МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 123 «Комп'ютерна інженерія»

Освітня програма: «Комп'ютерна інженерія»

Група: 2БКС-28

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

здобувача освіти денної форми навчання

БКС.28.14.000.КРБ

КОВАЛЕНКА
ЄВГЕНА
ОЛЕКСАНДРОВИЧА

м. Одеса
2024 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **123 «Комп'ютерна інженерія»**

Освітньо-професійна програма: **«Комп'ютерна інженерія»**

Група: **2БКС-28**

ПОЯСНЮВАЛЬНА ЗАПИСКА

До кваліфікаційної роботи бакалавра на тему: **«Дослідження можливостей платформи розробника STM32F407 для створення комплекту лабораторних завдань»**

Проектний матеріал складається з пояснювальної записки на 68 сторінках та графічного (презентаційного) матеріалу на 18 аркушах (слайдах)

Виконавець _____ (Коваленко Є.О.)

Керівник проекту _____ (Кривченко Ю.В.)

Консультанти:

з розділу охорони праці та техніки безпеки _____ (Чорновол Н.І.)

з нормоконтролю _____ (Петрашова В.І.)

старший консультант _____ (Кривченко Ю.В.)

До захисту допущений

Завідувач кафедри _____ (Іванова Л.В.)

Завідувач відділення _____ (Скорнякова О.В.)

Захист «26» 06 2024 р. Протокол ЕК № 1

Оцінка ЕК 4 (добре) 85 б

Секретар ЕК _____

АНОТАЦІЯ

У випускній кваліфікаційній роботі виконано дослідження можливостей платформи розробника STM32F407. Розроблено лабораторні завдання вивчення принципів роботи мікроконтролерів, їхньої основної периферії, організації передачі даних з їх використанням, керування іншими пристроями для вимірювання зовнішніх показників, налаштування відображення інформації, отриманої від зовнішніх пристроїв.

Реалізовано демонстраційні приклади на базі плати STM32F4Discovery, що дозволить здобувачам освіти на їх основі розробити власні проекти відповідно до індивідуальних завдань на лабораторних роботах.

Наведено опис архітектури ARM та 32-розрядних мікроконтролерів STM, надано інформацію, яка необхідна для початку роботи з відлагоджувальною платою STM32F4Discovery. Створено комплект з восьми лабораторних завдань, призначених для вивчення периферії мікроконтролерів Cortex-M4 ARM-архітектури сімейства STM32, можливостей, пристроїв та характеристик плати: ШІМ, АЦП, USART, SPI, DMA, таймерів.

Створені інструкції для самостійної розробки програмного забезпечення з використанням спеціалізованих середовищ розробки (CooCox CoIDE, Keil).

Виконано тестування розроблених мовою програмування С проектів у складі інструкцій до лабораторних робіт. Перевірено роботу всіх зазначених функцій, вказані особливості виконання проектів у середовищі програмування.

Описано заходи з охорони праці та техніки безпеки.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення *Комп'ютерних систем* Кафедра *Комп'ютерної інженерії*

Спеціальність *123 «Комп'ютерна інженерія»*

Освітньо-професійна програма *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР *Беркань І.В.*

"*15*" *01* 20 ___ р.

ЗАВДАННЯ

на кваліфікаційну роботу бакалавра

здобувачеві освіти *Коваленко Євген Олександрович*

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Дослідження можливостей платформи розробника STM32F407 для створення комплекту лабораторних завдань*

затверджена наказом по коледжу від "*02*" *листопада* 20*23* р. № *244-АГ-08*

2. Термін здачі студентом кваліфікаційної роботи *13.06.24р*

3. Вихідні дані до роботи *1. Специфікації мікроконтролеру STM32F407VG;*
2. Характеристики та перелік оснащення відлагоджувальної плати STM32F4 Discovery
3. Тактова частота роботи мікроконтролера STM32F407VG – 16..168 МГц;
4. Підключення відлагоджувальної плати до комп'ютера за допомогою miniUSB;
5. Середовище розробки CooCox CoIDE 1.7, Keil 4.xx, інструменти побудови проекту GNU

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
Архітектура ARM 32-розрядних мікроконтролерів STM
Аналіз оснащення відлагоджувальної плати STM32F4 Discovery
Розробка лабораторних завдань для вивчення периферії мікроконтролерів STM32 (використання портів введення/виведення, переривань, таймерів, генерування сигналу ШІМ, АЦП, USAR, SPI, контролеру ПДП, керування характеристиками сигналів).

5. Перелік графічного матеріалу (слайдів мультимедійної презентації) *Вбудовані модулі мікроконтролеру ARM Cortex-M4 STM32F407VG; Основні характеристики ядра мікроконтролерів STM32; Представлення цифрового периферійного пристрою для STM32; Зовнішній вигляд плати STM32F4 Discovery; Оснащення плати STM32F4 Discovery; Хід лабораторних робіт: з використання портів введення/виведення; з використання переривань і таймерів; з використання інтерфейсу SPI; з використання контролеру прямого доступу до пам'яті; з керування характеристиками генерованих сигналів та відображення інформації на LCD-дисплеї; Принципова електрична схема підключення компонентів плати*

6. Консультанти по кваліфікаційній роботі, із зазначенням розділів, що їх стосуються

Розділ	Консультант	ПІДПИС	
		Завдання видав	Завдання прийняв
Основний розділ	*Кривченко Ю.В.*		
Розділ охорони праці	*Чорновол Н.І.*		
Нормоконтроль	*Петрашова В.І.*		
Старший консультант	*Кривченко Ю.В.*		

7. Дата видачі завдання 15.01.24

Керівник роботи *Кривченко Ю.В.* _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	*Вступ. Постановка задачі проектування*	8.05.24	Виконав
2.	*Огляд архітектури 32-розрядних мікроконтролерів STM*	11.05.24	Виконав
3.	*Вивчення та опис оснащення відлагоджувальної плати*	13.05.24	Виконав
	STM32F4 Discovery		
4.	*Створення проекту в середовищі розробки CooCox CoIDE*	16.05.24	Виконав
5.	*Створення проекту в середовищі розробки Keil*	18.05.24	Виконав
6.	*Використання портів мікроконтролеру STM32F407VG*	20.05.24	Виконав
7.	*Використання переривань мікроконтролеру STM*	22.05.24	Виконав
8.	*Використання таймерів мікроконтролеру STM32F407VG*	25.05.24	Виконав
9.	*Генерування сигналу ШІМ та опис роботи*	28.05.24	Виконав
10.	*Використання АЦП на платі STM32F4 Discovery*	1.05.24	Виконав
11.	*Використання USART та інтерфейсу SPI, ПДП*	4.05.24	Виконав
12.	*Тестування розроблених програм у окремих проектах*	7.06.24	Виконав
13.	*Виконання розділу охорони праці та техніки безпеки*	8.06.24	Виконав
14.	*Виконання графічної частини роботи*	11.06.24	Виконав
15.	*Підготовка роботи до захисту*	13.06.24	Виконав

Здобувач освіти _____
(підпис)

Керівник роботи _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			БКС 28. 14 000. 00 КРБ	Випускна робота		
А4			БКС 28. 14 000. 00 КРБ ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	БКС 28. 14 000. 00 КРБ		
Розробив		Коваленко Є.О.		8.06	Дослідження можливостей платформи розробника STM32F407 для створення комплекту лабораторних завдань	Літ.	Аркуш 4 / Аркушів 68
Перевірив		Кривченко Ю.В.		8.06.		Н Д П	
Н. Контр.		Петрашова В.І.		09.06		ВСП "ОТФК ОНТУ" гр.2БКС-28	
Затверд.		Іванова Л.В.		10.06			

ЗМІСТ

ВСТУП

Для великої кількості систем, що вбудовуються (embedded systems), мікроконтролери АРМ є ключовим компонентом. Вони широко використовуються в мобільних телефонах, планшетах й інших пристроях. Наданий далі матеріал створено на основі вивчення автором можливостей 32-розрядних МК STM-32 й можливостей відлагоджувальної платформи Stm-32-f4-disc. Представлені лабораторні завдання передбачають вивчення побудови інформаційних систем, організації взаємодії декількох приладів між собою, використання передавання й відображення інформації у комп'ютерних системах, а разом із цим самостійну розробку програмного забезпечення із використанням спеціалізованих програм, проведення тестування й налагодження на реальних пристроях.

В ході здійснення лабораторних завдань вивчаються принципи діяльності МК, їхньої основної периферії, організації передавання байтів із їх використанням, управління іншими пристроями задля вимірювання зовнішніх показників, налаштування відображення інформації, отриманої від зовнішніх приладів. Здобувачі освіти зможуть отримати можливість на практиці самостійно налаштувати роботу демонстраційних прикладів й розробити власні відповідно біля індивідуального завдання. Задля успішного здійснення здобувачами освіти здійснення складених далі завдань біля лабораторних робіт необхідні знання основ теорії цифрової схемотехніки, проектування програмного забезпечення й алгоритмізації, а разом із цим знання мови програмування C++.

Випускна кваліфікаційна робота присвячена дослідженню можливостей платформи розробника STM-32-F407 задля створення комплекту лабораторних завдань по вивченню периферії МК STM-32. Робота містить матеріали, котрі будуть корисними задля вивчення МК архітектури АРМ Cortex-M4. В роботі наведено короткий опис архітектури АРМ й 32-розрядних МК STM, а разом із цим надано загальну інформацію, яка необхідна задля початку діяльності із відлагоджувальною платою Stm-32-f4-disc. Реалізовано декілька лабораторних робіт задля вивчення можливостей, приладів й характеристик платформи: PWM, АЦП, УСАПП, ППІ, ПДП, таймерів й ін.

1 ОСНОВНИЙ РОЗДІЛ

1.1 Архітектура ARM 32-розрядних МК STM

Мікроконтролери ARM засновані на RISC-архітектурі, що дозволяє зменшити споживання енергії процесором та, таким чином, робить їх добрим вибором задля систем, що вбудовуються. Однак є наступні відхилення від принципів RISC:

1. Змінна кількість циклів здійснення задля найпростіших інструкцій. Прості інструкції ARM можуть вимагати здійснення більш одного циклу. Наприклад, здійснення інструкцій Load й Save залежить від кількості регістрів, котрі їм передані;

2. Можливість з'єднувати команди зсуву й обертання із командами обробки інформації;

3. Умовне здійснення – інструкція виконується лише в випадку, якщо виконується конкретна умова. Це збільшує продуктивність та дозволяє позбутися операторів розгалуження;

4. Процесори ARM підтримують покращені DSP-інструкції задля операцій із цифровими сигналами.

Програміст спроможне розглядати ядро ARM як набір функціональних блоків – ALU, MMU й ін. – з'єднаних шиною байтів. Дані надходять в процесор через шину байтів. Декодер інструкцій обробляє інструкції перед виконанням. ARM можуть працювати тільки із даними, котрі записані у регістрах, тому перед виконанням інструкцій біля реєстрів записуються дані задля їх здійснення. ALU зчитує дані із регістрів, виконує необхідні операції й записує результат назад в регістр, звідки його можливо записати в зовнішню пам'ять [1].

Процесори ARM містять біля 18 регістрів: 16 регістрів байтів й 2 регістри процесів. Всі регістри містять 32 біти та називаються від R0 біля R15. Регістри R13, R14, R15 використовуються задля здійснення певних специфічних завдань:

- R13 використовується як показник стеку;
- R14 використовується як зв'язуючий регістр;
- R15 відіграє роль лічильника.

Залежно від контексту, ці регістри можуть використовуватися як регістри загального призначення. Разом із цим є два програмні регістри, котрі називаються CPSR (Current Program Status Register) й SPSR (Saved Program Status Register), котрі використовуються задля збереження стану процесора й команд. Ці процесори призначені задля використання в цифровій обробці сигналів (Digital Signal Processing, DSP). В загальному вигляді мікроконтролери, засновані на базі APM Cortex-M4 мають такі внутрішні модулі (рис. 1.1): мікроконтролер, встановлений на платі, що розглядається в даній роботі (STM32F407VG) використовує саме APM Cortex-M4 [4].

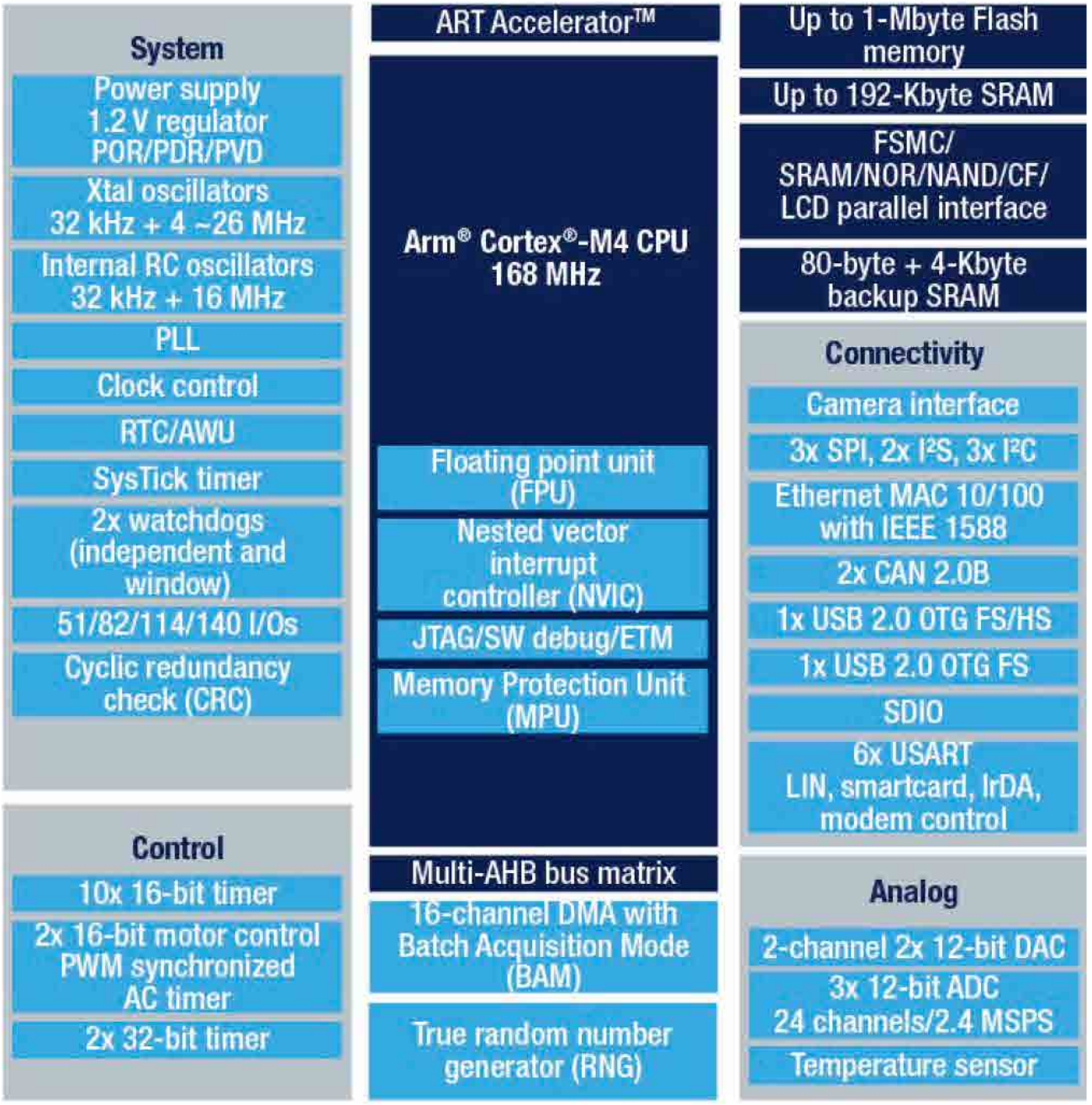


Рисунок 1.1. Вбудовані модулі мікроконтролеру APM Cortex-M4 STM32F407VG

На рис. 1.1 видно, що мікроконтролер сімейства STM-32 Cortex-M4 має багату периферію та спроможне використовуватися задля вирішення багатьох практичних завдань різної спрямованості.

Сімейство МК STM-32 побудовано із використанням 32-розрядного ядра Cortex різних версій (в мікроконтролері, встановленому на платі STM32F407VG, використовується ядро Cortex-M4). Деякі основні характеристики ядра МК STM-32 представлені в табл. 1.1.

Таблиця 1.1. Основні характеристики ядра МК STM-32

Характеристика	*Значення*
Ширина слів задля байтів, розрядів	32
Архітектура	Гарвардська
Конвеєр	3-ступінчастий
Набір інструкцій	RISC
Організація пам'яті програм, розрядів	32
Буфер передвиборки, розрядів	2x64
Середній розмір інструкції, байт	2
Тип переривань	Векторизовані
Затримка реагування на переривання	12 циклів
Режими управління енергоспоживанням	Сон, сон по виходу, глибокий сон
Інтерфейс задля відлагоджування	ST-LINK, JTAG

Мікроконтролери даного типу побудовані на гарвардській архітектурі й мають 3-ступінчастий конвеєр, який мінімізує час здійснення команд. Вони розроблені задля побудови систем із максимальною енергоефективністю й мають кілька режимів управління енергоспоживанням. Вони використовують внутрішні інтерфейси пам'яті шириною більше, ніж середня довжина інструкції. Це мінімізує кількість доступів біля шини пам'яті, а, отже, та споживання електроенергії, пов'язане із операціями по шині й читанням енергонезалежної пам'яті. Технологія безперервної обробки переривань із виключенням внутрішніх операцій над стеком (tail chaining) скорочує час реакції на переривання й виключає зайві операції [5].

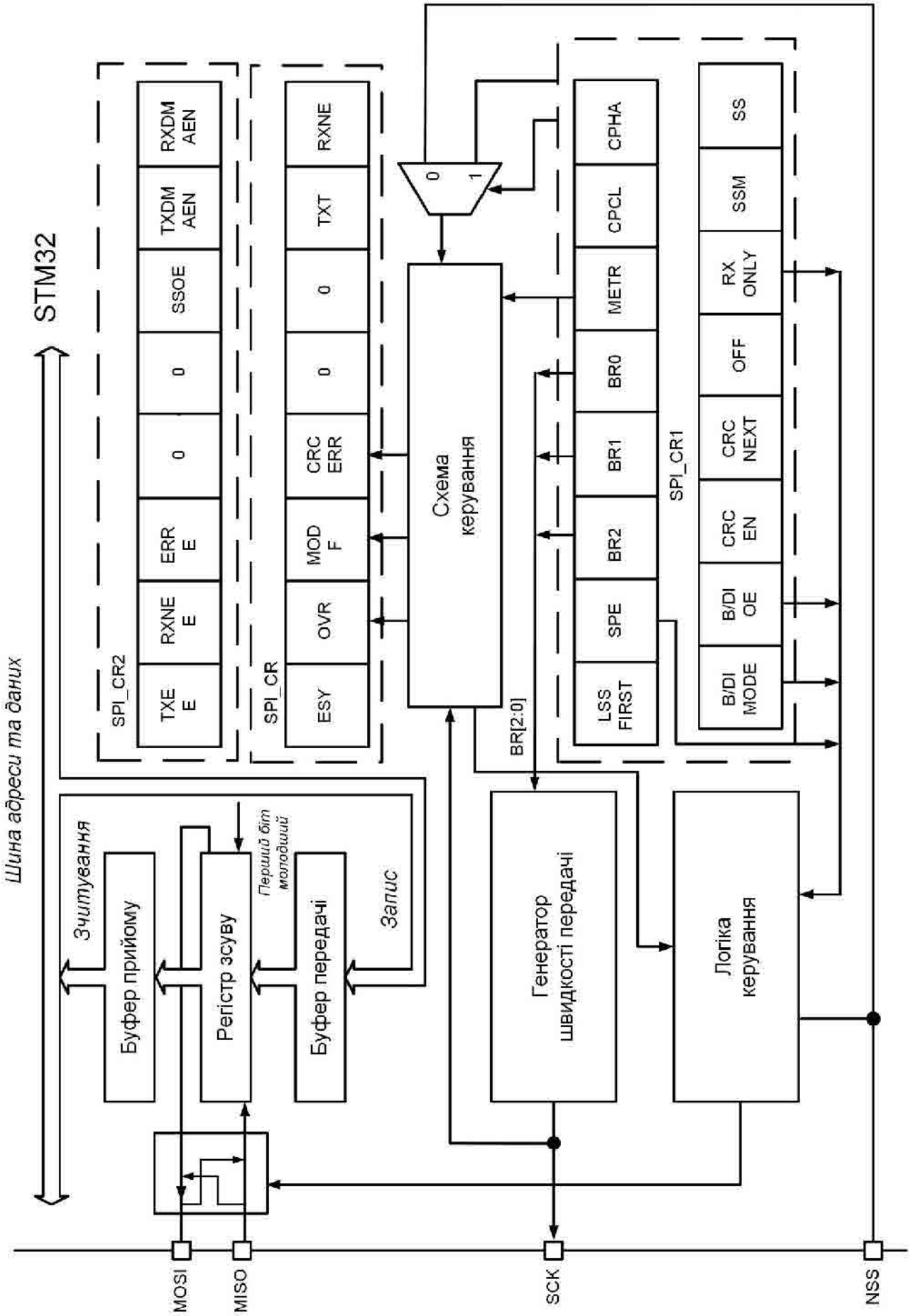


Рисунок 1.2. Представлення цифрового периферійного пристрою STM-32

На рис. 1.2 представлене спрощене уявлення цифрового периферійного пристрою задля STM-32. Периферійний вузол спроможне бути розділений на два головні блоки. Перший блок – це ядро, яке містить кінцеві автомати, лічильники й будь-який вид комбінаторної чи послідовної логіки. Воно призначене задля здійснення завдань, що не вимагають участі процесора, таких як прості завдання передавання байтів, управління аналоговими входами чи здійснення функцій, прив'язаних біля синхросигналів. Ядро периферійного вузла зв'язується із зовнішнім світом через порти вводу/виводу МК. Зовнішні із'єднання можуть складатися із кількох сигналів чи складних шин. Другий блок – налаштування й управління периферією, що здійснюються додатком через регістри, із'єднані із внутрішньою шиною, що поділяється із іншими ресурсами МК [6].

1.2 Оснащення відлагоджувальної платформи STM-32-F4

Відлагоджувальна плата STM-32-F4 Disc (рис. 1.3) призначена задля ознайомлення із можливостями 32-бітного МК на основі ARM-архітектури, а разом із цим задля реалізації власних приладів й програм із використанням апаратного забезпечення платформи.

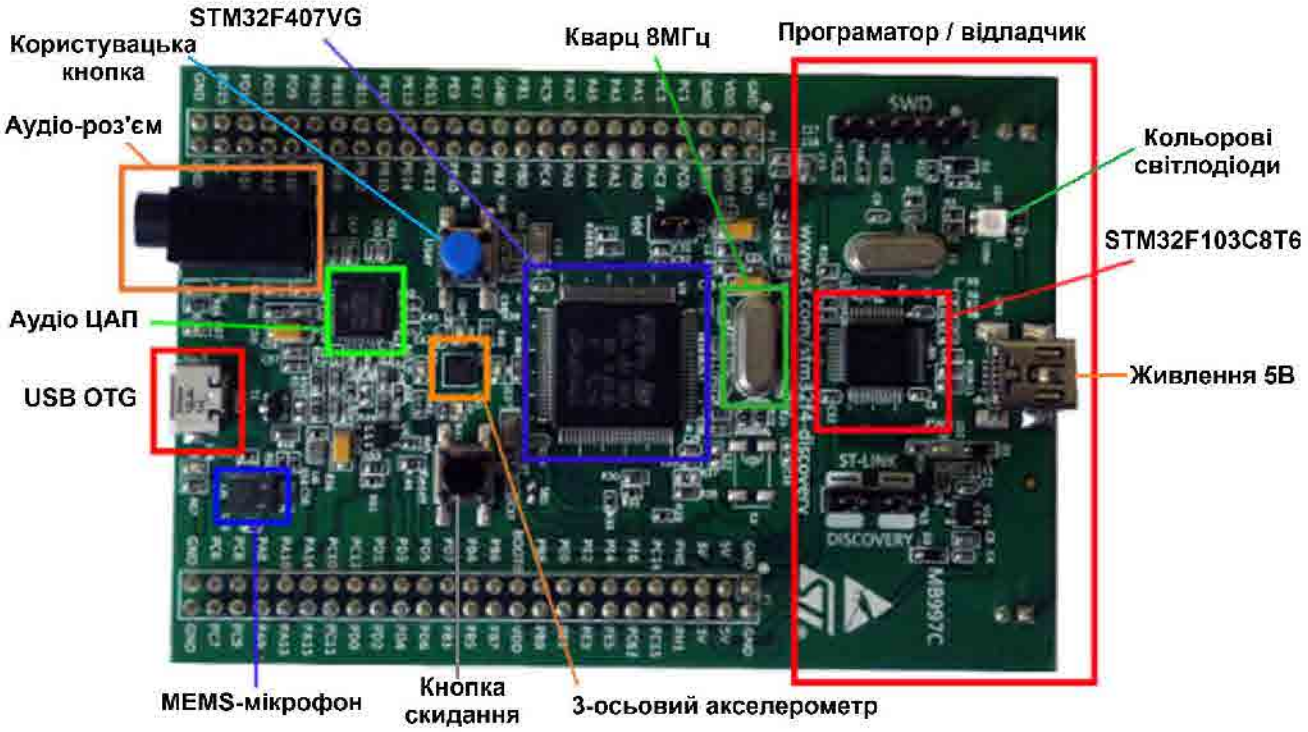


Рисунок 1.3. Зовнішній вигляд платформи STM-32-F4 Disc

Плату STM-32-F4 Disc (рис.1.4) обладнано:

- мікроконтролером STM32F407VGT6 із ядром Cortex-M4F тактовою частотою 168 МГц, 1 Мб Flash-пам'яті, 192 кб RAM у корпусі LQFP100;

- відладчиком ST-Link/V2 задля налагодження й програмування МК;

- живленням платформи через USB чи від зовнішнього джерела живлення 5 У;

- датчиком руху ST MEMS LIS302DL й виходами цифрового акселерометра по трьох осях;

- датчиком звуку ST MEMS MP45DT02;

- звуковим ЦАП CS43L22;

- вісьмома світлодіодами: LD1 (червоний/зелений) задля USB-підмикання, LD2 (червоний) задля живлення 3.3 У, чотири користувальницькі світлодіоди: LD3 (помаранчевий), LD4 (зелений), LD5 (червоний), LD6 (синій) й два світлодіоди задля USB On-The-Go – LD7 (зелений) й LD8 (червоний);

- двома кнопками (задля програмування користувачем й задля перезапуску).

Таким чином, відлагоджувальна плата оснащена великою кількістю периферії, що дозволяє реалізовувати на ній проекти різної складності.

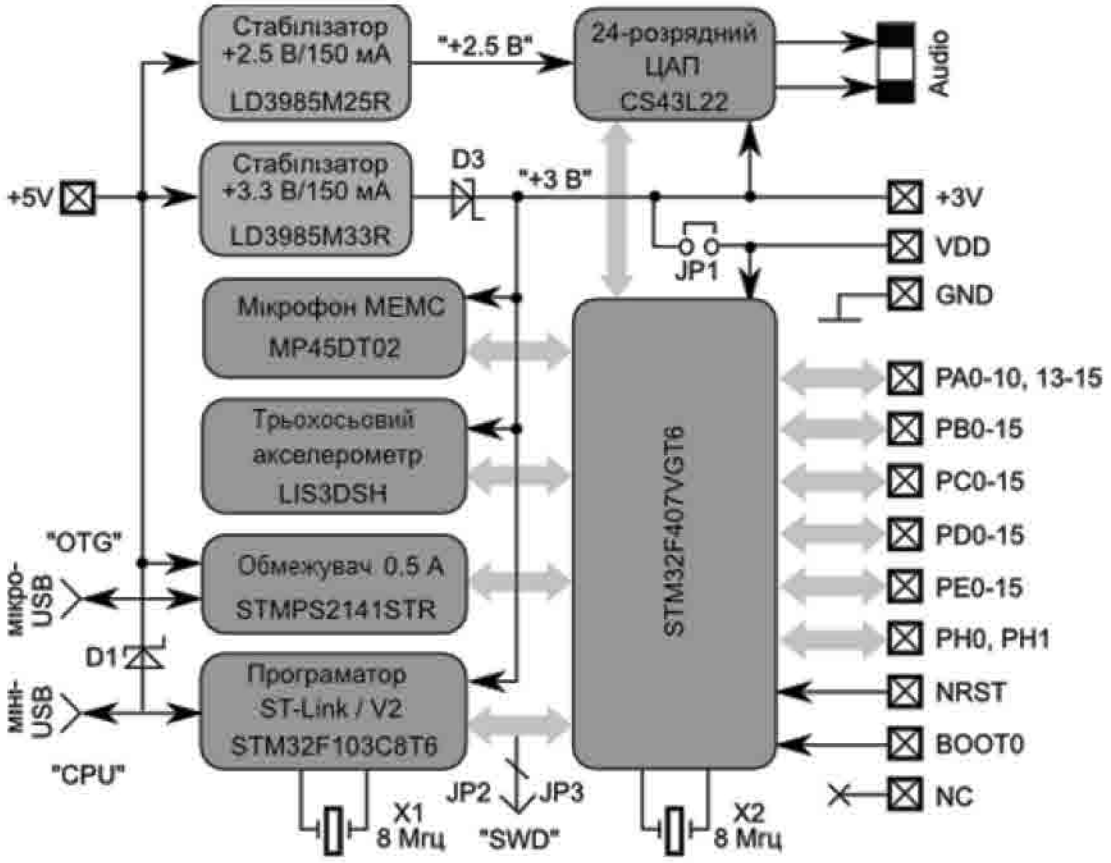


Рисунок 1.4. Оснащення платформи STM-32-F4 Disc

1.3 Розробка лабораторних завдань задля вивчення периферії МК STM-32

1.3.1 Створення проекту у середі проектування. Використання портів введення/виведення

Метою цієї діяльності є ознайомлення із процедурою створення проектів задля відлагоджувальної платформи STM-32-F4 Disc, вивчення її структури й принципів діяльності із портами введення/виведення, організації їхньої взаємодії.

Необхідним обладнанням й програмним забезпеченням є плата STM-32-F4 Disc, середовище проектування Cocox Coid 1.7, бібліотека CMSIS поза необхідності самостійного підмикання файлів біля проекту.

Контролер STM32F407VG містить п'ять 16-розрядних портів введення/виведення загального призначення, котрі позначені як GPIOx, де x спроможне мати значення A, B, C, D, E. Кожен порт GPIO має чотири 32-бітні регістри конфігурації (GPIOx_MODER, GPIOx_TYPER, GPIOx_TYPER, GPIOx_ORD), два 32-бітні регістри байтів (GPIOx_ODR, GPIOx_IDR) та два 32-бітові регістри вибору додаткових функцій (GPIOx_AFRH та GPIOx_AFRL).

Світлодіоди, призначені задля програмування, на платі підключені біля канала D (рис. 1.5).

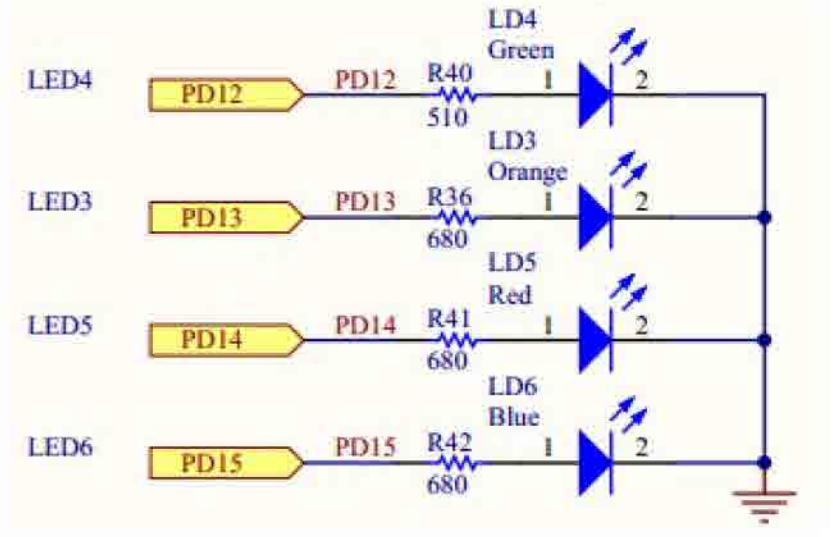


Рисунок 1.5. Схема підмикання led біля канала на відлагоджувальній платі

Інші чотири світлодіоди виконують службові функції індикації та у програмуванні певних дій не використовуються.

Задля здійснення лабораторних робіт рекомендується встановити середовище

Cocox Coid 1.7. Створення проекту виконується поза поміччю майстра, який відкривається під час здійснення команди меню Project/New Project. В майстрі слід крок поза кроком вказати ім'я проекту й його розташування, виробника й МК, задля якого призначена програма. Далі робота із проектом здійснюється поза поміччю вікна Repository, що дозволяє додати необхідні бібліотеки управління периферійними частинами МК, а разом із цим вікна Project. Вікно Repository разом із цим є майсром, який містить кілька вкладок, основною із яких є вкладка Peripherals (рис. 1.6).

Проект команд задля платформи STM-32-F4 Disc має подібну структуру всім засобам проектування. Зазвичай він включає два внутрішніх каталоги: один – задля файлів із бібліотеки CMSIS, а інший – задля файлів, призначених задля діяльності із периферією [7].

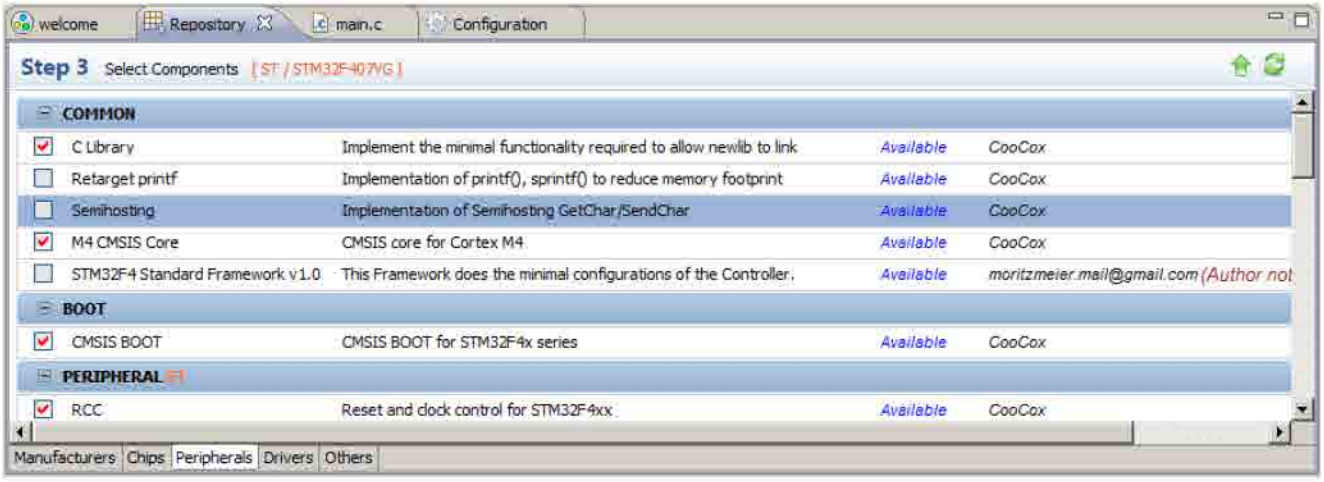


Рисунок 1.6. Вікно Repository середовища Cocox Coid

```
#include "stm32f4xx.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_gpio.h"
void Delay(uint32_t nCount)
{
while(nCount--)
{
}
}
int main(void)
```

Розглянемо приклад команд, яка демонструє роботу із портами введення/виведення. Вона дозволяє перемикати стани світлодіоду при натисканні кнопки користувача, яка знаходиться на відлагоджувальній платі.

```
{
GPIO_InitTypeDef gpioConf;
// ініціалізація входу, підключеного біля кнопки
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
gpioConf.GPIO_Pin = GPIO_Pin_0;
gpioConf.GPIO_Mode = GPIO_Mode_IN;
GPIO_Init(GPIOA, &gpioConf);

RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
// ініціалізація входу, підключеного біля світлодіоду
gpioConf.GPIO_Pin = GPIO_Pin_13;
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);
while(1) {
if(GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0) == 0) {
if (GPIO_ReadOutputDataBit(GPIOD, GPIO_Pin_13)) GPIO_ResetBits(GPIOD, GPIO_Pin_13);
else
GPIO_SetBits(GPIOD, GPIO_Pin_13);
Delay(5000);
while(GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0)==0);
Delay(5000);
}
}
}
```

Хід цієї лабораторної діяльності пропонується наступний:

1. На основі коду наданого прикладу команд здобувачеві освіти треба створити свій проект в середі проектування й перевірити його працездатність;

2. Ознайомитися із роботою функцій, переглянувши вихідний код, а разом із цим коментарі в вихідних файлах бібліотеки й в процесі налагодження;

3. Створити новий проект у середі проектування задля здійснення індивідуального завдання (табл.1.1);

4. Реалізувати потрібну функціональність;

5. Запрограмувати плату й продемонструвати роботу команд (рис.1.7).

Рисунок 1.7. Демонстрація діяльності портів введення/виведення

В роботі пропонуються індивідуальні завдання відповідно біля варіанту поза табл. 1.1 реалізувати схему увімкнення/вимкнення led, розташованих на платі:

Таблиця 1.1. Варіанти увімкнення led

Варіант	*1-й світлодіод*	*2-й світлодіод*	*3-й світлодіод*	*4-й світлодіод*
1	1	2	3	4
2	2	1	3	4
3	2	3	1	4
4	2	3	4	1
5	2	4	3	1
6	4	2	1	3
7	3	4	2	1
8	4	3	1	2
9	4	1	3	2
10	1	4	2	3

Номери увімкнення led присвоюються поза їх номерами в складі канала D (можливо знайти в документації відлагоджувальної платформи).

1.3.2 Переривання й їх використання. Використання таймерів

Метою цієї діяльності є ознайомлення із поняттям переривань, можливостями, котрі вони надають й навчитися використовувати таймери задля здійснення дій у затверджені часові інтервали.

Необхідним обладнанням й програмним забезпеченням є плата STM-32-F4 Disc, середовище проектування Cocox Coid 1.7, інструменти побудови проекту GNU Toolchain for APM Embedded Processors, бібліотека CMSIS поза необхідності самостійного підмикання файлів біля проекту.

Переривання – механізм, який дозволяє апаратному забезпеченню повідомляти про настання важливих подій в роботі. В момент, коли відбувається переривання, процесор перемикається із здійснення основної команд здійснення відповідного обробника переривань. Як тільки здійснення обробника завершено, продовжується здійснення основної команд із місця, у якому вона була перервана.

Задля використання переривань треба спочатку налаштувати регістр, який

називається Nested Vector Interrupt Controller (NVIC), вбудований контролер вектору переривань. Цей регістр є стандартною частиною архітектури ARM й зустрічається на всіх процесорах, незалежно від виробника.

NVIC розроблений таким чином, що затримка переривання є мінімальною. NVIC підтримує переривання із 16-ма рівнями пріоритету.

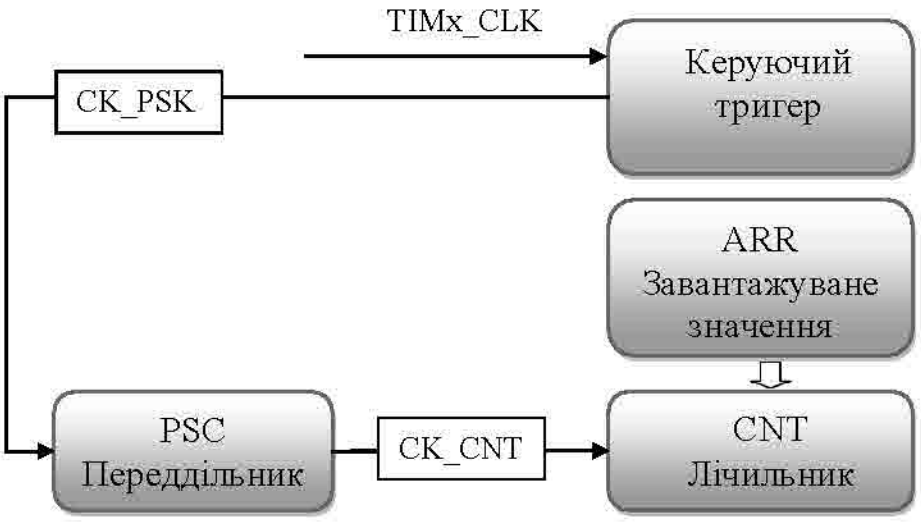


Рисунок 1.8. Схема управління підрахунком імпульсів

Мікроконтролер STM32F407VG містить 14 таймерів. В загальному вигляді схема управління підрахунком імпульсів представлена на рис. 1.8. Виробник мікроконтролеру поділяє всі таймери на три типи:

1) із розширеними можливостями;

2) загального призначення;

3) базові.

Кожен таймер спроможне мати біля 4 ліній захоплення/порівняння (саме вони використовуються у процесі генерації PWM).

Розглянемо приклад команд, яка демонструє роботу із перериваннями й таймерами. В ній реалізовано перемикання світлодіоду, підключеного біля канала введення/виведення, через певні інтервали часу.

```
#include «stm32f4xx.h»
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_tim.h"
#include "misc.h"
void INTTIM_Config(void);
```

```
void GPIO_Config(void);
int main(void)
{
GPIO_Config();
INTTIM_Config();
while(1)
{
}
}
void TIM2_IRQHandler(void) {
if (TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET) {
TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
GPIOD->ODR ^= GPIO_Pin_13;
}
}
void GPIO_Config(void) {
GPIO_InitTypeDef gpio_struct;
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE)
gpio_struct.GPIO_Pin = GPIO_Pin_13;
gpio_struct.GPIO_Mode = GPIO_Mode_OUT;
gpio_struct.GPIO_OType = GPIO_OType_PP;
gpio_struct.GPIO_PuPd = GPIO_PuPd_NOPULL;
gpio_struct.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_Init(GPIOD, &gpio_struct);
}
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 = 168 - 1;
tim_struct.TIM_ClockDivision = 0;
tim_struct.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM2, &tim_struct);
TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
TIM_Cmd(TIM2, ENABLE);
}
```

В додатковому заголовному файлі misc.h описані функції, перерахування, структури задля діяльності із перериваннями.

Хід цієї лабораторної діяльності пропонується наступний:

1. На основі коду наданого прикладу команд здобувачеві освіти треба створити свій проект у середі проектування й перевірити його працездатність;

2. Ознайомитися із роботою функцій, переглянувши вихідний код, а разом із цим коментарі в вихідних файлах бібліотеки й у процесі налагодження;

3. Створити новий проект у середі проектування задля здійснення індивідуального завдання;

4. Реалізувати потрібну функціональність;

5. Запрограмувати плату й продемонструвати роботу команд (рис.1.9).

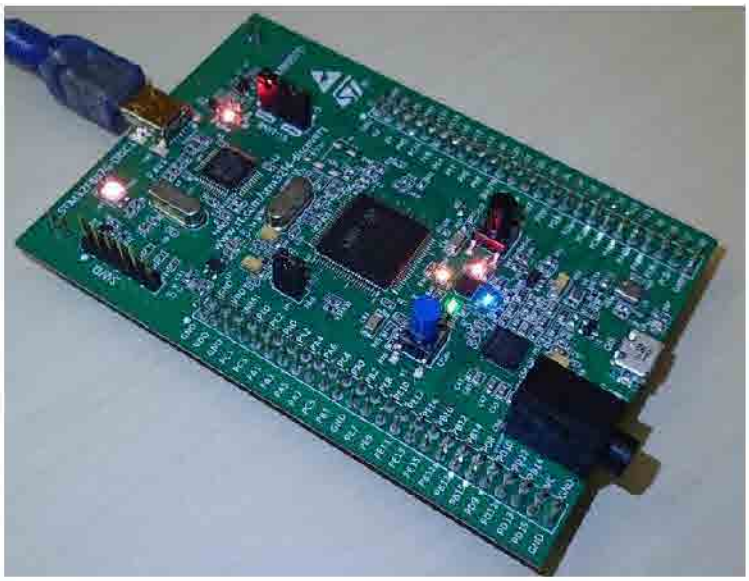


Рисунок 1.9. Демонстрація використання переривань й таймерів

В роботі пропонуються наступні індивідуальні завдання:

1. Поза поміччю одного із таймерів загального призначення згенерувати затримку тривалістю 1 с. (на основі байтів про робочу частоту). Затримка генерується на основі переривань таймера. Продемонструвати розрахунки, що підтверджують правильність завдання затримки;

2. Реалізувати поза поміччю переривання по переповненню таймера часову затримку із почерговим включенням led на платі по колу.

1.3.3 Генерування сигналу широтно-імпульсної модуляції

Метою цієї діяльності є ознайомлення із можливостями генерації PWM поза поміччю демонстраційної платформи й навчання генеруванню сигналу із заданими параметрами й його використання задля управління зовнішніми пристроями.

Необхідним обладнанням й програмним забезпеченням є плата Stm-32-f4-disc, середовище проектування Cocox Coid 1.7, інструменти побудови проекту GNU Toolchain for APM Embedded Processors, бібліотека CMSIS поза необхідності самостійного підмикання файлів біля проекту.

Широтно-імпульсна модуляція (PWM) – спосіб управління середнім значенням вольтажу на навантаженні шляхом зміни скважності імпульсів, керованих ключем. Мікроконтролери дозволяють генерувати PWM різної частоти. Оскільки виведення сигналу PWM не є основною функцією портів, котрі

підключені біля led, треба виконати відповідну їх конфігурацію. Задля цього слід задати здійснення портами альтернативних функцій. Генерація PWM пов'язана із використанням додаткових режимів таймера.

Перед підключенням пристрою мають бути відомі такі параметри PWM, як частота й коефіцієнт заповнення. Задля їх розрахунку у контролерах STM-32 треба визначити значення переддільника та значення, що автоматично завантажується в регістрі ARR (Auto-Reload Register). Розрахунок значення, яке слід записати біля переддільника, виконується наступним чином:

$$PSC = \frac{TIMxCLK}{TIMxCNT} - 1, \tag{1.1}$$

де PSC – значення переддільника,

$TIMxCLK$ – вхідна частота діяльності таймера,

$TIMxCNT$ – частота лічильника.

Задля отримання необхідної вихідної частоти слід записати значення в регістр ARR, що отримується із наступного співвідношення:

$$ARR\_VAL = \frac{TIMxCNT}{TIMx\_out\_freq} - 1, \tag{1.2}$$

де ARR_VAL – значення задля запису у регістр ARR,

$TIMxCNT$ – частота лічильника,

$TIMx\_out\_freq$ – потрібна вихідна частота PWM.

Останнім етапом є завдання потрібного коефіцієнту заповнення, що забезпечить потрібне значення вольтажу на виході. Це налаштування здійснюється поза поміччю регістру захвату/порівняння (capture/compare register, CCRx), виходячи із наступного співвідношення:

$$D = \frac{CCRx\_VAL}{ARR\_VAL} * 100\%, \tag{1.3}$$

где D – коефіцієнт заповнення,

$CCRx\_VAL$ – значення в регістрі CCRx (x – номер регістру задля конкретної лінії),

ARR _VAL – значення задля запису в регістр ARR.

Особливістю байтів МК є те, що е переддільник й інші регістри можливо записати будь-яке значення, яке можливо описати поза поміччю відведеної кількості розрядів.

Видача сигналу PWM на вихід є основним режимом діяльності виводів канала, а належить біля додаткових (альтернативних) режимів. Тому потрібно задати потрібний режим у налаштуваннях канала.

Задля підмикання альтернативних функцій біля портів введення/виведення треба:

1. Підключити порт біля альтернативної функції відповідного периферійного пристрою поза поміччю функції GPIO_PinAFConfig;

2. Налаштувати порт в режим здійснення альтернативної функції – GPIO_Mode_AF;

3. Виконати інші необхідні налаштування;

4. Викликати GPIO_Init задля застосування вказаних налаштувань.

Задля ознайомлення із можливістю генерації сигналу PWM й використання його задля управління яскравістю led надано наступну програму (в лістингу представлено зміст файлу із головною функцією).

```
#include "stm32f4xx.h"
#include
"stm32f4xx_gpio.h"
#include
"stm32f4xx_rcc.h"
#include "misc.h"
#include "stm32f4xx_tim.h"
#include "init.h"
int main(void)
{
init();
int brightness = 0;
int i;
while(1)
{
brightness++;
TIM4->CCR3 = 333 - (brightness + 0) % 333;
TIM4->CCR4 = 333 - (brightness + 166 / 2) % 333;
TIM4->CCR1 = 333 - (brightness + 333 / 2) % 333;
TIM4->CCR2 = 333 - (brightness + 499 / 2) % 333;
for(i=0;i < 10000; ++i);
```

```
for(i=0;i < 10000; ++i);
for(i=0;i < 10000; ++i);
}
}
```

У основній функції відбувається зміна значень регістру захвату/порівняння із певною затримкою. Функції із виконанням налаштувань перенесено біля окремого файлу init.c. Відповідно, оголошення функцій знаходяться в файлі init.h. В наступному лістингу представлено вміст файлу init.c.

```
#include "init.h"
#include "stm32f4xx.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_rcc.h"
#include "misc.h"
#include "stm32f4xx_tim.h"
void init() {
GPIOinit();
TimerInit();
}
void TimerInit() {
TIM_TimeBaseInitTypeDef time_init;
TIM_OCInitTypeDef oc_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 = 0;
time_init.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM4, &time_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);
oc_init.TIM_OutputState = TIM_OutputState_Enable;
oc_init.TIM_Pulse = 0;
TIM_OC2Init(TIM4, &oc_init);
TIM_OC2PreloadConfig(TIM4, TIM_OCPreload_Enable);
oc_init.TIM_OutputState = TIM_OutputState_Enable;
oc_init.TIM_Pulse = 0;
TIM_OC3Init(TIM4, &oc_init);
TIM_OC3PreloadConfig(TIM4, TIM_OCPreload_Enable);
oc_init.TIM_OutputState = TIM_OutputState_Enable;
oc_init.TIM_Pulse = 0;
TIM_OC4Init(TIM4, &oc_init);
TIM_OC4PreloadConfig(TIM4, TIM_OCPreload_Enable);
TIM_ARRPreloadConfig(TIM4, ENABLE);
 TIM_Cmd(TIM4, ENABLE);
}
void GPIOinit() {
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);
}
```

Як видно із прикладу, задля ініціалізації таймера треба відразу дві структури: одну – задля ініціалізації безпосередньо таймера, іншу – задля задання режиму порівняння виходу. Наведена програма дозволяє послідовно зменшувати яскравість світіння користувацьких led на платі.

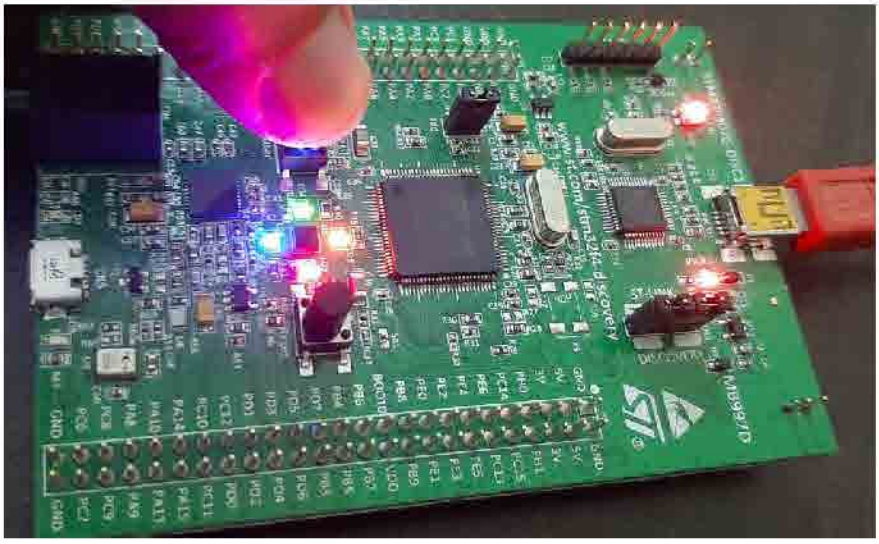


Рисунок 1.10. Демонстрація використання PWM

Хід цієї лабораторної діяльності пропонується наступний:

1. На основі коду наданого прикладу команд здобувачеві освіти треба створити свій проект в середі проектування й перевірити його працездатність;

2. Ознайомитися із роботою функцій, переглянувши вихідний код, а разом із цим коментарі в вихідних файлах бібліотеки й у процесі налагодження;

3. Створити новий проект в середі проектування задля здійснення індивідуального завдання, отриманого від викладача;

4. Реалізувати потрібну функціональність;

5. Запрограмувати плату й продемонструвати роботу команд (рис. 1.10).

1.3.4 Використання аналогово-цифрового перетворювача

Метою цієї діяльності є дослідження можливостей використання аналогово-цифрового перетворювача (АЦП) в складі відлагоджувальної платформи.

Необхідним обладнанням й програмним забезпеченням є плата STM-32-F4 Disc, середовище проектування Cocox Coid 1.7, інструменти побудови проекту GNU Toolchain for APM Embedded Processors, бібліотека CMSIS поза необхідності самостійного підмикання файлів біля проекту, джерело вольтажу (акумулятор), підключений біля входів АЦП.

Мікроконтролер STM32F407VG містить три АЦП. Розрядність всіх АЦП становить 12 біт. Кожен перетворювач здатний приймати сигнал із шістнадцяти зовнішніх каналів. Крім того, біля складу мікроконтролера входить датчик температури. Діапазон вхідної вольтажу становить 1.8…3.6 У. Датчик температури підключений біля вхідного канала АЦП_IN16, який використовується задля перетворення вихідної вольтажу сенсора на цифрове значення. Внутрішній датчик температури призначений задля відстеження зміни температури, а не задля її вимірювання, оскільки зсув показників датчика спроможне змінюватися під час змін параметрів процесу. Тому, якщо потрібне точне вимірювання абсолютних значень температури – краще використовувати зовнішній датчик.

В наступному прикладі наведено програму, яка дозволяє вимірювати напругу, яка подається на вхід АЦП. Перегляд значення вольтажу здійснюється в процесі налагодження (рис. 1.11).

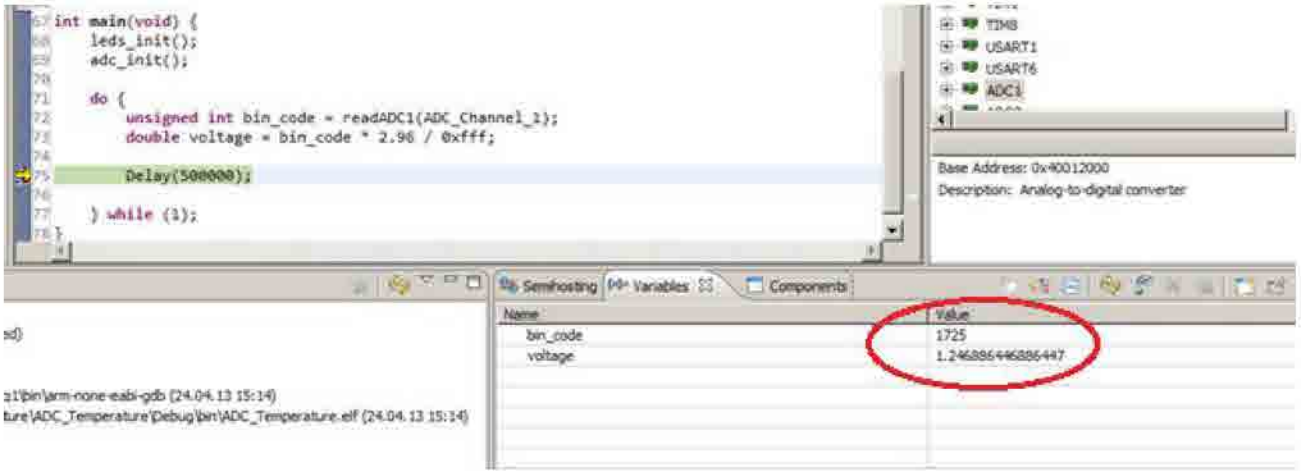


Рисунок 1.11. Контроль значення вольтажу в процесі відлагодження

Задля правильного визначення вольтажу треба провести додаткові вимірювання. Задля цього поза поміччю мультиметру чи вольтметру треба визначити напругу, яка відповідає 3 У, підключивши щупи вимірювального

приладу біля відповідних контактів на платі. Наприклад, мультиметр показав, що в даному випадку напруга дорівнює 2.96 У. Саме цей коефіцієнт треба безпосередньо записати біля коду команд. Розрахунок вольтажу виконується поза формулою:

$$U_{res} = \frac{U_{ref} * BIN_{res}}{BIN_{\max}}, \qquad (1.4)$$

де U_{res} – значення поданої вольтажу,

U_{ref} – напруга, відносно якої проводиться порівняння,

BIN_{max} – двійковий код максимально можливого значення, яке спроможне зберігатися в регістрі байтів АЦП. Залежить від його розрядності (в нашому випадку розрядність АЦП дорівнює 12 бітів),

BIN_{res} – значення двійкового коду із регістру байтів АЦП, записане після проведення вимірювань.

Відповідно, виходячи із формули 1.4, при подачі вольтажу 0 У, буде отримано мінімальне значення вольтажу. Здійснити це можливо із'єднавши контакти землі GND й першого контакту канала А перемичкою. В результаті цього значення у регістрі байтів АЦП зміниться так, як показано на рис. 1.12.

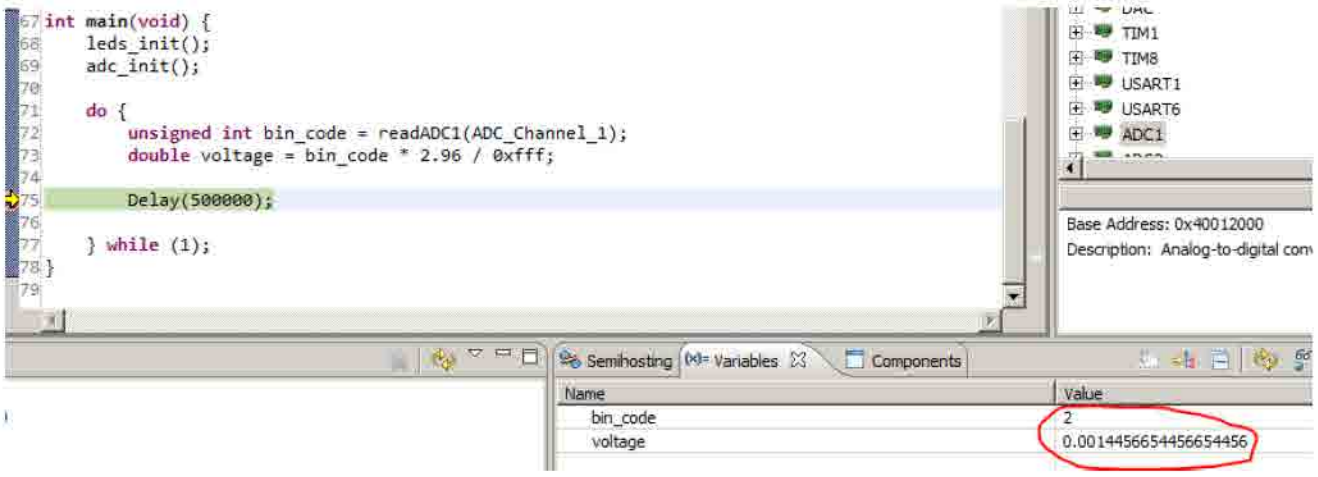


Рисунок 1.12. Контроль значення вольтажу в процесі відлагодження при підключенні землі біля входу АЦП

На рис.1.12 можливо помітити, що значення вольтажу досить близько біля нуля, що свідчить про коректну діяльності команд. При ініціалізації канала необхідне значення задля його діяльності в аналоговому процесі задається поза поміччю параметру GPIO_Mode_AN. Задля ініціалізації й зчитування значень із

АЦП треба налаштовати його таким чином, щоб перетворення здійснювалися програмно та виконувались поза поміччю виклику функції readADC1.

В наступному лістингу представлено вміст основного файлу main.c:

```
#include <stm32f4xx_rcc.h>
#include <stm32f4xx_gpio.h>
#include <stm32f4xx_adc.h>
void leds_init() {
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
GPIO_StructInit(&gpio);
gpio.GPIO_Mode = GPIO_Mode_AN;
gpio.GPIO_Pin = GPIO_Pin_1;
GPIO_Init(GPIOA, &gpio);
}
void adc_init() {
ADC_InitTypeDef ADC_InitStructure;
ADC_CommonInitTypeDef adc_init;
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
ADC_DeInit();
ADC_StructInit(&ADC_InitStructure);
adc_init.ADC_Mode = ADC_Mode_Independent;
adc_init.ADC_Prescaler = ADC_Prescaler_Div2;
ADC_InitStructure.ADC_ScanConvMode = DISABLE;
ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConvEdge_None;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;
ADC_CommonInit(&adc_init);
ADC_Init(ADC1, &ADC_InitStructure);
ADC_Cmd(ADC1, ENABLE);
}
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);
}
void Delay(unsigned int Val) {
for (; Val != 0; Val--);
}
int main(void) {
leds_init();
adc_init();
do {
unsigned int bin_code = readADC1(ADC_Channel_1);
double voltage = bin_code * 2.96 / 0xfff;
Delay(500000);
} while (1);
}
```

Хід цієї лабораторної діяльності пропонується наступний:

1. На основі коду наданого прикладу команд здобувачеві освіти треба створити свій проект в середі проектування й перевірити його працездатність;

2. Ознайомитися із роботою використовуваних функцій, переглянувши вихідний код, а разом із цим коментарі в вихідних файлах бібліотеки й у процесі налагодження;

3. Створити новий проект в середі проектування задля здійснення індивідуального завдання;

4. Реалізувати потрібну функціональність;

5. Запрограмувати плату й продемонструвати роботу команд (рис.1.13).

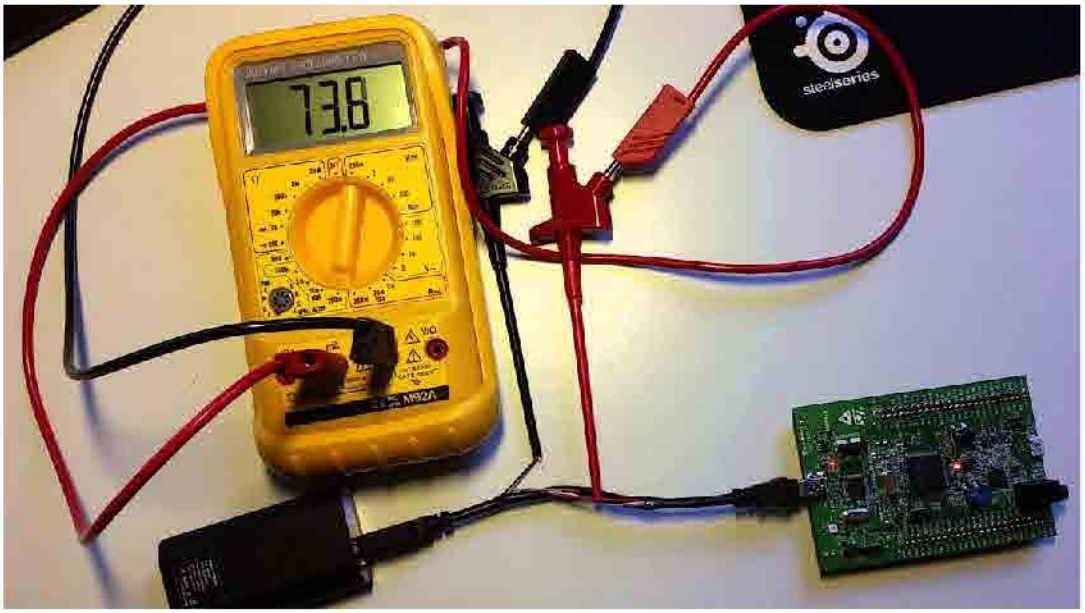


Рисунок 1.13. Демонстрація використання АЦП

Індивідуальні завдання поза даною темою:

1. Розробити програму, яка при подачі на АЦП вольтажу меншої, ніж половина від максимальної, вимикає світлодіод, а при подачі більшої вольтажу вмикає світлодіод. Підтвердити коректність діяльності команд поза поміччю мультиметру;

2. Продемонструвати прийняття байтів із кількох каналів АЦП;

3. Підключити біля АЦП задля вимірювання вбудований датчик зміни температури. При збільшенні температури вмикати червоний світлодіод, при зменшенні – синій;

4. Підключити біля АЦП вбудоване джерело вольтажу. При збільшенні вольтажу вмикати червоний світлодіод, при зменшенні – синій.

1.3.5 Використання універсального приймача/передавача УСАПП

Метою цієї діяльності є вивчення діяльності універсального синхронного/асинхронного приймача/передавача, організація передавання байтів біля інших приладів.

Необхідним обладнанням й програмним забезпеченням є плата STM-32-F4 Disc, середовище проектування Cocox Coid 1.7.

Мікроконтролер на демонстраційній платі містить загалом 6 приймачів/передавачів. При цьому 4 із них є синхронно-асинхронними (USART1, USART2, USART3, USART6) й 2 тільки асинхронними (UART4, UART5).

Розглянемо процес налаштування УСАПП задля прийняття байтів. Задля цього треба виконати такі дії:

1. Увімкнути тактування УСАПП та канала, виходи якого використовуються задля діяльності УСАПП;

2. Налаштувати відповідні виходи, вказавши при цьому режим діяльності здійснення альтернативної функції;

3. Підключити здійснення альтернативної функції біля зазначених виходів;

4. Задати налаштування діяльності УСАПП (частота, розмір передавання, кількість стоп-бітів, перевірка на парність);

5. Увімкнути переривання прийняття байтів задля УСАПП;

6. Увімкнути УСАПП.

В наступному прикладі розглянуто прийняття байтів поза поміччю USART1. Задля передавання й прийняття використовуються виходи канала А під номерами 9 (Transmit) й 10 (Receive). Задля кожного приймача-передавача виділено свої пари виводів задля прийому й передавання, котрі можливо знайти в документації біля мікроконтролера [3]. Насамперед, треба увімкнути тактування потрібних приладів:

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
```

Після цього треба провести ініціалізацію виводів канала й призначити здійснення альтернативної функції:

```
GPIO_InitStruct.GPIO_Pin = GPIO_Pin_9 | GPIO_Pin_10;
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_Init(GPIOA, &GPIO_InitStruct);
```

Далі треба вибрати альтернативну функцію, що виконується виводами канала. Робиться це поза поміччю функції GPIO_PinAFConfig().

```
GPIO_PinAFConfig(GPIOA, GPIO_PinSource9, GPIO_AF_USART1);
GPIO_PinAFConfig(GPIOA, GPIO_PinSource10, GPIO_AF_USART1);
```

Слід звернути увагу, що налаштуванню підлягають лише певні виводи, а інші виводи, наприклад, PA2 й PA3 використовувати в якості УСАПП не вдасться.

Далі треба ініціалізувати УСАПП. Код ініціалізації дуже схожий на ініціалізацію канала:

```
USART_InitStruct.USART_BaudRate = baudrate;
USART_InitStruct.USART_WordLength = USART_WordLength_8b;
USART_InitStruct.USART_StopBits = USART_StopBits_1;
USART_InitStruct.USART_Parity = USART_Parity_No;
USART_InitStruct.USART_HardwareFlowControl =
USART_HardwareFlowControl_None;
USART_InitStruct.USART_Mode = USART_Mode_Rx;
USART_Init(USART1, &USART_InitStruct);
```

На останньому етапі ініціалізації треба налаштувати переривання й увімкнути УСАПП:

```
USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
USART_Cmd(USART1, ENABLE);
```

Після вмикання переривання треба визначити його обробник. Обробник переривання має USART1_IRQHandler та він не повертає та не приймає жодних байтів. Крім того, УСАПП підтримує кілька переривань та задля всіх переривань використовується один обробник, тому треба перевіряти прапорець (RXNE) в регістрі статусу, щоб організувати обробку різних переривань в різних блоках коду обробника.

Після перевірки стану прапора можливо скинути прапор RXNE. Далі треба зчитати дані із регістру байтів й вивести їх на дисплей.

```
void USART1_IRQHandler(void){
if( USART_GetITStatus(USART1, USART_IT_RXNE) ){
USART_ClearITPendingBit(USART1, USART_IT_RXNE);
static uint8_t cnt = 0;
uint8_t t = USART_ReceiveData(USART1);
write_char(t);
}
}
```

В даному прикладі розглянуто прийняття байтів із іншого пристрою (використано плату STM8S Value Line, проте можливо використовувати будь-який інший пристрій із UART чи УСАПП) й відображення прийнятої інформації на символьному LCD-дисплеї.

Задля перевірки діяльності УСАПП навіть не обов'язково використовувати інший пристрій, тому що можливо замкнути між собою виходи передавача й приймача. Треба встановити однакові параметри повідомлення як на пристрої-передавачі, так та на приймачі.

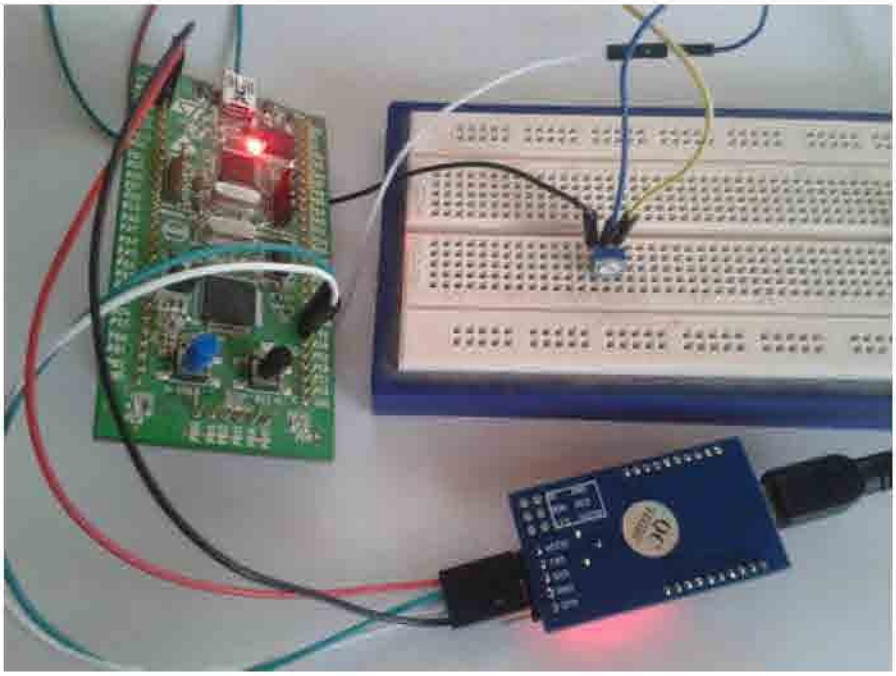


Рисунок 1.14. Демонстрація використання УСАПП

Хід цієї лабораторної діяльності пропонується наступний:

1. На основі коду наданого прикладу команд здобувачеві освіти треба створити свій проект в середі проектування й перевірити його працездатність;

2. Ознайомитися із роботою функцій, переглянувши вихідний код, а разом із цим коментарі в вихідних файлах бібліотеки й в процесі відлагодження;

3. Створити новий проект в середі проектування задля здійснення індивідуального завдання;

4. Реалізувати потрібну функціональність;

5. Запрограмувати плату й продемонструвати роботу команд (рис.1.14).

Індивідуальні завдання поза даною темою:

1. Задіяти два модулі УСАПП на платі й передати дані із одного в інший у асинхронному процесі. Задля демонстрації прийняття байтів змінювати стан одного із led на протилежний;

2. Продемонструвати прийняття/передачу в синхронному процесі. Задля

демонстрації прийому байтів змінювати стан одного із led на протилежний.

1.3.6 Використання інтерфейсу ППІ

Метою цієї діяльності є застосування інтерфейсу ППІ задля організації передавання байтів між пристроями, дослідження режиму ведучого й веденого приладів.

Необхідним обладнанням й програмним забезпеченням є плата STM-32-F4 Disc, середовище проектування Cocox Coid 1.7, Keil 4.xx, інструменти побудови проекту GNU Toolchain for APM Embedded Processors, бібліотека CMSIS, набір перемичок (3 шт.).

Serial Peripheral Interface (ППІ) – популярний інтерфейс задля обміну даними між мікросхемами. Шину ППІ організовано поза принципом "ведучий-ведений". В якості ведучого шини зазвичай виступає мікроконтролер, але ним разом із цим спроможне бути програмована логіка, DSP-контролер чи спеціалізована IC. Підключені біля провідної шини зовнішні пристрої утворюють підлеглі шини. У їх ролі виступають різноманітні мікросхеми. Зокрема: запам'ятовуючі пристрої (EEPROM, Flash-пам'ять, SRAM), годинник реального часу (RTC), АЦП/ЦАП, цифрові потенціометри, спеціалізовані контролери й ін.

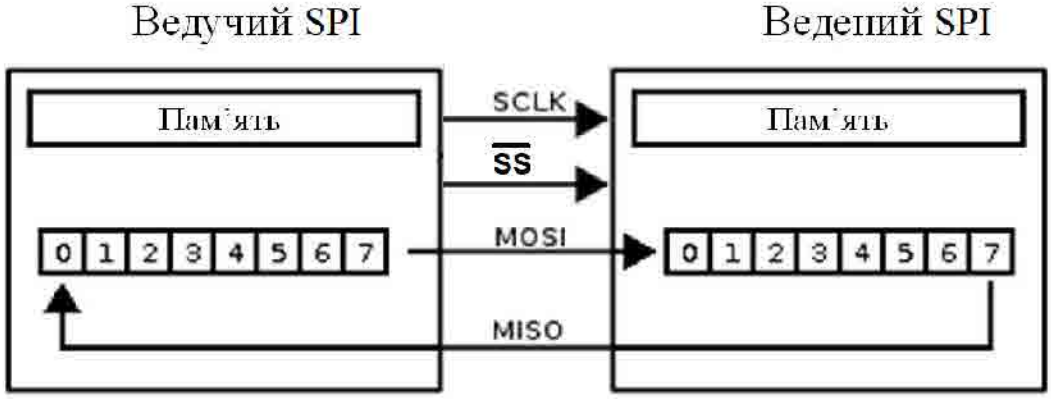


Рисунок 1.15. Схема простого підмикання із використанням ППІ

Головним складовим блоком інтерфейсу ППІ є звичайний зсувний регістр, сигнали синхронізації й введення/виведення бітового потоку якого та утворюють інтерфейсні сигнали. Таким чином, протокол ППІ правильніше назвати не протоколом передавання байтів, а протоколом обміну даними між двома зсувними регістрами, кожен із яких одночасно виконує функцію приймача та функцію

передавача. Обов'язковою умовою передавання по шині ППІ є генерація сигналу синхронізації шини. Цей сигнал має право генерувати тільки ведучий шини та від цього сигналу повністю залежить робота підлеглої шини.

Приклад найпростішого підмикання по шині ППІ показаний на рис. 1.15. Однойменні виводи із'єднуються між собою. Існують два канали передавання байтів (MOSI та MISO), один канал задля подачі тактових імпульсів (SCLK), а разом із цим лінія увімкнення веденого пристрою (SS). Ведений стає активним при подачі низького рівня по лінії SS.

ППІ – надзвичайно простий й поширений послідовний інтерфейс передавання байтів, що ґрунтується на зсувних регістрах. Його перевагою порівняно із УСАПП є можливість підмикання кількох відомих приладів. Однак в порівнянні із УСАПП він підтримує тільки синхронну передачу. Наявність кількох модулів ППІ в складі мікроконтролера дозволяє обійтися без підмикання додаткових приладів із таким інтерфейсом, а використовувати кілька приладів на платі, із'єднавши їхні входи між собою перемичками. Цей варіант є оптимальним задля навчальних цілей, оскільки вимагає мінімум додаткового обладнання задля початку діяльності.

В наступному прикладі продемонстровано застосування інтерфейсу ППІ задля організації передавання байтів між пристроями.

```
#include <stm32f4xx.h>
#include<stm32f4xx_rcc.h>
#include <stm32f4xx_gpio.h>
#include <stm32f4xx_spi.h>
#include <misc.h>
#define SPI_PINS GPIO_Pin_5 | \ GPIO_Pin_6 | \GPIO_Pin_7
void init(void);
void delay(int count);
void SPI1_IRQHandler(void) {
int res;
if (SPI_I2S_GetITStatus(SPI1, SPI_I2S_IT_RXNE) != RESET) {
SPI_I2S_ClearITPendingBit(SPI1, SPI_I2S_IT_RXNE);
res = SPI_I2S_ReceiveData(SPI1);
}
}
int main(void)
{
init();
int sendData = 0;
```

```
while(1)
      {
      delay(100); SPI_I2S_SendData(SPI2, sendData);
      while (SPI_I2S_GetFlagStatus(SPI2, SPI_I2S_FLAG_TXE) == RESET);
      sendData++;
      if (sendData == 0xff) sendData = 0;
      }
      }
      void init(void) {
      GPIO_InitTypeDef gpio_init;
      SPI_InitTypeDef spi_init;
      NVIC_InitTypeDef nvic_init;
      RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA | RCC_AHB1Periph_GPIOB |
      RCC_AHB1Periph_GPIOC, ENABLE);
      /* Configure slave pins */
      gpio_init.GPIO_Mode = GPIO_Mode_AF;
      gpio_init.GPIO_Pin = SPI_PINS;
      gpio_init.GPIO_Speed = GPIO_Speed_50MHz;
      gpio_init.GPIO_OType = GPIO_OType_PP;
      gpio_init.GPIO_PuPd = GPIO_PuPd_DOWN;
      GPIO_Init(GPIOA, &gpio_init);
      GPIO_PinAFConfig(GPIOA, GPIO_PinSource5, GPIO_AF_SPI1);
      GPIO_PinAFConfig(GPIOA, GPIO_PinSource6, GPIO_AF_SPI1);
      GPIO_PinAFConfig(GPIOA, GPIO_PinSource7, GPIO_AF_SPI1);
      gpio_init.GPIO_Pin = GPIO_Pin_2 | GPIO_Pin_3;
      GPIO_Init(GPIOC, &gpio_init);
      GPIO_PinAFConfig(GPIOC, GPIO_PinSource2, GPIO_AF_SPI2);
      GPIO_PinAFConfig(GPIOC, GPIO_PinSource3, GPIO_AF_SPI2);
      gpio_init.GPIO_Pin = GPIO_Pin_10;
      GPIO_Init(GPIOB, &gpio_init);
      GPIO_PinAFConfig(GPIOB, GPIO_PinSource10, GPIO_AF_SPI2);
      RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1, ENABLE);
      SPI_I2S_DeInit(SPI1);
      spi_init.SPI_Mode = SPI_Mode_Slave;
      spi_init.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
      spi_init.SPI_DataSize = SPI_DataSize_8b;
      spi_init.SPI_CPOL = SPI_CPOL_Low;
      spi_init.SPI_CPOL = SPI_CPHA_1Edge;
      spi_init.SPI_FirstBit = SPI_FirstBit_MSB;
      spi_init.SPI_NSS = SPI_NSS_Soft;
      spi_init.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_2;
      SPI_Init(SPI1, &spi_init);
      SPI_I2S_ITConfig(SPI1, SPI_I2S_IT_RXNE, ENABLE);
      RCC_APB1PeriphClockCmd(RCC_APB1Periph_SPI2, ENABLE);
      SPI_StructInit(&spi_init);
      SPI_I2S_DeInit(SPI2);
      spi_init.SPI_Mode = SPI_Mode_Master;
      spi_init.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
      spi_init.SPI_DataSize = SPI_DataSize_8b;
      spi_init.SPI_CPOL = SPI_CPOL_Low;
      spi_init.SPI_CPOL = SPI_CPHA_1Edge;
      spi_init.SPI_FirstBit = SPI_FirstBit_MSB;
      spi_init.SPI_NSS = SPI_NSS_Soft;
      SPI_Init(SPI2, &spi_init);
      nvic_init.NVIC_IRQChannel = SPI1_IRQn;
      nvic_init.NVIC_IRQChannelCmd = ENABLE;
      nvic_init.NVIC_IRQChannelPreemptionPriority = 0;
      nvic_init.NVIC_IRQChannelSubPriority = 0;
      NVIC_Init(&nvic_init);
      SPI_Cmd(SPI1, ENABLE);
      SPI_Cmd(SPI2, ENABLE);
      }
```

```
void delay(int count) {
while(--count);
}
```

Хід цієї лабораторної діяльності пропонується наступний:

1. На основі коду наданого прикладу команд здобувачеві освіти треба створити свій проект в середі проектування й перевірити його працездатність;

2. Ознайомитися із роботою функцій, переглянувши вихідний код, а разом із цим коментарі в вихідних файлах бібліотеки й у процесі налагодження;

3. Створити новий проект у середі проектування задля здійснення індивідуального завдання;

4. Реалізувати потрібну функціональність;

5. Запрограмувати плату й продемонструвати роботу команд (рис.1.16).

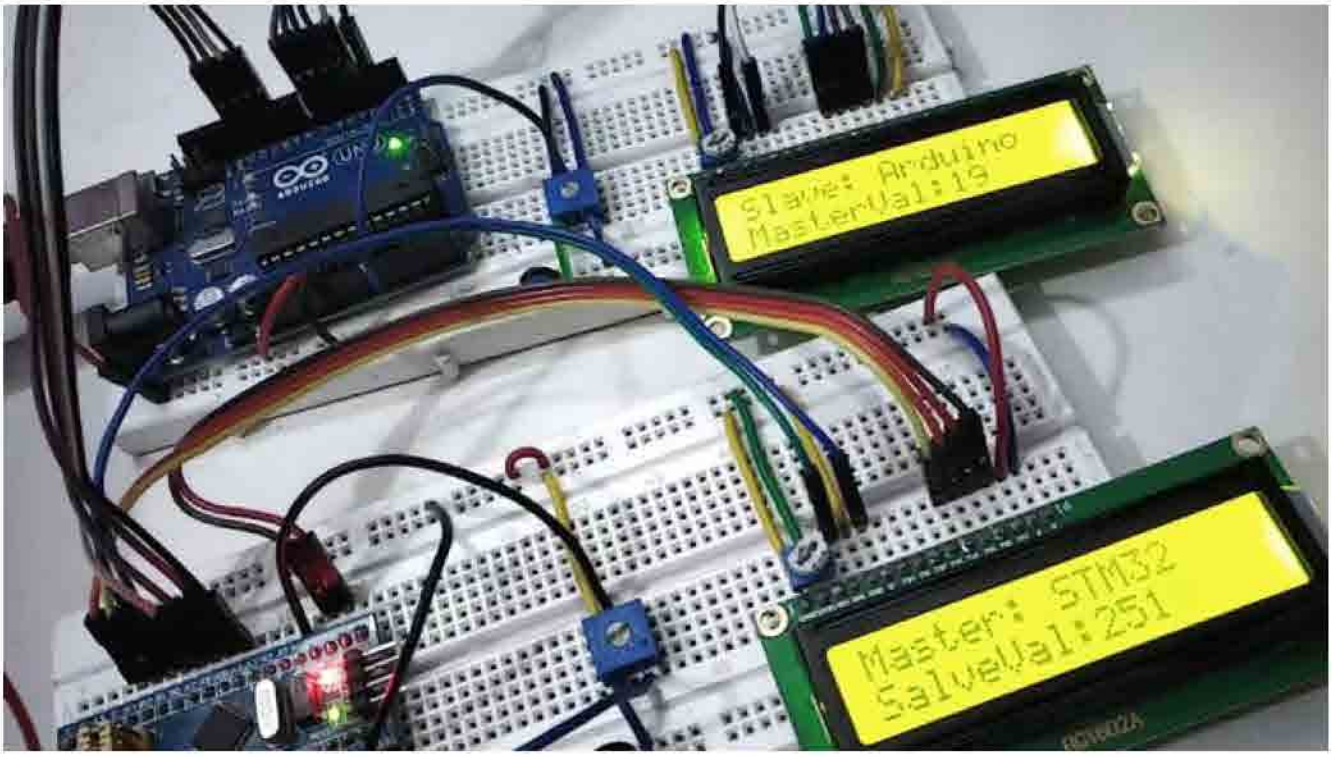


Рисунок 1.16. Демонстрація використання ППІ

Індивідуальні завдання поза даною темою:

1. Налаштувати на відлагоджувальній платі один із модулів в процесі прийому байтів й прийняти дані від іншого пристрою із відображенням на семисегментному індикаторі;

2. Продемонструвати роботу в процесі ведучого із підключенням декількох приладів: перший пристрій – інший модуль ППІ на платі, а другий – будь-який

інший пристрій, який спроможне виводити отримані дані на підключений семисегментний індикатор;

3. Організувати передачу байтів декільком провідним пристроям через певний проміжок часу. Підключити два пристрої біля одного модулю ППІ на відлагоджувальній платі та передавати дані поперемінно із інтервалом приблизно у 1 секунду із відображенням прийнятих байтів;

4. Організувати передачу байтів іншому модулю ППІ на відлагоджувальній платі (значення від 0x00 біля 0x0F) із відображенням двійкового значення на світлодіодах на платі;

5. Організувати одночасну передачу байтів декільком веденим пристроям, котрі відображають отримане значення на семисегментному індикаторі й в двійковому коді із використанням led.

1.3.7 Використання контролеру прямого доступу біля пам'яті

Метою цієї діяльності є використання режиму ПДП (прямого доступу біля пам'яті) й організація взаємодії із іншими пристроями в складі мікроконтролера.

Необхідним обладнанням й програмним забезпеченням є плата STM-32-F4 Disc, мультиметр, середовище проектування Cocox Coid 1.7, Keil 4.xx, інструменти побудови проекту GNU Toolchain for APM Embedded Processors, бібліотека CMSIS.

Direct Memory Access (ПДП, прямий доступ біля пам'яті) – механізм, який використовується у контролерах ARM задля переміщення байтів між пам'яттю й периферією без участі процесора. При використанні ПДП задля переміщення байтів не використовуються ресурси процесора, що спроможне бути особливо важливим при створенні додатків, що працюють із великою кількістю байтів та активно використовують периферію. Робота ПДП забезпечується окремим контролером, який виконує певні дії поза командою процесора. Схема взаємодії приладів наведена на рис. 1.17.

Контролер в складі відлагоджувальної платформи має в своєму розпорядженні відразу два контролери ПДП, котрі забезпечують загалом 16 потоків (по 8 потоків на кожен контролер), кожен із яких призначений задля управління запитами біля пам'яті від одного чи кількох приладів. Кожен потік спроможне забезпечити біля 8

каналів (запитів). Кожен контролер ПДП має пристрій вирішення конфліктів задля обробки запитів відповідно біля їх пріоритету.

Рисунок 1.17. Схема використання ПДП

Контролер ПДП дозволяє робити запис байтів в трьох напрямках:

- від периферії у пам'ять;
- із пам'яті біля периферії;
- із пам'яті у пам'ять.

Передача ведеться чи в процесі безпосередньої передавання, чи в процесі черги. Підтримується різний розмір байтів задля передавання, причому розмір байтів задля приймача та джерела спроможне бути неоднаковим. В такому разі ПДП визначає цю ситуацію й виконує необхідні дії задля оптимізації передавання, однак ця можливість підтримується лише в процесі черги.

Дуже корисною спроможне бути функція циклічної передавання, при використанні якої передача байтів починається із початкової адреси знову після передавання останньої одиниці байтів джерела. Програміст спроможне задавати пріоритети задля запиту кожного конкретного потоку чи пріоритет визначатиметься на основі значень поза замовчуванням.

В наступному прикладі команд продемонстровано використання ПДП задля передавання байтів із пам'яті у ЦАП. Зміна рівня сигналу на виході ЦАП відбувається циклічно із частотою зміни 1 с. Зміни відбуваються на основі значення, яке зчитується із пам'яті поза сигналом переповнення таймера. В результаті цього ЦАП відправляє запит біля ПДП та отримує дані, котрі записуються у регістр байтів, що призводить біля зміни рівня сигналу. В програмі слід звернути увагу на велику кількість параметрів, необхідних задля налаштування ПДП у порівнянні із іншими пристроями. ІЗ цим пов'язана складність використання ПДП, оскільки треба враховувати велику кількість

можливих налаштувань. Тим не менш, багато із них досить прості (напрямок передавання, значення адрес), чому сприяє бібліотека Standard Peripheral Library.

```
#include <stm32f4xx.h>
#include <stm32f4xx_rcc.h>
#include <stm32f4xx_dma.h>
#include <stm32f4xx_gpio.h>
#include <stm32f4xx_tim.h>
#include <stm32f4xx_dac.h>
uint8_t levels[] = {0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77};
void init(void);
void init_gpio(void);
void init_timer(void);
void init_dac(void);
void init_dma(void);
int main(void)
{
init();
while(1)
{
}
}
void init(void) {
init_gpio();
init_timer();
init_dac();
init_dma();
}
void init_gpio(void) {
GPIO_InitTypeDef gpio_init;
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
gpio_init.GPIO_Mode = GPIO_Mode_AN;
gpio_init.GPIO_Pin = GPIO_Pin_4;
gpio_init.GPIO_OType = GPIO_OType_PP;
gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_Init(GPIOA, &gpio_init);
}
void init_timer(void) {
TIM_TimeBaseInitTypeDef tim_init;
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM6, ENABLE);
tim_init.TIM_CounterMode = TIM_CounterMode_Up;
tim_init.TIM_Period = 16000 - 1;
tim_init.TIM_Prescaler = 1000 - 1;
TIM_TimeBaseInit(TIM6, &tim_init);
TIM_SelectOutputTrigger(TIM6, TIM_TRGOSource_Update);
TIM_Cmd(TIM6, ENABLE);
}
void init_dac(void) {
DAC_InitTypeDef dac_init;
RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);
DAC_StructInit(&dac_init);
dac_init.DAC_Trigger = DAC_Trigger_T6_TRGO;
dac_init.DAC_OutputBuffer = DAC_OutputBuffer_Enable;
dac_init.DAC_WaveGeneration = DAC_WaveGeneration_None;
DAC_Init(DAC_Channel_1, &dac_init);
}
void init_dma(void) {
DMA_InitTypeDef dma_init;
```

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA1, ENABLE);
     DMA_DeInit(DMA1_Stream5);
     dma_init.DMA_Channel = DMA_Channel_7;
     dma_init.DMA_PeripheralBaseAddr = (uint32_t)(DAC_BASE + 0x10);
     dma_init.DMA_Memory0BaseAddr = (uint32_t)&levels;
     dma_init.DMA_DIR = DMA_DIR_MemoryToPeripheral;
     dma_init.DMA_BufferSize = 8;
     dma_init.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
     dma_init.DMA_MemoryInc = DMA_MemoryInc_Enable;
     dma_init.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte;
     dma_init.DMA_MemoryDataSize = DMA_PeripheralDataSize_Byte;
     dma_init.DMA_Mode = DMA_Mode_Circular;
     dma_init.DMA_Priority = DMA_Priority_High;
     dma_init.DMA_FIFOMode = DMA_FIFOMode_Disable;
     dma_init.DMA_FIFOThreshold = DMA_FIFOThreshold_HalfFull;
     dma_init.DMA_MemoryBurst = DMA_MemoryBurst_Single;
     dma_init.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;
     DMA_Init(DMA1_Stream5, &dma_init);
     DMA_Cmd(DMA1_Stream5, ENABLE);
     DAC_Cmd(DAC_Channel_1, ENABLE);
     DAC_DMACmd(DAC_Channel_1, ENABLE);
     }
```

Після компіляції команд поза наведеним вище прикладом й прошивки відлагоджувальної платформи треба спостерігати поза напругою на виході PA4 поза поміччю мультиметру. Прилад має показати ступінчасту зміну вольтажу від 0 біля певного значення, після чого цикл повторюється. Крок збільшення однаковий задля всього циклу, що можливо побачити поза масивом значень задля запису в регістр байтів.

Хід цієї лабораторної діяльності пропонується наступний:

1. На основі коду наданого прикладу команд здобувачеві освіти треба створити свій проект в середі проектування й перевірити його працездатність (рис.1.18);

2. Ознайомитись із документацією по ПДП задля мікроконтролера вілагоджувальної платформи;

3. Змінити параметри адресації в програмі-прикладі;

4. Організувати взаємодію ПДП й іншого периферійного пристрою відповідно біля індивідуального завдання.

Індивідуальні завдання поза даною темою:

1. Реалізувати асинхронну передачу байтів через УСАПП, зчитуючи дані через певні проміжки часу поза поміччю ПДП;

2. Організувати прийняття байтів через УСАПП із записом біля пам'яті через

ПДП. Кількість байтів, що передаються, відома заздалегідь та дорівнює розміру масиву;

3. Продемонструвати передачу байтів через ППІ із пам'яті в циклічному процесі.

4. Продемонструвати прийом байтів через ППІ із записом в пам'ять. Режим запису – циклічний;

5. Реалізувати запис в пам'ять значень, отриманих поза поміччю діяльності АЦП через певні проміжки часу. Після запису перших 50 значень має починатися новий цикл запису.

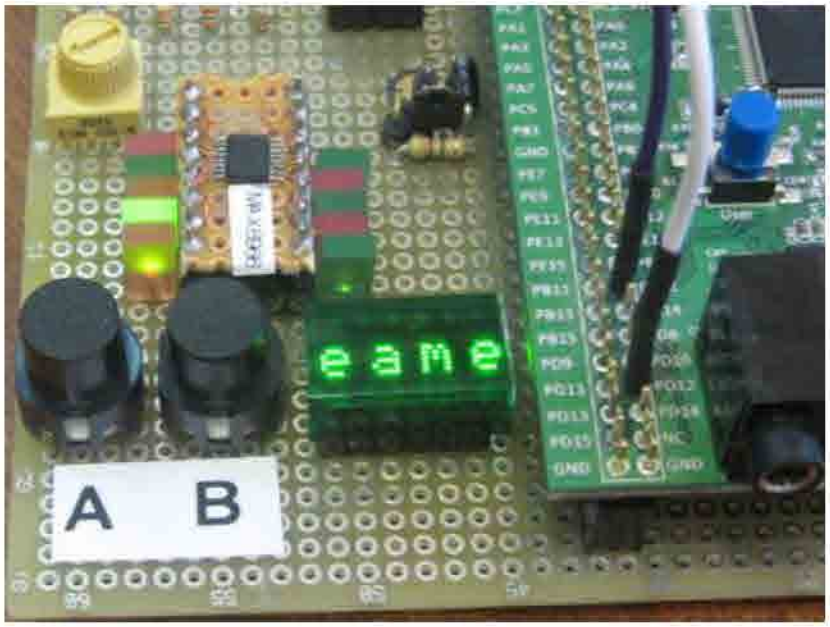


Рисунок 1.18. Демонстрація використання прямого доступу біля пам'яті

1.3.8 Управління характеристиками генерованих сигналів й відображення інформації на LCD-дисплеї

Метою цієї діяльності є вивчення можливості генерування сигналів й способи управління їх характеристиками із використанням відлагоджувальної платформи й відображенням інформації.

Необхідним обладнанням й програмним забезпеченням є плата STM-32-F4 Disc, символьний LCD-дисплей, 3 тактових кнопки й резистори задля їх підмикання, середовище проектування Cocox Coid 1.7, Keil 4.xx, інструменти побудови проекту GNU Toolchain for APM Embedded Processors, бібліотеки CMSIS й SPL.

В наступному прикладі буде складено програму, яка дозволяє на виході отримати сигнал прямокутної форми у заданому діапазоні частот й із заданим коефіцієнтом заповнення. Відповідно користувачеві треба надати можливість регулювання зазначені характеристики. Задля прикладу взяті такі значення:

- вихідна частота – 3-4 кГц;
- коефіцієнт заповнення ±50% від початкового значення, яким вважається значення 0,5.

Задля генерації сигналів прямокутної форми якнайкраще підходить вмикання таймеру у процесі PWM задля одного із вихідних каналів. Треба лише розрахувати значення задля налаштування самого таймера при зазначеній тактовій частоті пристрою (вони будуть різними задля 16 МГц та 168 МГц) та включити тактування й необхідний режим діяльності задля інших приладів в складі контролера.

Відповідно біля вказаних значень треба провести розрахунок задля налаштування таймера (розглядається випадок із внутрішньою частотою 16 МГц). Оскільки більш детально цей процес описаний в попередніх підрозділах, наведено лише результати обчислень. Задля того, щоб отримати на виході частоту у діапазоні 3-4кГц, треба встановити значення переддільника рівним 10, тоді діапазон зміни значень в регістрі автозавантаження буде змінюватися від 533 біля 400. Саме ці значення використовуються як граничні.

Значення коефіцієнту заповнення зберігається у програмі в вигляді дійсного числа, яке користувач спроможне змінювати, а виконані зміни записуються у регістр порівняння задля обраного вихідного канала.

В якості рішення задля надання користувачеві можливості регулювання параметрів вихідного сигналу пропонується підключити символьний LCD-дисплей та три кнопки задля здійснення регулювання. Функціональне призначення кнопок спроможне змінюватись відповідно біля дій користувача. В той же час, на екрані слід відображати інформацію про поточні зміни (зміна пунктів меню, значень характеристик), котрі виконує користувач.

Функціональна частина коду проекту оголошена й реалізована в файлах init.h й init.c. Далі наводиться лістинг файлу init.c:

```
#include "init.h"
#include "hd44780lib.h"
#include <stm32f4xx_gpio.h>
#include <stm32f4xx_rcc.h>
#include <stm32f4xx_exti.h>
#include <stm32f4xx_syscfg.h>
#include <stm32f4xx_tim.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <misc.h>
// ініціалізація портів, біля яких підключено дисплей
void init_gpio_lcd(void) {
GPIO_InitTypeDef gpio_init;
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD | RCC_AHB1Periph_GPIOE, ENABLE);
gpio_init.GPIO_Pin = DATA_PINS;
gpio_init.GPIO_Mode = GPIO_Mode_OUT;
gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_Init(DATA_PORT, &gpio_init);
gpio_init.GPIO_Pin = CONTROL_PINS;
gpio_init.GPIO_Mode = GPIO_Mode_OUT;
gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_Init(CONTROL_PORT, &gpio_init);
}
// ініціалізація каналу, біля якого підключені кнопки
void init_gpio_buttons(void) {
GPIO_InitTypeDef gpio_init;
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
gpio_init.GPIO_Mode = GPIO_Mode_IN;
gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
gpio_init.GPIO_Pin = BUTTON_PINS;
GPIO_Init(GPIOB, &gpio_init);
}
// конфігурування кнопок на платформі як джерел переривань
void configure_buttons(void) {
EXTI_InitTypeDef exti_init;
RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource0);
SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource1);
SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource2);
exti_init.EXTI_LineCmd = ENABLE;
exti_init.EXTI_Line = EXTI_Line0 | EXTI_Line1 | EXTI_Line2;
exti_init.EXTI_Mode = EXTI_Mode_Interrupt;
exti_init.EXTI_Trigger = EXTI_Trigger_Rising;
EXTI_Init(&exti_init);
}
// функція дозволу зазначеного переривання
void enable_interrupt(IRQn_Type irq) {
NVIC_InitTypeDef nvic_init;
nvic_init.NVIC_IRQChannel = irq;
nvic_init.NVIC_IRQChannelCmd = ENABLE;
nvic_init.NVIC_IRQChannelSubPriority = 1;
nvic_init.NVIC_IRQChannelPreemptionPriority = 1;
NVIC_Init(&nvic_init);
}
// дозвіл переривання по натисканню кнопки
void configure_interrupts(void) {
enable_interrupt(EXTI0_IRQn);
enable_interrupt(EXTI1_IRQn);
```

```
enable_interrupt(EXTI2_IRQn);
      }
      void configure_delay_timer(void) {
      TIM_TimeBaseInitTypeDef tim_init;
      tim_init.TIM_CounterMode = TIM_CounterMode_Up;
      tim_init.TIM_Prescaler = 1600 - 1;// налаштування переддільника таймеру задля
      затримок
      RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM7, ENABLE);
      TIM_TimeBaseInit(DELAY_TIMER, &tim_init);
      }
      // налаштування таймеру задля генерації ШІМ
      void configure_pwm_timer(void) {
      GPIO_InitTypeDef gpio_init;
      TIM_TimeBaseInitTypeDef tim_init;
      TIM_OCInitTypeDef oc_init;
      RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
      gpio_init.GPIO_Mode = GPIO_Mode_AF;
      gpio_init.GPIO_Pin = GPIO_Pin_0;
      gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
      gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
      GPIO_Init(GPIOA, &gpio_init);
      GPIO_PinAFConfig(GPIOD, GPIO_PinSource0, GPIO_AF_TIM2);
      RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
      tim_init.TIM_Prescaler = 10 - 1;
      tim_init.TIM_Period = frequency_value - 1;
      tim_init.TIM_CounterMode = TIM_CounterMode_Up;
      tim_init.TIM_ClockDivision = 0;
      tim_init.TIM_RepetitionCounter = 0;
      TIM_TimeBaseInit(PWM_TIMER, &tim_init);
      oc_init.TIM_OCMode = TIM_OCMode_PWM1;
      oc_init.TIM_Pulse = frequency_value / 2;
      oc_init.TIM_OutputState = TIM_OutputState_Enable;
      TIM_OC1Init(PWM_TIMER, &oc_init);
      TIM_Cmd(PWM_TIMER, ENABLE);
      }
      // заповнення меню
      void fill_menu(void) {
      main_option_index = 0;
      main_options[0] = create_menu_item("1.Change frequency", set_active_freq_menu);
      main_options[1] = create_menu_item("2. Change fill", set_active_fill_menu);
      main_menu.items[0] = create_menu_item("", down_menu);
      main_menu.items[1] = main_options[main_option_index];
      main_menu.items[2] = create_menu_item("", up_menu);
      freq_menu.items[0] = create_menu_item("", decrement_freq);
      freq_menu.items[1] = create_menu_item("", set_active_main_menu);
      freq_menu.items[2] = create_menu_item("", increment_freq);
      filling_menu.items[0] = create_menu_item("", increment_fill);
      filling_menu.items[1] = create_menu_item("", set_active_main_menu);
      filling_menu.items[2] = create_menu_item("", decrement_fill);
      set_active_menu(&main_menu);
      }
      void set_active_menu(struct menu * menu) {
      active_menu = menu;
      }
      void set_active_freq_menu(void) {
      active_menu = &freq_menu;
      show_freq_on_screen();
      }
      void set_active_fill_menu(void) {
      active_menu = &filling_menu;
      show_fill_on_screen();
      }
```

```
void set_active_main_menu(void) {
active_menu = &main_menu;
show_text_on_screen(&main_menu.items[1]);
}
// реалізація функції затримки
void delay(int times) {
DELAY_TIMER->CNT = 0;
DELAY_TIMER->ARR = times;
TIM_Cmd(DELAY_TIMER, ENABLE); // запуск таймеру
while(TIM_GetFlagStatus(DELAY_TIMER, TIM_FLAG_Update) == RESET); // очікування
TIM_ClearFlag(DELAY_TIMER, TIM_FLAG_Update);
TIM_Cmd(DELAY_TIMER, DISABLE); // зупинка таймеру
}
struct menu_item create_menu_item(char * text, action act) {
struct menu_item item;
strcpy(item.menu_text, text);
item.action = act;
return item;
}
void show_text_on_screen(struct menu_item * item) {
lcd_clear();
write_string(item->menu_text);
}
// збільшення значення змінної задля зміни частоти
void increment_freq(void) {
if (frequency_value == MAX_FREQ_VALUE) return;
frequency_value++;
TIM_SetAutoreload(PWM_TIMER, frequency_value - 1);
TIM_SetCompare1(PWM_TIMER, frequency_value * fill_value);
show_freq_on_screen();
}
// зменшення значення змінної задля зміни частоти
void decrement_freq(void) {
if (frequency_value == MIN_FREQ_VALUE) return;
frequency_value--;
TIM_SetAutoreload(PWM_TIMER, frequency_value - 1);
TIM_SetCompare1(PWM_TIMER, frequency_value * fill_value);
show_freq_on_screen();
}
void increment_fill(void) {
if (MAX_FILL_VALUE - fill_value < 0.01) return;
fill_value += 0.01;
TIM_SetCompare1(PWM_TIMER, frequency_value * fill_value);
show_fill_on_screen();
}
void decrement_fill(void) {
if (fill_value - MIN_FILL_VALUE < 0.01) return;
fill_value -= 0.01;
TIM_SetCompare1(PWM_TIMER, frequency_value * fill_value);
show_fill_on_screen();
}
void change_menu_item(struct menu_item * item, char * text, action act) {
strcpy(item->menu_text, text);
item->action = act;
}
void chamge_menu_item(struct menu_item * cur, struct menu_item * next) {
cur = next;
}
void up_menu(void) {
if (main_option_index == 1) {
} else {
main_option_index++;
```

```
main_menu.items[1] = main_options[main_option_index];
show_text_on_screen(&main_menu.items[1]);
}
}
void down_menu(void) {
if (main_option_index == 0) {
} else {
main_option_index--;
main_menu.items[1] = main_options[main_option_index];
show_text_on_screen(&main_menu.items[1]);
}
}
void show_freq_on_screen(void) {
char text[20];
double freq = (double)16000000 / (frequency_value * 10) / 1000;
gcvt(freq, 4, text);
strcat(text, " kHz");
lcd_clear(); write_string(text);
}
void show_fill_on_screen(void) {
char text[20];
gcvt(fill_value, 4, text);
strcat(text, " %"); lcd_clear();
write_string(text);
}
// ініціалізація всіх компонентів
void init_all(void) {
frequency_value = 533;
fill_value = 0.5;
init_gpio_buttons();
init_gpio_lcd();
configure_buttons();
configure_interrupts();
configure_delay_timer();
configure_pwm_timer();
fill_menu(); lcd_init();
show_text_on_screen(&main_menu.items[1]);
}                                                                    }
```

В заголовному файлі знаходяться оголошення функцій, а разом із цим використовуваних змінних й констант:

```
#ifndef INIT_H
#define INIT_H
#include <stm32f4xx.h>
#include <stm32f4xx_gpio.h>
#include <stm32f4xx_tim.h>
#define BUTTON_PINS GPIO_Pin_0 | \ GPIO_Pin_1 | \ GPIO_Pin_2
#define DELAY_TIMER TIM7
#define PWM_TIMER TIM2
#define MAX_FREQ_VALUE 533
#define MIN_FREQ_VALUE 400
#define MAX_FILL_VALUE 0.75
#define MIN_FILL_VALUE 0.25
typedef void (*action)(void);
struct menu {
struct menu_item items[3]; // пункти меню
};
int frequency_value; // змінна задля завдання частоти, має значення 400...533
double fill_value; // змінна задля завдання коефіцієнту заповнення
struct menu * active_menu; // поточне меню
```

```
struct menu_item main_options[3]; // пункти головного меню
struct menu main_menu;
struct menu freq_menu;
struct menu filling_menu;
int main_option_index; // індекс поточного пункту в головному меню
void init_gpio_lcd(void);
void init_gpio_buttons(void);
void configure_buttons(void);
void configure_interrupts(void);
void configure_delay_timer(void);
void configure_pwm_timer(void);
void init_all(void);
struct menu_item create_menu_item(char * text, action act);
void show_text_on_screen(struct menu_item * item);
void show_freq_on_screen(void);
void show_fill_on_screen(void);
void fill_menu(void);
void empty_action(void);
void change_menu_item(struct menu_item * item, char * text, action act);
void chamge_menu_item(struct menu_item * cur, struct menu_item * next);
void up_menu(void);
void down_menu(void);
void increment_freq(void);
void decrement_freq(void);
void increment_fill(void);
void decrement_fill(void);
#endif // INIT_H
```

В файлі із основною функцією викликається функція ініціалізації всіх приладів та описані обробники переривань задля натискань кнопок:

```
#include <stm32f4xx_exti.h>
#include <string.h>
#include "hd44780lib.h"
#include "init.h"
void EXTI0_IRQHandler(void) {
EXTI_ClearITPendingBit(EXTI_Line0);
if (active_menu != NULL)
delay(100);
active_menu->items[0].action();
}
void EXTI1_IRQHandler(void) {
EXTI_ClearITPendingBit(EXTI_Line1);
if (active_menu != NULL)
delay(100);
active_menu->items[1].action();
}
void EXTI2_IRQHandler(void) {
EXTI_ClearITPendingBit(EXTI_Line2);
if (active_menu != NULL)
delay(100);
active_menu->items[2].action();
}
int main(void)
{
init_all();
while(1)
{
}
}
```

В наведеному вище прикладі задіяно 3 порти введення/виведення (біля контактів канала А, на якому генерується сигнал, підключається вимірювальний прилад – мультиметр). Кнопки підключаються біля контактів 0-2 канала У. Вихідний сигнал можливо спостерігати на виході 0 канала А.

Разом із безпосередньою обробкою сигналів важливою частиною діяльності схеми є відображення інформації, результатів обробки задля користувача. В тому випадку, коли схема досить проста та спрямована на відстеження невеликої кількості показників, достатньо, наприклад, використовувати семисегментний індикатор чи символьний LCD-дисплей. Такі дисплеї часто можуть відображати більше інформації, ніж індикатори, складені із кількох семисегментних, а разом із цим можуть відображати набагато більше різних символів. При виконанні цієї діяльності передбачається використовувати дисплей RC-1602-A (рис. 1.19). Даний дисплей відображає 2 рядки по 16 символів в кожному й працює під керуванням контролера KS0066U. Він забезпечує наступну функціональність діяльності із дисплеєм:

- здійснення команд управління (управління розрядністю, курсором й інші операції);

- здійснення запису й зчитування пам'яті дисплея.

Використана схема підмикання дисплею представлена на рис. 1.20.

Рисунок 1.19. Зовнішній вигляд дисплея RC-1602-A

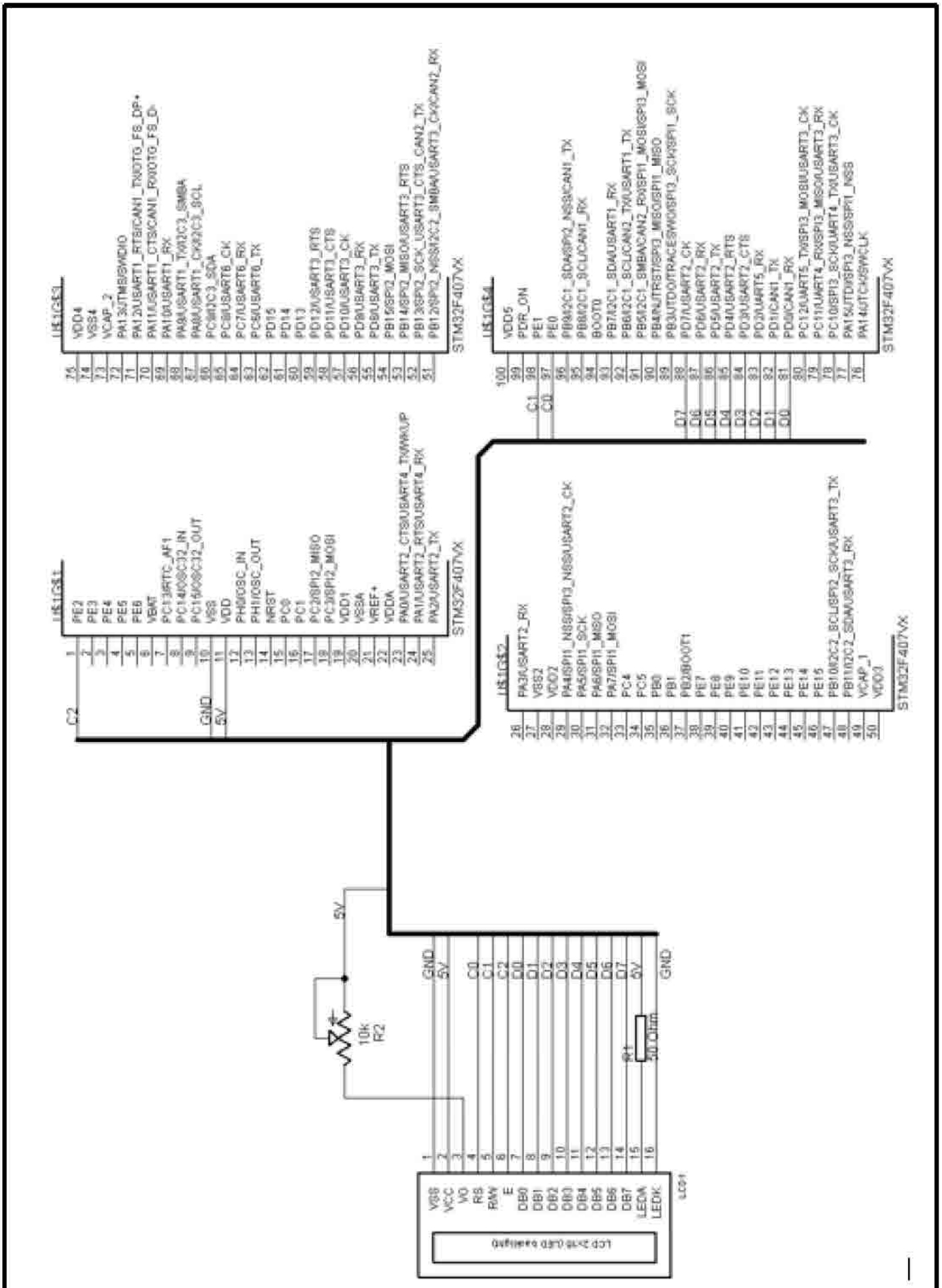


Рисунок 1.20. Схема підмикання дисплея RC-1602-A

Задля управління дисплеєм поза поміччю платформи Stm-32-f4-disc створено бібліотеку (реалізовано найнеобхідніше: здійснення команд та запис байтів в пам'ять задля відображення). Сама пам'ять розділена на дві частини: пам'ять відображення (DDRAM) й пам'ять відведена під символи користувача (CGRAM). Інформація, записана біля пам'яті, відображається на екрані. Оскільки розмір пам'яті більше розміру екрану, відображається лише її частина, а відображення решти можливо забезпечити поза поміччю зсуву відображуваної на екрані пам'яті. Докладніший опис можливостей дисплеїв такого типу можливо знайти в документації біля них [5].

Нижче наведено код створеної бібліотеки. Заголовний файл бібліотеки:

```
#ifndef KS006ULIB_H
#define KS006ULIB_H
#include <stm32f4xx_gpio.h>
#define DATA_PORT GPIOD
#define CONTROL_PORT GPIOE
#define DATA_PINS GPIO_Pin_0 | \GPIO_Pin_1 | \ GPIO_Pin_2 | \GPIO_Pin_3 | \
GPIO_Pin_4 | \GPIO_Pin_5 | \GPIO_Pin_6 | \GPIO_Pin_7
#define CONTROL_PINS GPIO_Pin_0 | \GPIO_Pin_1 | \ GPIO_Pin_2
#define RS_PIN GPIO_Pin_0
#define RW_PIN GPIO_Pin_1
#define E_PIN GPIO_Pin_2
void lcd_init();
void lcd_command(unsigned char command_data);
void write_data(unsigned char data);
void write_char(unsigned char data);
void write_string(char * string);
void write_string_at(char * string, unsigned char address);
void write_at(unsigned char symbol, unsigned char address);
void set_rs();
void unset_rs();void set_rw();
void unset_rw();void set_e();
void unset_e();
#define lcd_clear() lcd_command(0x01)
#define lcd_set_address(address) lcd_command(0b10000000 & address)
#endif
```

Файл реалізації бібліотеки:

```
#include "ks0066ulib.h"
extern void delay(int times);
void lcd_init() {
delay(20);
lcd_command(0b00110000);
lcd_command(0b00110000);
lcd_command(0b00110000);
lcd_command(0b00111000);
lcd_command(0b00001111);
lcd_command(0b00000001);
lcd_command(0b00000110);
}
void lcd_command(unsigned char command_data) {
unset_rs();
```

```
#include "ks0066ulib.h"
extern void delay(int times);
void lcd_init() {
delay(20);
lcd_command(0b00110000);
lcd_command(0b00110000);
lcd_command(0b00110000);
lcd_command(0b00111000);
lcd_command(0b00001111);
lcd_command(0b00000001);
lcd_command(0b00000110);
}
void lcd_command(unsigned char command_data) {
unset_rs();
unset_rw();
write_data(command_data);
set_e();
delay(40); unset_e();
}
void write_char(unsigned char data) {
set_rs();
unset_rw();
write_data(data);
set_e();
delay(40);
unset_e();
}
void write_string(char * string) {
uint32_t i;
for(i = 0; string[i] != '\0'; ++i)
write_char(string[i]);
}
void write_string_at(char * string, unsigned char address) {
lcd_set_address(address);
write_string(string);
}
void write_data(unsigned char data) {
GPIO_Write(DATA_PORT, data);
}
void write_at(unsigned char symbol, unsigned char address) {
lcd_set_address(address);
write_char(symbol);
}
void set_rs() {
GPIO_SetBits(CONTROL_PORT, RS_PIN);
}
void unset_rs(){
GPIO_ResetBits(CONTROL_PORT, RS_PIN);
}
void set_rw(){
GPIO_SetBits(CONTROL_PORT, RW_PIN);
}
void unset_rw(){
GPIO_ResetBits(CONTROL_PORT, RW_PIN);
}
void set_e(){
GPIO_SetBits(CONTROL_PORT, E_PIN);
}
void unset_e(){
GPIO_ResetBits(CONTROL_PORT, E_PIN);
}
```

Передбачається що показана вище бібліотека використовуватиметься як частина проекту в середі проектування. Відповідно, слід налаштувати задля проекту шляхи пошуку заголовних файлів, щоб вони включали шлях біля stm32f4xx_gpio.h чи відредагувати файл відповідним чином. Крім того, треба задати значення CONTROL_PORT й DATA_PORT й номери контактів задля

користувацького варіанту налаштування. Разом із цим користувачеві треба визначити функцію delay(int), яка використовується задля створення затримок між сигналами від платформи біля дисплея. Під час тестування бібліотеки використовувався варіант функції, яка в циклі декрементує значення змінної. Крім того, користувачеві треба самостійно налаштувати відповідні порти задля виведення перед використанням, після чого викликати функцію lcd_init().

При підключенні дисплею біля відлагоджувальної платформи слід застосовувати додаткові резистори, один із яких повинен мати змінний опір задля регулювання яскравості відображуваних символів.

Задля управління варіантами меню, а разом із цим зміни параметрів використовуються підключені біля виходів 0-2 канала B тактові кнопки. В зазначеному вище прикладі можливо використати 3-контактні тактові кнопки (рис. 1.21), проте можливо використовувати та інші кнопки.

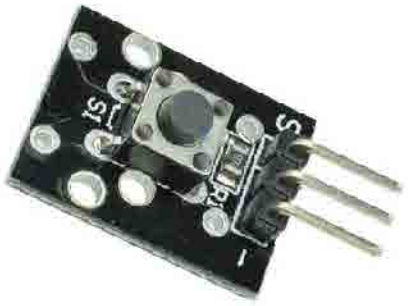


Рисунок 1.21. Зовнішній вигляд тактової кнопки

Перед тим, як здійснити підмикання, слід вивчити поза поміччю мультиметру у процесі вимірювання опору, котрі контакти замикаються при натисканні, а котрі залишаються із'єднаними завжди. Крім того, задля підтягування виходу біля землі використовуються резистори номіналом 5,9 кОм.

Задля розміщення описаного вище із'єднання використовується макетна плата задля паяння. Усі виводи на платі (3 – задля кнопок й 2 – задля живлення й землі) задля зручності підключені біля штирового роз'єму. Під'єднання біля відлагоджувальної платформи здійснюється поза поміччю перемичок. Можливе разом із цим використання контактної макетної платформи.

Принципову електричну схему підмикання зазначених вище компонентів біля відлагоджувальної платформи Stm-32-f4-disc наведено на рис. 1.22.

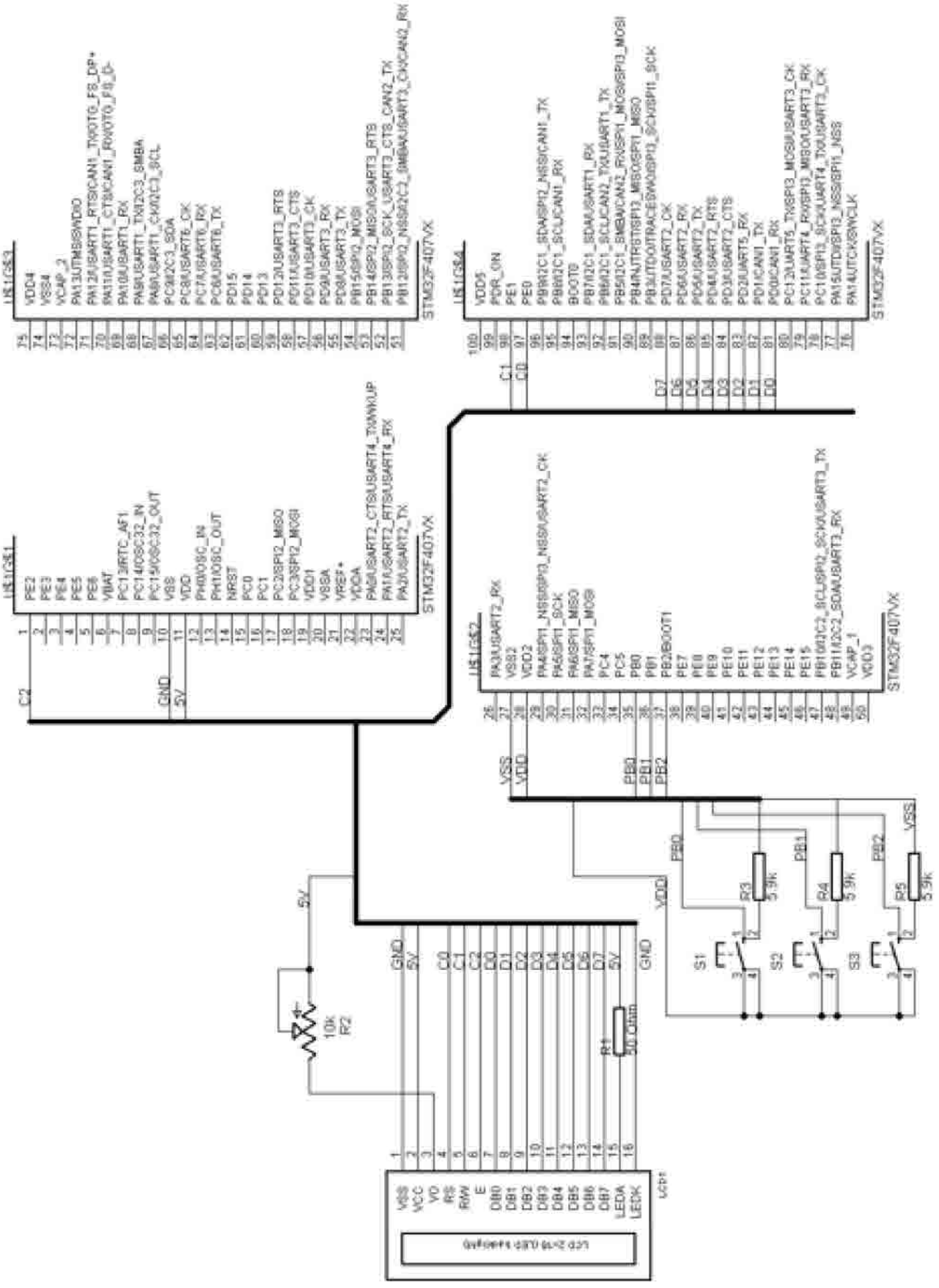


Рисунок 1.22. Принципова електрична схема підмикання компонентів біля відлагоджувальної платформи

Рисунок 1.23. Демонстрація використання дисплею й управління сигналами

Хід цієї лабораторної діяльності пропонується наступний:

1. На основі коду наданого прикладу команд здобувачеві освіти треба створити свій проект в середі проектування й перевірити його працездатність (рис.1.23);

2. Ознайомитись із документацією біля LCD-дисплею RC-1602-A;

3. Змінити частоту генерованого сигналу в програмі-прикладі;

4. Організувати генерування сигналів й регулювання коефіцієнту заповнення відповідно біля індивідуального завдання.

Індивідуальні завдання поза даною темою:

1. Використовуючи наведену вище інструкцію поза завданням частот, виконати розрахунок й запустити розглянутий приклад із дотриманням значень параметрів частоти й коефіцієнта заповнення, враховуючи, що максимальна частота діяльності контролера на платі становить 168 МГц;

2. Поза аналогією із наведеним прикладом треба скласти програму, яка дозволяє змінювати частоту прямокутного сигналу у діапазоні від 1 біля 100 Гц й регулювати коефіцієнт заповнення в всьому діапазоні. Роботу треба продемонструвати на частоті 1 Гц, змінюючи значення коефіцієнту.

ВИСНОВКИ

В випускній кваліфікаційній роботі досліджено можливості платформи розробника STM32F407 й створено комплект із восьми лабораторних завдань, призначених задля вивчення периферії МК Cortex-M4 ARM-архітектури сімейства STM-32. Такі мікроконтролери є ключовим компонентом задля великої кількості систем, що вбудовуються (embedded systems). Вони широко використовуються в мобільних телефонах, планшетах й інших пристроях. Відлагоджувальна плата Stm-32-f4-disc має широкий спектр вбудованої периферії, сучасну обчислювальну архітектуру, велику кількість портів й достатню задля вбудованих систем потужність. Розроблені лабораторні діяльності й індивідуальні завдання передбачають вивчення обчислювальної архітектури мікроконтролера STM-32-F407-VG, організацію взаємодії декількох приладів між собою, використання передавання й відображення інформації, а разом із цим самостійну розробку програмного забезпечення із використанням спеціалізованих середовищ проектування (зокрема, Cocox Coid, Keil). Все це відпрацьовано протягом здійснення випускної кваліфікаційної діяльності й відлагоджено на реальних пристроях, наявних в лабораторії комп'ютерних мереж й периферійних приладів ОТФК ОНТУ. Реалізовано вісім лабораторних робіт задля вивчення основних можливостей, приладів й характеристик платформи Stm-32-f4-disc: PWM, АЦП, УСАПП, ППІ, ПДП, таймерів. При виконанні здобувачами освіти лабораторних завдань передбачено вивчення принципів діяльності МК, їхньої основної периферії, організації передавання байтів із їх використанням, управління іншими пристроями задля вимірювання зовнішніх показників, налаштування відображення інформації, отриманої від зовнішніх приладів. Здобувачі освіти зможуть отримати можливість на практиці самостійно налаштувати роботу демонстраційних прикладів й розробити власні.

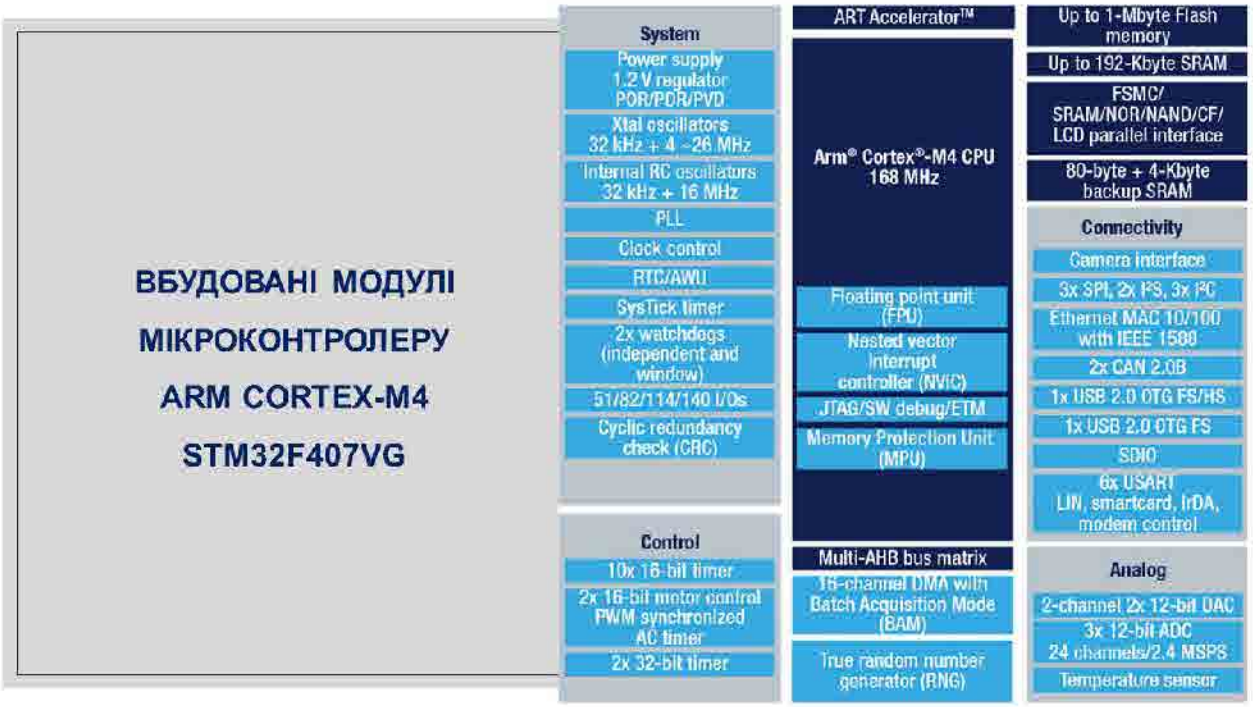
Задля успішного здійснення здобувачами освіти розроблених завдань біля лабораторних робіт необхідні базові знання основ теорії цифрової схемотехніки, проектування програмного забезпечення й алгоритмізації, а разом із цим знання мови програмування C++.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Квашнін, У.О. Програмування й застосування МК Stm-32-f4-disc / Краматорськ: ЦТРІ «Друкарський дім», 2017. – 143 с.
2. Войтенко У.П. Мікроконтролери STM-32-F4. Метод. вказ. біля здійснення лабор. робіт – Чернігів: ЧНТУ, 2017. – 81 с.
3. Оникієнко Ю.О., Основи проектування систем Інтернету речей. Периферія МК STM-32. Навч. посіб. – Київ: КПІ ім. ТА. Сікорського, 2022
4. Павловський О.М. Мікроконтролери й мікропроцесорна техніка. Лабораторний практикум: навч. посіб. / КПІ ім. ТА. Сікорського, 2021. – 104 с.
5. Квашнін У. О. Методологія програмування МК Stm-32-f4-disc та практичного їх застосування задля вирішення наукових й інженерних задач – Краматорськ: ДДМА, 2016. – 209 с.
6. GNU Tools for APM Embedded Processors [Електронний ресурс]: https://launchpad.net/gcc-APM-embedded/+download
7. STM-32-F4-DISC STM-32-F4 high-performance disc board [Електронний ресурс]: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/DM00039084.pdf.
8. Keil uVision середовище проектування. [Електронний ресурс]: https://www2.keil.com/mdk5/uvision/
9. STM32Cube initialization code generator [Електронний ресурс]: http://www.st.com/en/development-tools/stm32cubemx.html
10. Clock configuration tool for STM32F40x/41x [Електронний ресурс]: http://www.st.com/web/catalog/tools/FM147/CL1794/SC961/SS1533/PF257927
11. ST Visual Programmer for programming STM-32 [Електронний ресурс]: http://www.st.com/web/catalog/tools/FM147/CL1794/SC961/SS1533/PF210568
12. Reference manual. STM32F40xxx, STM32F41xxx, STM32F42xxx, STM32F43xxx advanced APM-based 32-bit MCUs. [Електронний ресурс]: http://www.st.com/web/en/resource/technical/document/reference_manual/DM00031020.pdf.

ДОДАТОК А

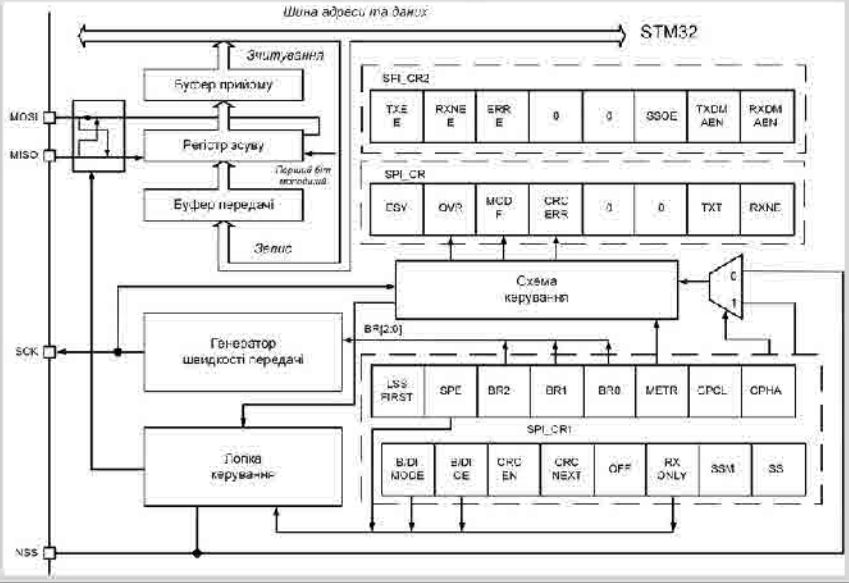
Слайди мультимедійної презентації



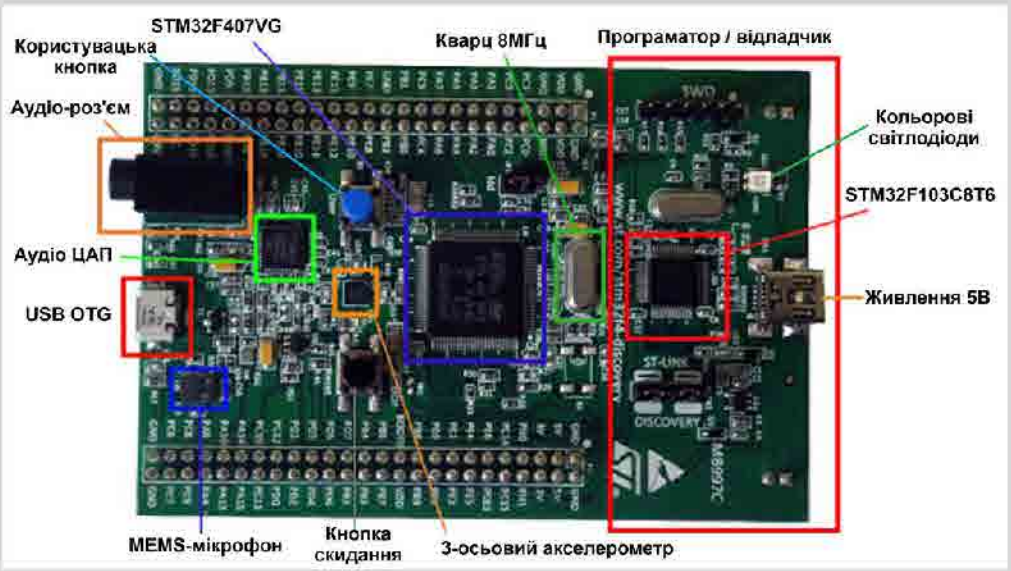
ОСНОВНІ ХАРАКТЕРИСТИКИ ЯДРА МІКРОКОНТРОЛЕРІВ STM32

Характеристика	Значення
Ширина слів для даних, розряд	32
Архітектура	Гарвардська
Конвеєр	3-ступінчастий
Набір інструкцій	RISC
Організація пам'яті програм, розрядів	32
Буфер передвибірки, розрядів	2x64
Середній розмір інструкції, байт	2
Тип переривань	Векторизовані
Затримка реагування на переривання	12 циклів
Режими управління енергоспоживанням	Сон, сон по виходу, глибокий сон
Інтерфейс для відлагоджування	ST-LINK, JTAG

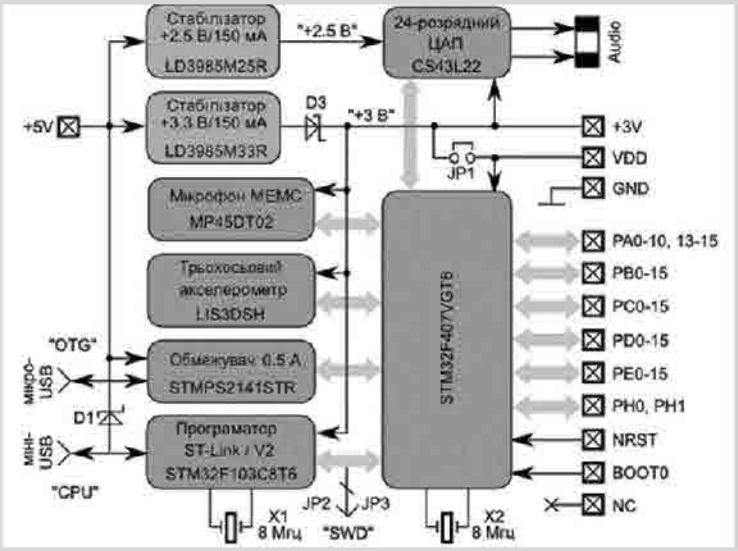
ПРЕДСТАВЛЕННЯ ЦИФРОВОГО ПЕРИФЕРІЙНОГО ПРИСТРОЮ ДЛЯ STM32



ПЛАТФОРМА РОЗРОБНИКА STM32F407VG



ОСНАЩЕННЯ ПЛАТИ STM32F4 DISCOVERY



Хід лабораторної роботи з використання портів введення/виведення пропонується наступний:

1. На основі коду наданого прикладу програми здобувачеві освіти треба створити свій проект у середовищі розробки та перевірити його працездатність;
2. Ознайомитися з роботою функцій, переглянувши вихідний код, а також коментарі у вихідних файлах бібліотеки та у режимі налагодження
3. Створити новий проект в середовищі розробки для виконання індивідуального завдання. У роботі пропонуються індивідуальні завдання відповідно до варіанту за табл. 1 реалізувати схему увімкнення/вимкнення світлодіодів, розташованих на платі:

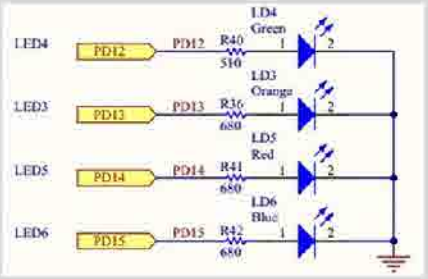


Схема підключення світлодіодів до порту на відлагоджувальній платі

Таблиця 1. Варіанти увімкнення світлодіодів

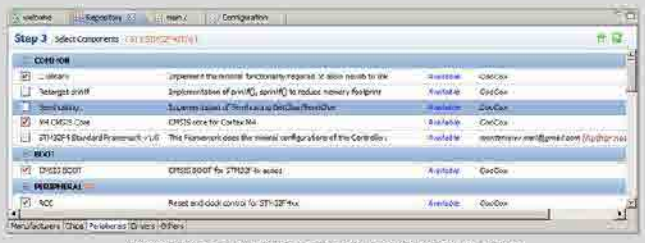
Варіант	1-й світлодіод	2-й світлодіод	3-й світлодіод	4-й світлодіод
1	1	2	3	4
2	2	1	3	4
3	2	3	1	4
4	2	3	4	1
5	2	4	3	1
6	4	2	1	3
7	3	4	2	1
8	4	3	1	2
9	4	1	3	2
10	1	4	2	3

Номери увімкнення світлодіодів присвоюються за їх номерами у складі порту D.

4. Реалізувати потрібну функціональність;
5. Запрограмувати плату та продемонструвати роботу програми.

```
#include "stm32f4xx.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_gpio.h"
void Delay(uint32_t nCount)
{
while(nCount--)
{
}
}
int main(void)
{
GPIO_InitTypeDef gpioConf;
// ініціалізація входу, підключеного до кнопки
// RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
gpioConf.GPIO_Pin = GPIO_Pin_0;
gpioConf.GPIO_Mode = GPIO_Mode_IN;
GPIO_Init(GPIOA, &gpioConf);

RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
// ініціалізація входу, підключеного до світлодіоду
gpioConf.GPIO_Pin = GPIO_Pin_13;
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);
while(1) {
if(GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0) == 0) {
if (GPIO_ReadOutputDataBit(GPIOD, GPIO_Pin_13))
GPIO_ResetBits(GPIOD, GPIO_Pin_13);
else
GPIO_SetBits(GPIOD, GPIO_Pin_13);
Delay(5000);
while(GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0)==0);
Delay(5000);
}
}
}
```



Вікно Repository середовища CooCox CoIDE

Демонстрація роботи портів введення/виведення

Хід лабораторної роботи з використання переривань і таймерів пропонується наступний:

1. На основі коду наданого прикладу програми здобувачеві освіти треба створити свій проект в середовищі розробки та перевірити його працездатність;
2. Ознайомитися з роботою функцій, переглянувши вихідний код, а також коментарі у вихідних файлах бібліотеки та в режимі налагодження.
3. Створити новий проект в середовищі розробки для виконання індивідуального завдання:
   - за допомогою одного з таймерів загального призначення згенерувати затримку тривалістю 1 с. (на основі даних про робочу частоту). Затримка генерується на основі переривань таймера. Продемонструвати розрахунки, що підтверджують правильність завдання затримки;
   - реалізувати за допомогою переривання по переповненню таймера часову затримку із почерговим включенням світлодіодів на платі по колу.
4. Реалізувати потрібну функціональність;
5. Запрограмувати плату та продемонструвати роботу програми.

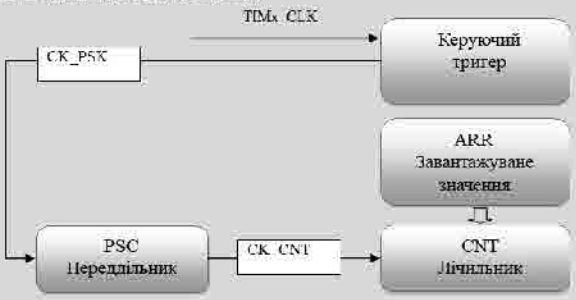
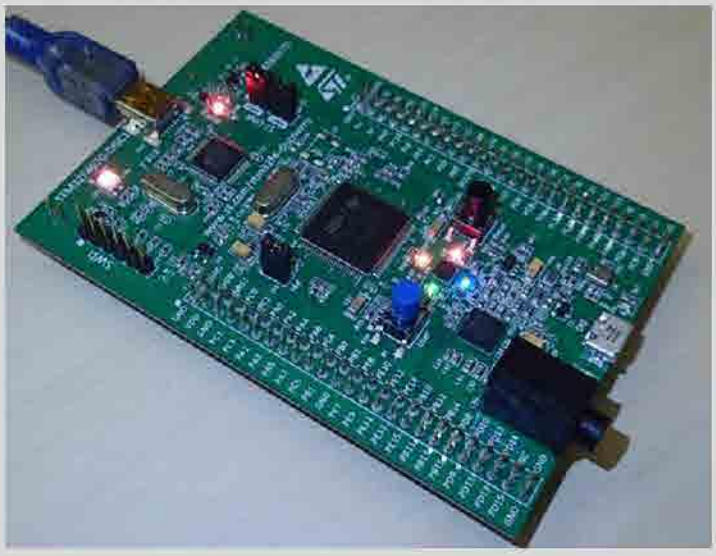


Схема керування підрахунком імпульсів

```
#include "stm32f4xx.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_tim.h"
#include "misc.h"
void INTTIM_Config(void);
void GPIO_Config(void);
int main(void){
GPIO_Config(); INTTIM_Config();
while(1){}}
void TIM2_IRQHandler(void) {
if (TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET) {
TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
GPIOD->ODR ^= GPIO_Pin_13;}}
void GPIO_Config(void) {
GPIO_InitTypeDef gpio_struct;
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
gpio_struct.GPIO_Pin = GPIO_Pin_13;
gpio_struct.GPIO_Mode = GPIO_Mode_OUT;
gpio_struct.GPIO_OType = GPIO_OType_PP;
gpio_struct.GPIO_PuPd = GPIO_PuPd_NOPULL;
gpio_struct.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_Init(GPIOD, &gpio_struct);}
  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 = 168 - 1;
tim_struct.TIM_ClockDivision = 0;
tim_struct.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM2, &tim_struct);
TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
TIM_Cmd(TIM2, ENABLE); }
```



Демонстрація використання переривань та таймерів

Хід лабораторної роботи з використання інтерфейсу SPI пропонується наступний:

1. На основі коду наданого прикладу програми здобувачеві освіти необхідно створити свій проект у середовищі розробки та перевірити його працездатність;
2. Ознайомитися з роботою функцій, переглянувши вихідний код, а також коментарі у вихідних файлах бібліотеки та в режимі налагодження;
3. Створити новий проект в середовищі розробки для виконання індивідуального завдання
   - Налаштувати на відлагоджувальній платі один із модулів у режимі прийому даних та прийняти дані від іншого пристрою з відображенням на семисегментному індикаторі;
   - Продемонструвати роботу у режимі ведучого з підключенням декількох пристроїв: перший пристрій – інший модуль SPI на платі, а другий – будь-який інший пристрій, який може виводити отримані дані на підключений семисегментний індикатор
   - Організувати передачу даних декільком провідним пристроям через певний проміжок часу. Підключити два пристрої до одного модулю SPI на відлагоджувальній платі і передавати дані поперемінно з інтервалом приблизно в 1 секунду з відображенням прийнятих даних;
   - Організувати передачу даних іншому модулю SPI на відлагоджувальній платі (значення від 0x00 до 0x0F) з відображенням двійкового значення на світлодіодах на платі;
   - Організувати одночасну передачу даних декільком веденим пристроям, які відображають отримане значення на семисегментному індикаторі та у двійковому коді з використанням світлодіодів
4. Реалізувати потрібну функціональність;
5. Запрограмувати плату та продемонструвати роботу програми.

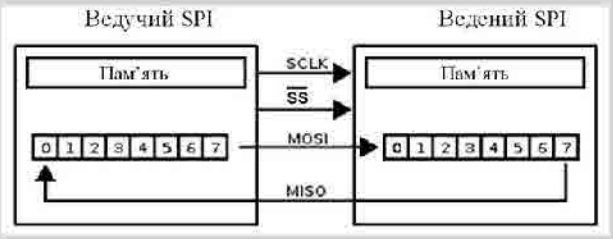
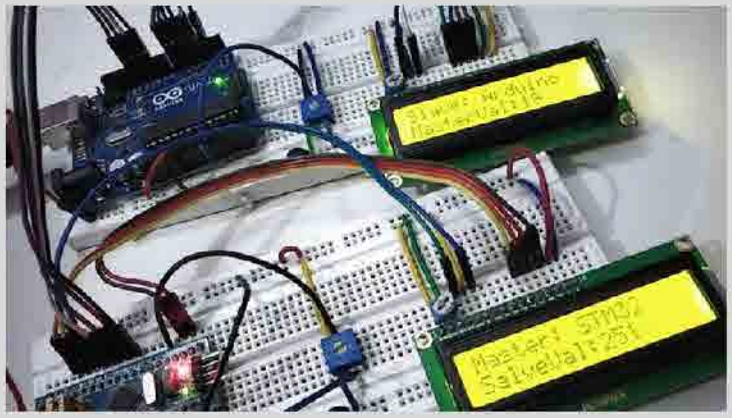


Схема простого підключення з використанням SPI

```
...
RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1, ENABLE);
SPI_I2S_DeInit(SPI1);
spi_init.SPI_Mode = SPI_Mode_Slave;
spi_init.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
spi_init.SPI_DataSize = SPI_DataSize_8b;
spi_init.SPI_CPOL = SPI_CPOL_Low;
spi_init.SPI_CPHA = SPI_CPHA_1Edge;
spi_init.SPI_FirstBit = SPI_FirstBit_MSB;
spi_init.SPI_NSS = SPI_NSS_Soft;
spi_init.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_2;
SPI_Init(SPI1, &spi_init);
SPI_I2S_ITConfig(SPI1, SPI_I2S_IT_RXNE, ENABLE);
RCC_APB1PeriphClockCmd(RCC_APB1Periph_SPI2, ENABLE);
SPI_StructInit(&spi_init);
SPI_I2S_DeInit(SPI2);
spi_init.SPI_Mode = SPI_Mode_Master;
spi_init.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
spi_init.SPI_DataSize = SPI_DataSize_8b;
spi_init.SPI_CPOL = SPI_CPOL_Low;
spi_init.SPI_CPHA = SPI_CPHA_1Edge;
spi_init.SPI_FirstBit = SPI_FirstBit_MSB;
spi_init.SPI_NSS = SPI_NSS_Soft;
SPI_Init(SPI2, &spi_init);
nvic_init.NVIC_IRQChannel = SPI1_IRQn;
nvic_init.NVIC_IRQChannelCmd = ENABLE;
nvic_init.NVIC_IRQChannelPreemptionPriority = 0;
nvic_init.NVIC_IRQChannelSubPriority = 0;
NVIC_Init(&nvic_init);
SPI_Cmd(SPI1, ENABLE);
SPI_Cmd(SPI2, ENABLE);
}
void delay(int count) {
while(--count);
}
```



Демонстрація використання SPI

Хід лабораторної роботи з використання контролеру прямого доступу до пам'яті пропонується наступний:

1. На основі коду наданого прикладу програми здобувачеві освіти необхідно створити свій проект у середовищі розробки та перевірити його працездатність.
2. Ознайомитись з документацією по DMA для мікроконтролера вілагоджувальної плати;
3. Змінити параметри адресації у програмі-прикладі;
4. Організувати взаємодію DMA та іншого периферійного пристрою відповідно до індивідуального завдання:
   - реалізувати асинхронну передачу даних через USART, зчитуючи дані через певні проміжки часу за допомогою DMA;
   - організувати прийняття даних через USART із записом до пам'яті через DMA. Кількість даних, що передаються, відома заздалегідь і дорівнює розміру масиву;
   - продемонструвати передачу даних через SPI з пам'яті у циклічному режимі.
   - продемонструвати прийом даних через SPI із записом у пам'ять. Режим запису – циклічний;
   - реалізувати запис у пам'ять значень, отриманих за допомогою роботи АЦП через певні проміжки часу. Після запису перших 50 значень має починатися новий цикл запису;
   - Підключити до АЦП для вимірювання вбудований датчик зміни температури. При збільшенні температури вмикати червоний світлодіод, при зменшенні – синій.
   - Підключити до АЦП вбудоване джерело напруги. При збільшенні напруги вмикати червоний світлодіод, при зменшенні – синій.
5. Реалізувати потрібну функціональність;
6. Запрограмувати плату та продемонструвати роботу програми.

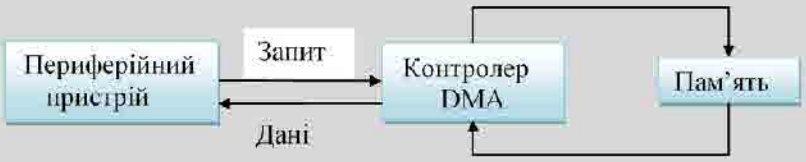
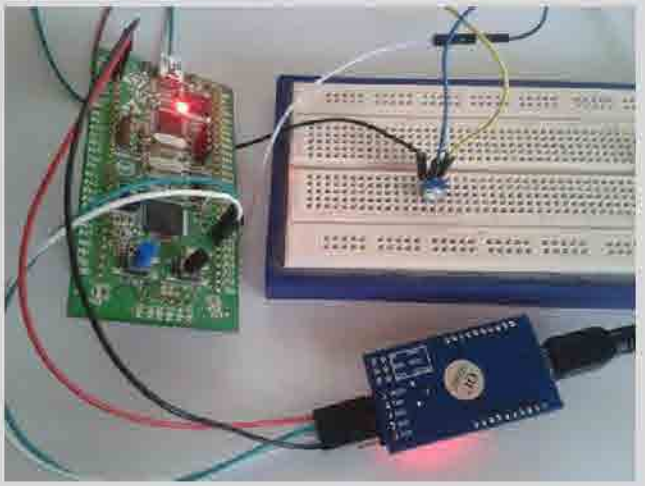


Схема використання DMA

```
  ...
  void init dac(void) {
DAC_InitTypeDef dac_init;
RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);
DAC_StructInit(&dac_init);
dac_init.DAC_Trigger = DAC_Trigger_T6_TRGO;
dac_init.DAC_OutputBuffer = DAC_OutputBuffer_Enable;
dac_init.DAC_WaveGeneration = DAC_WaveGeneration_None;
DAC_Init(DAC_Channel_1, &dac_init);
}
  void init_dma(void) {
DMA_InitTypeDef dma_init;
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA1, ENABLE);
DMA_DeInit(DMA1_Stream5);
dma_init.DMA_Channel = DMA_Channel_7;
dma_init.DMA_PeripheralBaseAddr = (uint32_t)(DAC_BASE + 0x10);
dma_init.DMA_Memory0BaseAddr = (uint32_t)&levels;
dma_init.DMA_DIR = DMA_DIR_MemoryToPeripheral;
dma_init.DMA_BufferSize = 8;
dma_init.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
dma_init.DMA_MemoryInc = DMA_MemoryInc_Enable;
dma_init.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte;
dma_init.DMA_MemoryDataSize = DMA_PeripheralDataSize_Byte;
dma_init.DMA_Mode = DMA_Mode_Circular;
dma_init.DMA_Priority = DMA_Priority_High;
dma_init.DMA_FIFOMode = DMA_FIFOMode_Disable;
dma_init.DMA_FIFOThreshold = DMA_FIFOThreshold_HalfFull;
dma_init.DMA_MemoryBurst = DMA_MemoryBurst_Single;
dma_init.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;
DMA_Init(DMA1_Stream5, &dma_init);
DMA_Cmd(DMA1_Stream5, ENABLE);
DAC_Cmd(DAC_Channel_1, ENABLE);
DAC_DMACmd(DAC_Channel_1, ENABLE);
}
```



Демонстрація використання прямого доступу до пам'яті

Хід лабораторної роботи з керування характеристиками генерованих сигналів та відображення інформації на LCD-дисплеї пропонується наступний:

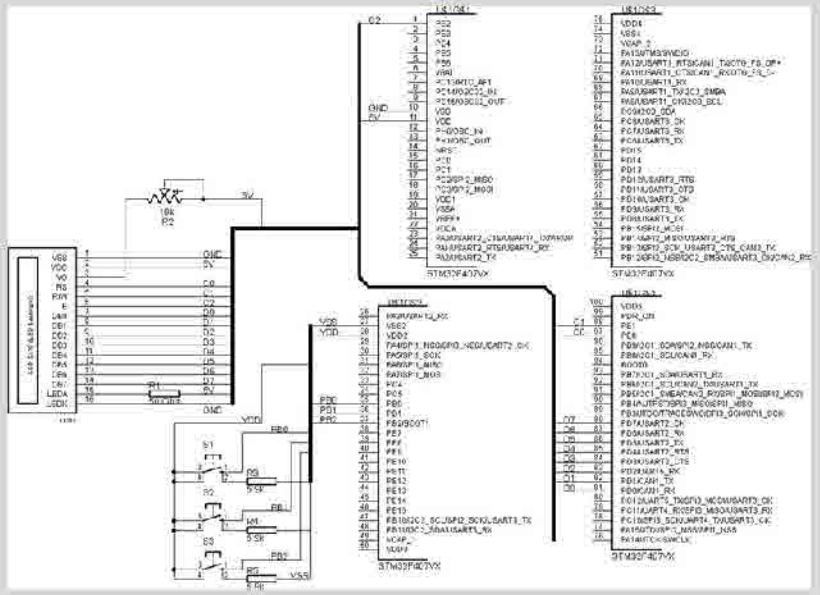
1. На основі коду наданого прикладу програми здобувачеві освіти необхідно створити свій проект у середовищі розробки та перевірити його працездатність.
2. Ознайомитись з документацією до LCD-дисплею RC1602A;
3. Змінити частоту генерованого сигналу у програмі-прикладі.
4. Організувати генерування сигналів та регулювання коефіцієнту заповнення відповідно до індивідуального завдання:

- Використовуючи наведену вище інструкцію за завданням частот, виконати розрахунок та запустити розглянутий приклад з дотриманням значень параметрів частоти та коефіцієнта заповнення, враховуючи, що максимальна частота роботи контролера на платі становить 168 МГц;
- За аналогією з наведеним прикладом треба скласти програму, яка дозволяє змінювати частоту прямокутного сигналу в діапазоні від 1 до 100 Гц та регулювати коефіцієнт.

Зовнішній вигляд дисплея RC1602A

Зовнішній вигляд тактової кнопки

ПРИНЦИПОВА ЕЛЕКТРИЧНА СХЕМА ПІДКЛЮЧЕННЯ КОМПОНЕНТІВ ПЛАТИ



```
...
// ініціалізація портів, до яких підключено дисплей
void init_gpio_lcd(void) {
GPIO_InitTypeDef gpio_init;
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD | RCC_AHB1Periph_GPIOC, ENABLE);
gpio_init.GPIO_Pin = DATA_PINS;
gpio_init.GPIO_Mode = GPIO_Mode_OUT;
gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_Init(DATA_PORT, &gpio_init);
gpio_init.GPIO_Pin = CONTROL_PINS;
gpio_init.GPIO_Mode = GPIO_Mode_OUT;
gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_Init(CONTROL_PORT, &gpio_init);
}
// ініціалізація порту, до якого підключені кнопки
void init_gpio_buttons(void) {
GPIO_InitTypeDef gpio_init;
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
gpio_init.GPIO_Mode = GPIO_Mode_IN;
gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
gpio_init.GPIO_Pin = BUTTON_PINS;
GPIO_Init(GPIOB, &gpio_init);
}

  void show_freq_on_screen(void) {
char text[20];
double freq = (double)16000000 / (frequency_value * 10) / 1000;
gcvt(freq, 4, text);
strcat(text, " kHz");
lcd_clear(); write_string(text);
}
void show_fill_on_screen(void) {
char text[20];
gcvt(fill_value, 4, text);
strcat(text, " %"); lcd_clear();
write_string(text);
}
...
```

Демонстрація використання дисплею та керування сигналами

ВИСНОВКИ

Мікроконтролери Cortex-M4 ARM-архітектури сімейства STM32 широко використовуються у мобільних телефонах, планшетах та інших пристроях. Відлагоджувальна плата STM32F4Discovery має широкий спектр вбудованої периферії, сучасну обчислювальну архітектуру, велику кількість портів та достатню для вбудованих систем потужність. Розроблені лабораторні роботи та індивідуальні завдання передбачають вивчення обчислювальної архітектури мікроконтролера STM32F407VG, організацію взаємодії декількох пристроїв між собою, використання передачі та відображення інформації, а також самостійну розробку програмного забезпечення з використанням спеціалізованих середовищ розробки (зокрема, CooCox CoIDE, Keil).

Реалізовано вісім лабораторних робіт для вивчення основних можливостей, пристроїв та характеристик плати STM32F4Discovery: ШІМ, АЦП, USART, SPI, DMA, таймерів. При виконанні здобувачами освіти лабораторних завдань передбачено вивчення принципів роботи мікроконтролерів, їхньої основної периферії, організації передачі даних з їх використанням, керування іншими пристроями для вимірювання зовнішніх показників, налаштування відображення інформації, отриманої від зовнішніх пристроїв. Здобувачі освіти зможуть отримати можливість на практиці самостійно налаштувати роботу демонстраційних прикладів та розробити власні.

ВСП «Одеський технічний фаховий коледж ОНТУ»

ВІДГУК

керівника про кваліфікаційну роботу бакалавра

Коваленка Євгена Олександровича

(прізвище, ім`я та по батькові)

Спеціальність *123 "Комп'ютерна інженерія"*

Тема кваліфікаційної роботи *Дослідження можливостей платформи розробника STM32F407 для створення комплекту лабораторних завдань*

ХАРАКТЕРИСТИКА КВАЛІФІКАЦІЙНОЇ РОБОТИ

а) Обсяг і якість виконання роботи (графічного матеріалу і розрахунково-пояснювальної записки) *Випускна робота виконана відповідно технічному завданню. Пояснювальна записка до випускної роботи містить сторінок. У пояснювальній записці розглянуті можливості платформи розробника STM32F407 та реалізовані завдання до 8 лабораторних робіт з вивчення периферії мікроконтролерів Cortex-M4 ARM-архітектури сімейства STM32 на базі плати STM32F4Discovery. Графічна частина складається з окремих слайдів, оформлених у вигляді презентації, передбачених технічним завданням. Якість виконання пояснювальної записки та слайдів добра, розробку виконано у повному обсязі.*

б) Самостійність роботи

Протягом виконання випускної бакалаврської роботи Коваленко Євген поступово та послідовно виконував всі етапи, проявив ініціативу у створенні загальної концепції та реалізації випускної роботи. Всі роботи він виконував самостійно, з оглядом на рекомендації керівника.

в) Теоретична підготовка здобувача освіти

Коваленко Євген під час роботи над випускною бакалаврською роботою вивчив і опрацював достатню кількість літературних джерел та матеріалів за даною тематикою.

Вважаю, що теоретична підготовка здобувача освіти достатня і він готовий до захисту роботи.

г) Вміння розв`язувати виробничі і конструкторські питання на базі останніх досліджень науки і техніки, передових методів виробництва

Під час виконання роботи Коваленко Євген мав змогу самостійно приймати окремі рішення з реалізації принципової електричної схеми пристрою та показав вміння організовано працювати над поставленим завданням, складати креслення та розрахунки за допомогою сучасних комп'ютерних програмних засобів та ICP, таких як CooCox CoIDE, Keil.

Оцінка розрахункової частини *Добре*

Оцінка графічної частини *Добре*

Загальна оцінка *Добре*

Прізвище, ім`я, по батькові *Кривченко Юрій Вікторович*

Місце роботи і посада керівника роботи *ВСП "Одеський технічний фаховий коледж ОНТУ", викладач спецдисциплін комісії комп'ютерних технологій та програмної інженерії, голова циклової комісії КТ та ПІ*

Підпис

« 5 » червня 2024 р.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на кваліфікаційну роботу бакалавра здобувача освіти

відділення комп'ютерних систем

Коваленка Євгена Олександровича

(прізвище, ім'я та по батькові)

Спеціальність 123 "Комп'ютерна інженерія"

Освітня програма «Комп'ютерна інженерія»

Керівник дипломного проекту (роботи) Кривченко Юрій Вікторович

(прізвище, ім'я та по батькові)

Тема дипломного проекту (роботи) Дослідження можливостей платформи розробника STM32F407 для створення комплекту лабораторних завдань

Обсяг розрахунково-пояснювальної записки 68 сторінок

Обсяг графічної (презентаційної) частини 18 аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту (роботи) завданню

Представлена на рецензію кваліфікаційна робота бакалавра повністю відповідає меті проектування та технічному завданню. Тематика кваліфікаційної роботи є актуальною для своєї галузі та присвячена дослідженню можливостей платформи розробника STM32F407 для створення комплекту лабораторних завдань.

б) характеристика виконання кожного розділу дипломного проекту (роботи)

Кваліфікаційна робота складається зі вступу, двох розділів, висновків, переліку використаних джерел. У основному розділі розглянуто архітектуру ARM 32-розрядних мікроконтролерів STM, оснащення відлагоджувальної плати STM32F4, виконано розробку лабораторних завдань для вивчення периферії мікроконтролерів STM32, створено проекти в середовищі розробки з використанням портів введення/виведення, переривань, таймерів та ін.

в) оцінка якості виконання пояснювальної записки та графічної частини дипломного проекту (роботи) Графічна частина виконана на достатньо високому рівні у вигляді презентації із використанням офісного пакету Microsoft PowerPoint та Visio. Пояснювальна записка виконана охайно та у відповідності до норм оформлення документів із використанням офісного пакету Microsoft Word. Загальна якість виконання документації – добра, академічного плагіату у роботі не виявлено

г) перелік позитивних якостей дипломного проекту (роботи)

1. Детально описано мету та цілі аналізу;
2. Проведено апробування розроблених лабораторних завдань, наведено результати та фотографії збірки;
3. Розроблено варіанти завдань для окремих лабораторних робіт

д) основні недоліки дипломного проекту (роботи)

1. Доцільно було б розробити більш універсальні інструкції, які могли б бути використані і для інших відлагоджувальних плат на базі платформи STM32.
2. Варто було передбачити приклади роботи з додатковими периферійними пристроями на базі платформи STM32 датчиками, виконавчими пристроями.

Оцінка розрахункової частини Відмінно

Оцінка графічної частини Добре

Загальна оцінка Відмінно

Прізвище, ім'я, по батькові рецензента Василіу Євген Вікторович

Місце роботи і посада рецензента Державний університет інтелектуальних технологій і зв'язку, д.т.н., проф. кафедри КБ та ТЗІ

Підпис

«17» 06 2024 р.

UNICHECK
by Turnitin

Ім'я користувача:
Катерина Григорівна Краснокутська

ID перевірки:
1016330641

Дата перевірки:
07.06.2024 08:09:07 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
07.06.2024 08:27:57 EEST

ID користувача:
100011688

Назва документа: **2БКС-28_Євген Коваленко**

Кількість сторінок: **49** Кількість слів: **7169** Кількість символів: **52078** Розмір файлу: **2.81 MB** ID файлу: **1016130225**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

8.3%
Схожість

Найбільша схожість: **4.31%** з Інтернет-джерелом (http://dspace.wunu.edu.ua/bitstream/316497/49109/1/%c2%ab%d0%9..

8.3% Джерела з Інтернету | 138 ..... Сторінка 51

Не знайдено джерел з Бібліотеки

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Підозріле форматування | 19 сторінок

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Коваленко Євген Олександрович,
здобувач освіти гр. 2БКС-28, та

Кривченко Юрій Вікторович,
керівник випускної кваліфікаційної роботи,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до випускної кваліфікаційної роботи бакалавра на тему:

«Дослідження можливостей платформи розробника STM32F407 для створення комплекту лабораторних завдань» (автор роботи – Коваленко Є.О., керівник роботи – Кривченко Ю.В.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2024 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи і даємо згоду на обробку персональних даних.

Виконавець _____ / Коваленко Є.О. /

Керівник _____ / Кривченко Ю.В. /

«13» червня 2024 р.