МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність:

123 – «Комп'ютерна інженерія»

Освітня програма:

«Обслуговування комп'ютерних систем і мереж»

Група: 4КС-55

Дипломний проект

здобувача освіти денної форми навчання
КС 55.02.000.00 ДП ПЗ

АРТЮХОВА
МАКСИМА ВОЛОДИМИРОВИЧА

м. Одеса
2022 р.

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

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

Освітня програма: **«Обслуговування комп'ютерних систем і мереж»**

Група: **4КС-55**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту (роботи) на тему:

Розробка лабораторних завдань для вивчення периферії мікроконтролерів STM32 з дисципліни "Периферійні пристрої"

Проектний матеріал складається з пояснювальної записки на 64 сторінках та графічного (презентаційного) матеріалу на 17 аркушах (слайдах).

Дипломник _____ (Артюхов М.В.)

Керівник _____ (Кривченко А.А.)

Консультанти:

з економічної частини _____ (Копайгородська Т.Г.)

з охорони праці_____ (Чорновол Н.І.)

з дотримання вимог ЄСКД _____ (Петрашова В.І.)

старший консультант _____ (Скорнякова О.В.)

До захисту допущений

Голова циклової комісії _____ (Скорнякова О.В.)

Завідувач відділення _____ (Суліма Ю.Ю.)

Захист «____»_____ 2022 р. Протокол ДКК №_____

Оцінка ДКК _____

Секретар ДКК _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення **комп'ютерних систем** Комісія **КТ та ПІ**

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

Освітня програма **«Обслуговування комп'ютерних систем і мереж»**

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР_____

Беркань І.В.

"_____"_____2022 р.

З А В Д А Н Н Я

на дипломний проект (роботу)

Здобувачеві (здобувачці) освіти Артюхову Максиму Володимировичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Розробка лабораторних завдань для вивчення периферії мікроконтролерів STM32 з дисципліни "Периферійні пристрої"

затверджена наказом по коледжу від "____"_____202___р. №_____

2. Термін здачі закінченого проекту (роботи) _____

3. Вихідні данні до проекту (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 Toolchain for ARM Embedded Processors, бібліотека CMSIS.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно 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 (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів)
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 проекту (роботі), із зазначенням розділів проекту, що їх стосується

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1.Технологічний	Кривченко А.А.		
розділ			
2.Екон. частина	Копайгородська Т.Г.		
3.Охорона праці	Чорновол Н.І.		
Нормоконтроль	Петрашова В.І.		

7. Дата видачі завдання _____

Керівник Кривченко А.А. _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проекту (роботи)	Термін виконання етапів дипломного проекту (роботи)	Відмітка про виконання
1.	Вступ. Постановка задачі проектування		
2.	Огляд архітектури 32-розрядних мікроконтролерів STM		
3.	Вивчення та опис оснащення відлагоджувальної плати		
	STM32F4 Discovery		
4.	Створення проекту в середовищі розробки CooCox CoIDE		
5.	Створення проекту в середовищі розробки Keil		
6.	Використання портів мікроконтролеру STM32F407VG		
7.	Використання переривань мікроконтролеру STM		
8.	Використання таймерів мікроконтролеру STM32F407VG		
9.	Генерування сигналу ШІМ		
10.	Використання АЦП на платі STM32F4 Discovery		
11.	Використання USART та інтерфейсу SPI, ПДП		
12.	Економічні розрахунки		
13.	Виконання розділу охорони праці та ТБ		
14.	Виконання графічної частини проекту		
15.	Підготовка проекту до захисту		

Дипломник _____
(підпис)

Керівник _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			КС 55. 02 000. 00 ДП	Дипломний проект		
А4			КС 55. 02 000. 00 ДП ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	КС 55. 02 000. 00 ДП		
Розробив		Артюхов М.В.			Розробка лабораторних завдань для вивчення периферії мікроконтролерів STM32 з дисципліни "Периферійні пристрої"	Літ.	Аркуш 4 / Аркушів 64
Перевірив		Кривченко А.А.				Н Д П	
Н. Контр.		Петрашова В.І.				ВСП "ОТФК ОНТУ" гр.4КС-55	
Затверд.		Скорнякова О.В.					

ЗМІСТ

ВСТУП

Мікроконтролери ARM є ключовим компонентом для великої кількості систем, що вбудовуються (embedded systems). Вони широко використовуються у мобільних телефонах, планшетах та інших пристроях. Наданий далі матеріал створено на основі вивчення автором можливостей 32-розрядних мікроконтролерів STM32 та можливостей відлагоджувальної плати STM32F4Discovery. Представлені лабораторні завдання передбачають вивчення побудови інформаційних систем, організації взаємодії декількох пристроїв між собою, використання передачі та відображення інформації в комп'ютерних системах, а також самостійну розробку програмного забезпечення з використанням спеціалізованих програм, проведення тестування та налагодження на реальних пристроях.

У ході виконання лабораторних завдань вивчаються принципи 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 відповідно до індивідуального завдання. Для успішного виконання здобувачами освіти виконання складених далі завдань до лабораторних робіт необхідні знання основ теорії цифрової схемотехніки, розробки програмного забезпечення та алгоритмізації, а також знання мови програмування C++.

Даний дипломний проект присвячено розробці лабораторних завдань для вивчення периферії мікроконтролерів STM32 з дисципліни "Периферійні пристрої". Пояснювальна записка та графічна частина проекту містить матеріали, які будуть корисними для вивчення мікроконтролерів архітектури ARM Cortex-M4. У роботі наведено короткий опис архітектури ARM та 32-розрядних мікроконтролерів STM, а також надано загальну інформацію, яка необхідна для початку роботи з відлагоджувальною платою STM32F4Discovery. Наведено декілька лабораторних робіт для вивчення основних можливостей, пристроїв та характеристик плати: ШІМ, АЦП, USART, SPI, DMA, таймерів та ін.

1 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

1.1 Архітектура ARM 32-розрядних мікроконтролерів STM

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

1. Змінна кількість циклів виконання для найпростіших інструкцій. Прості інструкції ARM можуть вимагати виконання більш одного циклу. Наприклад, виконання інструкцій Loadта Save залежить від кількості регістрів, які їм передані;

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

3. Умовне виконання – інструкція виконується лише у випадку, якщо виконується конкретна умова. Це збільшує продуктивність і дозволяє позбутися операторів розгалуження;

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

Програміст може розглядати ядро ARM як набір функціональних блоків – ALU, MMUта ін. – з'єднаних шиною даних. Дані надходять у процесор через шину даних. Декодер інструкцій обробляє інструкції перед виконанням. ARM можуть працювати тільки з даними, які записані в регістрах, тому перед виконанням інструкцій до реєстрів записуються дані для їх виконання. ALU зчитує дані з регістрів, виконує необхідні операції та записує результат назад у регістр, звідки його можна записати у зовнішню пам'ять [1].

Процесори ARM містять до 18 регістрів: 16 регістрів даних та 2 регістри процесів. Всі регістри містять 32 біти і називаються від R0 до R15. Регістри R13, R14, R15 використовуються для виконання певних специфічних завдань:

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

Залежно від контексту, ці регістри можуть використовуватися як регістри

загального призначення. Також є два програмні регістри, які називаються CPSR (Current Program Status Register) та SPSR (Saved Program Status Register), які використовуються для збереження стану процесора та програми. Одними з сучасних мікроконтролерів для систем, що вбудовуються, є процесори, засновані на архітектурі ARM Cortex-M4. Ці процесори призначені для використання у цифровій обробці сигналів (Digital Signal Processing, DSP). У загальному вигляді мікроконтролери, засновані на базі ARM Cortex-M4 мають такі внутрішні модулі (рис. 1.1): мікроконтролер, встановлений на платі, що розглядається у даній роботі (STM32F407VG) використовує саме ARM Cortex-M4 [4].

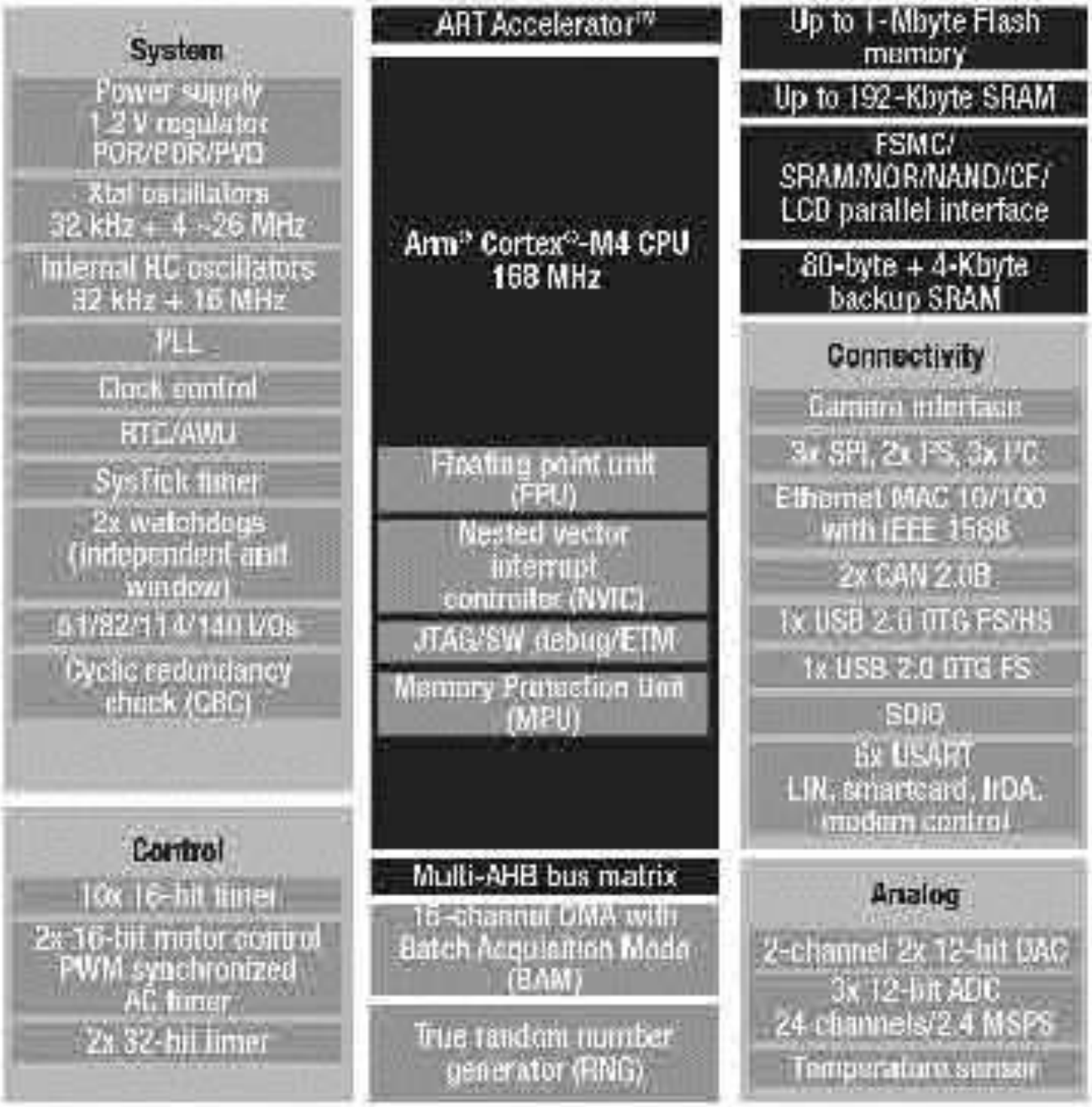

Рисунок 1.1. Вбудовані модулі мікроконтролеру ARM Cortex-M4 STM32F407VG

На рис. 1.1 видно, що мікроконтролер сімейства STM32 Cortex-M4 має багату периферію і може використовуватися для вирішення багатьох практичних завдань різної спрямованості.

Сімейство мікроконтролерів STM32 побудовано з використанням 32-розрядного ядра Cortex різних версій (у мікроконтролері, встановленому на платі STM32F407VG, використовується ядро Cortex-M4). Деякі основні характеристики ядра мікроконтролерів STM32 представлені у табл. 1.1.

Таблиця 1.1. Основні характеристики ядра мікроконтролерів STM32

Характеристика	*Значення*
Ширина слів для даних, розрядів	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

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

Рисунок 1.2. Представлення цифрового периферійного пристрою для STM32

На рис. 1.2 представлене спрощене уявлення цифрового периферійного пристрою для STM32. Периферійний вузол може бути розділений на два головні блоки. Перший блок – це ядро, яке містить кінцеві автомати, лічильники та будь-який вид комбінаторної чи послідовної логіки. Воно призначене для виконання завдань, що не вимагають участі процесора, таких як прості завдання передачі даних, управління аналоговими входами або виконання функцій, прив'язаних до синхросигналів. Ядро периферійного вузла з'єднується із зовнішнім світом через порти вводу/виводу МК. Зовнішні з'єднання можуть складатися з кількох сигналів чи складних шин. Другий блок – налаштування та управління периферією, що здійснюється додатком через регістри, з'єднані з внутрішньою шиною, що поділяється з іншими ресурсами МК [6].

1.2 Оснащення відлагоджувальної плати STM32F4 Discovery

Відлагоджувальна плата STM32F4 Discovery (рис. 13) призначена для ознайомлення з можливостями 32-бітного МК на основі ARM-архітектури, а також для реалізації власних пристроїв та програм з використанням апаратного забезпечення плати.

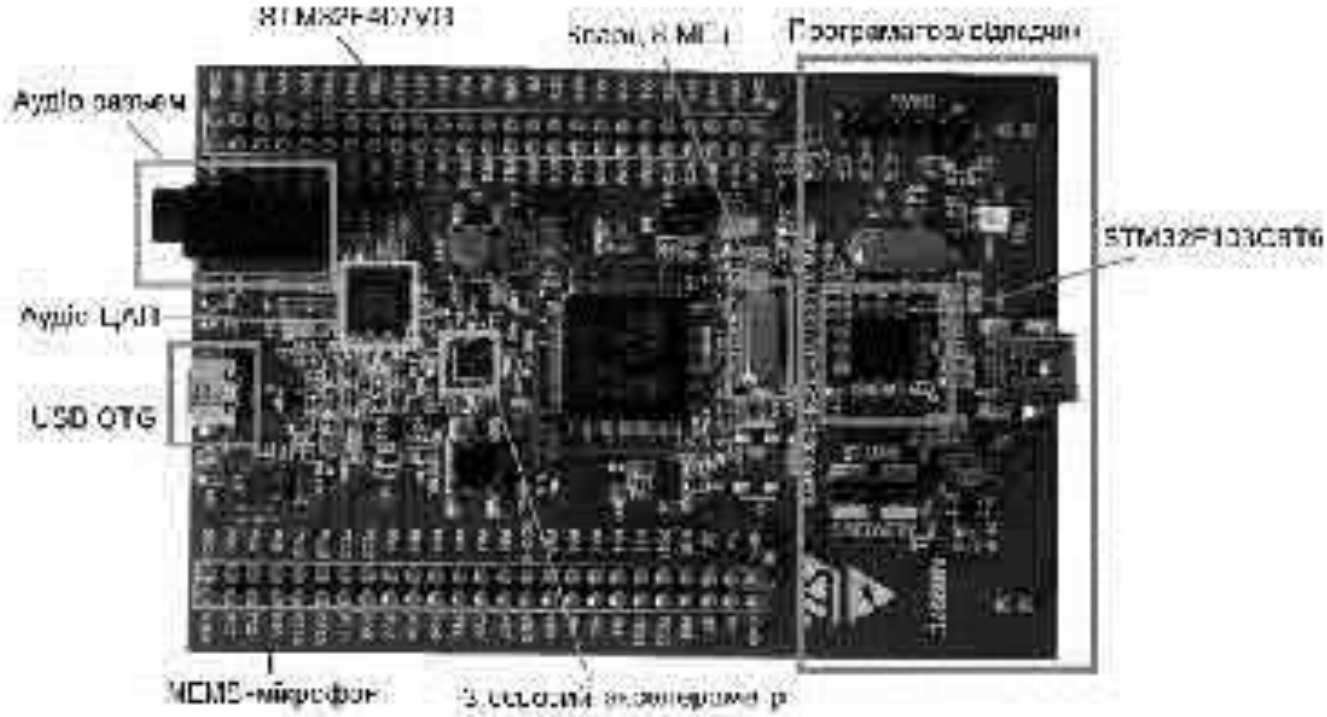

Рисунок 1.3. Зовнішній вигляд плати STM32F4 Discovery

Плату STM32F4 Discovery (рис.1.4) обладнано:

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

- відладчиком ST-Link/V2 для налагодження та програмування МК;

- живленням плати через USB або від зовнішнього джерела живлення 5 В;

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

- датчиком звуку ST MEMS MP45D T02;

- звуковим ЦАП CS43L22;

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

- двома кнопками (для програмування користувачем та для перезапуску).

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

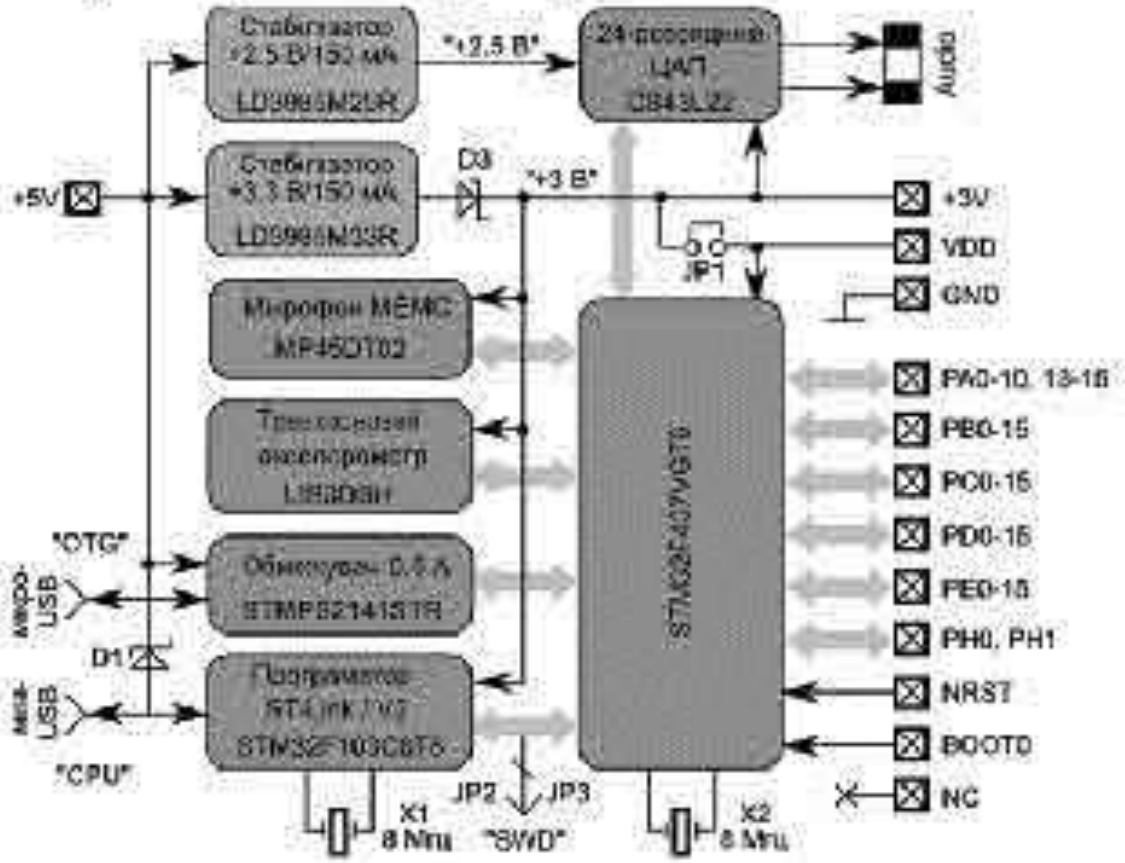

Рисунок 1.4. Оснащення плати STM32F4 Discovery

1.3 Розробка лабораторних завдань для вивчення периферії мікроконтролерів STM32

1.3.1 Створення проекту в середовищі розробки. Використання портів введення/виведення

Метою даної роботи є ознайомлення із процедурою створення проектів для відлагоджувальної плати STM32F4 Discovery, вивчення її структури та принципів роботи з портами введення/виведення, організації їхньої взаємодії.

Необхідним обладнанням та програмним забезпеченням є плата STM32F4 Discovery, середовище розробки Coo Cox CoIDE 1.7, інструменти побудови проекту GNU Toolchain for ARM Embedded Processors, бібліотека CMSIS за необхідності самостійного підключення файлів до проекту.

Контролер STM32F407VG містить п'ять 16-розрядних портів введення/виведення загального призначення, які позначені як GPIOx, де x може 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 даних (GPIOx_ODR, GPIOx_IDR) і два 32-бітовірегістри вибору додаткових функцій (GPIOx_AFRH і GPIOx_AFRL).

Світлодіоди, призначені для програмування, на платі підключені до порту D (рис. 1.5).

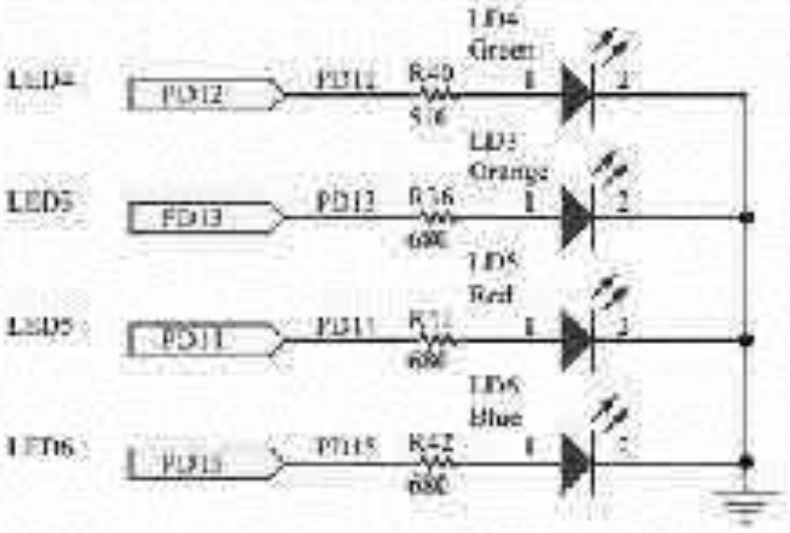

Рисунок 1.5. Схема підключення світлодіодів до порту на відлагоджувальній платі

Інші чотири світлодіоди виконують службові функції індикації і в програмуванні певних дій не використовуються.

Для виконання лабораторних робіт рекомендується встановити середовище CooCox CoIDE 1.7. Створення проекту виконується за допомогою майстра, який відкривається під час виконання команди меню Project/New Project. У майстрі під крок за кроком вказати ім'я проекту та його розташування, випробника та МК, для якого призначена програма. Далі робота з проектом здійснюється за допомогою вікна Repository, що дозволяє додати необхідні бібліотеки керування периферійними частинами МК, а також вікна Project. Вікно Repository також є майстром, який містить кілька вкладок, основною з яких є вкладка Peripherals (рис. 1.6).

Проект програми для плати STM32F4 Discovery має подібну структуру всім засобам розробки. Зазвичай він включає два внутрішніх каталоги: один – для файлів з бібліотеки CMSIS, а інший – для файлів, призначених для роботи з периферією [7].

Рисунок 1.6. Вікно Repository середовища CooCox CoIDE

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

```
#include "stm32f4xx.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_gpio.h"
void Delay(__IO 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 виходу, 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);
}
}
}
```

Хід даної лабораторної роботи 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 (табл.1.1);

4. Реалізувати потрібну функціональність;

5. Запрограмувати плату та продемонструвати роботу програми (рис.1.7).

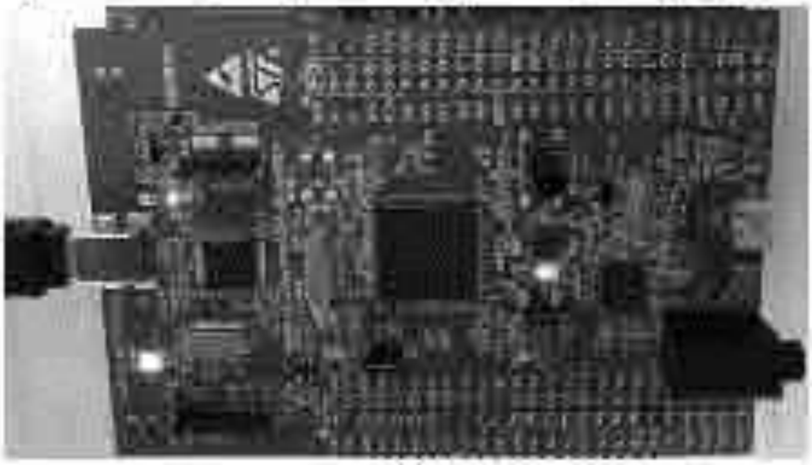

Рисунок 1.7. Демонстрація роботи портів введення/виведення

У роботі пропонується індивідуальні завдання відповідно до варіанту за табл. 1.1 реалізувати схему увімкнення/вимкнення світлодіодів, розташованих на платі

Таблиця 1.1. 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 їхніми номерами у складі порту D (можна знайти у документації відладжувальної плати).

1.3.2 Переривання та їх використання. Використання таймерів

Метою даної роботи є ознайомлення з поняттям переривань, можливостями, які вони надають та навчитися використовувати таймери для виконання дій в затверджені часові інтервали.

Необхідним обладнанням та програмним забезпеченням є плата STM32F4 Discovery, середовище розробки Coo Cox CoIDE 1.7, інструменти побудови проекту GNU Toolchain for ARM Embedded Processors, бібліотека CMSIS за необхідності самостійного підключення файлів до проекту.

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

Для використання переривань необхідно спочатку налаштувати регістр, який називається Nested Vector Interrupt Controller (NVIC), вбудований контролер вектору переривань. Цей регістр є стандартною частиною архітектури ARM та зустрічається на всіх процесорах, незалежно від виробника.

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

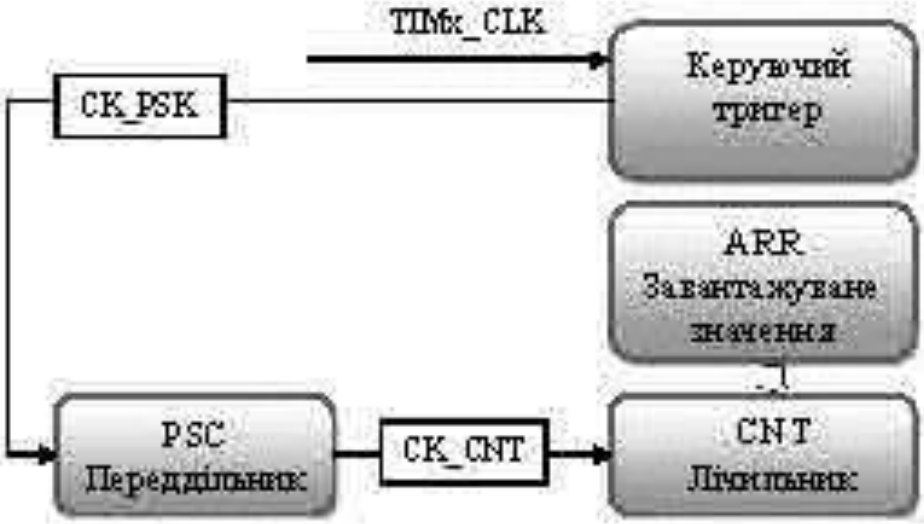

Рисунок 1.8. Схема керування підрахунком імпульсів

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

1) з розширеними можливостями;

2) загального призначення;

3) базові

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

Розглянемо приклад програми, яка демонструє роботу з перериваннями та таймерами. У ній реалізовано перемикання світлодіоду, підключеного до порту 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 inrrim_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_str;
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
gpio_str.GPIO_Pin = GPIO_Pin_13;
gpio_str.GPIO_Mode = GPIO_Mode_Out;
gpio_str.GPIO_OType = GPIO_OType_PP;
gpio_str.GPIO_PuPd = GPIO_PuPd_NOPULL;
gpio_str.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_Init(GPIOD, &gpio_str);
}
void INTTIM_Config(void)
{
NVIC_InitTypeDef nvic_str;
nvic_str.NVIC_IRQChannel = TIM2_IRQn;
nvic_str.NVIC_IRQChannelPreemptionPriority = 0;
nvic_str.NVIC_IRQChannelSubPriority = 1;
nvic_str.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&nvic_str);
TIM_TimeBaseInitTypeDef tim_str;
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
 tim_str.TIM_Period = 10000 - 1;
tim_str.TIM_Prescaler = 168 - 1;
tim_str.TIM_ClockDivision = 0;
tim_str.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM2, &tim_str);
TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
TIM_Cmd(TIM2, ENABLE);
}
```

У додатковому заголовному файлі misc.h описані функції, перерахування, структури для роботи з перериваннями.

Хід даної лабораторної роботи 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;

4. Реалізувати потрібну функціональність;

5. Запрограмувати плату та продемонструвати роботу програми (рис.1.9).

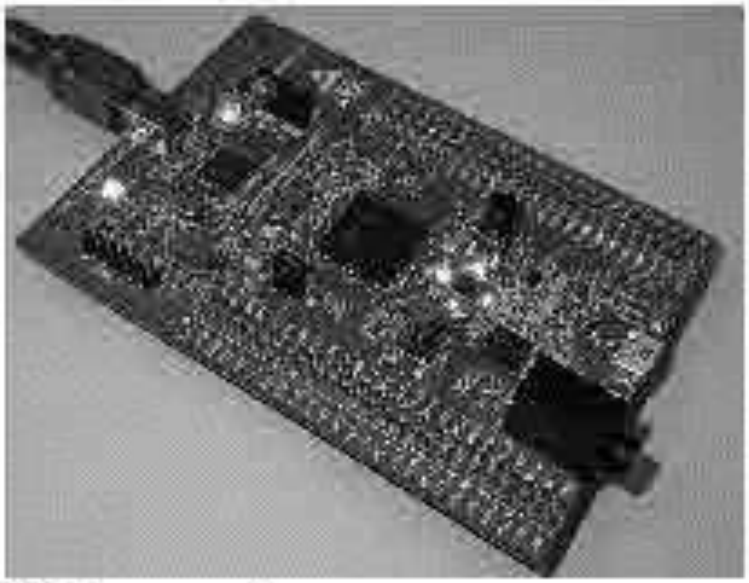

Рисунок 1.9. Демонстрація використання переривань та таймерів

У роботі пропонуються наступні індивідуальні завдання:

1. 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;

2. Реалізувати за допомогою переривання по переповненню таймера часову затримку з почерговим включенням світлодіодів на платі по колу.

1.3.3 Генерування сигналу широтно-імпульсної модуляції

Метою даної роботи є ознайомлення із можливостями генерації ШІМ за допомогою демонстраційної плати та навчання генеруванню сигналу із заданими параметрами та його використання для керування зовнішніми пристроями.

Необхідним обладнанням та програмним забезпеченням є плата STM32F4 Discovery, середовище розробки CooCox CoIDE 1.7, інструменти побудови проекту GNU Toolchain for ARM Embedded Processors, бібліотека CMSIS за необхідності самостійного підключення файлів до проекту.

Широтно-імпульсна модуляція (ШІМ) – спосіб керування середнім значенням напруги на навантаженні шляхом зміни скважності імпульсів, керованих ключем.

Мікроконтролери дозволяють генерувати ШІМ різної частоти. Оскільки виведення сигналу ШІМ не є основною функцією портів, які підключені до світлодіодів, необхідно виконати відповідну їх конфігурацію. Для цього слід задати виконання портами альтернативних функцій. Генерація ШІМ пов'язана з використанням додаткових режимів таймера.

Перед підключенням пристрою мають бути відомі такі параметри ШІМ, як частота та коефіцієнт заповнення. Для їх розрахунку в контролерах STM32 необхідно визначити значення переддільника і значення, що автоматично завантажується у регістрі ARR (Auto-Reload Register). Розрахунок значення, яке слід записати до переддільника, виконується наступним чином:

$$PSC = \frac{TIMxCLK}{TIMxCNT} - 1, \qquad (1.1)$$

де PSC – значення переддільника,

$TIMxCLK$ – вхідна частота роботи таймера,

$TIMxCNT$ – частота лічильника.

Для отримання необхідної вихідної частоти слід записати значення у регістр ARR, що отримується з наступного співвідношення:

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

де ARR_VAL – значення для запису в регістр ARR,

$TIMxCNT$ – частота лічильника,

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

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

$$D = \frac{CCRx\_VAL}{ARR\_VAL} * 100\%, \qquad (1.3)$$

где D – коефіцієнт заповнення,

CCRx _VAL – значення у регістрі CCRx (x – номер регістру для конкретної лінії),

ARR _VAL – значення для запису у регістр ARR.

Особливістю даних мікроконтролерів є те, що в переддільник та інші регістри можна записати будь-яке значення, яке можна описати за допомогою відведеної кількості розрядів.

Видача сигналу ШІМ на вихід є основним режимом роботи виводів порту, а належить до додаткових (альтернативних) режимів. Тому потрібно задати потрібний режим в налаштуваннях порту.

Для підключення алтернативних функцій до портів введення/виведення необхідно:

1. Підключити порт до альтернативної функції відповідного периферійного пристрою за допомогою функції GPIO_PinAFConfig;

2. Налаштувати порт у режим виконання альтернативної функції – GPIO_Mode_AF;

3. Виконати інші необхідні налаштування;

4. Викликати GPIO_Init для застосування вказаних налаштувань.

Для ознайомлення з можливістю генерації сигналу ШІМ та використання його для керування яскравістю світлодіодів надано наступну програму (у лістингу представлено зміст файлу з 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 counter = 0,
int i,
while(1)
{
counter++,
TIM4->CCR3 = 999 - (counter + 0) % 999,
TIM4->CCR4 = 999 - (counter + 666 / 2) % 999,
TIM4->CCR1 = 999 - (counter + 999 / 2) % 999,
TIM4->CCR2 = 999 - (counter + 499 / 2) % 999,
for(i=0,i < 30000, ++i),
```

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

В основній функції відбувається зміна значень регістру захвату/порівняння з певною затримкою. Функції з виконанням налаштувань перенесено до окремого файлу init.c. Відповідно, оголошення функцій знаходяться у файлі init.h. У 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(),
TIM4init(),
}
void TIM4init() {
TIM_TimeBaseInitTypeDef tim_init,
TIM_OCInitTypeDef oc_init,
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE),
uint16_t PrescalerValue = (uint16_t)((SystemCoreClock / 2) / 21000000),
tim_init.TIM_Period = 665,
tim_init.TIM_Prescaler = PrescalerValue,
tim_init.TIM_ClockDivision = 0,
tim_init.TIM_CounterMode = TIM_CounterMode_Up,
TIM_TimeBaseInit(TIM4, &tim_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 з прикладу, для ініціалізації таймера необхідно відразу дві структури: одну – для ініціалізації безпосередньо таймера, іншу – для задання режиму порівняння виходу. Наведена програма дозволяє послідовно змінювати яскравість світіння користувацьких світлодіодів на платі.

Рисунок 1.10. Демонстрація використання ШІМ

Хід даної лабораторної роботи 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 у середовищі розробки для виконання індивідуального завдання, отриманого від викладача;

4. Реалізувати потрібну функціональність;

5. Запрограмувати плату та продемонструвати роботу програми (рис. 1.10).

1.3.4 Використання аналогово-цифрового перетворювача

Метою даної роботи є дослідження можливостей використання аналогово-цифрового перетворювача (АЦП) у складі відлагоджувальної плати.

Необхідним обладнанням та програмним забезпеченням є плата STM32F4 Discovery, середовище розробки Coo Cox CoIDE 1.7, інструменти побудови проекту GNU Toolchain for ARM Embedded Processors, бібліотека CMSIS за необхідності самостійного підключення файлів до проекту, джерело напруги (акумулятор), підключений до входів АЦП.

Мікроконтролер STM32F407VG містить три АЦП. Розрядність всіх АЦП становить 12 біт. Кожен перетворювач здатний приймати сигнал із шістнадцяти зовнішніх каналів. Крім того, до складу мікроконтролера входить датчик температури. Діапазон вхідної напруги становить 1.8...3.6 В. Датчик температури підключений до вхідного каналу ADC_IN16, який використовується для перетворення вихідної напруги сенсора на цифрове значення. Внутрішній датчик температури призначений для відстеження зміни температури, а не для її вимірювання, оскільки зсув показників датчика може змінюватися під час змін параметрів процесу. Тому, якщо потрібне точне вимірювання абсолютних значень температури – краще використовувати зовнішній датчик.

У наступному прикладі наведено програму, яка дозволяє вимірювати напругу, яка подається на вхід АЦП. Перегляд значення напруги здійснюється у режимі налагодження (рис. 1.11).

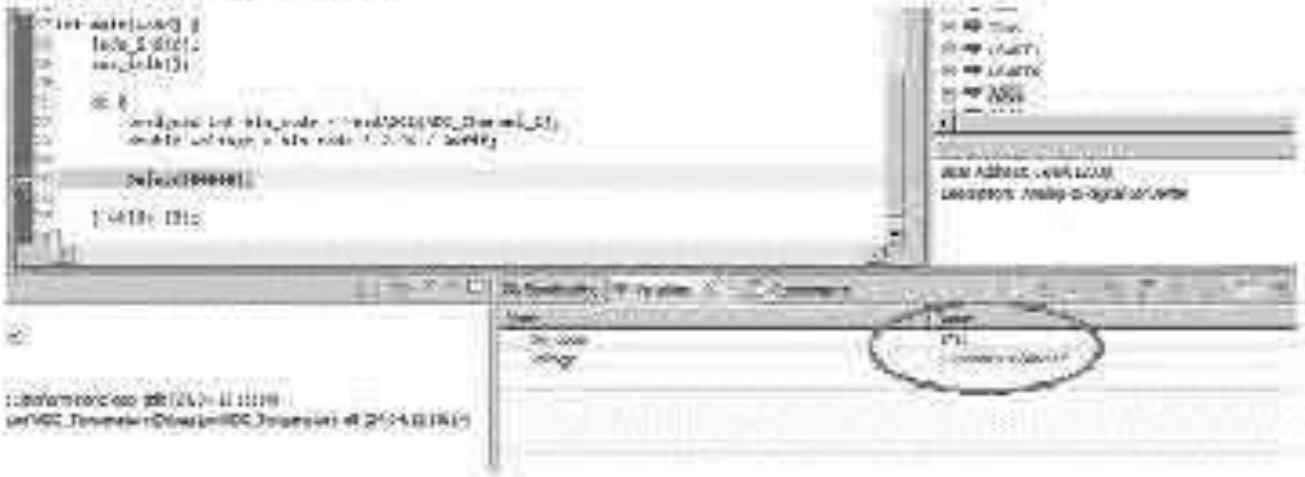

Рисунок 1.11. Контроль значення напруги у режимі відлагодження

Для правильного визначення напруги необхідно провести додаткові вимірювання. Для цього за допомогою мультиметру або вольтметру треба визначити напругу, яка відповідає 3 В, підключивши щупи вимірювального

приладу до відповідних контактів на платі. Наприклад, мультиметр показав, що у даному випадку напруга дорівнює 2.96 В. Саме цей коефіцієнт треба безпосередньо записати до коду програми. Розрахунок напруги виконується за формулою:

$$U_{вх} = \frac{U_{оп} * BIN_{вх}}{BIN_{max}} \quad (1.4)$$

де $U_{вх}$ – значення поданої напруги,

$U_{оп}$ – напруга, відносно якої проводиться порівняння,

BIN_{max} – дійсний код максимально можливого значення, яке може зберігатися у регістрі даних АЦП. Залежить від його розрядності (у нашому випадку розрядність АЦП дорівнює 12 бітів, тому максимальній напрузі відповідає число 0xFFF, у якого в молодших 12 розрядах всі одиниці),

$BIN_{вх}$ – значення дійсного коду з регістру даних АЦП, записане після проведення вимірювань.

Відповідно, виходячи з формули 1.4, при подачі напруги 0 В, буде отримано мінімальне значення напруги. Здійснити це можна з'єднавши контакти землі GND та першого контакту порту А перемичкою. У результаті цього значення в регістрі даних АЦП зміниться так, як показано на рис. 1.12.

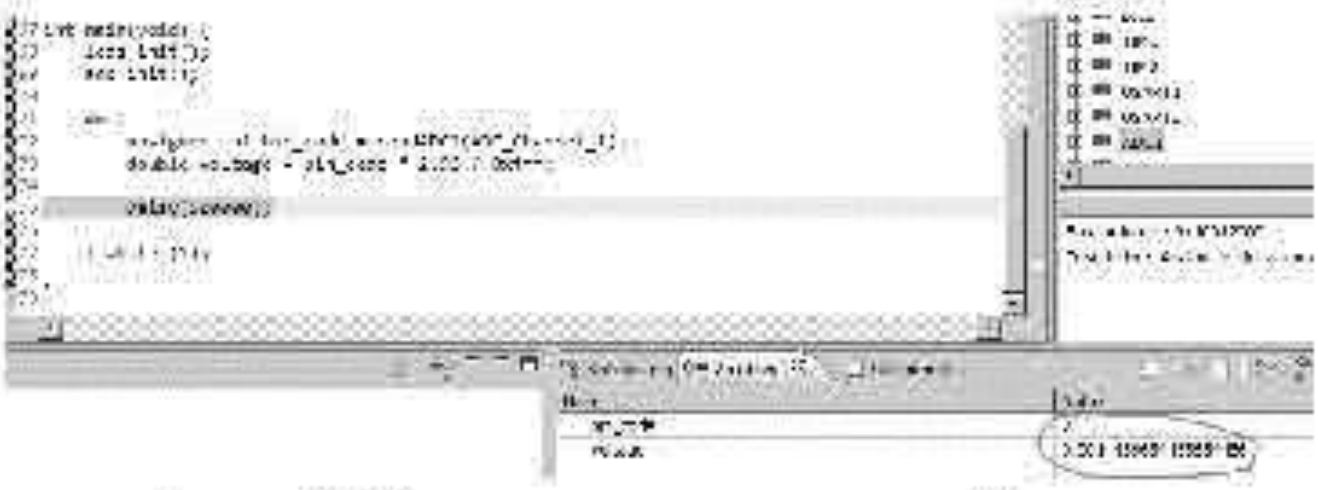

Рисунок 1.12. Контроль значення напруги у режимі відлагодження при підключенні землі до входу АЦП

На рис.1.12 можна помітити, що значення напруги досить близько до нуля, що свідчить про коректну роботи програми. При ініціалізації порту необхідне значення для його роботи у аналоговому режимі задається за допомогою параметру

GPIO_Mode_AN. Для ініціалізації та зчитування значень з АЦП треба налаштувати його таким чином, щоб перетворення здійснювалися програмно і виконувались за допомогою виклику функції readADC1.

У 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);
}
uint16_t readADC1(uint8_t 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(__IO uint32_t val) {
for (; val != 0; val--);
}
int main(void) {
leds_init();
adc_init();
do {
__IO uint16_t bit_code = readADC1(ADC_Channel_1);
double voltage = bit_code * 2.96 / 0xfff;
delay(100000);
} while (1);
}
```

Хід даної лабораторної роботи 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 використовуваних функцій, переглянувши

вихідний код, а також коментарі у вихідних файлах бібліотеки та в режимі налагодження;

3. Створити новий проект у середовищі розробки для виконання індивідуального завдання;

4. Реалізувати потрібну функціональність;

5. Запрограмувати плату та продемонструвати роботу програми (рис.1.13).

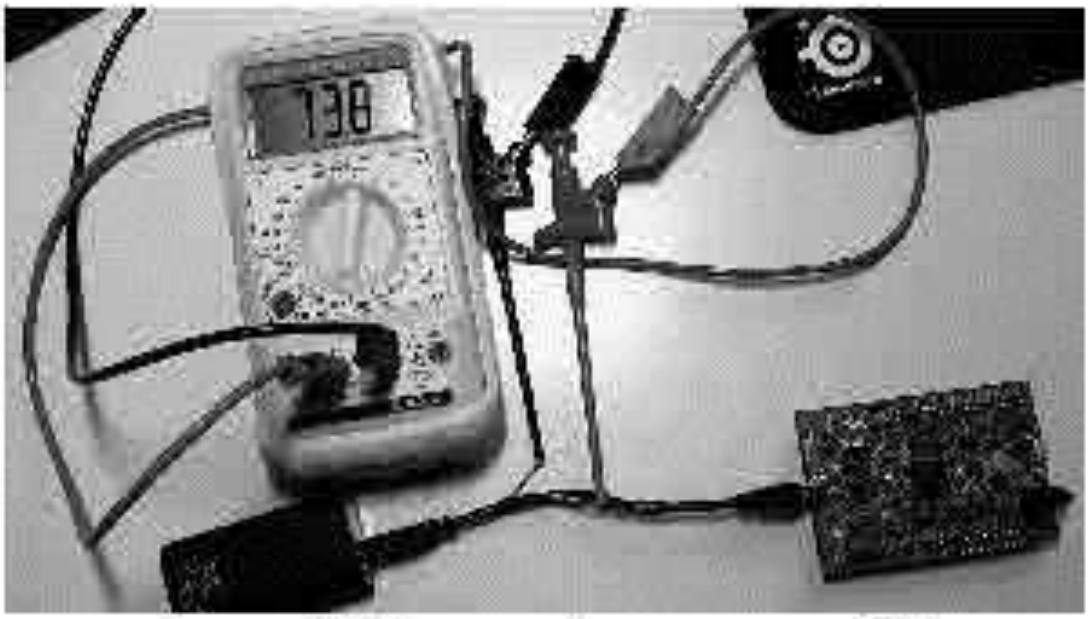

Рисунок 1.13. Демонстрація використання АЦП

Індивідуальні завдання за даною темою:

1. Розробити програму, яка при подачі на АЦП напруги меншої, ніж половина від максимальної, вимикає світлодіод, а при подачі більшої напруги вмикає світлодіод. Підтвердити коректність роботи програми за допомогою мультиметру;

2. Продемонструвати прийняття даних з кількох каналів АЦП;

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

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

1.3.5 Використання універсального приймача/передавача USART

Метою даної роботи є вивчення роботи універсального синхронного/асинхронного приймача/передавача, організація передачі даних до

інших пристроїв.

Необхідним обладнанням та програмним забезпеченням є плата STM32F4 Discovery, середовище розробки CooCox CoIDE 1.7.

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

Розглянемо процес налаштування USART для прийняття даних. Для цього необхідно виконати такі дії:

1. Увімкнути тактування USART і порту, виходи якого використовуються для роботи USART;

2. Налаштувати відповідні виходи, вказавши при цьому режим роботи виконання альтернативної функції;

3. Підключити виконання альтернативної функції до зазначених виходів;

4. Задати налаштування роботи USART (частота, розмір передачі, кількість стоп-бітів, перевірка на парність);

5. Увімкнути переривання прийняття даних для USART;

6. Увімкнути USART.

У наступному прикладі розглянуто прийняття даних за допомогою USART1. Для передачі та прийняття використовуються виходи порту A під номерами 9 (Transmit) та 10 (Receive). Для кожного приймача-передавача виділено свої пари виводів для прийому та передачі, які можна знайти у документації до мікроконтролера [3]. Насамперед, треба увімкнути тактування потрібних пристроїв:

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

Після цього треба провести ініціалізацію виводів порту та призначити виконання альтернативної функції:

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

Далі треба вибрати альтернативну функцію, що виконується виводами порту.

Робиться це за допомогою 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 та PA3 використовувати у якості USART не вдасться.

Далі необхідно ініціалізувати USART. Код ініціалізації дуже схожий на ініціалізацію порту:

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

На останньому етапі ініціалізації необхідно налаштувати переривання та увімкнути USART:

```
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 необхідно визначити його обробник. Обробник переривання має USART1_IRQHandler і він не повертає ні не приймає жодних даних. Крім того, USART підтримує кілька переривань і для всіх переривань використовується один обробник, тому необхідно перевіряти прапорець (RXNE) у регістрі статусу, щоб організувати обробку різних переривань у різних блоках коду обробника.

Після перевірки стану прапора можна скинути прапор RXNE. Далі треба зчитати дані з регістру даних та 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_usart(t);
}
}
```

У даному прикладі розглянуто прийняття даних з іншого пристрою

(використано плату STM8S Value Line, проте можна використовувати будь-який інший пристрій з UART або USART) та відображення прийнятої інформації на символьному LCD-дисплеї.

Для перевірки роботи USART навіть не обов'язково використовувати інший пристрій, тому що можна замкнути між собою виходи передавача та приймача. Необхідно встановити однакові параметри повідомлення як на пристрої-передавачі, так і на приймачі.

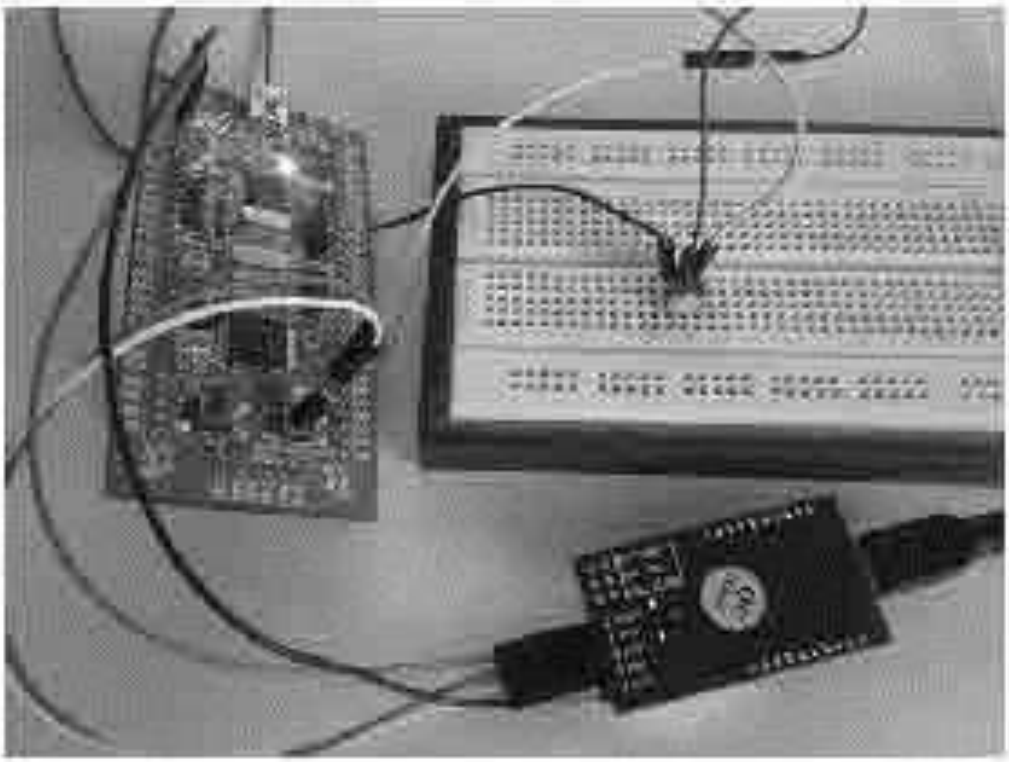

Рисунок 1.14. Демонстрація використання USART

Хід даної лабораторної роботи 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 у режимі відлагодження;

3. Створити новий проект у середовищі розробки для виконання індивідуального завдання;

4. Реалізувати потрібну функціональність;

5. Запрограмувати плату та продемонструвати роботу програми (рис.1.14).

Індивідуальні завдання за даною темою:

1. Задіяти два модулі USART на платі та передати дані з одного у інший в асинхронному режимі. Для демонстрації прийняття даних змінювати стан одного з світлодіодів на протилежний;

2. Продемонструвати прийняття/передачу у синхронному режимі. Для демонстрації прийому даних змінювати стан одного з світлодіодів на протилежний.

1.3.6 Використання інтерфейсу SPI

Метою даної роботи є застосування інтерфейсу SPI для організації передачі даних між пристроями, дослідження режиму ведучого та веденого пристроїв.

Необхідним обладнанням та програмним забезпеченням є плата STM32F4 Discovery, середовище розробки CooCox CoIDE 1.7, Keil 4.xx, інструменти побудови проекту GNU Tookhain for ARM Embedded Processors, бібліотека CMSIS, набір перемичок (3 шт.).

Serial Peripheral Interface (SPI) – популярний інтерфейс для обміну даними між мікросхемами. Шину SPI організовано за принципом "ведучий-ведений". У якості ведучого шини зазвичай виступає мікроконтролер, але ним також може бути програмована логіка, DSP-контролер або спеціалізована IC. Підключені до провідної шини зовнішні пристрої утворюють підлеглі шини. В їх ролі виступають різноманітні мікросхеми. Зокрема: запам'ятовуючі пристрої (EEPROM, Flash-пам'ять, SRAM), годинник реального часу (RTC), АЦП/ЦАП, цифрові потенціометри, спеціалізовані контролери та ін.

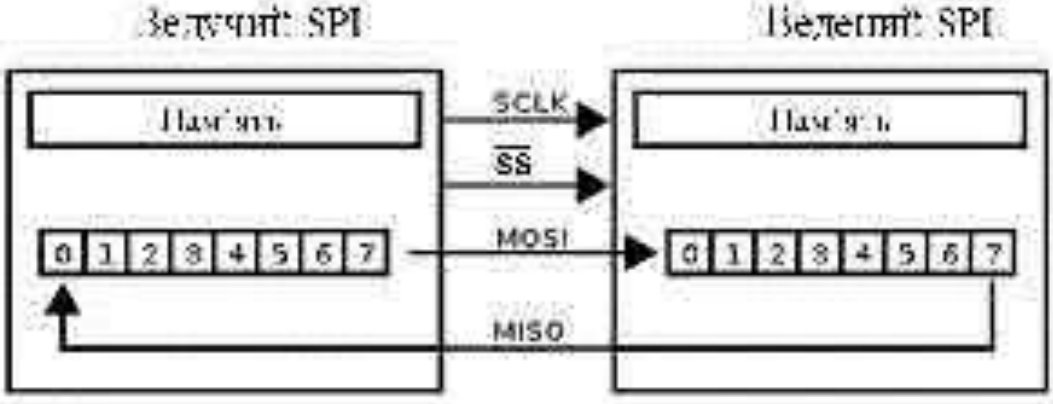

Рисунок 1.15. Схема простого підключення з використанням SPI

Головним складовим блоком інтерфейсу SPI є звичайний зсувний регістр, сигнали синхронізації та введення/виведення бітового потоку якого і утворюють

інтерфейсні сигнали. Таким чином, протокол SPI правильніше назвати не протоколом передачі даних, а протоколом обміну даними між двома зсувними регістрами, кожен з яких одночасно виконує функцію приймача і функцію передавача. Обов'язковою умовою передачі по шині SPI є генерація сигналу синхронізації шини. Цей сигнал має право генерувати тільки ведучий шини і від цього сигналу повністю залежить робота підлеглої шини.

Приклад найпростішого підключення по шині SPI показаний на рис. 1.15. Однойменні виводи з'єднуються між собою. Існують два канали передачі даних (MOSI і MISO), один канал для подачі тактових імпульсів (SCLK), а також лінія увімкнення веденого пристрою (SS). Ведений стає активним при подачі низького рівня по лінії SS.

SPI – надзвичайно простий та поширений послідовний інтерфейс передачі даних,що ґрунтується на зсувних регістрах. Його перевагою порівняно з USART є можливість підключення кількох відомих пристроїв. Однак у порівнянні з USART він підтримує тільки синхронну передачу. Наявність кількох модулів SPI у складі мікроконтролера дозволяє обійтися без підключення додаткових пристроїв із таким інтерфейсом, а використовувати кілька пристроїв на платі, з'єднавши їхні входи між собою перемичками. Цей варіант є оптимальним для навчальних цілей, оскільки вимагає мінімум додаткового обладнання для початку роботи.

У наступному прикладі продемонстровано застосування інтерфейсу SPI для організації передачі даних 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 nCount);
void SPI3_IRQHandler(void) {
int res;
if (SPI_I2S_GetITStatus(SPI3, SPI_I2S_IT_RXNE) != RESET) {
SPI_I2S_ClearITPendingBit(SPI3, SPI_I2S_IT_RXNE);
res = SPI_I2S_ReceiveData(SPI3);
}
}
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 i0,i1) {
while(--i0,i1),
}
```

Хід даної лабораторної роботи 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;

4. Реалізувати потрібну функціональність;

5. Запрограмувати плату та продемонструвати роботу програми (рис.1.16).

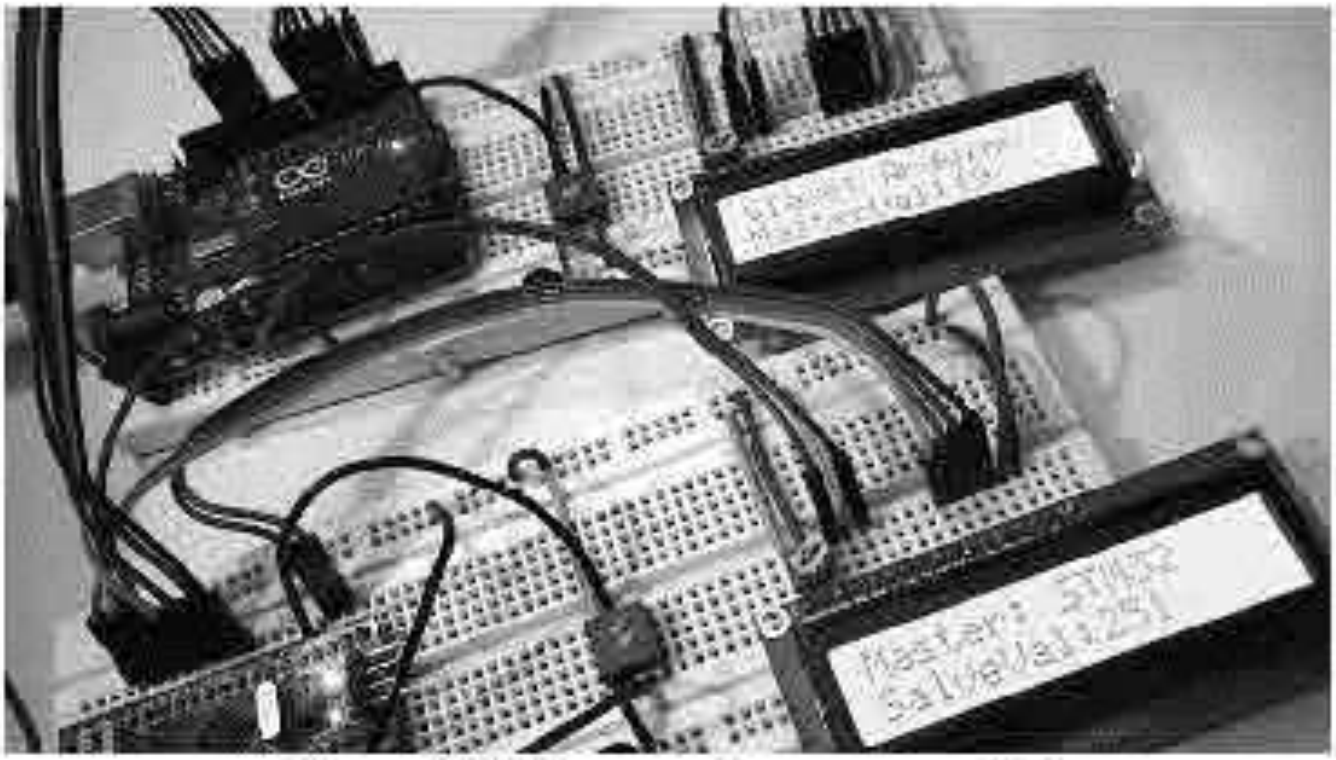

Рисунок 1.16. Демонстрація використання SPI

Індивідуальні завдання за даною темою:

1. Налаштувати на відладжувальній платі один із модулів у режимі прийому даних та прийняти дані від іншого пристрою з відображенням на семисегментному індикаторі;

2. 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;

3. 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;

4. Організувати передачу даних іншому модулю SPI на відлагоджувальній платі (значення від 0x00 до 0x0F) з відображенням двійкового значення на світлодіодах на платі;

5. Організувати одночасну передачу даних декільком веденим пристроям, які відображають отримане значення на семисегментному індикаторі та у двійковому коді з використанням світлодіодів.

1.3.7 Використання контролеру прямого доступу до пам'яті

Метою даної роботи є використання режиму DMA (прямого доступу до пам'яті) та організація взаємодії з іншими пристроями у складі мікроконтролера.

Необхідним обладнанням та програмним забезпеченням є плата STM32F4 Discovery, мультиметр, середовище розробки CooCox CoIDE 1.7, Keil 4.xx, інструменти побудови проекту GNU Tookhain for ARM Embedded Processors, бібліотека CMSIS.

Direct Memory Access (DMA, прямий доступ до пам'яті) – механізм, який використовується в контролерах ARM для переміщення даних між пам'яттю та периферією без участі процесора. При використанні DMA для переміщення даних не використовуються ресурси процесора, що може бути особливо важливим при створенні додатків, що працюють з великою кількістю даних і активно використовують периферію. Робота DMA забезпечується окремим контролером, який виконує певні дії за командою процесора. Схема взаємодії пристроїв наведена на рис. 1.17.

Контролер у складі відлагоджувальної плати має у своєму розпорядженні відразу два контролери DMA, які забезпечують загалом 16 потоків (по 8 потоків на кожен контролер), кожен з яких призначений для керування запитами до пам'яті від

одного або кількох пристроїв. Кожен потік може забезпечити до 8 каналів (запитів). Кожен контролер DMA має пристрій вирішення конфліктів для обробки запитів відповідно до їх пріоритету.

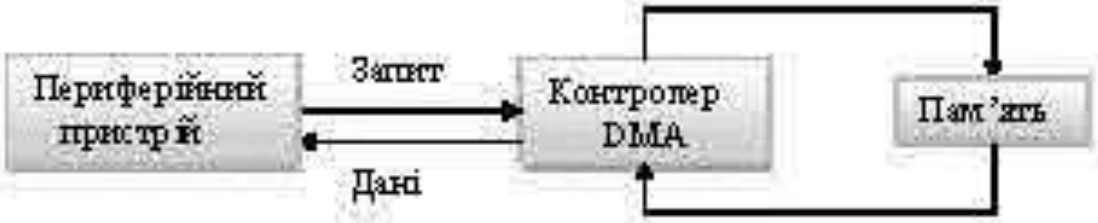

Рисунок 1.17. Схема використання DMA

Контролер DMA дозволяє робити запис даних у трьох напрямках:

- від периферії в пам'ять;
- із пам'яті до периферії;
- з пам'яті в пам'ять.

Передача ведеться або у режимі безпосередньої передачі, або у режимі черги. Підримуеться різний розмір даних для передачі, причому розмір даних для приймача і джерела може бути неоднаковим. У такому разі DMA визначає цю ситуацію та виконує необхідні дії для оптимізації передачі, однак ця можливість підримуеться лише у режимі черги.

Дуже корисною може бути функція циклічної передачі, при використанні якої передача даних починається з початкової адреси знову після передачі останньої одиниці даних джерела.

Програміст може задавати пріоритети для запиту кожного конкретного потоку або пріоритет визначатиметься на основі значень за замовчуванням.

У наступному прикладі програми продемонстровано використання DMA для передачі даних із пам'яті в ЦАП. Зміна рівня сигналу на виході ЦАП відбуваеться циклічно із частотою зміни 1 с. Зміни відбуваються на основі значення, яке зчитуеться з пам'яті за сигналом переповнення таймера. У результаті цього ЦАП відправляє запит до DMA і отримує дані, які записуються в регістр даних, що призводить до зміни рівня сигналу. У програмі слід звернути увагу на велику кількість параметрів, необхідних для налаштування DMA в порівнянні з іншими пристроями. З цим пов'язана складність використання DMA, оскільки необхідно

враховувати велику кількість можливих налаштувань. Тим не менш, багато з них досить прості (напрямок передачі, 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, 0x33, 0x22, 0x33, 0x44, 0x33, 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 = 0x0000 - 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 програми за наведеним вище прикладом та прошивки відлагоджувальної плати треба спостерігати за напругою на виході PA4 за допомогою мультиметру. Приклад має показати ступінчасту зміну напруги від 0 до певного значення,після чого цикл повторюється. Крок збільшення однаковий для всього циклу,що можна побачити за масивом значень для запису у регістр даних.

Хід даної лабораторної роботи пропонується наступний:

1. На основі коду наданого прикладу програми здобувачеві освіти необхідно створити свій проект у середовищі розробки та перевірити його працездатність (рис.1.18);

2. Ознайомитись з документацією по DMA для мікроконтролера відлагоджувальної плати;

3. Змінити параметри адресації у програмі-прикладі;

4. Організувати взаємодію DMA та іншого периферійного пристрою відповідно до індивідуального завдання.

Індивідуальні завдання за даною темою:

1. Реалізувати асинхронну передачу даних через USART,зчитуючи дані через певні проміжки часу за допомогою DMA;

2. Організувати прийняття даних через USART із записом до пам'яті через DMA. Кількість даних, що передається, відома заздалегідь і дорівнює розміру

масиву;

3. Продемонструвати передачу даних через SPI з пам'яті у циклічному режимі.

4. Продемонструвати прийом даних через SPI із записом у пам'ять. Режим запису – циклічний;

5. 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.

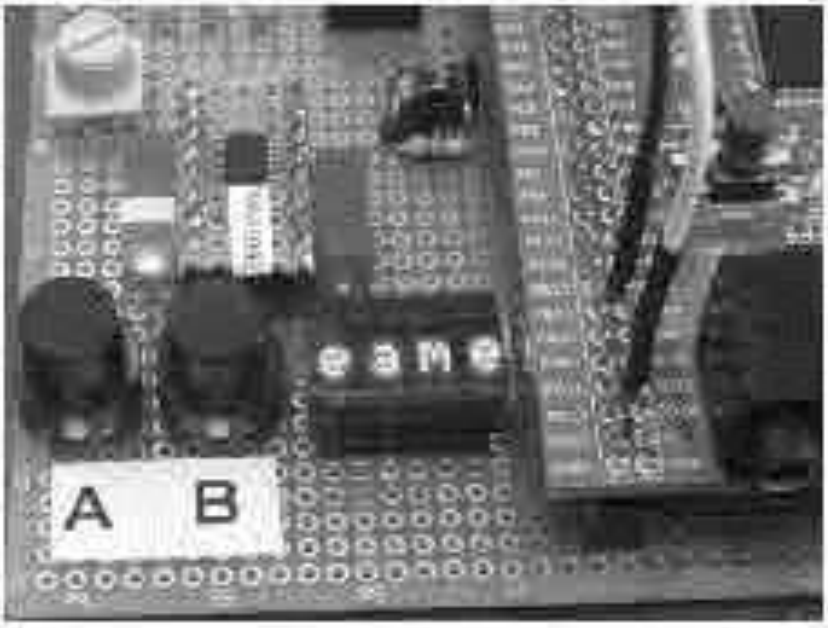

Рисунок 1.18. Демонстрація використання прямого доступу до пам'яті

1.3.8 Керування характеристиками генерованих сигналів та відображення інформації на LCD-дисплеї

Метою даної роботи є вивчення можливості генерування сигналів та способи керування їх характеристиками з використанням відлагоджувальної плати та відображенням інформації.

Необхідним обладнанням та програмним забезпеченням є плата STM32F4 Discovery, символьний LCD-дисплей, 3 тактових кнопки та резистори для їх підключення, середовище розробки CooCox CoIDE 1.7, Keil 4 xx, інструменти побудови проекту GNU Toolchain for ARM Embedded Processors, бібліотеки CMSIS та SPL.

У наступному прикладі буде складено програму, яка дозволяє на виході отримати сигнал прямокутної форми в заданому діапазоні частот та із заданим

коефіцієнтом заповнення. Відповідно користувачеві необхідно надати можливість регулювання зазначені характеристики. Для прикладу взяті такі значення:

- вихідна частота – 3-4 кГц;

- коефіцієнт заповнення ±50% від початкового значення, яким вважається значення 0,5.

Для генерації сигналів прямокутної форми якнайкраще підходить вмикання таймеру в режимі ШІМ для одного з вихідних каналів. Необхідно лише розрахувати значення для налаштування самого таймера при зазначеній тактовій частоті пристрою (вони будуть різними для 16 МГц і 168 МГц) і включити тактування та необхідний режим роботи для інших пристроїв у складі контролера.

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

Значення коефіцієнту заповнення зберігається в програмі у вигляді дійсного числа, яке користувач може змінювати, а виконані зміни записуються в регістр порівняння для обраного вихідного каналу.

У якості рішення для надання користувачеві можливості регулювання параметрів вихідного сигналу пропонується підключити символьний LCD-дисплей і три кнопки для виконання регулювання. Функціональне призначення кнопок може змінюватись відповідно до дій користувача. У той же час, на екрані слід відображати інформацію про поточні зміни (зміна пунктів меню, значень характеристик), які виконує користувач.

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

```
#include "init.h"
#include "stm32f4xx_gpio.h"
#include <stm32f4xx_rcc.h>
```

```c
#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, до 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 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);
}
// Конфігурування кнопок на роботу 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 заданого 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 кнопок
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 для затримок
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM7, ENABLE);
TIM_TimeBaseInit(DELAY_TIMER, &tim_init);
}
// налаштування таймеру для 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 функцій затримки
```

```
void delay(int times) {
DELAY_TIMER->CNT = 0;
DELAY_TIMER->ARR = times;
TIM_CMD(DELAY_TIMER, ENABLE); // Запуск таймера
while(TIM_GetFlagStatus(DELAY_TIMER, TIM_FLAG_Update) == RESET); // Ожидание
TIM_ClearFlag(DELAY_TIMER, TIM_FLAG_Update);
TIM_CMD(DELAY_TIMER, DISABLE); // Выключение таймера
}
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);
}
// Обработчик нажатия кнопки для изменения частоты
void increment_freq(void) {
if (freq.freq_value == MAX_FREQ_VALUE) return;
freq.freq_value++;
TIM_SetAutoreload(PWM_TIMER, freq.freq_value - 1);
TIM_SetCompare1(PWM_TIMER, freq.freq_value * fill_value);
show_freq_on_screen();
}
// Обработчик нажатия кнопки для изменения частоты
void decrement_freq(void) {
if (freq.freq_value == MIN_FREQ_VALUE) return;
freq.freq_value--;
TIM_SetAutoreload(PWM_TIMER, freq.freq_value - 1);
TIM_SetCompare1(PWM_TIMER, freq.freq_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, freq.freq_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, freq.freq_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 change_menu_item(struct menu_item * to, struct menu_item * from) {
to = from;
}
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 (menu_option_index == 0) {
} else {
menu_option_index--;
menu_MENU.items[3] = menu_options[menu_option_index];
show_text_on_screen(&menu_MENU.items[3]);
}
}
void show_freq_on_screen(void) {
char text[20];
double freq = (double)16000000 / (freq.cycle_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);
}
// Инициализация всех системных
void init_all(void) {
freq.cycle_value = 333;
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(&menu_MENU.items[3]);
}
```

У заголовному файлі знаходяться оголошення функцій, а також використовуваних змінних та констант:

```
#ifndef stm_h
#define stm_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 333
#define MIN_FREQ_VALUE 400
#define MAX_FILL_VALUE 0.75
#define MIN_FILL_VALUE 0.25
typedef void (*action)(void);
/* Структура для представления пункта меню */
struct MENU_item {
action action; // дія
char MENU_text[20]; // текст меню
},
struct MENU {
struct MENU_item items[3]; // пункти меню
},
int freq.cycle_value, // змінна для зберігання частоти, яка змінюється в межах 400...333
double fill_value, // змінна для зберігання 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 у 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 set_active_menu(struct menu * menu);
void set_active_freq_menu(void);
void set_active_fill_menu(void);
void set_active_main_menu(void);
void empty_action(void);
void change_menu_item(struct menu_item * item, char * text, action act);
void change_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
```

У файлі з основною функцією викликається функція ініціалізації всіх пристроїв і описані обробники переривань для 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)
```

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

Разом з безпосередньою обробкою сигналів важливою частиною роботи схеми є відображення інформації, результатів обробки для користувача. У тому випадку, коли схема досить проста і спрямована на відстеження невеликої кількості показників, достатньо, наприклад, використовувати семисегментний індикатор або символьний LCD-дисплей. Такі дисплеї часто можуть відображати більше інформації, ніж індикатори, складені з кількох семисегментних, а також можуть відображати набагато більше різних символів. При виконанні даної роботи передбачається використовувати дисплей RC1602A (рис. 1.19). Даний дисплей відображає 2 рядки по 16 символів у кожному та працює під керуванням контролера KS0066U. Він забезпечує наступну функціональність роботи з дисплеєм:

- виконання команд керування (керування розрядністю, курсором та інші операції);

- виконання запису та зчитування пам'яті дисплея.

Використана схема підключення дисплею представлена на рис. 1.20.

Рисунок 1.19. Зовнішній вигляд дисплея RC1602A

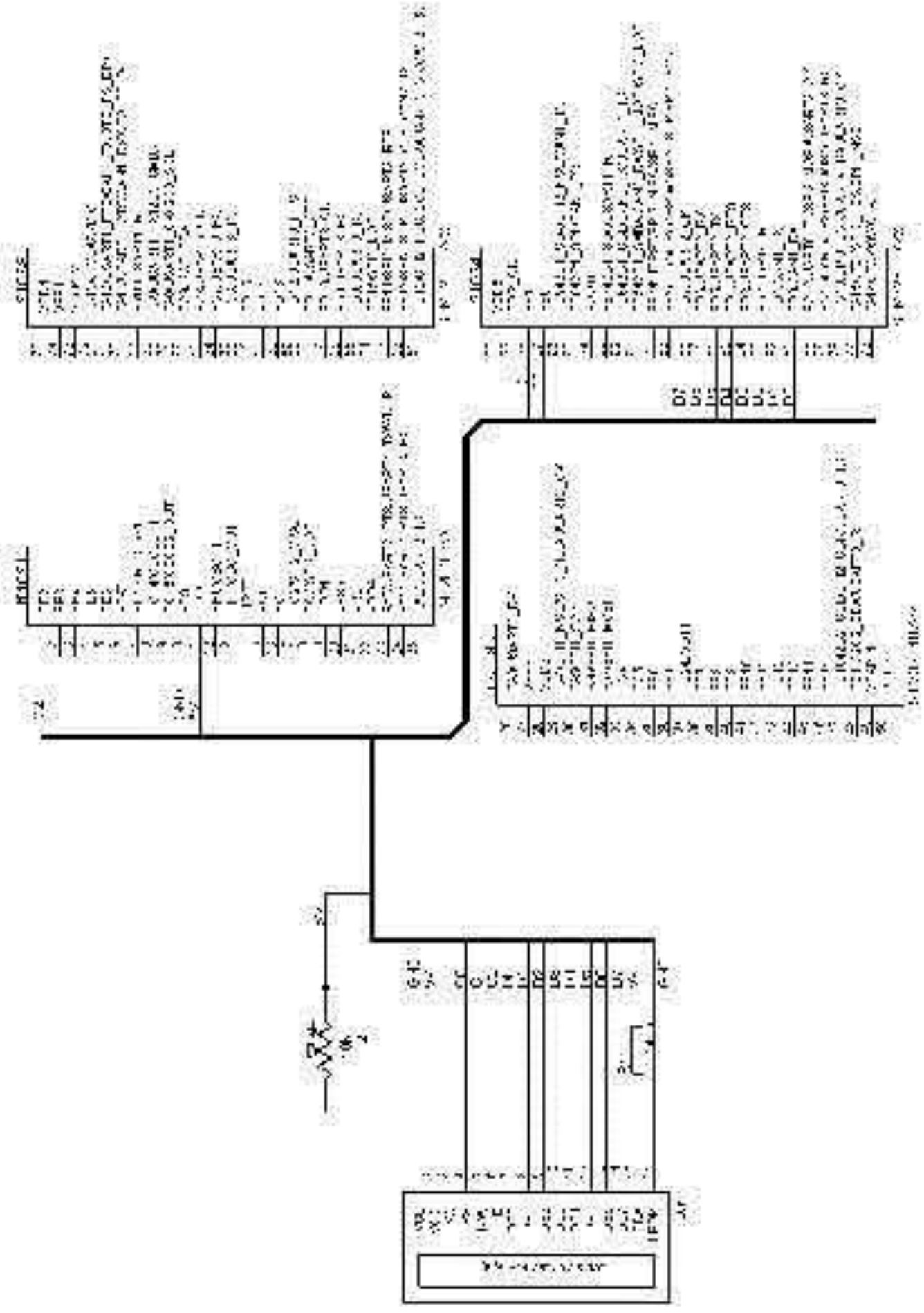

Рисунок 1.20. Схема підключення дисплея RC1602A

Для керування дисплеєм за допомогою плати STM32F4Discovery створено бібліотеку (реалізовано найнеобхідніше: виконання команді запис даних у пам'ять для відображення). Сама пам'ять розділена на дві частини: пам'ять відображення (DDRAM) та пам'ять відведена під символи користувача (CGRAM). Інформація, записана до пам'яті, відображається на екрані. Оскільки розмір пам'яті більше розміру екрану, відображається лише її частина, а відображення решти можна забезпечити за допомогою зсуву відображуваної на екрані пам'яті. Докладніший опис можливостей дисплеїв такого типу можна знайти у документації до них [5].

Нижче наведено код створеної бібліотеки. Заголовний файл бібліотеки:

```
#ifndef __LCD_H
#define __LCD_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 "stm32lib.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(),
```

```
reset_rw();
write_data(lcdMode_data);
set_e();
delay(40); reset_e();
}
void write_char(unsigned char data) {
set_rs();
reset_rw();
write_data(data);
set_e();
delay(40);
reset_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 reset_rs(){
GPIO_ResetBits(CONTROL_PORT, RS_PIN);
}
void set_rw(){
GPIO_SetBits(CONTROL_PORT, RW_PIN);
}
void reset_rw(){
GPIO_ResetBits(CONTROL_PORT, RW_PIN);
}
void set_e(){
GPIO_SetBits(CONTROL_PORT, E_PIN);
}
void reset_e(){
GPIO_ResetBits(CONTROL_PORT, E_PIN);
}
```

Передбачається що показана вище бібліотека використовуватиметься як частина проекту у середовищі розробки. Відповідно, слід налаштувати для проекту шляхи пошуку заголовних файлів, щоб вони включали шлях до stm32f4xx_gpio.h або відредагувати файл відповідним чином. Крім того, необхідно задати значення CONTROL_PORT та DATA_PORT та номери контактів для користувацького варіанту налаштування. Також користувачеві необхідно визначити функцію delay(int), яка використовується для створення затримок між сигналами від плати

до дисплея. Під час тестування бібліотеки використовувався варіант функції, яка у циклі декрементує значення змінної. Крім того, користувачеві необхідно самостійно налаштувати відповідні порти для виведення перед використанням, після чого викликати функцію lcd_init().

При підключенні дисплею до відлагоджувальної плати слід застосовувати додаткові резистори, один з яких повинен мати змінний опір для регулювання яскравості відображуваних символів.

Для керування варіантами меню, а також зміни параметрів використовуються підключені до виходів 0-2 порту B тактові кнопки. У зазначеному вище прикладі можна використати 3-контактні тактові кнопки (рис. 1.21), проте можна використовувати і інші кнопки.

Рисунок 1.21. Зовнішній вигляд тактової кнопки

Перед тим, як здійснити підключення, слід вивчити за допомогою мультиметру в режимі вимірювання опору, які контакти замикаються при натисканні, а які залишаються з'єднаними завжди. Крім того, для підтягування виходу до землі використовуються резистори номіналом 5,9 кОм.

Для розміщення описаного вище з'єднання використовується макетна плата для паяння. Усі виводи на платі (3 – для кнопок та 2 – для живлення та землі) для зручності підключені до шпилькового роз'єму. Під'єднання до відлагоджувальної плати здійснюється за допомогою перемичок. Можливе також використання контактної макетної плати.

Принципову електричну схему підключення зазначених вище компонентів до відлагоджувальної плати STM32F4 Discovery наведено на рис. 1.22.

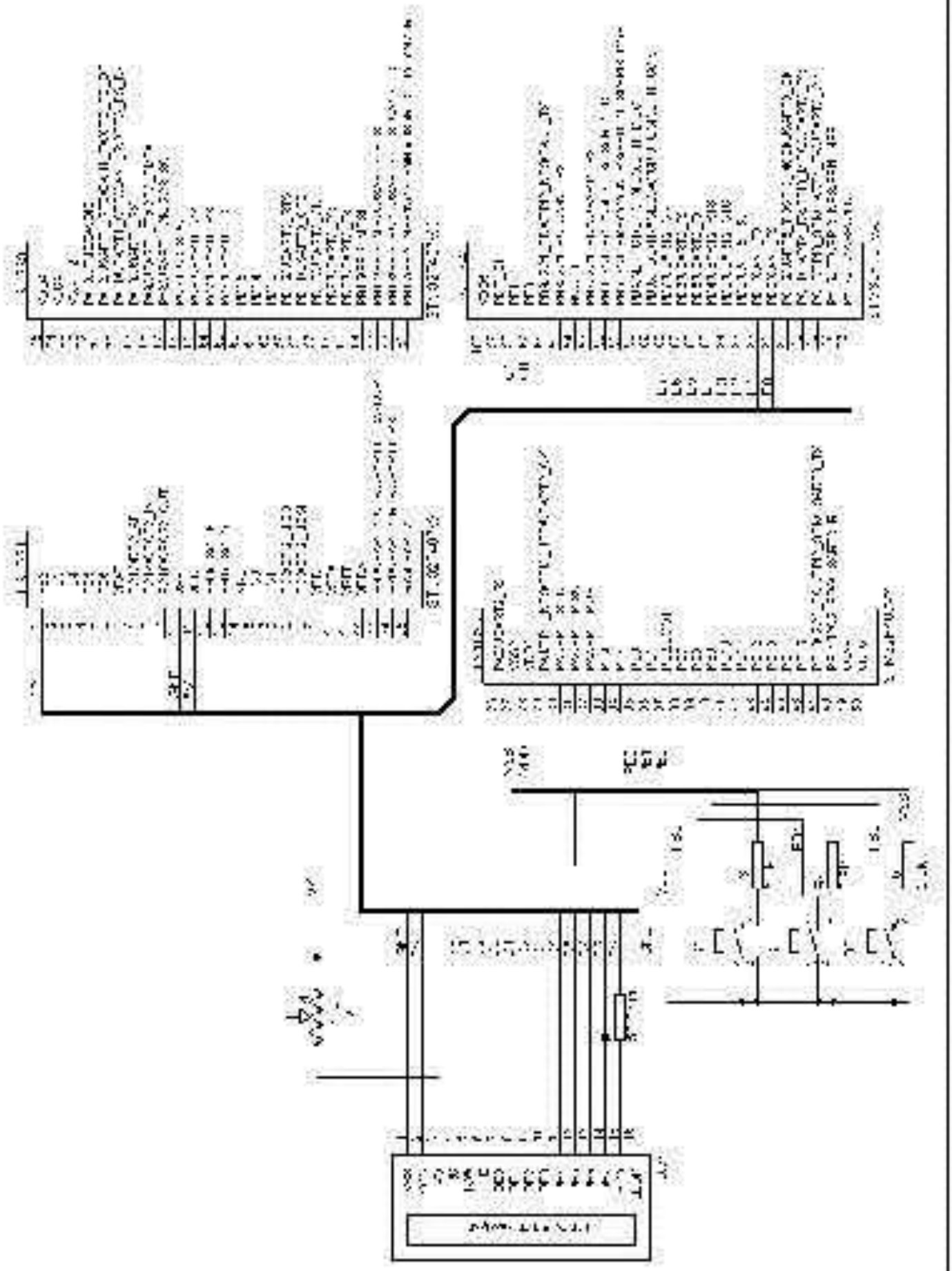

Рисунок 1.22. Принципова електрична схема підключення компонентів до відлагоджувальної плати

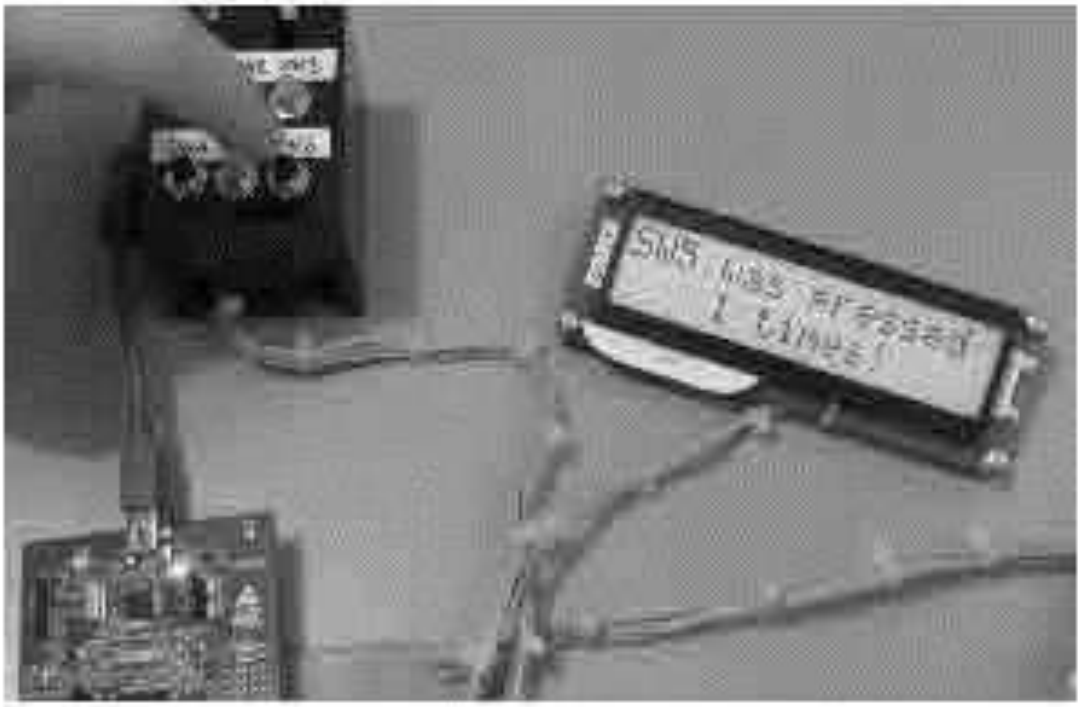

Рисунок 123. Демонстрація використання дисплею та керування сигналами

Хід даної лабораторної роботи пропонується наступний:

1. На основі коду наданого прикладу програми здобувачеві освіти необхідно створити свій проект у середовищі розробки та перевірити його працездатність (рис.123);

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.

Індивідуальні завдання за даною темою:

1. 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;

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

2 ЕКОНОМІЧНА ЧАСТИНА

Темою даного дипломного проекту є «Розробка лабораторних завдань для вивчення периферії мікроконтролерів STM32 з дисципліни "Периферійні пристрої». Основна мета проекту – створення матеріалу та лабораторного практикуму для допомоги у вивченні можливостей 32-розрядних мікроконтролерів та їх використання у побудові інформаційних систем, організації взаємодії декількох пристроїв між собою, використання передачі та відображення інформації в комп'ютерних системах, навчанні самостійної розробки програмного забезпечення з використанням спеціалізованих програм, проведенні тестування та налагодження на реальних пристроях.

Даний вид проекту відноситься до науково-дослідницької розробки. Оцінка якості розробленого проекту включає визначення трудомісткості і вартості його створення

Проведемо розрахунок трудомісткості виконання науково-дослідницької розробки. Розрахунок трудомісткості НДР здійснений в наступній послідовності:

1) Складений перелік всіх етапів і видів робіт, які необхідно виконати в ході даної НДР. Після узгодження з керівником проекту допущено виключення, доповнення, об'єднання окремих етапів і видів робіт;

2) По кожному виду робіт визначений кваліфікаційний рівень виконавців. Розподіл робіт по етапах і видах виконавців виробленій формою, наведено в таблиці 2.1.

Таблиця 2.1 Розподіл робіт по етапах і видах виконавців

Етап проведення	Вид робіт	Посада виконавця
Розробка технічного завдання (ТЗ)	1. Складання і затвердження ТЗ для НДР «Розробка лабораторних завдань для вивчення периферії мікроконтролерів STM32 з дисципліни "Периферійні пристрої».	Дипломник, керівник
Вибір напряму дослідження	1. Збір і вивчення науково-технічної літератури. 2. Формулювання можливих напрямів вирішення завдань, поставлених в технічному завданні НДР. 3. Вибір напряму проведення досліджень 4. Розробка плану проведення досліджень для подальшої розробки.	Дипломник керівник

Етап проведення	Вигляд робіт	Посада виконавця
Теоретичні і експериментальні дослідження	1. Загальний опис архітектури arm та 32-розрядних мікроконтролерів stm 2. Початок роботи зі відладочною платою stm32f4 discovery 3.Лабораторні роботи	Дипломник керівник консультанти
Узагальнення і оцінка результатів досліджень	1. Узагальнення результатів попередніх етапів роботи. 2. Складання і оформлення звіту. Розгляд результатів проведеної НДР і прийняття результатів в цілому.	Дипломник керівник консультанти

В умовах відсутності нормативної бази тривалість виконання окремих робіт розраховується на основі вірогідних оцінок робіт, що задаються виконавцями.

Таблиця 2.2 Очікувана трудомісткість робіт

Вигляд роботи	Очікуваний час виконання (дні)
1.Складання і затвердження ТЗ для НДР «Розробка лабораторних завдань для вивчення периферії мікроконтролерів STM32 з дисципліни "Периферійні пристрої».	1
2. Збір і вивчення науково – технічної літератури, технічної документації і інших матеріалів.	2
3. Формулювання можливих напрямів вирішення завдань, поставлених в технічному завданні НДР і їх порівняльна оцінка.	1
4. Вибір напряму проведення досліджень і способів вирішення поставлених завдань. Розробка плану проведення досліджень для подальшої розробки.	1
5. . Загальний опис архітектури arm та 32-розрядних мікроконтролерів stm	4
6. Початок роботи з відладочною платою stm32f4 discover	3
7. Лабораторні роботи	6
8. Економічна частина	2
9. Охорона праці	2
Всього:	22

Виходячи з особливостей створення науково – технічної продукції і її залежності від інтелектуальної праці, розрахунок собівартості і ціни виконання НДР включає наступні статті витрат: витрати на матеріали, основна і додаткова заробітна плата, відрахування до єдиного соціального фонду страхування, витрати на роботи, що виконуються сторонніми організаціями, і деякі інші.

1) Витрати на матеріали складаються з вартості придбаного паперу формата А4 і становитимуть 226 грн.

2) До витрат «Основна заробітна плата» відносяться оплата праці виконавців, безпосередньо причетних до її виконання. Розмір основної зарплати встановлюється виходячи з чисельності різних категорій виконавців, трудомісткості, що витрачається ними на виконання різних видів робіт, а також їх середньої заробітної плати (ставки) за один робочий день. Відповідно до статті 8 «Закону про Державний бюджет України на 2022» встановлено мінімальну заробітну плату у місячному розмірі з 1 січня 2022 року - 6500 гривень, мінімальну погодинну тарифну ставку – 39,26 грн.

Середня зарплата за один робочий день для кожного виконавця визначена по формулі:

$$Зден = птс. * 8;$$

де птс – погодинна тарифна ставка, грн..;

8 – тривалість робочого дня, год.

Зден дипломника = 39.26 *8 = 314,08 грн.

Зден керівника = 70,50 *8 = 564 грн.

Зден консультантів = 70,50 *8= 564 грн.

Витрати на основну заробітну плату, НДР, що включаються в собівартість, приведені в таблиці 2.3

Таблиця 2.3 Витрати на основну заробітну плату

Виконавець	Погодинна тарифна ставка, грн	Денна ставка, грн	Трудомісткість робочих днів	Сума основної зарплати, грн
Дипломник	39,26	314.08	22	6355,36
Керівник	70,50	564	1	564
Консультант по економічній частині	70,50	564	0,25	141
Консультант по охороні праці	70,50	564	0,25	141
Нормоконтроль	70,50	564	0,25	141
Всього (Зо)				7342,36

3) Витрати на додаткову заробітну плату визначаються у відсотках від основної У наукових закладах додаткова заробітна плата складає 10-20% від основної заробітної плати.

$$Зд = 10\% Зо = 7342,36 * 0,1 = 734,24 \text{ грн}$$

4)До складу собівартості НДР включаються податки, збори і інші обов'язкові платежі, встановлені системою оподаткування що діє. Відрахування до єдиного соціального внеску складає:

$$Зесв = 0,22*(Зо+Зд) = 0,22*(7342,36 + 734,24) = 1776,85 \text{ грн.}$$

5) До накладних витрат відносять витрати на управління і господарське обслуговування, що відносяться до всіх виконуваних НДР. У наукових закладах накладні витрати складають 40 - 120% від основної і додаткової заробітної плати.

$$Рнакл = (Зо+Зд)*0,4 = (7342,36 + 734,24)*0,4 = 3230,64 \text{ грн.}$$

На підставі отриманих даних по окремих статтях витрат складена калькуляція планової собівартості цілому НДР за формою, приведеною в таблиці 2.4

Калькуляція планової собівартості

Таблиця 2.4

Статті витрат	Сума, грн.
1. Матеріали	226,00
2. Основна заробітна плата	7342,36
3. Додаткова заробітна плата	734,24
4. Відрахування до єдиного соціального внеску	1776,85
5. Накладні витрати	3230,64
Планова собівартість (Спл)	13310,09

Плановий прибуток визначений по формулі:

$$Ппл = 0,1*Спл = 0,1*13310,09 = 1331,10 \text{ грн.}$$

Де 0,1 – норматив, який враховує граничний рівень рентабельності.

Договірна ціна визначається по формулі:

$$Цдір = Спл + Ппл = 13310,09 + 1331,10 = 14641,19 \text{ грн.}$$

Звідси ціна реалізації НДР становить:

$$Цр = Цдір + ПДВ;$$

$$Цр = 14641,19 + 0,2*14641,19 = 17569,43 \text{ грн.}$$

3 ОХОРОНА ПРАЦІ

Трудові права громадян охороняються законом. Захист трудових прав здійснюється державними органами, а також професійними спілками

Забезпечення здорових і безпечних умов праці покладається на адміністрацію підприємств, установ, організацій. Вона зобов'язана впроваджувати сучасні засоби техніки безпеки, попереджуючі виробничий травматизм і забезпечуючі санітарно-гігієнічні умови, що запобігають виникненню професійних захворювань.

Умови праці впливають на здоров'я, працездатність і всебічний розвиток особи трудящого. Узагальнюючи приведені вище положення, можна зробити висновок, що чим вища культура виробництва, тим краще умови праці, а отже, забезпечуються здоров'я і безпека працівників.

В розділі охорона праці дипломного проекту розглядаються питання умов праці програміста (користувача ПК), які повинні бути забезпечені на підприємстві для безпечної роботи працівника. У відповідності з Правилами охорони праці під час експлуатації ЕОМ на робочому місці користувача ПК повинні бути створенні умови для високопродуктивної праці. Розглянемо ці умови.

3.1 Аналіз небезпечних та шкідливих чинників, що впливають на працівника

В процесі роботи на користувачів ПК можуть мати вплив наступні небезпечні та шкідливі фактори:

- ✓ Невідповідність параметрів мікроклімату нормам;
- ✓ Недостатній рівень освітленості;
- ✓ Ураження електрострумом;
- ✓ Статична електрика;
- ✓ Порушення організації робочого місця тощо.

Користувач персонального комп'ютера має значне навантаження, як фізичне (сидяче положення, навантаження на очі тощо), так і розумове, що приводить до зниження його працездатності.

Розвитку стомлюваності сприяють наступні фактори – неправильна ергономічна

організація робочого місця, нераціональні зони розміщення устаткування по висоті від підлоги, характер протікання праці – чергування праці й відпочинку, зміна одних форм роботи на інші.

3.2 Розробка заходів з охорони праці

3.2.1 Виробничі приміщення

Розміщення робочих місць з ВДТ заборонено у підвальних приміщеннях та на цокольних поверхах заборонено. Для приміщень, які призначені для роботи з ВДТ, доцільно обрати орієнтацію вікон на північ або на північний схід. На вікнах повинні бути жалюзі, що регулюються, або штори, що дають можливість їх повністю закривати. Приміщення відповідно до ДБН В.2.5-28-2006 «Природне і штучне освітлення» повинні мати природне та штучне освітлення. При приміщеннях з ВДТ мають бути обладнані побутові приміщення для відпочинку, психологічного розвантаження тощо. Площа на одне робоче місце для користувачів повинна складати не менше 6 кв.м, а об'єм – не менше 20,0 куб.м. Стіни пофарбовані матовою фарбою, у відповідності з санітарними вимогами.

3.2.2 Мікроклімат робочої зони працівників, вентиляція

Найбільш значним фактором продуктивності й безпеки праці є виробничий мікроклімат. Він характеризується параметрами температури, вологості і швидкістю руху повітря. Порушення відповідності цих параметрів впливають на працездатність працівників, їх реакцій, збільшення кількості помилок. Тому в приміщенні повинні бути установлені оптимальні параметри мікроклімату: температура повітря 22-25 ^{0}С, вологість повітря – 40-60%, швидкість руху повітря – 0,1-0,2 м/с. Для цього приміщення має бути оснащене системами опалення й кондиціювання, що забезпечують постійне й рівномірне нагрівання, циркуляцію й очищення повітря від пилу й шкідливих речовин.

3.2.3 Освітлення робочого місця, шум, вібрація

Одним із основних питань охорони праці є організація раціонального освітлення виробничих приміщень і робочих місць.

Для освітлення приміщення, у якому працює користувач ПК, використовується змішане освітлення, тобто сполучення природного й штучного

освітлення.

Природне освітлення здійснюється через вікна в зовнішніх стінах будинку. Штучне освітлення використовують при недостатньому природному освітленні й здійснюють за допомогою двох систем: загального та місцевого освітлення. загального освітлення приміщення використовуються газорозрядні лампи типу ЛД. Норма для необхідної освітленості робочого місця становить 300-500 лк.

При розумовій праці, яка вимагає зосередженості припустимий рівень шуму становить 50дБ. Для зменшення шуму й вібрації в приміщенні устаткування, апарати й прилади встановлюють на спеціальні прокладки, що амортизують. Якщо стіни в приміщенні є джерелами шумоутворення, вони повинні бути облицьовані звукопоглинальним матеріалом. При розумовій праці, яка вимагає зосередженості припустимий рівень шуму становить 50дБ. Для зменшення шуму й вібрації в приміщенні устаткування, апарати й прилади встановлюють на спеціальні прокладки, що амортизують. Якщо стіни в приміщенні є джерелами шумоутворення, вони повинні бути облицьовані звукопоглинальним матеріалом

3.2.4 Електробезпека

На відміну від інших джерел небезпеки електричний струм не можна виявити без спеціального устаткування й приладів, тому вплив його на людину найчастіше виявляється.

Для попередження поразок електричним струмом необхідно:

- У повному обсязі виконувати правила провадження робіт і правил технічної експлуатації;
- Виключати можливість доступу працівника до частин устаткування, що працює під небезпечною напругою, неізольованим частинам, призначеним для роботи при малій напрузі й не підключеним до захисного заземлення;
- Застосовувати ізоляцію, що служить для захисту від поразки електричним струмом.

Заземлені конструкції, що знаходяться в приміщеннях, де розміщені робочі місця операторів (батареї опалення, водопровідні труби, кабелі із заземленим

відкритим екраном) мають бути надійно захищені діелектричними щитками або сітками з метою недопущення потрапляння працівника під напругу.

3.2.5 Організація робочого місця користувача ПК

Обладнання і організація робочого місця з ВДТ мають забезпечувати відповідність конструкцій всіх елементів робочого місця та їх взаємного розташування, ергономічним вимогам, з урахуванням характеру і особливостей трудової діяльності (ДСанПіН 3.3.2.-007-98).

Конструкція робочого місця й взаємне розташування всіх його елементів (сидіння, органи керування, засобу відображення інформації) відповідають антропометричним, фізіологічним і психологічним вимогам, а також характеру роботи. Конструкція робочих меблів дає можливість забезпечувати можливість індивідуального регулювання їх відповідно до потреб працівника для підтримки зручної пози. Робочий стіл повинен бути пофарбований матовою фарбою. Дисплей розташований так, що його верхній край перебуває на рівні очей, на відстані близько 70 см, що укладається в припустимі рамки від 60 до 90 см. Частота мерехтіння екрана дорівнює 100 Гц, що відповідає умові більше 70 Гц.

3.3 Пожежна безпека

Під пожежною безпекою розуміють систему державних і суспільних заходів, спрямованих на охорону від вогню людей і матеріальних цінностей

Заходи щодо пожежної безпеки підрозділяють на дві основні групи: попередження пожеж і ліквідація вже виниклих пожеж. Пожежна профілактика – це комплекс заходів, спрямованих на попередження пожежі, створення умов, сприяючих швидкій ліквідації пожежі.

Для ліквідації пожеж використовують первинні засоби пожежогасіння, які призначенні для гасіння пожеж у початковій стадії їх розвитку. Вони є у всіх виробничих приміщеннях, цехах.

Оснащення об'єктів первинними засобами пожежогасіння проводиться відповідно до Правил пожежної безпеки в Україні, введених в дію наказом внутрішніх справ України від 22.06.95 №400.

До первинних засобів пожежогасіння відносяться : вогнегасники, пожежний інвентар (покривала з негорючого теплоізоляційного полотна, грубошерстяної тканини або повсті, ящики з піском, бочки з водою, пожежні відра, совкові лопати) та пожежний інструмент (гаки, ломи, сокири тощо).

Для гасіння пожеж водою всередині будівель встановлюють внутрішні пожежні крани, які знаходяться у вбудованих або навісних шафках червоного кольору, якімають отвори для провітрювання іпристосовані для опломбування та візуального огляду їх без розкривання. На дверцятахпожежнихшаф із зовнішнього боку повинні бути вказані після літерного індексу пожежного крану «ПК» порядковий номер крана та номер телефону для виклику пожежної охорони.

Пожежні крани повинні постійно бути справними і доступними для використання.

Забезпеченняпожежної безпеки – це один із важливихнапрямків щодо охорони життята здоров'я людей, національного багатства і навколишнього середовища.

ВИСНОВКИ

У даному дипломному проекті розроблено 8 лабораторних завдань для вивчення периферії мікроконтролерів Cortex-M4 ARM-архітектури сімейства STM32 на базі відлагоджувальної плати STM32F4Discovery. Такі мікроконтролери є ключовим компонентом для великої кількості систем, що вбудовуються (embedded systems). Вони широко використовуються у мобільних телефонах, планшетах та інших пристроях. Відлагоджувальна плита 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 передач і та відображення інформації, а також самостійну розробку програмного забезпечення з використанням спеціалізованих середовищ розробки (зокрема, CooCox CoIDE, Keil). Все це відпрацьовано протягом дипломного проектування та відлагоджено на реальних пристроях, наявних у лабораторії "Периферійні пристрої" ОТФК ОНАХТ.

Наведено 8 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.

Для успішного виконання здобувачами освіти завдань до розроблених лабораторних робіт необхідні знання основ теорії цифрової схемотехніки, розробки програмного забезпечення та алгоритмізації, а також знання мови програмування C++.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. STM32F4DISCOVERY, Отладочный комплект на базе STM32F407VGT6 ARM CortexM4-F [Электронный ресурс] – Режим доступу: URL: http://www.chipdip.ru/product/stm32f4discovery/

2. Керниган Б., Ритчи Д. Язык программирования Си – 2-е изд. – М.: Вильямс, 2007. – С. 304.

3. GNU Tools for ARM Embedded Processors [Електронний ресурс] – Режим доступу: URL: https://launchpad.net/gcc-arm-embedded/+download

4. Программирование STM32F4. USART. Пример программы Приклад програми. [Електронний ресурс] – Режим доступу: URL: http://microtechnics.ru/programmirovanie-stm32f4-usart-primer-programmy/

5. Микроконтроллеры AVR. UART. Использование прерываний. [Електронний ресурс] – Режим доступу: URL: http://microtechnics.ru/mikrokontrollery-avr-uart-ispolzovanie-preryvanij/

6. Болл Стюарт Р. Аналоговые интерфейсы микроконтроллеров. – М.: Додэка-XXI, 2007. – 360 с.

7. STM32 ADC Примеры использования. Шаг 1 [Електронний ресурс] – Режим доступу: URL: http://mycontroller.ru/stm32-adc-primeryi-ispolzovaniya-shag-1/

8. Подключаем HD44780 дисплей к STM32. [Електронний ресурс] – Режим доступу: URL: http://easystm32.ru/indication/22-hd44780-and-stm32

9. STM32F4DISCOVERY STM32F4 high-performance discovery board [Електронний ресурс] – Режим доступу: URL: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/DM00039084.pdf.

10. Keil uVision середовище розробки. Джерело: https://www2.keil.com/mdk5/uvision/..