

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторного практикуму та
самостійної роботи
з дисципліни
ПРОГРАМУВАННЯ ВИМІРЮВАЛЬНИХ ПРИСТРОЇВ
для студентів спеціальності
152 „Метрологія та інформаційно-вимірювальна техніка“
денної й заочної форм навчання

Методичні вказівки до лабораторного практикуму та самостійної роботи з дисципліни «Програмування вимірювальних пристроїв» для студентів спеціальності 152 „Метрологія та інформаційно-вимірювальна техніка“ денної й заочної форм навчання / Укл.: В.І. Рева. – Запоріжжя: НУ «Запорізька політехніка», 2021. – 39 с.

Укладач: В.І. Рева, доц., канд. фіз.-мат. наук

Рецензент: О.В. Василенко, доц., канд. техн. наук

Відповідальний за випуск: А.В. Коротун, доц., канд. фіз.-мат. наук

Затверджено
на засіданні кафедри
мікро- та наноелектроніки

Протокол №5
від “ 20 “ жовтня 2020 р.

Рекомендовано до видання
НМК ФРЕТ
Протокол №3
від “ 22 “ жовтня 2020 р.

ЗМІСТ

Вступ	4
Підготовка до проведення лабораторної роботи та оформлення звіту	5
1. Лабораторна робота №1 “Робота з портами вводу/виводу”.	6
1.1 Теоретична частина	6
1.2 Приклад програми	7
1.3 Завдання	12
2. Лабораторна робота №2 “Робота з зовнішніми перериваннями”	12
2.1 Теоретична частина	12
2.2 Приклад програми	13
2.3 Завдання	15
3. Лабораторна робота №3 “Робота з АЦП. Пристрої індикації”	16
3.1 Теоретична частина.	16
3.2 Приклад програми	16
3.3 Завдання	20
4. Лабораторна робота №4 “Організація послідовної передачі інформації. Інтерфейс UART”	20
4.1 Теоретична частина.	20
4.2 Приклад програми	21
4.3 Завдання	23
5. Лабораторна робота №5 “Використання аналогового компаратора”	23
5.1 Теоретична частина.	24
5.2 Приклад програми	24
5.3 Завдання	26
6. Лабораторна робота №6 “Операційний підсилювач”	26
6.1 Теоретична частина.	26
6.2 Приклад програми	28
6.3 Завдання	29
Додаток А. Приклад оформлення титульної сторінки	31
Додаток Б Початок роботи з налагоджувальним модулем ...	32

ВСТУП

В умовах зростаючих потреб в потужності вбудованих систем зростають можливості основних компонентів таких систем - мікроконтролерів. Великим кроком у цьому процесі стала поява 32-розрядних мікроконтролерів, побудованих на базі архітектури ARM. Збільшення розрядності мало великі переваги, а також надало можливість збільшити потенційну продуктивність систем і забезпечити більший запас ресурсів при реалізації вбудованих систем.

Типовими представниками ARM-мікроконтролерів є мікроконтролери сімейства Cortex-M4, які також є одними з найбільш технологічних з апаратної точки зору. Широкий набір периферійних інтерфейсів, висока тактова частота, використання DMA – далеко не повний список можливостей, використання яких надають мікроконтролери даного сімейства.

Даний лабораторний практикум передбачає використання налагоджувального модуля STM32F3 Discovery для вивчення можливостей мікроконтролерів, побудованих на базі архітектури ARM і орієнтований на розвиток практичних навичок роботи з апаратним забезпеченням і програмними засобами для реалізації практичних завдань, в тому числі в області інформаційно-вимірювальної техніки.

ПІДГОТОВКА ДО ПРОВЕДЕННЯ ЛАБОРАТОРНОЇ РОБОТИ ТА ОФОРМЛЕННЯ ЗВІТУ

Лабораторна робота включає самостійне опрацювання теоретичного матеріалу, вивчення документації з апаратних засобів, методик проведення розрахунків, програмування і моделювання, обробку й інтерпретацію результатів. При проведенні лабораторного практикуму необхідно:

- підготуватися до експрес-опитування з теоретичного матеріалу, необхідного для виконання роботи (щоб отримати допуск до лабораторної роботи);

- підготувати і оформити план виконання лабораторної роботи;
- після виконання лабораторної роботи провести необхідні розрахунки і оформити звіт, який повинен містити:

- 1) титульний аркуш (див. Додаток А);
- 2) мету роботи;
- 3) опис словами;
- 4) алгоритм роботи у вигляді блок схеми;
- 5) необхідні математичні розрахунки роботи пристрою;
- 6) текст розробленої програми;
- 7) основні результати моделювання пристрою;
- 8) стислі висновки по роботі, в яких:

- вказати, що досліджувалось;
- вказати методику (чи метод) дослідження;
- вказати особливості алгоритму роботи пристрою;
- визначити характер роботи пристрою при моделюванні;
- відзначити подібність і відмінність результатів моделювання від теоретичних даних;
- здати звіт з виконаної лабораторної роботи та відповісти на запитання викладача (отримати залік за виконану роботу).

1 ЛАБОРАТОРНА РОБОТА

№1 “РОБОТА З ПОРТАМИ ВВОДУ/ВИВОДУ”

Мета роботи: навчитися працювати з портами вводу/виводу, організовувати код програми у вигляді функцій, організовувати базову структуру програми.

Устаткування: відладжувальна плата, модуль зі світлодіодним кубом розміром 3х3х3 (для включення окремого рівня і світлодіода використовується сигнал логічної 1), перемички для підключення між платами за необхідності.

1.1 Теоретична частина

На платі, яка використовується в ході виконання робіт, міститься відразу 6 портів вводу/виводу. Більшість з них є 16-розрядними. Для управління портами використовується 10 регістрів.

Є такі режими роботи висновків порту:

- робота в якості виходу;
- робота в якості входу;
- робота в якості аналогового виводу (використовується при роботі з АЦП і ЦАП);
- використання в якості виводу виконання альтернативної функції пристроїв, які входять до складу мікроконтролера (наприклад, UART, SPI і ін.).

При роботі в режимі виводу можливе використання 2 режимів: двухтактного виходу (push-pull) і відкритого стоку (open-drain). Режим двухтактного виходу використовується в більшості випадків, а робота в режимі відкритого стоку необхідно, наприклад, при використанні інтерфейсу передачі даних I2C. До кожного виводу є можливість підключення підтягуючого резистора. Є можливість підтягти сигнал на виході до лог. 0 або лог. 1. За замовчуванням підтягуючі резистори не використовуються.

Якщо необхідно отримати дані у вигляді логічних станів, то можна перевести вивід в режим входу. При такій конфігурації з'являється можливість зчитувати дані з виводів.

Проте, в більшості випадків для зчитування даних використовуються периферійні пристрої. Вони використовують виводи порту в

якості власних виводів. Такий режим роботи отримав назву альтернативної функції. При цьому відбувається відображення виводів порту на виводи пристрою. Такий режим розглянуто в наступних роботах.

1.2 Приклад програми

У прикладі розглядається режим роботи виводів у якості двухтактного виходу для простого ввімкнення/вимкнення пристроїв. Перейдемо до більш детального розгляду практичного прикладу.

Як приклад використовуємо роботу зі світлодіодним 3D-кубом і створенням візуальних ефектів на його основі.

Принцип, який використовується для створення ефектів з використанням світлодіодного 3D-куба, базується на використанні динамічної індикації. Використовується по чергове ввімкнення/вимкнення елементів схеми, яке непомітно для людського ока через те, що відбувається з великою частотою. Конкретна реалізація залежить від того, як організовано підключення світлодіодів в структурі куба.

Найчастіше використовується схема підключення, при якій виділяються виводи для управління ввімкненими рівнями, а також світлодіодів на ввімкненому рівні. Таким чином, для управління кубом розмірністю $3 \times 3 \times 3$ світлодіода необхідно 3 виходи для управління рівнями і 9 виходів для управління світлодіодами на рівні. Будемо вважати, що використовується схема із загальним катодом для організації структури. У такому випадку для включення одного світлодіода слід подати сигнал логічної одиниці на вихід, який відповідає за включення одного з рівнів, а також подати логічну одиницю на один з виходів, який відповідає за підключення світлодіодів.

Перейдемо до розгляду схеми підключення компонентів в структурі куба (рис. 1.1).

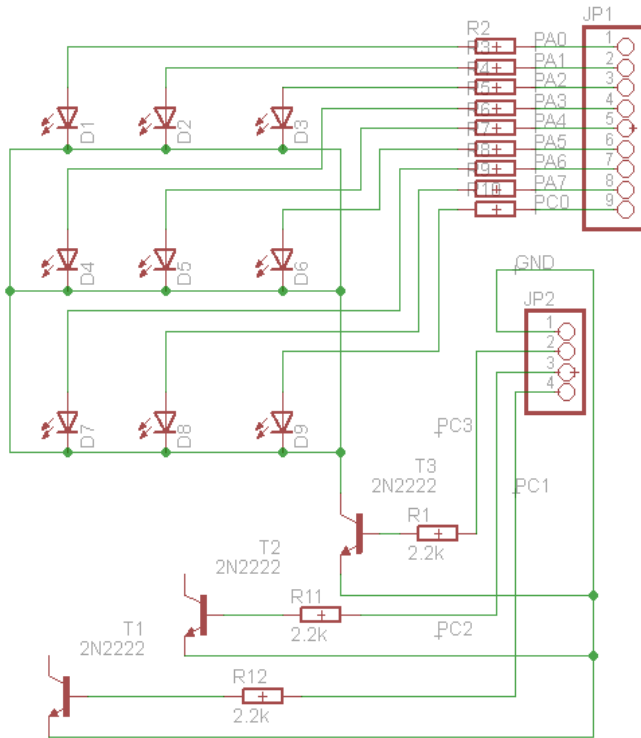


Рисунок 1.1 – Схема підключення компонентів при створенні куба

Все катоди світлодіодів на одному рівні об'єднуються і через транзистор підключаються до землі. В даному випадку транзистор працює в режимі ключа: при подачі логічної одиниці на базу, транзистор відкривається і з'являється можливість включити світлодіоди на рівні. Використання декількох рівнів призводить до того, що світлодіоди зручно групувати в стовпець по 3 елементи. Але слід підкреслити, що при цьому слід використовувати резистор з більшим опором, ніж при підключенні одного світлодіода. В даному випадку використовуються резистори з опором 220 Ом.

Розглянемо, як необхідно виконати підключення між налагоджувальний модулем і платою зі світлодіодним кубом. Підключення вказані в таблиці 1. Також слід перевірити підключення за допомогою рис. 1.1.

Таблиця 1.1 – З'єднання між модулями

Номер виводу на платі з кубом	Вивід на модулі
1	PC1
2	PC2
3	PC3
4	GND
5	PC0
6	PA7
7	PA6
8	PA5
9	PA4
10	PA3
11	PA2
12	PA1
13	PA0

Програмна реалізація ефекту площини, що обертається представлена далі. Основним моментом, на який слід звернути увагу, є ініціалізація портів вводу/виводу. Для цього використовується екземпляр структури типу `GPIO_InitTypeDef`. Спочатку виконується ініціалізація структури, після чого встановлюються параметри ініціалізації порту: режим роботи, швидкість, виводи, використання підтягуючих резисторів резисторів. Після установки необхідних значень викликається функція ініціалізації на основі значень, які містить структура - `GPIO_Init`. Також важливим моментом є виклик функції включення тактування пристрою. Це важливо, оскільки без включення тактування пристрої жоден периферійний модуль не виконуватиме заданих функцій. Підключення до тактування виконується викликом функції типу `RCC_AxxxPeriphClockCmd` (замість xxx підставити позначення шини, до якої підключається даний пристрій, - воно може мати значення АНВ, АРВ1, АРВ2; значення можна знайти по документації або за допомогою програмного коду, знайшовши визначення `RCC_AxxxPeriph_ууу`, де ууу - пристрій, який необхідно підключити до тактування, наприклад, `GPIOA`)

Для підвищення зрозумілості програмного коду використовуються визначення за допомогою директив препроцесора `#define` для позначення виходів, призначених для управління світлодіодами, зна-

чень затримок. Затримки в даному випадку реалізовані на основі таймера, але, оскільки таймери не є основною темою при виконанні даної роботи, то досить просто використовувати готову функцію затримки.

Лістинг 1.1 – Приклад програми

```
#include "stm32f30x_rcc.h"
#include "stm32f30x_gpio.h"
#include "stm32f30x_tim.h"
#define PORTA_LED 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 PORTC_LED GPIO_Pin_0
#define PORTC_LEVELS GPIO_Pin_1 | \
GPIO_Pin_2 | \
GPIO_Pin_3
#define LAYER1 GPIO_Pin_1
#define LAYER2 GPIO_Pin_2
#define LAYER3 GPIO_Pin_3
#define SMALL_DELAY 10
#define BIG_DELAY 1000
void init(void);
void delay_ms(int ms);
void set_layer_data(int layer, int data);
void turn_needed_layer(int layer);
void set_pict_by_layer(int data1, int data2, int data3);
void set_pict_by_layer_for(int time, int data1, int data2, int data3);
int main() {
    init();
    delay_ms(2);
    while(1) {
        set_pict_by_layer_for(BIG_DELAY, 0x7, 0x38, 0x1C0);
        set_pict_by_layer_for(BIG_DELAY, 0x38, 0x38, 0x38);
        set_pict_by_layer_for(BIG_DELAY, 0x1C0, 0x38, 0x7);
        set_pict_by_layer_for(BIG_DELAY, 0x0, 0x1ff, 0x0);
        set_pict_by_layer_for(BIG_DELAY, 0x7, 0x38, 0x1C0);
        set_pict_by_layer_for(BIG_DELAY, 0x38, 0x38, 0x38);
        set_pict_by_layer_for(BIG_DELAY, 0x1C0, 0x38, 0x7);
        set_pict_by_layer_for(BIG_DELAY, 0x0, 0x1ff, 0x0);
    }
}
void set_pict_by_layer_for(int time, int data1, int data2, int data3) {
    int times;
    int i;
    times = time / 30;
    for ( i = 0; i < times; ++i ) {
```

```

set_pict_by_layer(data1, data2, data3);
}
}
void set_pict_by_layer(int data1, int data2, int data3) {
set_layer_data(LAYER1, data1);
delay_ms(SMALL_DELAY);
set_layer_data(LAYER2, data2);
delay_ms(SMALL_DELAY);
set_layer_data(LAYER3, data3);
delay_ms(SMALL_DELAY);
}
void set_layer_data(int layer, int data) {
turn_needed_layer(layer);
GPIO_Write(GPIOA, data & 255);
GPIO_WriteBit(GPIOC, GPIO_Pin_0, ((data & 256) > 0) ? Bit_SET :
Bit_RESET);
}
void turn_needed_layer(int layer) {
int i;
for ( i = 1; i <= 3; i++ ) {
GPIO_WriteBit(GPIOC, 1 << i, ((1 << i) == layer) ? Bit_SET : Bit_RESET);
}
}
void delay_ms(int ms) {
TIM2->CNT = 0;
TIM2->PSC = 16000;
TIM2->ARR = ms;
TIM_Cmd(TIM2, ENABLE);
while (TIM_GetFlagStatus(TIM2, TIM_FLAG_Update) == RESET);
TIM_ClearFlag(TIM2, TIM_FLAG_Update);
TIM_Cmd(TIM2, DISABLE);
}
void init(void) {
GPIO_InitTypeDef gpio_init;
TIM_TimeBaseInitTypeDef time_init;
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA, ENABLE);
GPIO_StructInit(&gpio_init);
gpio_init.GPIO_Pin = PORTA_LED;
gpio_init.GPIO_Mode = GPIO_Mode_OUT;
gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
gpio_init.GPIO_OType = GPIO_OType_PP;
GPIO_Init(GPIOA, &gpio_init);
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOC, ENABLE);
gpio_init.GPIO_Pin = PORTC_LED | PORTC_LEVELS;
GPIO_Init(GPIOC, &gpio_init);
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
TIM_TimeBaseStructInit(&time_init);
time_init.TIM_CounterMode = TIM_CounterMode_Up;
time_init.TIM_RepetitionCounter = 0;
TIM_TimeBaseInit(TIM2, &time_init);
}

```

Для спрощення процесу створення ефектів реалізовано кілька функцій:

1. Включення необхідного рівня.
2. Включення світлодіодів на зворотному рівні.
3. Перемикання між рівнями з установкою необхідних рівнів сигналів для зміни зображення.
4. Затримка для контролю часових проміжків при відображенні.

Реалізація цих функцій дозволяє створити візуальний ефект (в даному прикладі - обертання площини) за допомогою декількох викликів функцій в циклі головної функції. Для цього необхідно визначити, які числові значення відповідають включенню світлодіодів на рівні і встановити правильну послідовність їх передачі на рівні.

1.3 Завдання

Реалізувати ефект обертання площини навколо осі, розміщеної діагонально.

2 ЛАБОРАТОРНА РОБОТА №2 “РОБОТА З ЗОВНІШНІМИ ПЕРЕРИВАННЯМИ”

Мета роботи: навчитися використовувати зовнішні переривання для виконання програмних функцій по зовнішньому сигналу.

Устаткування: оцінний модуль STM32F3 Discovery, модуль з 3 тактовими кнопками для підключення до плати.

2.1 Теоретична частина

Переривання – важлива подія в роботі мікроконтролера, яке може привести до припинення виконання основної програми і обробці даного переривання. Робота з перериваннями – важлива і основна складова при написанні програм для мікроконтролерів.

У даній роботі розглянемо використання зовнішніх переривань. Зовнішні переривання - переривання, які відбуваються внаслідок зміни логічного рівня сигналу на одному з виводів порту вводу/виводу.

Такий тип зовнішніх переривань є найбільш поширеним і використовується в STM32F303VC.

Оскільки до цього моменту переривання не використовувалися, то слід зазначити особливості використання переривань для даного апаратного забезпечення. Контроль роботи з перериваннями виконується за допомогою вбудованого контролера вектора переривань (Nested Vector Interrupt Controller, NVIC). З його допомогою виконується дозвіл і заборона переривань, установка пріоритетів переривань.

Управління зовнішніми перериваннями як окремим видом переривань виконується за допомогою модуля SYSCFG, а також модулями EXTI.

2.2 Приклад програми

В даному прикладі розглянемо простий приклад реакції на натискання кнопки за допомогою зовнішніх переривань. При натисканні кнопки, призначеної для програмування користувачем, відбувається почергове перемикавання двох світлодіодів (синього і червоного на платі).

Для програмної реалізації корисно знати, що імена всіх обробників переривань можна знайти в асемблерному файлі `startup_stm32f30x.s`. Обробник переривань являє собою функцію, яка не приймає параметрів і не повертає значень, тому в обох випадках слід вказати `void`. Ім'я переривання можна знайти в файлі `stm32f30x.c`, наприклад, `EXTI0_IRQn`; спільною рисою є те, що всі імена закінчуються на `IRQn`.

У порівнянні з попереднім прикладом додається використання програмних модулів для управління зовнішніми перериваннями, для управління вектором переривань і модулем SYSCFG.

Програмна реалізація прикладу наведено нижче. У ній слід звернути увагу на використання вже знайомої послідовності виконання ініціалізації периферійних пристроїв в складі мікроконтролера, а також на те, як саме виконується реалізація обробника переривань (він являє собою функцію).

Лістинг 2.1 – Приклад програми

```

#include "stm32f30x_rcc.h"
#include "stm32f30x_gpio.h"
#include "stm32f30x_exti.h"
#include "stm32f30x_misc.h"
#include "stm32f30x_syscfg.h"
typedef void (*action)(void);
action button_action;
void turn_on_red(void);
void turn_on_blue(void) {
    GPIO_SetBits(GPIOE, GPIO_Pin_8);
    GPIO_ResetBits(GPIOE, GPIO_Pin_9);
    button_action = turn_on_red;
}
void turn_on_red(void) {
    GPIO_SetBits(GPIOE, GPIO_Pin_9);
    GPIO_ResetBits(GPIOE, GPIO_Pin_8);
    button_action = turn_on_blue;
}
void EXTI0_IRQHandler(void) {
    EXTI_ClearITPendingBit(EXTI_Line0);
    button_action();
}
void init(void);
int main() {
    init();
    button_action = turn_on_blue;
    while(1) {
    }
}
void init(void) {
    GPIO_InitTypeDef gpio_init;
    EXTI_InitTypeDef exti_init;
    NVIC_InitTypeDef nvic_init;
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOE,
        ENABLE);
    GPIO_StructInit(&gpio_init);
    gpio_init.GPIO_Mode = GPIO_Mode_OUT;
    gpio_init.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_9;
    gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
    gpio_init.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOE, &gpio_init);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG,
        ENABLE);
    exti_init.EXTI_Line = EXTI_Line0;
    exti_init.EXTI_LineCmd = ENABLE;
    exti_init.EXTI_Mode = EXTI_Mode_Interrupt;
    exti_init.EXTI_Trigger = EXTI_Trigger_Rising;
    EXTI_Init(&exti_init);
    SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA,

```

```

EXTI_PinSource0);
nvic_init.NVIC_IRQChannel = EXTI0_IRQn;
nvic_init.NVIC_IRQChannelCmd = ENABLE;
nvic_init.NVIC_IRQChannelSubPriority = 1;
nvic_init.NVIC_IRQChannelPreemptionPriority =
1;
NVIC_Init(&nvic_init);
}

```

Особливістю запропонованої програмної реалізації є використання покажчика на функцію для виконання дій в обробнику зовнішнього переривання. Для цього описаний новий тип покажчика, action, і описана змінна даного типу. Згадана змінна ініціалізована таким чином, щоб посилатися на одну з функцій перемикання світлодіодів. У цих функціях відбувається присвоювання вказівником значення адреси іншої функції. Таким чином, забезпечується зміна функціональності при виникненні зовнішнього переривання.

Слід зазначити, що, відповідно до 10 правилами Хольцмана, які описують рекомендації до розробки програмного забезпечення для вбудованих систем, використання покажчиків на функції є практикою, від якої необхідно відмовлятися при побудові складних систем, які вимагають високої надійності, проте даний приклад не є складним, а використання покажчиків на функції спрощує реалізацію зміни функціональності і підвищує зрозумілість програмного коду. За аналогією з об'єктно-орієнтованим програмуванням, можна сказати, що таким чином забезпечується поліморфну поведінку системи.

2.3 Завдання

Підключити до відладочної платі модуль з 3 кнопками і реалізувати перемикання світлодіодів після натискання кнопок на модулі. Світлодіоди вибрати самостійно.

3 ЛАБОРАТОРНА РОБОТА

№3 “РОБОТА З АЦП. ПРИСТРОЇ ІНДИКАЦІЇ”

Мета роботи: дослідити принцип роботи мікроконтролера, з вбудованим АЦП, продемонструвати використання динамічної індикації при відображенні результатів вимірювань.

Устаткування: відлагоджувальний модуль, модуль 2 двухразрядного семісегментним індикатором, потенціометр, вольтметр.

3.1 Теоретична частина

Аналогово-цифровий перетворювач (АЦП) – пристрій, призначений для перетворення величини вхідної напруги в числове значення. У мікроконтролері, який використовується на відлагоджувальному модулі, доступні 4 АЦП з великою кількістю каналів для підключення сигналів. Є можливість налаштування розрядності АЦП, встановивши значення 6, 8, 10 або 12 біт. Швидкість перетворення для швидких і повільних каналів вхідної напруги по документації оцінюється наступними значеннями:

- для швидких - 0,19 мкс для 12-бітного перетворення (5,1 Мвибірки / с);
- для повільних - 0,21 мкс для 12 бітного перетворення (4,9 Мвибірки / с).

Ще більшої швидкості перетворення можна домогтися, зменшивши значення розрядності для перетворення.

Крім того, присутні внутрішні підключення між іншими периферійними пристроями (внутрішній датчик температури, операційні підсилювачі) і АЦП, що спрощує їх практичне використання при роботі та перевірці схем проектів.

3.2 Приклад програми

Як приклад роботи з АЦП розглянемо реалізацію вимірювання напруги на вході АЦП і виведення отриманої інформації на двухразрядний семісегментний індикатор (в даному прикладі використовується індикатор із загальним анодом, який не може відображати десяткову точку).

Як джерело вхідної напруги скористаємося підключенням потенціометра.

Для зменшення кількості використовуваних контактів реалізований принцип динамічної індикації. Виконується попеременне ввімкнення і вимикання конкретного розряду індикатора з виведенням інформації та затримкою. Для реалізації динамічної індикації використовується 9 ліній вводу/виводу:

- 2 для вибору індикатора, на який виводиться значення;
- 7 інших для виведення інформації.

Для програмної реалізації зручно, щоб ці 2 групи належали різним портам вводу/виводу, а також при підключенні розташовувалися по зростанню. Проте, це не є обов'язковою вимогою, оскільки лише трохи ускладнює процес програмування через необхідність складання таблиць відповідності для виводів портів і значень, які необхідно вивести.

Програмна реалізація приклад наведено далі.

Лістинг 3.1 – Приклад програми

```
#include "stm32f30x_rcc.h"
#include "stm32f30x_gpio.h"
#include "stm32f30x_adc.h"
#define DIGIT_PORT GPIOB
#define DIGIT_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 DIGIT_SELECTION_PORT GPIOB
#define DIGIT_SELECTION_PINS GPIO_Pin_4 | GPIO_Pin_5
#define FIRST_DIGIT_PIN GPIO_Pin_4
#define SECOND_DIGIT_PIN GPIO_Pin_5
int dig_code[] = { 0x01, 0x3d, 0x12, 0x18, 0x2c, 0x88, 0x80, 0x39,
0x00, 0x08 };
void init(void);
void delay(int count);
void show_converted_value(uint16_t convValue);
void turn_on_first_digit(void);
void turn_on_second_digit(void);
void turn_off_first_digit(void);
void turn_off_second_digit(void);
void set_digit(int);
int main() {
```

```

static uint16_t convertedValue;
init();
while (!ADC_GetFlagStatus(ADC1, ADC_FLAG_RDY));
ADC_StartConversion(ADC1);
while(1) {
    delay(10000);
    while (ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);
    convertedValue = ADC_GetConversionValue(ADC1);
    show_converted_value(convertedValue);
}
}

void init(void) {
    GPIO_InitTypeDef gpio_init;
    ADC_InitTypeDef adc_init;
    ADC_CommonInitTypeDef adc_common;
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA, ENABLE);
    GPIO_StructInit(&gpio_init);
    gpio_init.GPIO_Pin = GPIO_Pin_0;
    gpio_init.GPIO_Mode = GPIO_Mode_AN;
    GPIO_Init(GPIOA, &gpio_init);
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOD | RCC_AHBPeriph_GPIOB, ENA-
BLE);
    gpio_init.GPIO_Mode = GPIO_Mode_OUT;
    gpio_init.GPIO_OType = GPIO_OType_PP;
    gpio_init.GPIO_Pin = DIGIT_PINS;
    gpio_init.GPIO_PuPd = GPIO_PuPd_DOWN;
    gpio_init.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(DIGIT_PORT, &gpio_init);
    gpio_init.GPIO_Pin = DIGIT_SELECTION_PINS;
    gpio_init.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_Init(DIGIT_SELECTION_PORT, &gpio_init);
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_ADC12, ENABLE);
    RCC_ADCLKConfig(RCC_ADC12PLLCLK_Div2);
    ADC_VoltageRegulatorCmd(ADC1, ENABLE);
    delay(10000);
    ADC_SelectCalibrationMode(ADC1, ADC_CalibrationMode_Single);
    ADC_StartCalibration(ADC1);
    while (ADC_GetCalibrationStatus(ADC1) == RESET);
    ADC_GetCalibrationValue(ADC1);
    ADC_StructInit(&adc_init);
    adc_init.ADC_DataAlign = ADC_DataAlign_Right;
    adc_init.ADC_Resolution = ADC_Resolution_12b;
    adc_init.ADC_ContinuousConvMode = ADC_ContinuousConvMode_Enable;
    adc_init.ADC_NbrOfRegChannel = 1;
    ADC_Init(ADC1, &adc_init);
    ADC_CommonStructInit(&adc_common);
    adc_common.ADC_Mode = ADC_Mode_Independent;
    ADC_CommonInit(ADC1, &adc_common);
    ADC-RegularChannelConfig(ADC1, ADC_Channel_1, 1,
ADC_SampleTime_7Cycles5);
    ADC_Cmd(ADC1, ENABLE);
}

```

```

}
void delay(int count) {
for ( ; count > 0; count--);
}
void show_converted_value(uint16_t convValue) {
char first;
char second;
int i;
double voltageValue;
voltageValue = (double) convValue * 3.0 / 4095;
first = (int)voltageValue;
second = ((int)(voltageValue * 10) % 10);
for (i = 0; i < 500; i++) {
turn_off_first_digit();
turn_on_second_digit();
set_digit(first);
delay(10000);
turn_on_first_digit();
turn_off_second_digit();
set_digit(second);
delay(10000);
}
}
void turn_on_first_digit(void) {
GPIO_SetBits(DIGIT_SELECTION_PORT, FIRST_DIGIT_PIN);
}
void turn_on_second_digit(void) {
GPIO_SetBits(DIGIT_SELECTION_PORT, SECOND_DIGIT_PIN);
}
void turn_off_first_digit(void) {
GPIO_ResetBits(DIGIT_SELECTION_PORT, FIRST_DIGIT_PIN);
}
void turn_off_second_digit(void) {
GPIO_ResetBits(DIGIT_SELECTION_PORT, SECOND_DIGIT_PIN);
}
void set_digit(int index) {
GPIO_Write(DIGIT_PORT, dig_code[index]);
}

```

Так само, як для аналогового компаратора і операційного підсилювача, вхід АЦП це вивід, який працює в аналоговому режимі. Далі найбільшу увагу слід приділити тому, як виконується налаштування АЦП. Це вимагає порівняно великої кількості кроків, в порівнянні з розглянутими раніше пристроями. Слід зазначити те, що виконується калібрування АЦП, а також необхідно використовувати дві структури для ініціалізації різних типів: перша для ініціалізації конкретних властивостей АЦП, а інша - для ініціалізації загальних характеристик АЦП

в складі контролера. Перевірка готовності виконується перед входом в головний цикл програми. В тому ж місці починається виконання перетворення (виклик функції `ADC_StartConversion`). Оскільки при ініціалізації було поставлено параметр виконання постійного перетворення, то виконання подальших перетворень не вимагає повторного виклику даної функції.

3.3 Завдання

Детально ознайомтеся із запропонованим прикладом і продемонструйте його роботу. Для обраної схеми підключення скласти таблицю відповідності висновків і сегментів на індикаторі. За умови зміни послідовності підключення (власний варіант), продемонструвати нові значення в масиві для виведення. Перемістити реалізацію динамічної індикації з функції в головний цикл програми.

4 ЛАБОРАТОРНА РОБОТА

№4 “ОРГАНІЗАЦІЯ ПОСЛІДОВНОЇ ПЕРЕДАЧІ ІНФОРМАЦІЇ. ІНТЕРФЕЙС UART”

Мета роботи: дослідити можливості передачі даних між пристроями за допомогою послідовного інтерфейсу UART, організувати зв'язок між ПК і мікро контролером.

Устаткування: відлагоджувальний модуль, модуль з мікросхемою перетворювача рівнів MAX3232, перехідник USB/COM.

4.1 Теоретична частина

Універсальний асинхронний прийомо-передатчкі (англ. Universal Asynchronous Receiver-Transmitter, UART) – один з найбільш поширених інтерфейсів, який зустрічається в мікроконтролерах. Поширеність пояснюється простотою і невибагливістю до ресурсів: для організації передачі даних використовуються лише 2 виводи (зазвичай, вони називаються Rx і Tx). При підключенні двох пристроїв UART лінії передачі і прийому з'єднуються різноіменно. Свій розви-

ток отримав в тому, що деякі реалізації інтерфейсу в периферії підтримують синхронний режим передачі – в такому випадку модуль називають USART. Він використовує третю лінію для подачі тактового сигналу. Крім того, можуть підключатися інші висновки для виконання службових функцій.

4.2 Приклад програми

У практичному прикладі розглянемо роботу з UART для прийому даних.

Для виконання завдання знадобиться кабель-перехідник USB/COM, а також модуль з мікросхемою перетворювача рівнів MAX3232, яка забезпечує перетворення з рівнів сигналу COM-порту до рівнів TTL-логіки, і може живитися від 3 В (такий вивід присутній на отлагодочній платі). Схема підключення між перехідником, модулем і платою представлена на рис. 4.1.

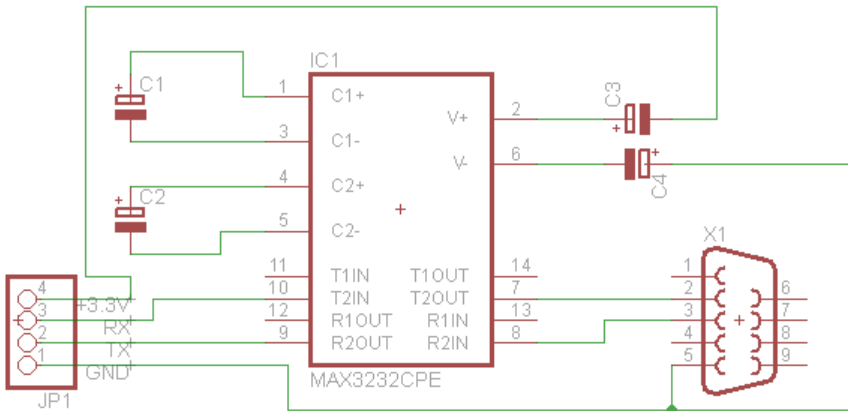


Рисунок 4.1 – Схема підключення перетворювача рівнів

Зверніть увагу на те, що схема на рисунку є спрощеною, оскільки не відображає виводи 15 і 16.

Для перевірки програмної реалізації скористаємося термінальною програмою Putty. Перейти в режим відладки (комбінація клавіш Ctrl + F5 або команда меню Debug\ Start/Stop Debug Session). При відправці символу продемонструвати проходження точки зупинки і отриманий символ.

Програмна реалізація, яка використовується при виконання вищеописаних дій, представлена нижче.

Лістинг 4.1 – Приклад програми

```
#include "stm32f30x_rcc.h"
#include "stm32f30x_gpio.h"
#include "stm32f30x_usart.h"
#include "stm32f30x_misc.h"
void init(void);
void delay(int count);
void USART1_IRQHandler(void) {
    char recData;
    if (USART_GetITStatus(USART1, USART_IT_RXNE) == SET) {
        USART_ClearITPendingBit(USART1, USART_IT_RXNE);
        recData = USART_ReceiveData(USART1);
        recData++;
    }
}
int main() {
    init();
    while(1) {
    }
}
void init(void) {
    GPIO_InitTypeDef gpio_init;
    USART_InitTypeDef usart_init;
    NVIC_InitTypeDef nvic_init;
    EXTI_InitTypeDef exti_init;
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA, ENABLE);
    GPIO_StructInit(&gpio_init);
    gpio_init.GPIO_Pin = GPIO_Pin_9 | GPIO_Pin_10; // 9 - Tx, 10 - Rx
    gpio_init.GPIO_Mode = GPIO_Mode_AF;
    gpio_init.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &gpio_init);
    GPIO_PinAFConfig(GPIOA, GPIO_PinSource9, GPIO_AF_7);
    GPIO_PinAFConfig(GPIOA, GPIO_PinSource10, GPIO_AF_7);
    nvic_init.NVIC_IRQChannel = USART1_IRQn;
    nvic_init.NVIC_IRQChannelCmd = ENABLE;
    nvic_init.NVIC_IRQChannelSubPriority = 1;
    nvic_init.NVIC_IRQChannelPreemptionPriority = 1;
    NVIC_Init(&nvic_init);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
    USART_StructInit(&usart_init);
    usart_init.USART_BaudRate = 19200;
    usart_init.USART_WordLength = USART_WordLength_8b;
    usart_init.USART_StopBits = USART_StopBits_1;
    usart_init.USART_Parity = USART_Parity_No;
    usart_init.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    usart_init.USART_HardwareFlowControl =
    USART_HardwareFlowControl_None;
```

```

USART_Init(USART1, &usart_init);
USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
USART_Cmd(USART1, ENABLE);
}
void delay(int count) {
for ( ; count > 0; count--);
}

```

Прийом символів виконується за допомогою переривання. Слід зазначити, що крім налаштування переривання в контролері переривань, слід також дозволити переривання для самого пристрою. Це робиться за допомогою біта в регістрі управління або функцією USART_ITConfig() зі стандартної бібліотеки. Слід це запам'ятати, оскільки на цьому прикладі можна бачити, які імена мають функції для настройки переривань окремих пристроїв: Periph_ITConfig (), де Periph - тип пристрою. Як параметри найчастіше виступають назву периферійного пристрою, скорочена назва переривання і новий стан біта, який відповідає за дозвіл переривання, відповідно, дозволено або заборонено.

4.3 Завдання

Ознайомитися з прикладом програми. На основі прикладу реалізувати функціональність включення світлодіодів при отриманні певного сигналу (символу). Значення символів і світлодіодів вибрати самостійно (8 світлодіодів на платі підключені в висновків PE8-PE15).

5 ЛАБОРАТОРНА РОБОТА №5 “ВИКОРИСТАННЯ АНАЛОГОВОГО КОМПАРАТО- РА”

Мета роботи: дослідити можливість використання аналогового компаратора в складі мікроконтролера на отладоочной платі.

Устаткування: відлагоджувальна плата, мультиметр або вольтметр.

5.1 Теоретична частина

Компаратор - аналоговий електронний пристрій, призначений для порівняння величин сигналів. Зазвичай він має два входи, на які подаються сигнали, які порівнюються. Сигнал на виході залежить від результатів порівняння: якщо перший сигнал більше - на виході лог. 1, якщо навпаки - лог. 0.

STM32F303VC містить 7 компараторів, які можуть використовуватися як самостійні пристрої або спільно з таймерами. Вони можуть використовуватися для різних функцій в складі самого пристрою:

- пробудження з режиму низького споживання енергії по аналоговому сигналу;
- порівняння сигналів;

Таким чином, даний пристрій використовується і як самостійний модуль, так і в складі апаратного комплексу.

5.2 Приклад програми

Як приклад розглянемо реалізацію перемикання рівня сигналу з 0 в 1 при перевищенні заданого рівня напруги. Це фактично і буде використання простого компаратора. Завдяки тому, що є можливість вибору сигналу для подачі напруги, необхідно подавати сигнал тільки на один з входів компаратора. В даному випадку в якості такого джерела вибрано значення половини напруги живлення мікроконтролера.

Програмна реалізація прикладу представлена далі.

Лістинг 5.1 – Приклад програми

```
#include "stm32f30x_rcc.h"
#include "stm32f30x_gpio.h"
#include "stm32f30x_syscfg.h"
#include "stm32f30x_comp.h"
#include "stm32f30x_misc.h"
#include "stm32f30x_exti.h"
void init(void);
void COMP1_2_3_IRQHandler(void) {
    int foo;
    uint32_t level;
    if (EXTI_GetITStatus(EXTI_Line21) == SET) {
        EXTI_ClearITPendingBit(EXTI_Line21);
        foo = 0;
        level = COMP_GetOutputLevel(COMP_Selection_COMP1);
        if (level == COMP_OutputLevel_High)
            foo = 1;
    }
}
```

```

}
}
int main() {
init();
while(1) {
}
}
void init(void) {
GPIO_InitTypeDef gpio_init;
COMP_InitTypeDef comp_init;
NVIC_InitTypeDef nvic_init;
EXTI_InitTypeDef exti_init;
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA, ENABLE);
GPIO_StructInit(&gpio_init);
gpio_init.GPIO_Mode = GPIO_Mode_AN;
gpio_init.GPIO_PuPd = GPIO_PuPd_DOWN;
gpio_init.GPIO_Pin = GPIO_Pin_1;
GPIO_Init(GPIOA, &gpio_init);
gpio_init.GPIO_Mode = GPIO_Mode_AF;
gpio_init.GPIO_Pin = GPIO_Pin_0;
GPIO_Init(GPIOA, &gpio_init);
GPIO_PinAFConfig(GPIOA, GPIO_PinSource0, GPIO_AF_8);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
COMP_StructInit(&comp_init);
comp_init.COMP_Mode = COMP_Mode_HighSpeed;
comp_init.COMP_OutputPol = COMP_OutputPol_NonInverted;
comp_init.COMP_NonInvertingInput = COMP_NonInvertingInput_IO1;
comp_init.COMP_InvertingInput = COMP_InvertingInput_1_2VREFINT;
COMP_Init(COMP_Selection_COMP1, &comp_init);
COMP_Cmd(COMP_Selection_COMP1, ENABLE);
exti_init.EXTI_Line = EXTI_Line21;
exti_init.EXTI_LineCmd = ENABLE;
exti_init.EXTI_Mode = EXTI_Mode_Interrupt;
exti_init.EXTI_Trigger = EXTI_Trigger_Rising;
EXTI_Init(&exti_init);
nvic_init.NVIC_IRQChannel = COMP1_2_3_IRQn;
nvic_init.NVIC_IRQChannelCmd = ENABLE;
nvic_init.NVIC_IRQChannelSubPriority = 1;
nvic_init.NVIC_IRQChannelPreemptionPriority = 1;
NVIC_Init(&nvic_init);
}

```

При роботі з компаратором видно, що є можливість підключити до входів різні значення сигналів. За аналогією з операційним підсилювачем входи компаратора називаються інвертується і неінвертуючий.

Переривання від аналогового компаратора в даному мікроконтролері реалізовані як зовнішні переривання (лінія 21). Тому для їх ви-

користання необхідно підключити ті ж модулі, що і для звичайного зовнішнього переривання. Крім того, за роботу компараторів відповідає вже згадуваний модуль SYSCFG, тому при використанні компараторів подача тактирования на даний модуль є обов'язковим.

5.3 Завдання

На основі прикладу, який пропонується для використання, а також документації для мікроконтролера, розробити програму, яка буде виконувати функцію контролю за рівнем напруги з індикацією: поки напруга нижче заданого рівня, потрібно включити один з світлодіодів, при переході через умовну допустиму межу, включати інший світлодіод, повідомляючи про це.

6 ЛАБОРАТОРНА РОБОТА №6 “ОПЕРАЦІЙНИЙ ПІДСИЛЮВАЧ”

Мета роботи: дослідити можливості плати при роботі аналоговими сигналами, особливості використання операційного підсилювача.

Устаткування: відладжувальна плата, набір перемичок для з'єднання виводів, мультиметр або вольтметр.

6.1 Теоретична частина

До складу мікроконтролера STM32F303VC також входять чотири операційних підсилювача.

Передбачається використання підсилювача в двох режимах в залежності від використання резисторів в схемі:

- самостійний режим (необхідно підключати зовнішні резистори, які визначають рівень посилення);
- програмний режим (виконується підключення до внутрішніх резисторам, які визначають рівень посилення);

За допомогою встановленого операційного підсилювача (ОП) можна побудувати такі відомі схеми, як диференційний підсилювач

або неінвертуючий підсилювач. У реальному використанні це дозволить уникнути установки додаткового пристрою за умови достатності параметрів, які забезпечує ОП в складі контролера. Проте, виробник, застерігає, що існують труднощі використання ОП, тому не рекомендує використовувати ці модулі в складних системах.

Режим, при якому управління резисторами, підключеними до ОП, виконується програмно, за документацією позначається як Programmable Gain Amplifier (PGA). Фактично в даному режимі маємо справу з інвертуючим підсилювачем. Схема підключення для неінвертуючий підсилювача представлена на рис. 6.1.

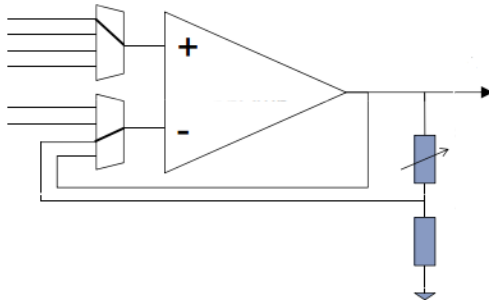


Рисунок 6.1 – Схема внутрішнього підключення при роботі в режимі неінвертуючого підсилювача

Значення вихідної напруги при такій схемі підключення визначається як $V_{out} = V_{in} \cdot \left(1 + \frac{R_2}{R_1}\right)$, де V_{in} – вхідна напруга; R_1, R_2 – значення опору резисторів.

Як зрозуміло з рисунку, значення опору другого резистора можна програмно вказати зі списку доступних значень. Для даного режиму доступні значення коефіцієнта посилення (одиниця вже врахована): 2, 4, 8, 16.

6.2 Приклад програми

Ознайомлення з роботою ОП у складі мікроконтролера проведемо на прикладі, який також використовує ЦАП для генерації напруги,

яке за допомогою ОП в режимі неінвертуючий підсилювача збільшується в 2 рази.

Оскільки використання виводів портів вводу/виводу не є додатковою функцією їх роботи, то знайти, які саме виводи будуть використовуватися при роботі ОП можна скористатися плагіном MicroXplorer для середовища розробки Eclipse. У ньому можна вибрати пристрій і дізнатися, як саме використовується вивід мікроконтролера (рис. 6.2).

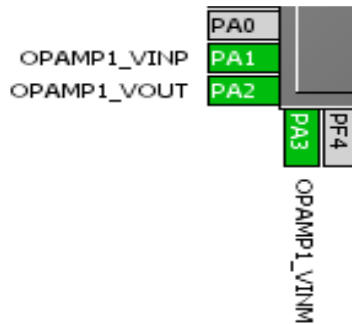


Рисунок 6.2 – Функціональне призначення виводів при використанні ОП

Найбільше слід звернути увагу на налаштування ОП. Для цього слід вказати, які лінії використовуються в якості входів. У нашому випадку вказуємо, що слід використовувати в якості неінвертуючого входу 4 лінію, якої відповідає 1 вивід порту А. На цю лінію подається напруга, яка генерується за допомогою ЦАП. Значення напруга на виході 4 порту А - приблизно 0,77 В. Поєднавши за допомогою перемички обидва Піна, слід перевірити напругу на виході АЦП – Пін 2 порту А. Показники вольтметра складуть приблизно 1,55 В. Слід звернути увагу на те, що всі перераховані вище Піни налаштовані на роботу в аналоговому режимі.

Програмна реалізація зазначеної функціональності наведена далі.

Слід звернути увагу на те, що зазначена функціональність реалізована за допомогою простої ініціалізації пристроїв в складі мікроконтролера. Серед цих пристроїв присутній також ЦАП. Важливо пам'ятати, як саме реалізується подача необхідного розрахункового напруги ЦАП, встановлюються значення в регістрі даних, яке перекладається в

потрібний рівень напруги. Значення вихідної напруги на виході ЦАП визначається на основі напруги живлення.

6.3 Завдання

Ознайомитися із запропонованим прикладом і документацією щодо даного розділу. Задати напругу, яку можна підсилити при КП = 16. Продемонструвати, що при подальшому збільшенні вхідної напруги посилення стає неможливим. Для отримання додаткових балів реалізувати схему інвертуючого підсилювача згідно з документацією.

Лістинг 6.1 – Приклад програми

```
#include "stm32f30x_rcc.h"
#include "stm32f30x_gpio.h"
#include "stm32f30x_opamp.h"
#include "stm32f30x_syscfg.h"
#include "stm32f30x_dac.h"
void init_opamp(void);
void init_dac(void);
int main() {
    init_dac();
    init_opamp();
    while(1);
}
void init_opamp(void) {
    GPIO_InitTypeDef gpio_init;
    OPAMP_InitTypeDef opamp_init;
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA, ENABLE);
    GPIO_StructInit(&gpio_init);
    gpio_init.GPIO_Pin = GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3;
    gpio_init.GPIO_Mode = GPIO_Mode_AN;
    GPIO_Init(GPIOA, &gpio_init);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
    OPAMP_StructInit(&opamp_init);
    opamp_init.OPAMP_InvertingInput = OPAMP_InvertingInput_PGA;
    opamp_init.OPAMP_NonInvertingInput = OPAMP_NonInvertingInput_IO4;
    OPAMP_Init(OPAMP_Selection_OPAMP1, &opamp_init);
    OPAMP_PGAConfig(OPAMP_Selection_OPAMP1, OPAMP_OPAMP_PGAGain_2,
    OPAMP_PGAConnect_No);
    OPAMP_Cmd(OPAMP_Selection_OPAMP1, ENABLE);
}
void init_dac(void) {
    DAC_InitTypeDef dac_init;
    GPIO_InitTypeDef gpio_init;
    GPIO_StructInit(&gpio_init);
    gpio_init.GPIO_Mode = GPIO_Mode_AN;
    gpio_init.GPIO_Pin = GPIO_Pin_4;
```

```
GPIO_Init(GPIOA, &gpio_init);
RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);
DAC_StructInit(&dac_init);
dac_init.DAC_WaveGeneration = DAC_WaveGeneration_None;
dac_init.DAC_OutputBuffer = DAC_OutputBuffer_Enable;
dac_init.DAC_Trigger = DAC_Trigger_None;
DAC_Init(DAC_Channel_1, &dac_init);
DAC_Cmd(DAC_Channel_1, ENABLE);
DAC_SetChannel1Data(DAC_Align_8b_R, 0x44);
}
```

Додаток А
Приклад оформлення титульної сторінки

Міністерство освіти та науки України
Національний університет «Запорізька політехніка»

Кафедра МiНЕ

ЗВІТ
З ЛАБОРАТОРНОЇ РОБОТИ №__

(тема роботи)
з дисципліни „МІКРОПРОЦЕСОРНІ ПРИСТРОЇ
КЕРУВАННЯ ТА ОБРОБКИ ІНФОРМАЦІЇ“

Виконав:
студент гр. РТ—

(Ініціали, Прізвище)

Прийняв:
(посада)

(Ініціали, Прізвище)

20__ р.

Додаток Б

Початок роботи з налагоджувальним модулем

Для роботи з налагоджувальним модулем STM32F3 Discovery необхідно наступне програмне забезпечення:

- ❖ середовище розробки Keil uVision версії не нижче 4.7 з підтримкою можливості створення проекту для мікроконтролера, який використовується на платі (STM32F303VC);
- ❖ утиліта STM32 ST-Link Utility для виконання прошивки, роботи з пам'яттю мікроконтролера; при установці утиліти також встановлюються драйвера для підключення пристрою.

Першим завданням для виконання є перевірка коректності підключення плати та її розпізнавання як пристрою програмними засобами. Підключіть плату до ПК за допомогою кабелю USB і відкрийте STM32 ST-LINK Utility. Виконайте команду меню Target / Connect. В результаті в списку Device Information (рис. Б.1) повинна з'явитися інформація про пристрій. Збережіть програму, яка знаходиться в мікроконтролері як hex-файл.

Дана програма може бути особливо корисною, коли виникають труднощі з програмуванням пристрою в результаті програмних і апаратних проблем, які складно відстежувати. В такому випадку слід виконати команду Target/Erase Chip, після чого виконати програмування пристрою.

Основна робота з модулем заснована на використанні середовища розробки Keil uVision. Процес установки в даному матеріалі не розглядається, представлена лише настройка проекту для роботи з налагоджування платою.

Створення проекту починається командою меню Project / New uVision Project. Необхідно вказати розташування проекту, для чого бажано попередньо створити окрему директорію. Далі слід вибрати фірму-виробник мікроконтролерів і власне сам мікроконтролер. На останньому кроці буде запропоновано додати до проекту файл початкової ініціалізації на мові асемблера, що необхідно підтвердити. Після цього сам проект буде містити лише один згаданий файл, який в дереві проекту буде відображатися в файлової групі Startup. За допомогою файлових груп забезпечується структурний і логічне поділ функціональності проекту. Файлові групи не відображаються в реальні директорії операційної системи. Тому при додаванні файлів в проект не обов'

язково створювати відповідні директорії. Нову файлову групу можна створити за допомогою контекстного меню у вікні структури проекту командою Add Group.

Наступним етапом налаштування проекту є додавання необхідних файлів. Для створення програмного забезпечення необхідні файли 2 бібліотек - CMSIS і Standard Peripheral Library. Перша з них описує базові функції для роботи і опису регістрів, а інша забезпечує роботу з периферійними пристроями. Завантажити бібліотеку можна на сайті виробника зі сторінки, присвяченій оцінний модуль. Зазвичай Standard Peripheral Library вже включає в себе файли бібліотеки CMSIS (піддиректорія CMSIS), тому досить завантажити тільки одну бібліотеку.

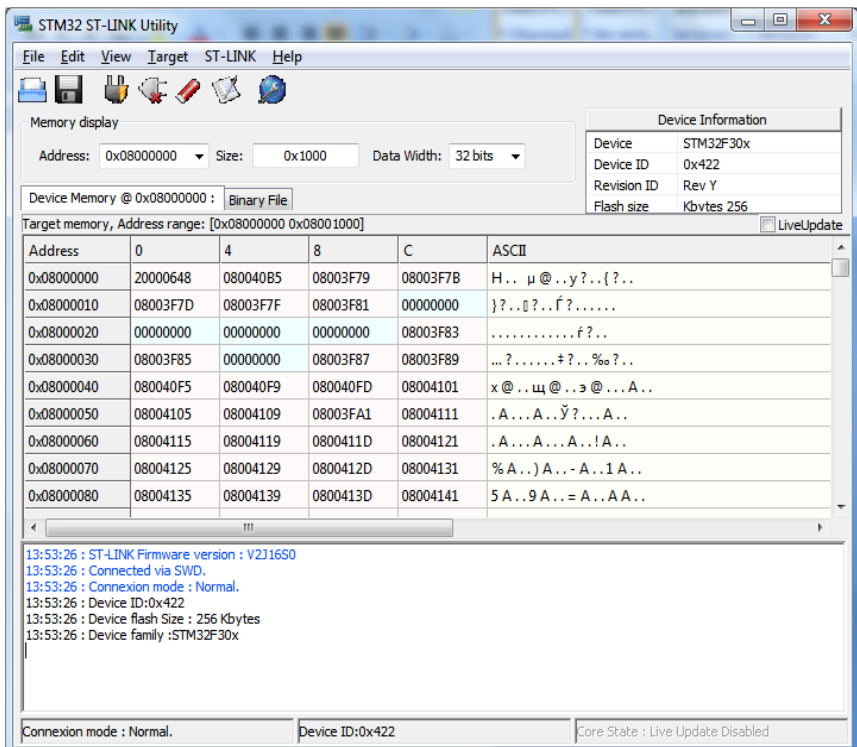


Рисунок Б.1. – Вікно утиліти після виконання команди

З бібліотеки CMSIS необхідні наступні файли:

- core_cm4.h;

- core_cmFunc.h;
- core_cmInstr.h;
- stm32f30x.h;
- stm32f30x_conf.h;
- system_stm32f30x.h
- system_stm32f30x.c.

Для початкового підключення їх слід скопіювати в директорію проекту.

Підключення файлів стандартної бібліотеки залежить від того, які пристрої будуть використані в роботі. Однак, при використанні функцій бібліотеки рекомендується обов'язково додати файли для налаштування тактової частоти пристрою (Reset Clock Control, RCC), оскільки без включення тактування робота пристроїв неможлива. Для роботи з конкретним пристроєм до проекту слід додати 2 файли - файл заголовків і файл вихідних кодів (відрізняються розширеннями .h і .c), в залежності від пристрою слід додати файли зі скороченим позначенням в кінці імені файлу, наприклад, для портів вводу/виводу - закінчення gpio.

Слід також зазначити, що файл з головною функцією main не створюється автоматично, тому його слід створити самостійно. Для цього слід виконати команду меню File/New і зберегти новий файл як main.c і додати в нього код стандартної функції main, яка не приймає параметрів і повертає цілочисельне значення і містить нескінченний цикл. Після збереження файл автоматично не додають до проекту, тому ця дія слід виконати самостійно. Рекомендується створити окрему групу файлів в проекті, в яку додавати файли (команда контекстного меню Add Files to Group ...), створені програмістом (наприклад, групу файлів можна назвати User), щоб відокремити їх від бібліотечних файлів. Таким же чином слід створити окремі групи для файлів бібліотек.

В результаті виконання зазначених дій отримуємо структуру проекту, приклад якої представлений на Рисунку Б.2. Даного стилю формування структури проекту рекомендується дотримуватися для того, щоб інші користувачі могли швидко орієнтуватися в ньому.

Також слід зазначити, що додавання нових файлів заголовків до проекту не є обов'язковим – обов'язково додавання тільки файлів вихідних кодів. Компілятор сам зможе знайти необхідні файли, якщо вони будуть знаходитися в місцях пошуку файлів заголовків.

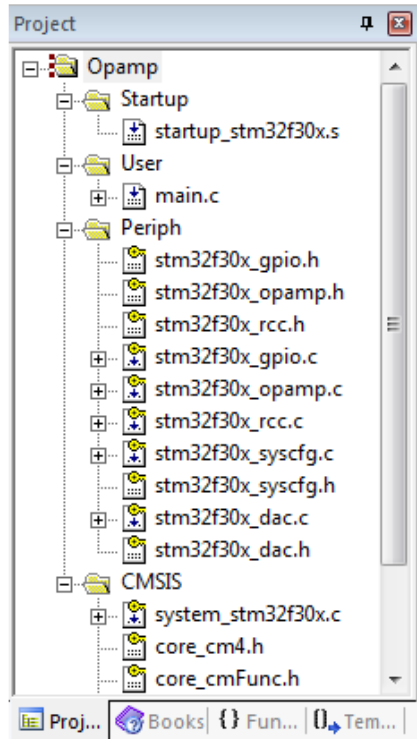


Рисунок Б.2 – Приклад структури робочого проекту

Після створення і налаштування структури проекту наступним завданням є налаштування компілятора, а також апаратних засобів для проведення програмування і відладки. Вікно основних налаштувань проекту можна відкрити командою Project/Options for Target.... Налаштування компілятора виконуються за допомогою вкладки C/C++. У ній вказуємо, що необхідно під час компіляції визначити такі символи: USE_STDPERIPH_DRIVER, STM32F30X; вказуємо це, ввівши зазначені значення в текстове поле Define (рис. Б.3). Оскільки при створенні проекту будемо дотримуватися правила копіювання файлів в директорію проекту і підключення їх з вказаній директорії, то налаштування директорій файлів заголовків можна не виконувати, як показано на Рис. Б.3 (поле Include Paths пусте).

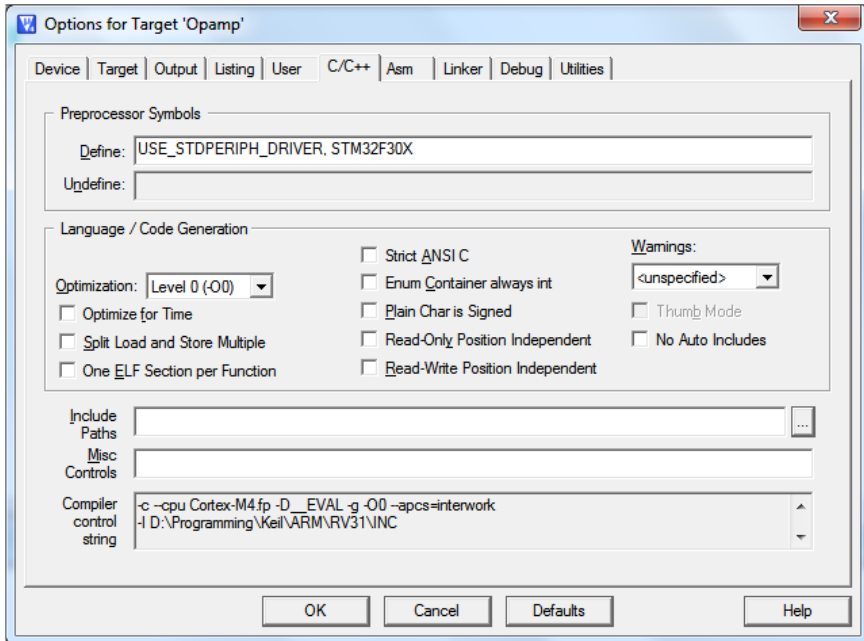


Рисунок Б.3 – Налаштування опцій компілятора

Далі виконаємо настройку налагодження. Ці дії проводяться у вкладці Debug. Для налагодження використовується інтерфейс ST-Link, який вже вбудований в плату. Тому встановлюємо відповідне значення кнопкою вибору і вибираємо вказаний інтерфейс зі списку (рис. Б.4).

Крім того, необхідно також виконати настройки самого пристрою налагодження, натиснувши кнопку Settings. В результаті відкривається додаткове діалогове вікно з 3 вкладками. У вкладці Debug змінюємо значення з JTAG на SW (рис. Б.5).

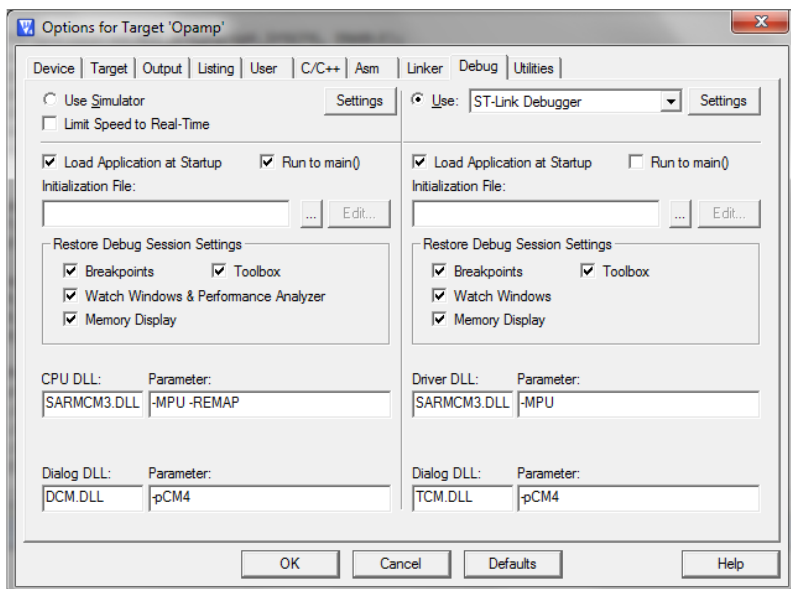


Рисунок Б.4 – Вкладка налаштування налагодження

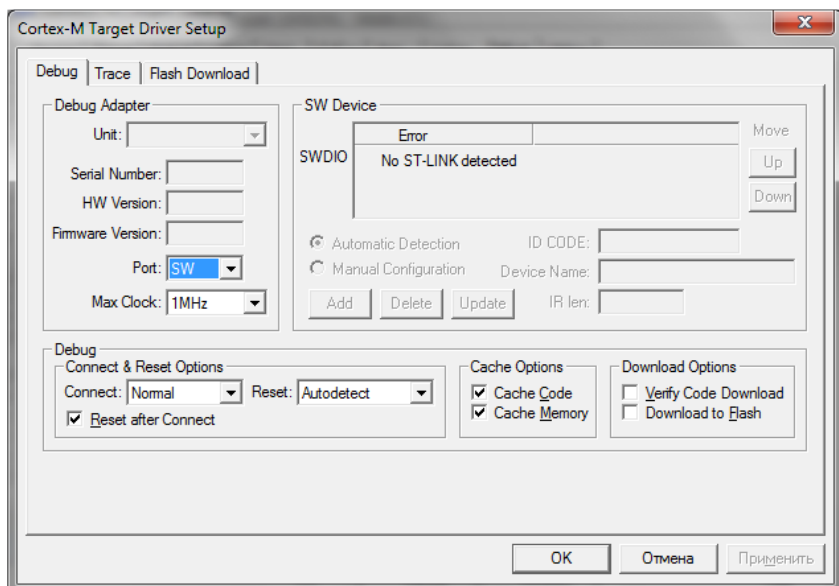


Рисунок Б.5 – Налаштування інтерфейсу підключення

На вкладці Flash Download виконуються настройки пам'яті пристрою для програмування і дій, які виконуються після виконання програмування. Рекомендується встановити прапор Reset and Run, щоб відразу після завантаження почати налагодження, інакше необхідно натиснути кнопку Reset на платі. Також слід додати тип пам'яті пристрою, як це показано на рис. Б.6.

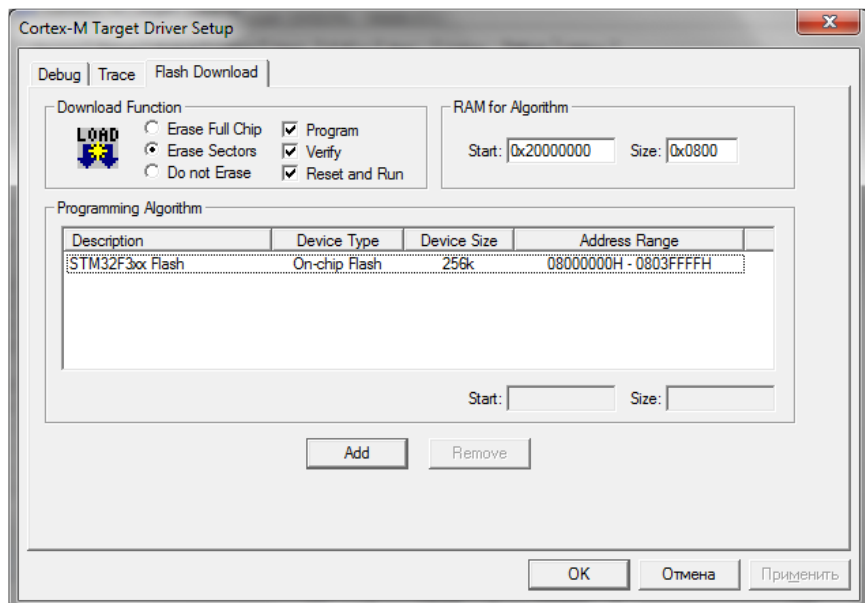


Рисунок Б.6 – Налаштування пам'яті пристрою для налагодження

Після виконання зазначених дій підтвердити зміни, натиснувши ОК.

Таким же чином слід виконати налаштування інтерфейсу для програмування (вкладка Utilities). Вибрати зі списку згаданий вище інтерфейс (рис. Б.7). Натиснувши кнопку Settings, перевірити, що настройки відповідають представленим на рис. Б.6.

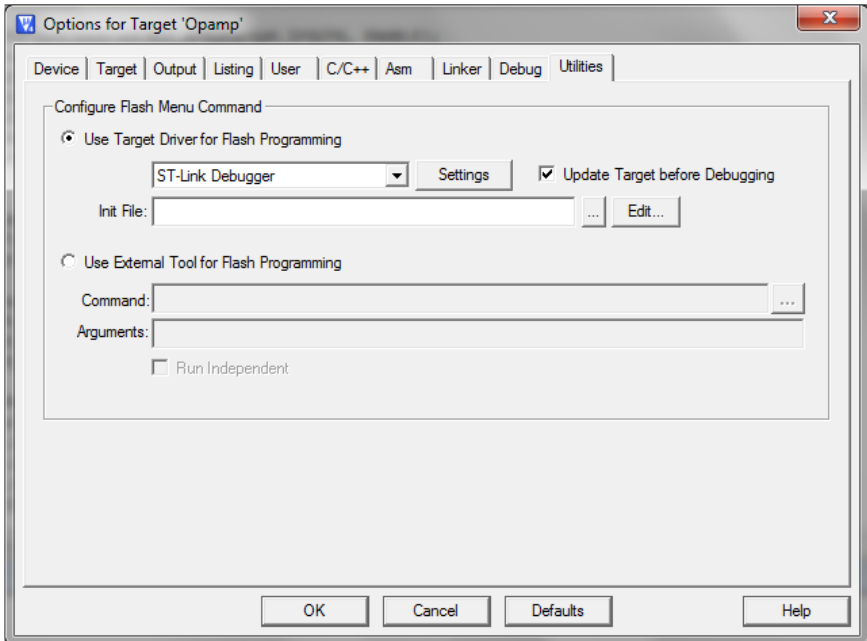


Рисунок Б.7 – Вибір засобів програмування

Після проведення зазначених дій слід виконати побудову проекту для перевірки правильності налаштувань. Запустити побудову проекту можна командою меню **Project / Build target** або клавішею **F7**. В результаті складання проекту повинна пройти без помилок. Якщо виникли помилки, то перевірте ще раз виконання зазначених вище кроків.