Міністерство освіти і науки України

Національний університет водного господарства та природокористування

Кафедра автоматизації, електротехнічних та комп'ютерно-інтегрованих технологій

04-03-373М

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт
з освітнього компонента
«Мікропроцесорні системи та програмування мікропроцесорних засобів»
для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійними програмами «Робототехніка та штучний інтелект» та «Автоматизація та комп'ютерно-інтегровані технології» спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології», «Біотехнології, біоробототехніка та біоенергетика» спеціальності 162 «Біотехнології та біоінженерія» та «Електроенергетика, електротехніка та електромеханіка» спеціальності 141 «Електроенергетика, електротехніка та електромеханіка» денної та заочної форм навчання

Рекомендовано науково-методичною радою з якості ННІЕАВГ
Протокол №5 від 25.01.2024 р.
Рекомендовано науково-методичною радою з якості ННІБА
Протокол № 4 від 31.01.2024 р.

Рівне – 2024

Методичні вказівки до лабораторних робіт з освітнього компонента «Мікропроцесорні системи та програмування мікропроцесорних засобів» для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійними програмами «Робототехніка та штучний інтелект» та «Автоматизація та комп'ютерно-інтегровані технології» спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології», «Біотехнології, біоробототехніка та біоенергетика» спеціальності 162 «Біотехнології та біоінженерія» та «Електроенергетика, електротехніка та електромеханіка» спеціальності 141 «Електроенергетика, електротехніка та електромеханіка» денної та заочної форм навчання [Електронне видання] / Реут Д. Т. – Рівне : НУВГП, 2024. – 78 с.

Укладач: Реут Д. Т., к.т.н., доцент кафедри автоматизації, електротехнічних та комп'ютерно-інтегрованих технологій.

Відповідальний за випуск: Древецький В. В., д.т.н., професор, завідувач кафедри автоматизації, електротехнічних та комп'ютерно-інтегрованих технологій.

Керівник групи забезпечення спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології»: Христюк А. О., к.т.н., доцент, доцент кафедри автоматизації, електротехнічних та комп'ютерно-інтегрованих технологій

Керівник групи забезпечення спеціальності 162 «Біотехнологія та біоінженерія»: Грицина О. О., к.т.н., доцент, доцент кафедри теплогазопостачання, вентиляції та санітарної техніки

Керівник групи забезпечення спеціальності 141 «Електроенергетика, електротехніка та електромеханіка»: Літковець С. П., к.т.н., доцент кафедри автоматизації, електротехнічних та комп'ютерно-інтегрованих технологій

Зміст

Вступ

Лабораторні роботи з навчальної дисципліни «Мікропроцесорні системи та програмування мікропроцесорних засобів» дають змогу освоїти основні прийоми розробки й програмування мікропроцесорних систем і пристроїв на базі мікроконтролерів Microchip ATmega328P, STMicroelectronics STM32F072C8, Espressif ESP8266.

У лабораторних роботах розглянуто:

– використання цифрових портів вводу-виводу, для зчитування стану кнопок та світлодіодної індикації;

– використання анологових входів мікроконтролера для отримання сигналів з давачів;

– програмна реалізація позиційного та ПІД-регулювання;

– обмін даними за допомогою модуля асинхронного прийомопередавача;

– використання операційної системи FreeRTOS з мікроконтролерами для реалізації багатозадачних систем;

– застосування контролера прямого доступу до пам’яті при роботі АЦП в режимі неперервного перетворення;

– реалізація веб-інтерфейсу мікропроцесорного пристрою.

Лабораторна робота 1. Режими роботи портів вводу-виводу. Використання кнопок та дискретних світлодіодів з мікроконтролерами

Мета роботи: навчитися програмувати мікроконтролер для роботи з механічними кнопками та дискретними індикаторними світлодіодами.

Теоретичні відомості

Практично всі мікропроцесорні пристрої мають засоби відображення та введення певної інформації. До

найпростіших засобів введення/виведення відносяться контактні давачі (кнопки, клавіатури, кінцеві вимикачі та ін.) та світлодіодні індикатори (дискретні світлодіоди, семисегментні, матричні, шкальні світлодіодні індикатори).

Окремі кнопки та набори кнопок – клавіатури, використовуються для введення внутрішніх параметрів пристроїв (уставок, завдань, коефіцієнтів і т.д.), а також для оперативного керування роботою пристрою. В залежності від кількості кнопок та їх призначення, клавіатури поділяють на: службові – кнопки керування роботою приладів, цифрові – для введення цифрової інформації, буквенно-цифрові, універсальні.

В залежності від того, яка кількість кнопок необхідна в пристрої і яка для цього є доступна кількість входів/виходів мікроконтролера, будують принципову електричну схему пристрою. В найпростішому варіанті коли необхідно лише декілька (звичайно до 4) кнопок і є необхідна кількість вільних портів мікроконтролера, використовують найпростіші схеми підключення кнопок (рис. 1) з підтяжкою до напруги живлення мікроконтролера або потенціалу землі.

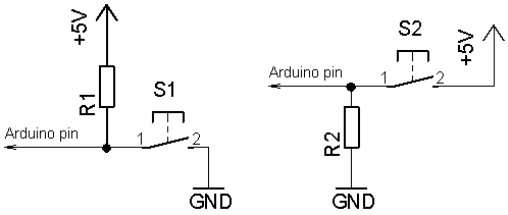

Рис. 1. Схеми підключення окремих кнопок

Для підключення більшої кількості кнопок (клавіатури) використовують матричну побудову схем (рис. 2). Всі механічні вимикачі мають одну негативну властивість, відому як “вібрація контакту” (рис. 3), що зумовлене коливанням гнучких контактів під час їх замикання і розмикання.

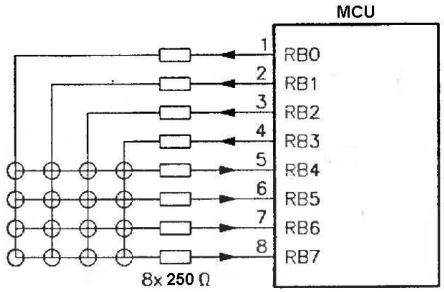

Рис. 2. Матрична схема підключення кнопок (клавіатури)

Час цих коливань всього декілька мілісекунд. При цьому замість “чистого” прямокутного імпульсу отримується кілька імпульсів.

Рис. 3. Явище “вібрації контакту”

Цей недолік можна ліквідувати з допомогою RS-тригерів, одновібраторів, інтегруючих RC-ланок, проте в пристроях на базі мікроконтролерів боротьбу з “вібрацією контакту” покладають на програму, в котрій здійснюється багатократне зчитування стану входу, визначаючи момент стійкої зміни його стану, або, найчастіше, роблять повторне опитування входу через певний проміжок часу (10 – 20 мс) після першого спрацювання контакту. Клавіатуру доцільно підключати до виводів мікроконтролера, що викликають переривання.

Для спрощення алгоритму роботи мікроконтролера, а також для економії кількості використовуваних портів вх/вих при використанні клавіатур із значною кількість кнопок, використовують додаткові апаратні засоби – дешифратори, спеціалізовані контролери клавіатур, або використовують окремий мікроконтролер для клавіатури.

У мікроконтролерах архітектури AVR напрямок роботи вхід-вихід обирається бітами в регістрі DDRx, де x – літера порта (DDRA, DDRB, …). Підтягуючі резистори активуються записом 1 у відповідний біт регістра PORTx за умови, що вивід налаштований на вхід (рис. 4).

DDxn	PORTxn	PUD (in MCUCR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low.
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output low (sink)
1	1	X	Output	No	Output high (source)

Рис. 4. Налаштування порта загального призначення

План роботи

1. Ознайомитися з середовищем розробки та налаштувати його. Завантажити тестову програму.

2. Зчитати стан кнопки та просигналізувати його за допомогою світлодіода.

3. Реалізувати зміну періоду блимання світлодіода двома кнопками.

Порядок виконання роботи

1. Запустити середовище Geany IDE. Створити файл main.c з наступним вмістом:

```
#define F_CPU 16000000UL // такт.сигнал 16 МГц

#include <avr/io.h>
#include <stdio.h>
#include <util/delay.h>

int main(int argc, char **argv)
{
    DDRB=0b00100000; //PB5 - вихід
    PORTB=0x00;
//зміна стану PB5 на протилежний кожні 0,2 с
    while(1)
    {
        if(PORTB&(1<<PB5)) PORTB&=~(1<<PB5); else
PORTB|=1<<PB5;
        _delay_ms(200);
```

```
    }
        return 0;
}
```

У цьому ж каталозі створти *makefile* з таким вмістом:

```
# Makefile for programming AVR in Geany
MK=atmega328p
compile:
	"C:\Program
Files\Arduino\hardware\tools\avr\bin\avr-gcc.exe" -
mmcu=$(MK) -Wall -gdwarf-2 -Os -std=gnu99 -
funsigned-char -funsigned-bitfields -fpack-struct -
fshort-enums -MD -MP -MT main.o -MF dep_main.o.d -
c                                                main.c
	"C:\Program Files\Arduino\hardware\tools\
avr\bin\avr-gcc.exe" -g -mmcu=$(MK) -o main.elf
main.o
	"C:\Program Files\Arduino\hardware\tools\
avr\bin\avr-objdump.exe" -h -S main.elf > main.lss
	"C:\Program Files\Arduino\hardware\tools\
avr\bin\avr-objcopy.exe" -j .text -j .data -O ihex
main.elf main.hex
	"C:\Program Files\Arduino\hardware\tools\
avr\bin\avr-size.exe" --mcu=$(MK) -C main.elf
program:
	"C:\Program
Files\Arduino\hardware\tools\avr\bin\avrdude.exe" -
C                                        "C:\Program
Files\Arduino\hardware\tools\avr\etc\avrdude.conf"
-v -patmega328p -carduino -PCOM4 -b115200 -D -
Uflash:w:main.hex:i
clean:
```

```
        del -f main.o dep_main.o.d main.map
main.eep.hex main.hex main.lss main.eep main.elf
size:
        "C:\Program
Files\Arduino\hardware\tools\avr\bin\avr-size.exe"
--mcu=$(MK) -C main.elf
```

Вказати в налаштуваннях IDE команду компіляції проекту – *make compile*, виконання проекту – *make program*. Скомпілювати та вивантажити програму в плату Arduino Uno. За потреби вказати в цілі *program* замість COM4 інший номер порту.

2. Відкрити datasheet для ATmega328P і в розділі 28. Electrical Characteristics знайти допустимі струми та напруги при роботі виводу на вхід та на вихід. Підібрати струмообмежувальні резистори для світлодіодів. Підключити за допомогою макетної плати світлодіоди та модифікувати програму таким чином, щоб при натиснутій кнопці світлодіод гаснув, інакше – засвічувався. Скомпілювати та вивантажити програму в плату Arduino Uno. Переконатись в правильності роботи програми.

3. Підключити ще одну кнопку до мікроконтролера. Задати величину затримки змінною, при натисканні однієї кнопки затримка збільшується, іншої - зменшується. Скомпілювати та вивантажити програму в плату Arduino Uno. Переконатись в правильності роботи програми.

4. Оформити звіт про виконання лабораторної роботи. Звіт повинен містити: назву та мету лабораторної роботи; тексти програм з коментарями; висновок про виконання роботи.

Контрольні запитання

1. Які засоби введення/виведення інформації користувача використовуються в мікропроцесорній техніці?
2. Для чого використовують засоби введення/виведення інформації у мікропроцесорних пристроях?
3. Які схеми використовуються при підключенні кнопок до мікропроцесорних пристроїв?
4. Які схеми використовуються при підключенні окремих світлодіодів до мікропроцесорних пристроїв?

Лабораторна робота 2. Виведення даних на світлодіодні динамічні індикатори

Мета роботи: навчитися програмувати мікроконтролер ATmega328P для роботи з світлодіодними семисегментними індикаторами.

Теоретичні відомості

Найпростішими засобами відображення інформації є світлодіодні індикатори. Окремі світлодіоди використовуються для відображення інформації про режим роботи пристрою, включення чи виключення приладів та ін. Семисегментні та матричні світлодіодні індикатори можуть виводити алфавітно-цифрову та графічну інформацію. Шкальні світлодіодні індикатори відображають значення певної неперервної величини у вигляді гістограм.

Для семисегментного світлодіодного індикатора, схема якого наведена на рис. 5, використовується динамічна індикація, при якій послідовно в циклі з певною частотою виводиться кожний розряд індикатора. Для задання символів кожного розряду (тобто комбінації сегментів a,b,c,d,e,f,g та роздільного знаку між розрядами)

використовуються одні і ті ж самі лінії та порти мікроконтролера, для комутації розрядів використовуються окремі виводи контролера.

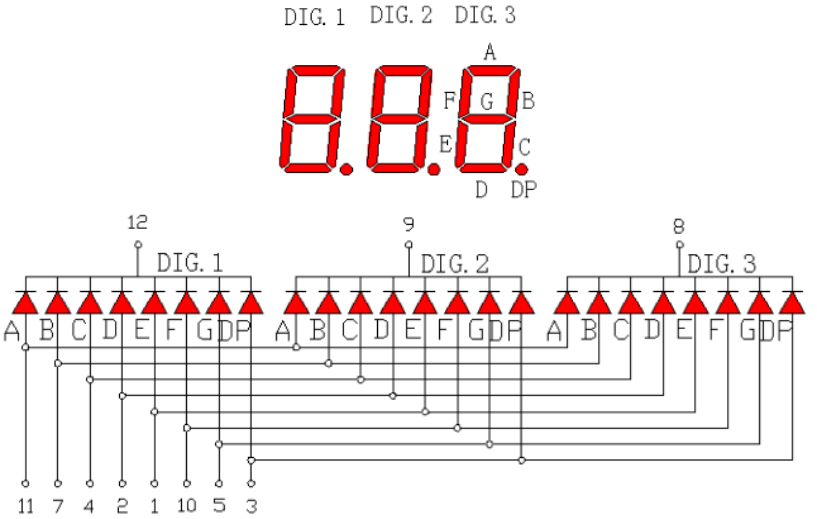

Рис. 5. Зовнішній вигляд та внутрішня схема семисегментного світлодіодного індикатора із спільним катодом

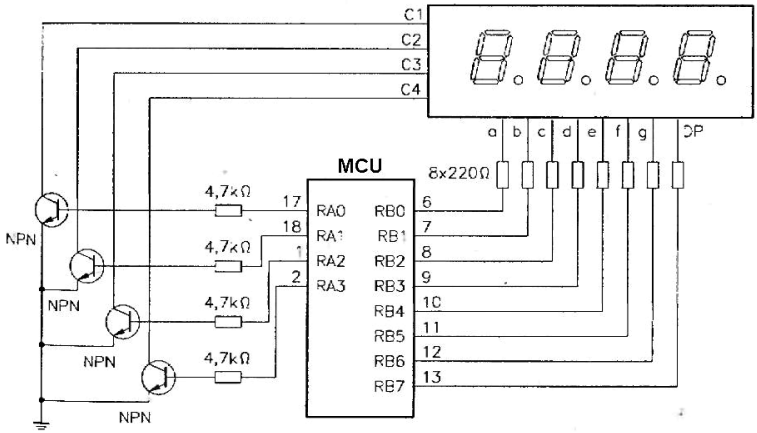

Рис. 6. Схема підключення семисегметного світлодіодного індикатора

Тобто для найпростішої схеми керування 4-розрядним семисегментним світлодіодним індикатором з роздільним знаком необхідно 8+4=12 портів мікроконтролера (рис. 6). Для зменшення кількості використовуваних ліній та спрощення програми можуть використовуватися дешифратори, зсуваючі регістри, спеціалізовані та універсальні контролери. Для побудови пристроїв з світлодіодними індикаторами та клавіатурою можуть використовуватись спеціальні схеми (рис. 7).

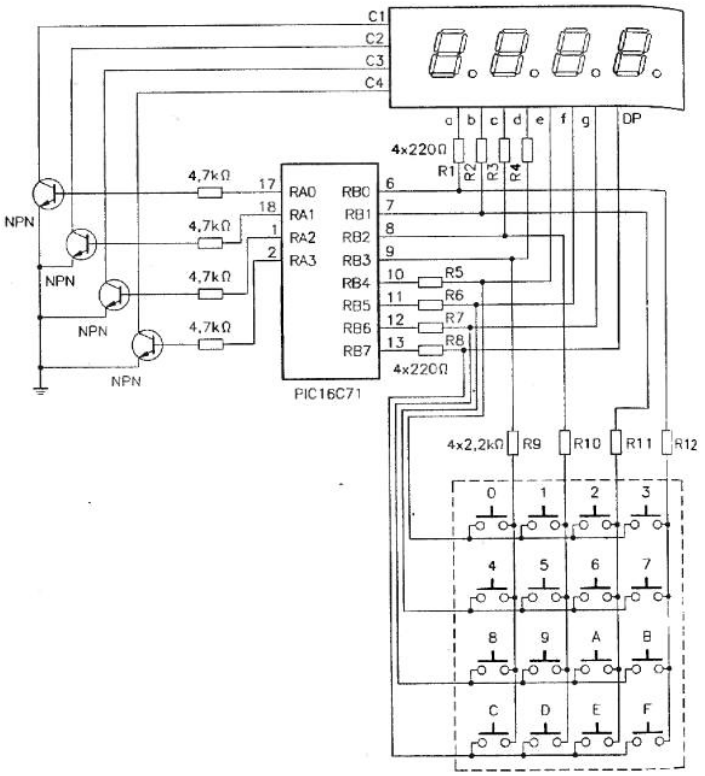

Рис. 7. Схема підключення семисегметного індикатора та клавіатури

При написанні програм користувача, для роботи з різноманітними периферійними пристроями, використовують програми-драйвери. Драйвером якого-небудь апаратного вузла називають набір підпрограм, які враховують всі апаратні особливості схеми включення цього вузла і максимально полегшують роботу з ним головній програмі. Часто виробники програмних продуктів (середовищ програмування та компіляторів) надають готові програми-драйвери (бібліотеки) для роботи з найрізноманітнішими периферійними пристроями, які можливо відразу ж використовувати при програмуванні, або провести незначне коректування під конкретний пристрій та схему включення.

Для перемикання розрядів індикатора, як правило використовують переривання від вбудованих таймерів.

Таймери використовується для вимірювання та формування імпульсних сигналів.

Мікроконтролер ATmega328P містить два 8-розрядні таймери та один 16-розрядний. Всі таймери мають регістри OCRxy, значення з яких може використовуватись для задання тривалості ШІМ-сигналу. Таймер 1 має додатково регістр захоплення OCR1, куди записується вміст таймера TCNT1 по події захоплення - зміні сигналу на вході ICP1 або виході аналогового компаратора.

Режим захоплення (capture)

У режимі захвату 16-ти бітне значення таймера (TCNT1) захоплюється в регістр OCR1 при кожній події (зміні сигналу) на вході ICP1 або виході аналогового компаратора. Подія для захоплення задається в регістрі TCCR1B (біт ICES1) та ACSR (біт ACIC).

Режим захоплення використовується для вимірювання тривалості між двома змінами сигналу, наприклад періоду (вимірюється час між сусідніми

зростаючими фронтами або сусідніми спадаючими фронтами), тривалості імпульсу (вимірюється час між зростаючим фронтом та спадаючим фронтом), скважності (вимірюється період і тривалість імпульсу з подальшим обчисленням відношення) і т.д.

Приклад 1. Вимірювання періоду дискретного сигналу (рис. 8)

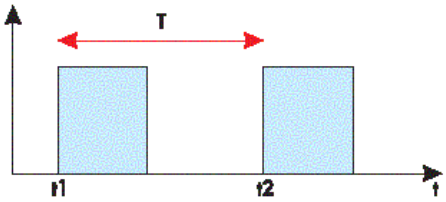

Рис. 8. Вимірювання періоду

Приклад 2. Вимірювання періоду з усередненням результату (рис. 9)

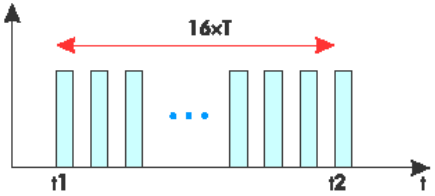

Рис. 9. Вимірювання періоду з усередненням

Приклад 3. Вимірювання тривалості імпульсу (рис. 10)

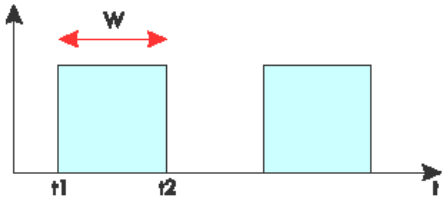

Рис. 10. Вимірювання тривалості імпульсу

Приклад 4. Вимірювання скважності імпульсів (рис. 11)

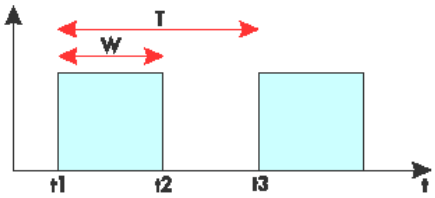

Рис. 11. Вимірювання скважності імпульсів

Режим порівняння (compare)

У режимі порівняння значення регістра OCRx порівнюються зі вмістом таймера. При збігу відбувається переривання і вихід контролера OCxy:

- встановлюється в 1;
- встановлюється в 0;
- стан не змінюється (як правило, використовується у випадку вимірювання часових проміжків).

Реакція виводу визначається бітами керуючого регістру TCCR1A.

Час відраховується від нуля до встановленого значення OCRx. Цей спосіб корисний для того, щоб зробити певні дії в точні інтервали часу. Звичайний режим роботи таймера з переповненням може використовуватися для виконання тих же самих функцій, однак у цьому випадку таймер потрібно буде перезавантажувати новим значенням при кожному переповненні. Режим порівняння також може автоматично змінювати стан виходу.

Режим ШІМ (PWM)

В ШІМ-режимі таймер використовується для генерування ШІМ-сигналу на відповідних виводах мікроконтролера. Відповідний біт настроювання виводу мікроконтролера DDRxy повинен бути встановлений в 1 для настроювання виводу на вихід.

В режимі Fast PWM значення в лічильному регістрі таймера збільшується й при рівності вмісту OCRx вихід OCx буде скинутий в 0, а значення продовжить рости до

моменту, коли таймер переповниться. При цьому вихід OCx буде встановлений в 1 та рахунок почнеться спочатку. Таким чином передподільник таймера задає період ШІМ, а вміст OCRx - тривалість імпульсу (рис. 12).

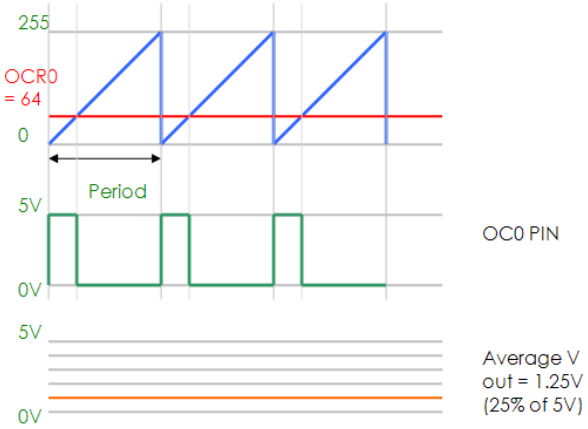

Рис. 12. Генерування ШІМ-напруги таймером

Приклад використання таймера в режимі порівняння

```
    #define F_CPU 16000000UL
    #include <avr/io.h>
    #include <avr/interrupt.h>
    #include <util/delay.h>
    volatile uint16_t time_count=0; //volatile забороняє компілятору оптимізовувати звертання до змінної
    ISR(TIMER1_COMPA_vect) { //переривання по співпадінню вмісту таймера з OCR1A спрацьовує кожні півсекунди
      time_count++;
      if(time_count==120) {
                          PORTB^=1<<PB5;
      //щохвилини змінювати стан
                          time_count=0;
```

```
                        }
    }

    int main() {
      DDRB=0b00100000;       //вихід            на
світлодіод на 13-му контакті (PB5)
        TCCR1A = 0;                 //таймер     1:
подільник 256, рахунок до 31250(-1)
        TCNT1=0;
        TCCR1C=0;
        TIMSK1 = 0b10; //дозволити    переривання
при події порівняння
        OCR1AH=0x7a;          //рахувати до
        OCR1AL=0x11;          //31249
        TCCR1B = 0b01100;     //  активувати  режим
4
      sei();            //дозволити переривання
    while(1) {
      if(PINB&~(1<<PINB4)) time_count--;
      _delay_ms(250); //будь-які операції
      }
      return 0;
    }
```

Якщо значення змінної модифікується в обробнику переривання, цю змінну потрібно проголошувати з ключовим словом `volatile`. Інакше компілятор може під час оптимізації замінити повторні звертання до змінної в SRAM на звертання до регістра загального призначення (доступ до якого займе менше тактів), де збережена її копія; непрогнозована для компілятора модифікація змінної в обробнику переривання призведе до продовження використання старого значення з регістра загального призначення замість тривалішого зчитування нового з SRAM.

План роботи

1. Проаналізувати схему підключення індикатора до мікроконтролера.

2. Скласти програму-таймер, що виводить кількість

секунд з моменту запуску на світлодіодний семисегментний динамічний індикатор.

Порядок виконання роботи

1. Використовуючи дані про підключення індикатора до мікроконтролера з таблиці 1, реалізувати функцію, що в якості аргументу приймає цифру та запалює її зображення на індикаторі, який розміщений на платі розширення.

Таблиця 1

Назва контакту індикатора	Номер контакту індикатора	Назва контакту плати Arduino	Назва піна мікроконтролера
A	11	5	PD5
B	7	7	PD7
C	4	A2	PC2
D	2	A4	PC4
DIG1	12	9	PB1
DIG2	9	2	PD2
DIG3	8	8	PB0
DP	3	A3	PC3
E	1	A5	PC5
F	10	6	PD6
G	5	4	PD4

Аналогічну функцію передбачити для вибору активного розряду. При реалізації функцій можна використати оператор множинного вибору за прикладом

```
        switch (digit) {
        case 1: //запалюємо перший розряд
(сотні)
         PORTB&=~(1<<PB1);
         PORTD|=(1<<PD2);
         PORTB|=(1<<PB0);
         break;
        case 2: //запалюємо другий розряд
```

```
(десятки)
         PORTB|=(1<<PB1);
         PORTD&=~(1<<PD2);
         PORTB|=(1<<PB0);
         break;
        case 3: //запалюємо третій розряд
(одиниці)
         PORTB|=(1<<PB1);
         PORTD|=(1<<PD2);
         PORTB&=~(1<<PB0);
         break;
      }
```

2. Налаштувати таймер та створити обробник переривання таймера ISR(TIMER1_COMPA_vect) {…}, в якому здійснювати перемикання розрядів індикатора та відлік часу. Скласти програму-таймер, що виводить кількість секунд з моменту запуску на світлодіодний семисегментний динамічний індикатор.

3. Скомпілювати, вивантажити програму в плату Arduino Uno, підключити плату розширення з індикатором, перевірити правильність роботи програми та налагодити її за потреби.

4. Оформити звіт про виконання лабораторної роботи. Звіт повинен містити: назву та мету лабораторної роботи; тексти програм з коментарями; висновок про виконання роботи.

Контрольні запитання

1. Які схеми використовуються при підключенні семисегментних світлодіодних індикаторів до мікропроцесорних пристроїв?

2. Які переваги і недоліки світлодіодних індикаторів?

3. Скільки таймерів має мікроконтролер ATmega328P?

4. В чому різниця між режимом таймера і лічильника?

5. Яка максимальна розрядність ШІМ мікроконтролера ATmega328P?

6. Яка максимальна розрядність захоплення імпульсу мікроконтролера ATmega328P?

7. Як працює таймер в режимі порівняння?

8. Як працює таймер в режимі ШІМ?

Лабораторна робота 3. Використання вбудованого АЦП

Мета роботи: навчитися вимірювати напругу на аналоговому вході мікроконтролера ATmega328P за допомогою АЦП.

Теоретичні відомості

Аналогово-цифровий перетворювач (АЦП, англ. Analog-todigital conventer, ADC) - пристрій, що перетворює вхідний аналоговий сигнал в цифровий код (цифровий сигнал).

У мікроконтролері ATmega328P є 10-розрядний АЦП. Вхідна напруга конвертується в 10-розрядне двійкове значення. Мінімальне значення відповідає 0 В, а максимальне - опорній напрузі V_{ref}. В даному мікроконтролері є 3 варіанти опорних напруг (напруга живлення, зовнішня опорна напруга, внутрішнє джерело 1,1 В), які перемикаються парою бітів при налаштуванні АЦП. Отриманий результат перетворення записується в 2 регістра: ADCH і ADCL. Результат перетворення можна розрахувати за формулою

$$ADC = \frac{V_{in} \cdot 1023}{V_{ref}}$$

Налаштування АЦП зводиться до наступних дій:

- налаштування регістра ADMUX (регістр настройки мультиплексора АЦП),

- налаштування регістра ADCSRA (регістр статусу і контролю А),

- за потреби, налаштування ADCSRB (регістр статусу і контролю B),

В ATmega328P АЦП реалізовано на основі ЦАП та компаратора. Коли напруга з виходу ЦАП зрівняється з вимірюваною напругою з обраного каналу АЦП, процес перетворення можна вважати закінченим та код, що в цей момент був на вході ЦАП, є результатом аналогово-цифрового перетворення. Внаслідок перехідних процесів швидкість АЦП є обмеженою. Оптимальний діапазон тактової частоти модуля АЦП складає 50-200 кГц, тому коефіцієнт подільника потрібно підібрати такий, щоб частота на його виході знаходилась у вказаних межах.

План роботи

1. Розробити схему підключення фоторезистора до мікроконтролера.

2. Скласти програму, що виводить на світлодіодний семисегментний динамічний індикатор число, пропорційне напрузі з фоторезистора.

Порядок виконання роботи

1. Увімкнути фоторезистор і постійний резистор на макетній платі в схему подільника напруги, яка подається на аналоговий вхід ADC0.

2. У програму з попередньої роботи додати налаштування модуля АЦП та в нескінченному циклі опитування та зчитування результату перетворення. Налаштування регістрів виконати відповідно до рекомендацій в datasheet. Отриманий результат промасштабувати в шкалу 0…100 та вивести на індикатор. Перевірити реакцію отриманої схеми на зміну освітленості, затінивши фоторезистор або підсвічуючи його спалахом камери телефону. Для опитування АЦП можна використати фрагмент коду

```
ADCSRA|=1<<ADSC ;//запустили АЦП
while(ADCSRA&(1<<ADSC)); //чекаємо, доки триває вимірювання напруги
```

```
uint16_t  res=ADCL|(ADCH<<8);  //зчитали
результат
```

3. Оформити звіт про виконання лабораторної роботи. Звіт повинен містити: назву та мету лабораторної роботи; тексти програм з коментарями; висновок про виконання роботи.

Контрольні запитання

1. Яка розрядність модуля АЦП мікроконтролера ATmega328P?
2. Яка кількість каналів модуля АЦП мікроконтролера ATmega328P?
3. Яким чином реалізована багатоканальність модуля АЦП мікроконтролера ATmega328P?
4. Які регістри використовуються для роботи з модулем АЦП мікроконтролера ATmega328P?
5. Яке призначення регістра ADMUX?
6. В яких регістрах зберігається результат перетворення модуля АЦП?

Лабораторна робота 4. Реалізація позиційного регулятора температури на мікроконтролері

Мета роботи: навчитися програмувати мікроконтролери для виконання задач позиційного регулювання.

Теоретичні відомості

Позиційними називають регулятори, які мають чітко визначену кількість вихідних станів (позицій) для всієї множини вхідних сигналів.

Теоретично може бути будь-яка кількість позицій регулятора, проте на практиці використовують 2 і 3-позиційні регулятори. Це пояснюють складністю практичної реалізації передачі багатьох регулюючих впливів регулятора на об’єкт регулювання, тобто обмеження створюються виконавчими механізмами (ВМ) і регулюючими органами (РО), які є проміжною ланкою між

регулятором та об’єктом регулювання. ВМ і РО з двома (електромагнітні реле, магнітні пускачі, клапани та ін.) та трьома (крани-змішувачі, різноманітні механізми позиціювання та ін.) станами є досить поширеними, на відміну від ВМ і РО з більшою кількістю дискретних станів, оскільки замість них доцільніше використовувати ВМ і РО з неперервною зміною вихідного впливу.

Позиційні регулятори часто називають релейними, оскільки вони мають статичні характеристики як у релейних елементів. Релейні елементи є нелінійними, їх поділяють на:

- двопозиційні;
- трипозиційні;
- ідеальні;
- реальні.

На рис. 13-16 зображені статичні характеристики релейних елементів.

На практиці в малоінерційних системах використовуються регулятори з характеристиками реальних релейних елементів, оскільки це усуває негативне явище занадто частого перемикання вихідних елементів регулятора (наприклад, переключення з високою частотою вихідного електромагнітного реле, що різко зменшує термін його служби). При значній інерційності об’єкта регулювання необхідність в зоні нечутливості (гістерезисі) зменшується.

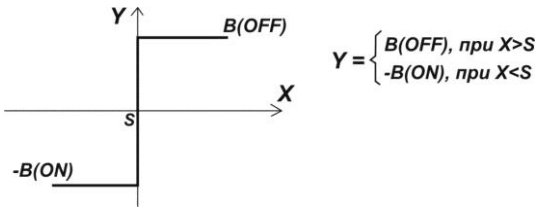

Рис. 13. Статична характеристика ідеального двопозиційного релейного елемента

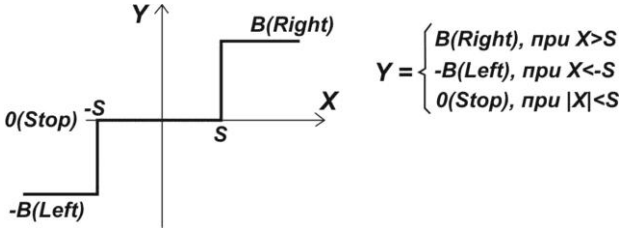

Рис. 14. Статична характеристика ідеального трипозиційного релейного елемента

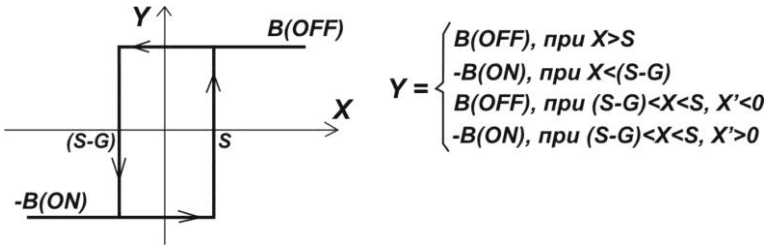

Рис. 15. Статична характеристика реального двопозиційного релейного елемента

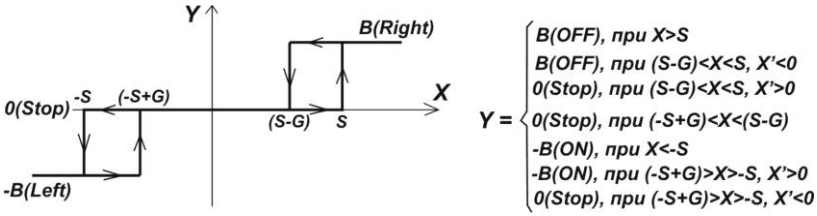

Рис. 16. Статична характеристика реального трипозиційного релейного елемента

Двопозиційні регулятори часто використовуються в задачах, де ставляться невисокі вимоги до якості регулювання параметрів, наприклад в побутових приладах (праски, електричні бойлери тощо). Ці регулятори прості в настроюванні (задається тільки точка завдання та величина гістерезису) і реалізації різноманітними засобами (електричні схеми, механічні пристрої та ін.).

Особливістю використання двопозиційних регуляторів в системах автоматичного регулювання є автоколивальний режим їх роботи.

Алгоритми реалізації двопозиційних регуляторів

Реалізація позиційних регуляторів на мікроконтролерах є досить простою задачею. Проте вирішити її можна декількома способами, одні з яких є більш простими, що зменшує вимоги до обчислювальної потужності мікроконтролера та зменшує обсяг використаної пам'яті – тобто є більш ефективними.

Програмна реалізація ідеальних позиційних регуляторів не являє складності – необхідно використовувати прості операції порівняння вхідного регульованого параметру по зонах і створювати відповідний вплив на дискретному виході (On/Off – для двопозиційного регулятора), або на 2 дискретних виходах (Right/Stop/Left – для трипозиційного регулятора).

У випадку із реальними позиційними регуляторами ситуація ускладнюється через наявність зон неоднозначності, в котрих можливі 2 значення вихідного сигналу.

Статична характеристика двопозиційного регулятора зображена на рис. 17.

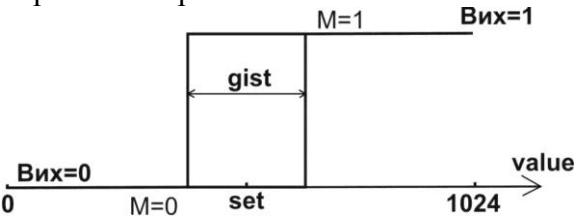

Рис. 17. Статична характеристика двопозиційного регулятора із зоною неоднозначності

Струм, що можуть пропустити виходи мікроконтролера, обмежений. Тому рідко навантаження, яким керує регулятор, безпосередньо підключають до

виводів мікроконтролера. Щоб вмикати потужне навантаження, використовуються схеми підсилення сигналу керування цим навантаженням: біполярні та польові транзистори, драйвери для вибраного навантаження, силові симістори та оптосимістори, електромагнітні та твердотільні реле.

План роботи

1. Реалізувати функцію перетворення коду АЦП в температуру.

2. Створити програму вимірювання температури за допомогою термістора.

3. Додати в програму функцію двопозиційного регулювання температури ввімкненням живлення зовнішнього нагрівача.

Порядок виконання роботи

1. Підключити до аналогового входу мікроконтролера термістор, включений в подільник напруги.

2. Створити функцію, яка обчислює температуру за сумою 32 результатів АЦП.

```
// Значення температури, що повертається, якщо
сума результатів АЦП більше першого значення
таблиці
#define TEMPERATURE_UNDER -250
// Значення температури, що повертається, якщо
сума результатів АЦП менше останнього значення
таблиці
#define TEMPERATURE_OVER 1250
// Значення температури, що відповідає першому
значенню таблиці
#define TEMPERATURE_TABLE_START -250
// Крок таблиці
#define TEMPERATURE_TABLE_STEP 20
// Тип кожного елемента в таблиці
typedef uint16_t temperature_table_entry_type;
//Тип індексу таблиці
```

```
typedef uint8_t temperature_table_index_type;
// Метод доступу до елемента таблиці повинен
відповідати temperature_table_entry_type
#define                TEMPERATURE_TABLE_READ(i)
pgm_read_word(&termo_table[i])
/* Таблиця сумарного значення АЦП залежно від
температури. Від більшого значення до меншого
    для побудови таблиці використані такі
параметри:
     R1(T1): 10кОм (25°C)
     B25/100: 3950
     Ra: 7.5кОм
     Напруги U0/Uref: 5В/5В
*/
const              temperature_table_entry_type
termo_table[] PROGMEM = {
    31149, 30942, 30713, 30463, 30189, 29890,
29566,  29215,  28838,  28432,  27998,  27537,
27047,  26529,  25985,  25415,  24821,  24204,
23566,  22910,  22237,  21551,  20854,  20148,
19438,  18725,  18012,  17303,  16600,  15905,
15221,  14549,  13893,  13252,  12629,  12025,
11441, 10878, 10335, 9814, 9314, 8836, 8379,
7942,  7527,  7132,  6756,  6399,  6061,  5740,
5436,  5149,  4877,  4619,  4376,  4147,  3930,
3725,  3532,  3350,  3177,  3015,  2862,  2717,
2580,  2451,  2330,  2215,  2106,  2003,  1907,
1815, 1729, 1647, 1569, 1496};

// Функція обчислює значення температури у
десятих частках градусів Цельсія на основі
сумарного значення результатів АЦП
  int16_t
calc_temperature(temperature_table_entry_type
adcsum) {
  temperature_table_index_type l = 0;
  temperature_table_index_type       r       =
(sizeof(termo_table) / sizeof(termo_table[0]))
```

```
- 1;
  temperature_table_entry_type     thigh     =
TEMPERATURE_TABLE_READ(r);

  // Перевірка виходу за межі граничних
значень
  if (adcsum <= thigh) {
    #ifdef TEMPERATURE_UNDER
      if (adcsum < thigh)
        return TEMPERATURE_UNDER;
    #endif
    return   TEMPERATURE_TABLE_STEP   *   r   +
TEMPERATURE_TABLE_START;
  }
  temperature_table_entry_type      tlow      =
TEMPERATURE_TABLE_READ(0);
  if (adcsum >= tlow) {
    #ifdef TEMPERATURE_OVER
      if (adcsum > tlow)
        return TEMPERATURE_OVER;
    #endif
    return TEMPERATURE_TABLE_START;
  }

  // Двійковий пошук по таблиці
  while ((r - l) > 1) {
    temperature_table_index_type  m   =   (l   +
r) >> 1;
    temperature_table_entry_type     mid     =
TEMPERATURE_TABLE_READ(m);
    if (adcsum > mid) {
      r = m;
    } else {
      l = m;
    }
  }
  temperature_table_entry_type       vl       =
TEMPERATURE_TABLE_READ(l);
```

```
  if (adcsum >= vl) {
    return   l   *   TEMPERATURE_TABLE_STEP   +
TEMPERATURE_TABLE_START;
  }
  temperature_table_entry_type       vr       =
TEMPERATURE_TABLE_READ(r);
  temperature_table_entry_type vd = vl - vr;
  int16_t res = TEMPERATURE_TABLE_START + r *
TEMPERATURE_TABLE_STEP;
  if (vd) {
    // Лінійна інтерполяція
    res    -=    ((TEMPERATURE_TABLE_STEP    *
(int32_t)(adcsum - vr) + (vd >> 1)) / vd);
  }
  return res;
}
```

3. Додати в програму з попередньої роботи створену функцію, щоб на індикатор виводилось поточне значення температури.

4. Додати в програму умовні оператори, що реалізовують двопозиційний регулятор з неоднозначністю. Вихід регулятора – реле на платі розширення. Щоб подати сигнал з виходу PB5 (контакт 13 в Arduino Uno), встановити перемичку Jmp4 в положення 1. Скомпілювати та вивантажити програму в плату Arduino Uno. Переконатись в правильності роботи програми, нагріваючи термістор.

5. Оформити звіт про виконання лабораторної роботи. Звіт повинен містити: назву та мету лабораторної роботи; тексти програм з коментарями; висновок про виконання роботи.

Контрольні запитання

1. Що таке позиційний регулятор?
2. Де використовуються позиційні регулятори?
3. Які вхідні і вихідні сигнали позиційних регуляторів?

4. Який вигляд має статична характеристика ідеального двопозиційного регулятора?

5. Який вигляд має статична характеристика двопозиційного регулятора із зоною неоднозначності?

6. Який вигляд має статична характеристика ідеального трипозиційного регулятора?

7. Який вигляд має перехідна характеристика системи автоматичного регулювання із двопозиційним регулятором без зони неоднозначності та високо інерційним об'єктом?

8. Який вигляд має перехідна характеристика системи автоматичного регулювання із двопозиційним регулятором із зоною неоднозначності?

9. Який вигляд має статична характеристика трипозиційного регулятора із зонами неоднозначності?

10. Якими способами можна визначити стан вихідного сигналу в зонах неоднозначності?

11. Яким чином найраціональніше реалізувати зону неоднозначності?

12. Який алгоритм реалізації двопозиційного регулятора із зоною неоднозначності на мікроконтролері?

13. Чому не використовують ідеальні позиційні регулятори із мало інерційними об'ктами?

Лабораторна робота 5. Використання модуля USART

Мета роботи: навчитися програмувати мікроконтролер для обміну даними з використанням модуля USART.

Теоретичні відомості

Мікроконтролери AVR мають у своєму складі модуль повнодуплексного універсального синхронного/асинхронного прийомопередавача USART. Через нього здійснюється прийом і передача інформації, представленої послідовним кодом, тому модуль UART

часто називають також послідовним портом. За допомогою цього модуля мікроконтролер може обмінюватися даними з різними зовнішніми пристроями.

Потік даних, що передаються по UART, являє собою сукупність посилок або кадрів. Кожен кадр містить стартовий біт, вісім або дев'ять бітів даних і стоповий біт. Стартовий біт має рівень логічного 0, стоповий - рівень логічної 1. Швидкість передачі даних може варіюватися в широких межах та задається числом в регістрі UBRR.

Відомо, що під час передачі даних можуть статися різні збої. Модуль UART, реалізований в мікроконтролерах, здатний при прийомі виявляти помилку формату і переповнення.

Виводи мікроконтролера, використовувані модулем UART, є лініями порту PORTD.

Прийняті і передані дані (вісім розрядів) зберігаються в регістрі UDR. Фізично регістр UDR складається з двох окремих регістрів, один з яких використовується для передачі даних, інший - для прийому. При читанні регістра UDR виконується звернення до регістру приймача, під час запису - до регістру передавача.

Налаштування модуля здійснюється в регістрах UCSRA, UCSRB, UCSRC, UBRR.

План роботи

1. Реалізувати передачу на комп’ютер через UART поточного значення температури.

2. Реалізувати прийом з комп’ютера через UART заданого значення температури.

Порядок виконання роботи

1. В програму з попередньої роботи додати налаштування модуля USART: швидкість передачі 9600 біт/с, без контролю парності, 8 біт даних, 1 стоп-біт, прийом байта викликає переривання (обробник – функція ISR(USART_RX_vect) {…}).

2. Додати надсилання байту з поточним невід'ємним цілим значенням температури через UART на ПК.

3. Додати прийом байту з поточним невід'ємним цілим завданням температури через UART на ПК. Переконатись в правильності роботи програми.

4. Оформити звіт про виконання лабораторної роботи. Звіт повинен містити: назву та мету лабораторної роботи; тексти програм з коментарями; висновок про виконання роботи.

Контрольні запитання

1. Чим відрізняється USART мікроконтролера від послідовного (COM) порта ПК?

2. Для чого використовується регістр UBRR?

3. Для чого використовується регістр UDR?

4. Як можна зчитувати отримані дані з USART поза функцією main()?

Лабораторна робота 6. Реалізація багатозадачності на мікроконтролері за допомогою FreeRTOS

Мета роботи: налаштувати операційну систему реального часу FreeRTOS для запуску на мікроконтролері ATmega328P; реалізувати систему з витісняючою багатозадачністю за допомогою FreeRTOS.

Теоретичні відомості

У більшості випадків при проектуванні систем на 8-бітних мікроконтролерах не використовують операційної системи, що дає розробнику повний контроль над всіма ресурсами мікроконтролера. За необхідності виконання декількох задач, що вимагають гарантованого часу реакції на зовнішню подію, структура програми значно ускладнюється, збільшується складність відлагодження та супроводу. Використання операційної системи реального часу дозволяє розподілити код програми в декілька відносно простих для супроводу задач, керувати доступом

до ресурсів мікроконтролера, встановлювати пріоритети задач, здійснювати псеводпаралельне виконання задач з однаковим пріоритетом.

В системі FreeRTOS кожен потік виконання називається *задачею* (*task*). Задачі реалізовуються на мові С у вигляді функцій типу *void*, які приймають єдиний аргумент типу *вказівник на void* та виконують нескінченний цикл, ніколи не завершуючись.

*void AtaskFunction(void *pvParameters);*

Одну й ту ж функцію можна використати для запуску кількох задач. Створення задачі здійснюється функцією

portBASE_TYPE ***xTaskCreate****(pdTASK_CODE pvTaskCode,*

*const signed portCHAR * const pcName,*
unsigned portSHORT usStackDepth,
*void *pvParameters,*
unsigned BaseType_t uxPriority,
*xTaskHandle *pxCreatedTask);*

Функція повертає значення pdTRUE у випадку успішного створення задачі. *pvTaskCode* – ім'я функції, що реалізує задачу; *pcName* – назва задачі, що використовуватиметься при відлагодженні; *usStackDepth* – розмір стеку задачі; *pvParameters* – вказівник на ділянку пам'яті, що містить параметри, які передаються в задачу при її виклику; *uxPriority* – пріоритет задачі; *pxCreatedTask* – handle створеної задачі, який використовується для маніпуляцій з задачею (призупинка, зміна пріоритету тощо).

Задача може знаходитись у наступних станах:

- Готова до виконання (READY). Задача готова прийняти на себе управління. Як тільки дійде її черга, планувальник (диспетчер) задач переведе задачу в режим RUN.

- Виконується (RUN). Планувальник передав управління задачі, процесор виконує програмний код задачі в даний момент. У цей момент задача споживає процесорний час і робить корисну роботу, заради якої вона була створена.
- Блокована/очікує (WAIT). Задача очікує подію, зовнішню (надходження зовнішнього сигналу переривання, отримання байту модулем USART) або внутрішню (закінчився час затримки). Планувальник не перемикається на неї, процесорний час не витрачається. Як тільки очікувана подія відбудеться, то FreeRTOS призначить цій задачі стан READY.
- Зупинена (SUSPEND). Задача не вивантажена з пам'яті, дані її збережені, але вона неактивна, на жодні події не реагує і сама з цього стану вийде. Вивести її з цього стану можна тільки API командою ОС з іншої задачі.

Знищити задачу і звільнити зайняту нею пам'ять можна функцією *vTaskDelete(pxCreatedTask)*.

У випадку витісняючої багатозадачності переремикання між задачами планувальник здійснює з частотою *configTICK_RATE_HZ* (макрос у файлі *FreeRTOSConfig.h*): в кінці кожного кванту часу (величина, обернена до *configTICK_RATE_HZ*) зберігає контекст поточної задачі, на початку наступного завантажує контекст задачі з найвищим пріоритетом серед готових до виконання та передає керування цій задачі. Задача може не чекати кінця виділеного їй кванту часу, а викликати диспетчер раніше викликом *taskYIELD()*. Щоб використовувати кооперативну багатозадачність замість витісняючої, потрібно у файлі конфігурації операційної системи *FreeRTOSConfig.h* встановити макрос

#define configUSE_PREEMPTION 0

При цьому перемикання між задачами диспетчер здіснюватиме лише після виклику *taskYIELD()*, не маючи

змоги перервати виконання задачі за власною ініціативою, тому для забезпечення мінімального часу реакції системи потрібно якомога раніше викликати диспетчер. Оскільки виклики диспетчера відбуваються в наперед відомі моменти, то при перемиканні між задачами потрібно зберігати менше даних, тому використання кооперативної багатозадачності потребує менших затрат оперативної пам'яті, ніж витісняючої.

Затримка у виконанні коду задачі може бути здійснена функцією

*void **vTaskDelay**(portTickType xTicksToDelay);*

Величина затримки *xTicksToDelay* задається у кількості спрацювань системного таймера (tick), для переходу до задання затримки в мілісекундах використовується константа *portTICK_RATE_MS:*

vTaskDelay(200 / portTICK_RATE_MS); //затримка 200 мс

Оскільки затримка відраховується з моменту виклику функції, то для виклику задачі періодично кожні, наприклад, 200 мс потрібно віднімати час між викликом задачі та викликом функції затрики, щоб період не збільшився до величини (200 мс + час між викликом задачі та викликом функції затримки). Для таких випадків передбачена функція

*void **vTaskDelayUntil**(portTickType * pxPreviousWakeTime, portTickType xTimeIncrement);*

Затримка рахується від моменту часу *pxPreviousWakeTime*, який повинен бути ініціалізований у задачі до виклику згаданої функції.

```
void vTaskFunction( void *pvParameters )
{
char *pcTaskName;
portTickType xLastWakeTime;
```

```
        /* Ініціалізація значення xLastWakeTime
*/
        xLastWakeTime = xTaskGetTickCount();

        while(1)
        {
        ...//дії, які задача повинна виконувати
кожні 200 мс
        vTaskDelayUntil(        &xLastWakeTime,
(200/portTICK_RATE_MS));              /*змінна
xLastWakeTime автоматично збільшується при
виклику функції vTaskDelayUntil*/
        }
      }
```

Для реалізації передачі даних між задачами першим методом, що спадає на думку, є використання глобальних змінних, до яких має доступ кожна задача. Оскільки виконання операцій зі змінними довжиною більше 1 байта 8-бітний мікроконтролером затрачається більше одного машинного циклу, то гарантувати правильність значення змінної без додаткових засобів неможливо: нехай перша задача змінює значення змінної з 255 (0b011111111) на 256 (0b100000000) записом 1 у старший байт і 0 у молодший байт; у момент між записом двох байтів закінчується квант часу, виділений першій задачі й диспетчер передає керування другій задачі, яка зчитує значення змінної як 1023 (0b111111111, старший байт 00000001 містить нове значення, молодший байт 11111111 - попереднє значення, яке перша задача не встигла змінити). Це явище є порушенням атомарності операції (атомарною є операція, яка виконується без переривань). Щоб виключити таку ситуацію, використовуються черги повідомлень (задача надсилає дані в чергу і далі вже операційна система відповідає за те, щоб інша задача зчитала з черги правильне значення), семафори (семафор встановлюється

перед записом та скидається після запису; друга задача зчитує змінну лише якщо семафор скинутий), критичні секції (ділянки програми, в яких заборонено виклик диспетчера і відповідно перемикання на іншу задачу).

Нехай перша задача записує в послідовний порт рядок "PV=158", а друга – "SP=120". При однаковому пріоритеті задач можлива ситуація, коли планувальник перерве виконання першої задачі в момент запису і відновить виконання другої, в результаті приймаюча сторона може отримати "PV=15SP=1280". Для арбітражу доступу кількох задач до одного апартного ресурсу служать м'ютекси. Задача перед доступом до апартного ресурсу перевіряє, чи вільний м'ютекс, що відповідає цьому ресурсу. Якщо так, задача захоплює м'ютекс, здійснює потрібні дії з ресурсом та звільняє м'ютекс. Якщо м'ютекс вже захоплений іншою задачею, операційна система переведе задачу в стан очікування (WAIT) до моменту, коли м'ютекс знову стане доступним, або до закінчення таймауту. М'ютекс реалізовується на базі семафору, тому для його створення використовується функція

xSemaphoreHandle SemaphoreCreateMutex(void),

яка повертає handle, що використовується для операцій з м'ютексом. Захопити та звільнити м'ютекс можна функціями

xSemaphoreTake*(SemaphoreHandle_t xMutex,*
TickType_t xTicksToWait),
xSemaphoreGive*(SemaphoreHandle_t Mutex)*

відповідно, де *xMutex* – handle м'ютекса. Якщо впродовж *xTicksToWait* задача не змогла захопити м'ютекс, виконання задачі буде продовжено, а функція поверне pdFALSE.

Ще одним засобом виключення одночасного доступу декількох задач до ресурсу є задача-gatekeeper. Лише вона має виключний доступ до ресурсу й всі інші задачі повинні обмінюватись даними з цією задачею для

роботи з ресурсом.

Черга повідомлень створюється функцією

xQueueHandle ***xQueueCreate****(unsigned BaseType_t uxQueueLength, unsigned BaseType_t uxItemSize);*

де *uxQueueLength* - максимальна кількість елементів у черзі, *uxItemSize* - розмір одного елемента. Функція повертає handle черги.

Для запису елементу в чергу служить функція

BaseType_t ***xQueueSend****(xQueueHandle xQueue, const void * pvItemToQueue, portTickType xTicksToWait);*

де *xQueue* – handle черги, *pvItemToQueue* – вказівник на елемент, який необхідно надіслати в чергу, *xTicksToWait* – максимальний час очікування вільного місця в черзі. Для надсилання даних поза чергою, в початок черги, служить функція *xQueueSendToFront* з аналогічним попередній прототипом.

Зчитати елемент з черги можна функцією

BaseType_t ***xQueueReceive****(xQueueHandle xQueue, const void * pvBuffer, portTickType xTicksToWait)*

Функція чекає час *xTicksToWait,* доки з'явиться елемент в черзі *xQueue*, та записує його за адресою, на яку вказує *pvBuffer*.

Якщо необхідно, щоб функція чекала необмежено довго потрібної події, потрібно вказати час очікування *xTicksToWait* рівним portMAX_DELAY.

Структура FreeRTOS

FreeRTOS розповсюджується у вигляді архіву з 2 каталогів: власне FreeRTOS та FreeRTOS-Plus. Остання містить реалізацію підтримки файлових систем та функцій роботи з IP-мережами. Каталог FreeRTOS містить каталоги Source (вихідний код операційної системи), Demo (каталог з демонстраційними проектами), License (містить файл з ліцензійною угодою). Демо-проекти розміщені в каталогах *ім'я_платформи_компілятор*, наприклад PIC18_MPLAB,

CORTEX_STM32F103_GCC_Rowley, та каталог Common зі спільним для всіх демо-проектів кодом.

У каталозі Source та його підкаталозі include розташовані архітектуронезалежні файли, а в каталозі Source/portable – специфічні для кожного компілятора та платформи файли port.c та portmacro.h, в яких реалізовано механізм перемикання контексту, та каталог MemMang, що містить файли з реалізацією операцій з купою (heap).

Щоб використати FreeRTOS у проекті, потрібно додати в нього файли tasks.c, list.c (містять реалізацію планувальника задач та API-функції роботи з задачами), *queue.c* (реалізація черг повідомлень), один з файлів *heap_x.c* з теки *MemMang* (реалізація функцій роботи з пам'яттю), *port.c* та файл *FreeRTOSConfig.h*, де здійснюється конфігурування операційної системи. У випадку використання семафорів або м'ютексів додатково потрібно підключити *semphr.h,* співпрограм – *croutine.h.*

План роботи

1. Налаштувати дистрибутив FreeRTOS для роботи на лабораторній платі.

2. Реалізувати псевдопаралельне виконання 2 задач блимання світлодіодами.

3. Реалізувати ПІД-імпульсний регулятор з індикацією та сигналізацією, кожну з функцій якого винести в окрему задачу.

Порядок виконання роботи

1. Завантажити з https://www.freertos.org/ архів з операційною системою. Розпакувати його та знайти спільні файли для всіх проектів, апаратно-залежну частину для мікроконтролерів ATmega323 й середовища WinAVR (IDE на основі компілятора AVR GCC), файл конфігурації. Модифікувати апаратно-залежну частину для ATmega328P (налаштування переривань таймера 1).

2. Створити проект, що містить дві задачі: *vBlinkPB5Task* здійснює блимання світлодіодом, підключеним до виводу PB5 мікроконтролера, *vBlinkPB4Task* – світлодіодом, підключеним до виводу PB4. Переконатись у псевдопаралельному виконанні обох задач. Головну функцію предаствити у вигляді

```
void main( )
{
xTaskCreate(vBlinkPB5Task,          "B5",
configMINIMAL_STACK_SIZE,   NULL,  0,  NULL  );
//створення першої задачі
xTaskCreate(vBlinkPB4Task,          "B4",
configMINIMAL_STACK_SIZE,  NULL,  0,  NULL  );
//створення другої задачі
vTaskStartScheduler();           //запуск
планувальника задач
}
```

Задачі можна оформити за зразком

```
static    void    vBlinkPB5Task(    void
*pvParameters )
{
      /* The parameters are not used. */
      ( void ) pvParameters;
      DDRB|=0b00100000;
      for( ;; )
      {
            PORTB|=1<<PB5;
            vTaskDelay(       400       /
portTICK_RATE_MS );
            PORTB&=~(1<<PB5);
            vTaskDelay(       400       /
portTICK_RATE_MS );
```

```
        }
    }
```

3. Оформити звіт про виконання лабораторної роботи. Звіт повинен містити: назву та мету лабораторної роботи; текст програми з коментарями; файл конфігурації операційної системи; висновок про виконання роботи.

Контрольні запитання

1. Які вимоги ставляться до функції, що реалізовує задачу?
2. У яких станах може знаходитись задача?
3. Як планувальник задач визначає, яку задачу запустити на виконання?
4. Як задається вид багатозадачності (витісняюча чи кооперативна) у FreeRTOS?
5. Де задається частота виклику планувальника задач у FreeRTOS?
6. Як реалізувати виконання функції PID() кожні 0,05 с?
7. Які засоби FreeRTOS використовуються, щоб виключити можливість порушення атомарності операції?
8. Що таке задача-gatekeeper?
9. Який файл містить специфічні для конкретної платформи механізми перемикання контексту?

Лабораторна робота 7. Реалізація ПІД-регулятора на базі FreeRTOS

Мета роботи: навчитися програмувати багатозадачні системи регулювання на базі мікроконтролера ATmega328P.

Теоретичні відомості

Для взаємодії задач у FreeRTOS використовуються семафори, м’ютекси, черги повідомлень, можуть використовуватись глобальні змінні за умови забезпечення

атомарності операцій доступу, наприклад, з використанням критичних секцій.

Функції роботи з семафорами та м'ютексами описані в **semphr.h**

Створення двійкового семафора

SemaphoreHandle_t ***SemaphoreCreateBinary****(void);*

Створення лічильного семафора

SemaphoreHandle_t ***xSemaphoreCreateCounting****(UBaseType_t uxMaxCount, UBaseType_t uxInitialCount);*

Захоплення (очікування) семафора

xSemaphoreTake*(SemaphoreHandle_t xSemaphore, TickType_t xTicksToWait);*

Таймаут xTicksToWait=*portMAX_DELAY* призведе до необмеженого очікування

Видача (звільнення) семафора

xSemaphoreGive*(SemaphoreHandle_t xSemaphore);*

Приклад використання двійкового семафора для сигналізації про настання події іншій задачі наведено нижче.

```
SemaphoreHandle_t xSemaphore = NULL;
void vATask( void * pvParameters )
{
   xSemaphore = xSemaphoreCreateBinary();
      ...
   xSemaphoreGive(     xSemaphore     );
//сигналізуємо іншій задачі
      ...
}

void vAnotherTask( void * pvParameters )
{
    if( xSemaphore != NULL )
    {
        if(  xSemaphoreTake(  xSemaphore,
( TickType_t ) 10 ) == pdTRUE )
```

```
            {          //виконуємо дії при
надходженні сигналу
            }
            else
            {     //виходимо по таймауту (10)
й виконуємо дії, передбачені при відсутності
сигналу від іншої задачі
            }
         }
      }
```

Для створення м’ютекса використовується функція *SemaphoreHandle_t* ***SemaphoreCreateMutex****(void)*, а для операцій з ним – функції роботи з семафорами:

```
      SemaphoreHandle_t xLcdMutex;
      void vATask( void * pvParameters )
      {  xLcdMutex = xSemaphoreCreateMutex();
         if( xLcdMutex != NULL )
         {
            //м’ютекс створено і може бути
використано
         }
      }
      void vBTask( void * pvParameters )
      {
          if( xLcdMutex != NULL )
          {
            //намагаємось захопити м’ютекс
              if(   xSemaphoreTake(   xLcdMutex,
( TickType_t ) 10 ) == pdTRUE )
            {     //виконуємо дії в режимі
ексклюзивного доступу до ресурсу
                LCD_print(“Hello, world”);
                xSemaphoreGive( xLcdMutex );
//звільняємо м’ютекс
            }
            else
            {   // не вдалось отримати доступ
до ресурсу
```

```
        }
    }
}
```

За можливості всі налаштування модулів мікроконтролера, потрібні для роботи задачі, варто виконувати на початку цієї задачі (до початку нескінченого циклу), а не виносити за її межі. Це полегшує повторне використання коду цієї задачі в інших проектах.

План роботи

1. Реалізувати задачу індикації.
2. Реалізувати задачі опитування АЦП і сигналізації.
3. Реалізувати ПІД-імпульсний регулятор з індикацією та сигналізацією з використанням розроблених задач.

Порядок виконання роботи

1. Відкрити main.c з попередньої лабораторної роботи. Перенести функції виведення на динамічний індикатор з лабораторної роботи 2, а перемикання розрядів реалізувати в окремій задачі замість обробника переривань таймера 1. Щоб заблокувати виконання задачі після підсвічування потрібного розряду індикатора, використати taskYIELD(). Додати в цю ж задачу налаштування портів вводу-виводу, що використовуються для індикації, скомпілювати програму й перевірити її роботу.

2. Реалізувати вимірювання регульованої величини з входу ADC0 кожні 10 мс. Регульована величина обчислюється за формулою PV=0,185*ADC0+53. Закінчення аналогово-цифрового перетворення повинно генерувати переривання. Про готовність результату аналогово-цифрового перетворення з обробника переривання просигналізувати семафором (функція xSemaphoreGiveFromISR). Поточне значення регульованої величини оновлювати на індикаторі (передавати для задачі

індикації) кожні 0,2 с. У випадку перевищення PV значення 200 повинна вмикатись сигналізація блиманням світлодіода на PB4 з частотою 2 Гц. Для забезпечення атомарності запису глобальної змінної зі значенням регульованої величини, використати критичні секції (taskENTER_CRITICAL(), taskEXIT_CRITICAL()).

3. Створити задачу ПІД-регулювання, використовуючи рекурентний алгоритм. Частота розрахунку керуючого впливу – 10 Гц. Тривалість імпульсу – від 0 до 5000 мс, пауза після кожного імпульсу – 1000 мс. Задача повинна передавати тривалість імпульсу наступній задачі. Для розрахунку керуючого впливу використати формулу u=u1+q0*e+q1*e1+q2*e2. Розрахунок коефіцієнтів q здійснити за формулою q0=k*(1+T0/Ti+Td/T0); q1=-k*(1+2*Td/T0-T0/(2*Ti)); q1=k*Td/T0;

4. Створити задачу, яка реалізовує імпульсний вихід на виводі PB5, на базі задачі vBlinkPB5Task з лабораторної роботи 6. Скомпілювати програму, переконатись в правильності роботи програми, за потреби здійснити налагодження.

5. Оформити звіт про виконання лабораторної роботи. Звіт повинен містити: назву та мету лабораторної роботи; текст програми з коментарями; висновок про виконання роботи.

Контрольні запитання

1. Як створити двійковий семафор?

2. Для чого призначений м'ютекс і який заголовний файл необхідно підключити для його використання у програмі?

3. Як захопити м'ютекс?

4. Чи є атомарною операція зчитування змінної типу float в мікроконтролерах архітектури AVR? Чому?

Лабораторна робота 8. Введення-виведення цифрових сигналів мікроконтролером STM32F072C8

Мета роботи: навчитися програмувати мікроконтролери серії STM32F0 для роботи з механічними кнопками та дискретними індикаторними світлодіодами з використанням портів вводу-виводу загального призначення.

Теоретичні відомості

Порти вводу-виводу загального призначення (GPIO) мікроконтролерів родини STM32F0 мають більше режимів роботи, між порти мікроконтролерів AVR. Порт (рис. 18) може бути налаштовано як:

- “плаваючий” (високоомний) вхід;
- вхід з підтяжкою до напруги живлення (pull-up);
- вхід з підтяжкою до потенціалу [віртуальної] землі (pull-down);
- вихід push-pull;
- вихід з відкритим стоком.

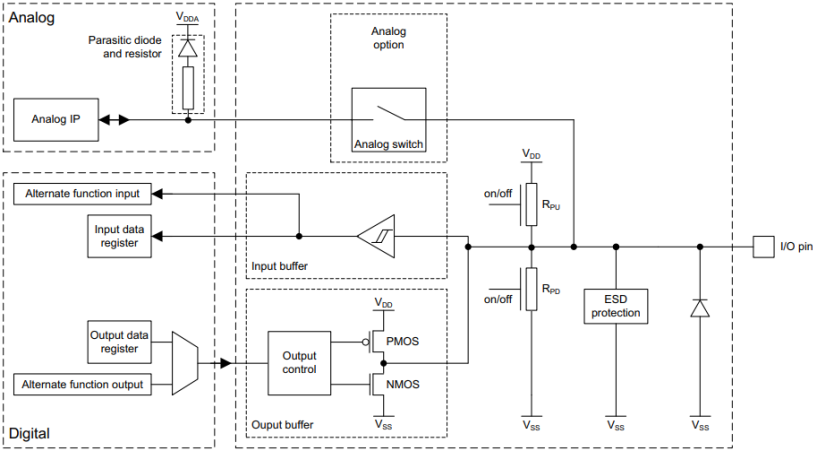

Рис. 18. Структура портів вводу-виводу загального призначення типу FT та TT

Режими роботи на вхід з підтяжкою до напруги живлення VDD або з підтяжкою до потенціалу землі VSS задіюють внутрішні підтягуючі резистори опором близько 40 кОм R_{PU} та R_{PD} відповідно.

В режимі push-pull порт працюватиме на вихід аналогічно єдиному вихідному режиму порта AVR-мікроконтролера ATmega328P: відкриватиме n-канальний транзистор NMOS і закриватиме p-канальний PMOS для виведення напруги логічного нуля та закриватиме n-канальний транзистор NMOS і відкриватиме p-канальний PMOS для виведення напруги логічної одиниці.

Режим виходу з вікдритим стоком задіює лише n-канальний транзистор NMOS, а p-канальний транзистор PMOS залишається завжди закритим. Таким чином вивід мікроконтролера може або підключатись до потенціалу землі VSS, або відключатись і не мати чітко заданого потенціалу. Відповідно, щоб отримати різну напругу при виведенні логічного нуля та одиниці, потрібно задіювати зовнішній або внутрішній підтягуючий резистор до напруги живлення.

Налаштування порта вводу-виводу можна здійснювати безпосереднім записом у кілька регістрів налаштувань GPIO або використати функцію

*HAL_GPIO_Init(GPIOx, *GPIO_InitTypeDef),*

де GPIO_InitTypeDef – вказівник на структуру налаштування порта, GPIOx – порт (GPIOA, GPIOB, …).

В структурі налаштувань порта режим роботи задається значеннями GPIO_MODE_OUTPUT_PP (вихід push-pull), GPIO_MODE_OUTPUT_OD (вихід з відкритим стоком), GPIO_MODE_INPUT (вхід). Конкретизується вхідний режим полем Pull, в яке можна записати значення GPIO_NOPULL (“плаваючий” високоомний вхід), GPIO_PULLUP (вхід з підтяжкою до напруги живлення),

GPIO_PULLDOWN (вхід з підтяжкою до потенціалу землі).

Стан входу (для прикладу, PC10) можна отримати викликом функції HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_10), яка повертає GPIO_PIN_SET (логічна 1) або GPIO_PIN_RESET (логічний 0).

Змінити стан виходу PC13 можна функціями

HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET); //скинути в 0

HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET); //встановити в 1

HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_13); // перемкнути в протилежний стан

Частота портів вводу-виводу загального призначення не задається однозначно частотою тактування обчислювального ядра, як в мікроконтролерах архітектури AVR, а може задаватись низькою (GPIO_SPEED_FREQ_LOW), середньою (GPIO_SPEED_FREQ_MEDIUM), високою (GPIO_SPEED_FREQ_HIGH) за допомогою бітів OSPEEDRx. Конкретні значення частоти вказуються в технічному описі (даташиті, datasheet) відповідного мікроконтролера. В STM32CubeIDE частотою GPIO за замовчуванням є низька, що при напрузі живлення 3,3 В, згідно таблиці 55 «I/O AC characteristics» в технічному описі, відповідає 2 МГц. Якщо напруга живлення менше 2 В, відповідні частоти будуть менші.

Порти вводу-виводу загального призначення відрізняються за своїми характеристиками. Звичайні порти вводу-виводу, розраховані на напругу 3,3 В, позначаються в технічному описі TC і не витримують вхідної напруги 5 В. Порти, позначені TTa, мають пряме підключення до АЦП й теж розраховані на напругу 3,3 В. Порти, що можуть

працювати з вхідною напругою 5 В, позначаються FT (5 V-tolerant I/O).

Всі порти вводу-виводу, за вийнятком SWDIO та SWCLK, під час та після скидання мікроконтролера налаштовуються як високоомні входи, напруга на яких не буде стабільною в непідключеному стані. Електромагнітні поля, що нас оточують, змінюватимуть напругу на високоомному вході, що призводитиме до хаотичного перемикання вхідного буфера порту то в стан логічної одиниці при підвищенні напруги до рівня VIH (див. таблицю 53 «I/O static characteristics» в технічному описі), то в стан логічного нуля при падінні напруги до VIL. Непідключений високоомний вхід споживатиме більший струм, ніж вхід, на якому встановлена напруга одного логічного рівня. Щоб уникнути зайвих перемикань стану входів, зменшити стрибки струму споживання мікроконтролера та його абсолютне значення, на непідключених входах варто активувати внутрішній підтягуючий резистор до потенціалу землі чи живлення або підключити цей вхід до «плюса» чи «мінуса» живлення мікроконтролера. Це забезпечить стабільний сигнал на вході.

Підключення кнопок та світлодіодів до мікроконтролера STM32F072C8T6 здійснюється за тими ж схемами (рис. 1), що й AVR-мікроконтролерів.

У схемі з підтяжкою до напруги живлення натиснутій кнопці на вході мікроконтролера відповідає напруга логічного нуля, відпущеній – логічної одиниці. У схемі з підтяжкою до землі натиснутій кнопці на вході мікроконтролера відповідає напруга логічної одиниці, відпущеній – логічного нуля.

План роботи

1. Реалізувати періодичне ввімкнення-вимкнення навантаження, підключеного до контакту PA8.

2. Зчитати стан кнопки на контакті PB5 та просигналізувати його за допомогою світлодіода, підключеного до контакту PA8.

3. Реалізувати зміну періоду блимання світлодіода двома кнопками, підключеними до контактів PC14 та PC15.

Порядок виконання роботи

1. Запустити STM32CubeIDE, створити новий проект мовою С й налаштувати в іос-файлі на вихід контакт PA8, до якого підключити світлодіод: анод світлодіода підключити до виходу мікроконтролера, катод – через струмообмежувальний резистор опором 470 Ом…2,2 кОм до GND. Використовуючи функції void HAL_GPIO_WritePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState) та void HAL_Delay (uint32_t Delay), реалізувати блимання світлодіода з частотою 1 Гц. Аргумент Delay – кількість мілісекунд, на які треба затримати виконання програми. Задамо тривалість ввімкнення 700 мс та тривалість вимкнення та 300 мс, що в сумі складе період 1000 мс й дасть частоту 1 Гц. Замінимо вміст нескінченого циклу в функції main() з попередньої програми на два виклики HAL_GPIO_WritePin(), розділені двома викликами функції затримки HAL_Delay().

Скомпілювати проект, завантажити програму в мікроконтролер, в режимі налагодження зупинити програму й пересвідчитись, що запаленому стану світлодіода відповідає 1 у регістрі ODR, а погашеному – 0.

2. Підключити кнопку до контакту PB5 (контакт INT на групі контактів GY-521) та налаштувати підтяжку до напруги живлення (рис 3.2.а). Для цього в іос-файлі двічі клацнути по контакту PB5, обрати режим GPIO_Input (рис. 19).

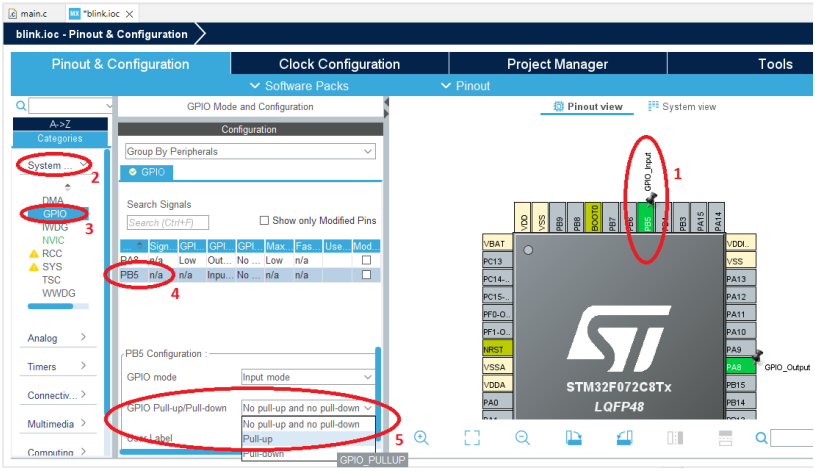

Рис. 19. Налаштування контакту PB5 як входу з підтяжкою до напруги живлення

Далі ліворуч обрати розділ «System Core», в ньому – GPIO, в отриманій таблиці виділити рядок з контактом PB5, обрати в списку властивості «GPIO Pull-up/Pull-down» варіант «Pull-up». Зберегти зміни й згенерувати оновлений програмний код.

Для зчитування логічного сигналу на вході використаємо функцію GPIO_PinState HAL_GPIO_ReadPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin). Далі модифікуємо основний цикл програми так, щоб при надходженні логічного нуля на вхід PB5 запалювався світлодіод, підключений до контакту PA8. Для цього додамо умову, що порівняє значення, повернуте функцією HAL_GPIO_ReadPin, з GPIO_PIN_RESET (натиснутій кнопці відповідає низький рівень напруги), і, якщо умова істинна, виводить високий рівень напруги (логічну одиницю) на вихід PA8 викликом HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_SET). Якщо умова не виконується, викликаємо

цю ж саму функцію, але з останнім аргументом GPIO_PIN_RESET.

Скомпілювати проект, прошити мікроконтролер і перевірити реакцію мікроконтролера на натиснення кнопки.

3. Змінити схему й активувати підтяжку до потенціалу спільного провідника (віртуальної землі) згідно рис. 1.б. Відповідно в іос-файлі в списку властивості «GPIO Pull-up/Pull-down» слід обрати варіант «Pull-down». Тепер натиснутій кнопці відповідатиме високий рівень напруги на вході, а відпущеній – низький. Змінюємо значення, з яким порівнюється значення, повернуте функцією HAL_GPIO_ReadPin(), на протилежне (GPIO_PIN_SET). В результаті умовний оператор матиме вигляд

```
if(HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_5) == GPIO_PIN_SET)
                    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_SET);
else
                    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET);
```

Скомпілювати проект, прошити мікроконтролер і перевірити реакцію мікроконтролера на натиснення кнопки, підключеної за іншою схемою.

4. Реалізувати збільшення частоти блимання світлодіода кнопкою Btn1 (вхід PC14), а зменшення – Btn3 (PC15). Кнопки Btn1 та Btn3 одним контактом з’єднані з GND/Vss, відповідно повинні використовуватись з підтяжкою до напруги живлення. Знову змінимо іос-файл, вказавши в списку властивості «GPIO Pull-up/Pull-down» варіант «Pull-up».

Далі передбачимо змінну, що зберігатиме значення затримки в мілісекундах. Розмістимо її проголошення у відповідній секції:

```
/* Private variables ---------------------------------------*/
/* USER CODE BEGIN PV */
int delayMs=500;
/* USER CODE END PV */
```

Початкову затримку задамо 500 мс. Додамо умови, яка проаналізують стан обох кнопок. По натисканню однієї кнопки будемо збільшувати затримку на 100 мс, іншої – зменшувати. Передбачимо обмеження зміни затримок у допустимих межах.

Перевірку двох умов за допомогою двох умовних операторів можна здійснити в одному, об’єднавши умови логічним «І». Тоді тіло головного циклу спроститься.

Скомпілювати проект, завантажити програму в мікроконтролер, в режимі налагодження перевірити реакцію мікроконтролера на натискання кнопок, переконатись в зміні значення змінної, що задає величину затримки.

5. Оформити звіт про виконання лабораторної роботи. Звіт повинен містити: назву та мету лабораторної роботи; тексти програм з коментарями; висновок про виконання роботи.

Контрольні запитання

1. Які засоби введення/виведення інформації користувача використовуються в мікропроцесорній техніці?
2. Які схеми використовуються при підключенні кнопок до мікропроцесорних пристроїв?
3. Які схеми використовуються при підключенні окремих світлодіодів до мікропроцесорних пристроїв?
4. Які види «підтяжок» доступні в мікроконтролерах родини STM32F0?

5. Як зчитати стан входу за допомогою бібліотеки HAL для мікроконтролерів STM32?

6. Як змінити стан виходу на протилежний за допомогою бібліотеки HAL для мікроконтролерів STM32?

7. Як змінити стан виходу на заданий за допомогою бібліотеки HAL для мікроконтролерів STM32?

8. В якому вхідному режимі має працювати вхід, щоб зчитувати вихідний сигнал датчика з виходом типу «сухий контакт»?

Лабораторна робота 9. Портування програми регулювання для STM32F072C8

Мета роботи: навчитися портувати програми для мікроконтролерів; навчитися використовувати АЦП мікроконтролерів родини STM32F0 в режимі неперервного перетворення.

Теоретичні відомості

Під час життєвого циклу мікропроцесорного пристрою може виникнути задача заміни мікроконтролера в його основі на інший внаслідок зміни цінової політики виробника мікроконтролера, появи дешевших замінників, зміни вимог до функціональності пристрою, виявлення апаратних недоліків. Це вимагає портування старої програми на новий пристрій. Портування в межах однієї родини мікроконтролерів може обмежитись переналаштування портів вводу-виводу, пов’язаних зі апаратними змінами у друкованій платі. Якщо в новому мікроконтролері використовуються відмінні модулі (наприклад, 10-бітний АЦП з програмованим підсиленням замість 10-бітного АЦП без нього), потрібно вносити зміни й у ділянки програми, пов’язані з цією периферією.

Портування на інші родини або навіть інші архітектури потребує більш значних змін. Може змінитись розмір доступної пам’яті, глибина стеку, доступні режими

сну, частоти. Інша архітектура може вимагати використання інших компіляторів, які передбачають відмінну розмірність типів даних, інший синтаксис роботи з бітами тощо.

STM32F072C8 містить 12-бітний АЦП з можливістю сканування групи каналів, неперервної роботи й запуску по події-тригеру, генеруванням переривання про готовність результату, а також передачею результату через DMA. Бібліотека HAL містить функції для роботи з АЦП, імена яких починаються на HAL_ADC_, зокрема:

Ініціалізація модуля АЦП

HAL_StatusTypeDef
HAL_ADC_Init*(ADC_HandleTypeDef * hadc)*

hadc — вказівник на структуру конфігурації модуля АЦП

повертає

HAL_OK при успішній ініціалізації; HAL_ERROR при помилці

Налаштування каналу АЦП

HAL_StatusTypeDef
HAL_ADC_ConfigChannel*(ADC_HandleTypeDef * hadc, ADC_ChannelConfTypeDef * sConfig)*

sConfig — вказівник на структуру конфігурації каналу

```
typedef struct
{
  uint32_t Channel;     // номер каналу
  uint32_t Rank;            // послідовність сканування каналу, в STM32F0 не викор.
  uint32_t SamplingTime;          // час вибірки
} ADC_ChannelConfTypeDef;
```

Запуск модуля АЦП

HAL_StatusTypeDef
HAL_ADC_Start*(ADC_HandleTypeDef * hadc)*

hadc — вказівник на структуру конфігурації модуля АЦП

повертає

HAL_OK при успішному запуску; HAL_ERROR при помилці

Очікування закінчення перетворення

HAL_StatusTypeDef ***HAL_ADC_PollForConversion****(ADC_HandleTypeDef * hadc, uint32_t Timeout)*

Timeout — таймаут в мілісекундах

Запуск модуля АЦП з перериванням

HAL_StatusTypeDef ***HAL_ADC_Start_IT****(ADC_HandleTypeDef * hadc)*

Запуск модуля АЦП з передачею результату за допомогою DMA

HAL_StatusTypeDef ***HAL_ADC_Start_DMA****(ADC_HandleTypeDef * hadc, uint32_t * pData, uint32_t Length)*

pData – вказівник на буфер даних,

Length – кількість даних.

Зчитування результату перетворення (ADC_DR)

uint32_t ***HAL_ADC_GetValue****(ADC_HandleTypeDef * hadc)*

зчитує черговий канал АЦП (в STM32F0 в порядку зростання номера каналу)

План роботи

1. Налаштувати 12-бітний АЦП мікроконтролера STM32F072C8T6.

2. Портувати програму двопозиційного регулювання з індикацією темпратури, написану для AVR-мікроконтролера ATmega328P, на мікроконтролер STM32F072C8T6 архітектури ARM Cortex-M0.

Порядок виконання роботи

1. Запустити середовище розробки STM32CubeIDE, створити проект мовою С, в ioc-файлі налаштувати АЦП на виконання неперервного перетворення напруги з ADC5 з максимальним часом вибірки 239,5 циклів (рис. 20). Щоб в регістрі результату АЦП завжди було найсвіжіше значення, задамо параметру «Overrun behaviour» значення «Overrun data overwritten». Це призведе до перезапису попереднього результату аналогово-цифрового перетворення, якщо він не зчитаний, новим результатом вимірювання напруги. На вкладці Clock Configuration задамо системну тактову частоту 48 МГц.

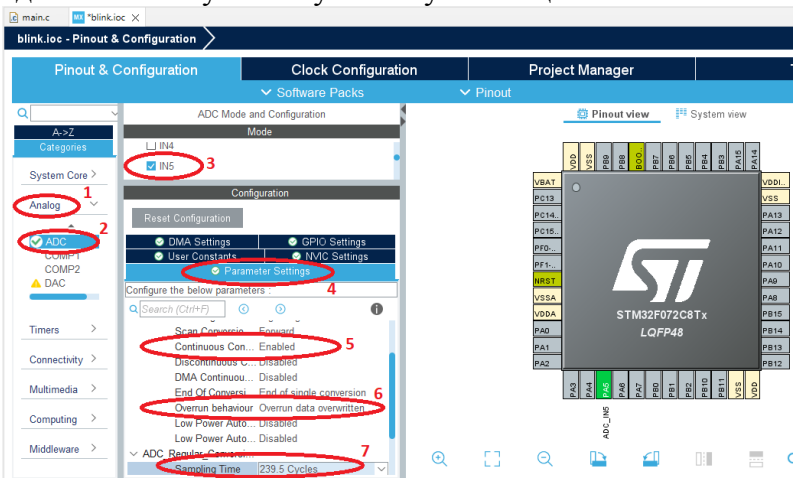

Рис. 20. Налаштування модуля АЦП в STM32CubeIDE

В функцію main() додамо проголошення змінної int adc_res=0 для збереження зчитаного результату аналогово-цифрового перетворення, запустимо АЦП в режимі неперервного перетворення перед нескінченим циклом викликом функції HAL_ADC_Start(&hadc), де hadc – хендл модуля АЦП. У тілі циклу потрібно буде почекати готовності результату аналогово-цифрового перетворення викликом функції HAL_ADC_PollForConversion(&hadc,

100), тут 100 – таймаут очікування результату. Врешті, залишається зчитати результат у створену раніше змінну: adc_res = HAL_ADC_GetValue(&hadc).

Скомпілювати програму й завантажити її в мікроконтролер. У режимі налагодження перевірити значення змінної adc_res: поки до аналогового входу ADC_IN5 нічого не підключено, результати аналогово-цифрового перетворення хаотично змінюватимуться, оскільки фактично на вході АЦП матимемо шум.

2. Перевіривши роботу АЦП, налаштуємо таймер. У програмі для AVR використовувався 16-розрядний таймер 1, а точна частота його спрацювання задавалась регістром OCR1A в режимі порівняння. У мікроконтролерах STM32 маємо можливість задати число, до якого рахуватиме таймер, в звичайному режимі.

У мікроконтролері STM32F072C8T6 є 9 таймерів (див. таблицю 7 «Timer feature comparison» в технічному описі [5]). Оскільки нам не потрібно генерувати сигнал з таймера на вихід мікроконтролера, можна не використовувати таймери загального призначення або таймер розширеного керування, а вибрати базовий таймер TIM6 або TIM7. Це 16-розрядні таймери з 16-розрядним передподільником. Оберемо таймер TIM6.

Розрахуємо коефіцієнт передподільника та «стелю» лічби таймера, при досягненні якої він переповнюватиметься (задається в регістрі TIM6_ARR). Задамось частотою оновлення всього динамічного світлодіодного дисплею 100 Гц. Оскільки дисплей трьохрозрядний, частота перемикання розрядів повинна бути 300 Гц. Розрахуємо, скільки тактів системного тактового сигналу 48 МГц відповідає періоду перемикання розрядів індикатора. 48000000 / 300 = 160000. Отримане

число займає більше, ніж 16 двійкових розрядів (216=65536), а отже не може бути записане в регістр автозавантаження таймера TIM6_ARR. Поділимо на 4 тактовий сигнал на вході таймера за допомогою передподільника. Тоді в регістр TIM6_ARR потрібно записати число 160000 / 4 = 40000. Згідно п. 21.4.7 технічного опису [5] частота на виході передподільника f = fCK_PSC / (PSC[15:0] + 1). Тому в регістр TIM6_PSC слід записати значення 4 – 1 = 3, щоб отримати ділення частоти на 4. Задамо відповідні параметри в ioc-файлі (рис. 21).

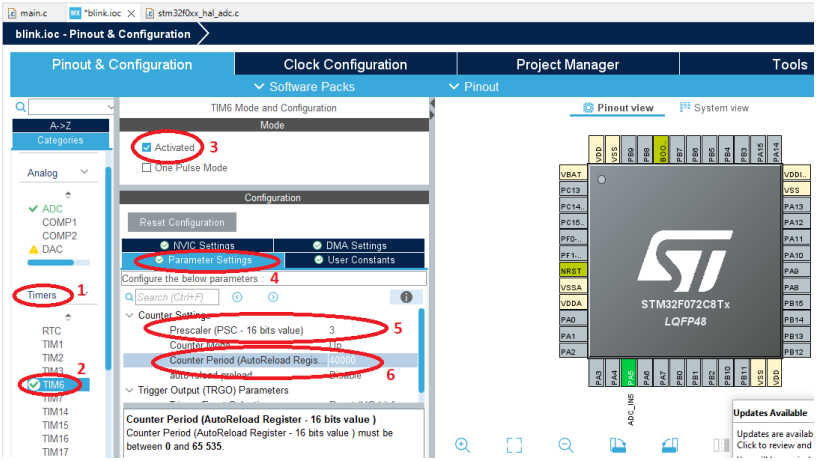

Рис. 21. Налаштування таймера TIM6

Переривання від таймера TIM6 можна дозволити на вкладці «NVIC Settings», відмітивши чекбокс «TIM6 global and DAC channel underrun error interrupts». Тепер кожну 1/300 секунди викликатиметься функція void TIM6_DAC_IRQHandler(void) з файлу stm32f0xx_it.c (файл для розміщення обробників переривань). У цій функції бачимо єдиний рядок – виклик функції

HAL_TIM_IRQHandler(&htim6), оскільки переривання від цифро-аналогового перетворювача (DAC) не дозволені. Функція HAL_TIM_IRQHandler() перевіряє, що спричинило переривання (подія захоплення чи порівняння на каналі 1, 2…, подія оновлення/переповнення, інші події) й викликає відповідну функцію-колбек.

У випадку переповнення генерується подія оновлення й викличеться колбек void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef *htim). Саме цю функцію повинен реалізувати користувач у своєму проекті, щоб виконати певні дії при переповненні таймера.

Збергіаємо іос-файл і генеруємо оновлений програмний код. Тепер у проекті повинна з'явитись функція ініціалізації таймера TIM6.

Таймер функцією MX_TIM6_Init() буде налаштовано, але не запущено. Запустити таймер потрібно викликом функції HAL_TIM_Base_Start_IT(&htim6), який слід розмістити до початку головного циклу. Функція запустить базовий таймер, на який вказує хендл htim6 з генеруванням переривань.

2. Налаштувати на вихід контакт PA8 – вихід регулятора, до якого підключити світлодіод. В іос-файлі налаштувати на вихід (GPIO_Output) контакти світлодіодного динамічного індикатора, підключеного до мікроконтролера, вказані в табл. 2.

Таблиця 2

Контакт індикатора	Контакт мікроконтролера
DIG1	PB4
DIG2	PB3
DIG3	PB0

A	PF0
B	PA15
C	PA9
D	PB9
E	PB8
F	PF1
G	PB2
DP	PA10

В результаті зроблених налаштуваннь отримуємо наступні використані входи-виходи мікроконтролера (рис. 22):

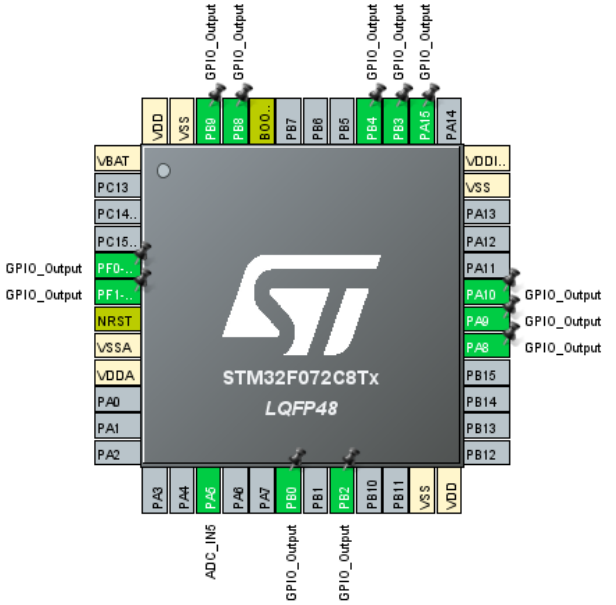

Рис. 22. Контакти мікроконтролера, задіяні в лабораторній роботі

3. Портувати програму з лабораторної роботи 4, яка реалізує двопозиційний регулятор температури з індикацією, на ARM-мікроконтролер. До аналогового входу мікроконтролера подаватиметься напруга з подільника напруги, утвореного термістором 10 кОм, B3950 та постійним резистором 7,5 кОм.

Очікувано на платі з іншим мікроконтролером зміняться назви й номери портів вводу-виводу, замість прямого запису в вихідні регістри портів використовуватимуться HAL-функції. Також зміняться частини програми, що стосуються налаштування таймера, АЦП, запуску і зчитування результату АЦП. Водночас функція розрахунку температури, позиційне регулювання, робиття на розряди для вивденення на дисплей можуть бути перенесені практично без змін.

Після виконання попередніх пунктів маємо налаштовану архітектурно-специфічну частину програми: вже виконано налаштування (функція MX_TIM6_Init()) і запуск (функція HAL_TIM_Base_Start_IT()) таймера TIM6, налаштовано, запущено й отримано результат АЦП (функції MX_ADC_Init(), HAL_ADC_Start(), HAL_ADC_PollForConversion(), HAL_ADC_GetValue()), налаштовано порти вводу-виводу загального призначення (функція MX_GPIO_Init()). Залишається перенести в STM32CubeIDE функцію розрахунку температури calc_temperature(), функцію засвічування цифри showDigit(), помістити в обробник переривання по переповненню таймера TIM6 тіло функції ISR(TIMER1_COMPA_vect), перенести тіло основного циклу програми з реалізацією опитування АЦП, розрахунком температури, двопозиційним регулюванням й розбиттям температури на розряди для індикації.

Оскільки в STM32F072C8T6 один адресний простір для команд і даних, глобальні константи зберігаються компілятором в flash-пам'яті програм і звертання до них здійснюється так само, як до даних в оперативній пам'яті (різниця лише в адресі), відпадає потреба в функції зчитування даних в flash-пам'яті програм.

Дії з перемикання розрядів світлодіодного динамічного семисегментного індикатора переносимо в функцію *void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)*.

Оскільки функція обробки переривання реалізована у іншому файлі, для звертання до змінних і функцій з файлу main.c потрібно використати ключове слово extern, яке вказуватиме, що це функція чи змінна з іншого файлу, відповідно пам'ять під змінну вже виділена при компіляції іншого файлу (main.c). В файлі stm32f0xx_it.c додамо в секцію глобальних змінних:

```
/* Private variables -------*/
/* USER CODE BEGIN PV */
uint8_t digit=1;
extern uint8_t disp[3];
/* USER CODE END PV */
```

Оскільки змінна digit використовується лише в цьому файлі, проголошення й виділення пам'яті здійснимо тут, а масив disp[3] заповнюється значеннями в main.c, тому в stm32f0xx_it.c залишимо вказівку компілятору, що цей масив із зовнішнього файлу (extern), а в файлі main.c у секцію /* USER CODE BEGIN PV */ … /* USER CODE END PV */ додамо власне проголошення масиву:

```
uint8_t disp[3]={1,2,3};
```

Перейдемо до переносу тіла функції main(). Для полегшення налагодження програми замість проголошення змінної temperature як локальної в функції main(), проголосимо її глобально, адже значення глобальних змінних можна легко перевірити в процесі налагодження. Додамо в секцію /* USER CODE BEGIN PV */ … /* USER CODE END PV */ рядок

```
int16_t temperature=0;
```

При реалізації кількаразового послідовного опитування для усереднення результатів і зменшення впливу шумів розрядність АЦП і кількість повторних опитувань визначають значення в таблиці відповідності «сума АЦП – температура», а відповідно й в масиві termo_table[ ]. При 10-бітному АЦП і 32 (25) ітераціях циклу опитування сума результатів АЦП 15-бітна. Якщо стоїть мета не перераховувати заново таблицю, то 15-бітну суму 12-бітних результатів АЦП можна отримати, якщо підсумовувати не 32 послідовні опитування, а 8 (23). Якщо кількість опитувань треба зберегти на рівні 32, то доведеться збільшити розрядність змінної для збереження суми, адже 21225=217. В такому випадку змінимо тип змінної temperature на int32_t й замінимо значення в масиві termo_table[ ] на наступні:

```
      const       temperature_table_entry_type
termo_table[] = {
         124597,   123767,   122853,   121851,
120755,   119561,   118264,   116862,   115350,
113728,   111993,   110146,   108187,   106117,
103941,  101661,  99284,  96816,  94265,  91639,
88949,  86203,  83414,  80593,  77750,  74898,
72048,  69212,  66399,  63619,  60883,  58197,
55570,  53008,  50518,  48102,  45766,  43511,
```

```
41341, 39256, 37256, 35343, 33514, 31770,
30108, 28526, 27023, 25596, 24242, 22959,
21744, 20594, 19506, 18478, 17506, 16588,
15720, 14901, 14128, 13399, 12710, 12060,
11446, 10867, 10321, 9805, 9318, 8858, 8424,
8014, 7626, 7260,  6914, 6587, 6277, 5985};
```

Переносимо програмну реалізацію регулювання, змінивши прямий запис у вихідний регістр порта AVR-мікроконтролера викликом HAL-функції HAL_GPIO_WritePin().

```
    if (temperature<sp-gist/2)
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8,
GPIO_PIN_SET);
    if (temperature>sp+gist/2)
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8,
GPIO_PIN_RESET);
```

Програмний код розбиття на розряди й запису в масив для індикації переносимо без змін:

```
    disp[0]=temperature/100;
    disp[1]=temperature/10-disp[0]*10;
    disp[2]=temperature%10;
```

Залишається замість виклику функції програмної затримки _delay_ms(200) додати HAL_Delay(200).

4. Оформити звіт про виконання лабораторної роботи. Звіт повинен містити: назву та мету лабораторної роботи; тексти програм з коментарями; висновок про виконання роботи.

Контрольні запитання

1. В яких режимах може працювати модуль АЦП в STM32F072?

2. Яка функція використовується для зчитування результату аналогово-цифрового перетворення?

3. Яка функція очікує закінчення аналогово-цифрового перетворення?

4. Яким чином працює АЦП в режимі неперервного перетворення?

5. Чи можливо в програмі для мікроконтролера серії STM32F0 так само, як для AVR-мікроконтролера, змінювати стан цифрового виходу безпосереднім записом у регістр, а не викликом HAL-функції? Чому?

6. Чим програмно відрізняється організація обробки переривань мікроконтролерів серії STM32F0 з використанням HAL від ISR-стилю AVR-мікроконтролерів?

Лабораторна робота 10. Застосування прямого доступу до пам'яті для отримання результату аналогово-цифрового перетворення

Мета роботи: навчитися використовувати DMA для передачі даних між периферією та оперативною пам'яттю.

Теоретичні відомості

DMA, Direct Memory Access (прямий доступ до пам'яті) — окремий блок, підключений до різноманітної периферії та пам'яті, який може самостійно (без участі CPU) переміщати дані

- з регістрів периферії в оперативну пам'ять і назад,
- з периферії в периферію,
- з пам'яті в пам'ять.

Розглянемо передачу даних з периферії в оперативну пам'ять на прикладі зчитування результату декількох аналогово-цифрових перетворень у масив.

Запуск модуля АЦП з передачею результату за допомогою DMA здійснюється функцією

HAL_ADC_Start_DMA*(ADC_HandleTypeDef * hadc, uint32_t * pData, uint32_t Length),*

де *pData – вказівник на буфер даних,*

Length – кількість даних.

Перед викликом функції повинен існувати масив *pData*, проголошений з ключовим словом *volatile* (без нього компілятор вважатиме вміст масиву незмінним, оскільки зміна даних в масиві відбувається без виконання обчислювальним ядром інструкцій запису в масив). Розмір масиву підбирається так, щоб вмістити потрібну кількість результатів опитування заданої кількості каналів. Наприклад, якщо потрібно опитати 3 канали по одному разу кожен, достатньо 3 елементів у масиві, а якщо 2 канали опитуються 16 раз підряд, то варто передбачити 32 елементи.

Також має бути дозволене переривання від DMA по закінченню передачі даних. Саме воно сигналізуватиме про отримання потрібної кількості результатів АЦП у масив і можливість перейти до їх обробки.

В нормальному режимі DMA закінчить роботу з модулем АЦП після запису в масив останнього результату. В циклічному (кільцевому) режмі DMA знову запустить АЦП і в масиві значення будуть перезаписані новими результатами аналогово-цифрового перетворення.

Для запису результату 12-бітного АЦП достатньо 16-бітної беззнакової цілочисельної змінної uint16_t, в той же час функція вимагає вказівник uint32_t*, тому, наприклад, для масиву *uint16_t buffer[32]* функція матиме вигляд

HAL_ADC_Start_DMA(&hadc1, (uint32_t)buffer, 32);*

План роботи

1. Налаштувати DMA для роботи в нормальному режимі.
2. Налаштувати DMA для роботи в циклічному режимі.

Порядок виконання роботи

1. Відкрити проект з попередньої лабораторної

роботи. В STM32CubeIDE відкрити ioc-файл і доналаштувати АЦП для використання DMA в нормальному режимі (рис. 23). Для цього обрати в лівій частині групу Analog, вибрати ADC, перейти на вкладку «DMA Settings», натиснути кнопку «Add», в списку обрати ініціатора DMA-запиту «ADC», залишити напрямок «з периферії в пам'ять», задати нормальний режим DMA, інкремент адреси пам'яті (результат АЦП зчитується з регістра результату, що має сталу адресу, а записується в масив у пам'яті, кожен наступний елемент в якому має більшу адресу), розмір даних – півслова (16 біт).

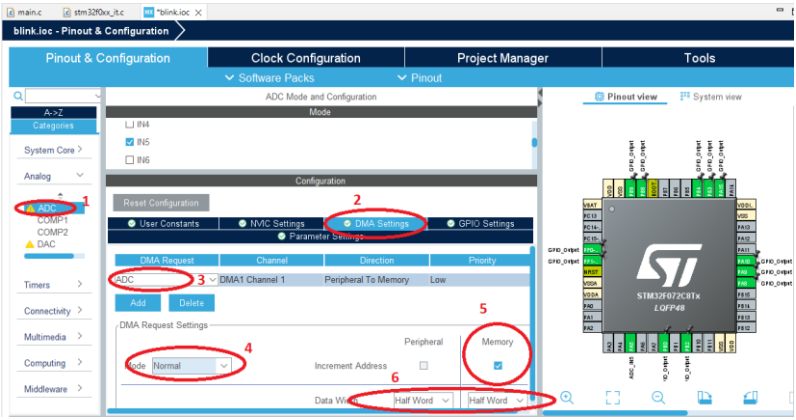

Рис. 23. Налаштування модуля АЦП для передачі результатів через DMA

Зберегти зміни й згенерувати оновлений програмний код проекту. В main.c повинна додатись функція ініціалізації DMA. В файлі stm32f0xx_it.c з'явиться обробник переривання DMA1_Channel1_IRQHandler(), який викличе функцію XferCpltCallback(DMA_HandleTypeDef *hdma) по закінченню передачі. Відповідно в цю функцію перенесемо обчислення суми:

```
    void    XferCpltCallback(DMA_HandleTypeDef
*hdma)
    {
     adc_res=0;
     for (int i=0; i<32; i++) {
                adc_res+=buffer[i];
       }
    }
```

Альтернативою є розміщення коду одразу в обробнику переривання між /* USER CODE BEGIN DMA1_Channel1_IRQn 1 */ … /* USER CODE END DMA1_Channel1_IRQn 1 */ (деякі версії HAL не викликають XferCpltCallback). Тоді його вигляд стане наступним:

```
    void DMA1_Channel1_IRQHandler(void)
    {
      /* USER CODE BEGIN DMA1_Channel1_IRQn 0
*/

      /* USER CODE END DMA1_Channel1_IRQn 0 */
      HAL_DMA_IRQHandler(&hdma_adc);
      /* USER CODE BEGIN DMA1_Channel1_IRQn 1
*/
      adc_res=0;
          for (int i=0; i<32; i++) {
                adc_res+=buffer[i];
            }
      /* USER CODE END DMA1_Channel1_IRQn 1 */
    }
```

Таким чином після закінчення обробки переривання по завершенню передачі DMA в змінній adc_res буде сума за 32 послідовні опитування аналогово-цифрового перетворювача.

Змінна adc_res і масив для збереження результатів buffer[32] проголошені в main.c, проте звертання до них здійснюється з обробника переривань з файлу stm32f0xx_it.c, тому в stm32f0xx_it.c їх потрібно

проголосити з ключовим словом extern. Секція проголошення глобальних змінних матиме вигляд

```
    /* Private variables ---------------------
-------------------------*/
    /* USER CODE BEGIN PV */
    uint8_t digit=1;
    extern uint8_t disp[3];
    extern volatile uint16_t buffer[32];
    extern volatile uint32_t adc_res;
    /* USER CODE END PV */
```

2. Модифікувати main.c з врахуванням того, що потреби 32 рази зчитувати результат АЦП вже немає, але потрібно синхронізувати початок обчислення темпертаури з готовністю нового значення суми за 32 опитування АЦП. Оскільки в нормальному режимі DMA закінчує роботу після передачі останнього результату АЦП, його потрібно запускати на початку кожної ітерації основного циклу while(1) викликом функції HAL_ADC_Start_DMA(&hadc, (uint32_t*)buffer, 32). Однократний запуск АЦП в режимі неперервного перетворення HAL_ADC_Start(&hadc) перед початком нескінченого циклу відповідно слід видалити або закоментувати.

Здачу синхронізації обчислень в обробнику переривання та в основному циклі класично можна вирішити проголошенням змінної-прапорця, яка встановлюватиметься з обробника переривання при готовності нового значення суми АЦП і слугуватиме умовою початку обчислення температури, регулювання та отримання масиву цифр для індикації. Коли свіжа сума результатів АЦП вже оброблена у основному циклі, прапорець скидається в нуль. Іншим варіантом може бути збереження попереднього значення змінної adc_res в змінній adc_res_old і порівняння на початку основного циклу цих змінних. Якщо в змінну adc_res у обробнику переривань буде записано нове значення суми результатів

АЦП, в основному циклі умова if(adc_res!=adc_res_old) поверне істину і запуститься обчислення температури, регулювання та отримання масиву цифр для індикації. Останній варіант дозволяє не виконувати повторні обчислення температури і регулювання, якщо нове значення суми результатів АЦП співпадає з попереднім.

Отримана програма після виклику функції HAL_ADC_Start_DMA() звільняється від потреби очікувати результат кожного перетворення, цей процесорний час може бути використаний для інших цілей. Готові результати АЦП зчитаються у обробнику переривання, буде обчислена сума, та якщо ця сума відрізнятиметься від попередньої, запуститься обчислення температури, позиційне регулювання та розбиття нового значення температури на розряди для індикації.

3. Скомпілювати проект, завантажити прошивку в мікроконтролер і налагодити програму.

Нами отримано більш ефективний код, який звільняє обчислювальне ядро мікроконтролера від потреби чекати результат АЦП і зчитувати його, але залишається потреба АЦП з DMA на початку кожного робочого циклу.

4. Змінити режим роботи DMA на кільцевий (рис. 24). Для цього обрати в лівій частині групу Analog, вибрати ADC, перейти на вкладку «DMA Settings», в списку обрати ініціатора DMA-запиту «ADC», залишити напрямок «з периферії в пам’ять», задати кільцевий режим DMA, інкремент адреси пам’яті (результат АЦП зчитується з регістра результату, що має сталу адресу, а записується в масив у пам’яті, кожен наступний елемент в якому має більшу адресу), розмір даних – півслова (16 біт). Перейти на вкладку «Parameter Settings» і дозволити неперервні запити DMA, задавши «DMA Continuous Requests» значення «Enabled». Зберегти зміни й згенерувати оновлений програмний код проекту.

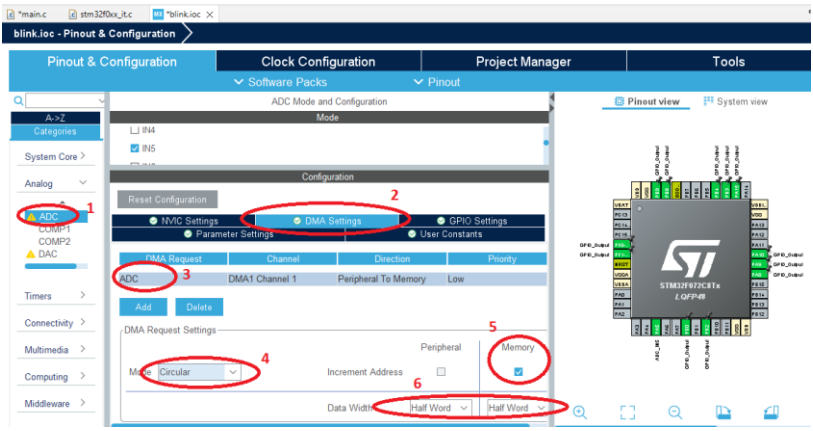

Рис. 24. Налаштування модуля АЦП для кільцевої передачі результатів через DMA

Тепер запущений одного разу АЦП з DMA буде працювати постійно. Коли останній результат АЦП із заданої кількості буде записано в масив у пам'яті, викличеться відповідний обробник переривання і обробить отриманий масив, а АЦП продовжить роботу, блок DMA записуватиме нові результати аналогово-цифрового перетворення спочатку виділеної під масив області оперативної пам'яті. Попередні результати АЦП у масиві буде перезаписано. Коли останній результат АЦП буде записано в кінець масиву в пам'яті, цикл повториться знову, DMA буде «по колу» записувати результати АЦП у масив у оперативній пам'яті.

З вищенаведеної причини виклик функції HAL_ADC_Start_DMA(&hadc, (uint32_t*)buffer, 8) можна винести з основного циклу й розмістити перед ним.

5. Скомпілювати проект, прошити мікроконтролер і налагодити програму. Тепер мікроконтролер виконує тіло основного циклу лише при істинності умови (adc_res!=adc_res_old), вільний процесорний час може використовуватись для виконання інших задач.

6. Оформити звіт про виконання лабораторної роботи. Звіт повинен містити: назву та мету лабораторної роботи; тексти програм з коментарями; висновок про виконання роботи.

Контрольні запитання

1. В яких режимах може працювати контролер DMA в STM32F072?

2. У яких випадках DMA може генерувати переривання?

3. У яких випадках доцільно використовувати контролер DMA?

4. Яка функція запускає модуль АЦП з передачею результату за допомогою DMA?

5. Який спосіб отримання результатів АЦП характеризується мінімальними затратами процесорного часу на зчитування 64 результатів?

Лабораторна робота 11. Використання ESP8266 для керування сервоприводом з веб-сторінки

Мета роботи: навчитися використовувати середовище Arduino IDE для програмування мікроконтролерів, що не підтримуються офіційно.

Теоретичні відомості

Середовище розробки Arduino IDE використовує проект Wiring. Для програмування плат з різними архітектурами мікроконтролерів потрібні різні компілятори й низькорівневі бібліотеки, тому для можливості програмування групи плат з однаковою архітектурою мікроконтролера потрібно встановити відповідну платформу (ядро, пакет плат).

“З коробки” доступна лише ядро/платформа з офіційними платами Arduino з мікроконтролерами AVR-архітектури. Середовище розробки Arduino IDE дозволяє

підключати додаткові плати, що не підтримуються офіційно, з використанням посилань, вказаних у налаштуванні «URL менеджерів додаткових плат». В результаті в Arduino IDE можна буде програмувати й інші мікропроцесорні плати, файли підтримки для яких автори зробили публічно доступними (рис. 25). Використання цієї можливості дозволяє розробляти прототипи мікропроцесорних систем на значній кількості мікроконтролерів, використовуючи звичний синтаксис і середовище розробки, а також велику кількість бібліотек, розроблених для Arduino.

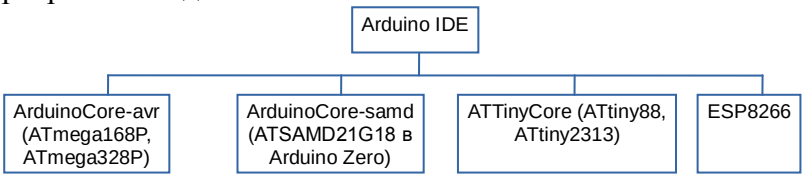

Рис. 25. Підтримка різних мікроконтролерів в Arduino IDE

Проте слід враховувати, що значна частка бібліотек використовує особливості мікроконтролерів з офіційних плат Arduino (таймери, модуль USART, переривання, регістри периферії, функції nop, sleep, sei тощо), а не лише високорівневі функції Arduino. Це виливається в неможливість скомпілювати програму з цими бібліотеками або некоректну її роботу на неофіційних платах, в результаті затрачається час на адаптацію бібліотеки для такої плати або від її використання відмовляються.

План роботи

1. Налаштувати Arduino IDE для програмування ESP8266.
2. Запрограмувати ESP8266 для керування сервоприводом SG90.
3. Запрограмувати ESP8266 для керування сервоприводом з веб-сторінки.

Порядок виконання роботи

1. Додати в налаштуваннях Arduino IDE новий URL менеджера додаткових плат https://arduino.esp8266.com/stable/package_esp8266com_index.json. Відкрити менеджер плат та встановити нову платформу — ESP8266.

2. Використовуючи бібліотеку Servo для ESP8266, розробити програму, що послідовно змінює положення сервопривода SG90 на 0, 90, 180 градусів. Перевірити виконання повороту сервоприводом.

3. Використовуючи клас ESP8266WebServer та приклад Файл->Приклади->ESP8266WebServer-> AdvancedWebServer, розробити програму, що виводить користувачу веб-сторінку, де він може задати положення сервоприводу, та розбирає отриманий GET-запит на наявність параметра-кута повороту. Числове значення параметра *angle* повинно використовуватись як аргумент методу write() об'єкта Servo з попереднього завдання. Перевірити реакцію сервопривода на дії на веб-сторінці, за потреби налагодити програму. Для формування веб-сторінки клієнту можна використати клас `StreamString`:

```
      StreamString temp;
        temp.reserve(500);    // Preallocate a
large chunk to avoid memory fragmentation
        temp.printf("\
      <html>\
        <head>\
          <meta            http-equiv='refresh'
content='5'/>\
          <title>ESP8266 Demo</title>\
          <style>\
            body { background-color: #cccccc;
font-family: Arial, Helvetica, Sans-Serif;
Color: #000088; }\
          </style>\
        </head>\
```

```
        <body>\
          <h1>Hello from ESP8266!</h1>\
          <p>Uptime: %02d:%02d:%02d</p>\
          <form action='/Servo' method='get'>\
            <label for='servoValue'>Enter value
(0-180): </label>\
            <input type='text' id='servoValue'
name='servoValue' maxlength='3' pattern='[0-
9]{1,3}' required>\
            <input type='submit'
value='Servo'>\
          </form>\
        </body>\
      </html>",
                    hr, min % 60, sec % 60);
        server.send(200,  "text/html",
temp.c_str());
        digitalWrite(led, 0);
      }
```

Обробку GET-запиту можна здійснити, створивши інлайн-функцію

```
        server.on("/Servo", HTTP_GET, [](){
          if (server.hasArg("servoValue")) {
            int servoValue =
server.arg("servoValue").toInt();
            myservo.write(servoValue);
            server.send(200, "text/plain",
"Servo value set");
          } else {
            server.send(400, "text/plain",
"Missing servoValue parameter");
          }
        });
```

4. Оформити звіт про виконання лабораторної роботи. Звіт повинен містити: назву та мету лабораторної роботи; тексти програм з коментарями; висновок про виконання роботи.

Контрольні запитання

1. Чи можна програмувати офіційні плати Arduino без використання Arduino IDE? Чому?

2. Що визначає перелік плат/мікроконтролерів, що можуть бути запрограмовані в Arduino IDE на конкретному комп'ютері?

3. Які обмеження можливі при використанні бібліотек Arduino з платами, що не підтримуються офіційно?

Список рекомендованої літератури

1. Проектування мікропроцесорних систем керування : навчальний посібник / І. Р. Козбур, П. О. Марущак, В. Р. Медвідь, В. Б. Савків, В. П. Пісьціо. Тернопіл ь: Вид-во ТНТУ імені Івана Пулюя, 2022. 324 с.

2. Мікропроцесори та мікроконтролери : навч. посіб. / Д. Д. Татарчук, Ю. В. Діденко. Київ : КПІ ім. Ігоря Сікорського, 2020. 238 с.

3. ATmega328/P AVR MCU with picoPower Technology Data Sheet [Електронний ресурс] / Microchip Technology Inc, 2018. URL: http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega328_P%20AVR%20MCU%20with%20picoPower%20Technology%20Data%20Sheet%2040001984A.pdf .

4. Noviello Carmine. Mastering STM32. Leanpub, 2018. (release 0.26). 792 с.

5. RM0091 Reference manual [Електронний ресурс] / STMicroelectronics, 2017. URL: https://www.st.com/resource/en/reference_manual/dm00031936-stm32f0x1stm32f0x2stm32f0x8-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

6. Michael Margolis. Arduino Cookbook. O'Reilly Media, 2011. 662 с.

7. Evans B. Arduino programming notebook [Електронний ресурс]. First edition. 2007. URL: https://playground.arduino.cc/uploads/Main/arduino_notebook_v1-1.pdf.

Інформаційні ресурси

1. FreeRTOS - Market leading RTOS (Real Time Operating System) for embedded systems with Internet of Things extensions. URL: http://www.freertos.org/.

2. AVR Libc Home Page. URL: http://www.nongnu.org/avr-libc/.