

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
УКРАЇНСЬКА АКАДЕМІЯ ДРУКАРСТВА

Пушак А.С., Вістовський В.В.

ПРОГРАМУВАННЯ РІС-КОНТРОЛЕРІВ

Лабораторний практикум

ЛЬВІВ – 2021

УДК 004(0758)

П78

*Затверджено Вченою радою Української академії друкарства
(протокол № 2/710 від 26.11.2020 року)*

Рецензенти:

М.І. Верхола, доктор технічних наук, професор кафедри автоматизації та комп'ютеризованих технологій Української академії друкарства.

Т.М. Демків, доктор фіз.-мат. наук, професор кафедри загальної фізики Львівського національного університету імені Івана Франка

Пушак А.С., Вістовський В.В.

П78 Пушак А.С. Програмування PIC-контролерів : лабор. практикум. Львів:

УАД, 2021. 118 с.

ISBN 978-966-322-520-3

Розглядається архітектура 8-розрядних мікроконтролерів сімейства PIC-16 та їхнє програмне середовище MPLAB X IDE. Приведено структурні блок-схеми периферійних модулів мікроконтролера PIC16F876A та опис їхніх регістрів налаштування. Наведено алгоритми ініціалізації периферійних модулів мікроконтролера та приклади кодів програм роботи цих модулів. Розглядаються алгоритми реалізації прийому та передачі інформації мікроконтролером, а також алгоритми обробки як цифрових так і аналогових сигналів. Розглянуто середовище proteus для симуляції роботи електричних схем в реальному часі.

УДК 004(0758)

ISBN 978-966-322-520-3

© Пушак А.С., Вістовський В.В.

© Українська академія друкарства, 2021

ЗМІСТ

ВСТУП.....	5
ЛАБОРАТОРНА РОБОТА № 1 Програмне забезпечення для роботи з мікроконтролерами PIC.....	6
ЛАБОРАТОРНА РОБОТА № 2 Вивчення роботи портів у режимі виводу інформації.....	13
ЛАБОРАТОРНА РОБОТА № 3 Вивчення портів мікроконтролера у режимі цифрового вводу інформації. Кнопка.....	22
ЛАБОРАТОРНА РОБОТА № 4 Керування кроковим двигуном.....	30
ЛАБОРАТОРНА РОБОТА № 5 Семисегментний індикатор. Статична індикація.....	40
ЛАБОРАТОРНА РОБОТА № 6 Семисегментні індикатори. Динамічна індикація.....	46
ЛАБОРАТОРНА РОБОТА № 7 Робота з рідкокристалічним дисплеєм. Ініціалізація дисплея LCD1602A	52
ЛАБОРАТОРНА РОБОТА № 8 Вивчення алгоритму виведення слів, багаторозрядних чисел та нестандартних символів на дисплей LCD1602A	65
ЛАБОРАТОРНА РОБОТА № 9 Вивчення модуля timer0. Генератор імпульсів.....	70
ЛАБОРАТОРНА РОБОТА № 10 Вивчення модуля CCP. Подільник частоти імпульсів.....	84
ЛАБОРАТОРНА РОБОТА № 11 Вивчення модуля CCP. Цифровий компаратор. Частотна модуляція	93

ЛАБОРАТОРНА РОБОТА № 12 Вивчення модуля ССР. Широтно-імпульсна модуляція (ШІМ).....	98
ЛАБОРАТОРНА РОБОТА № 13 Вивчення модуля ADC. Вимірювання напруги з допомогою модуля аналого-цифрового перетворювача	105
ЛІТЕРАТУРА.....	116

ВСТУП

Сучасні електронні пристрої, які використовуються у різних галузях науки і техніки, здебільшого містять інтегральні схеми, що підвищує надійність приладу та знижує його вартість порівняно з використанням дискретних напівпровідникових елементів. Окрім цього зменшення розмірів і ваги пристрою розширює межі його практичного застосування. Різноманітність інтегральних схем на сьогодні випускаються такими компаніями як Microchip, On Semiconductor, Texas Instrument, Analog Device, Linear Technology та багато інші, які є лідерами на світовому ринку. Одними з інтегральних мікросхем, які здатні виконувати різноманітні функції є мікроконтролери. Сьогодні стрімко розвиваються інформаційні і комп'ютерні технології, що дозволяє створювати автоматизовані технологічні лінії, якими можна керувати з допомогою комп'ютера. Одним з вузлів, що забезпечує взаємозв'язок комп'ютера з периферійним пристроєм є мікроконтролер. Для використання мікроконтролера у вузлах електроніки необхідно написати програмний код для реалізації тієї чи іншої задачі. Програмне налаштування параметрів мікроконтролера дозволяє керувати функцією відповідного пристрою без зміни параметрів електричної схеми, що є важливим фактором надійності пристрою в цілому. Одним з можливих застосувань мікроконтролерів є автоматизація фізичного експерименту, що дозволяє проводити вимірювання фізичних величин і передавати дані на комп'ютер. Тому вивчення дисциплін пов'язаних з програмуванням контролерів є актуальним на сьогодні завданням для підготовки фахівців в галузі комп'ютерної інженерії.

ЛАБОРАТОРНА РОБОТА № 1

Програмне забезпечення для роботи з мікроконтролерами PIC

Мета роботи: Ознайомитись із середовищем MPLAB, створити проєкт для програмування мікроконтролера PIC16F876A, скласти схему для роботи мікроконтролера, вивчити особливості роботи програматора PICKit3.

Теоретичні відомості

Мікроконтролер (МК) – це функціонально завершена мікропроцесорна система, виготовлена на одній надвеликій інтегральній схемі (НВІС). Мікроконтролер містить у собі процесор, оперативний запам'ятовуючий пристрій (ОЗП), постійний запам'ятовуючий пристрій (ПЗП), порти вводу/виводу для підключення зовнішніх пристроїв, модулі вводу аналогового сигналу – амплітудно-цифрові аналізатори (АЦП), таймери, контролери переривання, контролери інтерфейсів і т.д. Найпростіший МК представляє собою ВІС (велику інтегральну схему) площею не більше 1 см² і всього з вісьмома виходами.

Контролери, як правило, створюються для вирішення якоїсь окремої задачі або групи близьких задач. Вони, зазвичай, не мають можливостей підключення додаткових вузлів і пристроїв, наприклад, великої пам'яті, засобів вводу/виводу. Їх системна шина часто недоступна користувачеві. Структура контролера проста і оптимізована під максимальну швидкодію. У більшості випадків програми, що виконуються, зберігаються в постійній пам'яті і не змінюються. Конструктивно контролери виготовляються в одноплатному варіанті.

Основним призначенням мікроконтролерів PIC, як впливає з аббревіатури PIC (Peripheral Interface Controller), є виконання інтерфейсних функцій. Цим пояснюються особливості їхньої архітектури – RISC.

RISC – система команд, що характеризується малим набором одноадресних інструкцій (33, 35 або 55), кожна з яких має довжину в одне слово (12, 14 або 16 біт) і більшість виконується за один машинний цикл. У системі команд відсутні складні арифметичні команди (множення, ділення), гранично скорочений набір умовних переходів;

- висока швидкість виконання команд: за тактової частоти 20 МГц час машинного циклу складає 200 нс (швидкодія дорівнює 5 MIPS – млн. операцій/сек);
- наявність потужних вихідних каскадів (до 24 мА) на лініях портів вводу/виводу;
- низька споживана потужність;
- орієнтація на цінову нішу гранично низької вартості, що визначає використання дешевих корпусів з малою кількістю виводів (8, 14, 18, 28), відмова від зовнішніх шин адреси і даних (окрім PIC17C4X), використання спрощеного механізму переривань і апаратного (програмно недоступного) стека.

Для написання програмного коду компанією Microchip розроблено середовище MPLAB X IDE (Integrated Development Environment) (інтегроване середовище розробки), а також програма PICkit3 для прошивки коду в пам'ять мікроконтролера.

Програматор PICkit3

Для прошивки програмного коду у мікроконтролер використовується програматор PICkit3, який через USB порт під'єднується до комп'ютера, а його виводи під'єднуються до мікроконтролера. Зовнішній вигляд і опис виводів програматора PICkit3 приведено на рисунку 1.1.

Біля виводів на корпусі є мітка у вигляді трикутника, яка вказує на початок нумерації виводів (рис. 1.1 б).

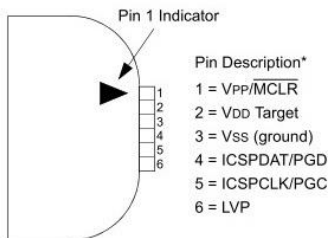
Опис виводів програматора PICkit3:

- 1 – Vpp (MCLR) (підключається через резистор 1-10 кОм до шини живлення +5 В);

- 2 – VDD (живлення +5 В);
- 3 – VSS (“земля”);
- 4 – ICSPDAT (PGD);
- 5 – ICSPCLK (PGC);
- 6 – LVP (PGM) не використовується.



а)

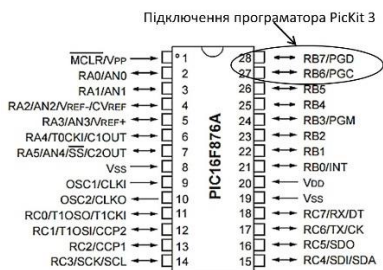


б)

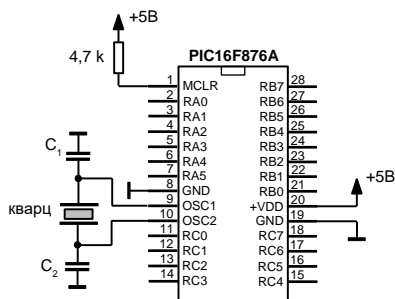
Рис. 1.1. а – зовнішній вигляд програматора PICkit3; б – нумерація та опис виводів програматора

Опис експерименту

У даній лабораторній роботі розглядається мікроконтролер PIC16F876A. З технічної документації дізнаємося опис виводів мікроконтролера та типову схему включення з використанням кварцового резонатора (рис. 1.2).



а)



б)

Рис. 1.2. а – опис виводів мікроконтролера PIC16F876A; б – типова схема включення мікроконтролера

Також з технічної документації мікроконтролера дізнаємося про виводи для підключення програматора. В таблиці 1.1 приведено відповідність під'єднання виводів програматора до мікроконтролера PIC16F876A

Таблиця 1.1

Номер виводу програматора PICKit3	Номер виводу мікроконтролера
1	1 (MCLR)
2	20 (VDD)
3	8, 19 (VSS)
4	28 (PGD)
5	27 (PGC)
6 (не використовується)	—

Згідно з технічними даними опису виводів мікроконтролера на рисунку 1.3 приведено схему під'єднання програматора до мікроконтролера.

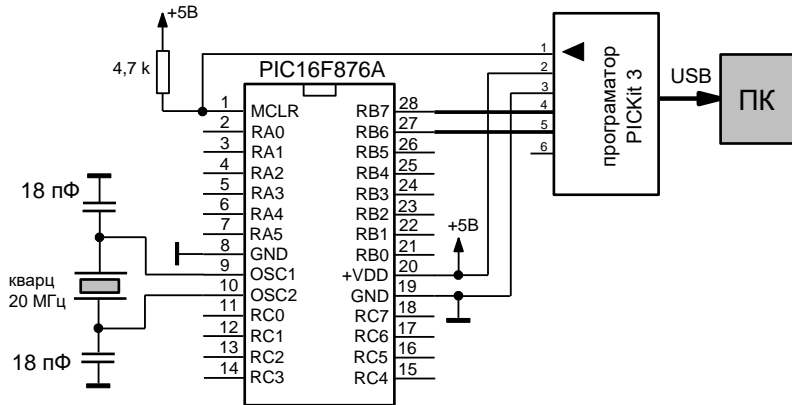


Рис. 1.3. Схема підключення програматора PICKit3 до мікроконтролерів PIC

Для прошивки мікроконтролера програмним кодом спочатку в середовищі необхідно створити проєкт в якому реалізується написання відповідного коду. Далі потрібно провести компіляцію

створеного коду з одержанням файлу прошивки (hex-файл). Далі з допомогою програми PICkit3 і програматора записують файл прошивки у мікроконтролер.

Хід роботи

1. Створення проєкту

1. Відкрийте середовище MPLAB.
2. Створіть новий проєкт, для цього у меню **file** вибираємо **New Project**, у вікні, яке появилось, вибираємо **Standalone Project** і натискаємо кнопку **Next**. У вікні, яке появилось, в полі **Family** вибираємо **Mind Range 8-Bit**, в полі **Device** вибираємо **PIC16F876A** і натискаємо кнопку **Next**. У наступному вікні вибираємо **Simulator** і натискаємо кнопку **Next**. Вибираємо компілятор XC8 і натискаємо кнопку **Next**. Далі в полі **Project Name** вказуємо ім'я свого проєкту, наприклад LAB_1, в полі **Project Location** вказуємо шлях до проєкту на жорсткому диску комп'ютера і натискаємо кнопку **Finish**.
3. У лівій верхній частині вікна середовища MPLAB появиться назва створеного проєкту, в якому створюємо файл для написання коду. Правою кнопкою миші на цьому проєкті вибираємо **new**, потім **main.c**. У вікні, яке появилось в полі **File Name**, вказуємо ім'я файлу і натискаємо кнопку **Finish**. Появиться поле з шаблоном для написання коду. В цьому полі, як вказує шаблон, задаємо вихідні дані проєкту (ім'я автора, дата створення, ім'я файлу).
4. У меню **Window** вибираємо **PIC Memory View**, а відтак **Configuration Bits**. В нижній частині вікна появиться поле для вибору бітів конфігурації. Вибираємо наступні біти:

FOSC = HS

WDTE = OFF

PWRTE = OFF

BOREN = OFF

LVP = OFF

```
CPD = OFF
WRT = OFF
CP = OFF
```

і натискаємо **Generate Source Code to Output**. Копіюємо згенеровані налаштування бітів конфігурації у поле для написання коду:

```
#pragma config FOSC = HS
#pragma config WDTE = OFF
#pragma config PWRTE = OFF
#pragma config BOREN = OFF
#pragma config LVP = OFF
#pragma config CPD = OFF
#pragma config WRT = OFF
#pragma config CP = OFF
```

5. Оголошуємо бібліотеку команд і оголошуємо тактову частоту кварцового генератора:

```
#include <xc.h>
#define _XTAL_FREQ 20000000
```

6. Створюємо головну функцію виконання алгоритму певної задачі:

```
void main (void)
{
    // команди та оператори користувача
}
```

Таким чином одержимо наступний вигляд коду:

```
//*****
// File:    LAB_1.c
// Project: LAB_1
// Author:  User_1.
// Created on 01 September 2020, 10:20
//
#pragma config FOSC = HS
#pragma config WDTE = OFF
```

```

#pragma config PWRTE = OFF
#pragma config BOREN = OFF
#pragma config LVP = OFF
#pragma config CPD = OFF
#pragma config WRT = OFF
#pragma config CP = OFF
#include <xc.h>
#define _XTAL_FREQ 20000000

//
void main (void)
{
    // команди та оператори користувача
}

//*****

```

2. Компіляція коду

В меню **Production** вибираємо **Build Main Project**. Якщо синтаксис коду правильний, то він буде успішно скомпільований, про що вказуватиме в нижній частині вікна повідомлення **BUILD SUCCESSFUL** із шляхом розміщення hex-файлу: **D:/LAB_1.X/dist/default/production/LAB_1X.production.hex**

3. Прошивання мікроконтролера

3.1. Під'єднуємо мікроконтролер до програматора, а останній через USB кабель до комп'ютера.

3.2. Відкриваємо програму **PICkit3**. У меню **File** вибираємо **Import Hex** і вибираємо скомпільований hex-файл прошивки. Натискаємо кнопку **Write** для виконання процесу прошивання. За умови коректного hex-файлу, процес прошивки відбудеться успішно, про що свідчитиме повідомлення: **Programming Successful**. Після цього мікроконтролер одразу виконуватиме запрограмовані команди.

Контрольні питання

1. Що означає PIC-контролери?
2. Яка архітектура мікроконтролерів PIC?
3. Що таке біти конфігурацій?
4. Опишіть типову схему включення мікроконтролера?
5. Для чого призначений програматор?
6. Опишіть виводи програматора PICkit3?

ЛАБОРАТОРНА РОБОТА № 2

Вивчення роботи портів у режимі виводу інформації

Мета роботи: Ознайомитись з портами мікроконтролера PIC16F876A. Ознайомитись із способами налаштування виводів портів. Вивчити налаштування порту А у режим цифрового вводу/виводу та аналогового входу.

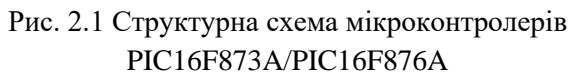
Теоретичні відомості

Структурну схему мікроконтролера PIC16F876A приведено на рисунку 2.1.

Для обміну інформацією з зовнішніми периферійними пристроями у мікроконтролерах слугують порти вводу/виводу. Як видно з рисунка, даний мікроконтролер має три порти вводу/виводу інформації: А, В, та С. За роботу цих портів відповідають регістри: PORTA, PORTB та PORTC відповідно. Порт А може бути налаштований в одному з двох режимів:

1. **режим цифрового порту вводу/виводу.** В цьому режимі, мікроконтролер може приймати або видавати інформацію у вигляді певного рівня напруги на виводах порту. Оскільки напруга живлення мікроконтролера 5 В, то рівень напруги приблизно 5 В відповідає логічній “1”, а рівень напруги приблизно 0 В відповідає логічному “0”.

Як видно з рисунка 2.1 порт А для мікроконтролера PIC16F876А є 6-розрядний, тобто має 6 виводів. Порти В і С є 8-розрядними.



З роботою порту А пов'язані такі регістри мікроконтролера: TRISA, ADCON1. У таблиці 2.2 приведено позначення бітів цих регістрів.

Регістр TRISA слугує для налаштування порту А на прийом або виведення інформації у цифровому режимі.

Регістр ADCON1 слугує для налаштування виводів порту А у режим цифрового вводу/виводу або аналогового входу.

Таблиця 2.1.

Позначення виводу	№ біта	Опис виводу порту А
RA0/AN0	0	цифровий ввід/вивід або аналогових вхід
RA1/AN1	1	цифровий ввід/вивід або аналогових вхід
RA2/AN2	2	цифровий ввід/вивід або аналогових вхід
RA3/AN3	3	цифровий ввід/вивід або аналогових вхід
RA4/T0CKI	4	використовується для модуля timer0
RA5/AN5	5	цифровий ввід/вивід або аналогових вхід

Таблиця 2.2.

регiстр	Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0
PORTA	-	-	RA5	-	RA3	RA2	RA1	RA0
TRISA	-	-	Регістр напряму даних порту А					
ADCON1	ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0

В таблиці 2.3 приведено режими робіт виводів порту А залежно від значення бітів PCFG3, PCFG2, PCFG1, PCFG0 регістра ADCON1.

Таблиця 2.3

PCFG3: PCFG0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0
0000	A	A	A	A	A
0001	A	V_{REF}^{+}	A	A	A
0010	A	A	A	A	A
0011	A	V_{REF}^{+}	A	A	A
0100	D	A	D	A	A

0101	D	V_{REF}^{+}	D	A	A
011x	D	D	D	D	D
1000	A	V_{REF}^{+}	V_{REF}^{-}	A	A
1001	A	A	A	A	A
1010	A	V_{REF}^{+}	A	A	A
1011	A	V_{REF}^{+}	V_{REF}^{-}	A	A
1100	A	V_{REF}^{+}	V_{REF}^{-}	A	A
1101	D	V_{REF}^{+}	V_{REF}^{-}	A	A
1110	D	D	D	D	A
1111	D	V_{REF}^{+}	V_{REF}^{-}	D	A

A – режим аналогового входу;

D – режим цифрового вводу/виводу;

V_{REF}^{+} – вхід опорної напруги для роботи АЦП (+5 В);

V_{REF}^{-} – вхід опорної напруги для роботи АЦП (“земля”).

Режим роботи виводів порту А налаштовують програмно так: у тілі головної функції середовища MPLAB для програмування мікроконтролерів записують стрічку:

TRISA = 0b00001110;

цей запис у середовищі MPLAB мовою С означає, що виводи RA1, RA2, RA3 налаштовані на ввід інформації, всі інші на вивід. Тобто значенню “1” відповідного біта у регістрі TRISA відповідає налаштування відповідного виводу порту А на ввід (прийм) цифрової інформації, а значенню “0” – виведення цифрової інформації.

Значення бітів у регістрі TRISA можна представити трьома способами:

1. Запис числа в регістр TRISA в двійковій формі.

Спочатку після запису “TRISA =” записують символи “0b”, що означає двійкову (binary) систему, після яких записують у двійковій

формі відповідне число (00001110), причому розрядність числа – 8 біт, оскільки регістр TRISA є 8-бітний. Розряди числа відповідають відповідним бітам регістра у зростаючій послідовності, починаючи з молодшого розряду (табл. 2.4).

Таблиця 2.4

RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0
0	0	0	0	1	1	1	0

2. Представлення числа в десятковій формі, наприклад:

`TRISA = 14;`

цей запис рівносильний запису

`TRISA = 0b00001110;`

3. Представлення числа в шістнадцятковій формі:

`TRISA = 0xE;`

символи “0x” після знаку “=” означають подання числа в шістнадцятковій системі, подальший символ “E” – це число 14 в шістнадцятковій системі.

Розглянутими способами ми налаштовуємо усі виводи порту A одночасно не залежно від того скільки виводів ми будемо використовувати. Є ще один спосіб, який дозволяє налаштовувати тільки один вивід порту A, наприклад, запис

`TRISAbits.TRISA2 = 1;`

або запис

`TRISA2 = 1;`

означає, що вивід RA2 порту A налаштовано на ввід цифрової інформації.

Розглянемо налаштування режимів роботи виводів порту A як цифрового (вводу/виводу) і аналогового (вводу). Для цього в регістр ADCON1 програмно заносимо відповідне значення бітів, наприклад:

`ADCON1 = 0b00000100;`

Як видно з таблиці 2.3 виводи RA0, RA1, RA3 є налаштовані як аналогові входи, а виводи RA2, RA5 як цифрові вводу/виводу.

Для налаштування режимів роботи портів В і С слугують регістри TRISB і TRISC відповідно. Опис бітів регістрів для портів В і С приведено в таблиці 2.5.

Таблиця 2.5.

регістр	Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0
PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
TRISB	Регістр напряму даних порту В (0 – вивід, 1 – ввід)							
PORTC	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0
TRISC	Регістр напряму роботи порту С (0 – вивід, 1 – ввід)							

Після налаштування режимів роботи виводів портів, проводиться виведення або прийом інформації.

Розглянемо процес виведення інформації в порт, наприклад порт В. Оскільки порт В є цифровим портом вводу/виводу, то для його налаштування використовується тільки один регістр TRISB. Налаштуємо порт В на вивід інформації:

TRISB = 0b00000000;

В даному випадку всі виводу порту будуть працювати на вивід цифрової інформації. Далі для того, щоб послати певну інформацію в порт В, тобто виставити на його виводах певні логічні рівні напруги +5 В або 0 В, що відповідають логічним “1” та “0”, потрібно в регістрі PORTB порту В виставити відповідні біти в “1” та “0”:

PORTB = 0b00001110;

У результаті на виводах RB1, RB2, RB3 порту В буде виставлено напругу високого логічного рівня (приблизно рівну напрузі живлення +5 В), а на інших виводах буде виставлено напругу, що відповідає низькому логічному рівню (приблизно 0 В). Розряди числа в двійковій формі відповідають відповідним бітам регістра PORTB у зростаючій послідовності, починаючи з молодшого розряду (табл. 2.6).

Таблиця 2.6

RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
0	0	0	0	1	1	1	0

Також можна виставляти певні логічні рівні напруги на окремих виводах портів, якщо інші виводи не використовуються або використовуються у режимі прийому інформації, наприклад:

```
RB1 = 1;
```

```
RB2 = 0;
```

```
RB3 = 1;
```

У результаті на виводах RB1; RB3 буде напруга високого логічного рівня, на виводі RB2 буде напруга низького логічного рівня. На інших виводах порту рівень напруги буде не визначений. Такий запис значно спрощує написання програмного коду. Також в середовищі MPLAB існує ще один спосіб задання значень окремих бітів порту, наприклад:

```
PORTBbits.RB0 = 1;
```

```
PORTCbits.RC2 = 0;
```

Такий запис рівносильний запису:

```
RB0 = 1;
```

```
RC2 = 0;
```

Також значення бітів регістра порту можна представляти числами у десятковій або шістнадцятковій формі:

```
PORTB = 14;
```

```
PORTB = 0xE;
```

Такі записи рівносильні запису:

```
PORTB = 0b00001110;
```

Опис експерименту

Схему для вивчення роботи портів мікроконтролера PIC16F876A приведено на рисунку 2.2. Для простоти схеми, стандартні елементи такі як кварцовий резонатор, конденсатори, лінії живлення на схемі не показано.

Резистори R служать для обмеження струму через світлодіоди, що запобігає виходу їх з ладу, а також запобігає виходу з ладу мікроконтролера, для якого вихідний струм не перевищує 30 мА. Значення резисторів R в даній схемі – 470 Ом.

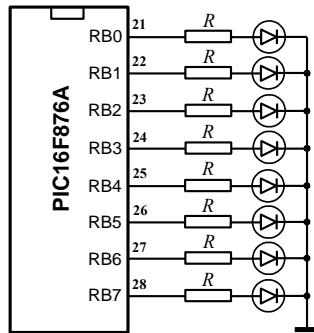


Рис. 2.2. Схема під'єднання світлодіодів до виводів порту мікроконтролера

Програмний код в середовищі MPLAB для вивчення роботи порту В у режимі виводу інформації має вигляд:

```

//*****
#pragma config FOSC = HS
#pragma config WDTE = OFF
#pragma config PWRTE = OFF
#pragma config BOREN = OFF
#pragma config LVP = OFF
#pragma config CPD = OFF
#pragma config WRT = OFF
#pragma config CP = OFF
#include <xc.h>
#define _XTAL_FREQ 20000000
void main (void)
{
    TRISB=0b00000000; //порту В налаштований на вивід
    while (1)
    {
        PORTB=14; // в порт В посилаємо число 14
    }
}
//*****

```

У даному програмному коді реалізується виведення числа 14 в порт В.

Частина програмного коду, який реалізує мигання світлодіоду під'єданого до виводу RB0, з певною частотою, має вигляд:

```
//*****  
while (1)  
{  
    RB0=1;  
    __delay_ms (500);  
    RB0=0;  
    __delay_ms (300);  
}  
//*****
```

Команда

```
__delay_ms (500);
```

виконує часову затримку 500 мс. Можна задавати затримку в мікросекундах, використовуючи команду:

```
__delay_us (400);
```

В даному прикладі реалізується затримка 400 мікросекунд.

Використовуючи різні часові затримки, можна генерувати цифрові імпульси різної скважності.

Хід роботи

1. У середовищі Proteus складіть схему, приведену на рисунку 2.2.
2. У середовищі MPLAB створіть проєкт і введіть приведенний вище код для виводу певної інформації в порт В.
3. Проведіть компіляцію коду та перевірити його роботу у середовищі Proteus.
4. Перевірте роботу коду на макетній платі, прошивши мікроконтролер з допомогою програматора PICkit3.

Завдання

1. Розробити алгоритм та написати програмний код для роботи

порту А у режимі цифрового вводу/виводу, та перевірити його роботу у середовищі Proteus використовуючи типову схему на рисунку 2.2.

2. Розробити код програми для мигання світлодіоду з частотою 1; 2; та 3 с.
3. Розробити алгоритм почергового короткочасного засвічування світлодіодів (“біжучий вогник”).
4. Перевірити роботу кодів пункту 2 і 3 у середовищі Proteus та на макетній платі.

Контрольні питання

1. Які є порти у мікроконтролері PIC16F876A?
2. Яка розрядність регістрів портів у мікроконтролері PIC16F876A?
3. Який порт мікроконтролера має аналоговий вхід?
4. Яку функцію виконує регістр TRISB?
5. Яку функцію виконує регістр ADCON1?

ЛАБОРАТОРНА РОБОТА № 3

Вивчення портів мікроконтролера у режимі цифрового вводу інформації. Кнопка

Мета роботи: Ознайомитись з портами мікроконтролера PIC16F876A. Налаштувати виводи портів на ввід інформації. Ознайомитись із схемами підключення кнопок до мікроконтролера. Вивчити алгоритм усунення брязкоту контактів кнопок.

Теоретичні відомості

Як відомо регістри TRISA, TRISB та TRISC портів мікроконтролера PIC16F876A задають напрям роботи портів, тобто налаштовують режим роботи портів на вивід або на ввід інформації. В режимі цифрового виводу ми програмно можемо виставляти певні

логічні рівні напруги на виводах портів. В режимі цифрового вводу ми програмно можемо зчитувати логічні рівні напруги, які є подані до виводів портів. Режим цифрового вводу використовується наприклад при роботі з кнопками або клавіатурою під'єднаною до мікроконтролера. В цьому випадку з допомогою кнопок ми подаємо певні рівні напруги на виводи портів, які несуть певну інформацію.

Для налаштування режиму роботи виводів порту (наприклад порту В) слугує регістр TRISB. Для налаштування виводів порту В у режим вводу потрібно в регістрі TRISB задати значення відповідних бітів як "1":

$$\text{TRISB} = 0b11111111;$$

В даному випадку всі виводи порту В налаштовані в режим вводу.

Розглянемо схеми під'єднання кнопок до мікроконтролера. На рисунку 3.1 приведено два способи підключення кнопок до виводів портів мікроконтролера (наприклад до виводу RB5 порту В).

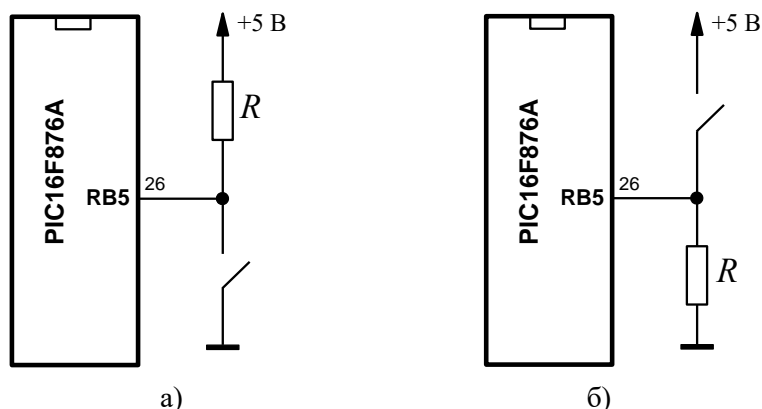


Рис. 3.1. Способи під'єднання кнопки до мікроконтролера.
Значення підтягуючого резистора R може бути в межах 1-10 кОм
залежно від схеми

Нехай маємо кнопку, яка замикає коло при натисканні. Коли кнопка не натиснута, на вивід мікроконтролера подається високий рівень напруги, що відповідає значенню логічної "1" (рис. 3.1 а). При натисканні кнопки відбувається замикання виводу порту на "землю",

що приводить до появи на цьому виводі нульового потенціалу, що відповідає значенню логічного “0”. Отже, при натисканні даної кнопки на виводі мікроконтролера відбувається зміна напруги з високого логічного рівня до низького логічного рівня.

Схема підключення кнопки, приведена на рисунку 3.1 б, протилежна за функціональністю до схеми (рис. 3.1 а), тобто, при натисканні кнопки на виводі мікроконтролера відбувається зміна напруги з низького логічного рівня до високого логічного рівня.

Схему під’єднання зовнішнього пристрою до мікроконтролера приведено на рисунку 3.2. Значення резистора R вибирають відповідно до вихідних електричних характеристик зовнішнього пристрою.

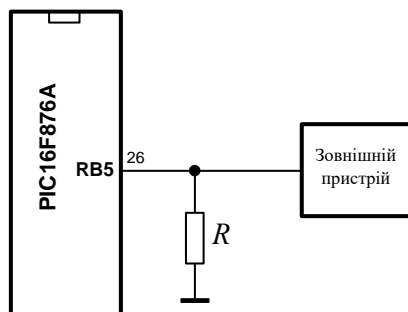


Рис. 3.2. Схема під’єднання зовнішнього пристрою до мікроконтролера

Опис експерименту

У даній роботі розглядається алгоритм мигання світлодіоду, підключеного до одного з виводів порту В, за умови натиснутої кнопки, під’єднаної до виводу RC4 (рис. 3.3). Кнопка така, що при натисканні її контакти замикаються.

Головна функція програмного коду, який реалізує мигання світлодіоду при натиснутій кнопці виглядає так:

```
//*****  
void main (void)  
{
```



```

TRISC=0b00010000; // вивід RC4 порту C налаштований на ввід
TRISB=0b00000000; // порт B налаштований на вивід
PORTB=0b00000000; // очищення регістра порту B
while (1)
{
    if (RC4==0) // умова натискання кнопки
    {
        RB1=1;
        __delay_ms (300);
        RB1=0;
        __delay_ms (300);
    }
}
}
//*****

```

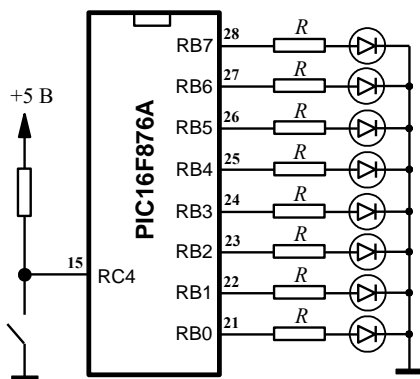


Рис. 3.3. Схема експериментальної плати для вивчення роботи кнопки

Вище приведений код даватиме коректну роботу мікроконтролера за умови ідеальної кнопки. Оскільки реальні кнопки не ідеальні, тобто в момент натискання може виникнути часте короткотривале замикання і розмикання контактів, так званий брязкіт контактів. На рисунку 3.4 приведено зміну рівня напруги в часі при натисканні ідеальної і реальної кнопок.

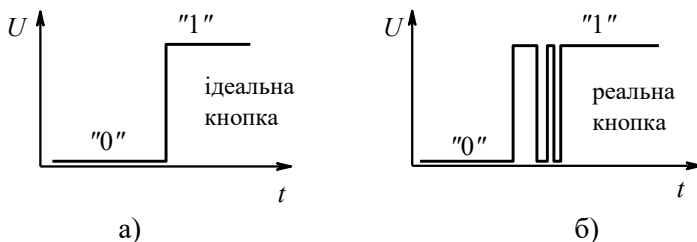


Рис. 3.4. Графік залежності зміни рівня напруги за натискання: а – ідеальної кнопки; б – реальної кнопки (наявний брязкіт контактів)

У вузлах цифрової схемотехніки явище брязкоту контактів неприпустиме, оскільки іншими вузлами електричної схеми такий брязкіт сприймається як багаторазове короточасне перемикання напруги, що приводить до некоректного сприймання цифрової інформації. Якщо кнопка під'єднана до мікроконтролера, то вплив ефекту брязкоту контактів кнопки можна усунути, зrealізувавши наступний алгоритм. Як тільки відбудеться перша зміна рівня напруги при натисканні кнопки, починається певна часова затримка, протягом якої брязкіт минає і настає новий стабільний рівень напруги на виводі порту мікроконтролера, який можна зчитувати.

У програмному коді створимо певну функцію для опрацювання умови натискання кнопки. Завданням цієї функції є зробити певну часову затримку за умови першої зміни рівня напруги (умова натискання кнопки) перед зчитуванням рівня напруги на виводі мікроконтролера. Протягом цієї затримки можливий брязкіт контактів минає і настає новий стабільний рівень напруги, який тепер можна зчитувати. Таким чином мікроконтролер правильно сприйматиме зміну напруги на його виводі за умови натискання кнопки. Функція опрацювання умови натискання кнопки має вигляд:

//**

```
unsigned char knopka (void)
{
    unsigned char result=0; // введемо змінну result
    unsigned int k=5000, i=0; // локальні змінні
```

```

while (RC4==0)                                // натиснули кнопку
{
    if (i<k)
    {
        i=i+1;
    }
    else
    {
        result=1;
        break;
    }
}
return result;
}
//*****

```

Розглянемо роботу цієї функції. При натисканні кнопки програма попадає в тіло циклу з умовою натиснутої кнопки `while (RC4 == 0)`. В цьому циклі відбувається інкрементування певної локальної змінної ($i = i + 1$;) до заданої величини ($i < k$) за умови, що кнопка натиснута. Значення межі інкрементування k підбирається експериментально для кожного виду кнопок. Таке інкрементування триває певний час, за який брязкіт минає. Як тільки порушиться умова ($i < k$), програма попадає в блок `else` де змінній `result` присвоюється значення рівне одиниці і відбувається вихід із циклу (`break`;) а далі функція `кнопка` повертає значення змінної `result` (`return result`;) . Весь цей час кнопка була натиснута, функція повернула значення змінної `result` рівне одиниці.

Коли хід виконання програми є в циклі `while (RC4 == 0)` і раптом через брязкіт контактів порушується умова `(RC4 == 0)`, тоді цикл закінчується і функція `кнопка` повертає значення змінної `result` рівне нулю, як присвоєно по замовчуванні.

Код програми для реалізації мигання світлодіоду за умови натискання кнопки має вигляд:

```

//*****
unsigned int knopka (void)
{
    unsigned int result=0;
    unsigned int k=5000, i=0;
    while (RC4==0)           // натиснули кнопку
    {
        if (i<k)
        {
            i=i+1;
        }
        else
        {
            result=1;
            break;
        }
    }
    return result;
}

void main (void)
{
    TRISC=0b00010000;   // вивід RC4 порту C налаштований на ввід
    TRISB=0b00000000;   // порт B налаштований на вивід
    PORTB=0b00000000;   // очищення регістра порту B
    while (1)
    {
        if (knopka ())   // умова натискання кнопки
        {
            RB1=1;
            __delay_ms (300);
            RB1=0;
            __delay_ms (300);
        }
    }
}
//*****

```

Хід роботи

1. У середовищі Proteus складіть схему зображену на рис. 3.4.
2. У середовищі MPLAB створіть проєкт з кодом для мигання світлодіоду за умови натиснутої кнопки (використайте код з функцією опрацювання натискання кнопки).
3. Проведіть компіляцію коду та прошити одержаним hex-фалом мікроконтролер.
4. Експериментально підберіть оптимальне значення часової затримки для зручного натискання кнопки.

Контрольні питання

1. Як налаштувати виводи порту на ввід інформації?
2. Опишіть способи під'єднання кнопок до виводів мікроконтролера?
3. Що таке брязкіт контактів кнопок?
4. Як усунути вплив брязкоту контактів кнопок у цифрових вузлах?
5. Опишіть алгоритм опрацювання умови натискання кнопки?

ЛАБОРАТОРНА РОБОТА № 4

Керування кроковим двигуном

Мета роботи: Вивчити принцип роботи крокового двигуна. Ознайомитись із схемою підключення крокового двигуна до мікроконтролера. Дослідити частотну характеристику крокового двигуна.

Теоретичні відомості

Крокові двигуни постійного струму набули поширення у верстатах з числовим програмним управлінням і робототехніці. Основною відмінністю даного електромотора є можливість задавання кута повороту вала. Вал крокового електродвигуна не обертається тривалий час, а лише повертається на певний кут. Цим забезпечується точне позиціонування робочого елемента в просторі. Електроживлення такого двигуна дискретне, тобто здійснюються імпульсами. Ці імпульси і повертають вал на певний кут. Кожен такий поворот називається кроком, звідси і пішла назва.

Кроковий двигун складається з статора і ротора. На роторі, зазвичай розташовані секції, набрані з листів електротехнічної сталі (“зубчаста” частина, рис. 4.1), а ті, в свою чергу, розділені постійними магнітами. На статорі розташовані обмотки у вигляді окремих котушок.

Принцип роботи крокового двигуна

Як працює кроковий електродвигун, можна розглянути на умовній моделі. В положенні 1 на обмотки А і В подається напруга певної полярності (рис. 4.2). У результаті в статорі утворюється електромагнітне поле. Оскільки різні магнітні полюси притягуються, ротор займе своє положення по осі магнітного поля. Крім того, магнітне поле мотора буде перешкоджати спробам зміни положення ротора ззовні. Магнітне поле статора забезпечуватиме утримання

ротора від зміни заданого положення (наприклад, при механічних навантаженнях на вал).

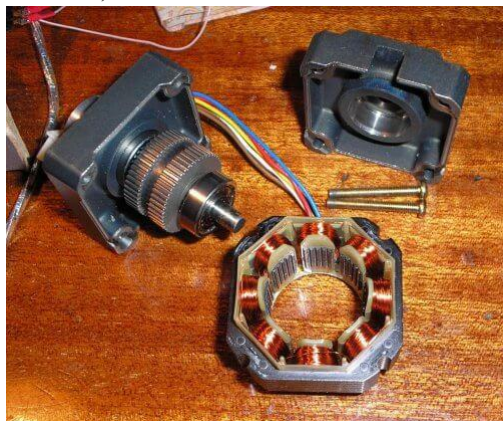


Рис. 4.1. Зовнішній вигляд елементів крокового двигуна

Якщо напругу тієї ж полярності подати на обмотки D і C, електромагнітне поле зміститься. Це змусить повернутися ротор з постійним магнітом в положення 2. У цьому випадку кут повороту дорівнює 90° . Цей кут і буде кроком повороту ротора.

Положення 3 досягається подачею напруги зворотної полярності на обмотки A і B. У цьому випадку електромагнітне поле стане протилежним положенню 1, ротор двигуна зміститься, і загальний кут буде 180° .

При подачі напруги зворотної полярності на обмотки D і C ротор повернеться на кут до 270° щодо початкової позиції. При підключенні позитивної напруги на обмотку A і негативної на B ротор займе початкове положення – закінчить оборот на 360° . Слід враховувати, що пересування ротора відбувається за найменшим шляхом, тобто з положення 1 в положення 4 за годинниковою стрілкою ротор повернеться тільки після проходження проміжних положень 2 і 3. При підключенні обмоток після положення 1 відразу в положення 4 ротор повернеться проти годинникової стрілки.

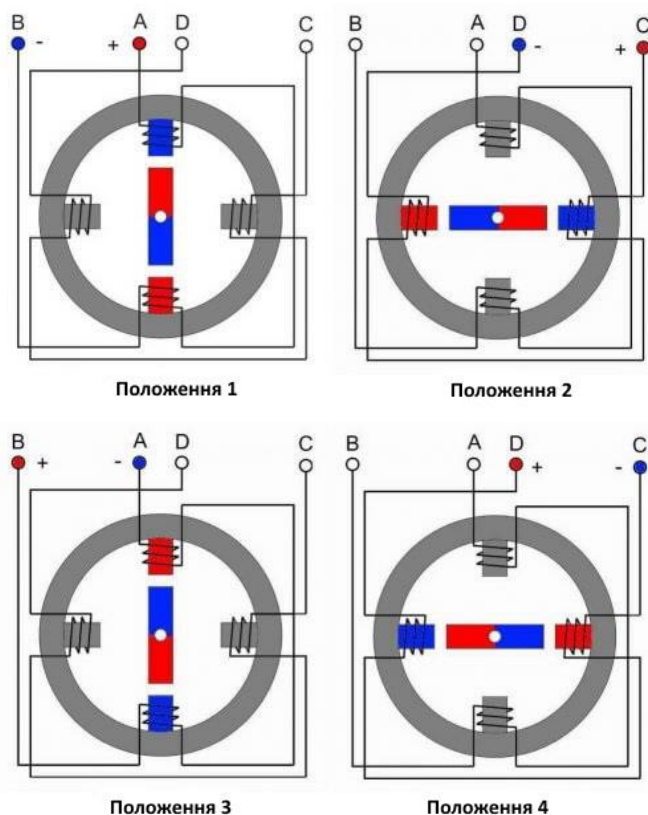


Рис. 4.2. Принцип роботи крокового двигуна

Види обмоток крокового двигуна

У крокових двигунах застосовуються біполярні і уніполярні обмотки. Принцип роботи, описаний вище, стосувався біполярного двигуна. Така конструкція передбачає використання різних фаз для живлення обмоток. Схема є складною і вимагає дорогих і потужних плат управління. Більш проста схема управління в уніполярних двигунах. У такій схемі обмотки з одного боку підключені до загальної шини живлення $+U$, а з другого – до окремих входів, на які по черзі подається сигнал “мінус” (рис. 4.3), і тим самим забезпечується обертання ротора. Також реалізується випадок, коли

одні виводи котушки з'єднані з шиною живлення $-U$ ("земля"), а до інших виводів по чергово подається живлення $+U$.

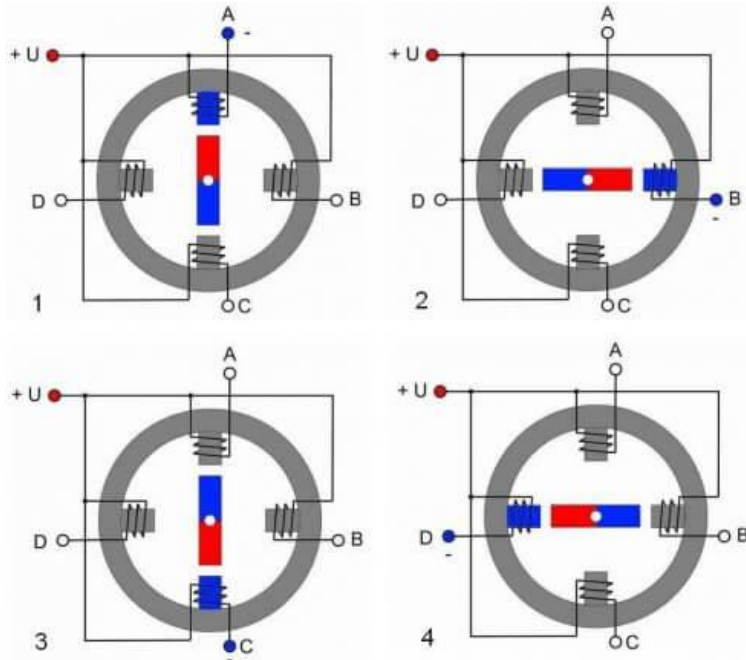


Рис. 4.3. Схема з'єднання обмоток уніполярного крокового двигуна

Для уніполярних крокових двигунів є декілька режимів роботи. Розглянемо два найбільш поширені режими.

Режим повного кроку

При подачі напруги на вивід обмотки А магніт ротора розташовується відповідно до магнітного поля, створюваного котушкою А (рис. 4.3, положення 1). Магнітне поле обмотки А тримає магніт ротора у фіксованому положенні певний проміжок часу, після чого живлення обмотки А вимикається, її магнітне поле при цьому зникає, і подається напруга на обмотку В. При цій зміні магнітного поля ротор повернеться в нове фіксоване положення в магнітному положенні обмотки В (рис. 4.3, положення 2). Такий поворот називається повним кроком двигуна, в даному випадку кут

повороту складає 90° . Після певного проміжку часу напруга живлення обмотки В вимикається, а вмикається на обмотці С, магнітне поле якої поверне ротор ще на кут 90° (рис. 4.3, положення 3). Знову через певний проміжок часу напруга живлення вимикається на обмотці С, а вмикається на обмотці D, магнітне поле якої поверне ротор на 90° (рис. 4.3, положення 4). Далі через певний проміжок часу напруга живлення вимикається на обмотці D, а вмикається на обмотці А, магнітне поле якої поверне ротор на 90° (рис. 4.3, положення 1). Таким чином ротор повернувся на 360° , зробивши 4 повних кроки.

На рисунку 4.4 приведено залежність від часу напруги на обмотках в режимі повного кроку.

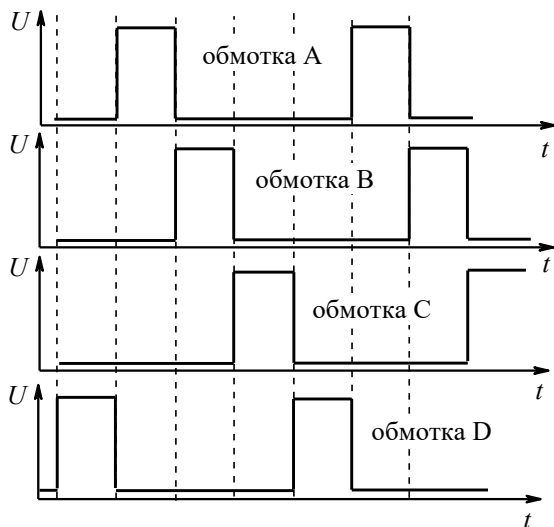


Рис. 4.4. Графік залежності напруги на обмотках крокового двигуна від часу у режимі повного кроку

Режим півкроку

У режимі півкроку є проміжки часу, коли напруга подається одночасно на дві обмотки. Розглянемо роботу двигуна в цьому режимі. Нехай в певний момент часу є подана напруга тільки на

обмотку А (рис. 4.5 а). У цьому випадку магніт ротора займе вертикальне положення, як показано на рисунку.

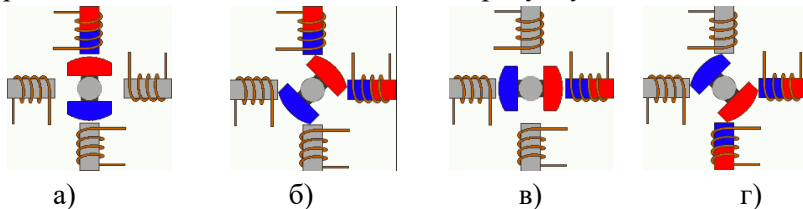


Рис. 4.5. Положення ротора крокового двигуна у режимі пів кроку: а – початкове положення; б – перший півкрок; в – другий півкрок; г – третій півкрок

Тепер, не вимикаючи напруги на обмотці А, вмикають напругу на обмотці В (рис. 4.5 б). У цьому випадку створюється магнітне поле двома обмотками і ротор займе проміжне положення між обмотками, тобто повернеться на кут 45° . Після певного проміжку часу напругу на обмотці А вимикають, а на обмотці В залишають (рис. 4.5 в), тоді магніт ротора займе горизонтальне положення, як показано на рисунку, тобто повернеться ще на 45° . Далі через певний проміжок часу, не вимикаючи напруги на обмотці В, вмикають напругу на обмотці С (рис. 4.5 г). Це приводить до повороту ротора ще на кут 45° , як показано на рисунку, а відтак напруга на обмотці В вимикається і ротор повернеться ще на 45° . І так далі до початкового положення ротора, в подальшому процес повторюється.

Графік залежності напруги на обмотках в режимі півкроку приведено на рисунку (рис. 4.6)

Опис експерименту

Схему експериментальної макетної плати підключення мікроконтролера до крокового двигуна приведено на рисунку 4.7.

Схема працює таким чином: мікроконтролер запрограмований так, що на виводах RB0-RB3 по чергово виставляють рівень напруги, що відповідає логічній “1”; ця напруга подається через резистори на бази транзисторів, які виконують роль силових ключів, тим самим

замикають коло живлення обмоток А, В, С, D відповідно (рис. 4.7). Необхідність використання силових ключів обумовлена тим, що

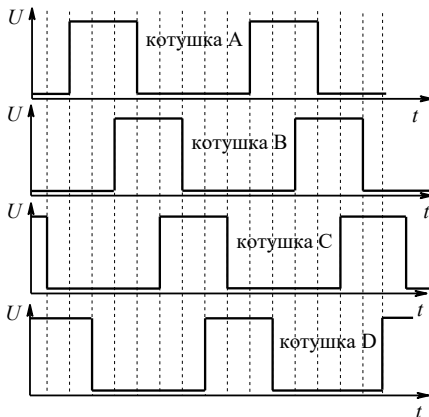


Рис. 4.6. Графік залежності напруги на обмотках крокового двигуна від часу у режимі пів кроку

двигун споживає набагато більший струм, ніж може видати мікроконтролер. Такі перехідні ланки називають драйвером крокового двигуна.

Для неперервної роботи двигуна необхідно періодично на виводах RB0-RB3 мікроконтролера по чергово виставляти логічну “1” з певною

часовою затримкою, яка визначає частоту обертання ротора двигуна. В таблиці 4.1 приведено значення бітів RB0-RB3 з кожним кроком двигуна у режимі повного кроку.

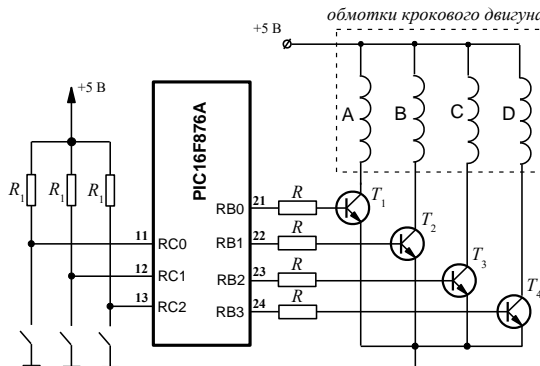


Рис. 4.7. Схема підключення крокового двигуна до мікроконтролера з використанням транзисторних ключів

Таблиця 4.1

Біти крок	RB0	RB1	RB2	RB3
Початковий стан	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	1	0	0	0

Найпростіший код програми для роботи двигуна в режимі повного кроку має вигляд:

```

//*****
void main (void)
{
    TRISB=0b00000000;    // порт В налаштований на вивід
    PORTB=0b00000000;    // очищення порту В
    while (1)
    {
        RB0=1;            // вмикаємо напругу на обмотку А
        __delay_ms (10);
        RB0=0;            // вимикаємо напругу на обмотці А
        RB1=1;            // вмикаємо напругу на обмотці В
        __delay_ms (10);
        RB1=0;            // вимикаємо напругу на обмотці В
        RB2=1;            // вмикаємо напругу на обмотці С
        __delay_ms (10);
        RB2=0;            // вимикаємо напругу на обмотці С
        RB3=1;            // вмикаємо напругу на обмотці D
        __delay_ms (10);
        RB3=0;            // вимикаємо напругу на котушці D
    }
}
//*****

```

У режимі півкроку один оберт ротора складається з 8 кроків. Відповідно до часової діаграми напруги на котушках двигуна (рис. 4.6) в режимі півкроку, в таблиці 4.2 приведено значення бітів (RB0-RB3) для кожного півкроку.

Таблиця 4.2

Біти півкрок	RB0	RB1	RB2	RB3	Число, в десятковій формі
початковий стан	1	0	0	0	1
1	1	1	0	0	3
2	0	1	0	0	2
3	0	1	1	0	6
4	0	0	1	0	4
5	0	0	1	1	12
6	0	0	0	1	8
7	1	0	0	1	9
8	1	0	0	0	1

Хід роботи

1. У середовищі Proteus складіть схему приведену на рис. 4.7.
2. Створіть новий проєкт у середовищі MPLAB.
3. Введіть код програми для роботи крокового двигуна в режимі повного кроку.
4. Скомпілюйте код та перевірте його роботу у середовищі Proteus.
5. Прошийте одержаним hex-файлом мікроконтролер на макетній платі.
6. Зменшуючи часову затримку, визначіть максимальну частоту обертання ротора двигуна.
7. Введіть код програми для роботи крокового двигуна в режимі півкроку та дослідіть його частотні характеристики.

Завдання

1. Оптимізувати код програми, використовуючи масиви.
2. Розробити алгоритм плавного збільшення частоти обертання ротора двигуна.

Контрольні питання

1. Скільки кроків робить кроковий двигун з чотирма обмотками у режимі повного кроку?
2. Скільки кроків робить кроковий двигун з чотирма обмотками у режимі пів кроку?
3. Як змінити напрям обертання ротора крокового двигуна?
4. Проведіть розрахунки частоти обертання ротора крокового двигуна?

ЛАБОРАТОРНА РОБОТА № 5

Семисегментний індикатор. Статична індикація

Мета роботи: Ознайомлення з семисегментним індикатором. Вивчення схем підключення семисегментного індикатора до мікроконтролера. Вивчення алгоритму виведення чисел на індикатор.

Теоретичні відомості

Сьогодні для відображення інформації все частіше використовують графічні дисплеї, однак, семисегментні індикатори також є актуальними. Якщо потрібно лише відображення чисел, то вони можуть стати більш зручним варіантом, тому що вони прості в управлінні і можуть використовуватися сумісно з будь-яким мікроконтролером з достатньою кількістю виводів. Назву семисегментний індикатор отримав у зв'язку з тим, що зображення символу формується з допомогою семи окремо керуючих елементів – сегментів (рис. 5.1). Ці елементи дозволяють відображати будь-яку цифру 0..9, а також деякі символи, наприклад: “-“, “A”, “B”, “C”, “d”, “E”, “F” та інші. Це дає можливість використовувати індикатор для виводу додатніх і від’ємних десяткових чисел і навіть текстових повідомлень. Зазвичай, індикатор має восьмий елемент – крапку, яка використовується при відображенні чисел з десятковою комою. Сегменти індикатора позначають буквами a, b, c, d, e, f, g (a – верхній елемент, далі букви присвоюють сегментам за годинниковою стрілкою; g – центральний сегмент; dp – крапка).

8 незалежних елементів, кожен з яких може знаходитися в одному з двох станів – “горить” або не “горить”, дають можливість відобразити десяткові числа і деякі символи (рис. 5.1).

Світлодіодні семисегментні індикатори мають гранично просту конструкцію, дешеві, надійні. Вони забезпечують високу яскравість і контрастність відображення. Існує велика різноманітність індикаторів: з різним кольором світіння сегментів, різного розміру,

різними схемами підключення світлодіодів (із спільним катодом або спільним анодом). За необхідності відображення декількох розрядів можна встановити кілька однорозрядних індикаторів поруч або вибрати потрібний варіант багаторозрядного індикатора.

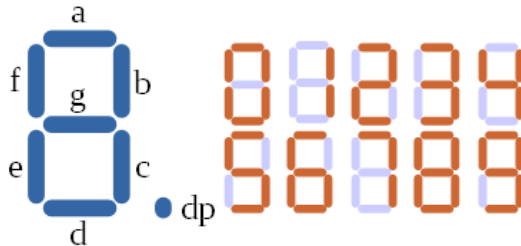


Рис. 5.1. Позначення сегментів семисегментного індикатора

Існує два типи індикаторів: з спільним катодом (рис. 5.2 а) або спільним анодом (рис. 5.2 б). Всього для підключення використовується 9 виводів: один спільний і 8 окремих виводів світлодіодів.

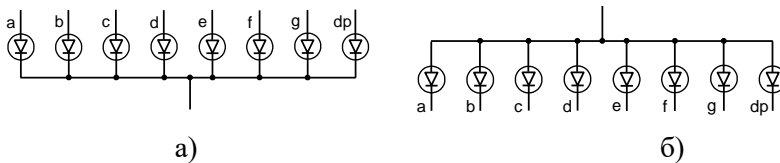












Рис. 5.2 Схема семисегментного індикатора з спільним: а – катодом; б – з спільним анодом

Для формування зображення символу на індикаторі використовують таблицю, в якій показано комбінації засвічених сегментів для кожного символу.

В таблиці 5.1 також приводиться представлення кожної комбінації у вигляді числа в десятковій і в шістнадцятковій системі, що є зручним у програмуванні мікроконтролерів. Посилаючи в певний порт відповідне число, на його виводах будуть виставлені відповідні логічні рівні для засвічування певних сегментів індикатора.

Таблиця 5.1

№	Символ	Рис.	g	f	e	d	c	b	a	HEX	DEC
0	0		0	1	1	1	1	1	1	0x3F	63
1	1		0	0	0	0	1	1	0	0x06	6
2	2		1	0	1	1	0	1	1	0x5B	91
3	3		1	0	0	1	1	1	1	0x4F	79
4	4		1	1	0	0	1	1	0	0x66	102
5	5		1	1	0	1	1	0	1	0x6D	109
6	6		1	1	1	1	1	0	1	0x7D	125
7	7		0	0(1)	0	0	1	1	1	0x07 (0x27)	7 39
8	8		1	1	1	1	1	1	1	0x7F	127
9	9		1	1	0	1	1	1	1	0x6F	111

Для символу “7” в таблиці подані два можливих варіанти відображення.

Опис експерименту

У даній роботі використовується семисегментний індикатор із спільним катодом, під'єднаний до мікроконтролера PIC16F876A. Схема підключення виводів індикатора до мікроконтролера приведена на рисунку (рис. 5.3).

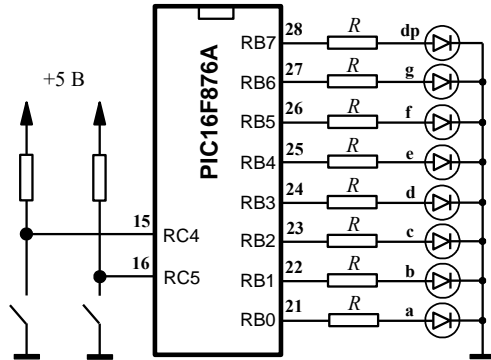


Рис. 5.3. Схема експериментальної макетної плати для вивчення статичної індикації

Значення резисторів R , що обмежують струм через світлодіоди – 470 Ом. З використанням даних таблиці 5.1, програмний код для висвічування цифр на індикаторі має вигляд:

//*****

```
void main (void)
```

```
{
```

```
    unsigned int a;
```

```
    TRISB=0b00000000;    // порт B налаштований на вивід
```

```
    PORTB=0b00000000;    // очищення регістра порту B
```

```
    a=5;                  // число для виводу на індикатор
```

```
    while (1)
```

```
    {
```

```
        switch (a)
```

```
        {
```

```
            case 0: PORTB=0b01111111; break;
```

```
            case 1: PORTB=0b00000010; break;
```

```

        case 2:  PORTB=0b01011011; break;
        case 3:  PORTB=0b01001111; break;
        case 4:  PORTB=0b01100110; break;
        case 5:  PORTB=0b01101101; break;
        case 6:  PORTB=0b11111111; break;
        case 7:  PORTB=0b00000111; break;
        case 8:  PORTB=0b11111111; break;
        case 9:  PORTB=0b01101111; break;
    }
}
}
//*****

```

Хід роботи

1. В середовищі proteus складіть схему приведену на рис. 5.3.
2. В середовищі MPLAB введіть код програми статичної індикації.
3. Проведіть компіляцію коду та перевірити його роботу у середовищі Proteus.
4. Прошийте одержаним hex-файлом мікроконтролер на макетній платі.

Завдання

1. Розробити алгоритм виводу на індикатор чисел від 0 до 9 у зростаючому порядку з затримкою 1 с.
2. Розробити алгоритм виводу на індикатор чисел у зростаючому порядку починаючи з нуля за кожним натисканням однієї з кнопок.
3. Оформити код для індикації чисел з використанням семисегментного індикатора зі спільним анодом.

Контрольні питання

1. Які бувають типи семисегментних індикаторів?

2. Опишіть схему під'єднання індикаторів до мікроконтролера?
3. Які символи можна вивести з допомогою семисегментних індикаторів?
4. В якому режимі повинні працювати виводи порту для під'єднання семисегментного індикатора із спільним анодом

ЛАБОРАТОРНА РОБОТА № 6

Семисегментні індикатори. Динамічна індикація

Мета роботи: Вивчення схеми підключення семисегментних індикаторів для виводу багаторозрядних чисел. Вивчення алгоритму для динамічної індикації багаторозрядних чисел.

Теоретичні відомості

На практиці, зазвичай, потрібно відображення чисел, що складаються більш ніж з одного розряду. Наприклад, для цифрового вольтметра щоб відображати значення з точністю до тисячних вольт потрібно 4 розряди, а для RLC-метра, що відображає дві величини або для частотоміра, необхідна кількість розрядів може скласти 8-10. Кількість розрядів у калькуляторі може перевищувати 12. Якщо з допомогою мікроконтролера управляти кожним розрядом індивідуально, то зі збільшенням їхньої кількості зростає число необхідних виводів мікроконтролера і кількість провідників для підключення індикатора. Так для керування N розрядами потрібно $8 \cdot N$ керуючих ліній і 1 загальний провід. У разі 5 розрядів, кількість керуючих ліній складе 40, а в 10-розрядному індикаторі – вже 80. А тим часом вільних 80 виводів може просто не бути навіть у мікроконтролера у 100-вивідному корпусі (з урахуванням виводів живлення, підключення кварцових резонаторів і реалізації важливих альтернативних функцій). А в 100-вивідних корпусах випускаються далеко не найдешевші мікроконтролери.

Вирішити цю проблему можна з допомогою динамічної індикації (рис. 6.1), суть якої полягає у почерговому засвічуванні розрядів числа використовуючи спільну шину для керування сегментами. Виводи однойменних сегментів усіх розрядів з'єднують разом, утворюючи загальну шину для керування сегментами. Включення потрібного розряду проводять за допомогою спільного виводу (катода або анода, в залежності від типу індикатора) цього розряду (рис. 6.1). Як правило, індикатори, що містять кілька

розрядів, випускають саме в розрахунку на динамічну індикацію і всі необхідні з'єднання виконані всередині пристрою. N -розрядний індикатор в цьому випадку має 8 виводів для управління сегментами і N виводів для керування включенням розрядів (спільний анод або катод розряду). Всього потрібно $8 + N$ виводів, що набагато менше, ніж $8 \cdot N + 1$ як при індикації кожного розряду окремо.

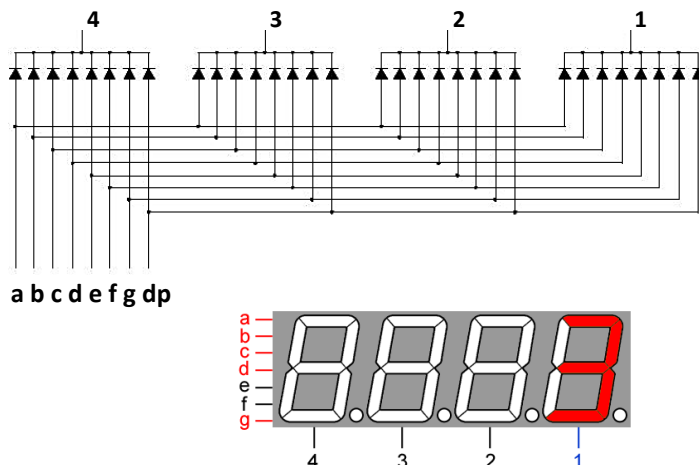


Рис. 6.1. Схема з'єднання виводів сегментів для реалізації динамічної індикації

В індикаторах, розряди яких виконані за схемою зі спільним катодом, сегменти загоряються високим рівнем на виводах керування сегментами, а розряд включається низьким рівнем на відповідному виводі.

Припустимо, що на індикаторі (рис. 6.1) ми хочемо вивести цифру “3” в молодшому розряді. Для цього виставляємо високий логічний рівень напруги на виводи, що відповідають сегментам: a; b; c; d; g; а на виводи решти сегментів виставляємо низький логічний рівень напруги. Оскільки нам потрібно, щоб засвітився тільки молодший розряд, то на вивід 1 (спільний катод для молодшого розряду) подаємо низький логічний рівень, тобто підключаємо на “землю”, а виводи 2, 3, 4 (спільні катоди для інших розрядів)

розмикаємо від шини живлення “земля”. Щоб вивести число на другому розряді, потрібно вивід 1 (спільний катод для молодшого розряду) від’єднати від шини “земля”, натомість під’єднати до землі вивід 2, який відповідає за другий розряд. Таким чином, по черговому включенню кожного розряду з певною частотою, оком буде сприйматися, ніби всі розряди одночасно працюють у статичному режимі незалежно один від одного. В цьому і полягає суть динамічної індикації.

Опис експерименту

У даній роботі використовується чотирирозрядний індикатор із спільним катодом (FYQ-3641Ax). Схему підключення виводів індикатора і виводів портів мікроконтролера PIC16F876A приведено на рисунку 6.2. Виводи RB0-RB7 порту В під’єднані до сегментів: а; b; c; d; e; f; g; dp відповідно через обмежуючі резистори номіналом 470 Ом. Виводи “1”, “2”, “3” і “4”, які відповідають за спільні катоди окремих розрядів індикатора, через транзисторні ключі під’єднані до шини “земля”. Бази транзисторів під’єднані до виводів RC4-RC7 порту С. Якщо на базу відповідного транзистора подати напругу високого логічного рівня, транзистор відкриється і тим самим замкне коло відповідного розряду. За напруги на базі транзистора, яка відповідає низькому логічному рівню, транзистор закритий і коло певного розряду розмикається, тим самим гаситься відповідний розряд. Отже, для того щоб включити відповідний розряд, необхідно на базу відповідного транзистора подати напругу високого логічного рівня.

Розглянемо алгоритм динамічної індикації. Нехай маємо деяке число x , яке треба вивести на індикатор. Кількість розрядів даного числа – k .

1. Знаходимо значення першого розряду ($n = x \% 10$) і включаємо відповідні сегменти для його індикації ($PORTB = \text{“відповідний код”}$). Включаємо перший розряд індикатора, тобто на вивід RC7 мікроконтролера подаємо логічну “1”.

2. Робимо деяку часову затримку для нормального візуального сприйняття виведеної цифри.
3. Очищаємо порт В ($PORTB = 0$) і відключаємо перший розряд ($RC7 = 0$). Після чого робимо невеличку затримку для повної релаксації свічення першого розряду.
4. Повторюємо пункти 1-3 для наступних розрядів, після чого весь процес повторюється знову.

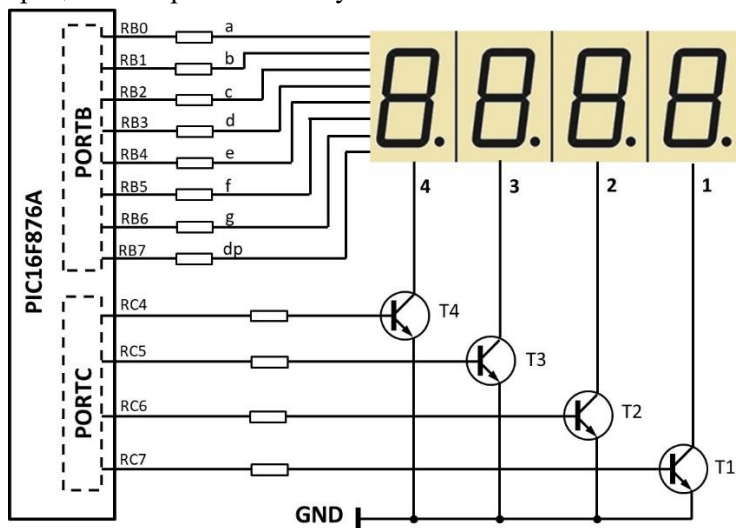


Рис. 6.2. Схема макетної плати для реалізації динамічної індикації

Код програми для реалізації динамічної індикації має вигляд:

```

//*****
void main (void)
{
    unsigned int k, x, a;
    TRISB=0b00000000;
    TRISC=0b00000000;
    PORTB=0b00000000;
    RC4=0;
    RC5=0;
    RC6=0;

```

```

RC7=0;
x=2345;           // число для відображення на індикаторі
while (1)
{
    a=x;           // вводимо змінну a для знаходження
                    // кількості розрядів
    k=1;           // вибираємо перший розряд (десяті)
    while (a>0 )
    {
        n=a%10;    // знаходимо значення відповідного розряду
        switch (k)  // включення відповідного розряду
        {
            case 1:  RC7=1;  break;
            case 2:  RC6=1;  break;
            case 3:  RC5=1;  break;
            case 4:  RC4= ;  break;
        }
        switch (n)   // виведення значення
                     // відповідного розряду
        {
            case 0:  PORTB=0b00111111;  break;
            case 1:  PORTB=0b00000110;  break;
            case 2:  PORTB=0b01011011;  break;
            case 3:  PORTB=0b01001111;  break;
            case 4:  PORTB=0b01100110;  break;
            case 5:  PORTB=0b01101101;  break;
            case 6:  PORTB=0b01111101;  break;
            case 7:  PORTB=0b00000111;  break;
            case 8:  PORTB=0b01111111;  break;
            case 9:  PORTB=0b01101111;  break;
        }
        __delay_ms (4); // часова затримка для чіткого
                        // зорового сприйняття зображення
    }
}

```

```

PORTB=0b00000000;    // очищення порту
RC4=0;                // вмикаємо перший розряд
RC5=0;                // вмикаємо другий розряд
RC6=0;                // вмикаємо третій розряд
RC7=0;                // вмикаємо четвертий розряд
__delay_us (10);      // часова затримка для повної
                      // релаксації свічення сегментів
k=k+1;                // перехід на наступний розряд
a=a/10;               // виділяємо число для знаходження
                      // значення наступного розряду
    }
}
}
//*****

```

Хід роботи

1. У середовищі proteus складіть схему приведену на рис. 6.2.
2. У середовищі MPLAB створіть новий проєкт та введіть вище приведений код динамічної індикації.
3. Проведіть компіляцію коду та перевірте його роботу у середовищі Proteus.
4. Прошийте одержаним після компіляції коду hex-файлом мікроконтролер на макетній платі.

Завдання

1. Знайти найменшу часову затримку між переключеннями розрядів, за якої спостерігається чітка індикація багаторозрядного числа.
2. Розробити алгоритм індикації чисел, що послідовно змінюються через певний інтервал часу.

Контрольні питання

1. В чому полягає динамічна індикація багаторозрядних чисел з

- допомогою семисегментних індикаторів?
2. Опишіть алгоритм роботи динамічної індикації?
 3. Як впливає інерційність зору на частоту повторення засвічування розрядів числа?
 4. Які характеристики світлодіодів у сегментах індикатора впливають на частоту повторення засвічування розрядів числа?

ЛАБОРАТОРНА РОБОТА № 7

Робота з рідкокристалічним дисплеєм.

Ініціалізація дисплея LCD1602A

Мета роботи: Вивчення принципу роботи рідкокристалічного дисплея LCD1602. Вивчення алгоритму ініціалізації у 8-бітному і 4-бітному режимі. Розробити алгоритм виведення на дисплей слів та багаторозрядних чисел.

Теоретичні відомості

Рідкокристалічний дисплей (LCD) 1602A це – символьний дисплей, здатний виводити символи у двох рядках по 16 символів у кожному (рис. 7.1). В технічній документації приводиться перелік символів, які можна вивести на екран, серед яких є арабські цифри, літери латинського та грецького алфавіту, розділові знаки, а також багато інших спеціальних символів. Дисплей містить внутрішній контролер HD44780, з допомогою якого відбувається прийом команд від зовнішнього пристрою та виведення символів на екран.



Рис. 7.1. Зовнішній вигляд рідкокристалічного дисплея LCD1602A

Приклад схеми підключення дисплея до мікроконтролера PIC16F876A приведено на рисунку 7.2.

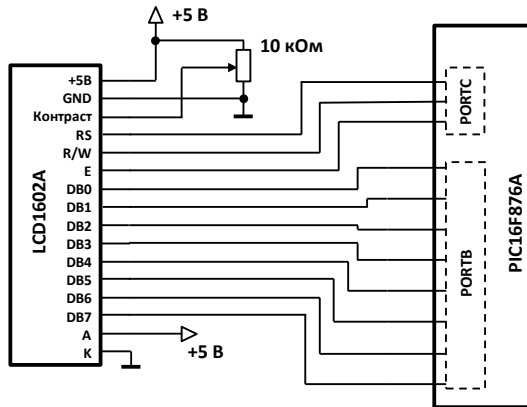


Рис. 7.2. Схема підключення дисплея до мікроконтролера

В таблиці 7.1 приведено опис виводів дисплея LCD1602A.

Таблиця 7.1.

Символ	Опис	
VSS	Живлення	GND (Земля)
VDD		+5В (живлення)
V0		Регулювання контрасту
RS	Регістр вибору (“1” – дані; “0” – команда)	
R/W	запис і читання даних (“0” – запис; “1” – читання)	
E	Стробування	
DB0-DB7	8-бітний порт даних і команд	
A	Підсвітка екрана	+5 В
K		GND (Земля)

Дисплей може працювати в одному з двох режимів при ширині лінії даних 8 біт або 4 біт. У 8 – бітному режимі використовуються 8

выводів (D0-D7) лінії даних і команд, у 4 – бітному режимі використовуються тільки 4 виводи (D4-D7).

Розглянемо механізм роботи дисплея.

Після включення живлення необхідно провести ініціалізацію дисплея, тобто задати параметри виводу символів, такі як: зсув курсора відносно попередньої позиції (вліво/вправо), видимість і мигання курсора, координату для виводу, вибрати кількість рядків для виводу.

При ініціалізації дисплея на вивід R/W подається логічний “0”, оскільки ми записуємо дані в мікроконтролер дисплея.

Якщо на вивід RS є подано логічний “0”, то код поданий на лінію DB0-DB7 сприймається внутрішнім контролером дисплея як команда. Якщо на вивід RS є подано логічну “1”, то код, поданий на лінію DB0-DB7, сприймається внутрішнім контролером дисплея як ASCII код символу, який треба вивести на дисплей.

Є декілька команд керування дисплеєм:

- **Очистка дисплея:**

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0	1

Всі символи стираються і адреса символу, який виводиться, стає 0x00.

- **Повернення назад:**

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	1	-

Адреса символу, який виводиться, встановлюється в 0x00 (при цьому всі символи на дисплеї залишаються незмінними).

- **Режими зсуву:**

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	1	I/D	S

I/D – Задає напрям руху курсора після вводу символу. (1 – вліво, 0 – вправо);

S – зсув курсора, що супроводжується зсувом символів.

- **Керування дисплеєм:**

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	1	D	C	B

D – включення дисплея (1 – включений, 0 – виключений);

C – видимість курсора (1 – видимий, 0 – невидимий);

B – мигання курсора (1 – мигає, 0 – відображається постійно).

- **Режим роботи:**

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	1	DL	N	F	-	-

DL – розрядність лінії даних і команд (1 – 8 біт, 0 – 4 біт);

N – кількість рядків для відображення даних (1 – два, 0 – один);

F – розмір символів (1 – 5×10 пікселів, 0 – 5×8 пікселів).

- **Вибір позиції курсора на дисплеї:**

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0

AC6-AC0 – адреса символу на дисплеї куди потрібно встановити курсор.

З технічної документації дізнаємося, що 8-розрядні коди позицій для символів першого становлять 128–143, а коди позицій для символів другого рядка становлять 192–207. Наприклад, якщо на лінію D7–D0 подати код 194, то символ виведеться в третій комірці другого рядка.

- **Виведення символу на дисплей:**

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
1	0	біт 7	біт 6	біт 5	біт 4	біт 3	біт 2	біт 1	біт 0

Значення бітів D0 – D7 відповідають коду символу, який виводиться.

Після установки коду команди, на вивід E подається стробуючий імпульс, тобто формується перехід з рівня логічної “1” в рівень логічного “0” з невеликою часовою затримкою (38 мкс), після чого на вивід E дисплея повертається рівень логічної “1”, що приводить його в готовність прийому нової команди.

У 8 – бітному режимі команди і дані приймаються за один такт поданням на виводи D0–D7 відповідний 8-бітний код. У 4 – бітному режимі команди і дані приймаються за два такти по 4 біти поданням на виводи D4–D7 спочатку старших 4 біти, а потім молодших 4 біти відповідного коду. Виводи D0-D3 у 4-бітному режимі не використовуються. Після кожного такту прийому коду необхідне стробування. Перевагою 4-бітного режиму над 8-бітним є використання меншої кількості з'єднувальних провідників (4 шт).

Алгоритм ініціалізації дисплея при ширині шини даних 8 біт згідно з технічною документацією приведено на рисунку 7.3.

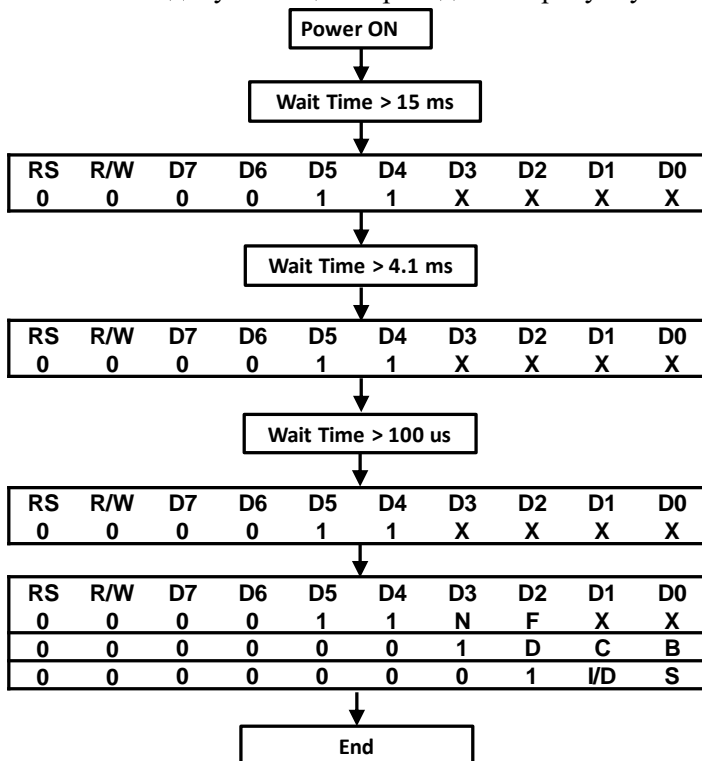


Рис. 7.3. Блок схема алгоритму ініціалізації дисплея у 8-бітному режимі

Як видно з рисунка 7.3, після подачі живлення на дисплей необхідно зробити часову затримку не менше 15 мс. Далі на лінію D0–D7, за умови подачі на виводи RS і R/W логічного “0”, подається відповідний код з відповідною часовою затримкою:

D7–D0 = 0b00110000;

затримка > 4,1 мс;

D7–D0 = 0b00110000;

затримка > 100 мкс;

D7–D0 = 0b00110000;

затримка > 38 мкс.

Після подачі на лінію D0–D7 відповідного коду (що відповідає числу 48), подаються команди для задання параметрів роботи дисплея, а саме: ширина лінії даних (8 або 4 біт), кількість рядків, включення дисплея, розмір шрифту, видимість і мигання курсору, напрям зсуву курсору після виведення символу (вліво/вправо), координати виводу символу та очистка дисплея. Коди усіх цих команд приведені вище.

Після команд налаштування можна вивести символ за вказаними координатами (рядок та положення в рядку), для цього необхідно на вивід RS подати логічну “1”, після чого на лінію D0–D7 подається ASCII код відповідного символу. Далі знову можна виводити символи, які будуть зміщуватись вліво або вправо залежно від команд налаштувань. Після кожної команди, як і після подачі даних для виводу, необхідно проводити стробування, тобто на вивід E подається логічний “0” із затримкою >38мкс після чого знову подається логічна “1”, і дисплей стає готовий до прийому нових команд.

Алгоритм ініціалізації дисплея при ширині шини даних 4 біти приведено на рисунку 7.4.

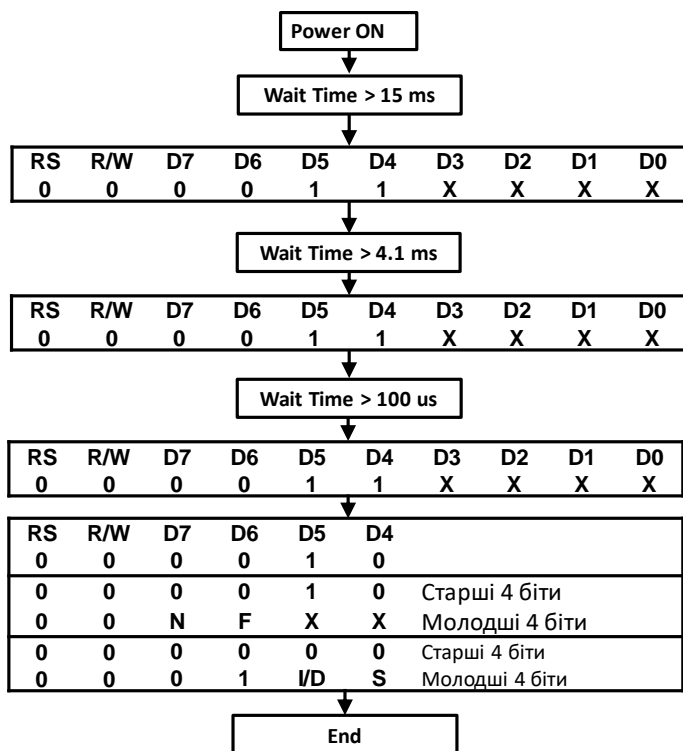


Рис. 7.4. Блок схема алгоритму ініціалізації дисплея у 4-бітному режимі

Як і у випадку 8-бітного режиму, спочатку на виводи D7-D4 подають відповідний код з відповідною часовою затримкою:

D7-D4 = 0b0011;

затримка > 4,1 мс;

D7-D4 = 0b0011;

затримка > 100 мкс;

D7-D4 = 0b0011;

затримка > 38 мкс.

Після чого на виводи D7-D4 подають команду вибору 4-бітного режиму. Далі всі наступні команди подають на виводи D7-D4 за два

такти по 4 біти: спочатку старші 4 біти потім молодші 4 біти (рис. 7.4):

$D7-D4 = 0b0010$ (4-бітний режим)

затримка > 38 мкс.

Після кожної команди проводиться стробування, тобто на вивід Е подається логічний “0” із затримкою > 38 мкс, після чого знову подається логічна “1” і дисплей стає готовий до прийому нових команд.

Опис експерименту

Схема експериментальної макетної плати приведена на рисунку 7.5.

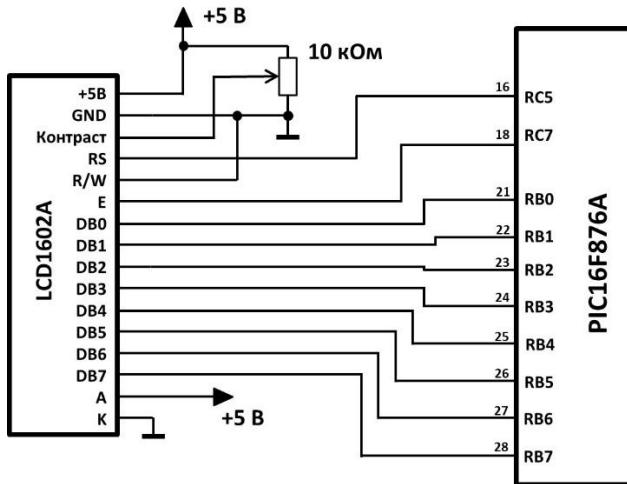


Рис. 7.5. Схема макетної плати роботи дисплея у 8-бітному режимі

Код програми ініціалізації дисплея у 8-бітному режимі має вигляд:

```

//*****
void strob ()                // функція стробування
{
    RC7=0;                  // зміна стану на стробуючому виводі
    __delay_us (50);        // затримка не менше 38 мкс
}

```

```

    RC7=1;
    __delay_us (50);
}
//_____
char x, char* y, int i; // оголошення потрібних змінних
void main (void)
{
    TRISB=0b00000000;    // порт В налаштовуємо на вивід
    TRISC=0b00000000;    // порт С налаштовуємо на вивід
    __delay_ms (20);      // необхідна часова затримка
    RC5=0;                // RS = 0, режим вводу команд
    RC7=1;                // E = 1, стробуючий вхід
    __delay_us (100);
    PORTB=0b00110000;     // ініціалізація, згідно документації
    __delay_ms (5);
    strob ();
    PORTB=0b00110000;     // ініціалізація, згідно документації
    __delay_ms (5);
    strob ();
    PORTB=0b00110000;     // ініціалізація, згідно документації
    __delay_ms (5);
    strob ();
//_____
    PORTB=0b00111000;     // 8-бітний режим, 2 рядки
    __delay_us (100);
    strob ();
//_____
    PORTB=0b00001111;     // видимість і мигання курсору
    __delay_us (100);
    strob ();
//_____
    PORTB=0b00000100;     // зсув курсору вправо
    __delay_us (100);

```

```

    strob ();

//
    PORTB=136;           // позиція курсору
    __delay_us (100);
    strob ();

//
    RC5=1;               // RS = 1, режим виводу даних
    PORTB='X';           // виводимо символ "X"
    __delay_us (50);
    strob ();

//
// команди для виведення інших символів
while (1)
{
    // інші команди користувача
}
}
//*****

```

Для 4-бітного режиму роботи дисплея виводи D0–D3 не задіяні, що дозволяє використовувати меншу кількість з'єднувальних провідників (рис. 7.6).

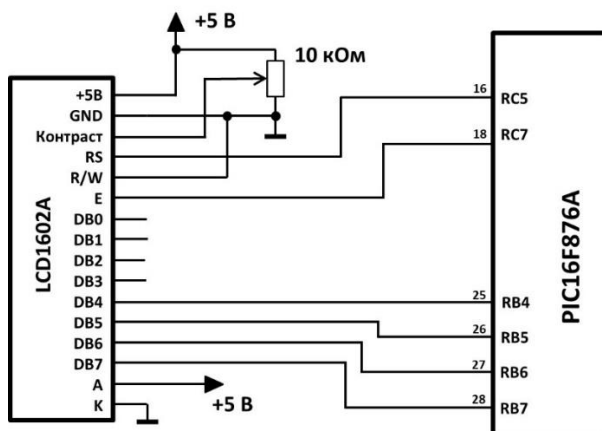


Рис. 7.6. Схема макетної плати роботи дисплея у 4-бітному режимі

Код програми ініціалізації дисплея у 4-бітному режимі має вигляд:

```
/**
void strob ()          // функція стробування
{
    RC7=0;             // перехід на виводі E з "1" до "0"
    __delay_us (50);   // затримка не менше 38мкс
    RC7=1;
    __delay_us (50);
}
//
char x, char* y, int i; // оголошення потрібних змінних
void main (void)
{
    TRISB=0b00000000;   // порт B налаштовуємо на вивід
    TRISC=0b00000000;   // порт C налаштовуємо на вивід
    __delay_ms (20);     // часова затримка за інструкцією
    RC5=0;               // RS = 0, режим вводу команд
    RC7=1;               // E = 1, ініціалізація стробу
    __delay_us (100);
//
    PORTB=0b00110000;    // код згідно документації
    __delay_ms (5);
    strob ();
    PORTB=0b00110000;    // код згідно документації
    __delay_ms (5);
    strob ();
    PORTB=0b00110000;    // код згідно документації
    __delay_ms (5);
    strob ();
//
    PORTB=0b0010<<4;    // 4-бітний режим, 2 рядки
    __delay_us (100);
}
```

```

        strob ();
//
PORTB=0b0010<<4; // старші 4-біти команди
    __delay_us (50);
        strob ();
PORTB=0b1000<<4; // молодші 4-біти команди
    __delay_us (50);
        strob ();
//
PORTB=0b0000<<4; // 4 старші біти команди мигання курсору
    __delay_us (50);
        strob ();
PORTB=0b1111<<4; // 4 молодші біти команди мигання курсору
    __delay_us (50);
        strob ();
//
PORTB=0b0000<<4; // 4 старші біти команди зсуву курсору
    __delay_us (50);
        strob ();
PORTB=0b0110<<4; // 4 молодші біти команди зсуву курсору
    __delay_us (50);
        strob ();
//
PORTB=136; // 4 старші біти команди позиції курсору
    __delay_us (50);
        strob ();
PORTB=136<<4; // 4 молодші біти команди позиції курсору
    __delay_us (50);
        strob ();
//
RC5=1; // RS = 1, режим виводу даних
PORTB='X'; // виводимо символ "X" (старші 4 біти)
    __delay_us (50);

```

```

        strob ();
PORTB= 'X' <<4;           // виводимо букву "X" (молодші 4 біти)
        __delay_us (50);
        strob ();

// _____
// команди для виведення інших символів
while (1)
{
    // інші команди та функції користувача
}
}
//*****

```

Хід роботи

1. У середовищі Proteus складіть схему приведену на рис. 7.5.
2. У середовищі MPLAB введіть код ініціалізації дисплея у 8-бітному режимі з виведенням символу "X".
3. Проведіть компіляцію коду та перевірити його роботу у середовищі Proteus.
4. У середовищі MPLAB введіть код ініціалізації дисплея у 4-бітному режимі з виведенням символу "Y", проведіть його компіляцію та перевірте його роботу у середовищі proteus.
5. Прошийте мікроконтролер на макетній платі hex-файлом одержаним під час компіляції коду для обидвох режимів дисплея.

Завдання

1. Вивести на дисплей у різних позиціях цифри та символи.
2. Вивести символи на дисплей з відображенням курсора та без нього.
3. Методом підбору визначити найменшу часову затримку для виконання команд дисплеєм, за якої дисплей нормально працюватиме.

Контрольні питання

1. Опишіть схему підключення дисплея до мікроконтролера?
2. У яких режимах може працювати дисплей LCD1602A?
3. В чому полягає ініціалізації дисплея?
4. Що таке стробування?
5. Які символи можна виводити на дисплей?

ЛАБОРАТОРНА РОБОТА № 8

Вивчення алгоритму виведення слів, багаторозрядних чисел та нестандартних символів на дисплей LCD1602A

Мета роботи: Вивчення алгоритму виведення на дисплей слів та багаторозрядних чисел.

Теоретичні відомості

Процедура ініціалізації рідкокристалічного дисплея LCD1602 описана в лабораторній роботі № 7. Поряд із символами та знаками, які є на клавіатурі, на дисплей можна виводити інші символи, наприклад, символи грецького алфавіту та багато інших символів. В технічній документації дисплея приведено таблицю кодів різних символів, які можна вивести на дисплей (рис. 8.1).

Для простоти розглянемо алгоритм виведення символів на дисплей у 8-бітному режимі.

Розглянемо приклад. Як видно з рисунка 8.1 код символу “ π ” є 0b11110111, а код символу “ Σ ” є 0b11110110. Отже, частина програми для виведення на дисплей одного із символів має вигляд:

```
//*********************************************************************  
// команди ініціалізації дисплея  
// команда задання позиції виводу символу  
PORTB=0b11110111;    // виводимо символ “ $\pi$ ”  
strob ();             // функція стробування  
//*********************************************************************
```

b7- b3 b0		0000	0010	0011	0100	0101	0110	0111	1010	1011	1100	1101	1110	1111
0000	CG RAM (1)		0	a	P	`	P		-	夕	Ξ	α	p	
0001	(2)		!	1	A	Q	a	q	。	ア	チ	△	ä	q
0010	(3)		"	2	B	R	b	r	「	イ	ツ	×	β	θ
0011	(4)		#	3	C	S	c	s	」	ウ	テ	ε	ε	ω
0100	(5)		\$	4	D	T	d	t	、	エ	ト	μ	μ	Ω
0101	(6)		%	5	E	U	e	u	・	オ	ナ	1	ε	Ü
0110	(7)		&	6	F	V	f	v	ヲ	カ	ニ	ヨ	ρ	Σ
0111	CG RAM (8)		'	7	G	W	g	w	フ	キ	ヌ	ラ	g	π
1000	CG RAM (1)		(8	H	X	h	x	ィ	ク	ネ	リ	フ	×
1001	(2))	9	I	Y	i	y	ゥ	ツ	ル		'	Y
1010	(3)		*	:	J	Z	j	z	エ	コ	ハ	レ	j	チ
1011	(4)		+	:	K	[k	[オ	サ	ヒ	ロ	*	斤
1100	(5)		、	<	L	¥	l	l	セ	シ	フ	ワ	φ	円
1101	(6)		-	=	M	I	m	}	ユ	ズ	ヘ	ン	も	÷
1110	(7)		。	>	N	^	n	→	ヨ	セ	ホ	°	ん	
1111	CG RAM (8)		/	?	0	_	o	+	ッ	リ	マ	"	ö	■

Рис. 8.1. Коды різних символів для виводу на дисплей LCD1602A

Розглянемо алгоритм виводу на дисплей слів. Для виводу слів на дисплей необхідно ввести певну змінну типу `char*` або `string`, якій буде присвоюватись слово або набір символів (коротке речення) і змінну для перебору символів у заданому слові. Алгоритм виведення слів на дисплей зводиться до почергового виведення символів заданого слова по порядку з першого по останній. Рахунок символів починається з нуля.

Частина програмного коду для виведення слів на дисплей має вигляд:

```

//*****
char* x;           // змінна для зберігання ряду символів
int i;             // змінна для перебору символів у слові
//
x="Hello World!"   // змінній "x" присвоїли коротке речення
                  // "Hello World!"
i=0;               // рахунок символів починається з нуля
while (x[i]!=0)    // цикл перебору символів
{
    PORTB=x[i];    // виводимо на дисплей i-й символ слова
    __delay_us (50);
    strob ();      // функція стробування
    i++;           // перехід до наступного символу у реченні
}
//*****

```

Розглянемо алгоритм виведення багаторозрядного числа. У цьому випадку також методом перебору розрядів числа виводиться на дисплей їх значення, однак спочатку значення кожного розряду необхідно перетворити у символний тип. У мові C++ для перетворення цілого числа у символний тип використовується наступний оператор:

```
x=a+'0';
```

де а – число цілого типу, а х – значення цього числа як змінна символного типу.

Програмний код виведення багаторозрядного числа має вигляд:

```

//*****

```

```

a=123456789;      // певне число цілого типу
while (a!=0)      // цикл для перебору розрядів числа
{
    x=a%10+'0';    // перетворення значення розряду в символний
                  // тип
    PORTB=x;       // виводимо значення на дисплей
    __delay_us (50);
}

```

```

        strob ();           // функція стробування
        a=a/10;             // перехід до наступного розряду
    }
//*****

```

Як видно з приведеного програмного коду, виведення починається з “9” і закінчується виводом “1”, тому для нормального відображення числа на дисплеї необхідно спочатку задати відповідний режим зсуву курсору після виводу символів, або виконати процедуру дзеркальної перестановки розрядів (тобто отримати число 987654321).

Опис експерименту

Схема макетної плати для роботи дисплея у 8-бітному режимі приведена на рисунку 8.2.

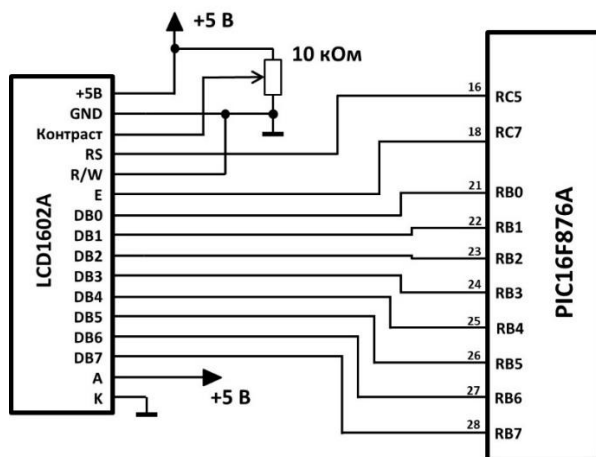


Рис. 8.2. Схема макетної плати для роботи дисплея у 8-бітному режимі

Хід роботи

1. У середовищі Proteus складіть схему приведену на рис. 8.2.
2. У середовищі MPLAB введіть код виводу на дисплей символів грецького алфавіту та інших символів приведених на рис. 8.1.

3. Проведіть компіляцію коду та перевірте його роботу у середовищі Proteus.
4. У середовищі MPLAB введіть програмний код для виводу на дисплей слів "Hello World" та перевірте його у середовищі Proteus.
5. Введіть код для виводу на дисплей багаторозрядних чисел, та перевірте його роботу у середовищі Proteus.
6. Прошийте мікроконтролер на макетній платі hex-файлом одержаним під час компіляції коду.

Контрольні питання

1. Як вивести на дисплей символи грецького алфавіту?
2. Як вивести на дисплей значення певної змінної?
3. Опишіть алгоритм виводу на дисплей слів та речень?
4. Опишіть алгоритм виводу на дисплей багаторозрядних чисел?

ЛАБОРАТОРНА РОБОТА № 9

Вивчення модуля timer0. Генератор імпульсів

Мета роботи: Ознайомитись із регістрами та принципом роботи модуля timer0 мікроконтролера PIC16F876A. Вивчити роботу функції опрацювання переривань таймера. Побудувати алгоритм генерування імпульсів різної скважності.

Теоретичні відомості

Мікроконтролери PIC містять різні модулі, серед яких – таймери. В мікроконтролері PIC16F876A є 3 модулі таймерів: timer0, timer1, timer2, кожен з них має свої особливості. З допомогою таймерів можна робити відлік реального часу, що дозволяє створити на базі мікроконтролера годинник, генератор імпульсів із заданою скважністю тощо. Головна властивість таймерів – це те, що відлік часу відбувається незалежно від ходу основного коду програми, чим забезпечується стабільна робота програми, так і економія ресурсів мікроконтролера. Коли у тілі програми використовується часова затримка у вигляді команди наприклад “__delay_ms(200);” то вся робота мікроконтролера зупиняється на зазначений проміжок часу після чого програма виконується далі. Використовуючи таймери можна робити відлік часу не залежно від ходу основної програми. Коли використовується затримка “__delay_ms(200);” робота програми зупиняється на визначений проміжок часу, що унеможливорює одночасно відслідковувати певні події. Наприклад, якщо в момент часової затримки на мікроконтролер поступить сигнал, то мікроконтролер його не зареєструє. Таймери працюють незалежно від ходу основної програми, що дозволяє паралельно до виконання певного алгоритму робити відлік часу, наприклад, для генерації сигналів певної частоти.

Робота таймера базується на рахунку тактових імпульсів від генератора тактової частоти. Оскільки виконання мікропроцесором однієї елементарної операції відбувається за 4 такти тактового

генератора, то відповідно частота рахунку імпульсів у 4 рази менша за частоту тактового генератора. Також таймер може рахувати імпульси від зовнішнього генератора, в цьому випадку частота рахунку таймера рівна частоті генератора імпульсів.

Модуль timer0

Модуль timer0 мікроконтролера PIC16F876A є 8-бітний, тобто таймер може рахувати від 0 до 255. Значення таймера записується в регістр **TRM0**, який є доступним для читання і запису. Швидкість рахунку імпульсів можна задавати, вибираючи відповідний коефіцієнт подільника частоти тактових імпульсів. У деяких мікроконтролерах даного сімейства (наприклад PIC18F4550) модуль timer0 є 16-бітним, тобто може рахувати від 0 до 65535. Блок схема модуля timer0 приведена на рисунку 9.1.

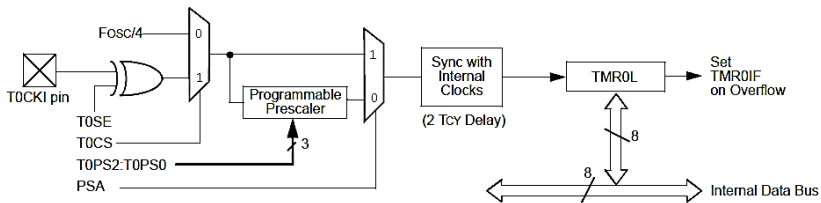


Рис. 1.9. Блок-схема модуля timer0 мікроконтролера PIC16F876A

Вивід T0CKI (RA4) слугує для підключення зовнішнього тактового генератора. Вивід RA4 порту A повинен бути налаштований як цифровий вхід з допомогою регістра **ADCON1**.

Біти налаштування модуля timer0 розміщені в регістрі **OPTION_REG**.

OPTION_REG							
RBPB	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
Біт 7							Біт 0

Біт 7 **RBPB**: Включення підтягуючих резисторів на входах порту B.

1 = підтягуючі резистори відключені;

0 = підтягуючі резистори включені.

Біт 6 **INTEDG**: Вибір активного фронту сигналу на вході зовнішнього переривання.

1 = переривання по передньому фронту сигналу;

0 = переривання по задньому фронту сигналу.

Біт 5 **T0CS**: Вибір тактового сигналу для TMR0.

1 = зовнішній тактовий сигнал з входу RA4/T0CKI;

0 = внутрішній тактовий сигнал CLKOUT.

Біт 4 **T0SE**: Вибір фронту приросту TMR0 при зовнішньому тактовому сигналі.

1 = приріст по задньому фронту сигналу (з “1” до “0”) на виводі RA4/T0CKI;

0 = приріст по передньому фронту сигналу (з “0” до “1”) на виводі RA4/T0CKI.

Біт 3 **PSA**: Вибір включення подільника частоти.

1 = подільник включений перед WDT;

0 = подільник включений перед TMR0.

Біт 2-0 **PS2:PS0**: Установка коефіцієнта подільника частоти.

Значення	Для TMR0	Для WDT
000	1:2	1:1
001	1:4	1:2
010	1:8	1:4
011	1:16	1:8
100	1:32	1:16
101	1:64	1:32
110	1:128	1:64
111	1:256	1:128

Особливості модуля **timer0**

- 8-бітний таймер (рахує від 0 до 255);
- Може працювати як від внутрішнього, так і від зовнішнього тактового генератора;
- Рахує по передньому або задньому фронту сигналу;
- 8-бітний подільник частоти імпульсів генератора;
- за переповнення таймера (при переході від 255 до 0) може генерувати переривання;
- таймер працює незалежно від основної програми.

Цей таймер можна використовувати для генерації імпульсів заданої ширини або для підрахунку входних імпульсів.

За умови переповнення таймера, тобто при переході значення регістра **TMR0** з 255 до 0, генерується переривання, тобто значення біта **TMR0IF** в регістрі **INTCON** встановлюється в “1” (повинен скидатися програмно). Дозвіл на переривання встановлюється бітом **TMR0IE** в регістрі **INTCON**. Біт **GIE** регістра **INTCON** встановлює дозвіл на переривання усіх модулів мікроконтролера.

Регістр **INTCON** містить наступні біти керування перериваннями від модуля **timer0**:

INTCON							
GIE	PEIE	TMR0IE	T0SE	RBIE	TMR0IF	INTF	RBIF
Біт 7							Біт 0

Біт 7 **GIE**: Глобальний дозвіл переривань.

1 = дозволено всі немасковані переривання;

0 = всі переривання заборонено.

Біт 6 **PEIE**: Дозвіл переривань від периферійних модулів.

1 = дозволено всі немасковані переривання периферійних модулів;

0 = переривання від периферійних модулів заборонені.

Біт 5 **TMR0IE**: Дозвіл переривання при переповненні TMR0.

1 = переривання дозволено;

0 = переривання заборонено.

Біт 4 **INTE**: Дозвіл зовнішнього переривання INT.

1 = переривання дозволено;

0 = переривання заборонено.

Біт 3 **RBIE**: Дозвіл переривання при зміні сигналу на входах RB7:RB4 PORTB.

1 = переривання дозволено;

0 = переривання заборонено.

Біт 2 **TMR0IF**: Прапорець переривання при переповненні регістра TMR0.

1 = відбулось переповнення TMR0 (обнуляється програмно);

0 = переповнення TMR0 не було.

Біт 1 **INTF**: прапорець зовнішнього переривання INT.

1 = виконано умову зовнішнього переривання на виводі RB0/INT (обнуляється програмно);

0 = зовнішнього переривання не було.

Біт 0 **RBIF**: прапорець переривання за зміни рівня сигналу на входах RB7:RB4 порту PORTB.

1 = зафіксовано зміну рівня сигналу на одному із входів RB7:RB4 (обнуляється програмно);

0 = не було зміни рівня сигналу ні на одному із входів RB7:RB4.

Розрахунок параметрів таймера

Для прикладу розглянемо наступні параметри:

- Нехай таймер працює від внутрішнього кварцового генератора частотою $F_{OSC} = 20 \text{ МГц}$;
- Вибираємо коефіцієнт подільника частоти імпульсів (наприклад $k = 2$);
- Маємо 8 – бітний таймер-лічильник від 0 до 255;

- Пам'ятаємо, що один машинний цикл відбувається за 4 такти генератора.

Час, протягом якого таймер порахує від 0 до 255, визначається за формулою:

$$T = \frac{4}{F_{\text{OSC}}} \cdot k \cdot N,$$

де k – коефіцієнт подільника частоти імпульсів, N – розрядність таймера (8 – біт, тобто $N = 256$), F_{OSC} – частота внутрішнього тактового генератора.

Для $k = 2$ і $F_{\text{OSC}} = 20$ МГц маємо:

$$T = \frac{4}{20 \text{ МГц}} \cdot 2 \cdot 256 = 1024 \text{ мкс.}$$

Тобто, за час 1024 мкс, таймер порахує від 0 до 255.

Модуль timer1

Особливості модуля timer1:

- 16-бітний таймер (рахує від 0 до 65535);
- може працювати як від внутрішнього, так і від зовнішнього тактового генератора;
- максимальний коефіцієнт подільника частоти імпульсів – 8;
- таймер працює по передньому фронту сигналу;
- переривання генерується за переповнення таймера (при переході від 65535 до 0);
- таймер працює постійно.

За умови переповнення таймера, тобто перехід значення регістра **TMR1** з 65535 до 0, генерується переривання, тобто значення біта **TMR1IF** в регістрі **PIR1** встановлюється в “1” (повинен скидатися програмно). Дозвіл на переривання встановлюється бітом **TMR1IE** в “1” в регістрі **PIE1**.

Як і для модуля timer0, для генерації переривань від модуля timer1 необхідно встановити дозвіл на переривання бітом **GIE** регістра **INTCON**.

Біти налаштування модуля timer1 розміщено в регістрі **T1CON**.

T1CON							
-	-	T1CKPS1	T1CKPS0	T1OSCEN	TMR1CS	TMR1CS	TMR1ON
Біт 7							Біт 0

Біт 7-6 **Не задіяні:** читаються як “0”.

Біт 5-4 **T1CKPS1:T1CKPS0:** Вибір коефіцієнта подільника частоти.

11 = 1:8;

10 = 1:4;

01 = 1:2;

00 = 1:1

Біт 3 **T1OSCEN:** Включення тактового генератора.

1 = генератор включений;

0 = генератор виключений.

Біт 2 **-TMR1CS:** Синхронізація зовнішнього тактового сигналу.

TMR1CS = 1:

1 = не синхронізувати зовнішній тактовий;

0 = синхронізувати зовнішній тактовий сигнал.

TMR1CS = 0:

Значення біта ігнорується. Модуль timer1 використовує внутрішній генератор коли **TMR1CS = 0**.

Біт 1 **TMR1CS:** Вибір джерела тактового сигналу.

1 = зовнішнє джерело з виводу RC0/T1OSO/T1CKI
(активним є передній фронт сигналу);

0 = внутрішнє джерело $F_{osc}/4$.

Біт 0 **TMR1ON:** включення модуля TMR1.

1 = модуль timer1 включений;

0 = модуль timer1 виключений.

Блок-схему модуля timer1 приведено на рисунку 9.2.

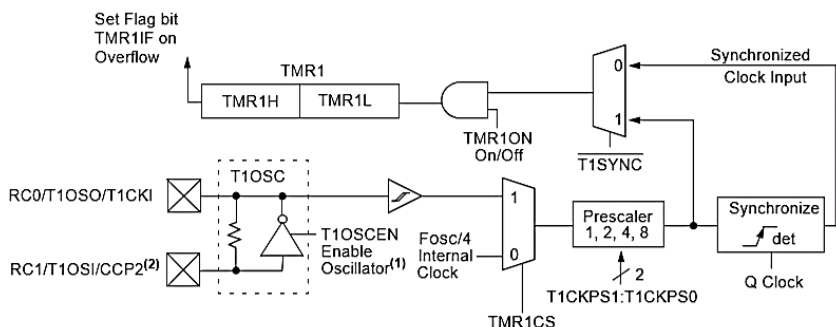


Рис. 9.2. Блок-схема модуля timer1 мікроконтролера PIC16F876A

Виводи RC0/T1OSO/T1CKI та RC1/T1OSI/CCP2 слугують для підключення зовнішнього кварцового генератора тактових імпульсів.

Регістр таймера TMR1 є 16-бітним і складається з двох 8-бітних регістрів TMR1H (старший байт) і TMR1L (молодший байт), які є доступні для читання та запису.

Модуль timer2

- 8-бітний таймер (рахує від 0 до значення в регістрі **PR2**);
- Працює тільки від внутрішнього тактового генератора;
- максимальний коефіцієнт подільника частоти імпульсів – 16;
- має вихідний 4-бітний подільник частоти переривань
- таймер працює по передньому фронту сигналу;
- переривання генерується за переповнення таймера (при переході від значення в регістрі **PR2** до 0);
- таймер працює постійно.

Цей таймер має інший принцип роботи. Таймер рахує імпульси від 0 до значення заданого в регістрі **PR2**. Після зрівняння регістра таймера **TMR2** і значення в регістрі **PR2** сигнал поступає на 4-бітний вихідний дільник для генерації переривання. Біт **TMR2IF** встановиться в “1” після певної кількості циклів таймера, яка задається коефіцієнтом вихідного подільника. Така система дозволяє генерувати імпульсні сигнали з малими частотами.

Таймер може посилати сигнали на модуль CCP (Capture/Compare/Pulse Wide Modulation) в якості базового таймера для широтно-імпульсної модуляції. Керування і налаштування модулем timer2 здійснюється регістром **T2CON**.

T2CON							
-	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS1
Біт 7							Біт 0

Біт 7 не реалізований.

Біт 6–3 **TOUTPS3:TOUTPS0**: Вибір коефіцієнта вихідного подільника.

0000 = 1:1;
 0001 = 1:2;

 1111 = 1:16.

Біт 2 **TMR2ON**: Включення модуля timer2.

1 = включений;
 0 = виключений.

Біт 1-0 **T2CKPS1:T2CKPS1**: Вибір коефіцієнта подільника частоти тактового генератора

00 = 1:1;
 01 = 1:4;
 1x = 1:16.

За умови переповнення таймера, тобто за переходу значення регістра **TMR2** від значення **PR2** до 0, генерується переривання і значення біта **TMR2IF** в регістрі **PIR1** встановлюється в “1” (повинен скидатися програмно). Дозвіл на переривання встановлюється бітом **TMR2IE** в “1” в регістрі **PIE1**.

Як для модулів timer0 і timer1 для генерації переривань від модуля timer2, необхідно встановити дозвіл на переривання бітом **GIE** регістра **INTCON**.

Блок-схема модуля timer2 приведена на рисунку 9.3.

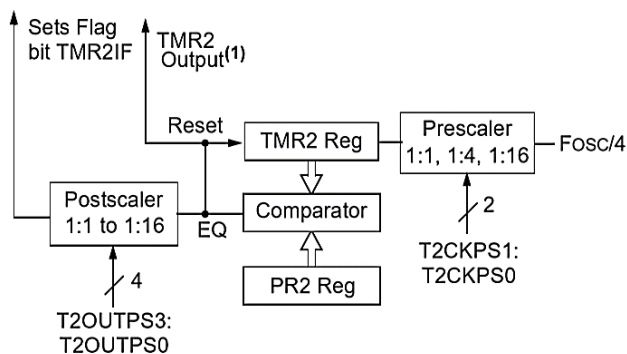


Рис. 9.3. Блок-схема модуля timer2 мікроконтролера PIC16F876A

Опис експерименту

Схема для вивчення роботи модуля timer0 в середовищі **proteus** приведено на рисунку 9.4.

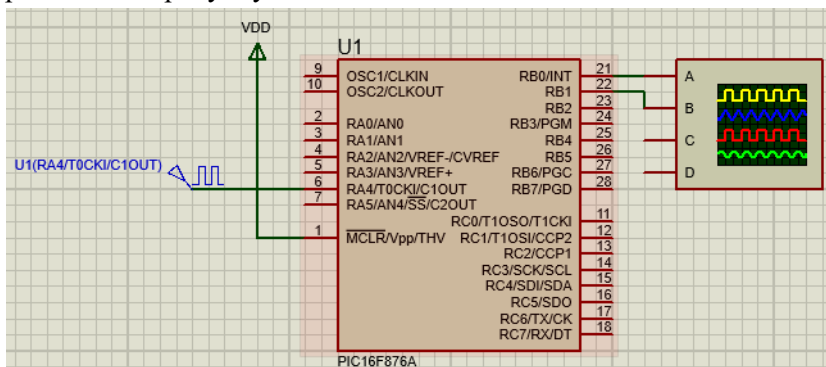


Рис. 9.4. Схема у середовищі Proteus для вивчення роботи модуля timer0 мікроконтролера PIC16F876A

За тактовий генератор для роботи мікроконтролера взято кварцовий резонатор частотою $F_{OSC} = 20$ МГц. Даний тактовий генератор також використовуватиметься як джерело тактових імпульсів для роботи модуля timer0. До виводу RA4 під'єднано зовнішній генератор тактової частоти для таймера, частота цього

генератора повинна бути меншою за частоту генератора від якого працює мікроконтролер.

Проведемо підрахунки параметрів таймера:

- $F_{OSC} = 20$ МГц (внутрішній тактовий генератор);
- $k = 256$ (вибираємо коефіцієнт подільника частоти тактового генератора);
- $N = 256$ (максимальне число значень регістра TMR0).

Проміжок часу, за який регістр TMR0 набуде значення 255, визначається за формулою:

$$T = \frac{4}{F_{OSC} \text{ МГц}} \cdot k \cdot N,$$
$$T = \frac{4}{20 \text{ МГц}} \cdot 256 \cdot 256 \approx 13,1 \text{ мс.}$$

Тобто через час 13,1 мс від початку рахунку таймера біт **TMR0IF** встановиться в “1”, про що можна відслідкувати програмно і за потребою користувача виконати певні дії (наприклад, поміняти стан на виводі RB0 на протилежний).

Код програми з використанням роботи модуля timer0 має вигляд:

```
//*****
// конфігураційні біти
#include <xc.h>
#define _XTAL_FREQ 20000000 // оголошення частоти
такового генератора
unsigned i, j, k; // оголошення потрібних змінних
void main (void)
{
    TRISB=0b00000000; // порт B на вивід
    PORTB=0b00000000; // очищення регістра порту B
    T0CS=0; // внутрішній генератор (20 МГц)
    T0SE=0; // приріс по передньому фронту при
// зовнішньому генераторі
```



```

PSA=0; // подільник частоти тактових
// імпульсів перед таймером TMR0
OPTION_REGbits.PS=0b111; // коефіцієнт подільника - 256
TMR0=0; // очищення регістра таймера
while (1)
{
    if (TMR0IF==1) // переповнення таймера TMR0
    {
        RB0=~RB0; // команди користувача
        TMR0IF=0; // обнулення прапорця переривання
        // інші команди користувача
    }
}
}
//*****

```

Даний код реалізовує генератор імпульсів певної частоти.

Код програми роботи модуля timer0 з використанням функції переривань має вигляд:

```

//*****
// конфігураційні біти
#include <xc.h>
#define _XTAL_FREQ 2000000 // оголошення частоти
// тактового генератора
int i, j, k; // оголошення потрібних змінних
// _____
void interrupt T0 (void) // функція опрацювання
// переривань
{
    RB0=~RB0; // команди користувача
    TMR0IF=0; // обнуляємо прапорець переривань
    // інші команди користувача
}
// _____

```

```

void main (void)
{
    TRISB=0b00000000;    // порт В налаштовуємо на вивід
    PORTB=0b00000000;    // очищення порту В
    T0CS=0;                // внутрішній тактовий генератор (20 МГц)
    T0SE=0;                // приріст таймера по передньому фронті імпульсу
                        // при зовнішньому генераторі
    PSA=0;                 // подільник тактових імпульсів
                        // перед таймером TMR0
    OPTION_REGbits.PS=0b111;    // коефіцієнт подільника
                        // частоти тактових імпульсів – 256
    GIE=1;                 // глобальний дозвіл на переривання
    TMR0IE=1;              // дозвіл на переривання від
                        // модуля timer0
    TMR0=0;                // очищення регістра таймера
    while (1)
    {
        RB1=~RB1;          // команди користувача
        __delay_ms (20);
        // інші команди користувача
    }
}
//*****

```

В результаті роботи цього коду на виводі RB1 генеруються імпульси тривалістю 20 мс відповідно до використаної часової затримки в тілі основної програми. Паралельно з тим на виводі RB0 генеруються імпульси тривалістю 13,1 мс, яка визначається налаштуванням модуля таймера.

Отже використання таймерів дає можливість із заданою періодичністю виконувати певні дії незалежно від ходу основної програми, де можуть використовуватися різні часові затримки.

Хід роботи

1. У середовищі Proteus складіть схему як на рисунку 9.4.
2. У середовищі MPLAB введіть вище приведений код з використанням роботи модуля timer0.
3. Проведіть компіляцію коду та перевірте його роботу у середовищі Proteus.
4. У середовищі MPLAB введіть код з використанням функції опрацювання переривань від модуля timer0, та перевірте його роботу в середовищі Proteus.

Завдання

1. Оформити програмний код роботи таймера з використанням зовнішнього тактового генератора.
2. Провести розрахунок і написати програмний код для генерування імпульсів тривалістю 0,1; 0,5 та 1 секунда.
3. Провести розрахунок та написати програмний код для генерації імпульсів частотою 1 МГц.

Контрольні питання

1. Які регістри налаштовують роботу модуля timer0?
2. Яка розрядність регістра TMR0?
3. За що відповідає біт TMR0IF?
4. Як коефіцієнт подільника частоти тактового генератора на роботу модуля timer0?

ЛАБОРАТОРНА РОБОТА № 10

Вивчення модуля ССР. Подільник частоти імпульсів

Мета роботи: Ознайомитись із регістрами та принципом роботи модуля ССР мікроконтролера PIC16F876A. Вивчити роботу модуля ССР в режимі захоплення імпульсів. Написати програмний код для реалізації подільника частоти імпульсів.

Теоретичні відомості

Абревіатура ССР утворена з перших букв слів Capture (Захоплення) Compare (Порівняння) Pulse-Wide Modulation (Широтно-Імпульсна Модуляція).

Модуль ССР призначений для розширення функціональних можливостей таймерів. З допомогою модуля ССР можна реалізовувати різні функціональні вузли цифрової електроніки, такі як: подільник частоти вхідних імпульсів, частотомір, цифровий компаратор а також реалізувати широтно-імпульсну модуляцію (ШІМ).

Модуль ССР може працювати в одному з трьох режимів: захоплення, порівняння, широтно-імпульсної модуляції.

Модуль ССР в різних режимах роботи використовує різні модулі таймерів:

Використання ресурсів модулем ССР	
Режим модуля ССР	Таймер
Захоплення	TMR1
Порівняння	TMR1
ШІМ	TMR2

В мікроконтролері PIC16F876A є два ідентичні модулі ССР1 і ССР2. Кожен модуль ССР містить 16-розрядний регістр, який може використовуватися в якості:

- 16-розрядного регістра захоплення даних;

- 16-розрядного регістра порівняння;
- Двох 8-розрядних (ведучий і підлеглий) регістрів ШІМ.

Робота модулів CCP1 і CCP2 ідентична за винятком функціонування тригера спеціальних подій.

Модуль CCP1:

Регістр CCPR1 модуля CCP складається з двох 8-розрядних регістрів: CCPR1L (молодший байт) і CCPR1H (старший байт).

У регістрі CCP1CON знаходяться біти керування модулем CCP1, доступні для запису і читання. В режимі порівняння тригер спеціальних подій скидає таймер TMR1.

Модуль CCP2:

Регістр CCPR2 модуля CCP складається з двох 8-розрядних регістрів: CCPR2L (молодший байт) і CCPR2H (старший байт). В регістрі CCP2CON знаходяться біти керування модулем CCP2, доступні для запису і читання. В режимі порівняння тригер спеціальних подій скидає таймер TMR1 і запускає перетворення АЦП (якщо АЦП включено).

Для налаштування модуля CCP використовується регістр **CCPxCON** ($x = 1; 2$).

CCPxCON							
-	-	CCPxX	CCPxY	CCPxM3	CCPxM2	CCPxM1	CCPxM0
Біт 7							Біт 0

Біт 7-6 **Не використовуються:** Читаються як “0”:

Біт 5-4 **CCPxX:CCPxY:** Молодші біти скважності ШІМ.

Режим захоплення імпульсів: не використовуються;

Режим порівняння: не використовуються;

Режим ШІМ: Два молодших біти скважності. Вісім старших бітів знаходяться в регістрі CCPRxL.

Біт 3-0 **CCPxM3:CCPxM0:** Режим роботи модуля CCPx.

0000 = Модуль CCPx виключений (скидання модуля CCPx);

- 0100 = захоплення по кожному задньому фронту сигналу;
 0101 = захоплення по кожному передньому фронту сигналу;
 0110 = захоплення по кожному 4-му передньому фронту сигналу;
 0111 = захоплення по кожному 16-му передньому фронту сигналу;
 1000 = Порівняння: встановлює вихідний сигнал (встановлюється прапорець CCPxIF в “1”);
 1001 = Порівняння: скидає вихідний сигнал (встановлюється прапорець CCPxIF в “1”);
 1010 = Порівняння: на вихідний сигнал не впливає (встановлюється прапорець CCPxIF в “1”);
 1011 = Порівняння: тригер спеціальних функцій (встановлюється прапорець CCPxIF в “1”; на вивід CCPx не впливає). CCP1 – скидання таймера TMR1. CCP2 – скидання таймера TMR1, запуск перетворення АЦП (якщо АЦП включено);
 11xx = ШІМ режим.

Блок-схема модуля CCP у режимі захоплення приведена на рисунку 10.1.

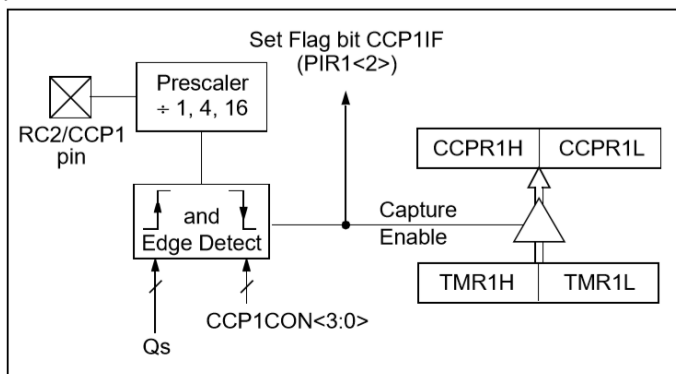


Рис. 10.1. Блок-схема модуля CCP в режимі захоплення імпульсів

Як видно з рисунка, вхідний сигнал з виводу RC2/CCP1 через подільник частоти запускає процес запису значення регістрів таймера TMR1H і TMR1L в регістри модуля CCP CCPR1H і CCPR1L відповідно.

Вивід RC2/CCP1 повинен бути налаштований на вхід виставленням біту TRISC<2> в “1”. У випадку використання зовнішнього тактового генератора з виводу RC1/T1OSI/CCP2 модуль таймера TMR1 повинен працювати в синхронізуючому режимі. В асинхронному режимі таймера TMR1 модуль CCP1 працювати не буде!

Коли змінюється режим роботи модуля CCP, необхідно забороняти переривання скиданням біта CCP1IE регістра PIR в “0” для запобігання спрацювань фальшивих переривань. Після зміни режиму роботи модуля CCP1, перед дозволом переривань необхідно скинути прапорець CCP1IF регістра PIR в “0”.

Опис експерименту

В даній роботі вивчається модуль CCP в режимі захоплення імпульсів. Цей режим дозволяє реалізувати схеми подільника частоти імпульсів та частотоміра періодичного сигналу. Схему для вивчення роботи модуля CCP в режимі захоплення приведено на рис. 10.2. Нехай частота імпульсів досліджуваного генератора дорівнює 25 кГц.

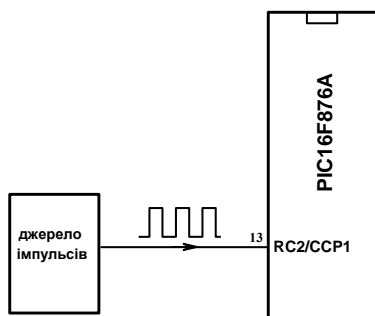


Рис. 10.2. Схема підключення зовнішнього генератора імпульсів до модуля CCP мікроконтролера PIC16F876A

При виникненні події захоплення, тобто коли на вивід RC2 мікроконтролера прийшов фронт імпульсу, 16-розрядне значення лічильника TMR1 заноситься в регістри CCP1L:CCP1H модуля CCP1. Подією захоплення може бути:

- Задній фронт імпульсу на вході RC2/CCP1
- Передній фронт імпульсу на вході RC2/CCP1
- Кожен 4-й передній фронт імпульсу на вході RC2/CCP1
- Кожен 16-й передній фронт імпульсу на вході RC2/CCP1

Тип події захоплення встановлюється бітами CCP1M3:CCP1M0 в регістрі CCP1CON.

Після виконання захоплення встановлюється прапорець переривання CCP1IF регістра PIR1 в “1”, який повинен скидатися в “0” програмно.

Розрахунок параметрів захоплення:

- Нехай маємо зовнішній генератор імпульсів частотою 25 кГц, під'єднаний до виводу RC2/CCP1 (цю частоту або період і будемо вимірювати).
- Модуль timer1, який є базовий для режиму захоплення модуля CCP1, нехай працює від внутрішнього кварцового генератора частотою 20 МГц. Коефіцієнт подільника виберемо $k = 1$.
- Модуль CCP налаштуємо на захоплення кожного імпульсу.

Робота модуля CCP у режимі захоплення виглядає таким чином. Після приходу фронту наростання імпульсу на вивід RC2 запускається модуль timer1, у момент приходу наступного імпульсу значення регістра таймера TMR1 записується в регістр CCP1 модуля CCP (рис. 10.3). Таким чином, значення регістра таймера прямо пропорційне до періоду надходження імпульсів, тобто за значенням таймера в момент приходу наступного імпульсу можна визначити період (або частоту) надходження імпульсів.

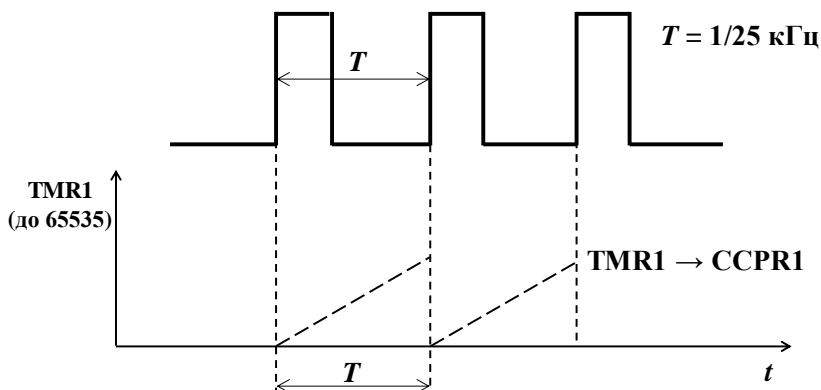


Рис. 10.3. Часова діаграма роботи модуля ССР в режимі захоплення імпульсів

Період T надходження імпульсів визначають за формулою:

$$T = \text{TRM1} \cdot T_0,$$

де TRM1 – значення регістра таймера в момент приходу чергового імпульсу, яке є задане в регістрі CCPR1, T_0 – період інкрементування таймера:

$$T_0 = \frac{4 \cdot k}{F_{\text{OSC}}} = \frac{4 \cdot 1}{20 \text{ МГц}} = 0,2 \text{ мкс},$$

$k = 1$ – коефіцієнт подільника таймера, F_{OSC} – частота тактового генератора (20 МГц). Знаючи, що період нашого досліджуваного періодичного сигналу:

$$T = \frac{1}{25 \text{ кГц}} = 40 \text{ мкс},$$

можна для самоперевірки взнати, до якого значення дорахує таймер. Регістр таймера набуде значення, яке дорівнює:

$$\text{TMR1} = \frac{T}{T_0} = \frac{40 \text{ мкс}}{0,2 \text{ мкс}} = 200.$$

Оскільки це значення є менше за 255, його можна візуалізувати в двійковій формі з допомогою світлодіодів під'єднаних до порту В (рис. 10.4).

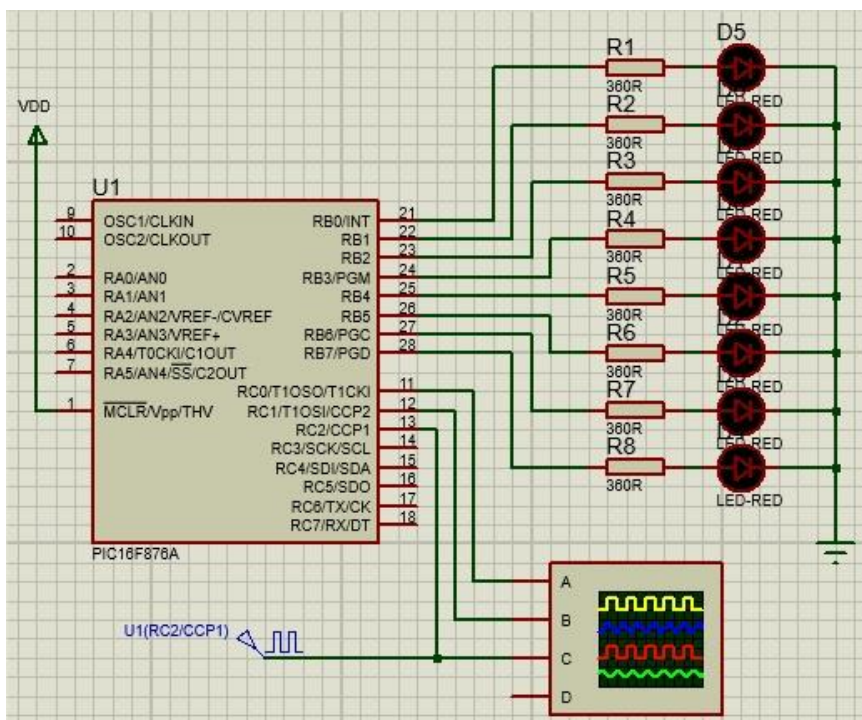


Рис. 10.4. Схема у середовищі proteus для вивчення роботи модуля CCP в режимі захоплення імпульсів

Код програми роботи модуля CCP в режимі захоплення має вигляд:

```
//*****
void interrupt Capture (void) // функція опрацювання переривань
{
    PORTB=CCPR1L; // в порт B посилаємо значення регістра
                  // CCPR1L (молодший байт)
    TMR1H=0;      // обнуляємо таймер (старший байт)
    TMR1L=25;     // для корекції таймера початкове значення
                  // TMR1L підібрано 25
    CCP1IF=0;     // обнуляємо прапорець переривання модуля CCP1
}
```

```
//
void main (void)
{
    TRISC=0b00000100; // вивід RC2 налаштовуємо на ввід
    TRISB=0b00000000; // порт B налаштовуємо на вихід
    GIE=1; // глобальний дозвіл переривань
    PEIE=1; // дозвіл переривань від периферійних
           // модулів
    CCP1IE=1; // дозвіл на переривання модулем CCP1
    T1CONbits.T1CKPS=0b00; // коефіцієнт подільника таймера – 1
    T1OSCEN=0; // вмикаємо RC генератор таймера
    TMR1CS=0; // внутрішній генератор  $F_{OSC} = F_{XTAL}/4$ 
    TMR1ON=0; // вмикаємо модуль timer1
    TMR1H=0; // обнуляємо регістр таймера
           // ( старший байт )
    TMR1L=0; // обнуляємо регістр таймера
           // ( молодший байт )
    CCP1CONbits.CCP1M=0b0101; // режим захоплення кожного
           // імпульсу
    CCP1IF=0; // занулюємо прапорець переривань від
           // модуля CCP1
    TMR1ON=1; // вмикаємо модуль timer1
    while (1)
    {
        RC0=~RC0; // команди користувача
        __delay_ms (10);
    }
}
//*****
```

Даний програмний код реалізує візуалізацію значення таймера в момент захоплення кожного імпульсу.

Хід роботи

1. У середовищі Proteus складіть схему приведену на рисунку 10.4.
2. В середовищі MPLAB введіть вище приведений програмний код роботи модуля ССР в режимі захоплення.
3. Проведіть компіляцію коду та перевірте його роботу в середовищі Proteus.
4. Дослідіть роботу модуля ССР за різних типів захоплення імпульсів.

Завдання

1. Розробити алгоритм подільника частоти, використовуючи модуль ССР.
2. Провести розрахунок і написати програмний код для вимірювання періоду надходження імпульсів.

Контрольні питання

1. Опишіть роботу модуля ССР в режимі захоплення імпульсів?
2. Які є типи захоплення імпульсів у модулі ССР?
3. Який модуль таймерів є базовим для роботи модуля ССР в режимі захоплення імпульсів?

ЛАБОРАТОРНА РОБОТА № 11

Вивчення модуля ССР. Цифровий компаратор. Частотна модуляція

Мета роботи: Ознайомитись з роботою модуля ССР в режимі порівняння (Compare). Вивчити алгоритм генерації імпульсних сигналів різної частоти.

Теоретичні відомості

Основні особливості і реєстри керування модулем ССР описані в лабораторній роботі № 10.

Робота модуля в режимі порівняння є полягає в наступному. 16-розрядний реєстр ССР1 порівнюється зі значенням реєстра ТМР1. Як тільки значення в реєстрах стають однаковими, модуль ССР1, в залежності від його налаштувань, може виконувати такі дії:

- встановлює високий рівень сигналу на виводі RC2/ССР1;
- встановлює низький рівень сигналу на виводі RC2/ССР1;
- на вивід RC2/ССР1 не впливає.

Дія при співпадінні значень реєстрів може бути вибрана бітами ССР1М3:ССР1М0 в реєстрі ССР1CON.

У момент зміни стану виводу RC2/ССР1 встановлюється прапорець переривання ССР1IF реєстра РІР1 в “1”, який повинен скидатися в “0” програмно.

Блок-схема модуля ССР у режимі порівняння приведена на рисунку 11.1.

Основні особливості роботи модуля ССР у режимі порівняння:

- Вивід RC2/ССР1 повинен бути налаштований на вихід.
- Зауваження: за умови очищення реєстра ССР1CON на виводі RC2/ССР1 з'явиться сигнал низького рівня, що не є результатом порівняння або даними з вихідної засувки реєстра PORTC.
- У випадку використання зовнішнього тактового генератора з виводу RC1/Т1ОSІ/ССР2, таймер ТМР1 повинен працювати в

синхронізованому режимі. В асинхронному режимі TMR1 модуль CCP1 працювати не буде!

- Програмна зміна рівня сигналу на виводі RC2/CCP1 не викликає генерацію переривання. Переривання генеруються тільки модулем CCP1.
- В режимі порівняння модуля CCP1 може бути включений тригер спеціальних подій. Тригер спеціальних подій CCP1 скидає значення таймера TMR1 при кожному позитивному результаті порівняння. Регістр CCPR1 є 16-розрядним програмним регістром періоду для TMR1. Тригер спеціальних подій модуля CCP2 скидає значення таймера TMR1 і запускає перетворення АЦП (якщо АЦП включено). Тригер спеціальних подій модулів CCP1 і CCP2 не встановлює прапорця TMR1IF в "1".

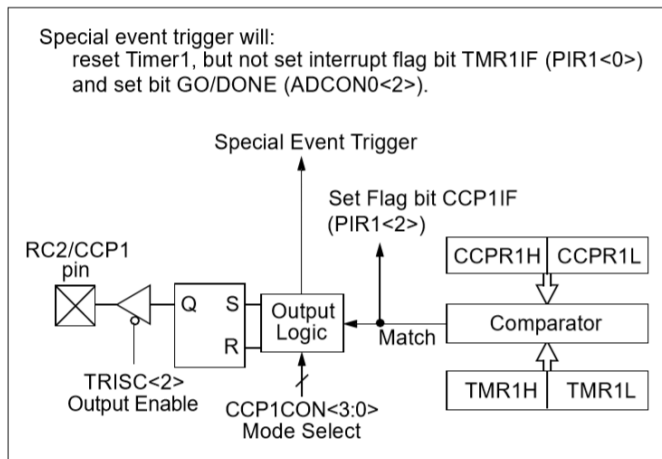


Рис. 11.1. Блок-схема модуля ССР у режимі порівняння

Опис експерименту

Схема установки для вивчення роботи модуля ССР у режимі порівняння приведена на рисунку 11.2.

Як видно з рисунка, вивід мікроконтролера RC2 налаштований на вихід, під'єднаний до осцилографа. Мікроконтролер працює від кварцового резонатора частотою 20 МГц.

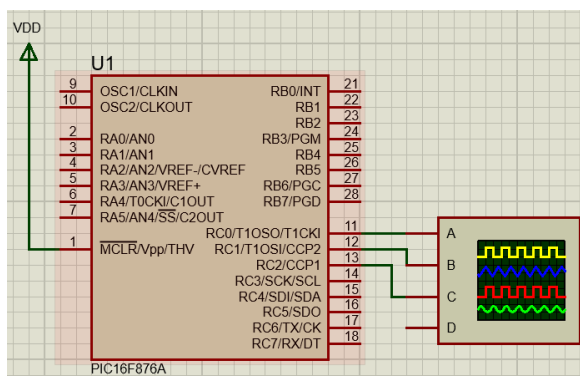


Рис. 11.2. Схема у середовищі proteus для вивчення роботи модуля CCP у режимі порівняння

Розрахунок параметрів порівняння:

Час T , за який таймер дорахує до значення CCPR1, визначається за формулою:

$$T = \text{CCPR1} \cdot T_0,$$

де CCPR1 – значення, до якого рахує таймер, T_0 – час за яких значення таймера збільшується на одиницю. Оскільки мікроконтролер працює від кварцового резонатора частотою $F_{\text{OSC}} = 20 \text{ МГц}$ з коефіцієнтом подільника $k = 1$, то

$$T_0 = \frac{4 \cdot k}{F_{\text{OSC}}} = \frac{4 \cdot 1}{20 \text{ МГц}} = 0,2 \text{ мкс.}$$

Задавши значення в регістр CCPR1 = 40000, обчислимо час, за який таймер досягне значення 40000:

$$T = T_0 \cdot \text{CCPR1} = 0,2 \text{ мкс} \cdot 40000 = 8 \text{ мс.}$$

Через цей час стан на виводі RC2 змінюється, в залежності від налаштувань модуля CCP, на високий або низький логічний рівень. Таким чином можна одержати генератор імпульсів заданої частоти. Змінюючи програмним способом значення в регістрі CCPR1, до якого дораховує таймер, можна регулювати частоту вихідного сигналу (частотна модуляція).

Програмний код роботи модуля CCP у режимі порівняння має вигляд:

```

//*****
void interrupt Compare (void)
{
    CCP1R1=40000;      // задаємо значення регістра CCP1R1
    TMR1H=0;           // очищаємо регістр таймера
                        // ( старший байт )
    TMR1L=0;           // очищаємо регістр таймера
                        // ( молодший байт )
    CCP1M0=~CCP1M0;    // змінюємо тип дії модуля за умови
                        // порівняння регістрів
    CCP1IF=0;          // скидаємо в "0" прапорець переривання
                        // від модуля CCP1
}
//_____
void main (void)
{
    TRISC=0b00000000;  // вивід RC2 налаштовуємо на вихід
    TRISB=0b00000000;  // порт B налаштовуємо на вихід
    GIE=1;              // глобальний дозвіл переривань
    PEIE=1;             // дозвіл переривань від периферійних
                        // модулів
    CCP1IE=1;           // дозвіл на переривання модулем CCP1
    T1CONbits.T1CKPS=0b00; // коефіцієнт подільника таймера – 1
    T1OSCEN=0;          // RC генератор таймера вмикаємо
    TMR1CS=0;           // вибираємо внутрішній тактовий
                        // генератор  $F_{OSC} = F_{XTAL}/4$ 
    TMR1ON=0;           // вмикаємо модуль timer1
    CCP1R1H=0;          // очищаємо регістр CCP1R1 (старший байт)
    CCP1R1L=0;          // очищаємо регістр CCP1R1 (молодший байт)
    CCP1CONbits.CCP1M=0b1000; // режим порівняння
    CCP1IF=0;           // занулюємо прапорець
                        // переривань від модуля CCP1
    TMR1ON=1;           // вмикаємо модуль timer1
}

```



```

while (1)
{
    RC0=~RC0;          // команди користувача
    __delay_ms (10);
}
}
//*****

```

Даний програмний код реалізує генерацію імпульсів тривалістю 8,03 мс на виводі RC2 мікроконтролера. Період надходження імпульсів відповідно дорівнює 2·8,03 мс.

Хід роботи

1. У середовищі Proteus складіть схему приведену на рис. 11.2.
2. У середовищі MPLAB введіть вище приведений код роботи модуля CCP у режимі порівняння.
3. Проведіть компіляцію коду і перевірте його роботу в середовищі Proteus.
4. Дослідіть роботу модуля CCP, змінюючи значення регістра CCPR1.

Завдання

1. Провести розрахунки та розробити алгоритм генерації імпульсів з частотою 1 МГц.
2. Провести розрахунки та розробити алгоритм генерації імпульсів зі зміною частоти від 0 до 1 кГц за 1с.

Контрольні питання

1. Опишіть роботу модуля CCP в режимі порівняння?
2. Що таке частотна модуляція?
3. Що таке цифровий компаратор?
4. Яка розрядність регістра CCPR1 у мікроконтролері PIC6A876A?
5. Скільки модулів CCP є у мікроконтролері PIC6A876A?

ЛАБОРАТОРНА РОБОТА № 12

Вивчення модуля ССР. Широтно-імпульсна модуляція (ШІМ)

Мета роботи: Ознайомитись із роботою модуля ССР у режимі PWM (широтно-імпульсної модуляції). Вивчити алгоритм генерації імпульсів заданої частоти та скважності. Вивчити алгоритм реалізації широтно-імпульсної модуляції. Оформити код для генерації імпульсів заданої частоти та тривалості імпульсу.

Теоретичні відомості

Один із режимів роботи модуля ССР мікроконтролера PIC16F876A є режим ШІМ (широтно-імпульсна модуляція). Основні особливості і регістри налаштування модуля ССР описані в лабораторній роботі №10.

Одним із способів представлення інформації з допомогою електричних сигналів є генерування електричних імпульсів певної тривалості, за сталої частоти імпульсів (або періоду, який називається періодом ШІМ) (рис. 12.1). Тривалість імпульсу, в даному випадку, несе певну інформацію.

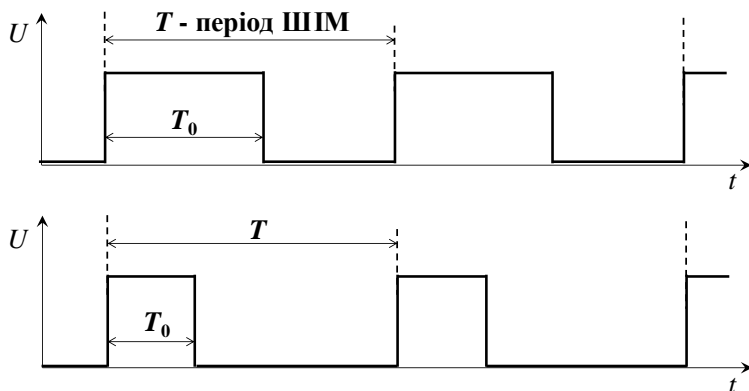


Рис. 12.1. Періодичні сигнали з однаковим періодом і різною тривалістю імпульсу

Отже, процес зміни ширини імпульсів за сталої їх частоти називається широтно-імпульсною модуляцією. Для характеристики ШІМ вводять такі поняття як S – скважність та D – коефіцієнт заповнення. Скважність – це відношення періоду надходження імпульсів до тривалості імпульсу:

$$S = \frac{T}{T_0}.$$

Коефіцієнт заповнення – це величина обернена до скважності:

$$D = \frac{T_0}{T}.$$

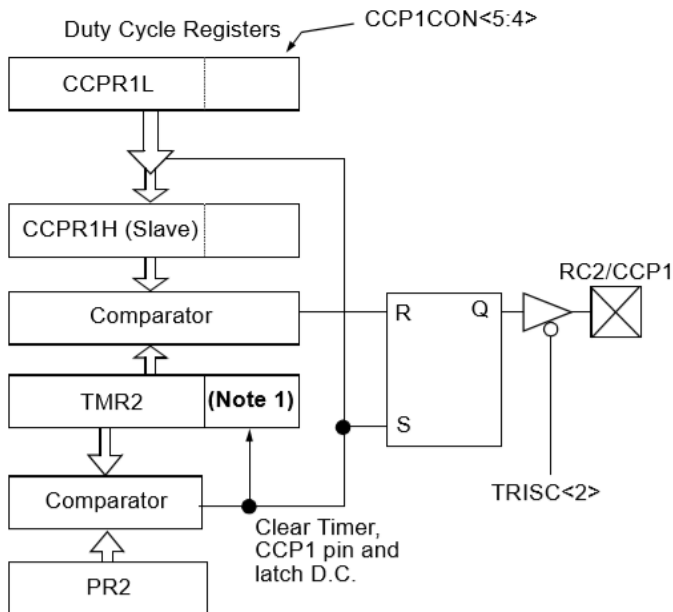
Іноді коефіцієнт заповнення виражають у відсотках, що є зручно представляти ширину імпульсу відносно періоду імпульсів.

З допомогою модуля ССР мікроконтролерів PIC можна легко генерувати імпульси заданої частоти та скважності. Структурна схема модуля ССР у режимі ШІМ приведена на рисунку 12.2.

Робота модуля ССР у режимі ШІМ полягає в наступному. Після запуску таймера, значення його регістра TMR2 порівнюється зі значенням в регістрі CCPR1H. Як тільки значення таймера досягне значення CCPR1H, то на виводі RC2 встановиться низький логічний рівень напруги, тобто логічний “0”. Паралельно цьому, значення регістра TMR2 таймера порівнюється зі значенням в регістрі PR2 і як тільки значення таймера зрівняється зі значенням PR2, на виводі RC2 встановлюється високий логічний рівень напруги, тобто логічна “1”. Роботу модуля ССР у режимі ШІМ ілюструє часова діаграма (рис. 12.3).

У момент запуску таймера на виводі RC2 встановлюється високий логічний рівень напруги (точка А на осі часу). В міру зростання значення таймера до значення яке занесене в регістр CCPR1, на виводі RC2 встановлюється низький логічний рівень напруги (точка В на осі часу). Подальше зростання значення таймера до значення, яке занесене в регістр PR2, на виводі RC2 знову встановлюється високий логічний рівень, і таймер починає рахувати з початку (точка С на осі часу). Таким чином, лише двома

параметрами, значеннями в регістрах PR2 і CCPR1, можна генерувати періодичний сигнал заданої частоти і скважності.



Note 1: The 8-bit timer is concatenated with 2-bit internal Q clock, or 2 bits of the prescaler, to create 10-bit time base.

Рис. 12.2. Структурна схема модуля CCP в режимі ШІМ

Період ШІМ визначається значенням регістра PR2, який є в модулі TMR2 і може бути визначений за формулою:

$$T = [(PR2) + 1] \cdot 4 \cdot T_{osc} \cdot k,$$

де T – період ШІМ, T_{osc} – період тактових імпульсів таймера, k – коефіцієнт подільника частоти таймера TMR2.

Тривалість імпульсу можна визначити за формулою:

$$T_0 = [CCPR1] \cdot T_{osc} \cdot k,$$

де T_0 – тривалість імпульсу, – період тактових імпульсів;
 k – коефіцієнт подільника частоти таймера TMR2.

10-бітне значення регістра CCPR1 складається з старших 8 біт регістра CCPR1L і двох біт регістра CCP1CON <5:4> (рис. 12.2).

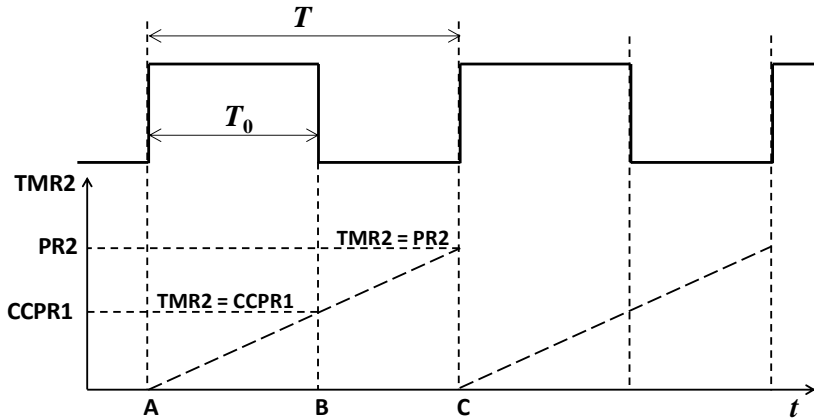


Рис. 12.3. Часова діаграма роботи модуля CCP у режимі ШІМ

Опис експерименту

Схема для вивчення роботи модуля CP у режимі ШІМ приведена на рисунку 12.4.

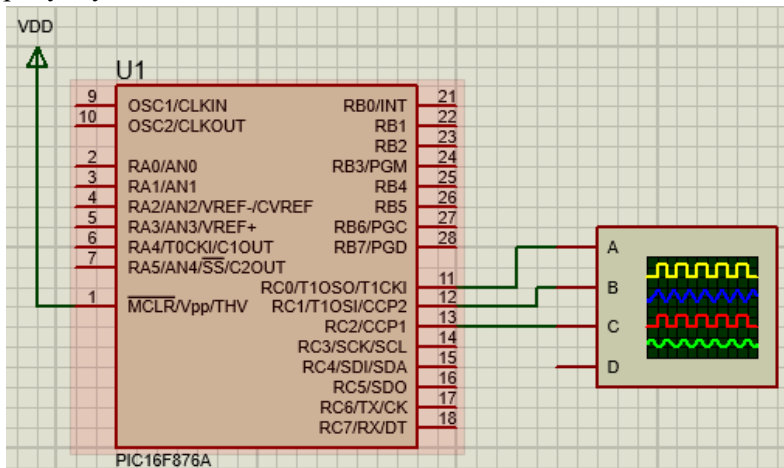


Рис. 12.4. Схема у середовищі proteus для вивчення роботи модуля CCP в режимі ШІМ

За формулою

$$T = \left[(PR2) + 1 \right] \cdot 4 \cdot T_{osc} \cdot k$$

знайдемо мінімальний та максимальний періоди ШІМ. Оскільки регістр PR2 8-бітний, його значення може бути від 0 до 255, $T_{osc} = 1/20 \text{ МГц} = 0,05 \text{ мкс}$ – період тактового генератора, k – коефіцієнт подільника частоти тактового генератора таймера timer2 мінімальне значення якого – 1, максимальне – 16. Отже мінімальний період ШІМ буде:

$$T_{min} = \left[(0) + 1 \right] \cdot 4 \cdot 0,05 \text{ мкс} \cdot 1 = 0,2 \text{ мкс}.$$

Максимальний період ШІМ буде:

$$T_{max} = \left[(255) + 1 \right] \cdot 4 \cdot 0,05 \text{ мкс} \cdot 16 = 819,2 \text{ мкс}.$$

Очевидним є те, що тривалість імпульсу ШІМ буде в межах від 0,2 до 819,2 мкс, що визначається значенням регістра CCP1L.

Послідовність налаштування модуля CCP у режимі ШІМ

- Встановити період ШІМ в регістрі PR2;
- Встановити тривалість імпульсу в регістрах CCP1L і CCP1CON <5:4>;
- Налаштувати вивід RC2/CCP1 як вихід, скинувши біт TRISC<2> в “0”;
- Налаштувати подільник і включити TMR2 в регістрі T2CON;
- Включити CCP1 в режимі ШІМ.

Програмний код для роботи модуля CCP у режимі ШІМ має вигляд:

```
//*****  
void interrupt PWM (void) // функція опрацювання переривань  
{  
    if (TMR2IF==1)  
    {  
        CCP1L=100;           // задаємо тривалість імпульсу  
        TMR2IF=0;           // скидаємо прапорець переривання таймера  
    }  
}
```

```

    }
}
// _____
void main (void)
{
    TRISC=0b00000000;    // вивід RC2 налаштовуємо на вихід
    TRISB=0b00000000;    // порт B налаштовуємо на вихід
    GIE=1;                // глобальний дозвіл переривань
    PEIE=1;                // дозвіл переривань від периферійних
                        // модулів
    TMR2IE=1;              // дозвіл переривань від модуля timer2
    TMR2=0;                // обнуляємо регістр таймера
    PR2=255;               // задаємо період імпульсів ШИМ
    CCP1L=100;             // тривалість імпульсу (старший байт)
    CCP1X=1;               // молодший біт тривалості імпульсу
    CCP1Y=1;               // молодший біт тривалості імпульсу
    T2CONbits.TOUTPS=0b1111; // коефіцієнт вихідного подільника
                        // таймера – 16
    T2CONbits.T2CKPS=0b11;  // подільник частоти таймера – 16
    TMR2ON=1;               // включаємо таймер
    CCP1CONbits.CCP1M=0b1111; // модуль CCP1 в режимі ШИМ
    while (1)
    {
        RC0=~RC0;           // команди користувача
        __delay_ms (2);
    }
}
//*****

```

Біти TOUTPS0:TOUTPS3 регістра T2CON задають коефіцієнт вихідного подільника генерації переривання від таймера, яким можна задавати швидкість зміни скважності ШИМ.

Хід роботи

1. У середовищі Proteus складіть схему приведену на рис.12.4.
2. У середовищі MPLAB введіть вище приведений програмний код роботи модуля ССР у режимі ШІМ.
3. Проведіть компіляцію коду та перевірте його роботу у середовищі Proteus.
4. Проведіть розрахунки для генерації імпульсів періодом 600 мкс і тривалістю імпульсу 200 мкс.
5. Дослідіть роботу модуля ССР, змінюючи значення регістра CCPR1L.

Завдання:

1. Провести розрахунки та розробити алгоритм генерації змінної тривалості імпульсів від мінімального значення до максимального за 1 с.
2. Провести розрахунки та розробити алгоритм генерації змінної тривалості імпульсів від максимального значення до мінімального за 2 с.

Контрольні питання

1. Що таке широтно-імпульсна модуляція?
2. Який регістр модуля ССР налаштовує його роботу у режим ШІМ?
3. Як задати тривалість імпульсу ШІМ?
4. Як задати період імпульсів ШІМ?

ЛАБОРАТОРНА РОБОТА № 13

Вивчення модуля ADC. Вимірювання напруги з допомогою модуля аналого-цифрового перетворювача

Мета роботи: Ознайомитись із роботою та регістрами налаштування модуля ADC мікроконтролера PIC16F876A. Вивчити порядок дій при роботі з модулем ADC. Провести вимірювання напруги та візуалізувати значення в двійковій формі з допомогою світлодіодів.

Теоретичні відомості

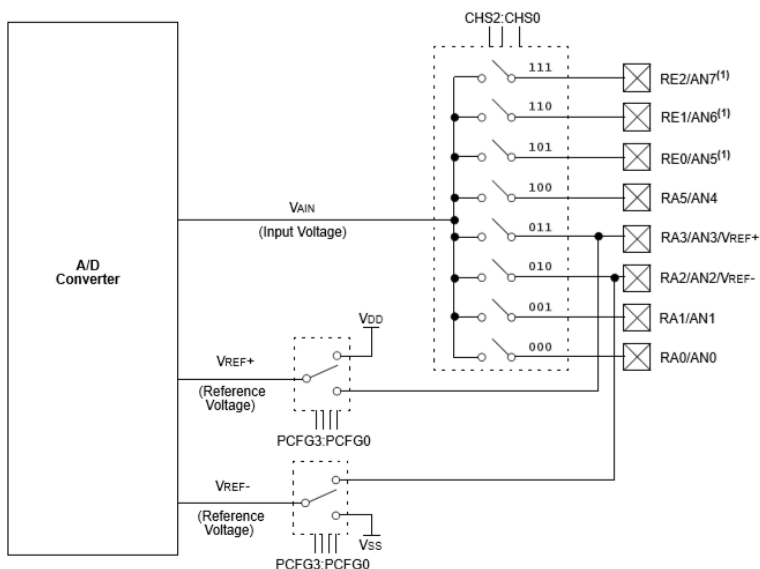
Модуль ADC (Analog Digital Converter) – аналого-цифровий перетворювач (АЦП). Аналого-цифрове перетворення в мікроконтролерах даного сімейства відбувається методом послідовного зрівноваження. Джерело верхньої і нижньої опорної напруги може бути програмно вибране з виводів V_{DD} , V_{SS} , RA2 або RA3.

Модуль АЦП має 5 каналів для 28 вивідних і 8 для 40/44 вивідних мікроконтролерів, які мультиплексно під'єднуються до входу АЦП (рис. 13.1)

У мікроконтролері PIC16F876A канали RE0, RE1, RE2 не реалізовані, оскільки він є 28 вивідним.

Особливості аналогового входу модуля ADC приведено на рисунку 13.2.

Як видно з рисунка, вхідна напруга відповідно вибраним каналом AN_x через електронний ключ SS заряджає конденсатор C_{HOLD} ємністю 120 пФ, і модуль ADC перетворює цю напругу в двійковий код. Для прецизійних вимірювань вхідної напруги потрібно також враховувати опір ключа, опір з'єднання елементів R_{IC} та втрати струму I_{LEAKAG} .



Note 1: Not available on 28-pin devices.

Рис. 13.1. Схема вхідної ланки модуля АЦП для 40/44 вивідних мікроконтролерів PIC

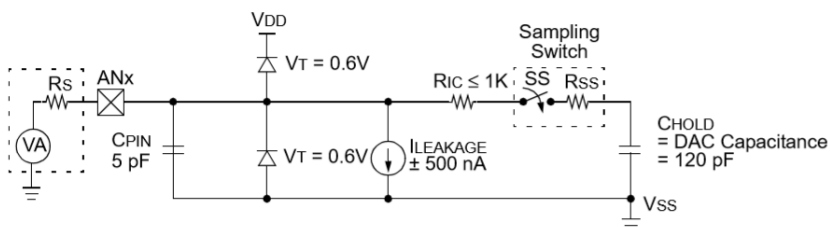


Рис. 13.2 Схема вхідної ланки модуля АЦП

- C_{PIN} – вхідна ємність
- V_T – порогова напруга
- $I_{LEAKAGE}$ – струм стоку виводу
- R_{IC} – опір з'єднання
- SS – переключатель
- R_{SS} – опір переключателя
- R_S – опір джерела аналогового сигналу
- C_{HOLD} – конденсатор

Для керування модулем АЦП у мікроконтролері PIC16F876A використовуються 4 регістри:

ADRESH – регістр результату АЦП (старший байт);

ADRESL – регістр результату АЦП (молодший байт);

ADCON0 – регістр налаштування модуля ADC;

ADCON1 – регістр налаштування модуля ADC.

Регістр керування ADCON0 використовують для налаштування роботи модуля АЦП, а з допомогою регістра ADCON1 підключають відповідний канал мікроконтролера до модулем АЦП, а також вибирають режим роботи виводів порту А (аналоговий вхід або цифровий порт вводу/виводу).

Регістр ADCON0 містить наступні біти:

ADCON0							
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/- DONE	-	ADON
Біт 7							Біт 0

Біт 7-6 **ADCS1:ADCS0:** Вибір джерела тактового сигналу.

00 = $F_{OSC}/2$;

01 = $F_{OSC}/8$;

10 = $F_{OSC}/32$;

11 = F_{RC} (внутрішній RC генератор модуля АЦП).

Біт 5-3 **CHS2:CHS0:** Вибір аналогового каналу.

000 = канал 0, (RA0/AN0);

001 = канал 1, (RA1/AN1);

010 = канал 2, (RA2/AN2);

011 = канал 3, (RA3/AN3);

100 = канал 4, (RA5/AN5);

101 = канал 5, (RE0/AN5)⁽¹⁾;

110 = канал 6, (RE1/AN6)⁽¹⁾;

111 = канал 7, (RE2/AN7)⁽¹⁾.

Біт 2 **GO/-DONE:** Біт статусу модуля АЦП.

Якщо ADON = 1:

1 = модуль АЦП виконує перетворення (установка біта викликає початок перетворення);

0 = стан очікування (апаратно скидається по завершенні перетворення).

Біт 1 **Не використовується:** (читається як “0”).

Біт 0 **ADON:** Біт включення модуля АЦП.

1 = модуль АЦП включений;

0 = модуль АЦП виключений і не споживає струму.

Примітка⁽¹⁾: Ці канали не реалізовані в мікроконтролерах PIC16F873/PIC16F876.

Регістр **ADCON1** містить наступні біти:

ADCON1							
ADFM	ADCS2	-	-	PCFG3	PCFG2	PCFG1	PCFG0
Біт 7							Біт 0

Біт 7 **ADFM:** Формат збереження 10-розрядного результату.

1 = праве вирівнювання, 6 старших біт ADRESH читаються як “0”;

0 = ліве вирівнювання, 6 молодших біт ADRESL читаються як “0”.

Біт 6-4 **Не використовується:** (читається як “0”).

Біт 3-0 **PCFG3:PCFG0:** Біти налаштування каналів АЦП.

Для мікроконтролера PIC16F876A біти **PCFG3:PCFG0** задають наступні параметри для виводів порту А (табл. 13.1).

Таблиця 13.1

PCFG3: PCFG0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	V _{REF+}	V _{REF-}
0000	A	A	A	A	A	V _{DD}	V _{SS}
0001	A	V _{REF+}	A	A	A	RA3	V _{SS}
0010	A	A	A	A	A	V _{DD}	V _{SS}

0011	A	V_{REF+}	A	A	A	RA3	V_{SS}
0100	D	A	D	A	A	V_{DD}	V_{SS}
0101	D	V_{REF+}	D	A	A	RA3	V_{SS}
011x	D	D	D	D	D	V_{DD}	V_{SS}
1000	A	V_{REF+}	V_{REF-}	A	A	RA3	RA2
1001	A	A	A	A	A	V_{DD}	V_{SS}
1010	A	V_{REF+}	A	A	A	RA3	V_{SS}
1011	A	V_{REF+}	V_{REF-}	A	A	RA3	RA2
1100	A	V_{REF+}	V_{REF-}	A	A	RA3	RA2
1101	D	V_{REF+}	V_{REF-}	A	A	RA3	RA2
1110	D	D	D	D	A	V_{DD}	V_{SS}
1111	D	V_{REF+}	V_{REF-}	D	A	RA3	RA2

A – аналоговий вхід;

D – цифровий канал вводу/виводу;

V_{REF+} – вхід опорної напруги для роботи АЦП (+5 В);

V_{REF-} – вхід опорної напруги для роботи АЦП (“земля”).

Після завершення перетворення результат перетворення записується в регістри ADRESH і ADRESL, після чого скидається прапорець GO/-DONE = “0” (ADCON0<2>) і встановлюється прапорець переривання ADIF = “1”.

Після включення АЦП і вибору каналів перед початком включення перетворення необхідно витримати певну часову паузу.

Розрахунок часової затримки

T_{ACQ} = час затримки підсилювача + час заряджання конденсатора
 C_{HOLD} + Температурний коефіцієнт.

$$\begin{aligned}
T_{ACQ} &= T_{AMP} + T_C + T_{COFF} = \\
&= 2 \text{ мкс} + T_C + [(\text{Температура} - 25^{\circ}\text{C}) \cdot (0,05 \text{ мкс}/^{\circ}\text{C})]; \\
T_C &= -C_{HOLD} (R_{IC} + R_{SS} + R_S) \ln(1/2047) = \\
&= -120 \text{ пФ} (1 \text{ кОм} + 7 \text{ кОм} + 10 \text{ кОм}) \cdot \ln (0,0004885) = \\
&= 16,47 \text{ мкс}; \\
T_{ACQ} &= 2 \text{ мкс} + 16,47 \text{ мкс} + [(50^{\circ}\text{C} - 25^{\circ}\text{C}) \cdot (0,05 \text{ мкс}/^{\circ}\text{C})] = \\
&= 19,72 \text{ мкс}.
\end{aligned}$$

Рекомендований порядок дій при роботі з АЦП

1. Налаштувати модуль АЦП:

- налаштувати виводи порту А як аналогові входи, входи V_{REF} або цифрові канали вводу/виводу з допомогою бітів регістра ADCON1;
- вибрати вхідний канал АЦП з допомогою регістра ADCON0;
- вибрати джерело тактових імпульсів для АЦП з допомогою регістра ADCON0;
- включити модуль АЦП.

2. Налаштувати переривання від модуля АЦП (якщо необхідно):

- скинути біт ADIF в “0”;
- встановити біт ADIE в “1”;
- встановити біт PEIE в “1”;
- встановити біт GIE в “1”;

3. Витримати паузу необхідну для зарядки конденсатора C_{HOLD} .

4. Почати аналого-цифрове перетворення:

- встановити біт GO/-DONE в “1”.

5. Очікувати завершення перетворення:

- очікувати поки біт GO/-DONE не буде скинутий в “0”, або
- очікувати переривання по завершенні перетворення.

6. Зчитати результат перетворення з регістрів ADRESH: ADRESL, скинути біт ADIF в “0”, якщо це необхідно.

7. Для наступного перетворення необхідно виконати кроки, починаючи з пункту 1 або 2. Час перетворення одного біта

визначається як час T_{AD} . Мінімальний час перед наступним перетворення повинен складати не менше $2T_{AD}$.

Час отримання одного біта результату визначається параметром T_{AD} .

Для 10-розрядного результату потрібно як мінімум $12T_{AD}$. Параметри тактового сигналу задають програмно, T_{AD} може приймати значення:

- $2T_{OSC}$;
- $8T_{OSC}$;
- $32T_{OSC}$;
- RC – внутрішній RC генератор модуля АЦП (2-6 мкс)

Для отримання коректного значення перетворення необхідно вибрати джерело тактового сигналу, для якого виконується умова $T_{AD} \geq 1,6$ мкс.

Опис експерименту

Схема макетної плати для вивчення роботи модуля ADC приведена на рисунку 13.3.

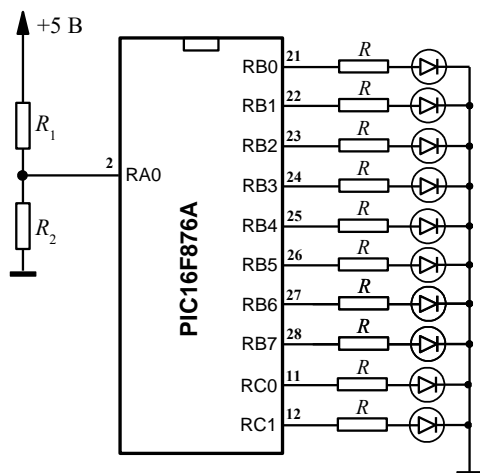


Рис. 13.3. Схема експериментальної макетної плати для візуалізації результату аналого-цифрового перетворення

Напруга з подільника на резисторах R_1 і R_2 подається до виводу RA0 мікроконтролера PIC16F876A. За опорну напругу для роботи модуля вибрано напругу живлення мікроконтролера – 5 В. Резисторами R_1 і R_2 можна регулювати напругу, яка подається на вхід АЦП. 10-бітний результат перетворення можна візуалізувати у вигляді двійкового коду з допомогою світлодіодів, підключених до виводів порту В та С (рис.13.3). Тобто результатом перетворення АЦП буде число:

$$N = \frac{U_{ex} \cdot 2^{10}}{5 B} = \frac{U_{ex} \cdot 1024}{5 B}, \quad (13.1)$$

яке буде візуалізовано у двійковій формі з допомогою світлодіодів.

Схема для вивчення роботи модуля ADC у середовищі симулятора Proteus приведена на рисунку 13.4.

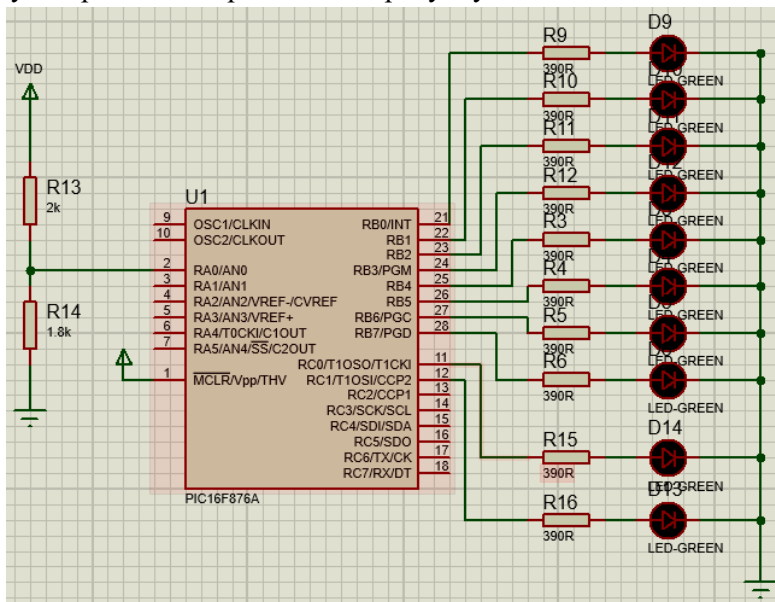


Рис.13.4. Схема у симуляторі Proteus для вивчення роботи модуля АЦП

Код програми, згідно з рекомендованим порядком дій, для роботи модуля АЦП має вигляд:


```

//*****
void interrupt ADC ()
{
    L=ADRESL;    // зчитуємо результат перетворення (молодший байт)
    H=ADRESH;    // зчитуємо результат перетворення (старший байт)
    PORTB=L;     // виводимо результат у порт B
    PORTC=H;     // виводимо результат у порт C
    ADIF=0;      // скидаємо прапорець переривання
    // інші команди користувача
}
//_____
void main (void)
{
    TRISA=0b11111111;    // порт A налаштовуємо на вхід
    TRISB=0b00000000;    // порт B налаштовуємо на вихід
    TRISC=0b00000000;    // порт C налаштовуємо на вихід
    PORTB=0b00000000;    // очищуємо порт B
    PORTC=0b00000000;    // очищуємо порт C
    ADCON1bits.PCFG=0b0000;    // порт A налаштовуємо як
                                // аналоговий вхід
    ADCON0bits.CHS=0b000;    // вибираємо аналоговий
                                // канал RA0
    ADCON0bits.ADCS=0b01;    // тактовий сигнал  $F_{osc}/8$ 
    ADFM=1;                  // праве вирівнювання результату
    ADON=1;                  // включаємо модуль АЦП
    GO=0;                    // стан очікування АЦП
    ADIF=0;                  // скидаємо прапорець переривання
    ADIE=1;                  // дозвіл на переривання від модуля АЦП
    PEIE=1;                  // дозвіл на переривання від різних модулів
    GIE=1;                   // глобальний дозвіл на переривання
    while (1)
    {
        __delay_us (30);
    }
}

```

```

GO=1; // починаємо перетворення
while (ADIF==0) // очікуємо завершення
// перетворення
{
    __delay_us (30); // затримка перед наступним
// перетворенням
}
}
//*****

```

Даний програмний код реалізовує вимірювання напруги поданої на вивід RA0 мікроконтролера PIC16F876A.

Хід роботи

1. У середовищі Proteus складіть схему приведену на рис.13.3. Резистор R_1 візьміть рівним 2 кОм, R_2 – 3 кОм та проведіть розрахунок вхідної напруги. Напруга живлення – 5 В.
2. У середовищі MPLAB введіть вище приведений програмний код роботи модуля АЦП.
3. Проведіть компіляцію коду і перевірте його роботу у середовищі Proteus.
4. Одержаний результат візуалізований з допомогою світлодіодів у двійковій формі перетворіть у десяткову форму та порівняйте з числом одержаним при розрахунку за формулою (13.1).
5. У середовищі Proteus змініть значення резистора R_2 (рис. 13.3) та проведіть вимірювання нового значення напруги.
6. Прошийте мікроконтролер hex-файлом одержаним під час компіляції коду та перевірте роботу модуля АЦП на макетній платі.

Завдання

1. Провести вимірювання вхідної напруги з інших аналогових каналів.
2. Дослідити роботу модуля АЦП за різних часових затримок.

Контрольні питання

1. Яка розрядність модуля АЦП мікроконтролера PIC16F876A?
2. Скільки аналогових каналів є 40/44 вивідних мікроконтролерів?
3. Від чого залежить швидкість аналого-цифрового перетворення?
4. Опишіть рекомендований порядок дій при роботі з модулем АЦП мікроконтролера?

ЛІТЕРАТУРА

1. Мухаммед Али Мазиди, Ролин Д. МакКинли, Дэнні Кусэй. Микроконтроллеры PIC и встроенные системы. Применение ассемблера и C для PIC18: Пер. с англ. – К.: «МК-ПРЕС», СПб.: «КОРОНА-ПРИНТ», 2009. – 784 с., ил.
2. Петин В.А. Проекты с использованием контролера Arduino. – 3-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2019. – 496 с.
3. Лесовой Л.В. Електроніка та мікропроцесорна техніка. Частина 2. Лабораторний практикум: навч. Посібник / Л.В. Лесовой, І.В. Костик, Я.В. Грень. – Львів: Видавництво Політехніки, 2014. – 268 с.
4. Microchip Technology Inc Home. – <http://www.microchip.com>
5. MPLAB® Integrated Development Environment
<https://www.microchip.com/mplab/mplab-x-ide>
6. PIC Microcontrollers – Programming in C. – Milan Verle
<http://learn.mikroe.com/ebooks/picprogramming/>
7. <http://ww1.microchip.com/downloads/en/DeviceDoc/51702a.pdf>
8. Яценков В.С. Микроконтроллеры Vicrochip. Практическое руководство – Москва: Горячая Линия-Телеком, 2002. – 296 с. ил.
9. Иванов Б.В. Программирование микроконтроллеров для начинающих. – К: МК-Пресс, 2010. – 176 с.
10. Алехин, В. А. Микроконтроллеры PIC : основы программирования и моделирования в интерактивных средах MPLAB IDE, microC, TINA, Proteus : практикум / В. А. Алехин. - Москва : Горячая линия-Телеком, 2016. - 248 с. : ил., табл.
11. Баранов В.Н. Применение микроконтроллеров AVR. Схемы, алгоритмы, программы. Москва: Додэка-XXI, 2004. — 288 с.
12. Дорф Р., Бишоп Р. Современные системы управления. Пер. с англ. Б. И. Копылова. — М.: Лаборатория базовых знаний, 2002. — 832 с.

13. Шагурин И.И. Современные микроконтроллеры и микропроцессоры Motorola. Москва: Горячая линия - Телеком, 2004. — 952 с.: ил.
14. Михаэль Хофманн Микроконтроллеры для начинающих. БХВ-Петербург, 2013. — 304 с. ил.
15. Цирульник С.М., Азаров О.Д., Крупельницький О.В., Трояновська Т.І. Мікропроцесорна техніка: навч. посіб. / Цирульник С.М., Азаров О.Д., Крупельницький О.В., Трояновська Т.І. — Вінниця : ВНТУ, 2017. — 123 с.

Навчальне видання

Пушак Андрій Степанович
Вістовський Віталій Володимирович

ПРОГРАМУВАННЯ
PIS-KONTPOJIEPIB

Лабораторний практикум

Комп'ютерний набір: *Пушак А.С.*
Комп'ютерне верстання: *Пушак А.С.*

Українська академія друкарства,
79020, м. Львів. вул. Під Голоском, 19
Свідоцтво про внесення до державного реєстру
ДК № 3050 від 11.12.2007 р.

Підписано до друку 12.01.2021 р. Формат 60х84/16.
Друк офсетний. Тираж 300 прим.
Зам. №_____

Віддруковано в НВЛПТ
Української академії друкарства
79008, м. Львів, вул. Личаківська 3