МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Центральноукраїнський національний технічний університет

Смірнов В.В., Смірнова Н.В., Пархоменко Ю.М.

АРХІТЕКТУРА ТА ПРОГРАМУВАННЯ ПЕРИФЕРІЙНИХ ІНТЕРФЕЙСНИХ КОНТРОЛЕРІВ

Підручник

Кропивницький - 2020

УДК: 004.41

ББК 32.97

С50

Рекомендовано Вченою радою Центральноукраїнського національного технічного університету до друку та використанню підручника у навчальному процесі здобувачами вищої освіти очної та заочної форми навчання за спеціальністю 123 «Комп'ютерна інженерія», протокол № 1 від 28.09.2020 року

Рецензенти:

Осадчий Сергій Іванович, доктор технічних наук, професор, завідувач кафедрою Автоматизації виробничих процесів Центральноукраїнського національного технічного університету.

Волков Юрій Іванович, доктор фізико-математичних наук, професор, завідувач кафедрою Математики Центральноукраїнського державного педагогічного університету імені Володимира Винниченка.

Неділько Віталій Миколайович, доцент, кандидат технічних наук, завідувач кафедрою Інформаційних технологій Льотної академії Національного авіаційного університету.

Смірнов В.В., Смірнова Н.В., Пархоменко Ю.М.

С50 Архітектура та програмування периферійних інтерфейсних контролерів: підручник. – Кропивницький : ЦНТУ, 2020. – 278 с.

У підручнику описані архітектура і вбудовані спеціалізовані модулі PIC – мікроконтролерів серії PIC18FXX2, методи та засоби для їх програмування. Представлені апаратно-програмні комплекси для розробки та програмування мікроконтролерних систем управління різними технологічними процесами, територіально-розподіленими системами, роботами, об'єктами і комплексами на базі PIC – мікроконтролерів.

Представлені практичні рішення навчальних завдань для кращого освоєння досліджуваного матеріалу, представлені варіанти навчальних завдань для самостійного придбання практичних навичок.

Підручник призначений для здобувачів вищої освіти очної та заочної форми навчання за спеціальністю 123 «Комп'ютерна інженерія».

Навчальне електронне видання комбінованого використання.

Можна використовувати в локальному та мережному режимах

УДК: 004.41

ББК 32.97

© В.В. Смірнов, Н.В. Смірнова, Ю.М. Пархоменко, 2020

© ЦНТУ, кафедра «Програмування комп'ютерних систем і мереж»

Зміст

ВСТУП 8

РОЗДІЛ 1. ОПИС І ХАРАКТЕРИСТИКИ МІКРОКОНТРОЛЕРІВ СЕРІЇ PIC18XX2 10

ОПИС АРХІТЕКТУРИ PIC - МІКРОКОНТРОЛЕРІВ 22

Генератори тактових імпульсів 22

Режими роботи тактового генератора 22

ПОРТИ ВВЕДЕННЯ/ВИВЕДЕННЯ 25

Регістри PORTA, TRISA, LATA 25

Регістри PORTB, TRISB, LATB 27

Регістри PORTC, TRISC, LATC 31

ТАЙМЕРИ 34

Сторожовий таймер 34

Таймер TMR0 35

Таймер TMR1 36

Таймер TMR2 39

Модуль таймера TMR3 41

Робота таймеру TMR3 43

Прескалер 44

ПЕРЕРИВАННЯ 45

Регістри INTCON 46

Регістри PIE 47

Скидання 50

АНАЛОГО-ЦИФРОВИЙ ПЕРЕТВОРЮВАЧ (АЦП) 51

Характеристики АЦП 51

Класифікація АЦП 55

ЦИФРО-АНАЛОГОВИЙ ПЕРЕТВОРЮВАЧ (ЦАП) 57

Характеристики ЦАП 57

Статичні характеристики ЦАП 58

Динамічні характеристики ЦАП 60
Класифікація ЦАП 62
ІНТЕРФЕЙСИ ОБМІНУ ДАНИМИ 64
Послідовний інтерфейс RS-232 64
Шина I²C 69
Інтерфейс SPI 80
Інтерфейс 1-Wire 88
Паралельний інтерфейс 98
ОРГАНІЗАЦІЯ ПАМ'ЯТІ МІКРОКОНТРОЛЕРА 101
Організація пам'яті програм 102
Стек 102
Організація пам'яті даних 104
Регістри загального призначення GPR 107
Регістри спеціального призначення SFR 107
EEPROM І ДОСТУП ДО ДАНИХ 108
ФОРМАТ ФАЙЛУ *.HEX 109
РОЗДІЛ 2. ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ МІКРОКОНТРОЛЕРІВ 112
АПАРАТНО-ПРОГРАМНІ КОМПЛЕКСИ 112
СИСТЕМА АВТОМАТИЗОВАНОГО ПРОЕКТУВАННЯ PROTEUS 120
ОПИС РОБОТИ З ІНТЕГРОВАНИМ СЕРЕДОВИЩЕМ РОЗРОБКИ ПРОГРАМ PCWH 122
Порядок створення проекту 123
Створення проекту у інтегрованому середовищі розробки PCWH за допомогою команди меню Project/Create 124
Створення проекту у інтегрованому середовищі розробки PCWH за допомогою майстра PIC Wizard 128
Компіляція проекту 136
Відкриття створеного проекту 138
Утиліти меню Tools 139

Завантаження бінарного коду у FLASH-пам'ять мікроконтролера 140
ОПИС РОБОТИ З СЕРЕДОВИЩЕМ МОДЕЛЮВАННЯ «PROTEUS» 143
Порядок виконання лабораторних практикумів для варіанту «Proteus» 143
РОЗДІЛ 3. ЛАБОРАТОРНІ ПРАКТИКУМИ 147
ЛАБОРАТОРНИЙ ПРАКТИКУМ № 1 - «IN_OUT» 149
Послідовність виконання лабораторного практикуму для варіанту АПК 150
Послідовність виконання лабораторного практикуму для варіанту «Proteus» 151
Послідовність виконання лабораторного практикуму для варіанту «Proteus» з використанням шаблону програми 152
Приклад створення проекту у інтегрованому середовищі розробки PCWH 153
Створення проекту у інтегрованому середовищі розробки PCWH за допомогою майстра PIC Wizard 153
Пояснення до виконання лабораторного практикуму «IN_OUT» 159
Приклади виконання лабораторного практикуму «IN_OUT» 164
Завдання для лабораторного практикуму «IN_OUT» 165
Результат виконання лабораторного практикуму «IN_OUT» для варіанту АПК 167
Результат виконання лабораторного практикуму «IN_OUT» для варіанту «Proteus» 169
Контрольні питання 171
ЛАБОРАТОРНИЙ ПРАКТИКУМ № 2 - «INTERRUPTS» 172
Послідовність виконання лабораторного практикуму для варіанту АПК 173
Послідовність виконання лабораторного практикуму для варіанту «Proteus» 174
Послідовність виконання лабораторного практикуму для варіанту «Proteus» з використанням шаблону програми 175
Приклад створення проекту у інтегрованому середовищі розробки PCWH 176
Створення проекту у інтегрованому середовищі розробки PCWH за допомогою майстра PIC Wizard 176

Створення проекту у інтегрованому середовищі розробки PCWH за допомогою команди меню Project/Create 185
Пояснення до виконання лабораторного практикуму «IN_OUT» 188
Програмне забезпечення для передачі даних за допомогою послідовного інтерфейсу RS-232 188
Програма – термінал Terminal 1.9b 189
Програма – термінал SIOW.exe 194
Приклади виконання лабораторного практикуму «INTERRUPTS» 196
Завдання для лабораторного практикуму «INTERRUPTS» 196
Результат виконання лабораторного практикуму «INTERRUPTS» для варіанту АПК 198
Результат виконання лабораторного практикуму «INTERRUPTS» для варіанту «Proteus» 201
Контрольні питання 204
ЛАБОРАТОРНИЙ ПРАКТИКУМ № 3 - «ADC» 205
Послідовність виконання лабораторного практикуму для варіанту АПК 206
Послідовність виконання лабораторного практикуму для варіанту «Proteus» 208
Послідовність виконання лабораторного практикуму для варіанту «Proteus» з використанням шаблону програми 208
Приклад створення проекту у інтегрованому середовищі розробки PCWH 209
Створення проекту у інтегрованому середовищі розробки PCWH за допомогою команди меню Project/Create 209
Створення проекту у інтегрованому середовищі розробки PCWH за допомогою майстра PIC Wizard 210
Пояснення до виконання лабораторного практикуму «ADC» 217
Робота модуля АЦП 220
Приклади виконання лабораторного практикуму «ADC» 222
Результат виконання лабораторного практикуму «ADC» для варіанту АПК 224
Результат виконання лабораторного практикуму «ADC» для варіанту «Proteus» 228
Контрольні питання 232

ЛАБОРАТОРНИЙ ПРАКТИКУМ № 4 - «DAC» 233
Послідовність виконання лабораторного практикуму для варіанту АПК 234
Послідовність виконання лабораторного практикуму для варіанту «Proteus» 236
Послідовність виконання лабораторного практикуму для варіанту «Proteus» з використанням шаблону програми 237
Приклад створення проекту у інтегрованому середовищі розробки PCWH 238
Створення проекту у інтегрованому середовищі розробки PCWH за допомогою майстра PIC Wizard 238
Пояснення до виконання лабораторного практикуму «DAC» 244
Цифро-аналоговий перетворювач MCP4921 244
Приклади реалізації програми лабораторного практикуму «DAC» 248
Приклади виконання лабораторного практикуму «DAC» 253
Результат виконання лабораторного практикуму «DAC» для варіанту АПК 255
Результат виконання лабораторного практикуму «DAC» для варіанту «Proteus» 257
Контрольні питання 260
ГЛОСАРІЙ 261
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 273

ВСТУП

Дисципліна **«Програмне забезпечення управляючих мікро-ЕОМ»** займає важливе місце серед інших навчальних дисциплін, які розширюють і поглиблюють знання, вміння і навички, отримані студентами в результаті освоєння матеріалів даної дисципліни.

Завданням навчальної дисципліни є підготовка фахівців в області розробки і програмування мікроконтролерних систем управління різними технологічними процесами, територіально-розподіленими системами, роботами, об'єктами і комплексами.

В основі побудови сучасних технічних систем лежить автоматизація процесів, контроль та управління станом систем за допомогою однокристальних мікроконтролерів (МК, MCU - MicroController Unit).

На відміну від мікропроцесорів МК включають всі пристрої, потрібні для реалізації цифрових систем управління: процесор, оперативна пам'ять, пам'ять команд, електрично програмована постійна пам'ять, внутрішній генератор тактових сигналів, АЦП, ЦАП, пристрій для зв'язку із зовнішнім середовищем, а також інші спеціалізовані модулі.

Тому мікроконтролери дозволяють реалізувати широке коло завдань управління об'єктами різного призначення.

Існує велика кількість МК різного рівня складності, які виготовляються провідними фірмами. Одним з важливих чинників в освоєнні технології розробки та програмування систем на базі МК є так званий «поріг входження», який визначається ступенем складності в освоєнні апаратної і програмної складової оточення МК.

Найнижчим «порогом входження» для освоєння МК мають мікроконтролери фірми Microchip Technology Inc. спільно з інтегрованим середовищем розробки програм **(IDE) PSWH** фірми Custom Computer Services Inc.

Завданням підручника є швидке навчання студентів навичкам роботи з мікроконтролерами у рамках виділених кредитів за допомогою спеціально розроблених апаратно-програмних комплексів які призначені для створення, програмування, налагодження мікроконтролерних систем управління і виконання лабораторних робіт в рамках курсу **«Програмне забезпечення управляючих мікро-ЕОМ»**, а також для підготовки програмістів в області розробки і управління територіально-розподіленими системами, роботами, об'єктами і комплексами.

Підручник призначений для здобувачів вищої освіти очної та заочної форми навчання за спеціальністю 123 «Комп'ютерна інженерія».

РОЗДІЛ 1. ОПИС І ХАРАКТЕРИСТИКИ МІКРОКОНТРОЛЕРІВ СЕРІЇ PIC18XX2

Периферійні інтерфейсні мікроконтролери (PIC - Peripheral Interface Controller**)** вироблені компанією Microchip Technology Inc., (http://www.microchip.com) засновані на модифікованій гарвардській RISC-архітектурі і поставляються в найрізноманітніших апаратних конфігураціях.

Відмінною особливістю мікроконтролерів PIC є їх програмна сумісність «знизу вгору», тобто програми, написані для старіших моделей, можуть легко і в багатьох випадках - без будь-яких модифікацій бути запущені в новіших пристроях. Базовий набір команд мікроконтролерів PIC містить всього лише 33 інструкції і більшість моделей використовує один і той же набір команд.

Мікроконтролери PIC реалізують такі апаратні можливості:

- цифрові порти введення/виведення;
- 8 і 16 - розрядні таймери з переддільниками частоти;
- сторожовий таймер;
- скидання по включенню електроживлення;
- «сплячий» режим, що дозволяє економити електроенергію;
- інтерфейс з зовнішнім тактовим генератором;
- пам'ять даних;
- пам'ять програм типу EEPROM або FLASH;
- пам'ять даних EEPROM;
- аналогові вхідні і вихідні канали;
- аналогові компаратори;
- зовнішні і внутрішні переривання;
- внутрішній осцилятор;
- вихідний сигнал з широтно-імпульсною модуляцією (ШІМ);
- інтерфейс USART для послідовного обміну даними;
- інтерфейс CAN;
- інтерфейс I^2C;

- інтерфейс SPI;
- інтерфейс з модулем ЖК-дисплея;
- інтерфейс USB;
- ремаппінг (Peripheral Pin Select, PPS).

Сімейства мікроконтролерів PIC працюють з 8, 16 і 32 - розрядними даними. Всі вони розбиті на основні підсімейства:

- мікроконтролери з 8-розрядними даними: серії 10xx, 12xx, 16XX, 18xx;
- мікроконтролери з 16-розрядними даними: серія 24xx;
- мікроконтролери з 32-розрядних даними: серія 32XX.

Всі мікроконтролери використовують одну і ту ж RISC-архітектуру та однаковий набір команд при наявності декількох додаткових інструкцій у моделях з 14-розрядним і 16-розрядним командним словом.

Розгляд серії **PIC18XX2** буде здійснюватися на прикладі мікроконтролера **PIC18F252**. Мікроконтролер **PIC18F252** випускається у 28-вівідніх корпусах SOIC та DIP (рис. 1.1).

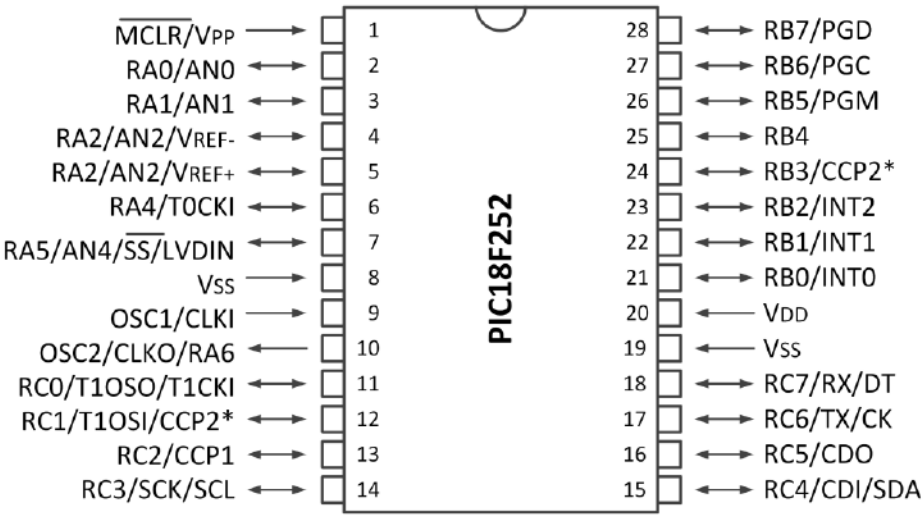


Рисунок 1.1 - Розташування виводів мікроконтролера **PIC18F252**

Примітки: RB3 - альтернативний вивід модуля CCP2.

Всі мікроконтролери цієї серії мають однакову архітектуру (рис. 1.2) і програмну сумісність. Розрізняються розміром RAM, EEPROM та

FLASH-пам'яті, а також наявністю/відсутністю деяких модулів, що обумовлено кількістю виводів мікроконтролера. 28-вівідні мікроконтролери не мають модуля веденого паралельного порту (PSP), а число АЦП обмежене 5 каналами.

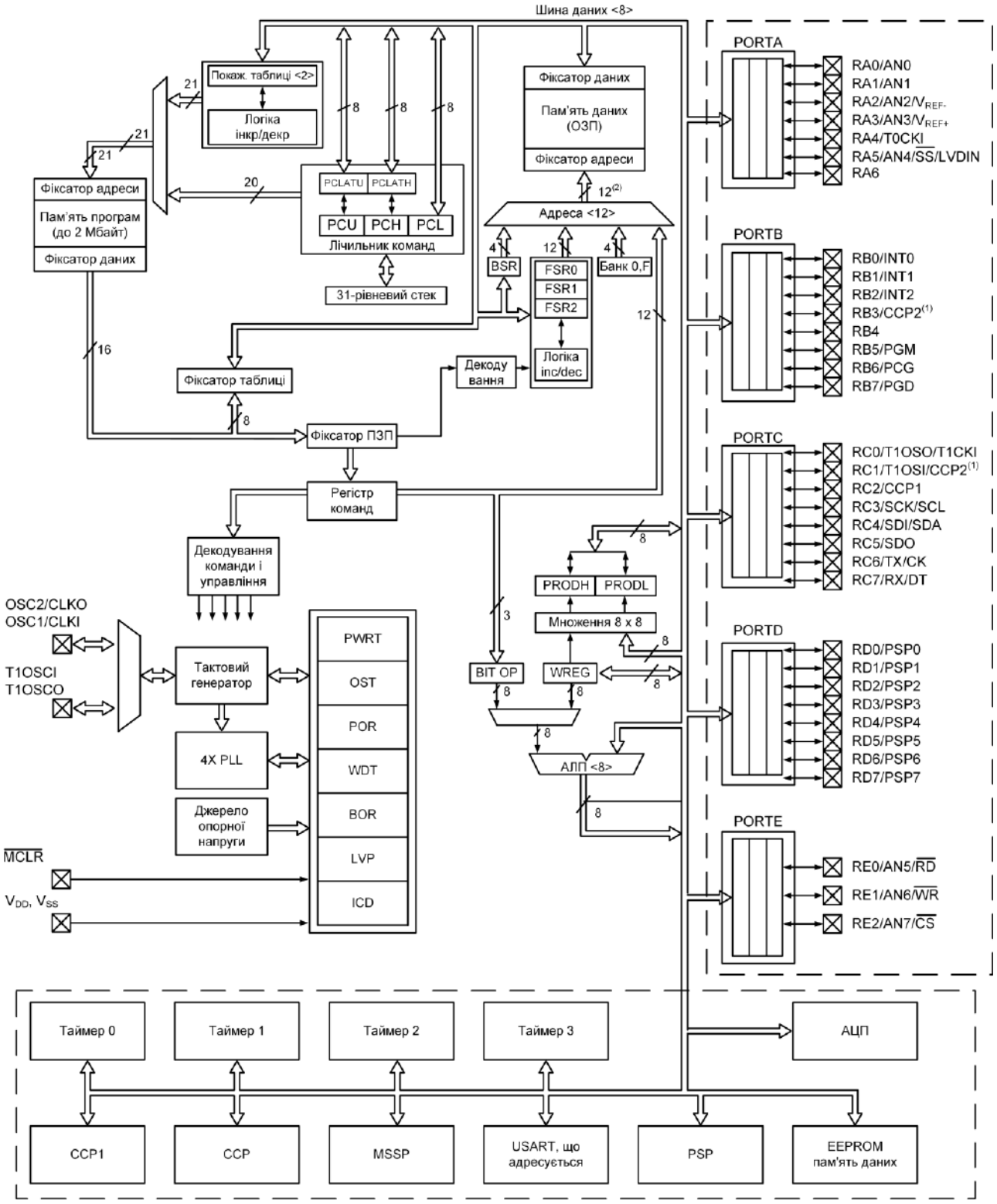


Рисунок 1.2 - Архітектура PIC мікроконтролерів серії 18FXX2

Примітки:

1) підключення виводу CCP2 до каналу порту введення/виведення визначається бітом конфігурації мікроконтролера;
2) старші біти адреси регістра ОЗП беруться з регістра BSR (крім команди MOVFF);
3) більшість каналів введення/виведення мультиплексуються з виводами периферійних модулів.

Основні характеристики мікроконтролера **PIC18F252** представлені у таблиці 1.1, а призначення виводів - у таблиці 1.2.

Таблиця 1.1 - Основні характеристики мікроконтролера PIC18F252

Параметр	**PIC18F252**
Тактова частота	40МГц
Пам'ять програм (байт)	32К
Пам'ять програм (команд)	16384
Пам'ять даних (байт)	1536
EEPROM пам'ять даних (байт)	256
Джерел переривань	17
Порти введення/виведення	PORT A, B, C
Таймери	4
Модуль CCP	2
Послідовні інтерфейси	MSSP, USART, що адресується
Паралельні інтерфейси	-
Модуль 10-розрядного АЦП	5 каналів
Скидання	POR, BOR, команда RESET, переповнення стеку, вичерпання стеку (PWRT, OST)
Програмований детектор зниженої напруги	Є
Програмоване скидання по зниженню напруги живлення (BOR)	Є
Команд мікроконтролера	75
Корпус	28DIP 28SOIC

Таблиця 1.2 - Призначення виводів мікроконтролера PIC18F252

Позначення	Номер виводу		Тип виводу	Тип буфера	Опис
	DIP	SOIC			
-MCLR/V_{PP}	1	1			Вхід скидання мікроконтролера або напруга програмування
-MCLR			I	ST	Вхід скидання мікроконтролера. Активний низький логічний рівень
V_{PP}			P	-	Вхід напруги програмування
OSC1/CLKIN	9	9			Кварцовий резонатор або вхід зовнішнього тактового сигналу
OSC1			I	ST	Вхід для підключення кварцового резонатора або зовнішнього тактового сигналу. ST буфер у RC режимі тактового генератора, CMOS в інших режимах
CLKIN			I	CMOS	Вхід зовнішнього тактового сигналу. Завжди пов'язаний з функціями OSC1 (див. OSC1 / CLKIN, OSC2 / CLKO)

Продовження таблиці 1.2

Позначення	Номер виводу		Тип виводу	Тип буфера	Опис
	DIP	SOIC			
OSC2/CLKO/RA6	10	10			Кварцовий резонатор або вихід тактового сигналу
OSC2			O	-	Вихід для підключення кварцового резонатора в режимі кварцового резонатора тактового генератора
CLKO			O	-	У RC режимі тактового генератора на виводі CLKO присутній сигнал з частотою Fosc/4, синхронний виконанню команд мікроконтролером
RA6			I/O	TTL	Канал порту введення / виведення
PORTA-двонаправлений порт введення/виведення					
RA0/AN0	2	2			
RA0			I/O	TTL	Цифровий канал порту введення / виведення.
AN0			I	AN	Аналоговий вхід 0
RA1/AN1	3	3			
RA1			I/O	TTL	Цифровий канал порту введення / виведення

Продовження таблиці 1.2

Позначення	Номер виводу		Тип виводу	Тип буфера	Опис
	DIP	SOIC			
AN1			I	AN	Аналоговий вхід 1
RA2/AN2/V_{REF-}	4	4			
RA2			I/O	TTL	Цифровий канал порту введення / виведення
AN2			I	AN	Аналоговий вхід 2
V_{REF-}			I	AN	Вхід опорної напруги АЦП
RA3/AN3/V_{REF+}	5	5			
RA3			I/O	TTL	Цифровий канал порту введення / виведення
AN3			I	AN	Аналоговий вхід 3
V_{REF+}			I	AN	Вхід опорної напруги АЦП
RA4/T0CKI	6	6			
RA4			I/O	ST /OD	Цифровий канал порту введення / виведення. Вихід з відкритим колектором
T0CKI			I	ST	Вхід тактового сигналу для TMR0
RA5/AN4/-SS/ LVDIN	7	7			
RA5			I/O	TTL	Цифровий канал порту введення / виведення
AN4			I	AN	Аналоговий вхід 4
-SS			I	ST	Вхід вибору веденого SPI
LVDIN			I	AN	Вхід детектора зниженої напруги
RA6					Дивіться OSC2/CLKO/RA6

Продовження таблиці 1.2

Позначення	Номер виводу		Тип виводу	Тип буфера	Опис
	DIP	SOIC			
PORTB - двонаправлений порт введення/виведення. На всіх входах PORTB можуть бути програмно включені резистори, що підтягуються					
RB0/INT0	21	21			
RB0			I/O	TTL	Цифровий канал порту введення / виведення
INT0			I	ST	Вхід зовнішнього переривання 0
RB1/INT1	22	22			
RB1			I/O	TTL	Цифровий канал порту введення / виведення
INT1			I	ST	Вхід зовнішнього переривання 1
RB2/INT2	23	23			
RB2			I/O	TTL	Цифровий канал порту введення / виведення
INT2			I	ST	Вхід зовнішнього переривання 2
RB3/CCP2	24	24			
RB3			I/O	TTL	Цифровий канал порту введення / виведення.
CCP2			I/O	ST	Вхід захоплення 2, вихід порівняння 2, вихід ШІМ 2
RB4	25	25	I/O	TTL	Цифровий канал порту введення / виведення. Переривання по зміні рівня сигналу на вході

Продовження таблиці 1.2

Позначення	Номер виводу		Тип виводу	Тип буфера	Опис
	DIP	SOIC			
RB5/PGM	26	26			
RB5			I/O	TTL	Цифровий канал порту введення / виведення. Переривання по зміні рівня сигналу на вході
PGM			I	ST	Включення режиму низьковольтного програмування ICSP
RB/PGC	27	27			
RB6			I/O	TTL	Цифровий канал порту введення / виведення. Переривання по зміні рівня сигналу на вході
PGC			I	ST	Вхід тактового сигналу для внутрішньосхемного налагодження та програмування ICSP
RB7/PGD	28	28			
RB7			I/O	TTL	Цифровий канал порту введення / виведення. Переривання по зміні рівня сигналу на вході
PGD			I	ST	Виведення даних для внутрішньосхемного налагодження та програмування ICSP

Продовження таблиці 1.2

Позначення	Номер виводу		Тип виводу	Тип буфера	Опис
	DIP	SOIC			
PORTC - двонаправлений порт введення/виведення					
RC0/T1OSO/T1CKI	11	11			
RC0			I/O	ST	Цифровий канал порту введення / виведення
T1OSO			O	-	Вихід для підключення кварцового резонатора TMR1
T1CKI			I	ST	Вхід тактового сигналу для TMR1/TMR3
RC1/T1OSI/CCP2	12	12			
RC1			I/O	ST	Цифровий канал порту введення / виведення
T1OSI			I	CMOS	Вхід для підключення кварцового резонатора TMR1
CCP2			I/O	ST	Вхід захоплення 2, вихід порівняння 2, вихід ШІМ
RC2/CCP1	13	13			
RC2			I/O	ST	Цифровий канал порту введення / виведення
CCP1			I/O	ST	Вхід захоплення 1, вихід порівняння 1, вихід ШІМ 1
RC3/SCK/SCL	14	14			
RC3			I/O	ST	Цифровий канал порту введення / виведення
SCK			I/O	ST	Вхід/вихід тактового сигналу в режимі SPI

Продовження таблиці 1.2

Позначення	Номер виводу		Тип виводу	Тип буфера	Опис
	DIP	SOIC			
SCL			I/O	ST	Вхід/вихід тактового сигналу в режимі I^2C
RC4/SDI/SDA	15	15			
RC4			I/O	ST	Цифровий канал порту введення / виведення
SDI			I	ST	Вхід даних в режимі SPI
SDA			I/O	ST	Вхід/вихід даних у режимі I^2C
RC5/SDO	16	16			
RC5			I/O	ST	Цифровий канал порту введення / виведення
SDO			O	-	Вихід даних у режимі SPI
RC6/TX/CK	17	17			
RC6			I/O	ST	Цифровий канал порту введення / виведення
TX			O	-	Вихід передавача USART в асинхронному режимі
CK			I/O	ST	Вивід синхронізації в синхронному режимі USART
RC7/RX/DT	18	18			
RC7			I/O	ST	Цифровий канал порту введення / виведення
RX			I	ST	Вхід приймача USART в асинхронному режимі
DT			I/O	ST	Вихід даних USART в синхронному режимі

Продовження таблиці 1.2

Позначення	Номер виводу		Тип виводу	Тип буфера	Опис
	DIP	SOIC			
Vss	8, 19	8, 19	P	-	Загальний вивід для логіки ядра і портів введення/виведення
VDD	20	20	P	-	Напруга живлення для логіки ядра і портів введення / виведення

Позначення:

TTL = ТТЛ сумісний вхід;

ST = вхід з тригером Шмітта і КМОП рівнями;

O = вихід;

OD = вихід з відкритим колектором (немає діода, підключеного до VDD);

CMOS = КМОП сумісний вхід/вихід;

I = вхід;

P = живлення;

AN = аналоговий вхід.

ОПИС АРХІТЕКТУРИ PIC - МІКРОКОНТРОЛЕРІВ

Всі мікроконтролери PIC, мають спільні особливості: наявність пам'яті програм і пам'яті даних; портів введення/виведення, таймерів і т.д.

Генератори тактових імпульсів

Для PIC-мікроконтролерів можуть використовуватися такі типи генераторів тактових імпульсів:

- вбудовані генератори;
- RC-генератори;
- генератори з кварцовими резонаторами;
- генератори з керамічними резонаторами;
- зовнішні генератори.

Режими роботи тактового генератора

Тактовий генератор **PIC18FXX2** може працювати у восьми режимах:

1) **LP** - низькочастотний кварцовий резонатор (мале енергоспоживання), (частота до 200 KHz);
2) **XT** - кварцовий/керамічний резонатор, (частота до 4 MHz);
3) **HS** - високочастотний кварцовий/керамічний резонатор, (частота до 24 MHz);
4) **HS** + **PLL** - високочастотний резонатор з включеним PLL модулем, (частота до 80 MHz);
5) **RC** - зовнішній резистор/конденсатор;
6) **RCIO** - зовнішній резистор/конденсатор з включеним каналом порту введення/виведення;
7) **EC** - зовнішній тактовий сигнал;
8) **ECIO** - зовнішній тактовий сигнал з включеним каналом порту введення/виведення.

Вбудований генератор застосовується в багатьох PIC-мікроконтролерах. Такий генератор зроблений на основі конденсатора і програмованого резистора. Генератор цього типу здатний працювати на частоті 4 МГц і забезпечує стабільність частоти тактових імпульсів не гірше 1,5%.

Генератор із зовнішнім RC-ланцюжком представлений на рис. 1.3.

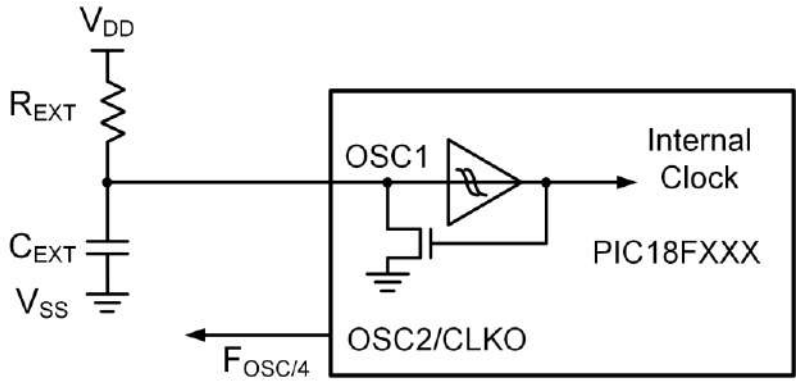


Рисунок 1.3 - Генератор із зовнішнім RC-ланцюжком

Його активним елементом є неінвертуючий буфер, який виконаний на основі тригера Шмітта і служить для відкривання або закривання ключа на МОП транзисторі з N-каналом. Величини ємності конденсатора і опору резистора можуть визначатися за специфікаціями фірми Microchip.

Схеми генераторів з кварцовими і керамічними резонаторами практично не відрізняються одна від одної. Підключення кварцового або керамічного резонатора проводиться відповідно до схеми, показаної на рис. 1.4.

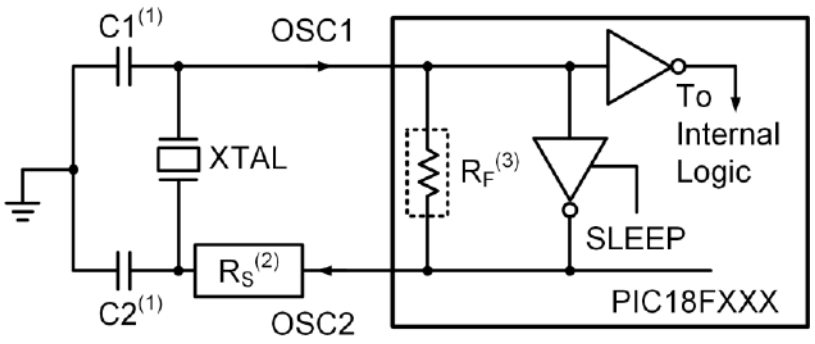


Рисунок 1.4 - Підключення кварцового резонатора

Примітки:

1) для деяких типів резонаторів може знадобитися послідовно включений резистор;

2) значення опору R_F залежить від обраного режиму тактового генератора.

Для нормального функціонування резонатора передбачається підключення двох додаткових конденсаторів.

Ємність цих конденсаторів оговорюється в специфікаціях фірми Microchip і зазвичай становить 15-22 пф.

У пристрої, що включає генератор з кварцовим або керамічним резонатором, вивід OCS2 може бути використаний як вихід тактової частоти для зовнішніх пристроїв.

Варіант тактування PIC-мікроконтролерів передбачає використання зовнішнього генератора, сигнал якого подається безпосередньо на вивід OSC1 (рис. 1.5).

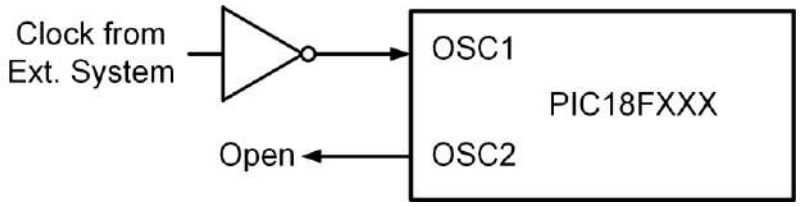


Рисунок 1.5 - Зовнішній тактовий генератор

Для мікроконтролерів **PIC18FXX2** існують різні варіанти тактування, що дозволяє розробнику додатків вибрати найбільш зручний.

Поряд з типовими варіантами тактування допускається застосування схеми чотириразового множення частоти з фазовим автопідстроюванням частоти - ФАПЧ.

При використанні такої схеми мікроконтролер виконує один командний цикл фактично за один період тактових імпульсів. Є також можливість тактування мікроконтролерів від тактових імпульсів таймера TMR1. Цей режим доцільно застосовувати е додатках, які передбачають мале споживання.

ПОРТИ ВВЕДЕННЯ/ВИВЕДЕННЯ

В залежності від кількості виводів (типу корпусу) мікроконтролера, реалізується один або декілька портів введення/виведення. Деякі канали портів введення/виведення мультиплексовані з додатковими функціями периферійних модулів мікроконтролера. У загальному випадку, коли використовується периферійна функція, вивід не може використовуватися як канал порту введення/виведення.

Кожному порту відповідає три керуючих регістра:

- **TRIS** (TRI-state buffer enable) призначені для керування напрямком передачі даних через порти. При записі "1" у будь-який розряд регістра відповідний вихідний буфер відключений і дані можуть надходити з входу у мікроконтролер (режим введення даних). При записі "0" до відповідного біту регістра TRIS вихідний буфер активізується (переходить у режим виведення даних), а величина, записана у розряд регістра даних, передається на відповідний вивід мікроконтролера;
- **PORT** - регістр порту (результатом читання є логічний рівень сигналу на виводах);
- **LAT** - засувка порту введення/виведення. Засувка порту введення / виведення LAT особливо корисна при використанні команд зі структурою «читання - модифікація - запис».

Регістри PORTA, TRISA, LATA

PORTA - 7-розрядний порт введення/виведення. Всі канали PORTA мають відповідні біти напрямку у регістрі TRISA, що дозволяють налаштовувати канал як вхід або вихід.

Запис "1" у TRISA переводить відповідний вихідний буфер у 3-й стан. Запис "0" у регістр TRISA визначає відповідний канал як вихід, вміст засувки PORTA передається на вивід мікроконтролера.

Читання регістра PORTA повертає стан на виводах порту, а запис здійснюється у засувку PORTA.

Регістр засувки LATA відображається на пам'ять даних. Операція типу «читання - модифікація - запис» з регістром LATA буде виконана з даними, записаними у порт введення/виведення PORTA.

Вивід RA4/T0CKI має тригер Шмітта на вході і відкритий сток на виході, мультиплексований з тактовим входом таймера TMR0.

Всі інші канали PORTA мають TTL буфер на вході і повнофункціональні вихідні КМОП буфери.

Інші канали PORTA мультиплексовані з аналоговими входами АЦП і аналоговим входом джерела опорної напруги V_{REF+} і V_{REF-}.

Біти управління режимом роботи каналів порту введення/виведення PORTA знаходяться у регістрі ADCON1 (керуючий регістр АЦП).

Примітка: після скидання по включенню живлення виводи RA5, RA3:RA0 налаштовуються як аналогові входи, читання дає результат "0". Виводи RA4, RA6 при скиданні по включенню живлення (POR) налаштовуються як цифрові входи.

Біти регістра TRISA управляють напрямком каналів PORTA, навіть коли вони використовуються як аналогові входи.

Користувач повинен упевнитися, що відповідні канали PORTA налаштовані на вхід при використанні їх в якості аналогових входів.

Таблиця 1.3 - Функціональне призначення виводів PORTA

Позначення	**№ біта**	**Буфер**	**Опис**
RA0/AN0	0	TTL	Вхід/вихід або аналоговий вхід
RA1/AN1	1	TTL	Вхід/вихід або аналоговий вхід
RA2/AN2/V_{REF-}	2	TTL	Вхід/вихід, аналоговий вхід або V_{REF-}
RA3/AN3/V_{REF+}	3	TTL	Вхід/вихід, аналоговий вхід або V_{REF+}
RA4/T0CKI	4	ST	Вхід/вихід або вхід тактового сигналу для TMR0. Вихід з відкритим колектором

Продовження таблиці 1.3

Позначення	№ біта	Буфер	Опис
RA5/AN4/-SS/LVDIN	5	TTL	Вхід/вихід, вхід вибору веденого SPI, аналоговий вхід або вхід детектора зниженої напруги
OSC2/CLKO/RA6	6	TTL	OSC2, вихід тактового сигналу або канал порту введення/виведення

Позначення:

ST = вхід з тригером Шмітта;

TTL = вхідний буфер ТТЛ.

Таблиця 1.4 - Регістри і біти, пов'язані з роботою PORTA

Адреса	Ім'я	Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0	Значення після POR, BOR
F80h	PORTA	-	RA6	RA5	RA4	RA3	RA2	RA1	RA0	-x0x 0000
F89h	LATA	-	Регістр вихідних даних							-xxx xxxx
F92h	TRISA		Регістр напряму даних							-111 1111
FC1h	ADCON1	ADFM	ADCS2	-	-	PCFG3	PCFG2	PCFG1	PCFG0	00-- 0000

Позначення:

x = невідомо;

u = не змінюється;

r = резерв;

- = не реалізований, читається як "0";

Затінені комірки на роботу не впливають.

Регістри PORTB, TRISB, LATB

PORTB - 8-розрядний двонаправлений порт введення/виведення. Біти регістра TRISB визначають напрямок каналів порту. Встановлення біту у "1" регістра TRISB переводить вихідний буфер у 3-й стан. Запис "0" у регістр TRISB

налаштовує відповідний канал як вихід, вміст засувки PORTB передається на вивід мікроконтролера.

Регістр засувки LATB відображається на пам'ять даних. Операція типу «читання - модифікація - запис» з регістром LATB буде виконана з даними, записаними у порт введення/виведення PORTB.

До кожного виводу PORTB підключений внутрішній підтягуючий резистор. Біт -RBPU (INTCON2<7>) визначає, підключені (-RBPU=0) чи ні (-RBPU=1) резистори, що підтягуються. Резистори, що підтягуються автоматично відключаються після скидання по включенню живлення POR, і коли канали порту налаштовуються на вихід.

Примітка: при скиданні POR канали порту введення/виведення PORTB налаштовуються як цифрові входи.

Чотири канали PORTB RB7:RB4, налаштовані на вхід, можуть генерувати переривання по зміні логічного рівня сигналу на вході. Якщо один з каналів RB7:RB4 налаштований на вихід, то він не може бути джерелом переривань. Сигнал на виводах RB7:RB4 порівнюється зі значенням, збереженим при останньому читанні PORTB. Що стосується розбіжності одного з значень встановлюється прапор RBIF (INTCON<0>), і якщо дозволено, генерується переривання.

Це переривання може вивести мікроконтролер з режиму SLEEP. У підпрограмі обробки переривань необхідно зробити наступні дії:

- виконати читання або запис у PORTB, виключивши невідповідність;
- скинути прапор RBIF у "0".

Невідповідність збереженого значення з сигналом на вході PORTB завжди встановлює біт RBIF у "1". Читання з PORTB перерве умову невідповідності і дозволить скинути прапор RBIF у "0".

Переривання по зміні сигналу на входах рекомендується використовувати для визначення натискання клавіш, коли PORTB повністю задіяний для реалізації клавіатури. Не рекомендується опитувати PORTB при використанні переривань по зміні вхідного сигналу.

Вивід RB3 може бути налаштований бітом конфігурації CCP2MX як додатковий периферійний вивід модуля CCP2 (CCP2MX=0).

Таблиця 1.5 - Функціональне призначення виводів PORTB

Позначення	**№ біта**	**Буфер**	**Опис**
RB0/INT0	0	TTL/ST(1)	Вхід/вихід або зовнішнє переривання INT0. Внутрішні резистори, що підтягуються
RB1/INT1	1	TTL/ST(1)	Вхід/вихід або зовнішнє переривання INT1. Внутрішні резистори, що підтягуються
RB2/INT2	2	TTL/ST(1)	Вхід/вихід або зовнішнє переривання INT2. Внутрішні резистори, що підтягуються
RB3/CCP2/ INT3(3)	3	TTL/ST(4)	Вхід/вихід або зовнішнє переривання INT3. Вхід захоплення 2, вихід порівняння 2, вихід ШІМ 2, якщо біт **CCP2MX = 0** у бітах конфігурації. Внутрішні резистори, що підтягуються
RB4	4	TTL	Вхід/вихід (переривання по зміні сигналу на вході). Внутрішні резистори, що підтягуються
RB5/PGM	5	TTL/ST(2)	Вхід/вихід (переривання по зміні сигналу на вході). Внутрішні резистори, що підтягуються. Включення режиму низьковольтного програмування ICSP
RB6/PGC	6	TTL/ST(2)	Вхід/вихід (переривання по зміні сигналу на вході). Внутрішні резистори, що підтягуються. Вхід тактового сигналу для внутрішньосхемного програмування ICSP

Продовження таблиці 1.5

Позначення	№ біта	Буфер	Опис
RB7/PGD	7	TTL/ST(2)	Вхід/вихід (переривання по зміні сигналу на вході). Внутрішні резистори, що підтягуються. Виведення даних для внутрішньосхемного програмування ICSP

Позначення: **ST** = вхід з тригером Шмітта; **TTL** = вхідний буфер ТТЛ.

Примітки:

1) вхідний буфер з тригером Шмітта при використанні зовнішніх переривань;
2) вхідний буфер з тригером Шмітта при роботі у режимі послідовного програмування;
3) режим роботи виводу (RB3 або CCP2) визначається бітом CCP2MX у регістрі конфігурації;
4) вхідний буфер з тригером Шмітта при використанні виводу як CCP2.

Таблиця 1.6 - Регістри і біти, пов'язані з роботою PORTB

Адреса	Ім'я	Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0	Значення після POR, BOR
F81h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx
F8Ah	LATB	Регістр вихідних даних								xxxx xxxx
F93h	TRISB	Регістр напряму даних								1111 1111
FF2h	INTCON	GIE/ GIEH	PEIE/ GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x
FF1h	INTCON2	RBPU	INTEDG0	INTEDG1	INTEDG2	-	TMR0IP	-	RBIP	1111 - 1-1
FF0h	INTCON3	INT2IP	INT1IP	-	INT2IE	INT1IE	-	INT2IF	INT1IF	11-0 0-00

Позначення:

x = невідомо;

u = не змінюється;

r = резерв;

- = не реалізований, читається як "0".

Затінені комірки на роботу не впливають.

Регістри PORTC, TRISC, LATC

PORTC - 8-розрядний двонаправлений порт введення/виведення. Біти регістра TRISC визначають напрямок каналів порту.

Встановлення біту у "1" регістра TRISC переводить вихідний буфер у 3-й стан. Запис "0" у регістр TRISC налаштовує відповідний канал як вихід, вміст засувки PORTC передається на вивід мікроконтролера.

Регістр засувки LATC відображається на пам'ять даних.

Операція типу «читання - модифікація - запис» з регістром LATC буде виконана з даними, записаними у порт введення/виведення PORTC.

Виводи PORTC мультиплексовані з декількома периферійними модулями (таблиця 1.7). На каналах PORTC присутній вхідний буфер з тригером Шмітта.

При використанні периферійних модулів необхідно відповідним чином налаштовувати біти регістра TRISC для кожного виводу PORTC.

Деякі периферійні модулі скасовують дію бітів TRISC, примусово налаштовуючи вивід на вхід або вихід.

Примітка: при скиданні POR канали порту введення/виведення PORTC налаштовуються як цифрові входи.

Реальний напрям даних каналу порту введення/виведення не завантажується у регістр TRISC, що дозволяє використовувати команди «читання - модифікація - запис» при зверненні до регістру TRISC без обмежень.

Режим роботи виводу RC1 управляється бітом CCP2MX у регістрах конфігурації. За замовчуванням вивід RC1 мультиплексований з виводом периферійного модуля CCP2 (CCP2MX = 1).

Таблиця 1.7 - Функціональне призначення виводів PORTC

Позначення	**№ біта**	**Буфер**	**Опис**
RC0/T1OSO/T1CKI	0	ST	Цифровий канал порту введення/виведення. Вихід для підключення кварцового резонатора TMR1. Вхід тактового сигналу для TMR1/TMR3
RC1/T1OSI/CCP2	1	ST	Цифровий канал порту введення/виведення. Вхід для підключення кварцового резонатора TMR1. Вхід захоплення 2, вихід порівняння 2, вихід ШІМ
RC2/CCP1	2	ST	Цифровий канал порту введення / виведення. Вхід захоплення 1, вихід порівняння 1, вихід ШІМ 1
RC3/SCK/SCL	3	ST	Цифровий канал порту введення / виведення. Вхід/вихід тактового сигналу у режимі SPI. Вхід/вихід тактового сигналу у режимі I^2C
RC4/SDI/SDA	4	ST	Цифровий канал порту введення / виведення. Вхід даних у режимі SPI. Вхід/вихід даних у режимі I^2C
RC5/SDO	5	ST	Цифровий канал порту введення / виведення. Вихід даних у режимі SPI

Продовження таблиці 1.7

Позначення	№ біта	Буфер	Опис
RC6/TX/CK	6	ST	Цифровий канал порту введення / виведення. Вихід передавача USART в асинхронному режимі. Вивід синхронізації у синхронному режимі USART
RC7/RX/DT	7	ST	Цифровий канал порту введення / виведення. Вхід приймача USART в асинхронному режимі. Виведення даних USART у синхронному режимі.

Позначення:

ST = вхід з тригером Шмітта.

Таблиця 1.8 - Регістри і біти, пов'язані з роботою PORTC

Адреса	Ім'я	Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0	Значення після POR, BOR
F82h	PORTC	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	xxxx xxxx
F8Bh	LATC	Регістр вихідних даних								xxxx xxxx
F94h	TRISC	Регістр напряму даних								1111 1111

Позначення:

x = невідомо;

u = не змінюється;

r = резерв;

- = не реалізований, читається як "0".

ТАЙМЕРИ

Важливою складовою частиною мікроконтролера є таймер.

Таймер - це синхронізований лічильник, який по кожному тактовою сигналу рахує в прямому або зворотному напрямку, тобто рахує зі збільшенням або зменшенням свого значення.

У мікроконтролері серії **PIC18FXX2** реалізовано 3 таймера. Таймери можуть використовуватися для самих різних цілей.

Кожен таймер має свій регістр, в якому зберігається значення рахунку в прямому або зворотному напрямку.

При переповненні або спустошенні цього регістра може викликатися переривання.

Сторожовий таймер

Сторожовий таймер (Watch Dog Timer - WDT) - спеціальний таймер, тактується від окремого вбудованого RC-генератора.

При переповненні він здійснює скидання мікроконтролера, таким чином запобігаючи «зависання» програми.

При нормальній роботі програма повинна періодично скидати сторожовий таймер за допомогою команди `clrwdt`, щоб він не переповнявся.

Максимальний інтервал при цьому дорівнює 18 мс. Структурна схема сторожового таймера (WDT) наведена на рис. 1.6.

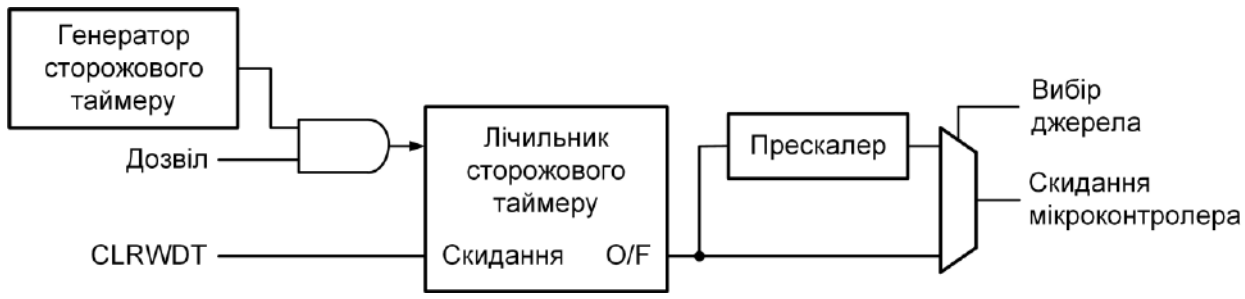


Рисунок 1.6 - Структурна схема сторожового таймера

Таймер TMR0

Таймер TMR0 виконаний на основі 8-розрядного інкрементуємого лічильника, який може попередньо встановлюватися (завантажуватися) за допомогою відповідної послідовності кодів.

Тактування цього лічильника здійснюється від зовнішнього джерела або частоти командних циклів. В останньому випадку використовується дільник частоти на два, тобто приріст таймера відбувається через кожні два командних цикли.

Структура таймера TMR0 показала на рис. 1.7.

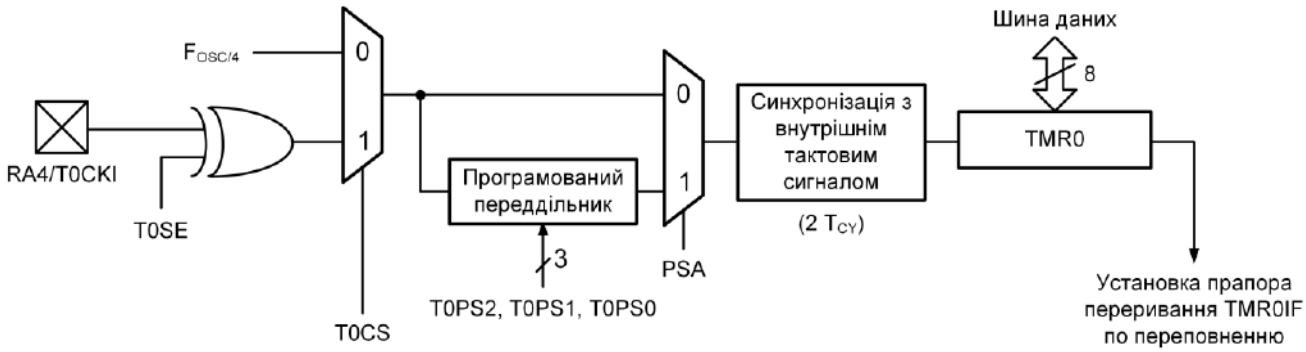


Рисунок 1.7 - Структура таймера TMR0

Примітка: після скидання мікроконтролера TMR0 працює у 8-розрядному режимі із зовнішнім тактовим сигналом (вивід T0CKI) і максимальним коефіцієнтом ділення переддільника.

Біти `TOCS` і `TOCE` застосовуються для вибору джерела синхронізуючих імпульсів, а також типу фронту, за яким здійснюється інкрементування вмісту лічильника TMR0 (по передньому або задньому фронту). Ці біти відносяться до регістру `OPTION`.

Таймер TMR0 може тактуватися від зовнішніх пристроїв через вхід T0CKI. Таймер TMR0 застосовується в якості інтервального таймеру для формування запитів на переривання при переповненні його лічильника.

Діапазон значень таймера становить від 0x000 до 0x0FF.

Частота вхідного тактового сигналу, що надходить на таймер TMR0, може піддаватися попередньому поділу прескалером.

В адресному просторі PIC-мікроконтролерів таймеру TMR0 відповідає адреса 0x001.

Операції запису і зчитування даних з регістра TMR0 дозволяється виконувати за допомогою прямої адресації.

Співвідношення тривалості часових інтервалів, які здатний формувати таймер TMR0, і стартових значень, що завантажуються у нього, визначається формулою:

$$TMR0_{СЗ} = 256 - (\tau * F_{OSC} / 8)$$

де:

TMR0$_{СЗ}$ - стартове значення TMR0;

τ - тимчасова затримка;

F$_{OSC}$ - частота синхронізуючих імпульсів.

Таймер TMR1

Таймер TMR1 виконаний на основі 16-розрядного регістра-лічильника і має чотири різних входи (рис. 1.8).

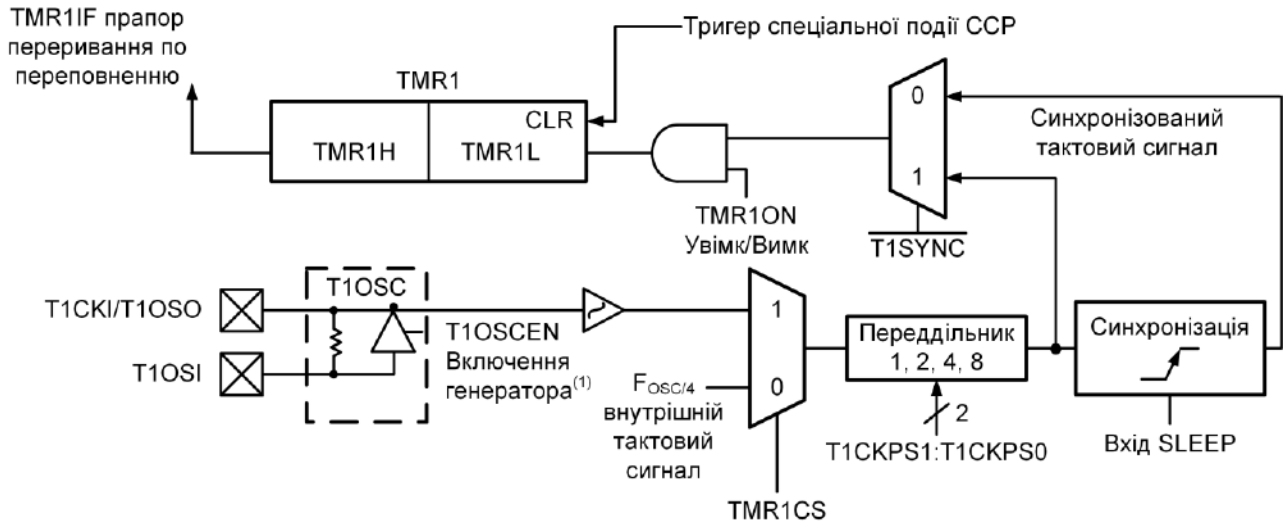


Рисунок 1.8 - Структура таймера TMR1

Примітка: якщо **T1OSCEN = 0**, то елемент, що інвертується та резистивний зворотний зв'язок вимкнені для зменшення струму споживання.

Доступ до вмісту регістра TMR1 забезпечується за допомогою запису і зчитування даних з регістрів TMR1L та TMR1H.

При записі у регістр TMR1 прескалер цього таймера скидається.

Запит на переривання (TMR1IF) формується при переповненні лічильника, для дозволу переривання повинен бути встановлений біт TMR1IE.

Таблиця 1.9 - Величини коефіцієнтів ділення прескалера.

Біти PS2 - PS0	**Коефіцієнт ділення**
000	1
001	2
010	4
011	8
100	16
101	32
110	64
111	128

Біти TMR1IF і TMR1IE зазвичай відносяться до регістрів PIR і PIE відповідно.

Для дозволу переривань повинні бути встановлені біти GIE і PIE регістра INTCON.

Управління таймером TMR1 здійснюється через регістр T1C0N, призначення бітів якого наводиться у табл. 1.10.

Таблиця 1.10 - Призначення бітів регістра T1C0N

Біт	Призначення
7-6	Не використовуються
5-4	T1CPS1 – T1CPS0 - вибір коефіцієнта ділення прескалера для таймера TMR1: 11 - коефіцієнт розподілу задається рівним 8; 10 - коефіцієнт розподілу задається рівним 4; 01 - коефіцієнт розподілу задається рівним 2; 00 - коефіцієнт розподілу задається рівним 1.
3	T10SLEN - біт дозволу використання вбудованого генератора схеми TMR1
2	T1SYNCH - біт дозволу синхронізації зовнішніх тактових імпульсів внутрішньою частотою командних циклів
1	TMR1CS - біт дозволу тактування таймера зовнішніми сигналами
0	TMR10N - біт дозволу використання таймера TMR1

У мікроконтролерах **PIC18FXX2** генератор таймера TMR1 виконує також функцію джерела тактових імпульсів мікроконтролера і забезпечує можливість роботи з відносно повільними, які не вимагають великих об'ємів обчислень програмними додатками, що не передбачають переведення мікроконтролера у режим очікування, щоб зменшити споживання.

Прескалер таймера TMR1 дозволяє формувати тривалі тимчасові інтервали (затримки). Для їх програмування необхідно завантажити відповідне значення у лічильник таймера і визначити коефіцієнт ділення прескалера.

Для розрахунку величини тимчасової затримки зручна наступна формула:

$$\tau = (const * K_{прс}) / F_{T1},$$

де:

τ - часова затримка;

const = (65536-TMR1Init);

Кпрс - коефіцієнт ділення прескалера;

F_{T1} - частота генератора таймера Т1.

Значення *TMR1Init*, яке необхідно завантажити у лічильник, визначається за формулою:

$$TMR1Init = 65536 - (\tau * F_{T1} / Kпрс).$$

Таймер TMR2

Таймер TMR2 (рис. 1.9) застосовується в якості таймера повторюваних подій.

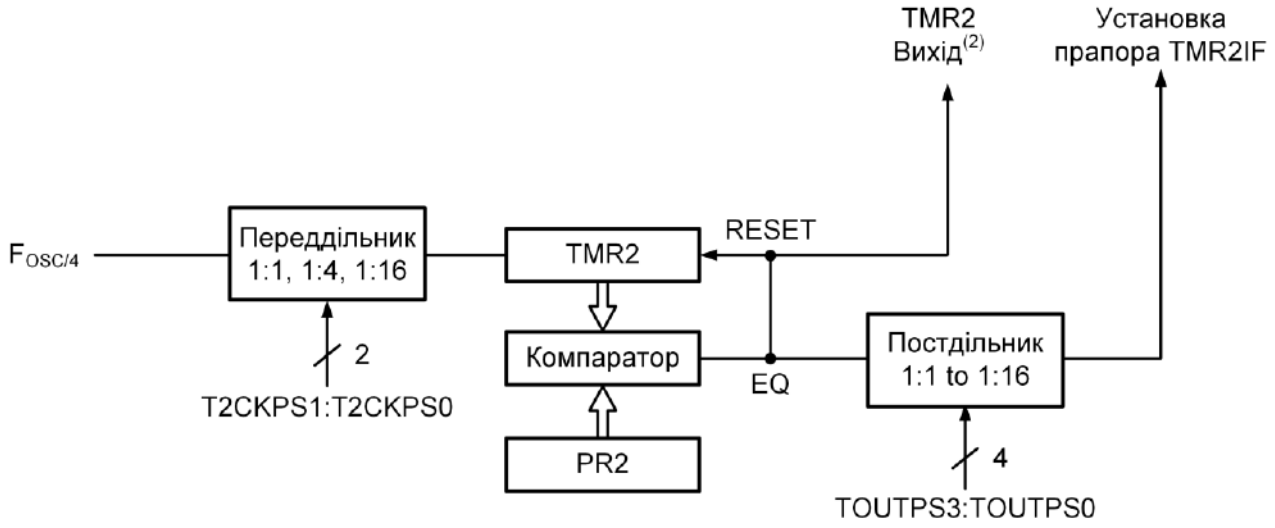


Рисунок 1.9 - Структура таймера TMR2

Примітка: вихідний сигнал TMR2 може використовуватися для програмної настройки швидкості передачі даних модуля MSSP

Спільна робота схеми TMR2 і модуля CCP дозволяє формувати сигнали з широтно-імпульсною модуляцією. У звичайних умовах ця схема може використовуватися для задання тимчасової затримки, відповідної за величиною циклу 16-розрядного лічильника.

Вміст регістра TMR2 постійно зіставляється з величиною, записаною у регістрі PR2.

Коли вміст регістра TMR2 збігається з вмістом регістра PR2, ініціюється скидання регістра TMR2, а інформація про це передається модулю CCP у вигляді повідомлення «перевстановлення TMR2».

Постскалер таймера TMR2 забезпечує розширення тимчасового діапазону подій і формує запити переривань.

Таймер TMR2 управляється за допомогою регістра T2C0N, призначення бітів якого наведено у табл. 1.11.

Таблиця 1.11 - Призначення бітів регістра T2C0N

Біт	**Призначення**
7	Не використовується
6-3	TOUTPS3 – TOUTPS0 - вибір коефіцієнта ділення постскалера для таймера TMR0: 1111 - коефіцієнт розподілу задається рівним 16 1110 - коефіцієнт задається рівним 15 0000 - коефіцієнт задається рівним 1
2	TMR20N - біт дозволу використання прескалера
1-0	T2CKPS1 - T2CKPS0 - вибір величини коефіцієнта ділення прескалера для таймера TMR2: 1x - коефіцієнт задається рівним 16 01 - коефіцієнт задається рівним 4 00 - коефіцієнт задається рівним 1

Регістр *PR2* містить значення модуля рахунку.

Величина тимчасової затримки, що формується таймером, визначається за формулою:

$$\tau = Kпрс * (PR2 + 1) / (Fosc / 4).$$

Якщо значення регістра *PR2* дорівнює нулю, тимчасова затримка визначається за формулою:

$$\tau = (Knpc * 256) / (Fosc / 4).$$

Для організації переривань використовуються біти TMR2IE і TMR2IF, призначення яких аналогічно призначенню відповідних бітів для таймера TMR1. Зазначені біти відносяться до регістрів PIR і PIE.

Таймер TMR2 зручний для роботи з програмними додатками, які передбачають регулярну зміну параметрів у часі, наприклад використовують обмін через послідовний асинхронний інтерфейс або забезпечують формування сигналів з широтно-імпульсною модуляцією.

Модуль таймера TMR3

Модуль таймера TMR3 (рис. 1.10) має такі особливості:

- 16-розрядний таймер/лічильник (з двома 8-розрядними регістрами TMR3H, TMR3L);
- значення таймера доступно для запису і читання (обидва регістра);
- вибір джерела тактового сигналу (зовнішнє або внутрішнє);
- генерація переривань по переповненню від FFFFh до 0000h;
- скидання таймера по сигналу тригера спеціальної події модуля CCP.

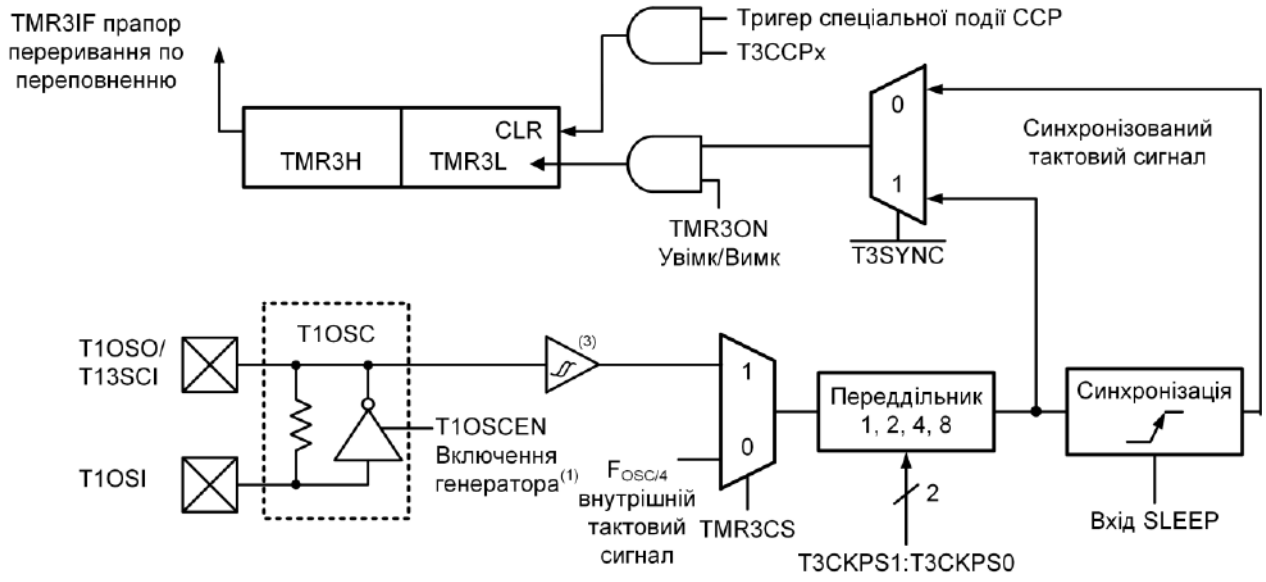


Рисунок 1.10 - Структура таймера TMR3

Примітка: якщо **T1OSCEN = 0**, то інвертор і резистивний зворотний зв'язок вимкнені для зменшення струму споживання.

Керуючий регістр T3CON (рис. 1.11) доступний для запису і читання. Цей реєстр містить біти управління таймера TMR3 і вибору тактового сигналу для модуля CCP.

R/W - 0	R/W - 0	R/W - 0	R/W - 0	R/W - 0	R/W - 0	R/W - 0	R/W - 0
RD16	T3CCP2	T3CKPS1	T3CKPS0	T3CCP1	-T3SYNC	TMR3CS	TMR3ON
Біт 7							Біт 0

Рисунок 1.11 - Регістр T3CON

Призначення бітів регістра:

Біт 7 **RD16:** Включення режиму 16-розрядного читання/запису.

1 = читання/запис регістрів TMR3 виконується за одну 16-розрядну операцію.

0 = читання/запис регістрів TMR3 виконується за дві 8-розрядні операції.

Біт 6, 3 **T3CCP2:T3CCP1:** Вибір джерела тактового сигналу для роботи модуля CCP.

1x = TMR3 використовується для роботи CCP модулів у режимі порівняння/захоплення.

01 = TMR3 використовується для роботи CCP2 модуля у режимі порівняння/захоплення.

TMR1 використовується для роботи CCP1 модуля у режимі порівняння/захоплення.

00 = TMR1 використовується для роботи CCP модулів у режимі порівняння/захоплення.

Біт 5-4 **T3CKPS1:T3CKPS0:** Коефіцієнт ділення переддільника TMR3.

11 = 1:8.

10 = 1:4.

01 = 1:2.

00 = 1:1

Біт 2 **-T3SYNC:** Синхронізація зовнішнього тактового сигналу.

TMR3CS = 1

1 = не синхронізувати зовнішній тактовий сигнал.

0 = синхронізувати зовнішній тактовий сигнал.

TMR3CS = 0

Значення біту ігнорується. Використовується внутрішній тактовий сигнал.

Біт 1 **TMR3CS:** Вибір джерела тактового сигналу.

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

0 = внутрішній тактовий сигнал Fosc/4.

Біт 0 **TMR3ON:** Біт дозволу роботи TMR3.

1 = таймер TMR3 включений.

0 = таймер TMR3 вимкнений.

Позначення:

R = читання біту; **W** = запис біту;

U = не використовується, читається як "0";

"**1**" = біт встановлений; **X** = невідомий стан;

"**0**" = біт скинутий; **- n** = значення після POR.

Робота таймеру TMR3

Модуль таймера TMR3 може працювати в одному з трьох режимів:

- таймер;
- синхронний лічильник;
- асинхронний лічильник.

Режим роботи визначається бітом вибору джерела тактового сигналу TMR3CS (T3CON <1>).

Якщо **TMR3CS = 0**, то значення таймера TMR3 інкрементується на кожному машинному циклі (якщо коефіцієнт переддільника 1:1).

Коли **TMR3CS = 1**, приріст відбувається по кожному передньому фронту зовнішнього тактового сигналу або сигналу генератора TMR1 (якщо він включений).

Коли включений генератор тактових імпульсів (T1OSCEN = 1), виводи RC1/T1OSI і RC0/T1OSO/T1CKI налаштовані як входи.

Значення бітів TRISC <1:0> ігнорується, а читання даних з цих виводів дає результат "0".

Модуль TMR3 має вхід внутрішнього скидання від CCP модуля.

Прескалер

Прескалер - попередній дільник тактової частоти, яка надходить на вхід таймера (рис. 1.12). Він виконаний на основі двійкового лічильника і застосовується спільно з таймерами.

Коефіцієнт розподілу задається програмно і може вибиратися від 1:2 до 1:256.

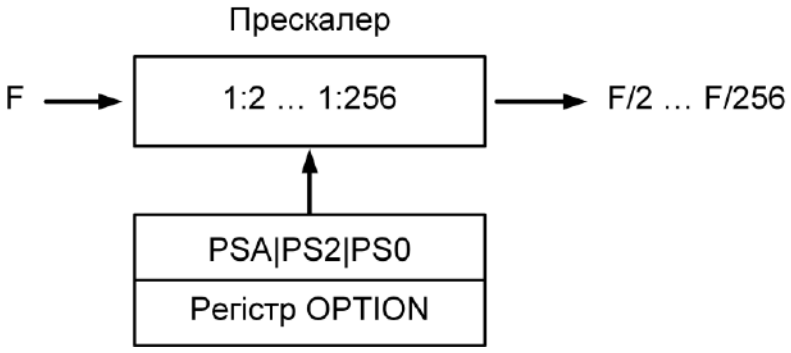


Рисунок 1.12 - Робота прескалера

Управління прескалером здійснюється за допомогою чотирьох бітів (PSA, PS2 - PS0) регістра OPTION.

ПЕРЕРИВАННЯ

Переривання - це реакція мікроконтролера на внутрішні або зовнішні події. В результаті переривання виконання програми зупиняється і відбувається перехід до відповідної підпрограми обробки переривання (функції).

Переривання бувають *внутрішніми*, *зовнішніми, апаратними* та *програмними*. Джерелами внутрішнього переривання є вбудовані модулі мікроконтролера (наприклад, таймер/лічильник або сторожовий таймер), або події програми. Зовнішні переривання викликаються скиданням (сигнал на виводі RESET) або сигналами встановленого рівня на виводах INT та інших.

У момент виникнення переривання у стек поміщається адреса повернення - адреса команди, яка повинна бути виконана першою після виходу з підпрограми обробки переривань.

Мікроконтролери **PIC18FXX2** мають кілька джерел переривань і функцію пріоритетної системи переривань, яка дозволяє для кожного джерела переривань призначити високий або низький пріоритет. При виникненні переривання з високим пріоритетом відбувається перехід по вектору 000008h, а при виникненні переривання з низьким пріоритетом - 000018h. Переривання з високим пріоритетом припиняють обробку переривань з низьким пріоритетом.

У мікроконтролерах **PIC18F252** передбачено 10 регістрів спеціального призначення для управління перериванъ:

- RCON;
- INTCON;
- INTCON2;
- INTCON3;
- PIR1, PIR2 - містить прапори запитів на переривання;
- PIE1, PIE2 - містить прапори дозволу переривань від периферійних пристроїв;
- IPR1, IPR2 - задають пріоритети переривань від периферійних пристроїв.

Кожному джерелу переривань відповідає три керуючих біта:

- прапор переривань, вказує на те, що виконана умова виникнення переривання;
- біт дозволу переривання, дозволяє перехід по вектору переривання при встановленні відповідного прапору;
- біт пріоритету, вибір низького або високого пріоритету переривання.

Регістри INTCON

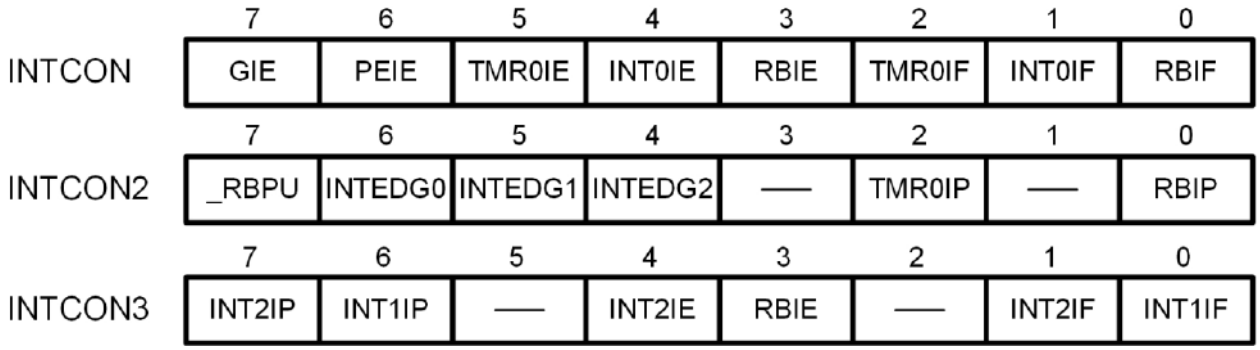


Рисунок 1.26 - Регістри INTCON

Призначення розрядів регістрів INTCON:

Регістр INTCON:

- GIE - прапор загального дозволу переривань;
- PEIE - прапор дозволу всіх запитів на переривання з низьким пріоритетом;
- TMR0IE - прапор дозволу переривань від таймера TMR0;
- INT0IE - прапор дозволу переривань по входу INT0;
- RBIE - прапор дозволу переривання по зміні стану порту B;
- TMR0IF - запит на переривання від таймера TMR0;
- INT0IF - запит на переривання по входу INT0;
- RBIF - запит на переривання по зміні стану порту B.

Регістр INTC0N2:

- INTEDG0 - вибір фронту для зовнішнього переривання по входу INT0 (лог. "1" - по наростаючому фронту);
- INTEDG1 - вибір фронту для зовнішнього переривання по входу 1NT1 (лог. "1" - по наростаючому фронту);
- INTEDG2 - вибір фронту для зовнішнього переривання по входу INT2 (лог. "1" - по наростаючому фронту);
- TMR0IP - задає високий пріоритет для переривання від таймера TMR0;
- RBIP - задає високий пріоритет для переривання по зміні стану порту B.

Регістр INTCON3:

- INT2IP - задає високий пріоритет для зовнішніх переривань по входу INT2;
- INT1IP - задає високий пріоритет для зовнішніх переривань по входу INT1;
- INT2IE - прапор дозволу зовнішніх переривань по входу INT2;
- INT1IE - прапор дозволу зовнішніх переривань по входу INT1;
- INT2IF - запит на переривання по входу INT2;
- INT1IF - запит на переривання по входу INT1.

Структура регістрів PIE показана на рис. 1.27.

Регістри PIE

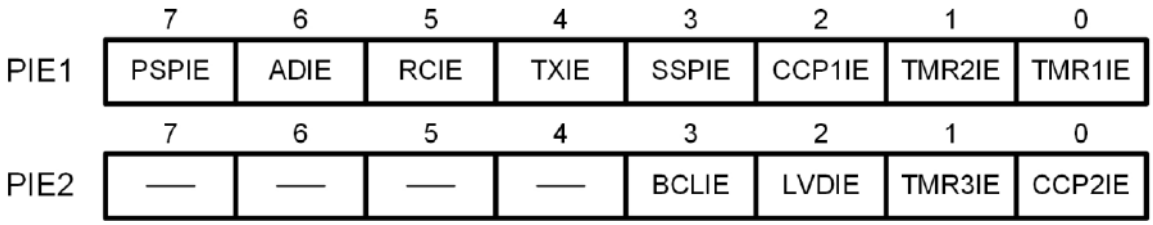


Рисунок 1.27 - Регістри PIE

Призначення розрядів регістрів PIE:

- PSPIE - прапор дозволу переривання від PSP при виконанні операцій читання/запису;
- ADIE - прапор дозволу переривання по завершенню аналого-цифрового перетворення;
- RCIE - прапор дозволу переривання від USART при прийомі даних;
- TXIE - прапор дозволу переривання від USART при передачі даних;
- SSPIE - прапор дозволу переривання від MSSP;
- CCP1IE - прапор дозволу переривання від модуля CCP1;
- TMR2IE - прапор дозволу переривання при збігу вмісту таймеру TMR2 і регістра PR2;
- TMR1TE - прапор дозволу переривання при переповненні таймера TMR1;
- BCLIE - прапор дозволу переривання при конфлікті на шині;
- LVDIE - прапор дозволу переривання при виявленні низького рівня живлячої напруги;
- TMR3IE - прапор дозволу переривання при переповненні таймера TMR3;
- CCP2IE - прапор дозволу переривання від модуля CCP2.

Регістри PIR і IPR мають подібну структуру і взаємозв'язок розрядів з різними пристроями та модулями.

Пріоритетна система переривань включена, якщо біт IPEN (RCON <7>) встановлений у "1". Для пріоритетної системи переривань передбачено два біти глобального дозволу переривань.

Встановлення у "1" біту GIEH (INTCON <7>) дозволяє переривання з високим пріоритетом (біт пріоритету цих переривань повинен бути встановлений).

Якщо біт GIEL (INTCON <6>) встановлений у "1", то дозволені всі переривання з низьким пріоритетом (біт пріоритету цих переривань повинен бути скинутий).

Коли прапор дозволеного переривання встановлений у "1" і дозволені переривання відповідного пріоритету, відбувається перехід по вектору 000008h або 000018h в залежності від пріоритетності переривання. Окремі переривання можуть бути заборонені скиданням відповідного біту дозволу.

Коли біт **IPEN = 0** (стан за замовчуванням), пріоритетна система переривань вимкнена (система переривань сумісна з мікроконтролерами PICmicro середнього сімейства). В цьому режимі біти пріоритету переривань не мають ніякого значення. INTCON<6> - PEIE, дозволяє/забороняє всі периферійні переривання. INTCON<7> - GIE, біт глобального дозволу переривань. При виникненні переривання завжди відбувається перехід по вектору 000008h.

При переході на обробку переривань біт глобального дозволу переривань скидається, щоб заборонити переривання відповідного пріоритету. Якщо біт **IPEN = 0**, то це біт GIE. Якщо пріоритетна система переривань включена, то це один з бітів GIEH або GIEL. Переривання з високим пріоритетом можуть призупиняти обробку переривань з низьким пріоритетом.

Адреса повернення поміщається у вершину стеку, а у лічильник команд PC поміщається вектор переривань (000008h або 000018h). У обробнику переривань конкретне джерело переривань може бути визначене перевіркою відповідних прапорів. Прапори переривань повинні бути скинуті в обробнику переривань для запобігання повторного переходу на обробку переривання.

Вихід з обробки переривань необхідно виконувати командою RETFIE, по якій буде встановлений відповідний біт глобального дозволу переривань (GIE, GIEH або GIEL).

Час переходу на обробку переривань від зовнішніх джерел (переривання INT, зміна рівня сигналу на входах PORTB та ін.) становить три-чотири цикли команд. Час переходу не залежить від типу виконуваної команди (однослівна або двохслівна). Прапори переривань встановлюються незалежно від стану бітів глобального та індивідуального дозволу переривань.

Скидання

Операція скидання призводить до початкового стану мікроконтролер і його регістри спеціального призначення.

Скидання виникає по одній з наступних причин:

- при включенні живлення;
- при появі сигналу низького рівня на вході MCLR;
- по переповненню сторожового таймера.

Найбільш поширені види скидання - при подачі живлення, коли вхід MCLR підключений до напруги живлення через резистор R (рис. 1.28). Крім того, у багатьох системах необхідно передбачити скидання мікроконтролера після натискання зовнішньої кнопки (S1 на рис. 1.28). У цій схемі вхід MCLR в ході роботи мікроконтролера знаходиться в стані лог. "1".

При натисканні кнопки він переходить у стан лог. "0", що призводить до скидання.

Коли кнопка скидання відпущена, мікроконтролер починає виконувати команди, починаючи з адреси 00h пам'яті програм.

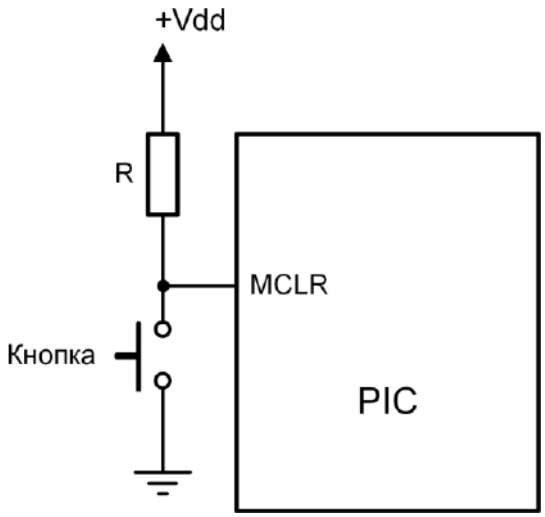


Рисунок 1.28 - Схема скидання мікроконтролера

АНАЛОГО-ЦИФРОВИЙ ПЕРЕТВОРЮВАЧ (АЦП)

Аналого-цифровий перетворювач (АЦП) являє собою пристрій для перетворення безупинно мінливих у часі аналогових сигналів в еквівалентні значення числових кодів. АЦП характеризується набором параметрів. АЦП виконаний або у вигляді модуля МК, або у вигляді окремого чіпу.

Характеристики АЦП

Характеристика перетворення АЦП - це залежність між напругою на його аналоговому вході і безліччю можливих значень вихідного коду, задана у вигляді таблиці, графіка або формули.

Розрізняють *номінальну* характеристику перетворення (рис. 1.29), встановлену у стандартах або технічних умовах на АЦП конкретного типу, і *дійсну* характеристику перетворення, знайдену експериментальним шляхом і настільки наближену до *істинної* характеристики перетворення конкретного АЦП, що для даної мети може бути використана замість неї.

Відхилення характеристики перетворення АЦП від прямої, що з'єднує точки *(0; 0)* і *(V_{REF}; 111 ... 1)*, характеризує *похибку квантування*.

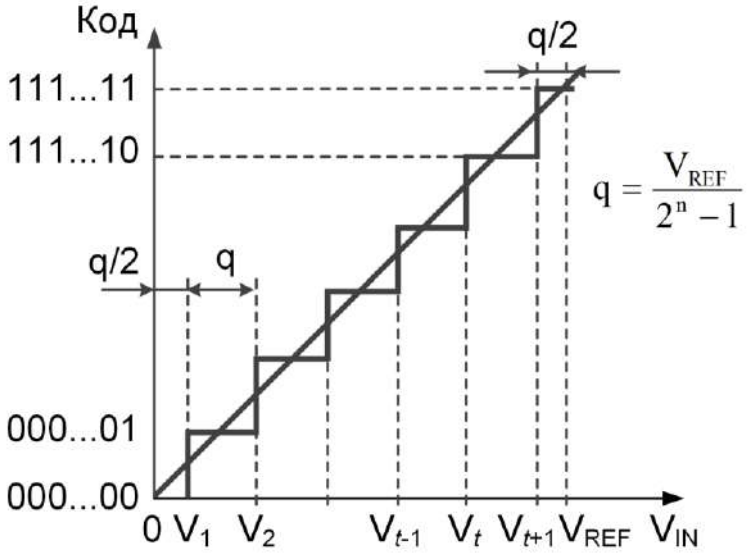


Рисунок 1.29 - Характеристика перетворення АЦП

Як видно з рис. 1.29, для *ідеального* АЦП, що має номінальну характеристику перетворення, максимальна величина цього відхилення становить *q/2* і залежить тільки від кількості розрядів АЦП.

Розрядність. Це основний параметр АЦП, він визначає максимальне двійкове число на виході.

Роздільна здатність (resolution) - це величина, зворотна максимальному числу кодових комбінацій на виході АЦП.

Роздільна здатність виражається у відсотках, розрядах або децибелах і характеризує потенційні можливості АЦП з точки зору досяжної точності.

Наприклад, 12-розрядний АЦП має роздільну здатність 1/4096, або 0.0245% від повної шкали, або -72.2 дБ.

У табл. 1.14 наведений ряд числових співвідношень між кількістю розрядів, роздільною здатністю, а також залежність значення молодшого значущого розряду (МЗР) від діапазону вхідної напруги і т.п.

Таблиця 1.14 - Ряд числових співвідношень між кількістю розрядів і іншими параметрами.

Розряди (n)		Дозвіл	dB	1/n	%	ppm	Значення молодшого значущого розряду (МЗП)				
							20 В	10 В	5 В	2.5 В	1 В
2	2^{-2}	1/4	-12	0.25	25	250000	5 В	2.5 В	1.25 В	625 мВ	250мВ
4	2^{-4}	1/16	-24.1	0.625	6.2	62500	1.25 В	625 мВ	312 мВ	156 мВ	62.5 мВ
6	2^{-6}	1/64	-36.1	0.015625	1.6	15625	312 мВ	156 мВ	78.1 мВ	39.1 мВ	15.6 мВ
8	2^{-8}	1/256	-48.2	0.003906	0.4	3906	78.1 мВ	39.1 мВ	19.5 мВ	9.77 мВ	3.91 мВ
10	2^{-10}	1/1024	-60.2	0.0009766	0.1	977	19.5 мВ	9.77 мВ	4.86мВ	244 мВ	977 мВ

Головний перехід. Під цим розуміють зміну коду з 011 ... 111 на 100 ... 000 або навпаки.

Частота Найквіста. Це частота, рівна 1/2 частоти дискретизації.

Відношення «сигнал/шум» *(Signal to Noise Ratio, SNR)* - це відношення ефективного значення вхідного сигналу до середньоквадратичного значення «шуму». «Шум» складається з усіх інших спектральних компонент, включаючи гармоніки, але виключаючи постійну складову. Рівень вхідного сигналу береться рівним (~ 1 дБ) від повної шкали.

Для ідеального АЦП визначається за формулою:

$$SNR = (6.02\ n + 1.76)\ дБ,$$

де n - кількість двійкових розрядів.

Таким чином, для ідеального 12-розрядного АЦП отримуємо:

$$SNR = 74\ дБ.$$

Інтермодуляційні викривлення *(IMD).* Коли на вхід подаються два гармонійних коливання з різними частотами (f_a і f_b), то на виході будь-якого пристрою з нелінійностямі будуть присутні викривлення порядку (m + n) на сумарних і різницевих частотах:

$$mf_a \pm n\,f_b,$$

де:

m, n = 0, 1, 2, 3 ...

Інтермодуляційними викривленнями називаються ті, для яких ні m, ні n не рівні нулю.

Інтегральна нелінійність *(Integral Non-Linearity, INL) (похибка нелінійності)* - це різниця між розрахунковим значенням вхідної напруги V_t' визначеним по лінеаризованій характеристиці перетворення АЦП, і дійсним значенням вхідної напруги V_t, відповідним заданій точці характеристики перетворення (рис. 1.30):

$$INL = V_t' - V_t$$

Диференціальна нелінійність *(Differential Non-Linearity, DNL)* - це різниця між значенням кванта перетворення (q_t) і середнім дійсним значенням кванта перетворення (q):

$$DNL = q_t - q = V_t - V_{t-1} - q$$

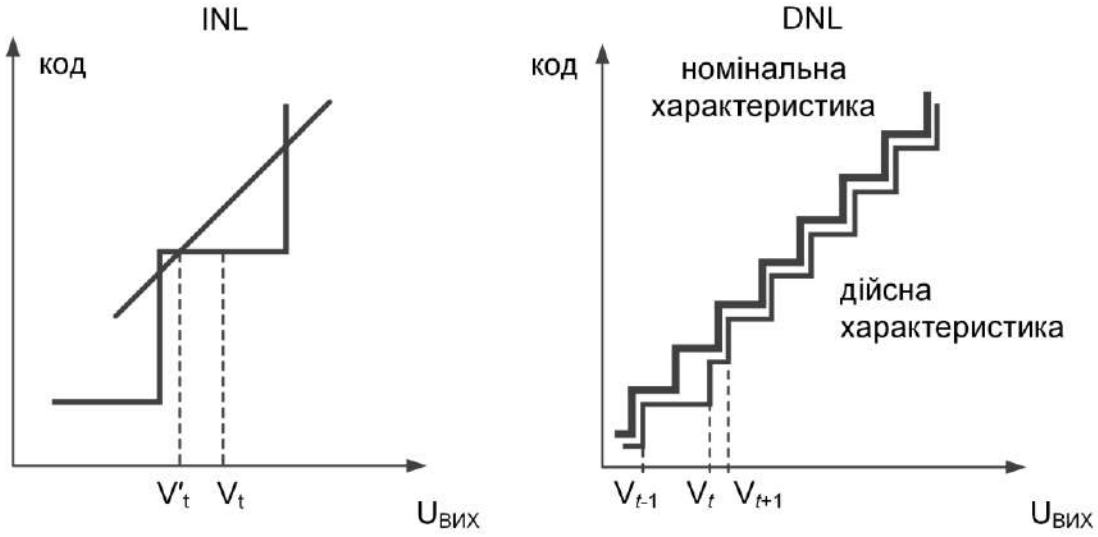


Рисунок 1.30 - Інтегральна та диференціальна нелінійність

Похибка нуля. Перемикання коду з головним переходом має відбуватися при рівні на аналоговому вході - 0,5 МЗР щодо аналогової землі AGND.

Похибка нуля визначається як відхилення фактичного рівня цього перемикання від наведеного значення.

Похибка нуля вказується при певній температурі, а температурний дрейф визначає її максимальну зміну у зазначеному температурному діапазоні.

Похибка біполярного нуля - це відхилення рівня, відповідного переключення коду в середині шкали, від ідеального значення (AGND - 0.5 МЗР).

Час відновлення після перенапруги - це час, який потрібен АЦП для досягнення номінальної точності після перенапруги на аналоговому вході (на 50% більше діапазону повної шкали) і вимірюється до того моменту, коли вхідна напруга повертається в межі вхідного діапазону АЦП.

Робоча смуга частот - це номінальний діапазон частот вхідного сигналу, заданий розробником приладу, в якому при заданій частоті дискретизації нормуються метрологічні характеристики. Вона вужче, ніж смуга пропускання частот повної потужності.

Час встановлення - це час, необхідний АЦП для досягнення номінальної точності після того, як на його вхід (або вхід вбудованого ПВЗ) був поданий ступінчастий сигнал, рівний повному діапазону вхідного сигналу.

Конвеєрна затримка (латентність) - це число тактових періодів між початком перетворення і появою на вихідних лініях відповідного результату. Після цього на кожному тактовому періоді виводиться новий результат перетворення.

Вхідний діапазон. Ця специфікація визначає мінімальні та максимальні вхідні напруги від нуля до повної шкали, які АЦП може сприймати і при цьому точно калібрувати посилення.

Класифікація АЦП

Всі АЦП можна розділити на дві групи, що істотно розрізняються між собою по нормованим характеристикам похибок і методів перевірки.

До першої групи належать АЦП, виконані у вигляді мікросхем (напівпровідникових, гібридних), які не є засобами вимірювань.

АЦП другої групи є засобами вимірювань.

АЦП першої групи широко використовуються не тільки для створення АЦП другої групи, але і в якості вузлів різних систем обробки аналогових сигналів.

До складу АЦП часто входять допоміжні вузли, що істотно поліпшують метрологічні характеристики і розширюють функціональні можливості АЦП:

- буферні підсилювачі;
- автоматичні перемикачі діапазонів;
- програмовані підсилювачі;
- пристрої вибірки-зберігання;
- схеми автокалібрування і автопідстроювання;
- екстраполятор;
- оперативні і постійні запам'ятовуючі пристрої;
- цифрові фільтри і т.п.

Практично всі АЦП орієнтовані на спільну роботу з мікропроцесорними системами і містять елементи інтерфейсу (буферні регістри, дешифратори адреси і т.п.).

Структурна схема модуля АЦП мікроконтролера серії PIC18FXX2 представлена на рис. 1.31.

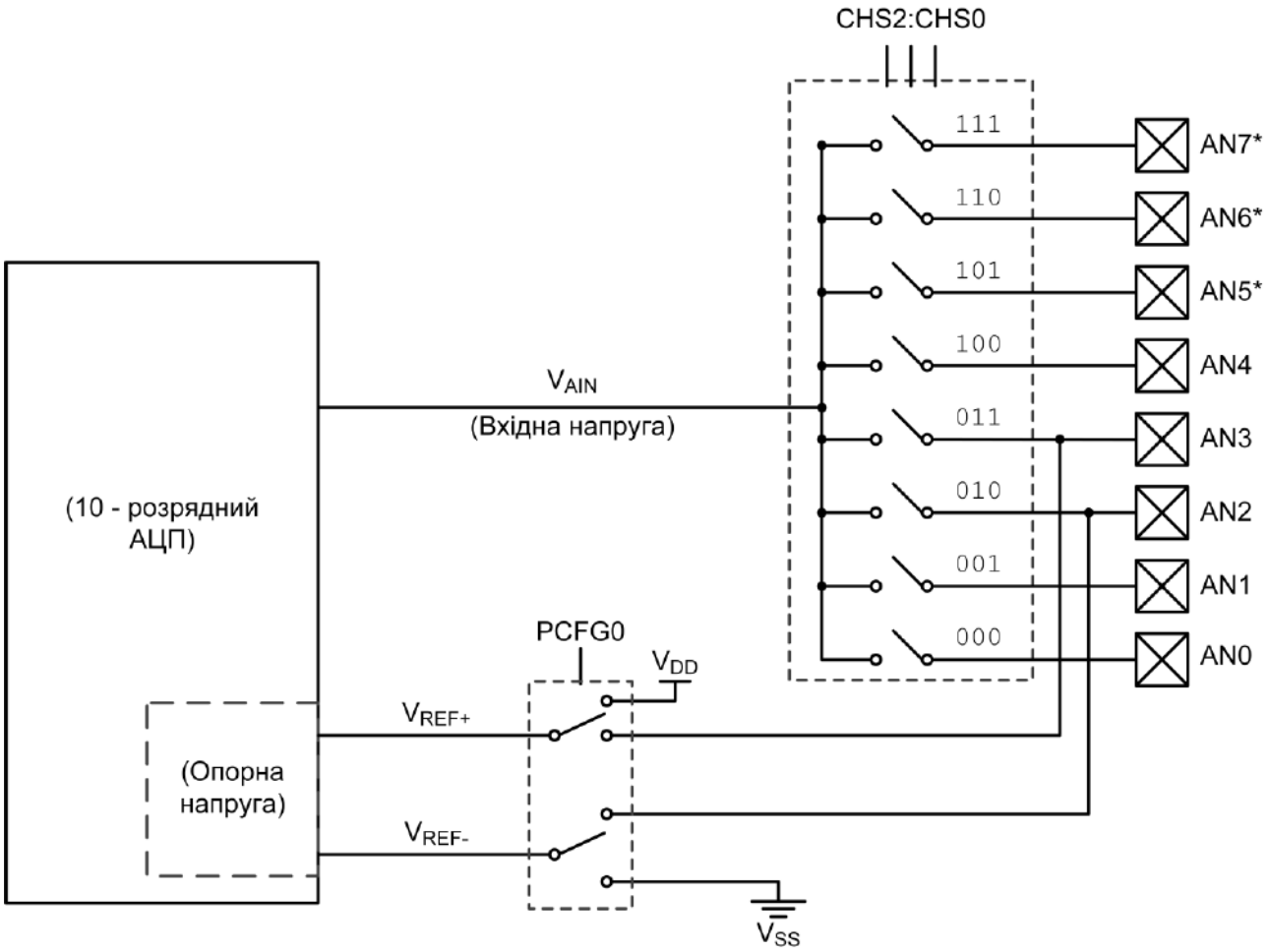


Рисунок 1.31 - Структурна схема модуля АЦП МК серії **PIC18FXX2**

ЦИФРО-АНАЛОГОВИЙ ПЕРЕТВОРЮВАЧ (ЦАП)

Цифро-аналоговий перетворювач *(ЦАП)* призначений для перетворення цифрового сигналу, заданого у вигляді двійкового *N*-розрядного коду, у відповідну напругу або струм.

Якість перетворення залежить від швидкості перетворення і розрядності ЦАП.

Характеристики ЦАП

Залежність вихідної напруги (струму) $U_{вих}(t)$ від вхідного цифрового сигналу $D(t)$, що змінюється від 0 до 2^{N-1}, називають характеристикою перетворення ЦАП. Ця характеристика може бути представлена у вигляді ступінчастої кривої. Величина сходинки відповідає одиниці молодшого значущого розряду (МЗР).

За відсутності апаратних похибок середні точки сходинок розташовані на прямій 1, якій відповідає ідеальна характеристика перетворення (рис. 1.32).

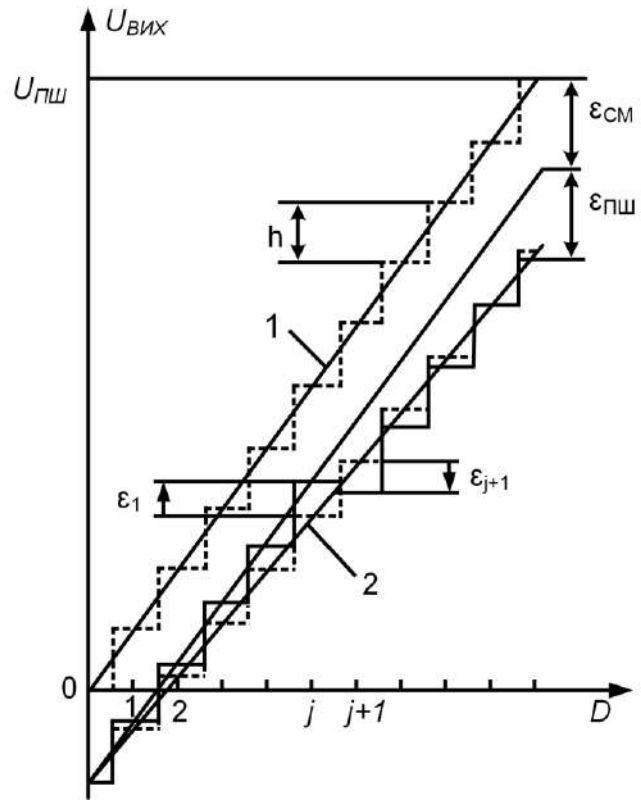


Рисунок 1.32 - Характеристика перетворення ЦАП

Реальна характеристика перетворення може відрізнятися від ідеальної розташуванням, розмірами і формою сходинок.

Для кількісного опису цих відмінностей використовуються характеристики, що називаються ***статичними***. Зміни вихідного сигналу в часі при стрибкоподібний зміні вхідного коду від мінімуму («всі нулі») до максимуму («всі одиниці») описуються динамічними характеристиками ЦАП.

Статичні характеристики ЦАП

Розрядність. Число розрядів (двійкових символів) N, що передають кодований вхідний сигнал. Йому відповідає число рівнів квантування 2^N.

Число каналів. Число аналогових виходів ЦАП, на яких формується вихідна напруга. Залежно від схеми побудови цифровий код для кожного з каналів подається або на різні цифрові входи ЦАП, або на один і той же з поділом за часом. Як правило, частота перетворення і напруга повної шкали для всіх каналів ЦАП рівні. Однак в деяких моделях ІМС ЦАП напруга повної шкали для кожного каналу задається окремо і залежить від опорної напруги.

Роздільна здатність. Приріст $U_{ВИХ}$ при збільшенні коду на одиницю називається *кроком квантування*. Номінальне значення кроку квантування становить:

$$h = \frac{U_{ПШ}}{2^N\text{-}1}$$

де:

$\boldsymbol{U_{ПШ}}$ - номінальна максимальна вихідна напруга ЦАП (напруга повної шкали);

$\boldsymbol{N}$ - розрядність ЦАП. Чим більше розрядність перетворювача, тим вища його роздільна здатність (менше h).

Похибка повної шкали. Відносна різниця між реальним і ідеальним значеннями межі шкали перетворення при відсутності зсуву нуля:

$$\delta_{ПШ} = \frac{\varepsilon_{ПШ}}{U_{ПШ}} \cdot 100\%$$

Вона є мультиплікативною складовою повної похибки. Іноді виражається відповідним числом молодшого значущого розряду.

Похибка зміщення нуля. Значення $U_{ВИХ}$, коли вхідний код ЦАП дорівнює нулю, називається *адитивною складовою повної похибки* ($\varepsilon_{СМ}$, рис. 1.32).

Зазвичай вказується в мілівольтах або у відсотках від повної шкали:

$$\delta_{ПШ} = \frac{\varepsilon_{СМ}}{U_{ПШ}} \cdot 100\%$$

Нелінійність *(інтегральна нелінійність. Integral Non-Linearity, INL).* Максимальне відхилення реальної характеристики перетворення $U_{ВИХ}(D)$ від оптимальної (лінія 2, рис. 1.32).

Оптимальна характеристика знаходиться емпірично так, щоб мінімізувати значення похибки нелінійності. Нелінійність зазвичай визначається у відносних одиницях.

Для характеристики, наведеної на рис. 1.32, цей параметр визначається за формулою:

$$\delta_{Л} = \frac{\varepsilon_j}{U_{ПШ}} \cdot 100\%$$

Диференціальна нелінійність *(Differential Non-Linearity, DNL).* Максимальна зміна (з урахуванням знаку) відхилення реальної характеристики перетворення $U_{ВИХ}(D)$ від оптимальної при переході від одного значення вхідного коду до іншого суміжного значення.

Визначається у відносних одиницях або у МЗР:

$$\delta_{ДЛ} = \frac{\varepsilon_j + \varepsilon_{j+1}}{U_{ПШ}} \cdot 100\%$$

Монотонність характеристики перетворення. Цей параметр характеризує збільшення (зменшення) вихідної напруги ЦАП $U_{ВИХ}$ при збільшенні (зменшенні) вхідного коду D. Якщо диференційна нелінійність більше відносного кроку квантування $h/U_{ПШ}$, то характеристика перетворювача немонотонна. У цьому випадку «оптимальна пряма» виходить за межі «сходинок».

Температурна нестабільність. Температурна нестабільність ЦАП характеризується температурними коефіцієнтами похибки повної шкали, похибки зміщення нуля і похибки нелінійності.

Похибки повної шкали та зміщення нуля можуть бути усунуті калібруванням (підстроюванням). Похибки нелінійності простими засобами усунути не можна.

Динамічні характеристики ЦАП

Частота перетворення. Максимальна частота, з якою ЦАП може формувати вихідну напругу або струм, відповідний вхідному коду.

Частота перетворення - це основний параметр, що характеризує швидкодію ЦАП

Час встановлення. Під цією величиною розуміють інтервал часу від моменту зміни вхідного коду ($t = 0$) до моменту, коли в останній раз виконується рівність (d - допуск на вихідну напругу або струм ЦАП):

$$\delta_{ДЛ} = \frac{\varepsilon_j + \varepsilon_{j+1}}{U_{ПШ}} \cdot 100\%$$

Зазвичай цей допуск рівний половині кроку квантування h.

Швидкість наростання. Це максимальна швидкість зміни $U_{ВИХ}(t)$ під час перехідного процесу, яка визначається як відношення приросту $\Delta U_{ВИХ}$ до часу Δt, за який відбулося це збільшення (рис. 1.33).

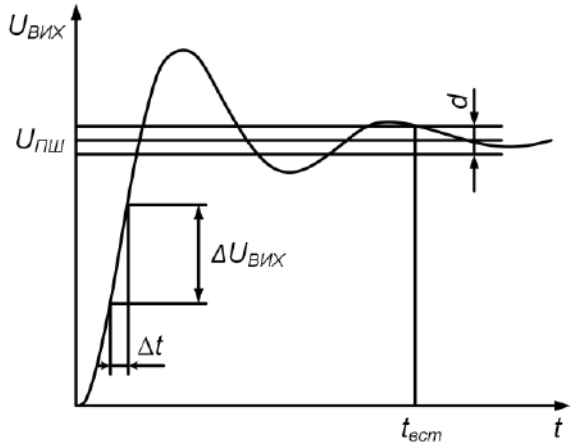


Рисунок 1.33 - Визначення часу встановлення і швидкості наростання

У ЦАП з струмовим виходом цей параметр у великій мірі залежить від типу вихідного операційного підсилювача.

Для перемножуючих ЦАП з виходом у вигляді напруги часто вказуються частота одиничного посилення і показники потужності смуги пропускання, які в основному визначаються властивостями вихідного підсилювача.

Шуми ЦАП. Шум на виході ЦАП може з'являтися з різних причин, що викликається фізичними процесами, які відбуваються в напівпровідникових пристроях.

Для оцінки якості ЦАП з високою роздільною здатністю прийнято використовувати поняття середньоквадратичного значення шуму.

Спектральну щільність шуму вимірюють зазвичай у $нВ/\sqrt{(Гц)}$ у заданій смузі частот.

Викиди *(імпульсні перешкоди)*. Викиди є круті короткі сплески чи провали у вихідній напрузі, що виникають під час зміни значень вихідного коду за рахунок несинхронності розмикання і замикання аналогових ключів в різних розрядах ЦАП.

Наприклад, якщо при переході від значення коду 011 ... 111 до значення 100 ... 000 ключ самого старшого розряду ЦАП з підсумовуванням вагових струмів відкриється пізніше, ніж закриються ключі молодших розрядів, то на виході ЦАП деякий час буде існувати сигнал, відповідний коду 000 ... 000.

Радикальним способом придушення викидів є використання пристроїв вибірки-зберігання. Викиди оцінюються по їх площі в одиницях $nB \cdot c$.

Класифікація ЦАП

За принципом роботи є дві основні групи ЦАП:

- послідовні;
- паралельні.

Перевагою послідовних ЦАП є простота схемотехнічної реалізації, недоліком невисока швидкодія.

Паралельні ЦАП забезпечують максимально можливу швидкодію, але це досягається за рахунок значного ускладнення схеми в порівнянні з послідовними ЦАП.

Класифікація ЦАП за схемотехнічними ознаками представлена на рис. 1.34

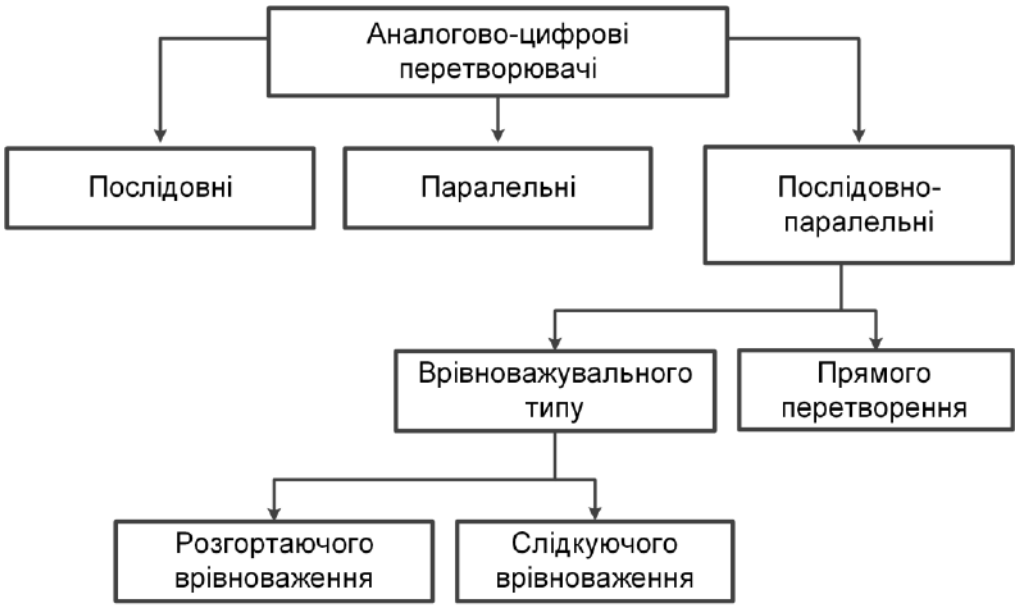


Рисунок 1.34 - Класифікація ЦАП за схемотехнічними ознаками

ЦАП можуть також класифікуватися по:

1. ***вигляду вихідного сигналу:***

- з струмовим виходом;
- з виходом у вигляді напруги;

2. ***полярності вихідного сигналу:***

- однополярні;
- біполярні;

3. ***характером опорного сигналу:***

- з постійним опорним сигналом;
- із змінним опорним сигналом;
- що множать;

4. ***швидкодії:***

- помірною швидкодією;
- високою швидкодією;

5. ***типу цифрового інтерфейсу (введення вихідною коду):***

- з послідовним введенням;
- з паралельним введенням;

6. ***числу ЦАП на кристалі:***

- одноканальні;
- багатоканальні.

ІНТЕРФЕЙСИ ОБМІНУ ДАНИМИ

Інтерфейси обміну даними між пристроями поділяються на:

- послідовні;
- паралельні.

У *послідовних* інтерфейсах дані передаються послідовно, біт за бітом.

У *паралельних* інтерфейсах дані передаються по 8/16/32 - розрядній шині даних.

Перевагою *послідовних* інтерфейсів є мала кількість провідників і відносно велика відстань між пристроями обміну інформацією. Тому послідовні інтерфейси у більшій частині є комунікаційними.

Перевагою *паралельних* інтерфейсів є висока швидкість обміну інформацією при малій відстані між пристроями і простота протоколу.

Послідовний інтерфейс RS-232

Послідовний інтерфейс RS-232 - це промисловий стандарт EIA RS-232-C, CCITT V.24 для послідовної двобічної асинхронної передачі даних.

У послідовному інтерфейсі RS-232 дані передаються послідовно, тобто по лінії зв'язку кожен біт посилається по черзі після надсилання попереднього біту.

Просте підключення через послідовний інтерфейс RS-232/UART

У персональному комп'ютері є роз'єм USB, до якого через перехідник USB-UART може приєднуватися кабель для послідовної передачі даних.

Схема підключення мікроконтролера до персонального комп'ютера через послідовний інтерфейс показана на рис. 1.35.

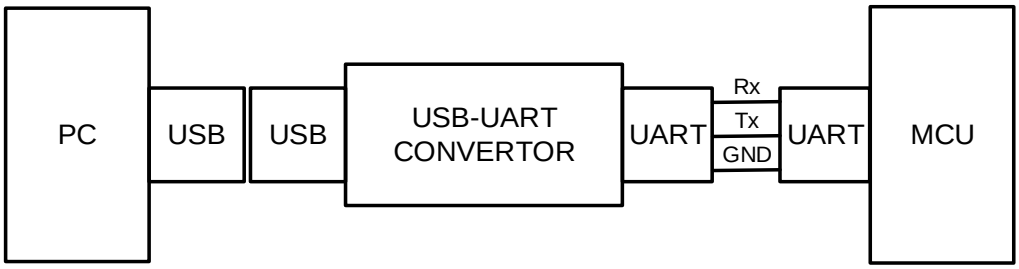


Рисунок 1.35 - Підключення мікроконтролера до PC
за допомогою послідовного інтерфейсу

У багатьох випадках вибирають саме цей варіант підключення, який дозволяє швидко зібрати необхідну електричну схему підключення апаратних засобів і досить просто розмістити керуючу програму у FLASH-пам'яті мікроконтролера.

Формат передачі даних RS-232/UART

Щоб дані могли послідовно передаватися по лінії, для них повинен бути визначений момент початку і кінця слова. Є різні види протоколів передачі по інтерфейсу. Проте в будь-якому випадку передача починається зі стартового біту і закінчується стоповим бітом. Формат даних і швидкість передачі, що використовуються у передавачі, повинні бути встановлені точно такими ж, як і у приймачі, оскільки інакше не може бути забезпечена надійна передача. Тому при проблемах передачі спочатку потрібно перевіряти всі налаштування. При найпростішому способі передачі застосовується 1 стартовий біт, 8 біт даних і 1 стоповий біт.

Якщо ж потрібна найбільш достовірна передача, то може бути доданий ще біт парності, який передається після слова даних. Біт парності призначений для захисту від помилок, який визначає окремі помилки при передачі даних. Якщо при передачі буде пошкоджений один біт, і відповідно невірно визначений, то ця помилка може бути визначена. Якщо ж будуть пошкоджені два або більша кількість бітів, то за допомогою біту парності визначити такі помилки вже не представляється можливим. У випадку використання біту парності можна вибирати між перевіркою на парність і непарність. При перевірці на парність

(EvenParity) у біті парності буде логічний "0", якщо сума одиниць у слові даних парна, або буде логічна "1", якщо сума непарна. При перевірці на непарність - все в точності навпаки.

У багатьох випадках від передачі біту парності відмовляються, т.я. у цьому випадку потрібні додаткові витрати на перевірку або установку біту відповідним чином.

Після того як визначено формат для передачі даних, потрібно встановлювати швидкість, з якою повинні передаватися дані. Оскільки загальна лінія з тактовим сигналом відсутня, то перед початком обміну даними потрібно повідомляти передавача і приймача, з якою швидкістю буде здійснюватися передача. Тому таку передачу називають асинхронною. Швидкість тактування або швидкість передачі вказується у бодах. Не можна плутати цю швидкість зі швидкістю передачі даних, т. я. при цьому мова йде про кількість корисних даних.

При швидкості передачі 19 200 бод або 19,2 кбод рівень сигналу змінюється 19 200 раз в секунду. Так як для передачі 1 байта ще потрібні стартовий і стоповий біт і швидкість передачі даних становить максимум 8/10 швидкості передачі. З цього випливає, що в цьому прикладі можуть передаватися максимум 15 360 біт в секунду або 1 920 байт в секунду. Оскільки в цьому випадку мається на увазі асинхронна передача даних, то стартовий біт може надсилатися в будь-який час. Приймач повинен тоді під час обміну зчитувати дані з тією ж самою швидкістю, з якою вони посилаються передавачем.

В інтерфейсі RS-232 дані, що передаються, повинні мати більш високі рівні напруги, ніж рівні ТТЛ (транзисторно-транзисторної логіки) в UART. Для передачі окремих бітів використовуються позитивні і негативні напруги. Причому рівень між +3 В і +15 В інтерпретується як низький логічний рівень (0, Low, Space), а напруги між -3 В і -15 В як високий (1, High, Mark). Як правило, для передачі використовуються напруги +12 В і -12 В.

На рис. 1.36 представлена процедура передачі даних по інтерфейсу UART / RS-232. Передача здійснюється молодшим розрядами вперед.

Параметри: 8 біт, біт парності - немає, стоп-біт - "1". У прикладі передається латинська буква "M" в ASCII-коді ("M" = 0x4D або 0b01001101).

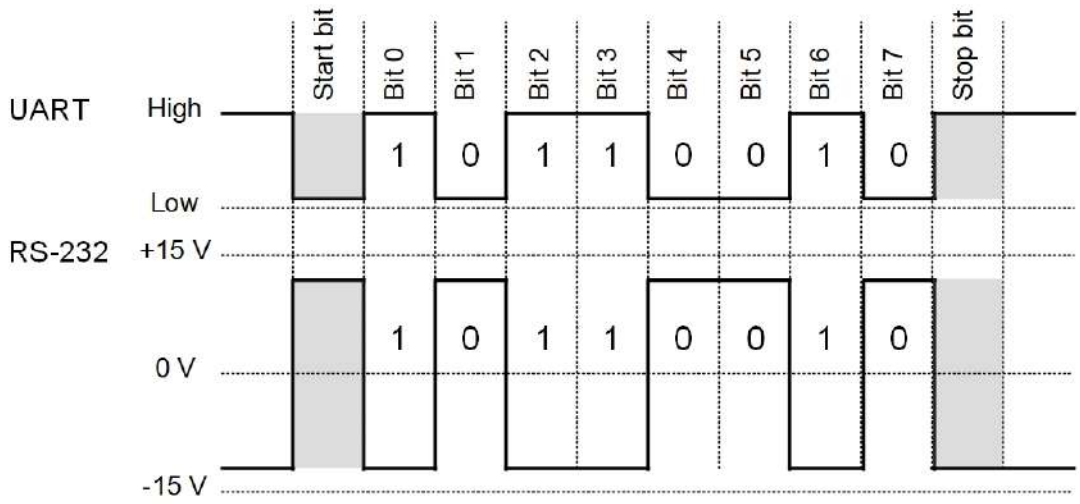


Рисунок 1.36 - Передача даних по інтерфейсу UART / RS-232

Процедура обміну даними між пристроями по інтерфейсу RS-232

Процедура обміну даними між пристроями по інтерфейсу RS-232 передбачає використання ліній протокольної взаємодії.

Таблиця 1.15 - Відповідність сигналів контактам для 9-ти і 25-ти контактних роз'ємів

Найменування	**Напрямок**	**Опис**	**Контакт (25-контактний роз'єм)**	**Контакт (9-контактний роз'єм)**
DCD	IN	Carrie Detect (Визначення несучої)	8	1
RXD	IN	Receive Data (Дані, що приймаються)	3	2
TXD	OUT	Transmit Data (Передані дані)	2	3

Продовження таблиці 1.15

Найменування	**Напрямок**	**Опис**	**Контакт (25-контактний роз'єм)**	**Контакт (9-контактний роз'єм)**
DTR	OUT	Data Terminal Ready (Готовність терміналу)	20	4
GND	-	System Ground (Корпус пристрою)	7	5
DSR	IN	Data Set Ready (Готовність даних)	6	6
RTS	OUT	Request to Send (Запит на відправку)	4	7
CTS	IN	Clear to Send (Готовність прийому)	5	8
RI	IN	Ring Indicator (Індикатор)	22	9

Лінії *«запит на пересилку»* (RTS) і *«ініціювання пересилання»* (CTS) використовуються зазвичай для управління *потоками даних*, що передаються між *комп'ютерами* (DCE) і *модемами* (DTE).

Після підготовки комп'ютера до передачі даних він активізує лінію RTS. Якщо *кінцевий пристрій* (DTE) готовий до прийому даних, він формує сигнал CTS. Якщо комп'ютер не в змозі прийняти дані (наприклад, через те, що його буфер заповнений або виконуються будь-які операції по обробці даних), сигнал

на лінії RTS видаватися не буде, тим самим відповідний пристрій повідомляється про неможливість прийому комп'ютером додаткової інформації.

Лінії *«готовність терміналу»* (DTR) і *«готовність модему»* (DSR) зазвичай застосовуються для підготовки *сеансу передачі даних*.

У випадку готовності до взаємодії з кінцевим пристроєм (DTE) комп'ютер видає в лінію DTR відповідний сигнал (повідомлення).

Якщо кінцевий пристрій може прийняти дані, він формує сигнал у лінії DSR для повідомлення комп'ютера про готовність до сеансу передачі даних.

При виникненні будь-яких помилок, пов'язаних з апаратними засобами, цей пристрій скасовує повідомлення у лінії DSR для повідомлення комп'ютера про виниклі проблеми. Аналогічним чином при зникненні *сигналу несучої* модеми скасують повідомлення DSR.

У лінії *«виявлення сигналу несучої»* (DCD) повідомлення формується, коли модемом встановлено зв'язок з іншими пристроями (модемом). За допомогою лінії «індикація сигналу виклику» (RI) комп'ютер інформується про генерацію *сигналів виклику*.

Ці лінії, поряд з лініями *«рукостискання»*, досить рідко використовуються у додатках з PIC-мікроконтролерами.

Пристрій передачі даних (DCE) і *кінцевий пристрій* (DTE) завжди пов'язані спільним («земляним») проводом. Ця лінія виявляється досить критичною для інтерфейсу RS-232, від неї залежить робота вхідних *перетворювачів рівнів*, за допомогою яких визначаються реальні логічні рівні вхідних напруг ліній.

Шина I²C

I²C - двопровідний інтерфейс, розроблений корпорацією Philips. У первісній технічній вимозі до інтерфейсу максимальна швидкість передачі даних становила 100 Кбіт/с. Однак з часом з'явилися стандарти на більш швидкісні режими роботи I^2C.

До однієї шині I^2C можуть бути підключені пристрої з різними швидкостями доступу, так як швидкість передачі даних визначається тактовим сигналом.

Протокол передачі даних розроблений таким чином, щоб гарантувати надійний прийом переданих даних.

Апаратна реалізація шини I^2C

При передачі даних один пристрій є «Master» або ведучий, який ініціює передачу даних і формує сигнали синхронізації.

Інший пристрій «Slave» або ведений, який починає передачу тільки по команді, що прийшла від «Master».

Тому важливо чітко визначити, який з пристроїв є головним, який задає правила і послідовність обміну, а який - підлеглим.

Така організація шини називається ***master-організацією*** і є найбільш типовим випадком (рис. 1.37).

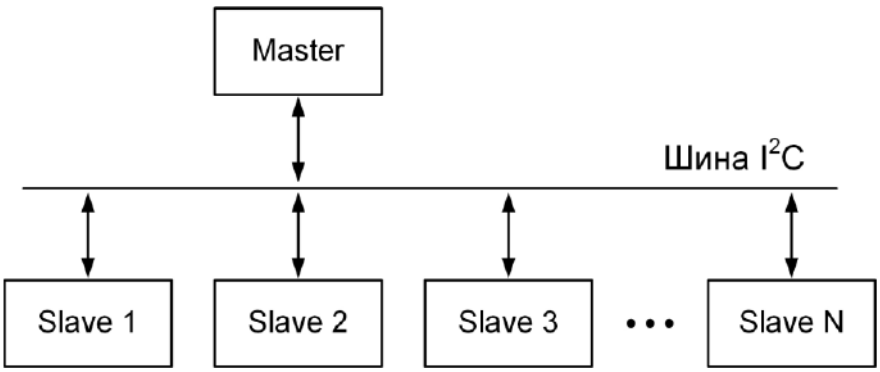


Рисунок 1.37 - Підключення пристроїв до шини I^2C

Master-пристроєм зазвичай призначається мікроконтролер. Саме він задає основний потік даних на шині, формує необхідні тимчасові інтервали і т.д.

Набагато рідше використовується в апаратурі режим multi-master коли до однієї шині підключено кілька master-пристроїв.

Складність такої організації шини складається головним чином в тому, що master-пристрої повинні вирішувати, хто з них буде працювати в даний момент зі slave-пристроями.

Одночасно на шині може здійснювати операції тільки один master-пристрій, інші зобов'язані відключатися. В іншому випадку виникає ситуація, звана ***шинним конфліктом***. Інформація може просто не дійти до адресата, порушиться робота пристрою.

Для того щоб виключити шинні конфлікти, у режимі multi-master повинні міститися процедури арбітражу і синхронізації, що встановлюють порядок роботи master-пристроїв.

Управляються всі ведені slave-абоненти по двох лініях (якщо не брати до уваги третю лінію - загальний провід схеми).

Slave-абонент повинен мати два виводи, які об'єднуються з такими ж виводами іншого slave-пристрою, а також з master-абонентом.

Концепція шини дуже проста, що дозволяє швидко розробляти принципову схему пристрою, випробувати її і у випадку необхідності «наростити» конструкцію новими елементами або видалити непотрібні безболісно для інших вузлів. Також може бути спрощений метод розробки програмного забезпечення, використані стандартні бібліотеки-підпрограми.

Практично всі мікросхеми з інтерфейсом I^2C мають такі характеристики, які дозволяють використовувати їх в низьковольтній портативній апаратурі з живленням від гальванічних елементів.

Наприклад, вони мають високу завадостійкість, низьке споживання струму, широкий діапазон живлячої напруги і слабку залежністю параметрів від температури навколишнього середовища.

Апаратна реалізація шини I^2C представлена на рис. 1.38.

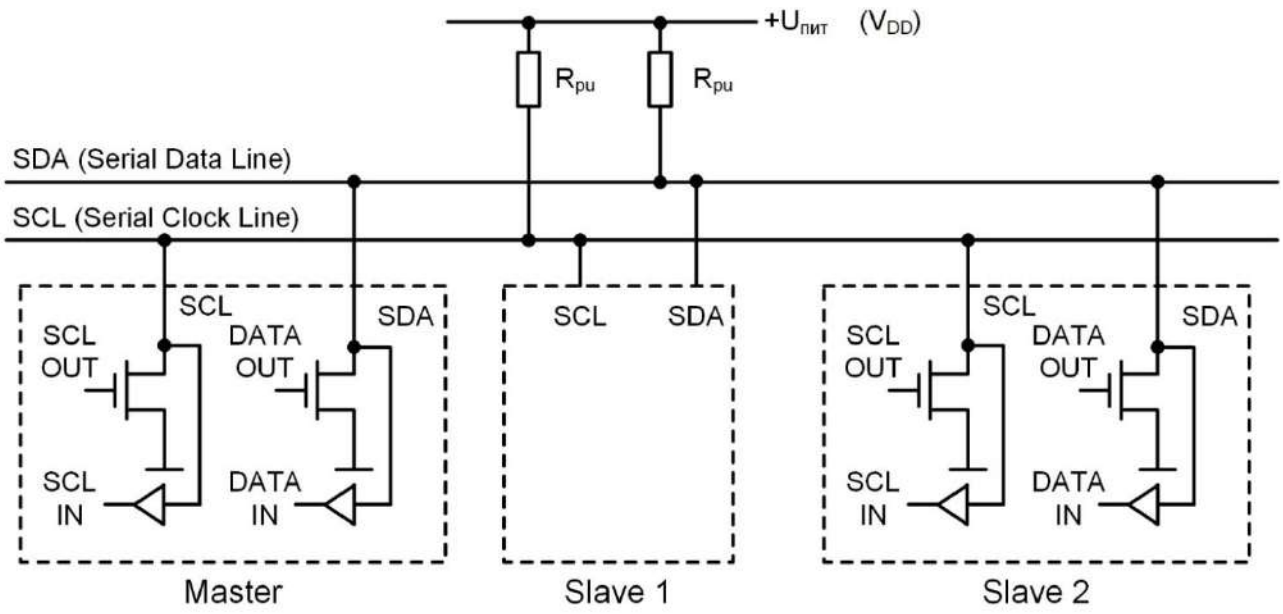


Рисунок 1.38 - Апаратна реалізація шини I^2C

Інтерфейс будь-якого пристрою, підключеного до шини I^2C складається з двох транзисторів з відкритим стоком (колектором) і двох буферних елементів з високим вхідним опором. Один з виводів названий **SDA** (**S**erial **Da**ta), призначається для зв'язку з ***лінією послідовних даних***. Другий вивід має скорочену назву **SCL** (**S**erial **Cl**ock), передбачений для зв'язку з ***лінією синхронізації***.

Як відомо, по будь-яких цифрових шинах передаються сигнали, що характеризуються лише двома електричними станами - "0" і "1" («низький рівень» і «високий рівень»). Стан, коли обидві лінії (SDA і SCL) встановлені у стан "1", ***вважається вільним шинним станом***. Шина в цей момент не зайнята і готова до трансляції даних (інакше кажуть, що шина знаходиться у стані очікування).

Для забезпечення логічних станів до шини підключаються два зовнішніх резистора R_{pu} (pull-up resistors), лінії, що «підтягують» до напруги живлення $U_{живл}$ Типова величина цих резисторів коливається у межах 1...10 кОм. Головна відмінність master-абонента від slave-абонента полягає в тому, що ***тільки master-абонент може генерувати сигнал SCL***.

Найважливішим критерієм, що визначає можливість використання тієї чи іншої шини, є її спектр технічних характеристик. Шина I^2C відноситься до класу двонаправлених асинхронних шин з послідовною передачею даних і, як наслідок, має досить низьку пропускну здатність. Основні технічні характеристики шини I^2C наведені у табл. 1.16.

Таблиця 1-16 - Основні технічні характеристики шини I^2C

Найменування параметра	**Значення параметра**
Швидкість обміну low-speed	не більше 100 кбіт/с
Швидкість обміну fast-speed	не більше 400 кбіт/с
Число адресованих пристроїв (7 біт)	до 128
Сумарна довжина ліній SCL і SDA	не більше 4 м
Сумарна паразитная ємність відносно загального проводу	не більше 400 пф
Вхідна ємність на кожен вивід абонента	не більше 10 пф

Шина I^2C не підходить для зв'язку між віддаленими абонентами, а значить, може бути використана тільки у складі одного приладу.

Ще однією важливою технічною характеристикою шини є її ***сумісність***. Раніше розроблені елементи, що володіють тільки можливостями низькошвидкісного обміну, повинні без проблем зв'язуватися з високошвидкісними, і навпаки.

Обмін даними може бути здійснений із швидкістю, доступною найповільнішому інтерфейсу.

Конфігурація Master-Slave і протокол передачі даних

Найпростішою конфігурацією шини I^2C є ***конфігурація Master-Slave (Ведучий-Ведений)***.

Передача будь-якого біту по шині відбувається за умови стробування даних SDA по лінії SCL. Припустимо, що master-пристрій виставив біт даних "0"

або "1" на лінію SDA. Slave-пристрій отримає цей біт тільки тоді, коли на лінії SCL відбудеться перепад сигналу з низького рівня у високий (так званий позитивний перепад). Тому зміна інформації на лінії SDA може бути здійснена тільки при нульовому стані лінії SCL (рис. 1.39).

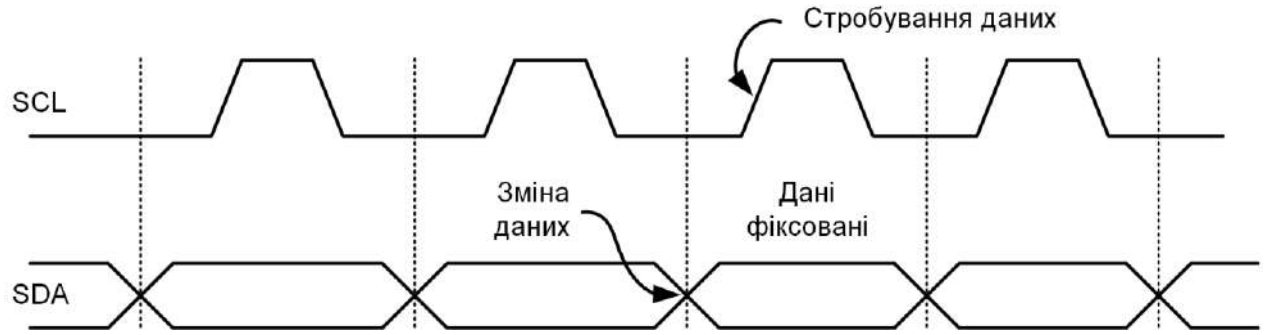


Рисунок 1.39 - Предача біту по шині I^2C

У неактивному стані шини на лініях SDA і SCL присутні високі рівні. Для розпізнавання початку і кінця передачі у специфікацію шини були введені умови Start і Stop.

На рис. 1.40 представлені умови **Start** і **Stop**. У фірмовій документації умова **Start** має умовне скорочення "**S**", умова **Stop** - "**P**".

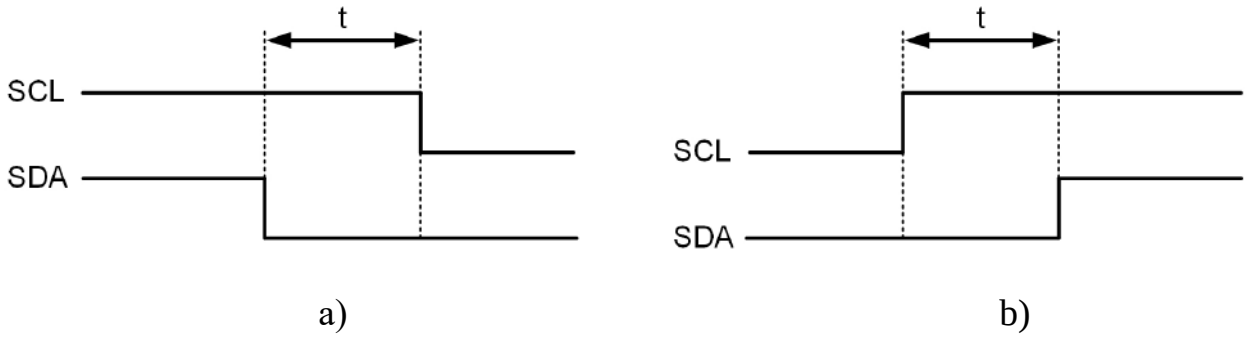


Рисунок 1.40 - a) - умова Start; b) - умова Stop

Умова **Start** утворюється при негативному перепаді сигналу на лінії SDA при одиничному стані лінії SCL. І навпаки, умова Stop виникає при позитивному перепаді лінії SDA при одиничному стані лінії SCL. Ці стани завжди повинні генеруватися master-пристроями.

Таким чином, інформаційний пакет, який передається по шині I²C, виглядає так, як показано на рис. 1.41.

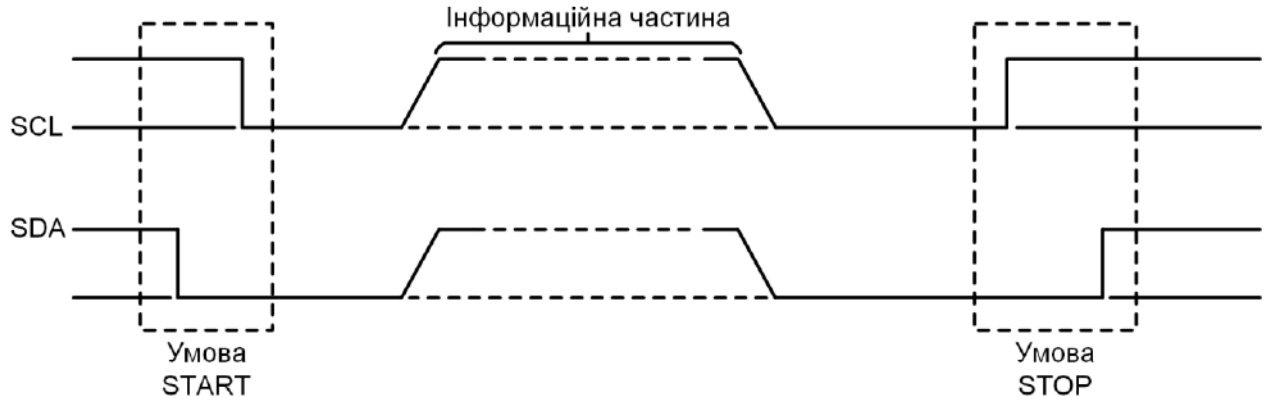


Рисунок 1.41 - Пакет даних, який передається по шині I²C

Для визначення станів Start і Stop у складі slave-пристрою зазвичай передбачається спеціальна апаратна схема, так як реалізувати програмно інтерфейс slave-абонента досить важко: з'являється необхідність постійної перевірки лінії SDA на предмет розпізнавання даних.

Існує також стан «повторний Start», який може виникнути у середині інформаційного пакета.

Передача даних по шині проводиться по 8 бітів, після чого слідує сигнал підтвердження (acknowledge, ACK, A). Сигнал підтвердження свідчить про те, що дані нормально прийняті і оброблені.

На рис. 1.42, представлений процес передачі байту по шині I²C.

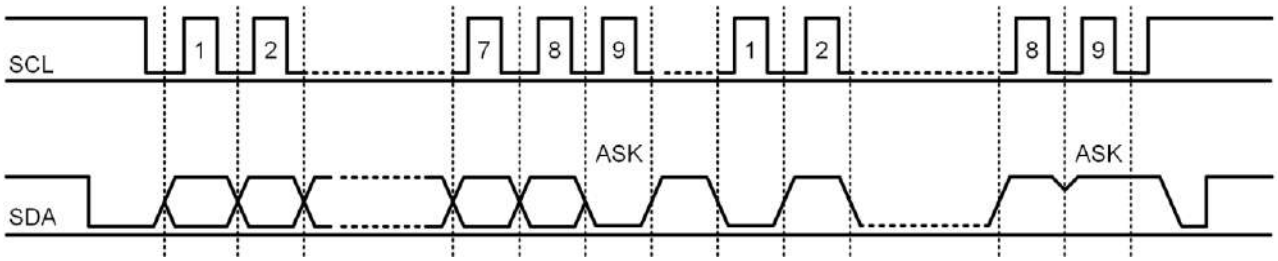


Рисунок 1.42 - Передача байта по шині I²C

Після відпрацювання стану Start передавач послідовно виставляє на лінії SDA дані, починаючи зі старшого біту (MSB) і закінчуючи молодшим (LSB). Дані стробуються по лінії SCL імпульсами 1...8.

Лінія SDA приймача (slave-абонента) у момент прийому інформаційних бітів (MSB-LSB) повинна бути виставлена в одиничний стан. Фізично це означає, що транзистор, підключений до лінії SDA, повинен бути закритий.

У момент негативного перепаду імпульсу 8 на лінії SCL slave-абонент повинен виставити на лінію SDA нульовий рівень - відкрити транзистор. Тим самим приймач підтверджує нормальний прийом байта. Передавач (master-абонент) повинен виставляти на лінію SDA одиничний стан.

Завдяки тому, що лінія організована за способом «монтажне І», її стан буде визначатися тільки slave-абонентом. Передавач повинен перевірити стан лінії SDA, потім видати дев'ятий стробуючий імпульс, з яким slave-абонент виставить на лінію SDA високий рівень, перевірити виконання цієї операції і лише після продовжити передачу.

Адресація пристроїв

Кожен пристрій визначається своєю унікальною адресою, у яку входить група приладів і номер конкретного приладу. Група визначає, чи є елемент мікроконтролером, LCD-індикатором, пам'яттю т.д.

Наприклад, усі пристрої пам'яті мають код 0x0A, таймери і годинник реального часу - 0x0D, пристрій телетексту - 0x02 і т.д.

Оскільки всі абоненти шини не мають інших способів спілкування, крім обміну даними по лініях SDA і SCL, в момент початку передачі всі slave-абоненти «слухають» лінію на предмет розпізнавання своєї slave-адреси. Упізнав свою адресу абонент продовжує прийом даних і видачу сигналів АСК, інші тільки стежать за моментом видачі стану Stop.

У зв'язку з цим можливі три формати передачі, що визначаються як 7-бітові (традиційні):

- master транслює дані на slave;
- master читає дані зі slave;
- комбінований формат трансляції/читання.

Історично склалося, що спочатку виник формат 7-бітної адресації, при якому здійснюється передача slave-адреси 7 бітами, а восьмий біт повинен містити ознаку операції «читання/запис» (R/W).

При 7-бітній адресації на шині можуть бути присутніми тільки 128 пристроїв з унікальними адресами.

На рис. 1.43 наведений формат передачі даних від master-пристрою до slave-абоненту.

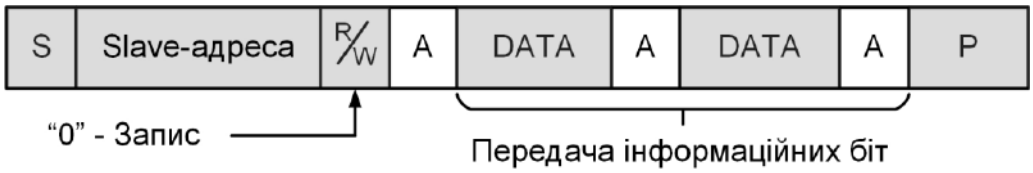


Рисунок 1.43 - Передача даних від master-пристрою до slave-абоненту.

Перший біт, передає slave-адресу пристрою, містить ознаку операції R/W. Коли цей біт встановлений у "0", буде здійснюватися запис у slave-пристрій, коли у "1" - читання з нього.

На рис. 1.44 наведений формат передачі даних від slave-абонента до master-пристрою.

Після читання інформаційного байту DATA master-абонент зобов'язаний підтвердити отримання байту сигналом ACK.

Рисунок 1.44 - Передача даних від slave-абонента до master-пристрою.

На рис. 1.45 показаний комбінований формат, що застосовується у випадках, коли здійснюється запис і читання в одному циклі.

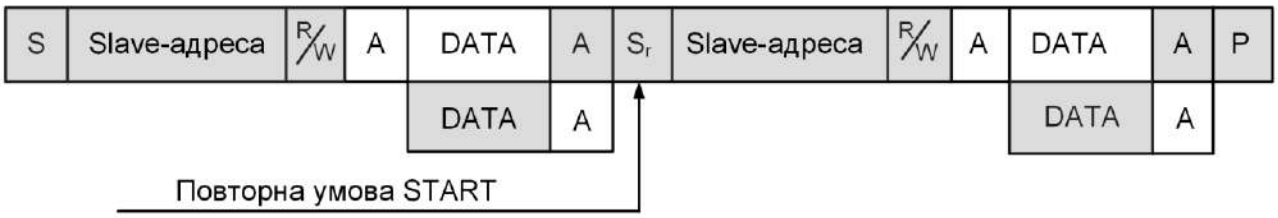


Рисунок 1.45 - Комбінований формат

Зазвичай комбіновані формати використовуються для роботи з послідовною пам'яттю (для скорочення часу доступу до даних). У комбінований формат введена повторна умова **Start** (repeated Start condition), що позначається на рисунку скороченням «Sr».

Програмний Start використовується тоді, коли пристрій, поєднаний з шиною I^2C, не має апаратного інтерфейсу. При цьому мікроконтролер зобов'язаний постійно опитувати шину на предмет отримання даних.

Якщо у мікроконтролері апаратно реалізований інтерфейс I^2C то досить запрограмувати мікроконтролер на переривання від шини, і при наявності даних на шині буде виконуватися програма обробки даних, що надходять з шини.

10-розрядна адресація шини I^2C

Оскільки форматом шини передбачена передача тільки «порції» з 8 байтів, 10-розрядні адреси доводиться транслювати 2 байтами. Перший байт має структуру 11110xx (R/W), в якому біти, позначені символом "x", є старшими розрядами 10-розрядної адреси slave-пристрою. У складі першого байту повинен бути переданий біт R/W.

Адресація за допомогою 10 розрядів аналогічна 7-розрядній адресації, тобто заснована на тих же принципах. Пристрій, отримавши службовий код у першому байті і упізнавши можливість прийому 10-розрядної адреси, підтверджує це і приймає другий байт. При збігу прийнятої адреси, з власною

адресою, що містяться всередині пристрою, видається підтвердження АСК і ведеться прийом даних у звичайному режимі до появи стану Stop.

На рис. 1.46 показаний формат передачі даних від master-абонента до slave-пристрою.

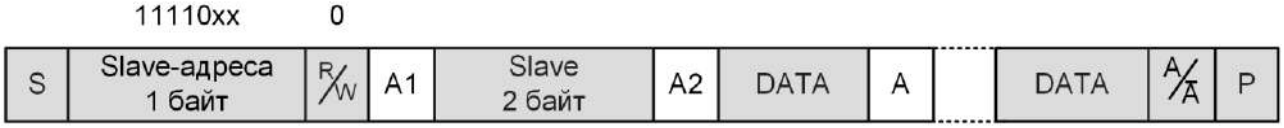


Рисунок 1.46 - Передача даних від master-абонента до slave-пристрою.

Slave - абоненти порівнюють перший отриманий байт зі своїми даними і, якщо вони збігаються, видають сигнал АСК. Очевидно, відразу кілька пристроїв можуть видати підтвердження. Тому при отриманні другого байта тільки один пристрій видасть сигнал АСК (А2). Решта пристроїв, які раніше видали сигнал АСК (А1), залишаться адресованими за першим байтом до отримання сигналу Stop.

На рис. 1.47 наведений формат передачі даних від slave-абонента до master-пристрою.

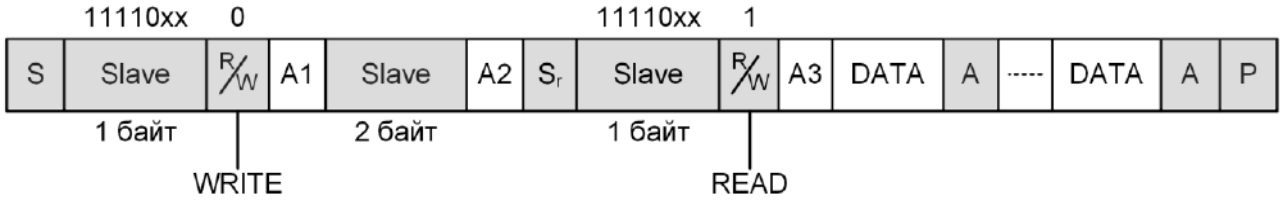


Рисунок 1.47 - Передача даних від slave-абонента до master-пристрою

Спочатку master-пристрій першим байтом адресує всі slave-абоненти відповідним кодом, і вони підтверджують його отримання сигналом АСК (А1). Потім другим байтом адресується конкретний пристрій з видачею сигналу АСК (А2).

Після виконання повторної умови Start (Sr) адресований slave-пристрій зберігає можливість звертатися до нього, тому досить повторити перший байт адреси, але вже з іншим значенням біту R/W і отримати АСК (А3).

Інтерфейс SPI

SPI (англ. Serial Peripheral Interface, SPI bus - послідовний периферійний інтерфейс, шина SPI) - послідовний периферійний інтерфейс (шина), служить для зв'язку мікроконтролера і периферійних пристроїв. Розроблений компанією Motorola.

Шина SPI використовує архітектуру Master-Slave. В якості ведучого шини зазвичай виступає мікроконтролер, але ним також може бути програмована логіка, DSP-контролер або спеціалізована IC.

Пристрої SPI взаємодіють у повнодуплексному режимі. Ведучий ініціює кадр для читання і запису.

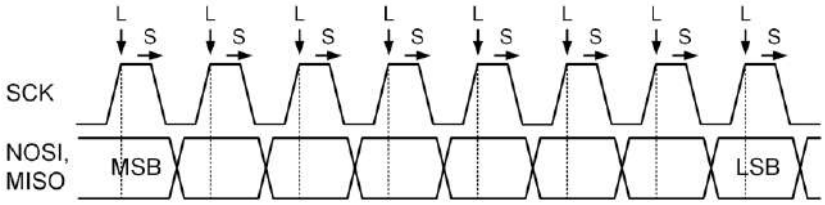
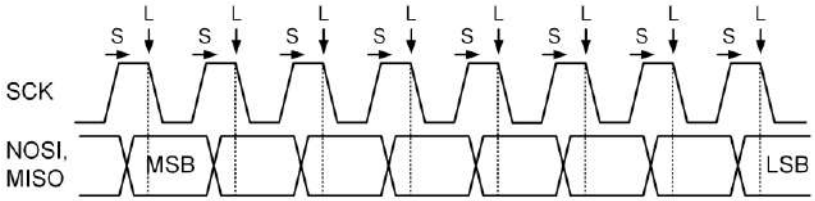
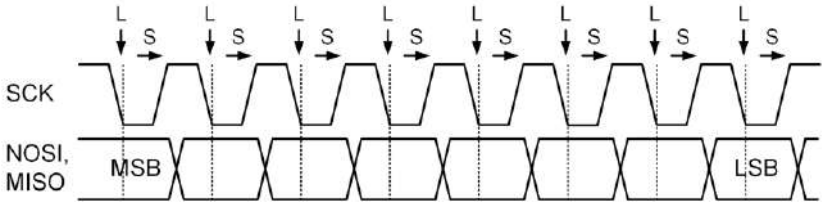
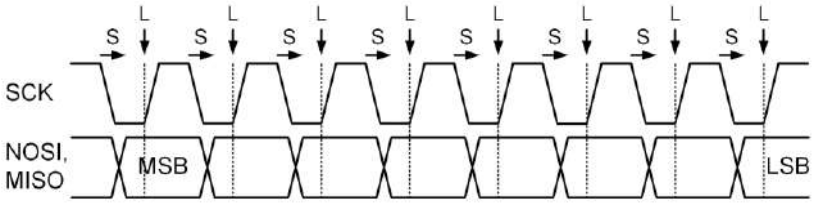
Опис інтерфейсу

У SPI використовуються чотири цифрових сигнали:

- **MOSI** або **SI** - вихід ведучого, вхід веденого
  (англ. Master Out Slave In).
  Служить для передачі даних від ведучого пристрою веденому;
- **MISO** або **SO** - вхід ведучого, вихід веденого
  (англ. Master In Slave Out).
  Служить для передачі даних від веденого пристрою ведучому;
- **SCK** або **SCLK** - послідовний тактовий сигнал
  (англ. Serial Clock).
  Служить для передачі тактового сигналу для ведених пристроїв;
- **CS** або **SS** - вибір мікросхеми, вибір веденого
  (англ. Chip Select, Slave Select).
  Як правило, вибір мікросхеми здійснюється низьким логічним рівнем.

В залежності від комбінацій полярності і фази синхроімпульсів можливі чотири режими роботи SPI (табл. 1.17).

Таблиця 1.17 - Режими роботи SPI

Режим SPI	Тимчасова діаграма
Режим SPI0 Активні рівень імпульсів - високий. Спочатку замикання, потім зсув.	
Режим SPI1 Активні рівень імпульсів - високий. Спочатку зсув, потім замикання.	
Режим SPI2 Активні рівень імпульсів - низький. Спочатку замикання, потім зсув.	
Режим SPI3 Активні рівень імпульсів - низький. Спочатку зсув, потім замикання.	

У таблиці прийнято:

- **MSB** - старший біт;
- **LSB** - молодший біт.

Ведучий настроюється на той режим, який використовується веденим. При обміні даними по інтерфейсу SPI мікроконтролер може працювати як ведучий (режим Master) або як ведений (режим Slave).

При цьому користувач може задавати такі параметри:

- режим роботи відповідно до таблиці;
- швидкість передачі;
- формат передачі (від молодшого біту до старшого або навпаки).

Операції

Шина SPI може працювати з одним ведучим пристроєм та з одним або декількома веденими пристроями. Якщо використовується один ведений пристрій, то SS може бути зафіксований на низькому логічному рівні, якщо це дозволяє ведений пристрій. Деякі ведені пристрої вимагають зниження сигналу чіпу для початку дій. З численними веденими пристроями, незалежний сигнал SS необхідний від ведучого для кожного веденого пристрою.

Більшість ведених пристроїв мають три стани виходів, тому їх сигнал MISO буде високо імпедансним, коли пристрій не вибраний.

Передача даних

З'єднання двох пристроїв по інтерфейсу SPI здійснюється відповідно до рис. 1.48.

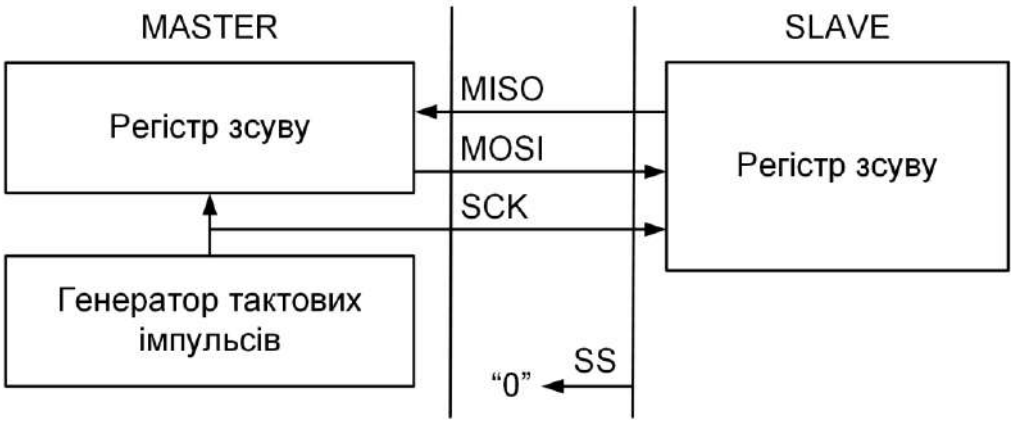


Рисунок 1.48 - З'єднання двох пристроїв по інтерфейсу SPI

Рівень регістрів зсуву інтерфейсу SPI представлений на рис. 1.49.

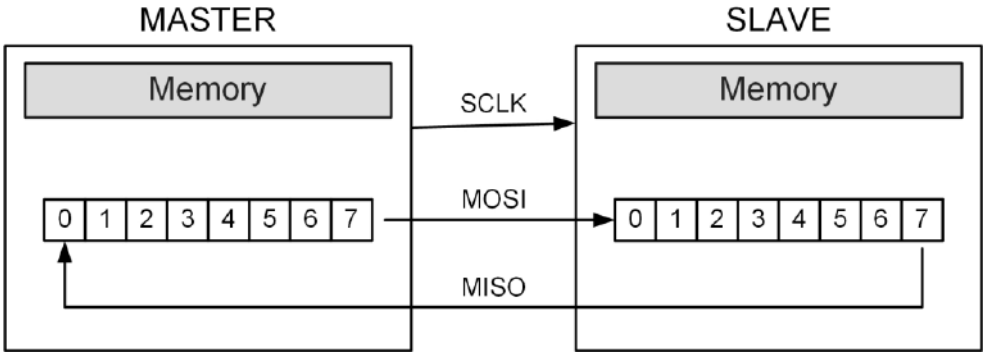


Рисунок 1.49 - Рівень регістрів зсуву інтерфейсу SPI

Протокол передачі по інтерфейсу SPI ідентичний логіці роботи зсувного регістру, яка полягає у виконанні операції зсуву і, відповідно, побітного введення і виведення даних за визначеними фронтами сигналу синхронізації.

Встановлення даних при передачі і вибірка при прийомі завжди виконуються по протилежних фронтах синхронізації. Це необхідно для гарантування вибірки даних після надійного їх встановлення.

Сигнали управління

Можливі чотири комбінації фази (CPHA) і полярності (CPOL) сигналу SCLK по відношенню до сигналів даних.

Режими роботи визначаються комбінацією біт CPHA і CPOL:

- **CPOL = 0** - сигнал синхронізації починається з низького рівня;
- **CPOL = 1** - сигнал синхронізації починається з високого рівня;
- **CPHA = 0** - вибірка даних проводиться по передньому фронту сигналу синхронізації;
- **CPHA = 1** - вибірка даних проводиться по задньому фронту сигналу синхронізації.

Для позначення режимів роботи інтерфейсу SPI прийнято наступну угоду:

- режим 0 (CPOL = 0, CPHA = 0);
- режим 1 (CPOL = 0, CPHA = 1);

- режим 2 (CPOL = 1, CPHA = 0);
- режим 3 (CPOL = 1, CPHA = 1).

CPOL - вихідний рівень сигналу синхронізації (якщо **CPOL = 0**, то лінія синхронізації до початку циклу передачі і після його закінчення має низький рівень (тобто перший фронт наростаючий, а останній - падаючий), інакше, якщо CPOL = 1, - високий (тобто перший фронт падаючий, а останній - наростаючий)).

CPHA - фаза синхронізації; від цього параметра залежить, в якій послідовності виконується встановлення і вибірка даних (якщо **CPHA = 0**, то по передньому фронту в циклі синхронізації буде виконуватися вибірка даних, а потім, по задньому фронту, - встановлення даних; якщо ж **CPHA = 1**, то встановлення даних буде виконуватися по передньому фронту у циклі синхронізації, а вибірка - по задньому).

Ведучі і ведені пристрої, що працюють у різних режимах SPI, є несумісними, тому, перед вибором ведених мікросхем важливо уточнити, які режими підтримуються ведучим шини.

Апаратні модулі SPI, інтегровані у мікроконтролери, в більшості випадків підтримують можливість вибору будь-якого режиму SPI і, тому, до них можливе підключення будь-яких ведених SPI-пристроїв. Крім того, протокол SPI у будь-якому з режимів легко реалізується програмно.

Топологія систем зв'язку на базі SPI

У найпростішому випадку до ведучого пристрою приєднаний єдиний ведений пристрій і необхідний двосторонній обмін даними. В такому випадку використовується трьохпровідна схема підключення.

Інтерфейс SPI дозволяє підключати до одного ведучого пристрою кілька ведених пристроїв, причому підключення може бути здійснено декількома способами, як показано на рисунках 1.50 і 1.51.

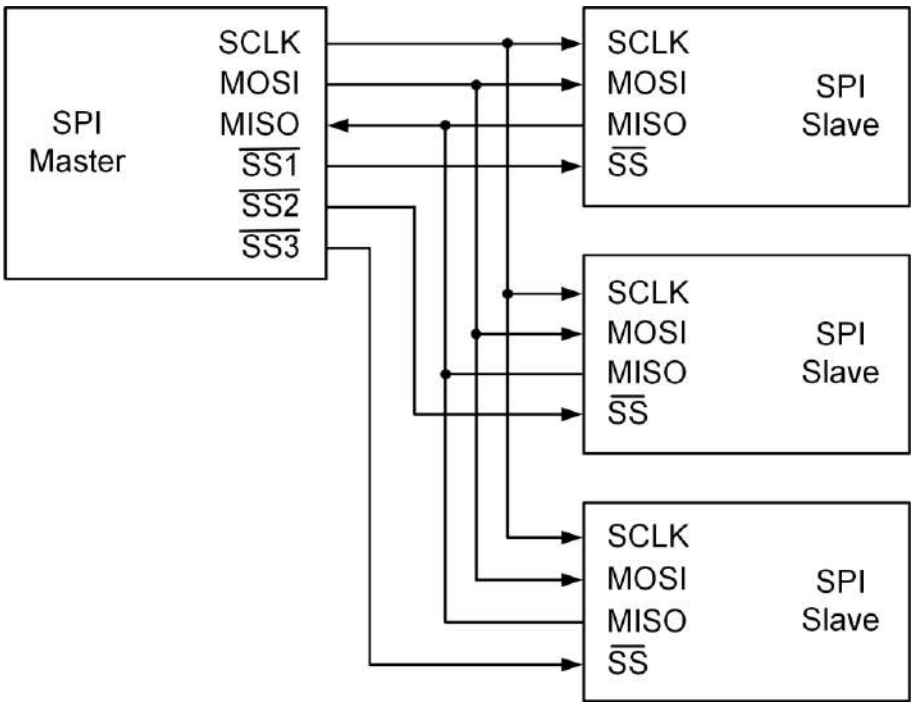


Рисунок 1.50 - Радіальна структура зв'язку з декількома веденими пристроями через SPI

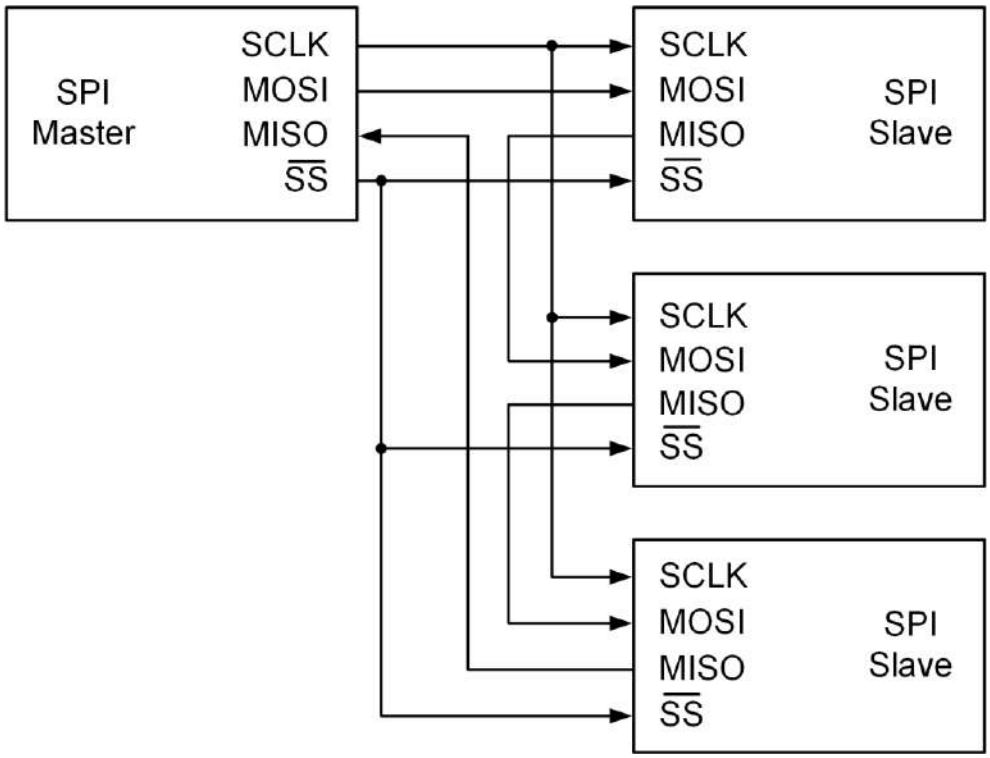


Рисунок 1.51 - Кільцева структура зв'язку
з декількома веденими пристроями через SPI

Перший спосіб дозволяє реалізувати радіальну структуру зв'язку (топологія типу «зірка»), його прийнято вважати основним способом підключення декількох ведених пристроїв.

У даному випадку для обміну більш ніж з одним веденим пристроєм ведучий пристрій має формувати відповідну кількість сигналів вибору веденого пристрою **Slave Select** (SS).

При обміні даними з веденим пристроєм, відповідний йому сигнал SS переводиться в активний (низький) стан, при цьому всі інші сигнали SS знаходяться в неактивному (високому) стані.

Виводи даних MISO ведених пристроїв з'єднані паралельно, при цьому вони знаходяться у неактивному стані, а перед початком обміну один з виходів (обраного веденого пристрою) переходить у активний режим.

Другий спосіб дозволяє реалізувати структуру зв'язку типу «кільце». У даному випадку для активації одночасно декількох ведених пристроїв використовується один сигнал SS, а виводи даних всіх пристроїв з'єднані послідовно і утворюють замкнений ланцюг.

При передачі пакета від ведучого пристрою цей пакет отримує перший ведений пристрій, який, в свою чергу, транслює свій пакет наступному веденому пристрою і так далі. Для того, щоб пакет від ведучого пристрою досяг визначеного пристрою, ведучий пристрій має відправити ще декілька пакетів.

Синхронізація у SPI

Частота слідування бітових інтервалів в лініях передачі даних визначається синхросигналом SCK, який генерує ведучий пристрій, ведені пристрої використовують синхросигнал для визначення моментів зміни бітів на лінії даних, при цьому ведені пристрої ніяк не можуть впливати на частоту слідування бітових інтервалів.

Як у ведучому пристрої, так і у веденому пристрої є лічильник імпульсів синхронізації (бітів).

Лічильник у веденому пристрої дозволяє останньому визначити момент закінчення передачі пакета. Лічильник скидається при виключенні підсистеми SPI, така можливість завжди є у ведучому пристрої.

У веденому пристрої лічильник зазвичай скидається деактивацією інтерфейсного сигналу SS.

Так як дії ведучого і веденого пристрою тактуються одним і тим же сигналом, то до стабільності цього сигналу не пред'являється ніяких вимог, за винятком обмеження на тривалість напівперіодів, яка визначається максимальною робочою частотою більш повільного пристрою.

Це дозволяє використовувати SPI у системах з нестабільною тактовою частотою, а також полегшує програмну емуляцію ведучого пристрою.

Переваги та недоліки інтерфейсу SPI

Переваги:

1) повнодуплексна передача даних за замовчуванням;
2) більш висока пропускна здатність у порівнянні з I²C або SMBus;
3) можливість довільного вибору довжини пакету, довжина пакета не обмежена вісьмома бітами;
4) простота апаратної реалізації:

- більш низькі вимоги до енергоспоживання у порівнянні з I²C і SMBus;
- можливе використання у системах з нізькостабільною тактовою частотою;
- веденим пристроям не потрібна унікальна адреса, на відміну від таких інтерфейсів, як I²C, GPIB або SCSI;

5) використовується тільки чотири виводи, що набагато менше, ніж для паралельних інтерфейсів;
6) односпрямований характер сигналів дозволяє при необхідності легко організувати гальванічну розв'язку між ведучим і веденими пристроями;

7) максимальна тактова частота обмежена тільки швидкодією пристроїв, що беруть участь в обміні даними.

Недоліки:

1) необхідно більше виводів, ніж для інтерфейсу I²C;
2) ведений пристрій не може керувати потоком даних;
3) немає підтвердження прийому даних з боку веденого пристрою (ведучий пристрій може передавати дані «в нікуди»);
4) немає визначеного стандартом протоколу виявлення помилок;
5) відсутність офіційного стандарту, що робить неможливим сертифікацію пристроїв;
6) за дальності передачі даних інтерфейс SPI поступається таким стандартам, як UART і CAN;
7) наявність безлічі варіантів реалізації інтерфейсу;
8) відсутність підтримки гарячого підключення пристроїв.

Інтерфейс 1-Wire

Інтерфейс 1-Wire був запропонований фірмою Dallas Semiconductor у кінці 90-х років минулого століття.

На основі цього інтерфейсу створюються шини, магістралі та мережі.

Системи 1-Wire привабливі завдяки легкості монтажу, низькій вартості пристроїв, можливості вибирати користувача при підключенні до функціонуючої мережі, великому числу пристроїв у мережі і т.д.

Типова система 1-Wire складається з керуючого мікроконтролера (майстра або ведучого) і одного або декількох пристроїв (ведених), приєднаних до загальної шини (рис.1.52).

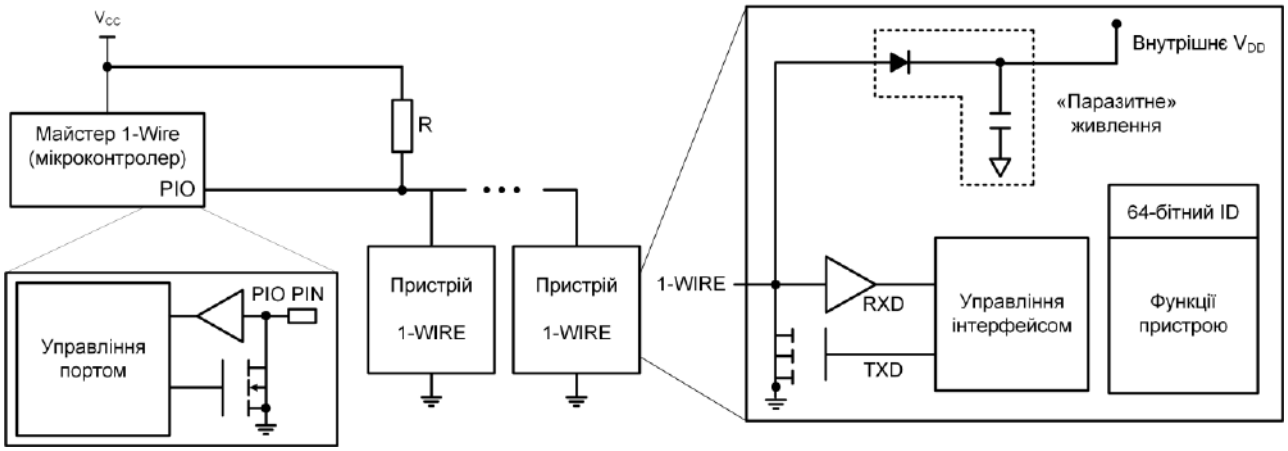


Рисунок 1.52- Архітектура інтерфейсу 1-Wire

Пристрої підключаються до шини за схемою з відкритим стоком і підтягуючим резистором. Рівень сигналів у шині - від 3 до 5 В.

У пасивному стані у лінії підтримується високий рівень напруги. Всі сигнали формуються за допомогою замикання сигнальної шини на землю (низький рівень напруги).

Основні параметри інтерфейсу 1-Wire

- максимальна довжина магістралі при використанні крученої пари - до 300 м;
- максимальна кількість абонентів на магістралі максимальної довжини - до 250;
- швидкість обміну по магістралі максимальної довжини - до 16,3 кбіт/с.

Головна особливість шини 1-Wire в тому, що вона використовує лише два дроти, один - сигнальний, інший - для заземлення пристроїв.

По сигнальному проводу можливо і електроживлення пристроїв 1-Wire - так зване паразитне живлення.

Джерелом живлення служить конденсатор, що заряджається від сигнальної лінії, що входить до складу ведених пристроїв ланцюга.

Протокол обміну даними по шині 1-Wire

Обмін даними по шині 1-Wire включає три фази (рис. 1.53):

- фазу скидання, що включає імпульс скидання від мікроконтролера і у відповідь імпульс підтвердження присутності від абонента (абонентів);
- фазу вибірки пристрою, що включає команду його вибірки (за кодом, без коду, групову, пошуку) і його код, якщо командою він передбачений;
- фазу запису/читання даних, що включає код команди і дані.

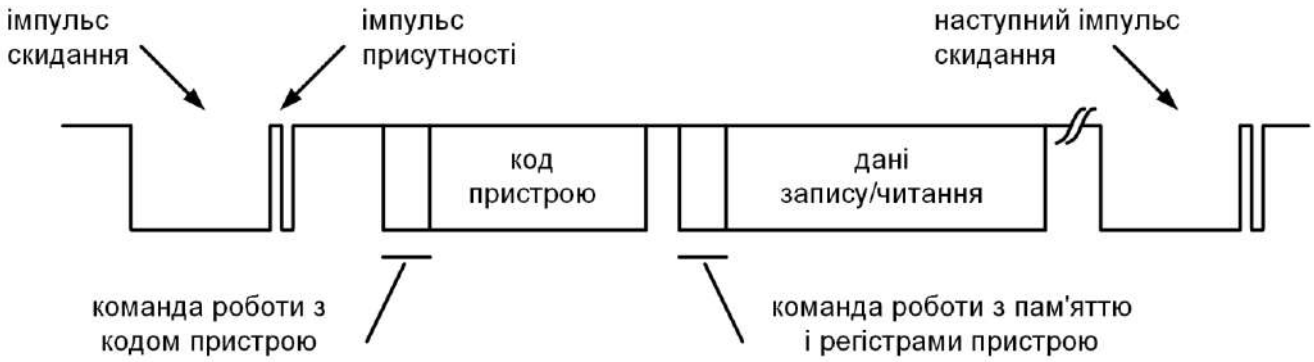


Рисунок 1.53 - Обмін даними по шині 1-Wire

Логіка всіх пристроїв тактується негативним фронтом сигналів мікроконтролера як в режимі запису, так і в режимі читання.

Біти кодуються тривалістю позитивного імпульсу:

- "1" передається довгим імпульсом;
- "0" - коротким.

У режимі запису всі імпульси даних формуються мікроконтролером.

У режимі читання мікроконтролер формує послідовність імпульсів синхронізації, у відповідь на кожен імпульс абонент виставляє на шину біт даних (рис. 1.54).

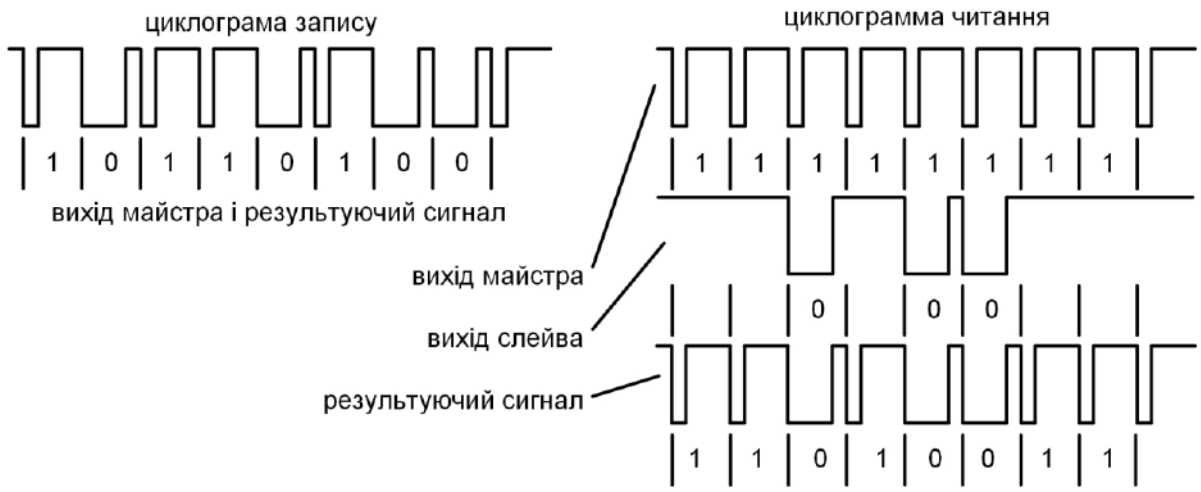


Рисунок 1.54 - Читання майстром даних від веденого

Режим передачі даних по шині 1-Wire - напівдуплексний: майстер і ведені пристрої передають дані по черзі. Кожна транзакція через інтерфейс 1-Wire починається з того, що майстер передає імпульс Reset. Для цього він переводить напругу у шині на низький рівень і утримує її у цьому стані протягом 480 мкс (рис. 1.55).

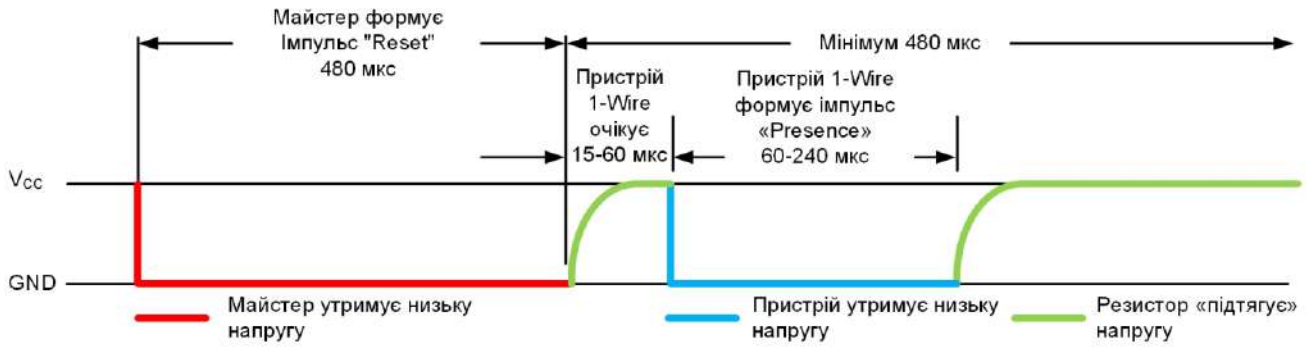


Рисунок 1.55 - Послідовність ініціалізації шини 1-Wire

Потім майстер відпускає шину, і підтягуючий резистор повертає напругу до високого логічного рівня. Всі ведені пристрої, виявивши сигнал Reset і дочекавшись його закінчення, передають свій сигнал - Presence. Він являє собою сигнал низького рівня тривалістю 100-200 мс.

Пристрій може генерувати сигнал Presence і без імпульсу Reset - наприклад, у такий спосіб він повідомляє про себе при підключенні до шини. Після передачі імпульсу Presence пристрій 1-Wire готовий до прийому команд.

Весь інформаційний обмін у шині відбувається під керуванням майстра. Для передачі кожного біту виділяється спеціальний часовий проміжок (таймслот) тривалістю близько 80 мкс.

На початку кожного тайм-слоту майстер переводить лінію на нульовий рівень. Якщо далі майстер хоче передати 0, він утримує напругу на низькому рівні як мінімум 60 мкс (рис.1.56а).

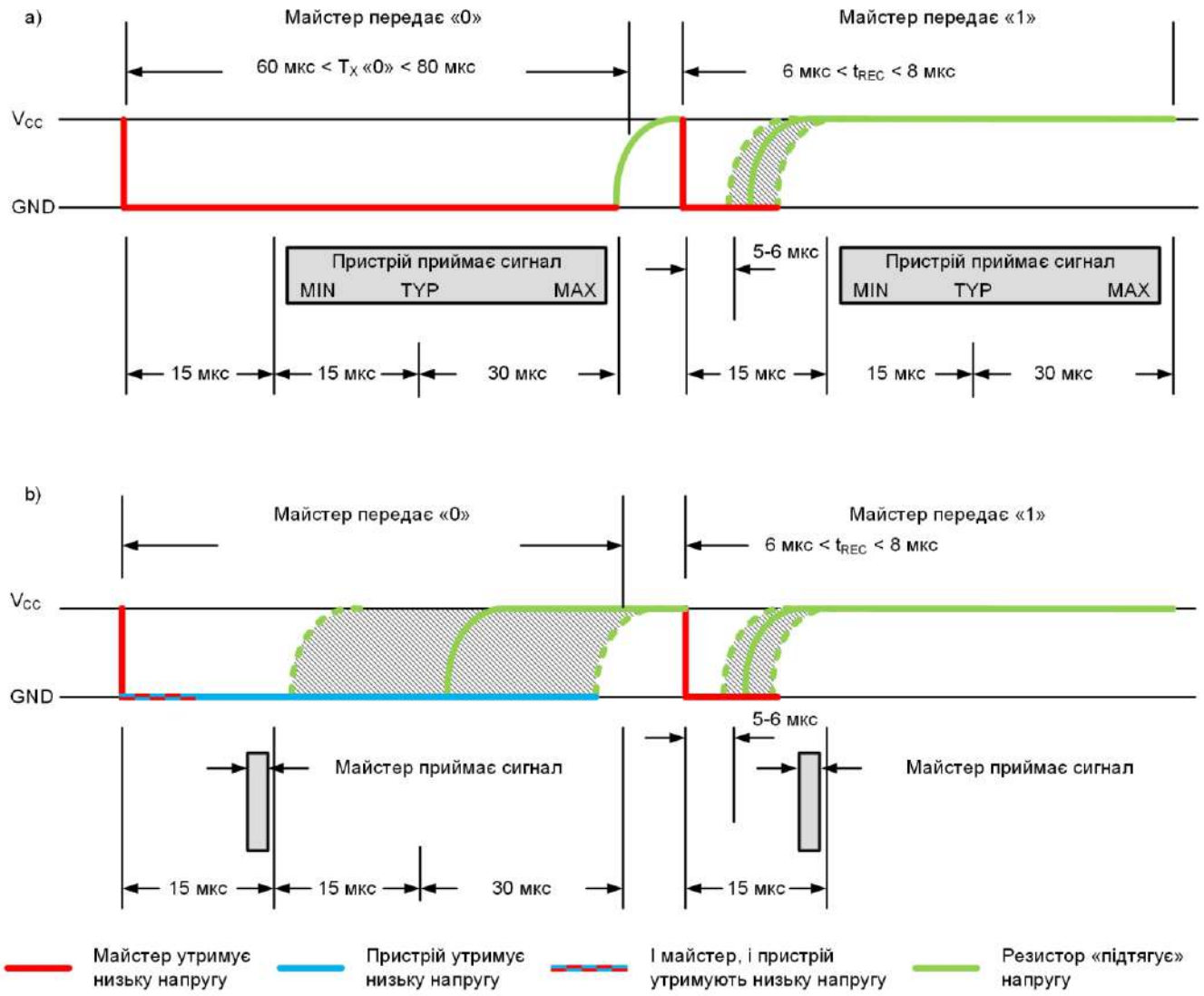


Рисунок 1.56 - Передача інформаційних бітів по шині 1-Wire:

a) - майстер передає сигнали;

b) - майстер зчитує сигнали

При передачі одиниці майстер утримує нульову напругу 5-6 мкс, а потім відпускає лінію і вичікує приблизно 60 мкс до початку формування наступного тайм-слоту. Якщо майстер очікує дані від ведених пристроїв, він також позначає початок тайм-слоту, обнуляючи лінію на 5-6 мкс, після чого перестає утримувати низьку напругу і протягом короткого часу слухає лінію (рис.1.56b).

Якщо пристрій хоче передати нуль, він сам обнуляє лінію відразу після реєстрації імпульсу початку тайм-слоту. Якщо пристрою потрібно передати одиницю, він ніяких дій не робить. Наведені значення тимчасових інтервалів відповідають стандартній швидкості передачі даних через інтерфейс 1-Wire. У режимі ***overdrive*** ці інтервали відповідно зменшуються.

Весь обмін на шині 1-Wire відбувається за допомогою спеціальних команд. Їх число для кожного типу пристроїв різне. Але є і мінімальний набір стандартних команд, які підтримують всі 1-Wire-пристрої - так звані ROM-команди.

Формат команд простий - ідентифікатор команди (1 байт), за яким можуть слідувати дані (ідентифікатор пристрою, корисні дані і т.п.). Всі пристрої в мережі знають довжину кожної команди.

Таблиця 1.18 - Інформація про значення бітів у адресах пристроїв 1-Wire

Істинний біт	**Інверсний біт**	**Інформація**
0	0	У поточному биті адрес є як "0", так і "1". Це так звана «розбіжність» (discrepancy)
0	1	У биті адрес присутні тільки нулі
1	0	У биті адрес присутні тільки одиниці
1	1	У пошуку не бере жоден пристрій

У кожного пристрою 1-Wire є 64-розрядний ідентифікатор (ID). Він складається з 8-розрядного коду сімейства, який ідентифікує тип пристрою і підтримувані ним функції, 48-розрядного серійного номера і 8-бітного поля коду циклічного надлишкового контролю (CRC-8).

ID вводиться при виготовленні пристрою і зберігається у ПЗП. Фірма Maxim гарантує, що один раз використана адреса ніколи не повториться в іншому пристрої.

Весь обмін командами ініціює майстер. Початок нового циклу транзакцій він зазначає командою Reset, і, отримавши підтвердження, вибирає пристрій спеціальною командою MATCH ROM, передаючи її ідентифікатор (5516) і 64 біту ID пристрою, що адресується.

Отримавши таку команду, ведений пристрій з даними ID очікує нових команд від майстра, а всі інші залишаються в пасивному стані до наступної команди Reset.

В системі з одним пристроєм можна не передавати ID, використовуючи команду SKIP ROM. В результаті ведений пристрій вважає себе обраним без отримання адреси.

Після того, як майстер вибрав пристрій для взаємодії, можна починати процес управління цим пристроєм і обміну даними з ним. Для цього використовуються команди, які специфічні для кожного типу пристроїв.

Але щоб почати роботу з визначеним пристроєм, майстер повинен знати його ID. Якщо у системі тільки один відомий пристрій, його адресу можна визначити за допомогою команди READ ROM. У відповідь на команду READ ROM пристрій передає свою 64-бітову адресу (рис.1.57).

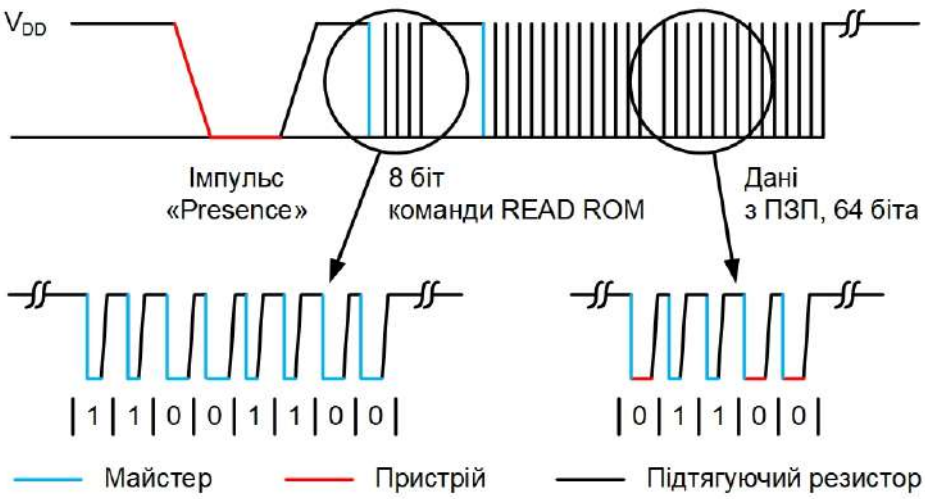


Рисунок 1.57 - Читання адреси пристрою

Якщо ж у системі декілька пристроїв з невідомими ID, спроба використовувати команду READ ROM призводить до колізії. У цьому випадку для визначення адрес використовується спеціальний алгоритм пошуку, в основі якого лежить команда SEARCH ROM.

Майстер передає команду SEARCH ROM. У відповідь всі пристрої, підключені до шини, висилають молодший біт своєї адреси.

Властивості шини 1-Wire такі, що при одночасній передачі сигналів усіма пристроями результат буде дорівнювати логічній "1" значень всіх посланих бітів.

Отже, сумарний відгук дорівнює "1", тільки коли сигнали від всіх пристроїв рівні "1". Після прийому першого біту адреси майстер ініціює наступний тайм-слот, у якому пристрій передає інвертований перший біт.

Зіставляючи значення результатів запиту істинного і інверсного бітів, можна отримати якусь інформацію про значення перших бітів адрес пристроїв (таб. 1.18). Таким чином, при комбінаціях "01" і "10" майстер знає значення першого біту адреси, фіксує його і по тій же схемі може переходити до визначення наступного.

Після отримання інверсного біту майстер передає визначений біт веденим пристроям.

Якщо його значення збігається зі значенням поточного біту з адреси пристрою, то пристрій продовжує брати участь у пошуку і видає у відповідь наступний біт своєї адреси.

Якщо не було «розбіжності», то значення виставляється майстром біту визначено. У випадку розбіжності майстер посилає нульовий біт.

Така послідовність - читання біту адреси та інверсного біта, передача біту майстром - повторюється для наступних 63 бітів адреси.

Таким чином, алгоритм пошуку послідовно виключає всі пристрої, поки не залишається один останній - його адреса і визначається в першому циклі пошуку.

Після того, як адресу першого пристрою визначено, пошук триває для наступного пристрою.

Алгоритм запам'ятовує місце останньої розбіжності і вибирає іншу гілку дерева пошуку (майстер посилає в цьому місці біт з іншим значенням). Процес триває до тих пір, поки не буде пройдена гілка, відповідна останньому пристрою.

В результаті пошуку стають відомі адреси усіх пристроїв, під'єднаних до шини, і їх число. Це означає, що прилад досить просто підключити до мережі, і всі подальші транзакції відбудуться автоматично.

Наприклад, так можна рахувати дані з пам'яті датчика, прочитати код електронної мітки або електронного ключа, прийняти масив значень від приладової мережі і т.п.

Інтерфейс 1-Wire відноситься до самосинхронізующихся, тобто не вимагає окремої лінії для передачі тактових сигналів.

Елементна база

Для реалізації інтерфейсу 1-Wire фірма Maxim/Dallas пропонує ряд пристроїв.

У ролі майстра інтерфейсу може виступати як ПК, так і спеціалізовані мікроконтролери.

У асортименті Maxim/Dallas присутні пристрої, які забезпечують перехід до 1-Wire інтерфейсу від стандартних інтерфейсів комп'ютера (наприклад, USB і RS-232), який керує роботою пристроїв 1-Wire.

Так, наприклад, чіпи MCP2221A або CY7C65211 служать мостом між інтерфейсами USB і 1-Wire.

Деякі пристрої можуть не тільки з'єднувати 1-Wire з іншими інтерфейсами, але також брати на себе частину функцій майстра шини 1-Wire.

Один з таких пристроїв - чіп DS2482-100. Це міст між інтерфейсами I^2C і 1-Wire.

Чіп DS2482-100 перетворює протоколи між керуючим I^2C мікро-контролером (майстром) і веденими 1-Wire пристроями (рис.1.58).

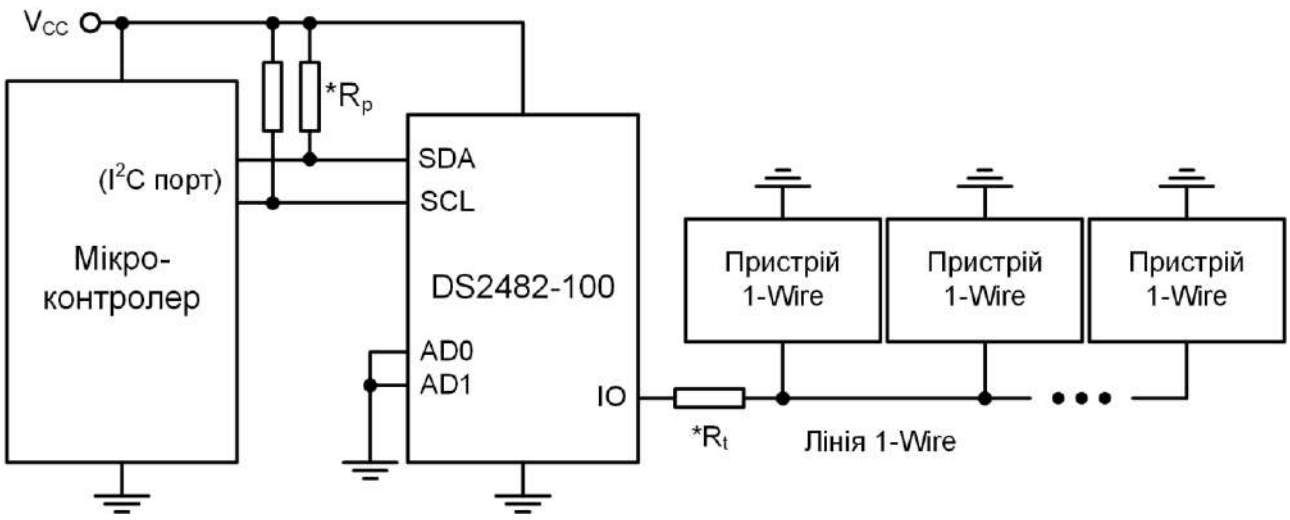


Рисунок 1.58 - Схема підключення DS2482-100.

де:

SDA - вхід/вихід даних шини I^2C;

SDL - вхід синхронізації шини I^2C;

AD0, AD1 - адресні входи I^2C.

По відношенню до цих пристроїв чіп DS2482-100 виступає у ролі майстра. За допомогою вбудованих таймерів DS2482-100 формує фронти переданих по шині сигналів, знімаючи це навантаження з керуючого мікроконтролера.

Для того щоб оптимізувати форму сигналів, чіп DS2482-100 контролює швидкості наростання і зменшення напруги в лінії і надає додаткові програмовані функції, які допомагають узгодити параметри сигналів і особливості ведених пристроїв.

Застосування інтерфейсу 1-Wire і програмне забезпечення

Наявність унікальних 64-бітних адрес дозволяє широко використовувати пристрої 1-Wire у системах аутентифікації. Тут вони часто застосовуються у пристроях iButton. Це мікросхема з введенням на етапі виробництва 64-бітної адреси, укладена у круглий корпус з нержавіючої сталі діаметром 16 мм (MicroCAN).

Такі пристрої функціонують, наприклад, у домофонних ключах. Чіпи з підтримкою 1-Wire (наприклад, DS2401, DS2431, DS28E01-100) використовуються також для ідентифікації картриджів принтерів, медичних сенсорів, ємностей з реагентами та ін.

Перевага мікросхем 1-Wire в тому, що для контролю ідентифікованого пристрою потрібен всього один контакт. Такі чіпи укладені у спеціальний плоский корпус (SFN - Single Flat No lead) розміром 6 × 6 мм, який полегшує їх приєднання до пристрою.

Ще одне поширене застосування 1-Wire - системи автоматизації. В першу чергу це системи багатоточкового вимірювання температури різних середовищ і моніторингу теплового режиму приміщень. Температуру можна вимірювати датчиками виробництва тієї ж Maxim/Dallas.

Найбільш популярний з них - цифровий термометр DS18S20. Він має розрядність 9 біт і вимірює температуру у діапазоні від -55 до 125°C. Точність вимірювань складає 0,5°C у діапазоні -10 ... 85°C. Оскільки кожен термометр, як і будь-який пристрій 1-Wire, має унікальну 64-бітову адресу, до однієї шині 1-Wire можна підключати безліч таких приладів.

Таким чином, завдяки своїм виграшним можливостям - один провід для передачі даних і управління пристроями, під'єднання пристроїв через один контакт, живлення підключених пристроїв по дроту передачі даних, наявність у кожного пристрою унікальної адреси, низька вартість елементної бази - інтерфейс 1-Wire широко представлений у найрізноманітніших виробах сучасної електроніки.

Паралельний інтерфейс

Паралельний інтерфейс використовується у системах, де потрібен швидкий обмін пакетами даних з розрядністю 8-32 біта.

У PIC мікроконтролерах паралельний інтерфейс реалізується модулем керованого паралельного порту.

Про *керований паралельний* порт кажуть у тих випадках, коли мікроконтролер має виділені виводи для читання/запису, що дозволяє легко здійснювати операції з передачі даних.

Структурна схема керованого паралельного порту приведена па рис. 1.59.

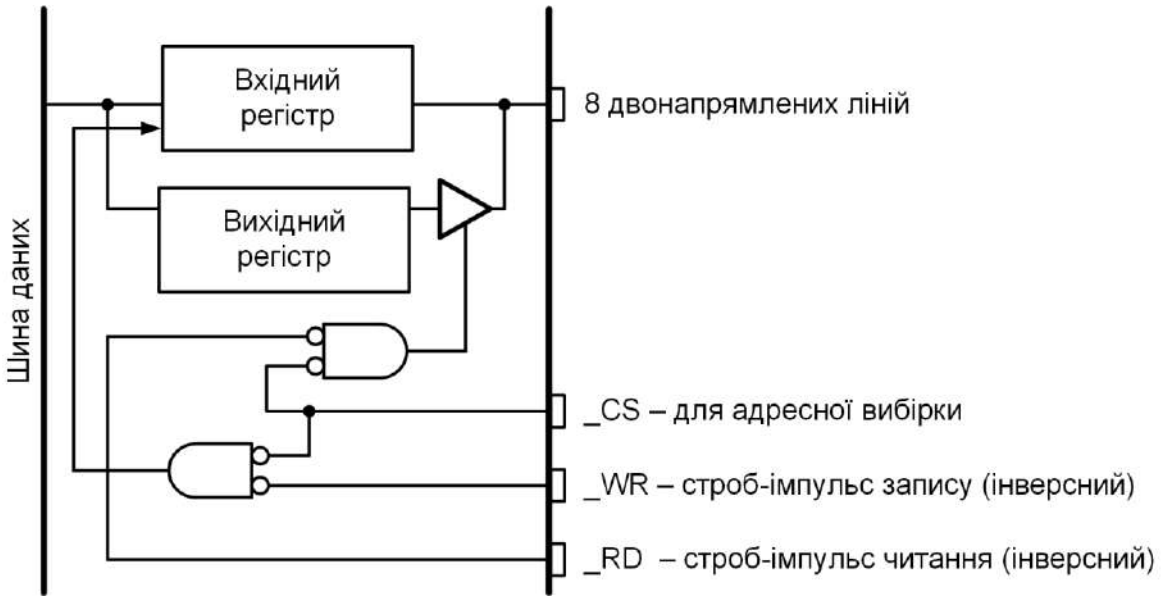


Рисунок 1.59 - Структурна схема керованого паралельного порту

Відповідні часові діаграми, що ілюструють процедуру запису/читання даних, показані на рис. 1.60.

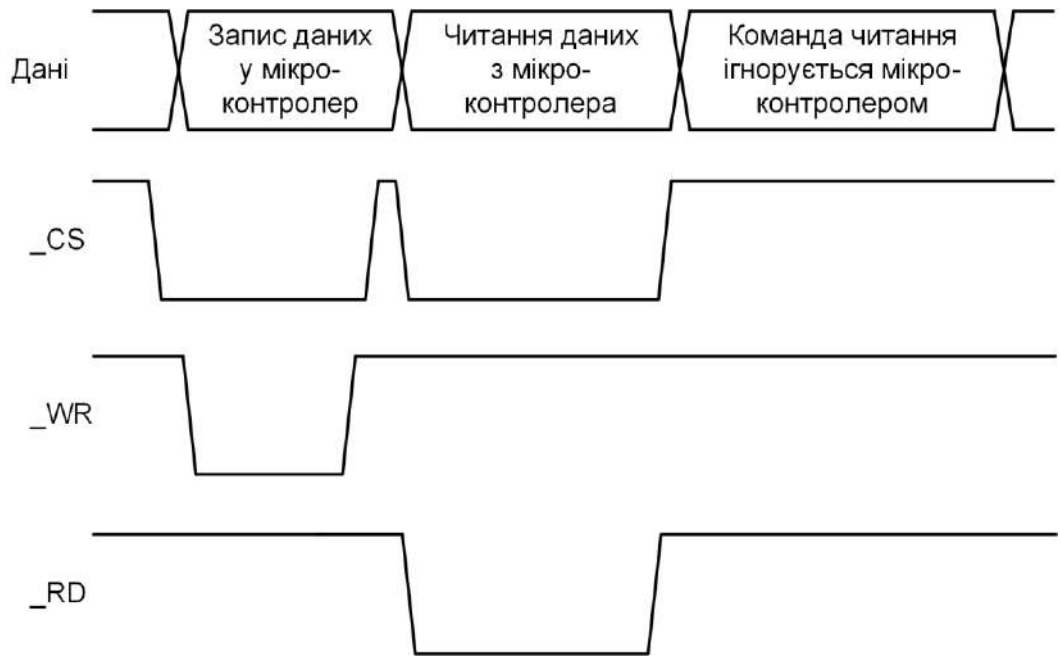


Рисунок 1.60 - Тимчасові діаграми роботи керованого паралельного порту

Мінімальний час доступу відповідає одному періоду тактових імпульсів мікроконтролера.

Для PIC-мікроконтролерів, що використовують синхронізуючі імпульси частотою 20 МГц, мінімальна величина часу доступу становить 50 нс.

Щоб дозволити використання керованого паралельного порту, потрібно попередньо встановити біт вибору режиму паралельного порту у регістрі TRISE. При установці цього біту управління портом D передається лініям _CS, _RD і _WR, які відповідають виводам RE2, RE1 і RE0 порту E.

Після того як буде встановлений біт вибору режиму порту, дані регістрів PORTD, PORTE, TRISD і TRISE ігноруються.

При використанні керованого паралельного порту, коли активізовані лінії _CS і _RD (на них рівень логічного 0), здійснюється виведення вмісту регістра OUTREG через порт D.

При записі у регістр OUTREG (адреса, відповідна регістру PORTD) проводиться встановлення біту 0BF («вихідний буфер повний») регістра TRISE. Але ця можливість передбачається не для всіх мікроконтролерів. Біт PBF скидається автоматично при зчитуванні байта з регістра OUTREG у режимі керованого паралельного порту.

При записі байта у мікроконтролер через керований паралельний порт (на лініях _CS і _WR логічні 0) величина, що записується зберігається у регістрі INREG до тих пір, поки вона не буде замінена новими даними.

У регістрі стану біт IBF («вхідний буфер повний») буде встановлюватися кожен раз під час запису даних у регістр INREG і стиратися кожен раз при зчитуванні байту програмою з регістра INREG.

Якщо раніше записаний байт НЕ зчитується до надходження наступного байту в регістр INREG, ініціюється встановлення біту переповнення вхідного буфера IBOV, який вказує, що виникли умови «затирання» даних.

ОРГАНІЗАЦІЯ ПАМ'ЯТІ МІКРОКОНТРОЛЕРА

У мікроконтролерах PIC18FXX2 реалізовано три типи пам'яті (рис 1.61):

- пам'ять програм;
- пам'ять даних;
- EEPROM пам'ять даних.

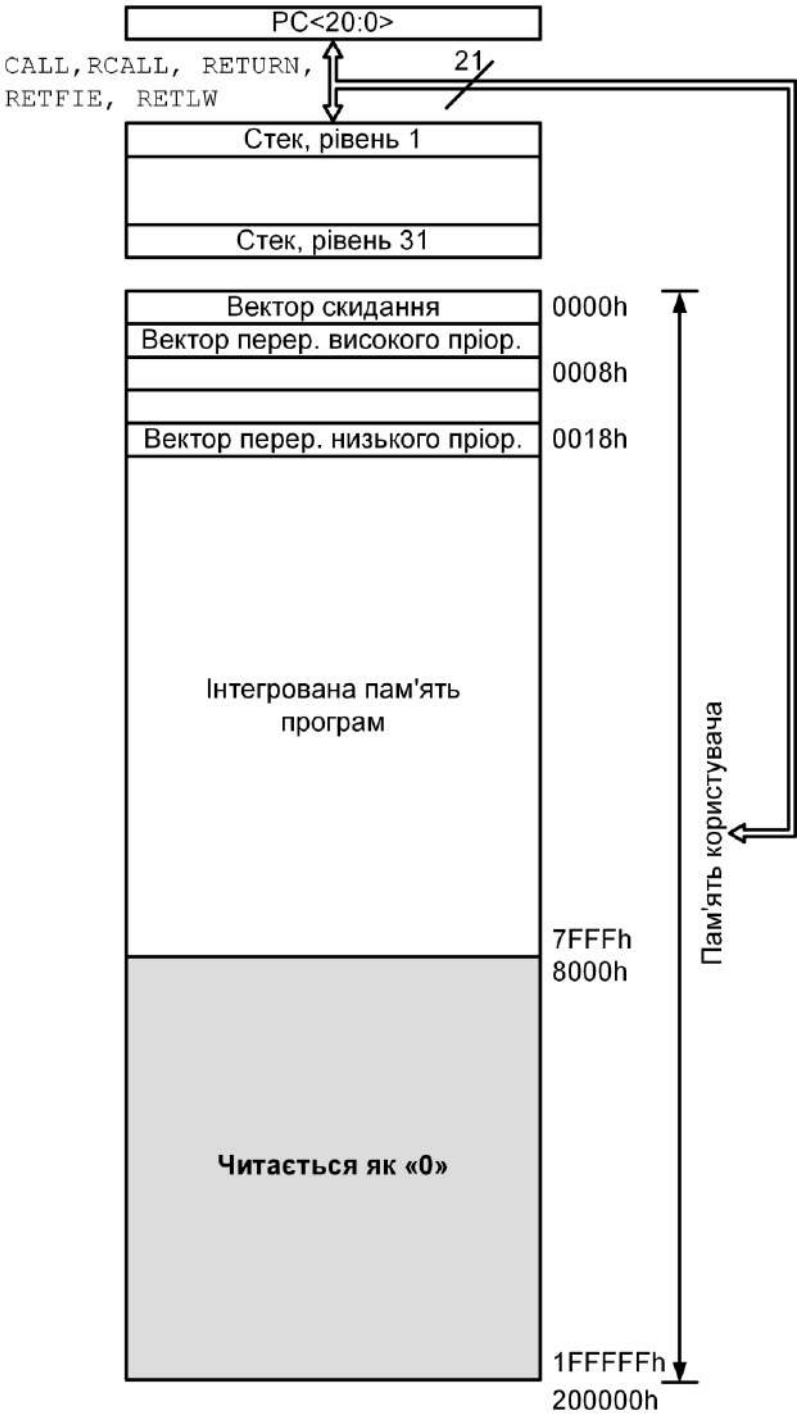


Рисунок 1.61 - Карта пам'яті програм і стеку МК PIC18F252

Звернення до пам'яті програм і пам'яті даних виконується по окремим шинам, що дозволяє організувати паралельний доступ до цих видів пам'яті.

Організація пам'яті програм

21-розрядний лічильник команд PC дозволяє адресувати 2Мбайт пам'яті програм. Фізично не реалізована пам'ять програм читається як "0" (команда NOP).

Мікроконтролери PIC18F252, PIC18F452 містять по 32 Кбайти FLASH-пам'яті програм, а PIC18F242 і PIC18F442 мають по 16 Кбайти FLASH-пам'яті програм.

Це означає, що мікроконтролери PIC18FX52 можуть мати до 16К окремих команд, а PIC18FX42 - до 8К окремих команд.

Адреса вектора скидання - 0000h.

Адреси векторів переривань - 0008h і 00018h.

Стек

Стек не є частиною пам'яті програм або пам'яті даних. Покажчик стеку доступний для запису і читання, він фактично є адресатом вершини стеку, яка може бути прочитана і змінена через регістри спеціального призначення. Дані можуть бути завантажені/прочитані у стек виконуючи операції з вершиною стеку. Біти статусу відображають стан покажчика стеку (переповнення, вичерпання стеку).

Стек дозволяє зберегти до 31 адреси повернення з підпрограми або обробки переривань. Значення лічильника команд PC поміщається у стек при виконанні команд CALL, RCALL або переході на підпрограму обробки переривань. За командою RETURN, RETLW або RETFIE значення з стеку завантажується у лічильник команд PC. При виконанні будь-якої команди переходу або повернення з підпрограми (переривань) значення регістрів PCLATU, PCLATH не змінюється.

Стек виконаний у вигляді 21-розрядного ОЗП об'ємом 31 слово. 5-розрядний покажчик стеку приймає значення 00000b після будь-якого виду скидання мікроконтролера.

Немає ніякого зв'язку з пам'яттю даних і значенням покажчика стеку 00000b. При виконанні команди типу CALL спочатку збільшується покажчик стеку, а потім значення лічильника команд PC поміщається у вершину стеку. При виконанні команди типу RETURN значення з вершини стеку завантажується у лічильник команд PC, потім покажчик стеку декрементується.

Доступ до вершини стеку

Вершина доступна для запису і читання. Три регістри спеціального призначення TOSU, TOSH і TOSL відображають стан вершини стеку, вказаної у регістрі STKPTR. Це дозволяє у разі необхідності виконувати операції зі стеком командами мікроконтролера. Після виконання команд CALL, RCALL або переходу на обробку переривань, користувач може прочитати вершину стеку через регістри TOSU, TOSH і TOSL.

Ці значення можуть бути поміщені у визначений користувачем програмний стек. При поверненні з процедури можна програмним способом змінити значення регістрів TOSU, TOSH, TOSL і виконати вихід з підпрограми.

При зміні значень стеку рекомендується вимикати переривання, щоб запобігти можливої некоректної зміни вмісту стеку.

Покажчик стеку (реєстр STKPTR)

Регістр STKPTR містить:

- біти покажчика стеку;
- біт STKFUL - прапор переповнення стеку;
- біт STKUNF - прапор вичерпання стеку.

Покажчик стеку може приймати значення від "0" до "31". Покажчик стеку збільшується, коли поміщається нове значення у стек, а при читанні вершини стеку декрементується. При будь-якому скиданні мікроконтролера покажчик

стеку стає рівним "0". Покажчик стеку доступний для запису і читання. Ця особливість може використовуватися системами RTOS для збереження адрес повернення з процедур.

Після запису у стек більше 31 рази (без читання вмісту стеку) встановлюється біт STKFUL, який може бути скинутий у "0" тільки програмним способом або скиданням по включенню живлення POR.

Дія, що виконується при переповненні стеку, залежить від стану біту конфігурації STVREN (дозвіл скидання мікроконтролера при переповненні стеку). Якщо біт STVREN встановлений у "1" (значення за замовчуванням), то при переході на процедуру (обробку переривань) у 31-у комірку стеку поміщається адреса повернення (PC + 2), встановлюється у "1" біт STKFUL, виконується скидання мікроконтролера (при цьому біт STKFUL залишається рівним "1", а покажчик стеку дорівнює "0").

Якщо **STVREN = 0**, STKFUL буде встановлений у "1" при записі у 31-у комірку стеку, покажчик стеку буде мати значення "31". Будь-який додатковий запис у стек не змінюватиме значення стеку, а покажчик стеку як і раніше буде мати значення "31".

При вичерпанні стеку (виконувалося повернення більше число раз, ніж переходів на підпрограми/обробку переривань) у лічильник команд PC завантажується 0000h, встановлюється у "1" біт STKUNF, покажчик стеку залишається рівним "0". Біт STKUNF скидається у "0" програмним способом і при скиданні по включенню живлення POR.

Організація пам'яті даних

Пам'ять даних реалізована як статичний ОЗП. Кожен регістр у пам'яті даних має 12-розрядну адресу, що дозволяє адресувати до 4096 байт пам'яті даних.

На рисунку 1.67 показана організація пам'яті даних мікроконтролерів PIC18FXX2.

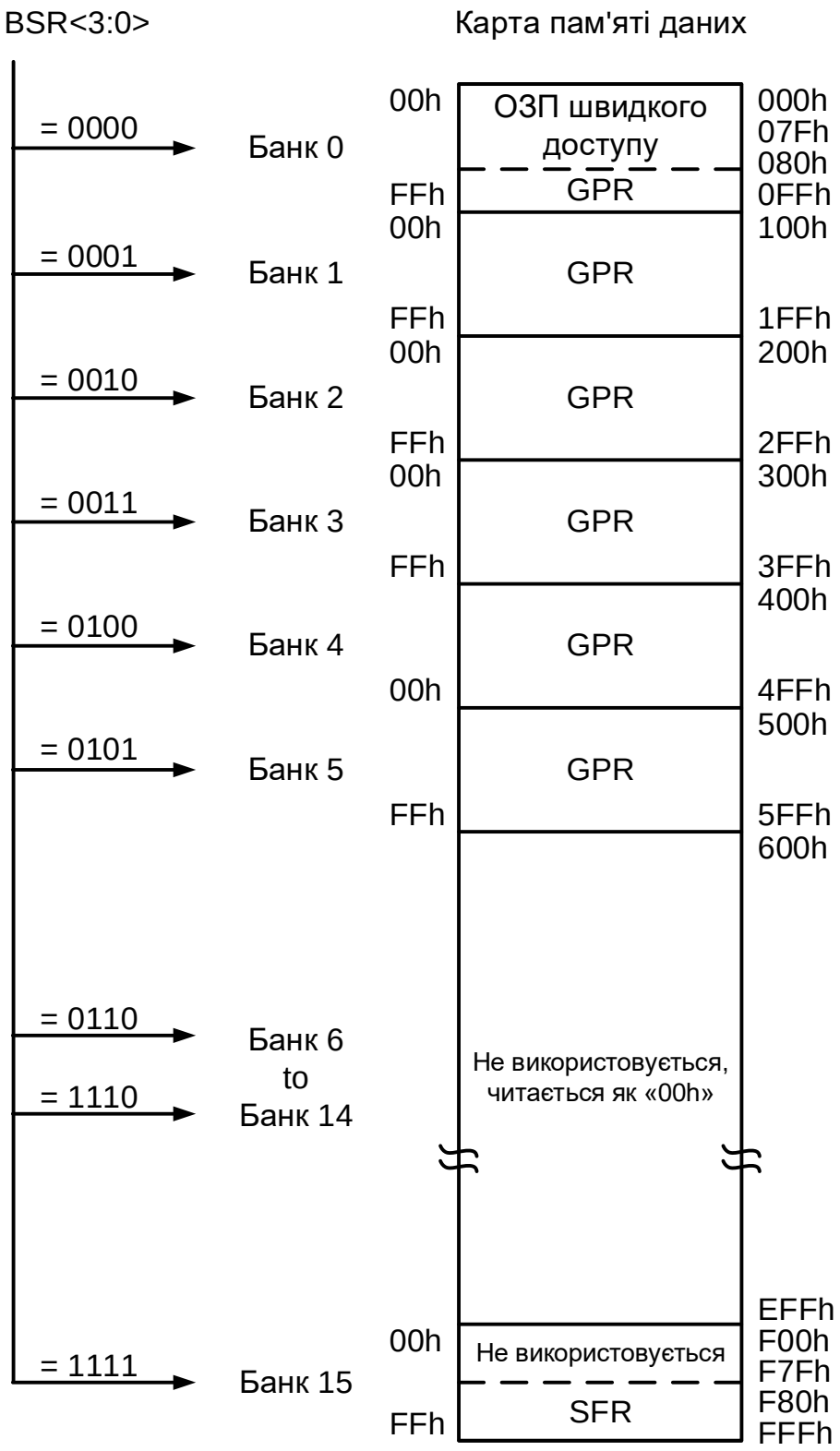


Рисунок 1.62 - Організація пам'яті даних МК PIC18FXX2

Пам'ять даних розділена на 16 банків, кожен з яких містить по 256 байт. Молодші 4 біти регістра BSR використовуються для вибору поточного банку пам'яті (BSR <3:0>). Старші 4 біти регістра BSR не реалізовані.

Пам'ять даних містить регістри спеціального (SFR) і загального (GPR) призначення.

Регістри SFR використовуються для управління ядром і периферійних модулів мікроконтролера, у той час як GPR використовуються для зберігання даних користувача.

Регістри SFR починаються з останнього байту 15-го банку пам'яті даних (0xFFF) і поширюються вниз по карті пам'яті. Будь-який незадіяний регістр в області SFR може використовуватися як регістр загального призначення.

Регістри GPR починаються в першому байті 0-го банку пам'яті даних і поширюються вгору по карті пам'яті. Читання не реалізованої пам'яті даних буде давати результат "0".

До будь-якого регістру пам'яті даних можна звернутися безпосередньо або непрямо.

При *прямій* адресації може знадобитися налаштування регістра BSR.

Непряма адресація вимагає настройки регістрів FSRn і звернення до пам'яті через відповідний регістр INDFn.

Кожен регістр FSR містить 12-розрядну адресу регістра в пам'яті програм, що дозволяє виконувати непряму адресацію без перемикання банків пам'яті даних.

Система команд PIC18FXX2 дозволяє виконувати операції з регістрами у всій області пам'яті програм. Це може бути виконано за допомогою непрямої адресації або командою MOVFF. Команда MOVFF є двослівною і двоцикловою, вона переміщує значення одного регістра до іншого.

Для звернення за один машинний цикл до регістрів спеціального і частини регістрів загального призначення був реалізований банк пам'яті швидкого доступу.

Незалежно від поточного значення регістра BSR відбувається звернення до частини банку 0 і банку 15.

Регістри загального призначення GPR

До регістрів загального призначення можна звернутися безпосередньо або непрямо.

Непряма адресація вимагає настройки регістрів FSR і звернення через регістр INDF.

Регістри загального призначення мають організацію у пам'яті даних по банкам, вони не ініціалізуються при скиданні по включенню живлення, а при інших видах скидання не змінюють свого значення.

Пам'ять даних доступна для звернення усіма командами мікроконтролера. Старша частина банку 15 містить регістри SFR, всі інші банки містять регістри GPR (починаючи з банку 0).

Регістри спеціального призначення SFR

Регістри спеціального призначення призначені для управління ядром мікроконтролера і периферійними модулями. Ці регістри реалізовані як статичне ОЗП.

Регістри SFR поділяються на дві основні групи:

- управління ядром мікроконтролера;
- управління периферійними модулями мікроконтролера.

EEPROM І ДОСТУП ДО ДАНИХ

Регістри EEADR, EEDATA, ECON1 і EEC0N2 забезпечують доступ до вбудованого EEPROM (*постійному запам'ятовуючому пристрою (ПЗП), що електрично стирається і програмується*), призначеного для зберігання даних. При цьому регістри EEADR і EEDATA необхідні для формування і введення адреси та даних у EEPROM, об'єм пам'яті якого може досягати 256 байт.

В свою чергу регістри EECON1 і EECON2 призначені для вибору режиму доступу та індикації моменту завершення операцій.

Регістр EEC0N2 являє собою «псевдорегістр», який не допускає зчитування даних, проте сюди можна записати коди 0x055/0x0AA, що дозволяє контролювати правильність виконання операції запису.

У табл. 1.20 вказується призначення бітів регістра EECON1, які відповідають за управління доступом.

Таблиця 1.20 - Призначення основних бітів регістра EECON1

Біт	**Призначення**
EEPCD	Встановлюється для дозволу доступу до програмної пам'яті; обнуляється для дозволу доступу до EEPROM
WRERR	Біт помилки запису, вказує на можливу некоректність записаного результату
WREN	Біт ініціювання операції запису у EEPROM
WR	Індицирує виконання операції запису; обнуляється після завершення операції запису
RD	Індицирує виконання операції зчитування; автоматично стирається

Для роботи з EEPROM використовуються наступні функції:

```
//читання EEPROM за адресою
read_eeprom(address);
//запис значення у EEPROM за адресою
write_eeprom(address, value);
//читання N байт EEPROM за адресою
read_eeprom(address, [N]);
//читання EEPROM у змінну за поточною адресою
read_eeprom(address, [variable]);
//читання N байт з EEPROM з адресацією
read_eeprom(address, pointer, N);
//запис N байт за адресою
write_eeprom(address, pointer, N);
```

ФОРМАТ ФАЙЛУ *.HEX

Асемблер середовища **IDE PCWH** і будь-які інші асемблери і компілятори перетворюють вихідний код програми PIC-мікроконтролерів у формат даних, яким може скористатися програматор для завантаження програм у PIC-мікроконтролер.

Найбільш популярним форматом файлів (який застосовує програматор фірми Microchip і більшість інших програматорів) є 8-бітний HEX-формат, запропонований корпорацією Intel.

HEX-файл (Example.hex) генерується при асемблюванні.

Інформація в ньому представляється у наступному вигляді:

```
:10000000FF308600831686018312A001A101A00B98
:0A0010000728A10B072886030728860307 2824
:02400E00F13F80
:00000001FF\
```

Кожен рядок містить початкову адресу і дані, які повинні бути розміщені за цією адресою.

У табл. 2.23 описано призначення позицій рядків.

Таблиця 2.23 – Позиції рядків HEX-файлу

Байт	**Призначення**
1	Двокрапка, завжди позначає початок рядка
2-3	Число байтів команд у рядку
4-7	Початкова адреса запису команд. Це формат фірми Motorola (за старшим байтом слідує молодший)
8-9	Тип рядка (00 - дані, 01 - кінець)
10-13	Перша команда, яка повинна завантажуватися у PIC - мікроконтролер за вказаною адресою. Ці дані представлені у форматі фірми Intel (за молодшим байтом слідує старший)
:	Інші команди також представлені в форматі фірми Intel
Дві останні позиції, що відображаються	Контрольна сума рядка
Дві останні позиції, що не відображаються	ASCII-символи повернення каретки і переведення рядка

Контрольна сума обчислюється шляхом додавання всіх байтів рядка і віднімання молодшого байта суми з `0x0100`. Для другого рядка в наведеному вище прикладі шістнадцяткового файлу ця сума розраховується наступним чином:

```
0A
00
10
00
07
28
A1
0B
07
28
86
03
07
+ 28
------
1DC
```

Для отримання контрольної суми з `0x0100` віднімається молодший байт суми (`0x00DC`):

```
0x010D
-0x00DC
----------
0x0024
```

Отримана в результаті обчислень величина контрольної суми `0x024` збігається з двома останніми байтами початкового рядка.

РОЗДІЛ 2. ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ МІКРОКОНТРОЛЕРІВ

АПАРАТНО-ПРОГРАМНІ КОМПЛЕКСИ

Для виконання лабораторних практикумів на базі **PIC – мікроконтролерів**, були розроблені **апаратно-програмні комплекси (АПК)** (рис 2.1, 2.2).

Рисунок 2.1 – Апаратно-програмний комплекс № 1

Апаратно-програмні комплекси призначені для створення та відлагодження програмного забезпечення для різних мікроконтролерних систем у рамках курсу **«Програмне забезпечення управляючих мікро-ЕОМ»**.

Рисунок 2.2 – Апаратно-програмний комплекс № 2

До складу Апаратно-програмного комплексу № 1 (рис. 2.1) входить:

- базовий інтегрований апаратно-програмний модуль;
- автономний апаратно-програмний модуль;
- модуль-емулятор об'єкта управління;
- навчальна програма курсу;
- завдання і методичні вказівки до виконання лабораторних практикумів;
- приклади виконання завдань;
- інтегроване середовище розробки програмного забезпечення на мові програмування **C**;
- драйвер для завантаження бінарного коду у FLASH-пам'ять програм мікроконтролера по інтерфейсу **ICSP**.

Завдання, які вирішуються за допомогою Апаратно-програмного комплексу № 1:

1) ознайомлення з мікроконтролерами і засобами розробки:
   - введення/виведення дискретних сигналів;
2) виведення інформації на LCD-дисплей;
3) робота з перериваннями мікроконтролера;
4) введення/виведення даних по інтерфейсу **RS-232**;
5) введення і обробка аналогових сигналів за допомогою АЦП:
   - організація введення з цифрової клавіатури;
6) виведення даних через зовнішній ЦАП по інтерфейсу **SPI**:
   - створення драйвера інтерфейсу **SPI**;
7) виведення інформації на семисегментний LED-дисплей:
   - створення драйвера динамічної індикації;
8) робота з модулями **PWM** (ШІМ):
   - управління потужністю у навантаженні;
   - управління двигуном постійного струму. Напрямок та швидкість обертання;
9) робота з зовнішньої **EEPROM** пам'яттю по інтерфейсу **I^2C**:
   - запис/читання байтів і блоків даних;
10) розробка ПІД-регулятора:
   - розрахунок параметрів ПІД-регулятора за методикою Ніколса;
   - оптимізація ПІД-регулятора;
   - управління об'єктом;
11) управління сервоприводом.

Структура Апаратно-програмного комплексу № 2

Апаратно-програмний комплекс № 2 (рис. 2.2) має наступну структуру (рис. 2.3):

- контролери;
- пристрої введення/виведення;
- зовнішні пристрої;
- операційні системи;
- мережеві протоколи;
- інтерфейси зв'язку з об'єктами і пристроями;
- бездротові інтерфейси;
- об'єкти управління.

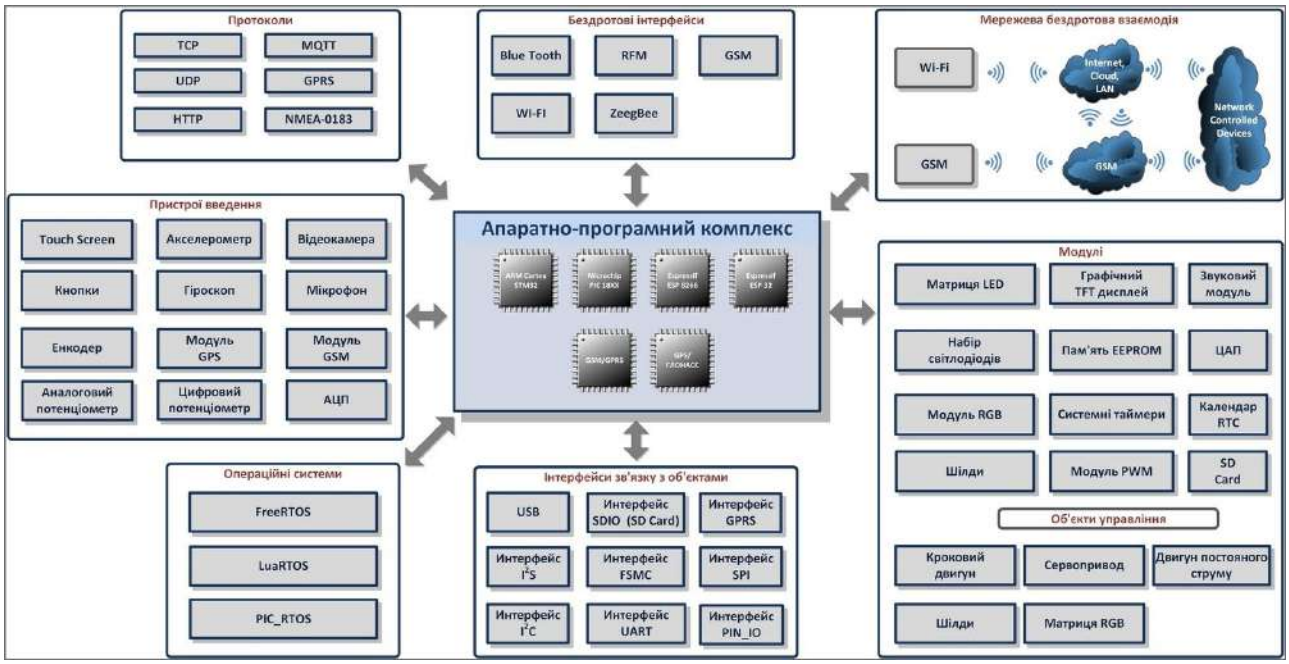


Рис. 2.3 - Структура Апаратно-програмного комплексу № 2

Склад апаратної частини комплексу № 2 (рис. 2.2)

1. Модулі контролерів:

- PIC 18F25K22;
- STM 32F103C8T6;
- ESP 32;
- ESP 8266 (рис. 2.4).

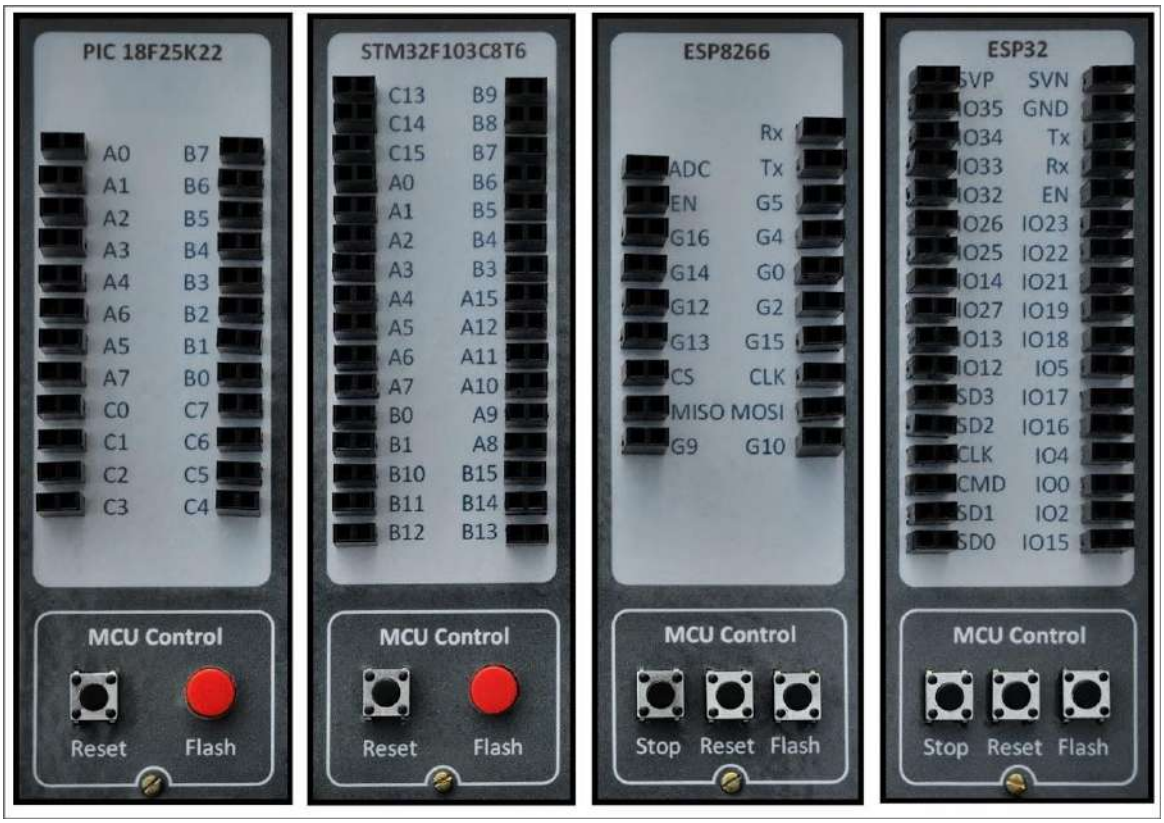


Рис. 2.4 - Змінні модулі контролерів

Передбачене встановлення будь-яких інших контролерів і програмованої логічної матриці. (ПЛМ).

2. Мережеві комунікаційні модулі:

- модуль Wi-Fi: ESP 8266;
- модуль Wi-Fi: ESP 32;
- модуль GSM: SIM800L.

3. Допоміжні модулі:

- гіроскоп: L3G4200D;
- акселерометр: ADXL345;
- модуль GPS: M8030 NEO-M8N;
- модуль SD Card (read/write).

4. Пристрої введення:

- TouchScreen SPI (TFT дисплей);
- аналоговий потенціометр;
- цифровий потенціометр: X9C103SZI;
- цифровий енкодер;
- набір кнопок;
- набір перемикачів;
- модуль UART;
- модуль Wi-Fi;
- модуль GSM;
- модуль GPS.

5. Пристрої виведення:

- TFT SPI дисплей;
- OLED I^2C serial дисплей;
- набір світлодіодів;
- набір RGB світлодіодів;
- матриця LED 8x8 SPI-UART;
- модуль Wi-Fi;
- модуль GSM.

6. Об'єкти управління:

- двигун постійного струму;
- кроковий двигун;
- сервопривід.

Розширення:

Для розробки і/або підключення інших модулів передбачена макетна панель і клеми розширення.

Склад програмної частини комплексу № 2 (рис. 2.2)

1. Мережеві модулі:

- FTP Server, Client;
- HTTP Server, Client;
- MQTT Client;
- Websocket Client;
- Redis Client;
- Net (UDP, TCP);
- CJSON;
- CoAP;
- IMAP (e-mail.);
- WiFi (Station, Access Point);
- WiFi Monitor;
- Sqlite3 (SQL);
- Crypto;
- TLS.

2. Бібліотеки та драйвери для:

- управління апаратними модулями комплексу і зовнішніми шилдами;
- управління модулем GPS / ГЛОНАСС;
- управління модулем GSM;
- бібліотека графічного інтерфейсу з елементами дискретного і пропорційного управління.

3. Інші бібліотеки і драйвери:

При необхідності легко встановлюються/використовуються необхідні бібліотеки і драйвери.

СИСТЕМА АВТОМАТИЗОВАНОГО ПРОЕКТУВАННЯ PROTEUS

Proteus Design Suite – пакет програм для автоматизованого проектування (САПР) електронних схем. Розробка компанії Labcenter Electronics (Великобританія).

Пакет являє собою систему схемотехнічного моделювання, що базується на основі моделей електронних компонентів, прийнятих у **PSpice** (Personal Simulation Program with Integrated Circuit Emphasis – програма симуляції аналогової і цифрової логіки).

Відмінною рисою пакета PROTEUS VSM є можливість моделювання роботи програмованих пристроїв: мікроконтролерів, мікропроцесорів, DSP NF та ін. Бібліотека компонентів містить довідкові дані. Додатково у пакет PROTEUS VSM входить система проектування друкованих плат.

Пакет **Proteus** складається з двох частин, двох підпрограм:

- **ISIS** – програма синтезу та моделювання безпосередньо електронних схем;
- **ARES** – програма розробки друкованих плат.

Разом з програмою встановлюється набір демонстраційних проектів для ознайомлення.

Також до складу восьмої версії входить середовище розробки **VSM Studio**, що дозволяє швидко написати програму для мікроконтролера, використовуваного у проекті, NF скомпілювати.

Пакет є комерційним. Безкоштовна ознайомча версія характеризується повною функціональністю, але не має можливості збереження файлів.

Примітною особливістю є те, що у ARES можна побачити 3D-модель друкованої плати, що дозволяє розробнику оцінити свій пристрій ще на стадії розробки.

Система підтримує підключення нових елементів (SPICE) і підключення різних компіляторів (PICOLO, ARM-подібні, AVR і т.д.).

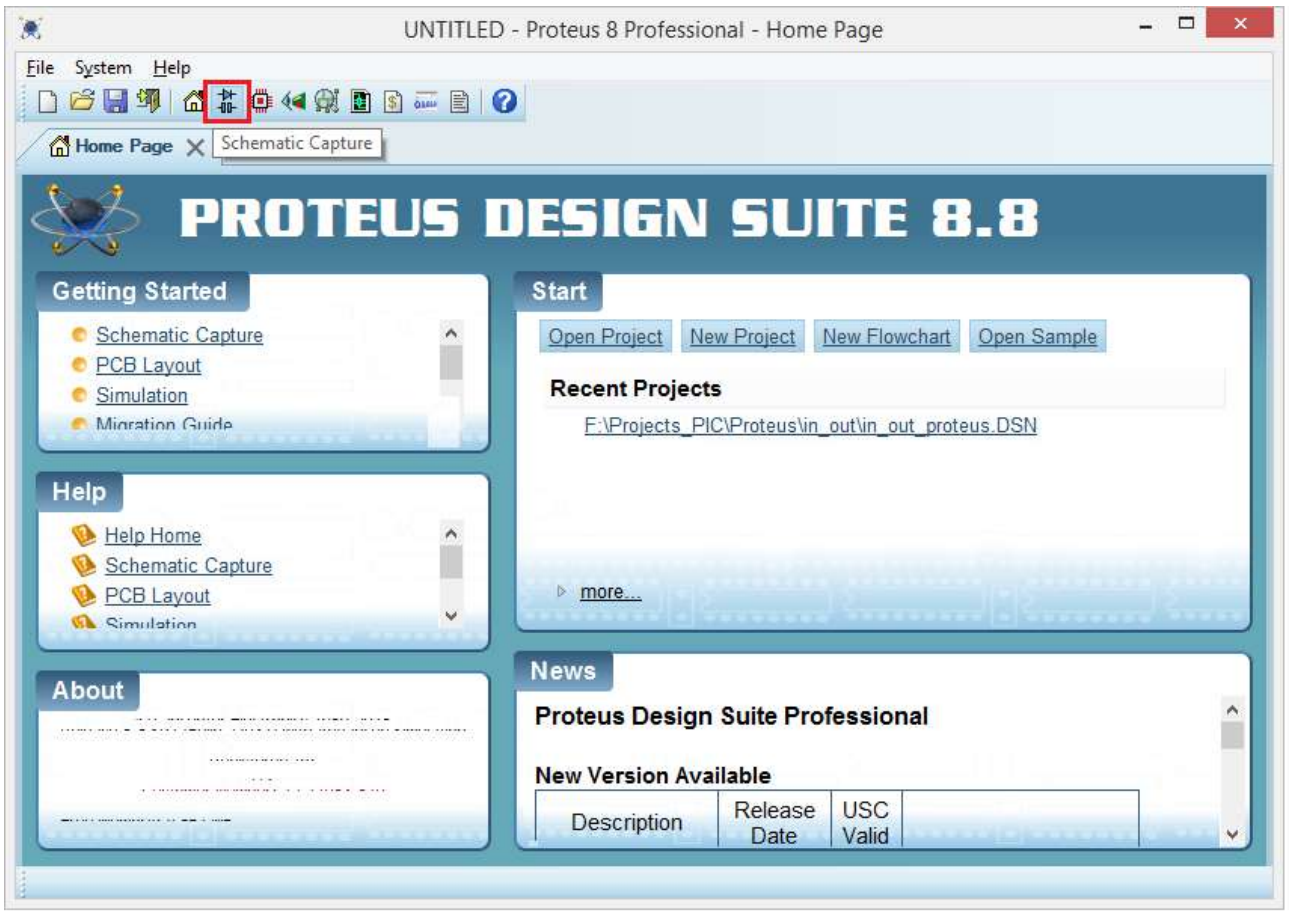


Рисунок 2.5 – Середовище моделювання **«Proteus»**

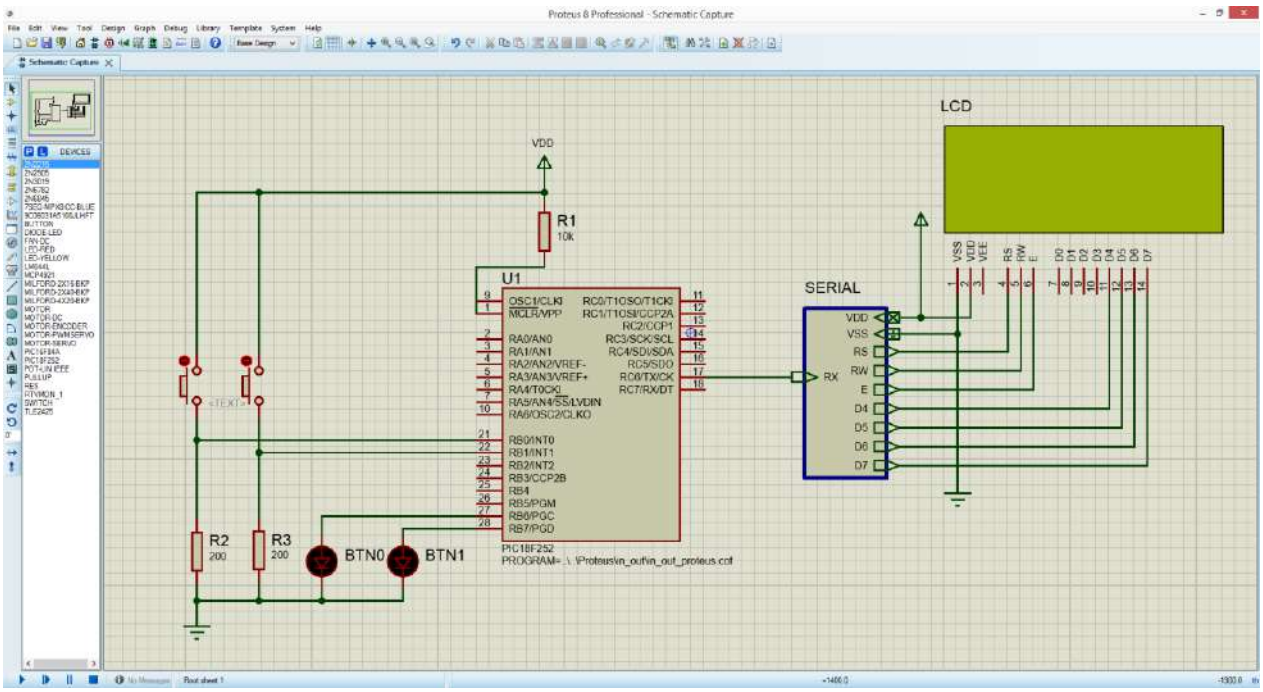


Рисунок 2.6 – Виконання лабораторного практикуму
у середовищі моделювання **«Proteus»**

ОПИС РОБОТИ З ІНТЕГРОВАНИМ СЕРЕДОВИЩЕМ РОЗРОБКИ ПРОГРАМ PCWH

До складу інтегрованого середовища розробки **(IDE) PCWH** входить компілятор **CCS-PICC**, який дозволяє користувачу створювати проекти з одного або декількох файлів вихідного коду, та компілювати вихідний код у виконувані файли, призначені для завантаження у цільовий мікроконтролер.

За замовчуванням, після установки **IDE PCWH** на **Робочому столі Windows** буде розміщений ярлик **«PIC C Compiler»**, який і використовується для запуску інтегрованого середовища розробки.

PCWH можна запустити по команді меню **Пуск › Все программы › PIC-C › PIC C Compiler** або **c:\Program Files\ PICC\PCW.exe**.

Вікно інтегрованого середовища розробки **PCWH** при першому запуску показане на рис. 2.7.

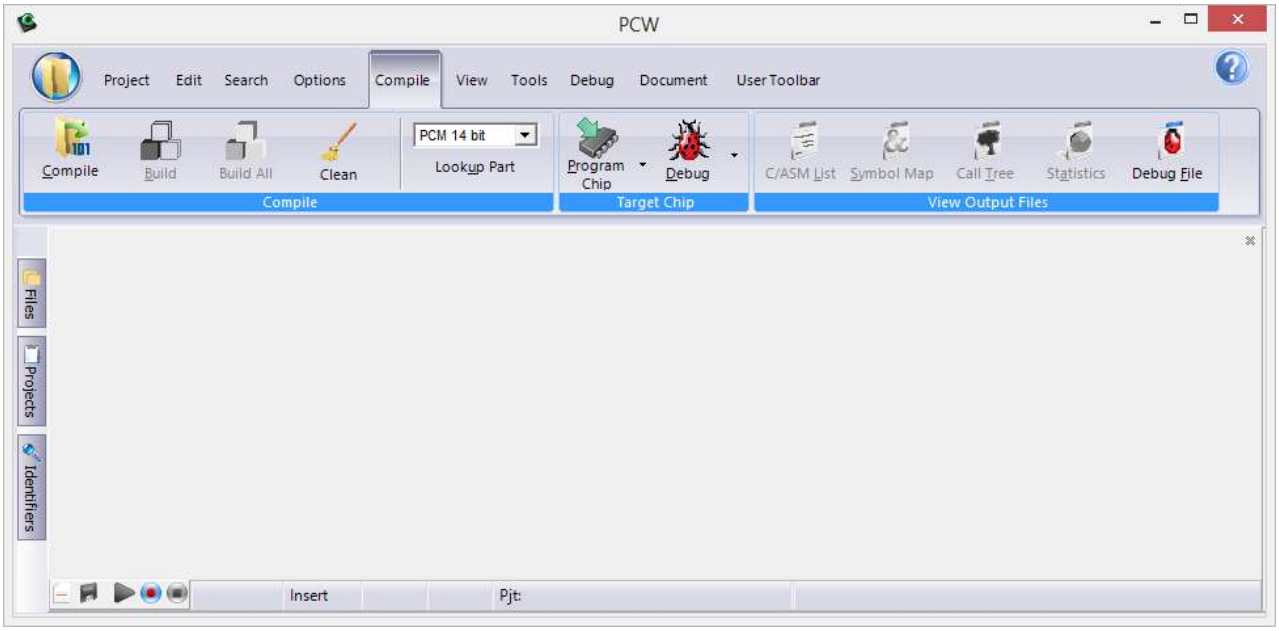


Рисунок 2.7 – Вікно інтегрованого середовища розробки **PCWH** при першому запуску

Порядок створення проекту

Проект складається з одного або більше файлів з вихідним кодом програми (головний файл проекту має розширення ***.pjt**). Новий проект можна створювати вручну за допомогою команди меню **Project › Create** (рис. 2.8),

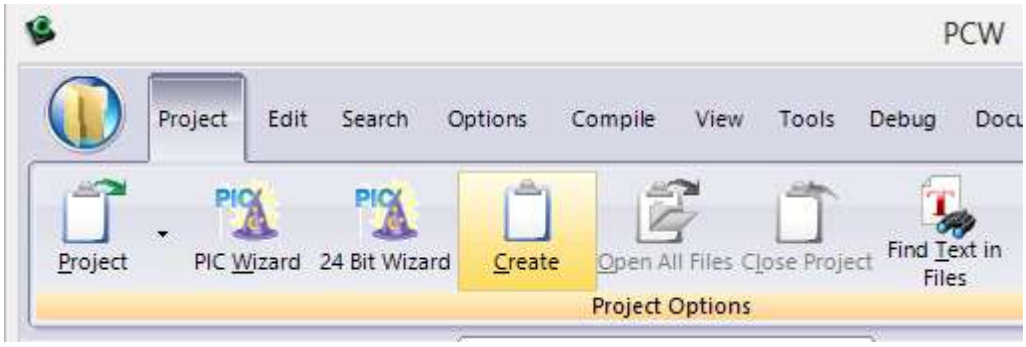


Рисунок 2.8 – Створення проекту у ручному режимі
за допомогою команди меню **Project / Create**

або згенерувати його автоматично за допомогою майстра **PIC Wizard**, який викликається по команді меню **Project › PIC Wizard або Project › 24 Bit Wizard** (рис. 2.9).

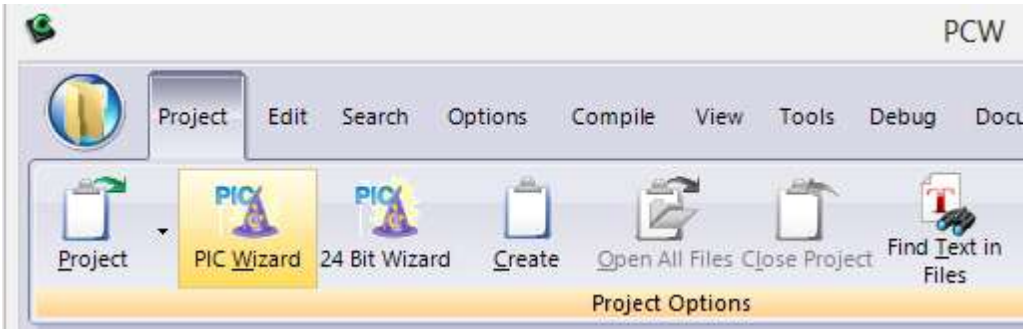


Рисунок 2.9 – Створення проекту автоматично
за допомогою майстра **PIC Wizard**

Обидва методи створення нового проекту запитують у користувача ім'я головного вихідного файлу для проекту і цільовий пристрій.

Створення проекту у інтегрованому середовищі розробки PCWH за допомогою команди меню Project/Create

Якщо було вибрано створення проекту за допомогою команди меню **Project/Create**, то спочатку з'явиться стандартне діалогове вікно, яке пропонує вибрати ім'я головного вихідного файлу проекту з розширенням ***.c** (рис. 2.10).

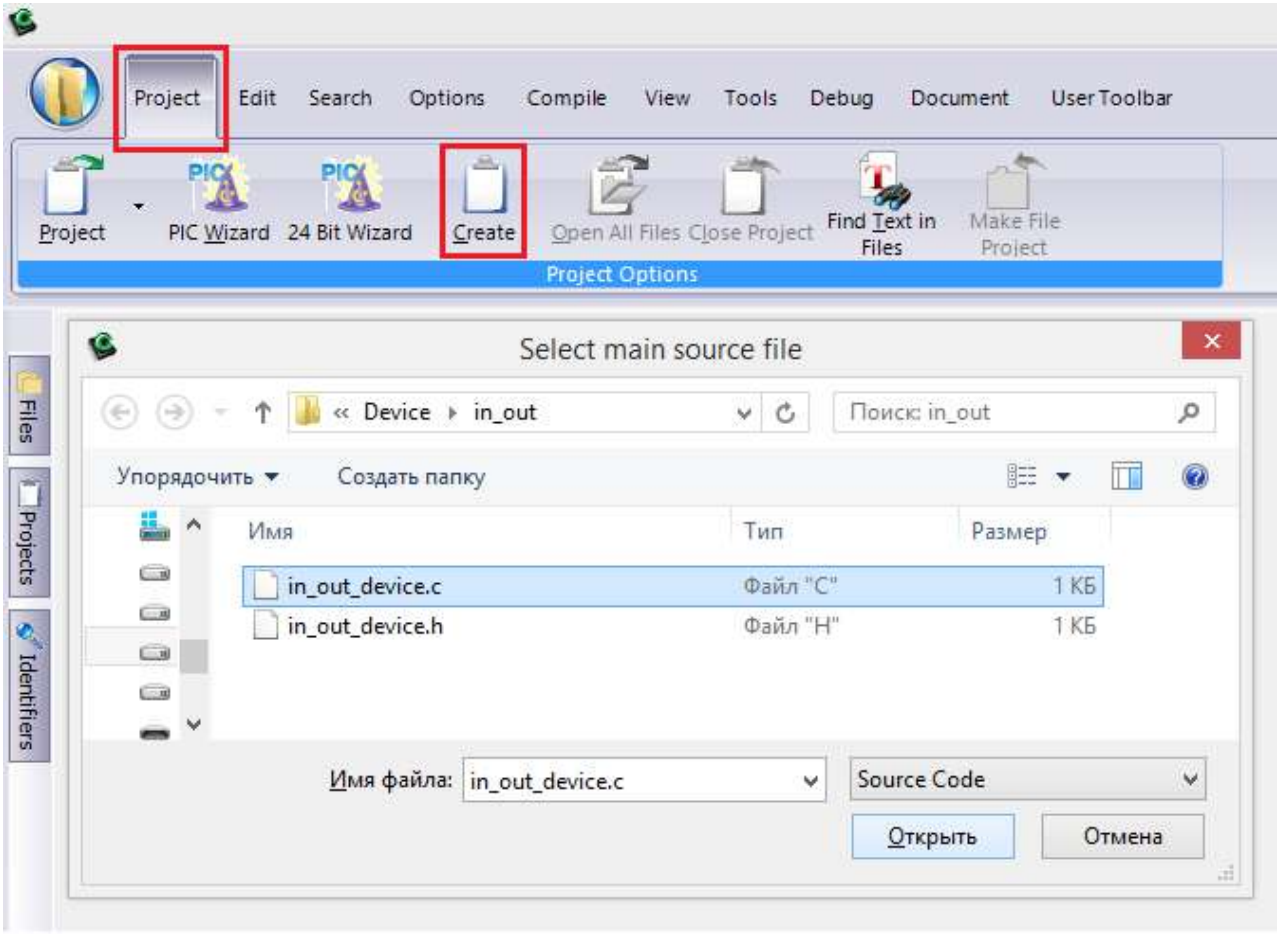


Рисунок 2.10 – Створення проекту в інтегрованому середовищі розробки **PCWH** у ручному режимі

Після вибору такого файлу відкриється діалогове вікно, яке пропонує вказати цільовий мікроконтролер. Наприклад: мікроконтролер **PIC18F252** (рис. 2.11).

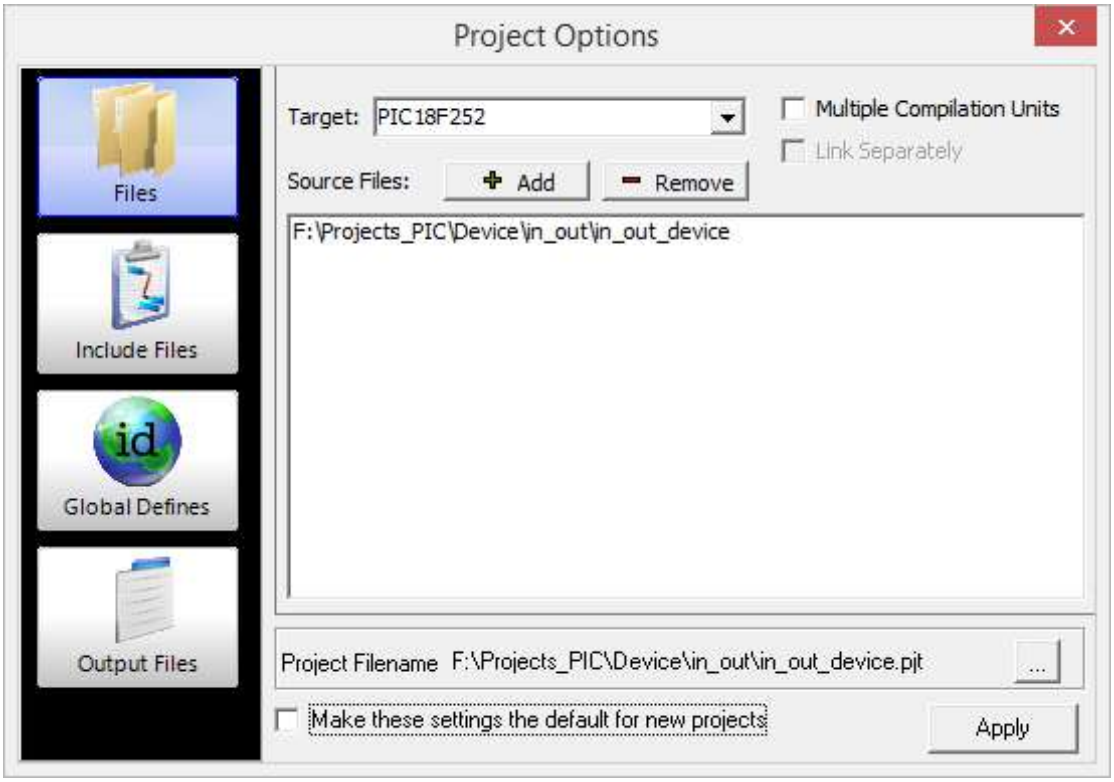


Рисунок 2.11 – Діалогове вікно **Project Options**

Це ж діалогове вікно можна відкрити у будь-який момент і після створення проекту по команді меню **Options › Project Options** (рис. 2.12).

Рисунок 2.12 – Діалогове вікно **Project Options**

Якщо готового вихідного коду немає, і необхідно почати створення проекту «з нуля», то можна вибрати будь-який файл ***.c** (наприклад, з папки `\Program Files\ PICC\Examples`) (рис. 2.13).

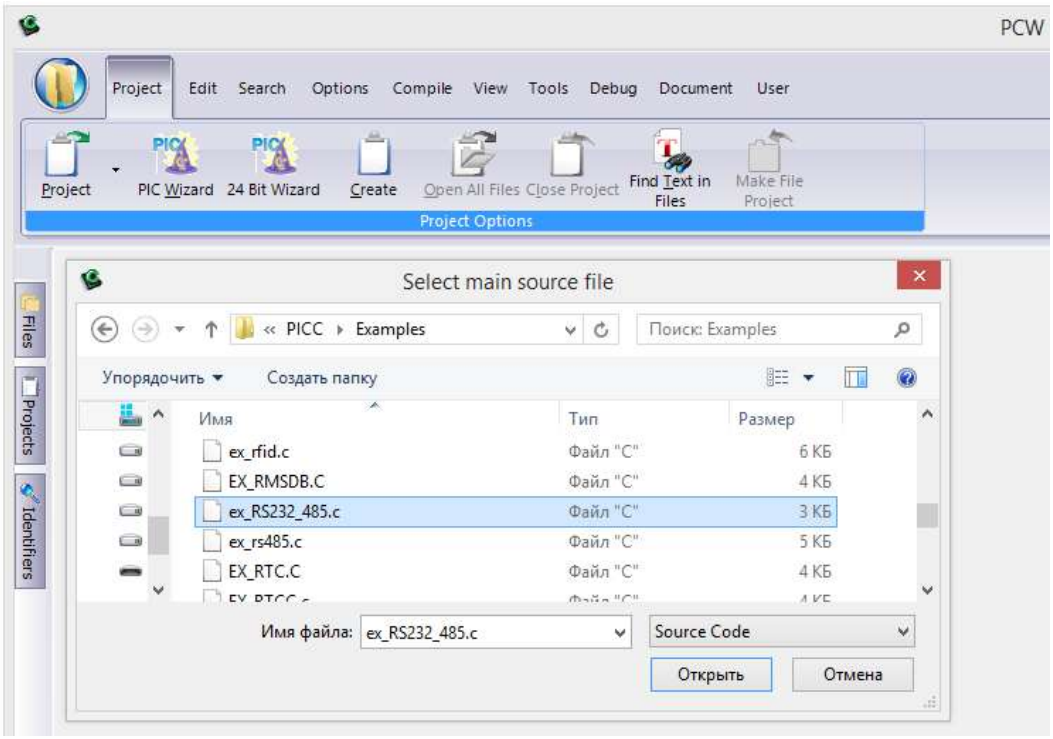


Рисунок 2.13 – Створення проекту «з нуля»

Вибрати тип мікроконтролера (наприклад, для виконання лабораторних практикумів вибрати тип мікроконтролера **PIC18F252)**. Потім виділити будь-який файл ***.c** у списку діалогового вікна **Project Options** і натиснути кнопку **Remove** (Видалити) (рис. 2.14).

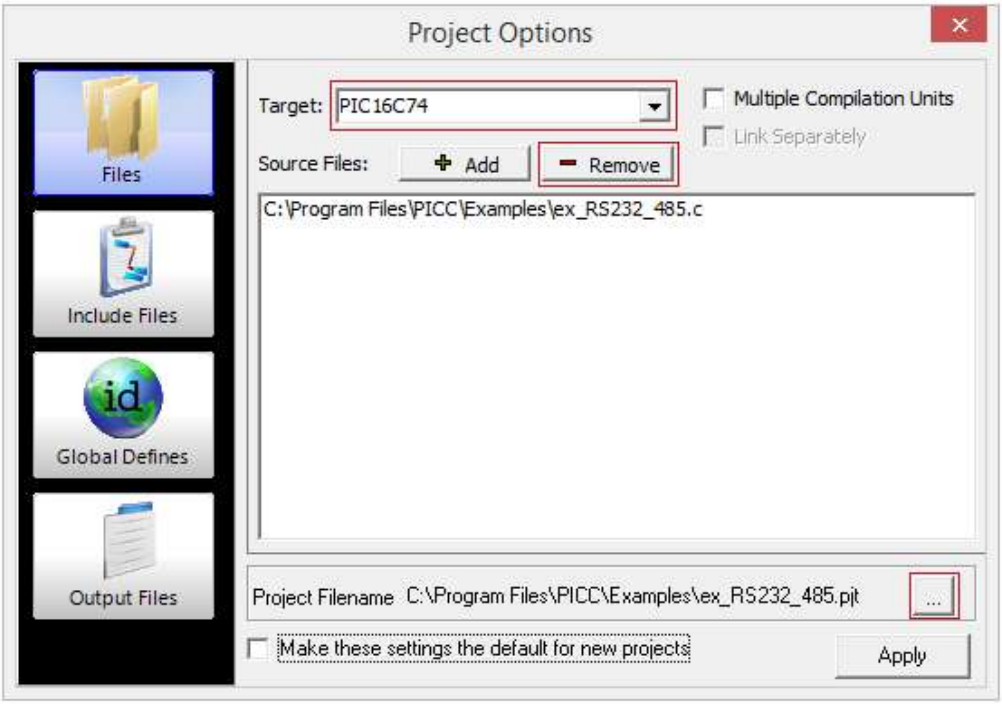


Рисунок 2.14 – Діалогове вікно **Project Options**

Надалі файл вихідного коду буде створений автоматично. У такому випадку знадобиться також змінити папку розміщення та ім'я проектного файлу. Для цього слід натиснути кнопку **…** праворуч від поля **Project FileName** і задати нове розташування та ім'я файлу ***.pjt**.

Для того щоб задати розміщення файлів з описом підтримуваних компілятором мікроконтролерів, відмінним від обраного за замовчуванням, у вікні **Project Options** слід натиснути кнопку **Include Files**. В результаті відкриється відповідний розділ параметрів проекту (рис. 2.15).

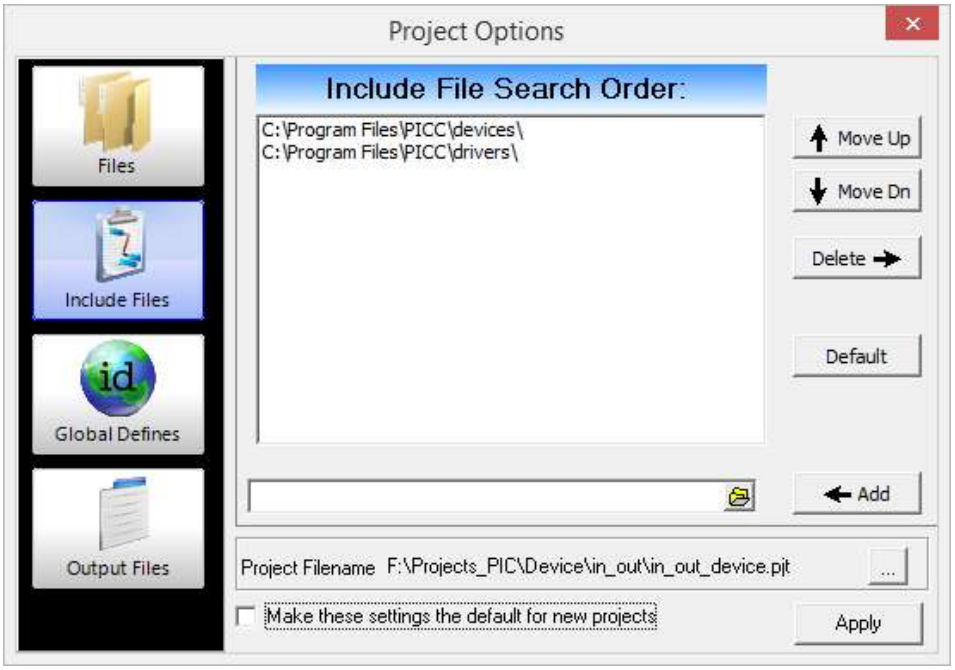


Рисунок 2.15 – Розділ **Include Files** діалогового вікна **Project Options**

Для того щоб додати новий шлях у список, слід ввести його у розташоване внизу поле (або знайти за допомогою діалогового вікна пошуку, натиснувши кнопку з зображенням папки) і натиснути кнопку **Add**.

Для видалення поточного елементу зі списку призначена кнопка **Delete**.

Після того як список шляхів пошуку сформований, можна натиснути кнопку **Apply** (Застосувати), щоб завершити створення проекту.

Якщо проект створюється на основі існуючого файлу ***.c**, то в одній з ним папці буде створений проектний файл з тим же ім'ям, але з розширенням ***.pjt**. В іншому випадку з'явиться запит на створення файлу **main.c**.

Після ствердної відповіді на цей запит буде створений проект з ім'ям, заданим у діалоговому вікні **Project Options**. У будь-якому випадку у **(IDE) PCWH** відкриється вихідний код головного файлу ***.c**.

Заголовні файли з розширенням ***.h** – це зовнішні файли описів, що підключаються до програмного модулю за допомогою директиви **#include**.

Створення проекту у інтегрованому середовищі розробки PCWH за допомогою майстра PIC Wizard

Для запуску майстра **PIC Wizard** слід виконати відповідну команду меню **Project**, а потім вказати розміщення і ім'я головного файлу проекту. В результаті відкриється вікно майстра, що складається з розділів з параметрами проекту (рис. 2.10). Зовнішній вигляд вікна майстра може відрізнятися в залежності від версії **IDE** і обраного типу мікроконтролера.

У розділі **General** (рис. 2.16) вибирається цільовий мікроконтролер (список **Device**, що розкривається), його робоча частота (поле **Oscillator Frequency**), а також настроюються загальні параметри, на зразок розрядів запобігання, типу джерела системної синхронізації, порогової напруги для скидання та ін.

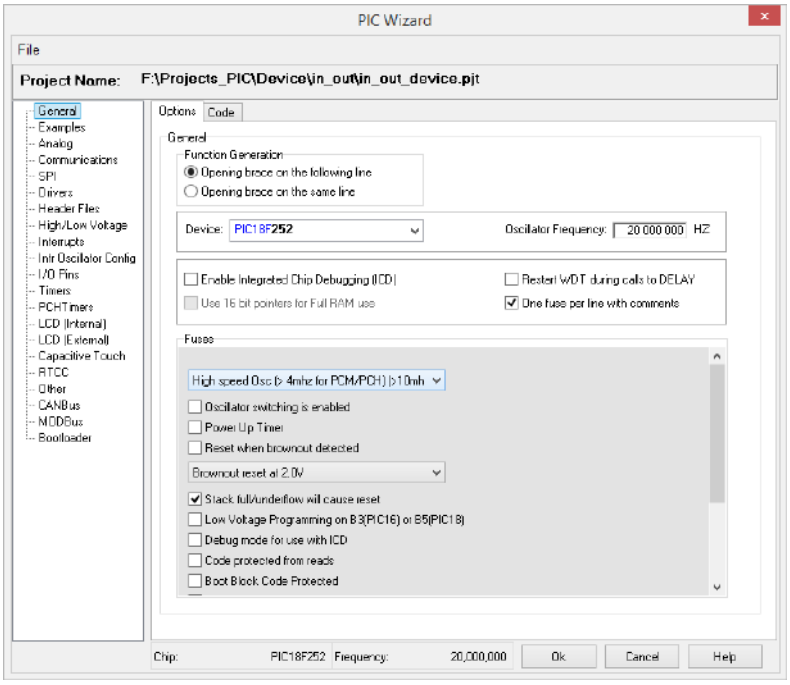


Рисунок 2.16 – Розділ **General** майстра **PIC Wizard**

Для перегляду програмного коду, який буде згенерований і доданий у вихідний файл відповідно до параметрів на поточній вкладці, слід вибрати вкладку **Code** (рис. 2.17).

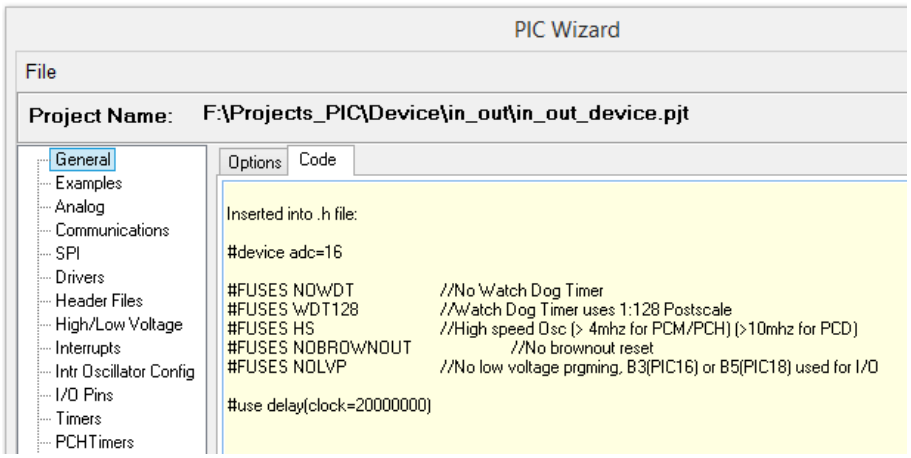


Рисунок 2.17 – Попередній перегляд коду, що генерується майстром **PIC Wizard** для поточного розділу

У розділі **Communications** (рис. 2.18) налаштовуються параметри введення/виведення для інтерфейсів **RS-232** і **I²C** (швидкість обміну даними, відповідні лінії портів введення/виведення, перевірка помилок, режим **Master/Slave** і т.д.), а також активізується/відключається апаратний порт **PSP**.

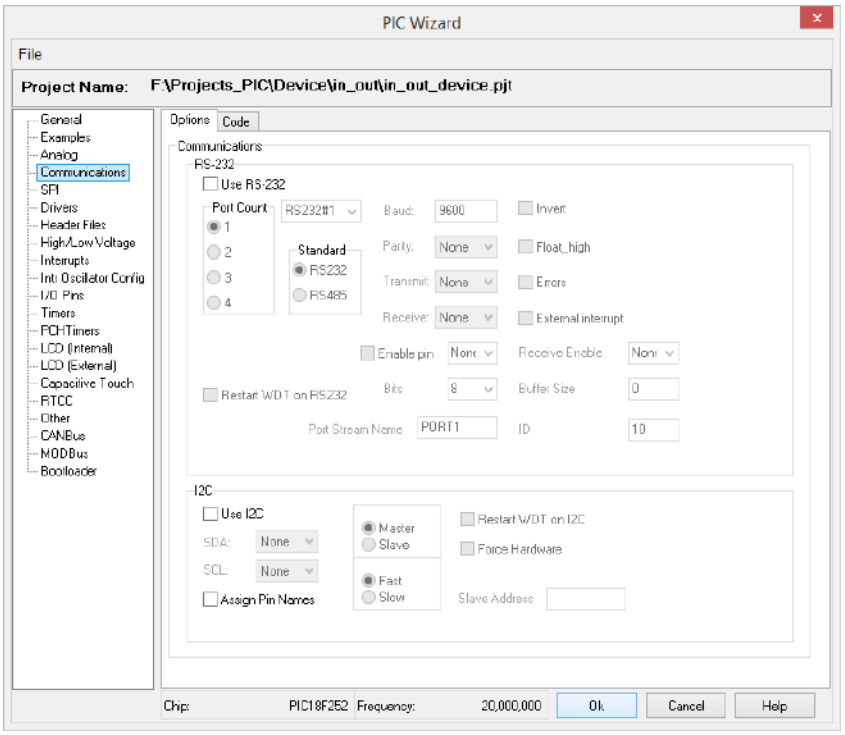


Рисунок 2.18 – Розділ **Communications** майстра **PIC Wizard**

У розділі **SPI** (рис. 2.19) користувач може активізувати апаратний інтерфейс **SPI** і налаштувати відповідні параметри обміну даними (режим **Master/Slave**, активний фронт тактового імпульсу, коефіцієнт ділення частоти системної синхронізації, використання виводу / **SS**).

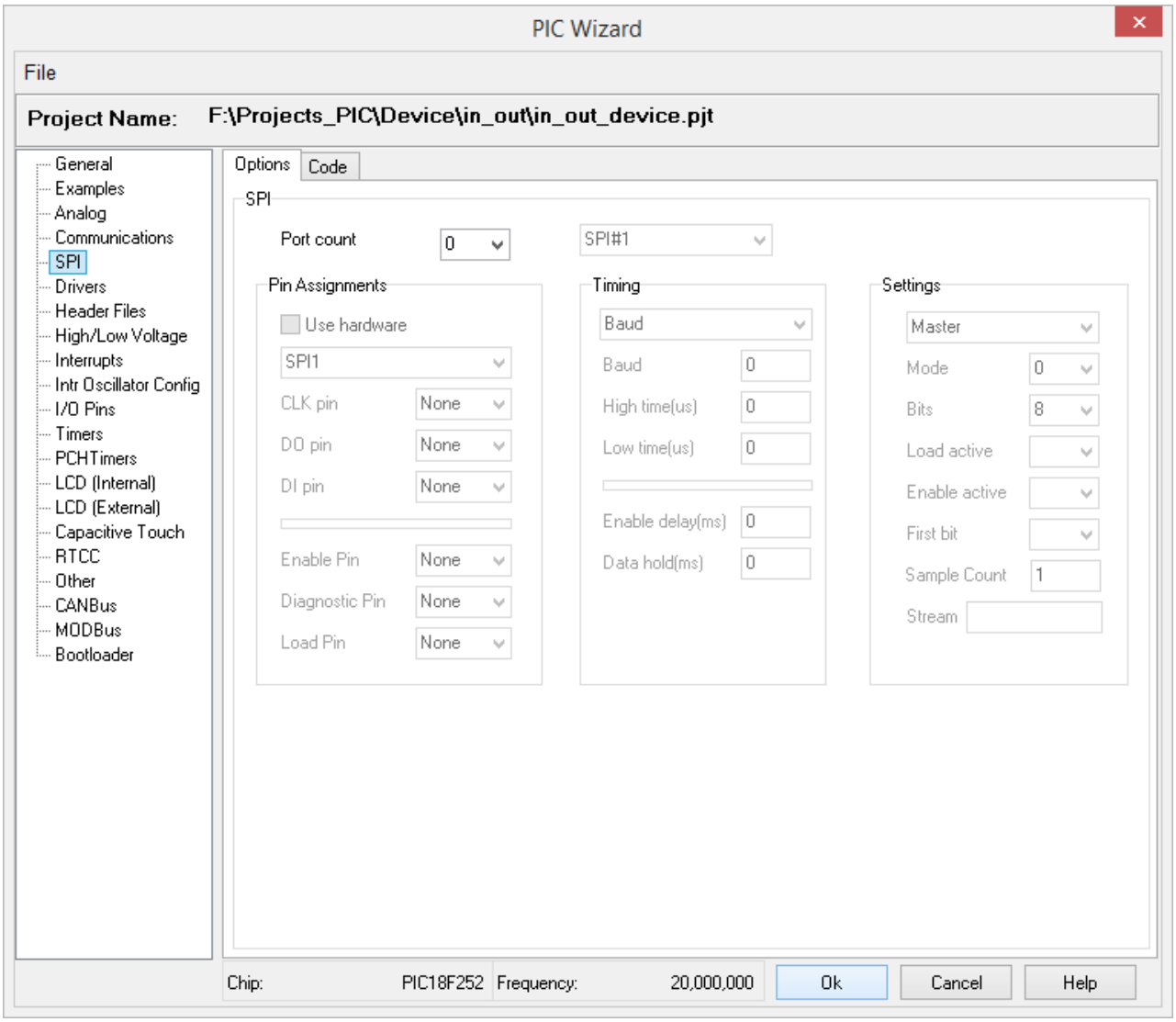


Рисунок 2.19 – Розділ **SPI** майстра **PIC Wizard**

У розділі **Timers** (рис. 2.20) налаштовуються параметри таймерів, включаючи сторожовий.

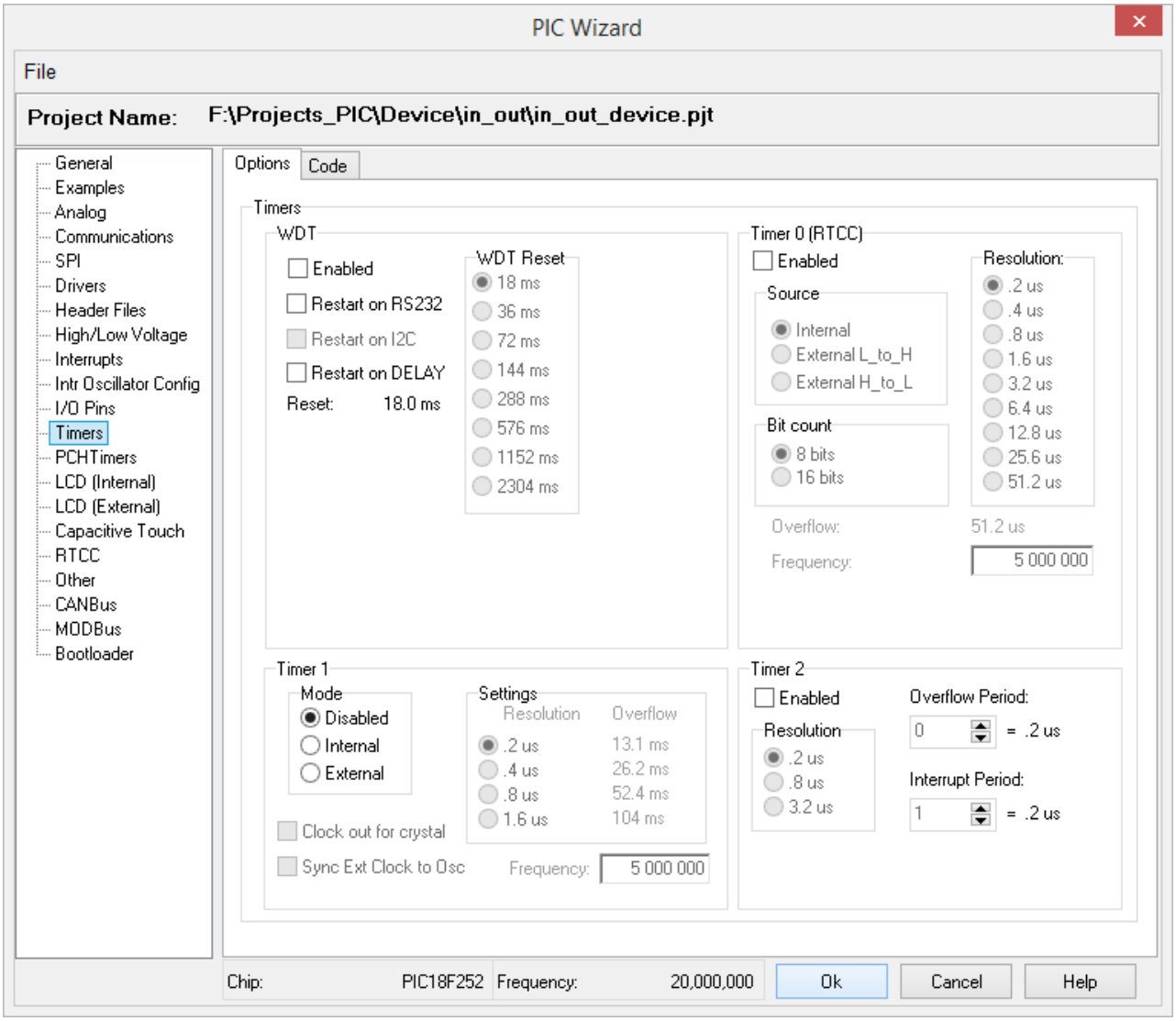


Рисунок 2.20 – Розділ **Timers** майстра **PIC Wizard**

Значення деяких параметрів:

- **WDT Reset** – період між сигналами скидання від сторожового таймера;
- **Source** – вибір джерела тактування таймера TMR0: внутрішній або зовнішній (по наростаючому або спадаючому фронту сигналу);
- **Frequency** – встановлення частоти у випадку вибору зовнішнього тактування таймера TMR0;
- **Resolution** – дозвіл таймера, що впливає на період між переповненнями лічильного регістру (для таймерів TMR0 і TMR1 обчислюється автоматично і відображається у полі **Overflow**);

- **Interrupt Period** – період між запитами на переривання від таймера TMR2.

Якщо обраний мікроконтролер надає додаткові таймери, їх параметри налаштовуються у розділі **PCH Timers** майстра **PIC Wizard**.

Розділ **Analog** (рис. 2.21) служить для налаштування вбудованого АЦП (конфігурація аналогових входів, частота і розрядність перетворення).

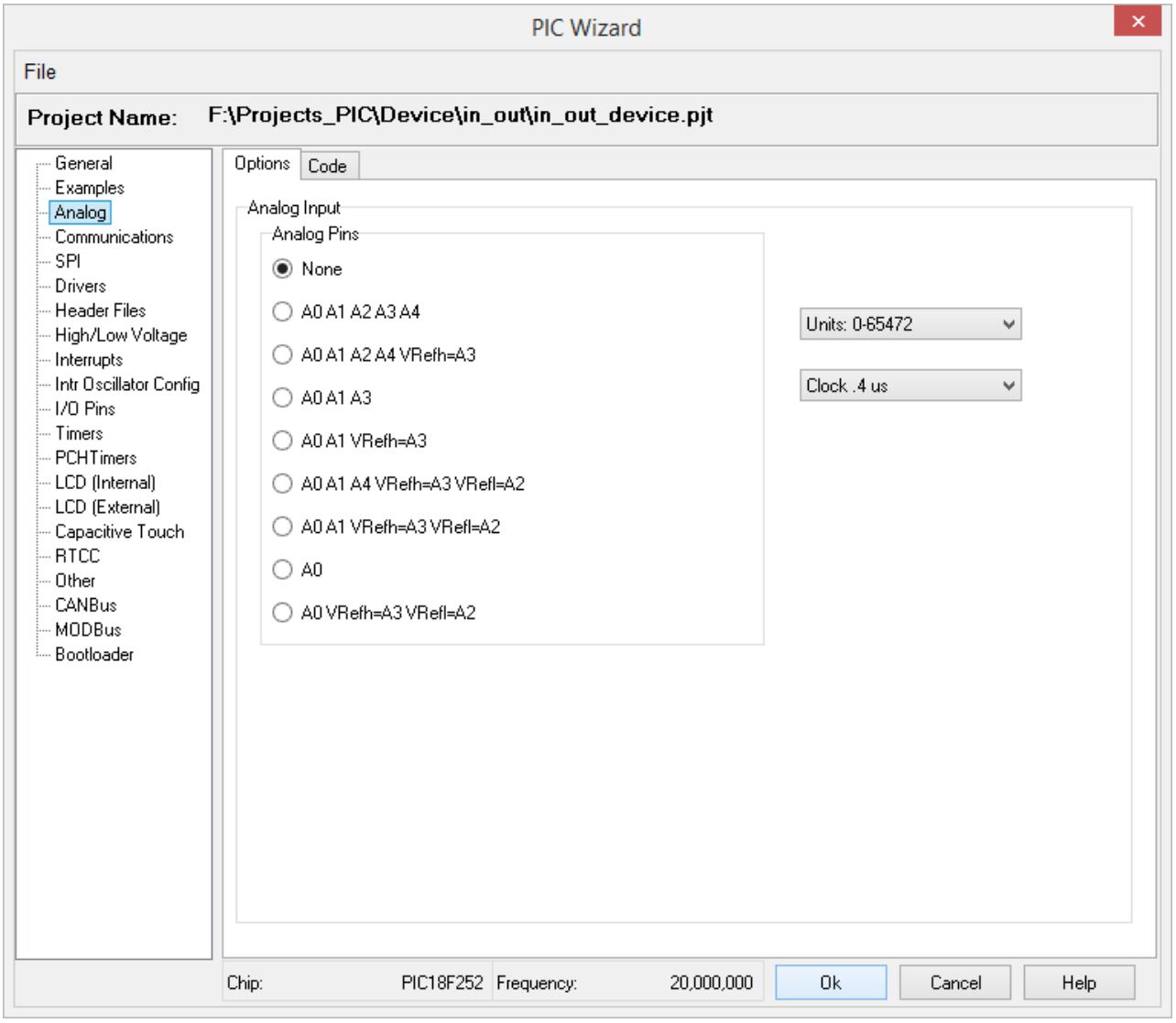


Рисунок 2.21 – Розділ **Analog** майстра **PIC Wizard**

У розділі **Other** (рис. 2.22) активізується модуль **CCP** і налаштовується режим його роботи, а також вибирається конфігурація входів компараторів напруги.

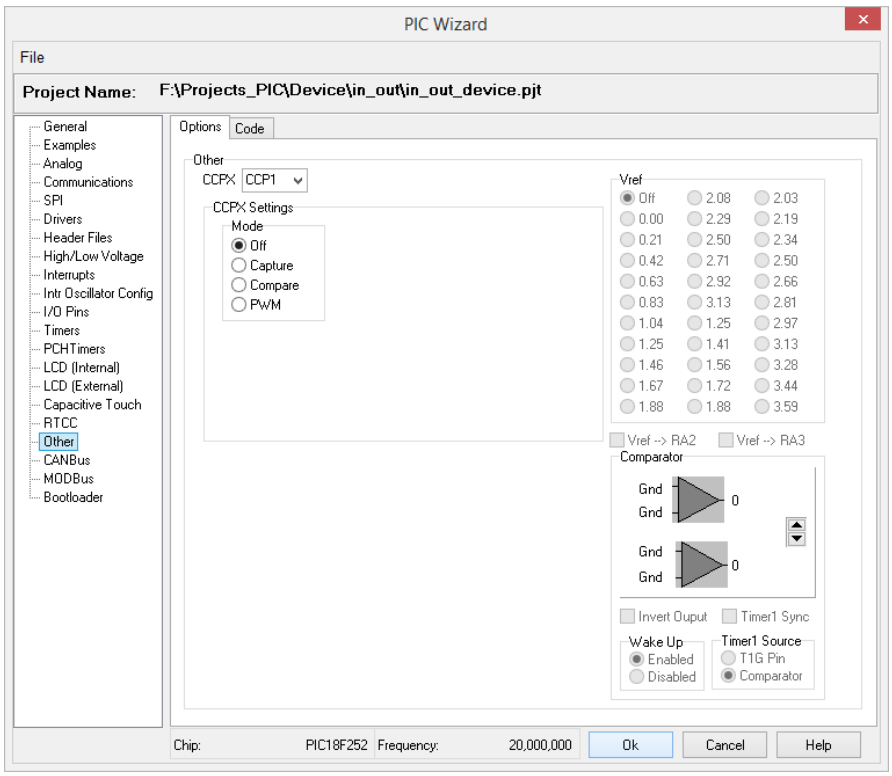


Рисунок 2.22 – Розділ **Other** майстра **PIC Wizard**

Розділ **Interrupts** (рис. 2.23) дозволяє за допомогою набору прапорців дозволити або заборонити те чи інше переривання, доступне для вибраного пристрою.

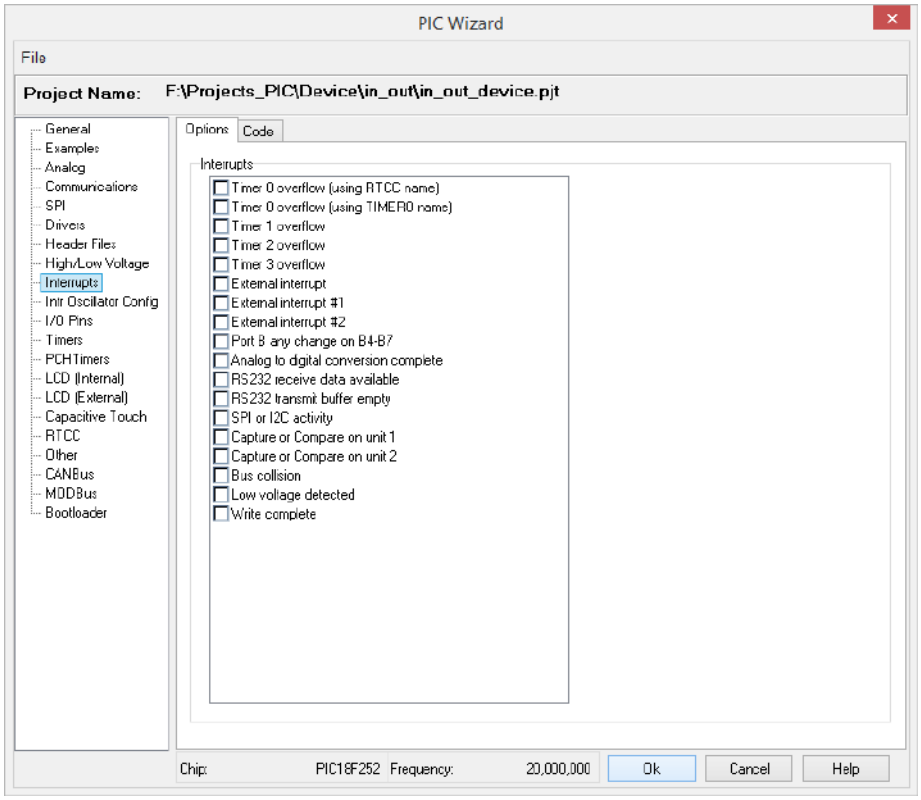


Рисунок 2.23 – Розділ **Interrupts** майстра **PIC Wizard**

Для кожного вибраного переривання у головну процедуру програми `main()` додається виклик відповідної підпрограми обробки переривання `enble_interrupts()`, а тіло самої підпрограми розміщується вище `main()` (приклад: рис. 2.24).

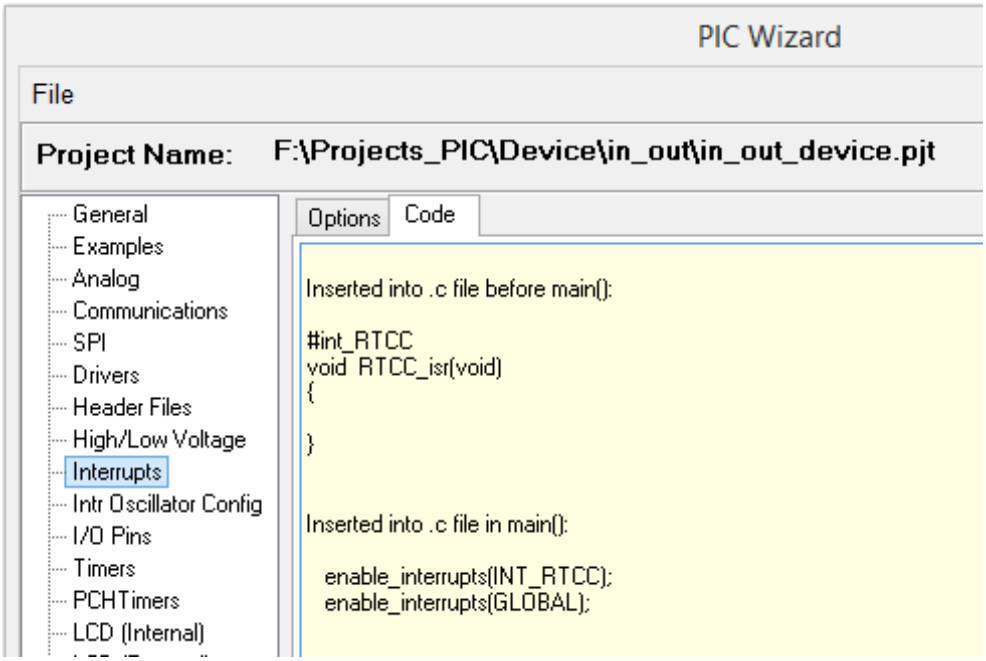


Рисунок 2.24 – Приклад програмного коду, згенерованого майстром **PIC Wizard** у розділі **Interrupts**

У розділі **Drivers** міститься список програмних драйверів, доступних для вибраного мікроконтролера.

Вибір конкретного драйверу призводить до того, що у проект включається відповідний заголовний файл, і викликається підпрограма для ініціалізації цього драйверу (за замовчуванням драйвери розміщені у папці `\Program Files\PICC\Drivers`).

Конфігурація виводів портів мікроконтролера налаштовується у розділі **I/O Pins** (рис. 2.25).

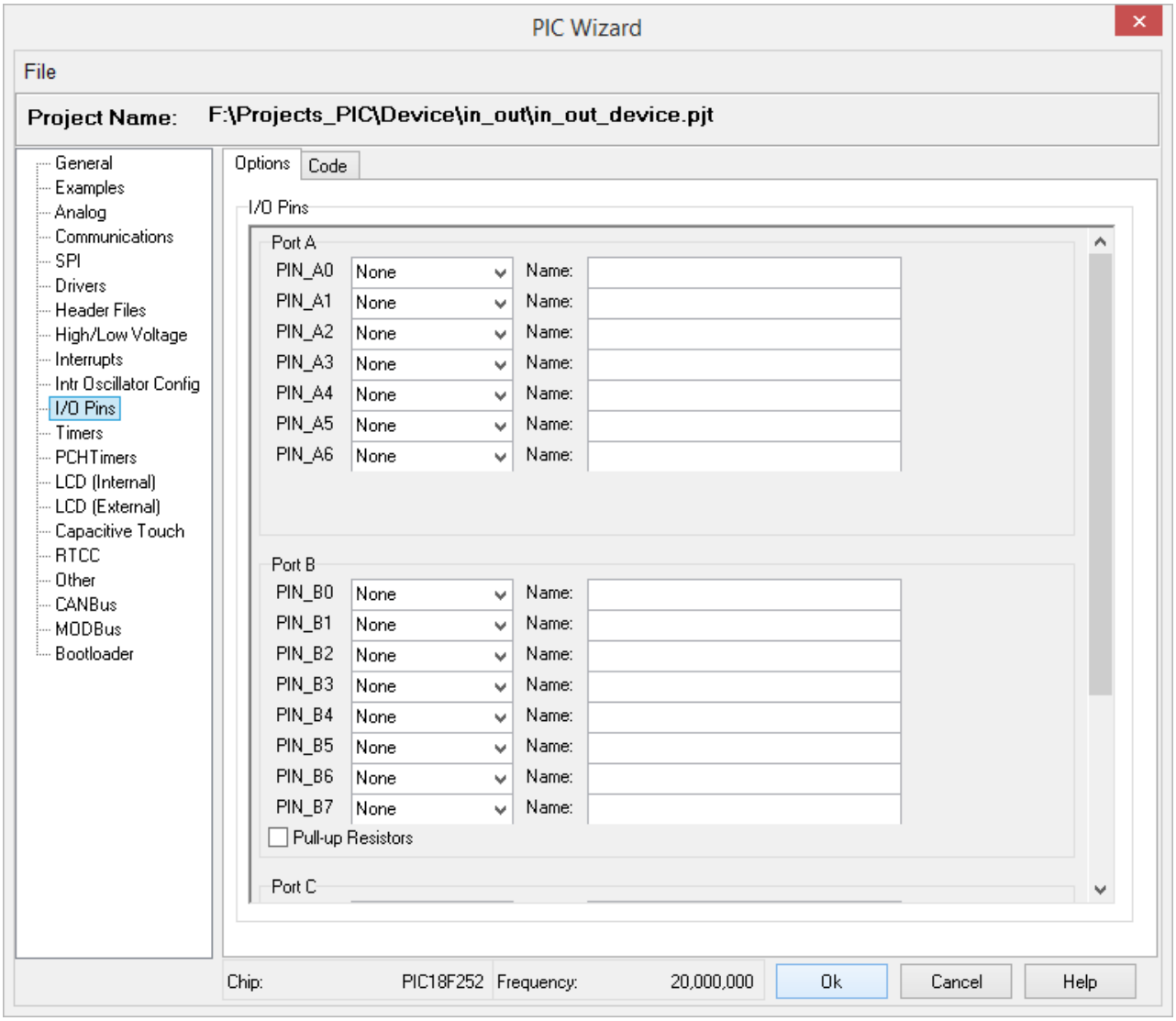


Рисунок 2.25 – Розділ **I/O Pins** майстра **PIC Wizard**

При цьому для кожного виводу можливі значення:

- **Input** (вхід);
- **Output** (вихід);
- **Input/Output** (вхід/вихід);
- **Analog** (аналоговий);
- **Not used** (не використовується).

Кожному виводу у програмі будуть відповідати ідентифікатори, зазначені через кому у стовпці **Identifiers**.

Якщо встановити прапорець **Pull-up Resistors (Port B)** у розділі **I/O Pins**, та прапорець **Timer 0 overlow (using RTCC name)** у розділі **Interrupts** то виводи порту **B** будуть зконфігуровані з підтягуючим резистором:

```
Inserted into .c file in main():
port_B_pullups(0xFF);
```

У розділі майстра **PIC Wizard Header Files** – за допомогою прапорців можна включити у проект додаткові заголовки, які використовуються, наприклад, для роботи з термінами або з числами з плаваючою комою.

Інші розділи майстра служать для налаштування різних апаратних функцій і інтерфейсів, наприклад:

- **High / Low Voltage** – виявлення підвищень і падінь рівня робочої напруги;
- **Internal Oscillator Configuration** – конфігурація внутрішнього осцилятора;
- **CAN Bus** – параметри шини CAN;
- **LCD** – параметри інтерфейсу для підключення ЖК-дисплея;
- **MOD Bus** – параметри шини MOD;
- **Bootloader** – активізація та параметри розміщення завантажувача.

Після налаштування всіх необхідних параметрів, у вікні майстра можна натиснути кнопку **ОК**, і в одній папці з самого початку заданим проектним файлом буде створений файл з розширенням ***.c** з тим же ім'ям, а також – всі необхідні файли, що включаються до заголовочного файлу ***.h**.

Компіляція проекту

Для компіляції поточного проекту можна виконати команду меню **Compile › Compile** або натиснути клавішу **<F9>** (рис. 2.26, рис 2.27).

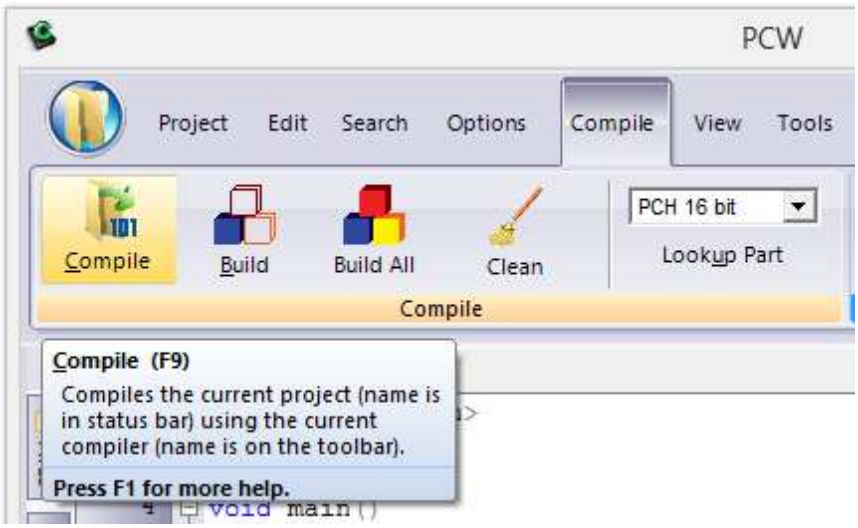


Рисунок 2.26 – Компіляція проекту

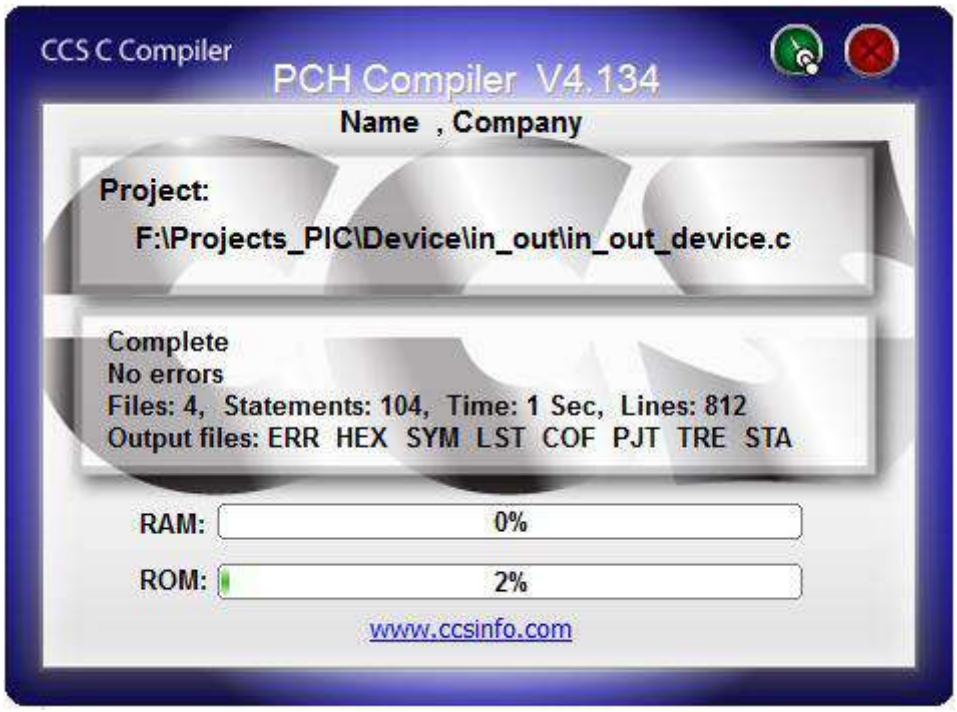


Рисунок 2.27 – Результати роботи компілятора **CCS-PICC**

У результаті компіляції у папці розміщення вихідних файлів будуть створені ще кілька файлів з тим же ім'ям, але іншими розширеннями:

- `.cof` – файл, який використовується у середовищі відлагодження;
- `.err` – файл з переліком помилок (якщо були виявлені); крім того, перша виявлена помилка буде виділена у початковому тексті програми, а її опис – відображений на червоному фоні у рядку стану;
- `.hex` – бінарний файл для завантаження у мікроконтролер;

- `.1st` – файл лістингу, в якому відображені відповідності між операторами на мові **C** і асемблерними наборами команд;
- `.sta` – файл статистики за результатами останньої компіляції;
- `.sym` – файл символів, що містить адреси змінних у пам'яті RAM;
- `.tre` – дерево викликів підпрограм.

У файлі з розширенням `*.h` містяться установки згенеровані **IDE PCWH:**

```
#include <18F252.h>
#device adc=16

#FUSES NOWDT                    //No Watch Dog Timer
#FUSES WDT128                   //Watch Dog Timer uses 1:128 Postscale
#FUSES HS                       //High speed Osc (> 4mhz for PCM/PCH)
                                //(>10mhz for PCD)
#FUSES NOBROWNOUT               //No brownout reset
#FUSES NOLVP                    //No low voltage prgming, B3(PIC16) or
                                //B5(PIC18) used for I/O

#use delay(clock=20000000)
#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8,stream=PORT1)
```

Відкриття створеного проекту

Відкрити проект можна наступним чином:

1. Запустити **IDE PCWH**, натиснути кнопку **Projects › Project** (рис. 2.28):

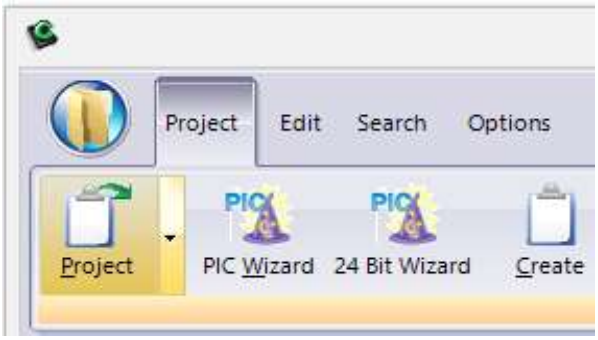


Рисунок 2.28 – Відкриття проекту у **IDE PCWH**
за допомогою майстра **PIC Wizard**

або натиснути кнопку у вигляді відкритої папки  .

2. Вибрати: **Open › Project** (рис. 2.29, рис. 2.30).

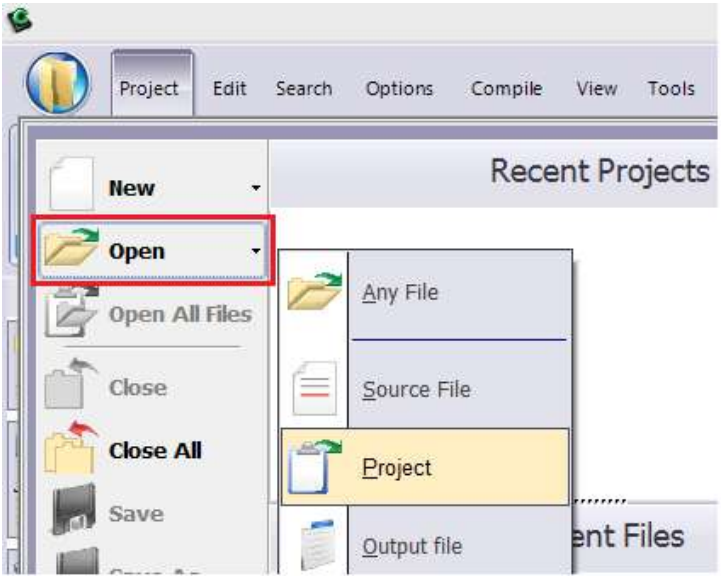


Рисунок 2.29 – Відкриття проекту у **IDE PCWH** за допомогою майстра **PIC Wizard**

3. Знайти і відкрити свій директорій і вибрати проект з розширенням ***.pjt** (рис. 2.30):

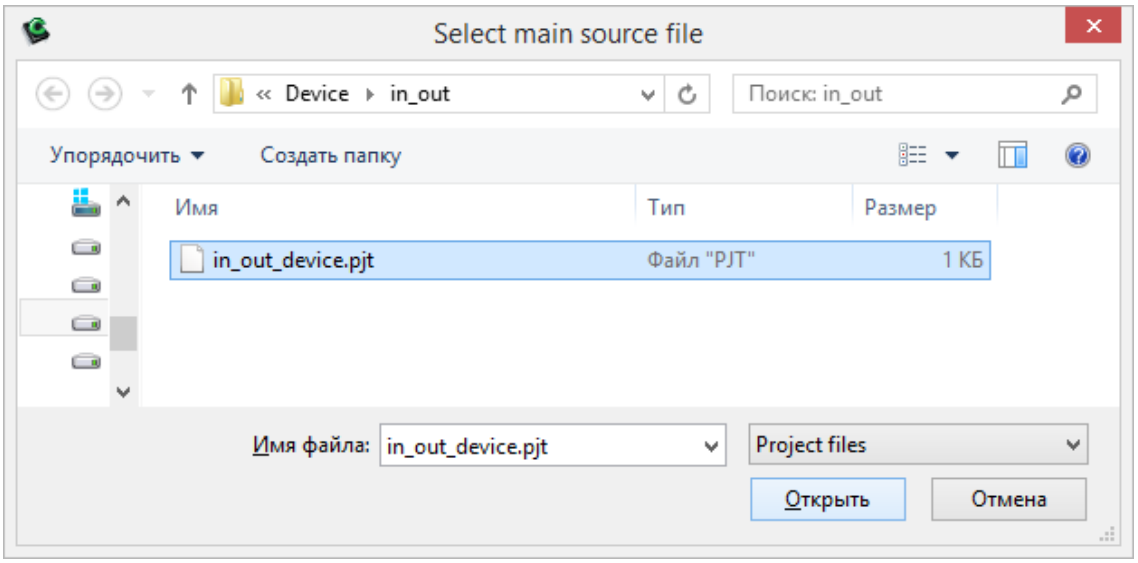


Рисунок 2.30 – Відкриття проекту у **IDE PCWH** за допомогою майстра **PIC Wizard**

Утиліти меню Tools

Меню **Toole** містить команди доступу до різних корисних утиліт:

- **Device Editor** – доступ до бази даних властивостей кожного підтримуваного мікроконтролера PIC;

- **Device Selector** – вибір цільового мікроконтролера і його властивостей;
- **File Compare** – утиліта порівняння файлів (вихідних кодів або лістингів). Якщо вибрано порівняння вихідних файлів, то виконується звичайне порядкове порівняння, якщо ж вибрано порівняння лістингів, то його можна налаштувати таким чином, щоб адреси пам'яті не враховувалися;
- **Numeric Converter** – засіб перетворення цілих і дійсних чисел в десятковому представленні в шістнадцяткове і навпаки;
- **Serial Port Monitor** – монітор послідовного порту для налагодження вбудованих систем, призначених для обміну через інтерфейс RS-232, RS-442, RS-485.

Завантаження бінарного коду у FLASH-пам'ять мікроконтролера

Двійковий файл з розширенням ***.HEX** завантажується у FLASH-пам'ять програм мікроконтролера через порт USB за допомогою програми **PICkit** (рис. 2.31).

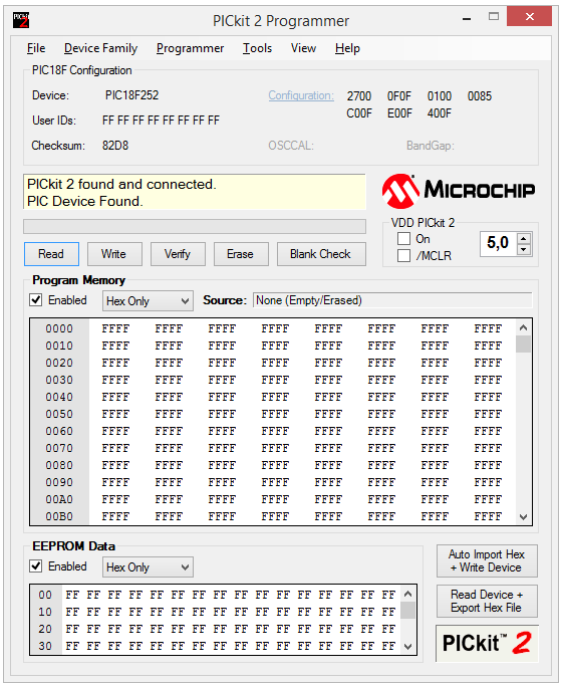


Рисунок 2.31 – Завантаження бінарного коду у FLASH-пам'ять мікроконтролера за допомогою програми **PICkit**

Для завантаження бінарного коду у FLASH-пам'ять мікроконтролера на передній панелі **АПК** натиснути кнопку **Pgm**.

Мікроконтролер готовий до приймання і завантаженню програми.

Потім у меню **File › Import HEX** ***(ctrl+I)*** вибрати ***.HEX**-файл і натиснути кнопку **Write** (рис. 2.32 – рис. 2.35):

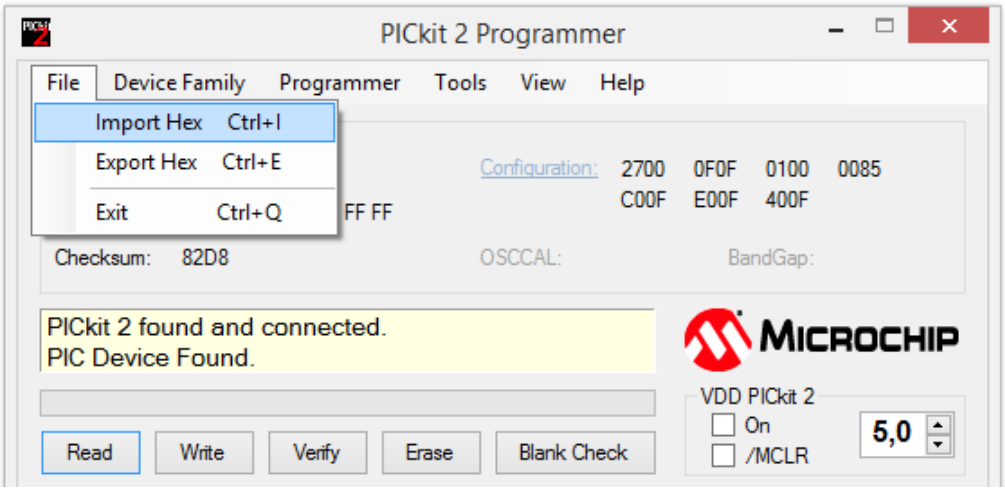


Рисунок 2.32 – Імпорт бінарного файлу «***.HEX**»
за допомогою програми **PICkit**

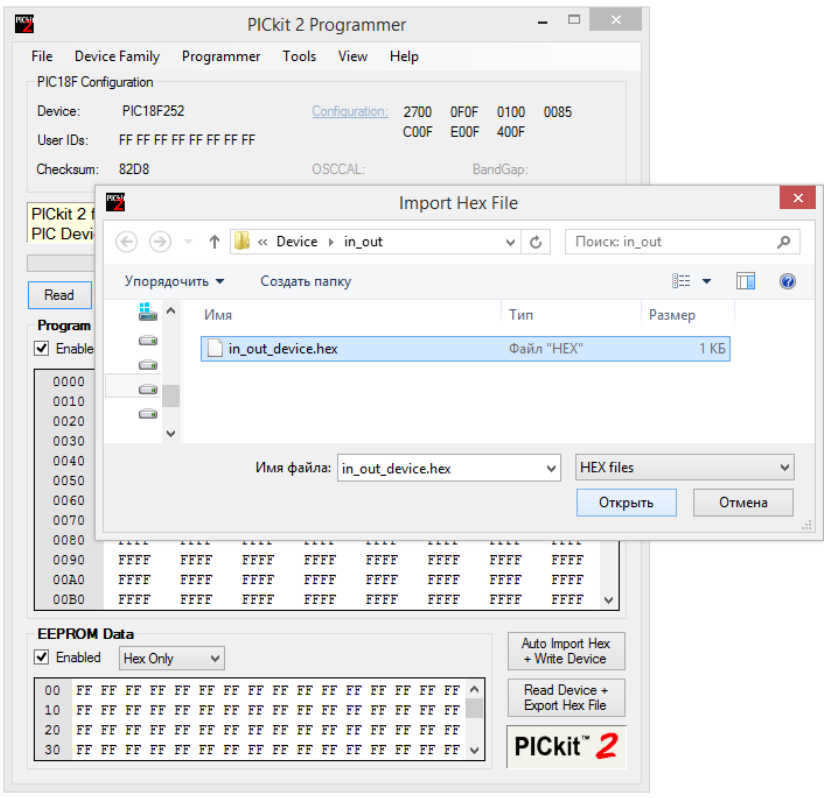


Рисунок 2.33 – Імпорт бінарного файлу «***.HEX**»
за допомогою програми **PICkit**

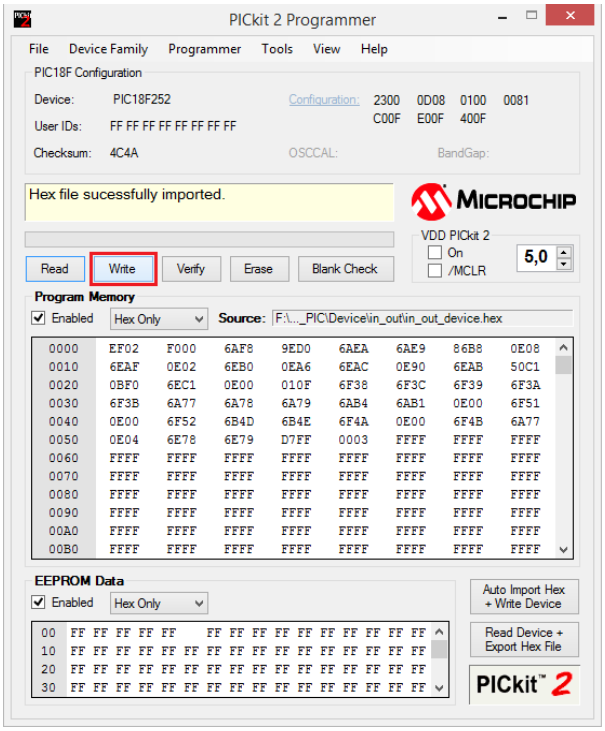


Рисунок 2.34 – Завантаження бінарного файлу «***.HEX**»
у FLASH-пам'ять мікроконтролера

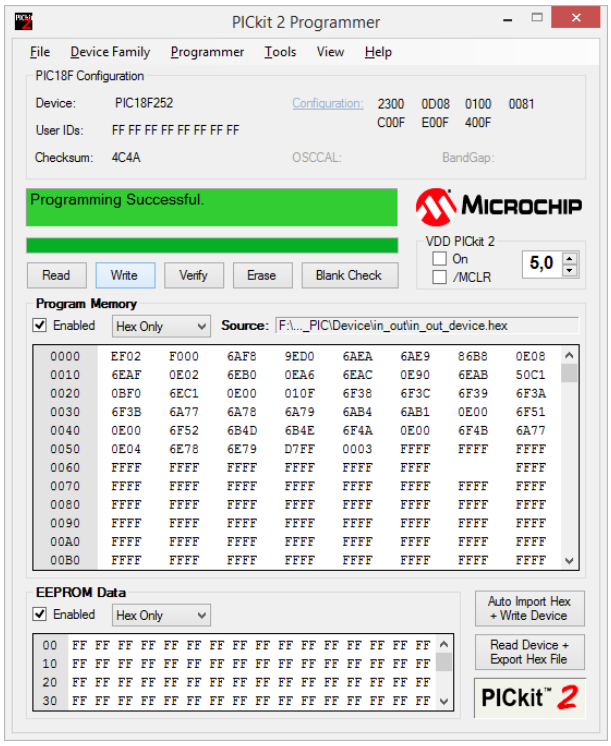


Рисунок 2.35 – Результат завантаження бінарного файлу «***.HEX**»
у FLASH-пам'ять мікроконтролера

Після завантаження програми на **АПК** натиснути кнопки **Pgm**, потім **Reset** і вона відразу починає виконуватися.

ОПИС РОБОТИ З СЕРЕДОВИЩЕМ МОДЕЛЮВАННЯ «PROTEUS»

Порядок виконання лабораторних практикумів для варіанту «Proteus»

1. У папці **«Proteus_students»** вибрати папку відповідного лабораторного практикуму (наприклад, **«IN_OUT»**).

2. Відкрити файл проекту з розширенням ***.DSN**.

або:

1. Відкрити середовище моделювання **«Proteus»** і натиснути на кнопку **«Schematic Capture»** або **«Open Progect»** (рис. 2.36).

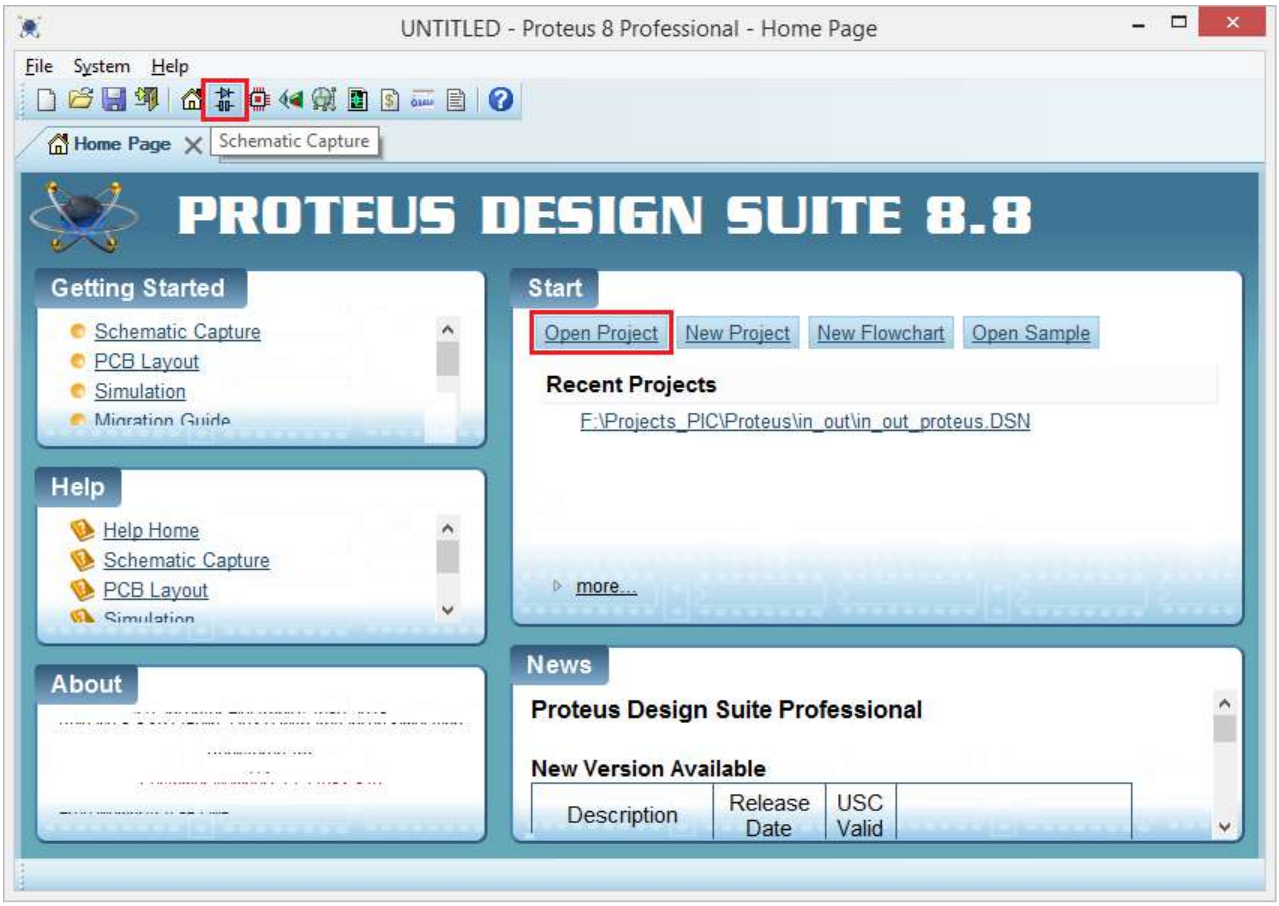


Рисунок 2.36 – Середовище моделювання **«Proteus»**

2. Відкрити файл проекту **File › Open Project (Ctrl+O)** (рис. 2.37) або натиснути на кнопку  .

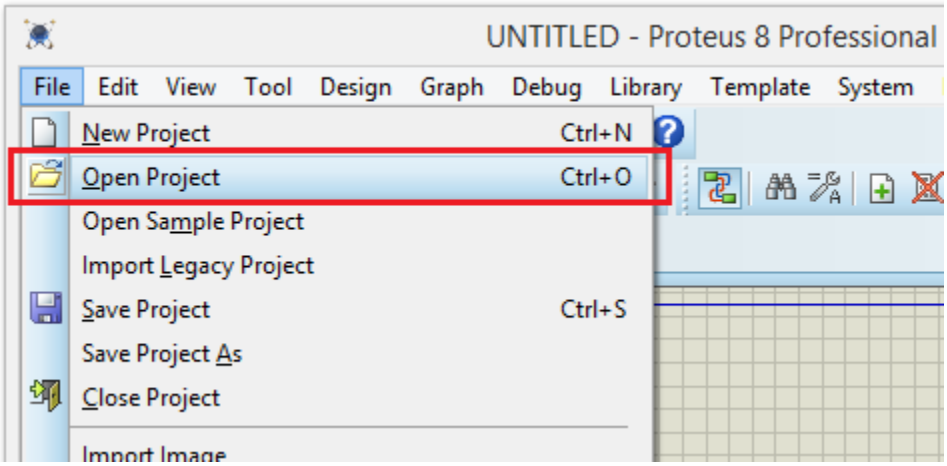


Рисунок 2.37 – Відкриття файлу проекту
у середовищі моделювання **«Proteus»**

3. У папці **«Proteus_students»** вибрати папку відповідного лабораторного практикуму (наприклад, **«IN_OUT»**).

4. Вибрати **«Тип файлов: » All files**.

5. Відкрити файл проекту з розширенням ***.DSN** (рис. 2.38).

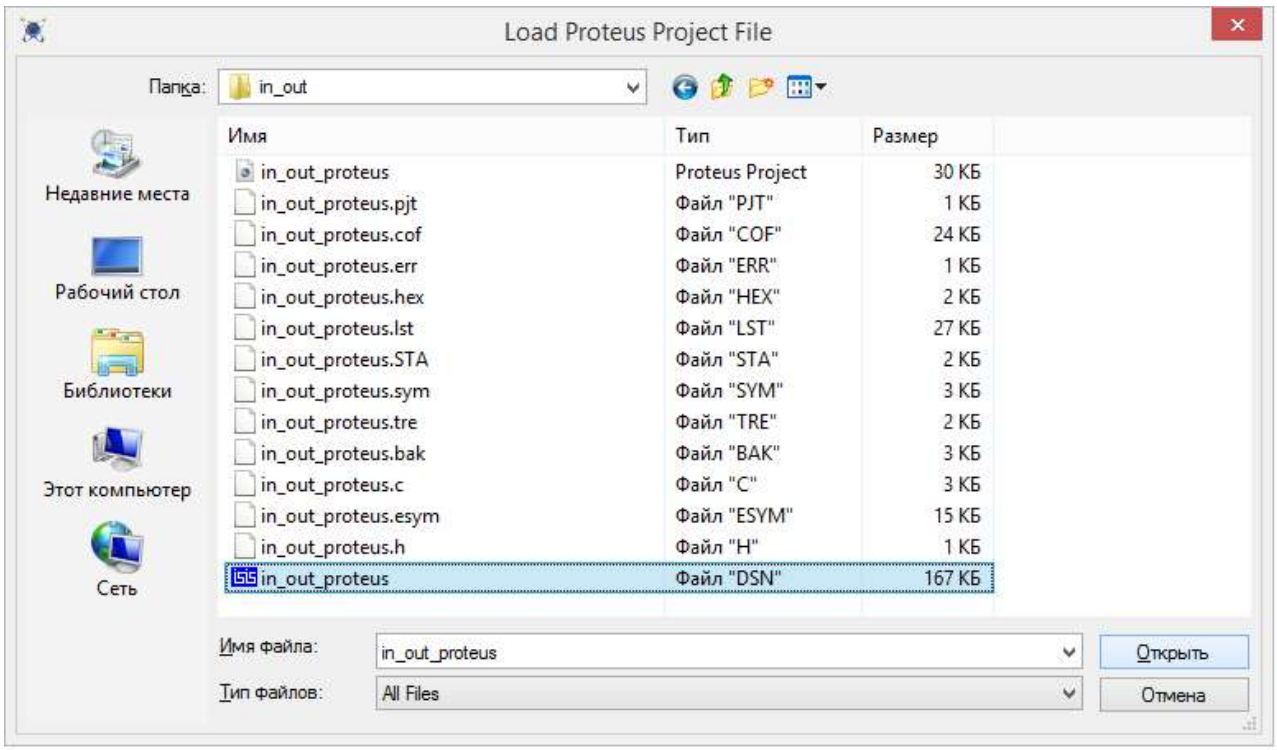


Рисунок 2.38 – Відкриття файлу проекту у
середовищі моделювання **«Proteus»**

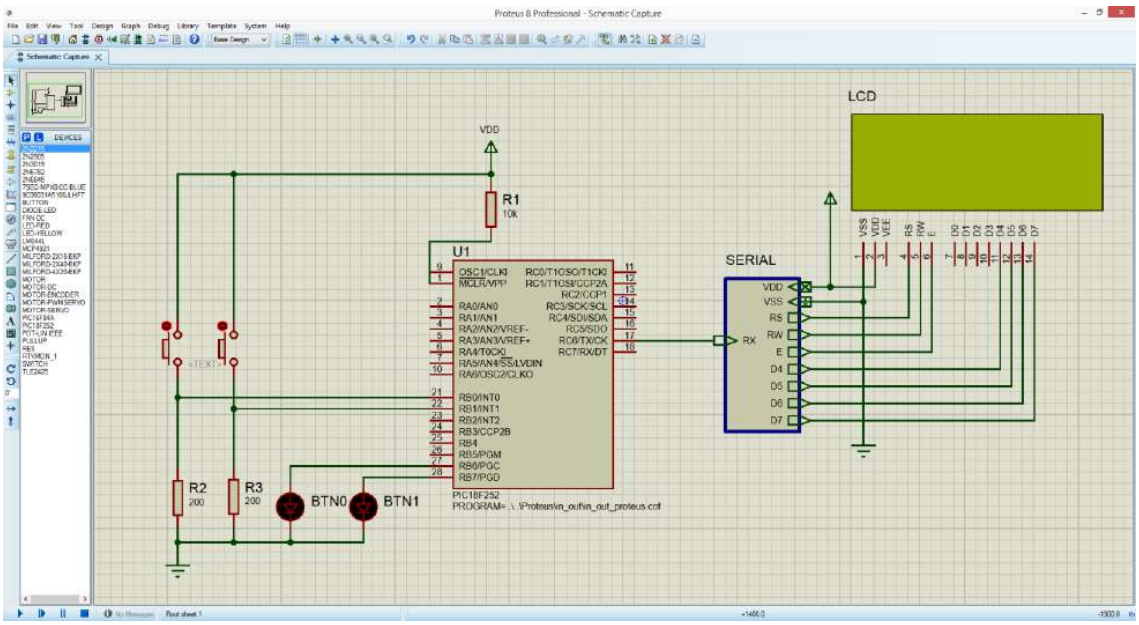


Рисунок 2.39 – Середовище моделювання **«Proteus»**

6. Змінити схему підключень відповідно до варіанту для виконання лабораторного практикуму (рис. 2.39).

7. Натиснути лівою кнопкою миші двічі по мікроконтролеру на схемі.

8. У вікні **Edit Component › Program File** натиснути кнопку (рис. 2.40).

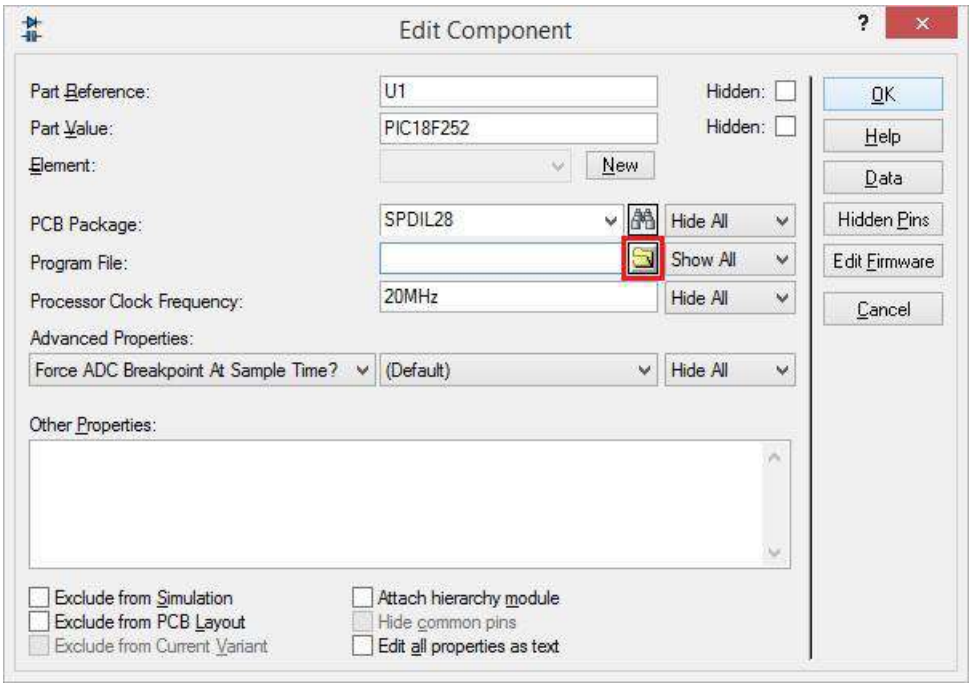


Рисунок 2.40 – Відкриття бінарного файлу з розширенням ***.HEX** або ***.COF** у середовищі моделювання **«Proteus»**

9. Завантажити заздалегідь скомпільований бінарний файл з розширенням ***.HEX** або ***.COF** у мікроконтролер (рис. 2.41).

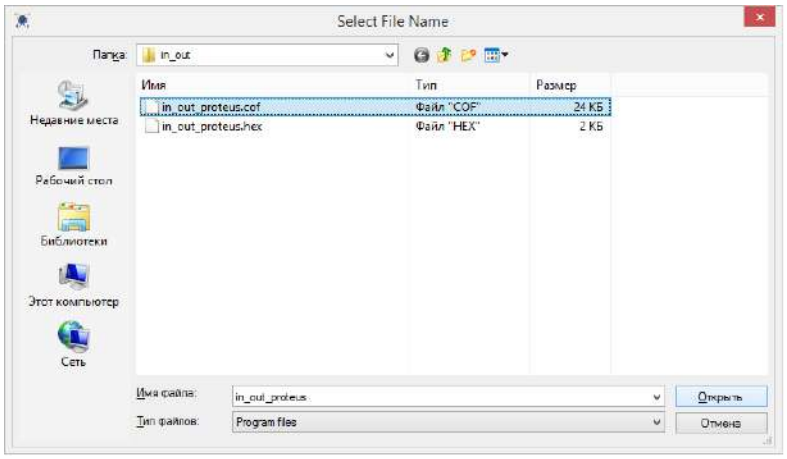


Рисунок 2.41 – Завантаження бінарного файлу з розширенням ***.HEX** або ***.COF** у середовищі моделювання **«Proteus»**

10. У вікні **Edit Component › Program File** натиснути кнопку **OK** (рис. 2.42).

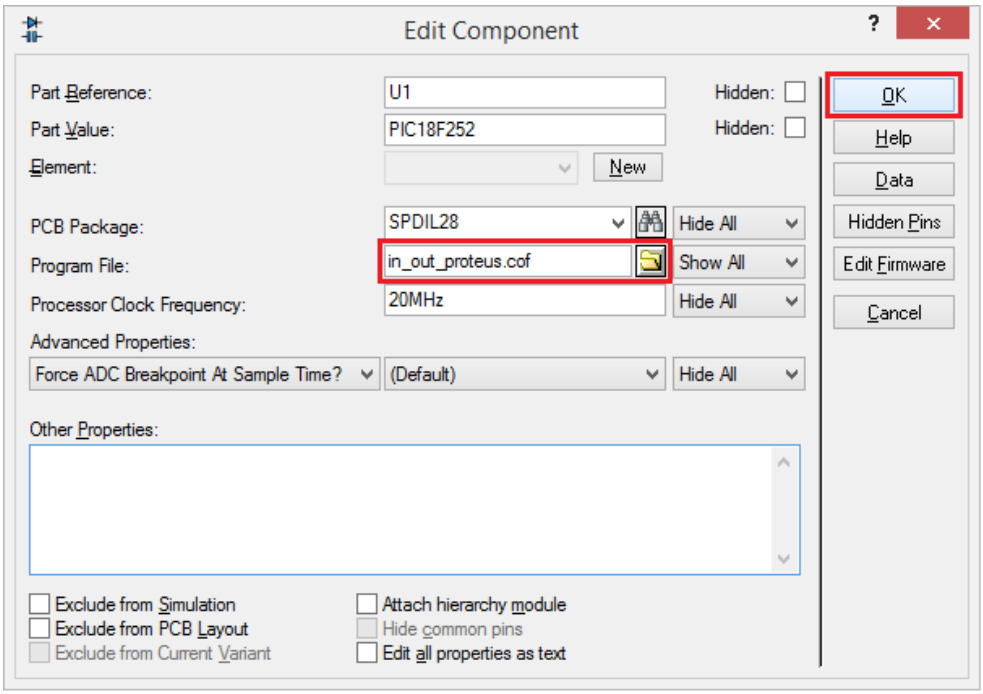


Рисунок 2.42 – Завантаження бінарного файлу з розширенням ***.HEX** або ***.COF** у середовищі моделювання **«Proteus»**

11. У середовищі моделювання **«Proteus»** виконати програму натиснувши кнопку **(Run the simulation).**

РОЗДІЛ 3. ЛАБОРАТОРНІ ПРАКТИКУМИ

Для мікроконтролерів PIC розробка програмного забезпечення здійснюється мовою програмування **C** у інтегрованому середовищі розробки **PCWH** фірми **CCS**.

Всі лабораторні практикуми можуть бути виконані і налагоджені за допомогою **апаратно-програмних комплексів (АПК)** (рис 2.1, рис 2.2) та у **програмі-симуляторі «Proteus»** (рис. 2.36).

Виконання лабораторних практикумів

Шаблон розробки принципової схеми лабораторних практикумів для варіанту **АПК** представлений на рис. 3.1:

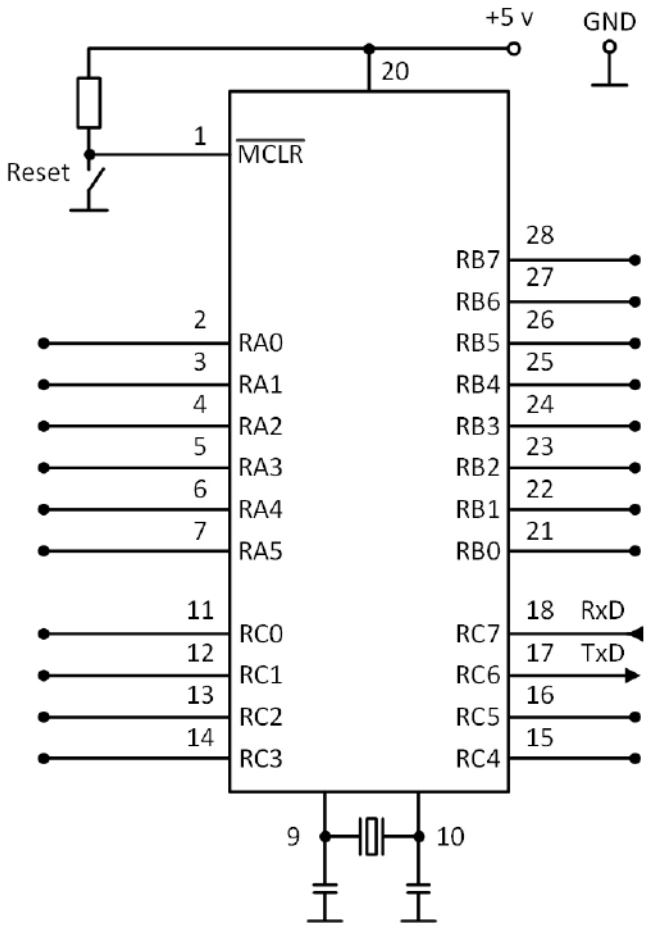


Рисунок 3.1 – Шаблон розробки принципової схеми лабораторних практикумів для варіанту **АПК**

Загальна схема процесу розробки програм для варіанту **АПК** виглядає у такий спосіб (рис. 3.2):

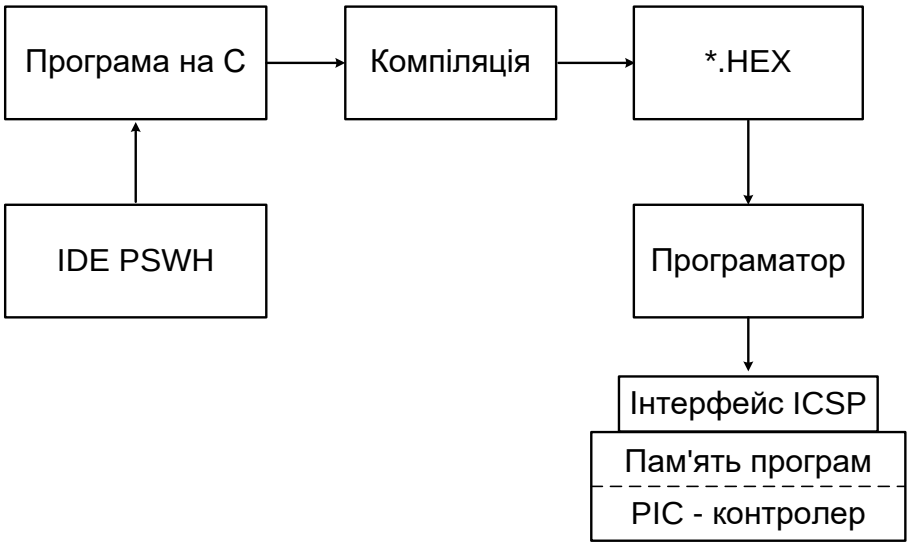


Рисунок 3.2 – Процес розробки програм для варіанту **АПК**

Загальна схема процесу розробки для **варіанту «Proteus»** виглядає наступний чином (рис. 3.3):

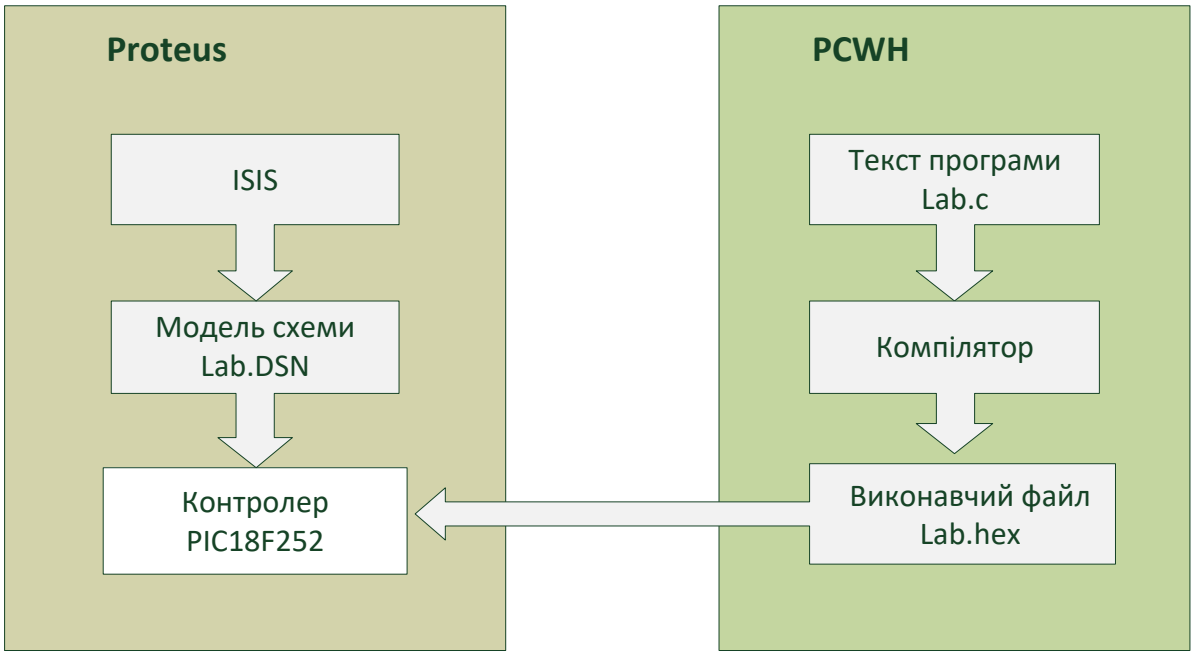


Рисунок 3.3 – Процес розробки програм для **варіанту «Proteus»**

ЛАБОРАТОРНИЙ ПРАКТИКУМ № 1 - «IN_OUT»

Тема: Введення/виведення дискретних сигналів

Ціль роботи:

Отримання навичок роботи з **IDE PCWH**, програмування мікроконтролерів, написання і налагодження програми введення/виведення дискретних сигналів.

Завдання лабораторного практикуму

- для варіанту **АПК** намалювати принципову схему підключень відповідно до варіанту для виконання лабораторного практикуму (таб. 3.1.1);
- для варіанту **«Proteus»** використовувати готову схему відповідно до варіанту для виконання лабораторного практикуму;
- створити алгоритм програми;
- здійснити введення дискретних сигналів із кнопок і тумблерів **АПК** відповідно до варіанту для виконання лабораторного практикуму;
- здійснити виведення результатів на LCD і на світлодіоди відповідно до варіанту для виконання лабораторного практикуму.

Виведення на LCD повинне містити:

- номер лабораторного практикуму;
- групу студента;
- прізвище та ініціали студента;
- вивід мікроконтролера, по якому у цей момент здійснюється читання сигналу.

Варіанти для виконання лабораторного практикуму «IN_OUT»

Таблиця 3.1.1 – Варіанти для виконання лабораторного практикуму **«IN_OUT»**

№	Введення (кнопки)								Виведення (світлодіоди)							
	A0	**A1**	**A2**	**A3**	**B0**	**B1**	**B2**	**B3**	**B0**	**B1**	**B2**	**B3**	**B4**	**B5**	**B6**	**B7**
1	+	+							+	+						
2		+	+							+	+					
3			+	+							+	+				
4	+			+					+			+				
5					+	+							+	+		
6						+	+							+	+	
7							+	+							+	+
8		+		+						+		+				
9					+		+						+	+		
10						+		+						+		+
11	+			+						+						+
12	+		+								+				+	
13		+		+								+		+		
14					+			+					+		+	
15						+		+							+	+

Послідовність виконання лабораторного практикуму для варіанту АПК

1) для варіанту **АПК** намалювати принципову схему підключень відповідно до варіанту для виконання лабораторного практикуму;
2) створити свій директорій для файлів проекту;
3) відкрити інтегроване середовище розробки **PCWH**;
4) створити проект у інтегрованому середовищі розробки **PCWH**;
5) створити алгоритм програми;
6) написати програму мовою програмування **C**, що реалізує створений алгоритм, відповідно до варіанту для виконання лабораторного практикуму;

7) скомпілювати програму, одержати бінарні файли з розширеннями ***.HEX** і ***.COF** (файли з розширеннями ***.HEX** і ***.COF** створюються підчас компіляції програми);
8) завантажити бінарний файл з розширенням ***.HEX** у пам'ять програм мікроконтролера програмою **PICkit.exe**;
9) на **АПК** зкумутувати схему підключень (кнопки, тумблери і світлодіоди) відповідно до варіанту для виконання лабораторного практикуму;
10) виконати програму.

Послідовність виконання лабораторного практикуму для варіанту «Proteus»

1) для середовища моделювання **«Proteus»** використовувати готову схему;
2) створити свій директорій для файлів проекту;
3) відкрити інтегроване середовище розробки **PCWH**;
4) створити проект у інтегрованому середовищі розробки **PCWH**;
5) створити алгоритм програми;
6) написати програму мовою програмування **C**, що реалізує створений алгоритм, відповідно до варіанту для виконання лабораторного практикуму;
7) скомпілювати програму, одержати бінарні файли з розширеннями ***.HEX** і ***.COF** (файли з розширеннями ***.HEX** і ***.COF** створюються підчас компіляції програми);
8) відкрити середовище моделювання **«Proteus»**;
9) у папці **«Proteus_students»** вибрати папку лабораторного практикуму **«IN_OUT»**;
10) відкрити файл проекту з розширенням ***.DSN**;
11) у середовищі моделювання **«Proteus»** змінити схему підключень відповідно до варіанту для виконання лабораторного практикуму;

12) завантажити заздалегідь скомпільований бінарний файл з розширенням ***.COF** або ***.HEX** у мікроконтролер;
13) у середовищі моделювання **«Proteus»** виконати програму.

Послідовність виконання лабораторного практикуму для варіанту «Proteus» з використанням шаблону програми

1) для середовища моделювання **«Proteus»** використовувати готову схему;
2) відкрити папку **«Proteus_students»**;
3) відкрити інтегроване середовище розробки **PCWH**;
4) у папці **«Proteus_students»** вибрати папку лабораторного практикуму **«IN_OUT»**;
5) відкрити файл проекту з розширенням ***.pjt**;
6) у редакторі **IDE PCWH** відкрити шаблон файлу програми з розширенням ***.c**;
7) відкрити середовище моделювання **«Proteus»**;
8) у папці **«Proteus_students»** вибрати папку лабораторного практикуму **«IN_OUT»**;
9) відкрити файл проекту з розширенням ***.DSN**;
10) у середовищі моделювання **«Proteus»** змінити схему підключень відповідно до варіанту для виконання лабораторного практикуму;
11) створити алгоритм програми;
12) у інтегрованому середовищі розробки **PCWH**, використовуючи шаблон програми самостійно написати програму мовою програмування **C**, що реалізує створений алгоритм, відповідно до варіанту для виконання лабораторного практикуму;
13) скомпілювати програму, одержати бінарні файли з розширеннями ***.HEX** і ***.COF** (файли з розширеннями ***.HEX** і ***.COF** створюються підчас компіляції програми);
14) у середовищі моделювання **«Proteus»** виконати програму.

Примітка: *файл демонстрації виконання лабораторного практикуму* ***«IN_OUT»*** *розташований на сайті* http://pksm.kntu.kr.ua/SCME.html.

Приклад створення проекту у інтегрованому середовищі розробки PCWH

Створення проекту у інтегрованому середовищі розробки PCWH за допомогою майстра PIC Wizard

Процес створення проекту у інтегрованому середовищі розробки **PCWH** за допомогою майстра **PIC Wizard** розглянемо на прикладі лабораторного практикуму «**IN_OUT**».

Для запуску майстра **PIC Wizard** слід виконати відповідну команду меню **Project**, а потім вказати розміщення і ім'я головного файлу проекту.

В результаті відкриється вікно майстра, що складається з розділів з параметрами проекту (рис. 3.1.4).

Зовнішній вигляд вікна майстра може відрізнятися в залежності від версії компілятора **CCS-PICC** і обраного типу мікроконтролера.

1. Запустити **IDE**, натиснути кнопку **Projects** майстра **PIC Wizard** (рис. 3.1.1),

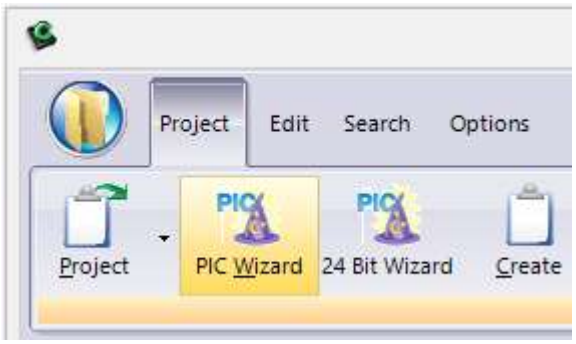


Рисунок 3.1.1 – Процес створення проекту у **IDE PCWH** за допомогою майстра **PIC Wizard**

або натиснути кнопку у вигляді відкритої папки  .

2. Вибрати: **New › Project Wizard** (рис. 3.1.2):

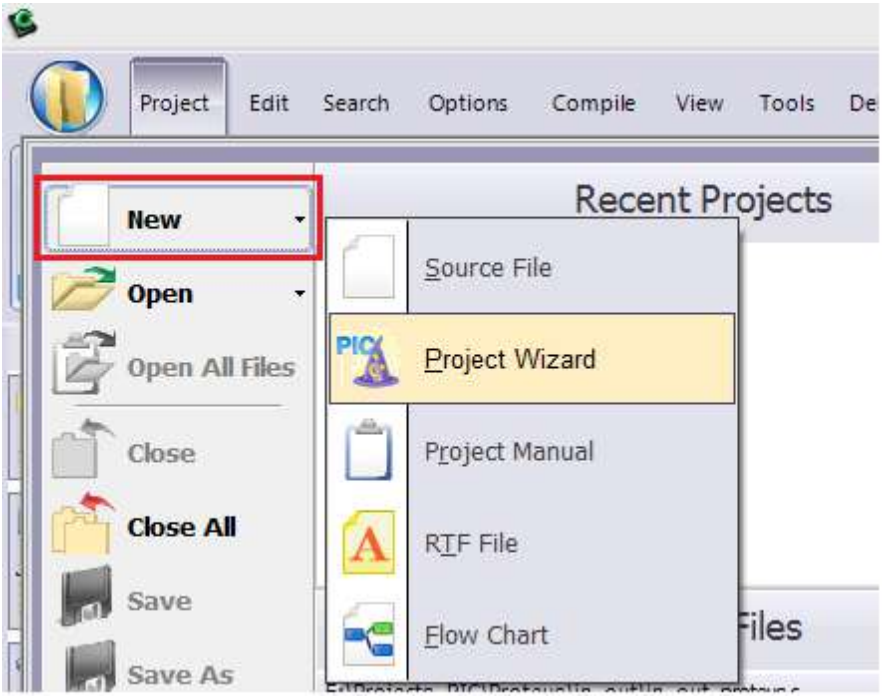


Рисунок 3.1.2 – Процес створення проекту у **IDE PCWH** за допомогою майстра **PIC Wizard**

3. Створити або вибрати свій директорій і записати у нього проект під будь-яким іменем латинським шрифтом, наприклад: «**in_out.pjt**» (рис. 3.1.3):

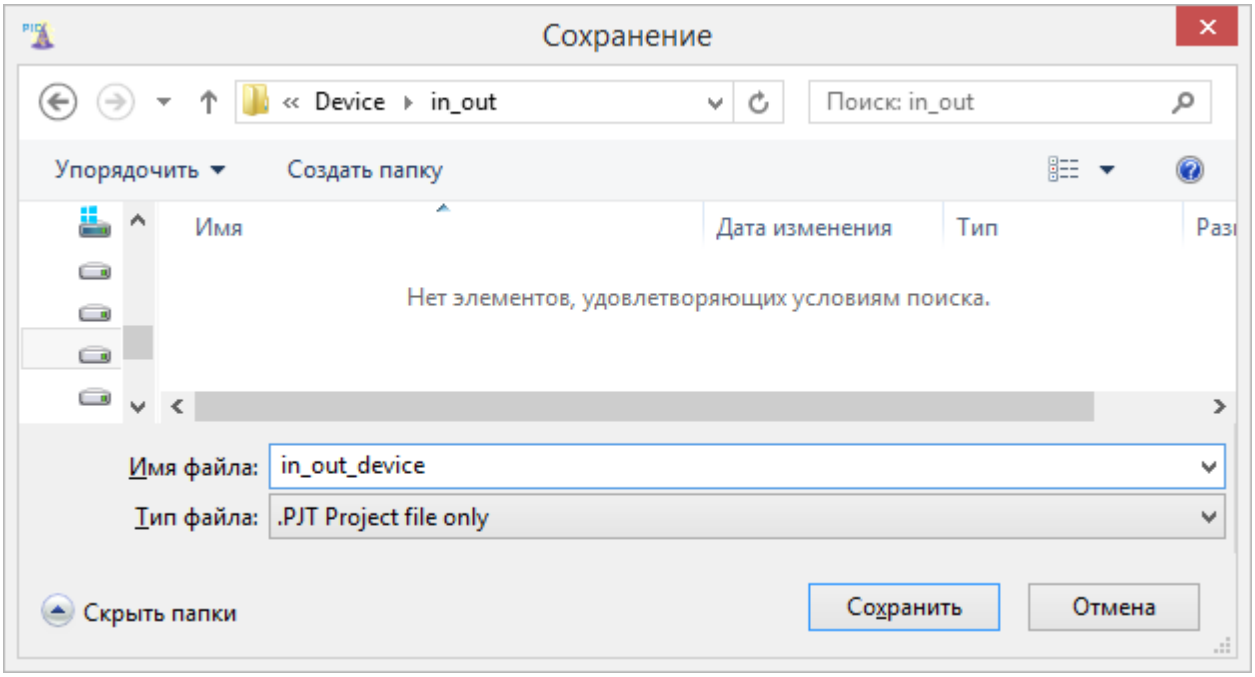


Рисунок 3.1.3 – Процес створення проекту у **IDE PCWH** за допомогою майстра **PIC Wizard**

У розділі **General** (рис. 3.1.4) вибирається цільовий мікроконтролер (список **Device**, що розкривається), його робоча частота (поле **Oscillator Frequency**), а також настроюються загальні параметри, на зразок розрядів запобігання, типу джерела системної синхронізації, порогової напруги для скидання та ін.

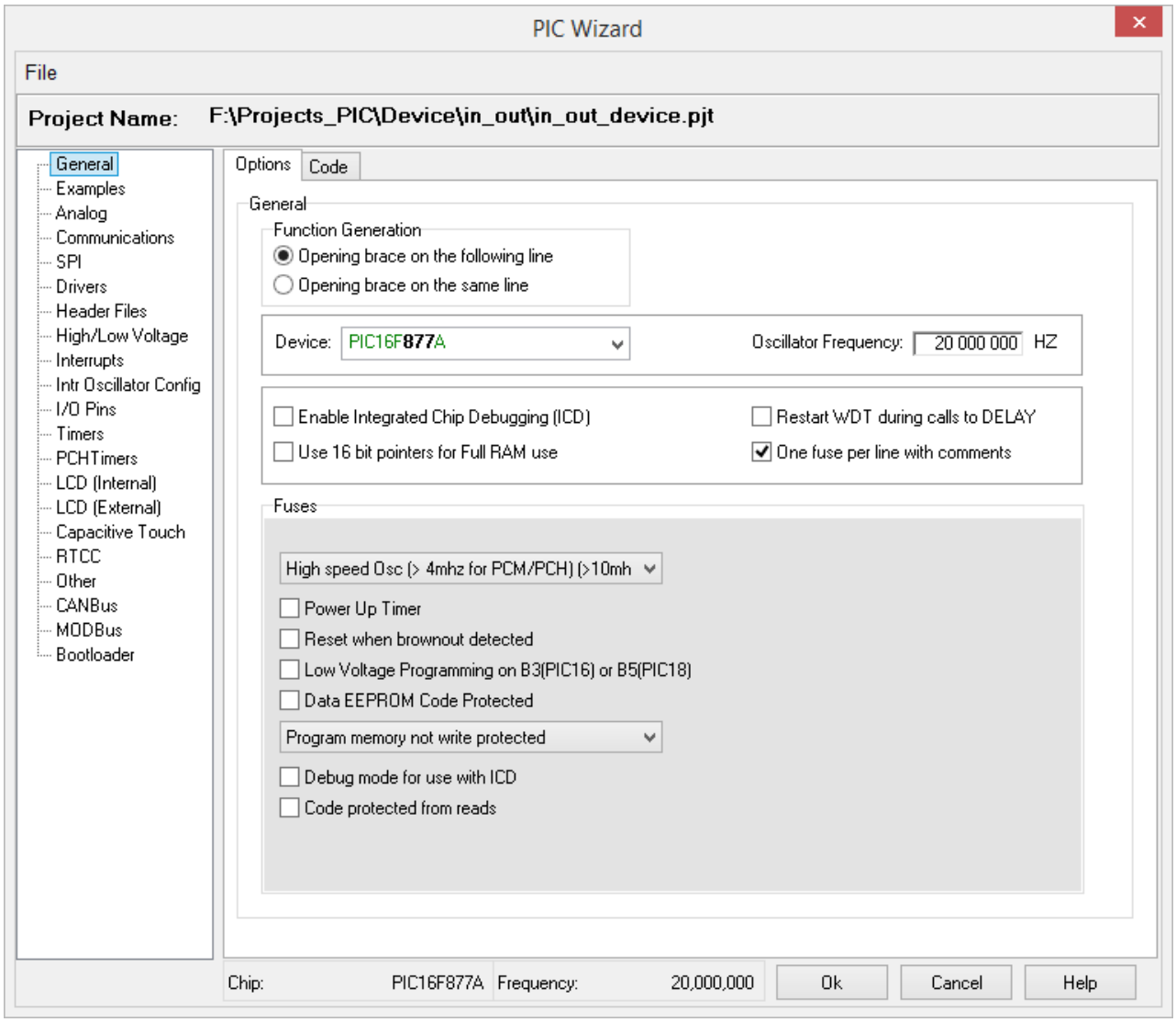


Рисунок 3.1.4 – Розділ **General** майстра **PIC Wizard**

Для перегляду програмного коду, який буде згенерований і доданий у вихідний файл відповідно до параметрів на поточній вкладці, слід вибрати вкладку **Code**.

4. У розділі **General › Device** вибрати тип мікроконтролера, наприклад **PIC18F252** (рис. 3.1.4, рис. 3.1.5):

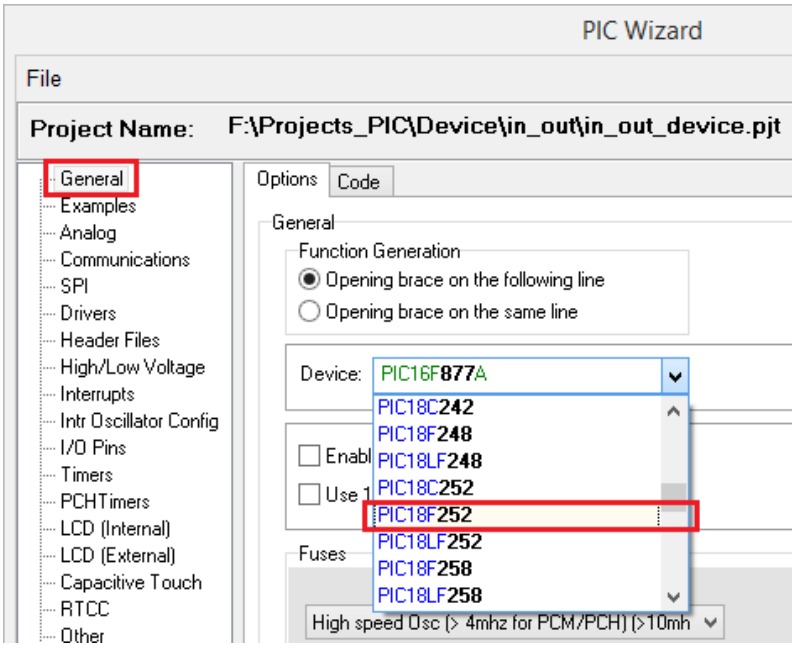


Рисунок 3.1.5 – Процес створення проекту **IDE PCWH** за допомогою майстра **PIC Wizard**

5. У розділі **General › Fuses** вибрати тип генератора **High speed Osc (> 4mhz for PCM/PCH) (>10mhz for PCD)** (рис. 3.1.6):

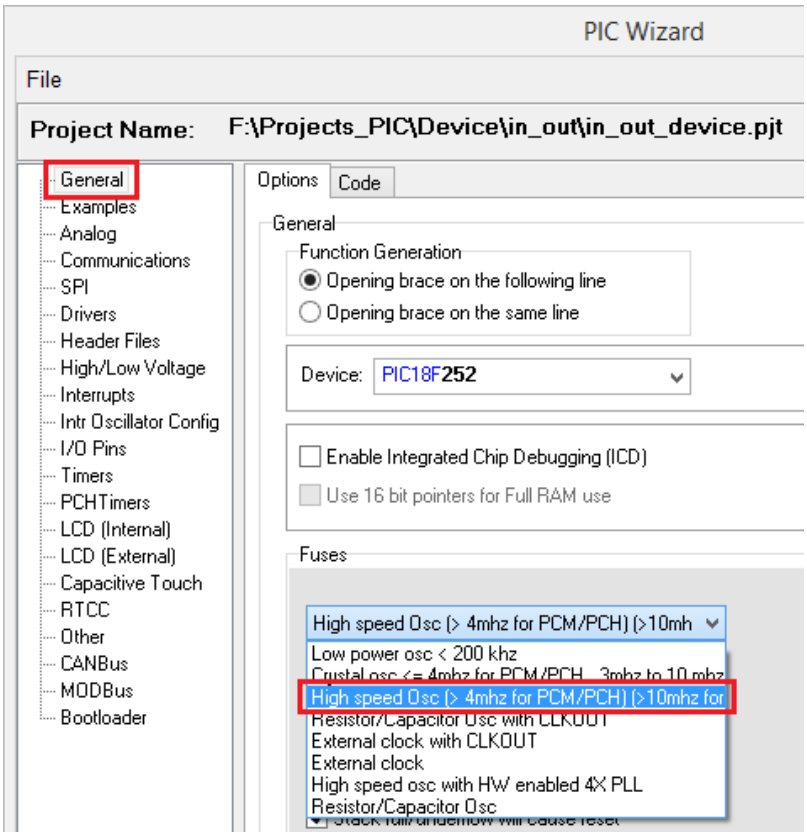


Рисунок 3.1.6 – Процес створення проекту у **IDE PCWH** за допомогою майстра **PIC Wizard**

У розділі **Communications** (рис. 3.1.7) налаштовуються параметри введення/виведення для інтерфейсів **RS-232** і **I²C** (швидкість обміну даними, відповідні лінії портів введення/виведення, перевірка помилок, режим **Master/Slave** і т.д.), а також активізується/відключається апаратний порт **PSP**.

6. У розділі **Communications** встановити мітку у полі **RS-232:**

- ☑ **Use RS-232**;

вказати:

- **Transmit:** **PIN C6** *(передача)*;
- **Receive:** **PIN C7** *(прийом)*.

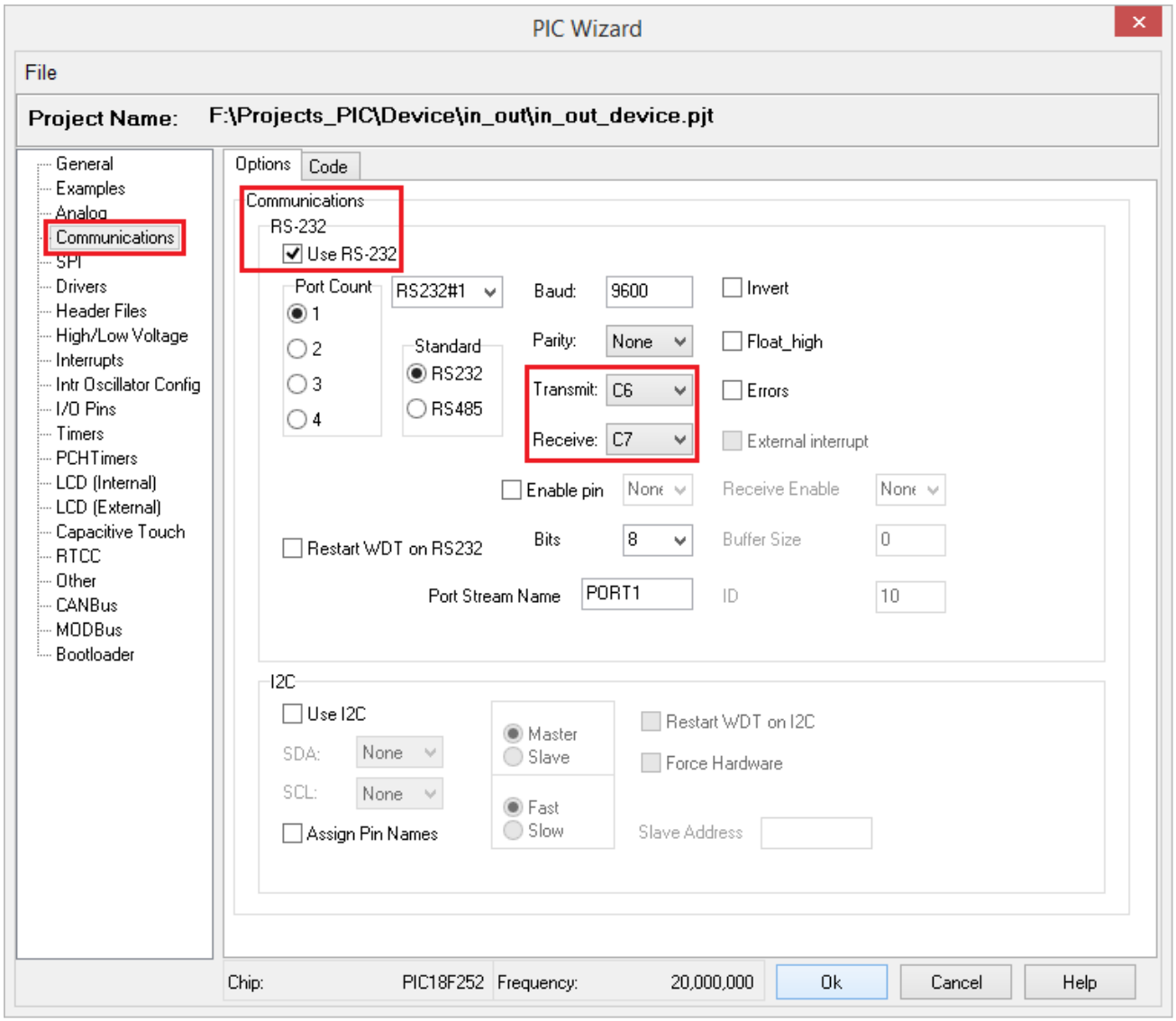


Рисунок 3.1.7 – Розділ **Communications** майстра **PIC Wizard**

Буде згенерований і доданий програмний код у вихідний файл відповідно до параметрів на поточній вкладці.

7. Для перегляду програмного коду, слід вибрати вкладку **Code** (рис. 3.1.8).

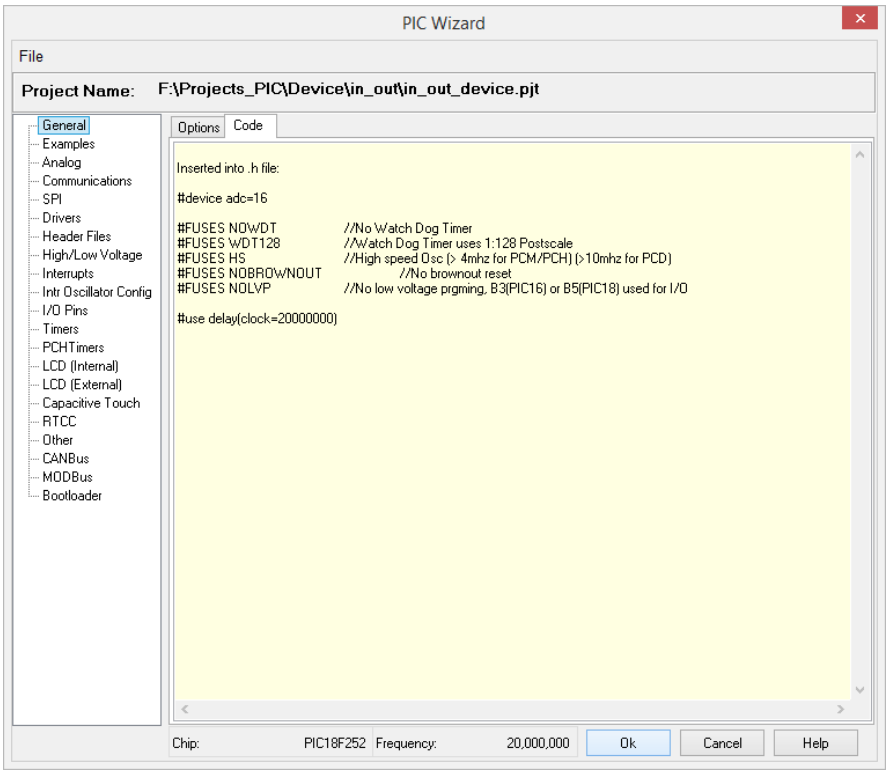


Рисунок 3.1.8 – Попередній перегляд коду, що генерується майстром **PIC Wizard** для поточного розділу **General**

8. Натиснути кнопку **OK**. Буде згенерований наступний код (рис. 3.1.9):

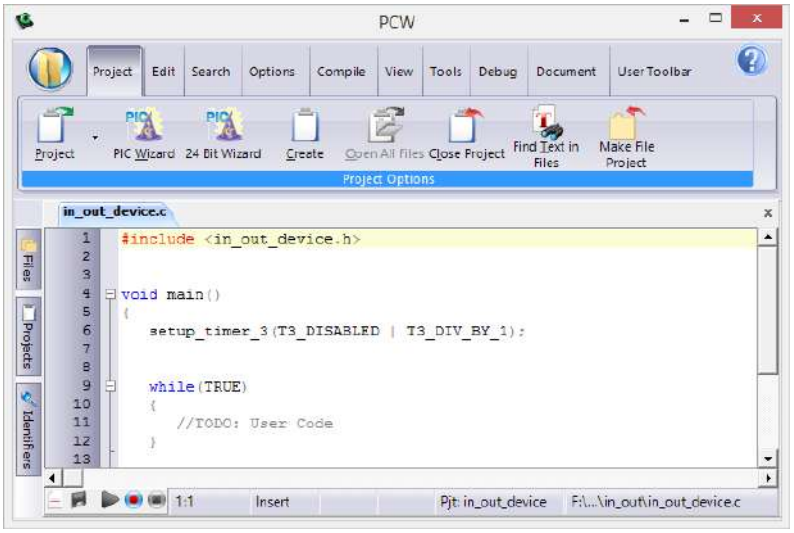


Рисунок 3.1.9 – Згенерований майстром **PIC Wizard** програмний код

Проект готовий, далі слід привести програму до наступного вигляду (лістинг 3.1.1) і використовувати як шаблон.

Можна приступати до написання програми.

Лістинг 3.1.1 – Шаблон програми

```
//#################################################################
// Найменування програми
// file: file_name.c
// Copyright (c) ПІБ / CNTU
//#################################################################

#include <in_out_device.h>
#include <swc_LCD.h>                //драйвер LCD дисплея
#CASE                               //враховувати регістр символів

//=================================================================
// Ініціалізація PIC
//=================================================================
void init()
{
       setup_timer_3(T3_DISABLED | T3_DIV_BY_1);
}

//=================================================================
// Функції програми
//=================================================================
void function_N()
{
       …
}
       …

//=================================================================
// Main
//=================================================================
void main()
{
       init();//ініціалізація мікроконтролера
       lcd_init();//ініціалізація LCD дисплея

       // Тут можна писати текст програми (виклики функцій)
}
```

Підключити драйвер LCD дисплея: `#include <swc_LCD.h>`

Пояснення до виконання лабораторного практикуму «IN_OUT»

Дискретними сигналами називаються сигнали, що змінюють свій рівень стрибком, без проміжних станів.

Активний рівень сигналу із кнопок і тумблерів **АПК** – логічний "0".

Активний рівень для запалювання світлодіоду – логічна "1".

Режим введення/виведення встановлюється функцією `set_tris_x(val)`.

Функції введення/виведення даних

Мікроконтролер може зчитувати дані із зовнішнього пристрою у порти і записувати дані у зовнішній пристрій через порт **A, B** і **C**.

Напрямок передачі даних визначається функціями:

```
set_tris_x(config_word);
```

У параметрі `config_word` встановити наступні значення для кожного розряду портів **A-C**:

- `0` – передача даних;
- `1` – прийом даних.

Приклад: потрібно порт B встановити у наступний режим:

- розряди 0-3 – читання даних;
- розряди 4-7 – виведення даних.

```
set_tris_b(0b00001111);//розташування розрядів: B7 B6 B5 B4 B3 B2 B1 B0
```

або:

```
set_tris_b(0x0F);      //розташування розрядів: B7 B6 B5 B4 B3 B2 B1 B0
```

Аналогічно для інших портів.

Читання даних з порту здійснюється функціями:

```
input_x();
```

Приклад: прочитати значення сигналів на вході порту A:

```
int8 val;
val = input_a();
```

Аналогічно для інших портів.

Виведення даних у порт здійснюється функцією:

```
output_x();
```

Приклад: вивести у порт **B** значення 0x55:

```
int8 val;
val = 0x55;
output_b(val);
```

Аналогічно для інших портів.

Операції з бітами портів здійснюється функціями:

```
output_high(PIN);
output_low(PIN);
```

Приклад: встановити розряди 0, 3 порту **B** у стан "1", а розряди 2, 4 порту **A** у стан "0":

```
output_high(PIN_B0);
output_high(PIN_B3);
output_low(PIN_A2);
output_low(PIN_A4);
```

Примітка:

- активний рівень кнопок і тумблерів **АПК** – логічний "0";
- активний рівень для роботи світлодіодів – логічна "1".

Виведення даних на дисплей

У процесі роботи програми необхідне виведення на дисплей налагоджувальної інформації і результату виконання програми. Інформація може бути виведена у вікно термінала на PC і на LCD дисплей **АПК** (4 рядка по 20 символів).

Команди виведення у вікно термінала стандартні:

```
putc();
puts();
printf();
```

Команди читання даних стандартні:

```
getc();
gets();
```

Команди для роботи з LCD дисплеєм:

```
void lcd_putc(int8 ch);                    //вивести символ
void printf();                             //вивести форматований рядок
void lcd_goto(int8 line,int8 pos);         //перейти до рядка і
                                           //знакомісця
void lcd_putc(int8 line, int8 pos, int8 ch); //вивести символ у
                                           //координатах
void lcd_clear_line(int8 line);            //почистити рядок
void lcd_clear();                          //почистити всі рядки
void lcd_cursor_on();                      //включити курсор
void lcd_cursor_off();                     //виключити курсор
void lcd_bs();                             //backspace
void lcd_cursor_left();                    //курсор вліво
void lcd_cursor_right();                   //курсор вправо
void lcd_cursor_up();                      //курсор нагору
void lcd_cursor_down();                    //курсор вниз
void lcd_running_string_start(); //запустити виведення рядка, що біжить
void lcd_running_string_stop();  //зупинити виведення рядка, що біжить
```

Приклади:

Виведення рядка, що біжить:

```
lcd_clear();
lcd_running_string_begin(4);       //початок повідомлення у рядку 4
printf("Програмне забезпечення управляючих мікро-ЕОМ");
lcd_running_string_end();          //кінець повідомлення
lcd_running_string_start();        //запустити рядок, що біжить
```

Виведення символу "Q" у другому рядку у п'ятому знакомісці:

```
lcd_putc(2,5,'Q');
```

Виведення повідомлення у другому рядку із третього знакомісця:

```
lcd_goto(2,3);
printf("Hello! How do You do?");
```

Виведення повідомлення на кілька рядків:

```
printf("1234567890\n 1234567890\n");
```

Повна принципова схема з'єднань для виконання всіх лабораторних практикумів представлена на рис. 3.1.10.

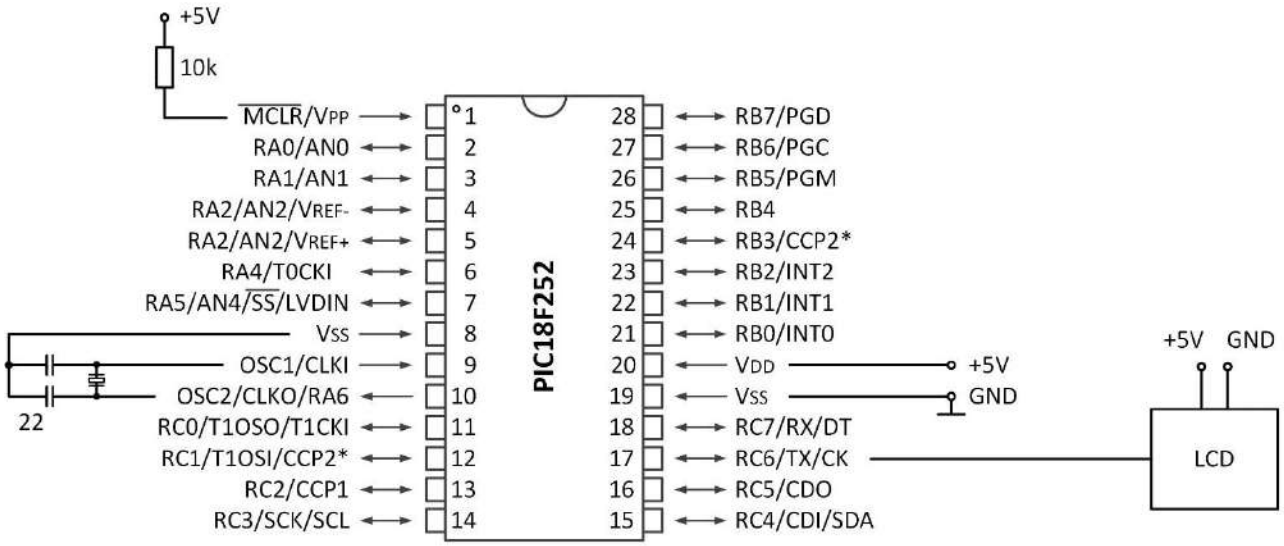


Рисунок 3.1.10 – Повна принципова схема з'єднань для виконання всіх лабораторних практикумів **«IN_OUT»**

Приклад комутації з'єднань **апаратно-програмного комплексу** представлений на рис. 3.1.11:

Рисунок 3.1.11 – Комутації з'єднань **АПК** для лабораторного практикуму **«IN_OUT»**

Приклади виконання лабораторного практикуму «IN_OUT»

Шаблон побудови та оформлення програми

Лістинг 3.1.2 – Шаблон побудови та оформлення програми

```
//#################################################################
// Найменування програми
// file: file_name.c
// Copyright (c) ПІБ студента / CNTU
//#################################################################

#include <in_out_device.h>
#include <swc_LCD.h>                //драйвер LCD дисплея
#CASE                               //враховувати регістр символів
//=================================================================
// Назва функції (Функція, що викликається з функції main())
//=================================================================
void init()
{
       …
}


//=================================================================
// Назва функції (Функція, що викликається з функції function_1())
//=================================================================
void function_1_1()
{
       …
}
//=================================================================
// Назва функції (Функція, що викликається з функції main())
//=================================================================
void function_1()
{
       …
       function_1_1();
       …
}
//=================================================================
// Назва функції (Функція, що викликається з функції main())
//=================================================================
void function_2()
{
       …
}
//=================================================================
// Main
//=================================================================
void main()
{
       init();                                  //ініціалізація
       …
       for(;;){
              function_1();                     //виклик функції
              function_2();                     //виклик функції
       }
}
```

Завдання для лабораторного практикуму «IN_OUT»

Програма має вигляд і структуру відповідно до лістингів 3.1.3, 3.1.4:

- лістинг 3.1.3 – для варіанту **АПК**;
- лістинг 3.1.4 – для варіанту **«Proteus»**.

Завдання для виконання лабораторного практикуму **«IN_OUT»** представлене у таблиці 3.1.2.

Таблиця 3.1.2 – Завдання для виконання лабораторного практикуму **«IN_OUT»**

Введення (кнопки)								**Виведення (світлодіоди)**							
A0	A1	A2	A3	B0	B1	B2	B3	B0	B1	B2	B3	B4	B5	B6	B7
	+	+												+	+

Алгоритм програми для лабораторного практикуму **«IN_OUT»** має вигляд (рис. 3.1.12):

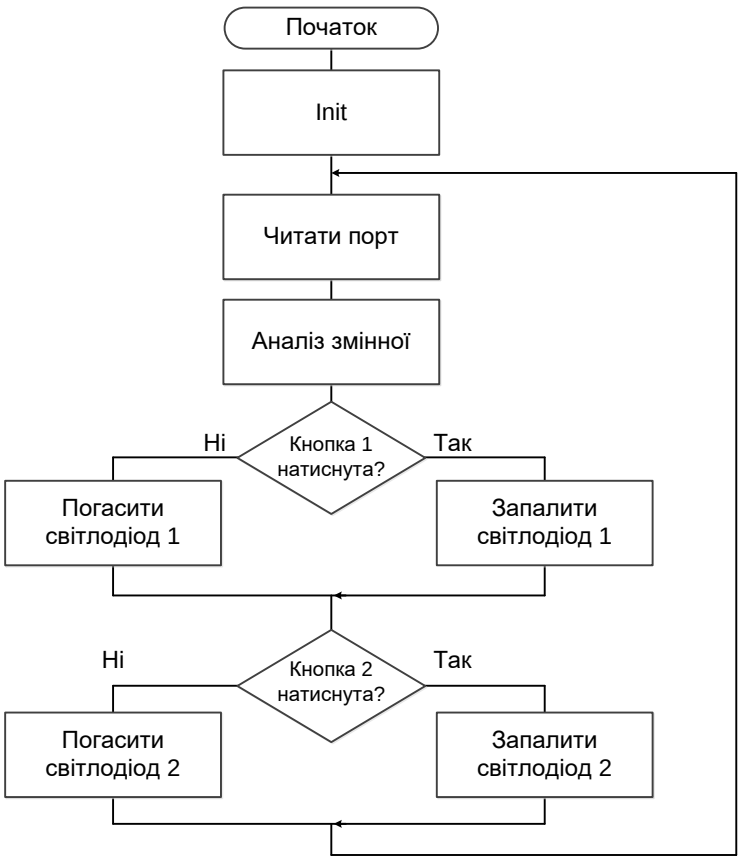


Рисунок 3.1.12 – Алгоритм програми лабораторного практикуму **«IN_OUT»**

Принципова схема підключень для варіанту АПК

Принципова схема підключень для варіанту **АПК**, для виконання лабораторного практикуму **«IN_OUT»** виглядає у такий спосіб (рис. 3.1.13):

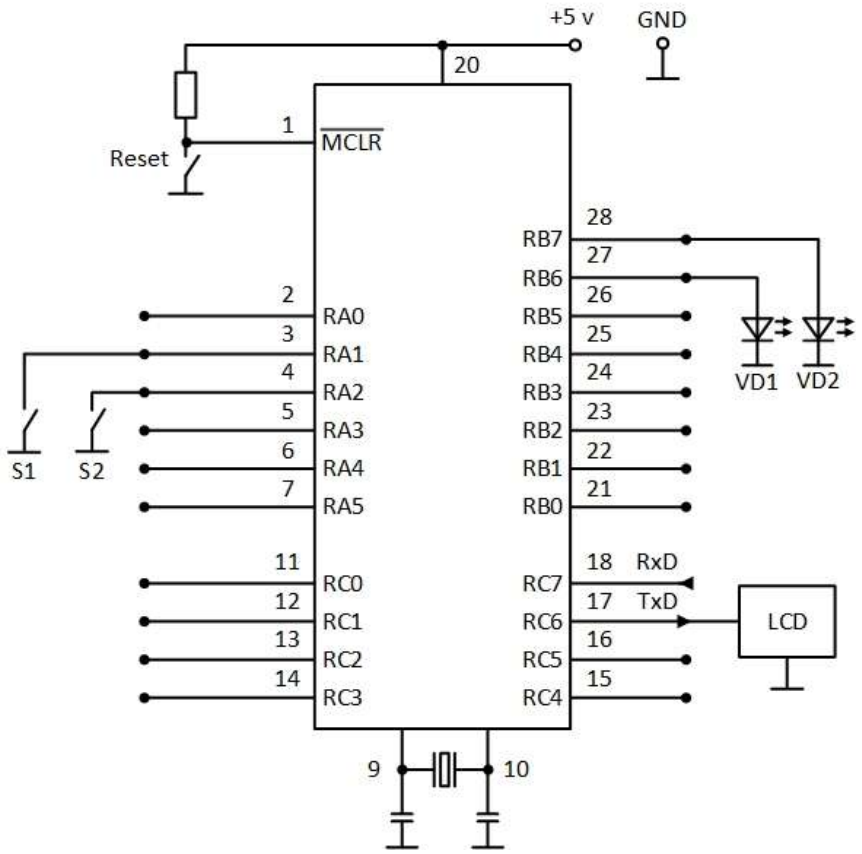


Рисунок 3.1.13 – Принципова схема підключень для лабораторного практикуму **«IN_OUT»** для варіанту **АПК**

Принципова схема підключень для варіанту «Proteus»

Принципова схема для варіанту **«Proteus»**, для виконання лабораторного практикуму **«IN_OUT»** виглядає у такий спосіб (рис. 3.1.14):

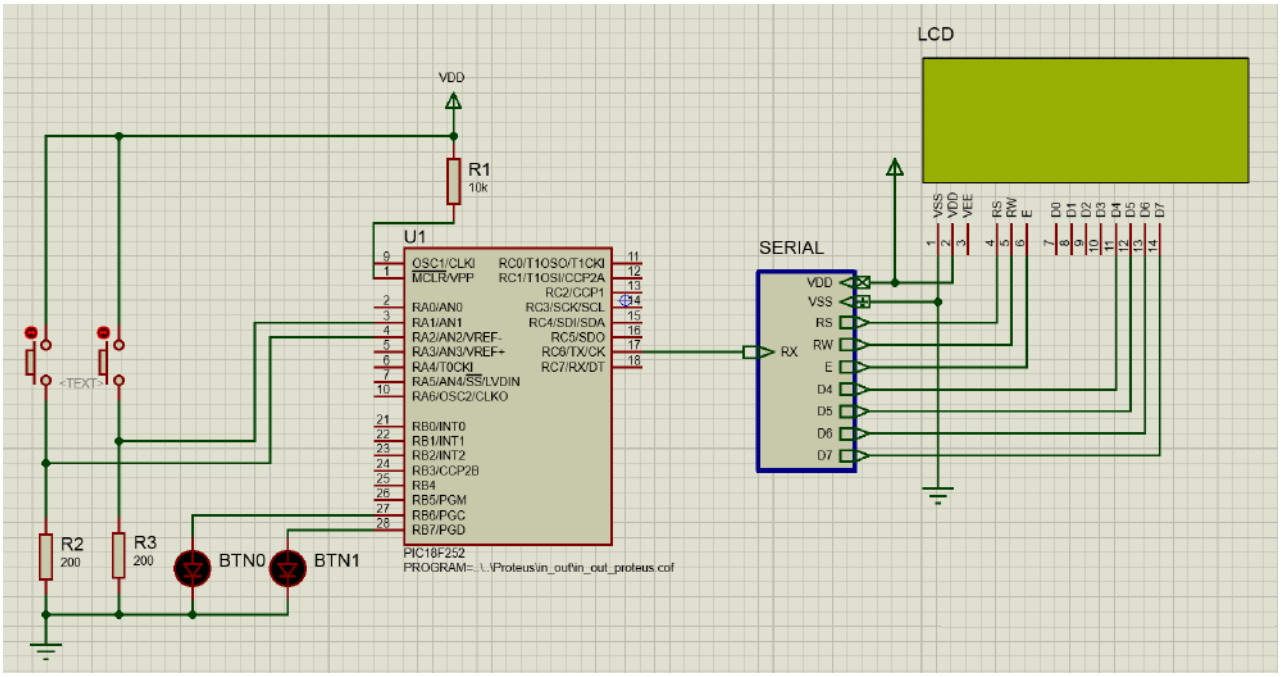


Рисунок 3.1.14 – Принципова схема підключень для лабораторного практикуму **«IN_OUT»** для варіанту **«Proteus»**

Результат виконання лабораторного практикуму «IN_OUT» для варіанту АПК

Лістинг 3.1.3 – Програма реалізації алгоритму (рис. 3.1.12) лабораторного практикуму **«IN_OUT»** для варіанту **АПК**

```
//###################################################################
// Лабораторний практикум «IN_OUT»
// file: in_out_device.c
// Copyright (c) Docent Smirnov V.V., Docent Smirnova N.V. / CNTU
//###################################################################

#include <in_out_device.h>
#include <swc_LCD.h>         //драйвер LCD дисплея
#CASE                        //враховувати регістр символів
//===================================================================
// Ініціалізація PIC
//===================================================================
void init()
{
     setup_timer_3(T3_DISABLED | T3_DIV_BY_1);
}

//===================================================================
// Запалити світлодіод 1
//===================================================================
void blink1()
{
     output_high(PIN_B6);
     delay_ms(50);
     output_low(PIN_B6);
     delay_ms(50);
```

```
}
//=============================================================================
// Запалити світлодіод 2
//=============================================================================
void blink2()
{
     output_high(PIN_B7);
     delay_ms(100);
     output_low(PIN_B7);
     delay_ms(100);
}
//=============================================================================
// Виведення заголовків
//=============================================================================
void titles()
{
     //====== виведення заголовка на дисплей
     lcd_goto(2,1);
     printf("Кафедра ПКСМ");
     lcd_goto(3,1);
     printf("Смірнов В.В.");
     lcd_goto(4,1);
     printf("Натиснута кнопка: ");

     //====== Виведення на дисплей рядка, що біжить
     lcd_running_string_begin(1);   // початок повідомлення у рядку 4
     printf("Лабораторний практикум IN_OUT");
     lcd_running_string_end();      // кінець повідомлення
     lcd_running_string_start();    // запустити біжучий рядок
}

//=============================================================================
// Введення/виведення сигналів
//=============================================================================
void in_out()
{
      int8 p;

      //=== читати порт у змінну
      p = input_a();
      //=== перевірити біт 1 на натискання клавіші (1)
      // варіант 1
      //test = bit_test(p,1);
      //if(test == 0)
      // варіант 2
      //if(!bit_test(p,1))

      // варіант 3
      if(bit_test(p,1) == 0)
      {
            //=== якщо 0 запалити світлодіод 1
            blink1();
            //=== виведення на дисплей
            lcd_goto(4,19);
            printf("A1");
      }else{
            //=== погасити світлодіод 1
            output_low(PIN_B6);
      }

      //=== перевірити біт 2 на натискання клавіші (0)
      // варіант 1
      //test = bit_test(p,2);
      // варіант 2
```

```
	//if(test == 0)

	// варіант 3
	if(!bit_test(p,2))
	{
		//=== якщо 0 запалити світлодіод 2
		blink2();
		//=== виведення на дисплей
		lcd_goto(4,19);
		printf("A2");
	}else{
		//=== погасити світлодіод 2
		output_low(PIN_B7);
	}
}
//=====================================================================
// Main
//=====================================================================
void main()
{
	init();				// ініціалізація мікроконтролера
	//====== конфігурація портів
	set_tris_a(0b11111111);
	set_tris_b(0b00111111);
	//====== затримка для lcd дисплея
	delay_ms(1000);

	//====== вивести заголовки
	titles();
	//====== головний цикл
	for(;;){
		in_out();
	}
}
```

Результат виконання лабораторного практикуму «IN_OUT» для варіанту «Proteus»

Лістинг 3.1.4 – Програма реалізації алгоритму (рис. 3.1.12) лабораторного практикуму **«IN_OUT»** для варіанту **«Proteus»**

```
//###################################################################
// Лабораторний практикум «IN_OUT»
// file: in_out_proteus.c
// Copyright (c) Docent Smirnov V.V., Docent Smirnova N.V. / CNTU
//###################################################################

#include <in_out_proteus.h>
#include <swc_LCD.h>			//драйвер LCD дисплея
#CASE					//враховувати регістр символів
//=====================================================================
// Ініціалізація PIC
//=====================================================================
void init()
{
	setup_timer_3(T3_DISABLED | T3_DIV_BY_1);
}
```

```
//=========================================================================
// Запалити світлодіод 1
//=========================================================================
void blink1()
{
       int i;
       output_high(PIN_B6);
       output_low(PIN_B7);
       delay_ms(300);
}

//=========================================================================
// Запалити світлодіод 2
//=========================================================================
void blink2()
{
       int i;
       output_high(PIN_B7);
       output_low(PIN_B6);
       delay_ms(300);
}
//=========================================================================
// Main
//=========================================================================
void main()
{
int8 p;
       init();
       //====== конфігурація портів
       set_tris_b(0b00111111);
       output_low(PIN_B6);
       output_low(PIN_B7);
       //====== головний цикл
       for(;;){
              lcd_goto(1,1);
              printf("IN_OUT");
              lcd_goto(2,1);
              printf("CS&NP Department");
              lcd_goto(3,1);
              printf("Docent Smirnov V.V.");

              //=== читати порт у змінну
              p = input_b();
              //=== перевірити біт 0 на натискання клавіші (0)
              if(bit_test(p,0))
              {
                     //=== виведення на дисплей
                     lcd_goto(4,1);
                     printf("Button: PIN B0");
                     //=== якщо 0 запалити світлодіод 1
                     blink1();
              }
              //=== перевірити біт 1 на натискання клавіші (0)
              if(bit_test(p,1))
              {
                     //=== виведення на дисплей
                     lcd_goto(4,1);
                     printf("Button: PIN B1");
                     //=== якщо 0 запалити світлодіод 2
                     blink2();
              }
              delay_ms(200);
       }
}
```

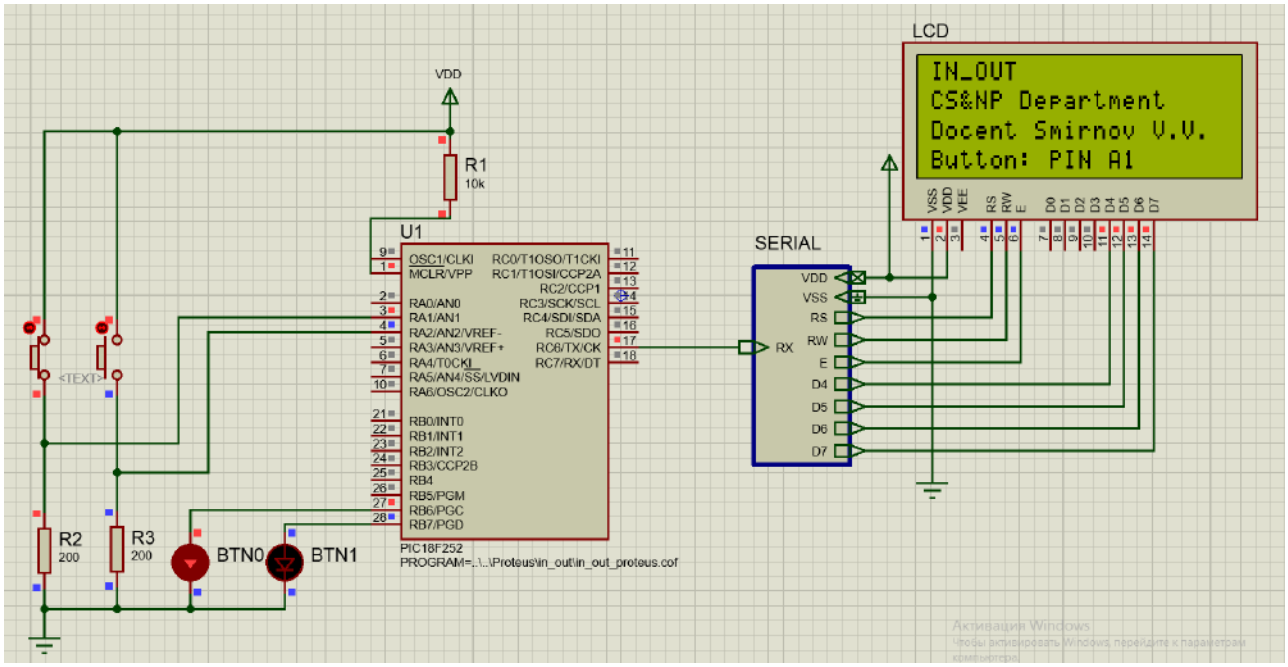


Рисунок 3.1.15 – Результат виконання лабораторного практикуму **«IN_OUT»** для варіанту **«Proteus»**

Контрольні питання

1) дати коротку характеристику мікроконтролера PIC18F252;
2) як управляти напрямком введення/виведення даних у мікроконтролері?
3) які команди здійснюють виведення інформації на LCD?
4) у яку пам'ять завантажується код програми?

ЛАБОРАТОРНИЙ ПРАКТИКУМ № 2 - «INTERRUPTS»

Тема: Робота з перериваннями мікроконтролера

Ціль роботи:

Отримання навичок роботи з перериваннями мікроконтролера, програмування обробників переривань мікроконтролерів, написання та налагодження програми обробників переривань порту RS-232, таймера, зовнішніх переривань.

Завдання лабораторного практикуму

- для варіанту **АПК** намалювати принципову схему підключень відповідно до варіанту для виконання лабораторного практикуму (таб. 3.2.1);
- для варіанту **«Proteus»** використовувати готову схему відповідно до варіанту для виконання лабораторного практикуму;
- створити алгоритм програми з обробниками переривань;
- здійснити виклик переривань із кнопок і тумблерів **АПК** та обробити їх відповідно до варіанту для виконання лабораторного практикуму;
- написати програму секундоміра в обробнику переривання таймера;
- здійснити приймання рядка символів з термінальної програми на PC по інтерфейсу RS-232, що містить ПІБ студента;
- здійснити виведення результатів на LCD і на світлодіоди з обробників переривань відповідно до варіанту для виконання лабораторного практикуму.

Виведення на LCD повинне містити:

- номер лабораторного практикуму;
- групу студента;
- прізвище та ініціали студента;
- секунди і хвилини в обробнику переривання таймера;

- номер зовнішнього переривання в обробнику зовнішнього переривання відповідно до варіанту для виконання лабораторного практикуму;
- рядок символів в обробнику переривання порту RS-232.

Варіанти для виконання лабораторного практикуму «INTERRUPTS»

Таблиця 3.2.1 – Варіанти для виконання лабораторного практикуму **«INTERRUPTS»**

№	Переривання				
	RTCC	RS-232	INT0	INT1	INT2
1	+	+	+	+	
2	+	+	+		+
3	+	+		+	+
4	+	+	+		+
5	+	+	+	+	
6	+	+	+		+
7	+	+		+	+
8	+	+	+		+
9	+	+	+	+	
10	+	+	+		+
11	+	+		+	+
12	+	+	+		+
13	+	+	+	+	
14	+	+	+		+
15	+	+		+	+

Послідовність виконання лабораторного практикуму для варіанту АПК

1) для варіанту **АПК** намалювати принципову схему підключень відповідно до варіанту для виконання лабораторного практикуму;
2) створити свій директорій для файлів проекту;
3) відкрити інтегроване середовище розробки **PCWH**;
4) створити проект у інтегрованому середовищі розробки **PCWH**;
5) створити алгоритм програми з обробниками переривань;

6) написати програму мовою програмування **C**, що реалізує створений алгоритм в обробнику переривань, відповідно до варіанту для виконання лабораторного практикуму;
7) скомпілювати програму, одержати бінарні файли з розширеннями ***.HEX** і ***.COF** (файли з розширеннями ***.HEX** і ***.COF** створюються підчас компіляції програми);
8) завантажити бінарний файл з розширенням ***.HEX** у пам'ять програм мікроконтролера програмою **PICkit.exe**;
9) на **АПК** зкумутувати схему підключень (кнопки і світлодіоди) відповідно до варіанту для виконання лабораторного практикуму.

Примітка: *Кнопками здійснювати виклик переривань INT0, INT1, INT2. Переривання порту RS-232 здійснювати шляхом посилки рядка символів з терміналу SIOW.exe або Terminal.exe;*

10) виконати програму.

Послідовність виконання лабораторного практикуму для варіанту «Proteus»

1) для середовища моделювання **«Proteus»** використовувати готову схему;
2) створити свій директорій для файлів проекту;
3) відкрити інтегроване середовище розробки **PCWH**;
4) створити проект у інтегрованому середовищі розробки **PCWH**;
5) створити алгоритм програми з обробниками переривань;
6) написати програму мовою програмування C, що реалізує створений алгоритм в обробнику переривань, відповідно до варіанту для виконання лабораторного практикуму;
7) скомпілювати програму, одержати бінарні файли з розширеннями ***.HEX** і ***.COF** (файли з розширеннями ***.HEX** і ***.COF** створюються підчас компіляції програми);

8) відкрити середовище моделювання **«Proteus»**;
9) у папці **«Proteus_students»** вибрати папку лабораторного практикуму **«INTERRUPTS»**;
10) відкрити файл проекту з розширенням ***.DSN**;
11) у середовищі моделювання **«Proteus»** змінити схему підключень відповідно до варіанту для виконання лабораторного практикуму;
12) завантажити заздалегідь скомпільований бінарний файл з розширеннями ***. COF** або ***. HEX** у мікроконтролер;
13) у середовищі моделювання **«Proteus»** виконати програму.

Послідовність виконання лабораторного практикуму для варіанту «Proteus» з використанням шаблону програми

1) для середовища моделювання **«Proteus»** використовувати готову схему;
2) відкрити папку **«Proteus_students»**;
3) відкрити інтегроване середовище розробки **PCWH**;
4) у папці **«Proteus_students»** вибрати папку лабораторного практикуму **«INTERRUPTS»**;
5) відкрити файл проекту з розширенням ***.pjt**;
6) у редакторі **IDE PCWH** відкрити шаблон файлу програми з розширенням ***.c**;
7) відкрити середовище моделювання **«Proteus»**;
8) у папці **«Proteus_students»** вибрати папку лабораторного практикуму **«INTERRUPTS»**;
9) відкрити файл проекту з розширенням ***.DSN**;
10) у середовищі моделювання **«Proteus»** змінити схему підключень відповідно до варіанту для виконання лабораторного практикуму;
11) створити алгоритм програми з обробниками переривань;
12) у інтегрованому середовищі розробки **PCWH**, використовуючи шаблон програми самостійно написати програму мовою

програмування **C**, що реалізує створений алгоритм в обробнику переривань, відповідно до варіанту для виконання лабораторного практикуму;

13) скомпілювати програму, одержати бінарні файли з розширеннями ***.HEX** і ***.COF** (файли з розширеннями ***.HEX** і ***.COF** створюються підчас компіляції програми);
14) у середовищі моделювання **«Proteus»** виконати програму.

Примітка: *файл демонстрації виконання лабораторного практикуму* ***«INTERRUPTS»*** *розташований на сайті* http://pksm.kntu.kr.ua/SCME.html.

Приклад створення проекту у інтегрованому середовищі розробки PCWH

Для виконання лабораторного практикуму **«INTERRUPTS»**, можна згенерувати новий проект автоматично за допомогою майстра **PIC Wizard**, який викликається по команді меню **Project › PIC Wizard**, або використовувати шаблон програми, що знаходиться у папці **«Proteus_students»** в якій вибрати папку лабораторного практикуму **«INTERRUPTS»**.

Створення проекту у інтегрованому середовищі розробки PCWH за допомогою майстра PIC Wizard

Для запуску майстра **PIC Wizard** слід виконати відповідну команду меню **Project**, а потім вказати розміщення і ім'я головного файлу проекту. В результаті відкриється вікно майстра, що складається з розділів з параметрами проекту (рис. 3.2.4).

Зовнішній вигляд вікна майстра може відрізнятися в залежності від версії компілятора **CCS-PICC** і обраного типу мікроконтролера.

1. Запустити **IDE PCWH**, натиснути кнопку **Projects** майстра **PIC Wizard** (рис. 3.2.1):

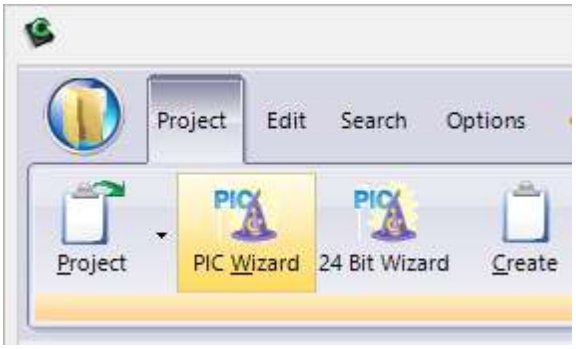


Рисунок 3.2.1 – Процес створення проекту у **IDE PCWH** за допомогою майстра **PIC Wizard**

або натиснути кнопку у вигляді відкритої папки 

2. Вибрати: **New › Project Wizard** (рис. 3.2.2):

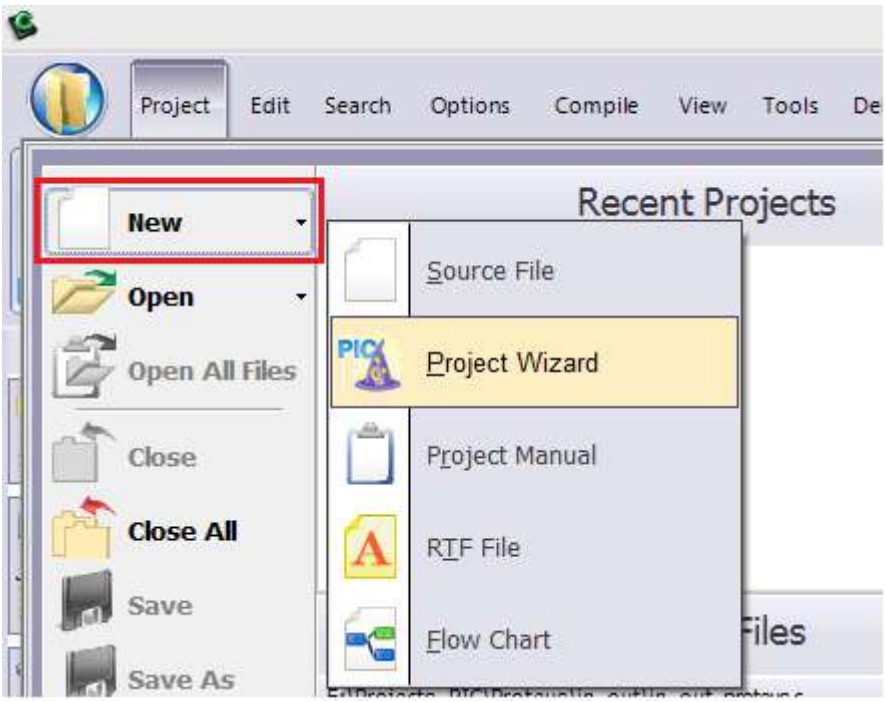


Рисунок 3.2.2 – Процес створення проекту у **IDE PCWH** за допомогою майстра **PIC Wizard**

3. Створити або вибрати свій директорій і записати у нього проект під будь-яким іменем латинським шрифтом, наприклад: «**interrupts.pjt**» (рис. 3.2.3):

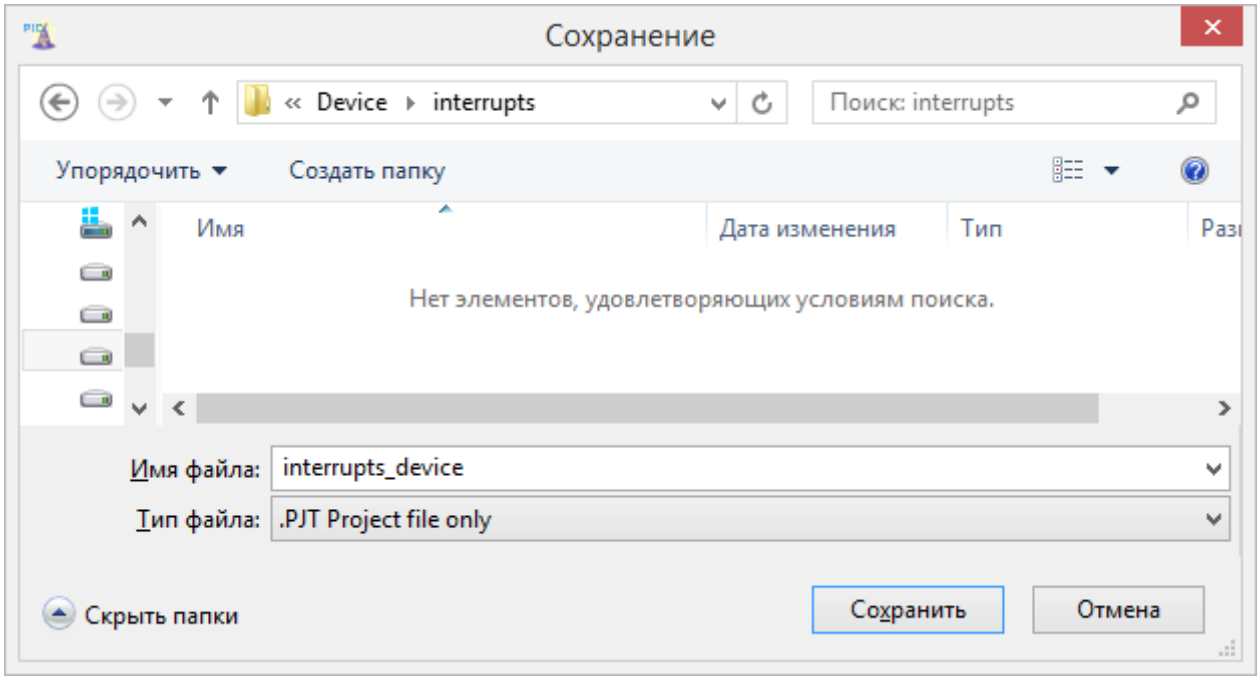


Рисунок 3.2.3 – Процес створення проекту у **IDE PCWH** за допомогою майстра **PIC Wizard**

У розділі **General** (рис. 3.2.4) вибирається цільовий мікроконтролер (список **Device**, що розкривається), його робоча частота (поле **Oscillator Frequency**), а також настроюються загальні параметри, на зразок розрядів запобігання, типу джерела системної синхронізації, порогової напруги для скидання та ін.

Для перегляду програмного коду, який буде згенерований і доданий у вихідний файл відповідно до параметрів на поточній вкладці, слід вибрати вкладку **Code**.

4. У розділі **General › Device** вибрати тип мікроконтролера, наприклад **PIC18F252**;

5. У розділі **General › Fuses** вибрати тип генератора **High speed Osc (> 4mhz for PCM/PCH) (>10mhz for PCD)** (рис. 3.2.4):

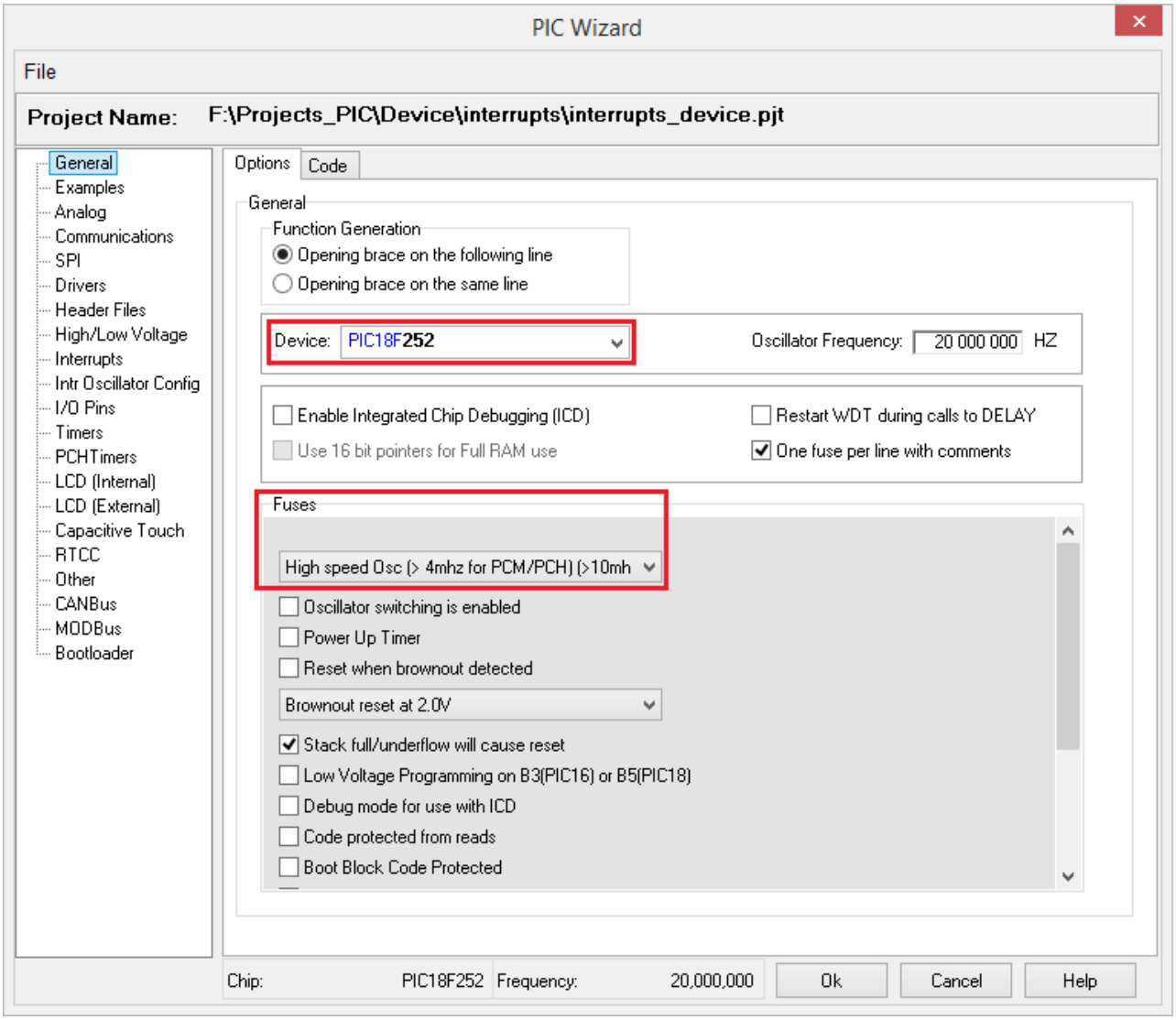


Рисунок 3.2.4 – Розділ **General** майстра **PIC Wizard**

У розділі **Communications** (рис. 3.2.5) налаштовуються параметри введення/виведення для інтерфейсів **RS-232** і **I²C** (швидкість обміну даними, відповідні лінії портів введення/виведення, перевірка помилок, режим **Master/Slave** і т.д.), а також активізується/відключається апаратний порт **PSP**.

6. У розділі **Communications** встановити мітку у полі **RS-232**:

- ☑ **Use RS-232**;

вказати:

- **Transmit:** **PIN C6** (передача);
- **Receive:** **PIN C7** (прийом).

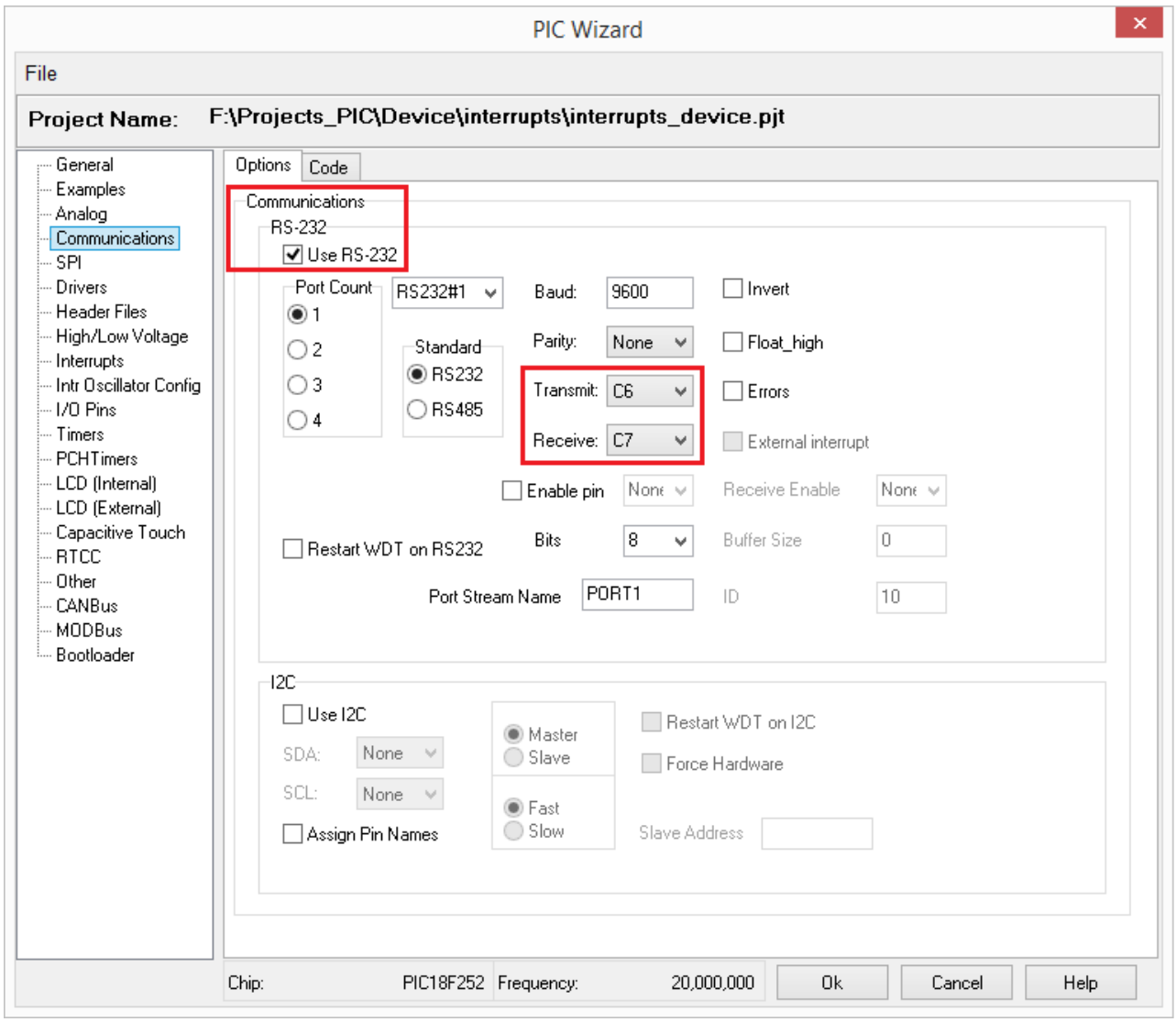


Рисунок 3.2.5 – Розділ **Communications** майстра **PIC Wizard**

Встановлення переривання мікроконтролера з IDE PCWH

Розділ **Interrupts** (рис. 3.2.6) дозволяє за допомогою набору прапорців дозволити або заборонити те чи інше переривання, доступне для вибраного пристрою.

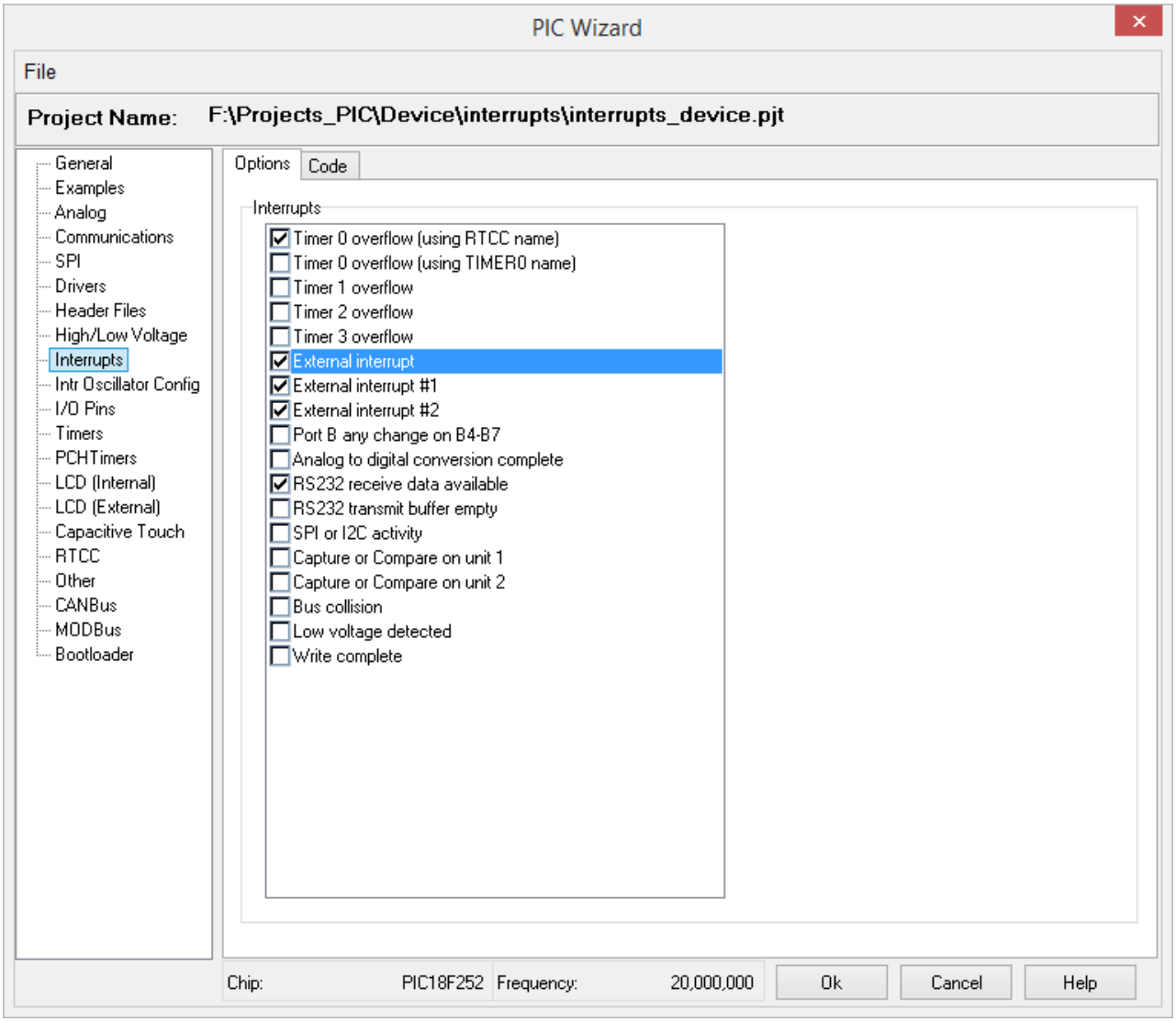


Рисунок 3.2.6 – Розділ **Interrupts** майстра **PIC Wizard**.

Для кожного обраного переривання у головну процедуру програми `main()` додається виклик відповідної підпрограми обробки переривання `enable_interrupts();`, а тіло самої підпрограми розміщується вище `main()` (лістінг 3.2.1).

Для роботи з перериваннями необхідно у інтегрованому середовищі розробки **PCWH** вказати компілятору, які переривання будуть використані у програмі (рис. 3.2.6).

7. У розділі **Interrupts › Options** вибрати переривання:
    1) Timer 0 overflow (using RTCC name);
    2) External interrupt;

3) External interrupt 1;
4) External interrupt 2;
5) RS-232 receive data available.

Встановлення переривань мікроконтролера з **IDE PCWH** представлене на рис. 3.2.6.

Після встановлення переривань мікроконтролера у розділі **Interrupts › Options** буде згенерований і доданий програмний код у вихідний файл відповідно до параметрів на поточній вкладці.

Для перегляду програмного коду, слід вибрати вкладку **Code** (рис. 3.2.7).

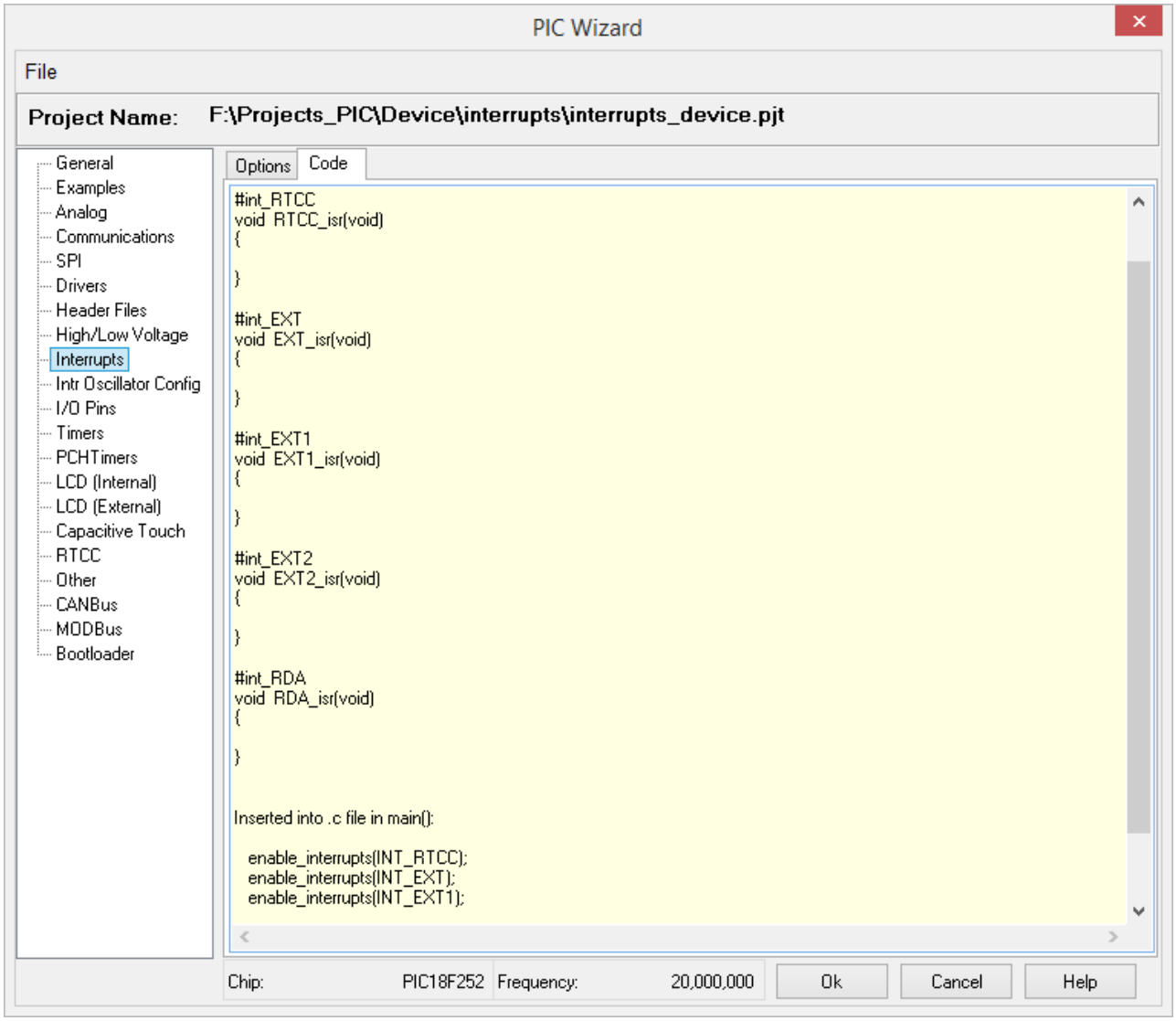


Рисунок 3.2.7 – Попередній перегляд коду, що генерується майстром **PIC Wizard** для розділу **Interrupts**

Встановлення таймера RTCC з IDE PCWH

8. У розділі **Timers** (рис. 3.2.8) налаштовуються параметри таймерів, включаючи сторожовий таймер.

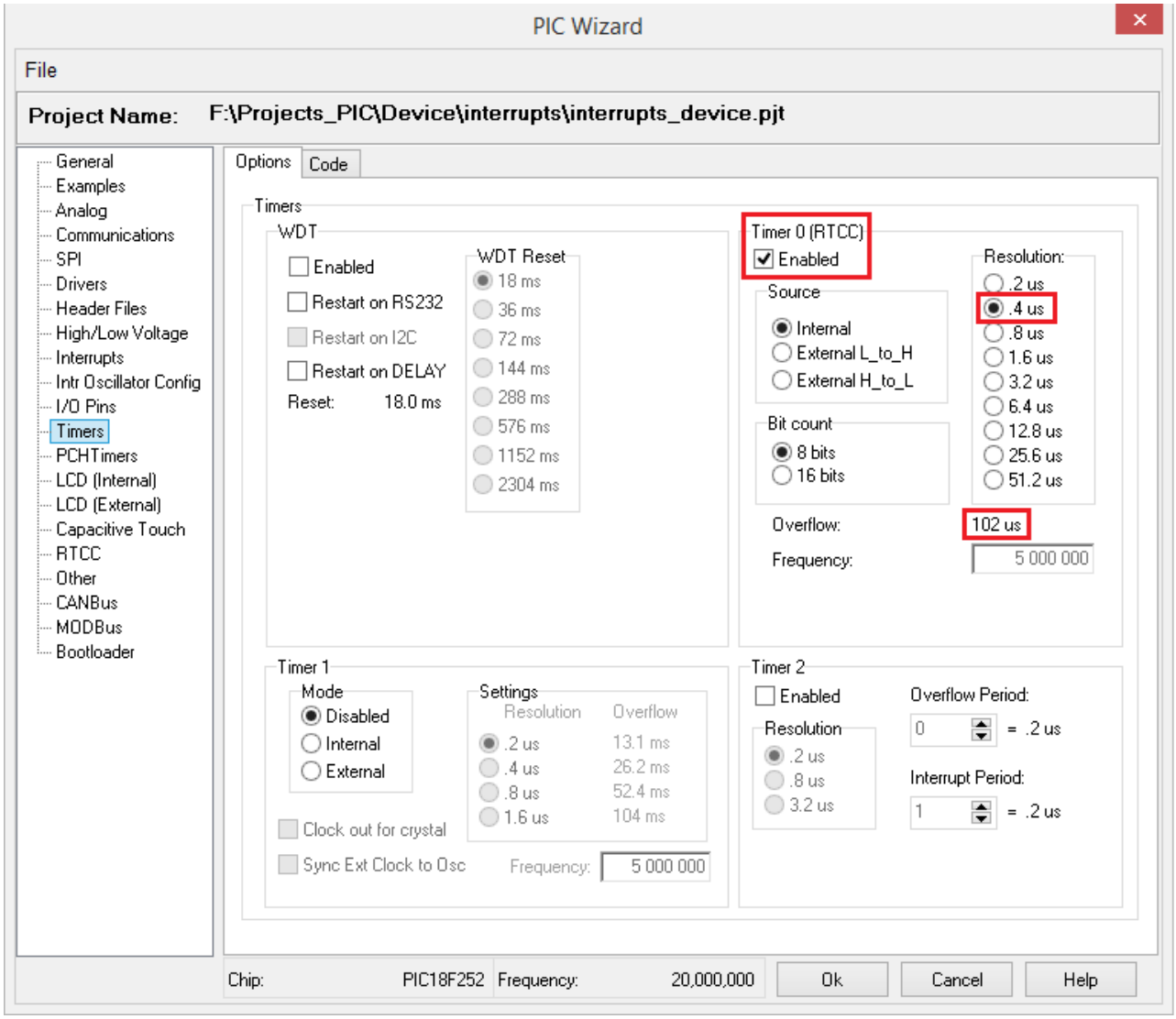


Рисунок 3.2.8 – Розділ **Timers** майстра **PIC Wizard**.

Значення деяких параметрів:

- **WDT Reset** – період між сигналами скидання від сторожового таймера;
- **Source** – вибір джерела тактування таймера TMR0: внутрішній або зовнішній (по наростаючому чи спадаючому фронті сигналу);
- **Frequency** – встановлення частоти у разі, коли вибране зовнішнє тактування таймера TMR0;

- **Resolution** – дозвіл таймера, що впливає на період між переповненнями лічільного регістру (для таймерів TMR0 і TMR1 обчислюється автоматично і відображається в полі **Overflow**);
- **Interrupt Period** – період між запитами на переривання від таймера TMR2.

Якщо обраний мікроконтролер надає додаткові таймери, їх параметри налаштовуються у розділі **PCH Timers** майстра **PIC Wizard**.

Встановлення переривань таймера **RTCC** з **IDE PCWH** представлене на рис. 3.2.8.

9. Натиснути кнопку **OK**.

Буде згенерований наступний код (рис. 3.2.9):

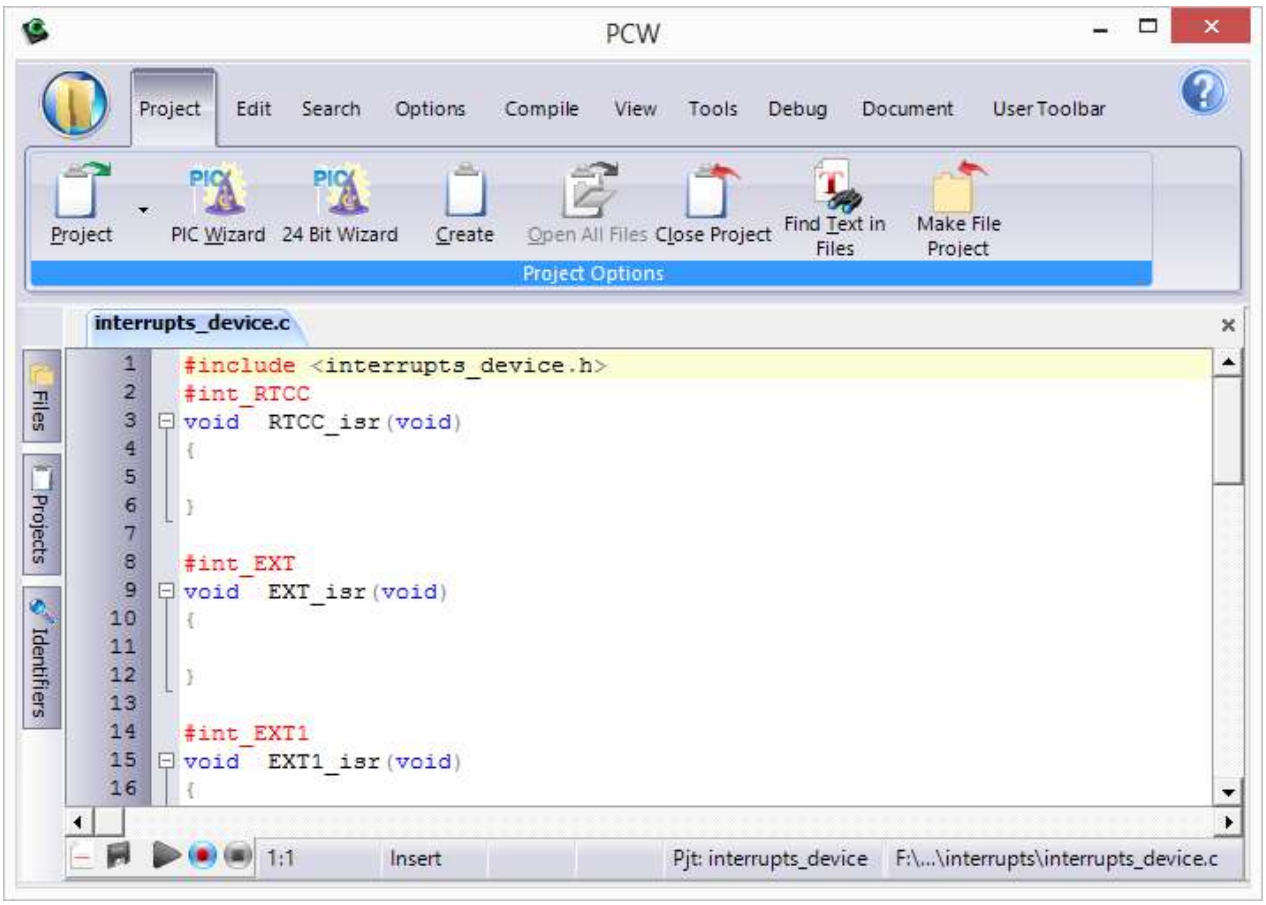


Рисунок 3.2.9 – Згенерований майстром **PIC Wizard** програмний код

Лістинг 3.2.1 – Приклад програмного коду, згенерованого майстром PIC Wizard

```
#include <interrupts_device.h>
#int_RTCC
void RTCC_isr(void)
{

}
#int_EXT
void EXT_isr(void)
{

}
#int_EXT1
void EXT1_isr(void)
{

}


#int_EXT2
void EXT2_isr(void)
{

}

#int_RDA
void RDA_isr(void)
{

}
void main()
{
       setup_timer_0(RTCC_INTERNAL|RTCC_DIV_2|RTCC_8_bit); //102 us
       overflow
       setup_timer_3(T3_DISABLED | T3_DIV_BY_1);
       enable_interrupts(INT_RTCC);
       enable_interrupts(INT_EXT);
       enable_interrupts(INT_EXT1);
       enable_interrupts(INT_EXT2);
       enable_interrupts(INT_RDA);
       enable_interrupts(GLOBAL);
       while(TRUE)
       {
              //TODO: User Code
       }
}
```

Проект готовий, можна приступати до написання програми.

Створення проекту у інтегрованому середовищі розробки PCWH за допомогою команди меню Project/Create

Новий проект можна створити вручну за допомогою команди меню **Project/Create** (рис. 3.2.10).

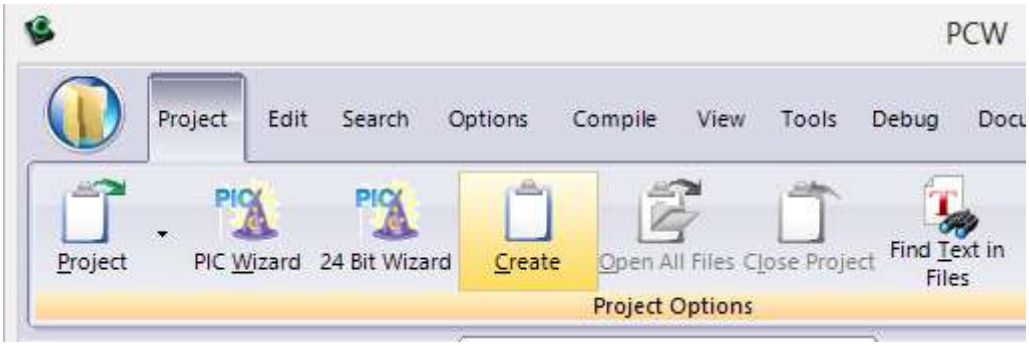


Рисунок 3.2.10 – Створення проекту у ручному режимі
за допомогою команди меню **Project / Create**

Якщо новий проект створений у ручному режимі, для роботи з перериванням таймера **RTCC** необхідно здійснити його ініціалізацію:

```
setup_timer_0(RTCC_INTERNAL|RTCC_DIV_8);
```

Заборонити використання інших таймерів можна директивами:

```
setup_timer_1(T1_DISABLED);
setup_timer_2(T2_DISABLED,0,1);
setup_timer_3(T3_DISABLED|T3_DIV_BY_1);
```

Для роботи з перериваннями їх необхідно дозволити:

```
enable_interrupts(INT_RTCC);        //дозволити переривання таймера RTCC
enable_interrupts(INT_EXT);         //дозволити зовнішнє переривання 0
enable_interrupts(INT_EXT1);        //дозволити зовнішнє переривання 1
enable_interrupts(INT_EXT2);        //дозволити зовнішнє переривання 2
enable_interrupts(INT_RDA);         //дозволити переривання порту RS-232
```

Далі необхідно дозволити всі зазначені переривання:

```
enable_interrupts(GLOBAL);          //дозволити всі зазначені переривання
```

Можна у довільному порядку заборонити переривання:

```
disable_interrupts(INT_RTCC);       //заборонити переривання таймера RTCC
disable_interrupts(INT_EXT);        //заборонити зовнішнє переривання 0
disable_interrupts(INT_EXT1);       //заборонити зовнішнє переривання 1
disable_interrupts(INT_EXT2);       //заборонити зовнішнє переривання 2
disable_interrupts(INT_RDA);        //заборонити переривання порту RS-232
disable_interrupts(GLOBAL);         //заборонити всі зазначені
                                    //переривання
```

Примітка: При роботі із зовнішніми перериваннями порту **B** мікроконтролера необхідно виводи порту **B** підключити до джерела живлення через резистори, тим самим забезпечивши наявність логічної "1" на виводах порту:

```
port_b_pullups(TRUE);        //входи порту B через резистори підключити
                             //до джерела 5 В
```

Розташування виводів зовнішніх переривань мікроконтролера представлене на рис. 3.2.11.

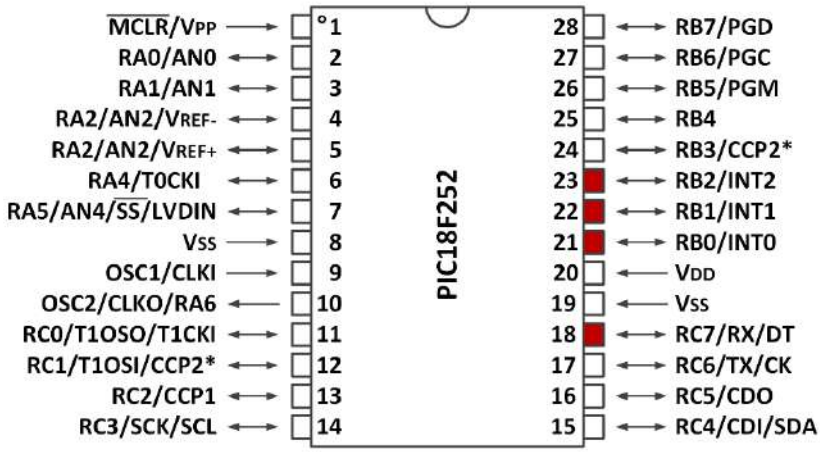


Рисунок 3.2.11 – Розташування виводів зовнішніх переривань мікроконтролера

Доступні переривання мікроконтролера **PIC18F252**:

- натиснути кнопку у меню **View › Valid Interrupts** (рис. 3.2.12 – 3.2.13):

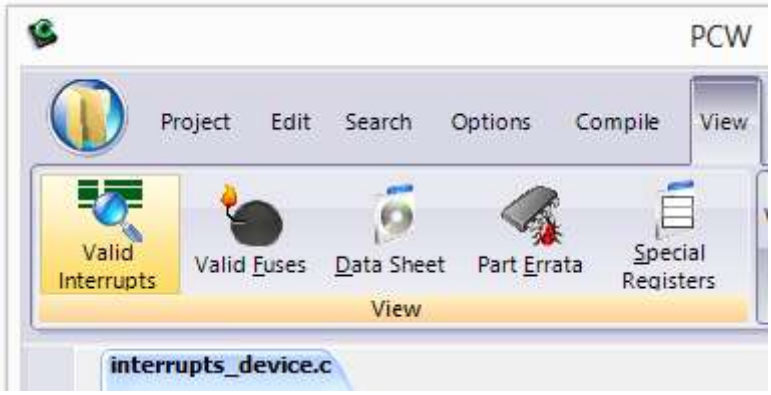


Рисунок 3.2.12 – Доступні переривання мікроконтролера **PIC18F252**

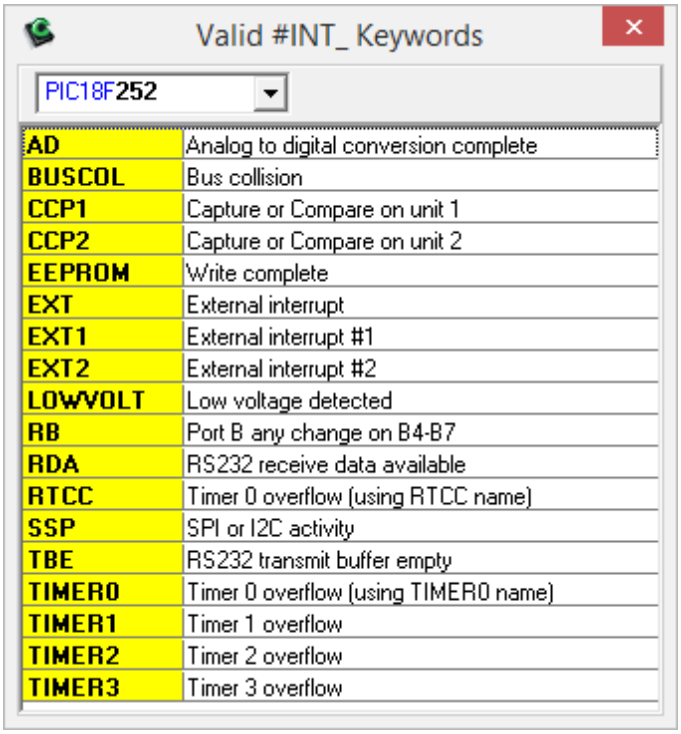


Рисунок 3.2.13 – Доступні переривання мікроконтролера **PIC18F252**

Пояснення до виконання лабораторного практикуму «IN_OUT»

Програмне забезпечення для передачі даних за допомогою послідовного інтерфейсу RS-232

Іноді потрібно для обміну даними підключати мікроконтролер до персонального комп'ютера (ПК). У такому випадку на персональному комп'ютері програмується графічний інтерфейс, за допомогою якого керують зовнішнім пристроєм, а підключають апаратні засоби до персонального комп'ютера за допомогою послідовного інтерфейсу.

Щоб передавати дані у мікроконтролер або приймати їх від нього за допомогою послідовного інтерфейсу RS-232, потрібне спеціальне програмне забезпечення на персональному комп'ютері, яке може відображати ці дані.

Після запуску програми потрібно вказувати кілька установок для нового підключення.

Всі зроблені установки можна змінювати також знову в більш пізній момент у відповідному меню.

Програма – термінал Terminal 1.9b

Програма – термінал **Terminal 1.9b** (рис. 3.2.14) є монітором COM порту персонального комп'ютера.

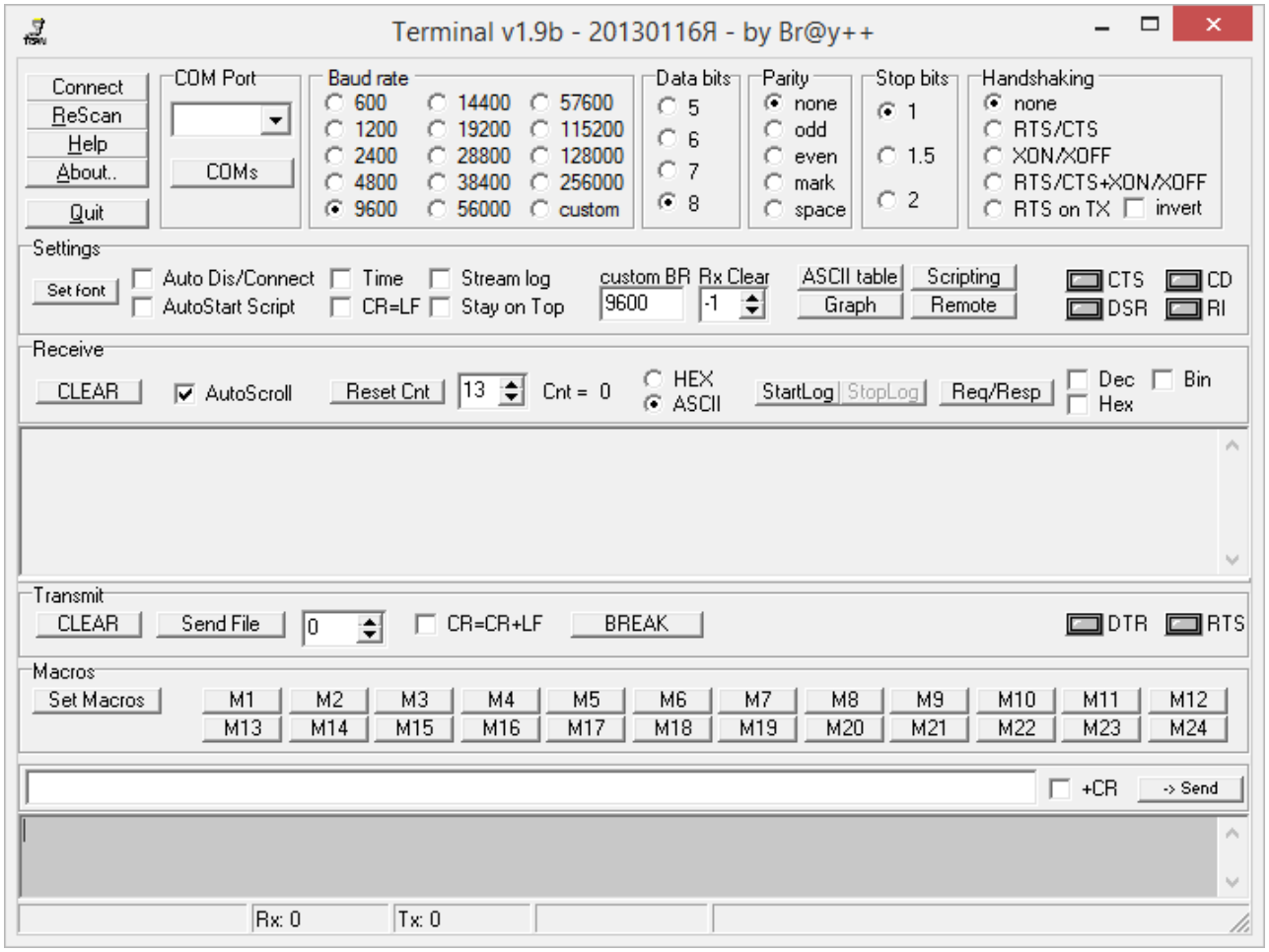


Рисунок 3.2.14 – Програма – термінал **Terminal 1.9b**

За допомогою програми можна легко відправляти і приймати дані через COM порт комп'ютера по протоколу RS-232. Серед переваг Terminal – гнучке налаштування програми під різні режими роботи. Інтерфейс програми простий і зрозумілий).

Основні можливості Terminal 1.9b:

- працює без установки. Вся програма – один exe-файл розміром близько 300Кб;
- є лічильник переданих і прийнятих байтів;
- можливість відправляти файли;

- крім стандартних швидкостей (baudrate) є можливість встановити свою нестандартну швидкість;
- підтримує до 64 COM-портів;
- можна весь лог роботи записувати у файл;
- можна призначити до 24 макросів;
- реалізовані Pascale-подібні скрипти.

Для роботи із програмою необхідно в першу чергу підключити пристрій, з яким збираєтеся працювати до комп'ютера через COM-порт. Підключіть живлення. Тепер запустіть програму - термінал **Terminal v1.9b**.

Інтерфейс і основні настройки підключення по порту

У верхньому полі знаходяться параметри підключення (рис. 3.2.15):

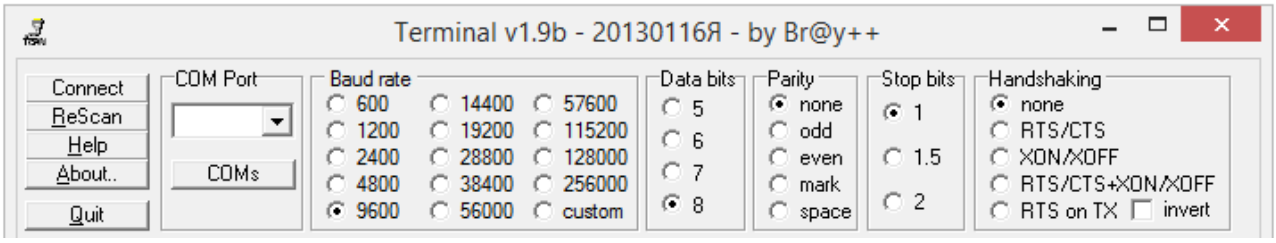


Рисунок 3.2.15 – Параметри підключення програми - терміналу **Terminal 1.9b**

Кнопка **Connect** – кнопка для відкриття COM-порту.

Кнопка **Rescan** – пересканувати список COM-портів.

Кнопка **Help** – довідка.

Кнопка **About ..** – про програму.

Кнопка **Quit** – вихід з програми.

Розділ **COM Port** – поле вибору номера COM-порту для підключення.

Розділ **Baud rate** – вибір швидкості COM-порту.

Розділ **Data bits** – вибір кількості біт даних.

Розділ **Parity** – вибір парності.

Розділ **Stop bits** – вибір кількості стопових біт.

Розділ **Handshaking** – вибір типу управління потоком.

У розділі **Settings** знаходяться додаткові параметри. Вони стануть в нагоді для написання скриптів, роботи з нестандартними швидкостями або для запису логу від пристрою.

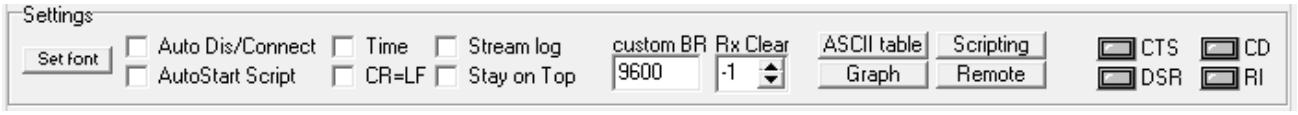


Рисунок 3.2.16 – Розділ **Setting**s

У розділі **Recieve** знаходяться параметри відображення відповіді від пристрою.

Рисунок 3.2.17 – Розділ **Recieve**

У розділі **Transmit** знаходяться параметри передачі даних на пристрій. Кнопки **DTR** і **RTS** встановлюють відповідні виводи у позитивний стан.

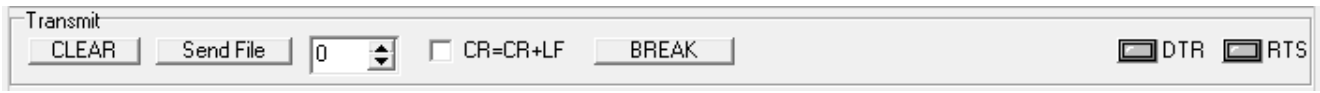


Рисунок 3.2.18 – Розділ **Transmit**

Макроси

Поле **Macros** призначене для створення користувацьких швидких клавіш.

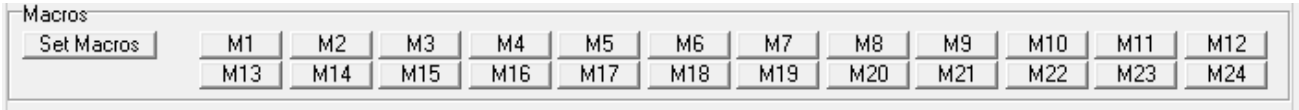


Рисунок 3.2.19 – Поле **Macros**

Для цього потрібно натиснути на кнопку **SetMacros** і у вікні, присвоїти кожній кнопці визначену послідовність символів, яка буде відправлятися на пристрій.

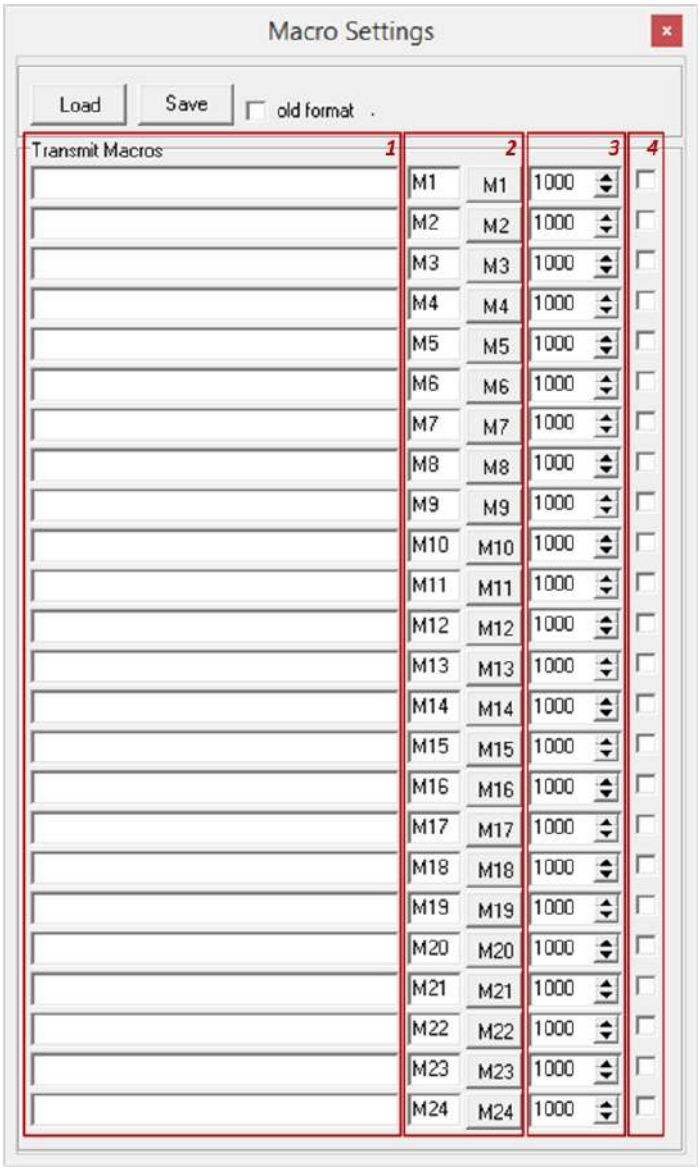


Рисунок 3.2.20 – Вікно **Macro Settings**

Блок №1:

Поле для введення послідовності символів для відправки.

Для того щоб відправити спеціальні символи, необхідно скористатися ASCII таблицею і ввести код символу, попередньо екранувати його знаком "$".

Блок №2:

У лівому полі задається ім'я кнопки, а в правому відображається сама кнопка.

Блок №3:

Задається затримка при автоматичному повторенні команд.

Блок №4:

Включення автоматичного повтору команди через інтервал часу.

Кнопки **Load** і **Save** дозволяють зберегти або завантажити файл з макросами, введеними в цьому вікні.

Відправлення та прийом даних

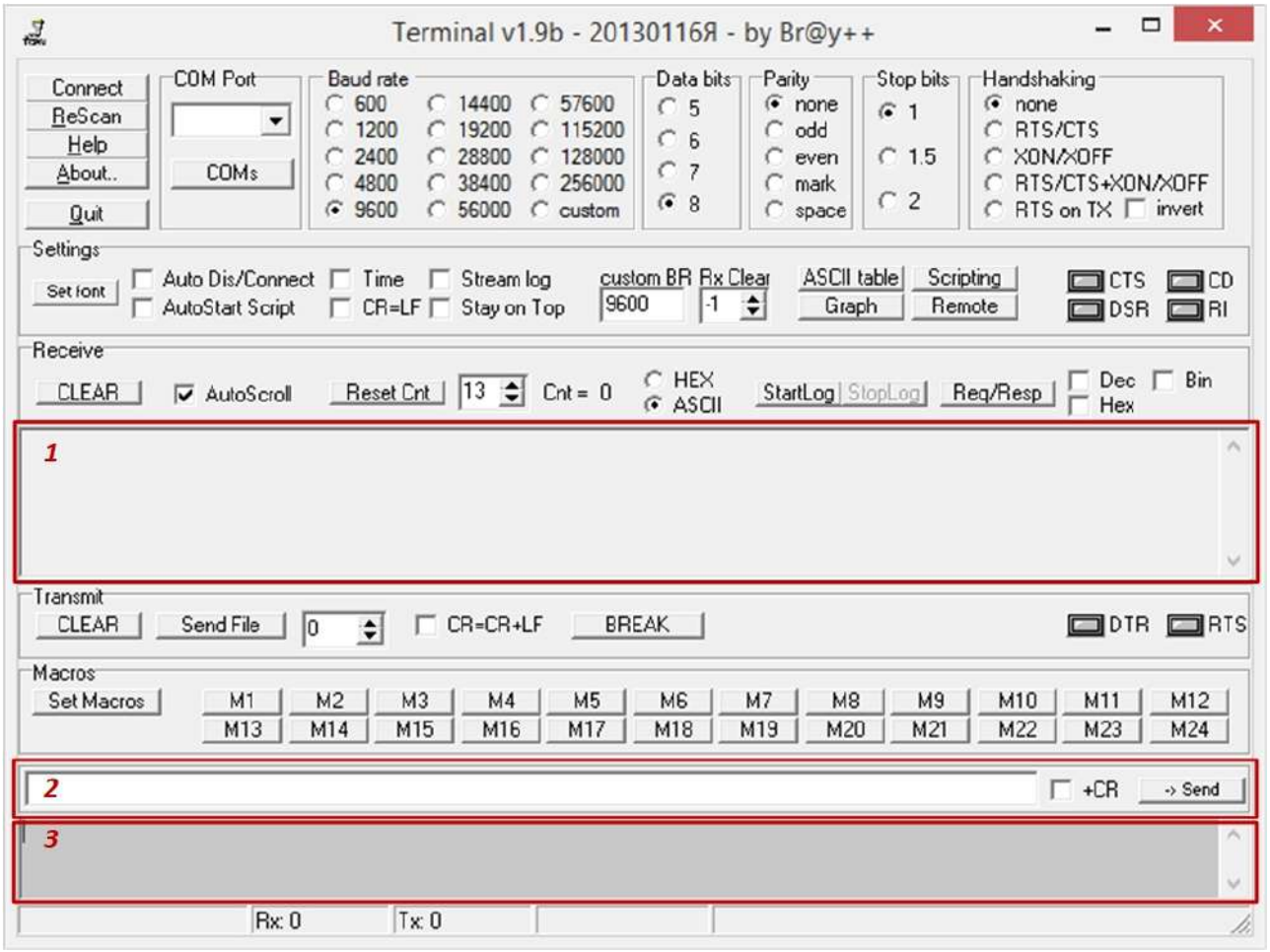


Рисунок 3.2.21 – Відправлення та прийом даних

Блок №1:

У цьому великому полі ви будете бачити відповіді від вашого пристрою.

Якщо у розділі «**Settings**» поставити галочку біля пункту «**Time**», то перед кожним рядком буде проставлена мітка часу. Це буває дуже корисно при аналізі логів з пристрою.

Блок №2:

Тут знаходиться поле для відправки тексту повідомлень.

Встановлена галочка біля пункту « + **CR** » буде дописувати до повідомленню, яке відправляється символ повернення каретки (емулювати натискання клавіші **Enter**).

Кнопка « → **Send** » відправить ваше повідомлення на пристрій.

Блок №3:

У самому низу знаходиться поле, де можна бачити відправлені на пристрій команди.

Програма – термінал SIOW.exe

Програму – термінал **SIOW.exe** можна запустити у директорії **PICC › SIOW.exe**. Вікно Програми - терміналу **SIOW.exe** представлено на рис. 3.2.22:

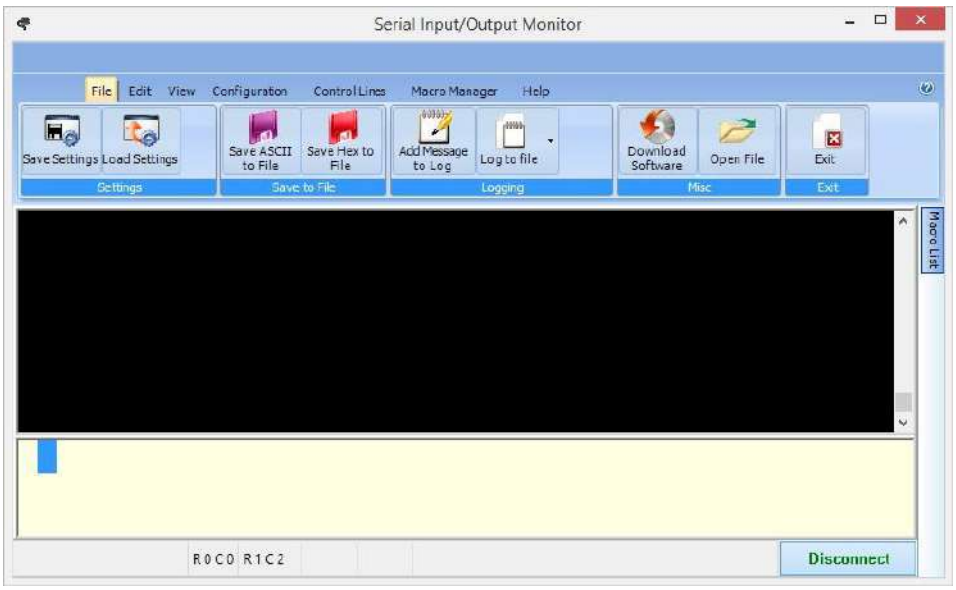


Рисунок 3.2.22 – Програма - термінал **SIOW.exe**

Для роботи із програмою необхідно вибрати порт і встановити параметри порту (рис. 3.2.23):

у меню **Configuration › Set Port Options**

вибрати: **ComPorts:** Direct to COM1

Baud rates: 9600

Натиснути кнопку **OK**

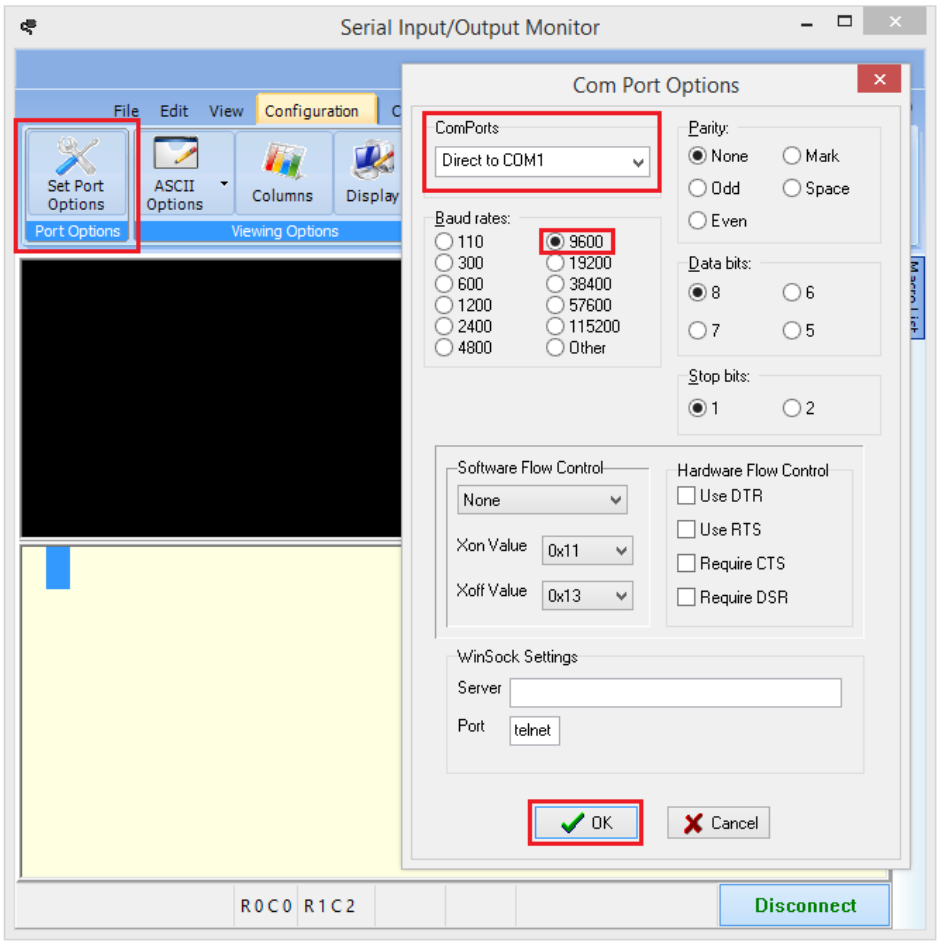


Рисунок 3.2.23 – Програма - термінал **SIOW.exe**

Потім у меню **File › Download Software** вибрати файл з розширенням ***.HEX**.

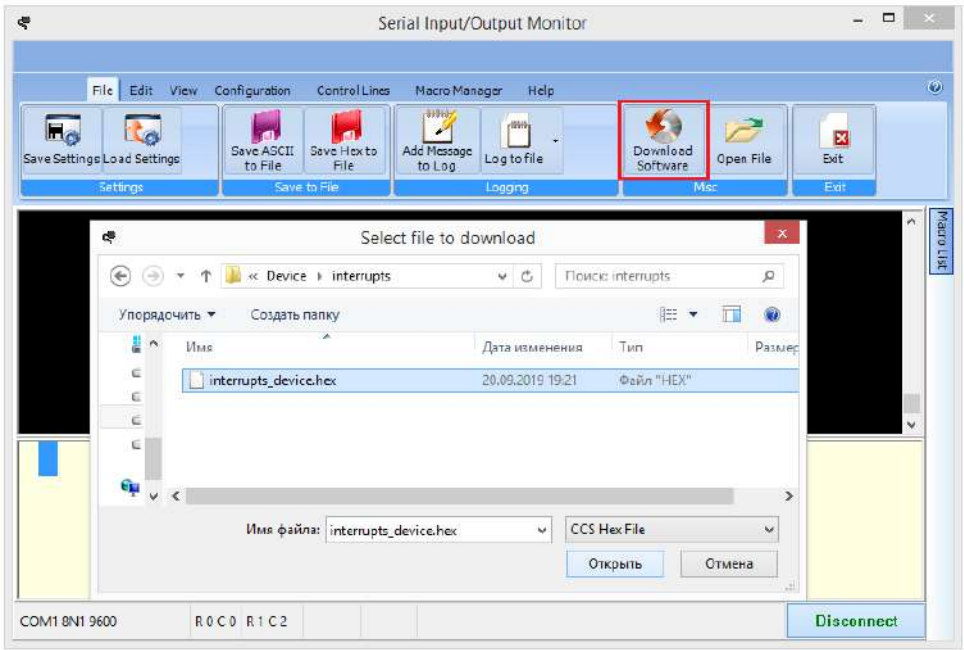


Рисунок 3.2.24 – Програма - термінал **SIOW.exe**

Примітка: у інтегрованому середовищі розробки **PCWH** в обробнику переривання порту **RS-232** необхідно очистити прийомний буфер, що б уникнути повторних переривань.

Очищення буфера здійснюється шляхом його читання функціями:

```
getc();
gets();
get_string();
```

Приклади виконання лабораторного практикуму «INTERRUPTS»

Завдання для лабораторного практикуму «INTERRUPTS»

Програма має вигляд і структуру відповідно до лістингів 3.2.2, 3.2.3:

- лістинг 3.2.2 – для варіанту **АПК**;
- лістинг 3.2.3 – для варіанту **«Proteus»**.

Завдання для виконання лабораторного практикуму **«INTERRUPTS»** представлене у таблиці 3.2.2.

Таблиця 3.2.2 – Завдання для виконання лабораторного практикуму **«INTERRUPTS»**

Переривання				**Виведення (світлодіоди)**		
RS-232	INT0	INT1	INT2	B5	B6	B7
+	+	+	+	+	+	+

Принципова схема підключень для варіанту АПК

Принципова схема підключень для варіанту **АПК**, для виконання лабораторного практикуму **«INTERRUPTS»** виглядає у такий спосіб (рис. 3.2.25):

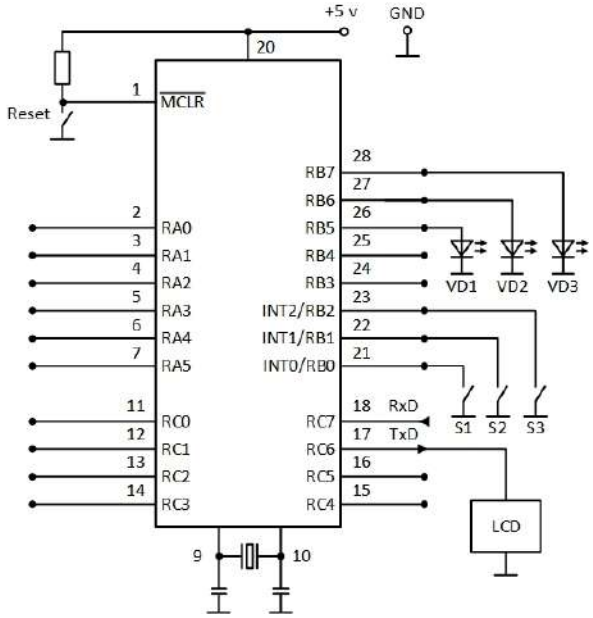


Рисунок 3.2.25 – Принципова схема підключень для лабораторного практикуму **«INTERRUPTS»** для варіанту **АПК**

Принципова схема підключень для варіанту «Proteus»

Принципова схема підключень для варіанту **«Proteus»**, для виконання лабораторного практикуму **«INTERRUPTS»** виглядає у такий спосіб (рис. 3.2.26):

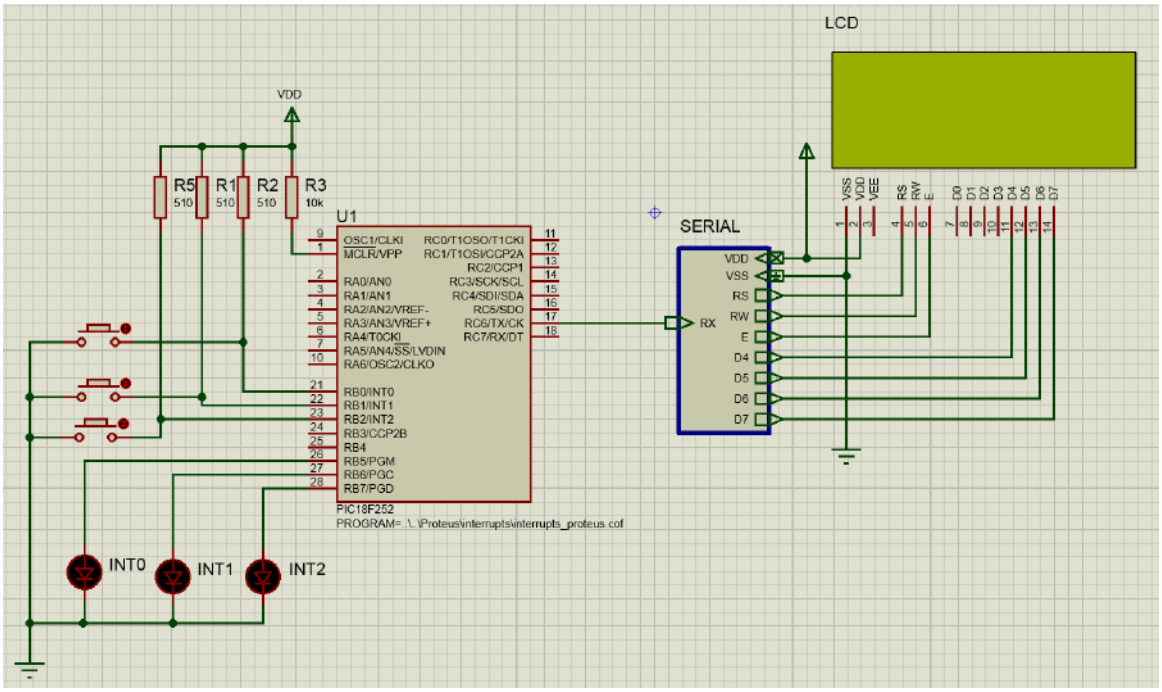


Рисунок 3.2.26 – Принципова схема підключень для лабораторного практикуму **«INTERRUPTS»** для варіанту **«Proteus»**

Результат виконання лабораторного практикуму «INTERRUPTS» для варіанту АПК

Лістинг 3.2.2 – Програма реалізації алгоритму лабораторного практикуму **«INTERRUPTS»** для варіанту **АПК**

```
//###################################################################
// Лабораторний практикум «INTERRUPTS»
// file: interrupts_device.c / переривання
// Copyright (c) Docent Smirnov V.V., Docent Smirnova N.V. / CNTU
//###################################################################

#include < interrupts _device.h>
#include <swc_LIB.h>
#include <swc_LCD.h>                //драйвер LCD дисплея
#CASE                               //враховувати регістр символів

//=== час
int16 t_ms = 0;
char t_s = 0;
char t_m = 0;

//===========================================================================
// Ініціалізація PIC
//===========================================================================
void init()
{
       setup_timer_0(RTCC_INTERNAL|RTCC_DIV_8);
       setup_timer_3(T3_DISABLED | T3_DIV_BY_1);
       enable_interrupts(INT_RTCC);
       enable_interrupts(INT_EXT);
       enable_interrupts(INT_EXT1);
       enable_interrupts(INT_EXT2);
       enable_interrupts(INT_RDA);
       enable_interrupts(GLOBAL);
       //входи порту B через резистори підключити до 5 В
       port_b_pullups(TRUE);
       enable_interrupts(GLOBAL);
       setup_oscillator(FALSE);
       set_timer0(65035);
}

//===========================================================================
// Відображення часу
//===========================================================================
void time_show()
{
       //=== секунди
       lcd_goto(1,1);
       printf("Секунди:");
       //=== секунди стерти
       lcd_goto(1,10);
       printf(" ");
       //=== секунди відобразити
       lcd_goto(1,10);
       printf("%u",t_s);     //%u – десяткове число цілого типу без знаку
       //=== хвилини
       lcd_goto(2,1);
       printf("Хвилини:");
```

```
        //=== хвилини стерти
        lcd_goto(2,10);
        printf(" ");
        //=== хвилини відобразити
        lcd_goto(2,10);
        printf("%u",t_m);
        lcd_goto(4,1);
        printf("Смірнов В.В., ПКСМ");
}

//=============================================================================
// Відлік часу
//=============================================================================
void time()
{
        //====== додати ms
        t_ms += 100;
        //====== перевірити ms на переповнення
        if(t_ms >= 1000){
                t_s ++;
                //t_ms -= 1000;
                t_ms = 0;
                time_show();
        }

        //====== перевірити секунди на переповнення
        if(t_s == 59){
                t_m ++;
                t_s =0;
        }
}

//=============================================================================
// Переривання таймера
//=============================================================================
#int_RTCC
void RTCC_isr(void)
{
        time();
}

//=============================================================================
// Зовнішнє переривання 0
//=============================================================================
#int_EXT
void EXT_isr(void)
{
        int i;
        lcd_goto(3,1);
        printf("Переривання 0 ");
        for(i=0; i<2; i++)
        {
                output_high(PIN_B5);
                delay_ms(200);
                output_low(PIN_B5);
                delay_ms(200);
        }
}
```

```
//=============================================================================
// Зовнішнє переривання 1
//=============================================================================
#int_EXT1
void EXT1_isr(void)
{
	int i;
	lcd_goto(3,1);
	printf("Переривання 1 ");
	for(i=0; i<2; i++)
	{
		output_high(PIN_B6);
		delay_ms(200);
		output_low(PIN_B6);
		delay_ms(200);
	}
}

//=============================================================================
// Зовнішнє переривання 2
//=============================================================================
#int_EXT2
void EXT2_isr(void)
{
	int i;
	lcd_goto(3,1);
	printf("Переривання 2 ");
	for(i=0; i<2; i++)
	{
		output_high(PIN_B7);
		delay_ms(200);
		output_low(PIN_B7);
		delay_ms(200);
	}
}

//=============================================================================
// Переривання RS-232
//=============================================================================
#int_RDA
void RDA_isr(void)
{
	get_string();
	lcd_goto(4,1);
	printf(INPUT_STRING);
}

//=============================================================================
// Main
//=============================================================================
void main()
{
	init();
	//====== конфігурація портів
	set_tris_b(0b00011111);
	lcd_init();
	delay_ms(1000);
	//====== головний цикл
	for(;;)
	{
		//нічого не робити, чекати переривання
	}
}
```

Результат виконання лабораторного практикуму «INTERRUPTS» для варіанту «Proteus»

Лістинг 3.2.3 – Програма реалізації алгоритму лабораторного практикуму **«INTERRUPTS»** для варіанту **«Proteus»**

```
//###################################################################
// Лабораторний практикум «INTERRUPTS»
// file: interrupts_ proteus.c / переривання
// Copyright (c) Docent Smirnov V.V., Docent Smirnova N.V. / CNTU
//###################################################################

#include < interrupts _device.h>
#include <interrupts_proteus.h>
#include <swc_LCD.h>                        //драйвер LCD дисплея
#include <swc_LIB.h>
#CASE                                       //враховувати регістр символів

//=== час
int16 t_ms = 0;
char t_s = 0;
char t_m = 0;

//===================================================================
// Ініціалізація PIC
//===================================================================
void init()
{
       setup_timer_0(RTCC_INTERNAL|RTCC_DIV_8);
       setup_timer_1(T1_DISABLED);
       setup_timer_2(T2_DISABLED,0,1);
       setup_timer_3(T3_DISABLED|T3_DIV_BY_1);
       enable_interrupts(INT_RTCC);
       enable_interrupts(INT_EXT);
       enable_interrupts(INT_EXT1);
       enable_interrupts(INT_EXT2);
       enable_interrupts(INT_RDA);
       enable_interrupts(GLOBAL);
       port_b_pullups(TRUE);        //входи порту B через резистори
                                    //підключити до 5 В
}

//===================================================================
// Відображення часу
//===================================================================
void time_show()
{
       lcd_goto(1,1);
       printf("CS&NP Department");
       lcd_goto(2,1);
       printf("Docent Smirnov V.V.");
       lcd_goto(3,1);       //%u – десяткове число цілого типу без знаку
       printf("Time: %u:%u",t_m,t_s);
}
```

```
//=========================================================================
// Відлік часу
//=========================================================================
void time()
{
	//====== додати ms
	t_ms += 100;
	//====== перевірити ms на переповнення
	if(t_ms >= 1000){
		t_s ++;
		//t_ms -= 1000;
		t_ms = 0;
		time_show();
	}
	//====== перевірити секунди на переповнення
		if(t_s == 59){
		t_m ++;
		t_s =0;
	}
}

//=========================================================================
// Перериванням таймера RTCC
//=========================================================================
#int_RTCC
void RTCC_isr(void)
{
	time();
}

//=========================================================================
// Зовнішнє переривання 0
//=========================================================================
#int_EXT
void EXT_isr(void)
{
	int i;
	lcd_goto(4,1);
	printf("Interrupt 0");
	for(i=0; i<5; i++)
	{
		output_high(PIN_B7);
		delay_ms(200);
		output_low(PIN_B7);
		delay_ms(200);
	}
}
//=========================================================================
// Зовнішнє переривання 1
//=========================================================================
#int_EXT1
void EXT1_isr(void)
{
	int i;
	lcd_goto(4,1);
	printf("Interrupt 1 ");
	for(i=0; i<5; i++)
	{
		output_high(PIN_B6);
		delay_ms(200);
		output_low(PIN_B6);
		delay_ms(200);
	}
}
```

```
//=========================================================================
// Зовнішнє переривання 2
//=========================================================================
#int_EXT2
void EXT2_isr(void)
{
       int i;
       lcd_goto(4,1);
       printf("Interrupt 2 ");
       for(i=0; i<5; i++)
       {
              output_high(PIN_B5);
              delay_ms(200);
              output_low(PIN_B5);
              delay_ms(200);
       }
}

//=========================================================================
// Зовнішнє переривання 0
//=========================================================================
#int_RDA
void RDA_isr(void)
{
       get_string();
       lcd_goto(2,1);
       printf(INPUT_STRING);
}
//=========================================================================
// Main
//=========================================================================
void main()
{
       init();
       //====== конфігурація портів
       set_tris_b(0b00011111);
       lcd_init();
       delay_ms(1000);
       //====== головний цикл
       for(;;)
       {
              //нічого не робити, чекати переривання
       }
}
```

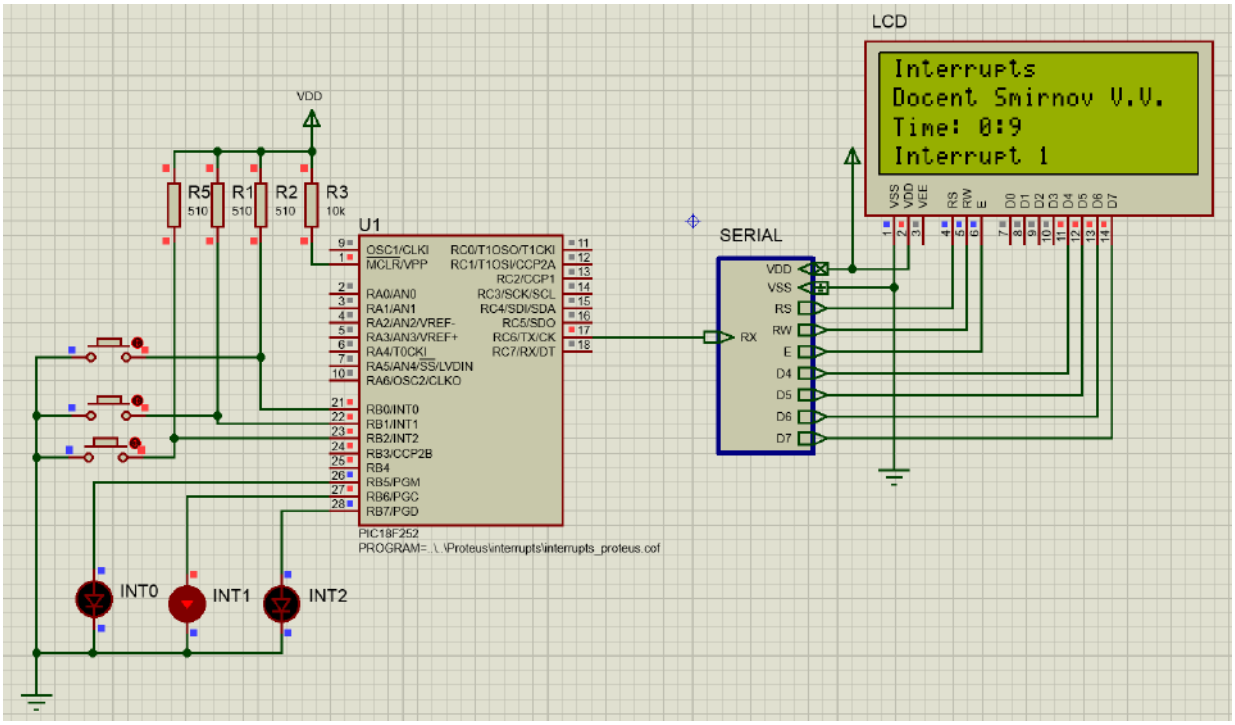


Рисунок 3.2.27 – Результат виконання лабораторного практикуму **«INTERRUPTS»** для варіанту **«Proteus»**

Контрольні питання

1) які умови і причини виникнення внутрішніх і зовнішніх переривань мікроконтролера **PIC18F252** (типи переривань)?
2) як управляти (дозволяти, забороняти) перериваннями у мікроконтролері?
3) чому у перериванні порту RS-232 необхідно очистити прийомний буфер?
4) якими по відношенню до мікроконтролера бувають переривання?
5) навіщо потрібні переривання у системах керування?

ЛАБОРАТОРНИЙ ПРАКТИКУМ № 3 - «ADC»

Тема: Робота з АЦП

Ціль роботи:

Отримання навичок роботи з **АЦП** мікроконтролера і написання програм обробки оцифрованих аналогових сигналів.

Завдання лабораторного практикуму

- для варіанту **АПК** намалювати принципову схему підключень відповідно до варіанту для виконання лабораторного практикуму (таб. 3.3.1);
- для варіанту **«Proteus»** використовувати готову схему відповідно до варіанту для виконання лабораторного практикуму;
- створити алгоритм програми роботи з АЦП для перетворення рівня напруги з потенціометра і клавіатури;
- здійснити зчитування рівня напруги з потенціометра, оцифрувати за допомогою АЦП і перетворити в десятковий формат відповідно до варіанту для виконання лабораторного практикуму;
- відобразити рівень поточної напруги у вольтах на LCD. Показання напруги на LCD повинно відрізнятися від показань вольтметра не більше, ніж на 0,01 В;
- написати програму визначення натиснутої кнопки на клавіатурі;
- здійснити виведення результатів на LCD відповідно до варіанту для виконання лабораторного практикуму.

Виведення на LCD повинне містити:

1) зчитані дані з АЦП:
- у HEX форматі;
- у десятковому форматі;
- у вольтах;

2) символ натиснутої клавіші;
3) прізвище та ініціали студента.

Варіанти для виконання лабораторного практикуму «ADC»

Таблиця 3.3.1 – Варіанти для виконання лабораторного практикуму **«ADC»**

	Канал АЦП								**Розрядність АЦП**	
	Потенціометр				**Клавіатура**					
№	**A0**	**A1**	**A2**	**A3**	**A0**	**A1**	**A2**	**A3**	**8**	**10**
1	+					+			+	
2		+					+			+
3			+					+	+	
4				+	+					+
5				+		+			+	
6			+			+				+
7	+							+	+	
8	+						+			+
9		+						+	+	
10			+		+					+
11				+	+				+	
12	+					+				+
13		+			+				+	
14			+					+		+
15				+			+		+	

Послідовність виконання лабораторного практикуму для варіанту АПК

1) для варіанту **АПК** намалювати принципову схему підключень відповідно до варіанту для виконання лабораторного практикуму;
2) створити свій директорій для файлів проекту;
3) відкрити інтегроване середовище розробки **PCWH**;
4) створити проект у інтегрованому середовищі розробки **PCWH**;
5) створити алгоритм програми роботи з АЦП для перетворення рівня напруги з потенціометра і клавіатури;

6) написати програму мовою програмування **С**, що реалізує створений алгоритм, відповідно до варіанту для виконання лабораторного практикуму;
7) скомпілювати програму, одержати бінарні файли з розширеннями ***.HEX** і ***.COF** (файли з розширеннями ***.HEX** і ***.COF** створюються підчас компіляції програми); завантажити бінарний файл з розширенням ***.HEX** у пам'ять програм мікроконтролера програмою **PICkit.exe**;
8) на **АПК** зкумутувати схему підключень (підключити потенціометр, клавіатуру і вольтметр до АЦП відповідно до рис. 3.3.1, рис. 3.3.2) відповідно до варіанту для виконання лабораторного практикуму.

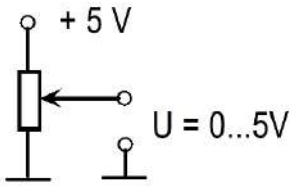


Рисунок 3.3.1 – Принципова схема потенціометра

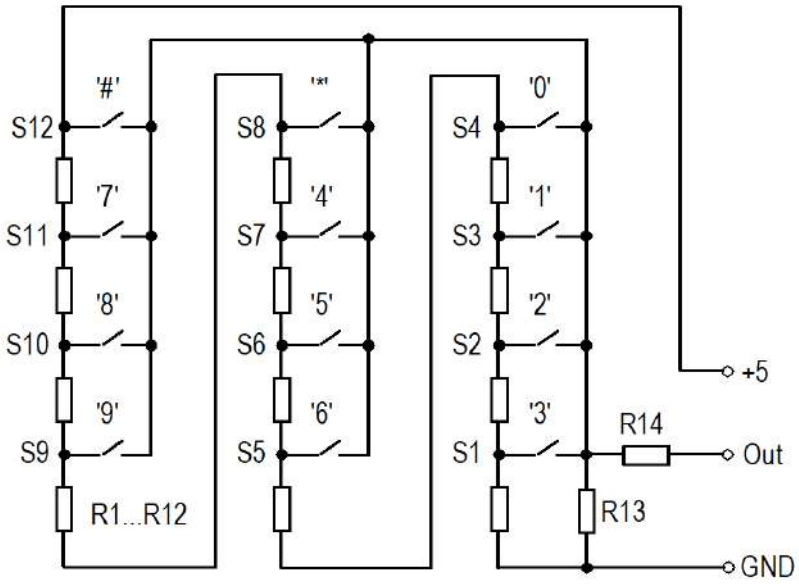


Рисунок 3.3.2 – Принципова схема клавіатури

9) виконати програму.

Послідовність виконання лабораторного практикуму для варіанту «Proteus»

1) для середовища моделювання **«Proteus»** використовувати готову схему;
2) створити свій директорій для файлів проекту;
3) відкрити інтегроване середовище розробки **PCWH**;
4) створити проект у інтегрованому середовищі розробки **PCWH**;
5) створити алгоритм програми;
6) написати програму мовою програмування **C**, що реалізує створений алгоритм, відповідно до варіанту для виконання лабораторного практикуму;
7) скомпілювати програму, одержати бінарні файли з розширеннями ***.HEX** і ***.COF** (файли з розширеннями ***.HEX** і ***.COF** створюються підчас компіляції програми);
8) відкрити середовище моделювання **«Proteus»**;
9) у папці **«Proteus_students»** вибрати папку лабораторного практикуму **«ADC»**;
10) відкрити файл проекту з розширенням ***.DSN**;
11) у середовищі моделювання **«Proteus»** змінити схему підключень відповідно до варіанту для виконання лабораторного практикуму;
12) завантажити заздалегідь скомпільований бінарний файл з розширенням ***.COF** або ***.HEX** у мікроконтролер;
13) у середовищі моделювання **«Proteus»** виконати програму.

Послідовність виконання лабораторного практикуму для варіанту «Proteus» з використанням шаблону програми

1) для середовища моделювання **«Proteus»** використовувати готову схему;
2) відкрити папку **«Proteus_students»**;
3) відкрити інтегроване середовище розробки **PCWH**;

4) у папці **«Proteus_students»** вибрати папку лабораторного практикуму **«ADC»**;
5) відкрити файл проекту з розширенням ***.pjt**;
6) у редакторі **IDE PCWH** відкрити шаблон файлу програми з розширенням ***.c**;
7) відкрити середовище моделювання **«Proteus»**;
8) у папці **«Proteus_students»** вибрати папку лабораторного практикуму **«ADC»**;
9) відкрити файл проекту з розширенням ***.DSN**;
10) у середовищі моделювання **«Proteus»** змінити схему підключень відповідно до варіанту для виконання лабораторного практикуму;
11) створити алгоритм програми;
12) у інтегрованому середовищі розробки **PCWH**, використовуючи шаблон програми самостійно написати програму мовою програмування **C**, що реалізує створений алгоритм, відповідно до варіанту для виконання лабораторного практикуму,
13) скомпілювати програму, одержати бінарні файли з розширеннями ***.HEX** і ***.COF** (файли з розширеннями ***.HEX** і ***.COF** створюються підчас компіляції програми);
14) у середовищі моделювання **«Proteus»** виконати програму.

Примітка: *файл демонстрації виконання лабораторного практикуму* ***«ADC»*** *розташований на сайті* http://pksm.kntu.kr.ua/SCME.html.

Приклад створення проекту у інтегрованому середовищі розробки PCWH

Створення проекту у інтегрованому середовищі розробки PCWH за допомогою команди меню Project/Create

Для виконання лабораторного практикуму **«ADC»**, новий проект можна створити вручну за допомогою команди меню **Project/Create** (рис. 3.3.3):

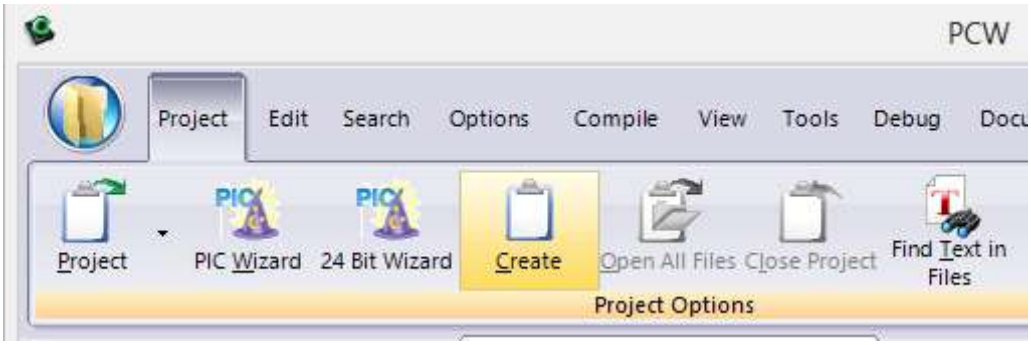


Рисунок 3.3.3 – Створення проекту у ручному режимі
за допомогою команди меню **Project/Create**

Якщо новий проект створений у ручному режимі за допомогою команди меню **Project/Create**, для конфігурування мікроконтролера для робота з АЦП, у заголовному файлі ***.h** необхідно ввести рядок, що вказує розрядність АЦП:

```
#device adc=8 або #device adc=10
```

а у функції ініціалізації ввести рядки:

```
setup_adc_ports(ALL_ANALOG);
setup_adc(ADC_CLOCK_INTERNAL);
```

Також новий проект можна згенерувати автоматично за допомогою майстра **PIC Wizard**, який викликається по команді меню **Project › PIC Wizard**, або використовувати шаблон програми, що знаходиться у папці **«Proteus_students»**. Необхідно вибрати папку відповідного лабораторного практикуму **«ADC»**.

Створення проекту у інтегрованому середовищі розробки PCWH за допомогою майстра PIC Wizard

Для запуску майстра **PIC Wizard** слід виконати відповідну команду меню **Project**, а потім вказати розміщення і ім'я головного файлу проекту. В результаті відкриється вікно майстра, що складається з розділів з параметрами проекту (рис. 3.3.7). Зовнішній вигляд вікна майстра може відрізнятися в залежності від версії компілятора **CCS-PICC** і обраного типу мікроконтролера.

1. Запустити **IDE PCWH**, натиснути кнопку **Projects** майстра **PIC Wizard** (рис. 3.3.4):

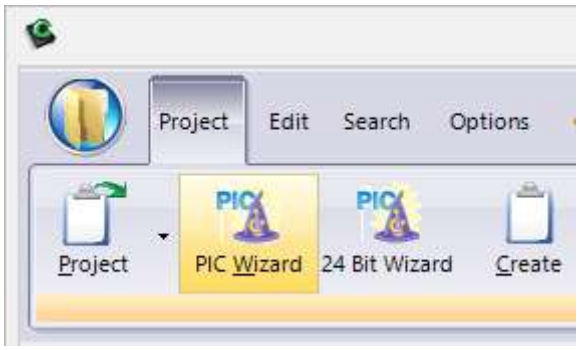


Рисунок 3.3.4 – Процес створення проекту у **IDE PCWH** за допомогою майстра **PIC Wizard**

або натиснути кнопку у вигляді відкритої папки  .

2. Вибрати: **New › Project Wizard** (рис. 3.3.5):

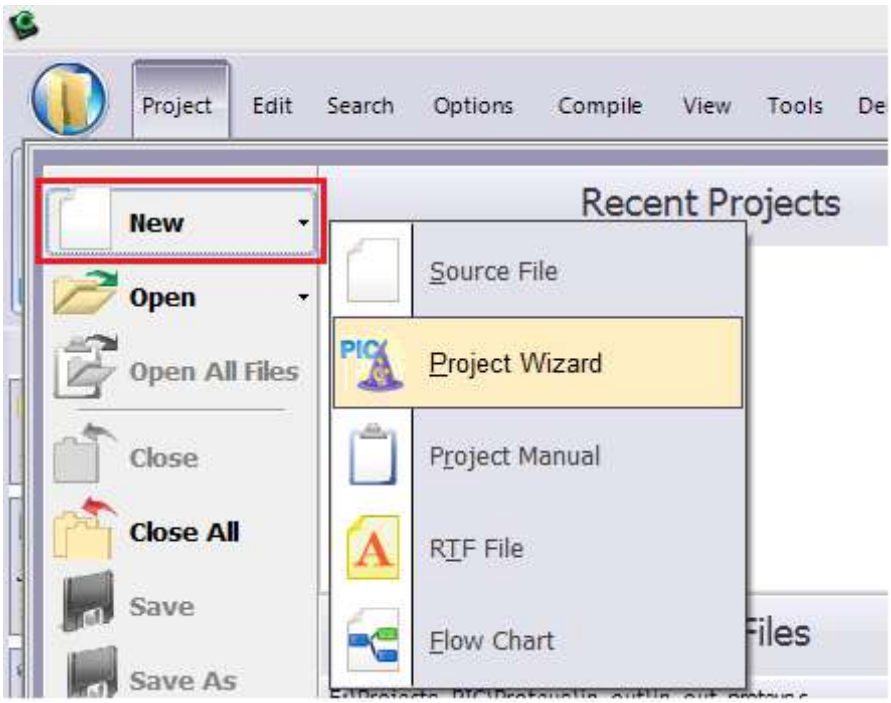


Рисунок 3.3.5 – Процес створення проекту у **IDE PCWH** за допомогою майстра **PIC Wizard**

3. Створити або вибрати свій директорій і записати у нього проект під будь-яким іменем латинським шрифтом, наприклад: «**adc.pjt**» (рис 3.3.6):

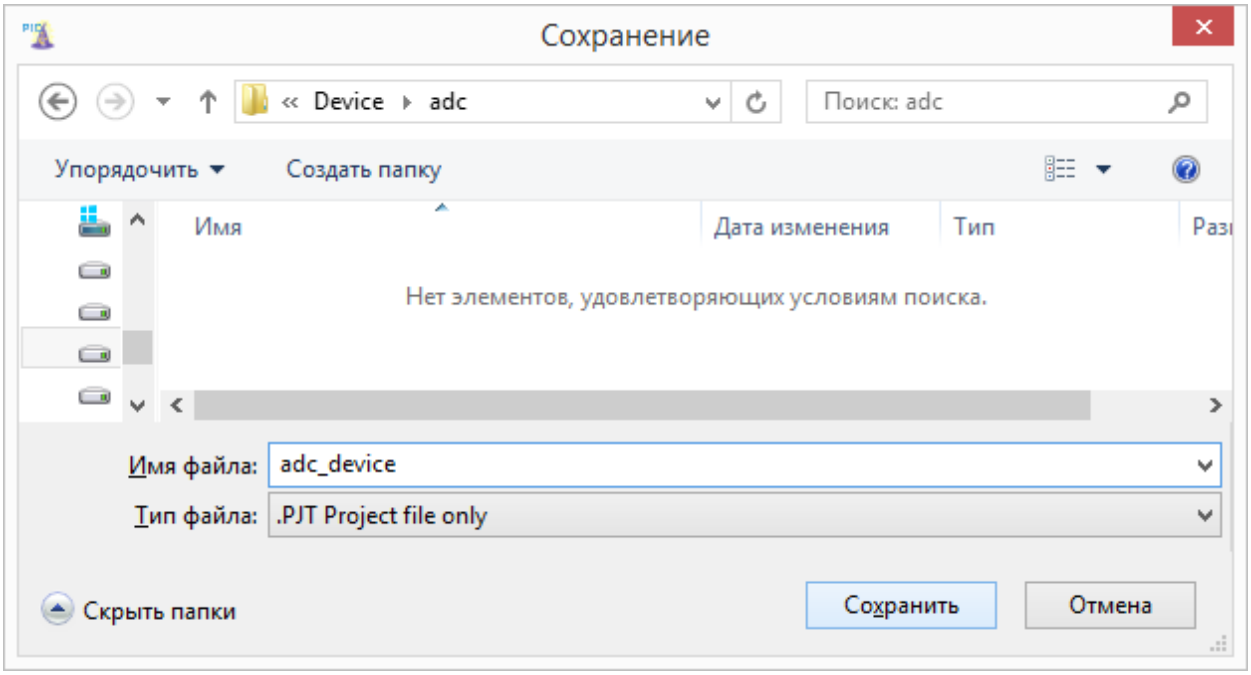


Рисунок 3.3.6 – Процес створення проекту у **IDE PCWH** за допомогою майстра **PIC Wizard**

У розділі **General** (рис 3.3.7) вибирається цільовий мікроконтролер (список **Device**, що розкривається), його робоча частота (поле **Oscillator Frequency**), а також настроюються загальні параметри, на зразок розрядів запобігання, типу джерела системної синхронізації, порогової напруги для скидання та ін.

Для перегляду програмного коду, який буде згенерований і доданий у вихідний файл відповідно до параметрів на поточній вкладці, слід вибрати вкладку **Code**.

4. У розділі **General › Device** (рис. 3.3.7) вибрати тип мікроконтролера, наприклад **PIC18F252**.

5. У розділі **General › Fuses** вибрати тип генератора **High speed Osc (> 4mhz for PCM/PCH) (>10mhz for PCD)**:

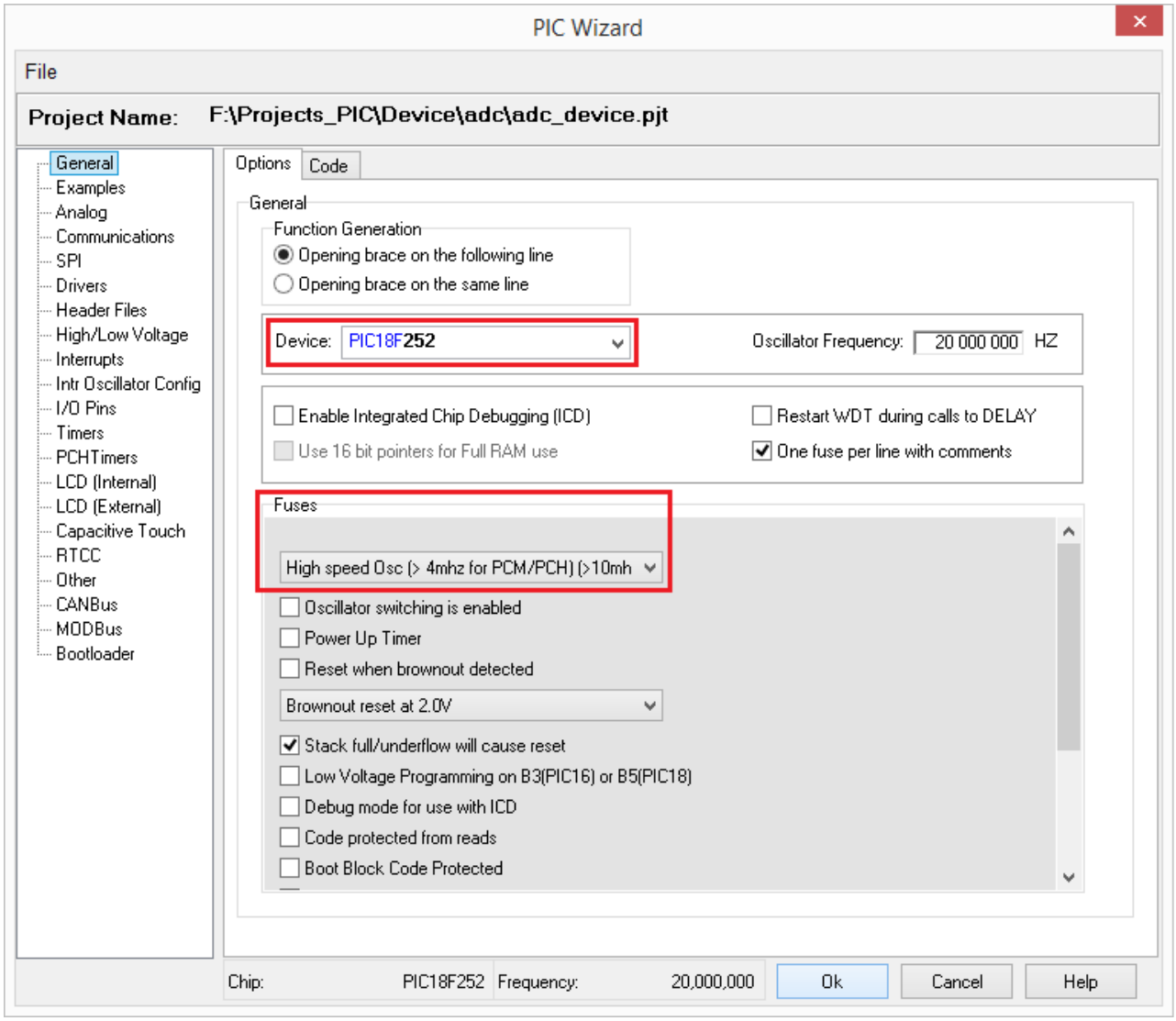


Рисунок 3.3.7 – Розділ **General** майстра **PIC Wizard**

У розділі **Communications** (рис. 3.3.8) налаштовуються параметри введення/виведення для інтерфейсів **RS-232** і **I²C** (швидкість обміну даними, відповідні лінії портів введення/виведення, перевірка помилок, режим **Master/Slave** і т.д.), а також активізується/відключається апаратний порт **PSP**.

6. У розділі **Communications** встановити мітку у полі **RS-232**:

- ☑ **Use RS-232**;

вказати:

- **Transmit:** **PIN C6** (передача);
- **Receive:** **PIN C7** (прийом).

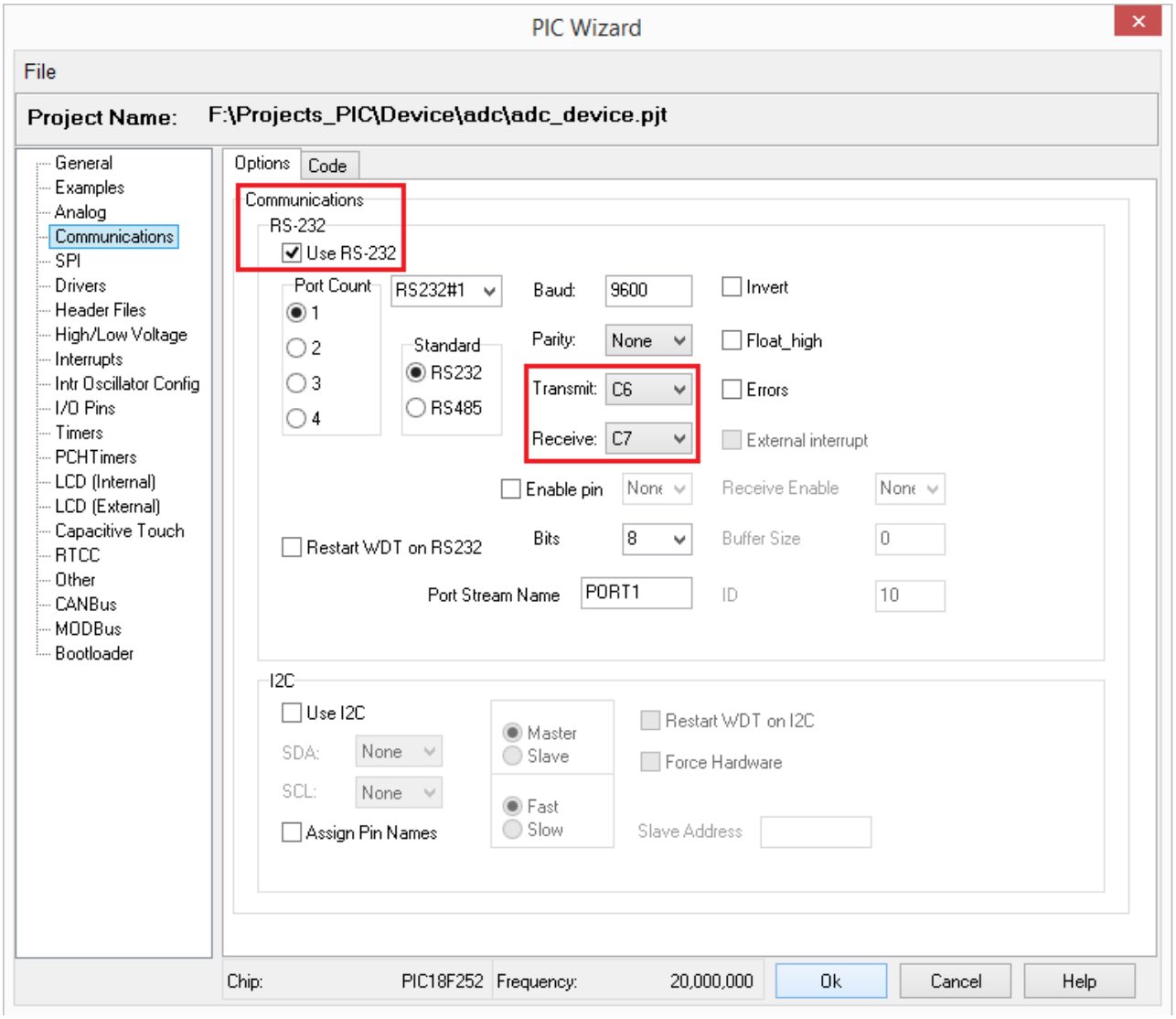


Рисунок 3.3.8 – Розділ **Communications** майстра **PIC Wizard**

Розділ **Analog** (рис. 3.3.9) служить для налаштування вбудованого АЦП (конфігурація аналогових входів, частота і розрядність перетворення).

7) У розділі **Analog** встановити:

у полі **Analog Input:**

- ◉ **A0 A1 A2 A3.**

Вказати:

розрядність АЦП: **8 (0-255)** або **10 (0-1023)** розрядів:

- **Units:** **0-255 / 0-1023**
- **Clock:** **4 us**

Конфігурування можна зробити з **IDE PCWH** при створенні проекту (рис. 3.3.9):

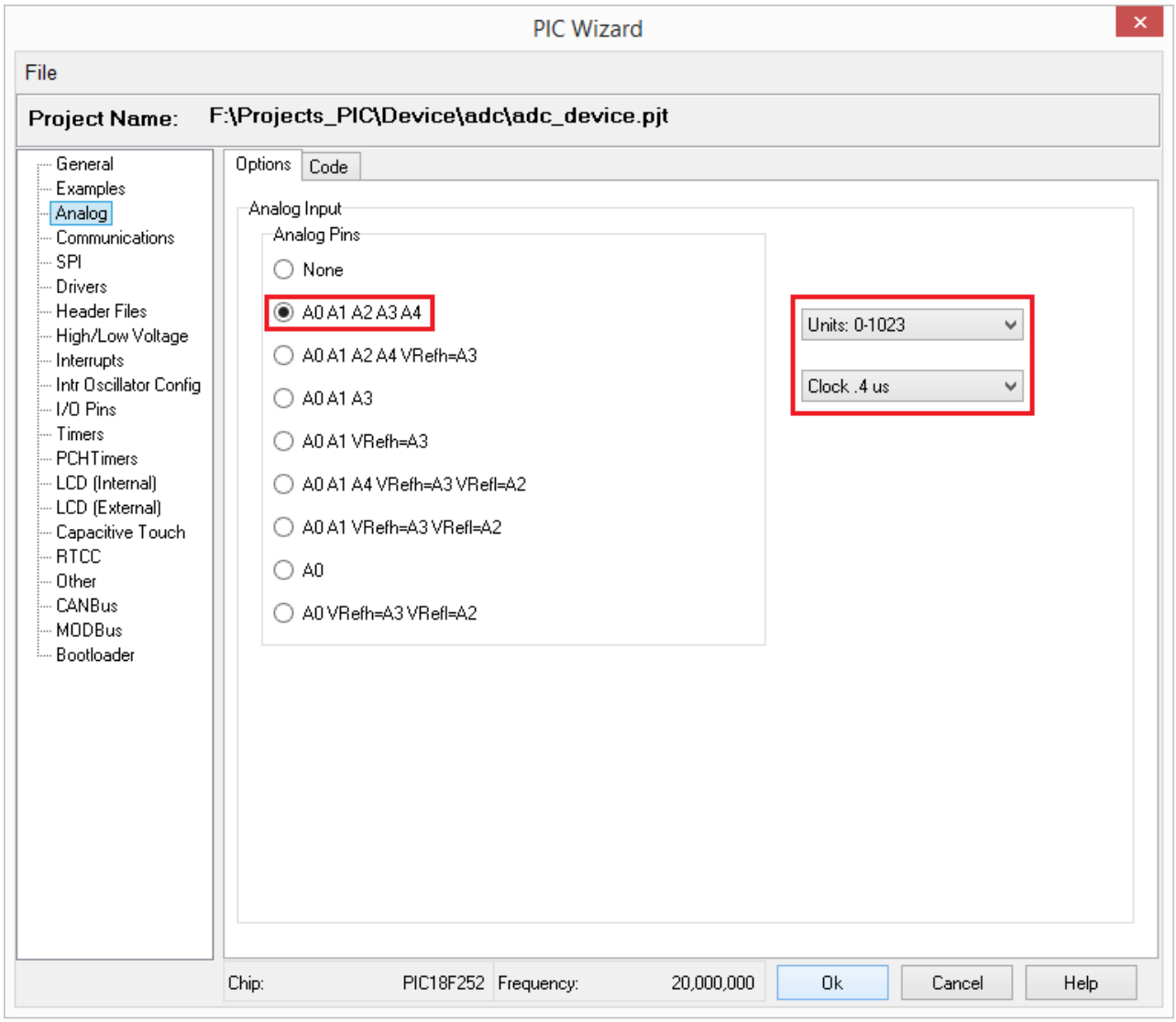


Рисунок 3.3.9 – Конфігурування АЦП із **IDE PCWH** при створенні проекту у розділі **Analog** майстра **PIC Wizard**

Буде згенерований і доданий програмний код у вихідний файл відповідно до параметрів на поточній вкладці.

Для перегляду програмного коду, слід вибрати вкладку **Code**.

8. Натиснути кнопку **OK**.

Буде згенерований наступний код (рис. 3.3.10):

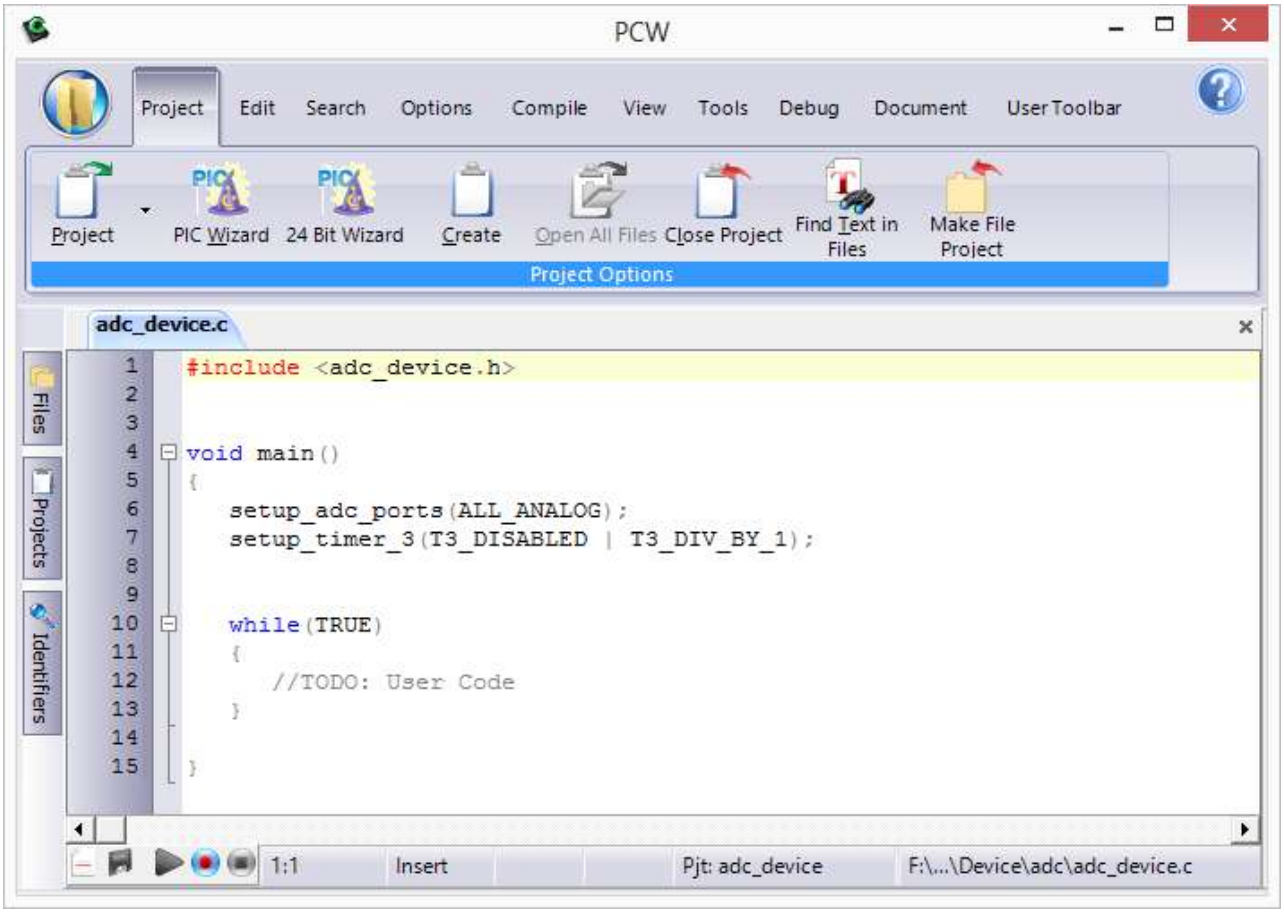


Рисунок 3.3.10 – Згенерований майстром **PIC Wizard** програмний код

Лістинг 3.3.1 – Приклад програмного коду, згенерованого майстром **PIC Wizard**

adc_device.c

```
#include <adc_device.h>

void main()
{
	setup_adc_ports(ALL_ANALOG);
	setup_timer_3(T3_DISABLED | T3_DIV_BY_1);
	while(TRUE)
	{
		//TODO: User Code
	}
}
```

adc_device.h

```
#include <18F252.h>
#device adc=10 //( або 8)

#FUSES NOWDT                    //No Watch Dog Timer
#FUSES WDT128                   //Watch Dog Timer uses 1:128
                                //Postscale
#FUSES HS      //High speed Osc (> 4mhz for PCM/PCH) (>10mhz for PCD)
#FUSES NOBROWNOUT               //No brownout reset
```

```
#FUSES NOLVP                          //No low voltage prgming, B3(PIC16)
                                      //or B5(PIC18) used for I/O

#use delay(clock=20000000)

#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8,stream=PORT1)
```

Проект готовий, можна приступати до написання програми.

Пояснення до виконання лабораторного практикуму «ADC»

Приклад зчитування даних з АЦП з каналу 2 і перетворення у вольти:

```
float volt;
int16 adc_val;
//====== вибрати номер каналу
set_adc_channel(2);
//====== затримка для роботи АЦП
delay_us(50);
//====== прочитати значення
adc_val = read_adc();
//====== визначити вольти
volt = adc_val * ADC_CVANT;
```

Змінна `ADC_CVANT` є ціною ділення шкали АЦП і обчислюється як частка від ділення V_{REF} на розрядність АЦП, виражену у двійкових одиницях:

```
//====== опорна напруга = напруга живлення
const float REF_VDD = 5.00;
const float ADC_8 = 255;
const float ADC_10 = 1023;

//====== ділення шкали: 5В/шкала (розрядність АЦП)
//=== 8 розрядів
const float ADC_CVANT = REF_VDD/ADC_8;
//=== 10 розрядів
const float ADC_CVANT = REF_VDD/ADC_10;
```

Для визначення натиснутої кнопки на клавіатурі необхідно визначити рівень напруги на кожній клавіші:

```
//====== коди клавіш у вольтах відповідно до розведення плати
const float KBD_CVANT = REF_VDD/12;              //усього 12 кнопок

//=== кнопки за схемою відповідно до розведення плати
const float ADC_BTN_0 = KBD_CVANT*4;
const float ADC_BTN_1 = KBD_CVANT*3;
...

const float ADC_BTN_STAR = KBD_CVANT*8;
const float ADC_BTN_DIEZ = KBD_CVANT*12;

//=== не натиснута жодна кнопка
const float ADC_BTN_NOT = KBD_CVANT/2;
```

Потім порівнювати значення АЦП у вольтах при натисканні клавіші з рівнями напруги, визначеним для кожної клавіші.

При збігу значень здійснюється ідентифікація натиснутою клавіші:

```
if((pushed_button > ADC_BTN_STAR-LUFT)&&(pushed_button <
ADC_BTN_STAR+LUFT))
{
        char_button = '*';
}

if((pushed_button > ADC_BTN_DIEZ-LUFT)&&(pushed_button <
ADC_BTN_DIEZ+LUFT))
{
        char_button = '#';
}

//====== клавіша не натиснута
if((pushed_button < ADC_BTN_NOT))
{
        char_button = 0;
}
```

Примітка: після виводу значень на LCD необхідно ввести затримку часу індикації:

```
delay_ms(100);
```

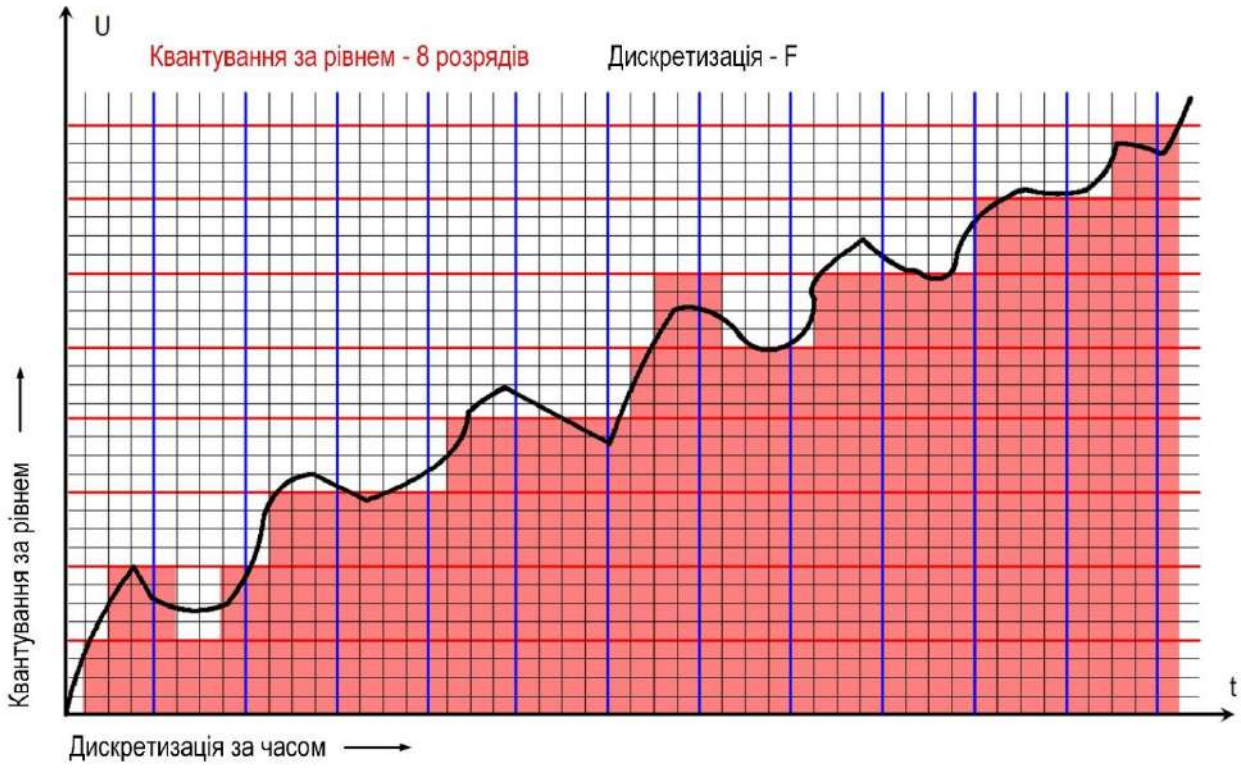


Рисунок 3.3.9 – Перетворення сигналу частотою дискретизації F і розрядністю = 8

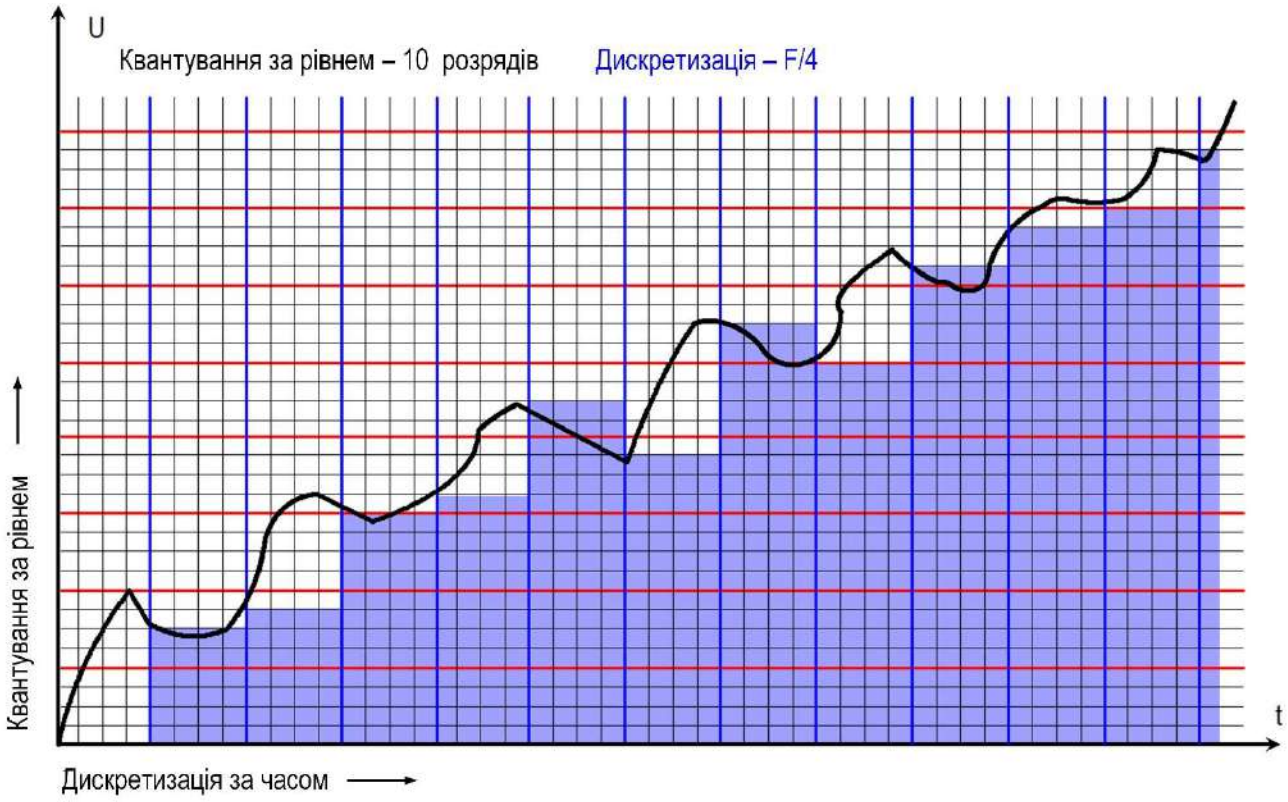


Рисунок 3.3.10 – Перетворення сигналу частотою дискретизації F/4 і розрядністю = 10

На рисунку 3.3.9 показані результати перетворення аналогового сигналу з низькою розв'язною здатністю АЦП – 8 розрядів (256 рівнів) і високою частотою дискретизації.

На рисунку 3.3.10 показані результати перетворення аналогового сигналу з більш високою розв'язною здатністю АЦП – 10 розрядів (1024) рівня і низькою частотою дискретизації.

Представлені графіки показують, що низька розв'язна здатність АЦП і низька частота дискретизації не дають можливості якісного перетворення аналогового сигналу.

Наприклад, у мікроконтролері **PIC18F252** розрядність АЦП – 8 і 10 розрядів. Частота дискретизації визначається програмістом.

Робота модуля АЦП

У мікроконтролері **PIC18F252** модуль аналого-цифрового перетворення (АЦП) має 5 аналогових входів (рис 3.3.11).

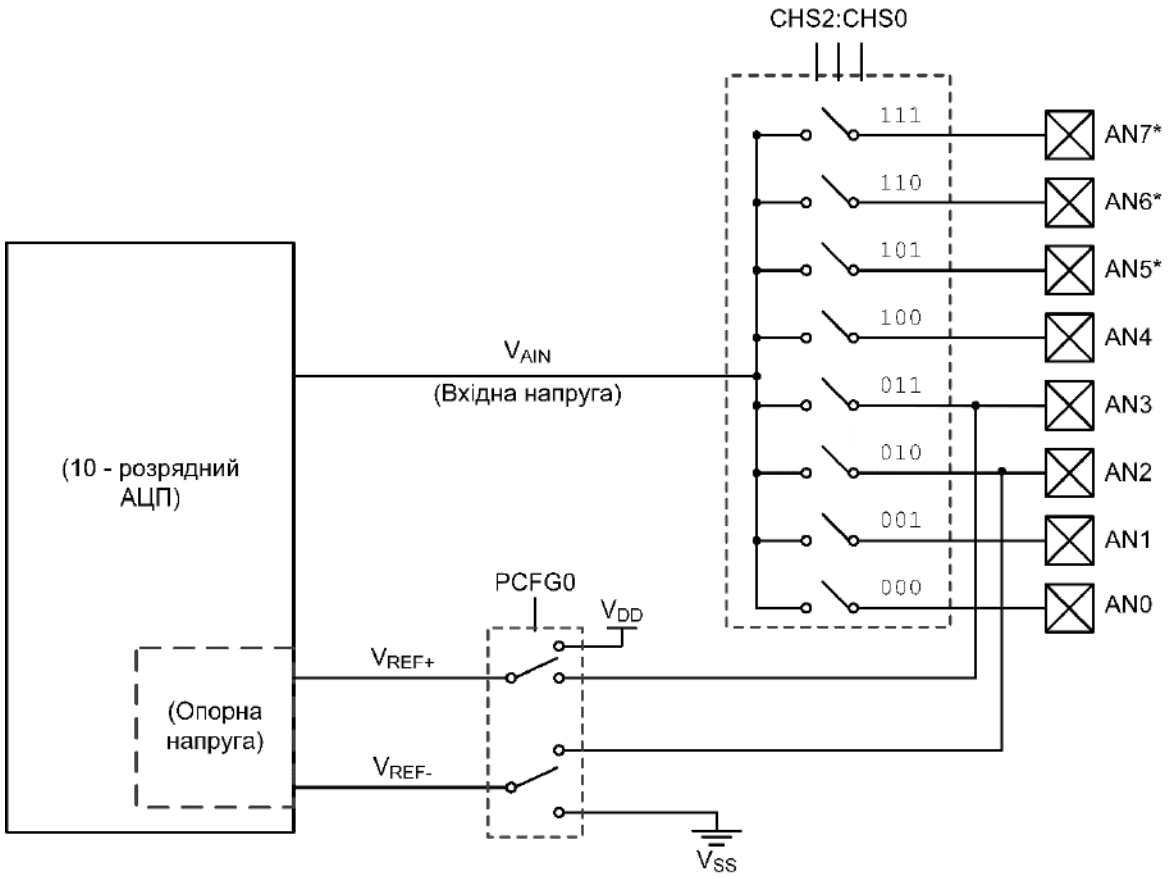


Рисунок 3.3.11 – Структурна схема модуля АЦП

Кожний канал порту, пов'язаний з модулем АЦП, може бути настроєний як аналоговий вхід (RA3 і RA2 як входи опорної напруги) або цифрового входу/виходу.

Вхідний аналоговий сигнал надходить на АЦП через комутатор каналів і перетворюється у відповідний 10-розрядний цифровий код.

Позитивний і негативний вхід опорної напруги може бути програмно обраний з виводів V_{DD} і V_{SS}, або з входом AN3 / V_{REF+} і AN2 / V_{REF-}

Допускається робота модуля АЦП у SLEEP режимі мікроконтролера, при цьому в якості джерела імпульсів для АЦП повинен бути обраний RS генератор.

Принцип перетворення представлений на рис. 3.3.12.

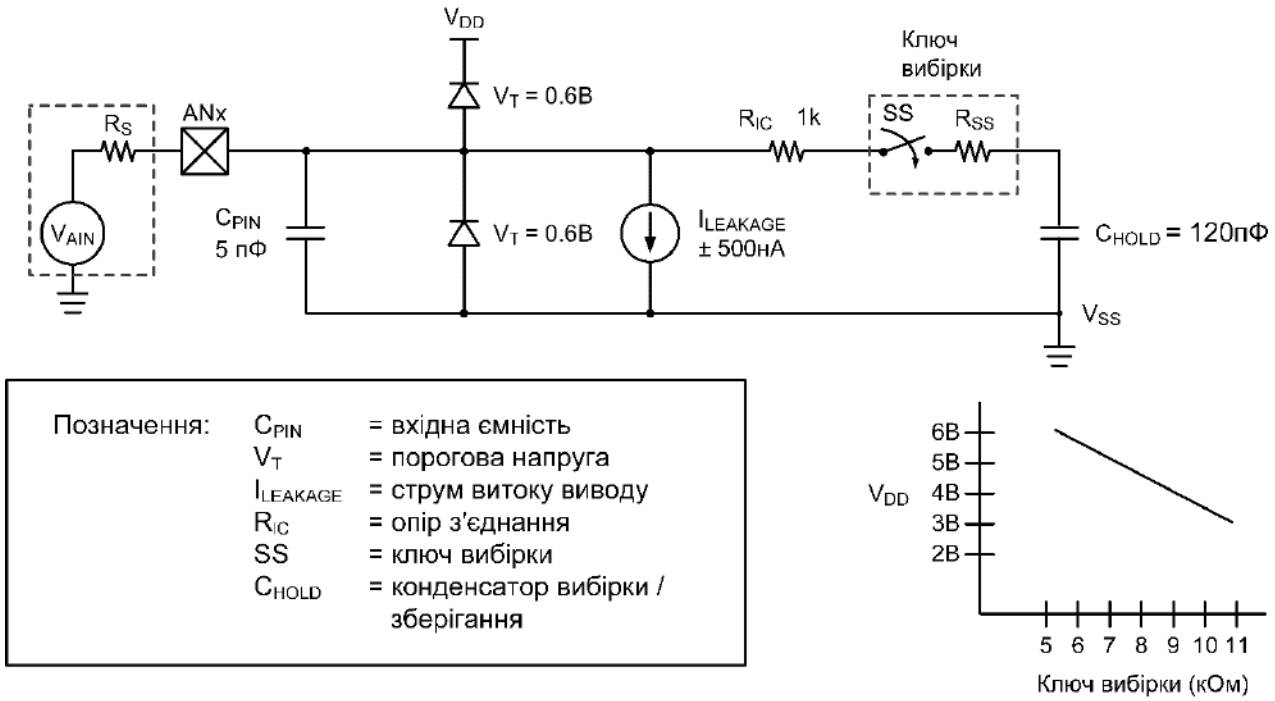


Рисунок 3.3.12 – Схема аналогового входу АЦП

Вхідний аналоговий сигнал через комутатор каналів заряджає внутрішній конденсатор АЦП C_{HOLD}.

Модуль АЦП перетворює напругу, яка утримувалась на конденсаторі C_{HOLD} у відповідний 10-розрядний цифровий код методом послідовного наближення.

При скиданні мікроконтролера значення всіх його регістрів встановлюються за замовчуванням.

Скидання виключає модуль АЦП, а також зупиняє процес перетворення, якщо він був початий.

Розташування виводів АЦП мікроконтролера представлено на рис. 3.3.13:

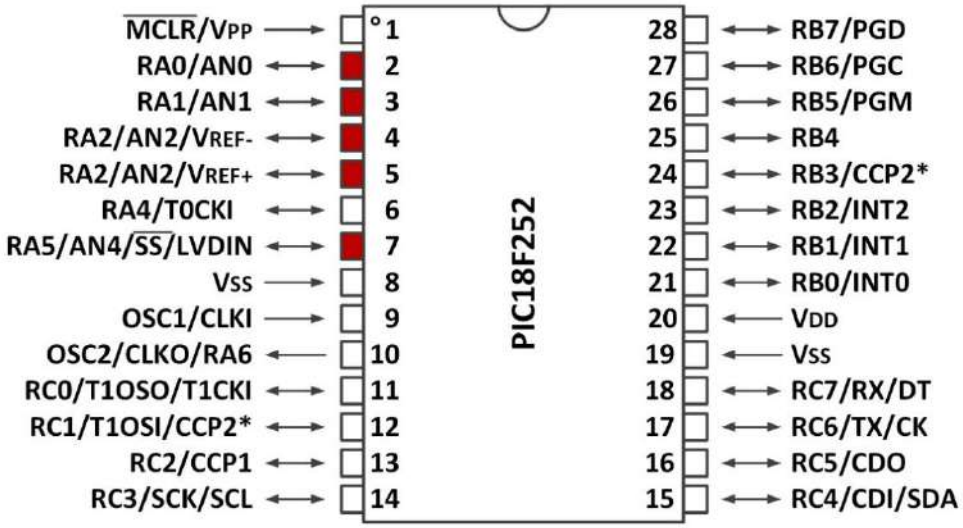


Рисунок 3.3.13 – Розташування виводів АЦП мікроконтролера

Приклади виконання лабораторного практикуму «ADC»

Завдання для лабораторного практикуму «ADC»

Програма має вигляд і структуру відповідно до лістингів 3.3.2, 3.3.3:

- лістинг 3.3.2 – для варіанту **АПК;**
- лістинг 3.3.3 – для варіанту **«Proteus»**.

Завдання для виконання лабораторного практикуму **«ADC»** представлене у таблиці 3.3.2.

Таблиця 3.3.2 – Завдання для виконання лабораторного практикуму **«ADC»**

Канал АЦП								**Розрядність АЦП**	
Потенціометр				**Клавіатура**					
A0	A1	A2	A3	A0	A1	A2	A3	8	10
			+			+			+

Принципова схема підключень для варіанту АПК

Принципова схема підключень для варіанту **АПК**, для виконання лабораторного практикуму **«ADC»** виглядає у такий спосіб (рис. 3.3.14):

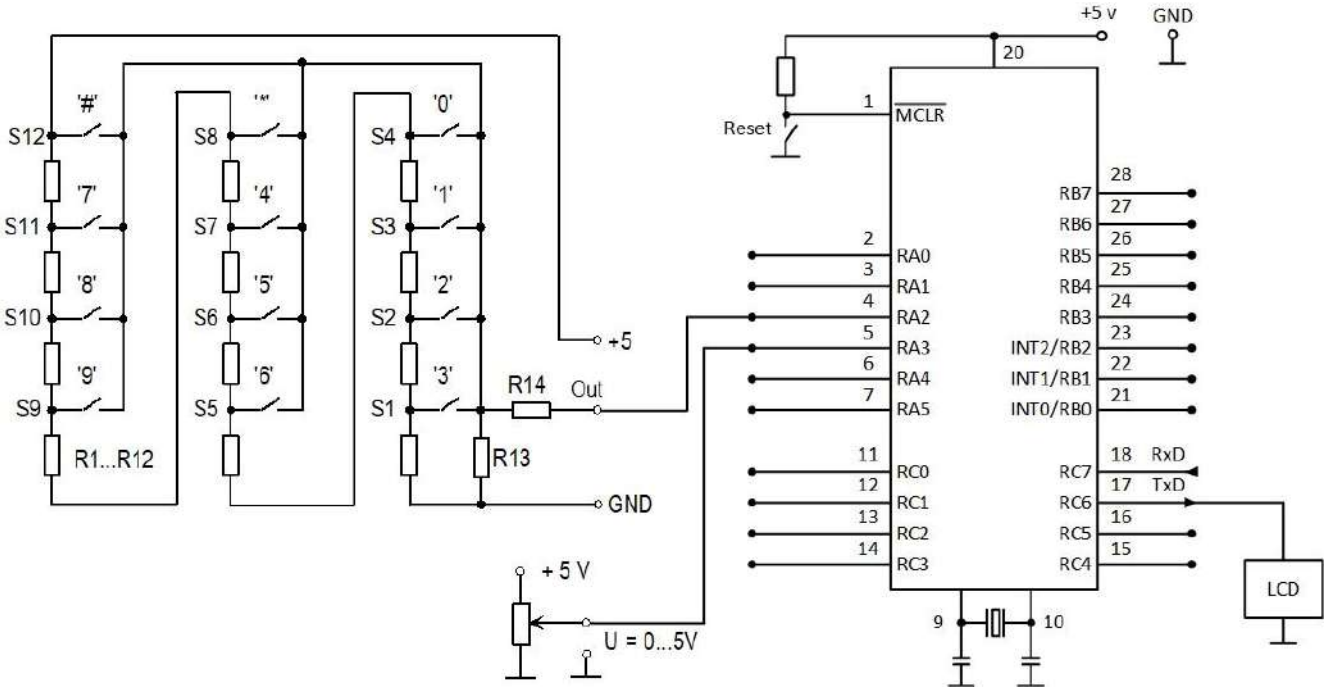


Рисунок 3.3.14 – Принципова схема підключень для лабораторного практикуму **«ADC»** для варіанту **АПК**

Принципова схема підключень для варіанту «Proteus»

Принципова схема для варіанту **«Proteus»**, для виконання лабораторного практикуму **«ADC»** виглядає у такий спосіб (рис. 3.3.15):

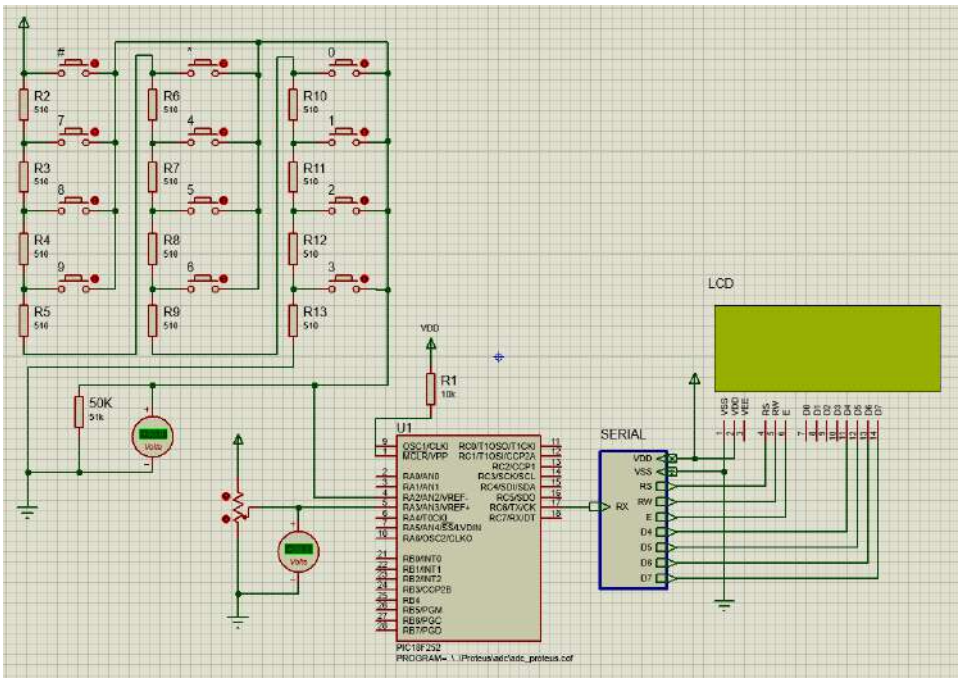


Рисунок 3.3.15 – Принципова схема підключень лабораторного практикуму **«ADC»** для варіанту **«Proteus»**

Результат виконання лабораторного практикуму «ADC» для варіанту АПК

Лістинг 3.3.2 – Програма реалізації алгоритму лабораторного практикуму **«ADC»** для варіанту **АПК**

```
//#################################################################
// Лабораторний практикум «ADC»
// file: adc_device.c / робота з АЦП
// Copyright (c) Docent Smirnov V.V., Docent Smirnova N.V / CNTU
//#################################################################

#include < adc _device.h>
#include <swc_LCD.h>                        //драйвер LCD дисплея
#CASE                                       //враховувати регістр символів
//================================================================================
// Инициализация PIC
//================================================================================
void init()
{
       setup_adc_ports(ALL_ANALOG);
       setup_adc(ADC_CLOCK_INTERNAL); //Add
       setup_spi(SPI_SS_DISABLED);
       setup_wdt(WDT_OFF);
       setup_timer_0(RTCC_INTERNAL);
       setup_timer_1(T1_DISABLED);
       setup_timer_2(T2_DISABLED,0,1);
       enable_interrupts(GLOBAL);
}

////////////////////////////////////////////////////////////////////////////////
//
//клавіатура і ADC
//====== опорна напруга = напруга живлення
const float REF_VDD = 5.00;
const float ADC_8 = 255;
const float ADC_10 = 1023;

//====== розподіл шкали: 5В / шкала (розрядність АЦП)
//=== 8 розрядів
const float ADC_CVANT = REF_VDD/ADC_8;
//=== 10 розрядів
//const float ADC_CVANT = REF_VDD/ADC_10;

//====== прототипи
float get_ADC_volt(int8 channel);
int16 get_ADC_hex(int8 channel);

//====== коди клавіш у вольтах відповідно до розводки плати
const float KBD_CVANT = REF_VDD/12;              //всього 12 кнопок
//=== кнопки за схемою відповідно до розводки плати
const float ADC_BTN_0 = KBD_CVANT*4;
const float ADC_BTN_1 = KBD_CVANT*3;
const float ADC_BTN_2 = KBD_CVANT*2;
const float ADC_BTN_3 = KBD_CVANT*1;
const float ADC_BTN_4 = KBD_CVANT*7;
const float ADC_BTN_5 = KBD_CVANT*6;
const float ADC_BTN_6 = KBD_CVANT*5;
const float ADC_BTN_7 = KBD_CVANT*11;
const float ADC_BTN_8 = KBD_CVANT*10;
```

```
const float ADC_BTN_9 = KBD_CVANT*9;
const float ADC_BTN_STAR = KBD_CVANT*8;
const float ADC_BTN_DIEZ = KBD_CVANT*12;

//=== не було натиснуто жодної кнопки
const float ADC_BTN_NOT = KBD_CVANT/2;
const float LUFT = KBD_CVANT/4;

//=============================================================================
// Прочитати значення ADC у HEX
//=============================================================================
int16 get_ADC_hex(int8 channel)
{
	int16 adc_val;

	//====== вибрати номер каналу
	set_adc_channel(channel);
	//====== затримка для роботи АЦП
	delay_us(50);
	//====== прочитати значення
	adc_val = read_adc();

	return adc_val;
}

//=============================================================================
// Прочитати значення ADC у вольтах з фільтрацією
//=============================================================================
float get_ADC_volt_filter(int8 channel,int16 filter_depth)
{
	float volt;
	float adc_val;
	int8 i;
	float volt_sum = 0;

	//====== прочитати значення АЦП HEX
	for(i = 0;i<filter_depth;i++){
		//====== значення в HEX
		adc_val = get_ADC_hex(channel);
		//====== визначити вольти
		volt = adc_val * ADC_CVANT;
		volt_sum += volt;
	}

	//====== отримати середнє
	volt = volt_sum/filter_depth;
	return volt;
}

//=============================================================================
// Прочитати значення ADC у вольтах
//=============================================================================
float get_ADC_volt(int8 channel)
{
	float volt;
	int16 adc_val;

	//====== прочитати значення АЦП HEX
	//adc_val = get_ADC_hex(channel);
	//====== вибрати номер каналу
	set_adc_channel(channel);
	//====== затримка для роботи АЦП
	delay_us(50);
```

```
        //====== прочитати значення
        adc_val = read_adc();
        //====== визначити вольти
        volt = adc_val * ADC_CVANT;

        return volt;
}

//=========================================================================
// Визначити натиснуту кнопку
//=========================================================================
int8 get_push_button(int8 channel)
{
        static int8 flag_btn_push;          //прапор натискання кнопки
        float pushed_button = 0;            //натиснута кнопка вхід
        int8 char_button;                   //натиснута кнопка вихід

        pushed_button = get_ADC_volt(channel);

        //====== визначити натиснуту клавішу
        if((pushed_button > ADC_BTN_0-LUFT)&&(pushed_button <
        ADC_BTN_0+LUFT)){
                char_button = '0';
        }
        if((pushed_button > ADC_BTN_1-LUFT)&&(pushed_button <
        ADC_BTN_1+LUFT)){
                char_button = '1';
        }
        if((pushed_button > ADC_BTN_2-LUFT)&&(pushed_button <
        ADC_BTN_2+LUFT)){
                char_button = '2';
        }
        if((pushed_button > ADC_BTN_3-LUFT)&&(pushed_button <
        ADC_BTN_3+LUFT)){
                char_button = '3';
        }
        if((pushed_button > ADC_BTN_4-LUFT)&&(pushed_button <
        ADC_BTN_4+LUFT)){
                char_button = '4';
        }
        if((pushed_button > ADC_BTN_5-LUFT)&&(pushed_button <
        ADC_BTN_5+LUFT)){
                char_button = '5';
        }
        if((pushed_button > ADC_BTN_6-LUFT)&&(pushed_button <
        ADC_BTN_6+LUFT)){
                char_button = '6';
        }
        if((pushed_button > ADC_BTN_7-LUFT)&&(pushed_button <
        ADC_BTN_7+LUFT)){
                char_button = '7';
        }
        if((pushed_button > ADC_BTN_8-LUFT)&&(pushed_button <
        ADC_BTN_8+LUFT)){
                char_button = '8';
        }
        if((pushed_button > ADC_BTN_9-LUFT)&&(pushed_button <
        ADC_BTN_9+LUFT)){
                char_button = '9';
        }
        if((pushed_button > ADC_BTN_STAR-LUFT)&&(pushed_button <
        ADC_BTN_STAR+LUFT)){
                char_button = '*';
        }
```

```
        if((pushed_button > ADC_BTN_DIEZ-LUFT)&&(pushed_button <
        ADC_BTN_DIEZ+LUFT)){
                char_button = '#';
        }
        //====== не була натиснута
        if((pushed_button < ADC_BTN_NOT)){
                char_button = 0;
        }
        //====== перевірка на відтиснення
        if(char_button == 0){
                //====== клавіша відпущена
                flag_btn_push = 0;
        }else{
                //====== клавіша натиснута
                flag_btn_push ++;
        }
        //printf("\n\r U = %f char - %c ",pushed_button,char_button);
        //
        if(flag_btn_push > 1){
                return 0;
        }else{
                return char_button;
        }
}

//=========================================================================
// Main
//=========================================================================
void main()
{
        int16 adc_val;              //значення, прочитане з ADC
        float volt;                 //вихідне значення у вольтах
        int8 button;
        //
        init();
        lcd_init();
        //====== титли
        lcd_running_string_begin(4);
        printf("Лабораторний практикум ADC / Кафедра ПКСМ / доц. Смірнов
        В.В.");
        lcd_running_string_end();
        lcd_running_string_start();
        //====== титли для 1 частини
        lcd_goto(1,1);
        printf("HEX Value: ");
        lcd_goto(2,1);
        printf("Вольти: ");
        //====== титли для 2 частини
        lcd_goto(3,1);
        printf("Нажата кнопка: ");
        //====== головний цикл
        for(;;){
                ////////////////////////////////////////////////
                //Робота з потенціометром
                //====== отримати значення АЦП у HEX форматі з каналу 3
                adc_val = get_ADC_hex(3);
                //====== отримати значення АЦП у вольтах з каналу 3
                volt = get_ADC_volt_filter(3,200);
                //volt = get_ADC_volt(3);
                //====== вивести результат
                //=== HEX
                lcd_goto(1,12);
                printf("%LX",adc_val);
```

```
                //=== Вольти
                lcd_goto(2,9);
                printf("%f",volt);

                ////////////////////////////////////////////
                //Робота з клавіатурою
                //====== отримати символ з клавітури з каналу 2
                if(button = get_push_button(2)){
                        //====== вивести результат
                        lcd_goto(3,16);
                        printf("%c",button);
                }
                delay_ms(100);
        }
}
```

Результат виконання лабораторного практикуму «ADC» для варіанту «Proteus»

Лістинг 3.3.3 – Програма реалізації алгоритму лабораторного практикуму **«ADC»** для варіанту **«Proteus»**

```
//#################################################################
// Лабораторний практикум «ADC»
// file: adc_proteus.c / робота з АЦП
// Copyright (c) Docent Smirnov V.V., Docent Smirnova N.V / CNTU
//#################################################################

#include "adc_proteus.h"
#include <swc_LCD.h>                        //драйвер LCD дисплея
#CASE                                       //враховувати регістр символів
//=================================================================
// Ініціалізація PIC
//=================================================================
void init()
{
        setup_adc_ports(ALL_ANALOG);
        setup_adc(ADC_CLOCK_INTERNAL);      //Add
        setup_spi(SPI_SS_DISABLED);
        setup_wdt(WDT_OFF);
        setup_timer_0(RTCC_INTERNAL);
        setup_timer_1(T1_DISABLED);
        setup_timer_2(T2_DISABLED,0,1);
        enable_interrupts(GLOBAL);
}

////////////////////////////////////////////////////////////////////
//
// клавіатура і ADC
//====== опорна напруга = напруга живлення
const float REF_VDD = 5.00;
const float ADC_8 = 255;
//const float ADC_10 = 1023;

//====== розподіл шкали: 5В / шкала (розрядність АЦП)
//=== 8 розрядів
const float ADC_CVANT = REF_VDD/ADC_8;
//=== 10 розрядів
//const float ADC_CVANT = REF_VDD/ADC_10;
```

```
//====== прототипи
float get_ADC_volt(int8 channel);
int16 get_ADC_hex(int8 channel);

//====== коди клавіш у вольтах відповідно до розводки плати
const float KBD_CVANT = REF_VDD/12;              //всього 12 кнопок

//=== кнопки за схемою відповідно до розводки плати
const float ADC_BTN_0 = KBD_CVANT*4;
const float ADC_BTN_1 = KBD_CVANT*3;
const float ADC_BTN_2 = KBD_CVANT*2;
const float ADC_BTN_3 = KBD_CVANT*1;
const float ADC_BTN_4 = KBD_CVANT*7;
const float ADC_BTN_5 = KBD_CVANT*6;
const float ADC_BTN_6 = KBD_CVANT*5;
const float ADC_BTN_7 = KBD_CVANT*11;
const float ADC_BTN_8 = KBD_CVANT*10;
const float ADC_BTN_9 = KBD_CVANT*9;
const float ADC_BTN_STAR = KBD_CVANT*8;
const float ADC_BTN_DIEZ = KBD_CVANT*12;
//=== не було натиснуто жодної кнопки
const float ADC_BTN_NOT = KBD_CVANT/2;
const float LUFT = KBD_CVANT/4;

//=========================================================================
// Прочитати значення ADC у HEX
//=========================================================================
int16 get_ADC_hex(int8 channel)
{
       int16 adc_val;

       //====== вибрати номер каналу
       set_adc_channel(channel);
       //====== затримка для роботи АЦП
       delay_us(50);

       //====== прочитати значення
       adc_val = read_adc();
       return adc_val;
}
//=========================================================================
// Прочитати значення ADC у вольтах з фільтрацією
//=========================================================================
float get_ADC_volt_filter(int8 channel,int16 filter_depth)
{
       float volt;
       float adc_val;
       int8 i;
       float volt_sum = 0;

       //====== прочитати значення АЦП HEX
       for(i = 0;i<filter_depth;i++){
       //====== значення у HEX
       adc_val = get_ADC_hex(channel);

       //====== визначити вольти
       volt = adc_val * ADC_CVANT;
       volt_sum += volt;
       }
       //====== отримати середнє
       volt = volt_sum/filter_depth;
       return volt;
}
```

```
//=========================================================================
// Прочитати значення ADC у вольтах
//=========================================================================
float get_ADC_volt(int8 channel)
{
	float volt;
	int16 adc_val;
	//====== прочитати значення АЦП у HEX
	//adc_val = get_ADC_hex(channel);
	//====== вибрати номер каналу
	set_adc_channel(channel);
	//====== затримка для роботи АЦП
	delay_us(50);
	//====== прочитати значення
	adc_val = read_adc();
	//====== визначити вольти
	volt = adc_val * ADC_CVANT;
	return volt;
}
//=========================================================================
// Визначити натиснуту кнопку
//=========================================================================
int8 get_push_button(int8 channel)
{
	static int8 flag_btn_push;		//прапор натискання кнопки
	float pushed_button = 0;		//натиснута кнопка вхід
	int8 char_button;			//натиснута кнопка вихід
	pushed_button = get_ADC_volt(channel);

	//====== визначити натиснуту клавішу
	if((pushed_button > ADC_BTN_0-LUFT)&&(pushed_button <
	ADC_BTN_0+LUFT)){
		char_button = '0';
	}
	if((pushed_button > ADC_BTN_1-LUFT)&&(pushed_button <
	ADC_BTN_1+LUFT)){
		char_button = '1';
	}
	if((pushed_button > ADC_BTN_2-LUFT)&&(pushed_button <
	ADC_BTN_2+LUFT)){
		char_button = '2';
	}
	if((pushed_button > ADC_BTN_3-LUFT)&&(pushed_button <
	ADC_BTN_3+LUFT)){
		char_button = '3';
	}
	if((pushed_button > ADC_BTN_4-LUFT)&&(pushed_button <
	ADC_BTN_4+LUFT)){
		char_button = '4';
	}
	if((pushed_button > ADC_BTN_5-LUFT)&&(pushed_button <
	ADC_BTN_5+LUFT)){
		char_button = '5';
	}
	if((pushed_button > ADC_BTN_6-LUFT)&&(pushed_button <
	ADC_BTN_6+LUFT)){
		char_button = '6';
	}
	if((pushed_button > ADC_BTN_7-LUFT)&&(pushed_button <
	ADC_BTN_7+LUFT)){
		char_button = '7';
	}
	if((pushed_button > ADC_BTN_8-LUFT)&&(pushed_button <
	ADC_BTN_8+LUFT)){
```

```
                char_button = '8';
        }
        if((pushed_button > ADC_BTN_9-LUFT)&&(pushed_button <
        ADC_BTN_9+LUFT)){
                char_button = '9';
        }
        if((pushed_button > ADC_BTN_STAR-LUFT)&&(pushed_button <
        ADC_BTN_STAR+LUFT)){
                char_button = '*';
        }
        if((pushed_button > ADC_BTN_DIEZ-LUFT)&&(pushed_button <
        ADC_BTN_DIEZ+LUFT)){
                char_button = '#';
        }
        //====== не було натиснуто
        if((pushed_button < ADC_BTN_NOT)){
                char_button = 0;
        }
        //====== перевірка на віджимання
        if(char_button == 0){
                //====== клавіша відпущена
                flag_btn_push = 0;
        }else{
                //====== клавіша натиснута
                flag_btn_push ++;
        }
        //printf("\n\r U = %f char - %c ",pushed_button,char_button);
        //
        if(flag_btn_push > 1){
                return 0;
        }else{
                return char_button;
        }
}
//=========================================================================
// Main
//=========================================================================
void main()
{
        int16 adc_val;              //значення, прочитане з ADC
        float volt;                 //вихідне значення у вольтах
        int8 button;
        init();
        lcd_init();
        //====== головний цикл
        for(;;){
                ////////////////////////////////////////////
                //Робота з потенціометром
                //====== отримати значення АЦП у HEX форматі з каналу 3
                adc_val = get_ADC_hex(3);
                //====== отримати значення АЦП у вольтах з каналу 3
                volt = get_ADC_volt_filter(3,200);
                //volt = get_ADC_volt(3);
                //====== вивести результат
                lcd_goto(1,1);
                printf("docent Smirnov V.V.");
                //=== HEX
                lcd_goto(2,1);
                printf("HEX: %LX",adc_val);
                //=== Вольти
                lcd_goto(3,1);
                printf("Volt:%f",volt);
```

```
                    ////////////////////////////////////////////
                    //Робота з клавіатурою
                    //====== отримати символ з клавітури з каналу 2
                    if(button = get_push_button(2)){
                           //====== вивести результат
                           lcd_goto(4,1);
                           printf("Btn: %c",button);
                    }
             }
      }
```

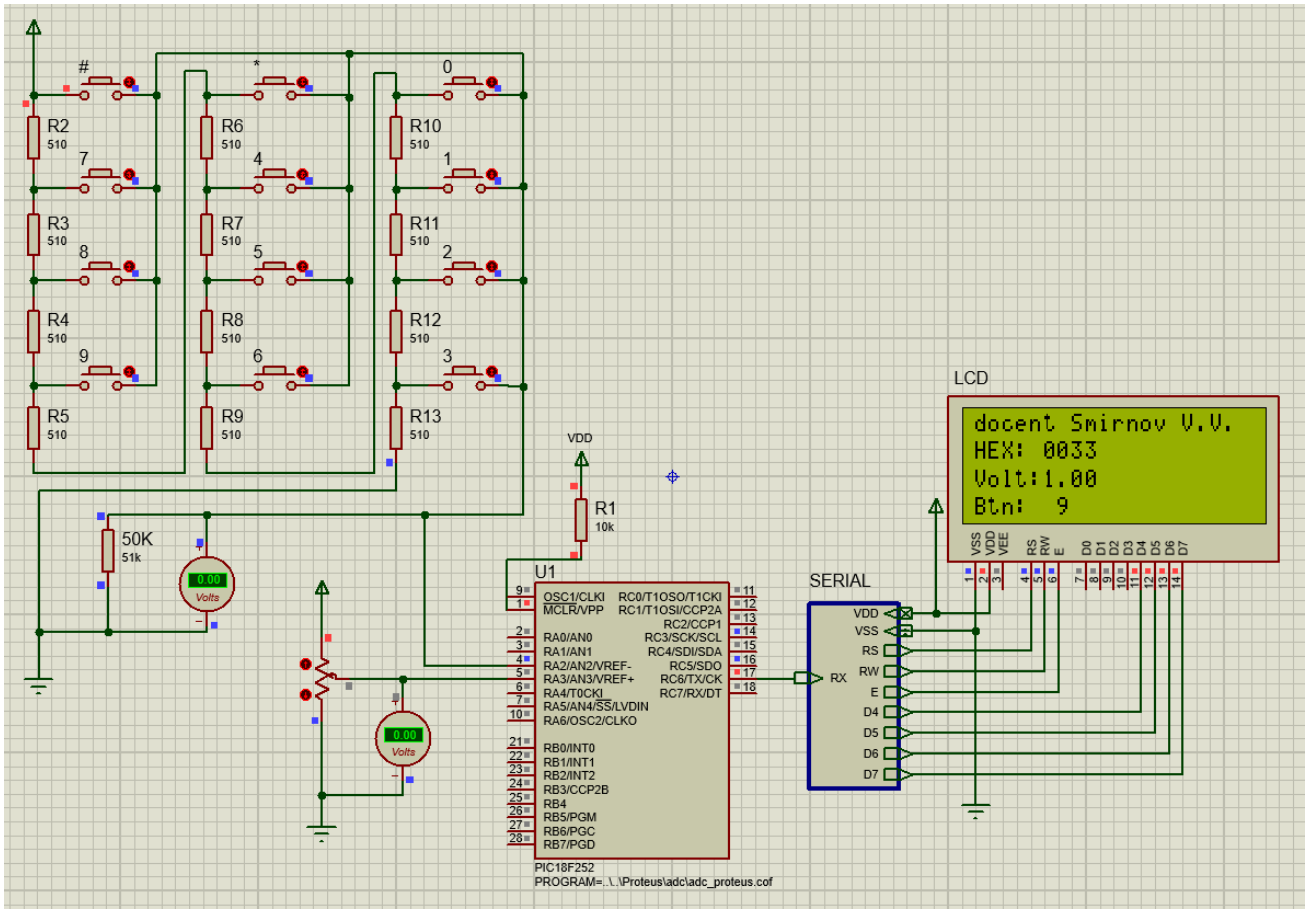


Рисунок 3.3.16 – Результат виконання лабораторного практикуму **«ADC»** для варіанту **«Proteus»**

Контрольні питання

1) призначення АЦП і методи перетворення;
2) конфігурування мікроконтролера для роботи з АЦП;
3) вплив розрядності АЦП на точність перетворення;
4) опорна напруга VREF і його вплив на параметри АЦП;
5) як визначається ціна ділення (вага) шкали АЦП?

ЛАБОРАТОРНИЙ ПРАКТИКУМ № 4 - «DAC»

Тема: Робота з ЦАП

Ціль роботи:

Отримання навичок роботи із ЦАП MCP4921 і написання драйвера послідовного інтерфейсу SPI.

Завдання лабораторного практикуму

- для варіанту **АПК** намалювати принципову схему підключень відповідно до варіанту для виконання лабораторного практикуму (таб. 3.4.1);
- для варіанту **«Proteus»** використовувати готову схему відповідно до варіанту для виконання лабораторного практикуму;
- створити алгоритм програми роботи з ЦАП по інтерфейсу SPI;
- написати програму драйвера послідовного інтерфейсу SPI;
- здійснити зчитування рівня напруги з потенціометра, оцифрувати за допомогою АЦП відповідно до варіанту для виконання лабораторного практикуму;
- відобразити рівень поточної напруги у вольтах на LCD;
- вивести цифрове значення напруги на ЦАП через інтерфейс SPI;
- здійснити зчитування рівня напруги з виходу ЦАП, оцифрувати за допомогою АЦП;
- відобразити рівень напруги з виходу ЦАП у вольтах на LCD;
- здійснити виведення результатів на LCD відповідно до варіанту для виконання лабораторного практикуму.

Виведення на LCD повинне містити:

- зчитані дані з АЦП у вольтах;
- зчитані дані з ЦАП у вольтах;
- групу студента;
- прізвище та ініціали студента.

Варіанти для виконання лабораторного практикуму «DAC»

Таблиця 3.4.1 – Варіанти для виконання лабораторного практикуму **«DAC»**

№	Канал АЦП							
	Потенціометр				ЦАП			
	A0	A1	A2	A3	A0	A1	A2	A3
1	+					+		
2		+					+	
3			+					+
4				+	+			
5		+			+			
6			+			+		
7	+						+	
8	+							+
9		+					+	
10			+		+			
11				+	+			
12	+					+		
13		+			+			
14			+					+
15				+			+	

Послідовність виконання лабораторного практикуму для варіанту АПК

1) для варіанту **АПК** намалювати принципову схему підключень відповідно до варіанту для виконання лабораторного практикуму та у відповідності зі структурною і функціональною схемою (рис. 8.1, рис. 3.4.2);
2) створити свій директорій для файлів проекту;
3) відкрити інтегроване середовище розробки **PCWH**;
4) створити проект у інтегрованому середовищі розробки **PCWH**;
5) створити алгоритм програми роботи з ЦАП по інтерфейсу SPI;
6) написати програму мовою програмування **C**, що реалізує створений алгоритм, відповідно до варіанту для виконання лабораторного практикуму;

7) скомпілювати програму, одержати бінарні файли з розширеннями ***.HEX** і ***.COF** (файли з розширеннями ***.HEX** і ***.COF** створюються підчас компіляції програми);
8) завантажити бінарний файл з розширенням ***.HEX** у пам'ять програм мікроконтролера програмою **PICkit.exe**;
9) на **АПК** зкумутувати схему підключень. Підключити потенціометр, вольтметр АЦП і ЦАП відповідно до варіанту для виконання лабораторного практикуму у відповідності з рис. 3.4.1, рис. 3.4.2;

Структурна схема підключень виглядає у такий спосіб (рис. 3.4.1):

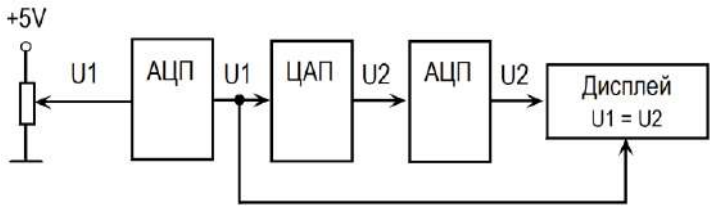


Рисунок 3.4.1 – Структурна схема підключень

Функціональна схема підключень виглядає у такий спосіб (рис. 3.4.2):

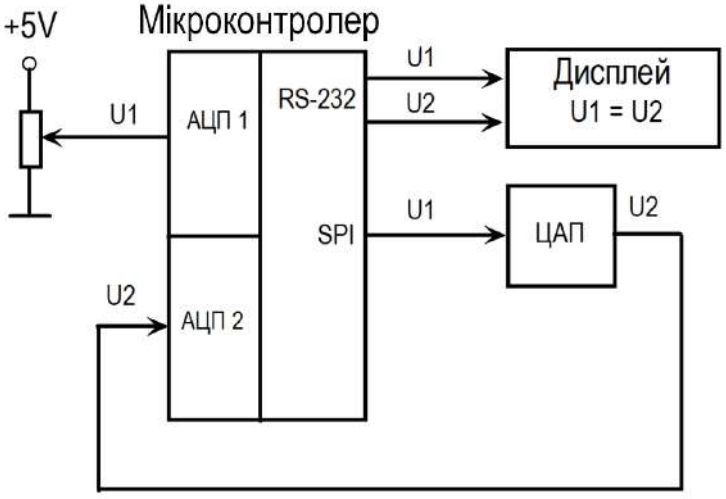


Рисунок 3.4.2 – Функціональна схема підключень

10) виконати програму.

Послідовність виконання лабораторного практикуму для варіанту «Proteus»

1) для середовища моделювання **«Proteus»** використовувати готову схему;
2) створити свій директорій для файлів проекту;
3) відкрити інтегроване середовище розробки **PCWH**;
4) створити проект у інтегрованому середовищі розробки **PCWH**;
5) створити алгоритм програми роботи з ЦАП по інтерфейсу SPI;
6) написати програму мовою програмування **C**, що реалізує створений алгоритм, відповідно до варіанту для виконання лабораторного практикуму;
7) скомпілювати програму, одержати бінарні файли з розширеннями ***.HEX** і ***.COF** (файли з розширеннями ***.HEX** і ***.COF** створюються підчас компіляції програми);
8) відкрити середовище моделювання **«Proteus»**;
9) у папці **«Proteus_students»** вибрати папку лабораторного практикуму **«DAC»**;
10) відкрити файл проекту з розширенням ***.DSN**;
11) у середовищі моделювання **«Proteus»** змінити схему підключень відповідно до варіанту для виконання лабораторного практикуму;
11) завантажити заздалегідь скомпільований бінарний файл з розширенням ***.COF** або ***.HEX** у мікроконтролер (файли з розширеннями ***.COF** і ***.HEX** створюються підчас компіляції програми);
12) у середовищі моделювання **«Proteus»** виконати програму.

Послідовність виконання лабораторного практикуму для варіанту «Proteus» з використанням шаблону програми

1) для середовища моделювання **«Proteus»** використовувати готову схему;
2) відкрити папку **«Proteus_students»**;
3) відкрити інтегроване середовище розробки **PCWH**;
4) у папці **«Proteus_students»** вибрати папку лабораторного практикуму **«DAC»**;
5) відкрити файл проекту з розширенням ***.pjt**;
6) у редакторі **IDE PCWH** відкрити шаблон файлу програми з розширенням ***.c**;
7) відкрити середовище моделювання **«Proteus»**;
8) у папці **«Proteus_students»** вибрати папку лабораторного практикуму **«DAC»**;
9) відкрити файл проекту з розширенням ***.DSN**;
10) у середовищі моделювання **«Proteus»** змінити схему підключень відповідно до варіанту для виконання лабораторного практикуму;
11) створити алгоритм програми роботи з ЦАП по інтерфейсу SPI;
12) у інтегрованому середовищі розробки **PCWH**, використовуючи шаблон програми самостійно написати програму мовою програмування **C**, що реалізує створений алгоритм, відповідно до варіанту для виконання лабораторного практикуму;
13) скомпілювати програму, одержати бінарні файли з розширеннями ***.HEX** і ***.COF** (файли з розширеннями ***.HEX** і ***.COF** створюються підчас компіляції програми);
14) у середовищі моделювання **«Proteus»** виконати програму.

Примітка: *файл демонстрації виконання лабораторного практикуму* ***«DAC»*** *розташований на сайті* http://pksm.kntu.kr.ua/SCME.html.

Приклад створення проекту у інтегрованому середовищі розробки PCWH

Створення проекту у інтегрованому середовищі розробки PCWH за допомогою майстра PIC Wizard

Для запуску майстра **PIC Wizard** слід виконати відповідну команду меню **Project**, а потім вказати розміщення і ім'я головного файлу проекту.

В результаті відкриється вікно майстра, що складається з розділів з параметрами проекту (рис. 3.4.6).

Зовнішній вигляд вікна майстра може відрізнятися в залежності від версії компілятора **CCS-PICC** і обраного типу мікроконтролера.

1. Запустити **IDE PCWH**, натиснути кнопку **Projects** майстра **PIC Wizard** (рис. 3.4.3):

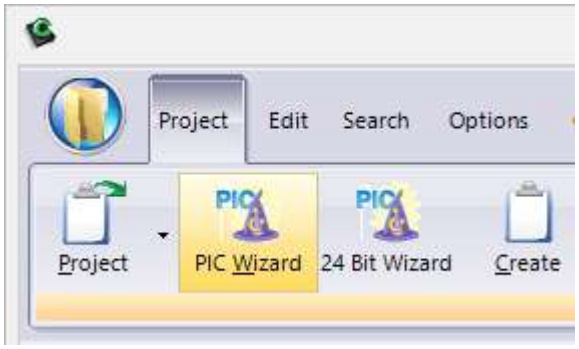


Рисунок 3.4.3 – Процес створення проекту у **IDE PCWH** за допомогою майстра **PIC Wizard**

або натиснути кнопку у вигляді відкритої папки 

2. Вибрати: **New › Project Wizard** (рис. 3.4.4):

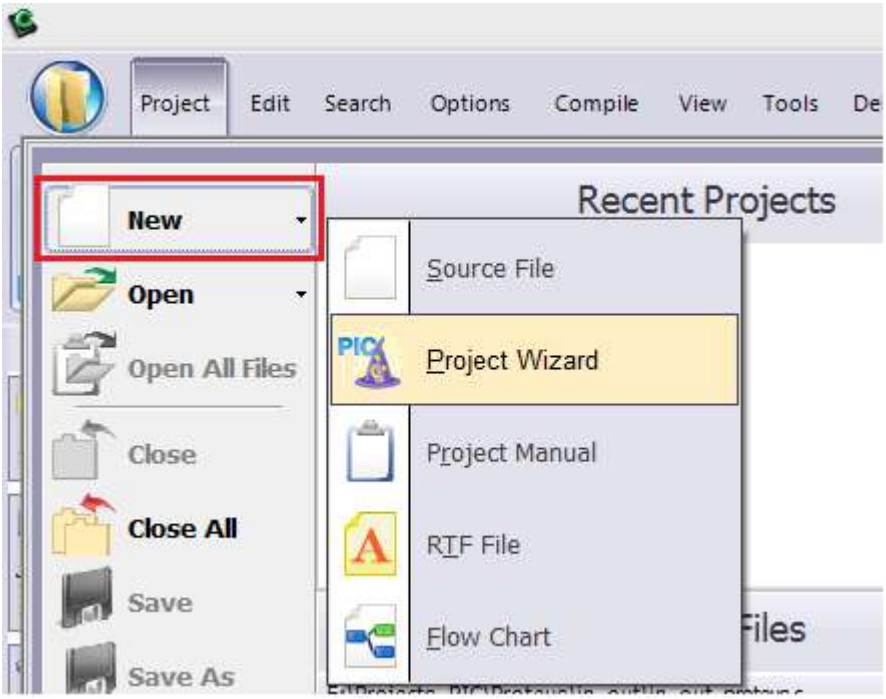


Рисунок 3.4.4 – Процес створення проекту у **IDE PCWH** за допомогою майстра **PIC Wizard**

3. Створити або вибрати свій директорій і записати у нього проект під будь-яким іменем латинським шрифтом, наприклад: «**dac.pjt**» (рис 4.5):

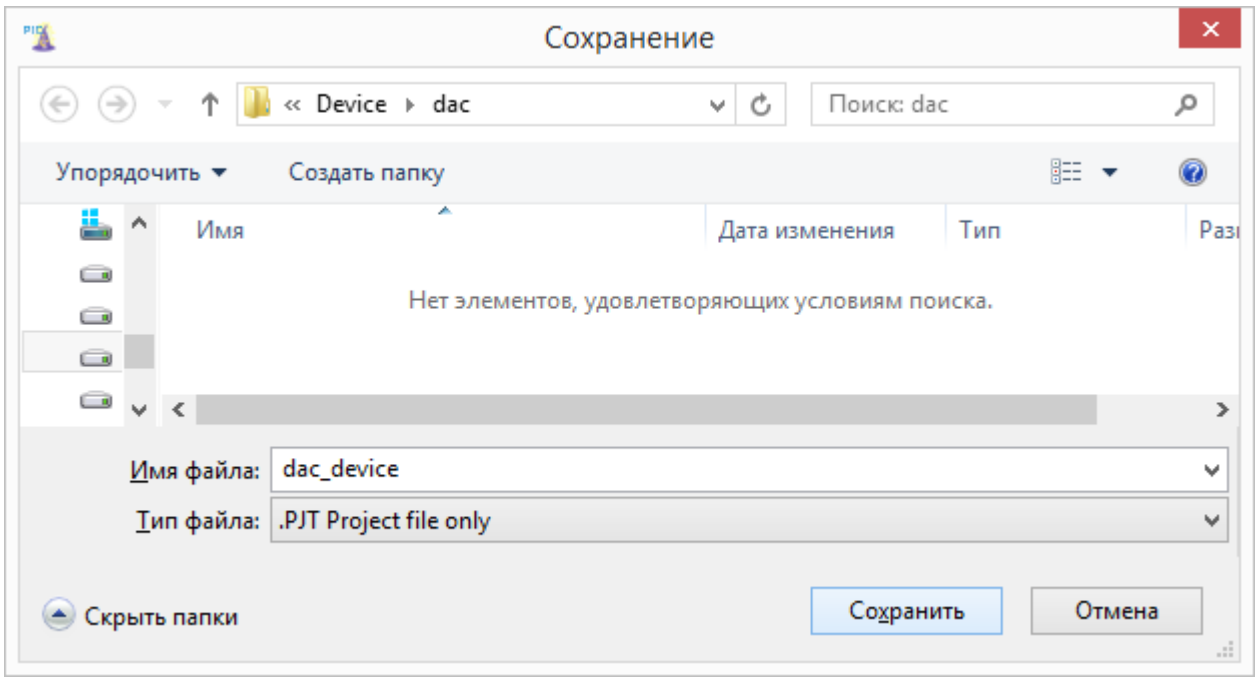


Рисунок 3.4.5 – Процес створення проекту у **IDE PCWH** за допомогою майстра **PIC Wizard**

У розділі **General** (рис 3.4.6) вибирається цільовий мікроконтролер (список **Device**, що розкривається), його робоча частота (поле **Oscillator Frequency**), а також настроюються загальні параметри, на зразок розрядів запобігання, типу джерела системної синхронізації, порогової напруги для скидання та ін. Для перегляду програмного коду, який буде згенерований і доданий у вихідний файл відповідно до параметрів на поточній вкладці, слід вибрати вкладку **Code**.

4. У розділі **General › Device** вибрати тип мікроконтролера, наприклад **PIC18F252**.

5. У розділі **General › Fuses** вибрати тип генератора **High speed Osc (> 4mhz for PCM/PCH) (>10mhz for PCD)**:

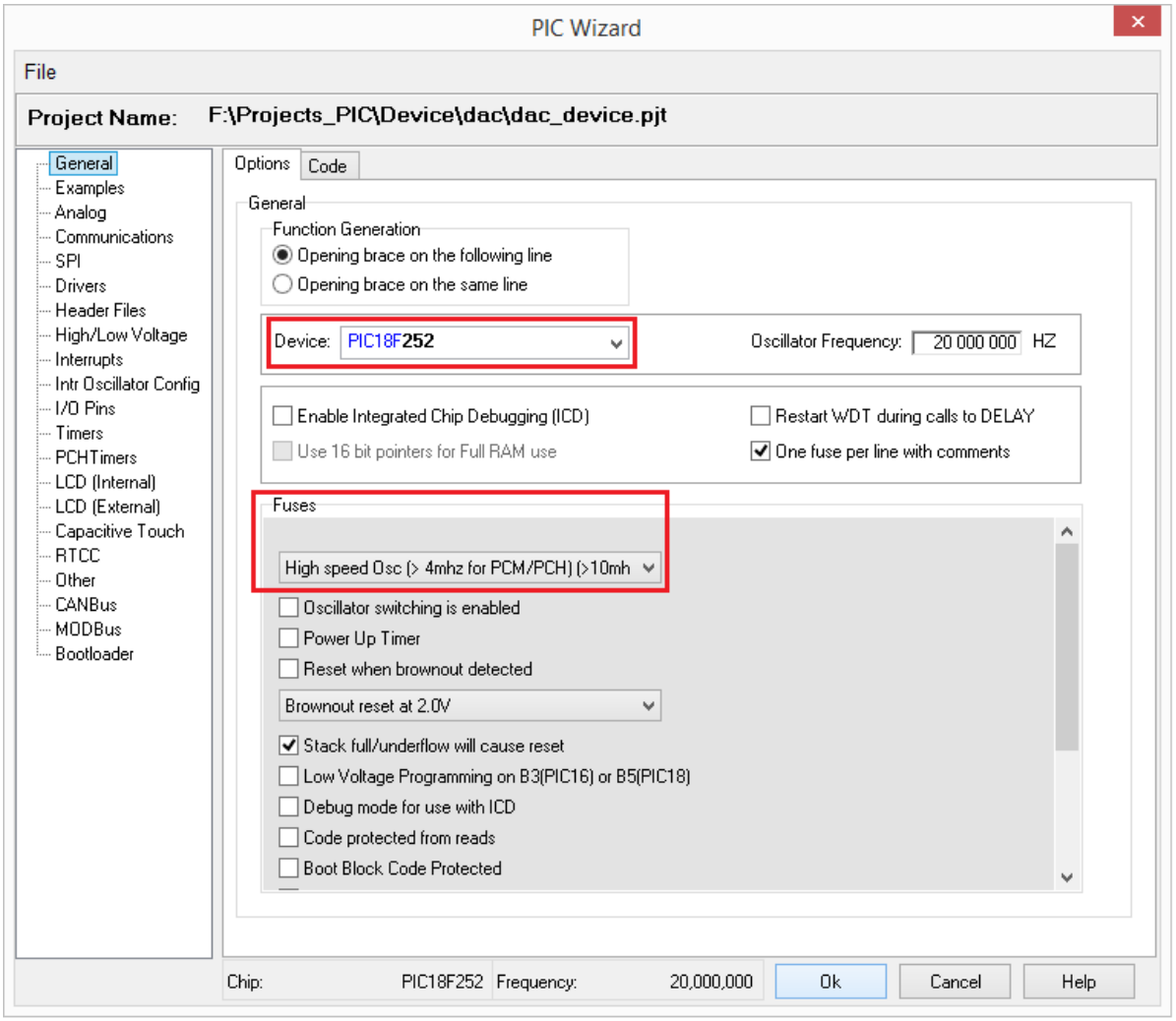


Рисунок 3.4.6 – Розділ **General** майстра **PIC Wizard**

У розділі **Communications** (рис. 3.4.7) налаштовуються параметри введення/виведення для інтерфейсів **RS-232** і **I²C** (швидкість обміну даними, відповідні лінії портів введення/виведення, перевірка помилок, режим **Master/Slave** і т.д.), а також активізується/відключається апаратний порт **PSP**;

6. У розділі **Communications** встановити мітку у полі **RS-232**:

- ☑ **Use RS-232**;

вказати:

- **Transmit:** **PIN C6** (передача);
- **Receive:** **PIN C7** (прийом).

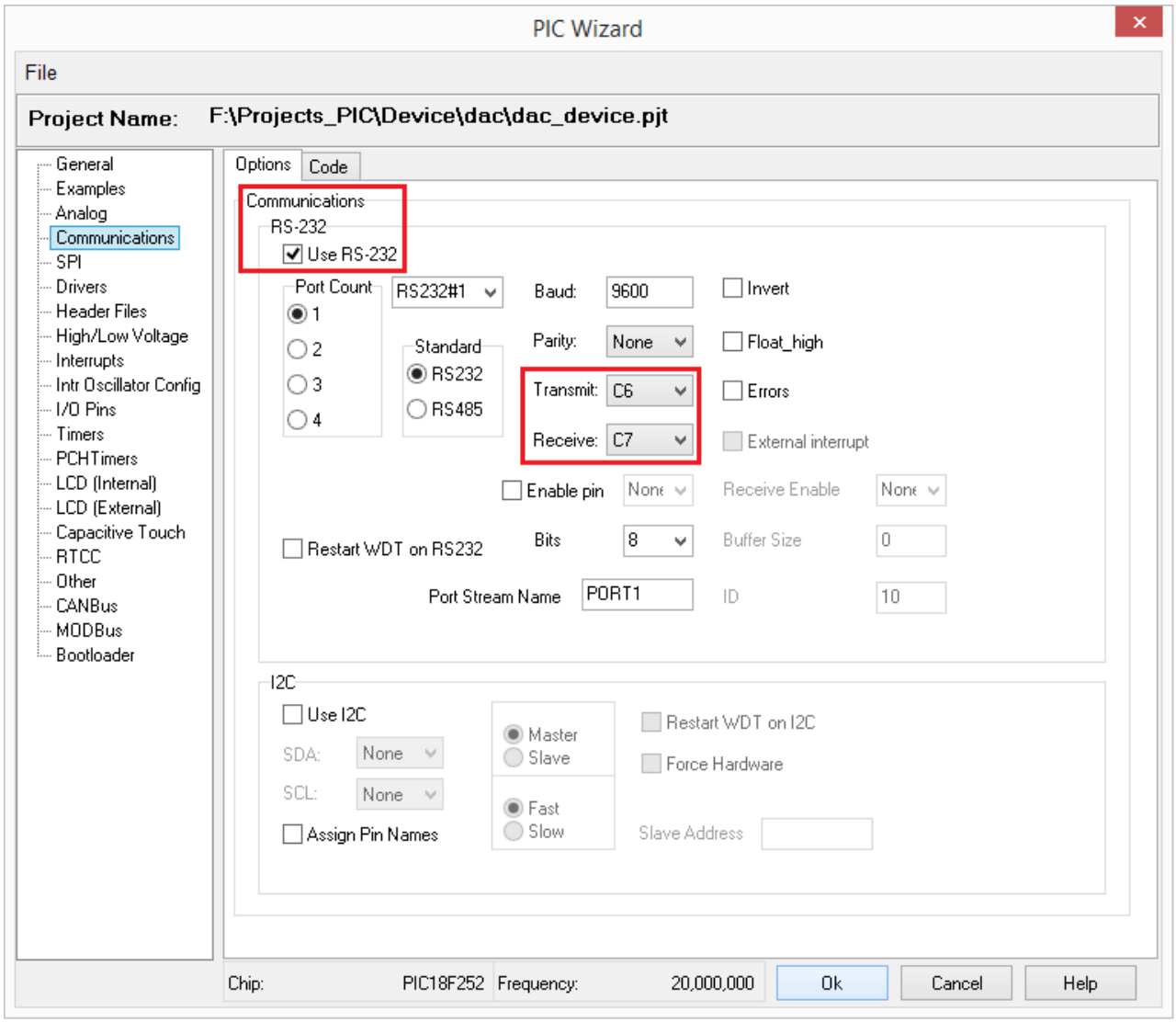


Рисунок 3.4.7 – Розділ **Communications** майстра **PIC Wizard**

Розділ **Analog** (рис. 3.4.8) служить для налаштування вбудованого АЦП (конфігурація аналогових входів, частота і розрядність перетворення).

7. У розділі **Analog** встановити:

   у полі **Analog Input:**

   - ◉ **A0 A1 A2 A3.**

   Вказати:

   розрядність АЦП: **8 (0-255)** або **10 (0-1023)** розрядів:

   - **Units:** **0-255 / 0-1023**
   - **Clock:** **4 us**

Конфігурування можна зробити з **IDE PCWH** при створенні проекту:

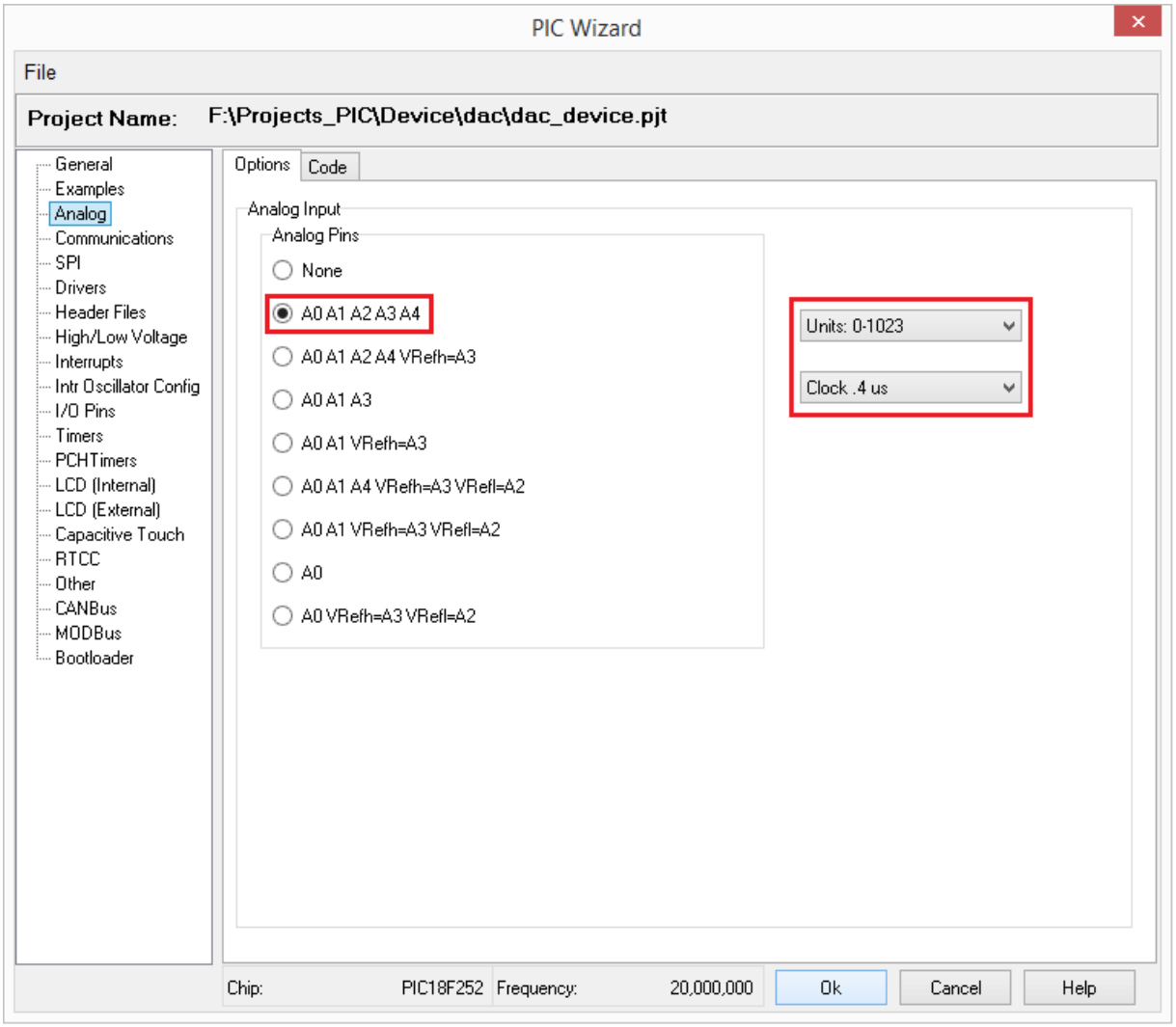


Рисунок 3.4.8 – Конфігурування АЦП із **IDE PCWH** при створенні проекту у розділі **Analog** майстра **PIC Wizard**

Буде згенерований і доданий програмний код у вихідний файл відповідно до параметрів на поточній вкладці.

Для перегляду програмного коду, слід вибрати вкладку **Code**.

8. Натиснути кнопку **OK**.

Буде згенерований наступний код (рис. 3.4.9):

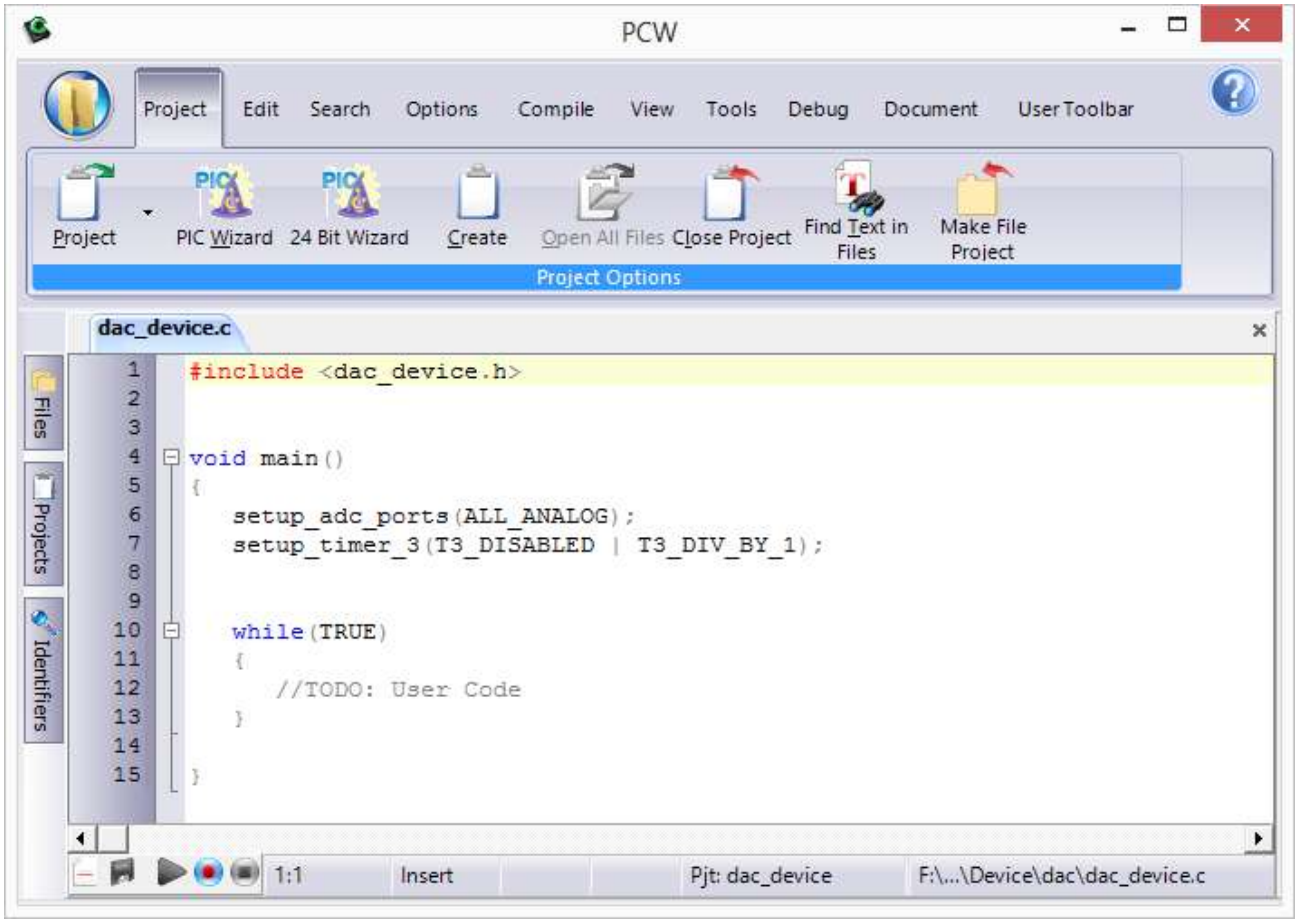


Рисунок 3.4.9 – Згенерований майстром **PIC Wizard** програмний код

Лістинг 3.4.1 – Приклад програмного коду, згенерованого майстром **PIC Wizard**

dac_device.c

```
#include <dac_device.h>
void main()
{
        setup_adc_ports(ALL_ANALOG);
        setup_timer_3(T3_DISABLED | T3_DIV_BY_1);\
        while(TRUE)
{
//TODO: User Code }}
```

dac_device.h

```
#include <18F252.h>
#device adc=10

#FUSES NOWDT          //No Watch Dog Timer
#FUSES WDT128         //Watch Dog Timer uses 1:128 Postscale
#FUSES HS             //High speed Osc (> 4mhz for PCM/PCH) (>10mhz for
PCD)
#FUSES NOBROWNOUT     //No brownout reset
#FUSES NOLVP          //No low voltage prgming, B3(PIC16)
                      //or B5(PIC18) used for I/O

#use delay(clock=20000000)
#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8,stream=PORT1)
```

Проект готовий, можна приступати до написання програми.

Пояснення до виконання лабораторного практикуму «DAC»

Цифро-аналоговий перетворювач MCP4921

MCP4921 пристрій являє собою одноканальний 12-бітний ЦАП, який використовує зовнішнє джерело опорної напруги. Цей пристрій забезпечує високу точність і низьке енергоспоживання і доступні в різних пакетах. Зв'язок з пристроєм здійснюється за допомогою простого послідовного інтерфейсу SPI. **MCP4921** пристрій є частиною сімейства, які використовують зовнішнє джерело опорної напруги (V_{REF}).

Ці пристрої забезпечують дуже високу точність і низький рівень шумів, і придатні для споживчих і промислових додатків. Низький рівень споживання енергії та малі варіанти пакета добре підходять для багатьох портативних і малопотужних додатків. Завдяки своїм компактним розмірам і малій потужності споживання, **ЦАП MCP4921** забезпечує значні переваги в просторі обмежених випадках, коли мінімальна потужність споживання має вирішальне значення.

Режими програмного забезпечення або апаратного виключення нададуть додаткову економію електроенергії, зменшення струму в режимі очікування до 0,5 мкА (тип.) при будь-якому виборі. Зв'язок з **MCP4921** здійснюється через 3-провідний SPI протокол.

Внутрішня структура **ЦАП MCP 4921** представлена на рис. 3.4.10.

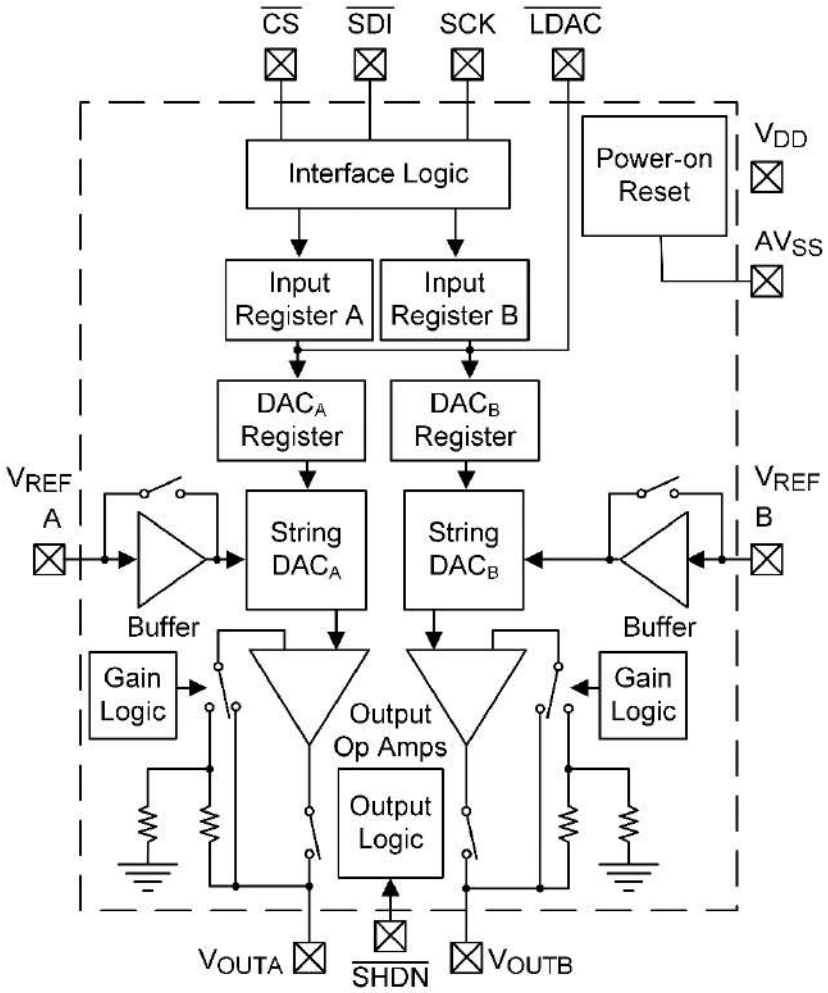


Рисунок 3.4.10 – Внутрішня структура **ЦАП MCP 4921**

Відмінні особливості:

- розрядність ЦАП: 12 розрядів;
- диференціальна нелінійність: ± 0,2 молодшого розряду (тип);
- інтегральна нелінійність: ± 2 молодших розряду (тип);
- виходи Rail-to-Rail;
- SPI-інтерфейс з частотою до 20 МГц;
- синхронні засувки даних на обох ЦАП;
- малий час встановлення: 4,5 мкс;
- вибір вихідного коефіцієнта посилення 1x або 2x;
- вхід для зовнішнього джерела опорної напруги V_{REF};
- діапазон напруги живлення: 2,7В ... 5,5В;
- кількість виводів: 8;
- тип входу: послідовний;

- температурний діапазон: –40 ° C ... + 125 ° C;
- корпуси: MSOP–8 (MS) і DIP–8 (P).

Розташування виводів **ЦАП MCP 4921** представлено на рис. 3.4.11.

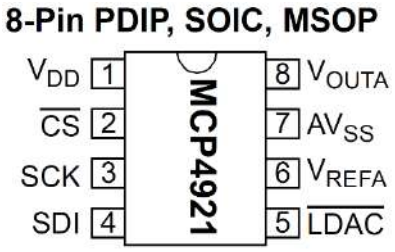


Рисунок 3.4.11 – ЦАП **MCP 4921**

Підключення **ЦАП** до мікроконтролера по інтерфейсу **SPI** представлено на рис. 3.4.12.

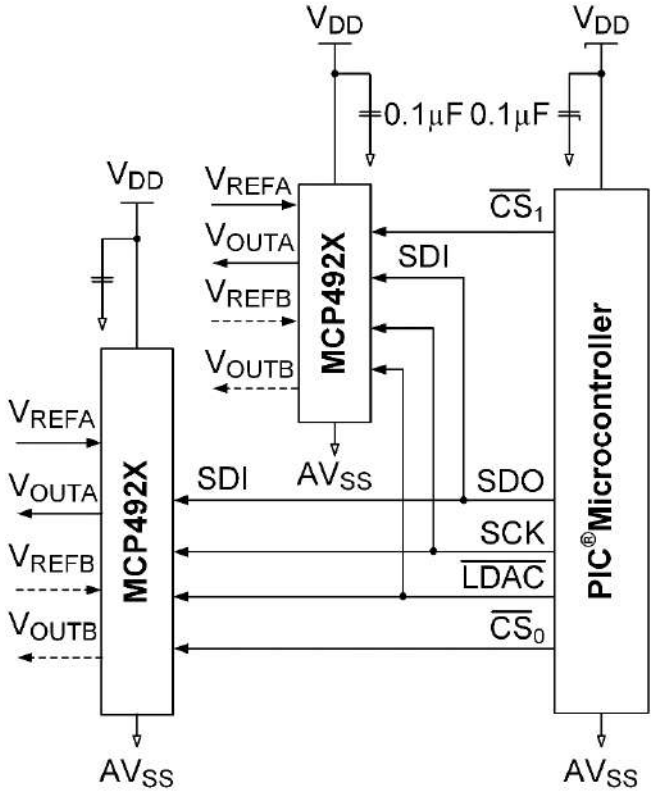


Рисунок 3.4.12 – Підключення **ЦАП** до мікроконтролера по інтерфейсу **SPI**

Таблиця 3.4.2 – Призначення виводів ЦАП **MCP 4921**

MCP4921 Pin No	**Symbol**	**Function**
1	V_{DD}	**Positive Power Supply Input (2.7V to 5.5V)** Споживана потужність живлення по відношенню до AV_{SS} може варіюватися від 2,7 до 5.5. Розв'язуючий конденсатор на V_{DD} рекомендується для досягнення максимальної продуктивності.
2	$\overline{CS}$	**Chip Select Input** Вхід вибору чіпу, який вимагає активний низький сигнал для включення послідовних годин і функцій даних
3	SCK	**Serial Clock Input** SPI сумісний послідовний ввід
4	SDI	**Serial Data Input** SPI сумісне послідовне введення даних
5	$\overline{LDAC}$	**Syncronization input used to transfer DAC settings from serial latches to the output latches** Засувка входу ЦАП. Передача введення регістрів засувки ЦАП при низькому логічному рівні
6	$VREF_A$	**DAC_A Voltage Input (AV_{SS} to V_{DD})** Аналоговий сигнал на цих виводах використовується для установки опорної напруги на рядок ЦАП. Вхідний сигнал може варіюватися в діапазоні від AV_{SS} до V_{DD}
7	AV_{SS}	**Analog ground** Аналогове заземлення
8	V_{OUTA}	**DAC_A Output** ЦАП підсилює сигнал на виводах в діапазоні AV_{SS}-V_{DD}

Приклади реалізації програми лабораторного практикуму «DAC»

Конфігурування мікроконтролера для роботи із ЦАП по інтерфейсу SPI можна зробити з **IDE PCWH** при створенні проекту або у функції ініціалізації ввести рядок:

```
setup_spi(SPI_MASTER|SPI_L_TO_H|SPI_CLK_DIV_4);
```

Проте, написання драйвера інтерфейсу SPI для роботи у двобайтовому режимі є одним із завдань лабораторного практикуму.

Приклад зчитування даних з АЦП з каналу 1 і передача даних у ЦАП:

```
float volt_in;                      //вихід потенціометра
float volt_out;                     //вихід ЦАП
int16 dac_hex;                      //вхід ЦАП

//====== послати значення у ЦАП
dac_hex = volt_in/DAC_CVANT;        //двійковий еквівалент напруги
SPI_send(dac_hex);                  //послати дані у ЦАП
```

Змінна **DAC_CVANT** є ціною розподілу шкали ЦАП і обчислюється як частка від ділення U_{REF} на розрядність АЦП, виражену у двійкових одиницях:

```
//====== опорна напруга = напруга живлення
const float REF_VDD = 5.00;
const float DAC_12 = 4095;

//====== розподіл шкали: 5В/шкала (розрядність ЦАП)
const float DAC_CVANT = REF_VDD/DAC_12;
```

Визначення двійкового значення величини `dac_hex` напруги $U_{ВИХ}$, що встановлюється:

```
dac_hex = volt_in/DAC_CVANT;
```

Відлік шкали перетворення проводиться відносно опорної напруги U_{REF}. Наприклад, якщо U_{REF} = 10В, а розрядність ЦАП = 8 розрядів (255 рівнів), то ціна (вага) одного розряду складе 10В/255 = 0.039В або 39 мВ.

Зазвичай U_{REF} встановлюють рівним напрузі живлення ЦАП, тобто:

$U_{ref} = v_{dd} = 5в.$

Примітка: після виведення значень на LCD необхідно ввести затримку часу індикації: `delay_ms(100);`

Послідовність побудови інтерфейсу SPI:

1) встановити виводи мікроконтролера, що беруть участь в роботі з SPI у режим передачі даних;
2) визначити інтерфейс:

```
#define CLK PIN_C3
#define CS PIN_C4
#define SDO PIN_C5
```

3) об`явити змінні для лічильника розрядів і затримки. Затримку прийняти рівною 25 мкс;
4) у пакет даних, що передається, вставити біт режиму `0x2000`;
5) перед початком передачі даних встановити сигнал CS у "0";
6) почати передачу пакета даних у циклі зі зсувом вліво. При передачі послідовно виставляти дані, затримки і строб відповідно до тимчасової діаграми на рис. 3.4.13.

```
//=== виставити біт
if(bit_test(val,c)){                    //c - номер біту 0-15
       output_high(SDO);         //SDO = 1
}else{
       output_low(SDO);          //SDO = 0
}
```

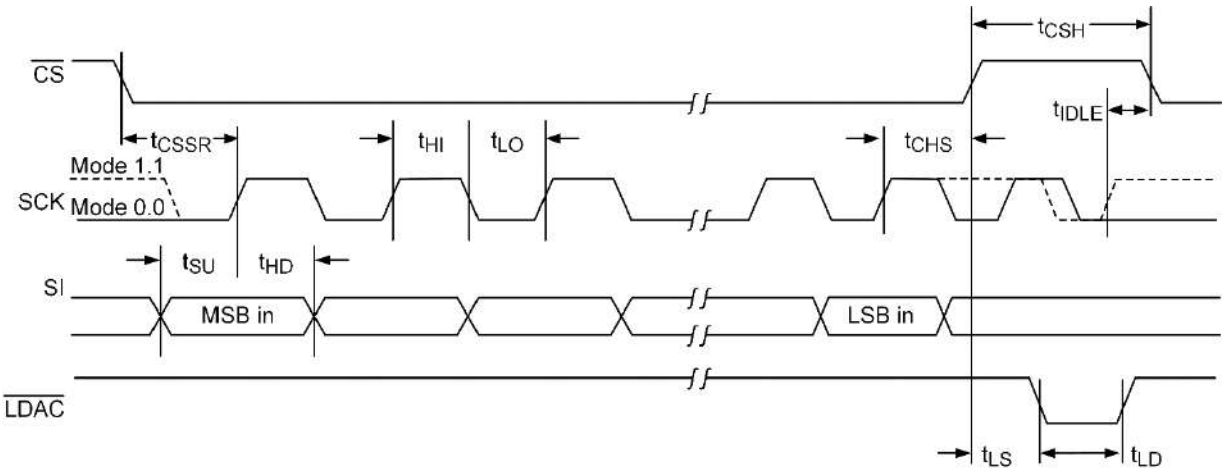


Рисунок 3.4.13 – Тимчасові діаграми роботи інтерфейсу SPI

7) після відправлення останнього біту встановити сигнал CS у "1" і виконати затримку.

На виході ЦАП з'явиться рівень напруги пропорційний заданому двійковому значенню.

Прототип функції:

```
void SPI_send(int16 val);
```

Процес пересилання пакета даних від мікроконтролера до ЦАП по інтерфейсу SPI представлений на рис. 3.4.14.

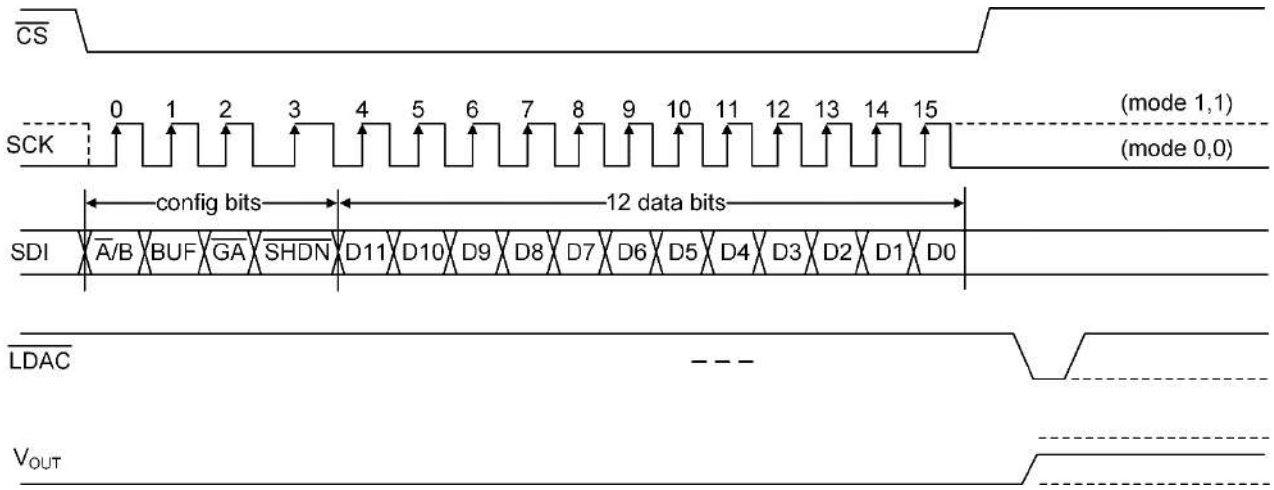


Рисунок 3.4.14 – Процес пересилання пакета даних від мікроконтролера до ЦАП по інтерфейсу SPI

Формат пакета даних:

Upper Half:							
W-x	W-x	W-x	W-0	W-x	W-x	W-x	W-x
$\overline{A}$/B	BUF	$\overline{GA}$	$\overline{SHDN}$	D11	D10	D9	D8
bit 15							bit 8

Lower Half:							
W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x
D7	D6	D5	D4	D3	D2	D1	D0
bit 7							bit 0

bit 15 **A/B:** DAC_A or DAC_B Select bit

1 = Write to DAC_B

0 = Write to DAC_A

bit 14 **BUF:** V_{REF} Input Buffer Control bit

1 = Buffered

0 = Unbuffered

bit 13 **GA:** Output Gain Select bit

1 = 1x ($V_{OUT} = V_{REF} * D/4096$)

0 = 2x ($V_{OUT} = 2 * V_{REF} * D/4096$)

bit 12 **SHDN:** Output Power Down Control bit

1 = Output Power Down Control bit

0 = Output buffer disabled, Output is high impedance

bit 11-0 **D11:D0:** DAC Data bits

12 bit number "D" which sets the output value. Contains a value between 0 and 4095.

Таблиця 4.3 – Тимчасові характеристики роботи інтерфейсу SPI

Electrical Specifications: Unless otherwise indicated, V_{DD}= 2.7V – 5.5V, TA= -40 to +125°C. Typical values are at +25°C.						
Parameters	**Sym**	**Min**	**Typ**	**Max**	**Units**	**Conditions**
Schmitt Trigger High-Level Input Voltage (All digital input pins)	V_{IH}	0.7 V_{DD}	—	—	V	
Schmitt Trigger Low-Level Input Voltage (All digital input pins)	V_{IL}	—	—	0.2 V_{DD}	V	
Hysteresis of Schmitt Trigger Inputs	V_{HYS}	—	0.05 V_{DD}	—		
Input Leakage Current	$I_{LEAKAGE}$	-1	—	1	µA	$\overline{SHDN}$ = $\overline{LDAC}$ = $\overline{CS}$ = SDI = SCK + V_{REF} = V_{DD} or AV_{SS}
Digital Pin Capacitance (All inputs/outputs)	C_{IN}, C_{OUT}	—	10	—	pF	V_{DD} = 5.0V, T_A = +25°C, f_{CLK} = 1 MHz
Clock Frequency	F_{CLK}	—	—	20	MHz	T_A = +25°C
Clock High Time	t_{HI}	15	—	—	ns	
Clock Low Time	t_{LO}	15	—	—	ns	
$\overline{CS}$ Fall to First Rising CLK Edge	t_{CSSR}	40	—	—	ns	Applies only when $\overline{CS}$ falls with CLK high.
Data Input Setup Time	t_{SU}	15	—	—	ns	
Data Input Hold Time	t_{HD}	10	—	—	ns	
SCK Rise to $\overline{CS}$ Rise Hold Time	t_{CHS}	15	—	—	ns	

Electrical Specifications: Unless otherwise indicated, V_{DD}= 2.7V – 5.5V, TA= -40 to +125°C. Typical values are at +25°C.						
Parameters	**Sym**	**Min**	**Typ**	**Max**	**Units**	**Conditions**
$\overline{CS}$ High Time	t_{CSH}	15	—	—	ns	
$\overline{LDAC}$ Pulse Width	t_{LD}	100	—	—	ns	
$\overline{LDAC}$ Setup Time	t_{LS}	40	—	—	ns	
SCK Idle Time before $\overline{CS}$ Fall	t_{IDLE}	40	—	—	ns	

Приклади виконання лабораторного практикуму «DAC»

Завдання для лабораторного практикуму «*DAC*»

Програма має вигляд і структуру відповідно до лістингів 3.4.2, 3.4.3:

- лістинг 3.4.2 – для варіанту **АПК;**
- лістинг 3.4.3 – для варіанту **«Proteus»**.

Завдання для виконання лабораторного практикуму **«DAC»** представлене у таблиці 3.4.2.

Таблиця 3.4.2 – Завдання для виконання лабораторного практикуму **«DAC»**

Канал АЦП							
Потенціометр				**ЦАП**			
A0	A1	A2	A3	A0	A1	A2	A3
	+					+	

Принципова схема підключень для варіанту АПК

Принципова схема підключень для варіанту **АПК**, для виконання лабораторного практикуму **«DAC»** виглядає у такий спосіб (рис. 3.4.15):

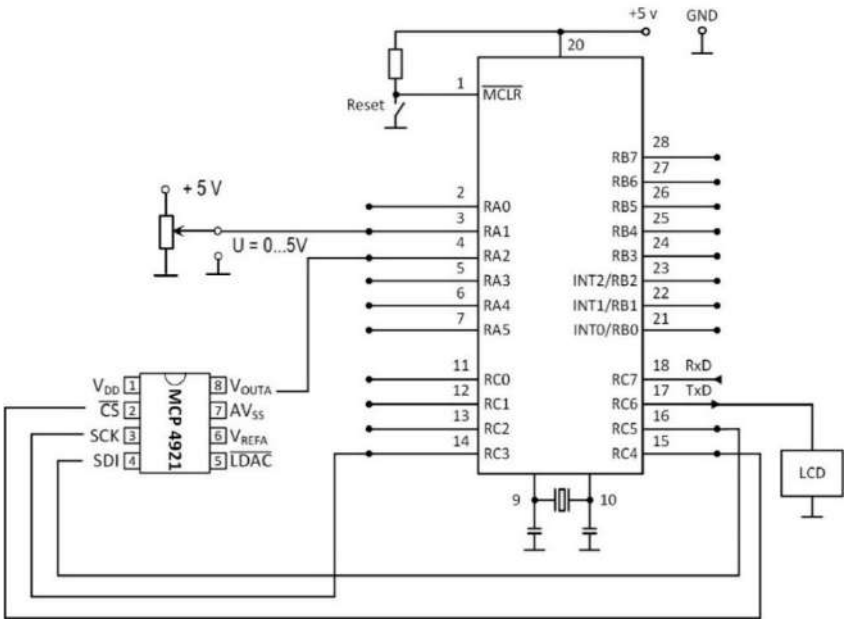


Рисунок 3.4.15 – Принципова схема підключень для лабораторного практикуму **«DAC»** для варіанту **АПК**

Принципова схема підключень для варіанту «Proteus»

Принципова схема для варіанту **«Proteus»**, для виконання лабораторного практикуму **«DAC»** виглядає у такий спосіб (рис. 3.4.16):

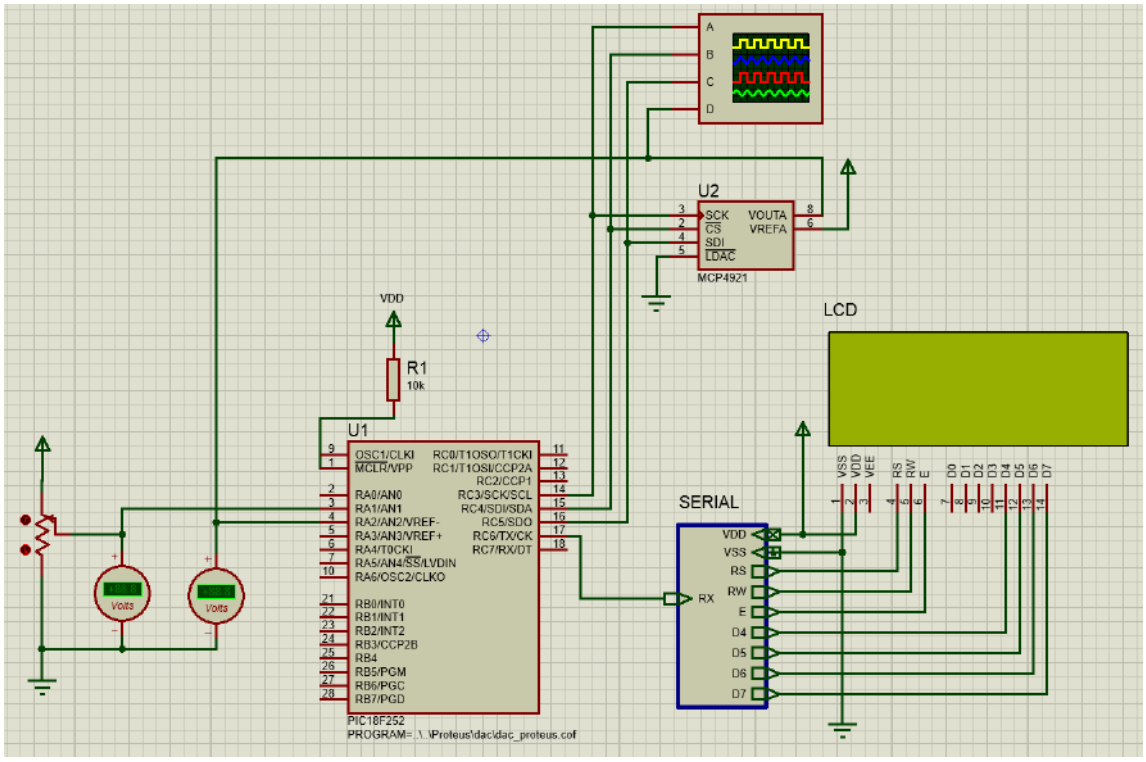


Рисунок 3.4.16 – Принципова схема підключень лабораторного практикуму **«DAC»** для варіанту **«Proteus»**

Результат виконання лабораторного практикуму «DAC» для варіанту АПК

Лістинг 3.4.2 – Програма реалізації алгоритму лабораторного практикуму **«DAC»** для варіанту **АПК**

```
//###################################################################
// Лабораторний практикум «DAC»
// file: dac_device.cРобота з ЦАП
// Copyright (c) Docent Smirnov V.V., Docent Smirnova N.V / CNTU
//###################################################################

#include < dac_device.h>
#include <swc_LIB.h>
#include <swc_LCD.h>                        //драйвер LCD дисплея


//====== визначити інтерфейс
#define CLK PIN_C3
#define CS PIN_C4
#define SDO PIN_C5

//====== визначення DAC
//====== опорна напруга = напруга живлення
//const float REF_VDD = 5.00;
const float DAC_12 = 4095;
//====== розподіл шкали: 5В / шкала (розрядність ЦАП)
//=== 12 розрядів
const float DAC_CVANT = REF_VDD/DAC_12;
//обчислення HEX: HEX = VOLT_TO_SEND/DAC_CVANT

//=========================================================================
// Ініціалізація PIC
//=========================================================================
void init()
{
       setup_adc_ports(ALL_ANALOG);
       setup_adc(ADC_CLOCK_INTERNAL);
       setup_spi(SPI_SS_DISABLED);
       setup_wdt(WDT_OFF);
       setup_timer_0(RTCC_INTERNAL);
       setup_timer_1(T1_DISABLED);
       setup_timer_2(T2_DISABLED,0,1);
       enable_interrupts(GLOBAL);
}

//=========================================================================
// Ініціалізація SPI
//=========================================================================
void SPI_init()
{
       int8 tris;
       //=== конфігурувати порт C не чіпаючи установок RS-232
       tris = get_tris_c();
       bit_clear(tris,3);                   //C3 = CLK
       bit_clear(tris,4);                   //C4 = CS
       bit_clear(tris,5);                   //C5 = SDO
       set_tris_c(tris);
```

```
	//====== скидання SPI
	//=== CS=1
	output_high(CS);
	//=== зняти строб
	output_low(CLK);			//CLK = 0
}

//=============================================================================
// Передати дані у ЦАП по SPI
//=============================================================================
void SPI_send(int16 val){
	int8 c;					//розрядність
	int8 delay;				//затримка
	delay = 25;
	//====== біт режиму
	val = val|0x2000;
	//====== виведення даних
	c = 16;
	//====== CS = 0
	output_low(CS);
	for(;;){
		//=== виставити біт
		if(bit_test(val,c)){
			output_high(SDO);		//SDO = 1
		}else{
			output_low(SDO);		//SDO = 0
		}

		//=== виставити строб
		delay_us(delay);
		//=== CLK = 1
		output_high(CLK);
		delay_us(delay);
		//=== зняти строб
		output_low(CLK);			//CLK = 0
		//=== зняти біт даних
		output_low(SDO);			//SDO = 0
		c--;
		if(c == 0){
			break;
		}
	}

	//====== виведення значення
	//=== CS=1
	delay_us(delay);
	output_high(CS);
	delay_us(delay);
}

//=============================================================================
// Main
//=============================================================================
void main()
{
	float volt_in;			//вихід потенциометра
	float volt_out;			//вихід ЦАП
	int16 dac_hex;			//вхід ЦАП
	//
	init();
	lcd_init();
	SPI_init();
```

```
        //====== титли
        lcd_running_string_begin(4);
        printf("Лабораторний практикум. Кафедра ПКСМ");
        lcd_running_string_end();
        lcd_running_string_start();

        //====== титли для 1 частини
        lcd_goto(1,1);
        printf("Вольти In:");
        lcd_goto(2,1);
        printf("Вольти Out:");
        lcd_goto(3,1);
        printf("Смірнов В.В.");

        //====== головний цикл
        for(;;){
               //====== отримати напругу з потенціометру
               volt_in = get_ADC_volt(1);
               //====== послати значення у ЦАП
               dac_hex = volt_in/DAC_CVANT;
               SPI_send(dac_hex);
               //====== отримати напругу з ЦАП
               volt_out = get_ADC_volt(2);

               //====== вивести результат
               lcd_goto(1,13);
               printf("%f",volt_in);
               lcd_goto(2,13);
               printf("%f",volt_out);
               delay_ms(200);
        }
}
```

Результат виконання лабораторного практикуму «DAC» для варіанту «Proteus»

Лістинг 3.4.3 – Програма реалізації алгоритму лабораторного практикуму **«DAC»** для варіанту **«Proteus»**

```
//#####################################################################
// Лабораторний практикум «DAC»
// file: dac_proteus.c / робота з ЦАП
// Copyright (c) Docent Smirnov V.V., Docent Smirnova N.V / CNTU
//#####################################################################

#include <dac_proteus.h>
#include <swc_LIB.h>
#include <swc_LCD.h>                        //драйвер LCD дисплея

//====== визначити інтерфейс
#define CLK PIN_C3
#define CS PIN_C4
#define SDO PIN_C5

//====== визначення DAC
//====== опорна напруга = напруга живлення
//const float REF_VDD = 5.00;
const float DAC_12 = 4095;
```

```
//====== розподіл шкали: 5В/шкала (розрядність ЦАП)
//=== 12 разрядов
const float DAC_CVANT = REF_VDD/DAC_12;
// обчислення HEX: HEX = VOLT_TO_SEND/DAC_CVANT

//=============================================================================
// Ініціалізація PIC
//=============================================================================
void init()
{
       setup_adc_ports(ALL_ANALOG);
       setup_adc(ADC_CLOCK_INTERNAL);
       setup_spi(SPI_SS_DISABLED);
       setup_wdt(WDT_OFF);
       setup_timer_0(RTCC_INTERNAL);
       setup_timer_1(T1_DISABLED);
       setup_timer_2(T2_DISABLED,0,1);
       enable_interrupts(GLOBAL);
}
//=============================================================================
// Ініціалізація SPI
//=============================================================================
void SPI_init()
{
       int8 tris;

       //=== конфігурувати порт С не чіпаючи установок RS-232
       tris = get_tris_c();
       bit_clear(tris,3);                          //C3 = CLK
       bit_clear(tris,4);                          //C4 = CS
       bit_clear(tris,5);                          //C5 = SDO
       set_tris_c(tris);

       //====== скидання SPI
       //=== CS=1
       output_high(CS);
       //=== зняти строб
       output_low(CLK);                            //CLK = 0
}
//=============================================================================
// Передати дані у ЦАП по SPI
//=============================================================================
void SPI_send(int16 val){
       /*output_high(SDO);                         //SDO = 1
       output_high(CLK);
       output_high(CS);
       delay_ms(50);
       output_low(SDO);                            //SDO = 0
       output_low(CLK);                            //CLK = 0
       output_low(CS);
       delay_ms(50);*/

       int8 c;                                     //розрядність
       int8 delay;                                 //затримка
       delay = 25;

       //====== біт режиму
       val = val|0x2000;

       //====== виведення даних
       c = 16;
       //====== CS = 0
       output_low(CS);
```

```
        for(;;){
                //=== виставити біт
                if(bit_test(val,c)){
                        output_high(SDO);           //SDO = 1
                }else{
                        output_low(SDO);            //SDO = 0
                }
                //=== виставити строб
                delay_us(delay);
                //=== CLK = 1
                output_high(CLK);
                delay_us(delay);

                //=== зняти строб
                output_low(CLK);                    //CLK = 0
                //=== зняти біт даних
                output_low(SDO);                    //SDO = 0
                c--;
                if(c == 0){
                        break;
                }
        }
        //====== виведення значення
        //=== CS=1
        delay_us(delay);
        output_high(CS);
        delay_us(delay);
}
//============================================================================
// Main
//============================================================================
void main()
{
        float volt_in;                              //вихід потенциометра
        float volt_out;                             //вихід ЦАП
        int16 dac_hex;                              //вхід ЦАП
        //
        init();
        lcd_init();
        SPI_init();

        //====== головний цикл
        for(;;){
                //SPI_send(0x255);
                //====== отримати напругу з потенціометру
                volt_in = get_ADC_volt(1);
                //====== послати значення у ЦАП
                dac_hex = volt_in/DAC_CVANT;
                SPI_send(dac_hex);
                //====== отримати напругу з ЦАП
                volt_out = get_ADC_volt(2);
                //====== вивести результат
                lcd_goto(1,1);
                printf("DAC");
                lcd_goto(2,1);
                printf("docent Smirnov V.V.");
                lcd_goto(3,1);
                printf("Input: %f",volt_in);
                lcd_goto(4,1);
                printf("Output:%f",volt_out);
                delay_ms(200);
        }
}
```

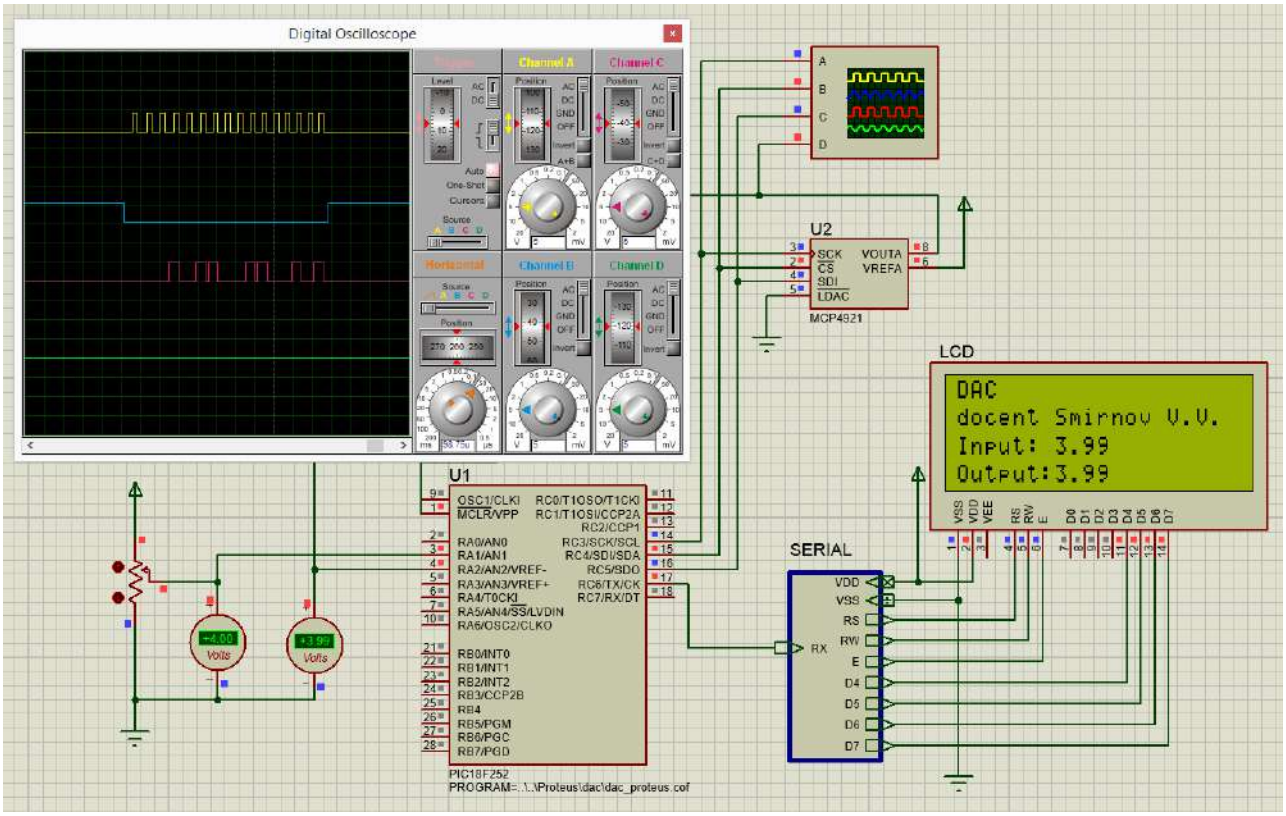


Рисунок 3.4.17 – Результат виконання лабораторного практикуму **«DAC»** для варіанту **«Proteus»**

Контрольні питання

1) призначення ЦАП і методи перетворення;
2) статичні і динамічні характеристики ЦАП **MCP4921** (SPI);
3) призначення ліній інтерфейсу SPI;
4) структура пакета даних ЦАП **MCP 4921** (SPI);
5) як визначається величина двійкового еквівалента необхідної вихідної напруги ЦАП?

ГЛОСАРІЙ

Acquisition Time - час заряду конденсатора АЦП до рівня вхідного сигналу. На цей параметр впливає опір джерела сигналу. Чим більше опір, тим більший час необхідний для заряду конденсатора. При установці біту GO/DONE аналоговий сигнал відключається і починається процес аналого-цифрового перетворення.

ADC (Analog To Digital Converter) - АЦП (аналогово-цифровий перетворювач). Цей пристрій конструктивно може бути окремою мікросхемою або вбудованим модулем мікроконтролерів фірми Microchip. Для окремо стоячих АЦП розрядність становить 12-13 біт. Можливо диференціальне включення. Для вбудованого модуля розрядність становить 10 біт.

AUSART - Addressable Universal Synchronous Asynchronous Receiver Transmitter - адресований USART. Відмінність полягає в тому, що є можливість використовувати 9-бітний протокол передачі даних для визначення адреси/даних.

Bank - спосіб адресації пам'яті. Так як у сімействі PIC16 є 7 біт для прямої адресації пам'яті, то можна працювати з 128 байтами пам'яті (включаючи спец. регістри). Для використання більшої кількості пам'яті, весь об`єм пам'яті ділиться на безперервні банки, кожен по 128 байт. Для вибору визначеного банку, необхідно встановити біти регістрів RP0:RP1, тобто можна працювати з 4 банками пам'яті. У сімействі PIC18 банкова система пам'яті замінена на більш досконалу.

Baud - бод (одиниця виміру) - дане поняття служить для обчислення кількості одиниць інформації (біт), переданих по послідовному каналу за інтервал часу в 1 секунду, тобто біт в секунду (bps).

BCD - Binary Coded Decimal - двійково-кодована десяткова форма обчислення. У шістнадцятковій формі обчислення кожна тетрада (послідовність з 4 біт) може приймати значення від 0000 до 1111 (від 0 до 15). Дана форма зручна для мікроконтролера, але не зручна для людини. Тому прийнято декодувати шістнадцяткову форму в двійково-десяткову. При цьому кожна тетрада може

приймати значення від 0000 до 1001 (від 0 до 9). У цьому випадку одним байтом можна записати число від 0 до 99.

BOR - Brown-out Reset - даний модуль є складовою частиною більшості мікроконтролерів фірми Microchip. Цей модуль переводить мікроконтролер у стан RESET, якщо живлення знизилося нижче нижче визначеної межі на визначений часовий інтервал. У більшості мікроконтролерів PIC16 дана межа встановлена апаратно. У мікроконтролерах PIC18 цю межу можна задавати на стадії програмування пристрою у слові конфігурації.

Brown-out - дане словосполучення означає, що сталося зниження напруги нижче визначеного робочого значення. Даний стан може виникати під час «осідання» напруги під час перемикання/включення потужного навантаження.

Capture - функція модуля CCP. У випадку включення модуля CCP у цей режим відбувається перезапис значення TMR1 у визначений регістр при виникненні деякої події. Такою подією може бути прихід фронту/зрізу імпульсу, прихід деякого числа імпульсів на ніжку модуля CCP.

CCP - Capture, Compare, Pulse Width Modulation (PWM) - захоплення, порівняння, широтно-імпульсна модуляція (ШІМ). Даний модуль є складовою частиною більшості мікроконтролерів фірми Microchip. Він служить для визначення довжини імпульсу, а також для реалізації широтно-імпульсної модуляції.

CERDIP - Ceramic dual in-line package - керамічний корпус з дворядним розташуванням ніжок. Відстань між ніжками становить 2.54мм.

CLC - Configurable Logic Cell - модуль, який дозволяє конфігурувати на апаратному рівні логічні елементи такі як «І», «АБО», «АБО, що виключає», різні тригери і т.д.

Common RAM - це область ОЗП, що має одне і теж розташування у всіх банках пам'яті. Цей вид ОЗП характерний для мікропроцесорів сімейства PIC16. «Загальний ОЗП» розташовується за адресами 70h-7Fh. Загальна область корисна для збереження необхідних змінних при перемиканні контексту, тому що не вимагає перемикання банків пам'яті. Ця частина пам'яті зазвичай

використовується для збереження вмісту акумулятора і деяких спеціальних регістрів.

Compare - порівняння - функція модуля CCP, сенс якої в тому, що пристрій здійснює визначену дію при збігу значень регістра таймера зі значенням у регістрі порівняння.

Configuration Word - слово конфігурації - спеціальний вид пам'яті, в якому розташовується інформація про різні режими роботи мікроконтролера. Цей блок пам'яті записується під час програмування мікроконтролера за допомогою програматора. В даному блоці розташовується інформація про тип генератора тактових імпульсів, про захист кристала від зчитування вмісту пам'яті програм / пам'яті даних EEPROM, про дозвіл роботи WDT і інших налаштуваннях. У різних мікроконтролерах різні слова конфігурації.

Conversion Time - час необхідний АЦП перетворення значення напруги на конденсаторі у цифрове значення. Даний час залежить від тактової частоти процесора, і обраного переддільника АЦП.

CPU - Central Processing Unit - центральний процесор

CTMU - CHARGE TIME MEASUREMENT UNIT - модуль для вимірювання заряду ємності і самої ємності. Основне застосування - це обробка сигналів ємнісних сенсорних панелей.

CWG - Complementary Waveform Generator - модуль який генерує комплементарний ШІМ сигнал з мертвим часом.

DAC - Digital-Analog Converter – цифро-аналоговий перетворювач (ЦАП). Даний модуль не є вбудованим модулем мікроконтролерів фірми Microchip. Він випускається у вигляді мікросхеми, що стоїть окремо. Однак цифро-аналогове перетворення можна виконати за допомогою модуля CCP, що працює у режимі PWM (ШІМ)

DIP - Dual In Package - корпус мікросхеми з дворядним розташуванням контактів.

Direct Addressing - пряма адресація. У цьому випадку адреса пам'яті знаходиться у команді/інструкції. Наприклад, `movwf 0x07`

DMA - Direct Memory Access - модуль для зчитування і запису RAM без задіяння процесора. Використовується для передачі і прийому пакетів даних через інтерфейси передачі даних, для збору даних від АЦП і т.д.

DSP - Digital Signal Processor - цифровий сигнальний процесор. Застосовуються для цифрової обробки сигналу, мають високу продуктивність ядра.

DCS - Digital Signal Controller - цифровий сигнальний контролер. Цей новий вид контролерів зараз починає випускати фірма Microchip під назвою dsPIC30F. Дані контролери з'єднують в одному ядрі функції властиві DSP і MCU.

ECAN - Enhanced Control Area Network - модуль для послідовної передачі даних між іншими CAN пристроями і мікроконтролерами. CAN надійніший ніж UART, так як має апаратну перевірку помилок передачі даних. В основному застосовується в автомобільній промисловості і для автоматизації на підприємствах.

ECCP - Enhanced Capture/Compare/PWM - покращений модуль CCP. Відрізняється від стандартного наявністю додаткових можливостей. Наприклад, даний модуль має можливість працювати з 2 або 4 вихідними модулями, змінювати полярність вихідного імпульсу, забезпечувати «мертву зону» для запобігання «наскрізних» струмів, забезпечувати екстрене відключення, автоматичне відключення і перезавантаження.

EEPROM - Electrically Erasable Programmable Read Only Memory - незалежний вид пам'яті. Дана пам'ять є складовою частиною більшості мікроконтролерів фірми Microchip. Однак її об`єм порівняно невеликий. Для мікроконтролерів сімейства PIC16 «типовим» значенням є 128 або 256 байт пам'яті EEPROM. Пам'яті такого об`єму цілком достатньо для зберігання калібрувальних констант, установочних даних та інших налаштувань користувача. Також дана пам'ять може служити «резервною областю», куди мікроконтролер може поміщати значення змінних у випадку критичного зниження напруги живлення.

EEPROM пам'ять також випускається у вигляді мікросхеми, що стоїть окремо з послідовним доступом (SPI або I^2C). У такому виконанні об`єм пам'яті може становити до 512Кбіт.

EPROM - Electrically Programmable Read Only Memory - електрично програмована постійна пам'ять. Пристрої даного типу пам'яті можна програмувати прямо в схемі. Стирання пам'яті проводиться опроміненням ультрафіолетом.

EMA - External Memory Addressing - модуль, що дозволяє мікроконтролеру виконувати програму, що знаходиться у зовнішній пам'яті.

EMI - Electromagnetic Interference - електромагнітна перешкода.

EXTRC - External Resistor-Capacitor (RC) - зовнішній ланцюг з резистора і конденсатора. У багатьох мікроконтролерах фірми Microchip є можливість підключення такого ланцюжка як генератор тактових імпульсів. Ця схема відрізняється дуже низькою ціною, але її стабільність набагато нижче, ніж у генератора на кварцовому резонаторі. У багатьох мікроконтролерах фірми Microchip є вбудований RC генератор - INTRC. Він калібрується на заводі-виробника і його точність становить 1%.

Flash memory - незалежний вид пам'яті. У процесорах фірми Microchip даний вид пам'яті використовується в якості пам'яті програм. Її можна програмувати і стирати прямо в схемі. Технологія даної пам'яті по функціональності майже еквівалентна EEPROM.

Fosc - частота тактового генератора. Крім свого «основного» значення - задавати швидкість роботи мікроконтролера, даний параметр відіграє важливу роль у багатьох вбудованих модулях мікроконтролерів фірми Microchip. Тому будьте уважні при зміні тактової частоти процесора!

GPIO - General Purpose Input/Output - введення/виведення загального призначення. Характерно для мікроконтролерів PIC12Fxxx.

GPR - General Purpose Register - регістр загального призначення. Частина пам'яті даних (один регістр), призначена для збереження в ній значень змінних.

HS - High Speed - висока швидкість. Один з режимів тактового генератора, в якому генератор налаштовується на роботу на високій частоті. Використовується для роботи з кварцами на частоті від 4 до 20 МГц. Встановлюється в слові конфігурації при програмуванні мікроконтролера.

IC - Integrated Circuit - інтегральна мікросхема.

ICD - In-Circuit Debugger - внутрішньосхемний відладчик. Вбудований модуль мікроконтролерів Microchip серій PIC12Fxx, Pic16F87x, Pic16F87xA, PIC18Fxxx. Наявність даного модуля дозволяє за допомогою спеціалізованого пристрою MPLAB ICD2 програмувати мікроконтролер і налагоджувати його внутрішньохемно, тобто мати доступ до всіх робочих регістрів, периферійним модулям і т.д.

ICSP - In-Circuit Serial Programming – внутрішньосхемне послідовне програмування. Для програмування мікроконтролера з цього протоколу використовується всього лише три виводи мікроконтролера + живлення. Завдяки наявності даного модуля програмування мікроконтролера можливо після установки на плату, що значно підвищує продуктивність при масовому виробництві.

IEEE - Institute of Electrical and Electronics Engineers - інститут інженерів з електротехніки та електроніки

IIC (I^2C) - Inter-Integrated Circuit - взаємно-інтегрований ланцюг, розроблений фірмою Philips. Один з режимів модуля SSP, що є вбудованим модулем мікроконтролерів фірми Microchip. Це двопровідний інтерфейс обміну даними між різними мікросхемами. Фірма Microchip випускає широкий спектр мікросхем, що працюють по протоколу I^2C.

IMC - інтегральні мікросхеми.

Indirect Addressing - непряма адресація. У цьому випадку адреса пам'яті не міститься в інструкції. Інструкція оперує INDF адресою, що представляє собою відображення адреси пам'яті у регістрі FSR. При виконанні команди дані беруться з комірки пам'яті, адреса якої вказується регістром FSR.

Instruction cycle - командний цикл - події, які відбуваються при виконанні команди. Основні етапи: Декодування, Читання, Виконання та Запис. Не всі етапи проходять всі команди. Один командний цикл Tcy виконується за чотири Tosc зовнішніх такти. Тобто кількість операцій за секунду дорівнює тактовій частоті, розділеної на чотири.

Interrupt - переривання - сигнал процесору, який направляє виконання програми по вектору переривання. Для мікроконтролерів сімейства PIC16 і PIC12Fxxx це адреса 0x0004H в пам'яті програм. Для мікроконтролерів фірми Microchip сімейства PIC18 існує два види переривань - переривання високого рівня і переривання низького рівня. Для цих переривань адреси рівні 0x0008H і 0x0018H відповідно. Перед зміною потоку виконання команди стан лічильника програми (Program Counter) заноситься в апаратний стек, і після обробки переривання виконання програми відновиться з цього ж місця. Під час виконання переривань користувач повинен подбати про збереження значення робочих регістрів.

IntRC - Internal Resistor-Capacitor (RC) - внутрішній ланцюг резистор-конденсатор. Багато мікроконтролерів фірми Microchip мають режим генератора, запуск якого здійснюється від внутрішнього RC-ланцюга. До таких мікроконтролерів відносяться практично всі пристрої, виконані за технологією NanoWatt. У таких пристроях внутрішній генератор калібрується на заводі і похибка не перевищує 1%. У даному режимі не потрібне підключення зовнішніх елементів, що дозволяє зменшити вартість виробу і звільнити ніжки, до яких раніше підключався кварцовий генератор або зовнішній RC ланцюжок. Встановлюється у слові конфігурації при програмуванні мікроконтролера.

LIN - Local Interconnect Network - протокол передачі даних по однопроводній шині.

LP - один з режимів генератора. Використовується для низькочастотних операцій в режимі зниженого енергоспоживання. Тактова частота - до 200 КГц. Встановлюється у слові конфігурації при програмуванні мікроконтролера.

LCD - Liquid Crystal Display - рідкокристалічний індикатор (РКІ).

LED - Light Emitting Diode - світлодіод

LSb - Least Significant Bit - найменш значущий біт.

LSB - Least Significant Byte - найменш значущий байт.

MI2C (MI2C) - Master Inter-Integrated Circuit - I^2C з апаратною підтримкою пристрою типу Master для реалізації топології мережі Multi-Master (в мережі присутні більше одного ведучого пристрою).

MOSFET - Metal Oxide Semiconductor Field Effect Transistor - польовий транзистор з МОН (метал-оксид-напівпровідник) структурою затвора. Даний вид пристроїв застосовується в якості ключового елемента в перетворювачах частоти, DC/DC перетворювачів, а також в пристроях управління двигунами. Фірма Microchip випускає даний вид транзисторів. Повний перелік знаходиться на сайті Microchip:
http://www.microchip.com/1010/pline/analog/anicateg/power/mosfet/index.htm

Motor control Kernel - апаратно-програмна реалізація управління двигуном. Включає в себе модулі ШІМ і програмне ядро, що дозволяє просто і зрозуміло програмувати ці модулі.

MSb - Most Significant bit - найбільш значущий біт.

MSB - Most Significant Byte - найбільш значущий байт.

NCO - NUMERICALLY CONTROLLED OSCILLATOR - модуль, що генерує сигнали прямокутної форми з малим відхиленням по частоті і з високою роздільною здатністю.

Op Amp - Operational Amplifier - операційний підсилювач.

OST - Oscillator Start-up Timer - таймер, який відповідає за надійний старт кварцового резонатора. При включенні живлення таймер чекає приходу 1024х імпульсів перед тим, як відпустити сигнал RESET.

OTP - одноразово програмована пам'ять програм

Pages - сторінки. Метод адресації пам'яті програми. Пристрої midrange мають 11-бітну адресацію на команди CALL і GOTO, що доводить довжину цих команд до 2К слів кожна. Для того щоб використовувати пам'ять великих об`ємів, пам'ять програми поділяється на безперервні сторінки, по 2К слів кожна.

Для звернення до тієї чи іншої сторінці необхідно встановити біти PCLATCH <5:4>. Якщо для маніпуляції є 2 біта, то відповідно до цього можна адресувати 4 сторінки.

PBOR - Programmable Brown-Out Detection/Reset - програмований модуль BOR. Даний модуль є складовою частиною мікроконтролерів Microchip серії PIC18. Цей модуль переводить мікроконтролер в стан RESET, якщо живлення знизилося нижче визначеної межі на визначений часовий інтервал. У мікроконтролерах PIC18 цю межу можна задавати на стадії програмування пристрою в слові конфігурації. Якість даного модуля в мікроконтролерах фірми Microchip настільки висока, що це дозволяє відмовитися від застосування зовнішніх супервізорів живлення.

PC - Program Counter - регістр, в якому знаходиться адреса пам'яті програми по якій знаходиться наступна команда.

PCB - Printed Circuit Board - друкована плата.

PSP - Parallel Slave Port - паралельний ведений порт. Паралельний порт використовується для взаємодії з мікропроцесорною шиною даних.

PDIP - Plastic DIP - пластиковий корпус з дворядним розташуванням ніжок. Відстань між ніжками становить 2.54мм.

PGA - Programmable-Gain Amplifier - підсилювач з програмованим підсиленням. Фірма Microchip випускає мікросхему MCP6S2x - операційний підсилювач з програмованим коефіцієнтом посилення і мультиплексором.

PLVD - Programmable Low-Voltage Detection - програмований модуль виявлення низької напруги. Даний модуль є складовою частиною мікроконтролерів фірми Microchip серії PIC18. При зниженні напруги нижче визначеного програмно-заданого рівня генерується переривання. Виникнення даного переривання може сигналізувати про початок відключення джерела живлення, наприклад, через відсутність напруги. Рекомендується зберегти всі важливі дані в незалежну пам'ять, наприклад EEPROM.

POR - Power-on Reset - вбудований модуль мікроконтролерів фірми Microchip, що робить скидання мікроконтролера після появи напруги живлення

і після перевищення ним визначеного рівня. Даний модуль покликаний забезпечити надійний старт мікроконтролера.

PPS - Peripheral Pin Select. У мікроконтролерах, які мають цю функцію, можна використовувати визначений набір периферійних пристроїв на більшості виводів мікроконтролера.

Pull-Up (resistor) - підтягуючий резистор. Резистор, що з'єднує будь-який ланцюг з джерелом живлення (Напр. +5).

Pull-Down (resistor) - підтягуючий резистор до землі. Резистор, що з'єднує будь-який ланцюг із землею.

PWM - Pulse Width Modulation - широтно-імпульсна модуляція. Ця функція в мікроконтролерах Microchip реалізується за допомогою CCP модуля.

PWRT - Power-up Timer - Внутрішній модуль мікроконтролерів фірми Microchip. Даний таймер утримує процесор в скинутому стані n мс (див. документацію на Ваш контролер), для того, щоб напруга живлення встигла піднятися до необхідного рівня. Робота даного модуля дозволяється/забороняється в слові конфігурації при програмуванні мікроконтролера.

RAM - оперативний запам'ятовуючий пристрій (ОЗП) - пам'ять даних мікроконтролера.

ROM - Read Only Memory - постійний запам'ятовуючий пристрій - пам'ять даних мікроконтролера (EEPROM)

RTCC - Real Time Clock and Calendar - модуль для відліку часу і дати, має також будильник.

Sampling Time - повний час аналого - цифрового перетворення. Включає час захоплення і перетворення.

SAW - Surface Acoustic Wave - поверхнева акустична хвиля (ПАХ)

SLAC - інтегруючий АЦП.

Sleep - режим низького енергоспоживання. У різних контролерах ця операція відбувається по-різному. У мікроконтролерах без технології nanoWatt цей режим досягається виключенням тактового генератора. Однак багато

модулів можуть зберігати свою працездатність. Наприклад продовжує працювати WDT, АЦП та інші модулі.

SOIC - тонкий корпус з відстанню між виводами 1.27 мм

SPI - Serial Peripheral Interface Protocol - протокол послідовного периферійного інтерфейсу. Даний вид протоколу є стандартним для багатьох мікроконтролерів фірми Microchip. Його наявність в мікроконтролері визначається наявністю модуля MSSP (SSP). Зазвичай це 3-х провідний інтерфейс - вихід, вхід, і тактова частота. Можливий синхронний обмін.

Tad - час необхідний для АЦ перетворення одного «біту» сигналу.

Tcy - час виконання команди. *Tcy = Fosc / 4.*

TSOP - Thin Small Outline Package - тонкий корпус із зменшеною відстанню між виводами.

USART - Universal Synchronous Asynchronous Receiver Transmitter - Універсальний синхронно-асинхронний приймач. Цей модуль є вбудованим модулем більшості мікроконтролерів фірми Microchip. Він може працювати як двобічний асинхронний порт або як напівдуплексний синхронний порт. В асинхронному режимі може бути підключений через перетворювач рівнів (наприклад, MAX232, ST232 і ін) до послідовного порту комп'ютера.

USB - Universal Serial Bus - універсальна послідовна шина. Даний вид передачі даних отримав широке розповсюдження в комп'ютерній техніці.

USB OTG - USB On-The-Go. На відміну від звичайного модуля USB, цей модуль дає можливість використовувати мікроконтролер не тільки як пристрій, але і як хост.

Vref - Voltage Reference - еталонна напруга для АЦП або напруга порівняння для компаратора. Фірма Microchip випускає дві мікросхеми такого характеру - MCP1525 і MCP1541. Також в деяких мікроконтролерах фірми Microchip є вбудований модуль Vref.

Wake-Up – «пробудження» мікроконтролера зі стану «sleep». Може бути викликано спрацьовуванням WDT або іншою зовнішньою (внутрішньою) подією.

WDT (Watch Dog Timer) - сторожовий таймер. Основне призначення - скидання процесора у випадку «зависання» програми. Скидання відбувається при переповненні таймера, для надійної роботи таймер має свій власний RC генератор. Для нормальної роботи програми його необхідно періодично обнуляти. Включається апаратно в слові конфігурації при програмуванні мікроконтролера.

XLP - eXtreme Low Power. Мікроконтролери з технологією XLP мають дуже низьке споживання в активному і сплячому режимі, що дозволяє їх застосовувати в пристроях, що живляться від батареї.

XT - один з режимів тактового генератора. Використовується для генерації від 100 КГц до 4 МГц. Встановлюється в слові конфігурації при програмуванні мікроконтролера.

ПЛМ - програмована логічна матриця (Programmable Logic Array, PLA) - функціональний блок, створений на базі напівпровідникової технології, для реалізації логічний. цифрових схем.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Microchip Technology Inc. PIC 18FXX2. Однокристальные 8-розрядные FLASH CMOS микроконтроллеры с 10 - разрядным АЦП компании Microchip Technology Incorporated / Microchip Technology Incorporated. - USA : ООО «Микро-Чип» (www.microchip.ru), 2003. - 299 с.
2. Microchip Technology Inc. PIC18FXX2. Data Sheet. High Performance, Enhanced FLASH. Microcontrollers with 10–Bit A/D / Microchip Technology Inc. – Microchip Technology Incorporated, USA (www.microchip.com), 2002. – 330 p.
3. Microchip Technology Inc. PIC18FXX2. Data Sheet. High Performance, Enhanced FLASH. Microcontrollers with 10–Bit A/D / Microchip Technology Inc. – Microchip Technology Incorporated, USA (www.microchip.com), 2006. – 332 p.
4. Microchip Technology Inc. MCP4921/4922. 12–Bit DAC with SPI™ Interface / Microchip Technology Inc. – Microchip Technology Incorporated, USA (www.microchip.com), 2007. – 42 p.
5. Microchip Technology Inc. 24AA256/24LC256/24FC256. Data Sheet. / 256K I^2C™ CMOS Serial EEPROM / Microchip Technology Inc. – Microchip Technology Incorporated, USA (www.microchip.com), 2005. – 26 p.
6. ООО «Микро–Чип» Программная реализация I^2C интерфейса (режим ведущего) / ООО «Микро–Чип». – М.: ООО «Микро–Чип» (www.microchip.ru), 2001. – 8 с.
7. ООО «Микро-Чип» Краткий обзор интерфейса I^2C / ООО «Микро–Чип». – М.: ООО «Микро-Чип» (www.microchip.ru), 2001. – 7 с.
8. ООО «Микро-Чип» Модуль 10–разрядного АЦП в микроконтроллерах PIC 18CXX2 / ООО «Микро–Чип» – М.: ООО «Микро–Чип» (www.microchip.ru), 2001. – 10 с.
9. Богатырев Е. А. Энциклопедия электронных компонентов. Большие интегральные схемы / Е.А. Богатырев, В.Ю. Ларин, А.Е. Лякин; под ред. А.Н. Еркина. - Т. 1. - М. : ООО «МАКРО ТИМ», 2006. - 224 с.

10. Брей Барри Применение микроконтроллеров PIC18. Архитектура, программирование и построение интерфейсов с применением С и ассемблера / Барри Брей. - К. : «МК-Пресс»; СПб : «КОРОНА-ВЕК», 2008. - 576 с.
11. Предко М. PIC-микроконтроллеры: архитектура и программирование / Майкл Предко. - Саратов : Профобразование, 2010. - 512 с.
12. Предко М. PIC-микроконтроллеры: архитектура и программирование / Майкл Предко. - Саратов : Профобразование, 2017. - 512 с.
13. Семенов Б.Ю. Шина I²C в радиотехнических конструкциях / Б.Ю. Семенов; изд. 2-е. доп. – М. : СОЛОН-Пресс, 2004. – 224 с.
14. Стюарт Болл Р. Аналоговые интерфейсы микроконтроллеров / Стюарт Болл Р.; пер. с англ. С. Щербинин. – М. : Додэка XXI, 2007. – 362 с.
15. Шилдт Герберт Полный справочник по C++ / Герберт Шилдт; пер. с англ. к.ф.–м.н. Д.А. Клюшина; 4–е изд. – М.: Издательский дом «Вильямс», 2006. – 800 с.
16. Шилдт Герберт Полный справочник по C / Герберт Шилдт; пер. с англ. А.Г. Беляева, И.В. Константинова; 4–е изд. – М.: Издательский дом «Вильямс», 2002. – 704 с.
17. Шпак Ю. А. Программирование на языке С для AVR и PIC микроконтроллеров / Ю. А. Шпак; 2-е изд. - К. : «МК-Пресс»; СПб. : «КОРОНА_ВЕК», 2011. - 544 с.
18. Хофманн М. Микроконтроллеры для начинающих / М. Хофманн; пер. с нем. - СПб. : БХВ-Петербург, 2010. - 304 с.
19. Афанасьев Илья Применение модуля захвата, сравнения, ШИМ в контроллерах Microchip / Илья Афанасьев // Журнал «Компоненты и технологии» №3, 2004 г. - 3 с.
URL: https://kit-e.ru/assets/files/pdf/2004_03_136.pdf,
http://www.microchip.com.ru/Support/tipsCCP%201.html,
http://www.radioradar.net/hand_book/documentation/microchip_shim.html,
20. Шина управления I²C. URL: https://cxem.net/comp/comp67.php

21. SPI (Serial Peripheral Interface) // Материал из Национальной библиотеки им. Н.Э. Баумана. URL: https://ru.bmstu.wiki/SPI_(Serial_Peripheral_Interface)
22. Последовательный интерфейс SPI. URL: https://prog-cpp.ru/micro-spi/
23. SPI Block Guide V03.06 / Motorola, Inc. - 38 с. URL: https://web.archive.org/web/20150413003534/http://www.ee.nmt.edu/~teare/ee308l/datasheets/S12SPIV3.pdf
24. Елисеев Н. Интерфейс 1-WIRE: устройство и применение / Н. Елисеев // ЭЛЕКТРОНИКА: Наука, Технология, Бизнес 8/2007. - 6 с. URL: http://www.electronics.ru/files/article_pdf/0/article_657_119.pdf
25. Морозов Е. Обзор шины 1-WIRE для создания умного дома / Егор Морозов. URL: https://www.iguides.ru/main/gadgets/other_vendors/obzor_platformy_1_wire dlya_sozdaniya_umnogo_doma/
26. Термины и аббревиатуры. URL: http://www.microchip.ua/index.php?p=termins.php&leng=rus&tl=%D2%E5%F0%EC%E8%ED%FB%20%E8%20%E0%E1%E1%F0%E5%E2%E8%E0%F2%F3%F0%FB
27. C keywords. URL: http://en.cppreference.com/w/c/keyword/
28. Terminal 1.9b. URL: https://sites.google.com/site/terminalbpp/https://micro–pi.ru/terminal–1–9b–%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0% B0%D0%B5%D0%BC–com–%D0%BF%D0%BE%D1%80%D1%82%D0 %BE%D0%BC/
29. Terminal 1.9b – работаем с COM-портом. URL: : https://micro–pi.ru/terminal–1–9b–%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%D0%B5%D0%BC–com–%D0%BF%D0%BE%D1%80%D1%82%D0%BE%D0%BC/
30. COM port development tool. URL: https://sites.google.com/site/terminalbpp/
31. Как пользоваться терминальной программой Terminal 1.9b. URL: https://faq.radiofid.ru/knowledge–bases/2/articles/13–kak–polzovatsya–terminalnoj–programmoj–terminal–19b /

32. Абашкин Иван Как пользоваться терминальной программой Terminal 1.9b. URL: https://faq.radiofid.ru/knowledge–bases/2/articles/13–kak–polzovatsya–terminalnoj–programmoj–terminal–19b/
33. Proteus (система автоматизированного проектирования). URL: https://ru.wikipedia.org/wiki/Proteus_(система_автоматизированного_проектирования)https://ru.wikipedia.org/wiki/Proteus_(система_автоматизированного_проектирования)/
34. PSpice. URL: https://ru.wikipedia.org/wiki/PSpicehttps://ru.wikipedia.org/wiki/PSpice/
35. Описание цифро-аналогового преобразователя MCP4921. URL: https://studbooks.net/2341863/tehnika/ opisanie_tsifro_analogovogo preobrazovatelya_mcp4921
36. G–NOR Lighting LED TECHNOLOGY / GNT–5631Ax–Bx (TRIPLE DIGIT LED DISPLAY) / G–NOR ELECTRONICS CO.,LTD. – 1 с.
37. Трехразрядный светодиодный цифровой дисплей. URL: https://old.radiodetali.com/td/displ/gnt5631.htm/ GNT–5631
38. I²C. URL: https://ru.wikipedia.org/wiki/I%C2%B2C/ I²C
39. Ермолович А.В. Программи́руемая логи́ческая ма́трица URL: https://bigenc.ru/technology_and_technique/text/3178939
40. Современные цифровые технологии / Справочник специалиста в АЦП. URL: http://www.centeradc.ru/guide/#:~:text=%D0%9C%D0%BB%D0%B0%D0%B4%D1%88%D0%B8%D0%B9%20%D0%B7%D0%BD%D0%B0%D1%87%D0%B0%D1%89%D0%B8%D0%B9%20%D1%80%D0%B0%D0%B7%D1%80%D1%8F%D0%B4%20(%D0%9C%D0%97%D0%A0)&text=%D0%9C%D0%B8%D0%BD%D0%B8%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5%20%D0%B2%D1%85%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5%20%D0%BD%D0%B0%D0%BF%D1%80%D1%8F%D0%B6%D0%B5%D

0%BD%D0%B8%D0%B5%20%D1%80%D0%B0%D0%B7%D1%80%D0%B5%D1%88%D0%B0%D0%B5%D0%BC%D0%BE%D0%B5%20%D0%90%D0%A6%D0%9F,%D0%98%D0%B7%D0%BC%D0%B5%D1%80%D1%8F%D0%B5%D1%82%D1%81%D1%8F%20%D0%B2%20%5B%D0%92%5D.

41. Сайт кафедри програмування комп'ютерних систем і мереж. URL: http://pksm.kntu.kr.ua/DEVELOPMENTS_2.html

Навчальне електронне видання комбінованого використання.

Можна використовувати в локальному та мережному режимах

Смірнов Володимир Вікторович
кандидат технічних наук, доцент

Смірнова Наталія Володимирівна
кандидат технічних наук, доцент

Пархоменко Юрій Михайлович
кандидат технічних наук, доцент

АРХІТЕКТУРА ТА ПРОГРАМУВАННЯ ПЕРИФЕРІЙНИХ ІНТЕРФЕЙСНИХ КОНТРОЛЕРІВ

Підручник

Редактор В.В. Смірнов

Технічний редактор В.В. Смірнов

Верстальник Н.В. Смірнова

Видавець

Центральноукраїнський національний технічний університет
25006, м. Кропивницький. пр. Університетський, 8,

тел. +38-091-608-75-02
E-mail: swckntu@gmail.com