МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Центральноукраїнський національний технічний університет

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

ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРНИХ СИСТЕМ

Навчальний посібник

Кропивницький - 2021

ББК 32.97

С50

Рекомендовано Вченою радою Центральноукраїнського національного технічного університету до друку та використанню навчального посібника у навчальному процесі здобувачами вищої освіти очної та заочної форми навчання за спеціальністю 123 «Комп'ютерна інженерія», протокол № 8 від 29 березня 2021 року

Рецензенти:

Павленко Іван Іванович, доктор технічних наук, професор, завідувач кафедрою Технології машинобудування Центральноукраїнського національного технічного університету.

Віхрова Лариса Григорівна, кандидат технічних наук, професор, декан факультету Автоматики та Енергетики Центральноукраїнського національного технічного університету.

С50

Смірнов В.В., Смірнова Н.В., Пархоменко Ю.М.
Програмування мікроконтролерних систем : навчальний посібник ;
М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т. –
Кропивницький : ЦНТУ, 2021. – 262 с.

У навчальному посібнику 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.

Навчальний посібник 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

ЗМІСТ

ВСТУП

Дисципліна **«Програмування мікроконтролерних систем»** 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 навчального посібника 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 дисципліни **«Програмування мікроконтролерних систем»**, а також для підготовки програмістів в області розробки і управління територіально-розподіленими системами, роботами, об'єктами і комплексами.

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

РОЗДІЛ 1. ОПИС НАВЧАЛЬНОЇ ДИСЦИПЛІНИ «ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРНИХ СИСТЕМ»

Анотація до дисципліни

Дисципліна «**Програмування мікроконтролерних систем**» викладається відповідно до навчального плану підготовки бакалаврів:

галузі знань: 12 Інформаційні технології;

спеціальності: 123 «Комп'ютерна інженерія»;

спеціалізації: «Комп'ютерні системи та мережі».

Дисципліна відноситься до категорії **вибіркових**.

Форма підсумкового контролю: екзамен.

Мета і завдання дисципліни

Основна мета дисципліни полягає в придбанні досконалих знань і навичок роботи з апаратним та програмним забезпеченням систем управління об'єктом на базі мікро-ЕОМ.

Компетентності, якими повинен оволодіти здобувач

В результаті засвоєння лекційного матеріалу і виконання лабораторних робіт студенти повинні отримати теоретичні знання та методику ефективної роботи з сучасними системами управління на базі мікро-ЕОМ.

Програмні результати навчання

- володіти теоретичними та практичними навичками з розробки та програмування мікроконтролерних систем;
- володіти знаннями і практичними навичками програмування основних інтерфейсів які використовуються у таких системах;
- володіти знаннями і практичними навичками проектування, програмування та використання автоматизованих систем збору даних, засобів робототехніки та автоматики.

Завдання вивчення дисципліни

- вивчення теоретичних основ мікроконтролерних систем;
- вивчення дротових інтерфейсів обміну даними;
- вивчення бездротових інтерфейсів обміну даними;
- вирішення завдань введення/виведення сигналів;
- вирішення завдань віддаленого управління об'єктами;
- набуття практичних навиків у сфері програмування мікроконтролерних систем.

Предметом навчальної дисципліни є архітектура мікроконтролерів і програмне забезпечення мікроконтролерів для систем управління об'єктом, провідні та безпровідні інтерфейси і програмне забезпечення мікроконтролерів на базі RTOS.

Результати навчання

У результаті вивчення дисципліни студент повинен:

знати:

- роботу дротових інтерфейсів обміну даними;
- роботу бездротових інтерфейсів обміну даними;
- роботу микроконтролера у режимі RTOS;

вміти:

- визначати параметри вхідних сигналів;
- створювати програмне забезпечення для роботи з дротовими і бездротовими інтерфейсами обміну даними;
- проводити віддалене управління об’єктами на базі мікроконтролерів;
- розробляти прикладні та системні програми для систем управління об’єктами на базі мікроконтролерів.

Теми лабораторних робіт

Таблиця 1.1 - Теми лабораторних робіт

№ з/п	Назва теми
№ 1 - «LED_display»	Динамічна індикація. Виведення інформації на LED – дисплей
№ 2 - «PWM»	Модуль CCP. Керування двигуном постійного струму
№ 3 - «EEPROM»	Робота із зовнішньою EEPROM - пам'яттю по інтерфейсу I^2C
№ 4 - «SERVO»	Сервоприводи

КРИТЕРІЇ ОЦІНЮВАННЯ ЗНАНЬ ЗДОБУВАЧІВ ВИЩОЇ ОСВІТИ

Реалізація основних завдань контролю знань здобувачів вищої освіти у ЦНТУ досягається системними підходами до оцінювання та комплексністю застосування різних видів контролю.

Згідно з діючою в університеті системою комплексної діагностики знань, з метою стимулювання планомірної та систематичної навчальної роботи, оцінка знань здобувачів вищої освіти здійснюється за 100-бальною системою.

Форми контролю знань здобувачів вищої освіти:

- поточний;
- рубіжний;
- семестровий підсумковий (залік, **екзамен**).

Оцінювання знань здобувачів вищої освіти в університеті здійснюється за 100-бальною шкалою, яка переводиться відповідно у:

національну шкалу:

- **«відмінно»**;
- **«добре»**;
- **«задовільно»**;
- **«незадовільно»,**

та шкалу європейської кредитно-трансферної системи ЄКТС:

- **A**; **B**; **C**; **D**; **E**; **FX**; **F**.

Поточний контроль проводиться на кожному семінарському, практичному/лабораторному занятті та за результатами виконання завдань самостійної роботи. Він передбачає оцінювання теоретичної підготовки здобувачів вищої освіти із зазначеної теми (у тому числі, самостійно опрацьованого матеріалу) під час роботи на семінарських заняттях та набутих практичних навичок під час виконання завдань лабораторних/практичних робіт.

Критерії поточного оцінювання знань здобувачів вищої освіти

Таблиця 1.2 - Критерії поточного оцінювання знань здобувачів вищої освіти

Усний виступ та виконання письмового завдання, тестування (бали)	**Критерії оцінювання**
5	В повному обсязі володіє навчальним матеріалом, вільно самостійно та аргументовано його викладає під час усних виступів та письмових відповідей, глибоко та всебічно розкриває зміст теоретичних питань та практичних завдань, використовуючи при цьому обов'язкову та додаткову літературу. Правильно вирішив усі тестові завдання.
4	Достатньо повно володіє навчальним матеріалом, обґрунтовано його викладає під час усних виступів та письмових відповідей, в основному розкриває зміст теоретичних питань та практичних завдань, використовуючи при цьому обов'язкову літературу. Але при викладанні деяких питань не вистачає достатньої глибини та аргументації, допускаються при цьому окремі несуттєві неточності та незначні помилки. Правильно вирішив більшість тестових завдань.

Продовження таблиці 1.2

Усний виступ та виконання письмового завдання, тестування (бали)	**Критерії оцінювання**
3	В цілому володіє навчальним матеріалом викладає його основний зміст під час усних виступів та письмових відповідей, але без глибокого всебічного аналізу, обґрунтування та аргументації, без використання необхідної літератури допускаючи при цьому окремі суттєві неточності та помилки. Правильно вирішив половину тестових завдань.
2	Не в повному обсязі володіє навчальним матеріалом. Фрагментарно, поверхово (без аргументації та обґрунтування) викладає його під час усних виступів та письмових відповідей, недостатньо розкриває зміст теоретичних питань та практичних завдань, допускаючи при цьому суттєві неточності, правильно вирішив меншість тестових завдань.
1	Частково володіє навчальним матеріалом не в змозі викласти зміст більшості питань теми під час усних виступів та письмових відповідей, допускаючи при цьому суттєві помилки. Правильно вирішив окремі тестові завдання.
0	Не володіє навчальним матеріалом та не в змозі його викласти, не розуміє змісту теоретичних питань та практичних завдань. Не вирішив жодного тестового завдання

Доповнення виступу:

2 бали – отримують здобувачі вищої освіти, які глибоко володіють матеріалом, чітко визначили його зміст; зробили глибокий системний аналіз змісту виступу, виявили нові ідеї та положення, що не були розглянуті, але суттєво впливають на зміст доповіді, надали власні аргументи щодо основних положень даної теми.

1 бал - отримують здобувачі вищої освіти, які виклали матеріал з обговорюваної теми, що доповнює зміст виступу, поглиблює знання з цієї теми та висловили власну думку.

Суттєві запитання до доповідачів:

1 бал - отримують здобувачі, які своїм запитанням до виступаючого суттєво і конструктивно можуть доповнити хід обговорення теми.

0,5 балів - отримують здобувачі вищої освіти, які у своєму запитанні до виступаючого вимагають додаткової інформації з ключових проблем теми, що розглядається.

Експрес-контроль:

1 бал - нараховуються здобувачам вищої освіти, які вільно володіють усім навчальним матеріалом, орієнтуються в темі та аргументовано висловлюють свої думки.

0,5 балів - отримують здобувачі вищої освіти, які частково володіють матеріалом та можуть окреслити лише деякі проблеми теми.

Ведення опорного конспекту лекції:

Опорний конспект лекції *(ОКЛ)* – вид навчально-методичного посібника, в якому у стисло і системно викладено основний теоретичний матеріал у формі основних понять і положень, що структурно й логічно пов’язані між собою.

Кожен здобувач повинен мати ОКЛ на лекціях і вести в ньому записи власноруч. Під час аудиторної роботи з ОКЛ здобувачі вищої освіти записують

основні тези лекції та пояснення викладача. Під час самостійної роботи рекомендується доповнити записи лекції.

1 бал нараховується здобувачам вищої освіти, які в повному обсязі самостійно і творчо опрацювали всі питання лекції і вільно володіють її змістом.

0,5 балів нараховується здобувачам вищої освіти, які опрацювали лише окремі питання лекції і не достатньо вільно володіють її змістом.

Рубіжний контроль знань здобувачів вищої освіти

Рубіжний контроль успішності здобувачів вищої освіти − це об'єктивна оцінка міри освоєння здобувачами вищої освіти денної форми навчання програм навчальних дисциплін; результатів у здобутті знань, дотримання навчальної дисципліни.

Рубіжний контроль успішності має на меті підвищення мотивації до навчання і свідомої навчальної дисципліни здобувачів вищої освіти.

Рубіжний контроль успішності здобувачів вищої освіти проводиться науково-педагогічними працівниками під час проведення всіх видів аудиторних занять з усіх дисциплін по завершеним темам всередині семестру та в останній тиждень семестру.

Оцінка рубіжного контролю носить комплексний характер і враховує досягнення здобувача вищої освіти по основних компонентах, які визначені робочою програмою навчальної дисципліни:

- рівень засвоєння навчального матеріалу;
- повнота виконання здобувачем вищої освіти усіх видів робіт, передбачених навчальною програмою дисципліни;
- відвідування занять;
- робота з дистанційними курсами на сайті дистанційної освіти ЦНТУ;
- самостійна робота здобувача вищої освіти;
- дослідницька робота тощо.

Результати поточних та рубіжних контролів є складовими оцінки семестрового підсумкового контролю.

Результати рубіжного контролю успішності з усіх дисциплін фіксуються викладачами двічі на семестр у встановлені графіком освітнього процесу терміни у факультетських журналах результатів рубіжного контролю і доводяться до відома кураторів академічних груп, обговорюються на засіданнях кафедр, рад факультетів

Загальна максимальна кількість балів, виділених для оцінки результатів під час одного рубіжного контролю робочою програмою навчальної дисципліни, при семестровому підсумковому контролі:

- у формі заліку складає 50 балів;
- **у формі екзамену складає 30 балів.**

Критерії рубіжного оцінювання знань здобувачів вищої освіти

Таблиця 1.3 - Критерії рубіжного оцінювання знань здобувачів вищої освіти

Загальна кількість балів	**Критерії оцінювання**
25 - 30	В повному обсязі володіє навчальним матеріалом, вільно самостійно та аргументовано його викладає під час усних та письмових відповідей глибоко та всебічно розкриває зміст теоретичних питань та практичних завдань, використовуючи при цьому обов'язкову та додаткову літературу. Правильно вирішив усі тестові завдання.

Продовження таблиці 1.3

Загальна кількість балів	**Критерії оцінювання**
21 - 24,5	Достатньо повно володіє навчальним матеріалом, обґрунтовано його викладає під час усних виступів та письмових відповідей, в основному розкриває зміст теоретичних питань та практичних завдань, використовуючи при цьому обов'язкову літературу. Але при викладанні деяких питань не вистачає достатньої глибини та аргументації, допускаються при цьому окремі несуттєві неточності та незначні помилки. Правильно вирішив більшість тестових завдань.
17 - 20,5	В цілому володіє навчальним матеріалом викладає його основний зміст під час усних виступів та письмових відповідей, але без глибокого всебічного аналізу, обґрунтування та аргументації, без використання необхідної літератури допускаючи при цьому окремі суттєві неточності та помилки. Правильно вирішив половину тестових завдань
12 - 16,5	Не в повному обсязі володіє навчальним матеріалом. Фрагментарно, поверхово (без аргументації та обґрунтування) викладає його під час усних виступів та письмових відповідей, недостатньо розкриває зміст теоретичних питань та практичних завдань, допускаючи при цьому суттєві неточності, правильно вирішив меншість тестових завдань.
10 - 15	Частково володіє навчальним матеріалом не в змозі викласти зміст більшості питань теми під час усних виступів та письмових відповідей, допускаючи при цьому суттєві помилки. Правильно вирішив окремі тестові завдання.

Продовження таблиці 1.3

Загальна кількість балів	**Критерії оцінювання**
0	Не володіє навчальним матеріалом та не в змозі його викласти, не розуміє змісту теоретичних питань та практичних завдань. Не вирішив жодного тестового завдання.

Сума балів, накопичених здобувачем вищої освіти за виконання всіх видів поточних навчальних завдань (робіт) на практичних (семінарських) заняттях та на підсумковому рубіжному контролі, свідчить про ступінь оволодіння ним програмою навчальної дисципліни на конкретному етапі її вивчення.

Протягом семестру здобувачі вищої освіти можуть набрати від 0 до 100 балів, що переводяться у національну шкалу оцінювання і відповідно у шкалу ЄКТС.

Кількість балів відповідає певному рівню засвоєння дисципліни:

Таблиця 1.4 - Шкала оцінювання: національна та ЄКТС

За системою ЦНТУ	**За шкалою ECTS**	**За національною системою**	**Визначення**
90 - 100	A	5 (відмінно)	Повно та ґрунтовно засвоїв всі теми навчальної програми вміє вільно та самостійно викласти зміст всіх питань програми навчальної дисципліни, розуміє її значення для своєї професійної підготовки, повністю виконав усі завдання кожної теми та рубіжного контролю в цілому. Брав участь в олімпіадах, конкурсах, конференціях.

Продовження таблиці 1.4

За системою ЦНТУ	**За шкалою ECTS**	**За національною системою**	**Визначення**
82 - 89	B	4 (дуже добре)	Недостатньо повно та ґрунтовно засвоїв окремі питання робочої програми. Вміє самостійно викласти зміст основних питань програми навчальної дисципліни, виконав завдання кожної теми та рубіжного контролю в цілому.
74 - 81	C	4 (добре)	Недостатньо повно та ґрунтовно засвоїв деякі теми робочої програми, не вміє самостійно викласти зміст деяких питань програми навчальної дисципліни. Окремі завдання кожної теми та рубіжного контролю в цілому виконав не повністю
64 - 73	D	3 (задовільно)	Засвоїв лише окремі теми робочої програми. Не вміє вільно самостійно викласти зміст основних питань навчальної дисципліни, окремі завдання кожної теми рубіжного контролю не виконав.
60 - 63	E	3 (достатньо)	Засвоїв лише окремі питання навчальної програми. Не вміє достатньо самостійно викласти зміст більшості питань програми навчальної дисципліни. Виконав лише окремі завдання кожної теми та рубіжного контролю в цілому.

Продовження таблиці 1.4

За системою ЦНТУ	За шкалою ECTS	За національною системою	Визначення
> 60	Fx	2 (незадовільно)	Не засвоїв більшості тем навчальної програми не вміє викласти зміст більшості основних питань навчальної дисципліни. Не виконав більшості завдань кожної теми та рубіжного контролю в цілому.

Підсумковий семестровий контроль

Семестровий підсумковий контроль проводиться з метою визначення рівня досягнення здобувачами вищої освіти запланованих результатів навчання, що визначені робочою програмою навчальної дисципліни (практики). Здобувач вищої освіти вважається допущеним до семестрового підсумкового контролю з конкретної навчальної дисципліни (семестрового екзамену, диференційованого заліку або заліку), якщо він виконав усі види робіт, які передбачені навчальним планом на відповідний семестр з цієї навчальної дисципліни, та виконав умови контракту.

Семестровий підсумковий контроль проводиться у формі екзамену, диференційованого заліку чи заліку, що визначено навчальним планом, у терміни, передбачені графіком освітнього процесу. Зміст екзаменів і заліків визначається робочими навчальними програмами дисциплін.

У випадку проведення семестрового підсумкового контролю у формі заліку, кожен з видів роботи (завдань), виконаних здобувачем вищої освіти протягом семестру, оцінюється визначеною кількістю балів відповідно до схеми нарахування балів, що представлена в робочій програмі навчальноїдисципліни. Здобувачі вищої освіти мають бути повідомлені про кількість набраних ними балів до початку екзаменаційної сесії.

Семестровий екзамен − це форма підсумкового семестрового контролю, що полягає в оцінці засвоєння здобувачем вищої освіти теоретичного та практичного навчального матеріалу з певної навчальної дисципліни протягом семестру, результати навчання за яким оцінюються за стобальною та чотирьохбальною шкалами оцінювання.

Екзамени складаються здобувачами вищої освіти з відповідних дисциплін, які передбачені навчальним планом, в період екзаменаційних сесій. Семестрові екзамени проводяться в письмовій, усній та тестовій формі.

Екзамен може завершуватись усною співбесідою зі здобувачами вищої освіти, їх відповідями на додаткові запитання.

Зміст, обсяг, структура, форма екзаменаційної роботи, система і критерії її оцінювання визначаються робочою програмою дисципліни. На початку семестру науково-педагогічний працівник повинен ознайомити здобувачів вищої освіти зі змістом, структурою, формою екзаменаційної (залікової) роботи та прикладами завдань. Обсяг матеріалу, що виноситься на підсумковий контрольний захід, має охоплювати весь зміст дисципліни відповідно до її робочої програми.

Оцінку підсумкового семестрового контролю у формі екзамену становить сума балів за результатами рубіжних контролів та балів, набраних здобувачем вищої освіти при складанні семестрового екзамену. Загальна кількість балів, виділених на проведення семестрового екзамену робочою програмою навчальної дисципліни, складає 40 балів. Кількість балів, одержана здобувачем вищої освіти на екзамені, додається до результатів рубіжних контролів, що разом складає оцінку знань здобувача вищої освіти з навчальної дисципліни за 100-бальною шкалою та переводиться в оцінку за шкалою ЄКТС і національною шкалою (“Відмінно”, “Добре”, “Задовільно”, “Незадовільно”).

Семестровий залік полягає в оцінці рівня засвоєння здобувачем вищої освіти навчального матеріалу на лекційних, практичних, семінарських або лабораторних заняттях і виконання індивідуальних завдань за стобальною та дворівневою («зараховано», «не зараховано») шкалою оцінювання результатів

навчання. Семестровий залік планується при відсутності екзамену. Семестровий залік з окремої дисципліни проводиться на останньому занятті, до початку екзаменаційної сесії.

Навчальний план передбачає при вивченні навчальної дисципліни виконання певних видів робіт на лекційних, практичних, семінарських, лабораторних заняттях, виконання індивідуальних завдань, інших видів навчальної діяльності, тому оцінка здобувачам вищої освіти вище 60 балів може виставлятися без виконання ними підсумкової залікової роботи. В такому разі виставлення оцінки підсумкового семестрового контролю не передбачає обов'язкової присутності здобувача вищої освіти на заліку. У разі, якщо сума рейтингових балів менша ніж 60, але виконані умови допуску до семестрового контролю, здобувач вищої освіти виконує на останньому за розкладом занятті залікову контрольну роботу. За бажанням, здобувач вищої освіти має право на виконання залікової контрольної роботи з метою підвищення кількості балів, які були набрані ним протягом семестру.

Заліки приймаються науково-педагогічними працівниками, які проводили практичні, семінарські та інші заняття в академічній групі або читали лекції з даної дисципліни.

Семестровий диференційований залік − це форма підсумкового контролю, що полягає в оцінці засвоєння здобувачем вищої освіти навчального матеріалу з певної дисципліни виключно на підставі результатів виконаних індивідуальних завдань (розрахункових, графічних, під час проходження практики тощо).

Семестровий диференційований залік може плануватися при відсутності екзамену з даної навчальної дисципліни. Здобувачі вищої освіти, які набрали за результатами поточного контролю менше мінімальної кількості балів, необхідної для виставлення заліку, допускаються до семестрового контролю після перескладання контрольних заходів, що проводилися в межах рубіжних контролів.

Здобувачі вищої освіти заочної форми навчання допускаються до семестрового контролю, якщо вони своєчасно виконали завдання із самостійної роботи з навчальних дисциплін семестру.

При складанні заліку оцінка підсумкового семестрового контролю виставляється як сума балів, набраних здобувачем вищої освіти за рубіжними контролями. У разі, якщо сума рентингових балів менша за 60, але виконані умови допуску до семестрового контролю з цієї навчальної дисципліни, здобувач вищої освіти виконує на останньому за розкладом занятті залікову контрольну роботу.

РОЗДІЛ 2. ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ МІКРОКОНТРОЛЕРІВ

АПАРАТНО-ПРОГРАМНІ КОМПЛЕКСИ

Для виконання лабораторних робіт 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 дисциплін **«Програмне забезпечення управляючих мікро-ЕОМ»** та **«Програмування мікроконтролерних систем»**.

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 дисципліни;
- завдання і методичні вказівки до виконання лабораторних робіт;
- 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 зовнішньою **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.

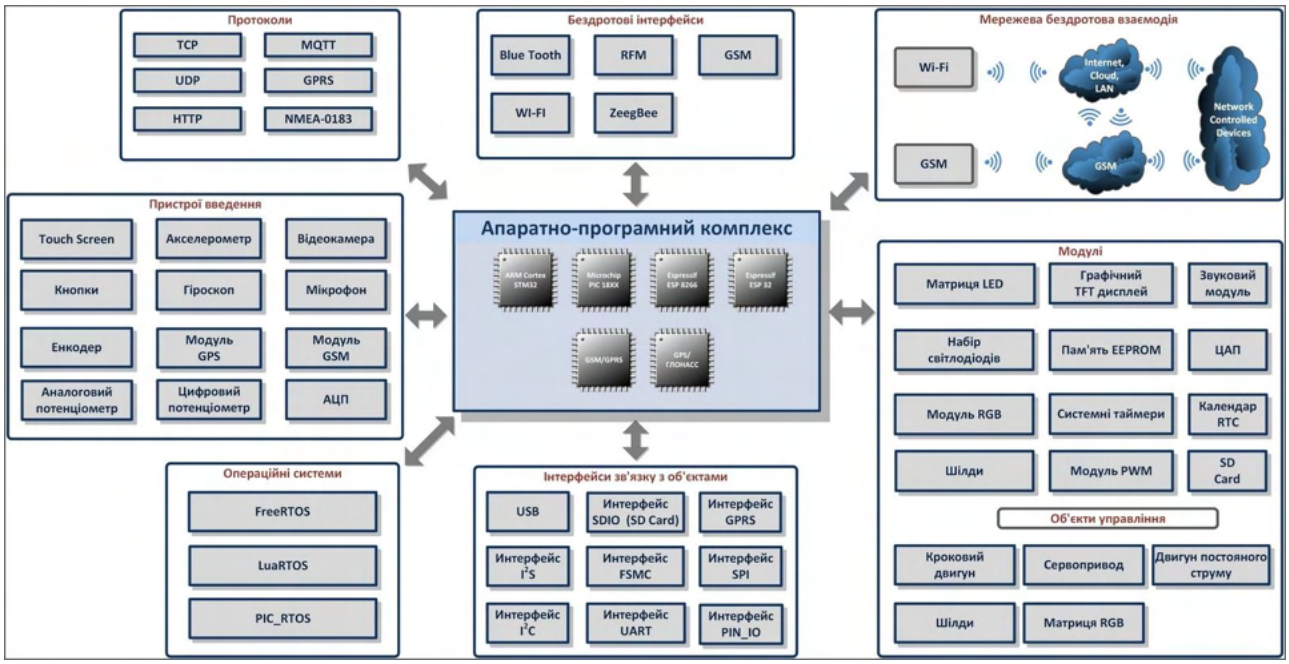

Рис. 2.3 - Структура Апаратно-програмного комплексу № 2

Склад апаратної частини комплексу № 2 (рис. 2.2)

1. Модулі контролерів:

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

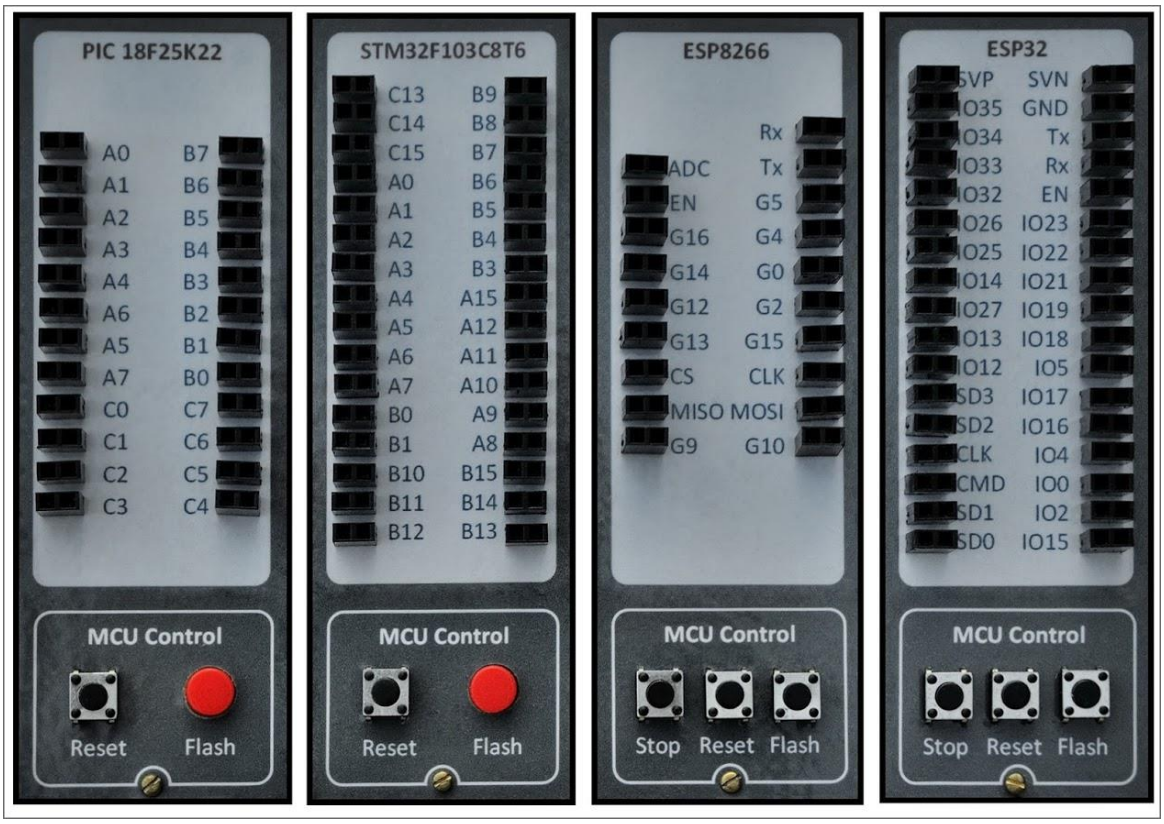

Рис. 2.4 - Змінні модулі контролерів

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.).

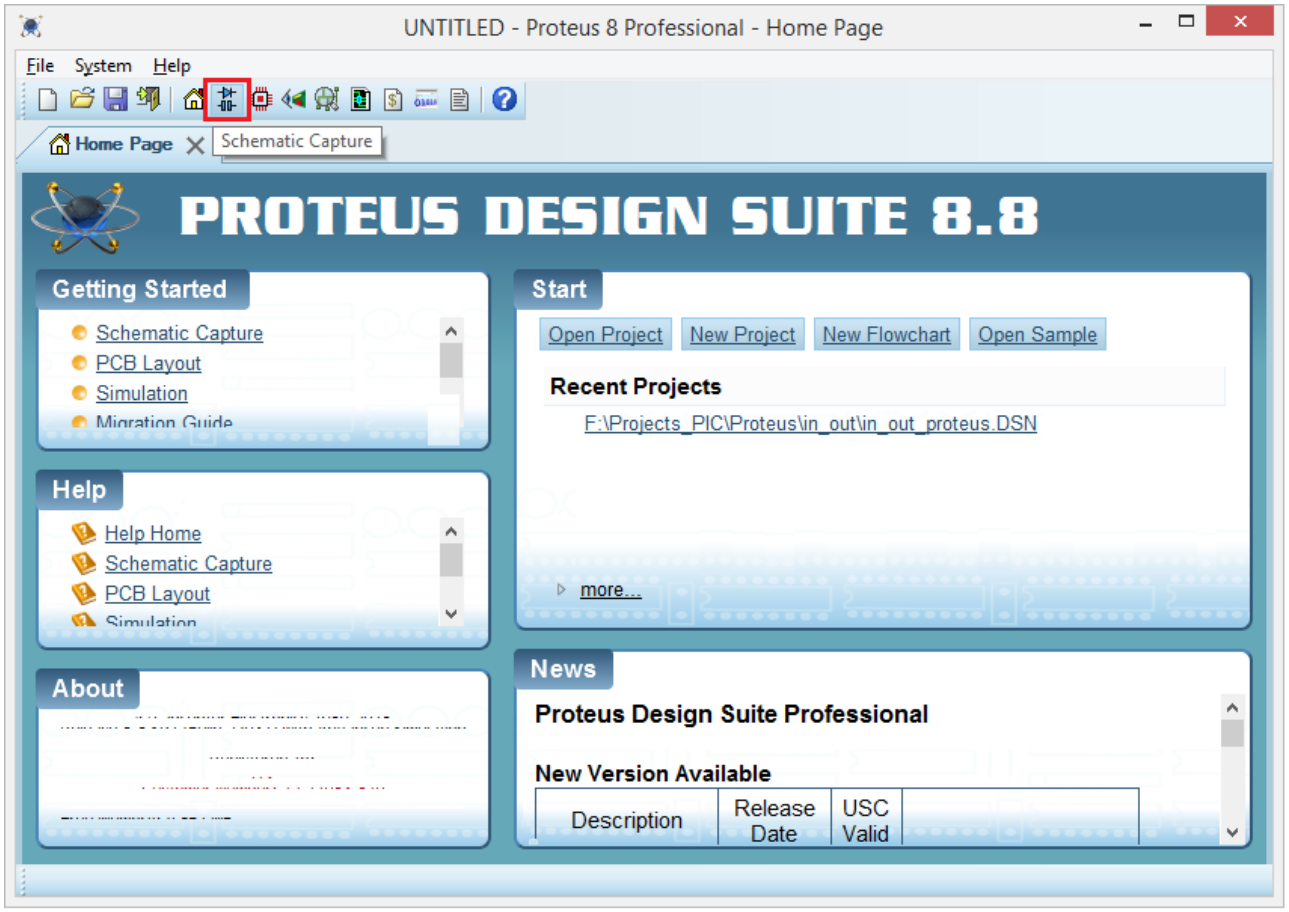

Рисунок 2.5 – Середовище моделювання «**Proteus**»

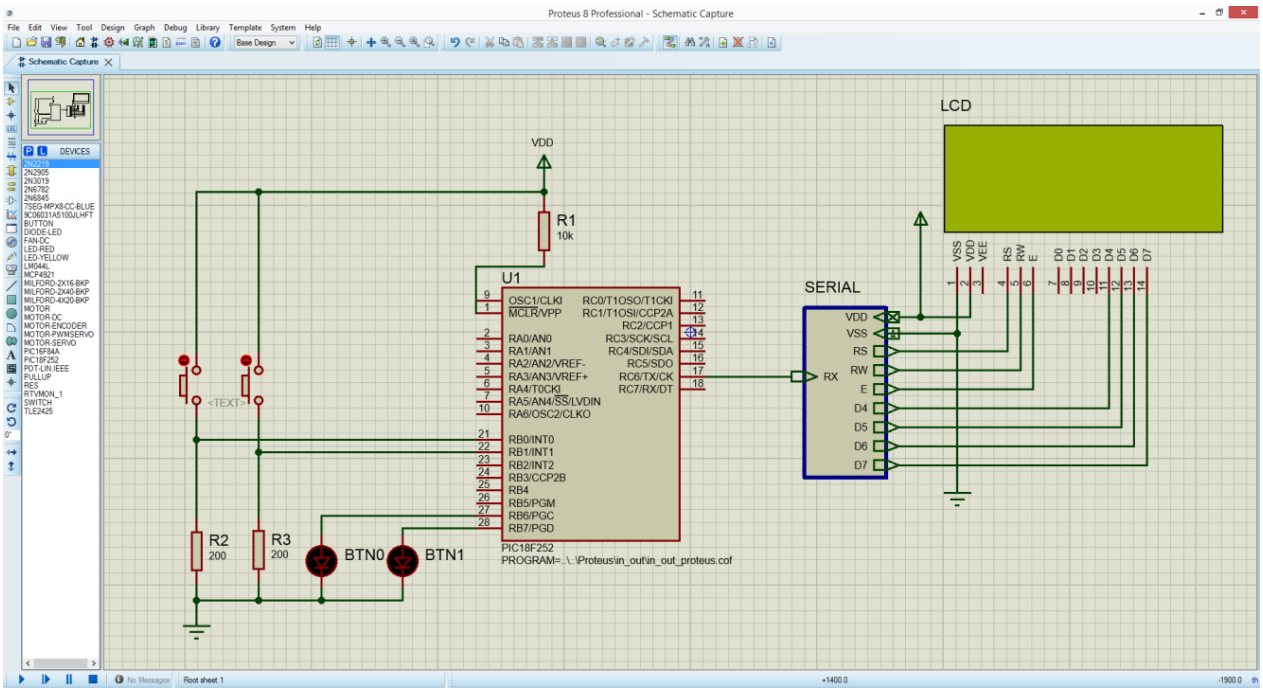

Рисунок 2.6 – Виконання лабораторної роботи
у середовищі моделювання «**Proteus**»

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.

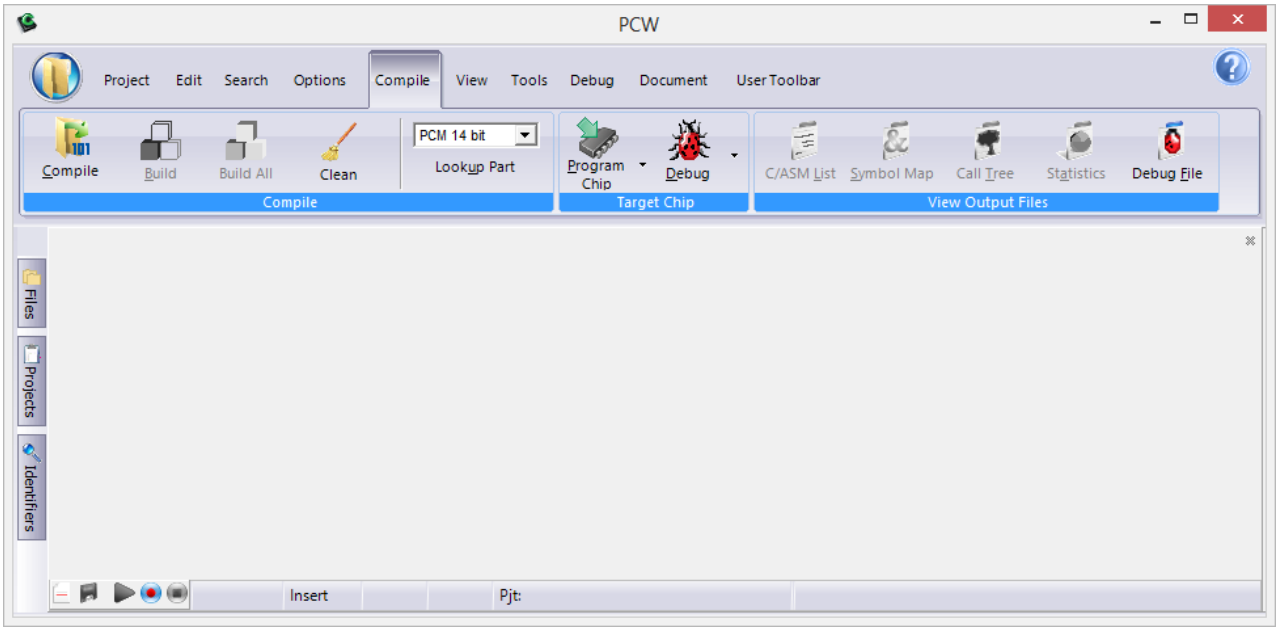

Рисунок 2.7 – Вікно інтегрованого середовища розробки **PCWH** при першому запуску

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),

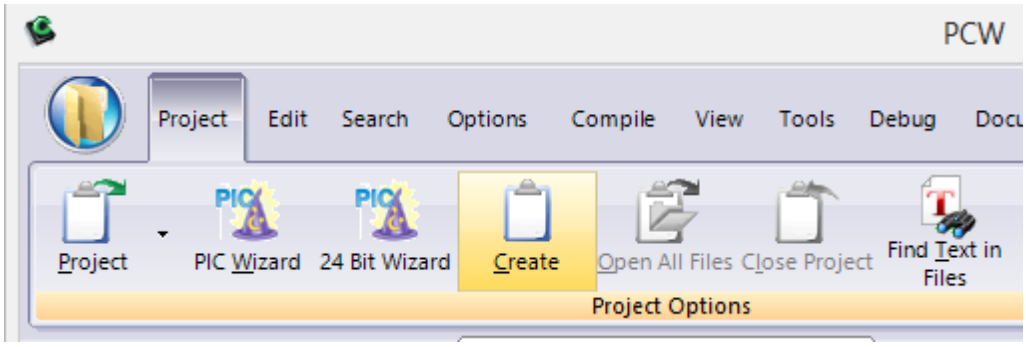

Рисунок 2.8 – Створення проекту у ручному режимі
за допомогою команди меню **Project / Create**

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).

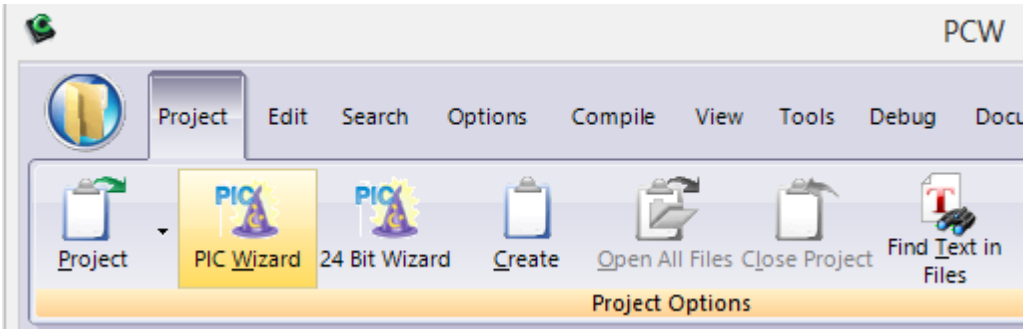

Рисунок 2.9 – Створення проекту автоматично
за допомогою майстра **PIC Wizard**

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).

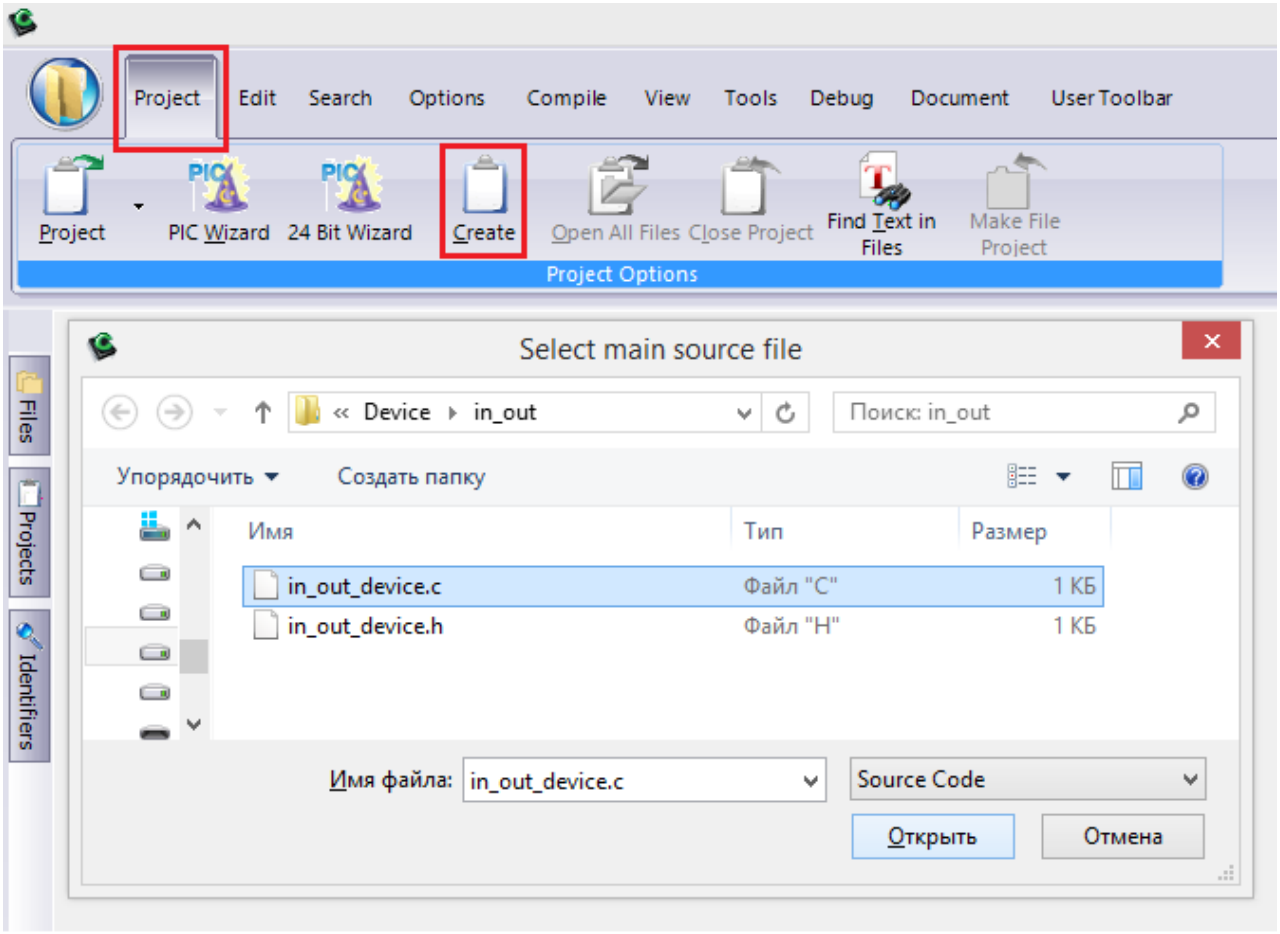

Рисунок 2.10 – Створення проекту в інтегрованому середовищі розробки **PCWH** у ручному режимі

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

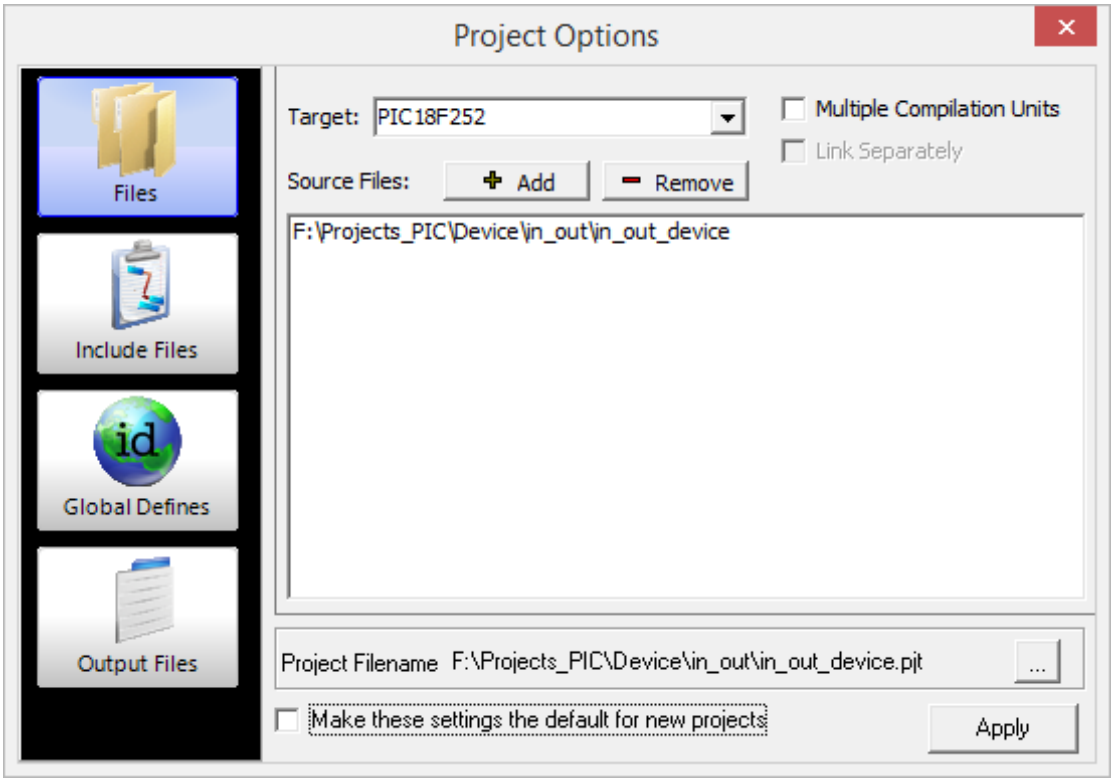

Рисунок 2.11 – Діалогове вікно **Project Options**

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

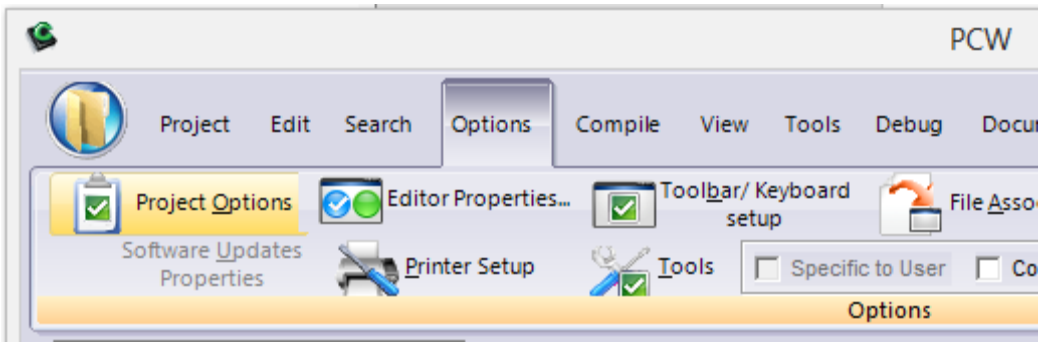

Рисунок 2.12 – Діалогове вікно **Project Options**

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).

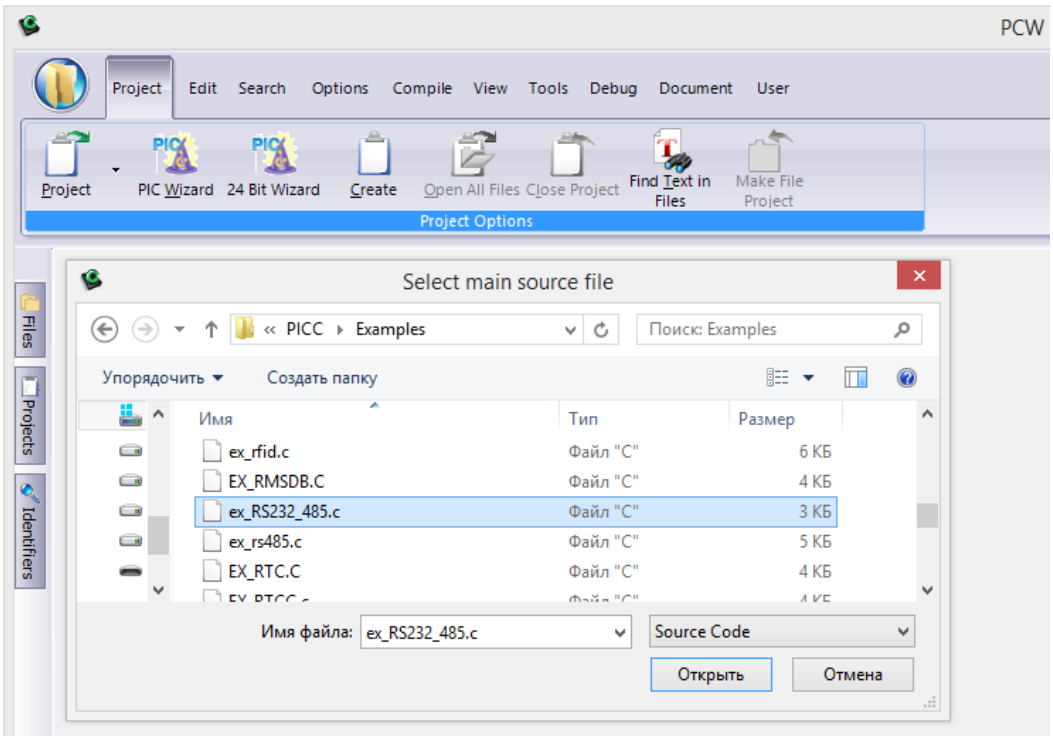

Рисунок 2.13 – Створення проекту «з нуля»

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

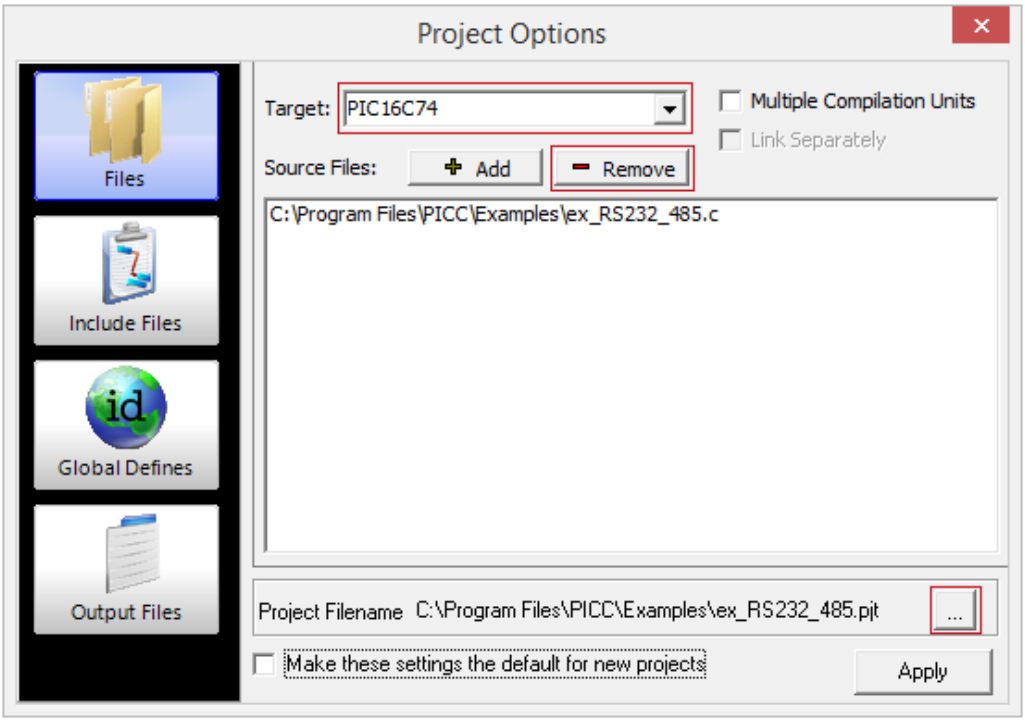

Рисунок 2.14 – Діалогове вікно **Project Options**

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).

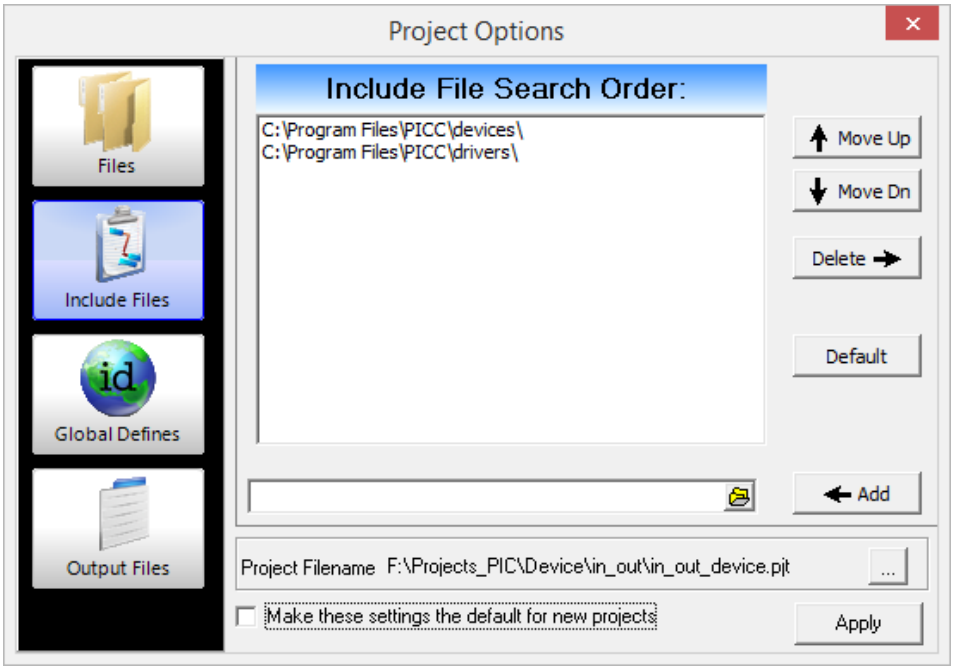

Рисунок 2.15 – Розділ **Include Files** діалогового вікна **Project Options**

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. 2.16). 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.

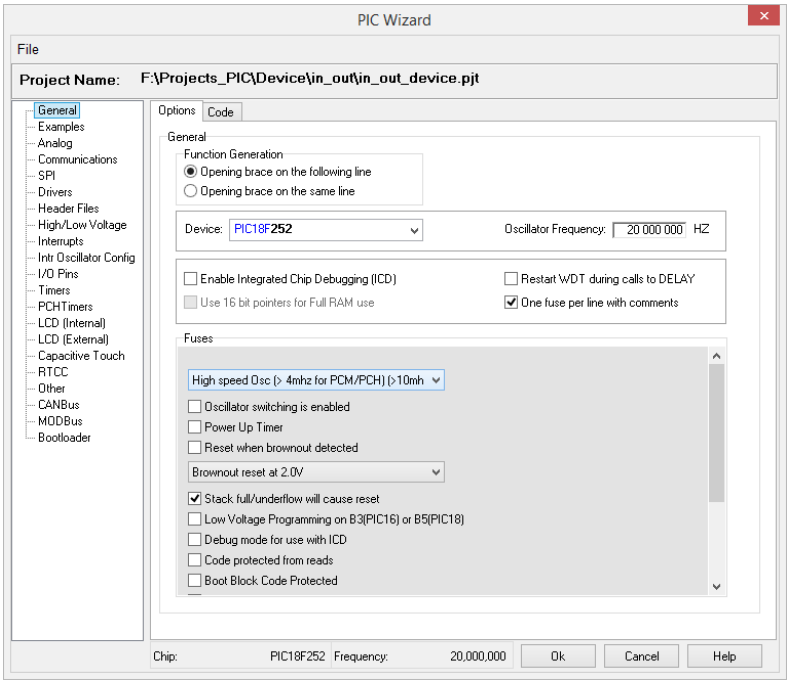

Рисунок 2.16 – Розділ **General** майстра **PIC Wizard**

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).

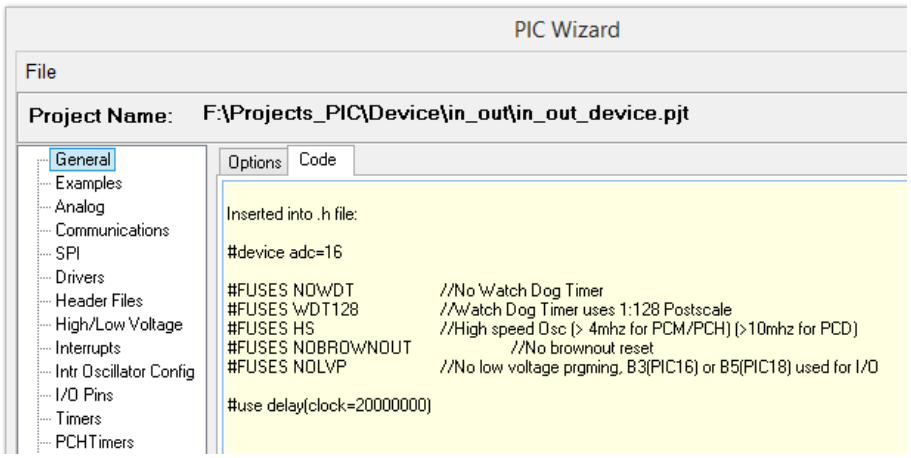

Рисунок 2.17 – Попередній перегляд коду, що генерується майстром **PIC Wizard** для поточного розділу

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**.

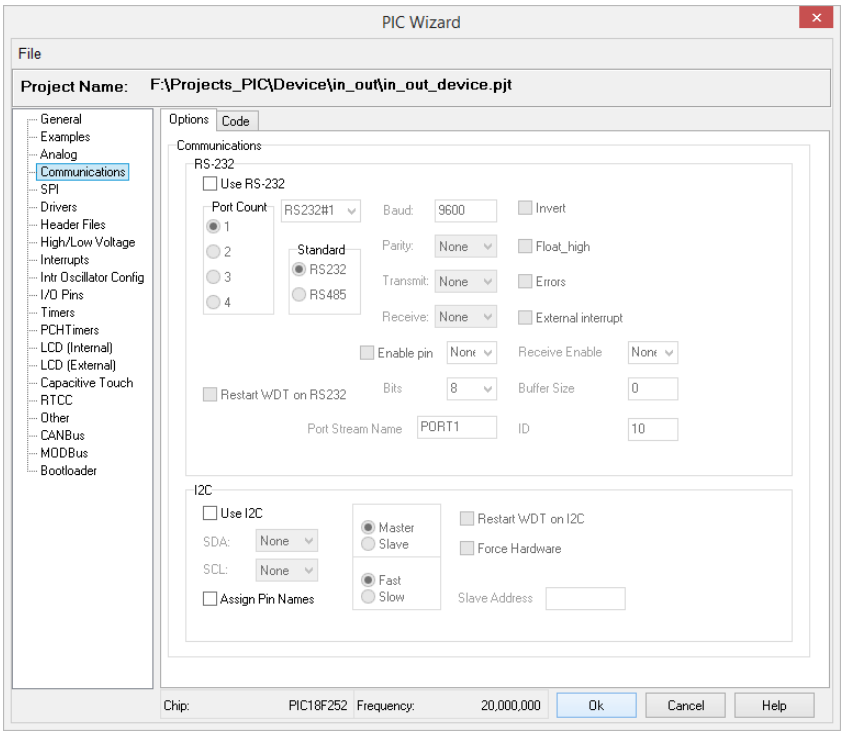

Рисунок 2.18 – Розділ **Communications** майстра **PIC Wizard**

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**).

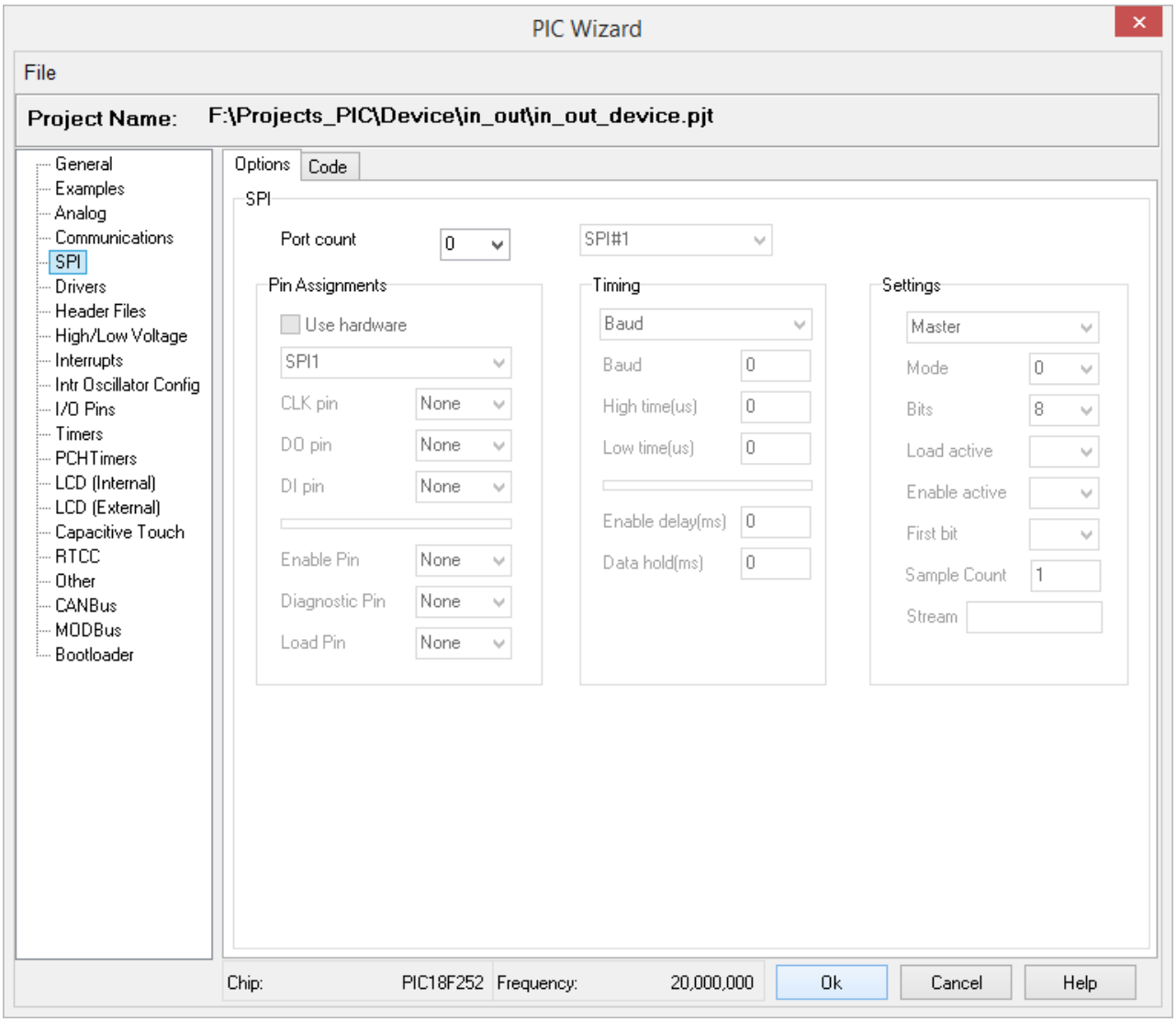

Рисунок 2.19 – Розділ **SPI** майстра **PIC Wizard**

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

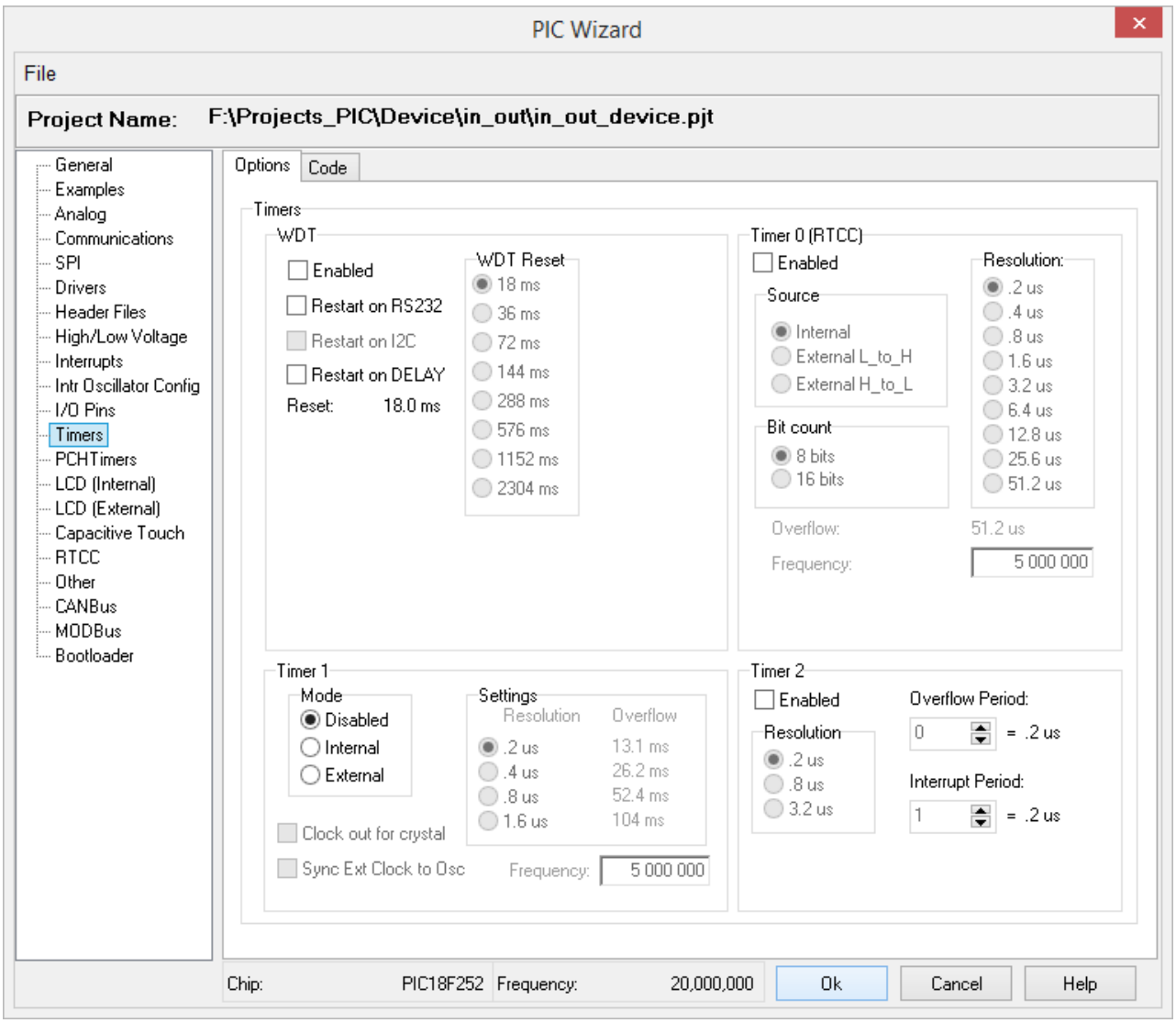

Рисунок 2.20 – Розділ **Timers** майстра **PIC Wizard**

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).

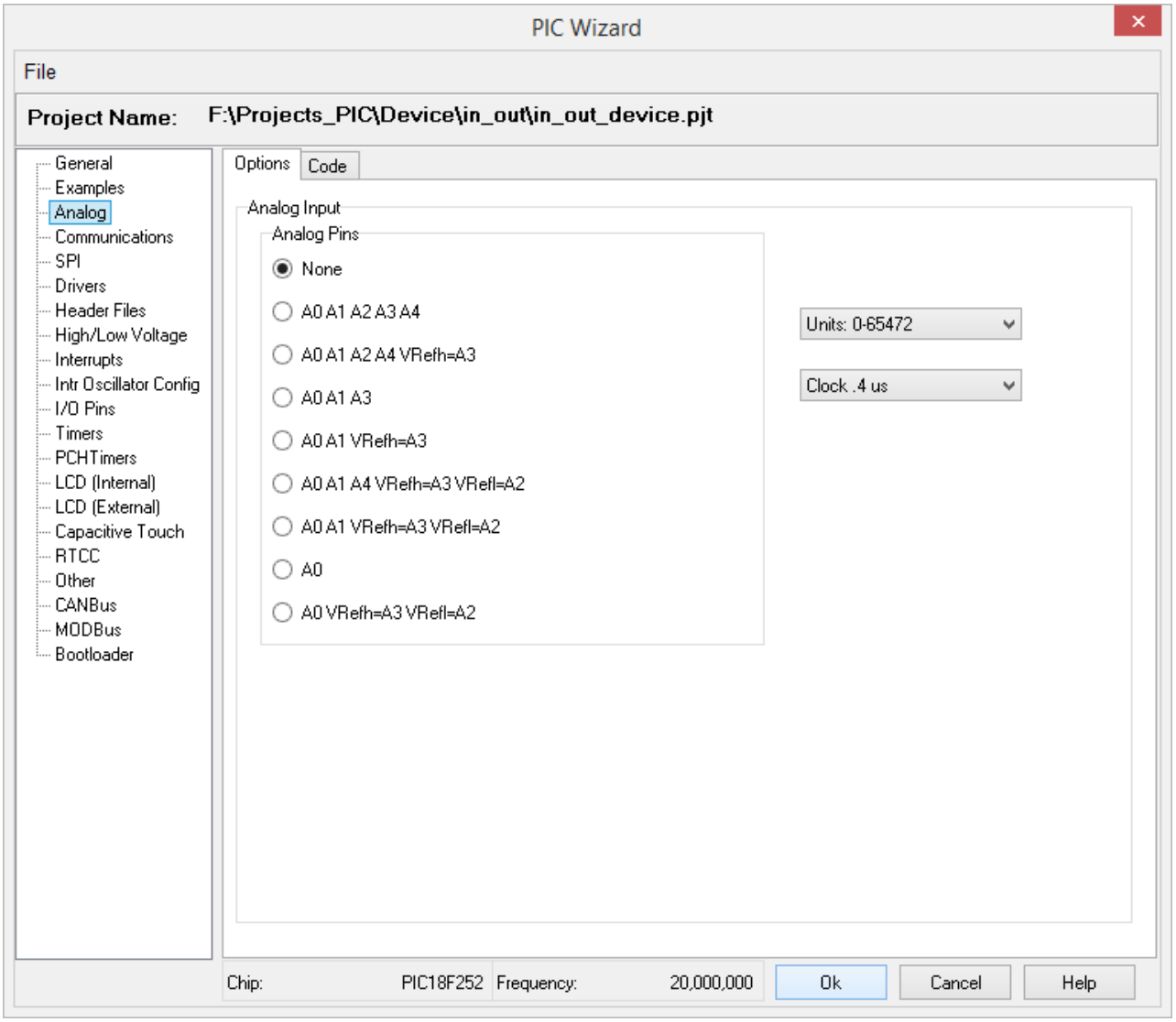

Рисунок 2.21 – Розділ **Analog** майстра **PIC Wizard**

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

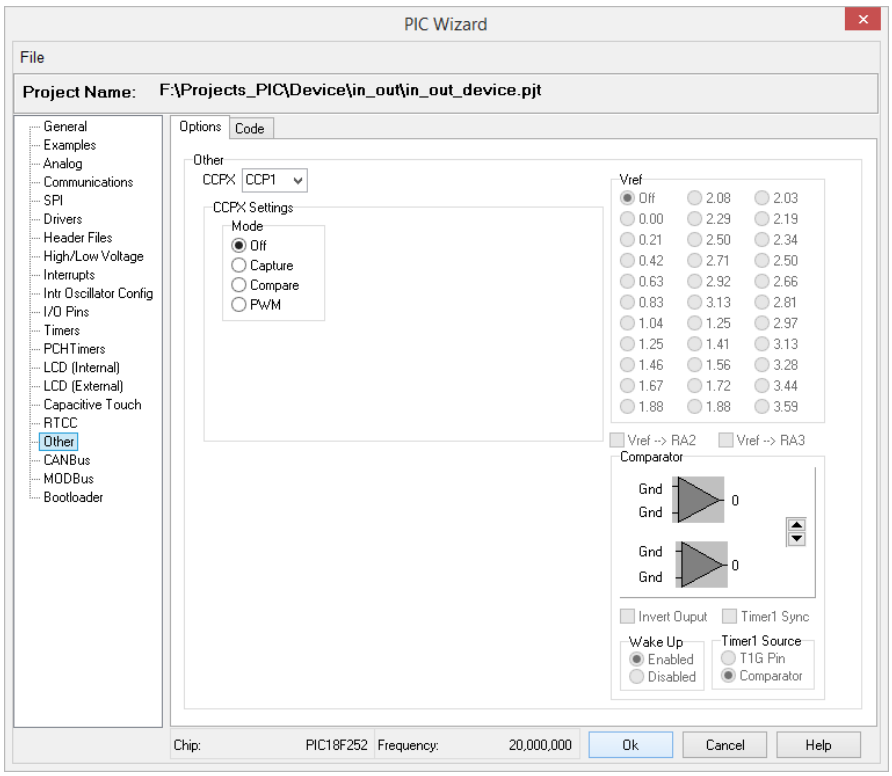

Рисунок 2.22 – Розділ **Other** майстра **PIC Wizard**

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

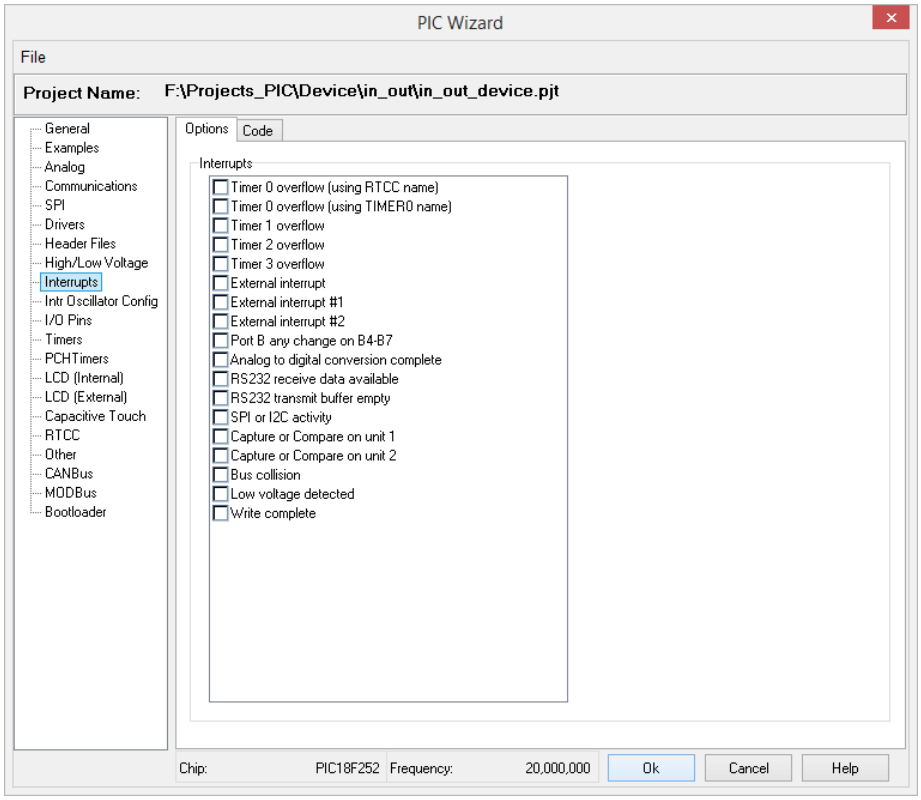

Рисунок 2.23 – Розділ **Interrupts** майстра **PIC Wizard**

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).

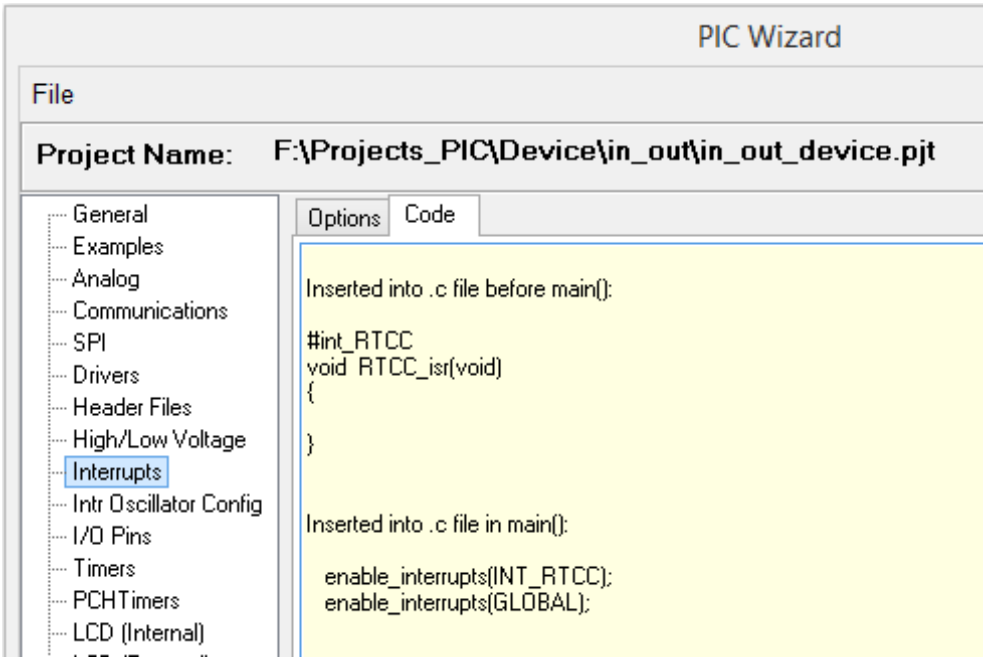

Рисунок 2.24 – Приклад програмного коду, згенерованого майстром **PIC Wizard** у розділі **Interrupts**

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).

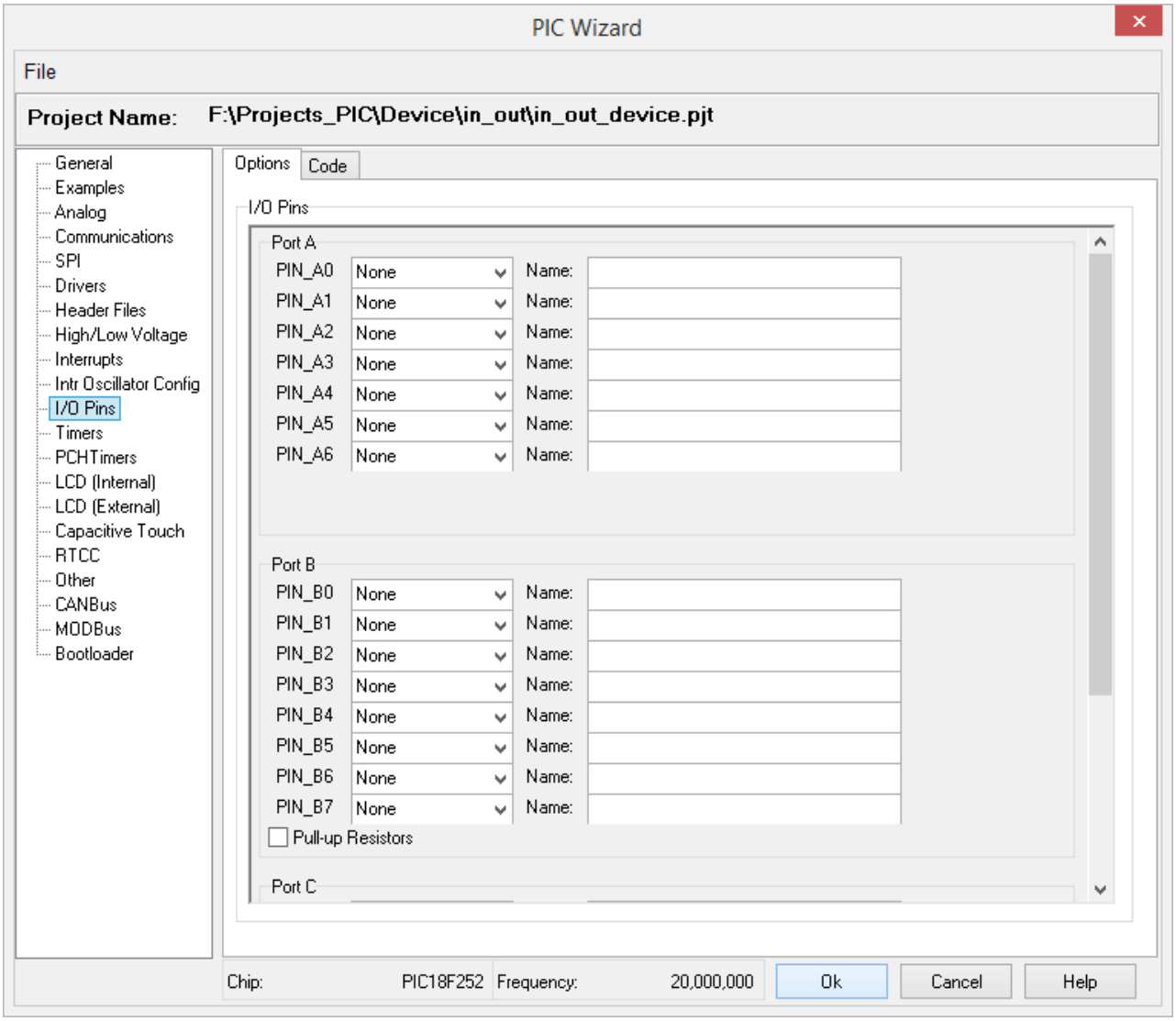

Рисунок 2.25 – Розділ **I/O Pins** майстра **PIC Wizard**

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).

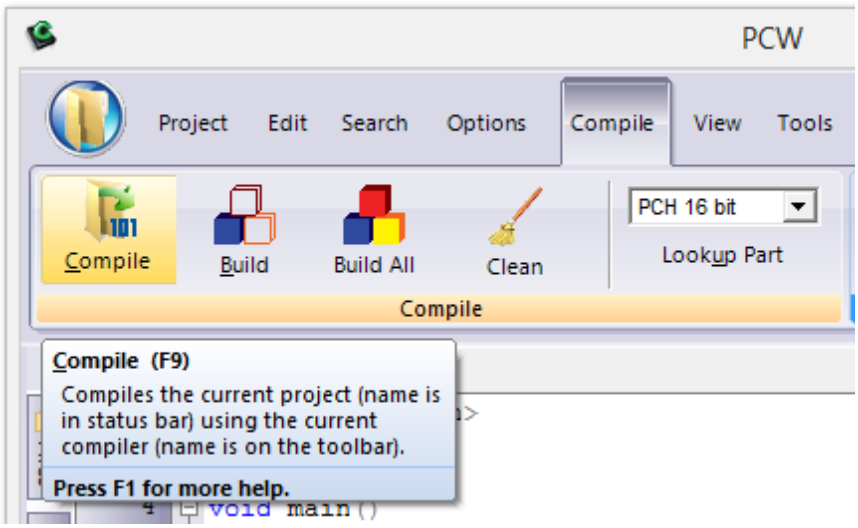

Рисунок 2.26 – Компіляція проекту

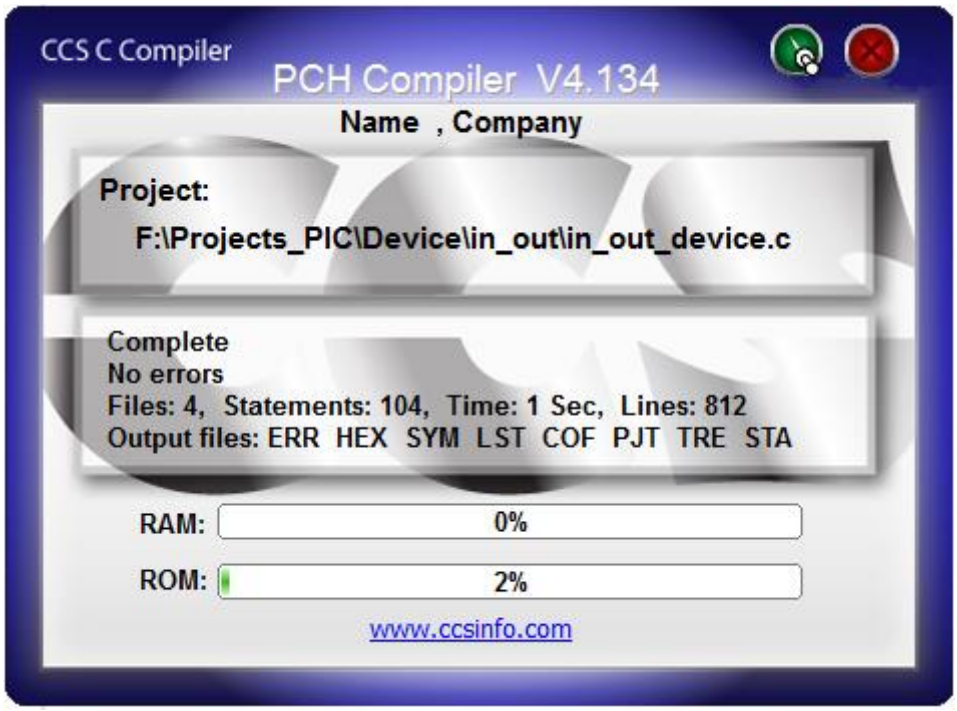

Рисунок 2.27 – Результати роботи компілятора **CCS-PICC**

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):

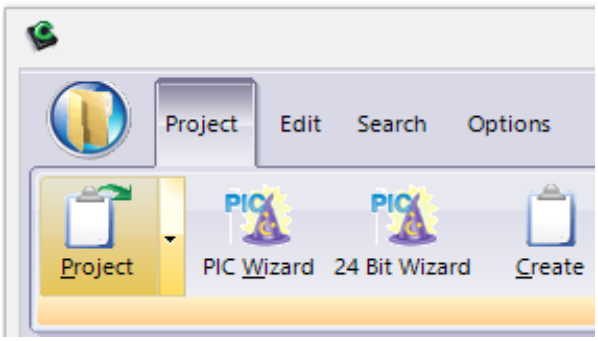

Рисунок 2.28 – Відкриття проекту у **IDE PCWH** за допомогою майстра **PIC Wizard**

або натиснути кнопку у вигляді відкритої папки  .

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

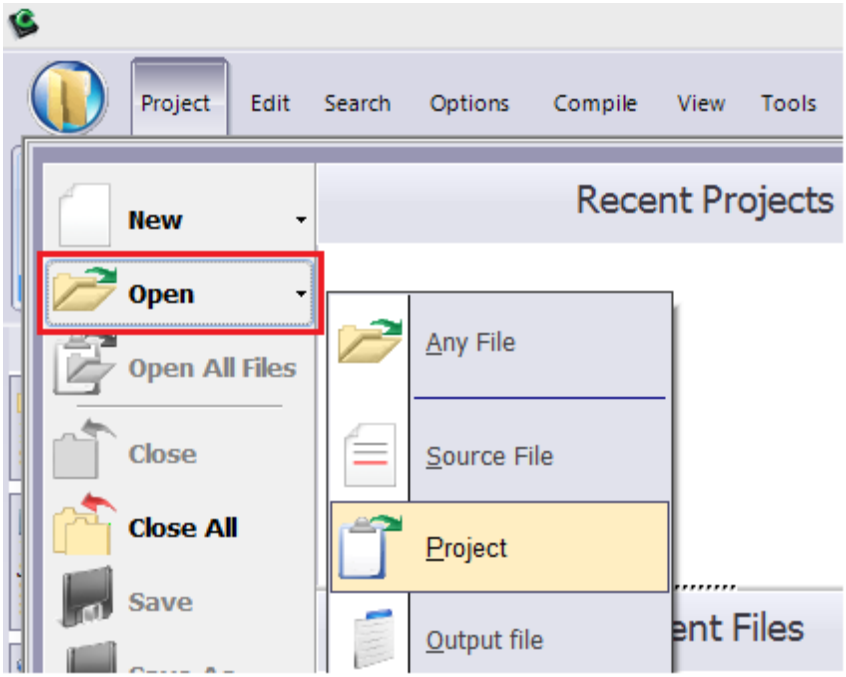

Рисунок 2.29 – Відкриття проекту у **IDE PCWH** за допомогою майстра **PIC Wizard**

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

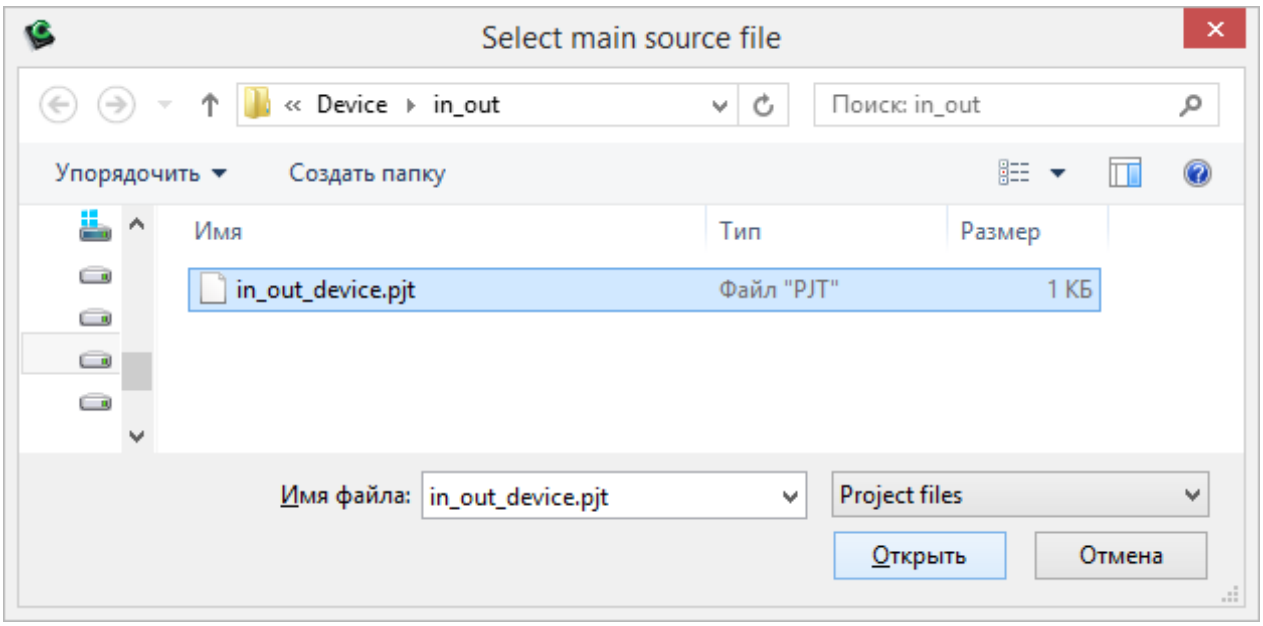

Рисунок 2.30 – Відкриття проекту у **IDE PCWH** за допомогою майстра **PIC Wizard**

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).

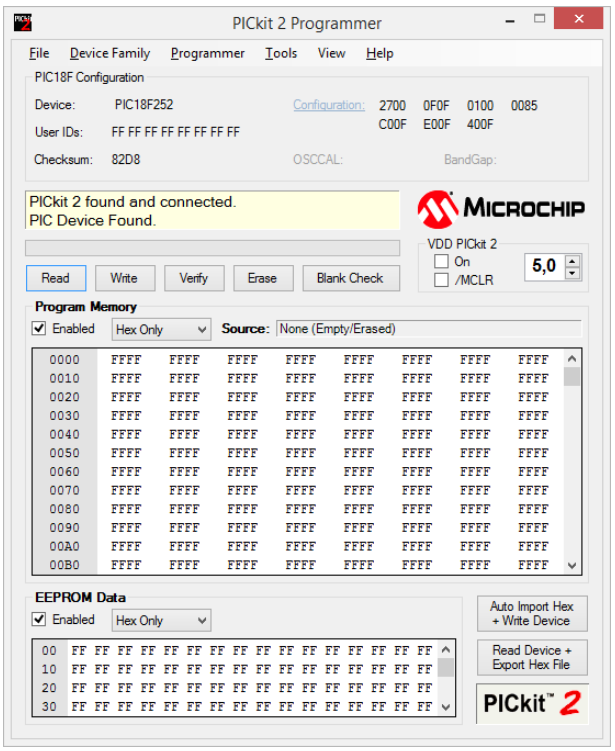

Рисунок 2.31 – Завантаження бінарного коду у FLASH-пам'ять мікроконтролера за допомогою програми **PICkit**

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):

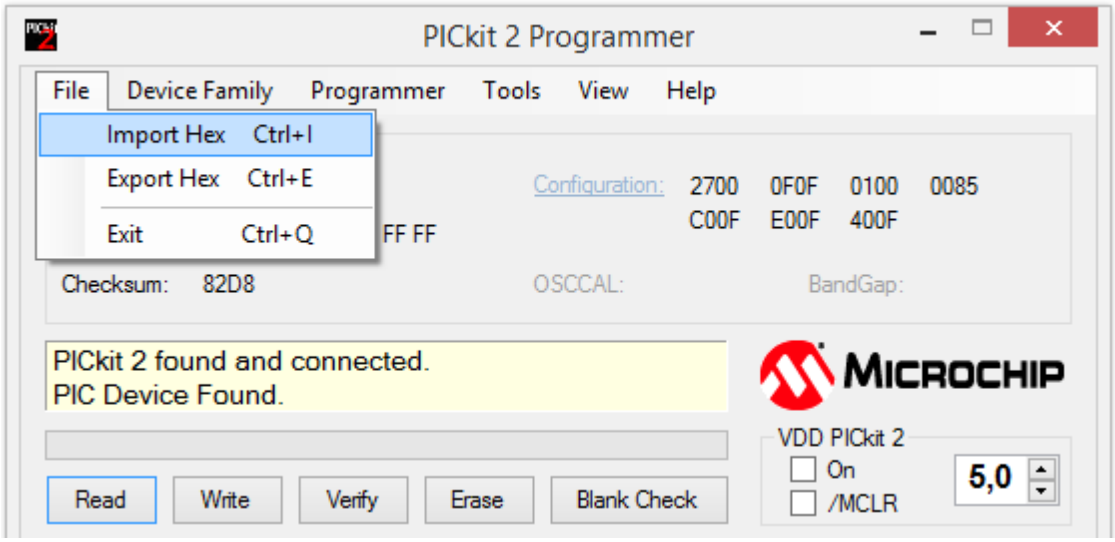

Рисунок 2.32 – Імпорт бінарного файлу «***.HEX**» за допомогою програми **PICkit**

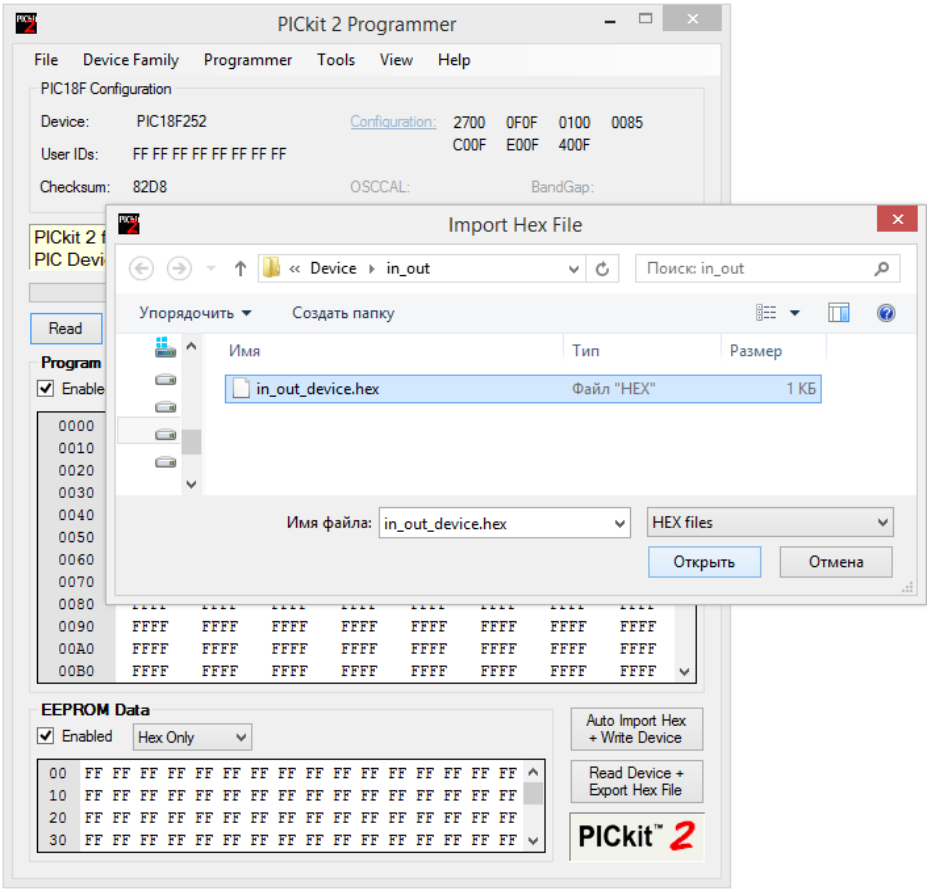

Рисунок 2.33 – Імпорт бінарного файлу «*.HEX»
за допомогою програми **PICkit**

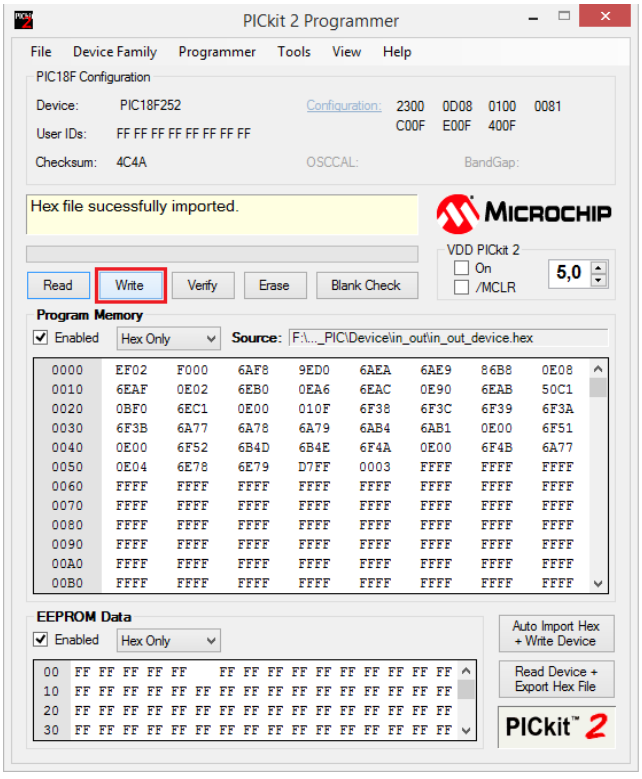

Рисунок 2.34 – Завантаження бінарного файлу «*.HEX»
у FLASH-пам'ять мікроконтролера

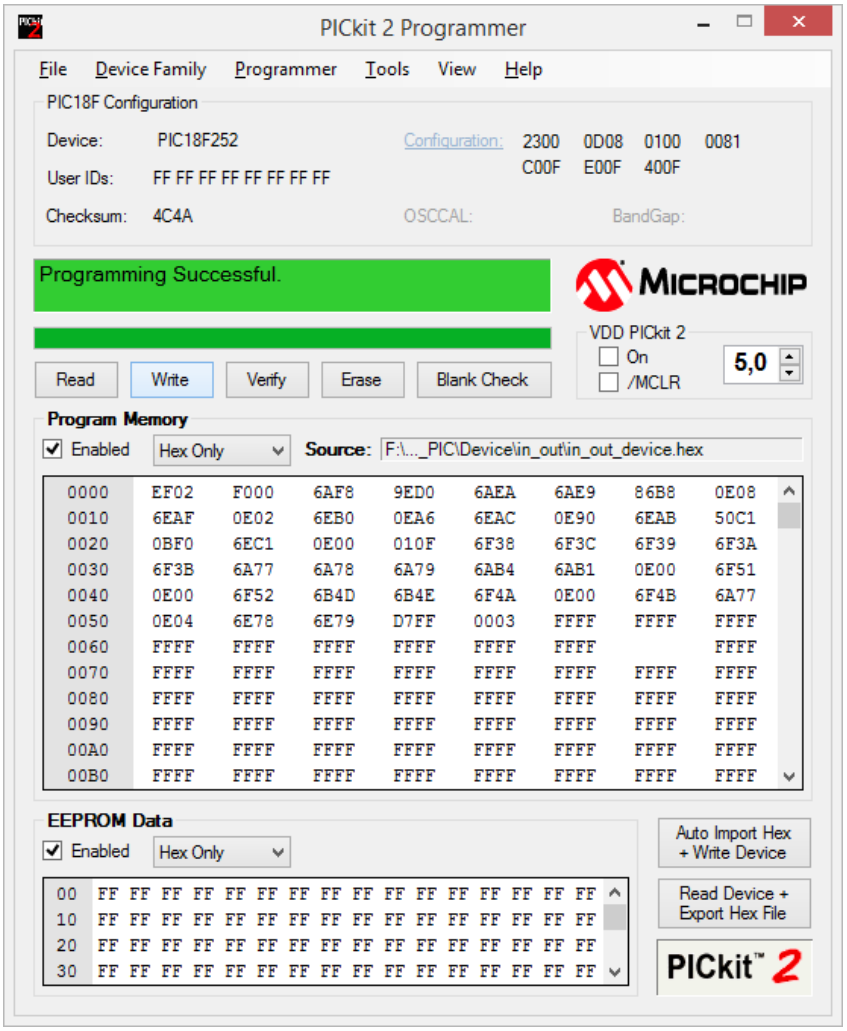

Рисунок 2.35 – Результат завантаження бінарного файлу **«*.HEX»** у FLASH-пам'ять мікроконтролера

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

ОПИС РОБОТИ З СЕРЕДОВИЩЕМ МОДЕЛЮВАННЯ «PROTEUS»

Порядок виконання лабораторних робіт для варіанту «Proteus»

1. У папці **«Proteus_students»** вибрати папку відповідної лабораторної роботи (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).

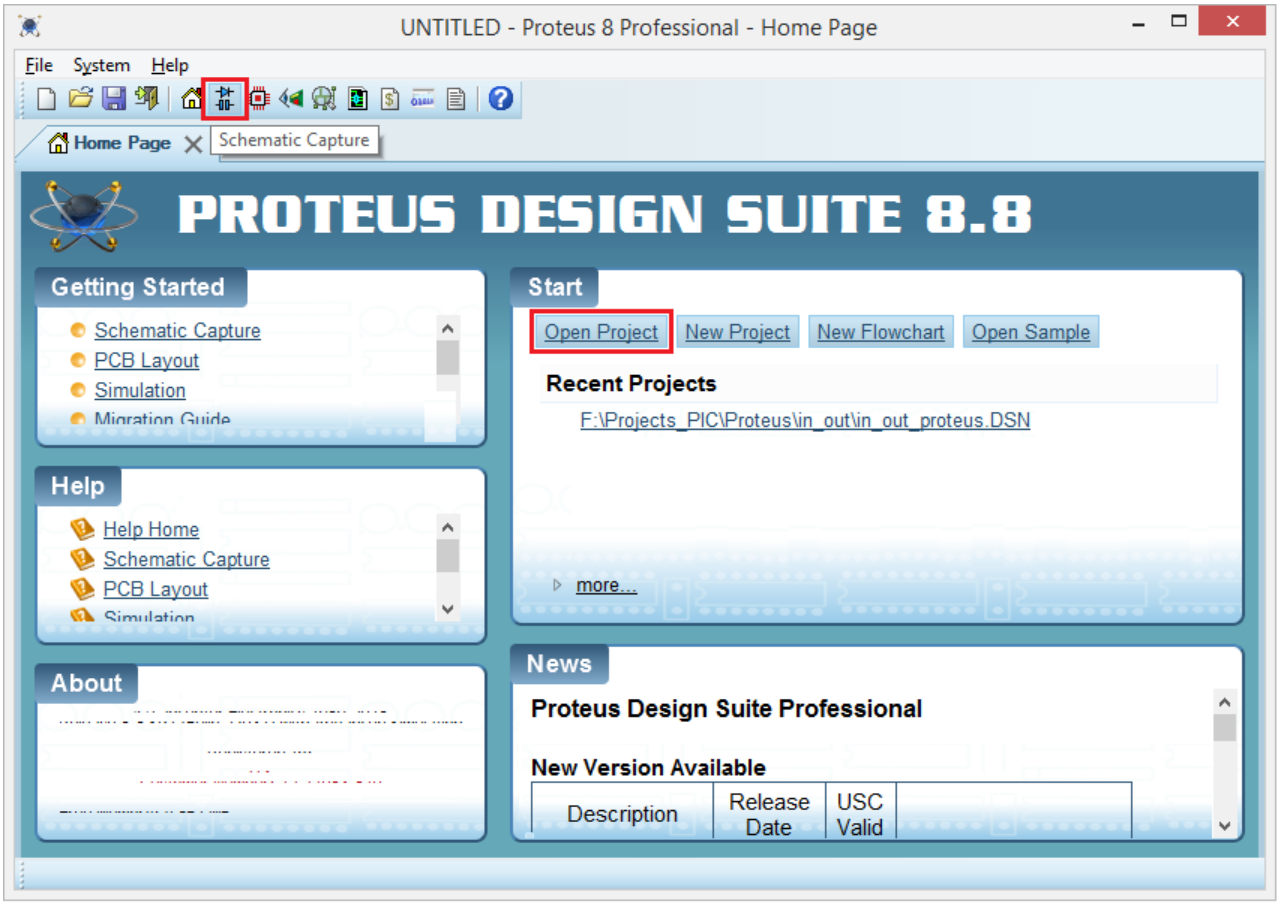

Рисунок 2.36 – Середовище моделювання **«Proteus»**

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

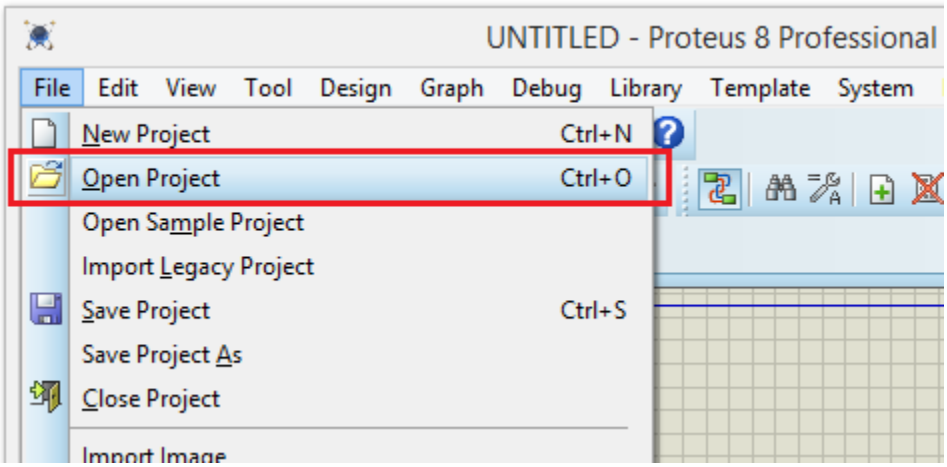

Рисунок 2.37 – Відкриття файлу проекту
у середовищі моделювання **«Proteus»**

3. У папці **«Proteus_students»** вибрати папку відповідної лабораторної роботи (наприклад, **«IN_OUT»**).

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

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

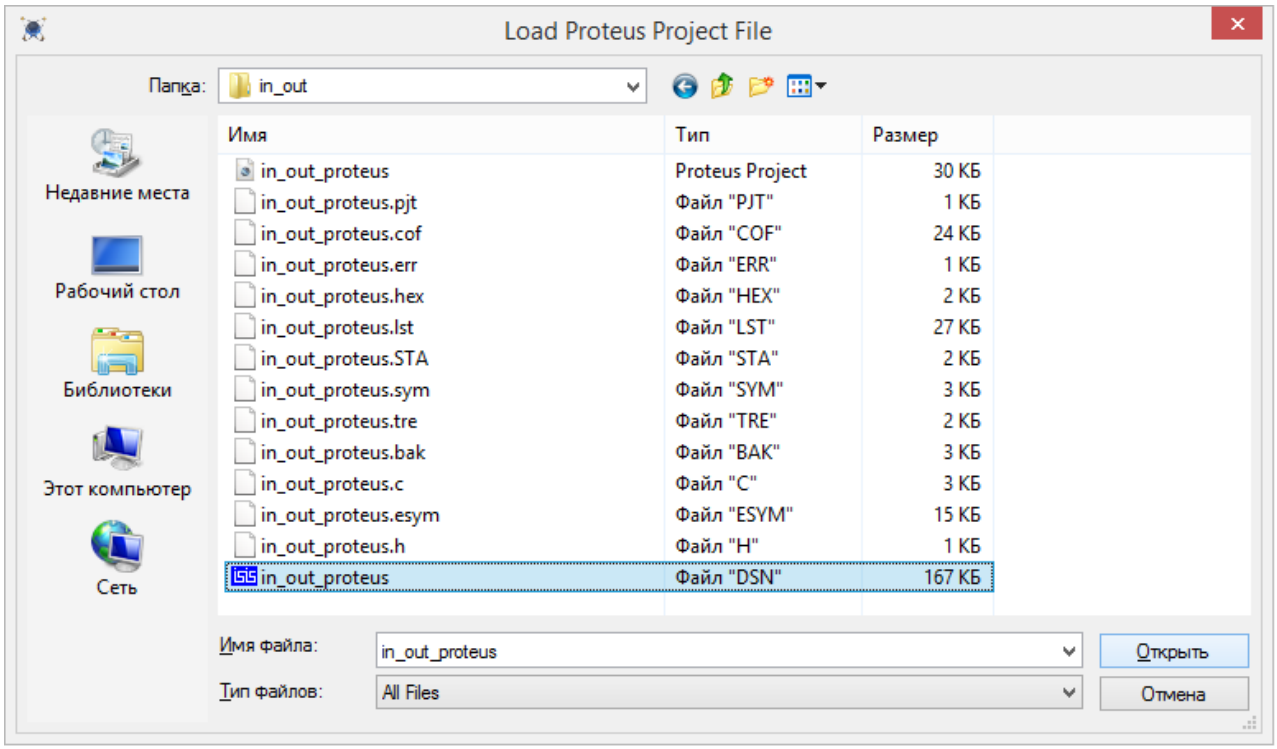

Рисунок 2.38 – Відкриття файлу проекту у середовищі моделювання **«Proteus»**

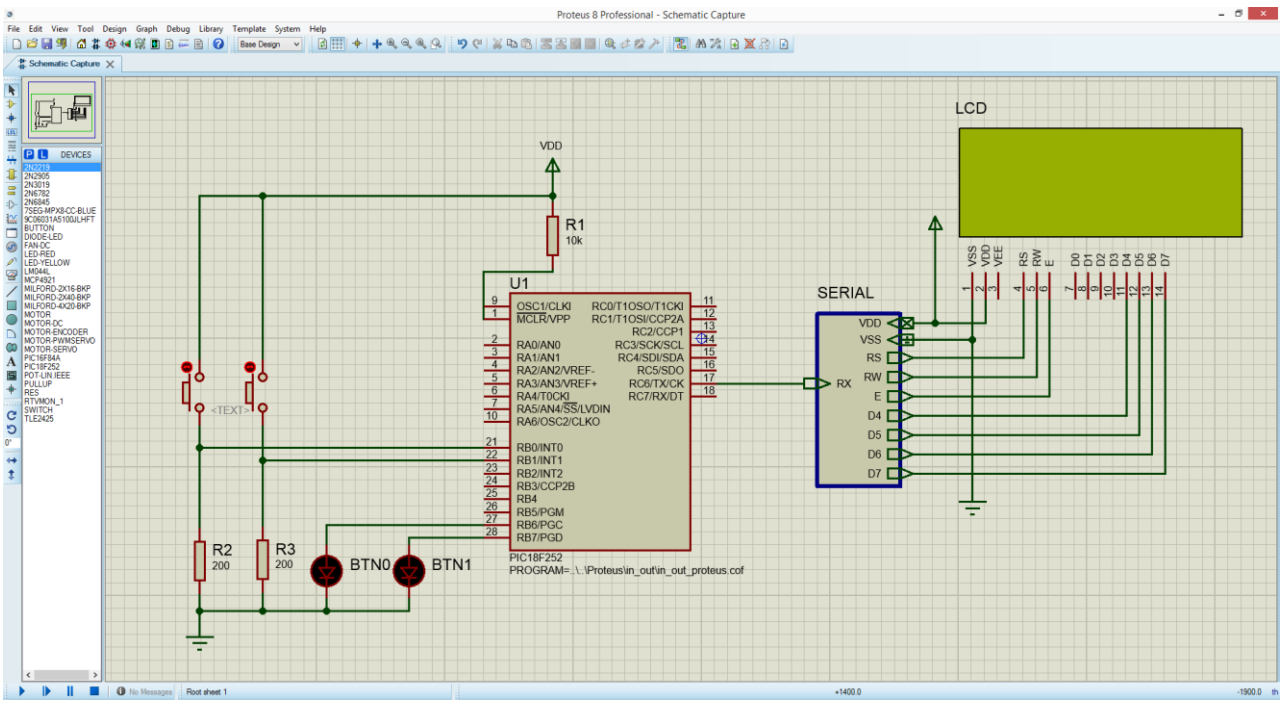

Рисунок 2.39 – Середовище моделювання **«Proteus»**

6. Змінити схему підключень відповідно до варіанту для виконання лабораторної роботи (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).

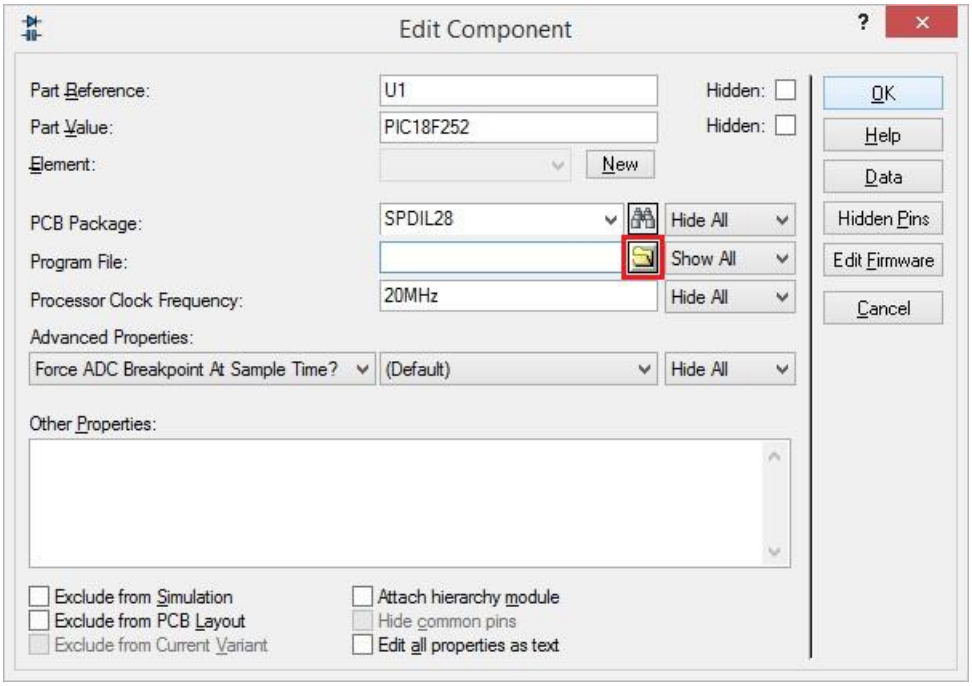

Рисунок 2.40 – Відкриття бінарного файлу з розширенням ***.HEX** або ***.COF** у середовищі моделювання **«Proteus»**

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

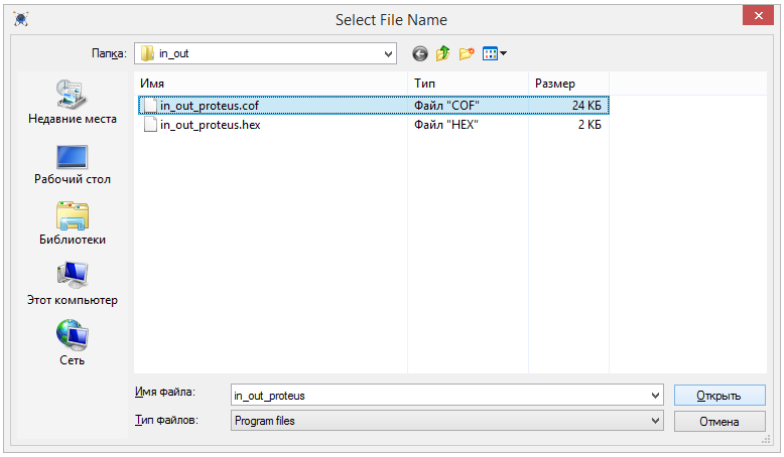

Рисунок 2.41 – Завантаження бінарного файлу з розширенням ***.HEX** або ***.COF** у середовищі моделювання **«Proteus»**

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

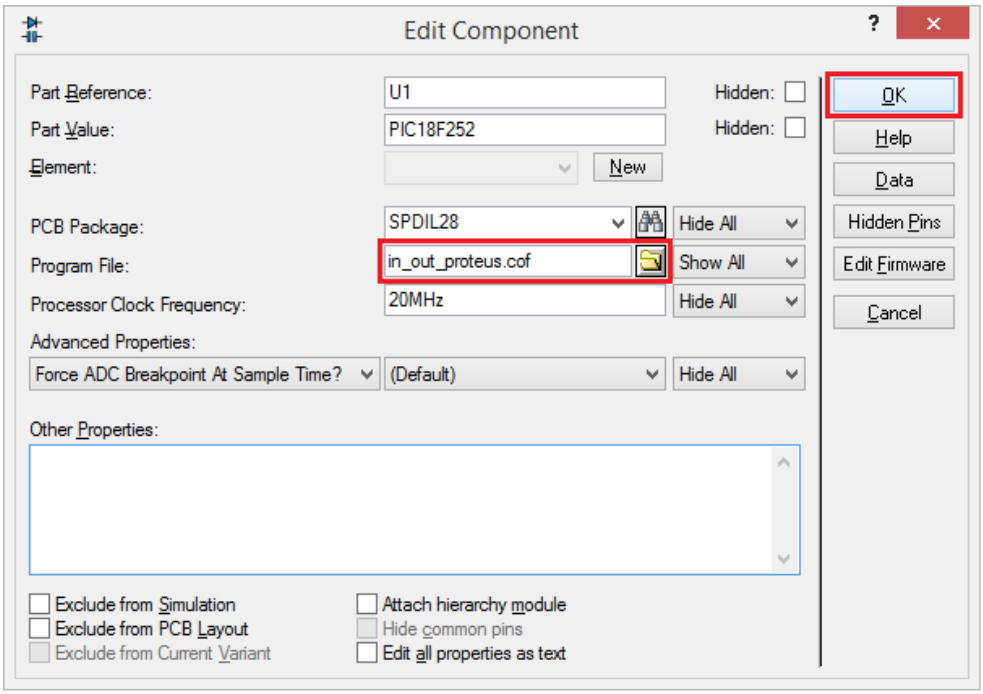

Рисунок 2.42 – Завантаження бінарного файлу з розширенням ***.HEX** або ***.COF** у середовищі моделювання **«Proteus»**

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

РОЗДІЛ 3. ЛАБОРАТОРНІ РОБОТИ

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

Всі лабораторні роботи 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 робіт

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

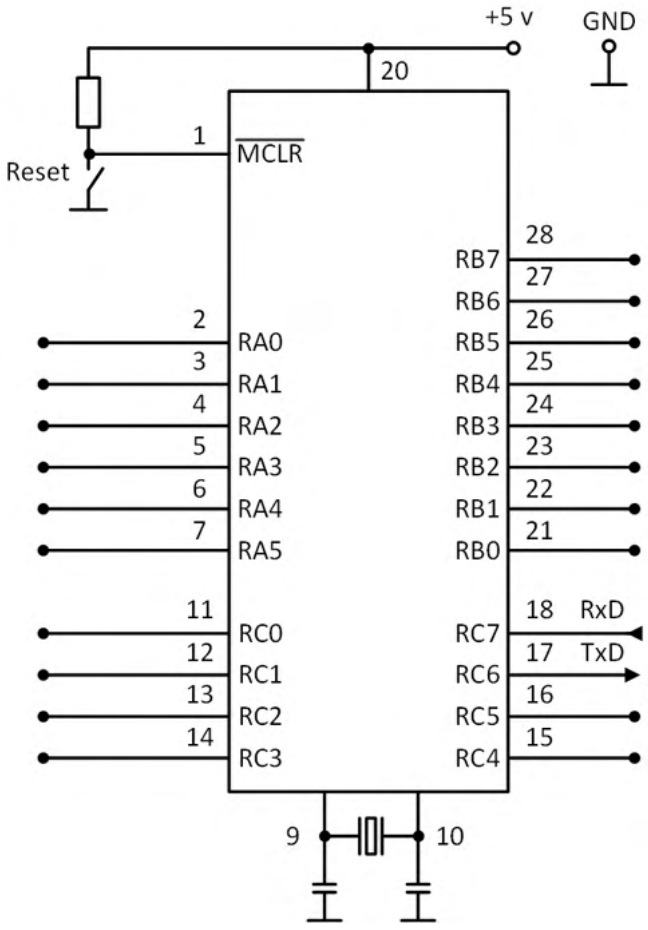

Рисунок 3.1 – Шаблон розробки принципової схеми лабораторних робіт для варіанту **АПК**

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

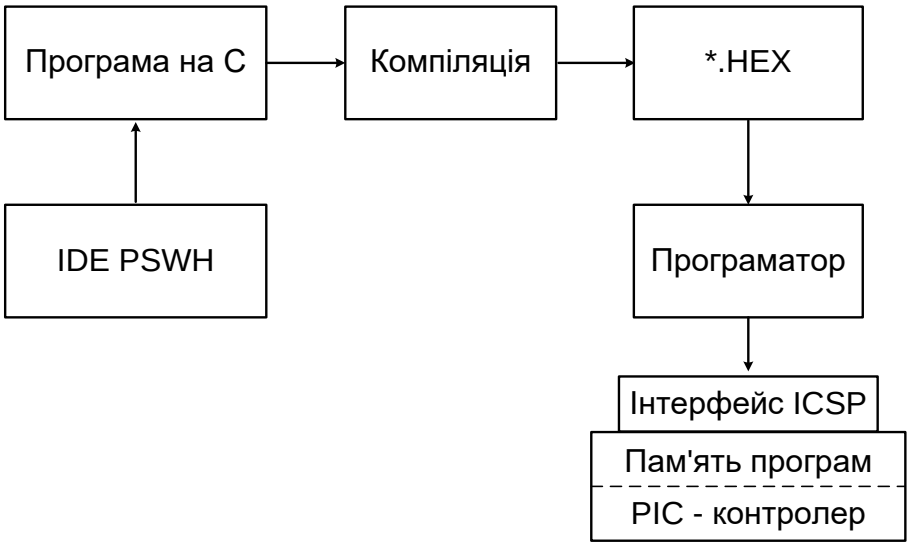

Рисунок 3.2 – Процес розробки програм для варіанту **АПК**

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

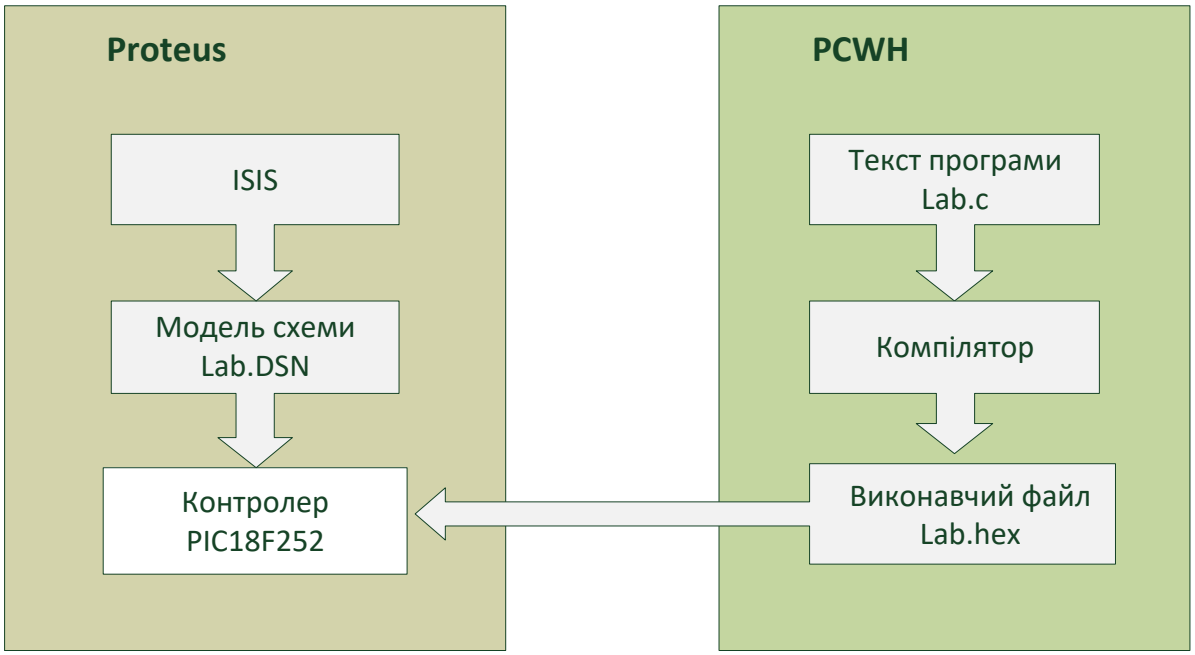

Рисунок 3.3 – Процес розробки програм для **варіанту «Proteus»**

ЛАБОРАТОРНА РОБОТА № 1 - «LED_DISPLAY»

Тема: Динамічна індикація. Виведення інформації на LED – дисплей

Ціль роботи:

Отримання навичок роботи із багаторозрядними LED-дисплеями і написання драйвера динамічної індикації для LED-дисплея.

Завдання лабораторної роботи

- для варіанту **АПК** намалювати принципову схему підключень відповідно до варіанту для виконання лабораторної роботи (таб. 3.1.1);
- для варіанту **«Proteus»** використовувати готову схему відповідно до варіанту для виконання лабораторної роботи;
- створити алгоритм програми роботи з LED-дисплеєм;
- написати програму драйвера динамічної індикації;
- здійснити виведення результатів на LCD відповідно до варіанту для виконання лабораторної роботи.

Виведення на LCD повинне містити:

- номер лабораторної роботи;
- групу студента;
- прізвище та ініціали студента;
- 3-х значне число лічильника секунд.

Варіанти для виконання лабораторної роботи «LED_display»

Таблиця 3.1.1 – Варіанти для виконання лабораторної роботи **«LED_display»**

	Число	**Керування**				**Дані**							
№	**N1 і N2**	**C0**	**C1**	**C2**	**C3**	**B0**	**B1**	**B2**	**B3**	**B4**	**B5**	**B6**	**B7**
1	N1 = № · 5 N2 = 150 - №	D1	D2	D3		A	B	C	D	E	F	G	DP
2			D1	D2	D3	A	B	C	D	E	F	G	DP
3		D1	D2	D3		DP	G	F	E	D	C	B	A
4			D1	D2	D3	DP	G	F	E	D	C	B	A
5		D1	D2	D3		A	B	C	D	E	F	G	DP
6			D1	D2	D3	A	B	C	D	E	F	G	DP
7		D1	D2	D3		DP	G	F	E	D	C	B	A
8			D1	D2	D3	DP	G	F	E	D	C	B	A
9		D1	D2	D3		A	B	C	D	E	F	G	DP
10			D1	D2	D3	A	B	C	D	E	F	G	DP
11		D1	D2	D3		DP	G	F	E	D	C	B	A
12			D1	D2	D3	DP	G	F	E	D	C	B	A
13		D1	D2	D3		A	B	C	D	E	F	G	DP
14			D1	D2	D3	A	B	C	D	E	F	G	DP
15		D1	D2	D3		DP	G	F	E	D	C	B	A

Заповнити таблицю 3.1.2 кодами цифр у 7-сегментному представленні десяткових цифр:

Таблиця 3.1.2 – 7-сегментне представлення десяткових цифр

Цифра	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**DP**
0	0	0	0	0	0	0	1	1
1	1	0	0	1	1	1	1	1
…								
8	0	0	0	0	0	0	1	1
9	0	0	0	0	1	0	0	1
DP	1	1	1	1	1	1	1	0

Вивести на LED-дисплей значення лічильника в інтервалі **N1 – N2** з періодом в 1 сек.

Значення N1 і N2:

- **N1** = номер за списком * 5;
- **N2** = 150 – номер за списком.

Примітка: Відображення цифр на LED-дисплеї можна здійснювати декількома варіантами:

a) конструкцією **if-then**;
b) конструкцією **switch**;
c) табличним методом і т.д.

Послідовність виконання лабораторної роботи для варіанту АПК

1) для варіанту **АПК** намалювати принципову схему підключень відповідно до варіанту для виконання лабораторної роботи;
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 LED-дисплеєм;
6) написати програму мовою програмування **C**, що реалізує створений алгоритм, відповідно до варіанту для виконання лабораторної роботи;
7) скомпілювати програму, одержати бінарні файли ***.HEX** і ***.COF** (файли з розширеннями ***.HEX** і ***.COF** створюються підчас компіляції програми);
8) завантажити бінарний файл ***.HEX** у пам'ять програм мікроконтролера програмою **PICkit.exe**;
9) на **АПК** зкумутувати схему підключень LED-дисплею відповідно до варіанту для виконання лабораторної роботи;
10) виконати програму.

Послідовність виконання лабораторної роботи 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 LED-дисплеєм;
6) написати програму мовою програмування **C**, що реалізує створений алгоритм, відповідно до варіанту для виконання лабораторної роботи;
7) скомпілювати програму, одержати бінарні файли ***.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 лабораторної роботи **«LED_display»**;
10) відкрити файл проекту з розширенням ***.DSN**;
11) у середовищі моделювання **«Proteus»** змінити схему підключень відповідно до варіанту для виконання лабораторної роботи;
12) завантажити заздалегідь скомпільований бінарний файл ***.COF** або ***.HEX** у мікроконтролер;
13) у середовищі моделювання **«Proteus»** виконати програму.

Послідовність виконання лабораторної роботи 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 лабораторної роботи **«LED_display»**;
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 лабораторної роботи **«LED_display»**;
9) відкрити файл проекту з розширенням ***.DSN**;
10) у середовищі моделювання **«Proteus»** змінити схему підключень відповідно до варіанту для виконання лабораторної роботи;
11) створити алгоритм програми роботи з LED-дисплеєм;
12) у інтегрованому середовищі розробки **PCWH**, використовуючи шаблон програми самостійно написати програму мовою програмування **C**, що реалізує створений алгоритм, відповідно до варіанту для виконання лабораторної роботи;
13) скомпілювати програму, одержати бінарні файли ***.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 лабораторної роботи «**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 PCWH**, натиснути кнопку **Projects** майстра **PIC Wizard** (рис. 3.1.1):

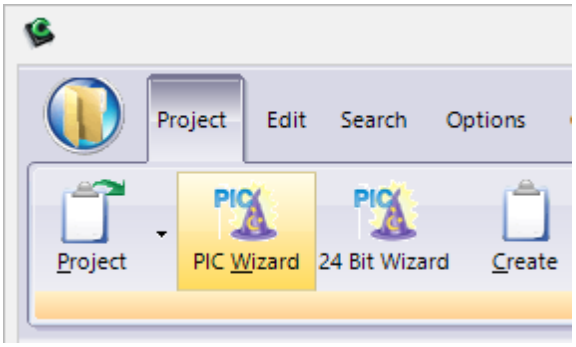

Рисунок 3.1.1 – Процес створення проекту у **IDE PCWH** за допомогою майстра **PIC Wizard**

або натиснути кнопку у вигляді відкритої папки  .

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

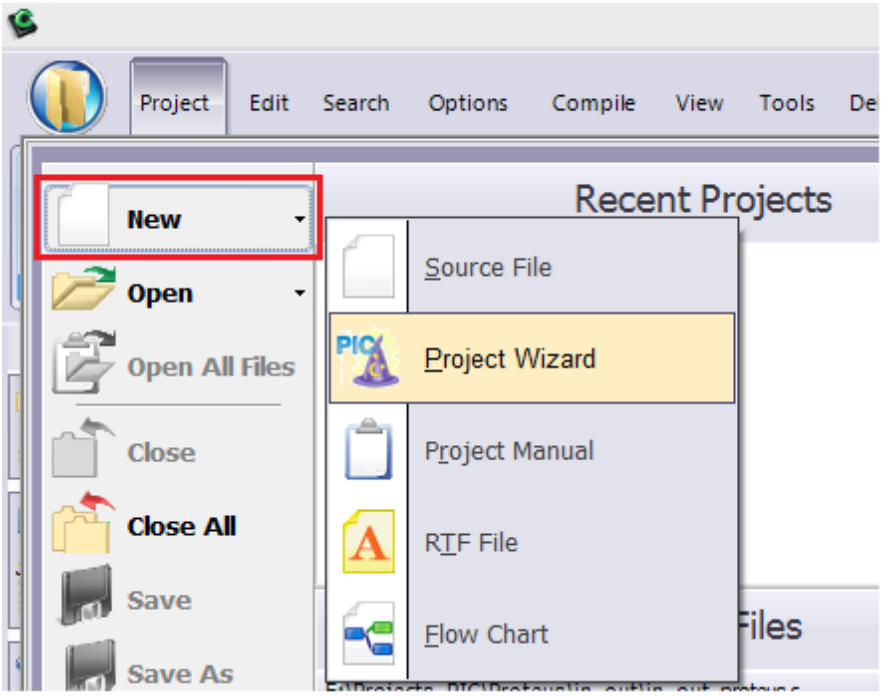

Рисунок 3.1.2 – Процес створення проекту у **IDE PCWH** за допомогою майстра **PIC Wizard**

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

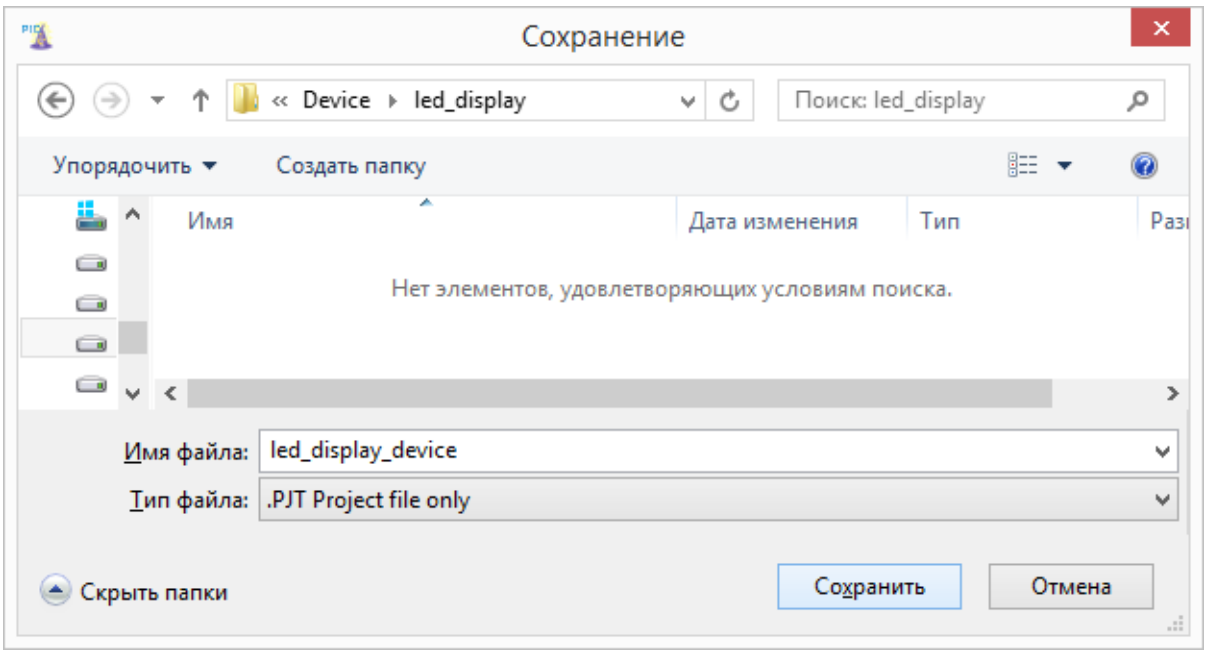

Рисунок 3.1.3 – Процес створення проекту у **IDE PCWH** за допомогою майстра **PIC Wizard**

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.

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)**:

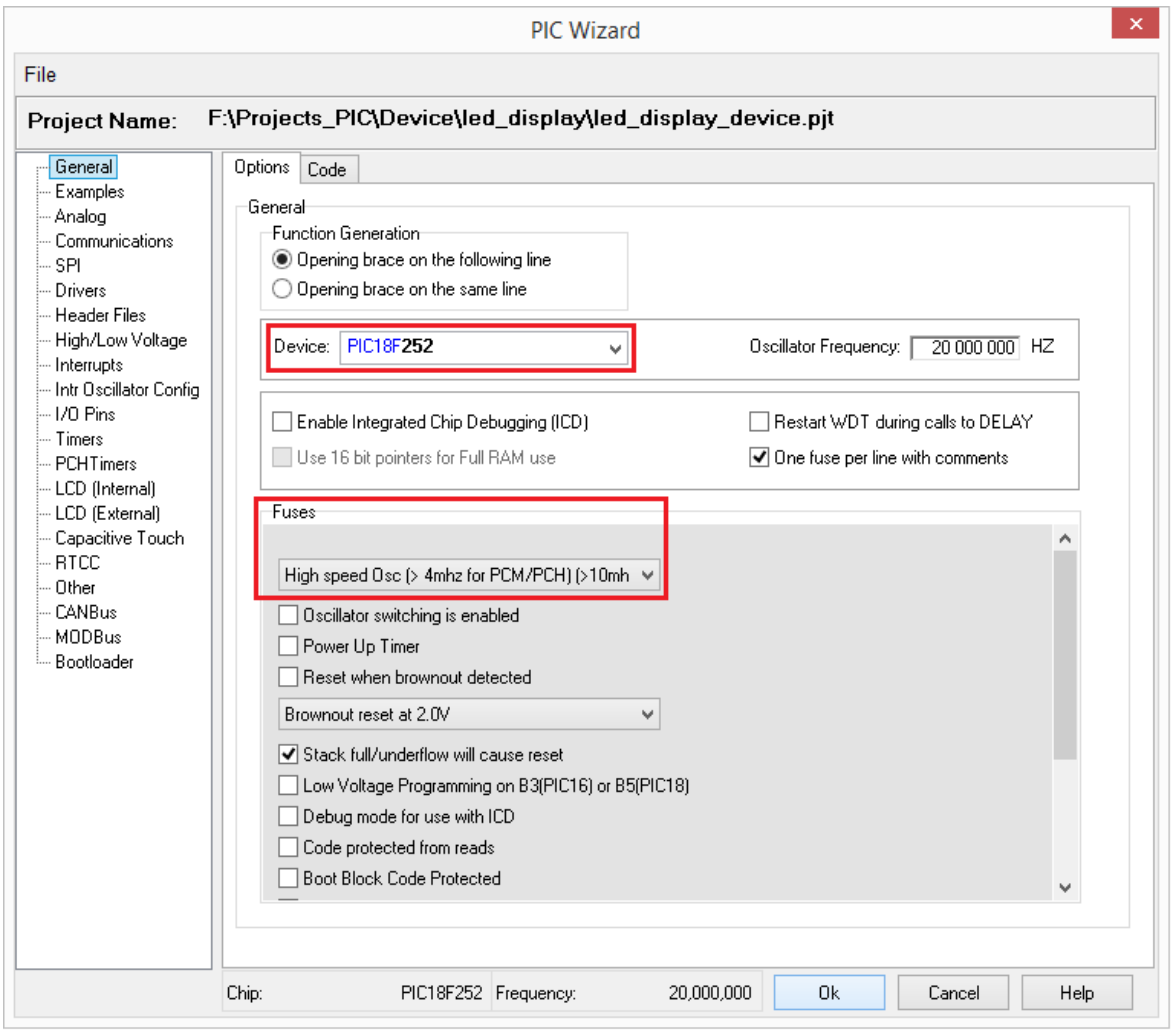

Рисунок 3.1.4 – Розділ **General** майстра **PIC Wizard**

У розділі **Communications** (рис. 3.1.5) 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. Розділі **Communications** встановити мітку у полі **RS-232**:

- ☑ **Use RS-232**;

вказати:

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

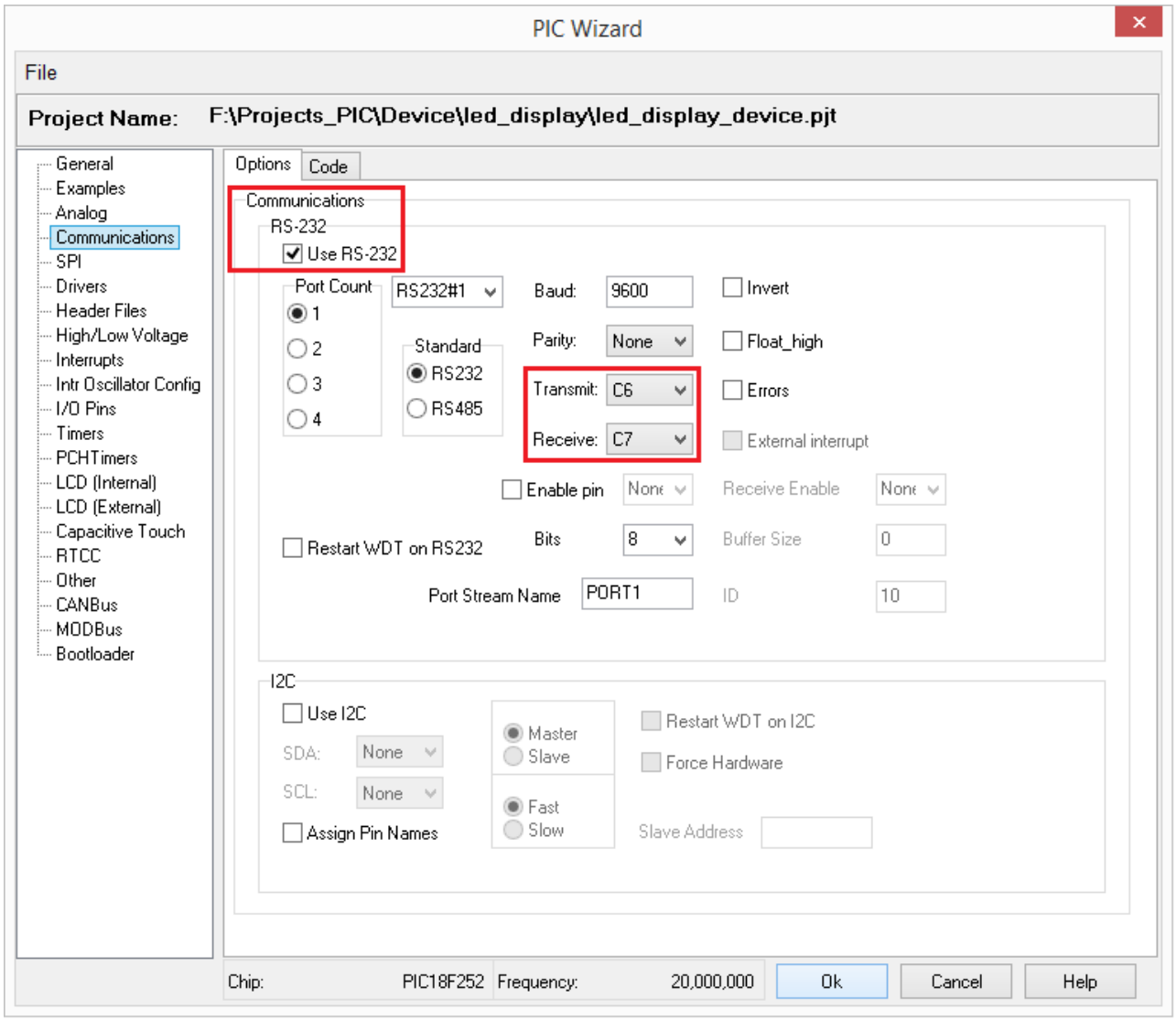

Рисунок 3.1.5 – Розділ **Communications** майстра **PIC Wizard**

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

Для перегляду програмного коду, слід вибрати вкладку **Code**.

7). Натиснути кнопку **OK**.

Буде згенерований наступний код (рис. 3.1.6):

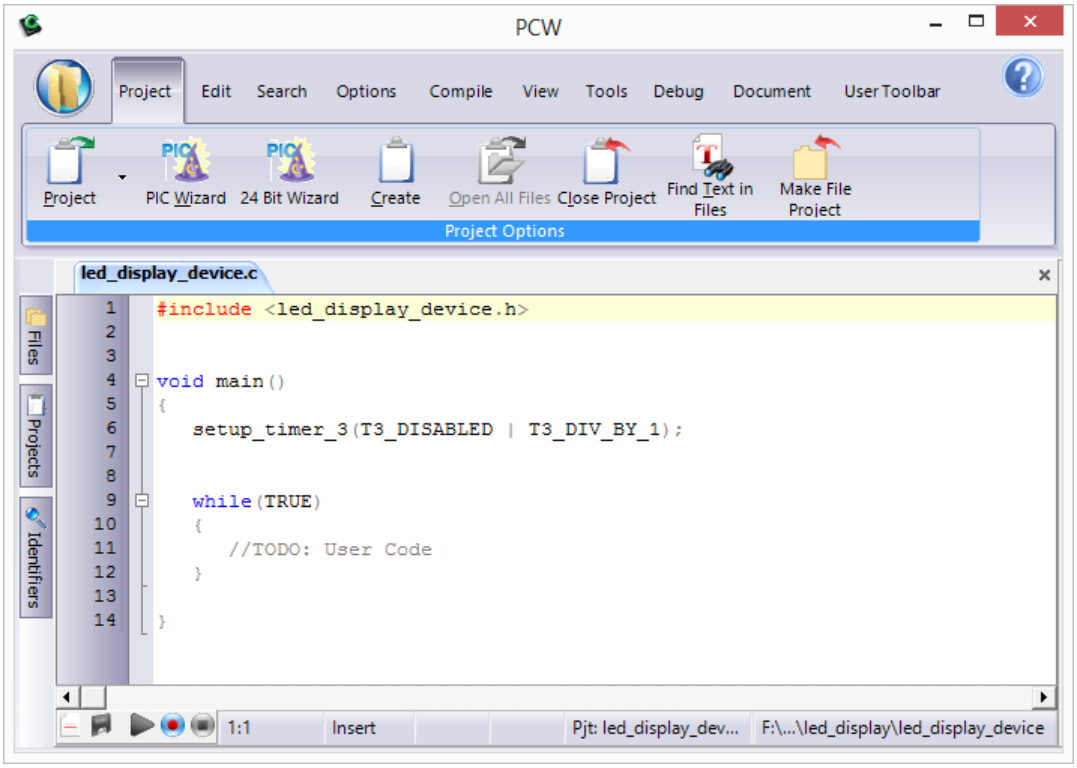

Рисунок 3.1.6 – Згенерований майстром **PIC Wizard** програмний код

Лістинг 3.1.1 – Приклад програмного коду, згенерованого майстром **PIC Wizard**

led_display_device.c

```
#include <led_display_device.h>
void main()
{
	setup_timer_3(T3_DISABLED | T3_DIV_BY_1);
	while(TRUE)
	{
			//TODO: User Code
	}
}
```

led_display_device.h

```
#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)
```

Проект готовий, можна приступати до написання програми.

ПОЯСНЕННЯ ДО ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ «LED_DISPLAY»

У мікроконтролерних системах керування одним з обов'язкових вузлів є підсистема індикації. Вона може складатися з одного світлодіода або декількох динамічних 7-сегментних дисплеїв.

У кожному разі необхідно виконати розрахунки допустимих струмів, що протікають через світлодіоди індикатору.

Розрахункова схема підключення світлодіода представлена на рис. 3.1.7.

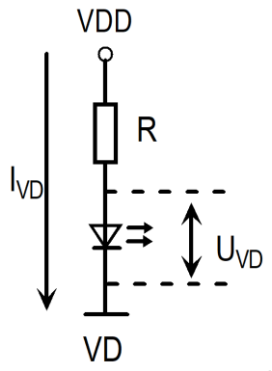

Рисунок 3.1.7 – Розрахункова схема підключення світлодіода

Зазвичай V_{DD} становить 5В, а спадання напруги на світлодіоді U_{VD} – 2,5 В. Робочий струм світлодіоду становить у середньому 5 мА.

Приклади підключення світлодіодів до мікроконтролера представлені на рис. 3.1.8.

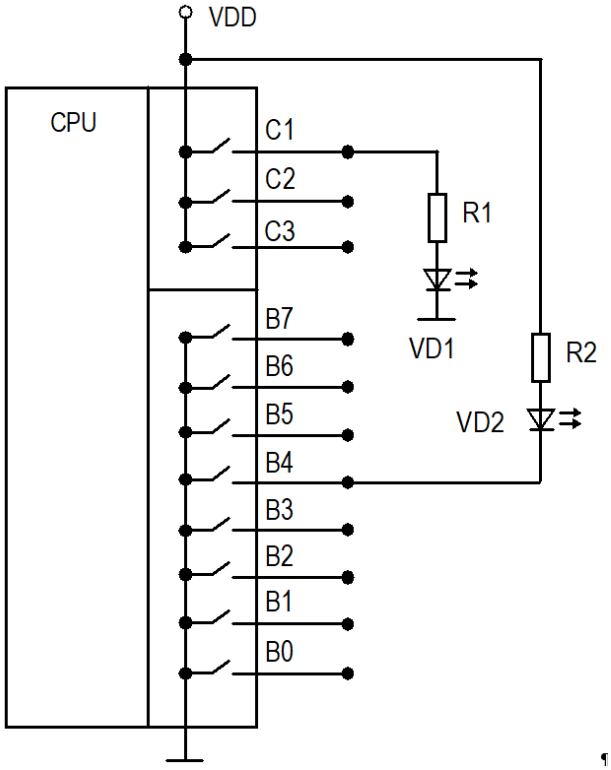

Рисунок 3.1.8 – Приклади підключення світлодіодів до мікроконтролера

LED-дисплей являє собою матрицю знакомісць, у кожному з них розташовано 8 світлодіодів – 7 інформаційних і 1 – десяткова крапка (рис 3.1.9).

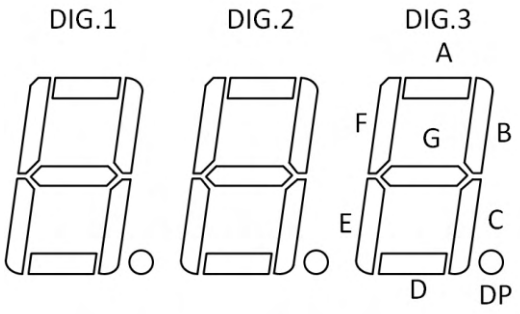

Рисунок 3.1.9 – LED – дисплей

Світлодіоди у дисплеях можуть бути включені за схемою із загальним катодом (рис. 3.1.10) або із загальним анодом (рис. 3.1.11)

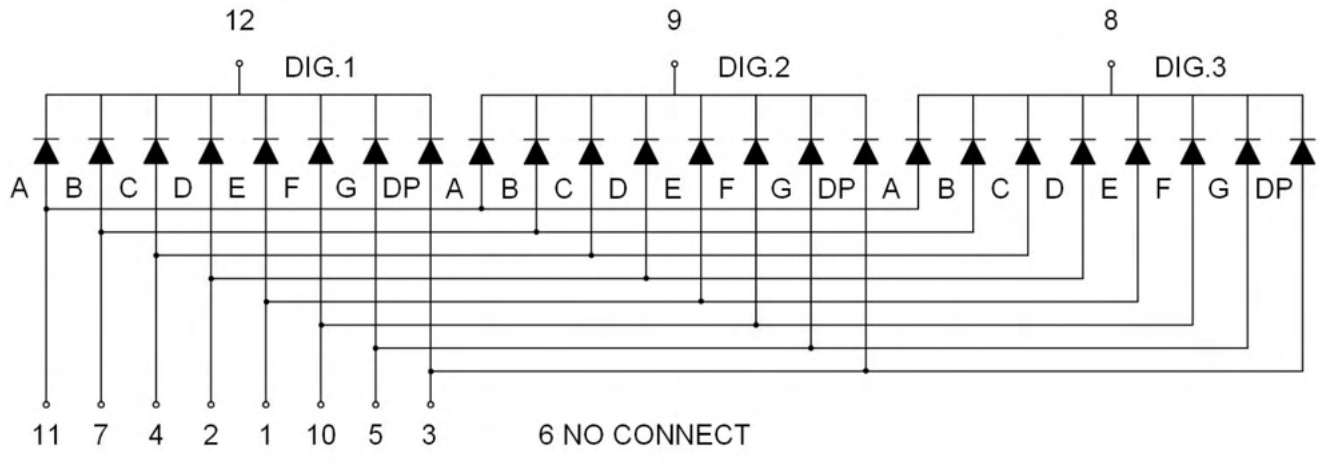

Рисунок 3.1.10 – Схема із загальним катодом

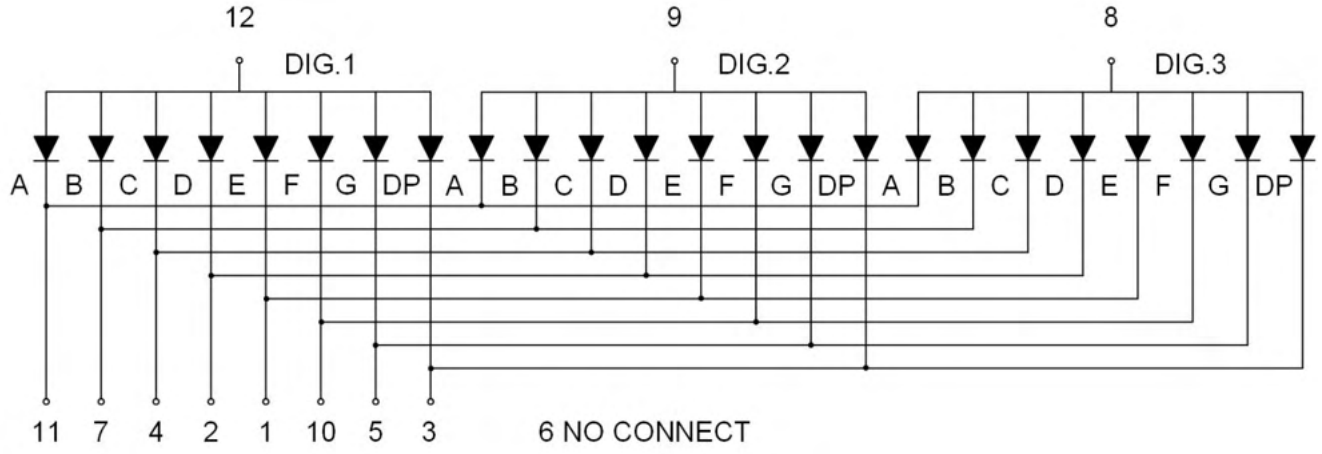

Рисунок 3.1.11 – Схема із загальним анодом

Схема підключення LED-дисплея до мікроконтролера представлена на рис. 3.1.12.

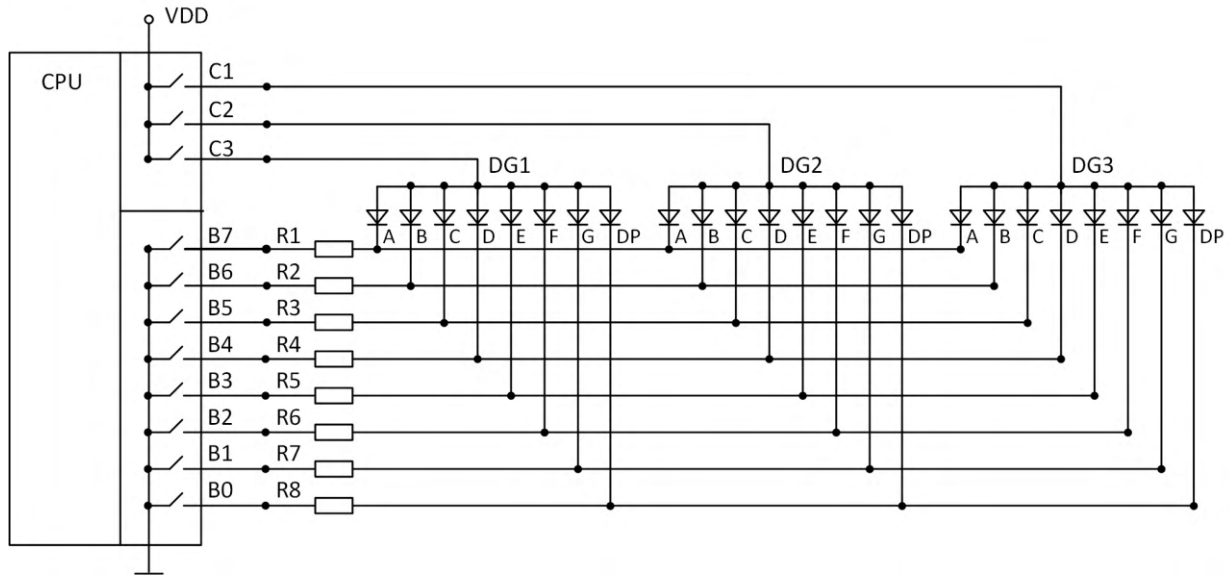

Рисунок 3.1.12 – Підключення LED-дисплея до контролера

Тимчасова діаграма роботи дисплея у динамічному режимі представлена на рис. 3.1.13.

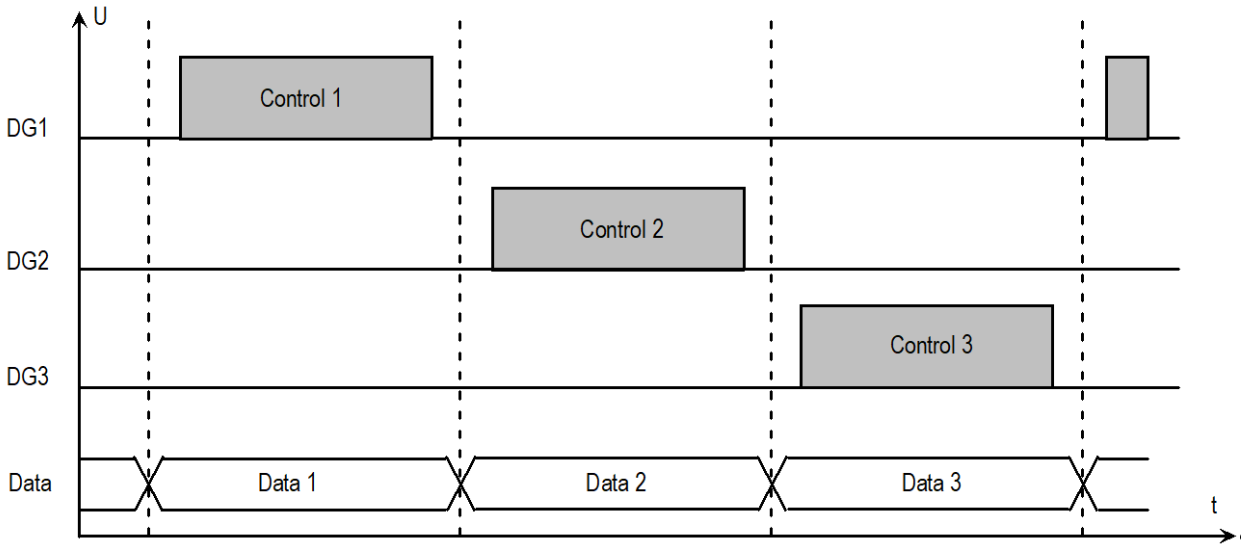

Рисунок 3.1.13 – Тимчасова діаграма роботи дисплея у динамічному режимі

У цьому режимі послідовно змінюються дані на інформаційних входах дисплея (A..G) і кожне знакомісце активується подачею логічної «1» на керуючий вхід (DG1..DG3).

Приклади реалізації програми лабораторної роботи «LED_display»

Приклад застосування конструкції if-then:

```
//=== відобразити 2-е знакомісце
output_c(D_2);                                  //включити знакомісце
if(digit == '1'){
       output_b(LED_digit);                     //відобразити значення
}
```

Приклад застосування конструкції switch:

```
//=== відобразити 2-е знакомісце
output_c(D_2);                                  //включити знакомісце
switch(digit){
       case '1':
              output_b(LED_digit);              //відобразити значення
              break;
...
```

Приклад застосування конструкції табличного методу:

```
//=== індекс масиву 7 сегментних кодів
idx = atoi(str_idx);

//=== читати з масиву 7 сегментний код числа
digit_out = LED[idx];

//=== відобразити знакомісце
output_c(D_2);                                    //включити знакомісце
output_b(digit_out);                              //відобразити значення

//=== час відображення
delay_ms(delay);
```

Для виключення мерехтіння затримки слід вибирати в межах 3-10 мс.

Приклади виконання лабораторної роботи «LED_display» відповідно до варіанту завдання для лабораторної роботи «LED_display»

Варіант завдання для виконання лабораторної роботи **«LED_display»** представлений у таблиці 3.1.3.

Таблиця 3.1.3 – Завдання для виконання лабораторної роботи **«LED_display»**

Число	**Керування**				**Дані**							
N1 і N2	C0	C1	C2	C3	B0	B1	B2	B3	B4	B5	B6	B7
000-999	D1	D2	D3		DP	G	F	E	D	C	B	A

Алгоритм відображення цифр на дисплеї:

1) зчитати із заданого числа один розряд;
2) перетворити у 7-сегментний код;
3) виключити попереднє знакомісце;
4) вивести 7-сегментний код;
5) включити поточне знакомісце;
6) включити затримку для індикації знакомісця;
7) перейти до п. 1.

Принципова схема підключень для варіанту АПК

Принципова схема підключень для варіанту **АПК**, для виконання лабораторної роботи **«LED_display»** виглядає у такий спосіб (рис. 3.1.14):

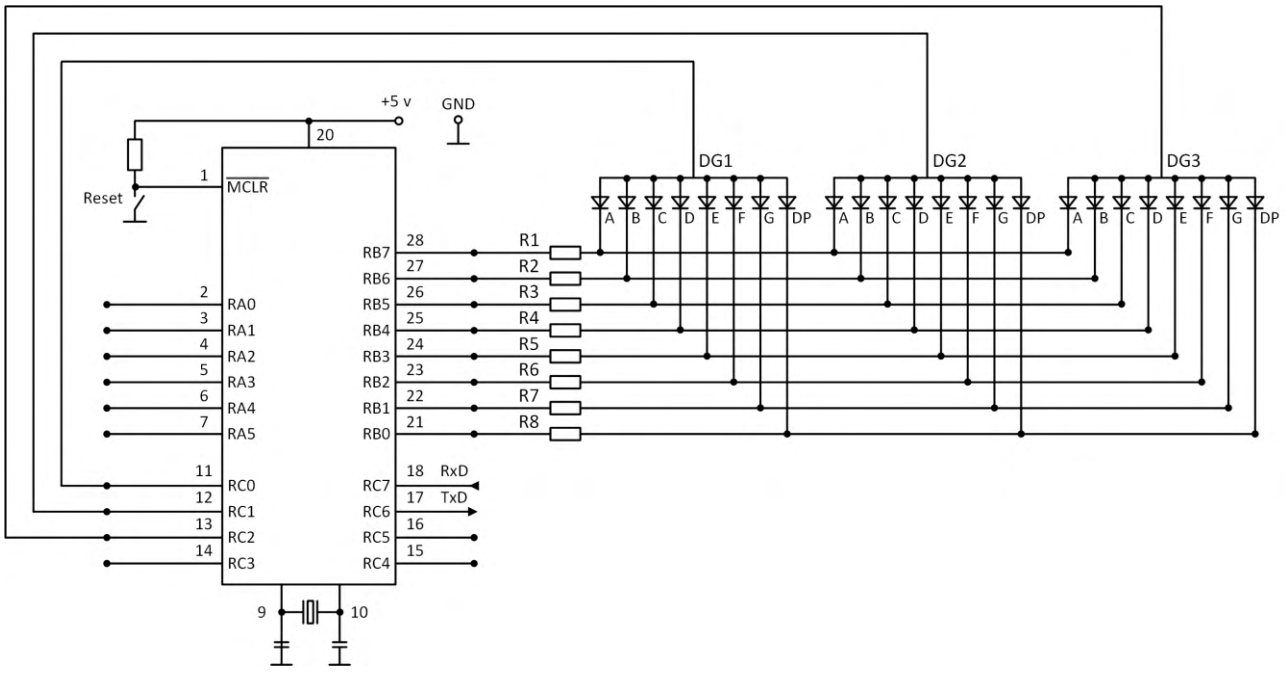

Рисунок 3.1.14 – Принципова схема підключень
для лабораторної роботи **«LED_display»** для варіанту **АПК**

Принципова схема підключень для варіанту «Proteus»

Принципова схема для варіанту **«Proteus»**, для виконання лабораторної роботи **«LED_display»** виглядає у такий спосіб (рис. 3.1.15):

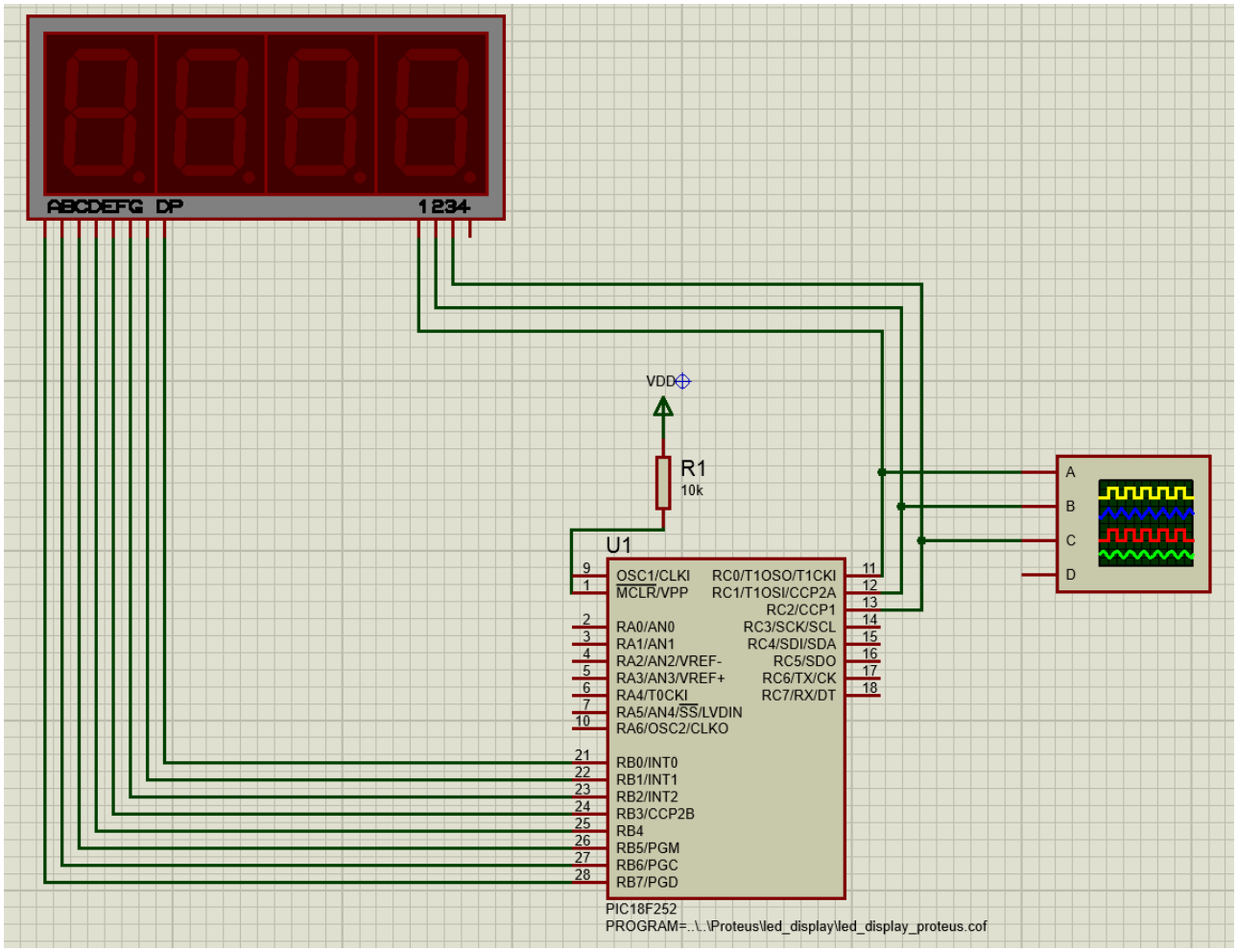

Рисунок 3.1.15 – Принципова схема підключень лабораторної роботи **«LED_display»** для варіанту **«Proteus»**

Результат виконання лабораторної роботи «LED_display» для варіанту «Proteus»

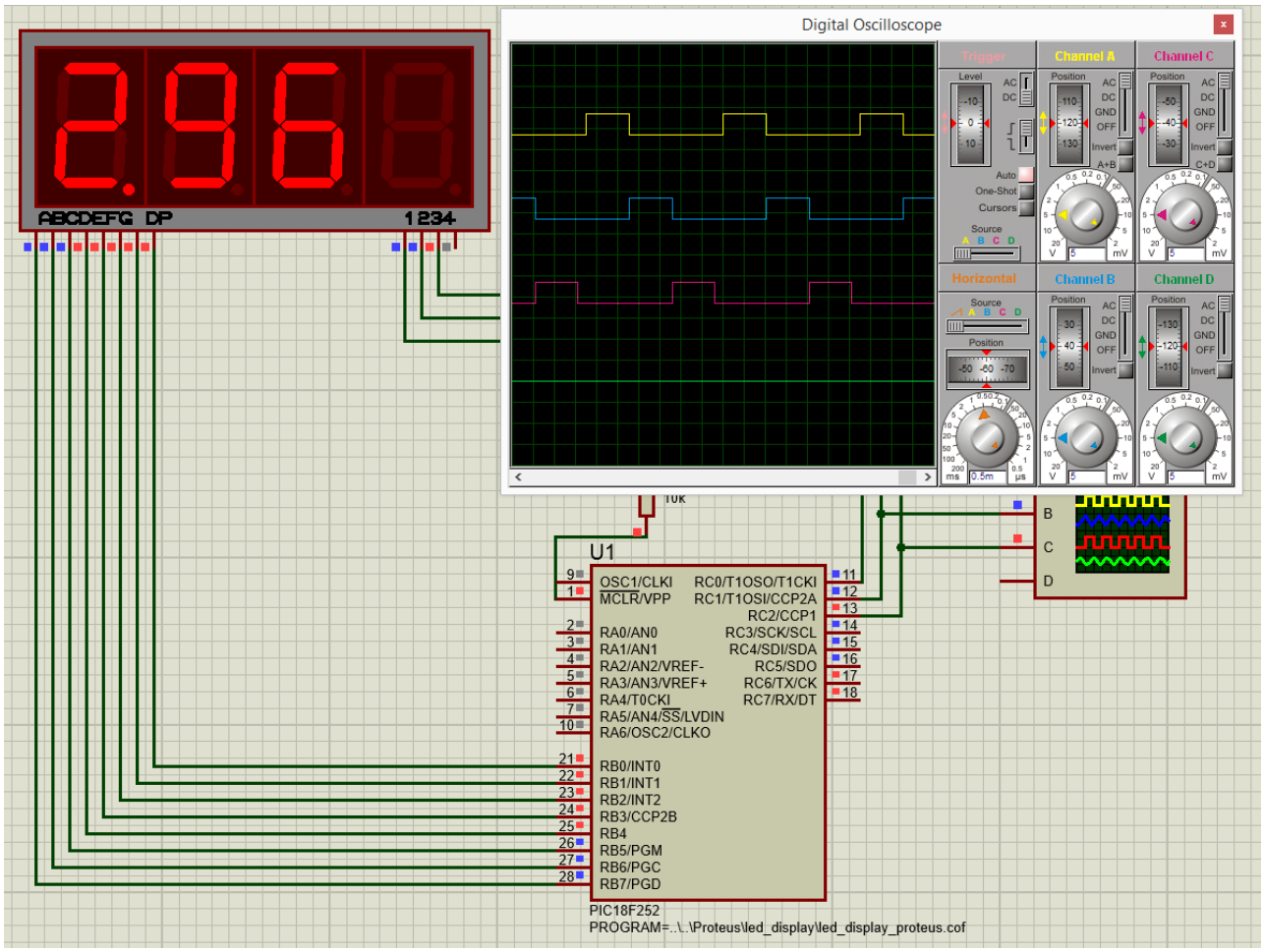

Рисунок 3.1.16 – Результат виконання лабораторної роботи для варіанту **«Proteus»**

Контрольні питання

1) схеми включення світлодіодів у LED-дисплеї;
2) розрахункова формула для визначення опору резистора в ланцюзі живлення світлодіода;
3) призначення виводів LED-дисплея;
4) принцип динамічної індикації;
5) перетворення десяткової цифри у 7-сегментний код.

ЛАБОРАТОРНА РОБОТА № 2 - «PWM»

Тема: Модуль CCP. Керування двигуном постійного струму

Ціль роботи:

Отримання навичок роботи з ШІМ і керування навантаженням за допомогою ШІМ.

Завдання лабораторної роботи

- для варіанту **АПК** намалювати принципову схему підключень відповідно до варіанту для виконання лабораторної роботи (таб. 3.2.1);
- для варіанту **«Proteus»** використовувати готову схему відповідно до варіанту для виконання лабораторної роботи;
- створити алгоритм роботи програми керування яскравістю світіння світлодіода та регулювання напрямку і швидкості обертання електродвигуна постійного струму;
- написати програму роботи ШІМ модуля;
- здійснити регулювання яскравості світіння світлодіоду;
- здійснити регулювання напрямку і швидкості обертання електродвигуна постійного струму.
- здійснити виведення результатів на LCD відповідно до варіанту для виконання лабораторної роботи.

Виведення на LCD повинне містити:

- поточний режим роботи двигуна;
- значення скважності у форматі ##.## %;
- номер лабораторної роботи;
- групу студента;
- прізвище та ініціали студента.

Варіанти для виконання лабораторної роботи «PWM»

Таблиця 3.2.1 – Варіанти для виконання лабораторної роботи **«PWM»**

	Керування режимами						
№	**Вліво**	**Вправо**	**Стоп**	**L_{UP}**	**R_{UP}**	**L_{DW}**	**R_{DW}**
1	Int 0	Int 1	Int 2	B7	B6	CCP 1	CCP 2
2	Int 1	Int 2	Int 0	B6	B5	CCP 2	CCP 1
3	Int 2	Int 0	Int 1	B5	B4	CCP 1	CCP 2
4	Int 0	Int 1	Int 2	B4	B7	CCP 2	CCP 1
5	Int 1	Int 2	Int 0	B7	B6	CCP 1	CCP 2
6	Int 2	Int 0	Int 1	B6	B5	CCP 2	CCP 1
7	Int 0	Int 1	Int 2	B5	B4	CCP 1	CCP 2
8	Int 1	Int 2	Int 0	B4	B7	CCP 2	CCP 1
9	Int 2	Int 0	Int 1	B7	B6	CCP 1	CCP 2
10	Int 0	Int 1	Int 2	B6	B5	CCP 2	CCP 1
11	Int 1	Int 2	Int 0	B5	B4	CCP 1	CCP 2
12	Int 2	Int 0	Int 1	B4	B7	CCP 2	CCP 1
13	Int 0	Int 1	Int 2	B7	B6	CCP 1	CCP 2
14	Int 1	Int 2	Int 0	B6	B5	CCP 2	CCP 1
15	Int 2	Int 0	Int 1	B5	B4	CCP 1	CCP 2

Послідовність виконання лабораторної роботи для варіанту АПК

1) для варіанту **АПК** намалювати принципову схему підключень відповідно до варіанту для виконання лабораторної роботи;
2) створити свій директорій для файлів проекту;
3) відкрити інтегроване середовище розробки **PCWH**;
4) створити проект у інтегрованому середовищі розробки **PCWH**;
5) створити алгоритм роботи програми керування яскравістю світіння світлодіода та регулювання напрямку і швидкості обертання електродвигуна постійного струму;
6) написати програму мовою програмування **C**, що реалізує створений алгоритм в обробнику переривань, відповідно до варіанту для виконання лабораторної роботи;
7) скомпілювати програму, одержати бінарні файли ***.HEX** і ***.COF** (файли з розширеннями ***.HEX** і ***.COF** створюються підчас компіляції програми);

8) завантажити бінарний файл ***.HEX** у пам'ять програм мікроконтролера програмою **PICkit.exe**;
9) на **АПК** зкумутувати схему підключень (світлодіоди, потенціометр і двигун постійного струму (рис. 3.2.9)) відповідно до варіанту для виконання лабораторної роботи;
10) виконати програму.

Послідовність виконання лабораторної роботи 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 роботи програми керування яскравістю світіння світлодіода та регулювання напрямку і швидкості обертання електродвигуна постійного струму;
6) написати програму мовою програмування **C**, що реалізує створений алгоритм в обробнику переривань, відповідно до варіанту для виконання лабораторної роботи;
7) скомпілювати програму, одержати бінарні файли ***.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 лабораторної роботи **«PWM»**;
10) відкрити файл проекту з розширенням ***.DSN**;
11) у середовищі моделювання **«Proteus»** змінити схему підключень відповідно до варіанту для виконання лабораторної роботи;
12) завантажити заздалегідь скомпільований бінарний файл ***.COF** або ***.HEX** у мікроконтролер;

13) у середовищі моделювання **«Proteus»** виконати програму.

Послідовність виконання лабораторної роботи 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 лабораторної роботи **«PWM»**;
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 лабораторної роботи **«PWM»**;
9) відкрити файл проекту з розширенням ***.DSN**;
10) у середовищі моделювання **«Proteus»** змінити схему підключень відповідно до варіанту для виконання лабораторної роботи;
11) створити алгоритм роботи програми керування яскравістю світіння світлодіода та регулювання напрямку і швидкості обертання електродвигуна постійного струму;
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 лабораторної роботи;
11) скомпілювати програму, одержати бінарні файли ***.HEX** і ***.COF** (файли з розширеннями ***.HEX** і ***.COF** створюються підчас компіляції програми);
12) у середовищі моделювання **«Proteus»** виконати програму.

Примітка: *файл демонстрації виконання лабораторної роботи* ***«PWM»*** *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. 3.2.4). 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):

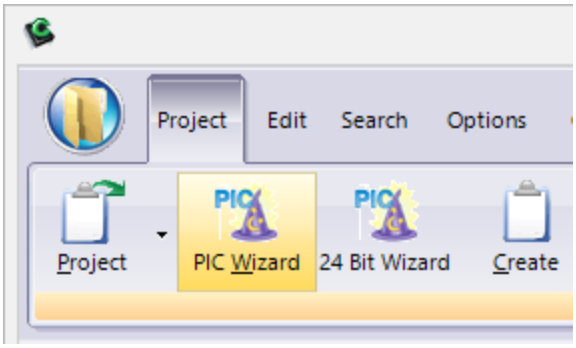

Рисунок 3.2.1 – Процес створення проекту у **IDE PCWH** за допомогою майстра **PIC Wizard**

або натиснути кнопку у вигляді відкритої папки  .

2. Вибрати: **New › Project Wizard** (рис. 3.2.2):

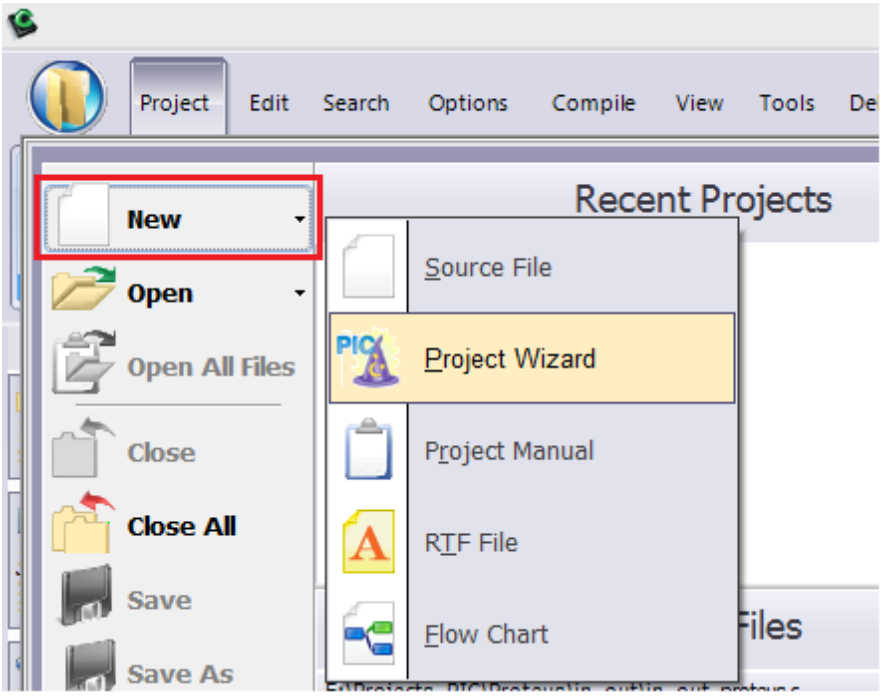

Рисунок 3.2.2 – Процес створення проекту у **IDE PCWH** за допомогою майстра **PIC Wizard**

3. Створити або вибрати свій директорій і записати у нього проект під будь-яким іменем латинським шрифтом, наприклад: «**pwm.pjt**» (рис. 3.2.3):

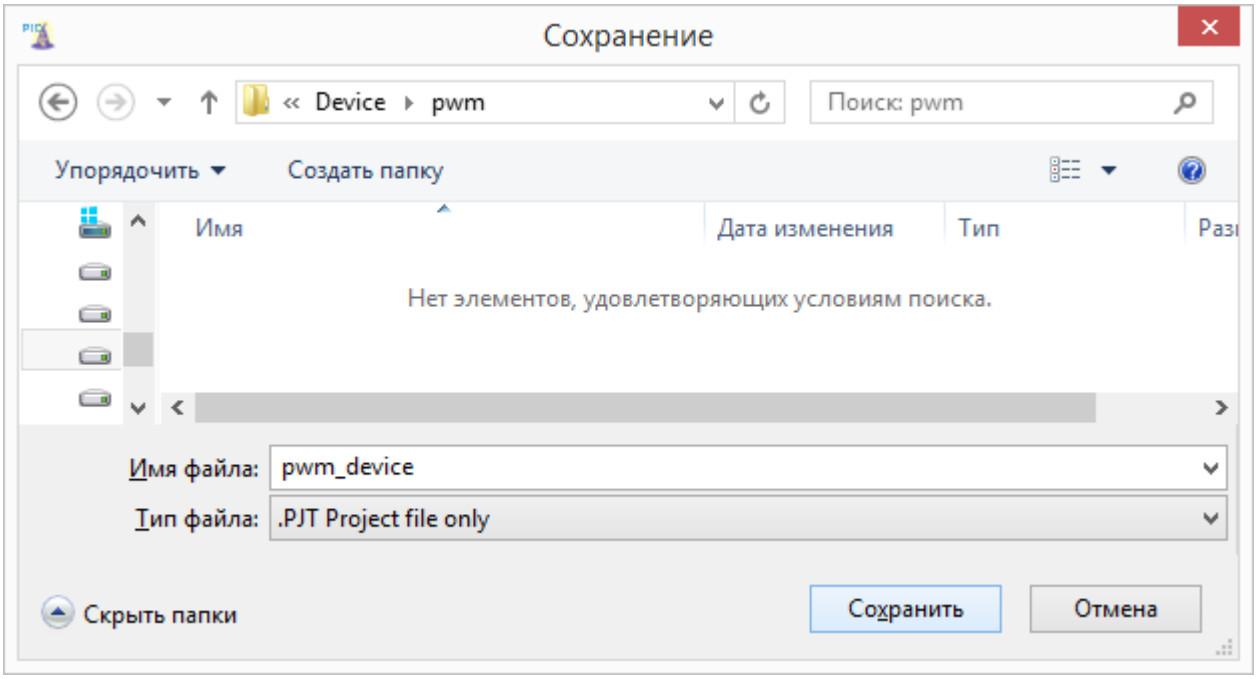

Рисунок 3.2.3 – Процес створення проекту у **IDE PCWH** за допомогою майстра **PIC Wizard**

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).**

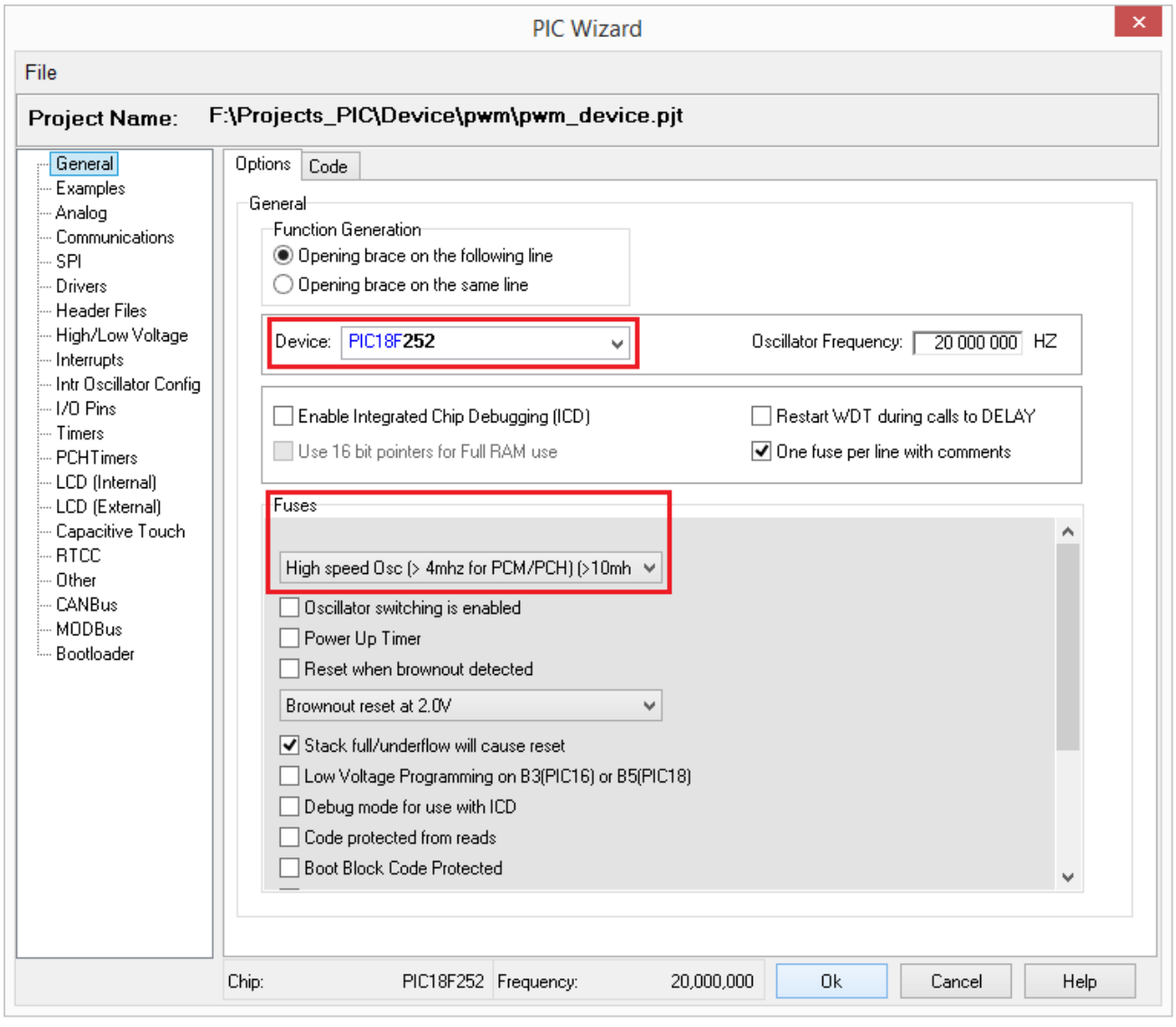

Рисунок 3.2.5 – Розділ General майстра PIC Wizard

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).

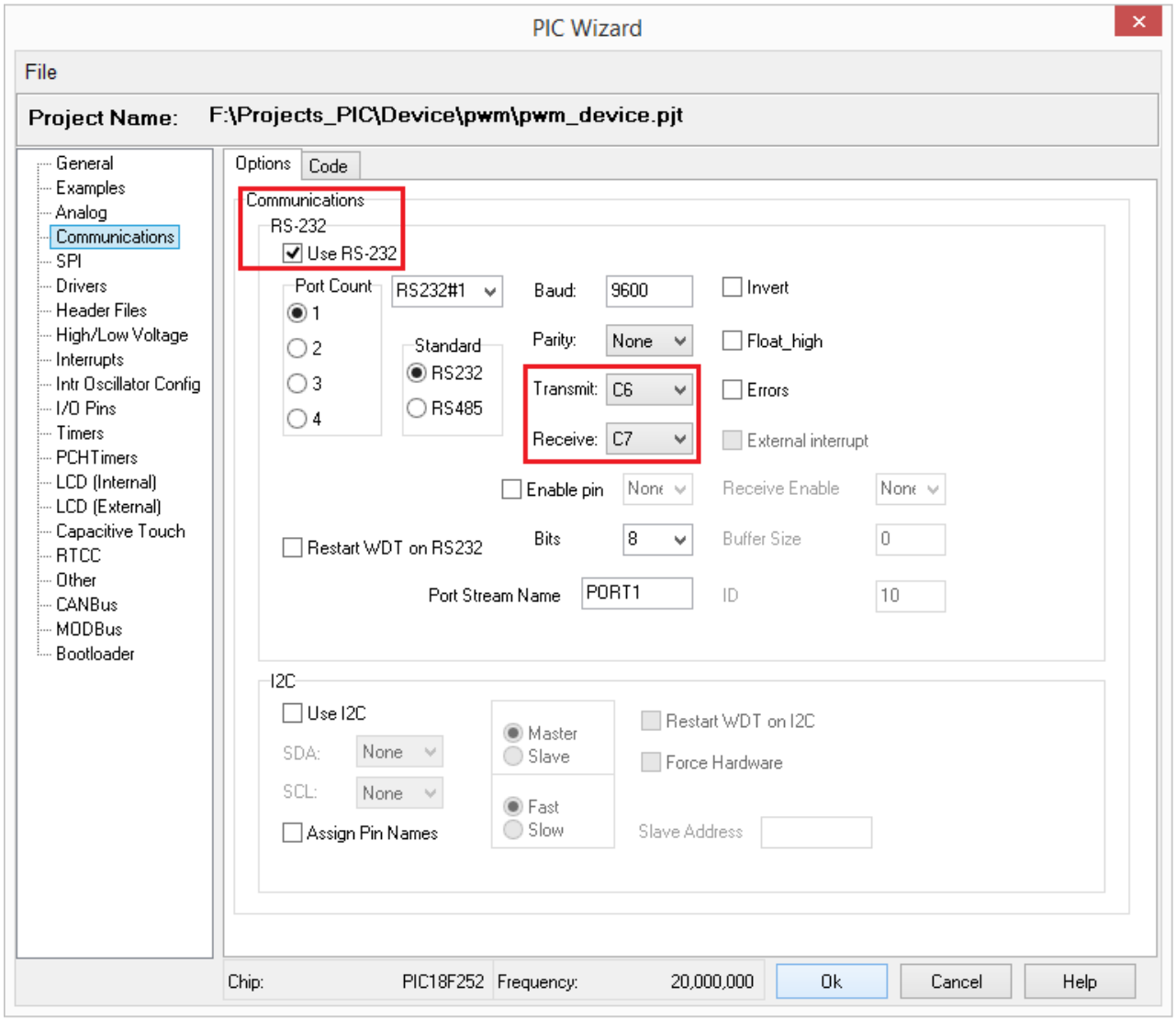

Рисунок 3.2.5 – Розділ **Communications** майстра **PIC Wizard**

Розділ **Analog** (рис. 3.2.6) 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 при створенні проекту (рис. 3.2.6):

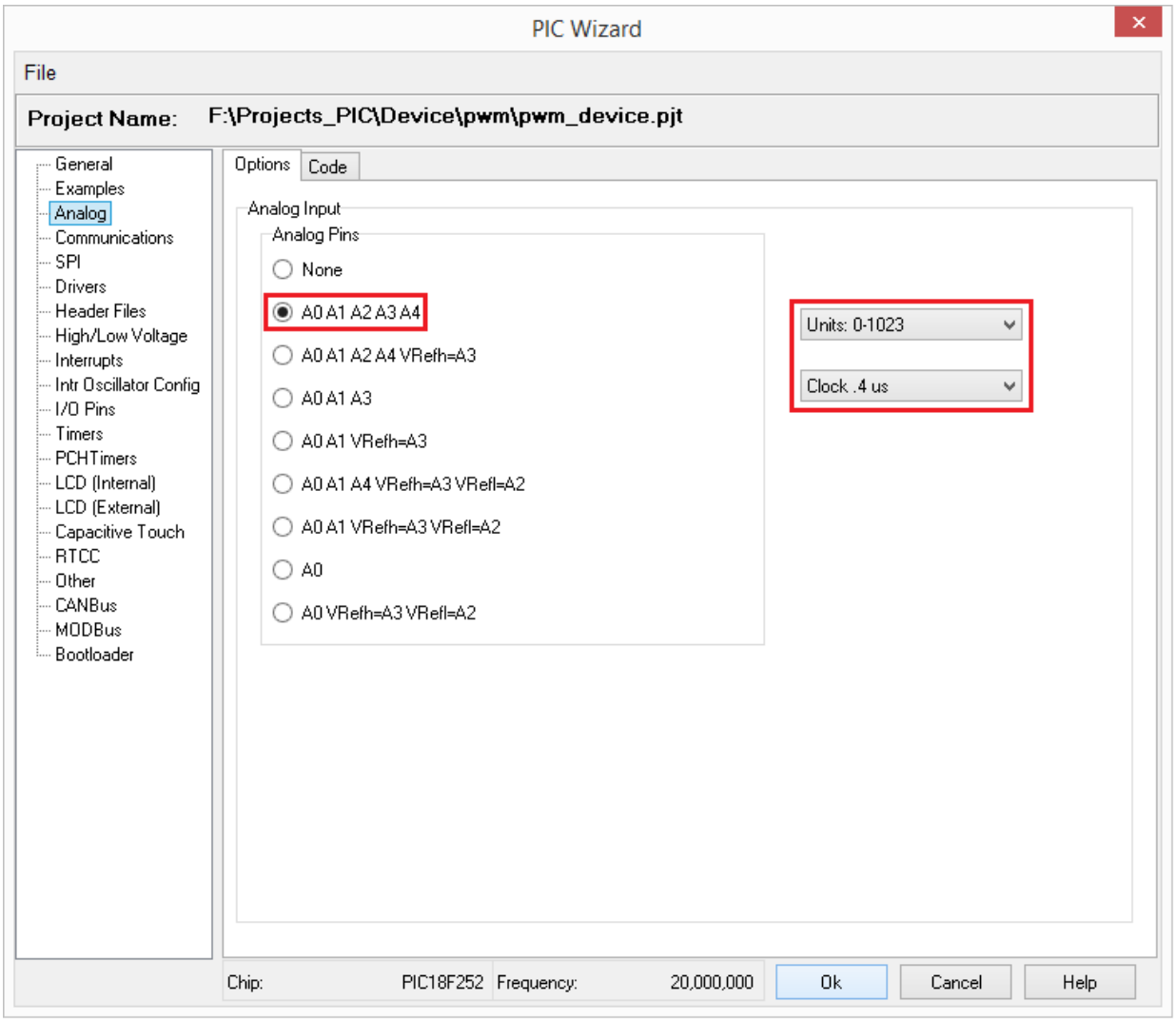

Рисунок 3.2.6 – Конфігурування **АЦП** із **IDE PCWH** при створенні проекту у розділі **Analog** майстра **PIC Wizard**

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. 3.2.7):

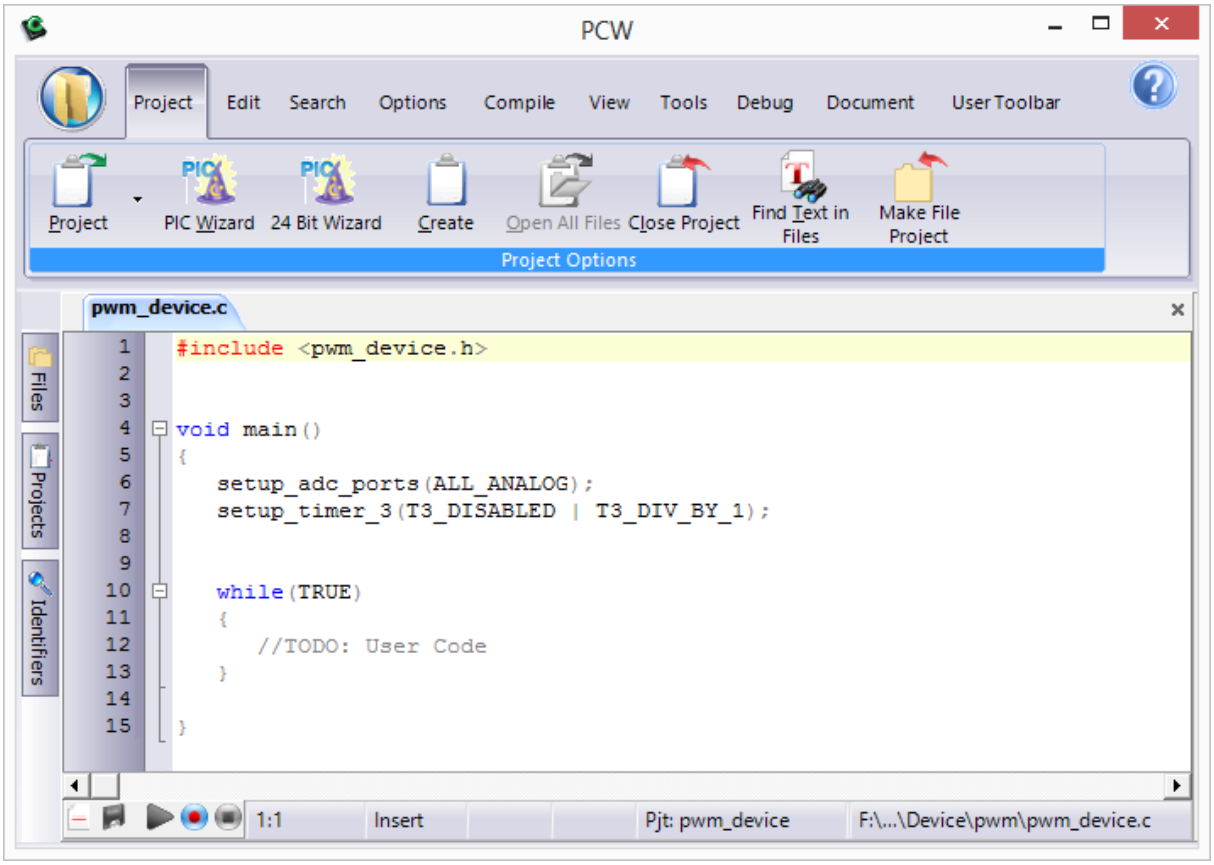

Рисунок 3.2.7 – Згенерований майстром **PIC Wizard** програмний код

Лістинг 3.2.1 – Приклад програмного коду, згенерованого майстром **PIC Wizard**

pwm_device.c

```
#include < pwm_device.h>

void main()
{
        setup_adc_ports(ALL_ANALOG);
        setup_timer_3(T3_DISABLED | T3_DIV_BY_1);
```

```
        while(TRUE)
        {
                //TODO: User Code
        }
}
```

pwm_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 ЛАБОРАТОРНОЇ РОБОТИ «PWM»

Підключення двигуна постійного струму

Крім швидкості обертання необхідно змінювати і напрямок обертання, тому електродвигун включається в мостову схему, як показано на рис. 3.2.8.

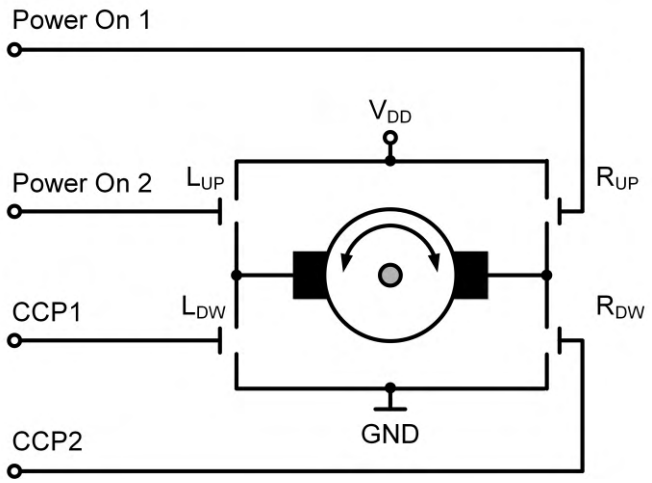

Рисунок 3.2.8 – Включення двигуна постійного струму в мостову схему

Керування напрямком обертання двигуна здійснюється в такий спосіб:

Обертання вліво:

- замкнути контакти в плечі моста $\mathbf{L_{UP}}$
  (подати сигнал керування **Power On 2**);
- замкнути контакти в плечі моста $\mathbf{R_{DW}}$
  (подати сигнал керування **CCP2**).

Обертання вправо:

- замкнути контакти в плечі моста $\mathbf{R_{UP}}$
  (подати сигнал керування **Power On 1**);
- замкнути контакти в плечі моста $\mathbf{L_{DW}}$
  (подати сигнал керування **CCP1**).

Режими роботи електродвигуна постійного струму:

- «Вправо»;
- «Вліво»;
- «Стоп»,

реалізувати дискретними кнопками.

Швидкість обертання електродвигуна постійного струму регулювати за допомогою потенціометра.

Керування режимами доцільно здійснювати через переривання **INT0-INT2**.

Конфігурування мікроконтролера для роботи з ШІМ

*Вмикання модуля **CCP**:*

```
setup_ccp1(CCP_PWM);
setup_ccp2(CCP_PWM);
```

*Вимикання модуля **CCP**:*

```
setup_ccp1(CCP_OFF);
setup_ccp2(CCP_OFF);
```

Модуль ШІМ працює з таймером **timer_2**.

Параметри вихідного сигналу на виходах **CCP1** і **CCP2** мікроконтролера визначаються установками таймера **timer_2** (рис. 3.2.9).

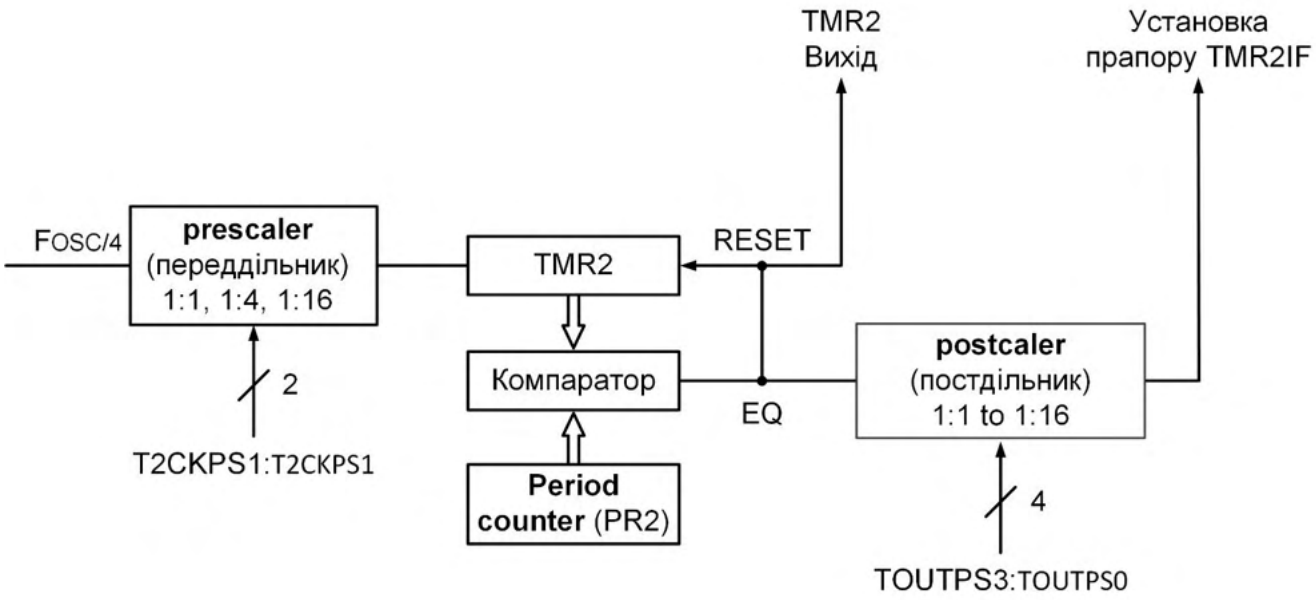

Рисунок 3.2.9 – Структурна схема **TMR2** (таймера 2)

Установка таймера:

Формат:

```
setup_timer_2(prescaler, period, postscaler);
```

наприклад:

```
setup_timer_2(T2_DIV_BY_16,255,1);
```

Перший параметр пропорційно змінює період проходження T і тривалість імпульсу τ.

T2_DIV_BY_1 – Частота ШІМ = 19.5 khz, період = 51.2 us

T2_DIV_BY_4 – Частота ШІМ = 4.9 khz, період = 204.8 us

T2_DIV_BY_16 – Частота ШІМ = 1.2 khz, період = 833.3 us

Залежність параметрів вихідного сигналу **CCP_X** від значення параметра 1 представлена на рис. 3.2.10.

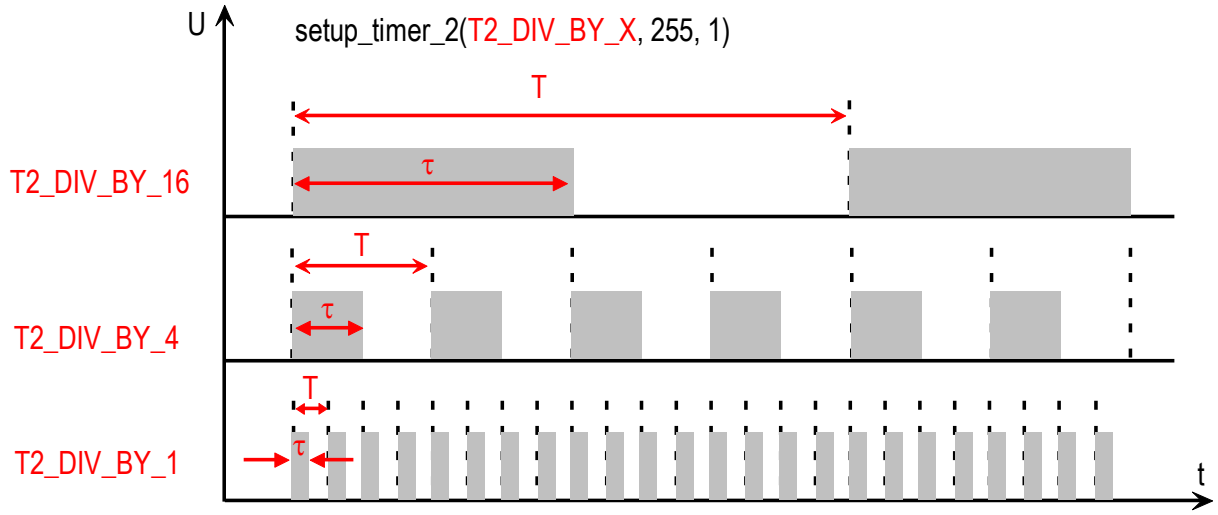

Рисунок 3.2.10 – Залежність параметрів вихідного сигналу від значення параметра 1

Другий параметр є лічильником періоду T.

Коефіцієнт приймає значення від 1 до 255.

Залежність періоду T вихідного сигналу CCP_X від значення параметра 2 представлена на рис. 3.2.11.

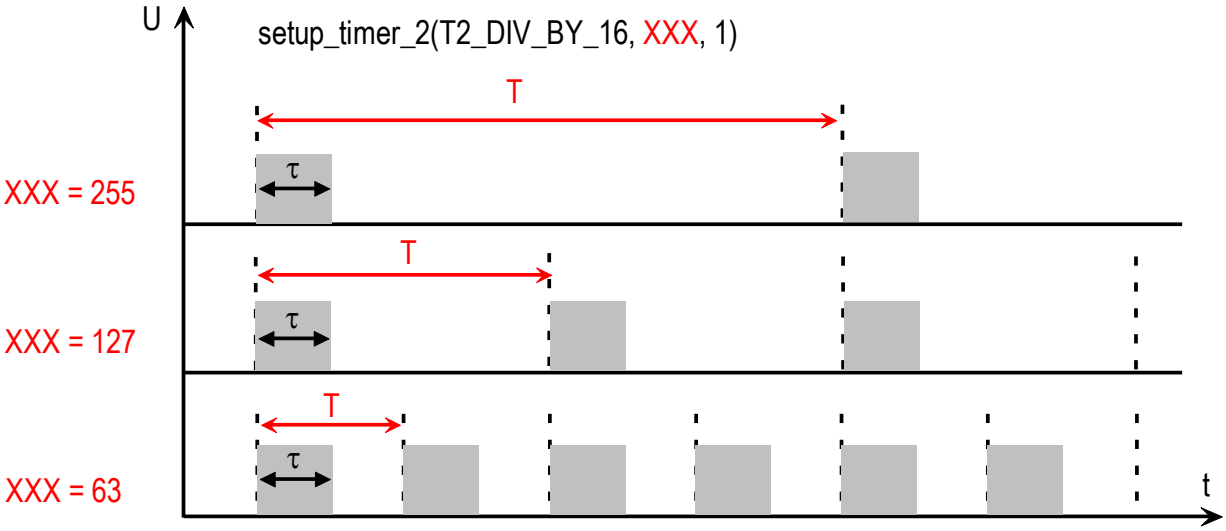

Рисунок 3.2.11 – Залежність параметрів вихідного сигналу від значення параметра 2

Примітка: Якщо параметр 1 пропорційно змінює і період проходження T і тривалість імпульсу τ (скважність S = const), то параметр 2 змінює тільки період проходження T. Наслідком є зміна скважності S. Змінювати скважність у такий спосіб не рекомендується.

Для виконання лабораторної роботи використовувати наступні параметри:

```
setup_timer_2(T2_DIV_BY_16,255,1);
```

Розташування виводів CCP мікроконтролера представлене на рис. 3.2.12.

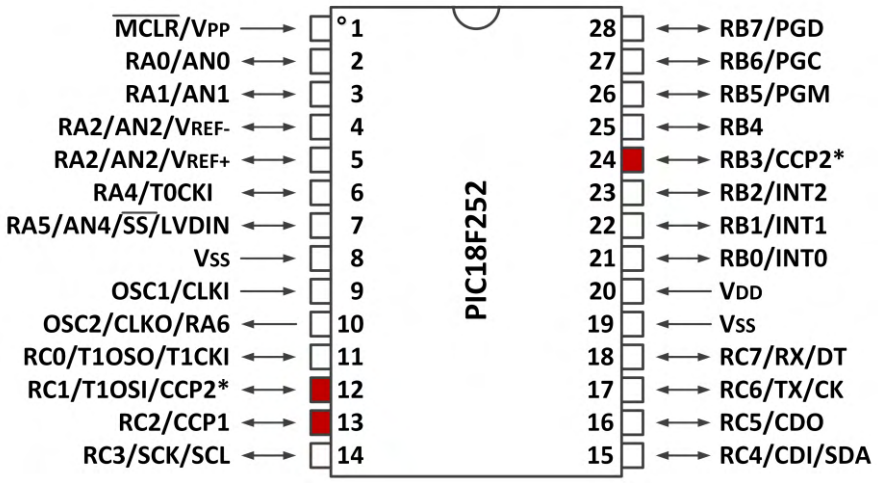

Рисунок 3.2.12 – Розташування виводів CCP мікроконтролера

Виконання лабораторної роботи

Підключення:

B7 – ЛВ

CCP1 – ЛН

B6 – ПВ

CCP2 – ПН

..

КН1 – INT0 – стоп

КН2 – INT1 – вправо

КН3 – INT2 – вліво

Конфігурація мікроконтролера для роботи з ШІМ

Включення модуля CCP:

```
setup_ccp1(CCP_PWM);
setup_ccp2(CCP_PWM);
```

Вимкнення модуля CCP:

```
setup_ccp1(CCP_OFF);
setup_ccp2(CCP_OFF);
```

Модуль ШІМ працює з таймером **timer_2**.

Параметри вихідного сигналу на виходах **CCP1** і **CCP2** мікроконтролера визначаються установками таймера **timer_2**.

Установка таймера:

Формат: `setup_timer_2(mode, period, postscale);`

наприклад: `setup_timer_2(T2_DIV_BY_16,255,1);`

Перший параметр пропорційно змінює період проходження і тривалість імпульсу.

T2_DIV_BY_1 – Частота ШІМ = 19.5 khz, період = 51.2 us

T2_DIV_BY_4 – Частота ШІМ = 4.9 khz, період = 204.8 us

T2_DIV_BY_16 – Частота ШІМ = 1.2 khz, період = 833.3 us

Примітка: Якщо параметр 1 пропорційно змінює і період проходження T і тривалість імпульсу t (скважність S = const), то параметр 2 змінює тільки період проходження T. Наслідком є зміна скважності S.

Змінювати скважність таким чином не рекомендується.

Для виконання лабораторної роботи використовувати такі параметри:

```
setup_timer_2(T2_DIV_BY_16,255,1);
```

Управління скважністю імпульсів:

Управління скважністю імпульсів здійснюється шляхом установки тривалості імпульсу t в періоді проходження T.

```
set_pwm1_duty(long pulse_width);
```

`pulse_width` може знаходитися у діапазоні 1..1023, що відповідає значенням скважності S в межах від 0.1% до 99.9%.

При цьому:

- тривалість імпульсу t не залежить від періоду слідування T.
  Якщо тривалість імпульсу буде більше періоду слідування, то на виході мікроконтролера **CCP_X** постійно знаходитися логічна '1';
- період слідування T не залежить від тривалості імпульсу t.

Тривалість імпульсу обчислюється за формулою:

$$pulse\_width * (1/clock) * t2\_div$$

Наприклад:

Частота генератору – 20 мГц.

Частота ШІМ = 1.2 кГц, період = 833.3 мкс.

Вибираємо дільник: T2_DIV_BY_16 (параметр 1).

Скважність 50% = 416 мкс.

pulse_width = 0.000416/(16*(1/20000000)) = 512.

Приклади виконання лабораторної роботи «PWM» відповідно до варіанту завдання для лабораторної роботи «PWM»

Варіант завдання для виконання лабораторної роботи **«PWM»** представлений у таблиці 3.2.2.

Таблиця 3.2.2 – Завдання для виконання лабораторної роботи **«PWM»**

Керування режимами							
Вліво	**Вправо**	**Стоп**	**Lup**	**Rup**	**Ldw**	**Rdw**	**Потенціометр**
Int 2	Int 1	Int 0	B7	B6	CCP 1	CCP 2	A1

Принципова схема підключень для варіанту АПК

Принципова схема підключень для варіанту **АПК**, для виконання лабораторної роботи **«PWM»** виглядає у такий спосіб (рис. 3.2.13):

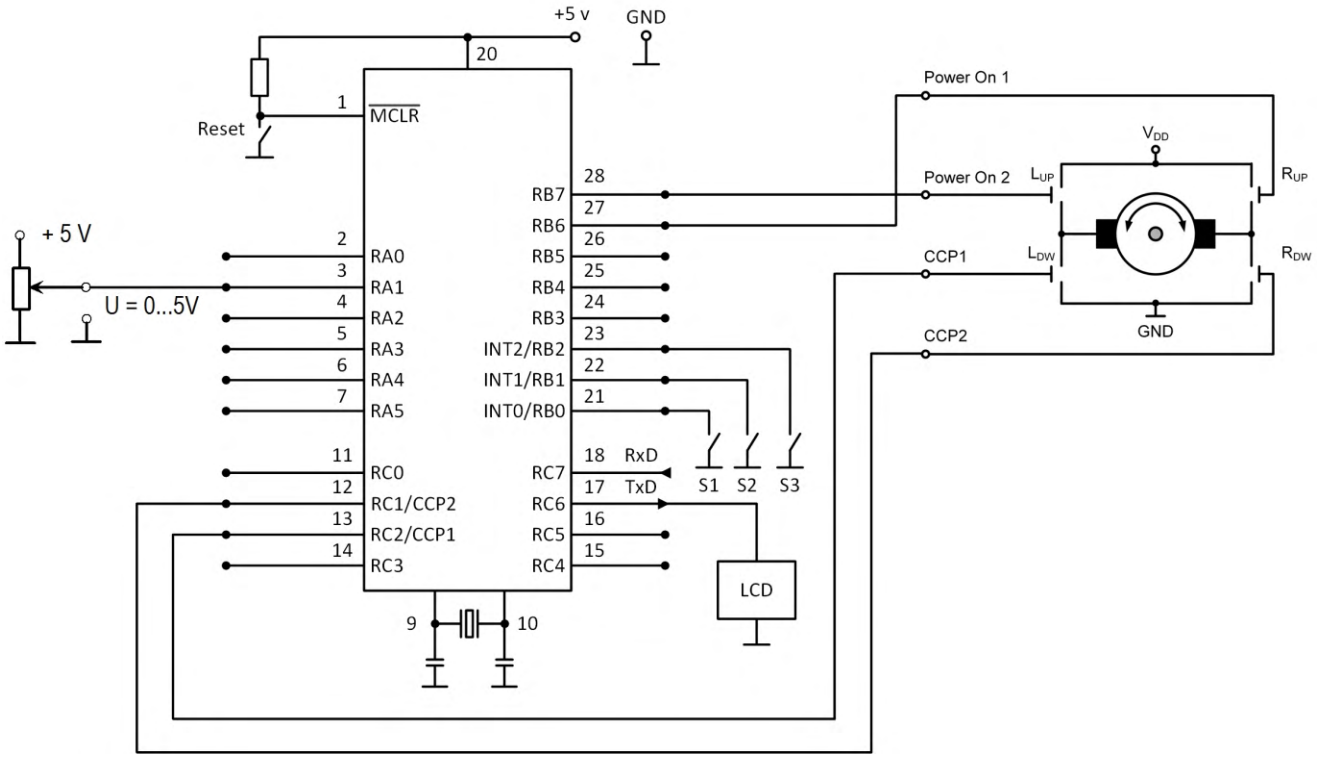

Рисунок 3.2.13 – Принципова схема підключень
для лабораторної роботи **«PWM»** для варіанту **АПК**

Принципова схема підключень для варіанту «Proteus»

Принципова схема для варіанту **«Proteus»**, для виконання лабораторної роботи **«PWM»** виглядає у такий спосіб (рис. 3.2.14):

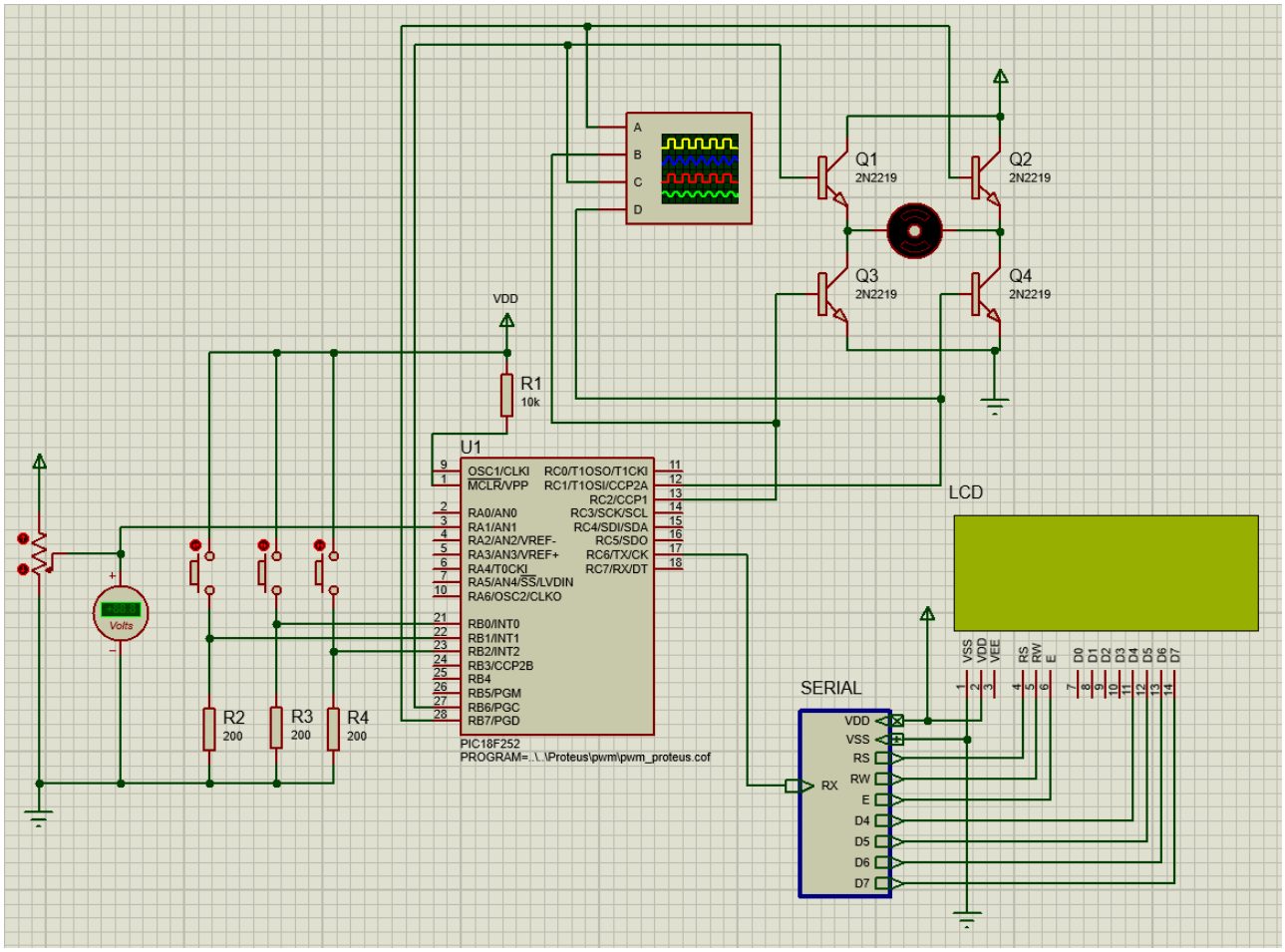

Рисунок 3.2.14 – Принципова схема підключень лабораторної роботи **«PWM»** для варіанту **«Proteus»**

Результат виконання лабораторної роботи «PWM» для варіанту «Proteus»

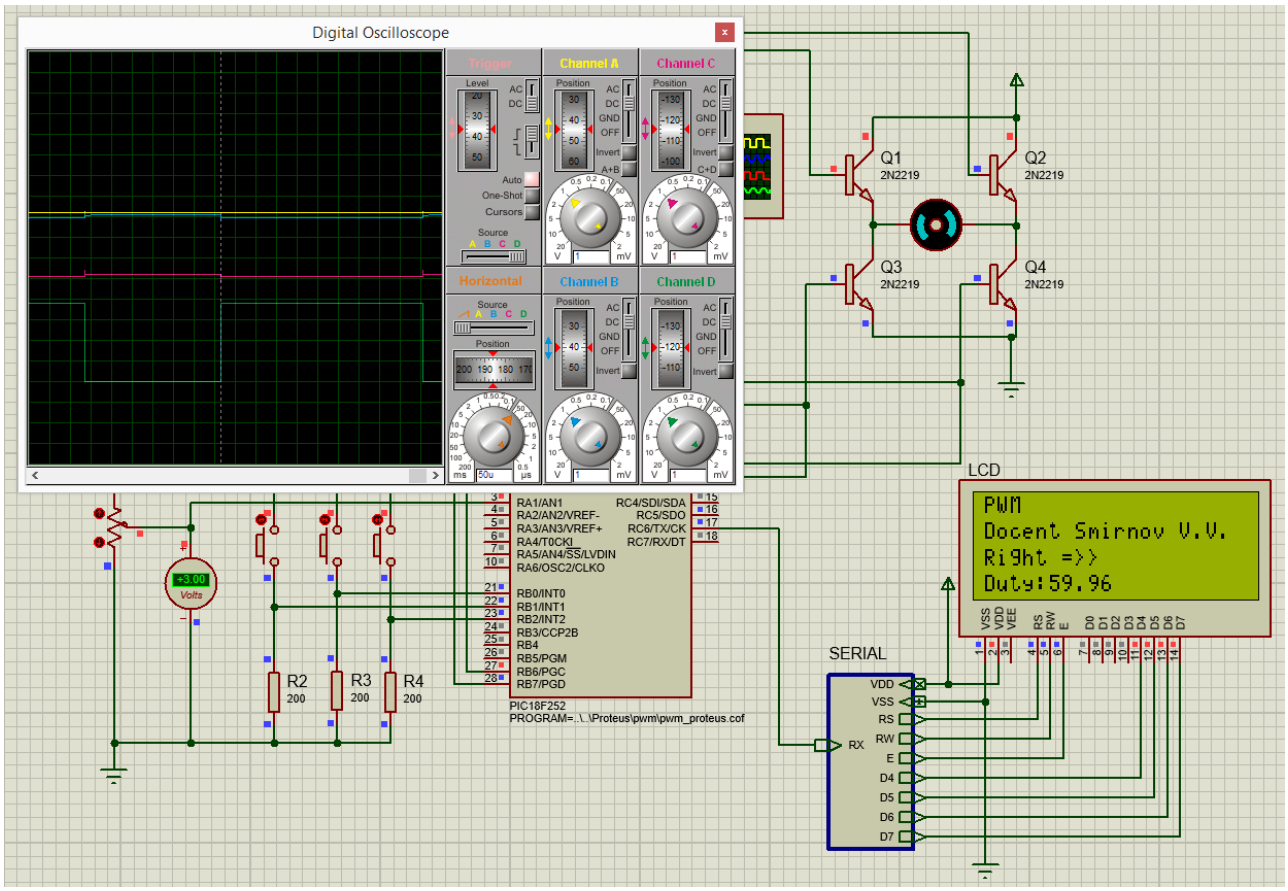

Рисунок 3.2.15 – Результат виконання лабораторної роботи **«PWM»** для варіанту **«Proteus»**

Контрольні питання

1) призначення ШІМ і принцип роботи;
2) загальна формула визначення скважності імпульсів і часна для завдання тривалості імпульсу;
3) призначення параметрів ініціалізації таймера;
4) яка залежність існує між скважністю імпульсів і формованою напругою в інтегруючому ланцюзі?
5) принцип керування напрямком і швидкістю обертання двигуна постійного струму.

ЛАБОРАТОРНА РОБОТА № 3 - «EEPROM»

Тема: Робота із зовнішньою EEPROM - пам'яттю по інтерфейсу I^2C

Ціль роботи:

Отримання навичок роботи з зовнішньою EEPROM - пам'яттю по інтерфейсу I^2C.

Завдання лабораторної роботи

- для варіанту **АПК** намалювати принципову схему підключень мікроконтролера і пам'яті до шини I^2C, відповідно до варіанту для виконання лабораторної роботи (таб. 3.3.1);
- для варіанту **«Proteus»** використовувати готову схему відповідно до варіанту для виконання лабораторної роботи;
- створити алгоритм роботи програми роботи с зовнішньою пам'яттю (запис і читання даних по заданій адресі):
  - запис даних типу int8, int16, int32, рядка символів;
  - читання даних типу int8, int16, int32, рядка символів;
- написати програму роботи з зовнішньою пам'яттю (запис і читання даних по заданій адресі);
- здійснити запис даних типу int8, int16, int32, рядок символів;
- здійснити читання даних типу int8, int16, int32, рядок символів;
- здійснити виведення результатів на LCD відповідно до варіанту для виконання лабораторної роботи.

Виведення на LCD повинне містити:

- записані дані;
- прочитані дані;
- номер лабораторної роботи;
- групу студента;
- прізвище та ініціали студента.

Примітка: робота із завданням може бути організована за допомогою меню або з демонстрацією роботи в послідовності виконання завдань із інтервалом у 2-5 сек.

Варіанти для виконання лабораторної роботи «EEPROM»

Таблиця 3.3.1 – Варіанти для виконання лабораторної роботи **«EEPROM»**

	Адреса		**Дані**			
№	**Intxx**	**Рядок**	**Int8**	**Int16**	**Int32**	**Рядок**
1	10	20	11	1111	11111111	Прізвище, ініціали
2	20	30	12	1212	12121212	Прізвище, ініціали
3	30	40	13	1313	13131313	Прізвище, ініціали
4	40	50	14	1414	14141414	Прізвище, ініціали
5	50	60	15	1515	15151515	Прізвище, ініціали
6	60	70	16	1616	16161616	Прізвище, ініціали
7	70	80	17	1717	17171717	Прізвище, ініціали
8	80	90	18	1818	18181818	Прізвище, ініціали
9	90	100	19	1919	19191919	Прізвище, ініціали
10	100	110	20	2020	20202020	Прізвище, ініціали
11	110	120	21	2121	21212121	Прізвище, ініціали
12	120	130	22	2222	22222222	Прізвище, ініціали
13	130	140	23	2323	23232323	Прізвище, ініціали
14	140	150	24	2424	24242424	Прізвище, ініціали
15	150	160	25	2525	25252525	Прізвище, ініціали

Послідовність виконання лабораторної роботи для варіанту АПК

1) для варіанту **АПК** намалювати принципову схему підключень мікроконтролера і пам'яті до шини I^2C, відповідно до варіанту для виконання лабораторної роботи;
2) створити свій директорій для файлів проекту;
3) відкрити інтегроване середовище розробки **PCWH**;
4) створити проект у інтегрованому середовищі розробки **PCWH**;
5) створити алгоритм роботи програми роботи с зовнішньою пам'яттю;
6) написати програму мовою програмування **C**, що реалізує створений алгоритм, відповідно до варіанту для виконання лабораторної роботи;

7) скомпілювати програму, одержати бінарні файли ***.HEX** і ***.COF** (файли з розширеннями ***.HEX** і ***.COF** створюються підчас компіляції програми);
8) завантажити бінарний файл ***.HEX** у пам'ять програм мікроконтролера програмою **PICkit.exe**;
9) на **АПК** зкумутувати схему підключень (підключити до мікроконтролера пам’ять, шину I^2C) відповідно до варіанту для виконання лабораторної роботи;
10) виконати програму.

Послідовність виконання лабораторної роботи 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 роботи програми роботи с зовнішньою пам'яттю;
6) написати програму мовою програмування **C**, що реалізує створений алгоритм, відповідно до варіанту для виконання лабораторної роботи;
7) скомпілювати програму, одержати бінарні файли ***.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 лабораторної роботи **«EEPROM»**;
10) відкрити файл проекту з розширенням ***.DSN**;
11) у середовищі моделювання **«Proteus»** змінити схему підключень відповідно до варіанту для виконання лабораторної роботи;
12) завантажити заздалегідь скомпільований бінарний файл ***.COF** або ***.HEX** у мікроконтролер;

13) у середовищі моделювання **«Proteus»** виконати програму.

Послідовність виконання лабораторної роботи 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 лабораторної роботи **«EEPROM»**;
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 лабораторної роботи **«EEPROM»**;
9) відкрити файл проекту з розширенням ***.DSN**;
10) у середовищі моделювання **«Proteus»** змінити схему підключень відповідно до варіанту для виконання лабораторної роботи;
11) створити алгоритм роботи програми роботи с зовнішньою пам'яттю;
12) у інтегрованому середовищі розробки **PCWH**, використовуючи шаблон програми самостійно написати програму мовою програмування **C**, що реалізує створений алгоритм, відповідно до варіанту для виконання лабораторної роботи;
13) скомпілювати програму, одержати бінарні файли ***.HEX** і ***.COF** (файли з розширеннями ***.HEX** і ***.COF** створюються підчас компіляції програми);
14) у середовищі моделювання **«Proteus»** виконати програму.

Примітка: *файл демонстрації виконання лабораторної роботи* ***«EEPROM»*** *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 лабораторної роботи **«EEPROM»**, новий проект можна створити вручну за допомогою команди меню **Project / Create** (рис. 3.3.1).

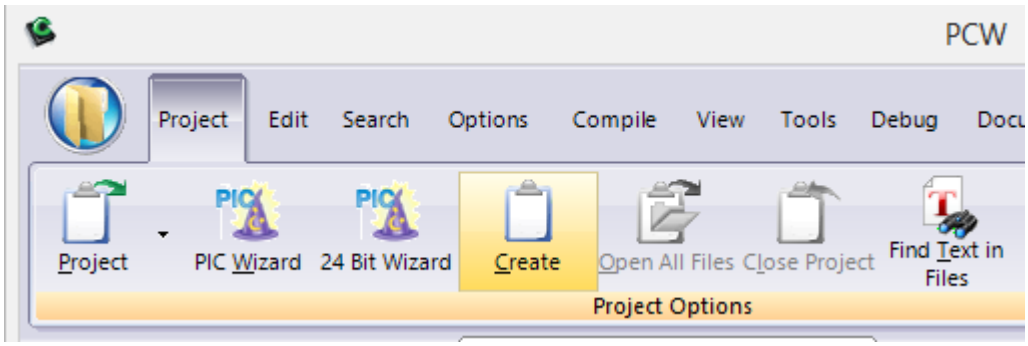

Рисунок 3.3.1 – Створення проекту у ручному режимі
за допомогою команди меню **Project/Create**

Якщо новий проект створений у ручному режимі за допомогою команди меню **Project/Create**, для конфігурування мікроконтролера для роботи з I^2C

У тексті програми або в заголовному файлі вставити строку:

```
#use i2c(Master,Slow,sda=PIN_C4,scl=PIN_C3);
```

Цей рядок призначає контролер Майстром, швидкість обміну – низька, дані прив'язані до виводу мікроконтролера `PIN_C4`, синхронізація – до виводу мікроконтролера `PIN_C3`.

У функцію ініціалізації вставити рядки:

```
output_float(PIN_C3);
output_float(PIN_C4);
```

Ці команди встановлюють виводи шини у режим роботи з відкритим стоком.

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 відповідної лабораторної роботи **«EEPROM»**.

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

Для запуску майстра **PIC Wizard** слід виконати відповідну команду меню **Projec**t, а потім вказати розміщення і ім'я головного файлу проекту.

В результаті відкриється вікно майстра, що складається з розділів з параметрами проекту (рис. 3.3.6).

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. 3.3.2),

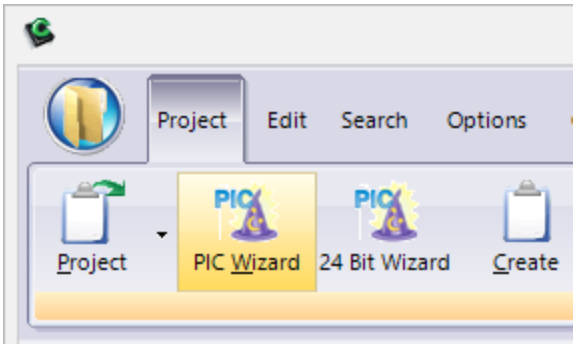

Рисунок 3.3.2 – Процес створення проекту у IDE PCWH
за допомогою майстра PIC Wizard

або натиснути кнопку у вигляді відкритої папки 

2. Вибрати: **New › Project Wizard** (рис. 3.3.3):

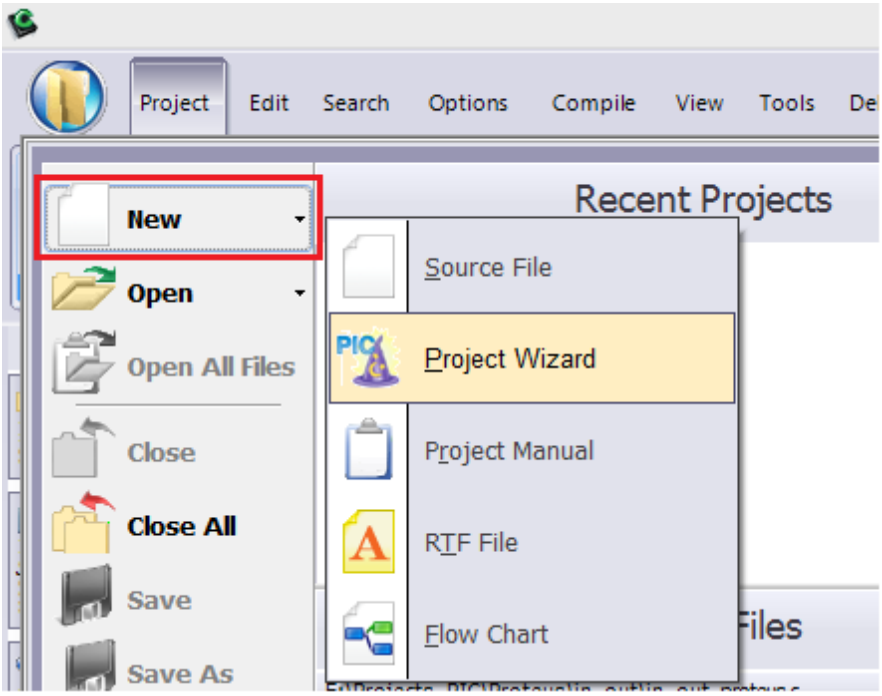

Рисунок 3.3.3 – Процес створення проекту у **IDE PCWH** за допомогою майстра **PIC Wizard**

3. Створити або вибрати свій директорій і записати у нього проект під будь-яким іменем латинським шрифтом, наприклад: «**eeprom.pjt**» (рис 3.3.4):

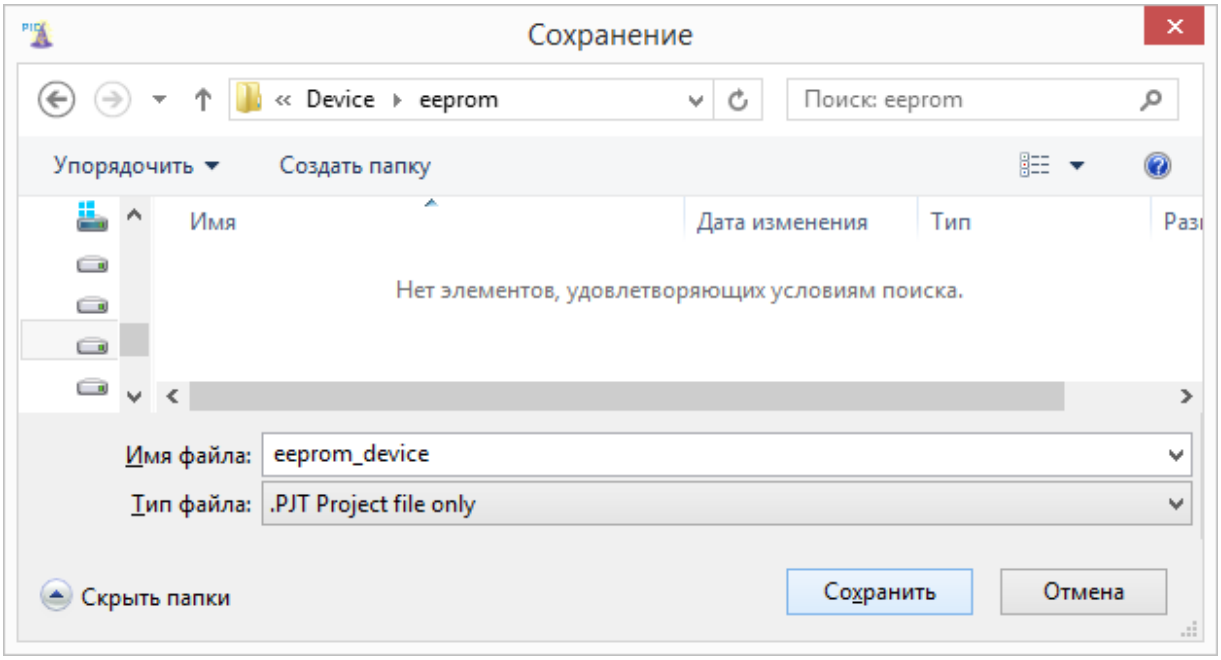

Рисунок 3.3.4 – Процес створення проекту у IDE PCWH за допомогою майстра PIC Wizard

У розділі **General** (рис 3.3.5) 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):**

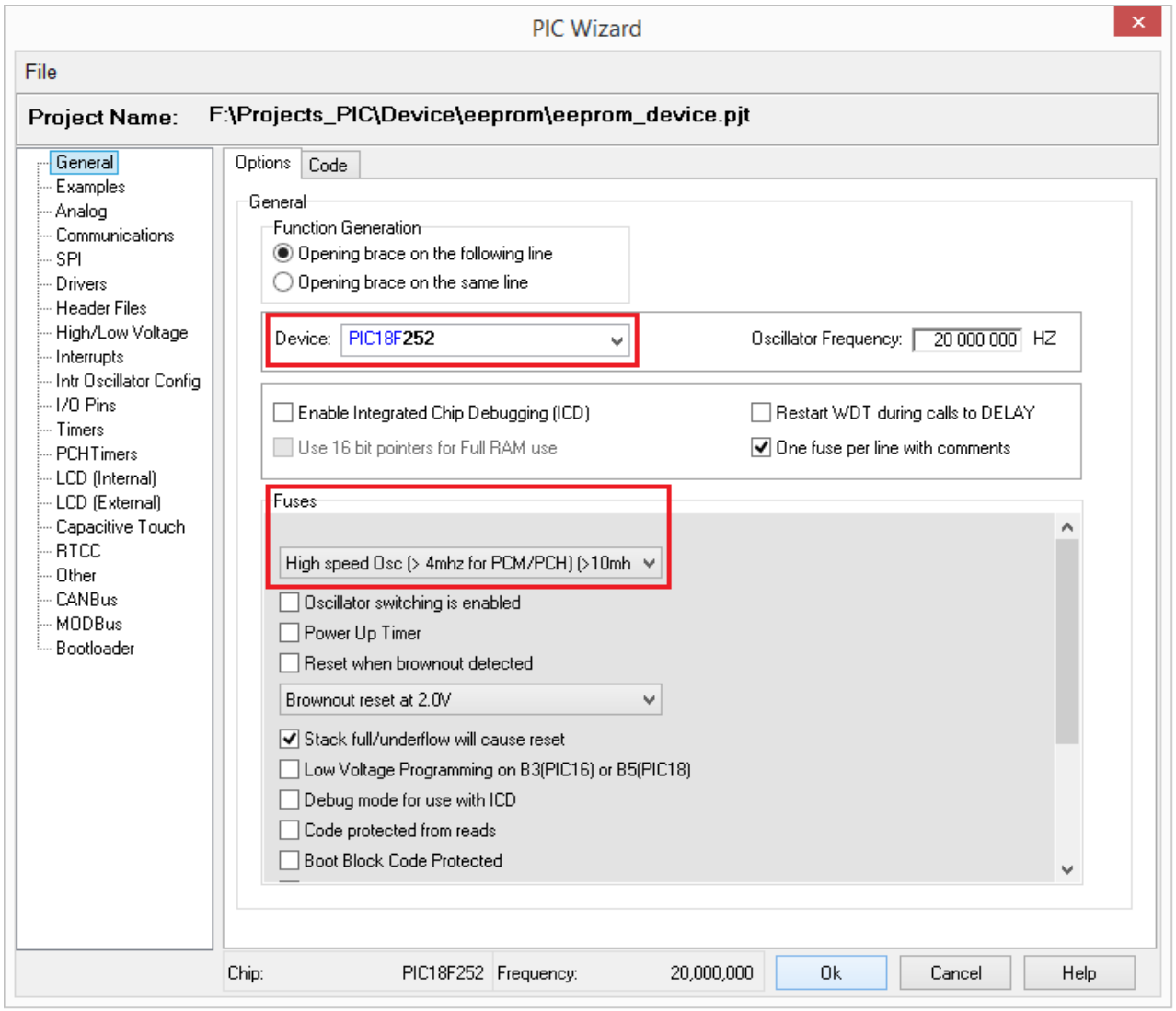

Рисунок 3.3.5 – Розділ **General** майстра **PIC Wizard**

У розділі **Communications** (рис. 3.3.6) 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**.

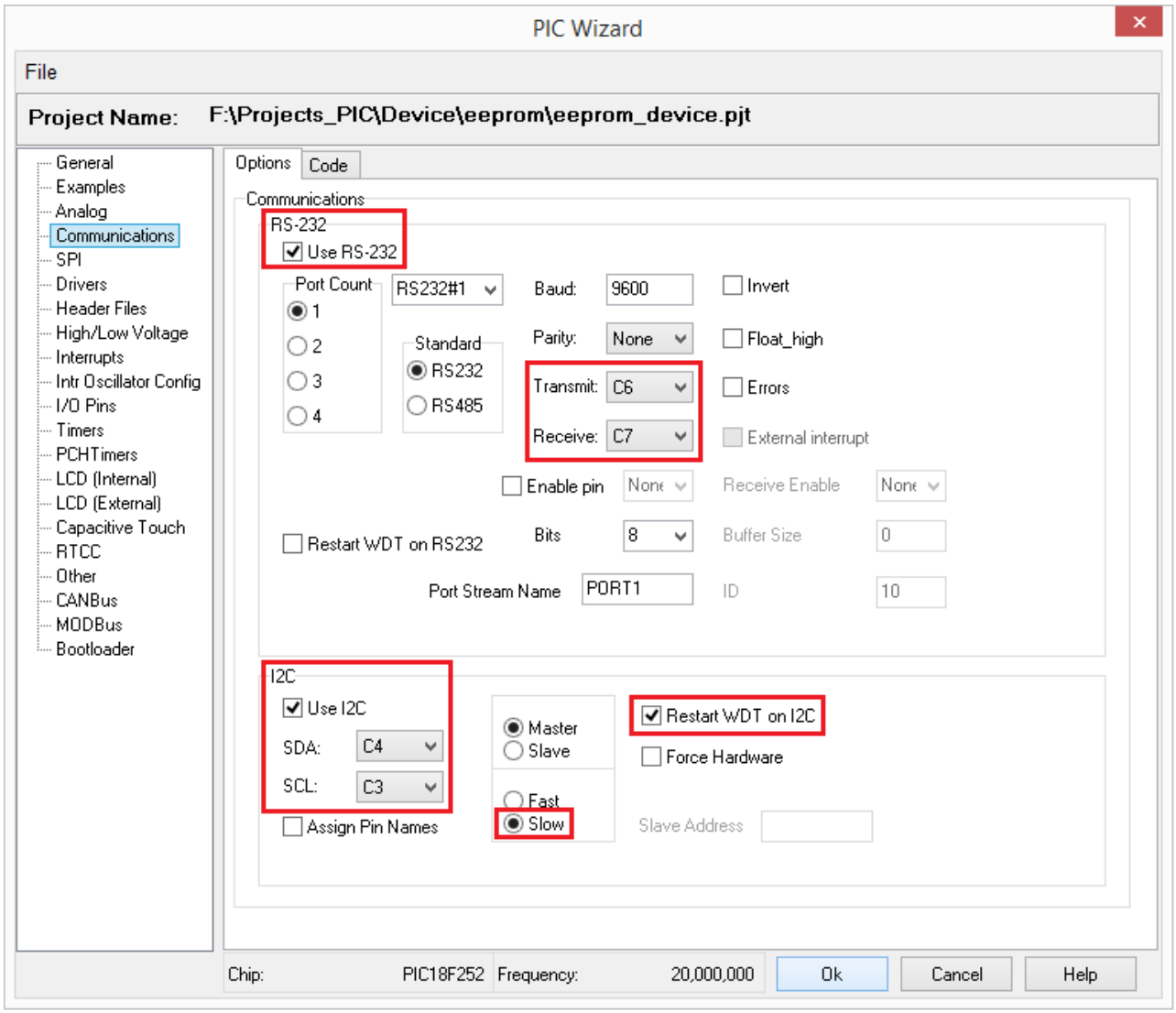

Рисунок 3.3.6 – Розділ **Communications** майстра **PIC Wizard**

6. У розділі **Communications**:

встановити мітку у полі RS-232:

- ☑ Use RS-232;

вказати:

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

встановити мітку у полі I²C:

- ☑ Use I²C;

вказати:

- **SDA:** **C4** ***(SDA**, англ. Serial Data – послідовна лінія даних)*;
- **SCL:** **C3** ***(SCL**, англ. Serial Clock – послідовна лінія тактування);*
- ◉ Slow;
- ☑ Restart WDT on I²C.

Розділ **Analog** (рис. 3.3.7) служить для налаштування вбудованого АЦП (конфігурація аналогових входів, частота і розрядність перетворення).

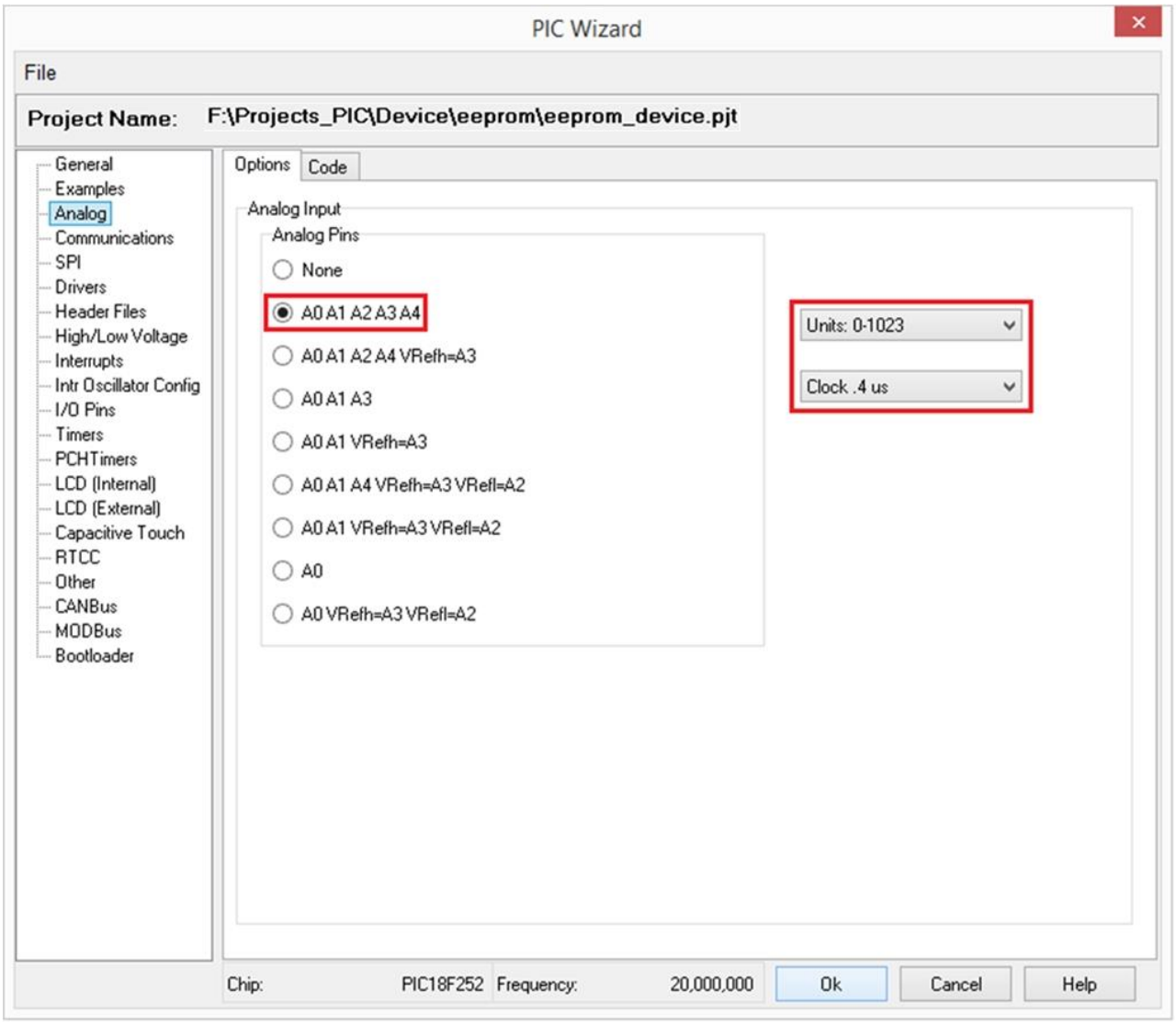

Рисунок 3.3.7 – Конфігурування **АЦП** із **IDE PCWH** при створенні проекту у розділі **Analog** майстра **PIC Wizard**

7. У розділі **Analog** встановити:

у полі **Analog Input**:

- ◉ **A0 A1 A2 A3.**

Вказати:

розрядність АЦП: **10 (0-1023)** розрядів:

- **Units:** **0-1023**
- **Clock:** **4 us**

Конфігурування можна зробити з **IDE PCWH** при створенні проекту (рис. 3.3.7).

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. 3.3.8):

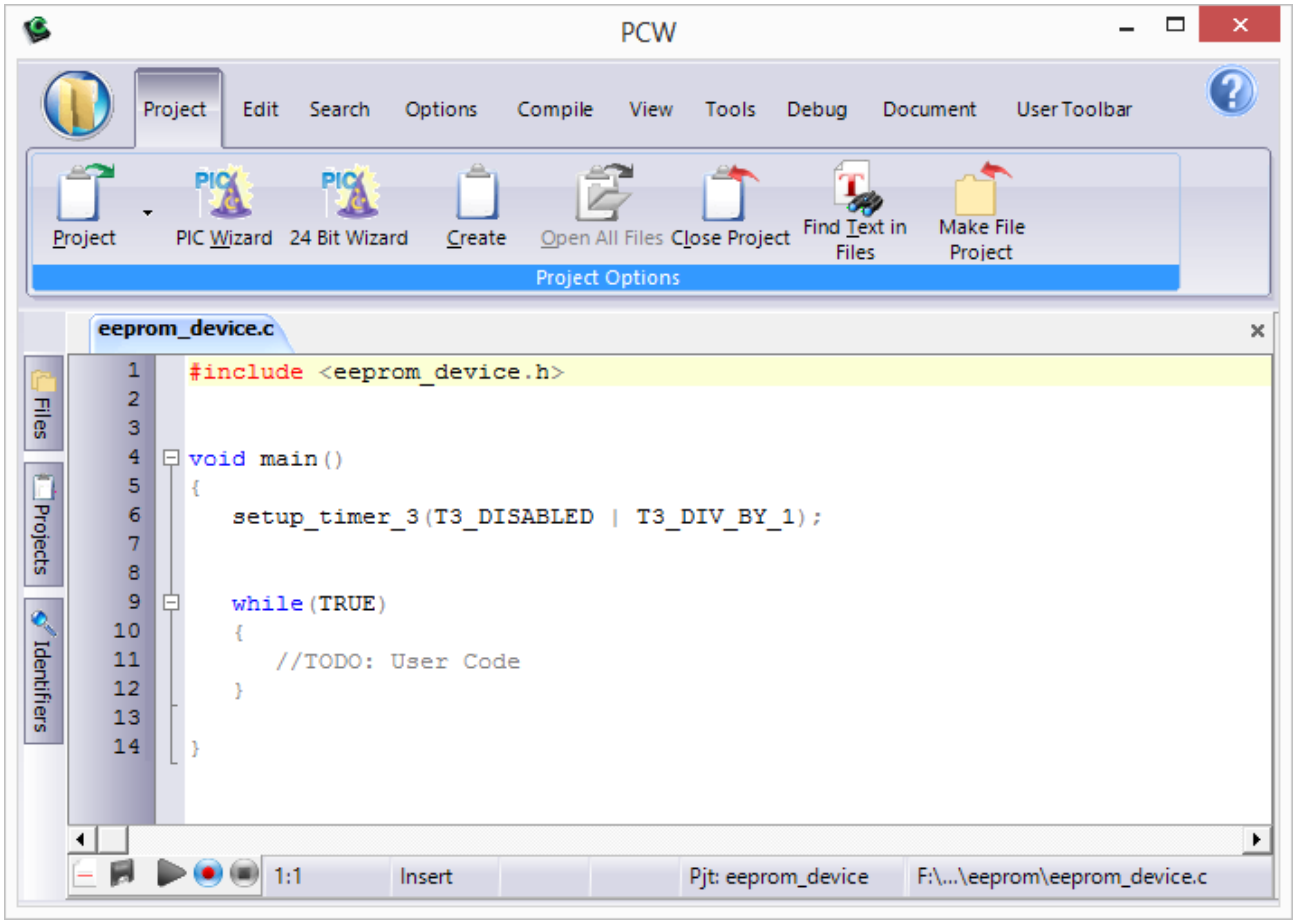

Рисунок 3.3.8 – Згенерований майстром **PIC Wizard** програмний код

Лістинг 3.3.1 – Приклад програмного коду, згенерованого майстром **PIC Wizard**

eeprom_device.c

```
#include <eeprom_device.h>
void main()
{
setup_timer_3(T3_DISABLED | T3_DIV_BY_1);
      while(TRUE)
      {
      //TODO: User Code
      }
}
```

eeprom_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)
#use i2c(Master,Slow,sda=PIN_C4,scl=PIN_C3,restart_wdt)
```

Проект готовий, можна приступати до написання програми.

ПОЯСНЕННЯ ДО ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ №3 - «EEPROM»

Послідовний інтерфейс I^2C

Передача біта

Для передачі одного біта даних використовується один імпульс сигналу синхронізації на лінії SCL, при цьому рівень на лінії SDA повинен бути незмінним протягом високого рівня на лінії SCL, і може змінюватися тільки при низькому рівні на SCL (рис. 3.3.9). Виключеннями служать два особливі стани – Start і Stop.

Стани Start і Stop

Існують два особливі стани шини I^2C – Start і Stop, які служать для індикації початку і кінця передачі і відповідно, переходу шини в неактивний стан. Доти, поки не встановлений стан Start, сигнали на лініях SDA і SCL можуть бути довільними (рис. 3.3.9). Це дозволяє використовувати одну лінію SDA і кілька ліній SCL.

Стан START – «1» на лінії SCL – перехід від «1» до «0» на лінії SDA.

Стан STOP – «1» на лінії SCL – перехід від «0» до «1» на лінії SDA.

Ці два стани завжди генеруються майстром.

Детектування станів Start і Stop зазвичай проводиться апаратно.

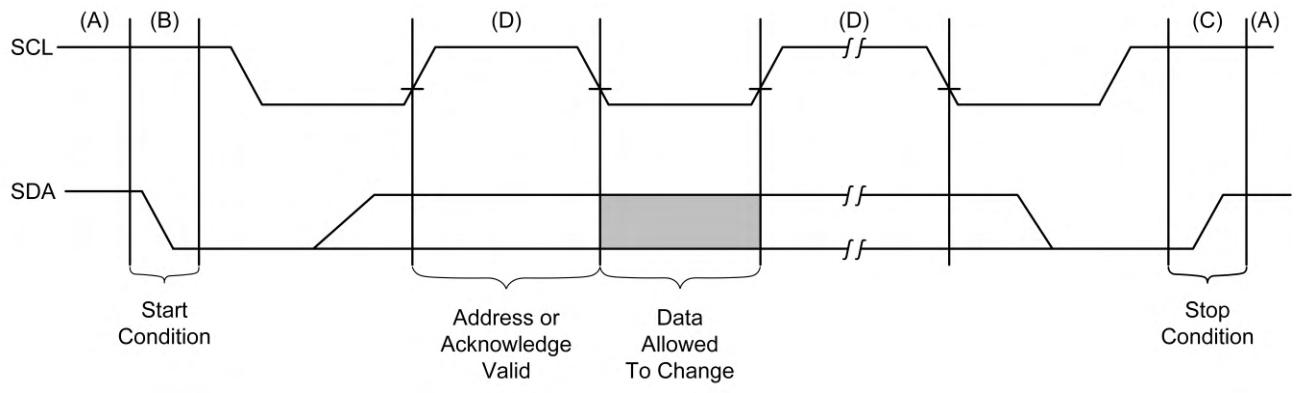

Рисунок 3.3.9 – Стани Start і Stop

Передача даних

Усі передачі проводяться 8-розрядними байтами. Число байтів, які можуть бути передані за одну передачу обмежене ємністю прийомного буфера приймача.

Кожний байт повинен супроводжуватися бітом підтвердження (АСК). Дані передаються починаючи зі старшого біта (рис. 3.3.10).

Квитування

Обмін даними по шині I^2C повинен бути детермінований. Для виконання цієї вимоги існує механізм квитування.

Для підтвердження передачі байту передавач встановлює лінію SDA у «1» протягом синхронізуючого імпульсу. Приймач при цьому повинен виставити квитанцію АСК – «0» на лінії SDA (рис. 3.3.10).

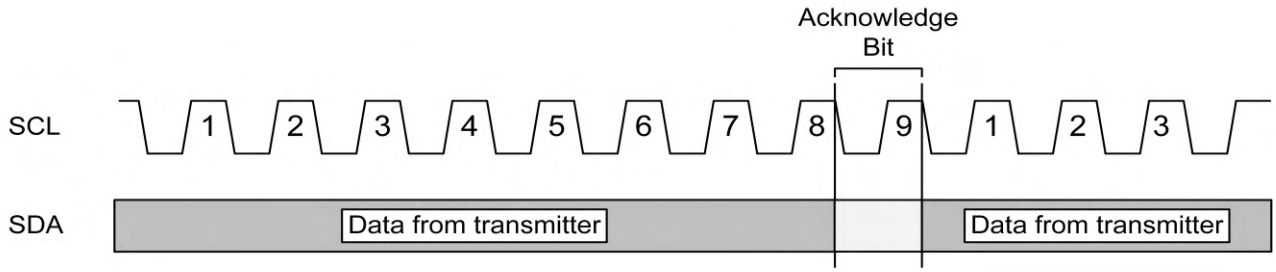

Рисунок 3.3.10 – Передача даних і квитування

Приймач повинен генерувати сигнал АСК після одержання кожного байту.

Якщо приймач не підтверджує підлеглу адресу (наприклад, пристрій не готовий, тому що виконує деяку внутрішню функцію), лінія SDA даних повинна бути залишена у «1». Майстер встановлює стан Stop, щоб перервати передачу.

Після передачі останнього байту майстер виставляє стан Stop, при цьому приймач повинен звільнити лінію даних.

EEPROM – пам'ять 24LC256

EEPROM – пам'ять 24LC256 має об'єм 32 kb. Інтерфейс передачі – I^2C. Структура пам'яті 24LC256 і розташування виводів представлене на рис. 3.3.11 і 3.3.12.

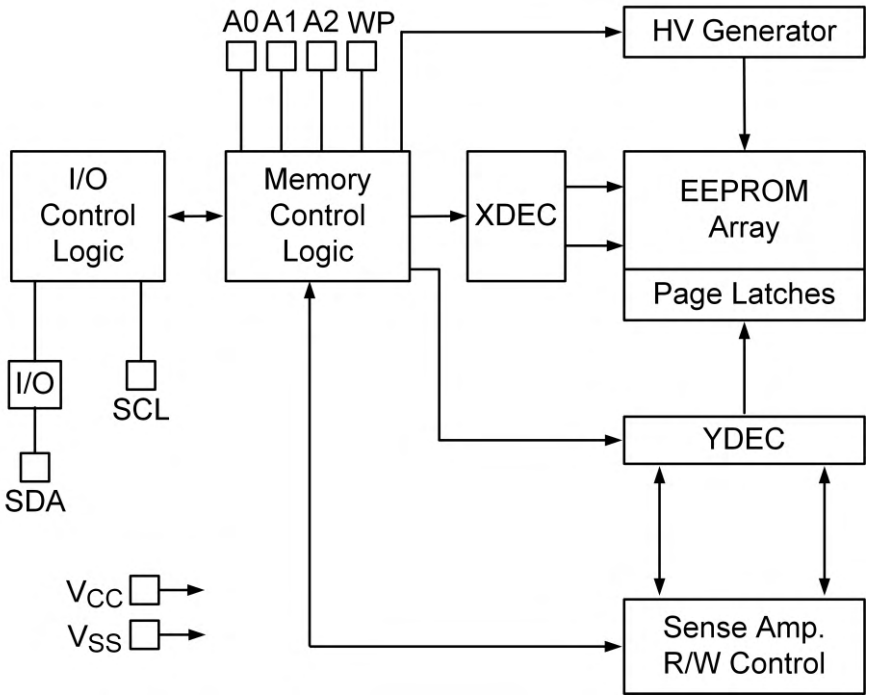

Рисунок 3.3.11 – Структура пам'яті 24LC256

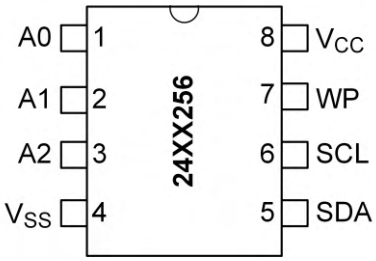

Рисунок 3.3.12 – Розташування виводів 24LC256

Таблиця 3.3.2 – Призначення виводів 24LC256

Name	8-pin PDIP	8-pin SOIC	8-pin TSSOP	8-pin MSOP	8-pin DFN	Function
A0	1	1	1	—	1	User Configurable Chip Select
A1	2	2	2	—	2	User Configurable Chip Select
(NC)	—	—	—	1,2	—	Not Connected
A2	3	3	3	3	3	User Configurable Chip Select
VSS	4	4	4	4	4	Ground
SDA	5	5	5	5	5	Serial Data
SCL	6	6	6	6	6	Serial Clock
(NC)	—	—	—	—	—	Not Connected
WP	7	7	7	7	7	Write-Protect Input
VCC	8	8	8	8	8	+1.8V to 5.5V (24AA256) +2.5V to 5.5V (24LC256) +1.8V to 5.5V (24FC256)

Робота з пам'яттю

Перед виконанням будь-якої операції з пам'яттю, на шину I^2C необхідно виставити керуючий байт. Формат керуючого байту представлений на рис. 3.3.13.

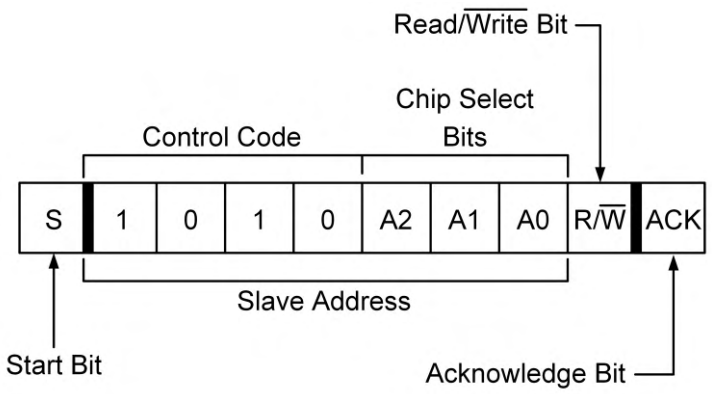

Рисунок 3.3.13– Формат керуючого байту

Старші біти 7-4 керуючого байту містять код 1010 – ідентифікатор типу пристрою.

Біти 3-2 містять адресу пристрою в системі і можуть приймати значення 000 – 111.

Таким чином, у системі може знаходитися 8 чипів пам'яті.

Біт 0 визначає операцію з пам'яттю:

- 0 – запис в пам'ять;
- 1 – читання з пам'яті.

На рисунку 3.3.14 представлений формат адресації чипа і адреси комірки пам'яті.

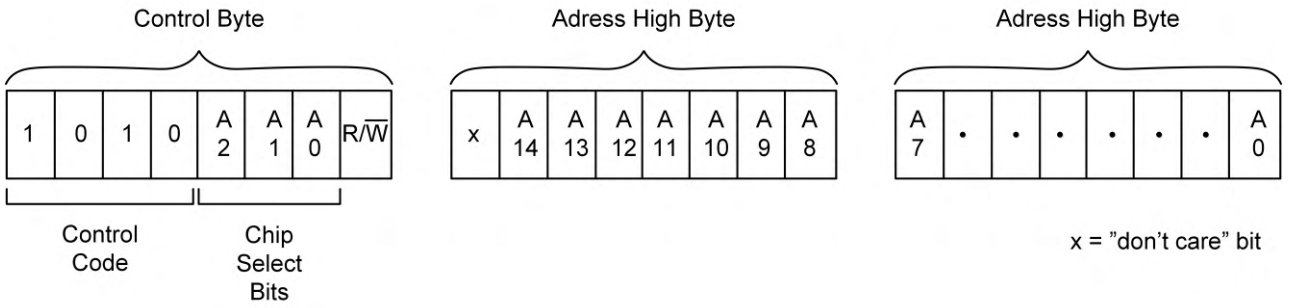

Рисунок 3.3.14 – Формат адресації чипа і адреси комірки пам'яті

Запис байту

Запис байту з мікроконтролера у пам'ять здійснюється у такий спосіб (рис. 3.3.15):

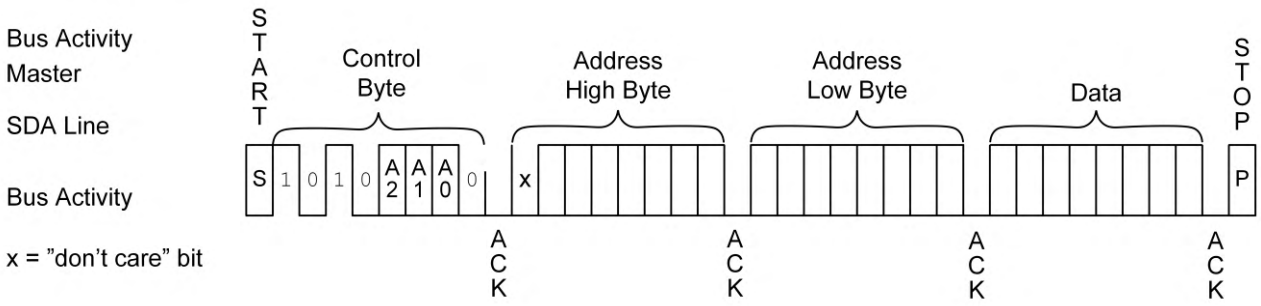

Рисунок 3.3.15 – Запис байту

- активізується шина I^2C – на лінії SDA встановлюється стан Start (0);
- посилається байт керування, у якому біт R/W = «0»;
- очікується квитанція АСК протягом 8–го біта синхронізації на лінії SCL;
- якщо квитанція надходить (0 на лінії SDA), то посилається старший байт адреси комірки пам'яті;
- очікується квитанція АСК;
- якщо квитанція надходить, то посилається молодший байт адреси комірки пам'яті;
- очікується квитанція АСК;
- якщо квитанція надходить, то посилається байт даних;
- очікується квитанція АСК;
- якщо квитанція надходить, то на лінії SDA встановлюється стан Stop (1).

Стан Stop встановлюється контролером у всіх випадках, якщо квитанція АСК не надходить вчасно.

Запис послідовності байт

При записі побайтно послідовності байт, на кожний байт даних доводиться три службові байти, що в кілька раз знижує швидкість обміну даними з пам'яттю.

Тому існує режим запису, який дозволяє записати в пам’ять послідовність байтів даних. Дані при цьому містяться в прийомний буфер.

При записі байту даних розряди 0-6 молодшого байту адреси автоматично інкрементується, тому немає необхідності адресації для кожного байту.

Процес запису послідовності байтів представлений на рис. 3.3.16.

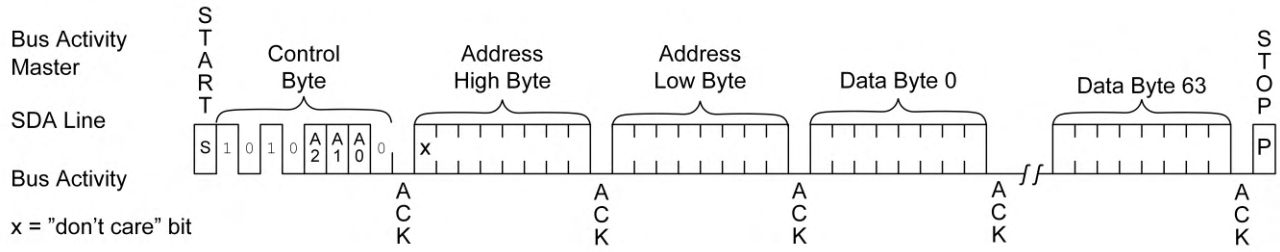

Рисунок 3.3.16 – Запис послідовності байтів

При послідовному записі байтів даних вони поміщаються в буфер пам'яті, адресуємий молодшим байтом адреси, а при надходженні стану Stop переписуються у комірки пам'яті.

Розмір буфера становить 64 байти. Якщо кількість байтів перевищить це значення, то в молодшому байті адреси виникне переповнення і дані в буфері будуть перезаписані.

Перевірка готовності

При роботі з пам'яттю завжди необхідно переконатися, що пам'ять знаходиться у стані готовності. Після прийому послідовності байт у прийомний буфер і одержання стану Stop, пам'ять переходить у режим запису вмісту буфера в комірки пам'яті. Оскільки запис байту в комірку пам'яті забирає тривалий час (5 мс), то ця пам'ять зайнята і квитанція АСК не виставляється.

Для одержання стану готовності необхідно:

1) після виконання попередньої операції завершити її станом Stop;
2) виставити стан Start;
3) послати керуючий байт, у якому біт R/W = «0»;
4) повірити стан шини SDA на «0»;
5) якщо квитанція не поступила, перейти до пункту 3.

Блок-схема алгоритму перевірки готовності пам'яті представлена на рис. 3.3.17.

Рисунок 3.3.17 – Блок-схема алгоритму перевірки готовності пам'яті

Читання байту по поточній адресі

Після виконання операції читання байту, внутрішній лічильник адреси 24LC256 інкрементується, тому при читанні наступного байту, його можна не адресувати (рис. 3.3.18).

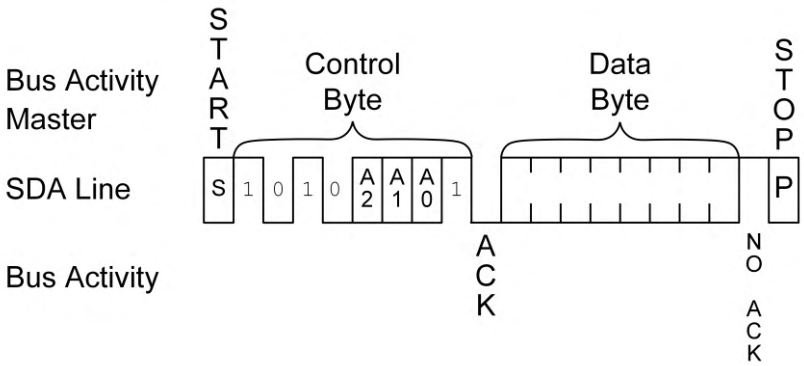

Рисунок 3.3.18 – Читання байта по поточній адресі

Читання байту по довільній адресі

Процедура читання байту по довільній адресі представлена на рис. 3.3.19.

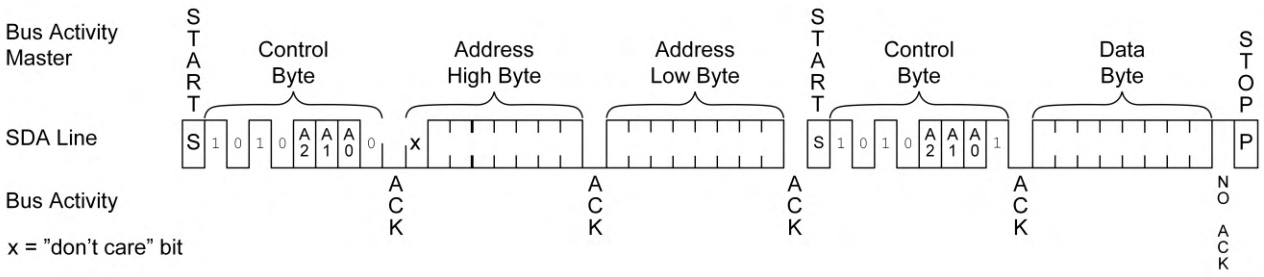

Рисунок 3.3.19 – Читання байту по довільній адресі

Процедура читання байту виконується у наступній послідовності:

- активізується шина I^2C – на лінії SDA встановлюється стан Start;
- посилається байт керування, у якому біт R/W = «0»;
- очікується квитанція АСК на протязі 8-го біта синхронізації на лінії SCL;
- якщо квитанція надходить (0 на лінії SDA), то посилається старший байт адреси комірки пам'яті;
- очікується квитанція АСК;
- якщо квитанція надходить, то посилається молодший байт адреси комірки пам'яті;
- очікується квитанція АСК;
- встановлюється стан Start;
- посилається байт керування, у якому біт R/W = «1» (читання);
- якщо квитанція надходить, то зчитується байт даних;
- квитанція після прочитаного байту не очікується;
- на лінії SDA встановлюється стан Stop.

Читання послідовності байт по довільній адресі

Процедура читання послідовності байтів аналогічна процедурі читання одного байту, за винятком того, що виключається адресація кожного байту, що

читається, оскільки внутрішній лічильник адреси пам'яті інкрементується автоматично.

Процедура послідовності байт по довільній адресі представлена на рис. 3.3.20.

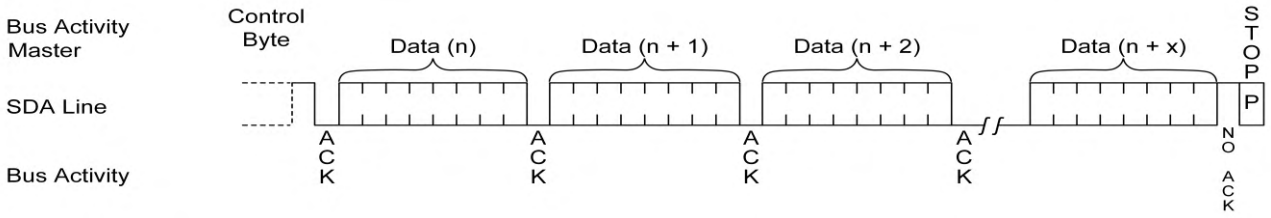

Рисунок 3.3.20 – Читання послідовності байтів по довільній адресі

Об'єм зчитаних даних обмежується об'ємом пам'яті.

Керування шиною I^2C

Керування шиною I^2C здійснюється функціями:

```
i2c_start();                                    //Стан Start
i2c_stop();                                     //Стан Stop
```

Посилка управляючого байту:

```
i2c_write(control_byte);            //ID пам'яті, адреса пам'яті, операція
```

Посилка адреси:

```
i2c_write(make8(address,1));                    //старший байт адреси
i2c_write(make8(address,0));                    //молодший байт адреси
```

Запис байту в пам’ять:

```
i2c_write(data);                                //записати байт
```

Читання байту з пам'яті:

```
Data = i2c_read();                              //читати байт
```

Перевірка готовності пам'яті

Перевірка готовності пам'яті здійснюється наступною функцією:

```
int8 EEP_ready()
{
      int1 ack;                      //квитанція
      i2c_start();                   //якщо команда одержить квитанцію,
      ack = i2c_write(ctrl_write);  //значить пристрій готовий
      i2c_stop();
      return !ack;
}
```

Приклад реалізації функції записи байту в пам'ять

```
EEP_write_byte(int16 address, int8 data)
{
      while(!EEP_ready());                 //чекати готовності пам'яті
      i2c_start();                         //старт
      i2c_write(ctrl_write);               //ID пам'яті
      i2c_write(make8(address,1));         //старший байт адреси
      i2c_write(make8(address,0));         //молодший байт адреси
      //====== записати байт
      i2c_write(data);                     //записати байт
      i2c_stop();                          //стоп
}
```

Розташування виводів шини I^2C мікроконтролера представлено на рис. 3.3.21.

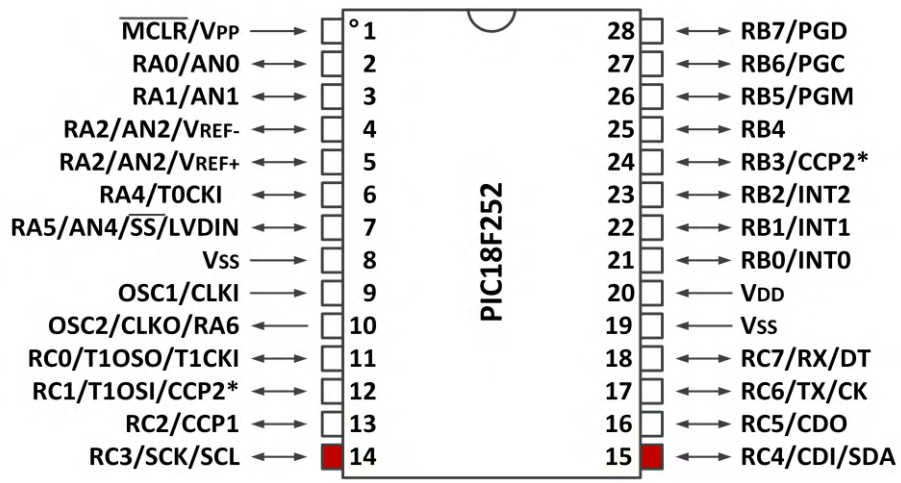

Рисунок 3.3.21 – Розташування виводів шини I^2C мікроконтролера

Приклади виконання лабораторної роботи «EEPROM» відповідно до варіанту завдання для лабораторної роботи «EEPROM»

Варіант завдання для виконання лабораторної роботи **«EEPROM»** представлений у таблиці 3.3.3.

Таблиця 3.3.3 – Завдання для виконання лабораторного практикуму **«EEPROM»**

Адреса		**Дані**				
Intxx	**Рядок**	**Int8**	**Int16**	**Int32**	**Рядок**	**Клавіатура**
100	15	25	2525	25252525	Смірнов В.В.	A1

Принципова схема підключень для варіанту АПК

Принципова схема підключень для варіанту **АПК**, для виконання лабораторного практикуму **«EEPROM»** виглядає у такий спосіб (рис. 3.3.22):

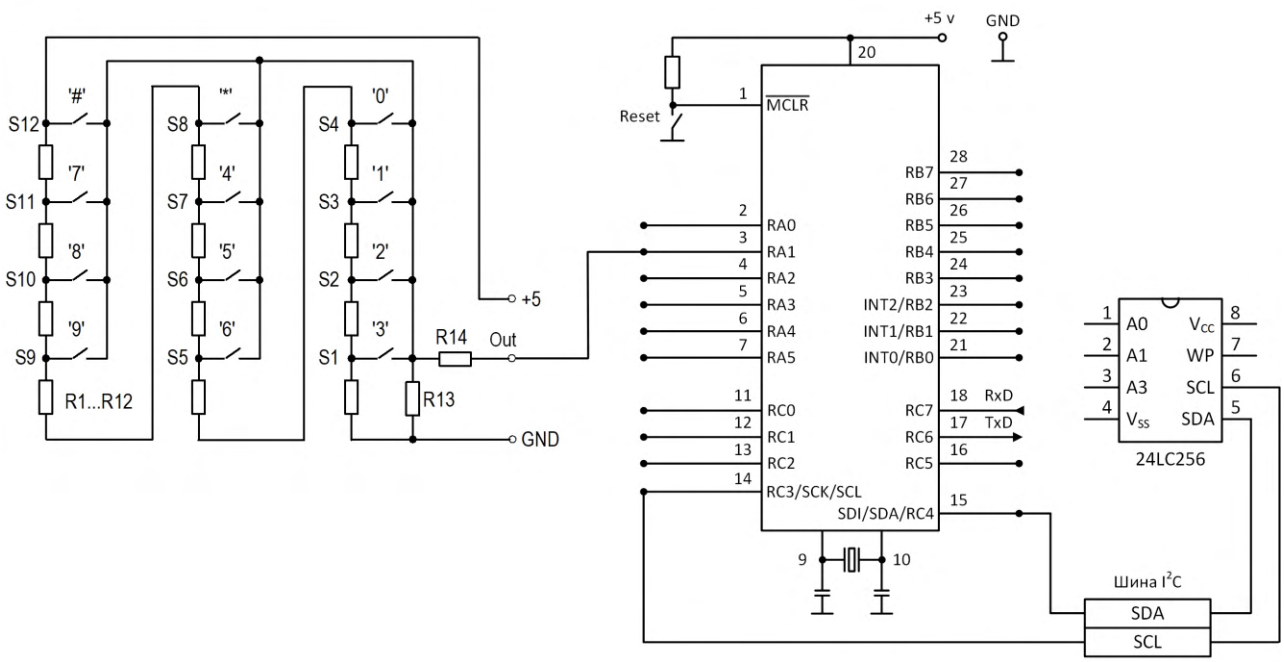

Рисунок 3.3.22 – Принципова схема підключень
для лабораторного практикуму **«EEPROM»** для варіанту **АПК**

Принципова схема підключень для варіанту «Proteus»

Принципова схема для варіанту **«Proteus»**, для виконання лабораторного практикуму **«EEPROM»** виглядає у такий спосіб (рис. 3.3.3):

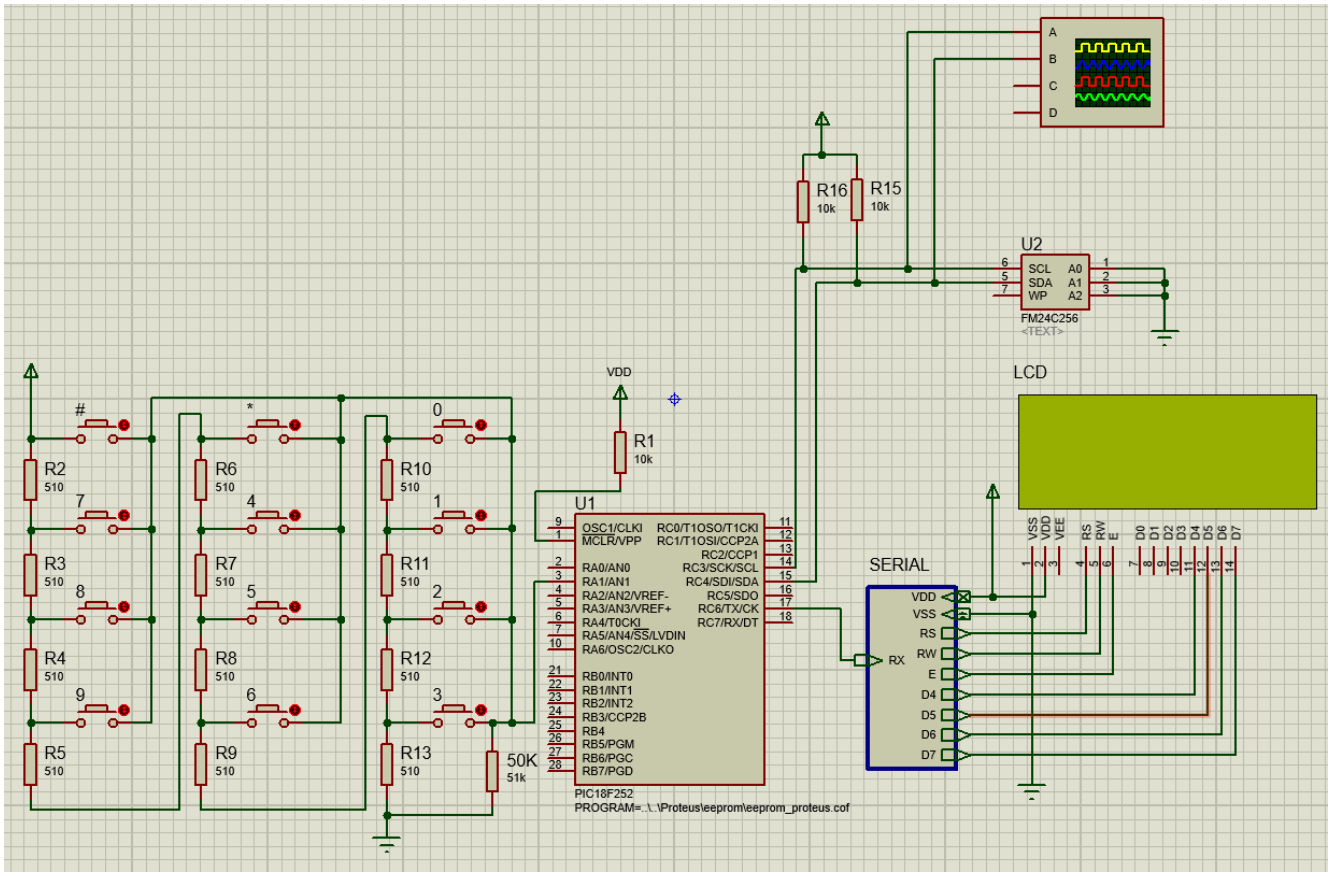

Рисунок 3.3.23 – Принципова схема підключень лабораторного практикуму **«EEPROM»** для варіанту **«Proteus»**

Результат виконання лабораторного практикуму «EEPROM» для варіанту «Proteus»

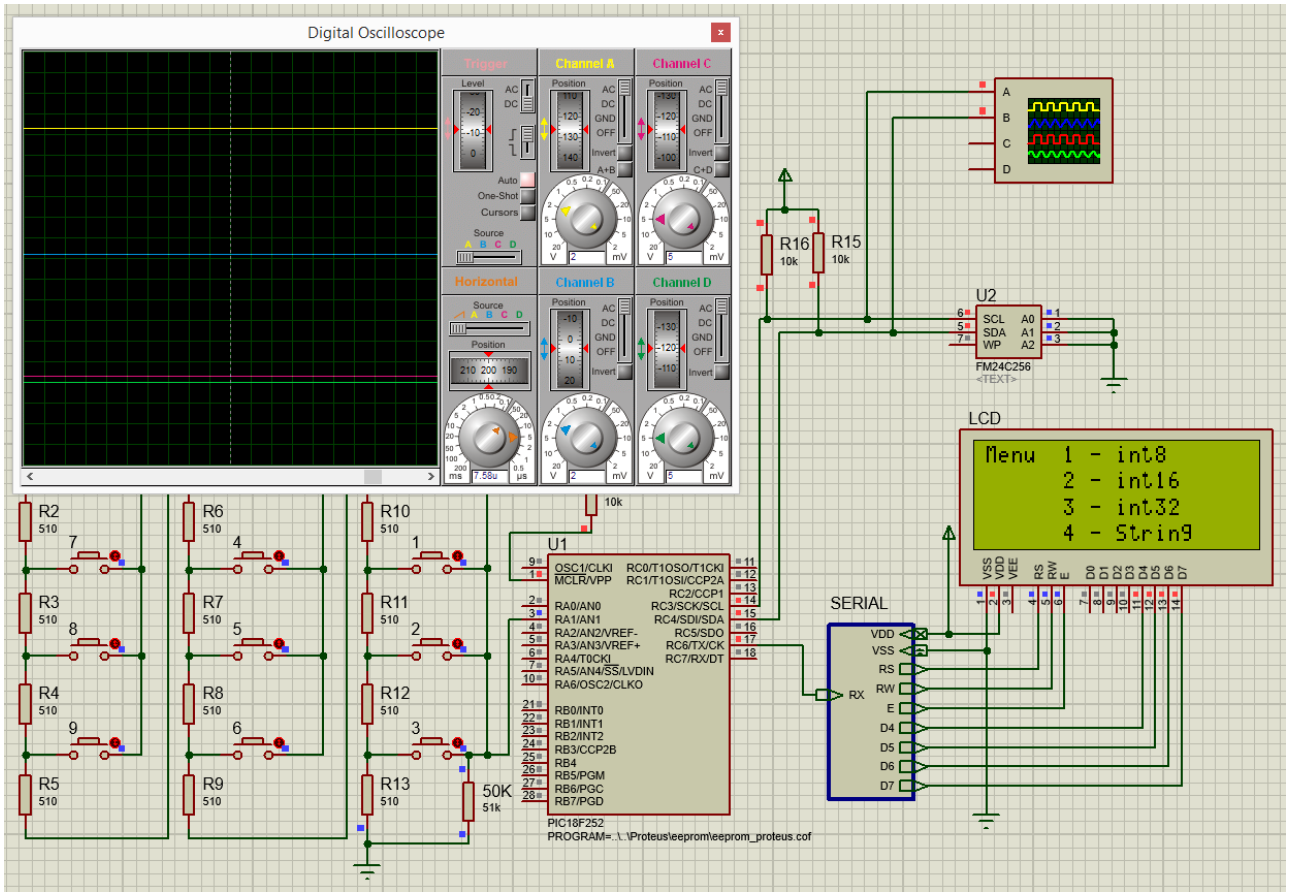

Рисунок 3.3.24 – Запис/читання даних типу int8, int16, int32, рядок символів

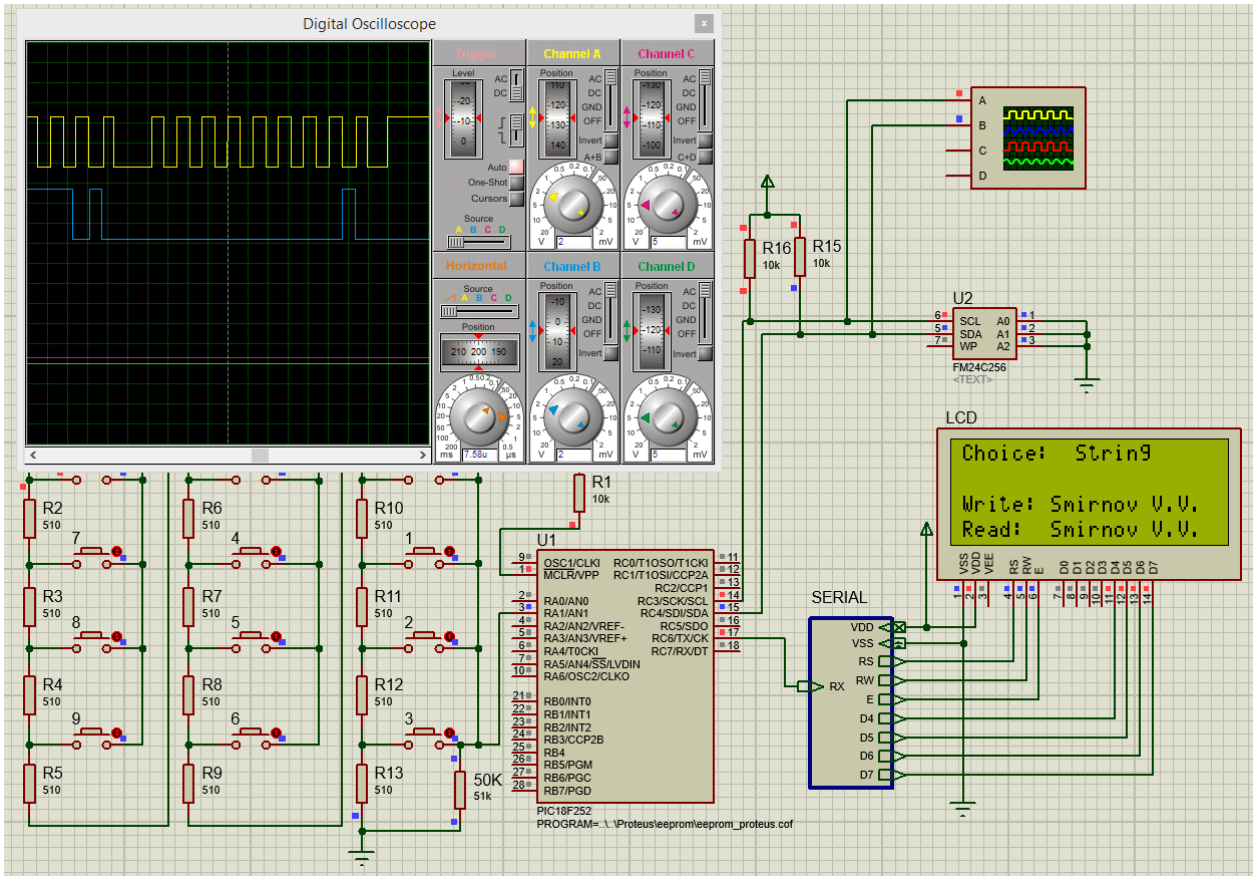

Рисунок 3.3.25 – Читання рядка символів

Контрольні питання

1) призначення шини I^2C і принцип роботи;
2) порядок ініціалізації мікроконтролера для роботи із шиною I^2C - і призначення параметрів команд;
3) процедура запису даних в пам’ять по шині I^2C;
4) процедура читання даних в пам’ять по шині I^2C;
5) алгоритм визначення готовності пам'яті.

ЛАБОРАТОРНА РОБОТА № 4 - «SERVO»

Тема: Сервоприводи

Ціль роботи:

Отримання навичок роботи керування сервоприводом.

Завдання лабораторної роботи

- для варіанту **АПК** намалювати принципову схему підключень відповідно до варіанту для виконання лабораторної роботи (таб. 3.4.1);
- для варіанту **«Proteus»** використовувати готову схему відповідно до варіанту для виконання лабораторної роботи;
- створити алгоритм програми керування валом сервоприводу;
- здійснити керування валом сервоприводу в межах 180 градусів за допомогою потенціометра і АЦП шляхом перетворення напруги на движку потенціометра у задаючий вплив (тривалість керуючого імпульсу) для сервоприводу;
- здійснити виведення на LCD тривалості керуючого імпульсу.

Виведення на LCD повинне містити:

- номер лабораторної роботи;
- групу студента;
- прізвище та ініціали студента;
- тривалість керуючого імпульсу.

Послідовність виконання лабораторної роботи для варіанту АПК

1) для варіанту **АПК** намалювати принципову схему підключень відповідно до варіанту для виконання лабораторної роботи;
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 керування валом сервоприводу;

6) написати програму мовою програмування **C**, що реалізує створений алгоритм, відповідно до варіанту для виконання лабораторної роботи;
7) скомпілювати програму, одержати бінарні файли ***.HEX** і ***.COF** (файли з розширеннями ***.HEX** і ***.COF** створюються підчас компіляції програми);
8) завантажити бінарний файл ***.HEX** у пам'ять програм мікроконтролера програмою **PICkit.exe**;
9) на **АПК** зкумутувати схему підключень (сервопривід та потенціометр) відповідно до варіанту для виконання лабораторної роботи;
10) виконати програму.

Послідовність виконання лабораторної роботи 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 керування валом сервоприводу;
6) написати програму мовою програмування **C**, що реалізує створений алгоритм, відповідно до варіанту для виконання лабораторної роботи;
7) скомпілювати програму, одержати бінарні файли ***.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 лабораторної роботи **«SERVO»**;
10) відкрити файл проекту з розширенням ***.DSN**;
11) у середовищі моделювання **«Proteus»** змінити схему підключень відповідно до варіанту для виконання лабораторної роботи;

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

Послідовність виконання лабораторної роботи 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 лабораторної роботи **«SERVO»**;
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 лабораторної роботи **«SERVO»**;
9) відкрити файл проекту з розширенням ***.DSN**;
10) у середовищі моделювання **«Proteus»** змінити схему підключень відповідно до варіанту для виконання лабораторної роботи;
11) створити алгоритм програми керування валом сервоприводу;
12) у інтегрованому середовищі розробки **PCWH**, використовуючи шаблон програми самостійно написати програму мовою програмування **C**, що реалізує створений алгоритм, відповідно до варіанту для виконання лабораторної роботи;
13) скомпілювати програму, одержати бінарні файли ***.HEX** і ***.COF** (файли з розширеннями ***.HEX** і ***.COF** створюються підчас компіляції програми);
14) у середовищі моделювання **«Proteus»** виконати програму.

Примітка: *файл демонстрації виконання лабораторної роботи* ***«SERVO»*** *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. 3.4.4). 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 3.4.1):

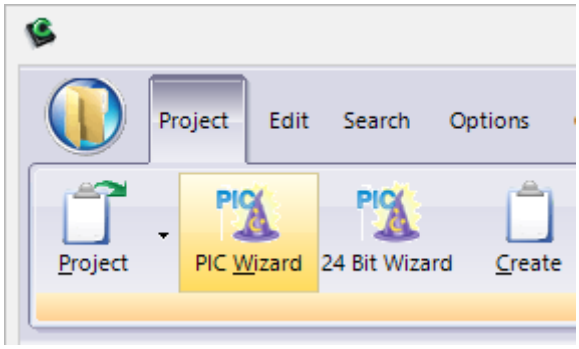

Рисунок 3.4.1 – Процес створення проекту у **IDE PCWH** за допомогою майстра **PIC Wizard**

або натиснути кнопку у вигляді відкритої папки  .

2. Вибрати: **New › Project Wizard** (рис 3.4.2):

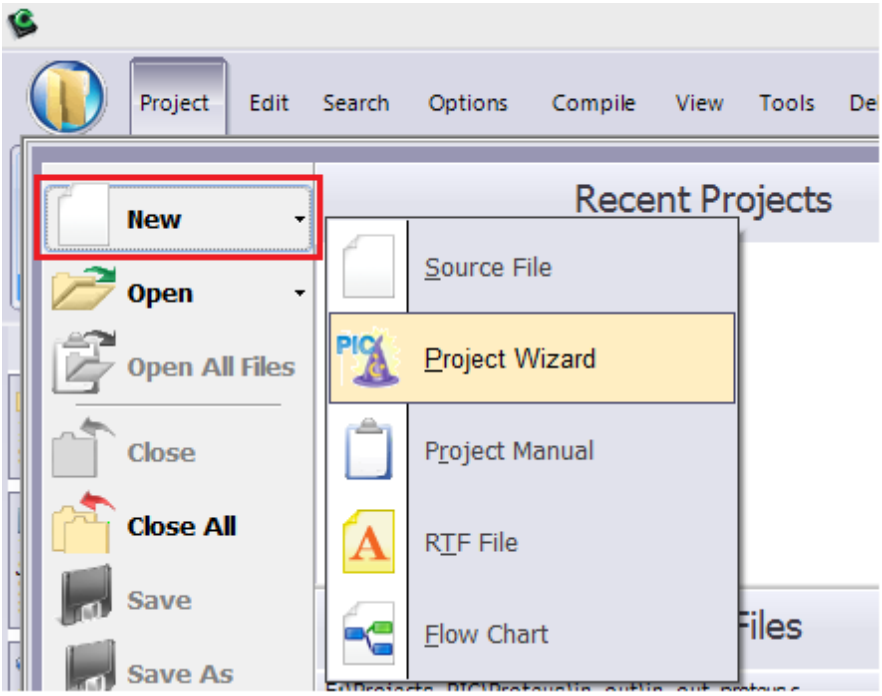

Рисунок 3.4.2 – Процес створення проекту у **IDE PCWH** за допомогою майстра **PIC Wizard**

3. Створити або вибрати свій директорій і записати у нього проект під будь-яким іменем латинським шрифтом, наприклад: «**servo.pjt**» (рис 3.4.3).

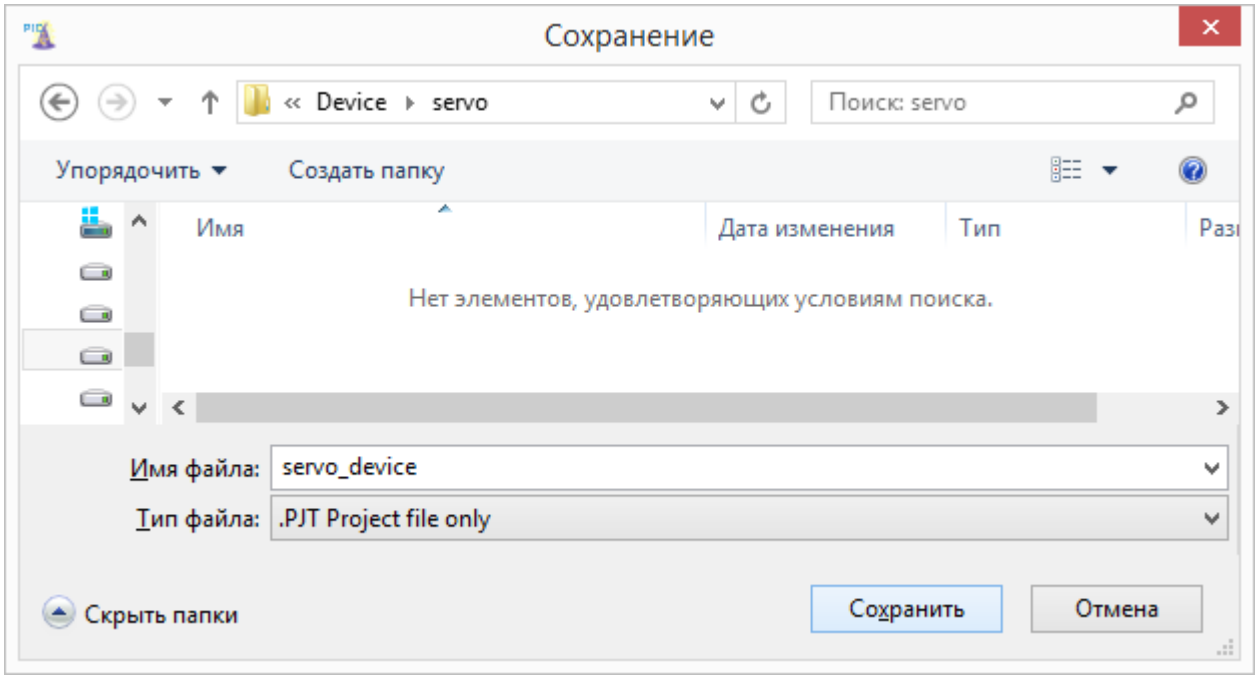

Рисунок 3.4.3 – Процес створення проекту у **IDE PCWH** за допомогою майстра **PIC Wizard**

У розділі **General** (рис 3.4.4) 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):**

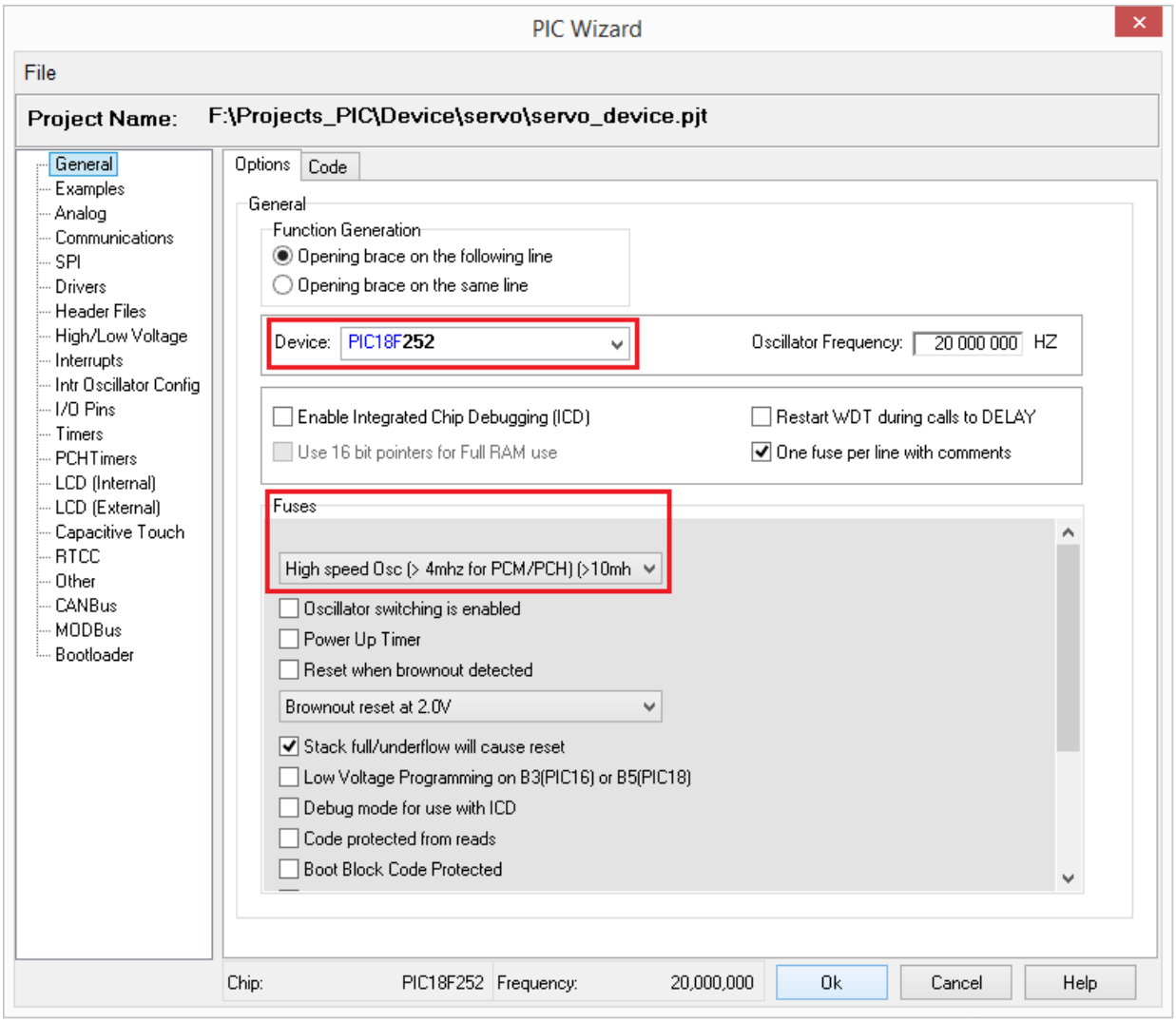

Рисунок 3.4.4 – Розділ **General** майстра **PIC Wizard**

У розділі **Communications** (рис. 3.4.5) 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).

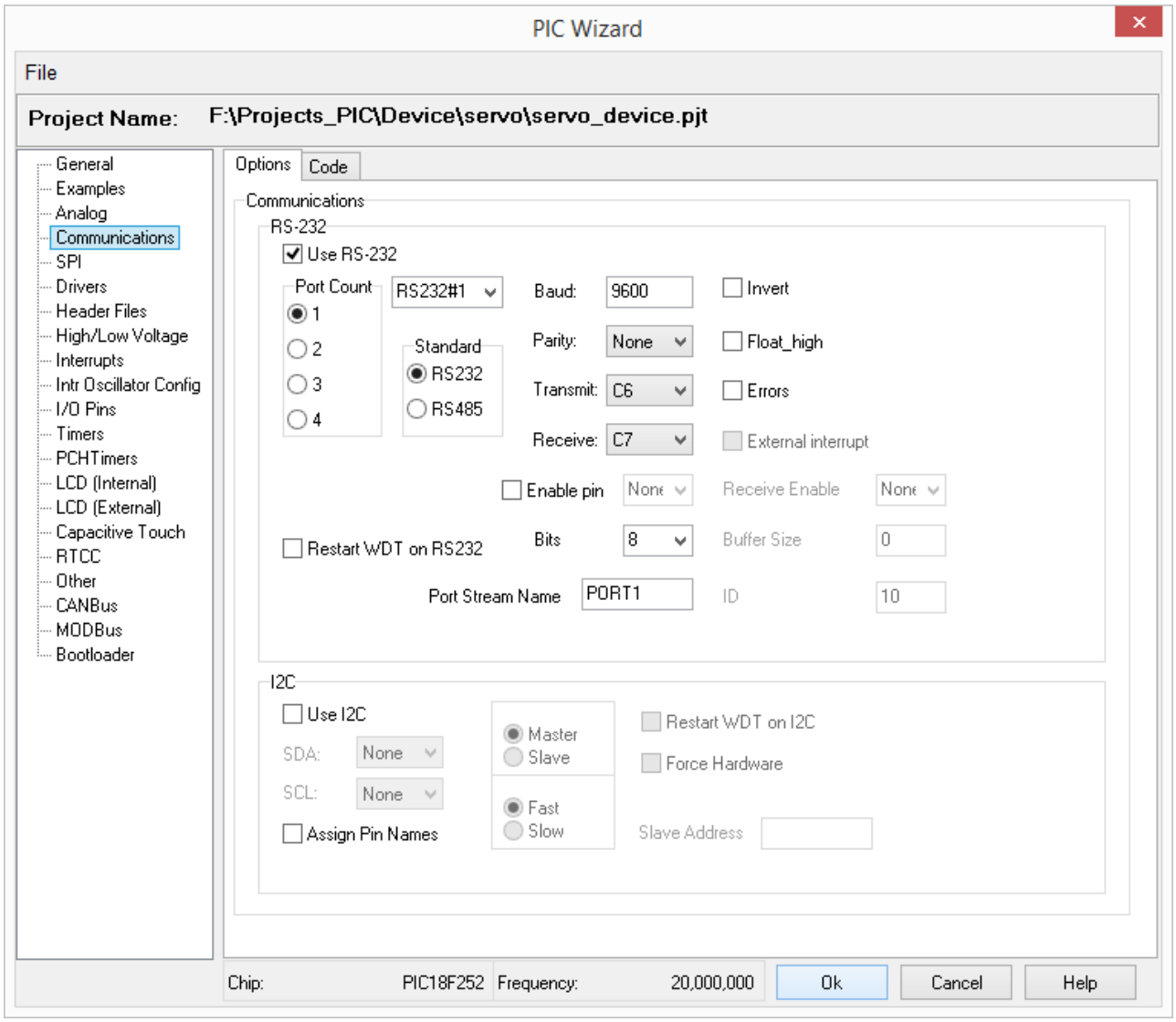

Рисунок 3.4.5 – Розділ **Communications** майстра **PIC Wizard**

Розділ **Analog** (рис. 3.4.6) 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 при створенні проекту:

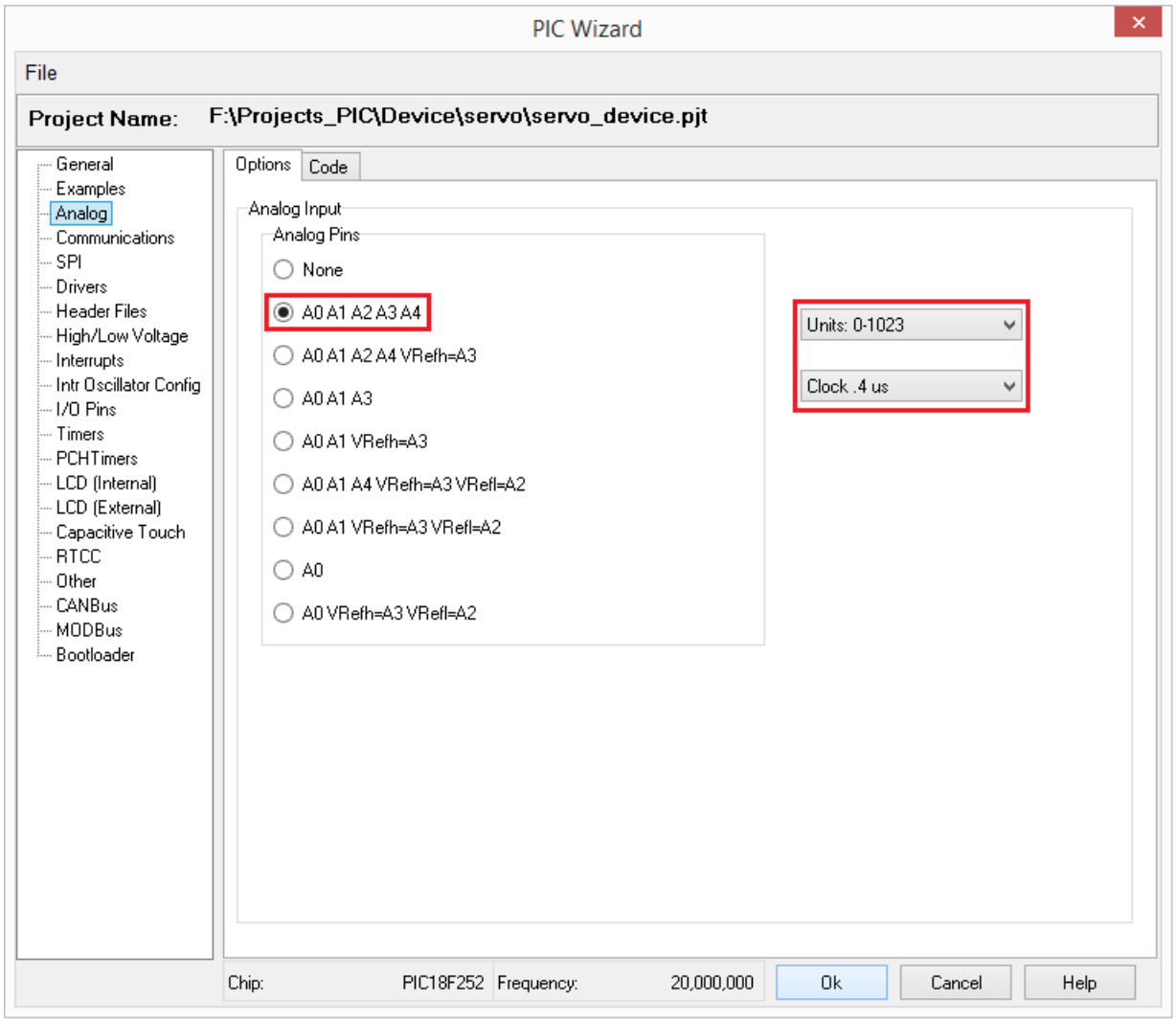

Рисунок 3.4.6 – Конфігурування АЦП із **IDE PCWH** при створенні проекту у розділі **Analog** майстра **PIC Wizard**

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. 3.4.7):

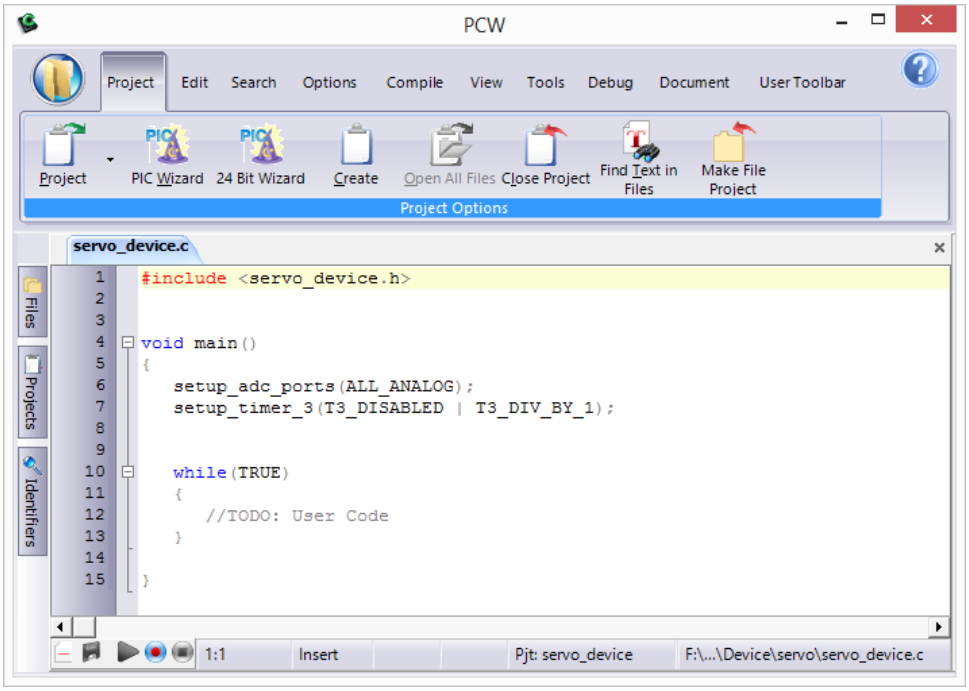

Рисунок 3.4.7 – Згенерований майстром **PIC Wizard** програмний код

Лістинг 3.4.1 – Приклад програмного коду, згенерованого майстром **PIC Wizard**

servo_device.c

```
#include <servo_device.h>

void main()
{
        setup_adc_ports(ALL_ANALOG);
        setup_timer_3(T3_DISABLED | T3_DIV_BY_1);

        while(TRUE)
        {
                //TODO: User Code
        }
}
```

servo_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.

Приклади виконання лабораторної роботи «SERVO» відповідно до варіанту завдання для лабораторної роботи «SERVO»

Принципова схема підключень для варіанту АПК

Принципова схема підключень для варіанту **АПК**, для виконання лабораторної роботи **«SERVO»** виглядає у такий спосіб (рис. 3.4.8):

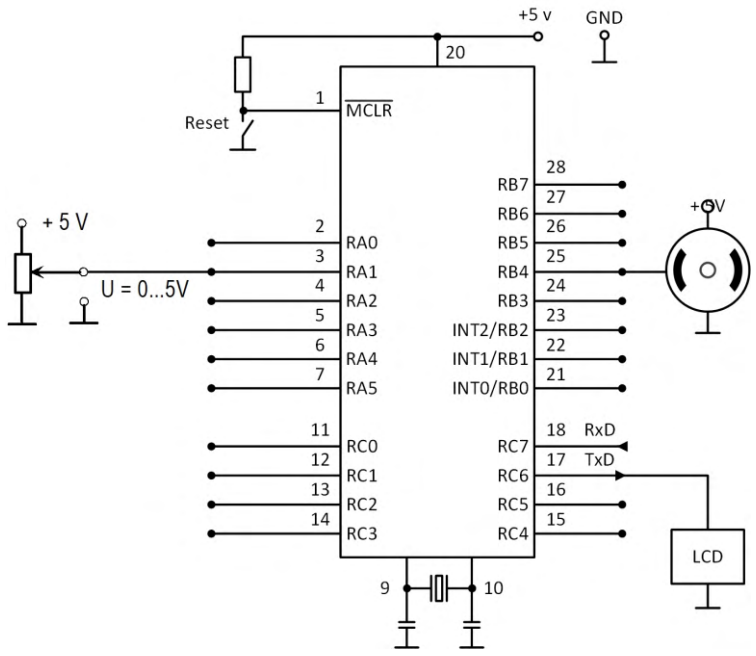

Рисунок 3.4.8 – Принципова схема підключень
для лабораторної роботи **«SERVO»** для варіанту **АПК**

Принципова схема підключень для варіанту «Proteus»

Принципова схема для варіанту **«Proteus»**, для виконання лабораторної роботи **«SERVO»** виглядає у такий спосіб (рис. 3.4.9):

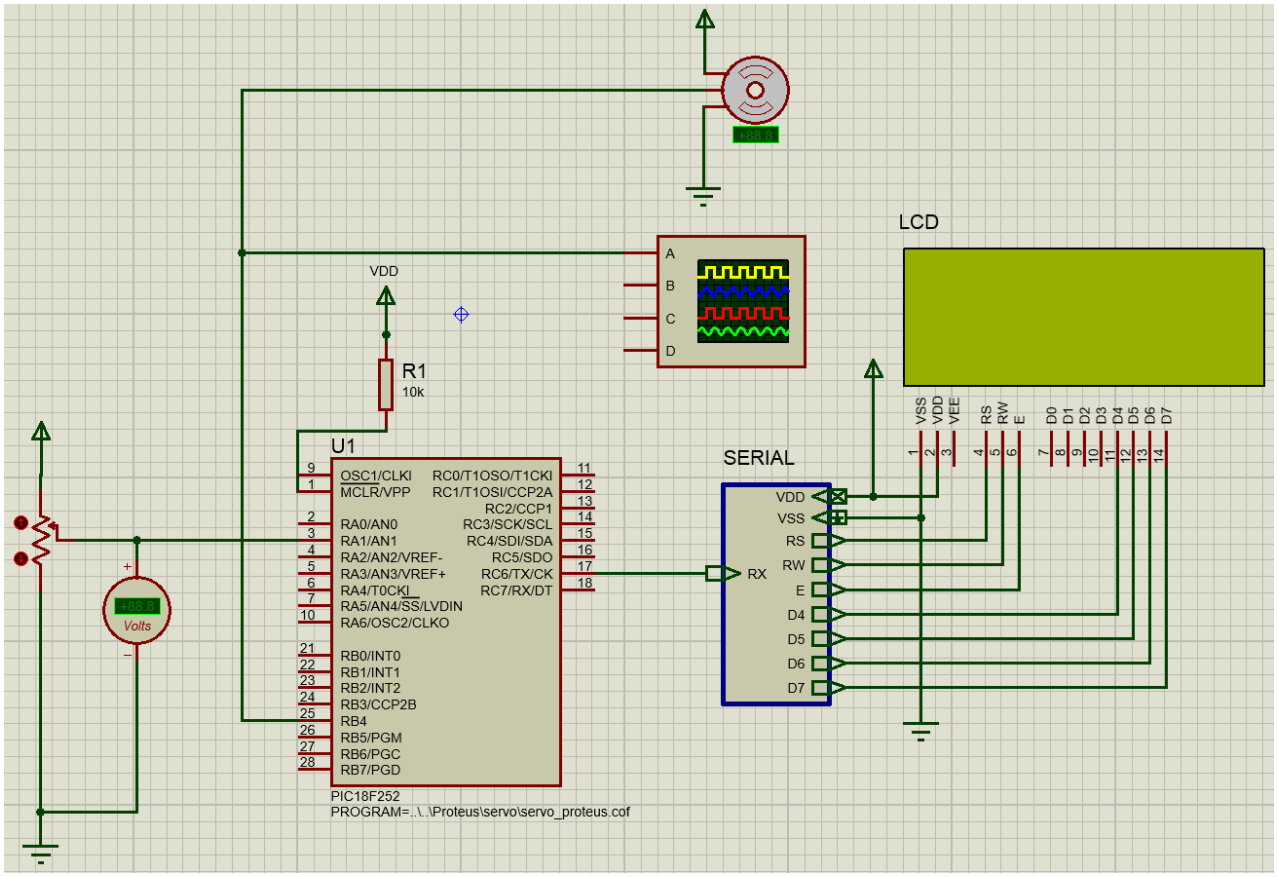

Рисунок 3.4.9 – Принципова схема підключень лабораторної роботи **«SERVO»** для варіанту **«Proteus»**

Результат виконання лабораторного практикуму «SERVO» для варіанту «Proteus»

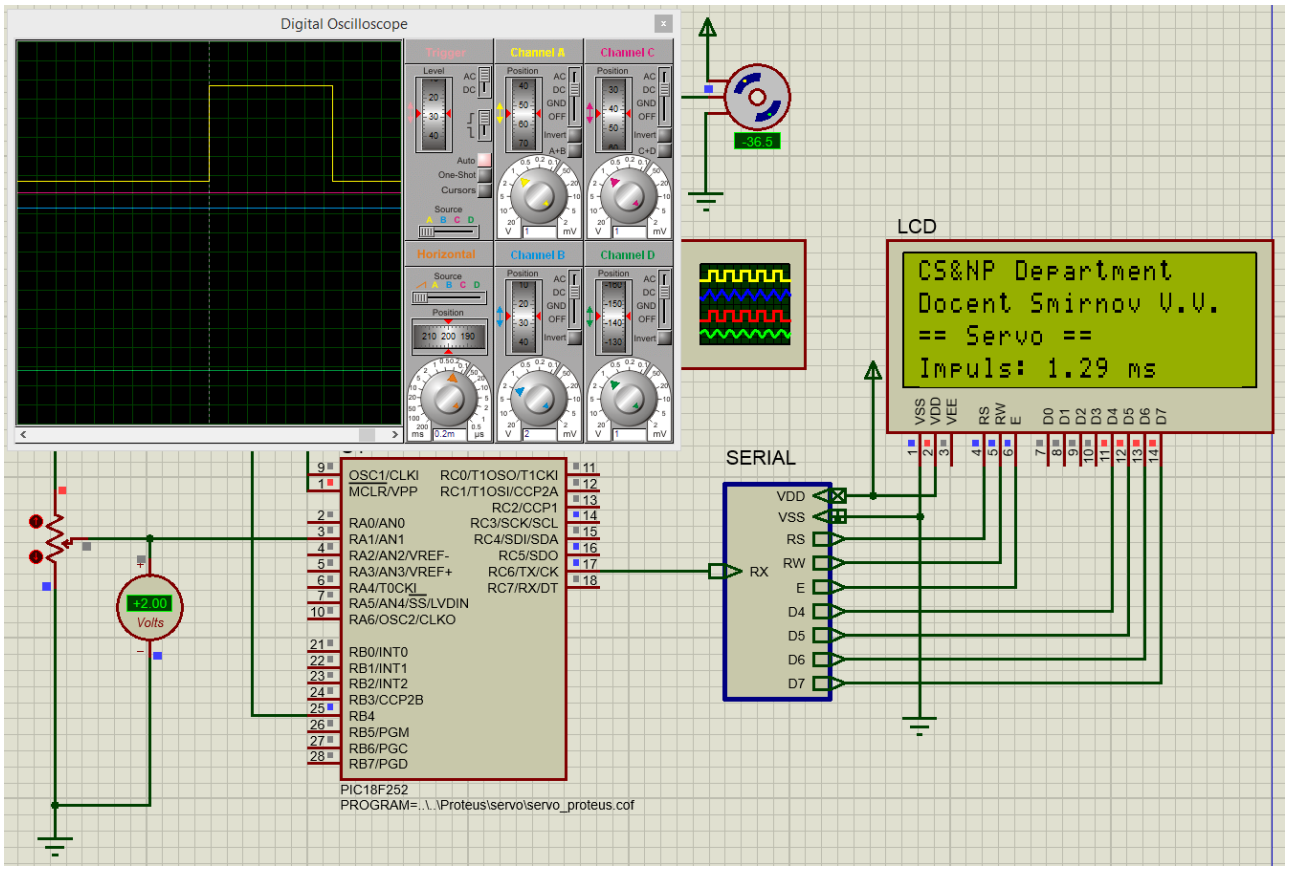

Рисунок 3.4.10 – Результат виконання лабораторної роботи **«SERVO»** для варіанту **«Proteus»**

Контрольні питання

1) призначення сервопривода;
2) область застосування сервопривода;
3) принцип роботи і керування сервоприводом;
4) тривалість керуючого імпульсу для даного сервопривода;
5) відмінність цифрового і аналового сервопривода.

РОЗДІЛ 4. ОСНОВИ МОВИ ПРОГРАМУВАННЯ С

Створення програм для PIC мікроконтролерів здійснюється на мовах програмування Асемблер і С.

Основним недоліком використання мови програмування Асемблер при створенні програм для мікроконтролерів є несумісність програм для різних серій. Несумісність викликана різною кількістю регістрів у мікроконтролерах різних серій і відмінністю їх адрес.

Програми, написані на мові програмування С без змін можуть працювати на всіх серіях мікроконтролерів, що є великою перевагою мови програмування С.

В умовах жорсткої ринкової конкуренції, фірма, яка витратила менше часу на адаптацію програмного забезпечення для нового мікроконтролера буде мати перевагу у часі виходу нового продукту на ринок, і відповідно, більший прибуток.

Тому мова програмування С в даний час є основною мовою створення системного програмного забезпечення для комп'ютерів і мікроконтролерів.

Мова С поєднує в собі властивості мов високого рівня з можливістю доступу до низькорівневих функцій і регістрів комп'ютерів та мікроконтролерів. Будучи мовою середнього рівня, мова С дозволяє здійснювати маніпуляції з бітами, байтами, адресами, машинними словами і покажчиками – основними елементами, з якими працюють функції операційної системи.

ВИЗНАЧЕННЯ МОВИ ПРОГРАМУВАННЯ С

Коментар - це пояснювальний текст, який при компіляції не враховується. Коментарі бувають багаторядковими (починаються з символів /* і закінчуються символами */) і однорядковими (починаються з символів //). В останньому випадку коментарем вважається вся частина рядка, розташована праворуч від символів //.

Приклади:

```
/* Багаторядковий коментар часто розміщують на початку файлу, де він
містить ім'я автора та опис програми */

#include <swc_LCD.h>                // драйвер LCD дисплея
```

Ключове слово - це зарезервоване слово чітко визначеного призначення. Ключові слова не можуть використовуватися в якості ідентифікаторів.

У таблиці 4.1 перелічений список зарезервованих ключових слів у мові С (стандартів C89, C99, C11 (ISO/IEC 9899: 2011)). Оскільки вони використовуються мовою, ці ключові слова недоступні для повторного визначення.

Таблиця 4.1 - Ключові слова

auto	float	signed	_Alignas (since C11)
break	for	sizeof	_Alignof (since C11)
case	goto	static	_Atomic (since C11)
char	if	struct	_Bool (since C99)
const	inline (since C99)	switch	_Complex (since C99)
continue	int	typedef	_Generic (since C11)
default	long	union	_Imaginary (since C99)
do	register	unsigned	_Noreturn (since C11)
double	restrict (since	void	_Static_assert (since C11)
else	C99)	volatile	_Thread_local (since C11)
enum	return	while	
extern	short		

Мова С чутлива до регістру (case sensitive), тобто великі та малі літери у ній розрізняються.

Літерал - постійне значення деякого типу, що використовується у виразах.

Приклади числових літералів:

`10` - число 10 у десятковій формі;

`0xA` - число 10 у шістнадцятковій формі (префікс `0x`);

`0b1010` - число 10 в двійковій формі (префікс `0b`);

`012` - число 10 у восьмеричній формі (префікс `0`);

`10,5` - число з плаваючою точкою;

`105e-1` - число 10,5 в експоненціальній формі;

`10U` - беззнакова константа (суфікс `U`);

`10L` - знакова константа (суфікс `L`).

Символьні літерали поміщаються в одинарні лапки, наприклад, 'B', '*'.

Для позначення недрукованих і спеціальних символів у літералах використовуються так звані escape-послідовності:

`'\a'` - звуковий сигнал;

`'\b'` - клавіша <Backspace>;

`'\f'` - прогін листа;

`'\n'` - символ перекладу рядка;

`'\r'` - повернення каретки;

`'\t'` - горизонтальна табуляція;

`'\v'` - вертикальна табуляція;

`'\0'` - нульовий символ;

`'\\'` - зворотна коса;

`'\''`- апостроф.

Крім того, будь-який символ можна представити за допомогою літерала по його ASCII-коду, наприклад, літерал `'\t'` рівнозначний `'\x09'` (у шістнадцятковому представленні).

Рядкові літерали обмежуються подвійними лапками, а в пам'яті зберігаються як послідовності символів, що закінчуються нульовим символом `'\0'`

Спеціальні символи всередині рядка повинні передувати зворотною косою (`"\"`).

Приклади строкових літералів:

`""` - порожній рядок: один символ `'\0'`;

`"B"` - два символи: `'B'` і `'\0'`;

`"A\tB\n"` - п'ять символів: `'A'`, табуляція, `'B'`, переклад рядка, `'\0'`.

Типи даних, змінні, константи

Тип даних визначає діапазон допустимих значень і простір, що відводиться у пам'яті даних, для змінних, констант та результатів, що повертаються функціями.

Базові типи даних

Основні типи даних у мові С:

`char` - символ;

`int` - ціле число;

`float` - число з плаваючою точкою;

`double` - число з плаваючою точкою подвійної точності;

`void` - змінна, яка не має значень.

На основі цих типів формуються інші типи даних.

Розмір змінних і діапазон їх значень залежить від типу процесора і компілятора. Однак у всіх випадках розмір символу дорівнює 1 байт.

Діапазон зміни змінних типу `float` і `double` залежить від способу представлення чисел з плаваючою точкою.

Стандарт мови С визначає мінімальний діапазон зміни чисел з плаваючою точкою:

від IE - 37 до IE + 37.

Мінімальна кількість цифр, що визначають точність чисел з плаваючою точкою, зазначена у таблиці 4.2.

Таблиця 4.2 - Типи даних, визначені у стандарті ANSI/ISO C Standard

Тип	Звичайний розмір, біт	Мінімальний діапазон
char	8	Від -128 до 127
unsigned char	8	Від 0 до 255
signed char	8	Від -128 до 127
int	16 або 32	Від -32768 до 32767
unsigned int	16 або 32	Від 0 до 65535
signed int	16 або 32	Такий же, як і у int
short int	16	Від -32768 до 32767
unsigned short int	16	Від 0 до 65535
signed short int	16	Такий же, як і у short int
long int	32	Від -2147483648 до 2147483647
signed long int	32	Такий же, як і у long int
unsigned long int	32	Від 0 до 4294967295
float	32	Шість значущих цифр
double	64	Десять значущих цифр
long double	80	Десять значущих цифр

Тип `void` використовується для визначення функції, що не повертає ніяких значень, або для створення узагальненого покажчика (generic pointer).

Таблиця 4.3 - Типи даних мови С, що використовуються у компіляторі CCS-PICC.

Тип	Розмір, у бітах	Діапазон значень
int1	1	0, 1
char signed int8	8	-128 … 127
int8	8	0 … 255
int signed intl6	16	-32768 … 32767
unsigned int intl6	16	0 … 65535
signed int32	32	- 2147483648 … 2147483647
int32	32	0 … 4294967295
float float32	32	$\pm 1{,}175 \cdot 10^{-38} \ldots \pm 3{,}402 \cdot 10^{38}$

Модифікація основних типів

За винятком типу `void`, основні типи даних можуть мати різні модифікатори (modifiers), які використовуються для більш точного налаштування.

Список модифікаторів типів:

- `signed;`
- `unsigned;`
- `long;`
- `short.`

Цілочисельні типи можна модифікувати за допомогою ключових слів `signed, short, long` і `unsigned`. Символьні типи можна уточнювати за допомогою модифікаторів `unsigned` і `signed`. Крім того, тип `double` можна модифікувати ключовим словом `long`. У таблиці 4.2 вказані можливі комбінації типів даних, а також їх мінімальні діапазони і приблизний розмір у бітах. У цій таблиці наведено мінімальні діапазони змінних, а не типові.

Застосування модифікатора `signed` допускається, але є зайвим, оскільки за замовчуванням всі цілі числа мають знак. Найбільш важливий цей модифікатор при уточненні типу `char` в тих реалізаціях мови С, де тип `char` за замовчуванням знаку не має.

Одиниці інформації

Біт - це базова одиниця інформації в обчислювальній техніці, що означає одне з двох станів: 0 або 1, високий рівень сигналу або низький рівень сигналу.

Наприклад, при підключенні світлодіодів до виводів мікроконтролера для кожного такого виводу програміст, оперує поняттям біта, оскільки у даному випадку мова йде про два рівня напруги, один з яких включає, а інший - відключає світлодіод. У загальному випадку, кажуть, що біт містить логічний "0" або логічну "1".

Наступна фундаментальна одиниця інформації - ***байт***, що складається з восьми бітів. З його допомогою можна представити 256 різних станів (0..255), що ілюструє табл. 4.4.

Таблиця 4.4 - Значення, які можна отримати за допомогою одного байта

Значення	**Двійкове представлення**	**Вісімкове представлення**	**Шістнадцяткове представлення**
0	0b00000000	0000	0x00
1	0b00000001	0001	0x01
2	0b00000010	0002	0x02
3	0b00000011	0003	0x03
4	0b00000100	0004	0x04
5	0b00000101	0005	0x05
6	0b00000110	0006	0x06
7	0b00000111	0007	0x07
8	0b00001000	0010	0x08
9	0b00001001	0011	0x09
10	0b00001010	0012	0x0A
11	0b00001011	0013	0x0B
12	0b00001100	0014	0x0C
13	0b00001101	0015	0x0D
14	0b00001110	0016	0x0E
15	0b00001111	0017	0x0F
16	0b00010000	0020	0x10
...	...	...	
254	0b11111110	0376	0xFE
255	0b11111111	0377	0xFF

Користувальницькі типи даних

Мова С дозволяє, крім стандартних, оголошувати власні типи. Для цього використовується ключове слово `typedef`, наприклад:

```
typedef unsigned char byte;              //оголошуємо тип byte
typedef unsigned int word;               //оголошуємо тип word
```

Іншими словами, для оголошення користувальницького типу використовується конструкція вигляду:

```
typedef стандартний_тип ідентіфікатор_користувальницького_типу;
```

Ідентифікатори

У мові С імена змінних, функцій, міток та інших об'єктів, визначених користувачем, називаються ***ідентифікаторами*** (indentifiers). Ідентифікатори можуть складатися з одного або декількох символів. Перший символ ідентифікатора повинен бути буквою або символом підкреслення, а наступні символи повинні бути буквами, цифрами або символами підкреслення. У таблиці 4.5 наведені приклади правильних і неправильних ідентифікаторів.

Таблиця 4.5 - Приклади правильних і неправильних ідентифікаторів

Правильно	**Неправильно**

`Count`
`test23`
`high_balance`
	`1count`
	`hi!there`
	`high...balance`

У мові С довжина ідентифікаторів може бути довільною. Однак не всі символи, що утворюють ідентифікатор, вважаються значущими.

Якщо ідентифікатор використовується в процесі редагування зовнішніх зв'язків, то значущими вважаються, шість його перших символів. Ці ідентифікатори називаються ***зовнішніми іменами*** (external names). До них відносяться імена функцій і глобальних змінних, використовуваних декількома файлами.

Якщо ідентифікатор не використовується в процесі редагування зовнішніх зв'язків, то значущими вважаються 31 перший символ. Такі ідентифікатори називаються ***внутрішніми іменами*** (internal names). До них відносяться, наприклад, імена локальних змінних.

Символи, набрані у верхньому і нижньому регістрі, розрізняються. Отже, `count`, `Count` і `COUNT` - це різні ідентифікатори.

У мові С не можна використовувати ключові слова в якості ідентифікаторів. Крім того, ідентифікатори не повинні збігатися з іменами функцій зі стандартних бібліотек.

Змінні

Змінна (variable) - це іменована величина визначеного типу, яка може змінюватися в ході виконання програми.

Змінна являє собою ім'я комірки пам'яті, яку можна використовувати для зберігання значення, що модифікується. Всі змінні повинні бути оголошені до свого використання.

Для оголошення змінних (тобто виділення для них пам'яті) у програмі на мові С використовується наступна конструкція:

```
тип_змінної список змінних;
```

Тут слово `тип` означає один з допустимих типів даних, включаючи модифікатори, а `список змінних` може складатися з одного або декількох ідентифікаторів, розділених комами.

Розглянемо кілька прикладів оголошень:

```
int i,j,l;
short int si;
unsigned int ui;
double balance, profit, loss;
```

Ім'я змінної ніяк не пов'язано з її типом.

Оголошення змінних

Як правило, змінні оголошують в трьох місцях:

- всередині функцій;
- у визначенні параметрів функції;
- за межами всіх функцій.

Відповідно такі змінні називаються ***локальними***, ***формальними параметрами*** і ***глобальними***.

Локальні змінні

Змінні, оголошені всередині функції, називаються ***локальними*** (local variables). Іноді ці змінні називаються ***автоматичними*** (automatic variables).

Локальні змінні можна використовувати тільки в операторах, розташованих усередині блоку, де вони оголошені. Інакше кажучи, локальні змінні невидимі зовні їх блоку. Блок обмежений відкриваючими і закриваючими фігурними дужками.

Найчастіше локальні змінні оголошуються всередині функцій.

Наприклад:

```
void func (void)
{
        int x;
        x = 10;
}
```

Мова С містить ключове слово `auto`, яке можна використовувати для оголошення локальних змінних. Однак, оскільки всі локальні змінні за замовчуванням вважаються автоматичними, це ключове слово на практиці ніколи не використовується.

З міркувань зручності, оголошують всі змінні, використовувані в функції, відразу після відкривачої фігурної дужки і перед усіма іншими операторами. Однак слід мати на увазі, що локальні змінні можна оголошувати в будь-якому місці блоку.

Оголошення змінних всередині блоків дозволяє уникнути побічних ефектів.

Оскільки локальна змінна поза блоком не існує (до неї неможливо звернутися ззовні, навіть з інших частин функції, що містить даний блок). Її значення неможливо змінити ненавмисно.

Наприклад:

```
void f(void)
{
        int t;
        scanf("%d%*c", &t);
        if(t==1) {
                char s[80]; /* Ця змінна створюється тільки при вході у
                даний блок */
                printf("Введіть ім'я:");
                gets(s);
                /* Деякі операції ... */
        }
        /* Тут змінна s невидима */
}
```

У мові С всі локальні змінні повинні бути оголошені на початку блоку до виконання будь-яких "активних" операторів. Наприклад, спроба оголосити локальні змінні так, як показано в наведеному нижче фрагменті, викличе помилку.

```
void f(void)
{
        int i;
        i = 10;
        int j;                  /* Цей оператор викличе помилку */
        j = 20;
}
```

Формальні параметри

Якщо функція має аргументи, слід оголосити змінні, які будуть приймати їх значення. Ці змінні називаються ***формальними параметрами*** (formal parameters). Усередині функції вони нічим не відрізняються від інших локальних змінних.

Тип формальних параметрів задається при їх оголошенні. Після цього їх можна використовувати як звичайні локальні змінні. Формальні параметри є динамічними і руйнуються при виході з функції.

Формальні параметри можна використовувати в будь-яких конструкціях і виразах. Незважаючи на те, що свої значення вони отримують від аргументів, переданих ззовні функції, в усьому іншому вони нічим не відрізняються від локальних змінних.

Глобальні змінні

На відміну від локальних, ***глобальні змінні*** (global variables) доступні з будь-якої частини програми і можуть бути використані де завгодно. Крім того, вони зберігають свої значення на протягом виконання всієї програми. Оголошення глобальних змінних повинні розташовуватися поза всіх функцій. Ці змінні можна використовувати в будь-якому виразі, в якому б блоці вони не знаходилося.

Якщо локальна і глобальна змінні мають одне і те ж ім'я, то при зверненні до неї всередині блоку, в якому оголошена локальна змінна, відбувається посилання на локальну змінну, а на глобальну змінну це ніяк не впливає.

Глобальні змінні зберігаються в окремій фіксованій області пам'яті, створеною компілятором спеціально для цього. Глобальні змінні використовуються у тих випадках, коли різні функції програми використовують одні і ті ж дані. Однак рекомендується уникати зайвого використання глобальних змінних, тому що вони займають пам'ять протягом усього часу виконання програми, а не тільки тоді, коли вони необхідні. Крім того, використання глобальної змінної робить функцію менш універсальною, тому що в цьому випадку функція використовує щось, визначене поза нею. До того ж велика кількість глобальних змінних легко призводить до помилок в програмі через небажаних побічних ефектів. При збільшенні розміру програми серйозною проблемою стає виконання довільного збільшення значення змінної десь в іншій частині програми, а коли глобальних змінних багато, запобігти цьому дуже важко.

Кваліфікатори const і volatile

У мові С існують два кваліфікатора, керуючих доступом і модифікацією: `const` і `volatile`. Їх слід вказувати перед модифікаторами типів та іменами типів, які вони уточнюють. Формально ці кваліфікатори називають ***cv-кваліфікаторами*** (cv-qualifiers).

Кваліфікатор const

Константа - це іменована величина визначеного типу, яка, на відміну від змінної, не може змінюватися в ході виконання програми, а має конкретне значення, визначене в момент оголошення.

Для оголошення констант в програмі на мові С використовується наступна конструкція:

```
const тип_константи ідентифікатор = значення;
```

Наприклад:

```
const int i = 10;           // Оголошення цілочисельної константи i
const float ADC_10 = 1023;
```

Змінні типу `const` не можуть змінюватися, проте їх можна ініціалізувати. Компілятор може помістити ці змінні у постійний запам'ятовуючий пристрій (Random Access Memory - ROM).

Величина, оголошена як константа, буде розміщена компілятором у пам'яті програм, а не в обмеженій області змінний в ОЗП.

Для доступу до константи використовується її ідентифікатор.

Приклади:

```
c = 'A';              //Помилка! Константа c ще не оголошена
int i;                //Оголошення цілочисельний змінної i
const c = 'A';        //Оголошення константи c
i = 2;                //Змінній i присвоєно значення 2
c = i;                //Помилка! Спроба привласнити значення константі
i = c;                //Змінній i присвоєно значення константи c.
                      //у даному випадку буде виконано автоматичне
                      //приведення типів, тобто i = 65 (ASCII-код символу
                      //'A')
```

Кваліфікатор `const` можна використовувати для захисту об'єктів, переданих у функцію в якості аргументів. Іншими словами, якщо у функцію передається покажчик, вона може модифікувати фактичне значення, на яке вона посилається. Але якщо покажчик уточнити кваліфікатором `const`, функція не зможе змінити значення у відповідній комірці.

Кваліфікатор `const` можна також використовувати для запобігання модифікації змінних. Змінну типу `const` можна модифікувати лише за межами

програми. Наприклад, її значення можна змінити за допомогою апаратних пристроїв. Однак, якщо змінна оголошена константною, можна гарантувати, що її зміна може відбутися лише внаслідок зовнішнього втручання.

Примітка: користувальницький тип, який використовується при оголошенні константи, повинен бути оголошений вище у тексті програми.

У мікроконтролерах PIC використовується роздільна пам'ять для даних (SRAM) і програм (Flash), а також пам'ять типу EEPROM. Змінні, якщо не вказано інше, за замовчуванням розміщуються у пам'яті SRAM, але, зустрічаючи слово `const`, компілятор відносить відповідні значення до постійної флеш-пам'яті.

Перелічувальні типи

Перелічувальний тип - це оголошення списку цілочисельних констант, які можна явно не ініціалізувати (в цьому випадку компілятор вважає, що перша константа в списку приймає значення 0, друга - 1 і т.д.).

Для подібного оголошення використовується ключове слово `enum`:

```
int n;
enum (zero, one, two);              //zero = 0; one = 1; two = 2
n = one;                            //n = 1
```

Якщо потрібно змінити початкове значення для списку констант, то можна вказати його явно при оголошенні, наприклад:

```
enum (three = 3, four, five);       //three = 3; four = 4; five = 5
```

Приведення типів

Приведення типів - це примусове перетворення значення одного типу до іншого, сумісного з вихідним. Це важливо при виконанні арифметичних операцій, коли отримані значення можуть виходити за допустимі межі.

Приведення типів буває ***явним*** і ***неявним***. Неявне приведення типів використовується в операторах присвоювання, коли компілятор сам виконує необхідні перетворення без. участі програміста.

Для явного приведення типу деякої змінної перед нею слід вказати в круглих дужках ім'я нового типу, наприклад:

```
int X;
int Y - 200;
char C = 30;
X = (int)C * 10 + Y;                // Змінна C приведена до типу int
```

Якби в цьому прикладі не було виконано явне приведення типів, то компілятор припустив би, що вираз `C * 10` - це восьмирозрядне множення (розрядності типу `char`) і замість коректного значення `300 (0x12C)` у стек було б вміщено урізане значення `44 (0x2C)`. Таким чином, в результаті обчислення виразу `C * 10 + Y` змінній `X` було б присвоєно значення `640`, а не коректне `3200`. В результаті приведення типу змінна `C` розпізнається компілятором як 16-розрядна і описаної вище помилки не виникає.

Оператор sizeof

Оператор `sizeof` застосовується для обчислення розміру області пам'яті (у байтах), що відводиться під деяку змінну, результат виразу або тип.

Наприклад:

```
int a;
float f;
a = sizeof(int);            //a = 2
a = sizeof(f);              //a = 4
f = 3.3;
a = sizeof(f + a);          //a = 4, оскільки тип результату - float
```

Кваліфікатор volatile

Кваліфікатор `volatile` повідомляє компілятору, що значення змінної може змінюватися неявно.

Функції

Функція являє собою "контейнер", в якому виконується лише певна частина програмного коду. Використання функцій спрощує написання і налагодження програм, оскільки в них зручно розміщувати повторювані групи операторів.

Будь-яка програма на мові С містить головну функцію під назвою `main()`. Ця функція при запуску програми виконується першою.

Користувальницькі функції визначаються після директив препроцесора і глобальних оголошень типів, змінних і констант у файлі з вихідним кодом або у заголовному файлі.

При цьому використовується наступний синтаксис:

```
Тип_значення_що_повертається Ім'я_функції (Список_параметрів)
{
       //тіло функції
}
```

Примітка: попереднє оголошення функції може бути відсутнім. В цьому випадку вона доступна тільки всередині того файлу, в якому визначена.

В добре написаному коді функція `main()` повинна містити, схему роботи всієї програми. Незважаючи на те що ім'я `main()` не включене у список ключових слів, за своєю природою воно є саме таким. Назвати змінну ім'ям `main` не можна, так як компілятор відразу видасть повідомлення про помилку.

Структура програми на мові С показана у лістингу 4.1. Функції з іменами `f1(),…, fN()` визначаються користувачем.

Лістинг 4.1 – Структура програми на мові С

```
//Оголошення глобальних змінних
тип_значення_що_повертається f1(список параметрів)
{
       послідовність операторів
}
тип_значення_що_повертається f2(список параметрів)
{
       послідовність операторів
}

.
.
.
```

```
тип_значення_що_повертається fN(список параметрів)
{
	послідовність операторів
}
тип_значення_що_повертається main(список параметрів)
{
	послідовність операторів
}
```

В якості типу значення, що повертається може використовуватися ключове слово `void`. Це означає, що функція не повертає ніякого значення (в деяких мовах програмування такі функції називають процедурами).

Функцію викликають по її імені із зазначенням в круглих дужках переліку переданих параметрів (якщо їх немає, то в дужках нічого не вказується), наприклад:

```
void Functionl(int n, char c)
{
	…
}
int Function2()
{
	…
}
int main()
{
	int x;
	char y;
	Functionl(x, y);
	x = Function2();
}
```

Параметри - це ідентифікатори, які можуть використовуватися всередині функції. Замість них підставляються відповідні значення, зазначені при виконанні функції (якщо у функцію передається більше одного параметра, то вони відокремлюються один від одного комами як при оголошенні, так і при виклику).

Значення, передані у функцію, фактично не змінюються, а просто копіюються у параметри, які в цьому сенсі виконують роль локальних змінних. При цьому слід стежити за тим, щоб тип переданих значень відповідав типу параметрів, оголошених у заголовку функції.

Значення, що повертаються

Якщо функція призначена для повернення значення деякого типу, то для цього в її тілі використовують ключове слово `return`, після якого (через пробіл) вказують значення, що повертається. При цьому всі оператори після слова `return` ігноруються, і відбувається повернення у функцію, що викликає. приклад:

```
int power3(int n)
{
	return n*n*n;
}

void main()
{
	int x;
	x = power3(2);			//x = 8
}
```

Слово `return` може також використовуватися без зазначення виразу, що повертається. У цьому випадку воно просто означає вихід з функції.

Прототипи функцій

У звичайному варіанті функції використовуються тільки після їх визначення, проте бувають випадки, коли функції викликають одна одну, і організувати їх "правильне" визначення неможливо. Обійти цю проблему дозволяють ***прототипи функцій***, які представляють собою оголошення до визначення. Таке оголошення являє собою тільки заголовок функції, причому у списку параметрів вказують тільки типи, без ідентифікаторів, наприклад:

```
…

int f1(int);
void f2(int, int);
int f1(int x)
{
	…
}
void f2(int a, int b)
{
	…
}
```

Прототипи функцій часто використовуються у заголовних файлах, що включаються у текст програми за допомогою директиви препроцесора `#include`.

Класи пам'яті при оголошенні локальних змінних

Локальні змінні можуть бути оголошені всередині функцій, що належать до одного з трьох класів пам'яті:

`auto` (значення за замовчуванням, можна явно не вказувати) - при оголошенні змінна не ініціалізується ніяким значенням (значення - поточний зміст області пам'яті, відведеної під змінну); при виході з функції змінна видаляється з пам'яті;

`static` - статична змінна доступна тільки у межах функції, хоча пам'ять для неї виділяється у просторі глобальних змінних; при першому зверненні до функції ініціалізується нульовим значенням і після виходу з функції з пам'яті не видаляється (таким чином, при наступних зверненнях до функції в ній міститься старе значення);

`register` - аналог автоматичної локальної змінної за тим винятком, що компілятор спробує виділити для неї не область пам'яті даних, а робочий регістр мікроконтролера, що значно прискорює звернення до значення змінної.

Приклад використання статичної змінної:

```
…
int plus5()
{
        static int x;
        return x + 5;
}
void main()
{
        int y;
        y = plus5();  //y = 5
        y = plus5();  //y = 10
        y = plus5();  //y = 15
}
```

Рекурсія

Рекурсія - це виклик функцією самої себе.

Приклад рекурсивного виклику:

```
void f1(int n)
{
	int x;
	f1(x);
}
```

Рекурсія досить часто використовується у стандартних бібліотечних функціях, а також у багатьох алгоритмах сортування. Проте, при програмуванні мікроконтролерів використання рекурсії, як правило, може спричинити проблеми через обмежений об’єм оперативної пам'яті. Справа у тому, що при кожному виклику рекурсивної функції частина пам'яті витрачається на збереження даних, що поміщають у стек. Ці дані зберігаються там до тих пір, поки не буде виконано повернення з функції. Таким чином, коли рекурсивна функція знову і знову викликає саму себе, у стеку залишається все менше і менше вільної пам'яті.

Це дуже часто призводить до виникнення помилкового стану, званого переповненням стеку. Область стеку зазвичай розміщується у верхній частині пам'яті і росте "вниз", тоді як область змінних розміщується у нижній частині пам'яті і росте "вгору". Тому якщо об’єм даних, які розміщені у стек, перевищить розмір області стеку, ці дані можуть досягти області змінних і затерти собою її значення. При цьому помилку переповнення стеку не завжди легко виявити, оскільки вона може проявлятися лише періодично.

Примітка: подібна проблема може також виникнути при багаторазовому вкладенні функцій, тобто коли функція `f1` викликає функцію `f2`, яка викликає функцію `f3`, яка викликає функцію `f4` і т.д.

Структури

Структура - це особливий тип даних, що складається з декількох різнотипних змінних (полів).

У загальному випадку оголошення структури має такий вигляд:

```
struct ім'я_структури {
	тип поле_1;
	...
	тип поле_N;
};
```

Як і будь-який інший тип, структуру можна в подальшому використовувати для оголошення змінних, наприклад:

```
struct MyStructure{          //Оголошення структури MyStructure
       int Fieldl;
       char Field2;
       float Field3;
};

//Оголошення змінних YourStruct і OurStruct типу MyStructure
struct MyStructure YourStruct, OurStruct;
```

Крім того, можна створювати змінні безпосередньо при оголошенні структури:

```
struct MyStructure{          //Оголошення структури MyStructure
       int Fieldl;
       char Field2;
       float Field3;
} YourStruct, OurStruct;
```

Допускається ініціалізація полів безпосередньо при оголошенні змінних структур за допомогою переліку значень у фігурних дужках, наприклад:

```
struct DATE {
       int Day;
       int Month;
       int Year;
} struct DATE MyBirthday = {1, 9, 2000};
```

Другий варіант цієї ж ініціалізації:

```
struct DATE {
       int Day;
       int Month;
       int Year;
} MyBirthday = {1, 9, 2000};
```

Багато компіляторів дозволяють визначати при оголошенні структури так звані бітові поля шириною від одного до 32 біт. Вони служать для економії пам'яті мікроконтролера і визначаються за допомогою символу. наприклад:

```
struct s1 {
       unsigned long A:10;
       unsigned long B:8;
       unsigned long C:11;
};
```

У цьому випадку структура `s1` займе в пам'яті чотири байти, в яких молодші 10 розрядів будуть відведені під поле `A`, наступні вісім - під поле `B`,

і 11 - під поле c. Старші три розряди області, займаною структурою, у такому випадку не використовуються.

Для доступу до полів структури у програмі є запис вигляду ім'я_структури.поле. Так, у представленому вище прикладі структури DATE для ініціалізації полів можна було скористатися наступними операторами:

```
MyBirthday.Day = 1;
MyBirthday.Month = 9;
MyBirthday.Year = 2000;
```

Структури, в свою чергу, можуть бути полями інших структур, наприклад:

```
struct ME {
       char MyName[30];              //Рядок довжиною 30 символів - ім'я
       struct DATE MyBirthday;       //Структура, що зберігає день
                                     //народження
}
…
struct ME MyData;
MyData.MyName = "John Smith";
MyData.MyBirthday.Day =1;
MyData.MyBirthday.Month = 9;
MyData.MyBirthday.Year = 2000;
```

Структури можуть виступати в якості параметрів функцій, а також результату, що повертається. Нижче представлений приклад функції, яка повертає структуру типу DATE:

```
struct DATE Januaryl(int CurYear)
{
       struct DATE Jan0l;
       Jan0l.Day = 1;
       Jan0l.Month = 1;
       Jan0l.Year = CurYear;
       return Jan0l;
}
…
struct DATE BeginOfTheYear;
BeginOfTheYear = Januaryl(2000);
```

Покажчики та адреси змінних

Покажчик - це змінна, яка містить адресу деякого елементу даних (змінної, константи, функції, структури). Для оголошення змінної як покажчика використовується оператор *:

```
int *p;              //p - покажчик на ціле число
```

Для присвоєння адреси деякої змінної вказівником використовується оператор &, наприклад:

```
char *p;              //покажчик на символ
char c;               //символьна змінна
c = 'A';
p = &c;               //p містить адресу змінної c (вказує на 'A')
```

Для того щоб вилучити значення змінної, на яку вказує покажчик, використовується оператор розіменування `*`.

```
char *p;
char c, b;
c = 'A';
p = &c;
b = *p;               //тепер b = 'A'
```

Аналогічним чином, цей оператор можна використовувати і для присвоєння деякого значення змінної, на яку вказує покажчик: `char *p`;

```
char c, b;
b = 'A';
p = &c;
*p = b;               //тепер c = 'A'
```

Передача в функції параметрів по посиланню

За допомогою покажчиків, використовуваних в якості параметрів функцій, можна організувати повернення більше одного значення. приклад:

```
int SumAndDiv(int *a, int *b)
{
       int bufA, bufB;
       bufA = *a;
       bufB = *b;
       *a = bufA / bufB;            //Покажчик посилається на результат
                                    //цілочисельного ділення без залишку
       *b = bufA % bufB;            //Покажчик посилається на залишок від
                                    //цілочисельного ділення
       return bufA + bufB;          //Функція повертає суму
}

int v1, v2, sum;
v1 = 10;
v2 = 3;
sum = SumAndDiv(&v1, &v2);          //sum = 13; v1 = 3; v2 = 1
```

У функції `SumAndDiv()` спочатку зберігаються у буферних змінних значення, на які вказують покажчики `a` і `b`. Потім у змінну, на яку вказує покажчик `a` записується результат ділення переданих у функцію значень без залишку, а в змінну, на яку вказує покажчик `b` - залишок від такого поділу.

Оскільки за допомогою покажчиків відбувається звернення до комірок пам'яті, а не до змінних, то після виходу з функції вміст цих комірок залишається незмінним.

При виконанні функції `SumAndDiv()` в неї передаються адреси змінних `v1` і `v2`, яким у тілі функції відповідають покажчики `a` і `b`.

Покажчики на структури

На структури можна створювати покажчики точно так же, як і для будь-якого іншого типу даних.

Приклад застосування такого покажчика:

```
struct DATE {
       int Day;
       int Month;
       int Year;
}
…
struct DATE MyBirthday, *dateP;
dateP = &MyBirthday;         //dateP вказує на структуру MyBirthday
```

У такому випадку для доступу до полів структури через покажчик використовується оператор `->`:

```
dateP->Day =1;
dateP->Month = 9;
dateP->Year = 2000;
```

Можна також використовувати і розіменування покажчика:

```
(*dateP).Day =1;
(*dateP).Month = 9;
(*dateP).Year = 2000;
```

Примітка: *в* останньому прикладі операція `*dateP` міститься в дужках, оскільки оператор розіменування `*` має менший пріоритет у порівнянні з оператором доступу до поля структури.

Покажчики також можуть бути полями структури. Це часто використовується для створення у пам'яті ланцюжків однотипних структур, коли кожна попередня вказує на наступну:

```
struct DATE {
       int Day;
       int Month;
       int Year;
       struct DATE *next_date;      //покажчик на іншу структуру
       //того ж типу
}
…
struct DATE datel, date2;
datel.next_date = &date2;
```

Тепер, наприклад, оператору `datel.next_date-> Year` відповідає доступ до поля `Year` структури `date2` (тобто це еквівалентно запису `date2.Year`).

Масиви і рядки

Масив - це тип даних, який використовується для представлення послідовності однотипних значень.

Масиви оголошуються подібно звичайним змінним, із зазначенням у квадратних дужках розмірності (кількості елементів у масиві):

```
int digits[10];             //Масив з десяти елементів типу
int char str[10];           //Масив з десяти символів (рядок)
```

Доступ до елементів масиву реалізується за допомогою індексу (порядкового номера елемента, починаючи з 0):

```
digits[0] = 0;
digits[1] = 1;
str[0] = 'A';
str[l] = 'B';
```

Примітка: ім'я масиву - це покажчик на його перший елемент. Так в останньому прикладі, оператор `str[0] = 'A';` можна було б замінити оператором `*str - 'A';`, а оператор `str[l] = 'B';` оператором `*(str + 1) = 'B';`.

Можна ініціалізувати масив безпосередньо при його оголошенні, наприклад:

```
int digits[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
char str[10] = {'T', 'h', 'e', ' ', 'l', 'i', 'n', 'e'}
int n;
char c;
n = digits[2];                  //n = 2
c = str[1];                     //c = 'h'
```

При розробці програм слід враховувати, що компілятор не завжди може відстежити вихід індексу за межі масиву.

Рядки

Рядок у мові С - це масив типу `char`.

Приклад: оголошення масиву типу `char`, відповідного рядку `"The line"`, можна виконати за допомогою строкового літерала:

```
char str[10] = "The line";
```

Однак в такому випадку слід пам'ятати, що компілятор неявно завершує рядкові літерали символом `'\0'`, і, таким чином, реальна довжина рядка більше на одиницю. Це слід враховувати, щоб при ініціалізації масиву не вийти за його межі.

При ініціалізації рядків розмірність масиву можна явно не вказувати:

```
char str[] = "The line";
```

В цьому випадку компілятор визначить розмірність самостійно (це правило може бути застосовано і до ініціалізації будь-яких інших масивів).

Примітка: для того щоб рядкові константи розміщувалися не в пам'яті SRAM, а у флеш-пам'яті мікроконтролера, їх слід оголошувати з використанням слова `const`:

```
const char str[];
```

Багатовимірні масиви

Мова С допускає використання багатовимірних масивів, тобто масивів, елементами яких є масиви.

Приклади оголошення таких масивів:

```
int a2[10][2];              //Двомірний масив 10x2
int a3[3][2][5];            //Тривимірний масив 3x2x5
```

Фактично, всі елементи багатовимірного масиву зберігаються у пам'яті послідовно, тому представлені нижче приклади ініціалізації аналогічні:

```
int a[2][3] = {{1, 2, 3}, {4, 5, 6}};
```

і

```
int a[2][3] = {1, 2, 3, 4, 5, 6};
```

Для доступу до елементів багатовимірного масиву використовуються індекси по кожній з розмірностей або операції розіменування покажчика.

Наприклад, щоб вилучити в деяку змінну `n` цифру `4` з представленого вище масиву `a`, можна скористатися одним з двох варіантів:

```
n = a[1][0];          //вилучаємо елемент з "рядку" 1 (другa),
                      //"стовпця" 0 (перший), тобто - цифру 4
n = *(a + 3);         //a - покажчик на перший елемент масиву,
                      //отже, a + 3 - покажчик на 4-й елемент
```

ОПЕРАТОРИ

Оператор - це символ, який вказує компілятору, які дії виконати над операндами. Деякі символи можуть трактуватися по-різному в залежності від контексту. Наприклад, знак "-" може використовуватися для зміни знаку числа або в якості оператора віднімання.

Оператори, що з'єднують операнди, являють собою ***вирази***. Вирази можуть бути укладені в круглі дужки і відокремлюються один від одного символом крапки з комою (";").

Об'єднані за певною ознакою послідовності виразів поміщені у фігурні дужки {}. Так позначаються межі функцій, а також блоки виразів у циклічних і умовних конструкціях.

Оператори поділяються на чотири основні групи:

- арифметичні (arithmetic);
- порівняння (relational);
- логічні (logical);
- побітові (bitwise).

Пріоритетність виконання операторів в виразах мови С вказана у табл. 4.6 (пріоритет з меншим номером рівня - вище).

Таблиця 4.6. Пріоритетність виконання операторів в виразах мови С

Рівень	**Оператори**	**Категорія**	**Опис**
1	`( )`		Круглі дужки
	`[ ]`		Елемент масиву
	`.`	Доступ до даних	Звернення до елементу структури, наприклад: `PORTB 1` - розряд `1` порту `B`
	->		до елементу структури, визначеної покажчиком, наприклад: `pStruct -> x` - елемент `x` структури, на яку вказує `pStruct`
	`++, --` (постфікси)	Арифметичні	Оператори автоінкремента і автодекремента після того як вираз, в якому задіяні відповідні операнди, обчислено. *Приклади:* `a = b++;` рівнозначно `a = b; b = b+1;` `a = b--;` рівнозначно `a = b; b = b-1;`
2	`++, --` (префікси)		Оператори автоінкремента і автодекремента перед тим як вираз, в якому задіяні відповідні операнди, буде обчислено. *Приклади:* `a = ++b;` рівнозначно `b = b+1; a = b;` `a = --b;` рівнозначно `b = b-1; a = b;`
	!	Логічні	Логічне (унарне) заперечення

	~	Порозрядні	Порозрядне заперечення
	&	Доступ до даних	Адреса
3	+, - (унарні)	Арифметичні	Зміна знака операнда
	* (унарний)	Доступ до даних	Розіменування покажчика
4	* (бінарний)	Арифметичні	Множення
	/		Ділення
	%		Залишок від ділення
5	+, - (бінарні)		Додавання і віднімання
6	<<	Порозрядні	Порозрядний зсув вліво
	>>		Порозрядний зсув вправо
7	<, >, <=, >=	Порівняння	Менше, більше і т.д.
8	==, ! =		Дорівнює, не дорівнює
9	&	Порозрядні	Порозрядне "І"
10	^	Порозрядні	Порозрядне "АБО, що виключає"
11	|	Порозрядні	Порозрядне "АБО"
12	&&	Логічні	Логічне "І"
13	||	Логічні	Логічне "АБО"
14	=	Присвоєння	Можливі поєднання оператора присвоювання з арифметичними і порозрядного операторами: `a + = b;` рівнозначно `a = a + b;`

Оператор присвоювання

Оператор присвоювання можна використовувати в будь-якому коректному виразі.

Загальний вигляд оператора присвоювання виглядає наступним чином:

```
ім'я змінної = вираз;
```

Вираз може складатися як з окремої константи, так і комбінації складних операторів.

В якості оператора присвоєння у мові С використовується знак рівності.

Мета (target), або ліва частина оператора присвоєння, повинна бути або змінною, або покажчиком, але не функцією або константою.

Перетворення типів в операторі присвоєння

Таблиця 4.7 – Правила перетворення типів

Результуючий тип	**Тип виразу**	**Можливі втрати**
`signed char`	`char`	Якщо значення `> 125`, результатом буде негативне число
`char`	`short int`	Старші 8 біт
`char`	`int (16 бит)`	Старші 8 біт
`char`	`int (32 бит)`	Старші 24 біт
`char`	`long int`	Старші 24 біт
`short int`	`int (16 бит)`	Немає
`short int`	`int (32 бит)`	Старші 16 біт
`int (16 Сит)`	`long int`	Старші 16 біт
`int (32 бит)`	`long int`	Немає
`int`	`float`	Дрібна частина і, можливо, щось ще
`float`	`double`	Точність, результат округляється
`double`	`long double`	Точність, результат округляється

Коли у виразі змішуються змінні різних типів, необхідно виконати ***перетворення типів*** (type conversion).

Правило перетворення типів для оператора присвоювання: значення правої частини (rvalue) перетворюється до типу лівої частини (*lvalue*).

Термін ***lvalue*** означає будь-який об'єкт, який може стояти в лівій частині оператора присвоювання. З практичної точки зору термін *lvalue* відноситься до змінних.

Терміном ***rvalue*** називають вираз, що стоїть в правій частині оператора присвоювання, точніше кажучи, значення цього виразу.

Множинні присвоювання

У мові С дозволяється присвоювати одне і те ж значення декільком змінним одночасно.

Наприклад:

```
x = y = z = 0;
```

Арифметичні оператори

У табл. 4.8 наведено список арифметичних операторів мови С

Таблиця 4.8 - Арифметичні оператори

Оператор	Дія
-	Віднімання, а також унарний мінус
+	Додавання
*	Множення
/	Ділення
%	Ділення по модулю
--	Декрементація
++	Інкрементація

Оператор ділення по модулю (%) повертає залишок цілочисельного ділення. Однак цей оператор не можна застосовувати до чисел з плаваючою крапкою.

Оператор ++ (інкрементації) додає 1 до свого операнду, а ***оператор*** -- (декрементаціі) віднімає її.

Таким чином, оператор

```
x = x + 1;
```

еквівалентний оператору

```
++x;
```

а оператор

```
x = x - 1;
```

еквівалентний оператору

```
x--;
```

Пріоритети арифметичних операторів:

Вищий	++ --
	- (унарний мінус)
	* / %
Нижчий	+ -

Оператори, що мають однаковий пріоритет, виконуються зліва направо. Для зміни порядку обчислення операторів можна застосовувати дужки.

Оператори порівняння і логічні оператори

У терміні ***оператор порівняння*** слово "порівняння" відноситься до значень операндів.

У табл. 4.9 наведено список операторів порівняння.

Таблиця 4.9 - Оператори порівняння

Оператор	Дія
>	Більше
>=	Більше або дорівнює
<	Менше
<=	Менше або дорівнює
!=	Не дорівнює

У терміні ***логічний оператор*** слово "логічний" відноситься до способу, яким встановлюються ці відносини.

У табл. 4.10 наведено список логічних операторів.

Таблиця 4.10 - Логічні оператори

Оператор	***Дія***
&&	І
||	АБО
!	НЕ

В основі операторів порівняння і логічних операторів лежать значення **"True"** і **"False"**. У мові С істинним вважається будь-яке значення, не рівне нулю. Хибне значення завжди дорівнює 0. Вирази, що використовують оператори порівняння і логічні оператори, повертають 0, якщо результат хибний, і 1, якщо результат істинний.

Побітові оператори

Побітові операції (bitwise operation), призначені для перевірки, установки і зсуву бітів, з яких складаються байти і машинні слова, що утворюють змінні типу `char` або `int`.

Побітові операції не можна застосовувати до змінних `float`, `double`, `long double`, `void`, `bool` і інших, більш складних типів.

Побітові оператори мови С перелічені у табл. 4.11. Ці оператори застосовуються до окремих бітів операндів.

Таблиця 4.11 - Побітові оператори

Оператор	**Дія**
&	І
|	АБО
^	виключаюче АБО
~	НЕ (заперечення, доповнення до одиниці)
>>	Зсув вправо
<<	Зсув вліво

Оператори розгалуження

Оператор розгалуження - це частина програми, яку можна виконати окремо. Це означає, оператор визначає якусь дію. Оператори мови С поділяються на такі категорії:

- умовні оператори;
- оператори циклу;
- оператори переходу;
- мітки;
- оператори виразу;
- блоки.

Оператори розгалуження використовуються для виконання того чи іншого блоку в залежності від деяких умов.

До таких аператорам у мові С відносяться ***умовні*** (conditional) оператори `if-else` і `switch-case`. (Умовні оператори іноді називають ***операторами розгалуження*** (selection statement)).

Оператори циклу (iteration statements) позначаються ключовими словами `while`, `for` і `do while`. Група операторів переходу складається з операторів `break`, `continue`, `goto` і `return`. Мітками служать оператори `case`, `default`.

Оператори-вирази - це оператори, що складаються з допустимих виразів. Блок являє собою фрагмент тексту програми, укладений у фігурні дужки. Іноді блоки називають складовими операторами (compound statements).

Вирази

Вирази складаються з операторів, констант, функцій і змінних. У мові С виразом вважається будь-яка допустима комбінація цих елементів.

Логічні значення `True` та `False` у мові С

При виконанні багатьох операторів мови С обчислюються значення умовних виразів і в залежності від отриманого значення вибирається та чи інша гілка обчислювального процесу. Умовний вираз може приймати одне з двох значень: `True` або `False`.

У мові С значення `True` представлено будь-яким ненульовим значенням, включаючи негативні числа. Значення `False` завжди представлено нулем.

Умовні оператори

У мові С передбачені два умовних оператора: `if` і `switch`. У деяких ситуаціях в якості альтернативи умовного оператору `if` можна застосовувати тернарний оператор "`?`".

Оператор if

Оператор `if` має наступний вигляд:

```
if (вираз) оператор;
else оператор;
```

Тут `оператор` може складатися з одного або декількох операторів або бути відсутнім зовсім (порожній оператор). Розділ `else` є необов'язковим.

Якщо вираз ***істинний*** (тобто не дорівнює нулю), виконується оператор або блок, зазначений у розділі `if`, в іншому випадку виконується оператор або блок, передбачений у розділі `else`. Оператори, вказані у розділах `if` або `else`, є взаємовиключними.

У мові С результатом умовного виразу є ***скаляр***, тобто ціле число, символ, покажчик або число з плаваючою крапкою.

Вкладені оператори if

Вкладеним (nested) називається оператор `if`, який знаходиться всередині іншого оператора `if` або `else`. У вкладеному умовному операторі розділ `else` завжди пов'язаний з найближчим оператором `if`, який знаходиться з ним в одному блоці і не пов'язаним з іншим оператором `else`.

```
if (i)
{
        if(j) оператор1;
        if(k) оператор2;              // даний if
        else оператор3;               // зв'язаний з даним оператором else
}
else оператор 4;                      // зв'язаний з оператором if(i)
```

Ланцюжок операторів if-then-else

У програмах па мові С часто використовується конструкція, яка називається ***ланцюжок*** `if-then-else` (if-then-alse ladder), або ***драбина*** `if-then-else` (if-then-else staircase). Загальний вигляд цієї конструкції виглядає так.

```
if(вираз)оператор;
else
        if(вираз)оператор;
        else
                if(вираз)оператор;
                …
                else оператор;
```

Ці умови обчислюються зверху вниз. Як тільки значення умовного виразу стає істинним, виконується пов'язаний з ним оператор, і решта конструкції ігнорується. Якщо всї умовні вирази виявилися хибними, виконується оператор, вказаний в останньому розділі `else`. Якщо цього розділу немає, то не виконується жоден оператор.

Оскільки в міру збільшення глибини вкладеності кількість відступів в рядку зростає, сходи `if-then-else` часто записують інакше

```
if(вираз)
        оператор;
else if(вираз)
        оператор;
```

```
else if(вираз)
        оператор;
…
else
        оператор;
```

Оператор switch

У мові С передбачений оператор ***багатоваріантного розгалуження*** `switch`, який послідовно порівнює значення виразу зі списком цілих чисел або символьних констант. Якщо буде виявлений збіг, виконується оператор, пов'язаний з відповідною константою.

Оператор `switch` має наступний вигляд:

```
switch(вираз)
{
        case константа1:
                послідовність операторів
                break;
        case константа2:
                послідовність операторів
                break;
        case константа3:
                послідовність операторів
                break;
        .
        .
        .
        default:
                послідовність операторів
}
```

Значенням *виразу* повинен бути символ або ціле число. Наприклад, вирази, результатом яких є число з плаваючою точкою, не допускаються. Значення виразу послідовно порівнюється з константами, зазначеними в операторах `case`.

Якщо буде виявлено збіг, виконується послідовність операторів, пов'язаних з даним оператором `case`, поки не зустрінеться оператор `break` або не буде досягнутий кінець оператора `switch`.

Якщо значення виразу не збігається ні з однією з констант, виконується оператор `default`. Цей розділ оператора `switch` є необов'язковим.

Якщо він не передбачений, за відсутності збігу не буде виконаний жоден оператор.

У мові С оператор `switch` допускає до 257 операторів `case`. На практиці кількість розділів `case` в операторі `switch` слід обмежувати, оскільки це впливає на ефективність програми. Незважаючи на те що оператор `case` є міткою, він використовується тільки всередині оператора `switch`.

Оператор `break` відноситься до групи операторів переходу. Його можна використовувати як в операторі `switch`, так і у циклах. Коли потік управління досягає оператора `break`, програма виконує перехід до оператора, що слідує за оператором `switch`.

Властивості оператора `switch`:

- оператор `switch` відрізняється від оператора `if` тим, що значення його виразу порівнюється виключно з константами, в той час як в операторі `if` можна виконувати будь-які порівняння або обчислювати будь-які логічні вирази;
- Дві константи в різних розділах `case` не можуть мати однакових значень, за винятком випадку, коли один оператор `switch` вкладений в інший.
- Якщо в операторі `switch` використовуються символьні константи, вони автоматично перетворюються в цілочисельні.

Оператори циклу

У мові С, як і в інших мовах програмування, оператори циклу служать для багаторазового виконання послідовності операторів до тих пір, поки виконується деяка умова. Умова може бути встановленою заздалегідь (як в операторі `for`) або змінюватися при виконанні тіла циклу (як у `while` або `do-while`).

Цикл for

Загальна форма оператора for наступна:

```
for(ініціалізація; умова; приріст) оператор;
```

Ініціалізація (initialization) - це присвоювання початкового значення змінної, яка називається параметром циклу.

Умова (condition) являє собою умовний вираз, що визначає, чи слід виконувати оператор циклу (його часто називають тілом циклу) в черговий раз.

Оператор *приріст* (increment) здійснює зміни параметра циклу при кожній ітерації.

Ці три оператора (вони називаються також *секціями оператора* `for`) обов'язково розділяються крапкою з комою.

Цикл `for` виконується, якщо вираз умова приймає значення `True`. Якщо вираз умова хоча б один раз прийме значення `False`, то програма виходить з циклу і виконується оператор, що слідує за тілом циклу `for`.

Оператор `for` може бути ***порожнім***. Це означає, що тіло циклу `for` (як і будь-якого іншого циклу) може не містити жодного оператора.

Безкінечний цикл

Хоча в якості нескінченного можна використовувати будь-який цикл, традиційно для цієї мети застосовується оператор `for`. Оскільки всі розділи оператора `for` є необов'язковими, його легко зробити нескінченним, не поставивши жодного умовного виразу.

```
for(;;) блок_операторів
printf("Цей цикл виконується нескінченно.\n");
```

Якщо умовний вираз не вказаний, він вважається істинним.

У випадку циклів `while` це буде виглядати наступним чином:

```
while(1) блок_операторів
```

Цикл while

Оператор `while` має такий вигляд.

```
while (умова) оператор;
```

Оператор може бути порожнім, окремим оператором або блоком операторів.

Умова може задаватися будь-яким виразом. Умова циклу вважається істиною, якщо значення цього виразу не дорівнює нулю. Як тільки умова циклу стає хибною, програма передає управління оператору, що слідує відразу після оператора `while`.

Як і цикл `for`, цикл `while` перевіряє свою умову перед початком виконання тіла. Отже, якщо умова циклу з самого початку є помилковою, його тіло не буде виконано жодного разу. Завдяки цій властивості немає необхідності окремо перевіряти умову циклу перед входом в нього.

Цикл do-while

На відміну від операторів `for` і `while`, які перевіряють умову на початку циклу, оператор `do-while` робить це в кінці. Іншими словами, цикл `do-while` виконується принаймні один раз.

Загальний вигляд оператора `do-while` такий:

```
do{
        оператор;
}while(умова);
```

Цикл `do-while` повторюється до тих пір, поки умова не стане хибною.

Оператори break і continue

Якщо в тілі будь-якого циклу зустрічається оператор `break`, то управління тут же передається на оператор, що слідує за оператором циклу, незалежно від істинності або неістинності умовного виразу. При цьому у вкладених циклах вихід здійснюється не на самий верхній рівень вкладеності, а лише на один рівень вгору.

При виконанні оператора `continue` всі оператори блоку, що знаходяться після нього, пропускаються, а управління передається в початок циклу для наступної ітерації. На практиці оператори `continue` використовуються набагато рідше, ніж оператори `break`, проте в складних циклах, що вимагають прийняття рішень на підставі багатьох чинників, використання операторів `continue` може бути вельми зручним.

СТАНДАРТНІ ФУНКЦІЇ ВВЕДЕННЯ/ВИВЕДЕННЯ

Стандартні функції введення/виведення дозволяють обмінюватися інформацією з виконуваною програмою, що, наприклад, дуже зручно в процесі налагодження. Всі вони побудовані на основі функцій посимвольного введення/виведення, які легко пристосувати до апаратних вимог системи.

Якщо при звичайному застосуванні мови С ці функції використовуються для введення даних з клавіатури і виведення на екран або принтер, то стосовно до мікроконтролерів PIC мова йде про обмін даними через приймач UART/USART (за замовчуванням) або послідовний порт. Таким чином, перш, ніж почати введення/виведення в програмі повинні бути виконані відповідні налаштування (наприклад, у компіляторі CCS-PICC для настройки частоти системної синхронізації і параметрів обміну даними через UART використовується директива препроцесора `#use`).

Введення/виведення символів за допомогою функцій getchar() і putchar()

Найпростішими функціями введення/виведення у мові С є `getchar()` і `putchar()`, які призначені для посимвольного обміну даними (обидві оголошені у стандартному заголовному файлі `stdio.h`).

Функція `getchar()` повертає символ, прийнятий від UART/USART або по послідовному інтерфейсу, а функція `putchar()`, навпаки, виводить символ.

Всі інші стандартні функції введення/виведення базуються на них.

Примітка:

Функції `getchar()` і `putchar()` у компіляторах для мікроконтролерів реалізують очікування готовності саме приймача UART/USART для передачі/ прийому символу. У тому випадку, якщо потрібно організувати обмін даними по іншому інтерфейсу (наприклад, по SPI), необхідно розробити власні функції аналогічного змісту.

Функції виведення рядків puts() і printf()

Функції `puts()` і `printf()` використовують функцію `putchar()` для виведення посимвольно рядку у порт RS-232, призначеним останнім. Наприклад, у нас є визначення рядка, призначеного для виведення через послідовний інтерфейс:

```
char s[6] = "12345";
```

Виклик функції `puts()` для виведення цього рядка виглядає просто як `puts(s);`. Функція `printf()` дещо складніше, оскільки дозволяє застосувати форматоване виведення.

Вона має наступний синтаксис виклику:

```
printf("Рядок, в який підставляються змінні",
       змінна1, змінна2, ....);
```

У рядку, що виводиться зазвичай використовуються специфікації форматування значень змінних, переданих в якості параметрів.

Кожна така специфікація починається зі знака після якого йдуть позначення параметрів форматування:

`c` - виведення параметра у вигляді символу ASCII;

`d` - виведення параметра у вигляді цілого числа;

`e` - виведення параметра в експоненційному форматі;

`f` - виведення параметра у вигляді дійсного числа;

`ld` - виведення параметра у вигляді довгого цілого зі знаком;

`lu` - виведення параметра у вигляді беззнакового довгого цілого;

`Lx` - виведення параметра у вигляді беззнакового довгого цілого у шістнадцятковому представленні, використовуючи нижній регістр символів;

`LX` - виведення параметра у вигляді беззнакового довгого цілого у шістнадцятковому представленні, використовуючи великі букви символів;

`S` - виведення параметра у вигляді рядка;

`U` - виведення параметра у вигляді беззнакового цілого;

`x` - виведення параметра у вигляді беззнакового цілого у шістнадцятковому представленні, використовуючи нижній регістр символів;

`X` - виведення параметра у вигляді беззнакового цілого у шістнадцятковому представленні, використовуючи великі букви символів;

`%` - виведення знака відсотка.

Кількість специфікацій має збігатися з кількістю змінних. При цьому специфікація, що зустрілася першою, відповідає першій змінній у списку, друга - другий і т.д.

У випадку виведення числових значень відразу ж після знака "%" може бути вказано кількість символів і формат для відображення чисел, наприклад:

`8` - вісім знаків;

`08` - вісім знаків з доповненням ведучими нулями;

`8.2` - вісім знаків, два знака після десяткового дробу.

Приклади використання специфікацій форматування:

```
int n = 123;
char c = '$';
float f = 23.5;
char str[] = "Hello";
printf("n = %d, str = %s", n, str);              //n = 123,str = Hello
printf("n = %d : 0x%04X", n, n);                 //n = 123:0x007B
printf("Salary = %c%8.2f", c, f);                //Salary = $23.50
```

Функції введення рядків gets() і scanf()

Функції `gets()` і `scanf()` виконують дії, зворотні функціям `puts()` і `printf()`: зчитують посимвольний рядок через призначений останнім послідовний інтерфейс.

Функція `scanf()` зчитує з вхідного потоку значення у форматі, визначеному у першому параметрі і зберігає їх у змінних адреси яких передані їй в якості наступних параметрів. Специфікації форматування для цієї функції такі ж, як і для `printf()`.

Зазвичай функція `scanf()` використовується у парі з функцією `printf()`:

```
int x, y;
puts("Enter two integers:");
scanf("%d, %x\n", &x, &y);          // Зчитує два числа, введені
                                    // через кому: одне - у десятковій,
                                    // а інше - у шістнадцятковій формі
printf ( "x =% d, y =% 04X", x, y);
```

Примітка: у середовищі CCS-PICC також реалізована функція `get_string()`, якою в якості першого параметра передається покажчик на символьний масив (буфер для зчитування рядка), а в якості другого - кількість зчитувальних символів (довжина рядка).

Директиви препроцесора

Директиви препроцесора, по суті, не є складовою частиною мови С, а використовуються для підстановки коду у текст програми.

Препроцесор - це особлива програма, яка виконує попередню обробку даних до початку компіляції.

Розглянемо стандартні директиви препроцесора, а також директиви, характерні для різних компіляторів.

Директива #include

Директива `#include`, використовується практично у всіх програмах для включення в текст програми заголовних файлів (з визначеннями), що має розширення `.h`, або файлів `.c` (що не містять функцію `main()`).

Ця директива може використовуватися у двох формах:

```
#include <ім'я_файлу>
```

або

```
#include "ім'я_файлу"
```

Якщо ім'я файлу поміщено між знаками "<" і ">", то він вважається частиною стандартної бібліотеки, що поставляється разом з компілятором.

Якщо ж ім'я файлу укладено в подвійні лапки (" "), то вважається, що він розташований в тій же папці, що і файл з вихідним кодом.

Директива #define

Директива `#define` вказує препроцесору на необхідність виконати підстановку у тексті програми визначеної послідовності символів іншою послідовністю.

Формат директиви:

```
#define замінювана_послідовність фрагмент_підстановки
```

Наприклад:

```
#define MyName "John Smith"
#define Condition (a > b)
#define Operation a = b

…
char str[] = MyName;         //рівнозначно char str[] = "John Smith";
int a, b;
if Condition Operatrion;     //рівнозначно if(a > b) a = b;
```

У директиві `#define` можуть використовуватися параметри, завдяки чому вона стає дуже потужним і гнучким інструментом, що дозволяє замінювати один простий текстовий елемент складним виразом. Такі підстановки називають макросами.

Наприклад, вираз, в якому вибирається більше з двох значень, можна представити у вигляді наступного макросу:

```
#define larger(x, y)((x)>(y) ? (x) : (y))
```

Якщо визначений такий макрос, то код, який використовує його, може мати такий вигляд:

```
int a = 9;
int b = 7;
int c = 0;
c = larger(a, b);
```

Нагадує виклик функції, однак це не функція - компілятор отримає від препроцесора останній рядок в наступному вигляді:

```
c = ((a)>(b) ? (a) : (b));
```

Основна відмінність макросу від функції полягає в тому, що код макропідстановки вставляється препроцесором у програмний код всюди, де зустрічається заданий текстовий елемент, код ж функції визначається тільки в одному місці, а в тих місцях, де вказано її ім'я, здійснюється виклик цього коду.

Є і ще одна відмінність, особливо важлива при програмуванні мікроконтролерів. При використанні макросів, на відміну від функцій, нічого не поміщається у стек, що дозволяє заощадити оперативну пам'ять.

Крім того, макроси перетворюються у звичайний код, який виконується швидше, ніж код функції, на виклик якого і повернення управління у функцію, що викликається, йде додатковий машинний час. При використанні макросів не потрібно формального оголошення типів даних.

Перерахуємо деякі правила використання директиви `#define`:

- при створенні коментарів у рядку з `#define` завжди використовується коментар вигляду `/* ... */`;
- кінець рядка - це кінець `#define`, і весь текст зліва замінить текст праворуч;
- для перетворення параметра макросу у текстовий рядок можна вказати перед ним символ "`#`" наприклад:

```
#define OutString(s) puts(#s)
OutString(Line);                        //Рівнозначно puts ("Line");
```

- для конкатенації двох параметрів можна скористатися оператором `##`, наприклад:

```
#define Concat (x, y) x ## y
int i = Concat (2, l);                  // Рівнозначно int i = 21;
```

- для перенесення тексту підстановки на інший рядок використовується символ зворотної косої "\" наприклад:

```
#define LongStr "012345678910 \
11 12 13 14 15 16 17 18 19 20"
```

- для скасування визначення використовується директива `#undef`, наприклад:

```
#define A_Char'A'

...

#undef A_Char
#define A_Char'a'
```

Директиви умовної компіляції

Багато мікроконтролерів відрізняються лише деякими параметрами, кількістю виводів, розміром пам'яті і розміщенням регістрів. Це дозволяє створювати на мові С програмний код для всього модельного ряду. Однак для цього слід якимось чином замінити ті параметри, які відрізняються у різних моделей.

Для таких цілей використовуються директиви умовної компіляції `#ifdef`, `# if-ndef`, `#else` і `#endif`, а також - `#if` і `#elif`.

Синтаксис для директиви `#ifdef`:

```
#ifdef ім'я_макросу
послідовність_операторів_1
#else
послідовність_операторів_2
#endif
```

Якщо ім'я макросу визначено у програмі, то компілюється перша послідовність операторів, в іншому випадку - друга послідовність (гілка `#else` може бути відсутня).

Приклад використання для компілятора CCS-PICC:

```
#define DEBUGGING_ON
#ifdef DEBUGGING_ON
#use rs232 (baud = 9600, xmit = PIN_C6, rcv = PIN_C7)
#endif
```

Синтаксис для директиви `#ifndef`*:*

```
#ifndef ім'я_макросу
послідовність_операторів_1
#else
послідовність_операторів_2
#endif
```

У даному випадку, на відміну від директиви `#ifdef`, перша послідовність операторів виконується у тому випадку, якщо ім'я макросу у програмі не визначено.

Для умовної компіляції можна також скористатися директивами `#if` і `#elif`.

Їх синтаксис:

```
#if вираз1
послідовність_операторів_1
#elif вираз2
послідовність_операторів_2
#else
послідовність_операторів_3
#endif
```

Ця конструкція працює аналогічно умовному оператору `if-else`. Компілятор оцінює вирази після `#if` і `#elif` до тих пір, поки один з них не дасть у результаті `TRUE`, після чого у текст програми підставляється відповідна послідовність операторів. Якщо обидва вирази дають `FALSE`, то підставляється послідовність операторів після директиви `#else` (якщо вона є).

Директива #error

Директива `#error` використовується спільно з директивами умовної компіляції. Зустрівши її, компілятор згенерує повідомлення про помилку, вказану праворуч від директиви.

Директиви, характерні для компілятора CCS-PICC

Компілятор CCS-PICC підтримує безліч вбудованих, нестандартних директив.

Розглянемо деякі з них.

Директива #bit

Директива препроцесора `#bit` використовується для отримання доступу до окремих розрядів регістрів або змінних.

Її синтаксис:

```
#bit ідентифікатор = x.y
```

де: `x` - константа, яка визначає регістр, або змінна;

`y` - константа від 0 до 7.

Наприклад, таким чином змінна `T0IF` оголошується як розряд 2 регістра за адресою `0x0B`:

```
#bit T0IF = 0x0B.2
```

Приклад для розряду змінної:

```
inc c;
#bit CBit0 = C.0;
...
if (CBit0 == 0) ...
```

Директива #byte

Директива `#byte` має наступний синтаксис:

```
#byte ідентифікатор = X
```

де `X` - ім'я змінної або константи.

Якщо ***ідентифікатор*** - це вже оголошене раніше ім'я змінної, то компілятор поміщає цю змінну за адресою `X`, в іншому випадку компілятор створює нову змінну і поміщає її за адресою `X` як восьмирозрядне ціле число. В обох випадках в комірку пам'яті з адресою `X` можуть бути поміщені будь-які інші змінні. Фактично, якщо `X` - ім'я змінної, то їй буде відповідати та ж адреса в пам'яті, що і для оголошеного ідентифікатора:

```
char c;              //Змінна c розміщується в пам'яті компілятором
#byte a = c;         //змінній b призначений та ж адреса, що і c
```

Директива #case

Директива препроцесора `#case` вказує компілятору бути чутливим до регістру символів. За замовчуванням, компілятор CCS-PICC не чутливий до регістру символів.

Директива #device

Директива `#device` визначає, який мікроконтролер є цільовим для програми. Її синтаксис:

```
#device ім’я_мікроконтроллера опції
```

Опції є необов'язковими і впливають на управління пам'яттю, аналого-цифровим перетворенням і можливістю налагодження.

Для управління АЦП призначена опція `ADC = x`, де `x` - кількість розрядів, зчитувальних з перетворювача за допомогою внутрішньої функції `read_adc()`.

Якщо необхідно, щоб згенерований код був сумісний з налагоджувальним програмним забезпеченням ICD від компанії Microchip, у директиву `#device` слід включити опцію `ICD = TRUE`.

Примітка: якщо опції не вказані, то компілятор використовує відповідні значення, вибрані за замовчуванням.

Директива #fuse

Директива `#fuse` визначає, які запобіжники повинні бути встановлені при програмуванні мікроконтролера. Її синтаксис:

```
#fuse опції
```

Набір опцій для різних пристроїв відрізняється. Для того щоб переглянути список запобіжників для конкретного мікроконтролера, у середовищі розробки **CCS-PICC** слід виконати команду меню **View ▸ Valid Fuses**.

Наприклад, для мікроконтролера **PIC18F252** використовується наступний перелік запобіжників:

- `BORV20` - скидання при падінні напруги живлення до 2 В;
- `BORV27` - скидання при падінні напруги живлення до 2,7 В;
- `BORV42` - скидання при падінні напруги живлення до 4,2 В;
- `BORV45` - скидання при падінні напруги живлення до 4,5 В;
- `BROWNOUT` - скидання при виявленні провалу живлення;
- `CPB` - захист коду у блоці початкового завантаження;
- `CPD` - включений захист даних в пам'яті EEPROM;
- `DEBUG` - активний режим налагодження за допомогою ICD;
- `EC` - зовнішнє джерело синхроімпульсів з CLKOUT;
- `EC_10` - зовнішнє джерело синхроімпульсів;
- `HS` - високошвидкісний осцилятор з частотою > 4 МГц;
- `LP` - осцилятор малої потужності з частотою < 200 кГц;
- `LVP` - низьковольтне програмування по виводу B3 (PIC16) або B5 (PIC18);
- `NOBROWNOUT` - скидання при провалі живлення не використовується;
- `NOCPB` - захист коду в блоці початкового завантаження відключений;
- `NOCPB` - захист коду в блоці початкового завантаження відключений;
- `NOCPD` - захист даних в пам'яті EEPROM відключений;
- `NODEBUG` - відсутність режиму відладки для ICD;
- `NOLVP` - низьковольтне програмування не використовується; вивід B3 (PIC16) або B5 (PIC18) задіяний для введення/виведення;
- `NOOSCSEN` - перемикання осцилятора заборонено;
- `NOPROTECT` - код не захищений від читання;
- `NOPUT` - таймер включення живлення не використовується;
- `NOSTVREN` - переповнення або вихід за нижню межу стека не приводить до скидання;
- `NOWDT` - сторожовий таймер не використовується;
- `NOWRT` - захист від запису в пам'ять програм відключена;
- `NOWRTB` - захист від запису в блок початкового завантаження відключена;

- NOWRTC - захист від запису в регістри конфігурації відключена;
- NOWRTD - захист від запису даних в пам'ять EEPROM відключена;
- OSCSEN - активно перемикання осцилятора;
- PROTECT - захист коду від читання;
- OSCSEN - активно перемикання осцилятора;
- PROTECT - захист коду від читання;
- PUT - використовується таймер включення живлення;
- RC - RC-осцилятор з CLKOUT;
- RC_10 - RC-осцилятор;
- STVREN - переповнення або вихід за нижню межу стека призводить до скидання;
- WDT - використовується сторожовий таймер;
- WDT1 - сторожевой таймер використовує постдільник 1:1;
- WDT2 - сторожовий таймер використовує постдільник 1:2;
- WDT4 - сторожовий таймер використовує постдільник 1:4;
- WDT8 - сторожовий таймер використовує постдільник 1:8;
- WDT16 - сторожовий таймер використовує постдільник 1:16;
- WDT32 - сторожовий таймер використовує постдільник 1:32;
- WDT64 - сторожовий таймер використовує постдільник 1:64;
- WDT128 - сторожовий таймер використовує постдільник 1:128;
- WRT - захист від запису в пам'ять програм;
- WRTB - захист від запису в блок початкового завантаження;
- WRTC - захист від запису в регістри конфігурації;
- WRTD - захист від запису даних в пам'ять EEPROM;
- XT - осцилятор на кристалі з частотою <= 4 МГц.

Так, наступний приклад директиви #fuses встановлює високошвидкісний осцилятор, активізує скидання при виявленні провалу живлення і відключає сторожовий таймер:

```
#fuses HS, BROWNOUT, NOWDT
```

Директива #locate

Директива препроцесора `#locate` за своїм призначенням і синтаксисом аналогічна директиві `#byte` за тим винятком, що призначає змінній комірку пам'яті, в яку не може бути поміщена ніяка інша змінна. наприклад:

```
#locate c = 0x50;           //Розміщення змінної c за адресою 0x50
```

Директива #org

Директива `#org` дозволяє компілятору визначити, де повинна бути розміщена в пам'яті наступна функція. Вона має синтаксис чотирьох видів:

```
#org початок, кінець
#org сегмент
#org початок, кінець{}
#org початок, кінець auto = 0
```

де `початок` завжди визначає першу, а `кінець` - останню комірку в області пам'яті, що використовується; `сегмент` - початок області з попередньої директиви `#org`. Якщо раніше був визначений деякий сегмент, і необхідно тільки додати до нього ще одну функцію, ідентифікатор закінчення може бути опущений.

Приклад використання директиви `#org`:

```
#org 0x1E00, 0xlFFF

fl()
{
       //Ця функція буде розміщена, починаючи з адреси 0xlE00
}

#org 0xlE00

//Сегмент - той же, який був визначений вище
f2()
{
       //Ця функція буде розміщена у сегменті 0xlE00-0xlFFF
}
#org 0x800, 0x820{}
//Порожній сегмент
```

Директива #opt

Директива `#opt` задає рівень оптимізації, яка застосовується при компіляції. Вона має синтаксис:

```
#opt рівень
```

де рівень - число від 0 до 9 (9 - повна оптимізація).

Директива #priority

Директива препроцесора `#priority` може бути використана для установки порядку, в якому опитуються прапори переривань. Деякі мікроконтролери PIC мають тільки один вектор переривань. Коли виникає переривання, управління передається за адресою, зазначеною у векторі.

За замовчуванням, компілятор **CCS-PICC** поміщає за цією адресою диспетчерську підпрограму, яка опитує прапори переривань, щоб визначити, яке з них мало місце, і викликати відповідну підпрограму обслуговування переривання.

Синтаксис директиви:

```
#priority список_переривань
```

Приклад використання:

```
#priority rtcc, rb
```

Примітка: переривання з великим пріоритетом розташовані у списку першими.

Директива #reserve

Директива `#reserve` визначає області ОЗП, не доступні для використання компілятором.

Її синтаксис має дві форми:

```
#reserve адреса, адреса, адреса ...
#reserve початкова_адреса_діапазону:кінцева_адреса_діапазону
```

Примітка: директиві `#reserve` повинна передувати директива `#device`, інакше вона не матиме ніякого ефекту.

Приклади резервування області `0x20..0x22`:

```
#reserve 0x10, 0x11, 0x12
#reserve 0x10: 0x12
```

Директива #rom

Директива препроцесора `#rom` дозволяє вставляти дані у файл `.hex`.
Її синтаксис:

```
#rom адреса = {список}
```

де `адреса` - адреса в пам'яті, а `список` - список слів через кому.

Приклад:

```
#rom 0x2100 = {1, 2, 3, 4, 5, 6, 7, 8}
```

Директива #type

Директива препроцесора `#type` дозволяє перевизначати типи, підтримувані компілятором. Її синтаксис:

```
#type стандартний_тип = розмір, стандартний_тип = розмір,...
```

Наприклад, за замовчуванням, компілятор **CCS-PICC** відводить під змінну типу `char` вісім біт, а під змінну типу `int` - 16 біт. Для того щоб змінити цю ситуацію, у текст програми можна додати наступну директиву:

```
#type char = 16, int = 32
```

Директива #use delay

Директива `#use delay` призначена для завдання робочої частоти процесора і дозволяє використання у програмі внутрішніх функцій `delay_ms()` (затримка у мілісекундах) і `delay_us()` (затримка в мікросекундах).

Її синтаксис має дві форми:

```
#use delay (частота)
#use delay (частота, restart_WDT)
```

Частота вказується у герцах. За допомогою опції `restart_DWT` можна вказати компілятору перезавантажувати сторожовий таймер під час програмних затримок.

Директиви #use xxx_io

Директиви вигляду `#use xxx_io` впливають на код, згенерований при реалізації доступу до портів введення/виведення, і мають відношення до установки регістрів TRIS.

У випадку стандартного доступу на введення/виведення (вибір за замовчуванням), компілятор генерує код, переводить деякий вивід мікроконтролера у стан входу або виходу при кожному зверненні до нього. Такий спосіб доступу активізується директивою препроцесора `#use standard_io`:

```
#use standard_io (X)
```

де `X` - символ порту (`A ... G`).

Якщо користувач повинен встановлювати розряди регістрів TRIS вручну перед зверненням до відповідних виводів безпосередньо або за допомогою вбудованої функції `set_tris_x()`, то у програмі повинна бути вказана директива препроцесора `#use fast_io`:

```
#use fast_io (X)
```

де `X` - символ порту (`A ... G`).

Подібна за змістом директива `#use fixed_io` приймає в якості параметрів ідентифікатори виводів, що встановлюються в якості входів/виходів:

```
#use fixed_io (X_outputs = вивід, вивід, вивід ...)
```

де `X` - символ порту (`A ... G`), вивід - константа, відповідна виводу (перелік таких констант можна знайти у заголовному файлі пристрою).

Приклади для виводу 0 порту B:

```
#use standard_io(B)
#use fast_io(B)
#use fixed_io(b_outputs = PIN_B0)
```

Директива #use i2c

Директива `#use i2c` визначає порт I^2C мікроконтролера.

Її синтаксис:

```
#use i2c (опції)
```

В якості `опції` (вказуються через кому) можуть використовуватися наступні значення:

- `ADDRESS = nn` - завдання адреси веденого пристрою;
- `FAST` - використання специфікації швидкої шини I^2C;
- `FORCE_HW` - використання апаратних функцій I^2C;
- `MASTER` - установка режиму ведучого пристрою;
- `RESTART WDT` - перезавантаження сторожового таймера під час очікування в процесі виконання вбудованої функції `i2c_read()`;
- `SCL = вивід` - завдання виводу SCL (вивід - адреса розряду);
- `SDA = вивід` - завдання виводу SDA;
- `SLAVE` - установка режиму веденого пристрою;
- `SLOW` - використання специфікації повільної шини I^2C.

У наступному прикладі в якості лінії даних встановлюється вивід 0 порту B, а в якості лінії, що тактується - вивід 1 того ж порту (режим ведучого пристрою):

```
#use i2c (MASTER, sda = PIN PIN B0, scl = PIN Bl)
```

Приклад для режиму веденого пристрою:

```
#use i2c (SLAVE, sda = PIN_C4, scl = PIN_C3, ADDRESS = 0xA0, FORCE_HW)
```

Примітка: директива `#use i2c` вважається активною доти, поки в тексті програми не зустрінеться інша директива `#use i2c`.

Ця директива дозволяє скористатися вбудованими функціями, які реалізують обмін даними по шині I^2C (якщо включена опція `FORCE_HW`, то програмні функції не генеруються і використовується апаратний порт MSSP).

Директива #use rs232

Директива препроцесора `#use rs232` використовується для ініціалізації послідовного порту, що працює за стандартом RS-232. Залежно від того, як задані виводи передачі і прийому, реалізація порту може бути апаратною або програмною. Якщо виводи відповідають приемопередатчику USART, то використовується апаратний порт, в іншому випадку - програмний UART.

Загальна форма директиви `#use rs232`:

```
#use rs232 (опція, опція, опція, ...)
```

Можливі такі опції:

- `BAUD = x` - встановлення швидкості передачі;
- `BITS = x` - встановлення розрядності у `x`, де `x` - 5-9 у випадку програмного UART і 8-9 у випадку апаратного UART;
- `ENABLE = вивід` - під час передачі компілятор переводить вказаний вивід у стан високого рівня;
- `ERRORS = вивід` - під час передачі компілятор переводить вказаний вивід у стан високого рівня;
- `ERRORS` - помилки прийому зберігаються у змінній `RS232_ERRORS`;
- `FLOAT HIGH` - використання на виході схем з відкритим колектором;
- `INVERT` - інвертує полярність виводів послідовного порту (використовується тільки з програмним прийомопередавачем UART);
- `PARITY = x` - контроль по парності: `x = N` - відсутня; `x = E` - контроль по парності; `x = 0` - контроль по непарності;
- `RCV = вивід` - установка виводу для прийому даних;
- `RESTART_WDT` - скидання сторожового таймера при очікуванні символу у функції `getchar()`;
- `STREAM = ім'я_потоку` - асоціює ідентифікатор потоку з портом RS-232 (цей ідентифікатор може використовуватися у функціях, на зразок `fputc()`).
- `XMIT = вивід` - установка виводу для передачі даних.

Примітка: при виборі швидкості передачі слід враховувати частоту системної синхронізації мікроконтролера. З цієї причини перед директивою `#use rs232` обов'язково повинна бути вказана директива `#use delay`. Якщо зазначена швидкість не може бути досягнута (з відхиленням 3%), то генерується повідомлення про помилку.

Директива #zero_ram

Директива препроцесора `#zero_ram` вказує компілятору обнуляти перед початком виконання програми всі внутрішні регістри, які можуть бути використані для зберігання змінних.

Обробка переривань в середовищі CCS-PICC

У середовищі компілятора CCS-PICC для визначення підпрограм обробки переривань використовуються директиви препроцесора `#int_default`, `#int_global` і `#int_xxx` (вказуються безпосередньо перед функцією). У багатьох мікроконтролерах PIC використовується єдиний вектор переривання, і, отже, у випадку виникнення переривання викликається тільки одна підпрограма, яка називається диспетчером переривань.

Диспетчер відповідає за опитування прапорів переривань і виклику відповідних підпрограм обслуговування. Якщо виявлено переривання, і жоден з прапорів переривання не встановлений, то диспетчер викликає функцію, позначену за допомогою директиви `#int_default`.

Директива `#int_global` дозволяє визначити функцію, яка замінює диспетчер компілятора (такі функції використовуються вкрай рідко, і з ними слід дотримуватися великої обережності).

Директива `#int_xxx` визначає функцію, що відповідає за обслуговування того чи іншого переривання.

Перелічимо найбільш поширені директиви цього типу:

- `#int_ad` - аналого-цифрове перетворення завершено;
- `#int_adof` - затримка аналого-цифрового перетворення;
- `#int_buscol` - конфлікт шини;
- `#int_button` - натискання кнопки;
- `#int_ccp1` - захоплення або збіг у модулі CCP1;
- `#int_ccp2` - захоплення або збіг в модулі CCP2;
- `#int_comp` - переривання від аналогового компаратора;
- `#int_eeprom` - запис у EEPROM завершений;
- `#int_ext` - зовнішнє переривання;
- `#int_ext1` - зовнішнє переривання 1;
- `#int_ext2` - зовнішнє переривання 2;
- `#int_i2c` - переривання від шини I^2C;
- `#int_lcd` - ЖК-дисплей активний;
- `#int_lowvolt` - виявлено низьку напругу;
- `#int_psp` - запис даних у порт PSP;
- `#int_rb` - зміна стану виводів 4-7 порту B;
- `#int_rc` - зміна стану виводів 4-7 порту C;
- `#int_rda` - доступний прийом даних по інтерфейсу RS-232;
- `#int_rtcc` - переповнення таймера TMR0;
- `#int_ssp` - активність інтерфейсу SPI або I^2C;
- `#int_tbe` - буфер передачі по інтерфейсу RS-232 порожній;
- `#int_timer0` - переповнення таймера TMR0;
- `#int_timer1` - переповнення таймера TMR1;
- `#int_timer2` - переповнення таймера TMR2;
- `#int_timer3` - переповнення таймера TMR3.

У ряді мікроконтролерів PIC використовується два вектора переривань. Це дозволяє встановити високий пріоритет деякому перериванню на апаратному рівні.

Компілятор **CCS-PICC** використовує цю апаратну можливість за допомогою опції `FAST` директиви `#int_xxx`:

```
#int_xxx FAST
```

Цей рівень пріоритету можна призначити тільки одному перериванню, тому прапор `FAST` може бути використаний в програмі тільки один раз.

У випадку використання прапора `FAST` відповідне переривання не зберігає всіх регістрів так, як це робить звичайне переривання. Зберігаються тільки основні регістри процесора. Таким чином, наприклад, можна швидко виконувати якісь дуже прості операції.

ФУНКЦІЇ І МАКРОСИ КОМПІЛЯТОРА CCS-PICC

Математичні макроси

Бібліотека: math.h

Таблиця 4.12 - Математичні макроси компілятора **CCS-PICC**

Макрос	Бібліотека	Визначення	Опис
LN2	math.h	#define LN2 0.6931471806	Натуральний логарифм від 2
LN10	math.h	#define LN10 2.30258509	Натуральний логарифм від 10
PI	math.h	#define PI 3.141592654	Число π
PI_DIV_BY_TWO	math.h	#define PI_DIV_BY_TWO 1.570796326794896	Число π / 2
RAND_MAX	stdlib.h	#define RAND_MAX 32767	Найбільше значення, яке може бути створене функцією `rand()`
SQRT2	math.h	#define SQRT2 1.41421356	Квадратний корінь з 2
TWOBYPI	math.h	#define TWOBYPI 0.6366197724	Число 2 / π

Функції для роботи з рядками

Бібліотека: stdlib.h

Специфічні функції компілятора **CCS-PICC**, призначені для роботи з рядками, перелічені у табл. 4.13.

Таблиця 4.13 - Функції компілятора **CCS-PICC** для роботи з рядками

Функція	Бібліотека	Заголовок	Опис
`atof`	`stdlib.h`	`float atof(char *str)`	Перетворює рядок `str` у дійсне число
`atoi32`	`stdlib.h`	`int32 atoi32(char *str)`	Перетворює рядок `str` у цілочисельне представлення (32-розрядне ціле)
`strcspn`	`string.h`	`unsigned char strcspn(char *str, char *set)`	Повертає індекс першого символу у рядку `str`, який збігається з символом у рядку `set`. Якщо жодного з символів `str` немає у `set`, повертається довжина `str`. В іншому випадку повертається `0`
`stricmp`	`string.h`	`signed char stricmp(char *sl, char *s2)`	Порівнює два рядки без урахування регістру символів і повертає число `< 0`, якщо `s1 < s2`; `0`, якщо `s1 = s2` або число `> 0`, якщо `s1 > s2`
`strpbrk`	`string.h`	`char* strpbrk (char *str, char *set)`	Переглядає рядок `str` у пошуках першого входження деякого символу з рядка `set`. Якщо входження знайдено, повертається покажчик на відповідний символ у рядку `str`. В іншому випадку повертається `NULL`

Продовження таблиці 4.13

Функція	Бібліотека	Заголовок	Опис
`Strspn`	`string.h`	`unsigned char strspn(char *str, char *set)`	Повертає індекс першого символу у рядку `str`, який не відповідає якому-небудь символу у рядку `set`. Якщо всі символи `str` присутні у `set`, повертається довжина `str`. Інакше повертається 0
`strtok`	`string.h`	`char* strtok (char *s, char const *delim)`	Переглядає рядок `s` у пошуку першого символу, що не входить у рядок `delim`. При цьому передбачається, що `s1` складається з послідовності підрядків, розділених символами з рядка `delim`. Перший виклик функції поверне покажчик на перший символ-роздільник, після якого в рядку буде вставлений символ "`\0`". Наступні виклики функції повертатимуть наступні подстроки зі вставкою замість роздільника символу "\0" до тих пір, поки не будуть переглянуті всі роздільники. Після цього повертається `NULL`

Функції для організації введення/виведення

Специфічні функції компілятора **CCS-PICC**, призначені для організації введення/виведення, перелічені у табл. 4.14.

Таблиця 4.14 - Функції компілятора **CCS-PICC** для організації введення/виведення

Функція	Бібліотека	Заголовок	Опис
`assert`	`assert.h` `#use rs232()`	`void assert (condition)`	Перевіряє будь-який умовний вираз `condition`. Якщо `condition` дає у результаті `FALSE`, то у стандартний потік `stderr` видається повідомлення про помилку (ім'я файлу і номер рядка, в якому була викликана функція `assert`). Якщо в програму включити директиву препроцесора `#define NODEBUG`, то компілятор проігнорує виклик функції `assert`
`fgetc`	`#use rs232()`	`char fgetc (stream)`	Очікує прийому символу з потоку USART, заданого параметром `stream`. Ім'я цього параметра має збігатися з тим, яке задано директивою `#use rs232`

Продовження таблиці 4.14

Функція	Бібліотека	Заголовок	Опис
fgets	#use rs232()	void fgets (char *str, stream)	Очікує прийому рядку з потоку USART, заданого параметром stream. Ім'я цього параметра має збігатися з тим, яке задано директивою #use rs232. Рядок зчитується у буфер str до тих пір, поки не зустрінеться символ повернення каретки. Приймальний буфер повинен мати достатній розмір, щоб помістити рядок і завершальний символ "\0"
fprintf	#use rs232()	void fprintf (stream, char *fmt, [argl, arg2, ...])	Виводить форматований рядок fmt у потік USART, заданий параметром stream (повинен збігатися із значенням у директиві #use rs232), з урахуванням підстановки параметрів зі списку arg1, arg2 ... (не обов'язковий).
fputc	#use rs232()	void fputc (char C, stream)	Виводить символ c у потік USART, заданий параметром stream (повинен співпадати із значенням у директиві #use rs232)

Продовження таблиці 4.14

Функція	Бібліотека	Заголовок	Опис
Fputs	#use rs232()	void fputs (char *str, stream)	Виводить рядок str у потік USART, заданий параметром stream (повинен співпадати із значенням у директиві #use rs232). Переданий рядок автоматично доповнюється символами "\n\r"
getc	#use rs232()	char getc()	Очікує прийому символу з потоку USART за замовчуванням (заданий останньою директивою #use rs232 перед викликом функції)
gets	#use rs232()	void gets (char *str)	Очікує прийому рядку з потоку USART за замовчуванням (заданий останньою директивою #use rs232 перед викликом функції). Рядок зчитується у буфер, на який вказує параметр str, до тих пір, поки не зустрінеться символ повернення каретки. Приймальний буфер повинен мати достатній розмір, щоб помістити рядок і завершальний символ "\0"

Продовження таблиці 4.14

Функція	Бібліотека	Заголовок	Опис
input	-	char input (const pin)	Повертає поточний стан виводу pin. Метод введення/виведення, що використовується, залежить від останньої директиви #use *10
input_X	-	char input X()	Зчитує байт з порту X(A, B, C, ...)
kbhit	#use rs232()	char kbhit()	Повертає 0 (FALSE), якщо функції getc необхідно очікувати приходу символу, або 1 (TRUE), якщо символ готовий. У випадку програмної реалізації USART функція повертає TRUE, якщо на вхідний вивід надійшов перший біт символу
output bit	-	void output_bit (const pin, char value)	Виводить value (0 або 1) через вивід pin. Метод введення/виведення, що використовується, залежить від останньої директиви #use *10
output_float	-	void output_float (const pin)	Переводить вивід pin у стан входу. Метод введення/виведення, що використовується, залежить від останньої директиви #use *10

Продовження таблиці 4.14

Функція	Бібліотека	Заголовок	Опис
output_high	-	void output_high (const pin)	Переводить вивід pin у стан лог. 1. Метод введення/виведення, що використовується, залежить від останньої директиви #use *10
output_low	-	void output_low (const pin)	Переводить вивід pin у стан лог. 0. Метод введення/виведення, що використовується, залежить від останньої директиви #use *10
output_X	-	void output_X (char value)	Виводить байт value у порт X (A, B, C ...)
port_b_pullups	-	void port_b_pullups char value)	Активізує (value = TRUE) або відключає (value = FALSE) резистори порту B, що підтягуються
printf	#use rs232()	void printf (char *fmt, [argl, arg2,…])	Виводить форматований рядок fmt у потік USART за замовчуванням з урахуванням підстановки параметрів зі списку argl, arg2 ... (не обов'язково)
putc	#use rs232()	void putc (char C)	Виводить символ C у потік USART за замовчуванням

Продовження таблиці 4.14

Функція	Бібліотека	Заголовок	Опис
`Puts`	`#use rs232()`	`void puts (char *str)`	Виводить рядок str у потік USART за замовчуванням. Переданий рядок автоматично доповнюється символами "`\n\r`"
`set_tris_X`	`#use fast io`	`void set_tris_X (char value)`	Задає напрям передачі даних через порт X (A, B, C ...) відповідно `value`. Якщо в деякому розряді `value` встановлена лог. 1, то відповідний вивід порту працює як вхід, інакше - як вихід
`set_uart_speed`	`#use rs232()`	`void set_uart_speed (const int32 baud)`	Змінює швидкість передачі даних через апаратний послідовний порт RS-232. `baud` - константа у діапазоні від 100 до 115200
`sprintf`	-	`sprintf(char *str, char flash *fmt, [arg1, arg2, …])`	Копіює форматований рядок `fmt` у рядок `str` з урахуванням підстановки параметрів зі списку `arg1, arg2 ...` (не обов'язково). Рядок-приймач повинен бути достатнього розміру для збереження сформованого форматованого рядку

Функції управління мікроконтролером

Функції компілятора **CCS-PICC**, призначені для управління мікроконтролером, перелічені у табл. 4.15.

Таблиця 4.15 - Функції управління мікроконтролером

Функція	Бібліотека	Заголовок	Опис
`delay_cycles`		`void delay_cycles (const unsigned char C)`	Створює затримку, довжина якої відповідає кількості `c` послідовностей з чотирьох тактів системної синхронізації. Значення `c` може знаходитися у діапазоні від 1 до 255. Функція не використовує таймерів, тому, в результаті виникнення запиту на переривання в процесі її роботи, фактична затримка може виявитися більше

`delay_ms`	`#use delay`	`void delay_ms (const unsigned char C)`
		
		`void delay_ms (const intl6 C)`
			Створює затримку на `c` мілісекунд. Якщо `c` - змінна, то її значення може знаходитися у діапазоні від 1 до 255. Якщо `c` - константа, то її значення може знаходитися в діапазоні від 1 до 65535. Функція не використовує таймерів, тому в результаті запиту на переривання у

Функція	Бібліотека	Заголовок	Опис
			процесі її роботи фактична затримка може виявитися більше
delay_us	#use delay	void delay_us(const unsigned char C) void delay_us(const int16 C)	Створює затримку на C мікросекунд. Якщо C - змінна, то її значення може знаходитися в діапазоні від 1 до 255. Якщо C - константа, то її значення може знаходитися в діапазоні від 1 до 65535. Ця функція не використовує таймерів, тому, в результаті виникнення запиту на переривання в процесі її роботи, фактична затримка може виявитися більше
disable_ interrupts	#int_xxx	void disable_interrupts (int_level)	Забороняє переривання на рівні int_level. int_level - це константа, яка представляє вид переривання (збігається з ім'ям директиви #int_xxx). Наприклад, переривання при переповненні TMR0 відповідає константа INT_RTCC, а зовнішньому перериванню - константа INT_EXT.

Функція	Бібліотека	Заголовок	Опис
			Всім переривань відповідає константа GLOBAL
enable_ interrupts	#int_xxx	void enable_interrupts (int level)	Дозволяє переривання на рівні int_level. int_level - це константа, яка представляє вид переривання (збігається з ім'ям директиви #int_xxx). Наприклад, переривання при переповненні TMR0 відповідає константа INT_RTCC, а зовнішньому перериванню - константа INT_EXT. Всім перериванням відповідає константа GLOBAL
ext_int_edge	-	void ext_int_edge ([source], edge)	Визначає, який фронт сигналу на вході зовнішнього запиту на переривання викликає переривання. Якщо edge = L_TO_H, то активним є наростаючий фронт, а значенням H_TO_L відповідає спадаючий фронт. Параметр source використовується тільки в мікроконтролерах, що дозволяють задати джерело зовнішнього переривання
reset_cpu		void reset_cpu()	Скидання мікроконтролера

Функція	Бібліотека	Заголовок	Опис
restart_cause	-	char restart_cause()	Повертає значення, яке вказує на причину останнього скидання процесора, наприклад: WDT_FROM_SLEEP, WDT_TIMEOUT, MCLR_FROM_SLEEP, NORMAL POWER UP та ін.
sleep		void sleep()	Виконує команду sleep

Функції для роботи з таймерами і модулем CCP

Функції компілятора **CCS-PICC**, призначені для роботи з таймерами і модулем CCP, перелічені у табл. 4.16.

Таблиця 4.16 - Функції компілятора **CCS-PICC** для роботи з таймерами і CCP

Функція	Бібліотека	Заголовок	Опис
get_rtcc	-	int8 get_rtcc() intl6 get_rtcc()	Повертає поточне значення лічильного регістра TMR0
get_timerX	-	int8 get_timer0() int16 get_timer1() int8 get_timer2() intl6 get timer3()	Повертає поточне значення лічильного регістра відповідного таймера / лічильника
restart_wdt	#fuses WDT	void restart_wdt()	Скидає сторожовий таймер
set_pwmX_ duty	-	void set_pwm1_duty (intl6 value) void set_pwm2_duty (intl6 value)	Записує у модуль CCP 10-ти розрядне значення value для установки робочого циклу ШІМ
set_rtcc	-	void set_rtcc (int8 value)	Встановлює значення лічильного регістра TMR0

Функція	Бібліотека	Заголовок	Опис
		void set_rtcc (intl6 value)	
set_timerX	-	void set_timer0 (int8 value) void set_timer0 (intl6 value) void set_timer1 (intl6 value) void set_timer2 (int8 value) void set timer3 (int16 value)	Встановлює значення лічильного регістра відповідного таймера / лічильника
setup_ccpX	-	void setup_ccp1 (const mode) void setup_ccp2 (const mode)	Ініціалізує CCP відповідно до значення константи mode (до лічильників CCP можна отримати доступ за допомогою глобальних змінних ccp_1 і ccp_2): - CCP_OFF - модуль CCP відключений; - CCP_CAPTURE_FE - захоплення стану TMR1 по спадаючому фронті на вивід CCPx; - CCP_CAPTURE_RE - захоплення стану TMR1 по наростаючому фронті на вивід CCPx; - CCP_CAPTURE_DIV_4 - захоплення стану TMR1 по кожному четвертому

Функція	Бібліотека	Заголовок	Опис
			наростаючому фронті на вивід CCPx; - CCP_CAPTURE_DIV_16 - захоплення стану TMR1 по кожному шістнадцятому наростаючому фронті на вивід CCPx; - CCP_COMPARE_SET_ON_MATCH - установка лог. 1 на виводі CCPx при збігу вмісту TMR1 і змінної ccp_x; - CCP_COMPARE_CLEAR_ON_MATCH - установка лог. 0 на вивід CCPx при збігу вмісту TMR1 і змінної ccp_x; - CCP_COMPARE_INT - формування запиту на переривання при збігу вмісту TMR1 і змінної ccp_x; - CCP_COMPARE_RESET_TIMER - скидання таймера при збігу вмісту TMR1 і змінної ccp_x; - CCP_PWM - режим ШІМ
setup_timer_X	-	void setup_timer_0 (mode) void setup_timer_1 (mode) void setup timer 2 (mode, unsigned char	Ініціалізація таймера / лічильника. mode - константа, що складається з однієї або декількох опцій. Параметр period (від 0 до 255)

Функція	Бібліотека	Заголовок	Опис
		period, int8 postscale) void setup_timer_3 (mode)	визначає момент скидання значення лічильника. Параметр postscale (від 1 до 16) визначає, скільки скидів таймера має відбутися перед виникненням переривання
setup_wdt	#fuses WDT	void setup_wdt (mode)	Ініціалізація сторожового таймера. Параметр mode - константа, що складається з однієї або декількох опцій, що визначають час до настання скидання від сторожового таймера: WDT_18MS, WDT_36MS, WDT_72MS, WDT_14 4MS, WDT_288MS, WDT_576MS, WDT_1152MS, WDT_2304MS

Функції для роботи з розрядами і пам'яттю

Функції компілятора **CCS-PICC**, призначені для роботи з розрядами і пам'яттю, перелічені у табл. 4.17

Таблиця 4.17 - Функції компілятора **CCS-PICC** для роботи з розрядами і пам'яттю

Функція	Бібліотека	Заголовок	Опис
bit_clear	-	void bit_clear (int8 var, char bit) void bit_clear (int16 var, char bit)	Очищає розряд bit цілочисельної змінної var. Якщо var має тип int8, bit може приймати значення від 0 до 7;

Функція	Бібліотека	Заголовок	Опис
		`void bit_clear (int32 var, char bit)`	для `int16` - від 0 до 15, а для `int32` - від 0 до 31

`bit_set`	-	`void bit_set (int8 var, char bit)`
		
		`void bit_set(intl6 var, char bit)`
		
		`void bit_set(int32 var, char bit)`
			Встановлює в лог. 1 розряд `bit` цілочисельної змінної `var`. Якщо `var` має тип `int8`, `bit` може приймати значення від 0 до 7; для `int 16` - від 0 до 15, а для `int32` - від 0 до 31

`bit_test`	-	`int1 bit_test(int8 var, char bit)`
		
		`int1 bit_test(intl6 var, char bit)`
		
		`int1 bit_test(int32 var, char bit)`
			Повертає стан розряду `bit` цілочисельної змінної `var`. У разі, якщо `var` має тип `int8`, `bit` може приймати значення від 0 до 7; для `int16` - від 0 до 15, а для `int32` - від 0 до 31

`input`	-	`char input(const pin)`	Повертає поточний стан виводу `pin` (адреса-константа виведення порту): 0 або 1. Метод введення/виведення, що використовується, залежить від останньої директиви `#use *io` перед викликом функції
`make16`	-	`intl6 makel6 (int8 var_high, int8 var_low)`	Повертає 16-розрядне значення, складене з двох восьмирозрядних величин `var_high` (старший байт) і `var low` (молодший байт).

Функція	Бібліотека	Заголовок	Опис
			Якщо розрядність одного з параметрів більше 8 біт, використовуються тільки молодші вісім розрядів
make32	-	int32 make32 (int8 varl [, int8 var2, int8 var3, int8 var4])	Повертає 32-розрядне значення, складене з одного, двох, трьох або чотирьох восьмирозрядних величин. Параметр var1 відповідає найстаршому байту, а var4 - наймолодшому. Якщо загальне число розрядів менше 32, то старші розряди в повернутому значенні заповнюються нулями
make8	-	int8 make8 (intl6 var, int8 offset) int8 make8(int32 var, int8 offset)	Повертає восьмирозрядне значення, отримане з 16- або 32-розрядної величини var шляхом зсуву на offset байтів
offsetof	stddef.h	int8 offsetof (structure_type, field)	Повертає зсув (в байтах) поля field у структурі типу structure type
offsetofbit	stddef.h	int8 offsetofbit (structure_type, field) .	Повертає зсув (в бітах) поля field у структурі типу structure type
read_bank	-	int8 read_bank (int8 bank, intl6 offset)	Зчитує байт даних з області RAM банку bank (1-3, в залежності від

Функція	Бібліотека	Заголовок	Опис
			мікроконтролера) по зміщенню `offset`
`rotate_left`	-	`void rotate_left (char *addr, char bytes)`	Виконує циклічний зсув вліво на один розряд `bytes` байтів в області пам'яті, на яку посилається `addr`
`rotate_ right`	-	`void rotate right (char *addr, char bytes)`	Виконує циклічний зсув вправо на один розряд `bytes` байтів в області пам'яті, на яку посилається `addr`
`shift_ left`	-	`int1 shift_left (char *addr, char bytes, int1 value)`	Виконує зсув вліво на один розряд `bytes` байтів в області пам'яті, на яку посилається `addr`. Перший розряд заповнюється значенням `value`. Функція повертає зсунутий розряд (останній перед зсувом)
`shift_ right`	-	`int1 shift_right (char *addr, char bytes, int1 value)`	Виконує зсув вправо на один розряд `bytes` байтів в області пам'яті, на яку посилається `addr`. Останній розряд заповнюється значенням `value`. Функція повертає зсунутий розряд (перший перед зсувом)
`swap`	-	`void swap(int8 x)`	Міняє місцями старший і молодший напівбайти `x`
`write_ bank`	-	`void write_bank (int8 bank, intl6 offset, int8 value)`	Записує байт даних `value` в область RAM банку `bank` (1-3, в залежності від

Функція	Бібліотека	Заголовок	Опис
			мікроконтролера) по зміщенню `offset`

Функції для роботи з пам'яттю EEPROM

Функції компілятора **CCS-PICC**, призначені для роботи з пам'яттю EEPROM, перелічені у табл. 4.18.

Таблиця 4.18 - Функції компілятора **CCS-PICC** для роботи з пам'яттю EEPROM

Функція	Бібліотека	Заголовок	Опис
`read_eeprom`	-	`int8 read_eeprom (int8 addr)`	Зчитує байт даних з пам'яті EEPROM за адресою `addr`
`write_ eeprom`	-	`void write_eeprom (int8 addr, int8 value)`	Записує байт даних `value` у пам'яті EEPROM за адресою `addr`

Функції для роботи з інтерфейсом SPI

Функції компілятора **CCS-PICC**, призначені для роботи з інтерфейсом SPI, перелічені у табл. 4.19.

Таблиця 4.19 - Функції компілятора **CCS-PICC** для роботи з інтерфейсом SPI

Функція	Бібліотека	Заголовок	Опис
`setup_spi`	-	`void setup_spi (const modes)`	Ініціалізує інтерфейс SPI відповідно до константи `modes`, яка задає режим і швидкість обміну даними. Можлива комбінація значень з наступних категорій:

Функція	Бібліотека	Заголовок	Опис
			- вибір режиму – SPI_SS_DISABLED, SPI_MASTER, SPI_SLAVE; - активний фронт синхроімпульсу – SPI_L_TO_H, SPI_H_TO_L; - частота тактування - SPI_CLK_DIV_4, SPI_CLK_DIV_16, SPI_CLK_DIV_64, SPI_CLK_T2
spi_data_is_in	-	char spi_data_is_in()	Повертає TRUE, якщо у порт SPI прийняті дані
spi_read	-	char spi_read()	Повертає значення, зчитане з порту SPI
spi_write	-	void setup_spi (char c)	Встановлює черговий байт даних для передачі у порт SPI. Якщо пристрій - ведучий, то функція генерує тактовий сигнал і передає байт. Якщо пристрій ведений, то байт не буде переданий до тих пір, поки не буде отримано тактовий сигнал від ведучого пристрою

Функції дя роботи з інтерфейсом PSP

Функції компілятора CCS-PICC, призначені для роботи з інтерфейсом PSP, перелічені у табл. 4.20.

Таблиця 4.20 Функції компілятора CCS-PICC для роботи з інтерфейсом PSP

Функція	Бібліотека	Заголовок	Опис
`psp_input_full`	-	`int1 psp_input_full()`	Повертає 0, якщо з порту PSP не зчитується ніяких даних, або 1, якщо є дані на читання. Дані можна зчитувати за допомогою глобальної змінної `psp data`
`psp_output_full`	-	`int1 psp_output_full()`	Повертає 0, якщо у порт PSP ще не було записано ніяких даних, або 1, якщо дані вже були записані. Дані можна записувати у PSP за допомогою глобальної змінної `psp data`
`psp_overflow`	-	`int1 psp_overflow()`	Повертає 0, якщо порт PSP не знаходиться у стані переповнення (дані записуються зовнішнім пристроєм до того як прийняті раніше дані були прочитані мікроконтролером), або 1 в іншому випадку
`setup_psp`	-	`void setup_psp(mode)`	Ініціалізує (`mode = PSP_ENABLED`) або відключає (`mode = PSP DISABLED`) порт PSP

Функції для робои з інтерфейсом I²C

Функції компілятора **CCS-PICC**, призначені для роботи з інтерфейсом I²C, перелічені у табл. 4.21.

Таблиця 4.21 - Функції компілятора **CCS-PICC** для роботи з інтерфейсом I²C

Функція	Бібліотека	Заголовок	Опис
`i2c_poll`	`#use i2c`	`inti i2c_poll()`	Використовується тільки в випадку присутності апаратного порту SSP (інтерфейс I²C у режимі Slave). Функція повертає TRUE, якщо прийнятий байт в буфер прийому, і FALSE, якщо буфер прийому порожній
`i2c_read`	`#use i2c`	`int8 i2c_read ([int1 ack])`	Зчитує байт по інтерфейсу I²C. У режимі Master ця функція генерує такт синхронізації, а у режимі Slave - очікує цього такту (для стробування сторожового таймера під час очікування у режимі Slave слід додати у директиву `#use i2c` прапор `RESTART WDT`). Необов'язковий параметр `ack` (значення 1 або 0) визначає, чи має бути підтвердження байта при читанні
`i2c_start`	`#use i2c`	`void i2c_start()`	Створює умови початку передачі, коли мікроконтролер

Функція	Бібліотека	Заголовок	Опис
			знаходиться у режимі I^2C Master. Після створення такої умови, рівень сигналу у тактовій лінії буде низьким до тих пір, поки не буде викликана функція `i2c_write`. Якщо до виклику функції `i2c_stop` функція `i2c_start` буде викликана ще раз, це створить умову рестарту
`i2c_stop`	`#use i2c`	`void i2c_stop()`	Створює умови завершення передачі, коли мікроконтролер знаходиться у режимі I^2C Master
`i2c_write`	`#use i2c`	`int1 i2c_write (int8 data)`	Передає байт по інтерфейсу I^2C. У режимі Master також генерує тактовий сигнал, а у режимі Slave очікує такт від ведучого пристрою. Повертає біт квитування. Молодший розряд першого запису після створення умови початку передачі визначає напрямок передачі: 0 - від ведучого відомому, 1 - від веденого ведучому

Функції для роботи з аналоговими сигналами

Функції компілятора **CCS-PICC**, призначені для роботи з аналоговими сигналами, перелічені у табл. 4.22.

Таблиця 4.22 - Функції компілятора **CCS-PICC** для роботи з аналоговими сигналами

Функція	Бібліотека	Заголовок
`read_adc`	`int8 read adc()` `int16 read adc()`	Зчитує результат аналого-цифрового перетворення, розрядність якого залежить від розрядності АЦП і параметрів директиви `#device adc`
`set_adc_channel`	`void set_adc_channel (int8 channel)`	Вибирає аналоговий канал `channel` для використання при наступному виклику функції `read adcc`
`setup_adc`	`void setup_adc(mode)`	Конфігурує АЦП відповідно до значення константи `mode`: `ADC_C LOCK_DIV_2`, `ADC_CLOCK_DIV_8`, `ADC_CLOCK_DIV_32` - ділення частоти; `ADC_OFF` - АЦП відключений; `ADC_CLOCK_INTERNAL` - частота без дільника
`setup_adc_ports`	`void setup_adc_port (mode)`	Конфігурує виводи АЦП відповідно до значення константи `mode`: - `NO_ANALOGS` - немає аналогових входів; - `ALL_ANALOG - A0, A1, A2, A3, A5, E0, E1, E2;` опорна напруга - на виводі Vdd; - `ANALOG_RA3_REF - A0, A1, A2, A5, E0, E1, E2;` опорна напруга - на виводі `A3`; - `A_ANALOG - A0, A1, A2, A3, A5;` опорна напруга - на виводі Vdd;

Функція	Бібліотека	Заголовок
		- A_ANALOG_RA3_REF - A0, A1, A2, A5; опорна напруга - на виводі A3; - RA0_RA1_RA3_ANALOG - A0, A1, A3; опорна напруга - на виводі Vdd; - RA0_RA1_ANALOG_RA3_REF - A0, A1; опорна напруга - на виводі A3; - ANALOG_RA3_RA2_REF - A0, A1, A5, E0, E1, E2; опорна напруга - на виводах A2, A3; - ANALOG_NOT_RE1_RE2 - A0, A1, A2, A3, A5, E0; опорна напруга - на виводі Vdd; - ANALOG_NOT_RE1_RE2_REF_RA3 - A0, A1, A2, A5, E0; опорна напруга - на виводі A3; - ANALOG_NOT_RE1_RE2_REF_RA3_ RA2 - A0, A1, A5; опорна напруга - на виводах A2, A3; - A_ANALOG_RA3_RA2_REF - A0, A1, A5; опорна напруга - на виводах A2, A3; - RA0_RA1_ANALOG_RA3_RA2_REF - A0, A1; опорна напруга - на виводах A2, A3; - RA0_ANALOG - A0; - RA0_ANALOG_RA3_RA2_REF - A0; опорна напруга - на виводах A2, A3
setup_comparator	void setup_comparator (mode)	Конфігурує аналоговий компаратор відповідно до значення константи mode (елементи представлених нижче констант відповідають входам C1, C1+, C2 і C2+): - A0_A3_A1_A2; - A0_A2_A1_A2; - NC_NC_A1_A2; - NC_NC_NC_NC; - A0_VR_A1_VR;

Функція	Бібліотека	Заголовок
		- A3_VR_A2_VR; - A3_A2_A1_A2; - A0_A2_A1_A2_OUT_ON_A3_A4

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 программаторов) є 8-бітний HEX-формат, запропонований корпорацією Intel.

HEX-файл (Example.hex) генерується при асемблюванні. Інформація в ньому представляється у наступному вигляді:

```
:10000000FF308600831686018312A001A101A00B98
:0A0010000728A10B072886030728860307288603072824
:02400E00F13F80
:00000001FF\
```

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

У табл. 4.23 описано призначення позицій рядків.

Таблиця 4.23 – 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)

Продовження таблиці 4.23

Байт	**Призначення**
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 шістнадцятирічного 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.

ТЕСТОВІ ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1) Схеми включення світлодіодів у LED - дисплеї.

a) Послідовно

b) Паралельно

c) Паралельно - послідовно

d) Із загальним анодом

e) Порозрядно

2) 1 – розрядний LED - дисплей має 9 виводів. 3 - розрядний LED - дисплей має виводів?

a) 27

b) 25

c) 12

d) 15

e) 11

3) 3 – розрядний LED - дисплей із загальним катодом має виводів?

a) 11

b) 13

c) 14

d) 15

e) 16

4) 3 – розрядний LED - дисплей із загальним анодом має виводів?

a) 11

b) 13

c) 14

d) 15

e) 16

5) Призначення ШІМ.

a) Створювати шум

b) Демпфірувати шум

c) Інтерфейсна модуляція

d) Модулювати аналоговий сигнал

e) Управляти навантаженням

6) Яка залежність існує між скважністю імпульсів і формованою напругою в інтегруючому ланцюзі?

a) Логарифмічна

b) Квадратурна

c) Білінійна

d) Лінійна

e) Уніполярна

7) Задати напрямок обертання двигуна постійного струму в мостовій схемі - вправо.

a) Відкрити верхні ключі

b) Відкрити нижні ключі

c) Відкрити лівий верхній і правий нижній ключ

d) Відкрити правий нижній і верхній ключ

e) Відкрити всі ключі

8) Призначення шини I^2C.

a) Послідовна передача даних

b) Паралельний прийом даних

c) Паралельно-послідовна передача даних

d) Паралельно-послідовний прийом даних

e) Паралельно-лінійна передача даних

9) Кількість провідників у шині I^2C.

a) 1

b) 2

c) 3

d) 4

e) 8

10) Відношення обладнань у концепції шини I^2C.

a) Master -Slave

b) Slave-SubSlave

c) Master- SubSlave

d) Slave -Slave

e) Master-Slave- Master

11) Ключові елементи шини I^2C включені за схемою:

a) Загальний емітер

b) Загальний колектор

c) Загальна база

d) Загальний істок

e) Загальний сток

12) Біти даних у шині I^2C передаються по провіднику:

a) SCL

b) SDA

c) SDB

d) DSA

e) CLS

13) Кількість таймерів у мікроконтролера PIC18F252

a) 1

b) 2

c) 3

d) 4

e) 6

14) Кількість CCP – модулів у мікроконтролера PIC18F252

a) 1

b) 2

c) 3

d) 4

e) 5

15) Кількість USB – модулів у мікроконтролера PIC18F252

a) 1

b) 2

c) 3

d) 4

e) 0

16) Кількість SPI – модулів у мікроконтролера PIC18F252

a) 1

b) 2

c) 3

d) 4

e) 5

17) Кількість I^2C – модулів у мікроконтролера PIC18F252

a) 1

b) 2

c) 3

d) 4

e) 5

18) Біти синхронізації в шині I^2C передаються по провіднику:

a) SCL

b) SDA

c) SDB

d) DSA

e) CLS

19) Кількість зовнішніх переривань типу INT у мікроконтролері PIC18F252

a) 1

b) 2

c) 3

d) 4

e) 5

20) Ініціалізація таймера RTCC здійснюється командою

a) setup_timer_4(RTCC_INTERNAL|RTCC_DIV_8);

b) setup_timer_3(RTCC_INTERNAL|RTCC_DIV_8);

c) setup_timer_2(RTCC_INTERNAL|RTCC_DIV_8);

d) setup_timer_1(RTCC_INTERNAL|RTCC_DIV_8);

e) setup_timer_0 (RTCC_INTERNAL|RTCC_DIV_8);

21) Дозвіл роботи переривання від таймера RTCC здійснюється командою

a) enable_interrupts (INT_RTCC);

b) enable_interrupts(EXT_RTCC);

c) enable_interrupts(RTCC_INT);

d) enable_interrupts(RTCC_EXT);

e) enable_interrupts(INT_RTTC);

22) Функція i2c_write (*data*) виконує

a) запис байту

b) запис біта

c) приймання байту

d) приймання біта

e) скидання байту

23) Модуль ШІМ працює з таймером:

a) timer_0

b) timer_1

c) timer_2

d) timer_3

e) timer_4

24) Функція i2c_start() виконує

a) стартову посилку

b) стопову посилку

c) передачу даних

d) приймання даних

e) кінець процедури приймання/передачі

25) Функція i2c_stop() виконує

a) стартову посилку

b) стопову посилку

c) передачу даних

d) приймання даних

e) кінець процедури приймання/передачі

26) Функція bit_clear(*var*, *bit*) виконує

a) скидання біта

b) установка біта

c) читання біта

d) скидання байту

e) установка байту

27) Функція bit_set(***var***, ***bit***) виконує

a) скидання біта

b) установка біта

c) читання біта

d) скидання байту

e) установка байту

28) Функція bit_test (*var*, *bit*) виконує

a) скидання біта

b) установка біта

c) перевірка біта

d) скидання байту

e) установка байту

29) Функція delay_ms (*time*) виконує

a) затримка виконання більше 1 мкс

b) затримка виконання більше 10 мкс

c) затримка виконання більше 100 мкс

d) затримка виконання більше 1000 мкс

e) затримка виконання більше 10000 мкс

30) Функція delay_us (*time*) виконує

a) затримка виконання більше 1 мкс

b) затримка виконання більше 10 мкс

c) затримка виконання більше 100 мкс

d) затримка виконання більше 1000 мкс

e) затримка виконання більше 10000 мкс

31) Функція value = input_x() виконує

a) приймання біта

b) передачу біта

c) приймання байту

d) передачу байту

e) скидання байту

32) Функція i2c_write (*data*) виконує

a) запис байту

b) запис біта

c) приймання байту

d) приймання біта

e) скидання байту

33) Функція i8 = MAKE8(*var*, *offset*) виконує

a) об'єднати дані і адресу

b) роз'єднати дані і адресу

c) з'єднати декілька байт в одну змінну

d) зі змінної одержати один байт

e) записати байт у змінну

34) Функція i16 = MAKE16(*varhigh*, *varlow*) виконує

a) об'єднати дані і адресу

b) роз'єднати дані і адресу

c) з'єднати декілька байт в одну змінну

d) зі змінної одержати один байт

e) зі змінної одержати два байти

35) Функція i32 = MAKE32(*var1*, *var2*, *var3*, *var4*) виконує

a) об'єднати дані і адресу

b) роз'єднати дані і адресу

c) з'єднати декілька байт в одну змінну

d) зі змінної одержати один байт

e) зі змінної одержати два байти

36) Функція i3c_stop() виконує

a) стартову посилку

b) нічого

c) передачу даних

d) приймання даних

e) кінець процедури приймання/передачі

37) Функція puts (*string*) виконує

a) Варіант відповіді:

b) запис байту

c) запис біта

d) запис рядка

e) приймання біта

f) скидання рядка

38) Функція Функція data = i2c_read() виконує

a) запис байту

b) запис біта

c) запис рядка

d) приймання біта

e) приймання байту

Варіанти відповідей

1	d.	**14**	b.	**27**	b.
2	e.	**15**	e.	**28**	c.
3	a.	**16**	a.	**29**	d.
4	a.	**17**	a.	**30**	a.
5	e.	**18**	a.	**31**	c.
6	d.	**19**	c.	**32**	a.
7	c.	**20**	e.	**33**	d.
8	a.	**21**	a.	**34**	c.
9	b.	**22**	a.	**35**	c.
10	a.	**23**	d.	**36**	b.
11	d.	**24**	a.	**37**	c.
12	b.	**25**	e.	**38**	e.
13	d.	**26**	a.		

ТЕМИ РЕФЕРАТІВ

Мікроконтролери.

1) дати короткий огляд основних характеристик мікроконтролерів;
2) дати опис середовища розробки програм PSW CSS;
3) дати короткі рекомендації по написанню програми, компіляції та програмування контролера через програматор;
4) висновки.

АЦП мікроконтролера.

1) дати короткий огляд типів і видів АЦП;
2) дати опис принципів роботи квантування за рівнем, дискретизації за часом та залежності якості перетворення від розрядності АЦП;
3) дати короткі рекомендації по використанню переривання АЦП та програмування контролера для роботи з АЦП;
4) висновки.

Датчики.

1) дати короткий огляд сучасних датчиків фізичних величин;
2) дати опис принципів роботи датчиків:

- датчики тиску;
- датчики температури;
- датчики переміщення;
- датчики вологості;
- тензометричні датчики та енкодери;

3) дати короткі рекомендації по застосуванню датчиків;
4) висновки.

ЦАП. Інтерфейс SPI.

1) дати короткий огляд типів і видів ЦАП;
2) дати опис принципів роботи квантування за рівнем, дискретизації за часом та залежності якості перетворення від розрядності ЦАП;
3) дати короткі рекомендації по програмуванню контролера для роботи з ЦАП по інтерфейсу SPI;
4) висновки.

Семисегментний LED – дисплей.

1) дати короткий огляд семисегментних LED-дисплеїв;
2) дати опис принципів роботи статичної та динамічної індикації;
3) дати короткі рекомендації по схемі підключення LED-дисплея до контролера, кодування сегментів та програмування контролера для роботи з LED-дисплеєм;
4) висновки.

Модуль CCP.

1) дати короткий огляд модуля CCP;
2) дати опис принципів роботи модуля **capture**, модуля **compare** та модуля **PWM**;
3) дати короткі рекомендації вимірюванню тимчасових інтервалів, тривалості імпульсу, скважності та керування навантаженням;
4) висновки.

EEPROM – пам'ять.

1) дати короткий огляд EEPROM – пам'яті;
2) дати опис принципів роботи інтерфейсу I^2C, архітектури Master-Slave та тимчасових діаграм роботи інтерфейсу I^2C;
3) дати короткі рекомендації по програмуванню контролера для роботи з інтерфейсом I^2C;
4) висновки.

Зовнішній таймер з інтерфейсом I^2C.

1) дати короткий огляд зовнішнього таймеру з інтерфейсом I^2C;
2) дати опис принципів роботи інтерфейсу I^2C, адресації таймера, та доступу до модулів таймера;
3) дати короткі рекомендації по програмуванню контролера для роботи з таймером;
4) висновки.

LCD – дисплей.

1) дати короткий огляд LCD-дисплеїв;
2) дати опис принципів LCD-дисплея та інтерфейсу LCD-дисплея;
3) дати короткі рекомендації по програмуванню контролера для роботи з LCD;
4) висновки.

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 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.

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 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. Смірнов В.В., Смірнова Н.В., Пархоменко Ю.М. Архітектура та програмування периферійних інтерфейсних контролерів: підручник. Кропивницький : ЦНТУ, 2020. 278 с.
   URL: http://dspace.kntu.kr.ua/jspui/handle/123456789/10313
2. Сайт кафедри програмування комп'ютерних систем і мереж.
   URL: http://pksm.kntu.kr.ua/DEVELOPMENTS_2.html
3. Положення про критерії оцінювання знань здобувачів вищої освіти в Центральноукраїнському національному технічному університеті. – Кропивницький : ЦНТУ, 2020. – 15 с.
4. Смірнов В.В., Смірнова Н.В. Програмне забезпечення управляючих мікро-ЕОМ : метод. вказ. до виконан. лаб. робіт для студ. ден. та заоч. форми навч. за напрямом підготовки 6.050102 «Комп’ютерна інженерія»; Кіровоградський. нац. техн. ун-т (КНТУ). Кіровоград : КНТУ, 2015. 109 с.
   URL: http://dspace.kntu.kr.ua/jspui/handle/123456789/7600
5. Смірнов В.В., Смірнова Н.В. Програмне забезпечення управляючих мікро-ЕОМ : метод. вказ. до виконан. самост. робіт для студ. ден. та заоч. форми навч. за напрямом підготовки 6.050102 «Комп’ютерна інженерія» ; Кіровоградський. нац. техн. ун-т (КНТУ). – Кіровоград : КНТУ, 2015. – 41 с.
   URL: http://dspace.kntu.kr.ua/jspui/handle/123456789/7601
6. 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.
7. 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.
8. 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.

9. Microchip Technology Inc. MCP4921/4922. 12–Bit DAC with SPI™ Interface / Microchip Technology Inc. Microchip Technology Incorporated, USA (www.microchip.com), 2007. 42 p.
10. 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.
11. ООО «Микро–Чип» Программная реализация I^2C интерфейса (режим ведущего) / ООО «Микро–Чип». М.: ООО «Микро–Чип» (www.microchip.ru), 2001. 8 с.
12. ООО «Микро-Чип» Краткий обзор интерфейса I^2C / ООО «Микро–Чип». М.: ООО «Микро-Чип» (www.microchip.ru), 2001. 7 с.
13. ООО «Микро-Чип» Модуль 10–разрядного АЦП в микроконтроллерах PIC 18CXX2 / ООО «Микро–Чип». М.: ООО «Микро–Чип» (www.microchip.ru), 2001. 10 с.
14. 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.
15. 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.
16. Предко М. PIC-микроконтроллеры: архитектура и программирование / Майкл Предко. Саратов : Профобразование, 2010. 512 с.
17. Предко М. PIC-микроконтроллеры: архитектура и программирование / Майкл Предко. Саратов : Профобразование, 2017. 512 с.
18. Семенов Б.Ю. Шина I^2C в радиотехнических конструкциях / Б.Ю. Семенов; изд. 2-е. доп. М. : СОЛОН-Пресс, 2004. 224 с.
19. Стюарт Болл Р. Аналоговые интерфейсы микроконтроллеров / Стюарт Болл Р.; пер. с англ. С. Щербинин. М. : Додэка XXI, 2007. 362 с.
20. 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.

21. 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.
22. Шпак Ю. А. Программирование на языке C для AVR и PIC микроконтроллеров / Ю. А. Шпак; 2-е изд. К. : «МК-Пресс»; СПб. : «КОРОНА_ВЕК», 2011. 544 с.
23. Хофманн М. Микроконтроллеры для начинающих / М. Хофманн; пер. с нем. СПб. : БХВ-Петербург, 2010. 304 с.
24. 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,
25. Шина управления I^2C. URL: https://cxem.net/comp/comp67.php
26. SPI (Serial Peripheral Interface) // Материал из Национальной библиотеки им. Н.Э. Баумана. URL: https://ru.bmstu.wiki/SPI_(Serial_Peripheral_Interface)
27. Последовательный интерфейс SPI. URL: https://prog-cpp.ru/micro-spi/
28. 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
29. 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
30. 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/
31. 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

32. C keywords. URL: http://en.cppreference.com/w/c/keyword/
33. 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/
34. 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/
35. COM port development tool. URL: https://sites.google.com/site/terminalbpp/
36. Как пользоваться терминальной программой Terminal 1.9b. URL: https://faq.radiofid.ru/knowledge–bases/2/articles/13–kak–polzovatsya–terminalnoj–programmoj–terminal–19b /
37. 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/
38. Proteus (система автоматизированного проектирования). URL: https://ru.wikipedia.org/wiki/Proteus_(система_автоматизированного_проектирования)https://ru.wikipedia.org/wiki/Proteus_(система_автоматизированного_проектирования)/
39. PSpice. URL: https://ru.wikipedia.org/wiki/PSpicehttps://ru.wikipedia.org/wiki/PSpice/
40. Описание цифро-аналогового преобразователя MCP4921. URL: https://studbooks.net/2341863/tehnika/ opisanie_tsifro_analogovogo preobrazovatelya_mcp4921
41. G–NOR Lighting LED TECHNOLOGY / GNT–5631Ax–Bx (TRIPLE DIGIT LED DISPLAY) / G–NOR ELECTRONICS CO.,LTD. – 1 с.
42. Трехразрядный светодиодный цифровой дисплей. URL: https://old.radiodetali.com/td/displ/gnt5631.htm/ GNT–5631
43. I²C. URL: https://ru.wikipedia.org/wiki/I%C2%B2C/ I²C
44. Ермолович А.В. Программи́руемая логи́ческая ма́трица URL: https://bigenc.ru/technology_and_technique/text/3178939

45. 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%D0%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.

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

ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРНИХ СИСТЕМ

Навчальний посібник

Редактор В.В. Смірнов

Технічний редактор В.В. Смірнов

Верстальник Н.В. Смірнова

Видавець

Центральноукраїнський національний технічний університет
25006, м. Кропивницький. пр. Університетський, 8,

E-mail: swckntu@gmail.com