М. М. Биков, Т. В. Грищук, В. В. Ковтун

МІКРОПРОЦЕСОРНІ ЗАСОБИ СИСТЕМ УПРАВЛІННЯ

Міністерство освіти і науки України
Вінницький національний технічний університет

М. М. Биков, Т. В. Грищук, В. В. Ковтун

МІКРОПРОЦЕСОРНІ ЗАСОБИ СИСТЕМ УПРАВЛІННЯ

Лабораторний практикум

Вінниця
ВНТУ
2019

УДК 681.32
Б60

Рекомендовано до друку Вченою радою Вінницького національного технічного університету Міністерства освіти і науки України (протокол № 2 від 04.10.2018 р.)

Рецензенти:
А. Я. Кулик, доктор технічних наук, професор
П. І. Кулаков, доктор технічних наук, професор
С. Г. Кривогубченко, кандидат технічних наук, доцент

Биков, М. М.

Б60 Мікропроцесорні засоби систем управління : лабораторний практикум / Биков М. М., Грищук Т. В., Ковтун В. В. – Вінниця : ВНТУ, 2019. – 120 с.

У лабораторному практикумі наводяться теоретичні відомості, завдання та приклади виконання 7 лабораторних робіт, які формують практичні навички проектування мікропроцесорних систем управління на мовах Асемблер і Сі з використанням програмних середовищ AVR Studio, Atmel Studio, Code Vision AVR, Proteus VSM та аппаратного відлагоджувача STK 500.

УДК 681.32

ЗМІСТ

ВСТУП

Останнім часом найбільшого поширення набули цифрові системи управління, ядром яких є спеціалізовані мікропроцесори у вигляді мікроконтролерів. Постійне вдосконалення та поновлення номенклатури МП засобів та цифрової схемотехніки, жорсткі обмеження на тривалість і вартість розробки вносять додаткові ускладнення в процес проектування цифрових систем управління та автоматики на базі МП пристроїв та цифрових схем. Тому для успішного подолання таких складнощів в інженерній діяльності фахівців спеціальності 151 – «Автоматизація та комп'ютерно-інтегровані технології» в навчальний план цієї спеціальності внесена дисципліна «Мікропроцесорні засоби систем управління» (МПЗСУ). **Мета** дисципліни МПЗСУ – формування у студентів знань з апаратних та програмних засобів МП та цифрової схемотехніки, необхідних для побудови сучасних мікропроцесорних систем автоматики та управління.

Метою лабораторних робіт є практичне закріплення знань студентів і набуття ними навичок використання цифрових систем на базі мікропроцесорів та програмного забезпечення МП техніки при розробці та налагодженні різноманітних автоматичних й автоматизованих систем управління. Задачею проведення лабораторних робіт є виконання студентами на лабораторних стендах та комп'ютерній техніці досліджень з питань проектування та налагодження технічного й програмного забезпечення мікропроцесорних систем управління.

Лабораторні роботи дають практичні навички студентам щодо використання таких апаратних і програмних засобів, як STK 500, AVR Studio, Atmel Studio, Proteus та Code Vision AVR, які дозволяють виконати повний цикл проектування мікропроцесорної системи управління. Цей цикл охоплює створення програм в редакторі мовами Асемблер чи Сі, їх налагоджування, симулювання їх роботи, запис у флеш пам'ять мікроконтролера, моделювання роботи електронної схеми спільно з програмним забезпеченням та розробку друкованої монтажної плати, розроблюваної МПСУ.

В результаті проведення лабораторних занять студент має:

- знати склад сучасних мікропроцесорних комплектів і їх технічні характеристики; архітектуру сучасних МП та принципи їх програмування; засоби кроссасемблерування та налагодження програмних модулів на емуляторах;

- вміти розробляти апаратні засоби МПСУ; розробляти програмне забезпечення МПСУ; здійснювати інтеграцію апаратних і програмних засобів МП систем управління.

ЛАБОРАТОРНА РОБОТА № 1
Знайомство з апаратним налагоджувачем STK500 і програмним середовищем AVR Studio

Мета роботи – вивчення принципів роботи з апаратним стендом і програмним середовищем AVR Studio.

1.1 Порядок виконання роботи

1.1.1 Ознайомитися з улаштуванням стенда STK500 і способами його з'єднання з комп'ютером.

1.1.2 Ознайомитися з принципами розробки і налагоджування програм в середовищі AVR Studio.

1.1.3 Записати та налагодити в середовищі AVR Studio задану програму, завантажити її в стенд STK-500 і виконати, скласти звіт про роботу.

1.2 Теоретичні відомості

1.2.1 Опис стенда STK500

Навчально-налагоджувальний стенд виконаний на базі універсальної налагоджувальної плати STK500 та мікроконтролерів ATmega8515 і ATmega16, що використовуються як цільові.

Зовнішній вигляд стенда STK500 та його компоненти показані на рис. 1.1. Плата STK500 живиться від нестабілізованого джерела напругою 10–12 В (конектор 1). Стабілізація напруги реалізується на платі за допомогою стабілізованого джерела живлення, яке підключається вимикачем 2. Індикація підключення напруги здійснюється індикатором 3 на світлодіоді. Цільові мікроконтролери, що підлягають програмуванню, вставляються в гнізда 4, розташовані на ділянці плати між роз'ємами роширення 5. Пристрій підтримує програмування переважної більшості мікроконтролерів AVR, що випускаються на даний час (для підтримки мікроконтролерів, що випускаються тільки в корпусах TQFP64, наприклад, Atmega128, потрібна плата розширення STK501/502, що під'єднується до STK500 за допомогою роз'ємів роширення 5, а для мікроконтролерів Xmega потрібна плата STK600). Всі порти, наявні в допустимих цільових мікроконтролерах (PortA, PortD, PortE, PortB, PortC) виведені на роз'єми 6. Окрім виводів портів на ці роз'єми виведені лінії живлення GND і VTG. Це дозволяє з'єднати з портами за допогою 10-жильних шлейфів зовнішні пристрої і лінійки світлодіодів 9 і кнопок 10 на самій платі (відповідно роз'єми 7 і 8).

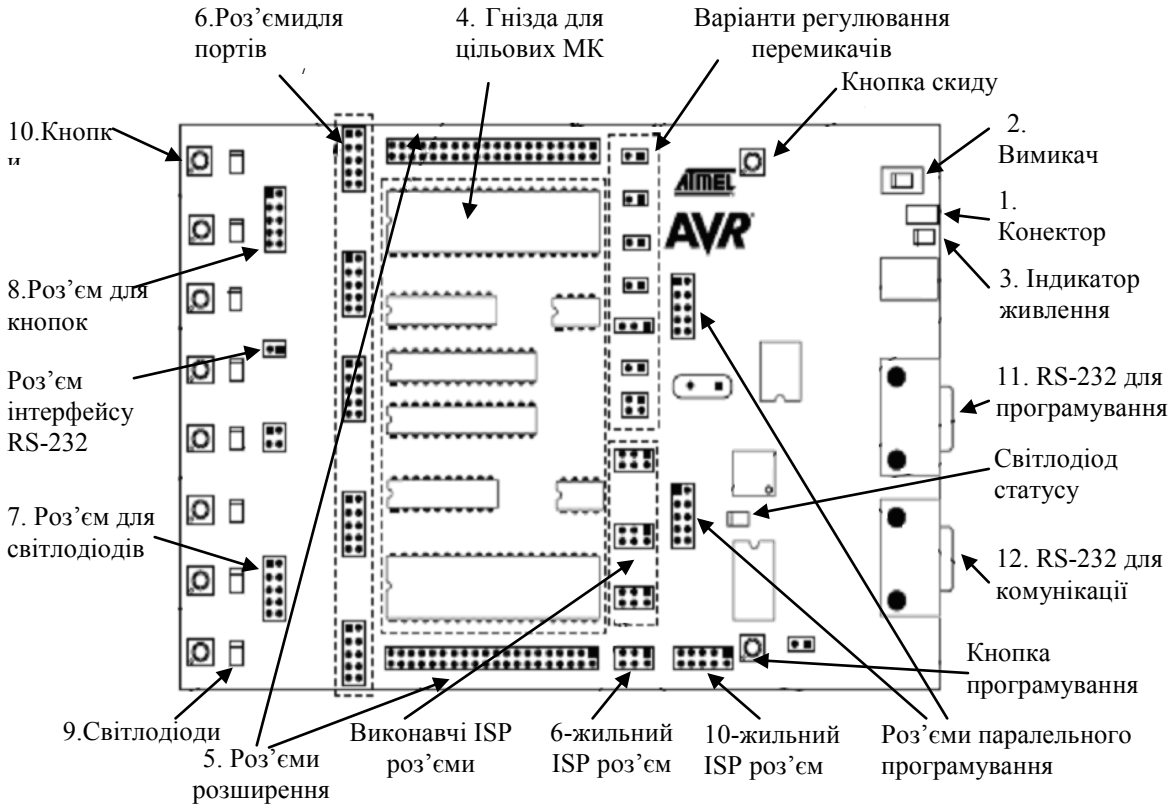


Рисунок 1.1 – Компоненти STK500

Універсальність STK500 полягає в тому, що його можна використовувати як плату налагоджування, як паралельний програматор (для всіх типів мікроконтролерів AVR) і як послідовний внутрісхемний програматор (для мікроконтролерів AVR, що мають режим послідовного внутрісхемного програмування).

Функції внутрісхемного програмування і керування STK500 реалізовані на двох мікроконтролерах: AT90S1200-12SG і AT90S8535-8AC. На платі STK500 змонтовані також: схема оперативного запам'ятовувального пристрою ємністю 2 Мбайти на мікросхемі DataFlash AT45D021; перетворювачі рівнів сигналів (для випадку, коли напруга живлення цільового мікроконтролера відрізняється від напруги живлення мікроконтролерів керування); кероване програмно джерело опорної напруги для внутрішнього АЦП мікроконтролера; інтерфейс RS232 для зв'язку з AVR Studio комп'ютера керування (роз'єм 11); комунікаційний інтерфейс RS232 для зв'язку цільового мікроконтролера з комп'ютером (роз'єм 12).

Для тактування мікроконтролерів, що налагоджуються на платі STK500, передбачені два джерела тактових сигналів.

Одно з них являє собою генератор із кварцовою стабілізацією частоти, побудований на логічних схемах. Користувач має можливість задавати частоту цього генератора, встановлюючи кварцовий резонатор на

необхідну частоту в спеціальну панель CRYSTAL. Друге джерело тактових сигналів являє собою вихід установленого на платі STK500 керівного мікроконтролера AT90S8535-8AC. Частота цього тактового сигналу може бути задана програмно.

Перемикач OSCSEL служить для вибору одного з джерел тактового сигналу для мікроконтролера. Якщо ж за тактовий необхідно використовувати внутрішній RC-генератор мікроконтролера, то зовнішній тактовий сигнал має бути відключений джампером XTAL1.

На платі STK500 не передбачене підключення до цільового мікроконтролера зовнішнього ОЗП.

Керування STK500 відбувається через COM-порт ПК. Схема з'єднання плати STK500 з персональним комп'ютером наведена на рис. 1.2.

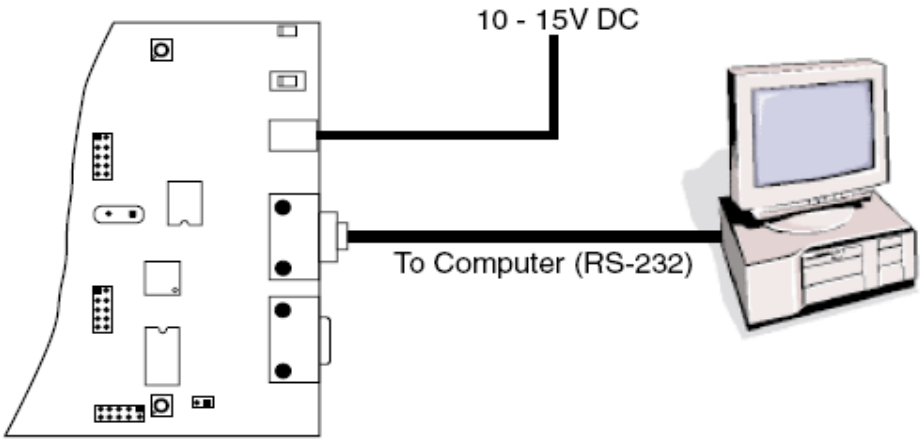


Рисунок 1.2 – Схема підключення стенда STK500 до ПК

Опис роботи світлодіодів. Стартовий набір STK500 містить 8 жовтих світлодіодів. Світлодіоди пов'язані з роз'ємом 7, розташованим на друкованій платі.

У STK500 використовуються транзистор і два резистори (схема рис. 1.3) для підтримки постійної яскравості свічення світлодіодів при будь-якому значенні напруги живлення мікроконтролера (VTG), а також для вимкнення світлодіодів, коли VTG відсутній. Напруга, що подається на світлодіод, лежить в межах [1.8–6] вольт.

Для підключення світлодіодів до порту цільового мікроконтролера| потрібен 10-жильний кабель, яким з'єднуються роз'єм панелі світлодіодів і роз'єм ліній введення/виведення порту.

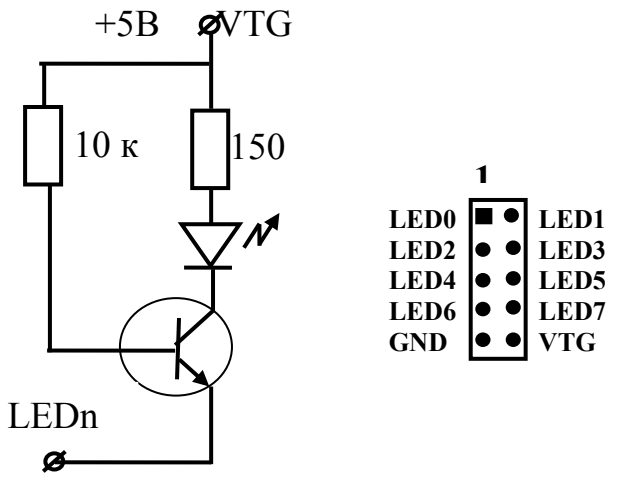


Рисунок 1.3 – Схема під’єднання світлодіодів

Опис роботи кнопок. Схема підключення кнопок до входів портів мікрококонтролера в STK-500 наведена на рис. 1.4.

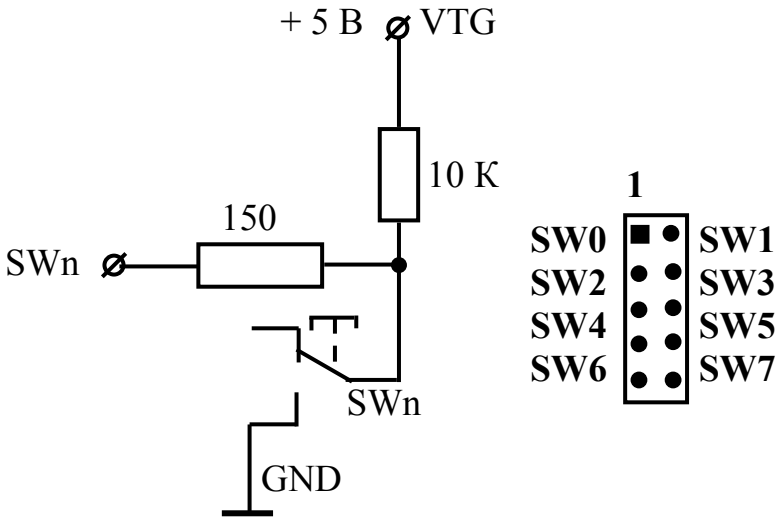


Рисунок 1.4 – Схема підключення кнопки

У схему додані зовнішні підтягувальні резистори 10 кОм| для формування логічної «1» на виводах SWn при ненатиснутому стані кнопок. При натиснутій кнопці на неї надходить нульовий потенціал (логічний «0»). Резистор 150 Ом виконує функцію захисного струмообмеження|, наприклад, у разі помилкового налаштування ліній введення-виведення, пов’язаних з кнопками, на виведення. На лініях портів введення-виведення AVR-мікроконтролерів є можливість активізації внутрішніх вбудованих підтягувальних резисторів до плюса живлення. Ця властивість дозволяє не використовувати в схемах зовнішні підтягувальні резистори.

Використання інтерфейсу RS-232. STK500 містить два роз’єми інтерфейсу RS-232. Один з них використовується для того, щоб

з’єднуватися з AVR Studio. Інший може використовуватися для того, щоб зв’язати цільовий мікроконтролер AVR у гнізді з послідовним портом ПК, пов’язаним з RS-232. Щоб використати RS-232, виводи UART AVR мають бути фізично пов’язані з RS-232.

Роз’єм на платі з двома виводами, позначений «RS232 SPARE», використовується для з’єднання другого роз’єму RS-232 з виводами цільового UART мікроконтролера AVR у слоті. Для цього використайте 2-жильний з’єднувальний шлейф так, як це показано показане на рис. 1.5. Структурна схема з’єднання RS-232 з входом послідовного порту UART показана на рис.º1.6.

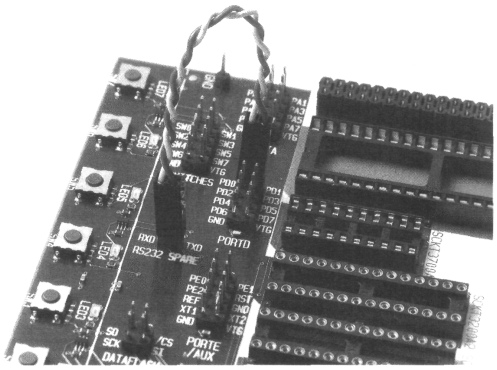


Рисунок 1.5 – З’єднання виводів порту введення/виведення з UART

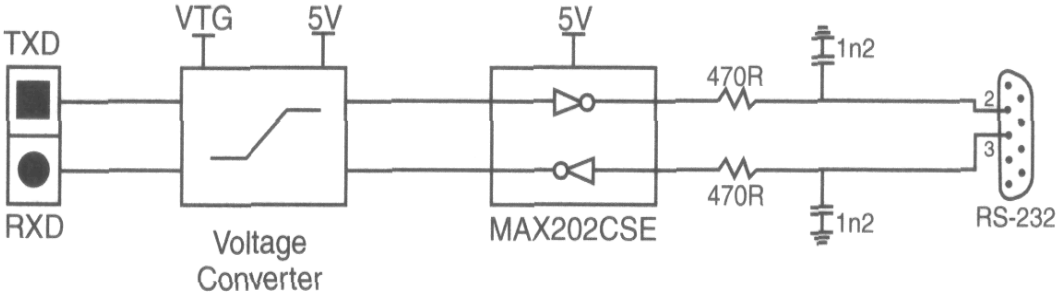


Рисунок 1.6 – Схемне рішення з’єднання виводів UART

1.2.2 Налагодження STK 500 для роботи з AVR Studio

Керування STK500 відбувається через COM-порт персонального комп’ютера. Керівна програма є складовою частиною *AVR Studio* і запускається з меню **Tools -> STK500/AVRISP/JTAGICE.** Крім STK500 ця керівна програма підтримує такі апаратні засоби налагодження, як внутрісхемний программатор ATAVRISP і внутрісхемний емулятор JTAGICE. На рис. 1.7 показано вікно програми, що керує STK500.

Закладка Board служить для встановлення параметрів STK500:

- напруги живлення цільового мікроконтролера **(VTarget),**
- опорної напруги АЦП мікроконтролера (**ARef**),
- частоти керованого джерела тактового сигналу **(Oscillator),**

а також слугує для індикації версії прошивки *(firmware)* керівних мікроконтролерів **(Revision).**

STK500 підтримує різні режими програмування цільових мікроконтролерів: режим внутрісхемного послідовного програмування і два режими програмування з використанням підвищеної програмувальної напруги – паралельний і послідовний. Вибір режиму програмування здійснюється у вікні керування параметрами STK500 при обраній закладці **Program** (рис. 1.7).

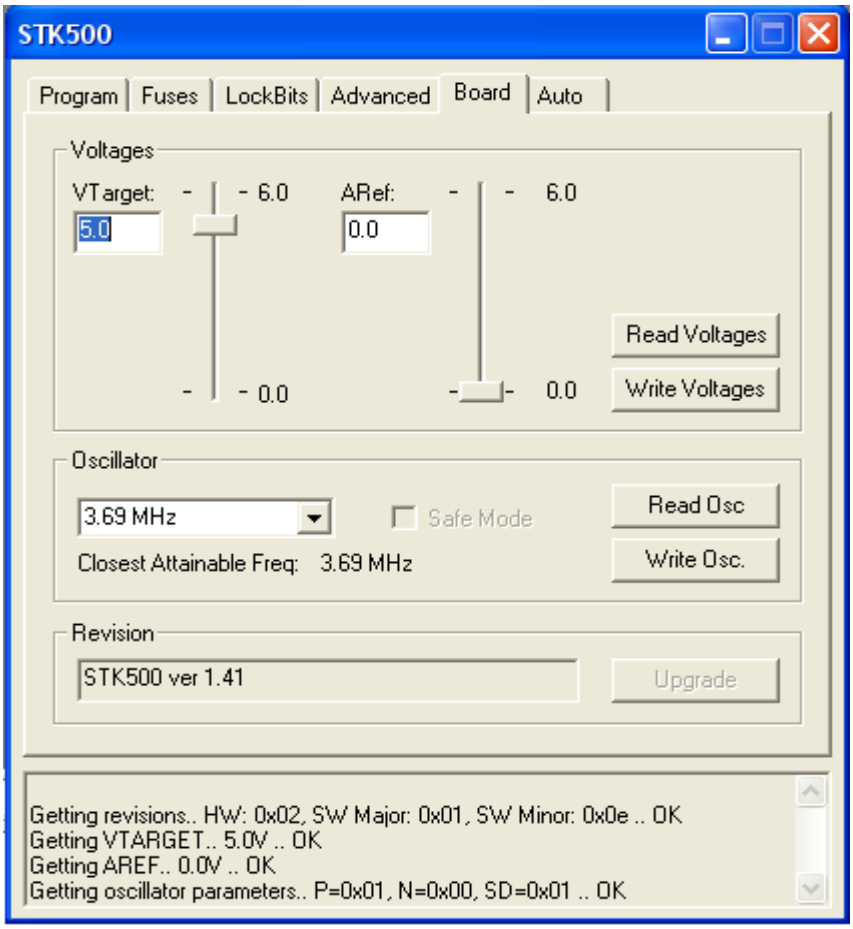


Рисунок 1.7 – Вікно управління параметрами STK500

STK500 також може бути використаний як внутрісхемний програматор для мікроконтролерів, встановлених у цільовому пристрої. Для цього потрібно плоским десятижильним кабелем з'єднати цільовий пристрій з роз'ємом **ISP10PIN** на платі STK500.

Крім того, у *AVR Studio* передбачена можливість відновлення прошивки (*firmware)* пам'яті програм керувальних мікроконтролерів **(Revision Upgrade).** Оновлені версії *firmware* вносять до складу *AVR Studio* як прикладне програмне забезпечення. При запуску керівна програма перевіряє зв'язок COM-порту персонального комп'ютера з STK500 і у випадку його присутності запитує версію прошивки. Для входу в режим перепрограмування потрібно, щоб у момент включення живлення на платі STK500 була натиснута кнопка **Program**.

1.2.3 Порядок створення програм в AVR Studio

AVR Studio — це інтегроване налагоджувальне середовище розробки програм (IDE) для мікроконтролерів сімейств AVR (AT90S, ATmega, ATtiny) фірми *Atmel.*

IDE *AVR Studio* містить:

• транслятор мови Асемблер *(Atmel AVR macroassembler);*

• налагоджувач *(Debugger);*

• програмне забезпечення верхнього рівня для підтримки внутрісхемного програмування (*In-System Programming,* ISP).

Налагоджувач *AVR Studio* підтримує всі типи мікроконтролерів AVR і має два режими роботи: режим програмної симуляції і режим керування різними типами внутрісхемних емуляторів *(In-Circuit Emulators)* виробництва фірми *Atmel.* Важливо відзначити, що інтерфейс користувача не змінюється залежно від обраного режиму налагодження.

Налагоджувальне середовище підтримує виконання програм як у вигляді асемблерного тексту, так і у вигляді вихідного тексту мови С.

Після запуску *AVR Studio* для створення нового проекту необхідно в меню **Project** вибрати команду New. У результаті на екрані з'являється діалогове вікно (рис. 1.8), у якому необхідно ввести назву проекту **(Project name)** і його розташування **(Location).** Новий проект зручніше створювати в окремій папці.

Далі вибирається тип проекту:

• **AVR Assembler**. Використовує вбудований макроасемблер *AVR Studio*;

• **Generic 3rd party C compiler.** Використовує зовнішній компілятор С, що має інтерфейс командного рядка.

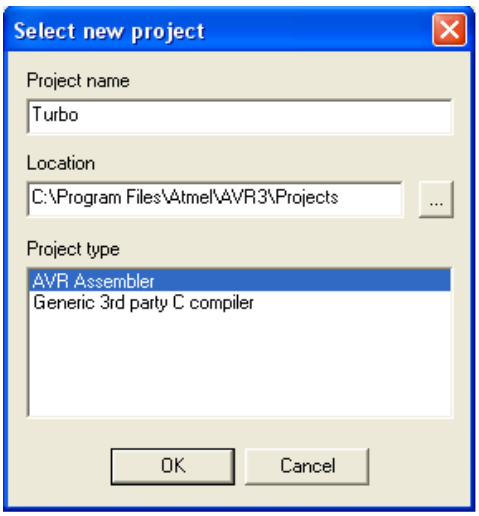


Рисунок 1.8 – Вікно створення нового проекту

При виборі пункту **AVR Assembler** після натискання кнопки **ОК** на екрані з'являється вікно організації проекту (рис. 1.9), що показує всі пов'язані з проектом файли.

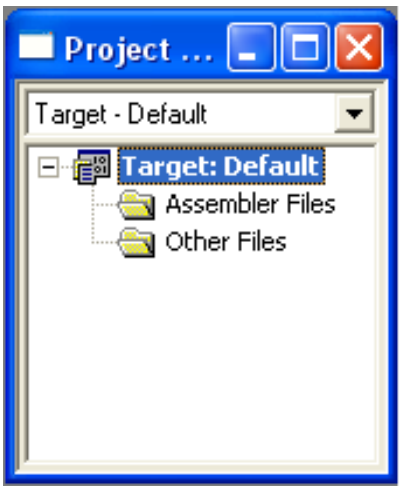


Рисунок 1.9 – Вікно організації проекту

Далі до проекту має бути доданий файл програми мовою Асемблер. Це можна зробити різними способами: або в проект додається вже існуючий файл із розширенням *.asm,* або створюється новий.

Для створення нового файлу необхідно в цьому вікні вибрати групу **Assembler Files** і в меню **Project** вибрати пункт **Add File.** У вікні, що відкрилося, потрібно ввести назву файлу з розширенням *.asm.* Якщо файл був створений раніше, то його необхідно знайти на диску і подвійним клацанням миші занести в рядок «Ім'я файлу».

Створений (або знайдений) у такий спосіб файл буде поміщений у групу **Assembler Files** у вікні організації проекту. Подібним же чином можна внести в проект й інші асемблерні файли, але група **Assembler Files** може містити тільки один файл, з якого надалі буде починатися трансляція проекту. Назвемо цей файл вхідним асемблерним файлом проекту. Значок цього файлу у вікні організатора проекту відзначений червоною стрілкою, спрямованої вправо, всі інші файли проекту будуть відзначені синіми стрілками, спрямованими вниз (рис. 1.10). Усі файли проекту мають бути внесені у вхідний файл проекту за допомогою асемблерної директиви *.include.* Для зміни вхідного файлу проекту на інший треба встановити курсор миші на потрібний файл у вікні організацій проекту і клацнути правою кнопкою миші. У висхідному вікні потрібно вказати цей файл як **Assembler entry file.**

Для того, щоб у тексті асемблерних файлів замість адрес внутрішніх регістрів мікроконтролера використовувати їхні символічні імена, потрібно внести директивою .include до проекту файл, що містить призначення символічних імен усіх регістрів обраного мікроконтролера (звичайно його називають файлом, що вноситься), наприклад *.include «m8515def.inc».* Файли, що вносяться, входять у прикладне програмне забезпечення *AVR Studio* і при інсталяції поміщаються в папку Appnotes у директорії, де встановлений *AVR Studio.*

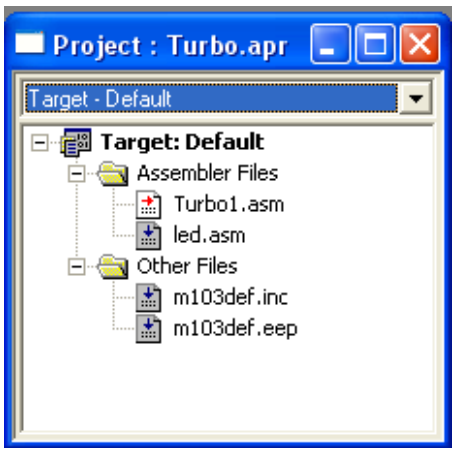


Рисунок 1.10 – Вікно організації проекту
з внесеними файлами проекту

Для редагування вихідного тексту програми необхідно в папці **Assembler Files** у вікні організації проекту відкрити потрібний файл із розширенням *.asm*. У вікно, що відкрилося, редагувати файл можна з клавіатури або через буфер комп'ютера ввести текст програми мовою Асемблер (рис. 1.11).

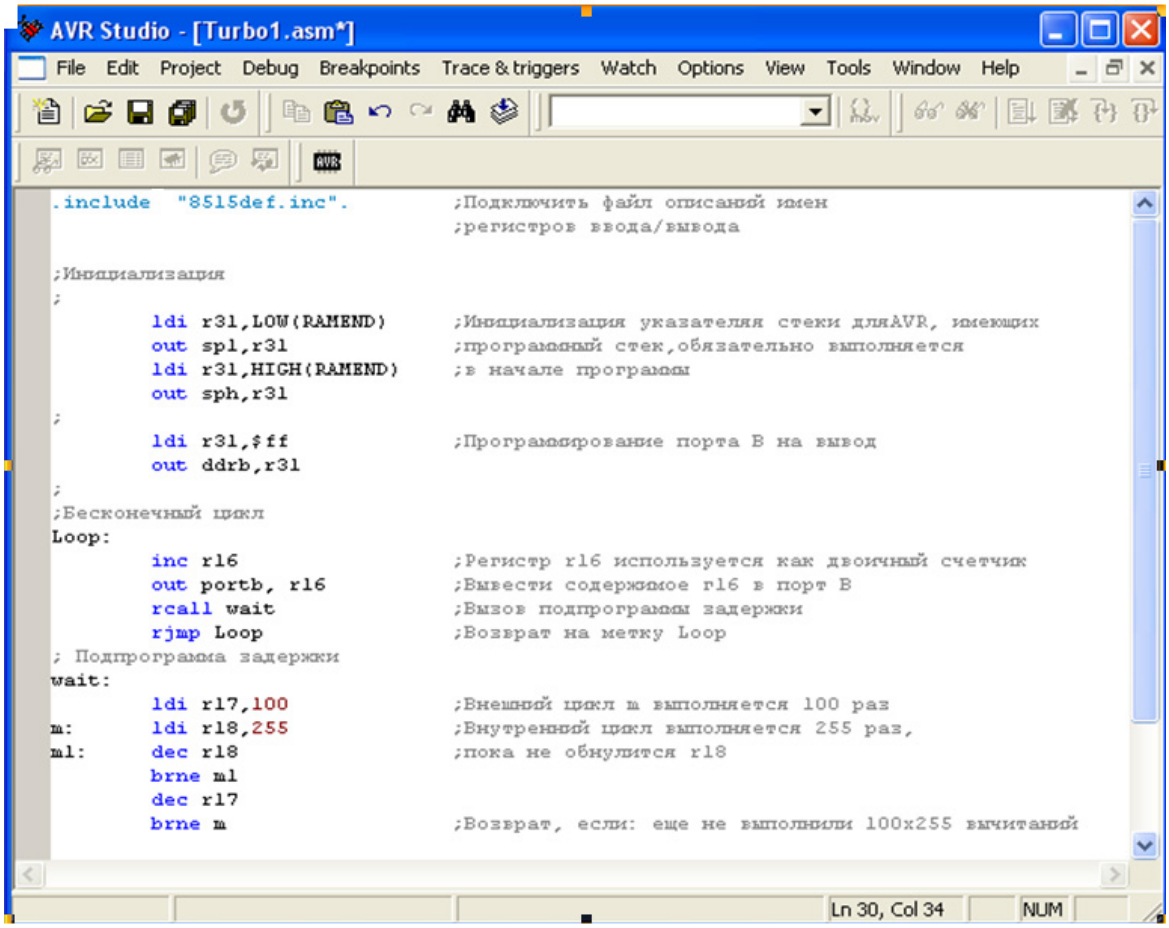


Рисунок 1.11 – Вікно редагування програми мовою Асемблер

Перед трансляцією потрібно задати встановлення проекту. У пункті меню **Project** вибирається **Project Settings,** і у висхідному вікні встановлень проекту вказується необхідний формат вихідного файлу. *AVR Studio* підтримує такі вихідні формати:

- Object;
- Generic;
- Intel Intellec 8/MDS (Intel Hex);
- Motorola S-Record.

Для налагодження в *AVR Studio* необхідний файл у форматі *Object* (об'єктний файл). Однак більшість програматорів як вхідні використовують файли у форматі *Intel Hex.*

Далі здійснюється трансляція програми і перевірка правильності її написання. Вибирається пункт **Assembler** у меню **Project.** Вікно **Project Output** містить повідомлення асемблера. У це вікно виводиться інформація про кількість слів коду і даних, про наявність помилок й інша інформація (рис. 1.12).

Для локалізації помилок трансляції у випадку їхньої наявності можна у вікні повідомлень асемблера встановити курсор миші на повідомлення про помилку і два рази клацнути лівою кнопкою миші. При цьому у вікні редагування вихідного тексту програми курсор буде встановлений на рядок, що викликав повідомлення про помилку, і цей рядок буде виділений кольором.

Рисунок 1.12 – Вікно повідомлень асемблера

У результаті трансляції створюється вихідний файл у зазначеному форматі. Якщо вихідний асемблерний текст містив сегмент енергонезалежних даних (оголошений директивою *.eseg*), то при трансляції буде створений також файл із розширенням *.eep*. Цей файл містить дані для внутрішньої EEPROM мікроконтролера і має той же формат, що і вихідний файл.

Якщо в результаті трансляції не видається повідомлень про помилки, можна приступати до налагоджування проекту.

1.2.4 Налагоджування програм в AVR Studio

Як уже говорилося, *AVR Studio* дозволяє робити налагоджування програм з використанням убудованого програмного симулятора або зовнішнього внутрісхемного емулятора. Коли користувач запускає *AVR Studio,* програма перевіряє наявність емулятора на одному з послідовних портів комп'ютера. Якщо емулятор знайдений, то він вибирається як базова система налагоджування. Якщо емулятор не знайдений, то налагоджування буде виконуватися на вбудованому програмному симуляторі AVR.

Внутрісхемний емулятор дозволяє робити налагоджування проекту на реальній цільовій платі. Необхідно пам'ятати, що в режимі реального часу емулятор працює істотно швидше, ніж програмний симулятор.

Відзначимо ще раз, що, незалежно від режиму роботи налагоджувача, інтерфейс *AVR Studio* не змінюється. При переході від одного режиму налагоджування на інший всі параметри середовища зберігаються. Інформація про поточний стан налагоджувача виводиться в рядку стану *AVR Studio.*

Для запуску налагоджувача необхідно виконати процедуру **Build and run,** що викликається при натисканні на відповідну кнопку на панелі керування. Процедура **Build and run** виконується в два етапи. Спершу відбувається трансляція вхідного асемблерного файлу, при якій, незалежно від установок проекту, крім вихідного файлу заданого формату генерується й об'єктний файл. Потім цей об'єктний файл завантажується у налагоджувач.

Для сумісності з колишніми версіями в *AVR Studio* передбачений ще один варіант запуску налагоджувача — завантаження отриманого в результаті трансляції об'єктного файлу **(File -> Open).** Але при цьому користувач не має можливості редагувати вихідний текст програми безпосередньо у налагоджувачі. Крім того, варто пам'ятати, що для того, щоб транслятор згенерував необхідний для запуску налагоджувача об'єктний файл, потрібно у вікні установок проекту вказати його як формат вихідного файлу транслятора Object.

При першому для проекту запуску налагоджувача викликається вікно вибору цільового мікроконтролера (рис. 1.13). У цьому вікні зі списку вибирається потрібний мікроконтролер **(Device),** наприклад AT90S8515, і тактова частота процесорного ядра **(Frequency).** Опції **Memory і Architecture** при виборі стандартного пристрою у вікні не використовуються. Це ж вікно може бути викликане в процесі роботи налагоджувача **(Options -> Simulator Options).**

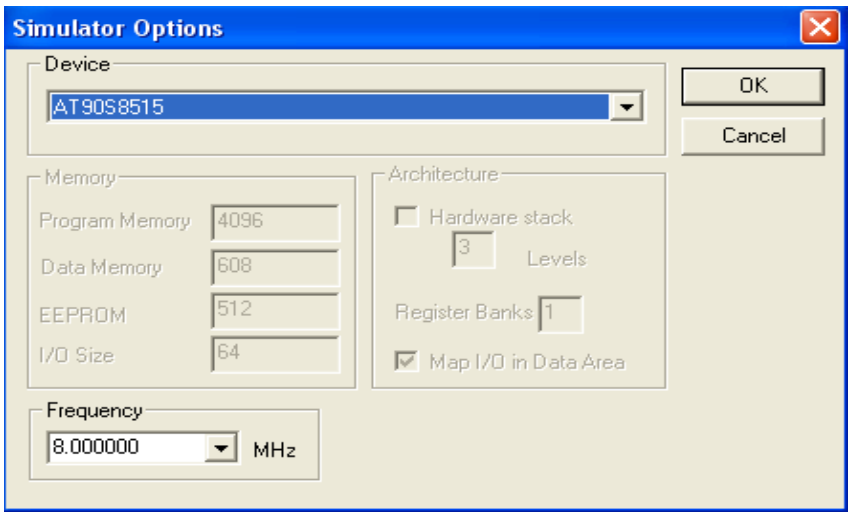


Рисунок 1.13 – Вікно вибору цільового мікроконтролера

Екран *AVR Studio* у режимі налагоджування наведений на рис. 1.14.

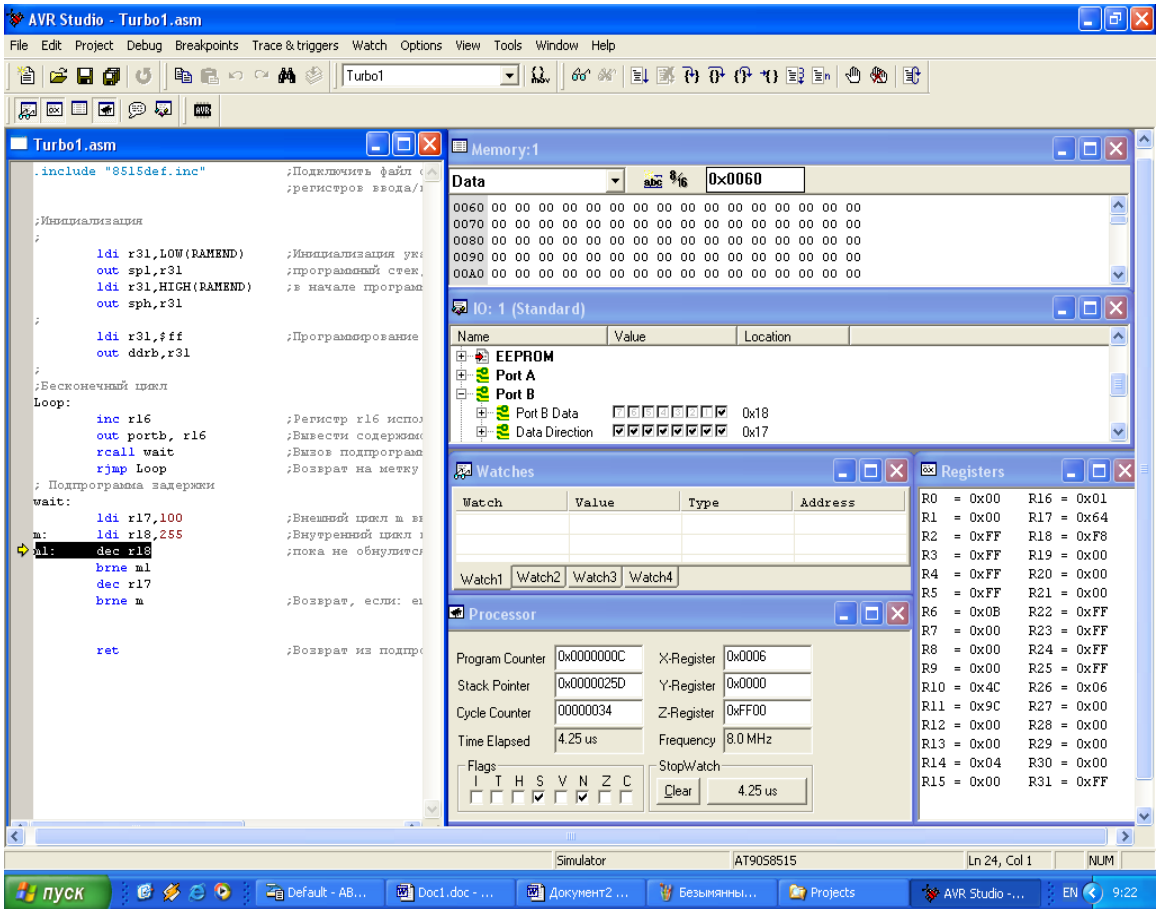


Рисунок 1.14 – Екран *AVR Studio* у режимі налагоджування

Під час виконання процедури **Build and run** (або при завантаженні об’єктного файлу) автоматично відкривається вікно вихідного тексту програми, що виконується мікроконтролером. У вікні відображається код,

що виконується в налагоджувальному оточенні (емуляторі або програмному симуляторі).

Після вибору опцій симулятора в лівому полі вікна асемблерної програми з'являється жовта стрілка, що вказує позицію програмного лічильника мікроконтролера (рис. 1.15). Цей покажчик завжди знаходиться на рядку, що буде виконаний в наступному циклі.

Користувач може виконувати програму повністю у покроковому режимі, трасуючи блоки функцій або виконуючи програму до того місця, де знаходиться курсор. У програмі можна визначати необмежене число точок зупинки, кожна з яких може бути задіяна або незадіяна. Точки зупинок зберігаються між сесіями роботи.

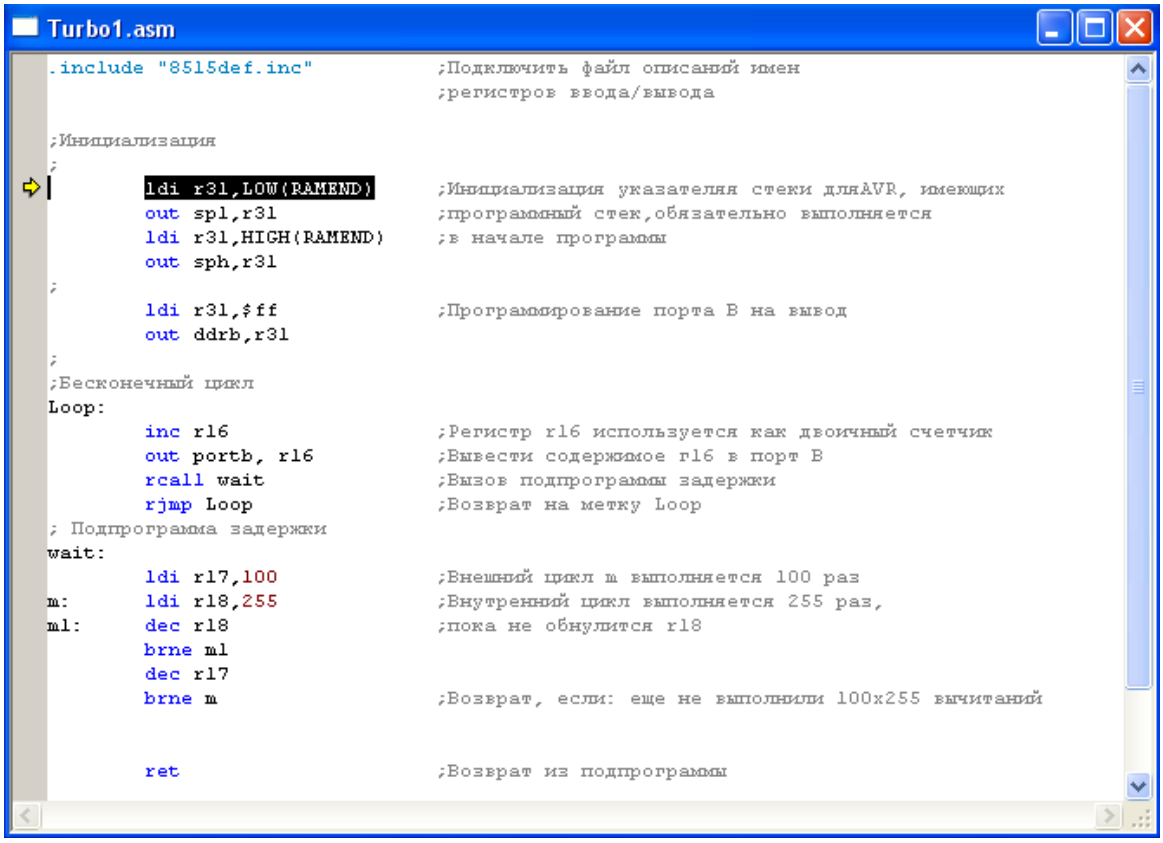


Рисунок 1.15 – Вікно вихідного тексту програми в режимі налагоджування

У *AVR Studio* для налагоджування програми передбачені дві команди покрокового режиму: **Step Over і Trace into.** Різниця між ними в тому, що команда **Step Over** не працює в підпрограмах. За допомогою команд покрокового режиму можна відстежити зміни значень у регістрах пристроїв введення/виведення, пам'яті і реєстрового файлу. До команд крокового режиму відносять також **Auto Step** і **Multi Step.** Вибравши в меню **Debug** пункт **Debug Options,** можна встановити у висхідному вікні

параметри режимів **Auto Step** і **Multi Step**, а також деякі інші опції симулятора, мова про які піде нижче.

Крім покрокового режиму можливе налагоджування програми з використанням точок переривання **(Break points).** Командою **Go** запускається виконання програми. Програма буде виконуватися до зупинки користувачем або до виявлення точки переривання.

Для встановлення точки переривання в *AVR Studio* служить пункт меню **Breakpoints -> Toggle Breakpoint.** Точка переривання ставиться в рядку, відзначеному курсором (рис. 1.16).

Червона відмітка у лівому полі вікна вихідного тексту програми показує встановлену точку переривання.

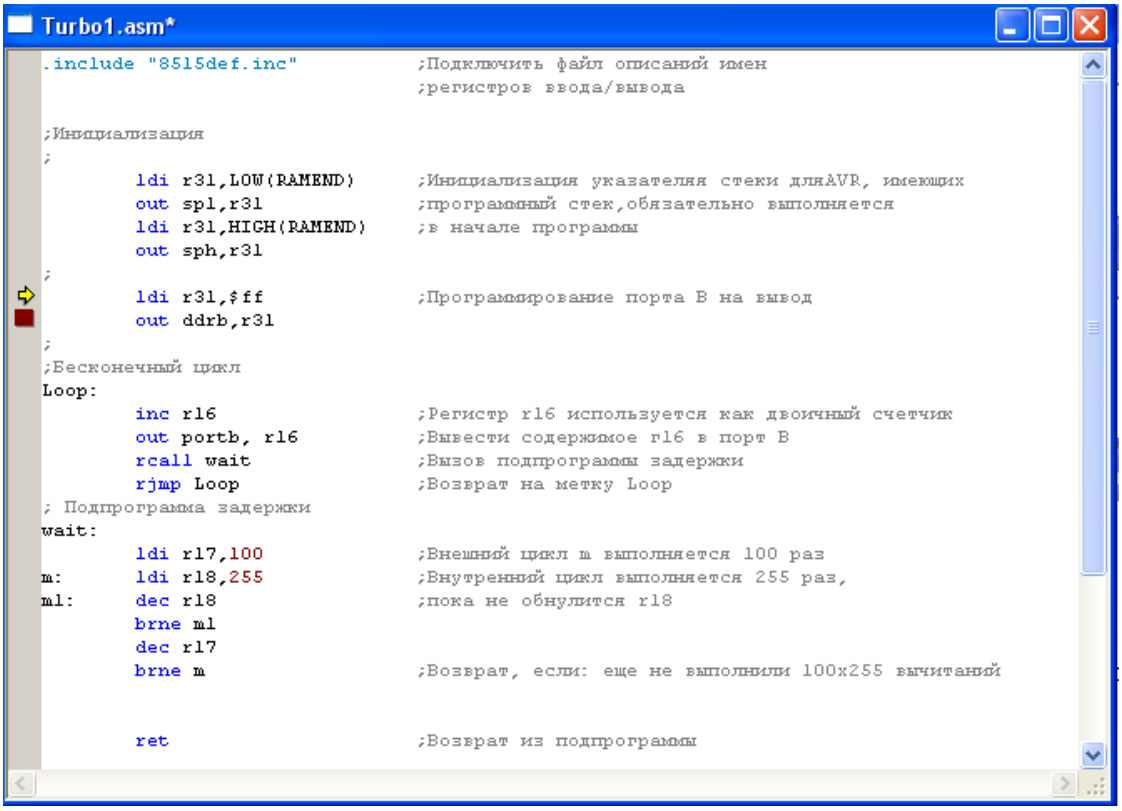


Рисунок 1.16 – Точка переривання у вікні вихідного тексту програми в режимі налагоджування

У процесі відлагодження також можна вибрати пункт меню **Debug -> Run To Cursor.** При виборі цього пункту код, що виконується, буде виконуватись до досягнення команди, позначеної курсором. При цьому, якщо налагоджувач виявляє точку переривання, встановлену раніше положення курсору, то переривання буде виконано тільки у випадку його дозволу у вікні **Debug Option,** у противному випадку виконання не припиняється. Якщо команда, позначена курсором, не досягається,

налагоджувач продовжує виконувати код програми доти, поки виконання не буде перервано користувачем. Оскільки режим **Run To Cursor** залежить від позиції курсору, він доступний тільки при активному вікні вихідного тексту.

Для переривання виконання програми користувачем служить команда **Break.** У стані переривання ця команда недоступна.

При налагоджуванні з використанням точок переривання або якщо адреса переривання зазначена курсором у вікні вихідного тексту, модифікація інформації в усіх вікнах відбувається тільки при досягненні точки переривання (або при припиненні виконання програми користувачем). Пункт меню **Debug -> Reset** виконує скидання мікроконтролера. Якщо програма при цьому виконується, то її виконання буде зупинено чи призупинено. Після скидання інформація в усіх вікнах модифікується.

Для спостереження за роботою програми можна відкрити кілька вікон, що відображають стан різних вузлів мікроконтролера. Вікна відкриваються натисканням відповідних кнопок на панелі інструментів або при виборі відповідного пункту меню **View.**

Регістровий файл мікроконтролера AVR відображається у вікні **Registers** (рис. 1.17). Якщо в процесі виконання програми в черговому циклі значення якого-небудь регістра зміниться, то цей регістр буде виділений червоним кольором. Таке ж колірне виділення реалізоване у вікнах пристроїв вв/вив, пам’яті і змінних.

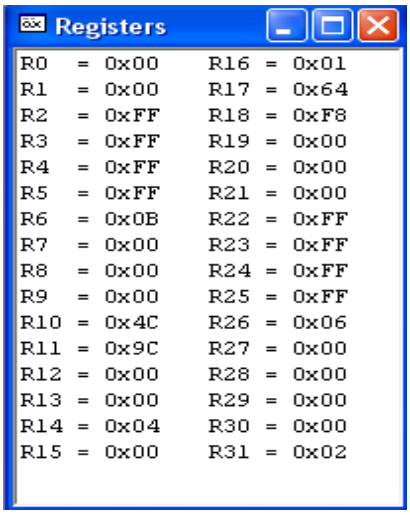


Рисунок 1.17 – Вікно стану регістрового файлу

Стан убудованих периферійних пристроїв мікроконтролера відображено у вікні **I/O Window** (рис. 1.18).

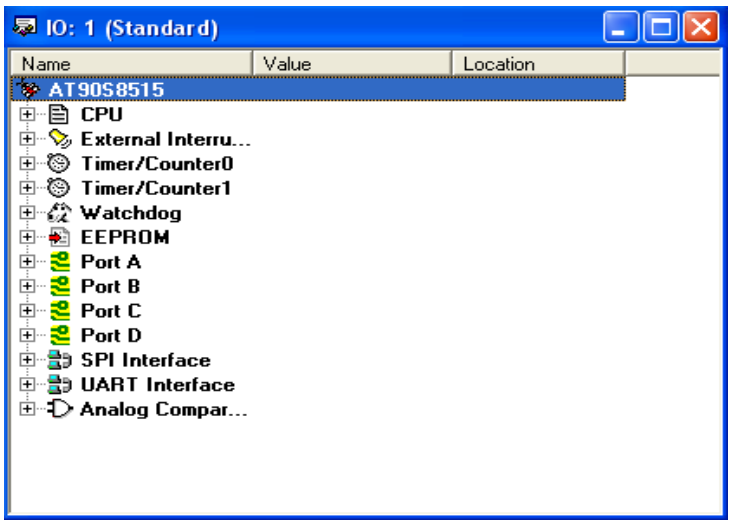


Рисунок 1.18 – Вікно стану пристроїв введення/виведення

У цьому вікні відображені всі функціональні блоки мікроконтролера. Будь-який блок може бути розкритий натисканням на його значок. При розкритті блока у вікні відображаються адреси і стани всіх його регістрів і окремих, доступних для модифікації, бітів (рис. 1.19). Кожен доступний для модифікації біт може бути встановлений або скинутий як програмою в ході її виконання, так і користувачем вручну (клацнувши лівою кнопкою миші на потрібному біті, користувач може змінити значення біта на зворотне) – у режимі програмної симуляції це є способом імітації вхідного впливу на мікроконтролер.

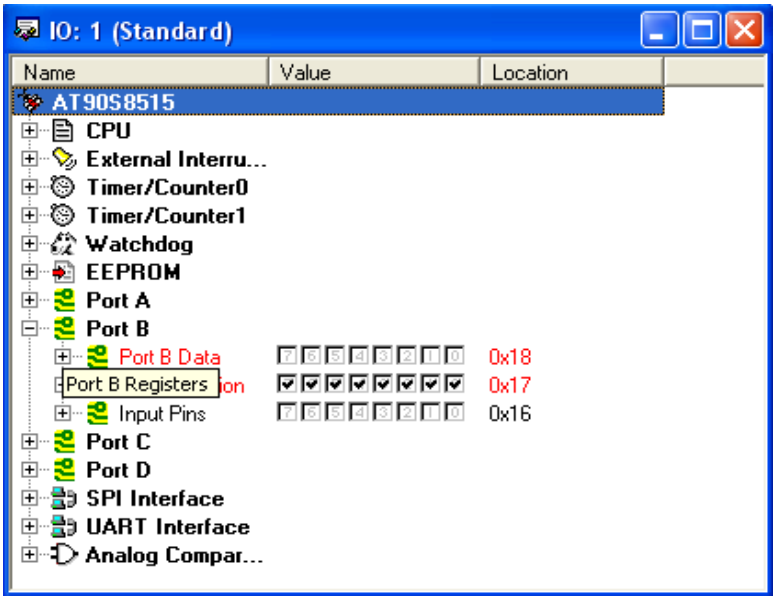


Рисунок 1.19 – Розгорнутий порт PORTB у вікні пристроїв введення/виведення

Іншим способом задання вхідного впливу на мікроконтролер у режимі симулятора є використання зовнішніх файлів вхідних впливів. Формат файлу вхідного впливу дуже простий:

000000000:00
000000039:01
000000040:00
9999999999:FF

Тут значення, зазначене після роздільника «:», – це шіснадцяткове подання сигналів, що впливають на порт мікроконтролера. Значення, зазначене до роздільника, – це десятковий номер циклу (з моменту скидання мікроконтролера), у якому зазначений вплив надходить на виходи порту мікроконтролера. Файл вхідного впливу має закінчуватися рядком з завідомо великим номером циклу, в іншому випадку буде видане повідомлення про помилку. Для задіяння файлу вхідного впливу служить пункт меню **Options -> Simulator Port Stimuli.** У висхідному вікні потрібно вказати порт мікроконтролера, на який потрібно подавати вплив, і файл цього впливу. Користувач може створювати файли впливів, записувати зміни значень на виходах портів мікроконтролера у файл (формат цього файлу той же, що й файлу вхідних впливів). Для запису служить пункт меню **Options-> Simulator Port Logging.** У висхідному вікні потрібно вказати порт мікроконтролера й ім'я файлу для запису. Записуваний файл буде видалятися і створюватися знову при кожному скиданні мікроконтролера **(Debug -> Reset). К**ористувач має сам підключати файл вхідного впливу або задавати ім'я файлу для запису при кожному запуску симулятора.

Для спостереження за змінами змінних призначене вікно **Watch. З**мінні, визначені в програмі, можуть бути відображені в цьому вікні. Якщо в процесі виконання програми значення цих змінних будуть змінюватися, то всі зміни можна буде спостерігати в цьому вікні (рис. 1.20).

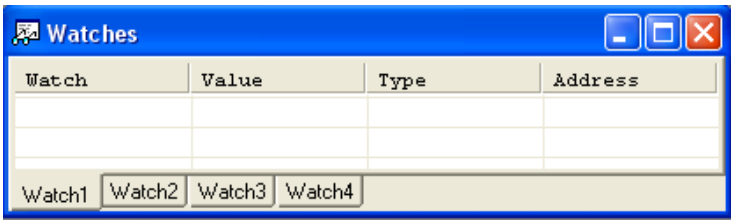


Рисунок 1.20 – Вікно перегляду змінних

Для індикації стану програмного лічильника, покажчика стека, вмісту регістра статусу SREG і індексних регістрів X, Y і Z у процесі налагоджування програми призначене вікно **Processor** (рис. 1.21).

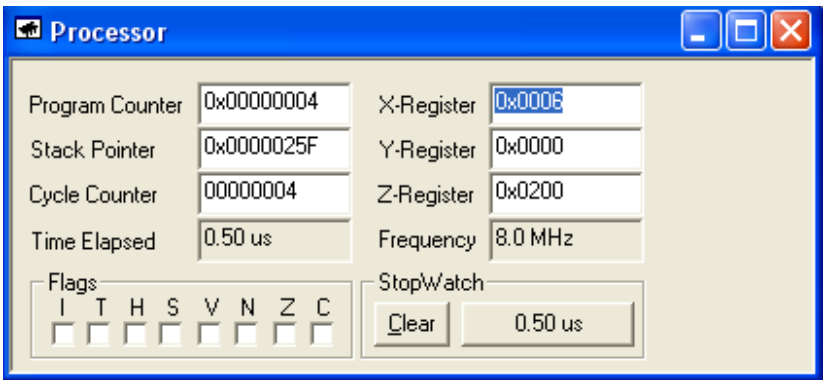


Рисунок 1.21 – Вікно стану процесорного ядра

У цьому ж вікні відображається поточний час виконання програми і тактова частота ядра мікроконтролера.

Перегляд комірок пам'яті програм, пам'яті даних, EEPROM і регістрів портів вв/вив в ході виконання програми можливий також за допомогою діалогового вікна **Memory.** Низхідне меню діалогового вікна дозволяє вибрати один з чотирьох масивів комірок пам'яті: Data, IO, Eeprom, Program Memory. Для одночасного перегляду декількох областей вікно Memory може бути відкрито кілька разів. Інформація в діалоговому вікні може бути подана у виді байтів або у вигляді слів у шіснадцятковій системі числення, а також у вигляді ASCII-символів (рис. 1.22).

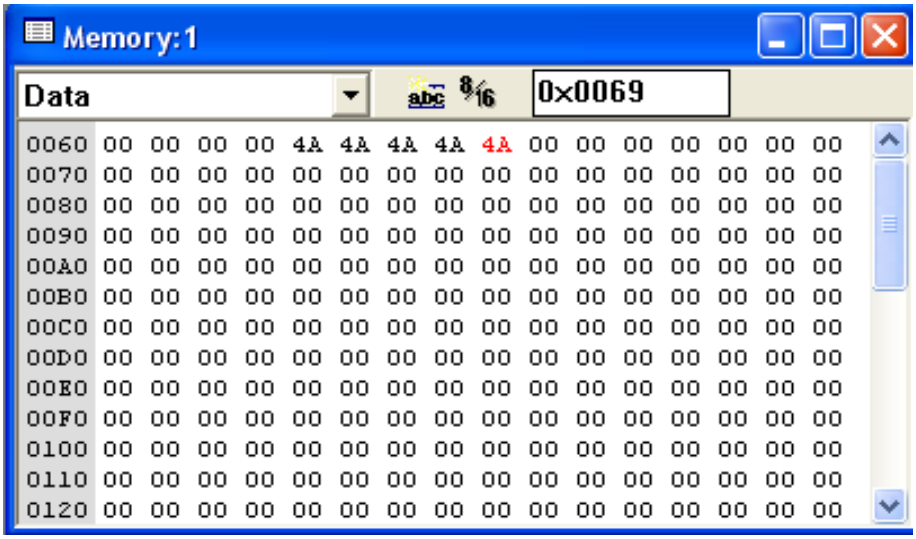


Рисунок 1.22 – Вікно перегляду вмісту пам'яті

У процесі налагоджування користувач може ініціалізувати внутрішній ОЗП або EEPROM мікроконтролера (наприклад, даними, що утримуються в отриманому при трансляції файлі *.eep),* або зберегти вміст ОЗП і EEPROM у вигляді файлів у форматі *Intel Hex.* Для цього служить пункт меню **File -> Up/Download Memory.**

Для внесення змін у програму в процесі налагоджування необхідно редагувати її вихідний текст. При спробі запуску симулятора на виконання програми після редагувань на екрані зэ'являється вікно, що повідомляє про зміну програми та необхідність її компіляції.

Для збереження проекту потрібно скористатися пунктом меню **Project -> Close.** При закритті проекту зберігаються всі його налаштування. Під час наступного завантаження налаштування будуть автоматично відновлені.

Працюючи з програмним симулятором пакета *AVR Studio,* варто пам'ятати, що він поки не підтримує деякі режими роботи мікроконтролерів AVR і їхні периферійні вузли:

- аналого-цифровий перетворювач;
- аналоговий компаратор;
- режим реального часу;
- режим зниженого енергоспоживання (інструкція «sleep» інтерпретується програмним симулятором як «nop»);

Як уже говорилося, крім програмного симулятора IDE *AVR Studio* містить у собі програмне забезпечення верхнього рівня для керування апаратними засобами підтримки розробок. Меню Tools містить команди запуску керівних програм.

1.3 Приклад виконання роботи AVR Studio

1.3.1 Завдання. Записати, ввести і налагодити в середовищі AVR Studio програму керування світлодіодами на платі STK500.

Хід виконання лабораторної роботи. Наведена нижче програма дає можливість ознайомитися з загальною структурою побудови проекту, написаного асемблером, звертає увагу на об'язкову ініціалізацію вказівника стека на початку програми, демонструє способи організації циклів і виклику підпрограм. Згідно з наведеною на рис. 1.3 схемою підключення світлодіодів, ввімкненому стану світлодіода має відповідати логічний 0 на виходах AVR мікроконтролера. Програма генерує на виходах порту B ступінчату двійкову послідовність. Для того, щоб спостерігати цю послідовність візуально, в програмі вводится затримка.

Текст програми керування світлодіодами

```
.include «m8515def.inc»          ; підключення файлу опису імен регістрів вв\вив
;Ініціалізація
ldi r31, LOW(RAMEND)         ; Ініціалізація вказівника стека для AVR,  що мають
out spl, r31          ;програмний стек, обов'язково виконується на початку програми
ldi r31, HIGH(RAMEND)
```

```
out sph, r31
ldi r31, $ff                ; Підготування порту B до виведення інформаціїї
out ddrb, r31
ldi r16, $00                ;Скидання регістра ступінчатої послідовності в 0
;Нескінченний цикл
Loop:
inc r16                     ; Регістр r16 використовується як двійковий лічильник
out portb, r16              ; Вивести вміст r16 в порт B
rcall wait                  ; Виклик підпрограми затримки
rjmp Loop                   ; Повернення на мітку Loop
;Підпрограма затримки
wait:
ldi r17, 100
m: ldi r18, 255             ; Зовнішній цикл m виконується 100 разів
m1: dec r18                 ; Внутрішній цикл m1 виконується 255 раз
brne m1                     ; поки не обнулиться r18
dec r17
brne m                      ; Перехід, якщо ще не виконались 100x255 віднімань
ret                         ; Вихід з підпрограми
```

1.3.2 Порядок завантаження програми в стенд

1. Приєднайте блок з 8 світлодіодів до STK500 плоским кабелем на порт B (див. рис. 1.1).

2. Приєднайте STK500 до ПК і блока живлення (див. рис. 1.2).

3. На персональному комп'ютері завантажити AVRStudio: кнопки меню «Пуск->Atmel AVR Tools->AVRStudio».

4. Створити новий проект, кнопки «Project->New Project». Ввести| ім'я проекту, натиснути| Finish (рис.1.23, 1.24).

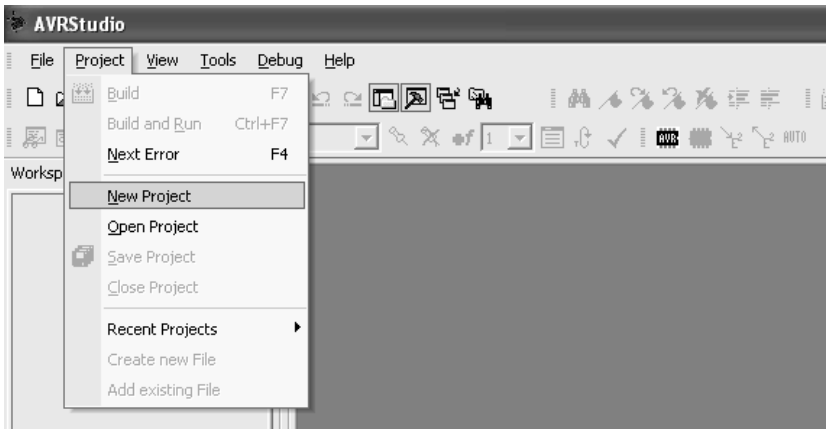


Рисунок 1.23 – Головне вікно інтерфейсу AVR Studio

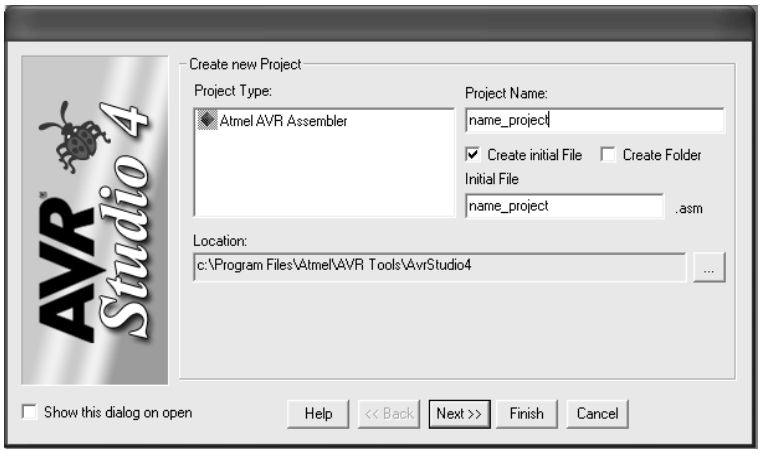


Рисунок 1.24 – Вікно створення проекту

5. У вікні редактора ввести код програми (рис. 1.25).

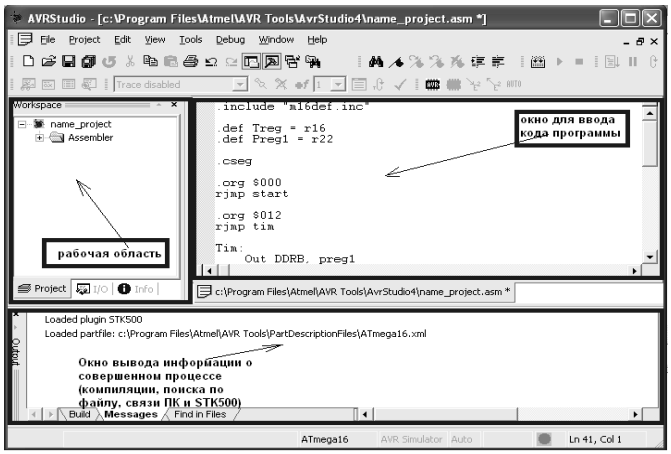


Рисунок 1.25 – Вікно запису програми

6. Зберегти набраний файл із розширенням *.ASM (*.C).

7. Скомпілювати набрану програму – кнопка «F7» на клавіатурі або натиснути кнопку .

8. Можливі помилки в програмі можна проглянути у вікні «message».

9. Після усунення всіх помилок, скомпілювати програму знову і записати дані файлу з розширенням *.HEX у мікроконтролер.

Для цього:

- натиснути кнопку «AVR» на панелі «AVRStudio toolbar», з’явиться вікно «STK500» (рис. 1.26);

- у закладці| «Program» вибрати «Device» – пристрій| (Atmega8515);

- у розділі «Flash» в полі «Input HEX File» вказати розташування файлу, що його скомпілювали.

Під час передачі даних з персонального комп'ютера в стенд STK500 дані відображуються на індикаторі стенда. Горять світлодіоди Led7, Led6, Led5.

Під час записування програми в МК| переконайтеся, що плата STK500 включена.

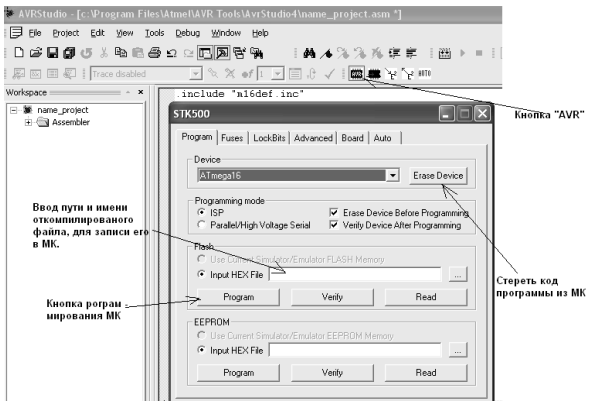


Рисунок 1.26 – Запис скомпільованої програми у мікроконтролер

Запис нової програми можливий у будь-який момент часу роботи завантаженої програми.

1.4 Контрольні запитання та завдання

1. Опишіть склад стенда STK 500 і його можливості.
2. Зарисуйте схему підключення STK 500 до ПК.
3. Покажіть, яким чином підключаються кнопки і світлодіоди до виходів портів.
4. Як з’єднати виходи послідовного порта UART мікроконтролера з роз’ємом інтерфейсу RS 232?
5. Розкажіть про порядок налагодження стенда STK 500 для роботи з AVR Studio.
6. Опишіть порядок створення програм в середовищі AVR Studio.
7. Опишіть порядок налагоджування програм в середовищі AVR Studio.
8. Опишіть порядок запису програм в цільовий мікроконтролер і їх виконання.
9. Запишіть текст програми засвічування світлодіодів у заданій комбінації, налагодіть і виконайте на цільовому МК.

Комбінації станів світлодіодів

№ вар.	Задана комбінація світлодіодів							
	LED7	LED6	LED5	LED4	LED3	LED2	LED1	LED0
1	1	1	0	0	1	1	0	0
2	0	1	1	0	0	1	1	0
3	0	0	1	1	0	0	1	1
4	0	1	0	1	0	1	0	1
5	1	1	1	0	0	0	1	1
6	0	0	0	1	1	1	0	0
7	1	0	0	1	0	0	1	1
8	1	0	1	0	1	1	0	0

ЛАБОРАТОРНА РОБОТА № 2
Архітектура мікроконтролера AVR і система його команд. Принципи програмування портів введення/виведення

Мета роботи – вивчення архітектури і системи команд мікроконтролерів AVR, прийомів програмування введення/виведення даних через порти.

2.1 Порядок виконання роботи

2.1.1 Ознайомитися з архітектурою мікроконтролера Atmega 8515 і системою його команд.

2.1.2 Ознайомитися з будовою портів введення/виведення мікроконтролера Atmega 8515.

2.1.3 Записати і налагодити в середовищі AVR Studio задану програму, яка реагує на стан кнопок на вході порту D засвіченням світлодіодів на виході порту B, завантажити її в стенд STK-500 і виконати.

2.1.4 Скласти звіт про виконання роботи.

2.2 Теоретичні відомості

2.2.1 Архітектура мікроконтролера та його програмна модель

Всі мікроконтролери AVR мають гарвардську архітектуру, яка припускає розділення пам’яті програм і даних. Способи адресації, що використовуються при цьому, дозволяють створювати ефективні програми з високою швидкодією.

Спрощена структурна схема мікроконтролера Atmega 8515 наведена на рис. 2.1. Ядро мікроконтролера утворюють блок процесора, який об’єднує арифметико-логічний пристрій (АЛП) з регістром ознак (SREG) і пристрій управління, пам’ять програм (Flash) обсягом 8 Кбайтів, 32 регістри загального призначення, пам’ять даних статичного типу (SRAM) обсягом 512ºбайтів. Пристрій керування містить схему синхронізації, регістр управління мікроконтролера (MCUCR), генератор, а також регістр команд з дешифратором, програмний лічильник і покажчик стека.

До периферійних пристроїв відносять:

- 8-розрядні порти введення/виведення PA, PB, PC, PD;
- 3-розрядний порт PE;
- послідовний асинхронний приймач/передавач UART (Universal Asynchronous Receiver-Transmitter);
- послідовний синхронний порт SPI (Serial Peripheral Interface);
- 8-розрядний таймер T0;
- 16-розрядний таймер T1;

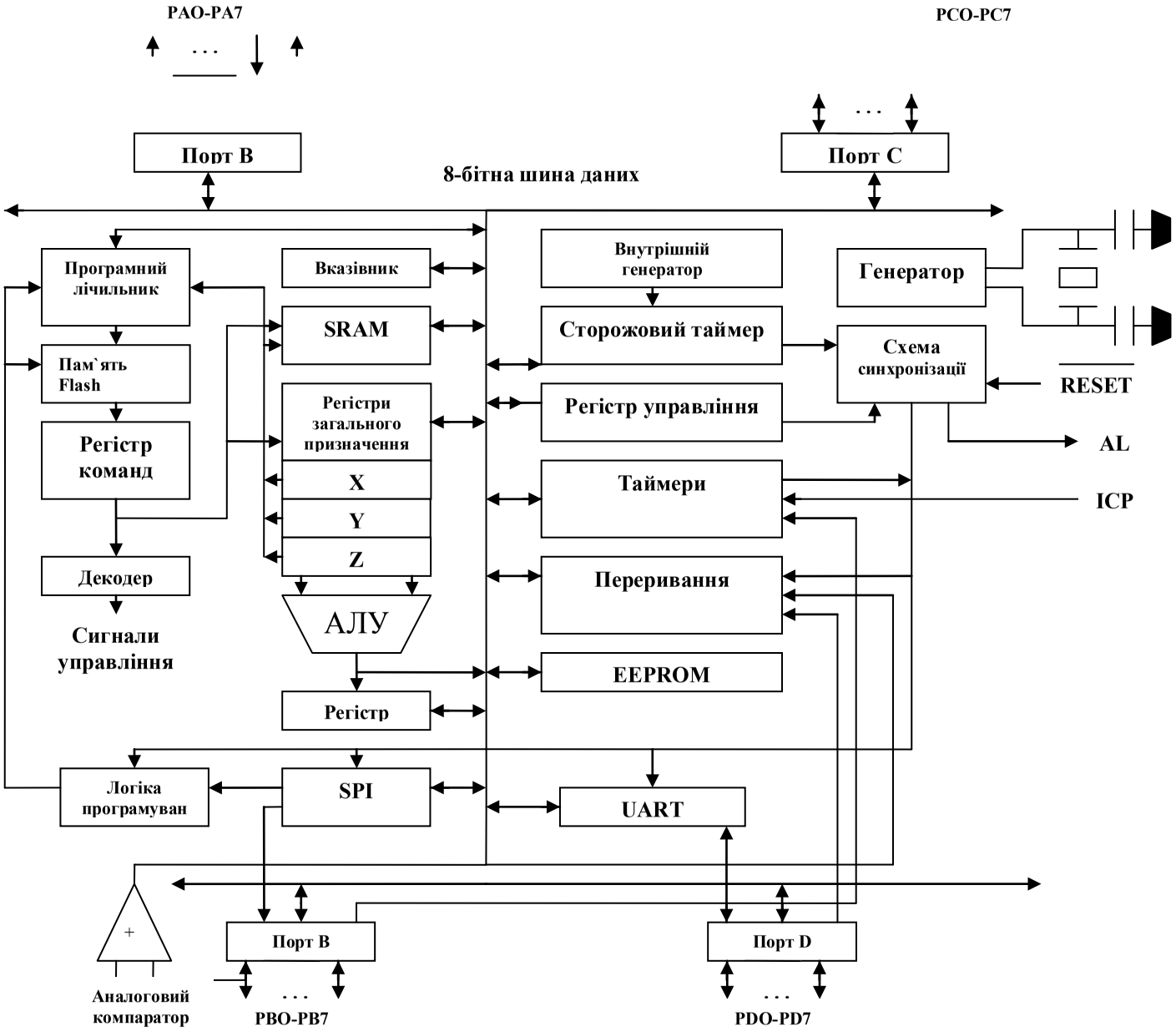


Рисунок 2.1 – Структурна схема мікроконтролера ATx8515

- сторожовий таймер;
- широтно-імпульсний модулятор PWM;
- енергозалежна пам’ять EEPROM обсягом 512 байтів;
- блок переривань;

Рисунок 2.2 – Карта пам’яті мікроконтролерів ATx8515

Пам’ять програм Flash відособлена, її розмір складає 8 Кбайтів. Кожний осередок Flash-пам’яті містить 16 розрядів. Пам’ять даних ділиться на три частини: регістрова, оперативна статична SRAM і енергонезалежна EEPROM.

Регістрову пам’ять складають 32 регістри загального призначення і 64 регістри введення/виведення, що є периферійними пристроями.

Оперативна пам’ять обсягом 512 байтів призначена для зберігання даних при виконані програми. Регістрова та оперативна пам’яті утворюють єдиний адресний простір: регістри загального призначення займають адреси $0000–$001F, за ними розташовуються регістри введення/виведення $0020–$005F, потім – елементи оперативної пам’яті $0060–$025F. Розширення адресного простору аж до верхньої межі $FFFF можна здійснити за рахунок підключення зовнішнього запам’ятовувального пристрою ERAM. Для довготривалого зберігання даних, які можуть змінюватися в процесі роботи мікроконтролера, використовують пам’ять EEPROM обсягом 512 байтів. Пам’ять EEPROM має відособлений адресний простір, кожний осередок містить вісім розрядів. Дані в EEPROM можуть бути записані при програмуванні мікроконтролера. При вимкненні живлення дані зберігаються.

Регістри загального призначення розбиті на дві групи: R0...R15 і R16...R31. Належність регістра до тієї або іншої групи потрібно враховувати при написанні програми.

Мікроконтролер AT90S8515 містить 44 регістри введення/виведення, що використовуються у складі периферійних пристроїв (ще 20 регістрів зарезервовано), з них 14 – 8-розрядні регістри даних, інші є регістрами керування та стану. Якщо врахувати, що вони подані бітами керування та стану, загальна кількість яких 187 (!), і частина їх з'єднана полями з кодами керування, то неважко зрозуміти, наскільки багатоманітні функціональні можливості цих пристроїв (наприклад, одне 3-розрядне поле керування дозволяє задати вісім керівних функцій). Список регістрів введення/виведення та їх адреси в адресному просторі SRAM наведені в табл. 2.1.

Таблиця 2.1 – Список регістрів введення/виведення

Адреса	20	30	40	50
0	Резерв	PIND	Резерв	Резерв
1	Резерв	DDRD	WDTCR	Резерв
2	Резерв	PORTD	Резерв	TCNT0
3	Резерв	PINC	Резерв	TCCR0
4	Резерв	DDRC	ICR1L	Резерв
5	Резерв	PORTC	ICR1H	MCUCR
6	Резерв	PINB	Резерв	Резерв
7	Резерв	DDRB	Резерв	Резерв
8	ACSR	PORTB	OCR1BL	TIFR
9	UBRR*	PINA	OCR1BH	TIMSK
A	UCR*	DDRA	OCR1AL	GIFR
B	USR*	PORTA	OCR1AH	GIMSK*
3	UDR	EECR	TCNT1L	Резерв
D	SPCR	EEDR	TCNT1H	SPL
E	SPSR	EEARL	TCCR1B	SPH
F	SPDR	Резерв	TCCR1A	SREG

Всі регістри мають штатні імена, які можна використовувати при написанні програм. Для цього в програму мовою Асемблер необхідно внести файл визначень 8515def.inc, а в програму мовою Сі – файл 90S8515.h. Розшифровування назв регістрів, формат і призначення кожного біта окремо наведені в подальших розділах практикуму при описі роботи відповідних периферійних пристроїв. Регістри, імена яких відзначені знаком *, в моделі ATmega8515 одержали інші імена: за адресою $29 – UBRRL, за адресою $2A – UCSRB, за адресою $2B – UCSRA, за адресою $5B – GICR.

Список регістрів введення/виведення мікроконтролера ATmega8515 і їх адреси в адресному просторі SRAM наведені в табл. 2.2. Мікроконтролер ATmega8515 містить 55 регістрів уведення/виведення. Файл визначень штатних імен регістрів для ATmega8515, що входять в програми на Асемблері, носить ім'я m8515def.inc, на Сі – mega8515.h.

Таблиця 2.2 – Список регістрів введення-виведення МК ATmega8515

Адреса	20	30	40	50
0	Резерв	PIND	UBRRH	SFIOR
1	Резерв	DDRD	WDTCR	OCR0
2	Резерв	PORTD	Резерв	TCNT0
3	Резерв	PINC	Резерв	TCCR0
4	OSCCAL	DDRC	ICR1L	Резерв
5	PINE	PORTC	ICR1H	MCUCR
6	DDRE	PINB	Резерв	EMCUCR
7	PORTE	DDRB	Резерв	SPMCR
8	ACSR	PORTB	OCR1BL	TIFR
9	UBRL	PINA	OCR1BH	TIMSK
A	UCSRB	DDRA	OCR1AL	GIFR
B	UCSRA	PORTA	OCR1AH	GICR
З	UDR	EECR	TCNT1L	Резерв
D	SPCR	EEDR	TCNT1H	SPL
E	SPSR	EEARL	TCCR1B	SPH
F	SPDR	Резерв	TCCR1A	SREG

Контролер переривань обробляє переривання зовнішні і від периферійних пристроїв (таймерів, портів послідовного введення/виведення, аналогового компаратора й ін.). Всі переривання є маскованими. Адреси, маски і прапорці переривань вказані в табл. 2.3. В табл. наведені:

INTO, INT1 – сигнали зовнішніх переривань, що надходять по лініях порту PD2, PD3. Маски зовнішніх переривань подані розрядами INTO і INT1 (відповідно 6-й і 7-й розряди регістра GIMSK мікроконтролера AT90S8515 або регістра GICR мікроконтролера ATmega8515). Сигнали зовнішніх переривань встановлюють в 1 прапорці переривань INTF0 і INTF1 (відповідно 6-й і 7-й розряди регістра GIFR);

переривання від таймерів T1, T0 мають адреси $003–$007. Маскування переривань від таймерів здійснюється бітами регістра TIMSK. Прапорці переривань таймерів розташовуються в регістрі TIFR;

подальші адреси переривань для запитів від послідовних каналів введення/виведення SPI ($008), UART або USART ($009, $00A, $00B) і аналогового компаратора ($00C). Маскування переривань здійснюється розрядами регістрів: для каналу UART – UCR (або UCSRB), для каналу SPI – SPCR, для компаратора – ACSR. Відповідні прапорці переривань розташовуються в регістрах стану USR (або UCSRA), SPSR, ACSR; решта переривань (зовнішні переривання INT2, від схеми порівняння таймера TO, після закінчення запису в пам’ять EEPROM і в пам’ять Flash), помічених знаком *, підтримуються мікроконтролером ATmega8515.

Запит з меншою адресою має більш високий пріоритет. Прапорець загального дозволу переривання I розташований в регістрі стану мікроконтролера SREG (біт 7). При I = 1 і одиничному значенні маски запиту переривання даного типу дозволено. Під час надходження запиту встановлюється прапорець переривання в одному з регістрів

введення/виведення, що може викликати апаратне переривання. Стан прапорця може бути також опитаний програмою.

Обробка переривання починається після завершення поточної команди, для чого може знадобитися декілька тактів залежно від типу виконуваної команди. При обробці переривання розряд I в регістрі SREG скидається в стан 0, забороняючи обробку всіх інших запитів. В стеку зберігається адреса повернення і виконується перехід за вектором переривання на першу команду обробника переривання. При виході з програми переривання розряд I знову встановлюється в стан 1, дозволяючи обробку переривань. Програма, що виконується під час запуску мікроконтролера і використовує вектор запиту RESET, не залежить від стану розряду I. Для обробки переривань вона має виконати команду дозволу переривань, що встановлює прапорець I в стан 1.

Таблиця 2.3 – Адреси і маски переривань ATx8515

Запит	Адреса	Маска	Прапорець
RESET	TOB	-	-
INTO	001	GIMSK.6/GICR.6	GIFR.6
INT1	002	GIMSK.7/GICR.7	GIFR.7
T/C1 З АРТ	003	TIMSK.3	TIFR.3
T/Cl COMPA	004	TIMSK.6	TIFR.6
T/Cl COMPB	005	TIMSK.5	TIFR.5
T/Cl OVF	006	TIMSK.7	TIFR.7
T/CO OVF	007	TIMSK.l	TIFR.1
SPI STC	008	SPCR.7	SPSR.7
UART RXC/	009	UCR.7/	USR.7/
USART RXC		UCSRB.7	UCSRA.7
U ART DRE/USART DRE	00A	UCR.5/UCSRB.5	USR.5/UCSRA.5
U ART TXC/USART TXC	00B	UCR.6/UCSRB.6	USR.6/UCSRA.6
ANA COMP	OOC	ACSR.3	ACSR.4
INT2*	00D	GICR.5	GIFR.5
T/CO COMP*	00E	TIMSK.O	TIFR.O
EERDY*	OOF	EECR.3	-
SPMRDY*	010	SPMCR.7	-

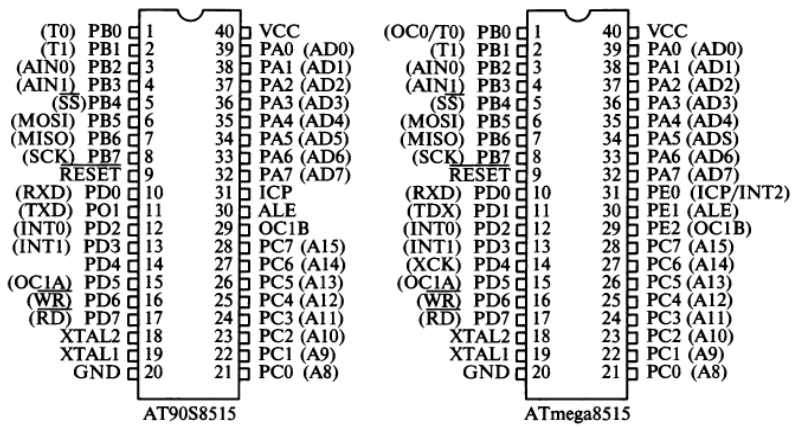


Рисунок 2.3 – Інтерфейси мікроконтролерів ATx8515

Аналоговий компаратор порівнює за значенням аналогові сигнали, що надходять на входи порту PB2, PB3, і формує запит переривання ANA COMP при зміні знака різниці, а також сигнал захоплення для таймера.

На рис. 2.3 наведені умовні графічні позначення мікросхем мікроконтролерів AT90S8515 і ATmega8515, що використовуються. Всі виводи портів мають альтернативні функції, що конфігуруються автоматично периферійними пристроями мікроконтролера. Їх імена наведені в дужках. Порівняно з AT90S8515 (рис. 2.3) в мікроконтролері ATmega8515 виводи 29–31 утворюють 3-розрядний порт PE. Мікроконтролери ATx8515 випускають в різних корпусах: TQFP, PLCC, PDIP. В STK500 мікроконтролери, що поставляються зі стартовим набором, виконані в 40-вивідному корпусі PDIP, встановлюваному в панелі STK500, і допускають заміну в процесі експлуатації.

2.2.2 Способи адресації в командах Асемблера AVR та система його команд

Для програмування мікроконтролера AT90S8515 використовується 118 команд, а ATmega8515 – 130. Всі команди можна розбити на групи операцій:

- арифметичних і логічних;
- пересилання і завантаження;
- передачі управління;
- роботи з бітами.

В табл. 2.4–2.7 наведено опис базового набору команд мікроконтролерів AVR. При описі команд використані такі позначення:

Rd, Rr – регістри загального призначення з номерами d і r;

Rdh:Rdl – пара регістрів;

K – константа (дані);

P,b – розряд b (b = 0 ...,7) порту P;

Rr(b) – розряд b (b = 0 ...,7) регістра Rr;

(X), (У), (Z) – вміст осередків, що адресуються регістровими парами X, У, Z відповідно;

Rd – регістр-приймач операнда;

Rr – регістр-джерело операнда;

n – номер біта;

s – номер розряду в регістрі SREG;

PC – вміст програмного лічильника;

k – приріст в лічильнику команд (адреса);

q – 6-розрядний зсув;

STACK – область пам’яті SRAM, що адресується покажчиком стека SP;

C, Z, N, V, S, H, T, I – біти регістра стану SREG;

d, r = 0 … 31 у всіх випадках, окрім спеціально відзначених.

Таблиця 2.4 – Арифметичні і логічні операції

Мнемоніка	Опис команди	Операція	Вплив на SREG
ADD Rd, Rr	Складання двох регістрів	Rd ← Rd + Rr	Z, C, N, V, H
ADC Rd, Rr	Складання двох регістрів і перенесення	Rd ←Rd + Rr + C	Z, C, N, V, H
ADIW Rdl, K	Складання регістрової пари з константою	Rdh:Rdl←Rdh:Rdl+K	Z, C, N, V, S
SUB Rd, Rr	Віднімання двох регістрів	Rd← Rd - Rr	Z, C, N, V, H
SUBI Rd, K	Віднімання константи з регістра	Rd← Rd - K d= 16-31	Z, C, N, V, H
SBC Rd, Rr	Віднімання двох регістрів із позикою	Rd← Rd - Rr - C	Z, C, N, V, H
SBCI Rd, K	Віднімання константи з регістра із позикою	Rd ← Rd-K-C d =16 - 31	Z, C, N, V, H
SBIW Rdl, K	Віднімання константи з регістрової пари	Rdh:Rdl← Rdh:Rdl - K	Z, C, N, V, S
AND Rd, Rr	Логічне І двох регістрів	Rd← Rd ∧ Rr	Z, N,V
ANDI Rd, K	Логічне І регістра і константи	Rd← Rd ∧ K d= 16-31	Z,N,V
OR Rd, Rr	Логічне АБО двох регістрів	Rd← Rd ∨ Rr	Z, N, V
ORI Rd, K	Логічне АБО регістра і константи	Rd← Rd ∨ K d= 16-31	Z,N,V
EOR Rd, Rr	Виключне АБО регістрів	Rd← Rd ⊕ Rr	Z,N,V
LSL Rd	Логічний зсув вліво	Rd(n+ l) ← Rd(n) Rd(0) ← 0	Z, C, N, V
LSR Rd	Логічний зсув управо	Rd(n) ← Rd(n+l), Rd(7) ← 0	Z, C, N, V
ROL Rd	Зсув вліво через перенесення	Rd(0) ← C Rd(n+ 1) ← Rd(n)C← Rd(7)	Z, C, N, V
ROR Rd	Зсув управо через перенесення	Rd(7) ← C Rd(n) ← Rd(n+1) C← Rd(0)	Z, C, N, V
ASR Rd	Арифметичний зсув управо	Rd(n) ← Rd(n+l) n = 0...6	Z, C, N, V
CP Rd, Rr	Порівняння регістрів	Rd-Rr	Z, N, V, C, H
CPC Rd, Rr	Порівняння регістрів з урахуванням позики	Rd - Rr – C	Z, N, V, C, H
CPI Rd, K	Порівняння регістра з константою	Rd-K,d= 16-31	Z,N, V, C, H
COM Rd	Інверсія регістра	Rd ← $FF - Rd	Z, C, N, V
NEG Rd	Зміна знака	Rd ← $00 - Rd	Z, C, N, V, H
SBR Rd, K	Логічне АБО регістра і константи	Rd ← Rd ∨ K d= 16-31	Z,N,V
CBR Rd, K	Логічне І Rd з інверсією константи	Rd← Rd ∧ ($FF - K)	Z, N, V
INC Rd	Інкремент регістра	Rd ← Rd + 1	Z, N, V
DEC Rd	Декремент регістра	Rd← Rd-1	Z, N, V
TST Rd	Перевірка регістра	Rd ← Rd ∧ Rd	Z, N, V
CLR Rd	Скидання регістра в 0	Rd ← Rd ⊕ Rd	Z, N, V
SER Rd	Встановлення 1 в розрядах регістра	Rd ← $FF d= 16-31	

Таблиця 2.5 – Команди пересилання

Мнемоніка	Опис команд	Операція
MOV Rd, Rr	Пересилання між регістрами	Rd← Rr
MOVW Rd, Rr	Пересилання між парами регістрів	Rd+l:Rd← Rr+l:Rr
SWAP Rd	Обмін тетрадами	Rd(3...0) Rd(7...4)
LDI Rd, K	Завантаження константи в регістр	Rd← K, d= 16-31
LD Rd, X	Непряме завантаження регістра	Rd ← (X)
LD Rd, X+	Непряме завантаження з постінкрементом	Rd ← (X), X← X + 1
LD Rd, -X	Непряме завантаження з переддекрементом	X←X-l, Rd← (X)
LD Rd, У	Непряме завантаження регістра	Rd ← (У)
LD Rd, Y+	Непряме завантаження з постінкрементом	Rd ← (У), У ← У + 1
LD Rd, -Y	Непряме завантаження з преддекрементом	Y←Y- l, Rd← (У)
LDD Rd,Y+q	Непряме завантаження відносне	Rd ← (У + q)
LD Rd, Z	Непряме завантаження регістра	Rd ← (Z)
LD Rd, Z+	Непряме завантаження з постінкрементом	Rd← (Z), Z← Z+1
LD Rd, -Z	Непряме завантаження з преддекрементом	Z← Z- l, Rd ← (Z)
LDD Rd, Z+q	Непряме завантаження відносне	Rd ← (Z + q)
LDS Rd, k	Пряме завантаження регістра	Rd ← (k)
ST X, Rr	Непряме збереження	(X) ← Rr
ST X+, Rr	Непряме збереження з постінкрементом	(X) ← Rr, X ← X + 1
ST-X,Rr	Непряме збереження з переддекрементом	X← X- l, (X) ← Rr
ST У, Rr	Непряме збереження	(У) ← Rr
ST Y+, Rr	Непряме збереження з постінкрементом	(У) ← Rr, У ← У + 1
ST -Y, Rr	Непряме збереження з переддекрементом	У ← У - 1, (У) ← Rr
STD У + q, Rr	Непряме збереження відносне	(У + q) ← Rr
ST Z, Rr	Непряме збереження	(Z) ← Rr
ST Z+, Rr	Непряме збереження з постінкрементом	(Z) ← Rr, Z ← Z + 1
ST -Z, Rr	Непряме збереження з переддекрементом	Z← Z-l,(Z) ← Rr
STD Z + q, Rr	Непряме збереження відносне	(Z + q) ← Rr
STS k, Rr	Пряме збереження	(k) ← Rr
LPM	Завантаження з програмної пам’яті в R0	R0 ← (Z)
LPM Rd, Z+	Завантаження з програмної пам’яті в регістр Rd з постінкрементом	Rd ← (Z), Z← Z+l
SPM	Збереження в програмній пам’яті	(Z) ← R1:R0
IN Rd, P	Читання регістра введення/виведення	Rd ← P, P = 0 - 63
OUT P, Rr	Запис у регістр введення/виведення	P← Rr, P = 0-63
PUSH Rr	Збереження в стеку	STACK ← Rr
POP Rd	Витягання зі стека	Rd ← STACK

Таблиця 2.6 – Команди управління

Мнемоніка	Опис команди	Операція
RJMP k	Перехід	PC ← PC + k + 1
IJMP	Непрямий перехід по (Z)	PC← Z
RCALL k	Виклик підпрограми	PC ← PC + k + 1
ICALL	Непрямий виклик по (Z)	PC← Z
RET	Повернення з підпрограми	PC ← STACK
RETI	Повернення з переривання	PC ← STACK
CPSE Rd, Rr	Порівняти і пропустити команду, якщо дорівнює	якщо (Rd = Rr), то PC← PC + 2(3)
SBRC Rr, b	Пропустити, якщо біт в регістрі дорівнює 0	якщо (Rr(b)= 0), то PC← PC + 2(3)
SBRS Rr, b	Пропустити, якщо біт в регістрі дорівнює 1	якщо (Rr(b)= 1), то PC← PC + 2(3)
SBIC P, b	Пропустити, якщо біт порту дорівнює 0	якщо (P(b)= 0), то PC ← PC + 2(3)
SBIS P, b	Пропустити, якщо біт порту дорівнює 1	якщо (P(b)= 1), то PC← PC + 2(3)
BRES s, k	Перейти, якщо прапорець в SREG = 1	якщо (SREG(s)= 1) то PC← PC+ k +1
BRBC s, k	Перейти, якщо прапорець в SREG = 0	якщо (SREG(s)= 0) то PC← PC + k + 1
BREQ k	Перейти, якщо дорівнює	якщо (Z = 1), то PC← PC + k + 1
BRNE k	Перейти, якщо не дорівнює	якщо (Z=0),то PC←PC+k+1
BRCS k	Перейти, якщо C = 1	якщо (C = 1), то PC← PC + k + 1
BRCC k	Перейти, якщо C = 0	якщо (C = 0), то PC← PC + k + 1
BRSH k	Перейти, якщо більше або дорівнює	якщо (C = 0), то PC← PC + k+ 1
BRLO k	Перейти, якщо менше	якщо (C = 1), то PC← PC + k + 1
BRMI k	Перейти, якщо мінус	якщо (N = 1), то PC← PC + k + 1
BRPL k	Перейти, якщо плюс	якщо (N = 0), то PC← PC + k + 1
BRGE k	Перейти, якщо більше або дорівнює (з урахуванням знаку)	якщо (N ⊕ V = 0) то PC← PC + k + 1
BRLT k	Перейти, якщо менше (з урахуванням знаку)	якщо(N ⊕ V = 1) то PC← PC + k + 1
BRHS k	Перейти, якщо міжтетрадне перенесення H = 1	якщо (H = 1) то PC← PC + k + 1
BRHC k	Перейти, якщо міжтетрадне перенесення H = 0	якщо (H = 0) то PC ← PC + k + 1
BRTS k	Перейти, якщо прапорець T = 1	якщо (T = 1) то PC ← PC + k + 1
BRTC k	Перейти, якщо прапорець T = 0	якщо (T = 0) то PC ← PC + k + 1
BRVS k	Перейти, якщо прапорець переповнення V= 1	якщо (V = 1) то PC ← PC + k + 1
BRVC k	Перейти, якщо прапорець переповнення V = 0	якщо (V = 0) то PC ← PC + k + 1
BRIE k	Перейти, якщо прапорець переривання I = 1	якщо (I = 1) то PC ←PC + k + 1
BRID k	Перейти, якщо прапорець переривання I = 0	якщо (I = 0) то PC ← PC + k + 1

Таблиця 2.7 – Операції з бітами

Мнемоніка	Операція	Мнемоніка	Операція
SBI P, b	(P,b) ← 1,P=0-31	SES	S←1
CBI P, b	(P,b) ← 0, P=0-31	CLS	S←0
BSET S	SREG(S) ← 1	SEV	V←1
BCLR S	SREG(S) ← 0	CLV	V← 0
BST Rr, b	T ← Rr(b)	SET	T← 1
BLD Rd, b	Rd(b) ← T	CLT	T← O
SEC	C ← 1	SEH	H ← 1
CLC	C ← 0	CLH	H ← O
SEN	N ← 1	NOP	Немає операції
CLN	N← 0	SLEEP	Режим енергозбереження
SEZ	Z← 1	WDR	Скидання WDT
CLZ	Z← 0		
SEI	I ← 1		
CLI	I← 0		

2.2.3 Способи адресації даних в мікроконтролерах AVR

Ключовим моментом до розуміння функцій, виконуваних кожною командою, крім коду операції, є способи адресації даних (операндів), що використовуються командою.

Мікроконтролери AVR застосовують різноманітні способи адресації даних. За кількістю та способами адресації вони перевершують можливості мікроконтролерів MCS-51. При тому або іншому способі адресації можна здійснити доступ до будь-якої області пам’яті даних (регістрів загального призначення, регістрів уведення/виведення, пам’яті SRAM), а також Flash-пам’яті програм і енергозалежної пам’яті даних EEPROM. При цьому часто до одного і того ж об’єкта можна звернутися різними способами, використовуючи для цього відповідний вид адресації. Розглянемо кожний з них докладніше.

Пряма регістрова адресація з одним регістром Rd. При цьому способі адресації дані знаходяться в регістрі d(Rd), адреса якого міститься безпосередньо в команді (рис. 2.4, а). Прикладом команд, що використовують цей метод адресації, є команди загального призначення для роботи зі стеком (PUSH, POP), обміну тетрадами в регістрі (SWAP), ряд команд арифметичних і логічних операцій.

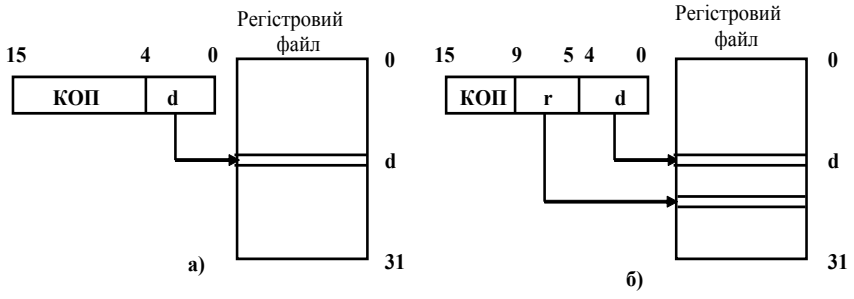


Рисунок 2.4 – Пряма регістрова адресація одного (а) і двох (б) регістрів

Приклади запису команд з такою адресацією:
LSL R18
INC R20
SER R24
COM R19

Пряма регістрова адресація з двома регістрами Rd і Rr. Даний спосіб адресації застосовується в командах, які використовують два регістри загального призначення: d(Rd) і r(Rr) (рис. 2.4, б). Цей вид адресації використовують команди пересилання даних з регістра в регістр і більшість команд арифметичних операцій, ряд команд логічних операцій. При цих операціях результат операції зберігається в регістрі d (Rd). Приклади запису команд з такою адресацією:
ADD R4, R5
OR R16, R22
MOV R16, R0

Пряма адресація регістра введення/виведення. Даний вид адресації використовують для виконання обміну між регістром введення/виведення, розташованим в адресному просторі введення/виведення, і одним з регістрів загального призначення за командами IN і OUT (рис. 2.5).

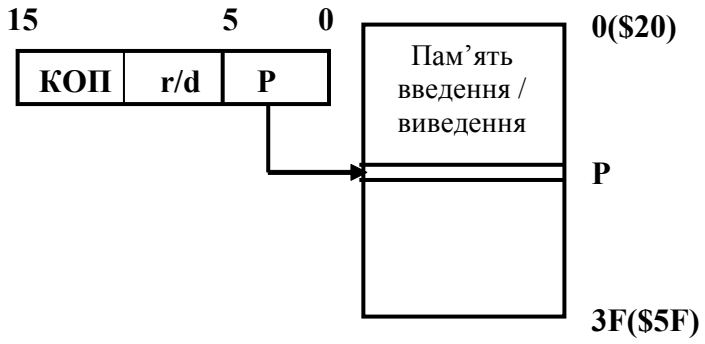


Рисунок 2.5 – Пряма адресація регістра введення/виведення

Приклади запису команд:
IN R17, PORTD
OUT PORTC, R16

Пряма адресація пам'яті даних. Даний спосіб адресації застосовується при зверненні до будь-якого осередку адресного простору SRAM. Є всього дві команди: LDS і STS, кожна завдовжки в два слова (32 розряди). Перше слово містить код операції і адресу регістра загального призначення, друге – 16-розрядна адреса осередку, до якого виконується звернення (рис. 2.6).

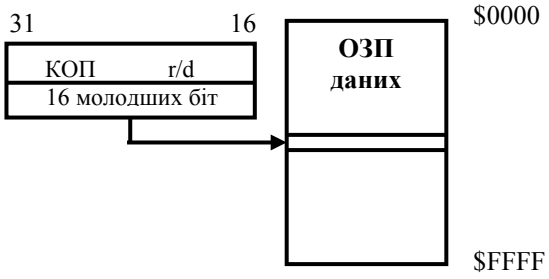


Рисунок 2.6 – Пряма адресація пам’яті даних

Приклади команд:
LDS R17, 0x045
STS $081, R20

Непряма адресація пам’яті даних. При непрямій адресації звернення, обхід направлений до елемента пам’яті, адреса якого знаходиться в 16-розрядному індексному регістрі X, У або Z (рис. 2.7). В ролі цих регістрів виступають пари регістрів: R26, R27 (регістр X), R28, R29 (регістр У) і R30, R31 (регістр Z). Приклади команд:

ST У, R23
LD R21, X

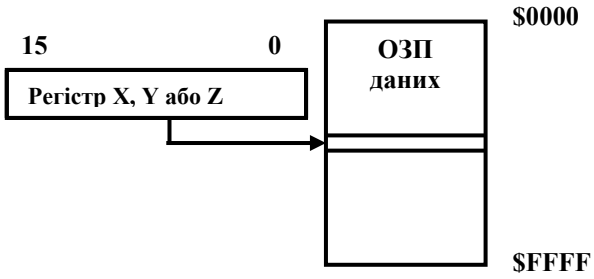


Рисунок 2.7 – Непряма адресація пам’яті даних

Непряма адресація пам’яті даних із зсувом. При цьому способі адреса елемента пам’яті визначається шляхом підсумовування вмісту індексного регістра У або Z з 6-розрядним зсувом, що задається в команді (рис. 2.8). Цей спосіб адресації використовують команди LDD (пересилання байта з елемента пам’яті SRAM в регістр Rd) і STD (пересилання байта з регістра Rr в осередок SRAM).

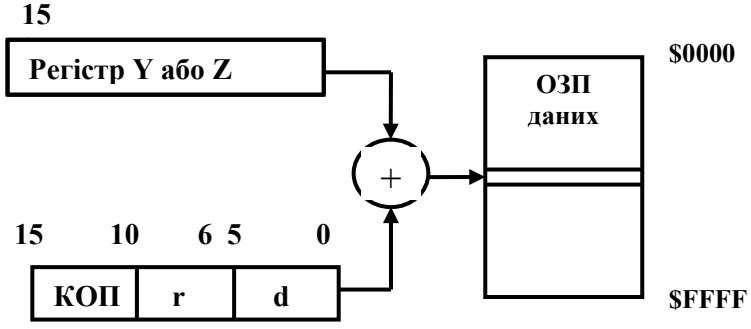


Рисунок 2.8 – Непряма адресація пам’яті даних із зсувом

Приклади команд:
LDD R10, Y+12
STD Z+1, R25

Непряма адресація пам'яті даних з переддекрементом. При цьому способі адресації вміст індексного регістра X, У або Z спочатку зменшується на 1, а потім проводиться звернення до пам'яті за одержаною адресою (рис. 2.9). Цей спосіб адресації використовують команди LD (пересилання байта даних з пам'яті в регістр Rd) і ST (пересилання байта даних з регістра Rr в пам'ять), всього шість команд – по дві для кожного регістра.

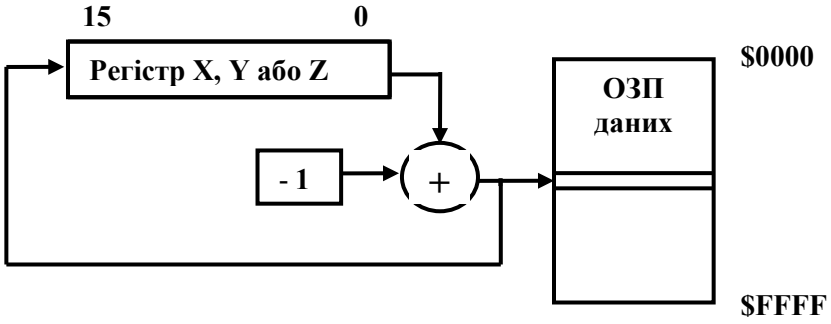


Рисунок 2.9 – Непряма адресація з переддекрементом

Приклади команд:
ST -Y, R16
LD Rd, -X

Непряма адресація пам'яті даних з постінкрементом. При цьому способі адресації вміст індексного регістра X, У або Z спочатку використовується як адреса звернення до пам'яті даних, а потім збільшується на 1 (рис. 2.10). Цей спосіб адресації використовують команди LD (пересилання байта даних з пам'яті в регістр Rd) і ST (пересилання байта даних з регістра Rr в пам'ять), всього шість команд – по дві для кожного регістра.

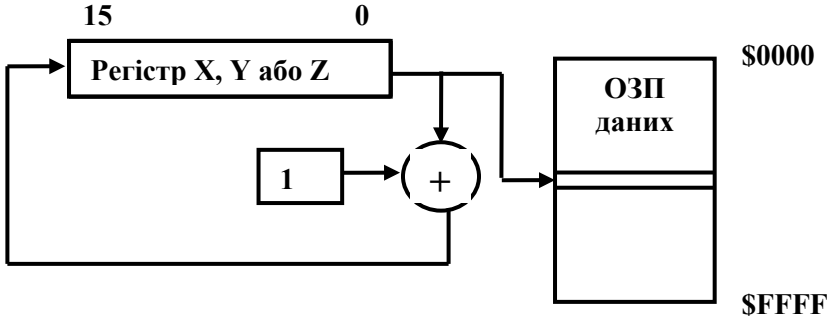


Рисунок 2.10 – Непряма адресація з постінкрементом

Приклади команд:

LD R24, Y+

ST X+, R16

Непряма адресація пам'яті програм. Мікроконтролери AVR дозволяють звернутися до елементів пам'яті програм для прочитування констант, а в моделях сімейства Mega і для запису даних у Flash-пам'ять програм, використовуючи механізм непрямої адресації через регістр Z. При цьому старші 15 розрядів регістра визначають адресу слова, а молодший нульовий розряд – молодший або старший байт слова. Якщо молодший розряд адреси дорівнює 0, вибирається молодший байт, інакше – старший байт (рис. 2.11). Даний вид адресації використовують команди читання з елемента пам'яті в регістр R0 (LPM) і записи в пам'ять з регістрів R1:R0 (SPM).

Окрім простої непрямої адресації при читанні константи в регістр Rd можна застосувати **непряму адресацію з постінкрементом** (команда LPM, Rd, Z+).

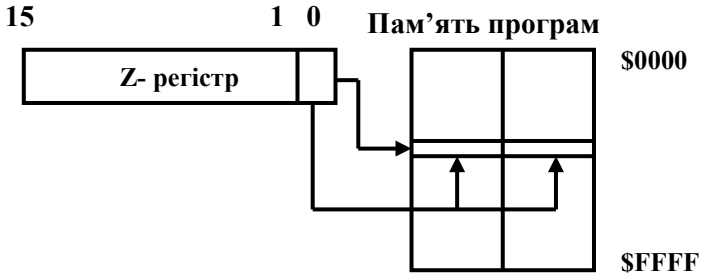


Рисунок 2.11 – Непряма адресація констант в пам'яті програм

Приклади команд:

SPM ; Збереження в програмній пам'яті (Z) ←R1:R0

LPM ; Завантаження з програмної пам'яті R0 ← (Z)

LPM R16,Z+ ;Завантаження з програмної пам'яті в регістр Rd з постінкрементом
; Rd ← (Z),Z←Z+l

Крім команд, пов'язаних з передачею даних, непряма адресація може бути використана в командах **непрямого переходу за адресою в регістрі** Z(IJMP) і непрямого виклику підпрограми через регістр Z(ICALL) (рис 2.12).

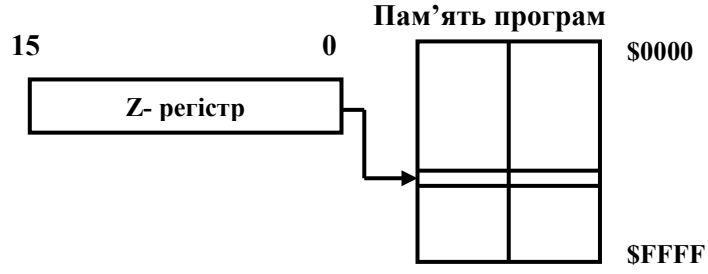


Рисунок 2.12 – Непряма адресація пам'яті програм

Відносна адресація пам'яті програм. При цьому способі адресу обчислюють шляхом складання вмісту програмного лічильника PC і константи k, що задається в команді (рис. 2.13). Відносну адресацію використовують команди відносного переходу (RJMP) і відносного виклику підпрограми (RCALL), численна група команд умовних переходів.

Безпосередня адресація. Даний вид адресації має на увазі вказання одного з операндів (константи k) безпосередньо в команді. Безпосередня адресація використовується командою пересилання константи в регістр LDI, а також деякими командами арифметичних і логічних операцій.

Приклади команд:
CPI R18, 0x086
ANDI R20, 12

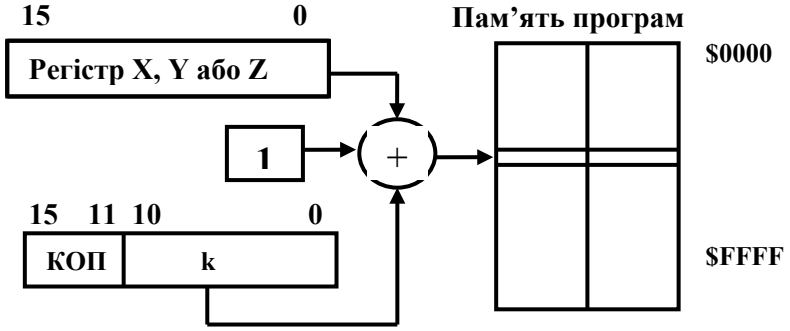


Рисунок 2.13 – Відносна адресація пам'яті програм

Бітова адресація. Цей вид адресації дозволяє вказати один з восьми бітів будь-якого з 32 регістрів загального призначення або першої половини регістрів введення/виведення з номерами 0–31, а також регістра SREG. Для цього потрібно вказати ім'я регістра загального призначення Ri (i = 0...31) або ім'я регістра введення/виведення Pi (i = 0.. .31), або ім'я SREG і номер біта b (b = 0 ... 7). Командами SBI і CBI здійснюється встановлення в 1 і скидання в 0 вказаного біта регістра введення/виведення, командами BLD, BST – обмін значеннями біта T з регістра SREG і адресованого біта з регістра загального призначення. Крім цього є група команд бітових операцій, що забезпечує встановлення і скидання бітів регістра стану SREG. При записі мнемоніки команд допускається використання символічних (штатних) імен регістрів вв/вив та імен бітів. Існує група команд умовного переходу, де як умова використовується або значення біта в регістрі загального призначення (SBRC, SBRS), або значення біта в регістрі вв/вив (SBIC, SBIS).

Приклади команд:
SBRC R17, 1 ; Пропустити, якщо біт 1 в регістрі R17 дорівнює «0»
SBIS PORTA, 0 ; Пропустити, якщо біт 0 порту A дорівнює «0»

```
SBI PORTD, 2       ; Встановити біт 2 порту D в «1» стан
```

2.2.4 Загальний опис команд

Раніше були наведені таблиці з описом команд мікроконтролерів AVR, розбиті по групах. Окрім операції і операндів для кожної команди вказані ознаки результату, формовані в регістрі SREG. Мнемоніки команд, виконуваних тільки в мікроконтролерах сімейства Mega, помічені *.

Група арифметичних і логічних операцій за складом достатньо традиційна для архітектури 8-розрядних мікроконтролерів, за винятком мікроконтролерів сімейства Mega. Так, наприклад, в групу арифметичних команд мікроконтролера ATmega8515 додано шість операцій множення: беззнакових чисел MUL, чисел із знаком MULS, беззнакового числа на число зі знаком MULSU, дробових беззнакових чисел FMUL, дробових із знаком FMULS, дробового беззнакового та дробового зі знаком FMULSU. Дробові співмножники мають формат 1.7, їх добуток – формат 1.15, де праворуч від крапки вказано число дробових розрядів. У всіх операціях множення джерелами операндів є регістри Rd і Rr, добуток формується в регістрах Rl :R0.

При виконанні операцій додавання/віднімання приймачем результату є один з регістрів загального призначення, в якому до операції знаходиться один з операндів. Таким чином, можна говорити про реалізацію АЛП акумуляторного типу відносно будь-якого регістра загального призначення (порівняйте: мікроконтролери з ядром MCS-51 мають всього лише один акумулятор, що, безумовно, погіршує ефективність обробки даних і веде до зниження продуктивності в цілому).

Особливістю системи команд мікроконтролерів AVR є відсутність команди двійково-десяткової корекції.

Команди пересилання можна використовувати для передачі даних з регістра в регістр, для пересилань між регістрами та осередками адресного простору SRAM, регістрами введення/виведення та регістрами загального призначення, що побічно адресуються, для збереження і витягання даних із стека, читання констант з Flash-пам’яті програм і навіть запису в Flash-пам’ять (в моделі ATmega8515). Слід звернути увагу, що безпосереднє завантаження константи в регістри загального призначення першої половини (R0...R15) неможливе. Для цього необхідно заздалегідь завантажити константу в один з регістрів другої половини (R16...R31), а потім переслати вміст допоміжного регістра в регістр першої половини.

Широко презентована група команд передачі управління. Крім традиційних команд безумовної та умовної передачі управління за прапорцем є команди непрямого переходу і непрямого виклику підпрограм. Команди умовних переходів поділяться на два типи. Команди першого типу при виконанні умови забезпечують перехід за адресою, обчислюваною як сума (PC + k + 1). При невиконанні умови відбувається

перехід до наступної команди програми за адресою (PC + 1). Команди другого типу при виконанні умови забезпечують перехід до команди, наступної за черговою, тобто за адресою (PC + 2), якщо довжина чергової команди складає одне слово, або за адресою (PC + 3), якщо довжина чергової команди складає два слова. Якщо умова не виконується, відбувається перехід до наступної команди за адресою (PC +1). Така різноманітність команд управління сприяє ефективній роботі компіляторів програм, написаних мовою Сі.

В бітових операціях слід звернути увагу, що всі операції пересилання бітів в області пам'яті регістрів загального призначення здійснюються тільки через допоміжний біт T регістра стану SREG. В області пам'яті регістрів уведення/виведення біти можна змінювати тільки шляхом установлення 1 або 0. Програмно шляхом установлення 1 або 0 можна змінювати стани прапорців регістра стану SREG. В цю групу входять також операції WDR (скидання сторожового таймера), SLEEP (перехід в енергоощадний режим), пуста операція NOP.

2.2.5 Директиви Асемблера

При написанні програм мовою Асемблер використовуються директиви, які вказують компілятору положення програми в пам'яті, визначають макроси, ініціалізують пам'ять і ін. Список директив і їх опис наведено в табл. 2.8. Запис всіх директив починається з крапки. Стисло перерахуємо виконувані директивами функції в кожному з сегментів.

Сегмент програми відкривається директивою .CSEG. Якщо програма починається з цього сегмента, директива може не бути. В сегменті програми за допомогою директиви .ORG можна вказати початок сегмента.

Директива .DB в сегменті визначає один байт або групу байтів, констант, записуваних у Flash-пам'ять. Директива .DW визначає слово або групу слів, записуваних в пам'ять як константи. Початок запису констант визначається міткою, яка стоїть перед відповідною директивою. Перераховувані константи розділяються комами.

Директива .DEF присвоює регістру символічне ім'я. Директиви .EQU, .SET присвоюють значення імені. Ім'я, якому присвоєно значення директивою .EQU, не можна перепризначувати. Ім'я, присвоєне директивою .SET, може бути змінено іншою директивою .SET.

Директива .DEVICE визначає тип цільового мікроконтролера, який буде використаний для виконання програми. Наявність цієї директиви підключає засоби контролю інструкцій програми стосовно фізичного пристрою, попереджаючи про неможливість виконання деяких інструкцій, розмірів використовуваної пам'яті та ін.

Директива .INCLUDE з ім'ям файлу використовується для внесення в текст програми іншого файлу.

Таблиця 2.8 – Список директив

Директива	Опис
.BYTE	Резервувати байти в ОЗП
.CSEG	Сегмент програми
.DB	Визначити байт-константу в Flash-пам’яті або EEPROM
.DEF	Призначити регістру символічне ім’я
.DEVICE	Визначає пристрій, для якого компілюється програма
.DSEG	Сегмент даних
.DW	Визначає слово в Flash-пам’яті
.ENDM	Кінець макросу
.EQU	Встановити постійний вираз
.ESEG	Сегмент EEPROM
.EXIT	Вихід з файлу
.INCLUDE	Вкласти інший файл
.LIST	Ввімкнути генерацію лістингу
.LISTMAC	Ввімкнути розвертання макросів в лістингу
.MACRO	Початок макросу
.NOLIST	Вимкнути генерацію лістингу
.ORG	Встановити положення в сегменті
.SET	Встановити для змінної еквівалентний вираз

Директиви .MACRO і .ENDMACRO обрамляють макроозначення. Макроозначення може мати до 10 параметрів з фіксованими іменами @0...,@9. При виклику макроозначення параметри задають у вигляді списку в порядку нумерації.

Сегмент даних починається директивою .DSEG. В сегменті можуть бути використані директиви .ORG і .BYTE. Директива .BYTE визначає кількість байтів, до яких проводитиметься звернення при виконанні програми. Резервована область починається за адресою, визначуваною міткою перед директивою.

Сегмент типа EEPROM починається директивою .ESEG. В сегменті можуть бути використані директиви .ORG, .DB, .DW. Директива .DB в сегменті визначає один або групу байтів, записуваних в EEPROM. Директива .DW визначає слово або групу слів, записуваних в пам’ять EEPROM парами по 2 байти. Початок запису байтів і слів визначається міткою, яка стоїть перед відповідною директивою.

Директиви .LIST, .NOLIST, .LISTMAC використовують для управління виведенням лістингу.

2.2.6 Вирази

При записі команд на Асемблері можуть використовуватися вирази, за якими в процесі асемблювання програми обчислюються значення. Операндами виразів можуть бути:

- числа (десяткові, шістнадцяткові і двійкові);
- цілі;
- коди символів ASCII ('A') і рядка ASCII;
- символічні імена, які позначають змінні,

визначені директивою .SET, і константи, визначені директивою .EQU;

- поточне значення лічильника команд (PC).

Для позначення шістнадцяткових чисел використовують покажчики 0x або $ (0xla, 0xff, $ff), для двійкових чисел – 0b (0b0000l111, 0b11111111), десяткові числа не мають покажчиків (255, 0).

Крім операндів у вирази можуть входити функції, наприклад:

LOW (вираз) – повертає молодший байт виразу;

HIGH (вираз) – повертає старший байт виразу;

EXP2 (N) – повертає 2^N;

LOG2 (N) – повертає цілу частину log2N.

Під час запису виразів можна використовувати арифметичні, логічні та операції відношення. Групу арифметичних операцій утворюють додавання двох чисел або виразів (N + M), віднімання (N - M), множення (N*M), ділення (N/M), зміна знака числа (-N). Групу логічних операцій утворюють інверсія (~N), побітове І (N&M), побітове АБО (N | M), побітове виключне АБО (N⊕M), зсув вліво (N<<M – зсунути N вліво на M розрядів), зсув управо N>>M. Операції відношення:

- логічне заперечення (!N – повертає 1, якщо N = 0, і 0, якщо N=1);
- менше (N < M – повертає 1, якщо N < M, і 0, якщо N > M);
- більше (N > M – повертає 1, якщо вираз N > M, і 0 якщо N < M);
- менше або дорівнює (N <= M – повертає 1, якщо N <= M, і 0 якщо N > M);
- більше або дорівнює (N >= M – повертає 1, якщо N >= M, і 0 якщо N < M);
- дорівнює (N = M – повертає 1, якщо N = M, і 0, якщо N ≠ M);
- не дорівнює (N != M – повертає 1, якщо N ≠ M, і 0, якщо N = M);
- логічне І (N&&M – повертає 1, якщо N ≠ 0 і M ≠ 0 інакше 0);
- логічне АБО (N||M – повертає 0, якщо N = 0 і M = 0 інакше 1).

Для вказання черговості операцій можна використовувати круглі дужки.

2.2.7 Приклади запису програм в командах Асемблера

Як приклад підрахуємо в масиві пам’яті кількість байтів з одиницею в нульовому біті. Масив розташований в пам’яті з адреси 0x61 по 0x81. Результат записати в комірку пам’яті з адресою 0x88, якщо результат більший чи дорівнює десяти, а інакше – в комірку з адресою 0x89.

Блок-схема алгоритму роботи програми, яка наведена на рис. 2.14.

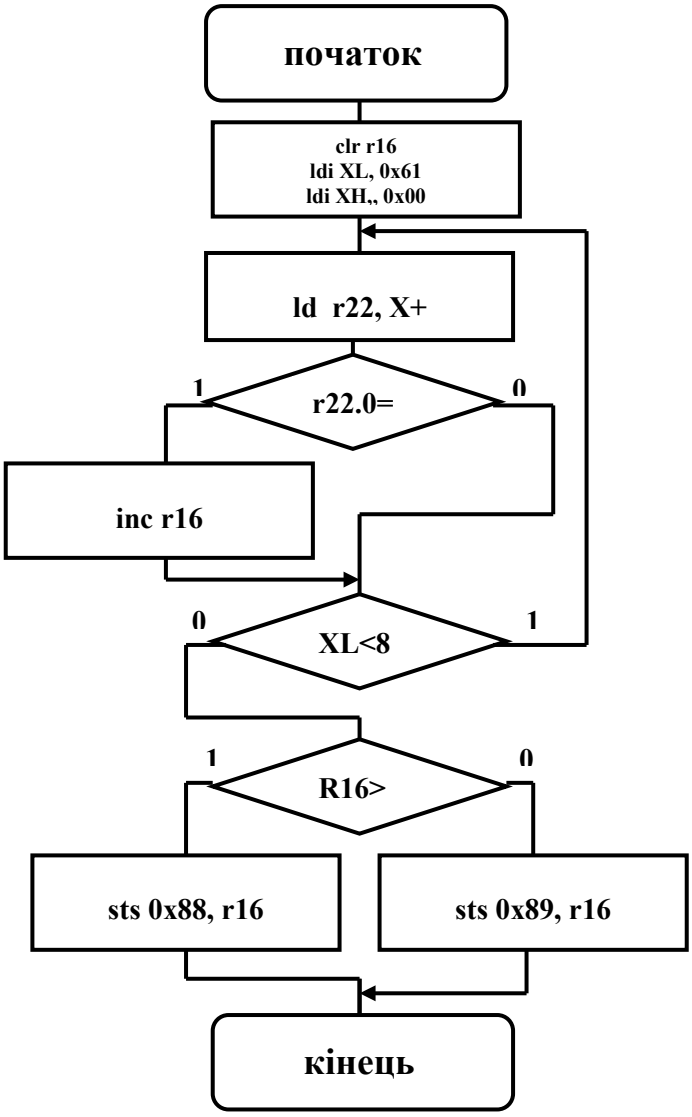


Рисунок 2.14 – Блок-схема алгоритму роботи програми

Текст програми

```
.include "2313def.inc"   ; підключення бібліотеки мікроконтролера
.org 00                  ; установлення початкової адреси програми
clr r16                  ; очищення регістра R16 для формування результату
ldi XL,0x61
ldi XH,0x00      ; завантаження у вказівник X початкової адреси 0x61
loop:  ld r22,X+         ; завантаження r22 елемента масиву з пам'яті
sbrc r22,0               ; пропуск наступного рядка, якщо R22.0 = 0
inc r16                  ; корекція результату
cpi XL,(0x81+1)    ; порівняння вказівника XL з максим. адресою
brne loop                ; перехід на мітку loop, якщо отримана нерівність
cpi r16,10               ; порівняння R16 з числом 10
brlo st89                ; перехід на мітку st89, якщо R16<10
```

```
        sts 0x88,r16        ; запис результату за адресою 0x88
        rjmp exit           ; безумовний перехід на мітку exit
st89:   sts 0x89,r16        ; запис результату за адресою 0x89
exit:
```

2.3 Програмування портів введення/виведення

Мікроконтролери AVR фірми Atmel мають широкі можливості з введення і виведення даних. Мікроконтролери моделей ATx8515 мають чотири паралельні 8-розрядні порти Px (x = A, B, C, D) і один 3-розрядний порт PE (в моделі ATtega8515). Всі лінії портів можуть програмуватися на введення або виведення даних незалежно одна від одної і підключатися через внутрішні «підтягувальні» резистори $R_{підт}$ з опором 35.....120 кОм до шини живлення VCC.

До складу кожного порту Px (PA, PB, PC і ін.) входять три регістри з іменами DDRx, PORTx і PINx. В мікроконтролері AT90S8515 регістр PINx не має апаратної реалізації. Це ім'я використовується для читання ліній інтерфейсу. На рис. 2.14 наведена загальна структурна схема 8-розрядних портів Px і структурна схема одного розряду порту Px.Y (У = 0, 1...7) мікроконтролера AT90S8515.

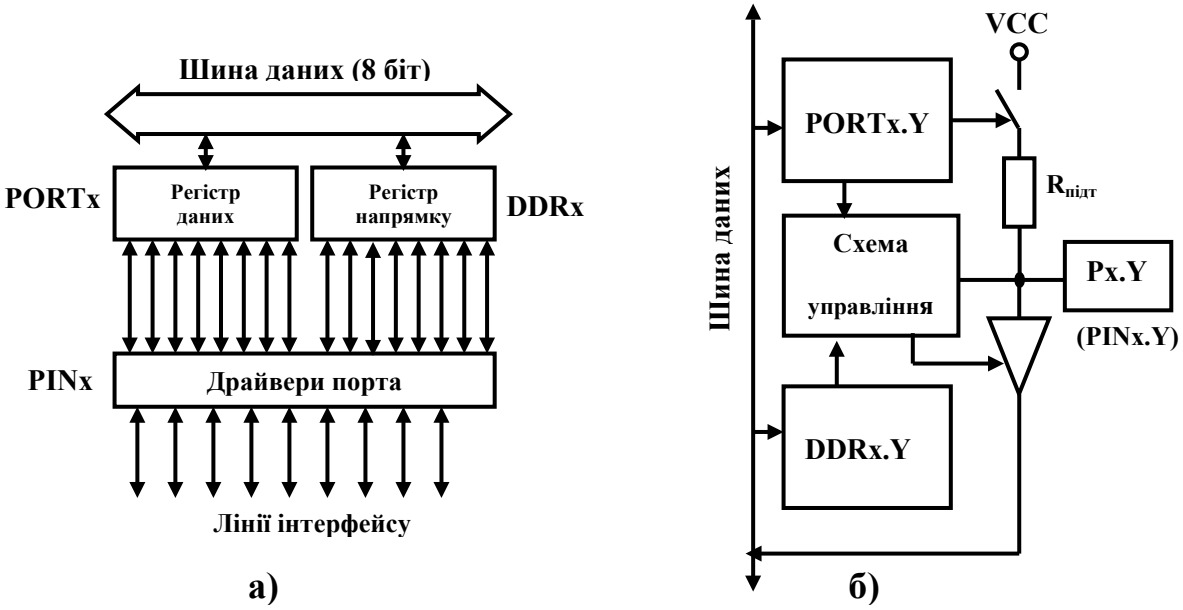


Рисунок 2.14 – Структура порту Px (а) і схема одного розряду порту (б)

Стан розряду DDRx.Y визначає напрям передачі біта даних через вивід порту Px.Y. При DDRx.Y = 0 виведення порту Px.Y є входом, при DDRx.Y = 1 – виходом.

В режимі входу стан розряду PORTx.Y визначає стан виводу Px.Y. При PORTx.Y = 1 вивід порту через внутрішній резистор підключено до шини

живлення VCC. При PORTx.Y = 0 резистор відключається, вивід Px.Y знаходиться у високоімпедансному стані (Z-стан).

В режимі виходу стан розряду PORTx.Y визначає значення сигналу на виводі Px.Y. При PORTx.Y = 0 на виводі встановлюється напруга низького рівня, при PORTx.Y = 1 – високого.

При пуску і перезапуску мікроконтролера всі розряди регістрів DDRx і PORTx скидаються в нульовий стан, унаслідок чого виводи портів працюють в режимі входу і знаходяться в Z-стані.

При одночасному використанні всіх розрядів порту для введення байта даних використовують команди з мнемонікою IN Rd, PINx, для виведення – OUT PORTx, Rr (d, г = 0...31). Значення вихідного сигналу на окремому виводі порту можна задати за допомогою команд установлення 0 (CBI PORTx,Y) і 1 (SBI PORTx,Y). Вхідний сигнал на окремому виводі порту можна перевірити, використовуючи команди умовного переходу SBIC PINx,Y або SBIS PINx,Y, які передбачають пропуск наступної команди за нульовим або одиничним значенням Px.Y.

2.4 Приклад виконання програми

Завдання. Підготувати програму, яка при натисканні кнопки START виконує почергове перемикання світлодіодів, при натисканні кнопки STOP зупиняє перемикання і відновлює при повторному натисканні кнопки START.

Виконання

Приклад програми наведений далі. В програмі для мікроконтролера AT90S8515 використовується файл визначень 8515def.inc, для ATmega8515 – m8515def.inc.

В програмі лінії порту PB використані для індикації і, отже, проініціалізовані на виведення, а лінії 0 і 1 порту PD, що з'єднуються з кнопками, – на введення. Після натискання кнопки START починається послідовне перемикання світлодіодів із затримкою і перевірка стану кнопки STOP.

Програма 2.1

```
**************************^
;Програма 2.1 для мікроконтролерів ATx8515:
;перемиканн світлодіодів (СД) при натисканні на кнопку START
;(SW0), після натискання кнопки STOP (SW1) перемикання
;припиняється і поновлюється з місця зупинки
;при повторному натисканні на кнопку START
.*****************************
.include "8515def.inc"                    ;файл визначень для AT90S8515
```

```
.include "m8515def.inc"                ;файл визначень для ATmega8515
.def temp = rl6                        ;тимчасовий регістр
.def reg_led = r20                     ;стан регістра світлодіодів
.equ START = 0                         ;0-й розряд порту PD
.equ STOP = 1                          ;1-й розряд порту PD
.org $000
rjmp init
.**ініціалізація* **
INIT: ldi reg_led,0xFE                 ;скидання reg_led.0 для включення
LED0
sec                                    ;C=l
set                                    ;T=1 – прапорець напряму
ser temp                               ; ініціалізація
out DDRB, temp                         ; порту PB на вивід
out PORTB, temp                        ; погасити СД
clr temp                               ;ініціалізація
out DDRD, temp                         ;порту PD на введення
ldi temp, 0x03                         ;включення «підтягувальних»
out PORTD, temp                        ; резисторів до ліній 0 і 1 порту PD
WAITSTART:                             ;очікування
sbic PIND,START                        ;натискання
rjmp WAITSTART                         ; кнопки START
LOOP: PORTB, reg_led                   ;включення СД
;***3атримка (два вкладені цикли)***
ldi r17, 2
     dl:ldi rl8, 2
    d2:dec rl8
brne d2
dec r17
brne dl

sbic PIND, STOP                        ;якщо натиснута кнопка STOP
rjmp MM                                ; то перехід
rjmp WAITSTART                         ; для перевірки кнопки START
  MM:ser temp                          ;інакше вимикання світлодіодів
out PORTB, temp
brts LEFT                        ;перехід, якщо прапорець T встановлений
sbrs reg_led, 0                        ;пропуск наступної команди, якщо
; 0-й розряд reg_led встановлений
set                                    ;T=1 – перемикання прапорця
ror reg_led                            ;зсув reg_led управо
rjmp LOOP
```

```
LEFT: sbrs reg_led,7          ;пропустити наступну команду, якщо
; 7-й розряд reg_led встановлений
clt                           ;T=0 – перемикання прапорця
rol reg_led                   ;зсув reg_led вліво
rjmp LOOP
```

2.5 Контрольні запитання та завдання

1. Яка роль «підтягувальних» резисторів? Яким чином можна ввімкнути і вимкнути резистори?

2. Як задається альтернативна функція розряду порту?

3. Які лінії портів мікроконтролера можна використовувати для зовнішніх переривань? Які адреси векторів зовнішніх переривань? Як управляти зовнішніми перериваннями? Яке з них має більш високий пріоритет?

4. Яким сигналом (логічний 0 або логічна 1) можна ввімкнути світлодіод STK500? Як має виглядати біт порту PINx.Y в AVR Studio 4 при замиканні (розмиканні) кнопки?

5. Як підвищити точність програмної затримки?

6. Скласти програму, налагодити в AVR Studio і виконати за допомогою STK 500, яка б за натисканням кнопки SW0 на вході PD.6 порту D засвічувала б світлодіоди на виході порту B, а за натисканням кнопки SW1 на вході PD.7 порту D гасила б їх. Варіанти комбінації станів світлодіодів вказані в таблиці.

№ вар.	Задана комбінація світло діодів							
	LED7	LED6	LED5	LED4	LED3	LED2	LED1	LED0
1	1	1	0	0	1	1	0	0
2	0	1	1	0	0	1	1	0
3	0	0	1	1	0	0	1	1
4	0	1	0	1	0	1	0	1
5	1	1	1	0	0	0	1	1
6	0	0	0	1	1	1	0	0
7	1	0	0	1	0	0	1	1

ЛАБОРАТОРНА РОБОТА № 3
Організація переривань в мікроконтролерах AVR

Мета роботи – вивчення принципів програмування задач управління з використанням механізму переривань мікроконтролерів.

3.1 Порядок виконання роботи

3.1.1 Ознайомитися з призначенням системи переривань.

3.1.2 Ознайомитися з принципами програмування функцій з обробкою переривань.

3.1.3 Записати і налагодити в середовищі AVR Studio задану програму, завантажити її в стенд STK500 і виконати.

3.1.4 Скласти звіт про виконану роботу.

3.2 Теоретичні відомості

3.2.1 Призначення системи переривань

Важливим елементом мікроконтролера є система переривань. Ця система наявна в будь-якому сучасному мікроконтролері. Вона також є в усіх мікроконтролерах AVR. Система переривань мікроконтролера обслуговує декілька джерел переривань. Кількість джерел переривань для різних мікроконтролерів є різною.

Основне призначення системи переривань – організація взаємодії всіх засобів мікроконтролера при вирішенні прикладних задач. У всіх моделях мікроконтролерів AVR містяться засоби апаратної, а, отже, паралельної реалізації різноманітних стандартних задач, які мають вирішуватися при управлінні технічними об’єктами. Всі апаратні засоби мікроконтролера (процесор, таймери, інтерфейси, АЦП і т. п.) можуть працювати і працюють паралельно та незалежно один від одного. Очевидно, що можливість паралельної реалізації декількох функцій вельми істотно розширює можливості мікроконтролера і збільшує швидкодію. В той же час кожен з пристроїв забезпечує виконання будь-яких обмежених функцій, які є тільки елементами загальної прикладної задачі. Ця спільна задача на певних етапах вимагає обміну даними між пристроями, запуску або зупинки окремих процедур, зміни параметрів або режимів роботи і т. ін. Система переривань є необхідним і найважливішим елементом управління роботою апаратних засобів.

Система векторів переривань дозволяє виділити моменти часу, коли завершуються чергові операції й потрібна певна реакція інших елементів мікроконтролера для продовження роботи.

Саме процедури обробки переривань в мікроконтролерах AVR дозволяють виконувати досить гнучкий, програмно-керований контроль

реалізованих функцій. Засоби керування роботою апаратних засобів в обробці переривань перераховані нижче:

• встановлені пріоритети векторів переривань – при одночасному надходженні декількох запитів переривань спочатку викликається вектор переривання з найменшою адресою;

• програмне управління прапорцем загального дозволу переривання (прапорець I регістру SREG) – в будь-яких програмах на довільних етапах за допомогою цього прапорця можна програмно забороняти або дозволяти обробку всіх переривань;

• апаратне керування прапорцем загального дозволу переривання – при виклику будь-якого вектора переривання прапорець глобального дозволу I очищається апаратно і відновлюється (командою reti) при завершенні обробки переривання (крім того, підпрограма обробки будь-якого вектора переривання також може програмно змінювати стан прапорця I, дозволяючи обробку інших переривань);

• вектори переривань містять у відповідних регістрах введення/виведення індивідуальні прапорці дозволу (маскування) – це дозволяє програмно дозволяти або забороняти виклик кожного з векторів переривань на будь-яких етапах роботи незалежно від інших векторів;

• прапорці виклику векторів переривань в регістрах введення/виведення очищаються апаратно при зверненні до підпрограми обробки переривання;

• прапорці виклику векторів переривань можуть очищатися програмно записом одиниці в ці біти регістрів введення/виведення – це дозволяє, за необхідності, скасовувати виклик підпрограм обробки переривань.

Стандартний розподіл паралельно реалізованих в мікроконтролері функцій для керування технічним об’єктом може бути таким, як описано нижче (кожен пункт в даному прикладі – окремий паралельно реалізований процес).

1. АЦП виконує перетворення аналогового сигналу, що надходить від датчика (час перетворення близько 100 мкс, за цей період процесор може виконати декілька сот команд робочої програми).

2. Процесор, виконуючи команди основної програми, робить обробку кодів, що надійшли раніше від АЦП і інших інтерфейсів мікроконтролера, та формує нові значення сигналів керування і даних для індикації.

3. Паралельні порти введення/виведення забезпечують виведення сформованих раніше сигналів керування та індикації.

4. Таймери формують сигнали з заданими раніше в програмі часовими характеристиками і/або в режимі модулятора ШІМ видають сигнали управління виконавчими пристроями.

5. Інтерфейс SPI передає отримані від процесора дані для управління символьним або матричним індикатором.

6. Інтерфейс UART реалізує обмін даними з COM-портом персонального комп'ютера для координації роботи мікроконтролера з іншими пристроями.

7. Зовнішні переривання забезпечують прийом і обробку сигналів цифрових датчиків і сигналів управління, що вимагають швидкої реакції мікроконтролера (наприклад, сигнал аварійної зупинки системи управління).

Очевидно, що перераховані процеси асинхронні один відносно одного, кожен з пристроїв реалізує задані функції зі своїми режимами і часовими межами. Основну координувальну роль може виконувати тільки процесор, забезпечуючи необхідний обмін даними між пристроями і керуючи їх режимами та параметрами через регістри введення/виведення. Основна вимога, яка висувається до організації взаємодії, – безконфліктний доступ до ресурсів, оскільки різні засоби мікроконтролера можуть одночасно виставляти запити на доступ до одних і тих же ресурсів. Коректне управління має забезпечуватися навіть за умови зниження ефективності роботи окремих апаратних засобів. Необхідні координувальні функції мають бути передбачені в алгоритмі основної робочої програми і можуть бути виконані завдяки ефективній системі обробки переривань. Для реалізації координувальних функцій необхідні певні, в першу чергу програмні, ресурси мікроконтролера, ці функції зазвичай досить складні і є найважливішою частиною загального алгоритму роботи.

3.2.2 Механізм реалізації переривань в мікроконтролерах

Переривання – це запуск спеціальної підпрограми (званої «обробником переривання» чи «програмою обслуговування переривання»), і викликається воно сигналом апаратури. На час виконання цієї підпрограми реалізація поточної програми зупиняється (рис. 3.1). Термін «запит на переривання» (interrupt request) використовується тому, що іноді програма відмовляється підтвердити переривання і запустити обробник переривання негайно.

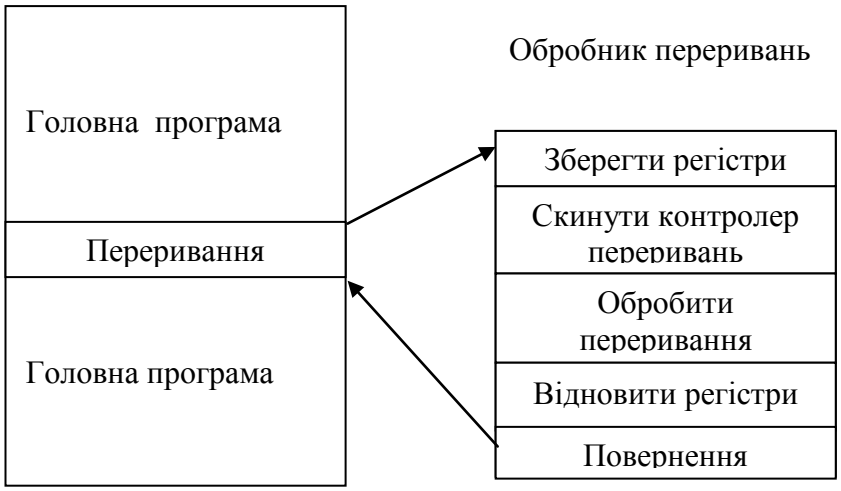


Рисунок 3.1 – Послідовність дій під час реалізації переривання

МК може не реагувати на переривання, поки не завершиться виконання поточної задачі – це реалізується шляхом заборони (маскування) обслуговування запиту переривання. Після розв'язання задачі можливий один з двох варіантів: скидання маски і дозвіл обслуговування переривання, що приведе до виклику обробника переривання, чи аналіз значення бітів, що вказують на надходження запитів переривання і безпосереднє виконання програми обслуговування без виклику обробника переривання. Такий метод обробки переривання використовується, коли потрібно забезпечити заданий час виконання основної програми, оскільки будь-яке переривання може порушити реалізацію необхідного інтерфейсу.

Обробник завжди забезпечує нижчевказану послідовність дій.

1. Зберегти вміст регістрів контексту.
2. Скинути контролер переривань і устаткування, що викликало запит.
3. Обробити дані.
4. Відновити вміст регістрів контексту.
5. Повернутися до перерваної програми.

Регістри контексту – це регістри, що визначають поточний стан виконання основної програми. Зазвичай до них відносять лічильник команд ЛК, регістри стану і акумулятор. Інші регістри МК, наприклад, індексні регістри, можуть бути використані в процесі обробки переривання, тому їхній вміст також необхідно зберегти. Всі інші регістри є специфічними для конкретного типу МК і його застосування.

Після скидання у вихідний стан контролер переривань готовий сприймати наступний запит, а устаткування, що викликає переривання, готове надсилати запит, коли виникають відповідні причини. Якщо надійде новий запит переривання, то регістр маскування переривань процесора запобіжить обробці переривання, але регістр стану переривань зафіксує цей запит, який буде очікувати свого обслуговування. Після завершення обслуговування поточного переривання маска переривання буде скинута, і запит, що надійшов, знову надійде на обробку. Вкладені переривання складно виконати деяким типам МК, які не мають стека. Ці переривання так само можуть викликати проблеми, пов'язані з переповненням стека.

Іноді МК може швидко відреагувати на запит переривання, прийнявши необхідні дані, які будуть потім використані після вирішення поточної задачі. Це реалізується шляхом збереження надійшовших даних в масиві пам'яті та наступної їх обробки, коли виконання вихідної програми буде завершено. Такий спосіб обслуговування є гарним компромісом між негайною повною обробкою переривання, що може потребувати багато часу, та ігноруванням переривання, що може призвести до втрати інформації про подію, яка викликала переривання.

При обробці переривання вміст регістра стану звичайно (але не завжди) автоматично зберігається разом із вмістом лічильника команд

(ЛК) перед обробкою переривання. Це позбавляє від необхідності зберігати його в пам’яті програмними засобами за допомогою команди пересилання, а потім відновлювати при поверненні до вихідної програми. Однак таке автоматичне збереження реалізується не в усіх типах МК.

Якщо вміст регістра стану зберігається перед початком обробки переривання, то за командою повернення виробляється його автоматичне оновлення. Якщо вміст інших регістрів змінюється при виконанні обслуговування переривання, то воно також має бути збережене в пам’яті до зміни і відновлено перед поверненням в основну програму.

«Вектор переривання» – це адреса, яка завантажується в лічильник команд при переході до обробника переривання. Існує кілька типів векторів. Адреса, що завантажується в ЛК при запуску МК (RESET), називається «вектором скиду». Для різних переривань можуть бути задані різні вектори. Але іноді різним перериваннях призначається один вектор. Це не має викликати проблем при роботі з МК, оскільки найчастіше він виконує одну єдину програму. У МК, де апаратна частина добре відома, не має виникнути жодних проблем при спільному використанні векторів переривань.

3.3 Управління перериваннями в мікроконтролерах AVR

У мікроконтролерах AVR за керування перериваннями відповідають, головним чином, чотири регістри:

- GIMSK (General Interrupt Mask Register – GICR для Atmega): дозволяє або забороняє зовнішні переривання за входом INT0/INT1;
- GIFR (General Interrupt Flag Register) – регістр прапорців зовнішніх переривань;
- TIMSK (Timer/Counter Interrupt Mask Register) – регістр маскування переривань від таймера/лічильника T/C0 і T/C1;
- TIFR (Timer/Counter Interrupt Flag Register) – регістр прапорців переривань від таймерів/лічильників.

Про стан переривання сигналізує відповідний прапорець, який встановлюється або скидається в регістрі прапорців. Навіть якщо в регістрі маски переривань встановлений відповідний окремий розряд дозволу переривання, то переривання можуть активізуватися тільки тоді, коли в регістрі стану SREG установлений розряд загального дозволу переривань I (розряд 7). Якщо це має місце і настає переривання, то виконання програми відгалужується за відповідною адресою (див. табл. 1.4) і розряд загального дозволу переривань I в регістрі SREG скидається в стан логічного 0, блокуючи тим самим подальші переривання. Якщо потрібно перервати підпрограму іншим перериванням, то після входу в підпрограму

обробки переривання програма користувача має встановити прапорець I в логічну 1.

Разом з входом в підпрограму обробки переривання апаратно скидається також і відповідний прапорець, що викликав переривання. Деякі прапорці переривань можуть бути скинуті самим користувачем за допомогою встановлення відповідного прапорця в логічну 1.

Регістр GIMSK (GICR). Регістр GIMSK (GICR для Atmega 8515) (рис. 3.2) розташований в області введення/виведення за адресою 003B (адреса в SRAM – 0x005B), використовується для дозволу зовнішніх переривань.

7	6	5	4	3	2	1	0
INT1	INT0	–	–	–	–	–	–

Рисунок 3.2 – Структура регістра GIMSK (GICR)

Якщо розряд INT1/INT0 установлений у логічну 1, то зовнішнє переривання за входом INT1/INT0 буде дозволено до тих пір, поки встановлений у стан 1 розряд I в регістрі стану SREG.

Регістр GIFR. Стан зовнішнього переривання визначається за регістром GIFR (рис. 3.3), який розташований в області введення/виведення за адресою 0x003A (адреса SRAM – 0x005A).

7	6	5	4	3	2	1	0
INTF1	INTF0	–	–	–	–	–	–

Рисунок 3.3 – Структура регістра GIFR

Прапорець INTF1/INTF0 встановлюється в логічну 1, якщо виникає зовнішнє переривання за сигналом на виводі INT1/INT0. При вході в підпрограму обробки переривання цей розряд переводиться апаратно в початковий стан логічного 0.

Регістри TIMSK і TIFR. Регістр TIMSK (рис. 3.4) розташований в області введення/виведення за адресою 0x0039 (адреса в SRAM - 0x0059), використовується для дозволу переривань від таймерів/лічильників.

7	6	5	4	3	2	1	0
TOIE1	OCIE1A	OCIE1B	–	TICIE1	–	TOIE0	–

Рисунок 3.4 – Структура регістра TIMSK

Стан переривань, що стосується таймерів/лічильників мікроконтролерів AVR, визначається за регістром TIFR (рис. 3.5), який

розташований в області введення/виведення за адресою 0x0038 (адреса SRAM – 0x0058).

7	6	5	4	3	2	1	0
TOV1	OCF1A	OCF1B	–	ICF1	–	TOV0	–

Рисунок 3.5 – Структура регістра TIFR

Коли розряд TOIE1 і розряд I в регістрі стану SREG встановлені в логічну 1, то дозволено переривання при переповненні T/C1. У разі переповнення в регістрі TIFR встановлюється прапорець TOV1.

Якщо розряд OCIE1A і розряд I в регістрі стану SREG встановлені в логічну 1, то дозволено переривання при збігу вмісту регістра порівняння A з поточним станом T/C1. У разі збігу в регістрі TIFR встановлюється прапорець OCF1A.

Якщо розряд OCIE1B і розряд I в регістрі стану SREG встановлені в логічну 1, то дозволяється переривання при збігу вмісту регістра порівняння B з поточним станом T/C1. У разі збігу в регістрі TIFR встановлюється прапорець OCF1B.

Якщо розряд TICIE1 і розряд I в регістрі стану SREG встановлені в логічну 1, то дозволяється переривання при виконанні умови захоплення. Коли виникає спрацьовування за захопленням, в регістрі TIFR встановлюється прапорець ICF1.

Якщо розряд TOIE0 і розряд I в регістрі стану SREG встановлені в логічну 1, то дозволяється переривання при переповненні таймера/лічильника T/C0. У такому випадку в регістрі TIFR встановлюється прапорець TOV0.

Встановлення в логічну 1 одного з прапорців в регістрі TIFR веде до переходу за відповідним вектором переривання. При вході в підпрограму обробки переривання прапорець в регістрі TIFR апаратно скидається в логічний 0.

3.4 Обробка переривань в мікроконтролері ATmega 8515

Як входи зовнішніх переривань використовуються входи портів з альтернативною функцією: PD2, PD3 – для переривань INT0, INT1 і PE0 – для переривання INT2 в мікроконтролері ATmega8515. Запити зовнішніх переривань INT0, INT1 можуть бути подані низьким рівнем сигналу переривання (L), переходом від високого рівня сигналу до низького (HL – за негативним фронтом), переходом від низького рівня сигналу до високого (LH – за позитивним фронтом), запит INT2 тільки переходами (LH) і (HL). Залежно від типу запиту в регістрі управління мікроконтролера MCUCR необхідно встановити біти ISCx0 і ISCxl згідно з

табл. 3.1 для кожного з переривань INTx (x = 0,1) і визначити біт ISC2 в регістрі EMCUCR для переривання INT2. При ISC2 = 0 переривання здійснюється за негативним фронтом, при ISC2 = 1 – за позитивним.

Таблиця 3.1 – Таблиця вибору типу запиту

ISCx1	ISCx0	Тип запиту
0	0	L
0	1	-
1	0	HL
1	1	LH

Далі підготуємо програму перемикання світлодіодів з використанням зовнішнього переривання від кнопки STOP. Згідно з поставленими умовами, в блок ініціалізації мікроконтролера внесемо низку змін:

- додамо вектор переривань;
- покажчик стека встановимо на останній осередок ОЗП;
- дозволимо зовнішнє переривання INTO (за сигналом 0 на лінії 2 порту PD) і переривання взагалі.

Оскільки зовнішнє переривання INTO подано сигналом на вході порту PD2, як STOP використовуємо кнопку SW2 і програмуємо PD2 на введення. Натискання кнопки STOP викликає переривання.

Приклад програми наведено нижче. Затримка подана підпрограмою DELAY.

Програма 3.1

```
;*****************************************
;Програма  для почергового перемикання світлодіодів при натисканні
;на кнопку START (SW0). Після натискання на кнопку STOP
; перемикання припиняється і поновлюється (SW2)
;з місця зупинки при повторному натисканні на кнопку START
;*************************************
.include "m8515def.inc"          ;файл визначень для ATmega8515
.def temp = r16                   ;тимчасовий регістр
.def reg_led = r20                ;стан регістра світлодіодів
.equ START = 0                    ;0-й розряд порту PD
;***Вектори переривань***
        .org $000
rjmp INIT                        ;обробка скидання
.org $001
rjmp STOP_PRESSED                ;обробка зовнішнього переривання
                                  ; INT0 (STOP)
        ;*** Ініціалізація МК ***
INIT: ldi reg_led,0xFE
ldi temp,$5F                      ; встановлення
ldi SPL, temp                     ; покажчика стека
```

```
ldi temp, $02                    ; на останню
out SPH, temp                    ; комірку ОЗП
sec                              ;C=1
set                              ;T=1
ser temp                         ;ініціалізація виводів
out DDRB, temp                   ;порту PB на вивід
out PORTB, temp                  ;погасити світлодіоди
clr temp                         ;ініціалізація
out DDRD, temp                   ;порту PD на введення
ldi temp, 0x05                   ; ввімкнення підтягувальних
out PORTD, temp                  ;резисторів порту PD
ldi temp, 0x7F                   ;дозвіл переривання INTO
out GICR, temp                   ; (6-й біт  GICR)
ldi temp, 0x00                   ;обробка переривання INTO
out MCUCR, temp                  ;за низьким рівнем
sei                              ; дозвіл переривань
WAITSTART: sbic PIND, START      ;очікування натискання
rjmp WAITSTART                   ; кнопки START
LOOP:  out PORTB, reg_led        ; ввімкнення світлодіодів
rcall DELAY                      ;затримка
ser temp                         ;вимикання
out PORTB, temp                  ;світлодіодів
brts LEFT                  ;перехід, якщо прапорець T встановлений
sbrs reg_led, 0                  ;пропуск наступної команди, якщо
                                 ; 0-й розряд reg_led встановлений
set                              ;T=1
ror reg_led                      ;зсув reg_led управо
rjmp LOOP
LEFT: sbrs reg_led,7             ;пропуск наступної команди, якщо
                                 ;1-й розряд reg_led встановлений
clt                              ;T=0
rol reg_led                      ;зсув reg_led вліво
rjmp LOOP
;***Обробка переривання від кнопки STOP***
STOP_PRESSED:
WAITSTART_2:                     ;очікування
sbic PIND, START                 ;натискання
rjmp WAITSTART_2                 ; кнопки START
reti
;*** Затримка ***
DELAY: ldi r17, 128
d1: ldi r18, 100
d2: dec r18
brne d2
```

```
dec r17
brne d1
ret
```

3.5 Контрольні запитання і завдання

1. Яке основне призначення системи переривань мікроконтролера?
2. Що називаєтьсяя перериванням?
3. Які регістри мікроконтролера визначають поточний стан виконання основної програми?
4. Що називається вектором переривання.
5. Які регістри використовують для управління перериваннями в мікроконтролерах AVR?
6. Опишіть структуру регістра GIMCK (GICR) мікроконтролера Atmega 8515.
7. Опишіть структуру регістра GIFR мікроконтролера Atmega 8515.
8. Опишіть структуру регістра TIMCK мікроконтролера Atmega 8515.
9. Опишіть структуру регістра TIFR мікроконтролера Atmega 8515.
10. Біт якого регістра дає загальний дозвіл переривань?
11. Якими рівнями сигналів можуть бути представлені запити на входах зовнішніх переривань INT0, INT1 I INT2 мікроконтролера Atmega 8515?
12. Які біти регістра MCUCR задають тип сигналу запиту на входах зовнішніх переривань?
13. **Завдання**. Скласти програму мовою Асемблер для мікроконтролера Atmega 8515, що реалізує такі дії, як програма 3.1, але затримку організувати на таймері/лічильникові T/C1. Вважати, що частота тактового генератора вибрана 1024 кГц (1,024 мГц).

Час затримки для кожного варіанта вказаний в таблиці 3.2.

Таблиця 3.2 –Варіанти завдань

№ вар.	1	2	3	4	5	6
Час затр.	0,25 сек	0,5 сек	0,75 сек	1,0 сек	1,25 сек	1,5 сек

ЛАБОРАТОРНА РОБОТА № 4
Таймери мікроконтролера AVR і їх програмування

Мета роботи – вивчення принципів роботи таймерів і їх програмування.

4.1 Порядок виконання роботи

4.1.1 Ознайомитися з принципами побудови та роботи таймерів/лічильників.

4.1.2 Ознайомитися з принципами програмування функцій МК з використанням таймерів/лічильників.

4.1.3 Записати і налагодити в середовищі AVR Studio задану програму, завантажити її в стенд STK500 і виконати.

4.1.4 Скласти звіт про виконану роботу.

4.2 Теоретичні відомості

4.2.1 Таймери мікроконтролерів ATx8515

Мікроконтролери AVR, залежно від класу (Tiny, Mega, Xmega) і типу моделі, мають у своєму складі від одного до трьох таймерів/лічильників загального призначення – T/C0, T/C1 та T/C2.

У мікроконтролера Atmega 8515 є 2 таймери/лічильники:

1. Таймер/лічильник T/C0 (8-розрядний);
2. Таймер/лічильник T/C1 (16-розрядний).

Залежно від моделі МК T/C0 може виконувати такі функції:

- відлік та вимірювання часових інтервалів;
- лічильник зовнішніх подій;
- генерація ШІМ сигналів фіксованої розрядності;
- асинхронний (щодо тактового сигналу мікроконтролера) режим годинника реального часу.

Таймер/лічильник T/C1 реалізує такі функції:

- відлік та вимірювання часових інтервалів;
- лічильник зовнішніх подій;
- запам’ятовування свого значення за зовнішнім сигналом (захоплення);
- генерація ШІМ сигналів змінної розрядності.

Кожен таймер/лічильник використовує один або більше виводів мікроконтролера. Ці виводи можуть бути або лініями портів введення/виведення з альтернативною функцією, або виділеними виводами мікроконтролера. Всі виводи мікроконтролерів ATx8515, що їх відносять до таймерів/лічильників, і їх функції наведено в табл.4.1.

Таблиця 4.1 – Виводи, що використовуються таймерами/лічильниками загального призначення

Назва	ATx8515	STK500	Опис
(OC0)T0	PB0	PB0	Вхід зовнішнього сигналу таймера T0
T1	PB1	PB1	Вхід зовнішнього сигналу таймера T1
ICP	ICP/PEO*	PE0	Вхід захоплення таймера T1
OC1A	PD5	PD5	Вихід схеми порівняння таймера T1
OC1B	OC1B/PE2*	PE2	Вихід схеми порівняння таймера T1

* У мікроконтролері AT90S8515 виділений вивід, в ATmega 8515 – вивід порту PE.

При використанні ліній портів введення/виведення необхідно сконфігурувати виводи відповідно до їх функціонального призначення (вхід або вихід).

Управління роботою таймерів та обробка їх сигналів проводиться через відповідні регістри файлу регістрів введення/виведення. Регістр масок TIMSK і регістр прапорців переривань TIFR – загальні для всіх таймерів і програмно доступні для читання та запису. Інші регістри управління індивідуальні для кожного таймера. Взаємодію з таймерами при формуванні заданих часових інтервалів практично завжди необхідно проводити процедурами введення/виведення за відповідними перериваннями. Основне призначення регістрів TIMSK і TIFR – управління перериваннями таймерів.

Регістр TIMSK (адреса $39) містить біти дозволу, а регістр TIFR (адреса $38) – прапорці всіх переривань, що виробляються таймерами.

Структура регістра масок переривань таймерів/лічильників TIMSK (Timer/Counter Interrupt Mask Register) зображена на рис. 4.2, а призначення його розрядів наведено в таблиці 4.2.

7	6	5	4	3	2	1	0
TOIE1	OCIE1A	OCIE1B	-	TICIE1	-	TOIE0	OCIE0
R/W	R/W	R/W	R	R/W	R	R/W	R/W

Рисунок 4.2 – Регістр масок переривань таймера TIMSK

Таблиця 4.2 – Призначення розрядів регістра TIMSK

Ім’я прапорця	Опис призначення прапорця
TOIE1	Прапорець дозволу перивання за подією переповнення T/C1
OCIE1A	Прапорець дозволу перивання за подією « Збіг A» T/C1
OCIE1B	Прапорець дозволу перивання за подією « Збіг B» T/C1
TICIE1	Прапорець дозволу перивання за подією « Захоплення» в T/C1
TOIE0	Прапорець дозволу перивання за подією переповнення T/C0
OCIE0	Прапорець дозволу перивання за подією « Збіг» T/C0

Для дозволу якого-небудь зазначеного в таблиці 4.2 переривання від Т/Сх необхідно встановити в «1» відповідний розряд регістра TIMSK і прапорець I регістра SREG. У наступному прикладі показано, як дозволити переривання виходу порівняння (за подією «Збіг А» Т/С1:

```
ldi r16,1<<OCIE1A
out TIMSK,r16       ;Дозвіл переривання за виходом порівняння Т/С1
sei         ;Дозвіл загального переривання встановленням біта I регістра SREG
```

У випадку виникнення переривань від Т/С0 або Т/С1 у відповідних їм розрядах регістра TIFR встановлюються прапорці (логічні «1»). Структура регістра TIFR наведена на рис.4.3, а опис його прапорців – в таблиці 4.3.

7	6	5	4	3	2	1	0
TOV1	OCF1A	OCF1B	-	ICF1	-	TOV0	OCF0
R/W	R/W	R/W	R	R/W	R	R/W	R/W

Рисунок 4.3 – Регістр прапорців переривань TIFR таймерів/лічильників

Таблиця 4.3 – Опис прапорців регістра TIFR

Ім’я прапорця	Опис призначення прапорця
TOV1	Прапорець переривання, встановлений подією переповнення Т/С1
OCF1A	Прапорець переривання, встановлений подією «Збіг А» Т/С1
OCF1B	Прапорець переривання за подією «Збіг В» Т/С1
ICF1	Прапорець переривання за подією «Захоплення» в Т/С1
TOV0	Прапорець переривання за подією переповнення Т/С0
OCF0	Прапорець переривання за подією «Збіг» Т/С0

За тактовий сигнал (сигнал синхронізації) CLK таймерів/лічильників можуть використовуватись:

- тактовий сигнал мікроконтролера;
- зовнішній тактовий сигнал на входах Т0 (PB0), Т1 (PB1);
- масштабований тактовий сигнал з переддільника таймера.

Для скидання переддільника використовується розряд PSR10 регістра SFIOR (0-ий розряд цього регістра).

Вибір режиму роботи (джерела тактового сигналу), а також запуск і зупинка таймера/лічильника здійснюються за допомогою розрядів CS02–CS00 регістра керування таймером TCCR0 (табл. 4.4). Відповідність між станом цих розрядів і режимом роботи таймера/лічильника наведено в табл. 4.5. Інші розряди регістра доступні тільки для читання і містять 0.

Таблиця 4.4 – Формат регістра керування таймером TCCR0

№ розряду	7	6	5	4	3	2	1	0
Ім’я	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

Таблиця 4.5 – Задання режимів таймера/лічильника регістром TCCR0

CS02	CS01	CS00	Джерело тактового сигналу
0	0	0	Таймер/лічильник зупинений
0	0	1	СК (тактовий сигнал мікроконтролера)
0	1	0	СК/8
0	1	1	СК/64
1	0	0	СК/256
1	0	1	СК/1024
1	1	0	Вивід T0, інкремент лічильника виконується за спадним фронтом імпульсів
1	1	1	Вивід T0, інкремент лічильника виконується за позитивним фронтом імпульсів

У всіх мікроконтролерах сім’ї AVR наявний також сторожовий таймер, який є неодмінним атрибутом усіх сучасних мікроконтролерів. Цей таймер використовується для запобігання зациклень програми.

4.2.2 Таймер/лічильник T/C0 мікроконтролера ATmega8515

Структурна схема таймера/лічильника T/C0 мікроконтролера ATmega8515 наведена на рис. 4.4.

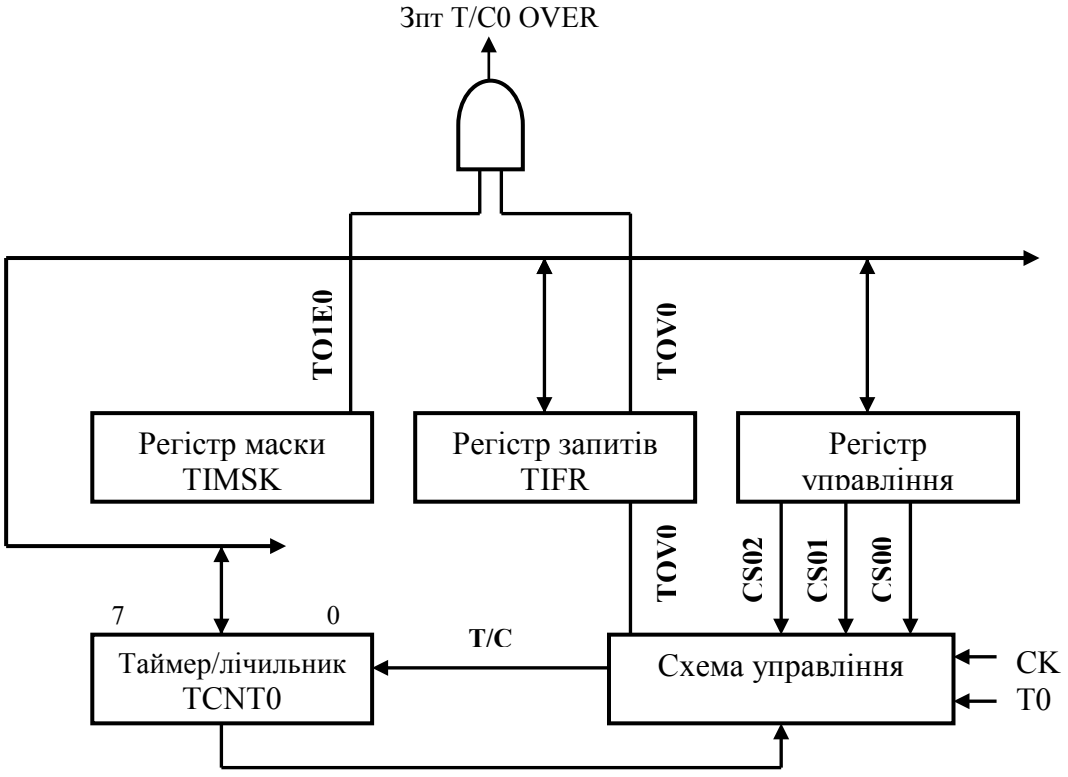


Рисунок 4.4 – Структурна схема таймера/лічильника

Таймер містить базовий лічильник TCNT0 ($33), регістр управління TCCR0 ($32), регістр порівняння TCCR0 ($31) і схему управління. Крім того, до його складу входять по одному розряду регістра запитів переривань TIFR і маски переривань TIMSK.

Тактовий сигнал мікроконтролера СК надходить в перерахункову схему (дільник) ПС, що являє собою десятирозрядний лічильник, де виконується ділення частоти тактового сигналу на 8, 64, 256 і 1024. Сигнали з чотирьох виходів перерахункової схеми надходять у схему управління СУ (мультиплексор). При наявності в мікроконтролері таймера/лічильника Т/С1 ці ж сигнали надходять в Т/С1.

У схему управління надходять також тактовий сигнал СК і сигнал із зовнішнього джерела, що приймається на вхід Т0. За вхід Т0 у мікроконтролерів типу 1200, 2313 і 4433 використовується вивід порту PD4, у мікроконтролерів типу 2323, 2343 і серії ATtny – вивід порту PB2, а у мікроконтролерів типу 8515, 8535 і m163 – вивід порту PB0.

Таймер/лічильникТ/С0 в ATmega 8515 має 2 запити на переривання:

- TOV0, переривання за подією переповнення;
- OCF0, переривання за порівнянням при рівності регістрів TCNT0 і OCR0 (регістра-лічильника і регістра порівнянь).

Прапорці обох переривань знаходяться в регістрі TIFR, а дозвіл чи заборона цих переривань здійснюється встановленням чи скиданням відповідних розрядів регістра (TOIE0 чи OCIE відповідно) регістра TIMSK.

Лічильник TCNT0 доступний в будь-який момент часу як для читання, так і для запису. При запису в лічильник TCNT0 під час його роботи рахунок буде продовжений в наступному за командою запису машинному циклі. Після подачі напруги живлення лічильник TCNT0 переходить в нульовий стан.

Регістр-лічильник TCNT0 збільшує або зменшує свій вміст на 1 за кожним тактовим імпульсом залежно від режиму роботи. При переході регістра TCNT0 таймера/лічильника зі стану $ FF у стан $ 00 встановлюється в 1 прапорець TOV0 в регістрі TIFR і генерується запит на переривання. Дозвіл переривання здійснюється встановленням в 1 розряду TOIE0 регістра маски TIMSK. Також прапорець загального дозволу переривання I регістра SREG мікроконтролера також має бути встановлений в 1.

Регістр порівняння OCR0 (на рис. 4.4 не показаний, під’єднаний через схему порівняння до регістра TCNT0) містить двійкове слово (число), яке у кожному циклі роботи порівнюється з вмістом регістра TCNT0. У разі збігу вмісту цих регістрів в наступному машинному циклі встановлюється прапорець OCF0 регістра TIFR і генерується переривання, якщо воно дозволено. Крім цього, може змінитися стан виводу OC0 (PB0) мікроконтролера залежно від налаштування розрядів FOC0 та COM00, COM01 регістра TCCR0. Опис функцій цих розрядів наведено в таблиці

4.6. Для цього розряд регістра DDRB.0 має бути налаштований на виведення (встановлений в 1).

Таблиця 4.6 – Функції окремих розрядів регістра TCCR0

Ім'я прапорця	Опис призначення прапора
FOC0	Примусова зміна стану виводу OC0 (PB0). Якщо FOC0=1, то стан змінюється відповідно до встановлених значень COM00,COM0 1
WGM01, WGM00	Режим роботи T/C0: 0,0 –Normal; 0,1 – Phase correct PWM, 1,0 – CTC (скидання за збігом)
COM01, COM00	Визначають поведінку виводу OC0 при настанні події «збіг» (табл. 4.7)

Таблиця 4.7 – Керування поведінкою виводу OC0

COM01	C0M00	Опис
0	0	Вивід відключений від таймера/лічильника
0	1	Стан виводу змінюється на протилежний
1	0	Вивід скидається в 0
1	1	Вивід скидається в 1

При використанні таймера/лічильника в режимі рахунку зовнішніх подій необхідно пам'ятати, що сигнал, присутній на виведенні T0, синхронізується частотою тактового генератора мікроконтролера (стан виведення T0 зчитується по наростаючому фронті внутрішнього тактового сигналу). У зв'язку з цим для забезпечення коректної роботи таймера від зовнішнього сигналу проміжок часу між сусідніми імпульсами повинен бути більше періоду тактового сигналу мікроконтролера. Інкремент вмісту таймера/лічильника при роботі в режимі рахунку зовнішніх подій проводиться навіть у тому випадку, якщо висновок TO налаштований як вихід. Ця особливість дає користувачеві можливість програмно керувати процесом рахунку.

Приклад програмного керування таймером/лічильником T/C0

```
ml: clr rl                ; очищення r1 (r1=0),
m2: out TCCRO, rl         ; пересилання байта з rl (0) в регістр
                          ; управління TCCR0,
M3: out TCNTO, rl         ; пересилання байта з rl (0) в регістр таймера TCNT0,
m4: Idi r 16, 1           ; завантаження в r16 константи 1,
m5: out TIMSK, r 16       ; пересилання в регістр TIMSK константи 1
                          ; для дозволу переривания таймера TCNT0,
mб: Idi r 16, 3           ; завантаження в r 16 константи управління таймера TCNT0,
m7: out TCCRO, rl6        ; запуск таймера 0
```

Команди m1–m3 виконують підготовку таймера 0 для роботи (m2 – зупинка таймера, m3 – скидання лічильника в нульовий стан). Команди

m4, m5 необхідні для дозволу переривання таймера 0. Команди m6, m7 виробляють запуск таймера 0. Після виконання команди m7 таймер починає працювати: частота вхідного сигналу $f_{mx}/64$ відповідно до констант управління, що дорівнює 3, через кожні 256 вхідних імпульсів формується прапорець переривання TOV0. Отже, виклик відповідногоо вектора переривання буде проводитися кожні 64 * 256 = 16384 такти роботи мікроконтролера.

4.3 Приклади програмування таймера/лічильника T/C0

Режим лічильника. Підготувати програму для дослідження таймера/лічильника T/C0 в режимі лічильника подій. Подією в даному випадку може бути замикання однієї з кнопок SWx на платі STK500. Результат роботи програми відобразити засобами індикації. За допомогою 10-проводового шлейфу підключимо виводи порту PB до виводів кнопок SW0–SW7 (в даному випадку буде використано лише вивід PB0 – вхід зовнішнього сигналу таймера T/C0). За допомогою другого 10-проводового шлейфу з'єднуємо виводи порту PD з виводами світлодіодів LED0–LED7. Програмуємо вивід PB0 на введення, всі виводи порту PD – на виведення. Налаштовуємо таймер на режим підрахування зовнішніх подій (натискання кнопки SW0). Після декількох, наприклад чотирьох, натискань має статися переповнення таймера (отже, початкове значення лічильника TCNT0 – $FC) і виклик обробника переривань. Обробник має включити світлодіоди, показуючи, що програма виконана коректно, і наново ініціалізувати лічильник таймера для продовження роботи, якщо планується кількаразове повторення програми. Час включення світлодіодів встановимо, використовуючи підпрограму затримки.

Програма 4.1

```
        ; Программа для МК ATmega8515:
        ; демонстрація работи таймера T0 в режимі лічильника  подій
        ; подія – натискання кнопки SW0
        ;  світлодіоди включаються після четвертого натискання на кнопку SW0
        ; з'єднати  шлейфами: порт PB-SW, порт PD-LED
        ;*******
;************************************************************
      .include "m8515def.inc"                ;файл визначень ATmega8515
      .def temp = rl6                        ; тимчасовий  регістр

        ; ******Таблиця векторів переривань ********
       .org $000
       rjmp INIT                                 ; перехід на обробку скиду
       .org $007
      rjmp T0_OVF                          ; перехід на обробку переповнення таймера

       ; ****** ініціалізація МК **********
```

```
INIT:   ldi temp,  low(RAMEND)          ; установлення
        out SPL,  temp                  ; вказівника стека
        ldi temp,  high(RAMEND)         ; на останню
        out SPH, temp                   ; комірку SRAM (ОЗП)
        clr temp                        ; ініціалізація порту PB
         out DDRB, temp                 ;  на введення
         ldi temp, 0x01                 ; включення
         out PORTB, temp                ; підтягувального резистора до PB0
         ser temp                       ; ініціалізація порту PD
         out DDRD, temp                 ; на виведення
         out PORTD, temp                ; виключення світлодіодів
         ldi temp, 0x20                 ; біт SE=1 – дозвіл переходу
         out MCUCR, temp                ; в режим Idle (сплячий режим)

         ;******налаштування таймера T0 на режим лічильника подій*******
         ldi temp, 0x02                 ; дозвіл переривання за
         out TIMSK, temp                ; переповненням таймера
         ldi temp,0x07                  ; перемикання таймера
        cbi PORTB, 0                    ; за позитивним фронтом на PB0
        out TCCR0, temp
        sei                             ; дозвіл переривань
        ldi temp, 0xFC                  ; занесення  в лічильник $FC=-04 для
        out TCNT0, temp                 ; відліку чотирьох натискань
LOOP:  sleep                            ;перехід в режим пониженого
        nop                             ; енергоспоживання
        rjmp LOOP

        ;******обробка переривання при переповненнні таймера T0*******
T0_OVF: clr temp
        out PORTD, temp                 ; включення світлодіодів
        rcall DELAY                     ; затримка
        ser temp
        out PORTD, temp                 ; виключеннння світлодіодів
        ldi temp,0xFC                   ;$FC=-04 для
        out TCNT0, temp                 ; відліку чотирьох натискань
        reti

        ;*** Затримка ***
DELAY: ldi rl9,10
        ldi r20,255
        ldi r21,255
    dd: dec r21
        brne dd
        dec r20
        brne dd
        dec rl9
        brne dd
        ret
```

Результат роботи програми буде такий. При четвертому натисканні на кнопку SW0 загоряються всі світлодіоди (у разі брязкоту контактів світлодіоди можуть включитися раніше). Тривалість часу, протягом якого вони горять, визначається затримкою DELAY. Далі дії можуть бути повторені. Як вже було сказано в теоретичній частині, події для таймера/лічильника T0 можна генерувати програмно. Для цього необхідно налаштувати вивід PB0 як вихід. В даному випадку інкремент лічильника буде відбуватися після виконання команд програми, що емулюють позитивний або негативний фронт, як цього вимагає налаштування таймера:

```
; позитивний (наростаючий) фронт сигналу на PB0
 cbi PORTB, 0
 sbi PORTB, 0
 ; негативний (спадний) фронт сигналу на PB0
 sbi PORTB, 0
 cbi PORTB, 0
```

Режим таймера. Підготувати програму для дослідження таймера лічильника T/C0 в режимі таймера. Для наочності роботи таймера в цьому режимі можна запрограмувати такі дії: при натисканні на першу кнопку на вхід таймера надходять сигнали з частотою, що дорівнює частоті тактового генератора СК, при натисканні на другу кнопку на вхід таймера надходять сигнали з частотою, наприклад, СК/8. В обох випадках відразу після натискання загоряються світлодіоди, а після переповнення таймера і обробки відповідного переривання світлодіоди гаснуть. Таким чином, у другому випадку час світіння світлодіодів буде у вісім разів довший, ніж у першому, що і означає правильність виконання програми. Якщо встановити частоту тактового генератора мікроконтролера такою, що дорівнює 512 Гц, то час включення світлодіодів в першому випадку 0.5 с, у другому – 4 с (0.5×8). (Увага! Частоту тактового генератора необхідно змінювати після програмування мікроконтролера.) На платі STK500 необхідно з'єднати висновки порту PD з кнопками SW0–SW7, виводи порту PB – зі світлодіодами LED0–LED7. Для обробки натискання кнопок використовуємо метод послідовного опитування стану кнопок.

Програма 4.2

```
    .include "m8515def.inc"        ; файл визначень ATmega8515
    .def  temp  =  r16              ;призначення імені temp регістру R16
    .equ  SW2  =  2               ;присвоєння змінній SW константи 2 (2-ий
вивід PD)
    .equ SW3 = 3                   ;3-й вивід порту PD

        ;*****Таблиця векторів преривань******
    .org $000
     rjmp INIT                    ;обробка скиду МК
    .org $007
```

```
  rjmp T0_OVF           ;обробка переповнення таймера T0

     ;***Ініціалізація МК***
INIT: ldi temp,low(RAMEND)  ;установлення
out SPL,temp              ;вказівника стека
ldi temp,high(RAMEND)     ; на останню
out SPH,temp              ; комірку ОЗП
clr temp                  ;ініціалізація порту PD
out DDRD,temp             ; на введення
ldi temp,0x0C             ;включення підтягувальних
out PORTD,temp            ; резисторів порту PD
ser temp                  ;ініціалізація порту PB
out DDRB,temp             ; на виведення
out PORTB,temp            ;виключення світлодіодів

    ;*****Налаштування таймера T0 на режим таймера*****
ldi temp,0x02             ;дозвіл переривання по
out TIMSK,temp            ; переповненню таймера TO
clr temp                  ;таймер T0
out TCCR0,temp            ; зупинено
sei                       ; дозвіл всіх переривань
clr temp                  ;підрахунок імпульсів у таймері (TCNT0)
out TCNT0,temp            ; починається з 0

     ;*****Очікування натискання кнопок*****
WAITSET_0: sbic PIND,SW2    ;перевірка натискання
rjmp WAITSET_1          ; кнопки SW2

    ;*****Обробка натискання кнопки SW2******
ldi temp,0x01             ;установлення частоти тактових сигналів -
CK
rcall LED_ON              ;включення світлодіодів
WAITSET_1: sbic PIND,SW3     ; перевірка натискання
rjmp WAITSET_0            ; кнопки SW3

    ;***** Обробка натискання кнопки SW3 ****
ldi temp,0x02             ;частота сигналів - CK/8
rcall LED_ON              ;включення світлодіодів
rjmp WAITSET_0

;*****Обробка переривання при переповненні таймера T0******
T0_OVF: ser temp          ; виключення
 out PORTB,temp     ; світлодіодів
 clr temp           ; зупинка
 out TCCR0,temp     ; таймера T0
 clr temp           ; скид лічильника
 out TCNT0,temp     ; в нульовий стан (очищення)
 reti               ; вихід з переривання

     ;*****Включення світлодіодів******
LED_ON: out TCCRO,temp    ; налаштування тактової частоти
        clr temp           ; включення
        out PORTB,temp     ; світлодіодів
        ret                ; вихід з підпрограми
```

4.4 Контрольні запитання і завдання

1. Які функції можуть виконувати таймери/лічильники мікроконтролерів?
2. Скільки, зазвичай, є таймерів/лічильників в мікроконтролерах AVR?
3. Зарисуйте структурну схему таймера/лічильника T/C0.
4. Які регістри мікроконтролера використовуються для управління таймерами/лічильниками?
5. Яке призначення регістра масок TIMSK?
6. Опишіть призначення бітів регістра TIMSK.
7. Яке призначення регістра прапорців TIFR?
8. Опишіть призначення бітів регістра TIFR.
9. Опишіть призначення бітів регістра управління TCCR0.
10. Які функції виконує регістр TCNT0 таймера/лічильника T/C0?
11. Які функції регістра OCR0 таймера/лічильника T/C0?
12. До яких ліній портів мікроконтролера Atmega 8515 підключені входи T0 і T1 сигналів зовнішніх подій під час їхнього підрахунку?
13. Який коефіцієнт поділу частоти тактового сигналу можна встановити в лічильнику?

Варіанти завдань на лабораторну роботу

Варіант 1. Змінити програму 4.1, додавши в неї обробку натискання кнопки, що усуває вплив брязкоту.

Варіант 2. Змінити програму 4.1 так, щоб при замиканні кнопки START (SW1) відбулося виведення на світлодіодні індикатори числа зареєстрованих подій.

Варіант 3. Змінити програму 4.1 таким чином, щоб затримка реалізовувалась таймером/лічильником T/C1.

Варіант 4. Підготувати програму за прикладом 4.2 для перевірки роботи таймера/лічильника T/C1 в режимі таймера. Час включення світлодіодів 1 і 8 с.

Варіант 5. Написати програму, яка кожні 50 мс за запитом переривання від таймера прочитує стан кнопкового регістра SW, при замиканні визначає номер замкнутої кнопки і висвічує його на лінійці світлодіодів.

Варіант 6. Скласти програму мовою Асемблер для виконання таких дій:

а) Налаштувати виводи порту B та молодшу половину виводів порту D як виходи, а виводи порту C та старшу половину виводів порту D налаштувати як входи.

б) Видати на виводи порту B число 10101111. Для 1-го, 2-го та 7-го виводів порту C ввімкнути внутрішні підтягувальні резистори. На молодшу половину виводів порту D видати число 1000, а для старшої половини виводів порта D ввімкнути внутрішні підтягувальні резистори.

в) Налаштувати переривання за переповненням будь-якого таймера з частотою 0,1–0,2 Гц при тактовій частоті роботи мікроконтролера 1 МГц.

г) У процесі обробки переривання за переповненням таймера отримати стан порту D та видати його в порт B.

На рис. 4.5 наведено рекомендований алгоритм виконання завдання.

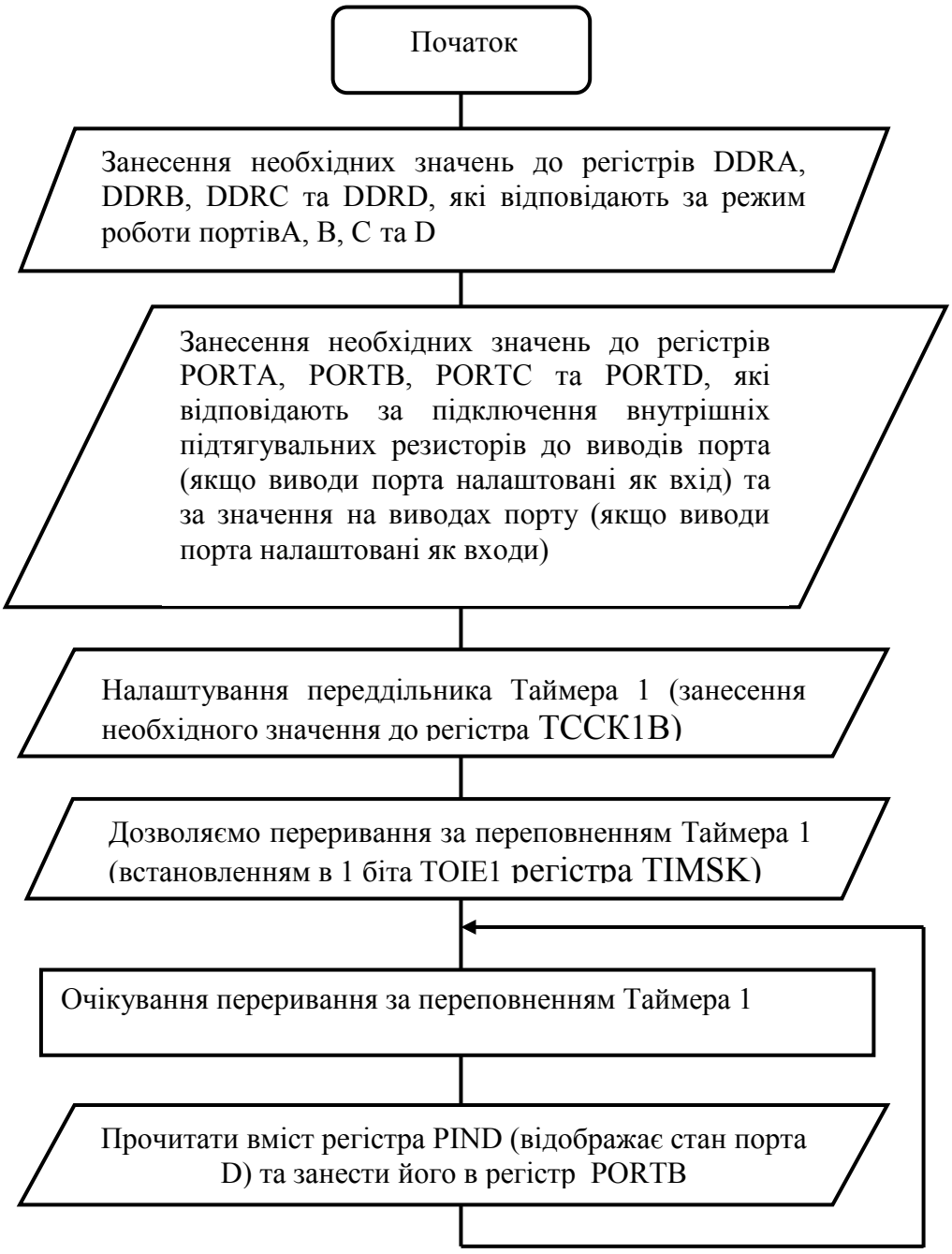


Рисунок 4.5 – Алгоритм виконання завдання

ЛАБОРАТОРНА РОБОТА № 5
Програмування мікроконтролерів AVR мовою Сі в середовищі Code Vision AVR (CV AVR)

Мета роботи – вивчення принципів програмування задач управління на мовах високого рівня

5.1 Порядок виконання роботи

5.1.1 Ознайомитися з принципами роботи в середовищі CV AVR.

5.1.2 Ознайомитися з принципами програмування мікроконтролерів мовою Сі.

5.1.3 Записати та налагодити в середовищі CV AVR задану программу, завантажити її в стенд STK-500 і виконати.

5.1.4 Скласти звіт про виконану роботи.

5.2 Теоретичні відомості

5.2.1 Розробка програм мовою Сі в середовищі CodeVision AVR

Найбільш поширеними компіляторами програм з мови Сі в об'єктний код для сімейства мікроконтролерів типу AVR є компілятор IAR Embedded Workbench фірми «IAR Systems» (Швеція) (http://www.iar.com) комерційного поширення та компілятор CodeVision AVR (Румунія) фірми «HP Info Tech» (http://www.hpinfotech.com) некомерційного використання («freeware»).

CodeVision містить компілятор Сі відповідно до ANSI стандарту, графічну оболонку та автоматичний генератор програм для мікроконтролерів AVR сімейства.

Об'єктні файли компілятора дозволяють здійснювати налагодження та перегляд вмісту змінних за допомогою AVR Studio, починаючи з версії 3.55.

В складі компонентів CodeVision AVR присутня дуже корисна термінальна програма Terminal, яка дозволяє тестувати програми, що використовують передачу даних послідовною лінією.

Компілятор CodeVision AVR може генерувати шаблони програм для роботи з такими зовнішніми компонентами, які можуть бути вказані як опції при створенні проекту програми для CodeVision AVR, а саме:

- стандартним рідкокристалічним індикатором із вбудованим контролером;
- передача даних по послідовній двопроводовій шині 1^2C;
- робота з сенсором температури LM75 фірми Nationfl Semiconductor;
- робота з годинниками реального часу PCF8583 фірми Philips, DS1302 DS1307 фірми Dallas Semiconductor;

- робота з однопроводовим інтерфейсом фірми Dallas Semiconductor;
- робота з сенсорами температури DS1820 nf DS1822 фірми Nationfl Semiconductor;
- робота з сенсором температури/термостатом DS1621 фірми Nationfl Semiconductor;
- робота з енергонезалежною пам'яттю EEPROM DS2430, DS2433 фірми Nationfl Semiconductor;
- передача даних по послідовній синхронній трипроводовій шині SPI;
- керування режимами зниженого енергоспоживання мікроконтролера;
- формування програмних модулів для реалізації затримок в програмі.

Компілятор CodeVision AVR дозволяє автоматично генерувати програми для виконання таких функцій:
- ініціалізація портів введення/виведення;
- ініціалізація зовнішніх переривань;
- ініціалізація доступу до зовнішньої пам'яті;
- визначення джерела переривання від сигналу RESET;
- ініціалізація таймерів/лічильників;
- ініціалізація сторожового таймера;
- ініціалізація блока послідовного обміну даними UART;
- ініціалізація аналогового компаратора.

Головне вікно інтегрованого середовища CodeVision AVR, призначеного для створення програм для AVR мікроконтролерів мовою Сі, має такий вигляд (рис. 5.1).

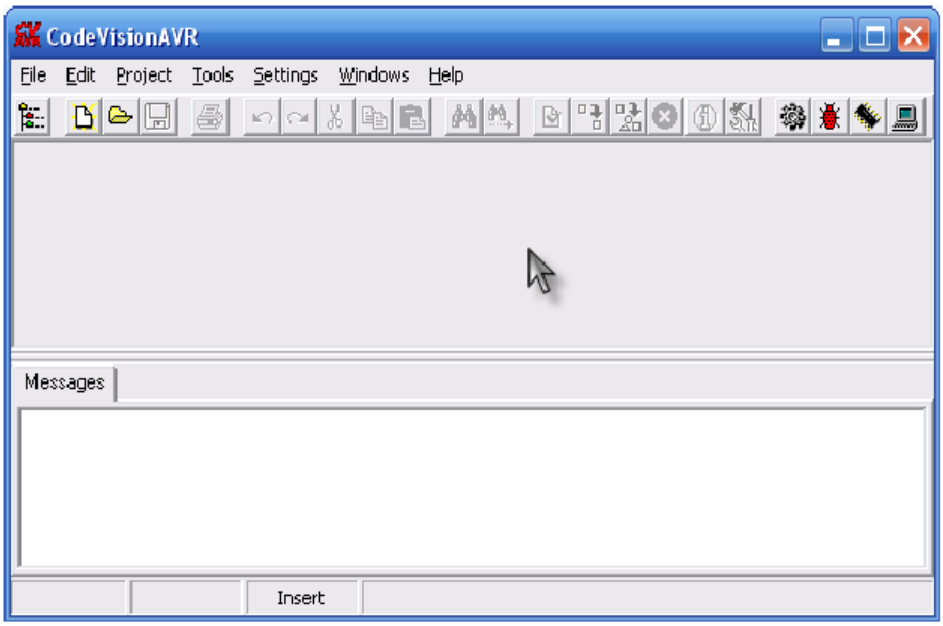


Рисунок 5.1 – Вигляд головного вікна програми CodeVision AVR

Для створення будь-якої програми в Code Vision AVR потрібно спочатку створити новий проект або відкрити вже існуючий. Новий проект створюється таким чином.

1. У головному вікні програми натискаємо File->New. На екрані з’явиться діалог (рис. 5.2).

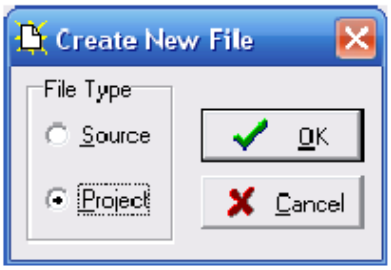


Рисунок 5.2 – Вікно створення проекту Code Vision AVR

2. Вибираємо File Type «Project». Натискаємо «OK». На екрані з’явиться діалог (рис. 5.3).

Рисунок 5.3 – Вікно створення проекту Code Vision AVR

3. Натискаємо «Yes». На екрані з’явиться діалог налаштувань, який дозволяє вибрати тип мікроконтролера та сконфігурувати його параметри й периферію (рис. 5.4).

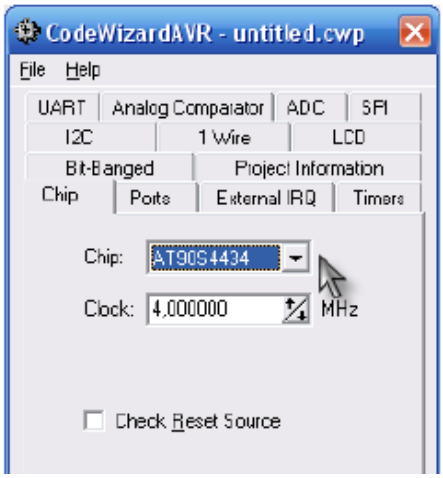


Рисунок 5.4 – Вікно конфігурування мікроконтролера та його периферії

На рисунку 5.5 наведено фрагмент екрана, який дозволяє налаштувати опції програмного проекту, що дасть змогу отримати шаблон програми з налаштуванням ліній порту введення/виведення.

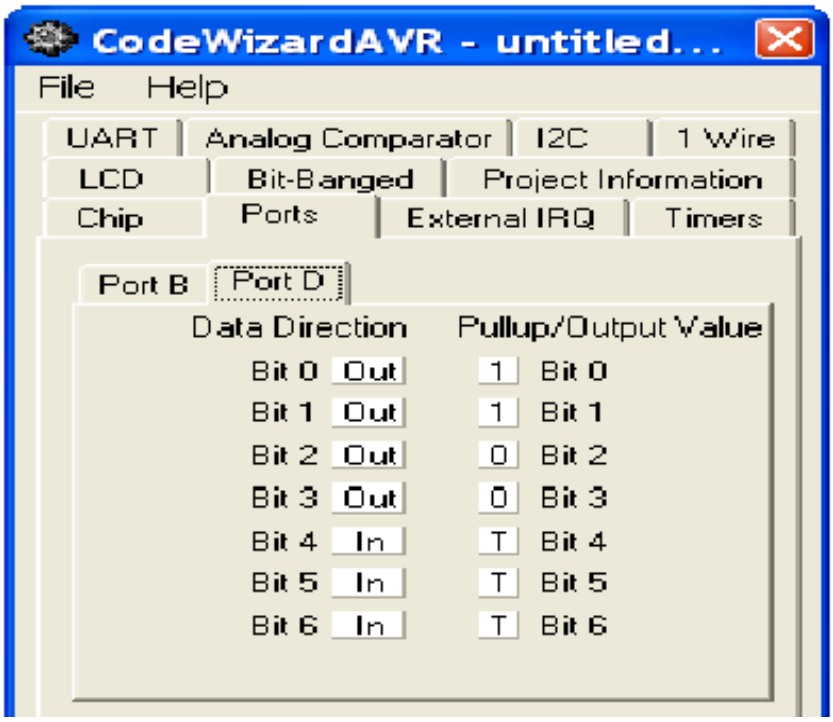


Рисунок 5.5 – Налаштування ліній порту введення/внведення проекту

Фрагмент екрана, який ілюструє створення шаблона програми з вибором опцій таймера/лічнльннка, наведено на рисунку 5.6.

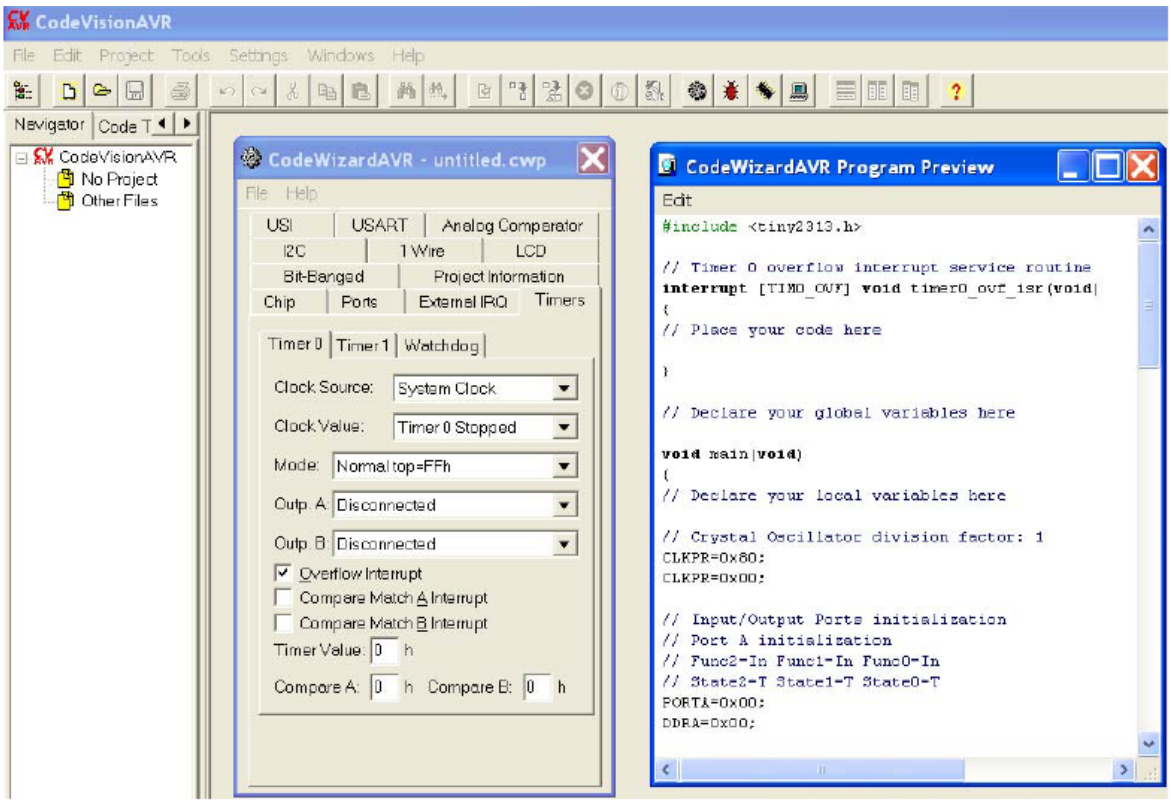


Рисунок 5.6 – Шаблон екрана для роботи з таймером-лічнльником

Після встановлення необхідних налаштувань потрібно натиснути File->Generate, Save and Exit для генерації програмного коду згідно зі зробленими налаштуваннями та збереження сформованих файлів проекту у вибраний каталог.

4. На екрані з'явиться текстове вікно, що вже містить базовий текст програми (рис. 5.7).

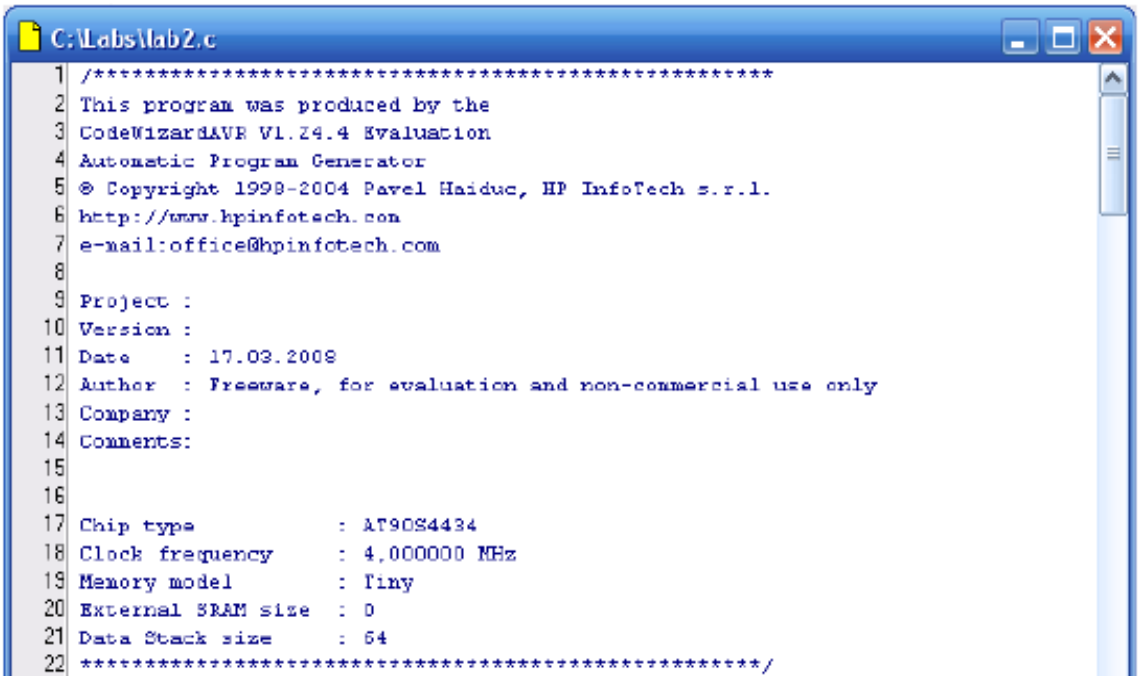


Рисунок 5.7 – Вікно редактора створення та редагування програми

В даному вікні існує нескінченний цикл, в якому пропонується розмістити програму користувача.

```
while (1)
{
      // Piece your code here
      };
```

5. Створену програму потрібно відкомпілювати. Для цього потрібно натиснути Project->Compile. На екрані з'явиться вікно, яке повідомляє результати компілювання (рис. 5.8).

У прикладі на рис. 5.8 в результаті компілювання помилки та системні попередження відсутні. У випадку наявності помилок, їх необхідно виправити та перекомпілювати програму.

6. Для компіляції та запуску програми на виконання необхідно натиснути Project->Make. На екрані з'явиться вікно, яке повідомляє про результати компілюваннння та асемблювання програми (рис. 5.9).

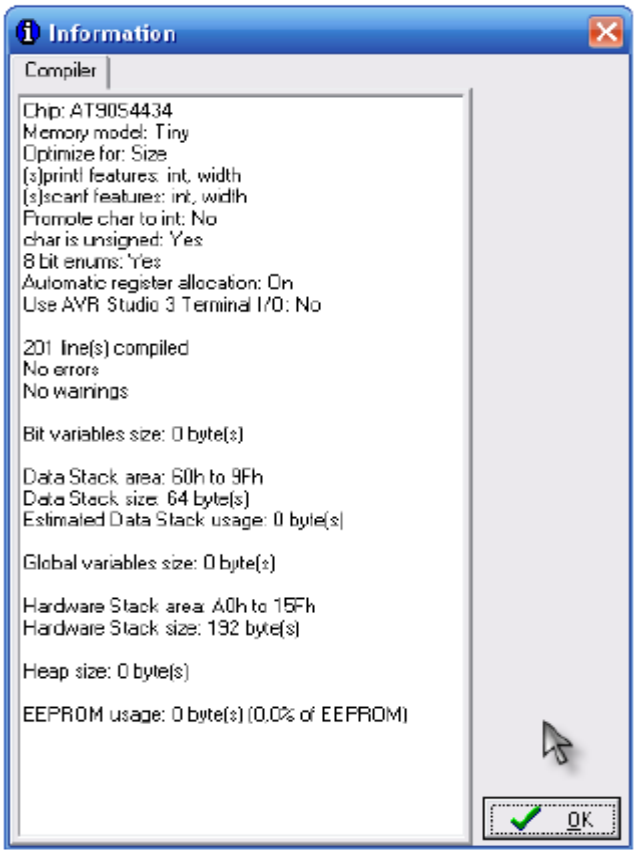


Рисунок 5.8 – Вікно результатів компілювання програми

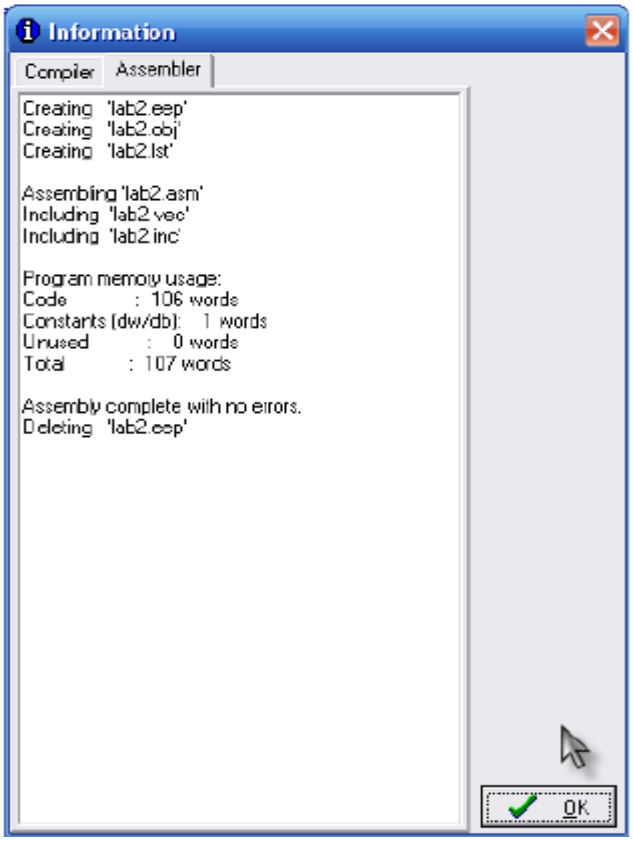


Рисунок 5.9 – Вікно результатів компілювання та асемблювання програми

7. Для тестування програми її можна емулювати за допомогою AVR Studio. Для цього потрібно вибрати AVR Studio як програму налагодження для CodeVision AVR за замовчуванням, натиснувши Settings->Debugger.

На екрані з'явиться діалог пошуку можливої програми для налагодження (рис. 5.10), в якому потрібно вказати шлях до файлу AVRStudio.exe та натиснути «ОК».

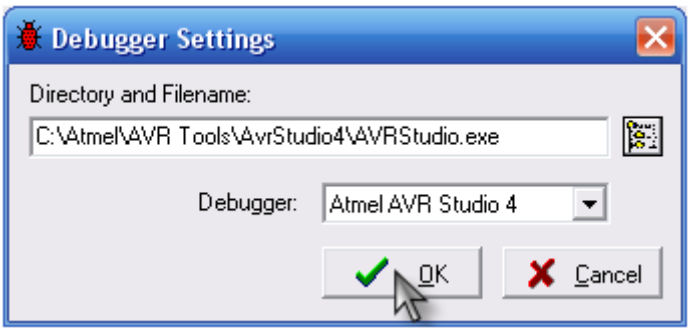


Рисунок 5.10 – Вікно вибору програми налагодження

8. Для виконання налагодження створеної програми необхідно натиснути Tools->Debugger. В результаті запускається програма AYR Studio, в якій відкривається вікно з програмою, яку ми тестуємо (рис. 5.11). Отже, запустивши налагодження, ми отримуємо можливість протестувати текст програми.

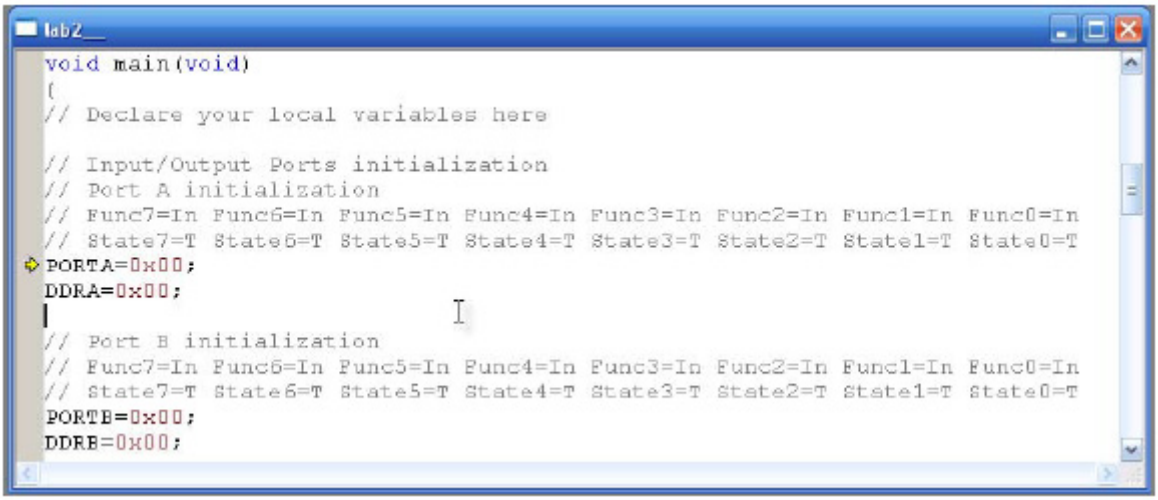


Рисунок 5.11 – Вікно тестування програми в середовищі AVR Studio

Для тестування програми в покроковому режимі можна також запустити програму AVR Studio та вибрати через меню File результат трансляції програми з розширенням *.cof.

5.3 Приклад розробки програми для МК AVR в середовищі CV AVR

Розглянемо програмні реалізації типових задач, які зустріються в обчислювальних засобах з використанням мікроконтролерів. Реалізацію таких задач наведемо з використанням мов Асемблер та С в середовищі CodeVision AVR.

5.3.1 Прорама мовою Сі для керування світлодіодами

Для прикладу розглянемо просту програму, створену в CodeVision, яка вмикас та вимикає світлодіод, який підключено до сьомої лінії порту В з періодом біля однієї секунди. Текст програми має нижченаведений вигляд.

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Author : Freeware, for evaluation and non-commercial use only Company :
Comments:
Chip type:  AT90S2313
Clock frequency:  4.000000 MHz
Memory model:  Tiny
External SRAM size: 0  Data Stack size: 32
**************************************************************************

#include <90s2313.h>
unsigned int count_delay=0;
unsigned char delay_flag=0;
                // Timer 0 overflow interrupt service routine
// Fov=500 000 Hz/256 = 1953 Hz  Tov=0.5l2 msec
 interrupt  [TIM0_OVF] void timer0_ovf_isr(void)
         { count_delay++; }
void delay_msec(unsigned int msec)         //return 1 if yes delay, return 0 if no
{
 count_delay=0;
    while(count_delay < 2*msec)  #asm("nop");
       }

void main(void)
{
   // Input/Output Ports initialization
    // Port B initialization
PORTB=0x00;
DDRB |= (1<<7);                  //PORTB,7 output
     // Port D initialization
PORTD=0x00;
DDRD=0x00;
// Timer/Counter 0 initialization
// Clock value: Fclk= 4 000 kHz/8 = 500 kHz
```

```
TCCR0=0x02;
TCNT0=0x00;
// External Interrupt(s) initialization
// INTO: Off, INTI: Off
GIMSK=0x00;
MCUCR=0x00;
//Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x02;
// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
// Global enable interrupts
#asm(«sei»)
while (1)
{
PORTB &= ~(1<<7);        // «0» on PORTB,7 – «on»
delay_msec( 1000);
PORTB |= (1<<7);         // «l» on PORTB,7 – «off»
delay_msec( 1000);
  }
}      //end main
```

5.3.2 Модифікація окремих бітів оперативної пам’яті

Для модифікації окремих бітів комірки пам’яті чи змінної можна використовувати логічні операції з константою для накладання маски на комірку пам’яті чи змінну.

Наприклад, потрібно очистити третій біт комірки пам’яті з адресою 0x60, а потім встановити цей же біт, не модифікуючи інших бітів цієї комірки пам’яті.

При використанні мови асемблера фрагмент програми може мати такий вигляд:

а) модифікація біта з використанням логічної операції та маски;

```
ldi  r30, 0x60          ; завантаження в регітрову пару R30:R31 (Z)
ldi  r31, 0             ;  адреси  0x60     Z=0x60
ld     r16, Z           ; запис байта пам’яті з ОЗП в регістр R16
andi  r16, 0xf7         ; очищення біта 3  в регістрі логічною операцією I
st      Z, r16          ; запис модифікованого байта в ОЗП за адресою 0x60
ld r16, Z               ; запис байта пам’яті з ОЗП в регістр R16
ori       r16, 0x08     ; установлення біта 3  в регістрі логічною операцією АБО
st         Z, r16       ; запис модифікованого байта в ОЗП за адресою 0x60
```

б) модифікація біта з використанням логічної операції та операторів зсуву, які підтримує Асемблер;

```
ldi  r30, 0x60          ; завантаження в регітрову пару R30:R31 (Z)
```

```
ldi  r31, 0              ;  адреси  0x60     Z=0x60
ld   r16, Z              ; запис байта пам'яті з ОЗП в регістр R16
andi  r16, ~(1<<3)       ; очищення біта 3  в регістрі логічною операцієюI
st   Z, r16              ; запис модифікованого байта в ОЗП за адресою 0x60
ld   r16, Z              ; запис байта пам'яті з ОЗП в регістр R16
ori   r16, (1<<3)        ; установлення біта 3 в регістрі логічною операцією АБО
st    Z,  r16            ; запис модифікованого байта в ОЗП за адресою 0x60
```

При використанні мови С та маски фрагмент програми може мати такий вигляд:

```
unsigned char    var;
var &= 0xf7;                //clear bit 3
var  |=  0x08;              // set   bit 3
```

Але краще користуватись логічними операціями та операторами зсуву, що дає змогу символічно описувати біт, який має бути модифіковано:

```
unsigned char    var;
unsigned char    bit=3;
 var  &= ~(1 <<  bit );          //clear bit 3
var   | =  (1 << bit );          // set   bit 3
```

5.3.3 Реалізація програмних затримок в мікроконтролері

Для реалізації затримок в програмі можна використовувати два підходи – програмний та апаратний.

Програмний підхід передбачає використання як затримки часу виконання певного числа команд мікроконтролера, оскільки час виконання кожної команди відомий. Недоліком такого підходу є неможливість мікроконтролера виконувати під час програмної затримки програму основної задачі, а також залежність величини затримки від частоти роботи тактового генератора.

Наприклад, реалізація програмної затримки на асемблері може мати такий вигляд.

```
          ; Fosc=5 МГц, call= 4 clock
delay_5mksec:
          push  r21          ;   2   clock
          ldi     r21,8      ;   1   clock
l_5mksec:
          dec   r21          ;   1   clock
          brne   l_5mksec    ;  1/2 clock
          pop   r21          ;  2   clock
          ret                ;  4   clock
          ; ---------------------------------------------
delay_1msec:
           push     r21
```

```
            ldi       r21,1000/5
loop_1msec:
             call  delay_5mksec
            dec   r21
            brne   loop_100mksec
             pop       r21
          ret
```

Приклад реалізації програмної затримки мовою С.

```
void delay_mksec (unsigned int pause)    // pause N mksec
{
 unsigned int j;
 char i;

  for (j=0; j<=pause; j++)
  for (i=0; i<1; i++) __no_operation();   //for  7 MHz
 //for (i=0; i<2; i++) __no_operation();  //for 14 MHz
}
 //---------------------------------------
void delay_msec (unsigned int pause)     // pause - N msec
{
  unsigned int i;

  for (i=0; i<=pause; i++)
  delay_mksec(1000);
}
```

Приклад іншої реалізації програмної затримки мовою Сі з урахуванням частоти роботи тактового генератора мікроконтролера.

```
#define F_clock  7372800                //CPU Clock

#define delay_mksec(del)
__delay_cycles((unsigned long)((double)F_clock*del/1000000))

#define delay_msec(del)
__delay_cycles((unsigned long)((double)F_clock*del/1000))
//---------------------------------------
```

Апаратний підхід передбачає, як правило, використання внутрішніх апаратно реалізованих блоків таймерів/лічильників.

При налаштуванні таймера/лічильника як лічильника зовнішній сигнал відомої частоти надходить на вхід лічильника. Тоді відрізок часу буде визначатись кількістю сигналів, які зафіксовані лічильником.

При налаштуванні таймера/лічильника як таймера внутрішній сигнал мікроконтролера відомої частоти надходить на вхід лічильника. Відрізок часу буде визначатись кількістю сигналів, які будуть зафіксовані лічильником.

При такому підході мікроконтролер може виконувати основну програму паралельно з роботою таймера/лічильника.

5.3.4 Програмний доступ до енергонезалежної пам'яті (EEPROM)

Мікроконтролери містять область енергонезалежної пам'яті (EEPROM), інформація в якій електрично стирається, а сама пам'ять забезпечує до 100 000 циклів перезапису. В такій пам'яті зручно зберігати тимчасові константи, інформацію про конфігурацію мікропроцесорного пристрою.

Основні операції, які виконує така пам'ять – це операції запису та читання. Робота з такою пам'ятю забезпечується програмним доступом до трьох керувальних регістрів – адреси (EEAR), даних(EEDR) та регістра керування (EECR).

Варіант програми на асемблері, який підтримує роботу з пам'ятпю EEPROM може мати такий вигляд:

```
                        ; EEPROM_write:
                        ;Wait for completion of previous write
EEPROM_write:
        sbic EECR,EEWE
        rjmp EEPROM_write
                        ;Set up address (r18:r17) in address register
        out  EEARH, r1 8
        out  EEARL, r1 7
                        ; Write data (r16) to data register
        out EE DR, r16
                        ; Write logical one to EEMWE
        sbi EECR, EEMWE
                        ; Start eeprom write by setting EEWE
        sbi  EECR, EEWE
        ret

                        ; EEPROM_read:
                        ; Wait for completion of previous write
EEPROM_read:
        sbicEECR, EEWE
        rjmpEEPROM_read
                        ; Set up address (r18:r17) in address register
        out EEARH, r18
        out EEARL, r17
                        ; Start eeprom read by writing EERE
    sbiEE CR,EERE
                        ; Read data from data register
```

```
in r1 6,EEDR
ret
```

Варіант програми мовою С, який підтримує роботу з пам'яттю EEPROM, може мати такий вигляд:

```
void EEPROM_write(unsigned int uiAddress, unsigned char ucData)
{
 while(EECR & (1<<EEWE));          // Wait for completion of previous write
 EEAR = uiAddress;                 // Set up address and data registers
 EEDR = ucData;
 EECR |= (1<<EEMWE);               // Write logical one to EEMWE
 EECR |= (1<<EEWE);                // Start eeprom write by setting   EEWE
}

unsigned char EEPROM_read(unsigned int uiAddress)
{
 while(EECR & (1<<EEWE));          //Wait for completion of previous write
 EEAR = uiAddress;                 //  Set up address register
 EECR |= (1<<EERE);                // Start eeprom read by writing EERE
 return EEDR;                      // Return data from data register
}
```

При відсутності бібліотечного опису керувальних бітів регістра EECR їх можна перед використанням оголосити:

```
#define EERE   0
#define EEWE 1
#define EEMWE 2
```

5.3.6 Реалізація програмного доступу до портів введення/виведення

Кожному двонаправленому порту X введення.виведення поставлено у відповідність три регістри – регістр напрямку DDRX, регістр даних PORTX та регістр логічного стану зовнішніх ліній порту PINX. Тому робота з портами зводиться до програмного доступу до регістрів порту.

Приклад програми мовою Асемблера, який налаштовує лінії 0–3 порту в режим виведення, встановлює високий рівень на лініях 0–3 порту, читає лінії 0–3 порту, може мати такий вигляд:

```
...
ldi   r16,(1<<3) | (1<<2) | (1<<1) | (1<<0)
ldi   r17,(1<<3) | (1<<2) | (1<<1) | (1<<0)
out   DDRB,   r16          ; установлення ліній порту 0–3 на виведення
out   PORTB,  r17          ; установлення логічного стану ліній порту
nop                        ; пауза синхронізації
in   r16, PINB             ; читання ліній порту
...
```

Аналогічні дії над лініями порту, які описані мовою С, мають такий вигляд:

```
    unsigned char i;
    ...
    DDRB   = (1<<3) | (1<<2) | (1<<1) | (1<<0);
    PORTB = (1<<3) | (1<<2) | (1<<1) | (1<<0);
    NOP();                  // пауза синхронізації
    i = PINB;               // читання ліній порту
…
```

5.4 Контрольні запитання і завдання

1. Які компілятори з мови Сі для МК типу AVR Вам відомі?
2. Чи можна налагоджувати об’єктні файли середовища Code Vision AVR (CV AVR) за допомогою середовища AVR Studio?
3. Шаблони яких програм для роботи зы стандартними зовнішніми компонентами може генерувати CV AVR?
4. Для виконання яких функцій CV AVR може автоматично генерувати програми мовою Сі?
5. Опишіть послідовність роботи з інтерфейсом для створення проекту в середовищі Code Vision AVR.
6. Які опції інтерфейсу середовища CV AVR потрібно використати для компіляції та запуску розробленої мовою Сі програми?
7. Які дії потрібно виконати для тестування програми?
8. Скласти алгоритм і програму мовою Сі згідно з заданим викладачем варіантом задачі. Відкомпілювати програму і реалізувати на STK500 в середовищі AVR Studio. На STK500 встановлено МК Atmega 8515.

Варіанти задач

1. Скласти програму мовою Асемблер AVR і мовою Сі, яка за допомогою таймера підраховує число натискань на кнопку SW0. Потім після натискання на кнопку SW2 перемикає світлодіод LED7 за значенням таймера. Затримка перемикання світлодіода 500 мсек. У режимі очікування ввімкнено світлодіод LED6. Кнопка SW0 підключена до входу T0 (PB0) таймера/лічильника T/C0, кнопка SW2 підключена до входу зовнішнього переривання INT1 (PD2), світлодіоди led7 і led6 підключені до виходів порту PB7 і PB6 відповідно.
2. Скласти програму мовою Асемблер AVR і мовою Сі, яка за допомогою таймера перемикає парні і непарні світлодіоди, підключені до виходу порту PORTC. Затримка перемикання – 0,5 с, тактова частота – 1МГц.
3. Розробити програму керування світлодіодною лінійкою плати налагоджування STK500 мовами Асемблер і Сі. Програма має циклічно вмикати всі світлодіоди та вимикати з інтервалом в 1 секунду. Для відліку часу використовувати таймер мікроконтролера.

Вхідні дані: мікроконтролер – ATmtga8515, світлодіоди під’єднані до порту B мікроконтролера, засвічення світлодіода відбувається подачею логічного нуля на відповідний роз’єм порту введення/виведення.

4. Розробити програму керування світлодіодною лінійкою плати налагоджування STK500 мовами Асемблер і Сі. Програма має попарно вмикати світлодіоди плати з інтервалом 1,5 секунди. Тобто, спочатку вмикаються 1-ий та 2-ий свідлодіоди, через 1,5 секунди вони вимикаються і вмикаються 3-ій та 4-ий світлодіоди, потім 5-ий та 6-ий, 7-ий та 8-ий, і все спочатку. Для відліку часу використовувати таймер мікроконтролера.

Вхідні дані: мікроконтролер – ATmega8515, світлодіоди під'єднані до порту А мікроконтролера, засвічення світлодіода відбувається подачею логічного нуля на відповідний роз'єм порту введення/виведення.

5. Розробити програму керування світлодіодною лінійкою та блоком кнопок плати налагоджування STK500 мовами Асемблер і Сі. Програма має висвічувати двійковий код натиснутої кнопки, тобто, наприклад, якщо користувач натискає кнопку № 5 (двійковий код 00000101), то програма відповідно має увімкнути 6-ий та 8-ий світлодіоди (одиниці в двійковому коді означають, що відповідний світлодіод має бути ввімкнутим). Для опитування кнопок використовувати таймер, інтервал опитування 100ºмсек.

Вхідні дані: мікроконтролер – ATmega8515, світлодіоди під'єднані до порту А мікроконтролера, блок кнопок – до порту В, засвічення світлодіода відбувається подачею логічного нуля на відповідний роз'єм порту введення/виведення, кнопка вважається натиснутою, якщо на вхід роз'єму порту прийшла логічна одиниця.

6. Розробити програму керування світлодіодною лінійкою та блоком кнопок плати налагоджування STK500 мовами Асемблер і Сі. Програма має засвічувати кількість світло діодів, еквівалентну порядковому номеру натиснутої кнопки, тобто, наприклад, якщо користувач натискає кнопку № 4, то програма відповідно має ввімкнути перших чотири світлодіоди, якщо кнопку 6 – перших шість світлодіодів і т. д. Для опитування кнопок використовувати таймер, інтервал опитування 200ºмсек.

Вхідні дані: мова програмування – С, середовище – CVAVR, мікроконтролер – ATmega8515, світлодіоди під'єднані до порту В мікроконтролера, блок кнопок – до порту А, засвічення світлодіода відбувається подачею логічного нуля на відповідний роз'єм порту введення/виведення, кнопка вважається натиснутою, якщо на вхід роз'єму порту прийшла логічна одиниця.

ЛАБОРАТОРНА РОБОТА № 6
Програмування мікроконтролерів AVR в середовищі Atmel Studio 6

Мета роботи – вивчити принципи написання програм в інтегрованому середовищі розробки Atmel Studio 6.

6.1 Порядок виконання роботи

6.1.1 Ознайомитися з середовищем Atmel Studio 6.
6.1.2 Ознайомитися з основними елементами інтерфейсу Atmel Studio 6.
6.1.3 Ознайомитися зі створенням нового проекту в Atmel Studio 6.
6.1.4 Перенесення проекту з AVR Studio 4 в Atmel Studio 6.
6.1.5 Ознайомитися з роботою Atmel Studio 6.
6.1.6 Налагодити програму.
6.1.7 Виконати програму в покроковому та безперервному режимах. Візуалізовані результати виконання програми скопіювати та додати до звіту з лабораторної роботи.
6.1.8 Оформити звіт з лабораторної роботи.

6.2 Теоретичні відомості

6.2.1 Ознайомлення з Atmel Studio 6

Atmel Studio (до 6-ої версії AVR Studio) – інтегроване середовище розробки для програмування та налагоджування програм для мікроконтролерів AVR та AVR32 в операційних системах Windows. Після шостої версії середовище може працювати як і з AVR-контролерами, так і з системами з ARM-архітектурою.

Програмний пакет AVR Studio розробляється компанією Atmel з 2004ºроку. Починаючи з версії 6.0, компанія змінила назву програми на Atmel Studio та додала можливість програмувати системи на базі ARM-архітектури. Раніше існував і фірмовий асемблер під ОС Windows (wavrasm.exe) від Atmel, який поєднував асемблер і редактор, проте, невдовзі після появи AVR Studio, відмовились від його подальшого розвитку.

Atmel Studio містить в собі такі інструмени, як вбудований C/C++-компілятор, симулятор мікропроцесорної системи для налагоджування програм, менеджер проектів, редактор коду, модуль внутрішньосхемного налагоджування, а також інтерфейс командного рядка. Крім стандартних елементів середовище підтримує низку таких інструментів, як компілятор GCC та плагін AVR RTOS (операційної системи реального часу). Крім C/C++ середовище дозволяє програмувати також на асемблері.

6.2.2 Інтерфейс програмного середовища Atmel Studio 6

Інтерфейс середовища наведено на рис. 6.1.

Ключове вікно в Atmel Studio – це вікно початкового тексту програми (1). У вікні відображається код, який можна виконувати повністю або до місця, де є курсор.

Memory windows (2). Вікно показує вміст пам'яті програм, даних, портів введення/виведення та енергозалежного ПЗП.

Processor windows (3). У вікні відображається важлива інформація про ресурси мікроконтролера, дані параметри можуть модифікуватися під час зупинки програми.

Disassemly windows (4). У вікні відображається вихідний код, з якого формується код збирання, код байтів подання фактичної машини, символьні імена для адрес пам'яті, номери рядків, відповідні вихідному коду.

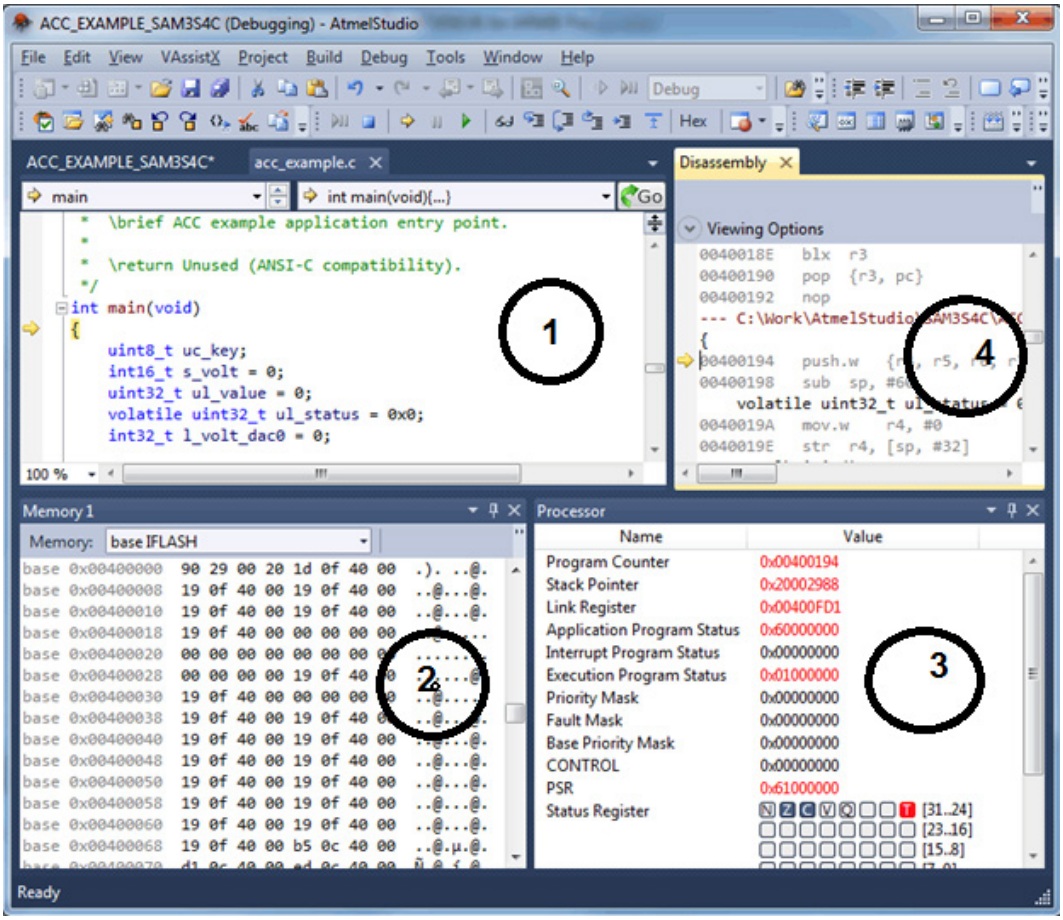


Рисунок 6.1 – Інтерфейс програмного середовища Atmel Studio 6

6.2.3 Створення нового проекту в Atmel Studio 6

Під час запуску Atmel Studio пропонує або створити новий проект (*New Project*) або відкрити вже існуючий (Open). У вікні новий проект (*New Project*) (рис. 6.2), заповнюємо поле ім'я проекту англійською мовою

(*Name*) і шлях до файлу англійською мовою (*Location*) та обираємо мову програмування, якою буде написана програма. Натискаємо *OK*.

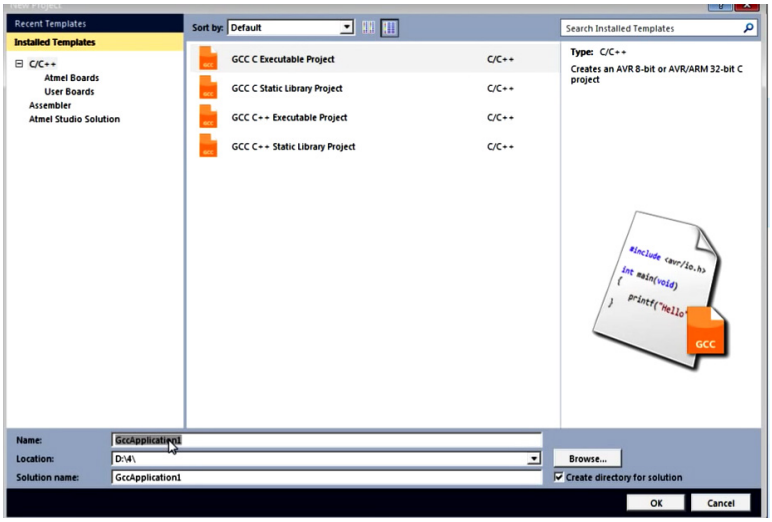


Рисунок 6.2 – Вікно створення нового проекту

З’явиться вікно *Device Selection*, в ньому потрібно обрати процесор, для якого пишеться програма. В пункті меню «*Device Family*» (рис. 6.3) обираємо процесор для роботи в подальшому. Натискаємо *OK*.

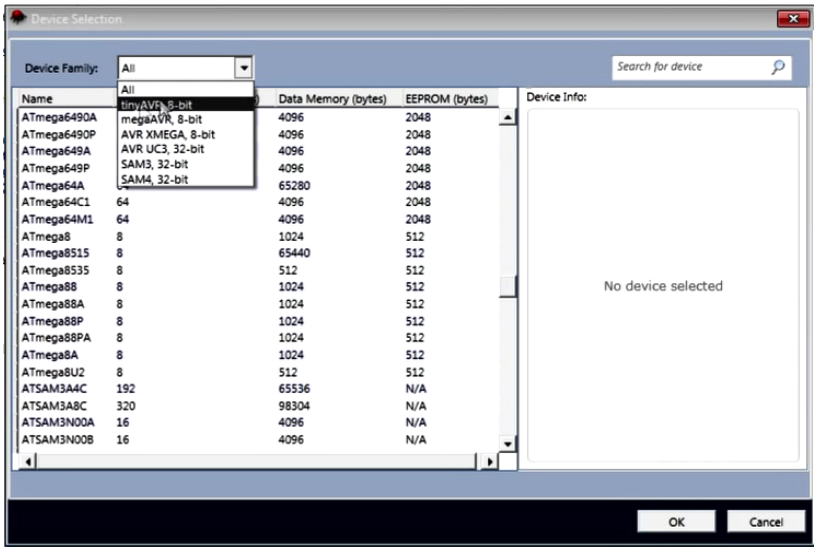


Рисунок 6.3 – Вікно вибору типу процесора (мікроконтролера)

6.2.4 Перенесення проекту з AVR Studio 4 в Atmel Studio 6

Для того щоб перенести вже створені проекти з AVR Studio 4 в Atmel Studio 6, щоб їх не переписувати, потрібно натиснути на панелі меню *File-Import-AVR Studio 4 Project.. or Ctrl+4* (рис. 6.4).

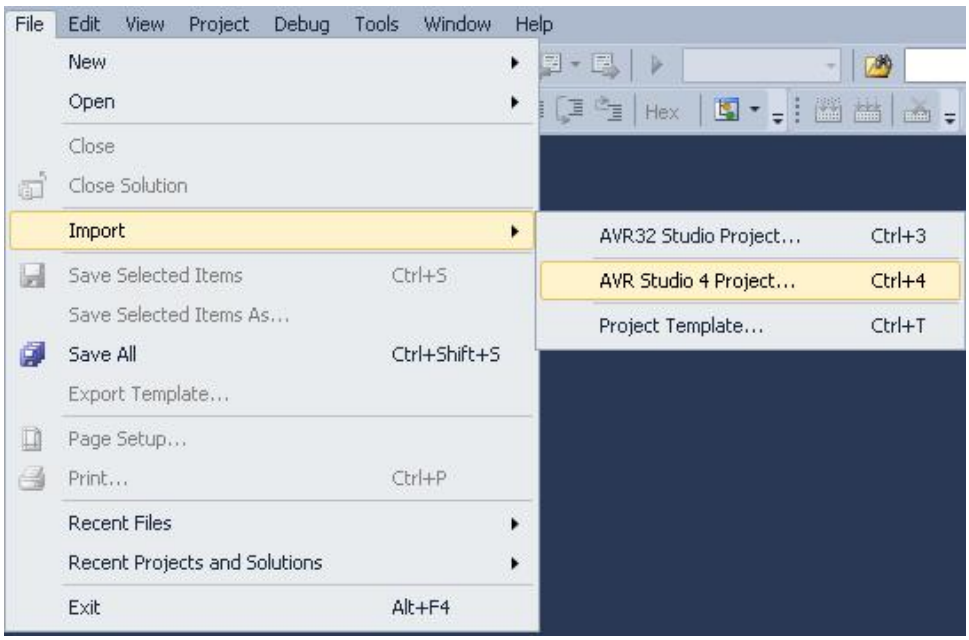


Рисунок 6.4 – Вікно діалогу для вибору опції імпорту проекту з AVR Studio 4 в Atmel Studio

З’явиться вікно Import AVR Studio 4 Project (рис. 6.5), у якому потрібно ввести назву вашого проекту або перейти до папки проекту, натиснувши на кнопку *Browse* на вкладці *APS File location Tab*. Натиснути клавішу *Convert*, після чого Atmel Studio буде перевіряти проект на помилки. Натиснути *OK*.

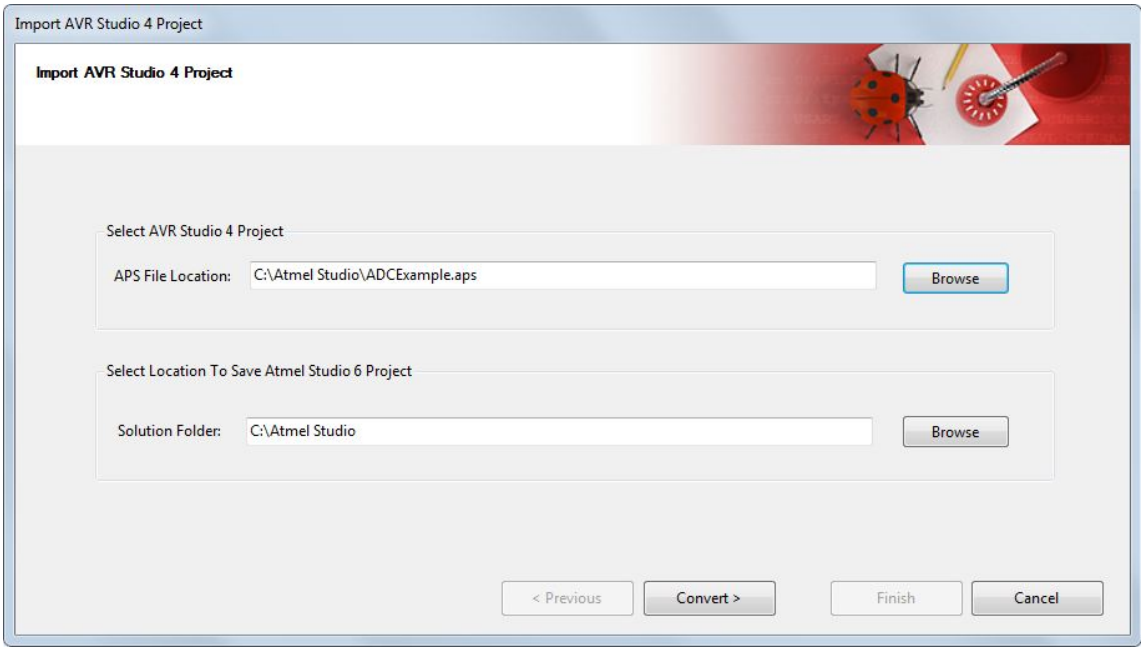


Рисунок 6.5 – Вікно імпорту проекту з AVR Studio 4 в Atmel Studio

6.2.5 Робота з коментарями в Atmel Studio 6

В Atmel Studio 6 коментарі в програмі, написані російською чи українською мовами, будуть підкреслені червоною лінією (рис. 6.6), оскільки Atmel Studio їх не розуміє та сприймає як помилку.

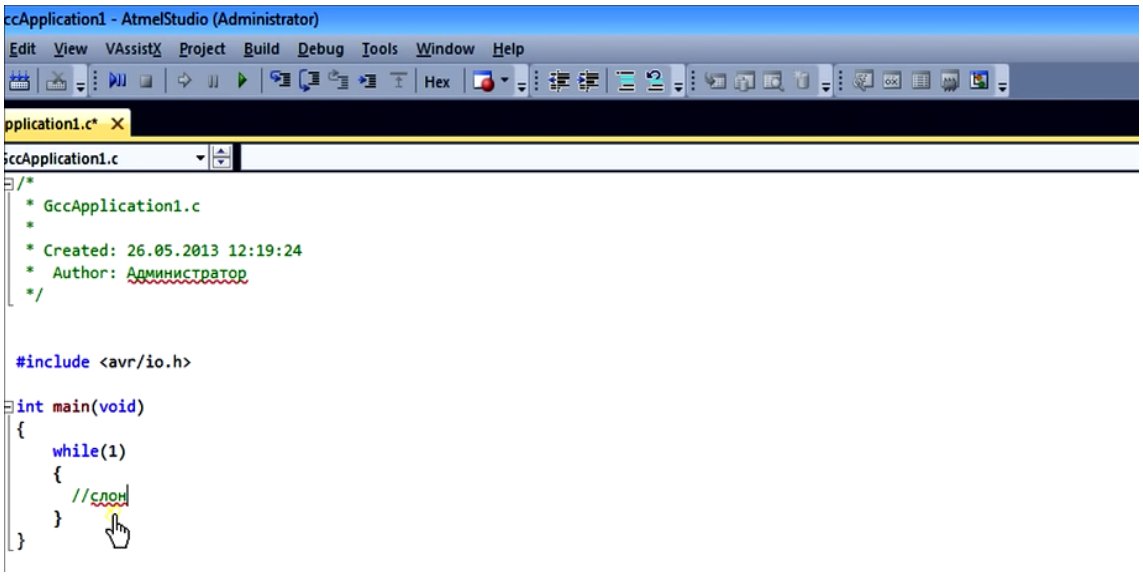


Рисунок 6.6 – Підкреслювання коментарів, написаних кирилицею

Існує два способи виправити помилку в Atmel Studio:

1) Натиснути правою кнопкою миші на слово (рис. 6.7):

– *Add word to dictionary* – запам'ятати слово в бібліотеку, більше в проектах дане слово не буде підкреслюватися.

– *Ignore all occurrences* – ігнорувати слово, яке не буде збережене в бібліотеку і не запам'ятовується.

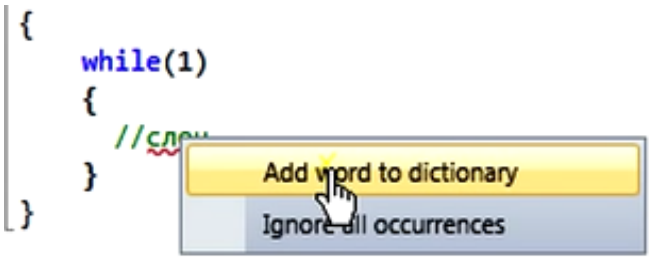


Рисунок 6.7 – Усунення підкреслень натисканням правої кнопки правою кнопкою миші

2) Усунення підкреслення, даний спосіб буде ефективнішим та кращим за перший. Для цього потрібно натиснути на панелі меню *VAssistX – Visual Assist X Options* (рис. 6.8):

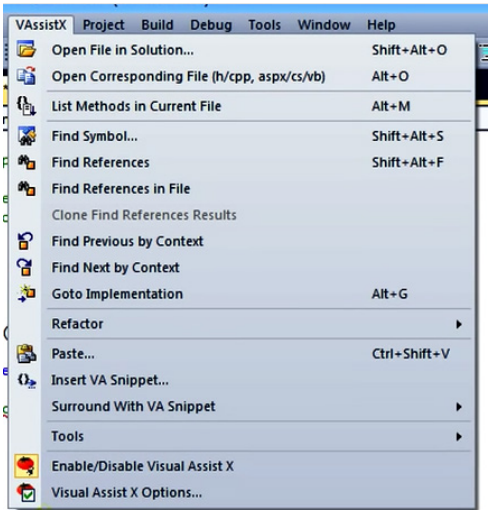


Рисунок 6.8 – Вибір опції усунення підкреслень

Після цього з’явиться вікно *Visual Assist X Options* (рис. 6.9), в якому потрібно обрати *Advanced – Underlines – Underline spelling errors in comments and strings using, забрати галочку*, натиснути *Применить*.

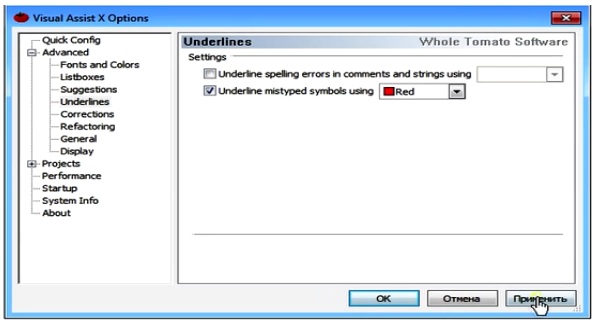


Рисунок 6.9 – Вікно усунення підкреслень другим методом

В результаті виділення червоним коментарів як помилки не буде.

Також на панелі інструментів є кнопка, яка додає коментарі в текст програми (рис. 6.10), та кнопка, що забирає коментарі (рис. 6.11).

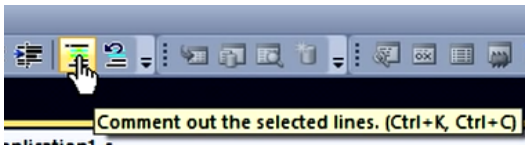


Рисунок 6.10 – Кнопка додання коментарів

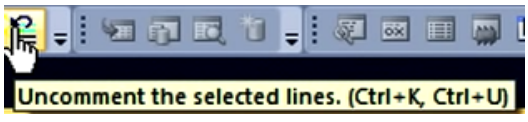


Рисунок 6.11 – Кнопка вилучення коментарів

6.2.6 Відлагодження програми в Atmel Studio 6

Компіляція – процес перекладу тексту програми, написаної мовою програмування, у виконавчий модуль, який містить машинні команди конкретного процесора. Компіляція асемблерних команд мікроконтролера називається асемблеруванням. Асемблерування – це перетворення мови асемблера в команди машинної мови.

Якщо при написанні тексту програми були допущені синтаксичні помилки, компіляція переривається і у вкладці *Bild* виводиться повідомлення про допущені помилки. При успішній компіляції у вкладці *Bild* показується звіт про проходження процесу асемблерування і таблиця використаних ресурсів.

Після вдалої компіляції можна переходити до фази симуляції.

Симуляція – моделювання процесу виконання програми мікроконтролером на персональному комп'ютері. Інакше кажучи – режим налагоджування (Debugging).

Налагоджування – етап комп'ютерного розв'язання задачі, при якому відбувається усунення явних помилок в програмі. Часто проводиться з використанням спеціальних програмних засобів –налагоджувачів.

Для управління режимом налагоджування призначені такі кнопки (рис.°6.12):

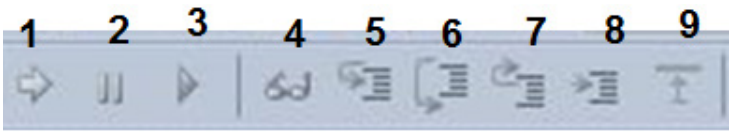


Рисунок 6.12 – Панель з кнопками управління режимом налагоджування

1 – стрілка жовтого кольору, показує де зупинилася програма;
2 – зупинка програми;
3 – запуск програми;
4 – пошук у програмі;
5 – поетапне проходження програми;
6 – виконання програми, але не виконання функцій;
7 – виконання функцій та повернення у наступний рядок програми;
8 – зупинка програми там, де знаходиться курсор;
9 – скинути програму, курсор автоматично повертається на початок програми.

Контрольна точка – інструкція в програмі, дійшовши до якої виконання програми призупиниться. Встановлена контрольна точка відзначена червоним кружком (рис. 6.13).

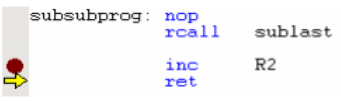


Рисунок 6.13 – Позначення контрольної точки у вікні налагоджування

Оглядове вікно – призначене для перегляду і редагування значень всіх визначених символів (рис. 6.14).

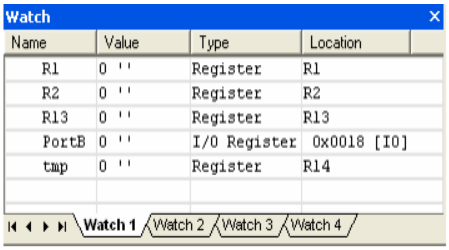


Рисунок 6.14 – Вікно перегляду і редагування символів

Вікно регістрів – призначене для перегляду і редагування значень регістрів (рис. 6.15).

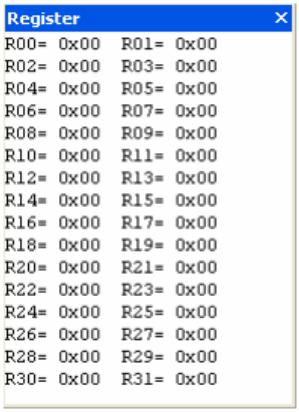


Рисунок 6.15 – Вікно перегляду і редагування значень регістрів

Вікно пам’яті (рис. 6.16) – показує розподіл і вміст пам’яті в мікроконтролері (пам’яті програм, регістрів загального користування, регістрів введення/виведення, EEPROM і ін.).

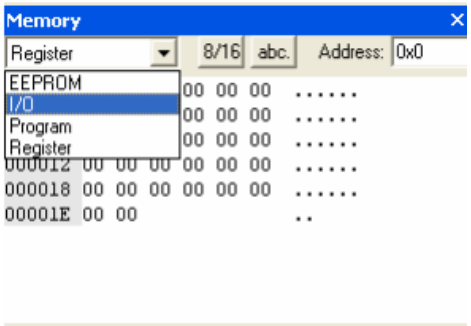


Рисунок 6.16 – Вікно перегляду пам’яті мікроконтролера

Вікно дизасемблерування (рис. 6.17) – показує, як програма транслювалася в машинні інструкції і за якими адресами вони розташовані.

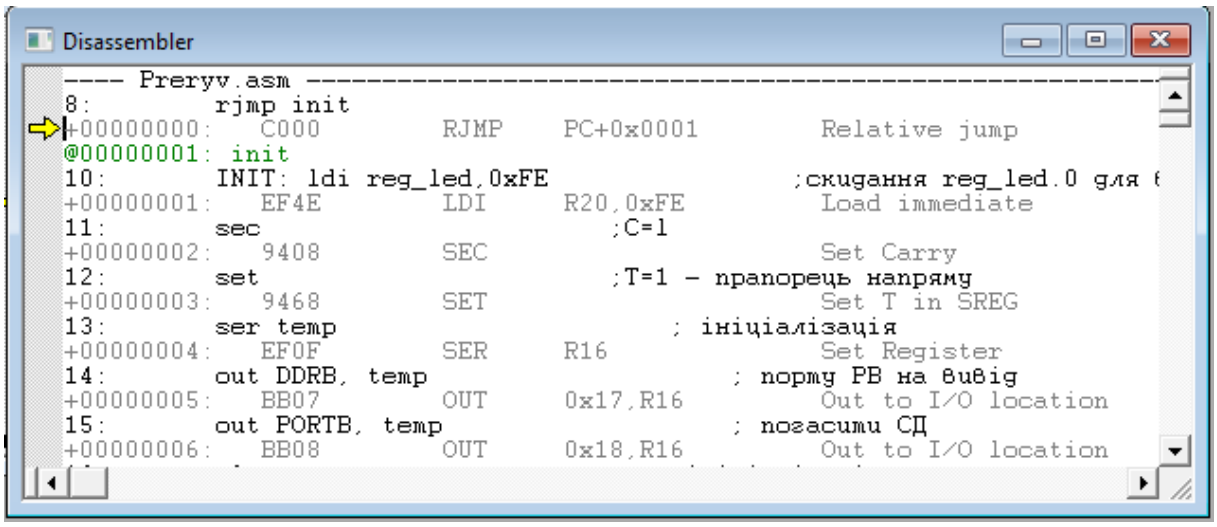


Рисунок 6.17 – Вікно дизасемблерування

Дизасемблерування – переклад машинних кодів будь-якої програми в її подання мовою Асемблер. Інформація про регістри введення/виведення, процесор і регістри загального користування розташована і розподілена по групах у вкладці «Перегляд введення/виведення» – I/O View (рис. 6.18).

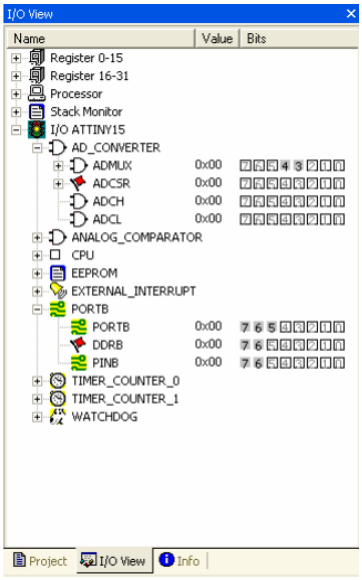


Рисунок 6.18 – Вікно перегляду регістрів введення/виведення

Після того, як проект пройшов налагоджування, дану програму можна записувати в мікроконтролер.

6.2.7 Корисні функції в Atmel Studio 6

Заміна багатьох імен функцій, а також імен глобальних і локальних змінних: *Права клавіша миші на ім'я фунції– Refactor– Rename*.

Пошук змінних в проекті: *Права клавіша миші на змінну– Refactor– Find References*. Якщо змінна зеленого кольору, то в ній нічого не відбувається, червоного – зміни відбулися.

Виділення глобальних змінних: *Vissual AssistX Options– TextEditors– Local symbols in bolt– Применить*.

Параметри шрифту: *Tools– Options– Enviroment– FontsandColor– Size– OK*.

6.2.8 Створення бібліотек в Atmel Studio 6

Створення бібліотек є дуже зручною опцією, тому що їх можна використовувати декілька раз для різних проектів.

Покроковий опис створення бібліотек:

1) Завантажити чи створити проект;
2) Створити новий лист: *File– New– File– IncFile*;
3) Зберегти його в проекті: *File–Save–Папка, де проект знаходиться– Save*;
4) Написання коду, який ми хочемо зберегти, як бібліотеку;
5) Переходимо в програму, в якій ми хочемо використати нашу бібліотеку, здійснюємо під'єднання: #include «Назва бібліотеки.h».

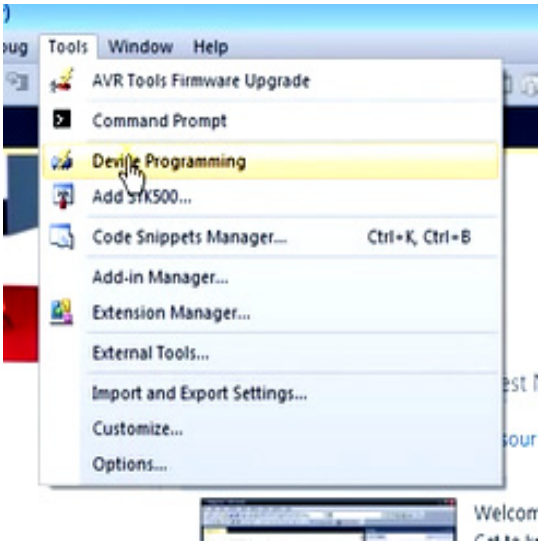
Якщо назва бібліотеки в <Назва бібліотеки.h> – бібліотека Atmel Studio, а якщо ``Назва бібліотеки.h`` – використовується бібліотека, яка збережена в папці з проектом;

6) Звертаємся до функції в основній програмі, яка збережена в бібліотеці: SF();
7) Компілюємо проект.

6.2.9 Завантаження HEX.файлу та зчитування з мікропроцесора

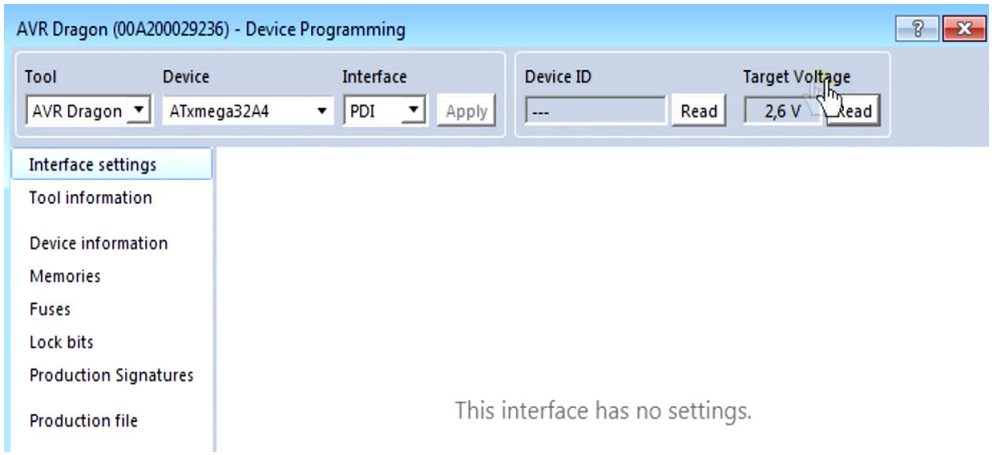
Розглянемо покроково, як завантажити з процесора дані або зчитати HEX.файл.

1) *Tools– Device Programming*.

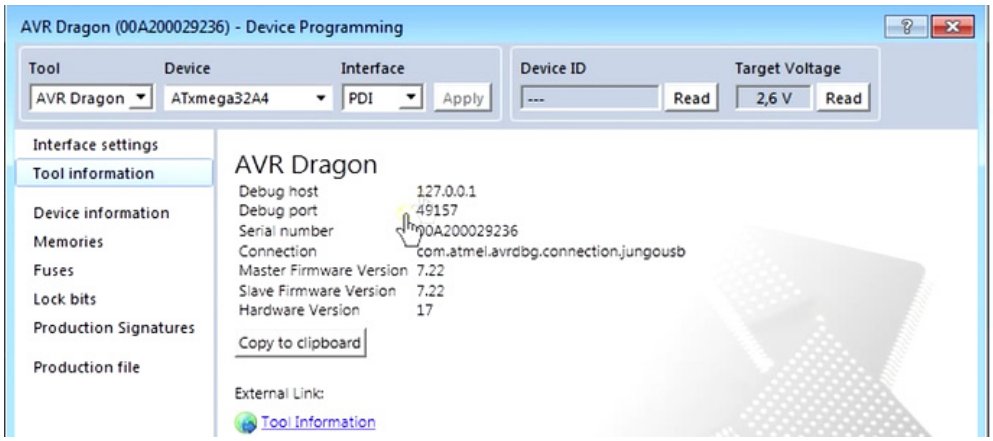


2) Вікно «Device Programming», в якому:
Tool – вказуємо тип програматора;
Device – вказуємо мікросхему;
Interface – інтерфейс, на якому буде виконуватися компіляція.

3) Натиснути клавішу «Apply» – підключення та клавішу «Target Voltage» – яка показує напругу процесора.

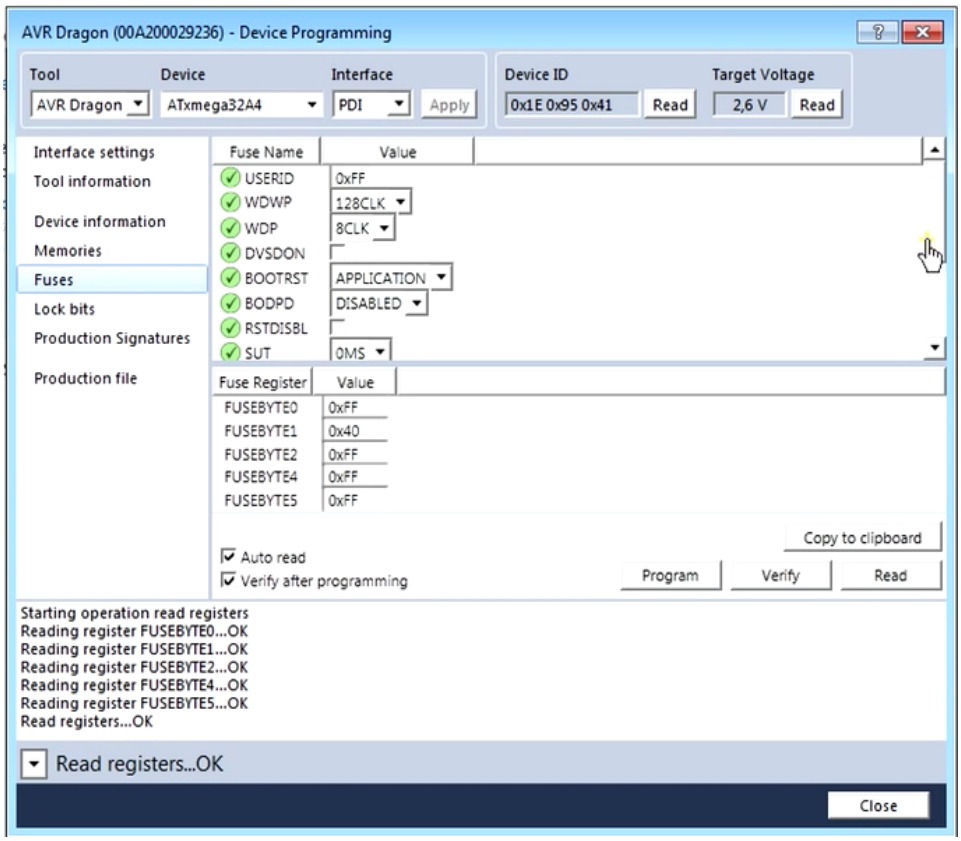


Значення вкладок після підключення у вікні «AVR Dragon–Device Programming»:
Interface settings – дозволяє налаштувати інтерфейс;
Tool information – інформативна вкладка;

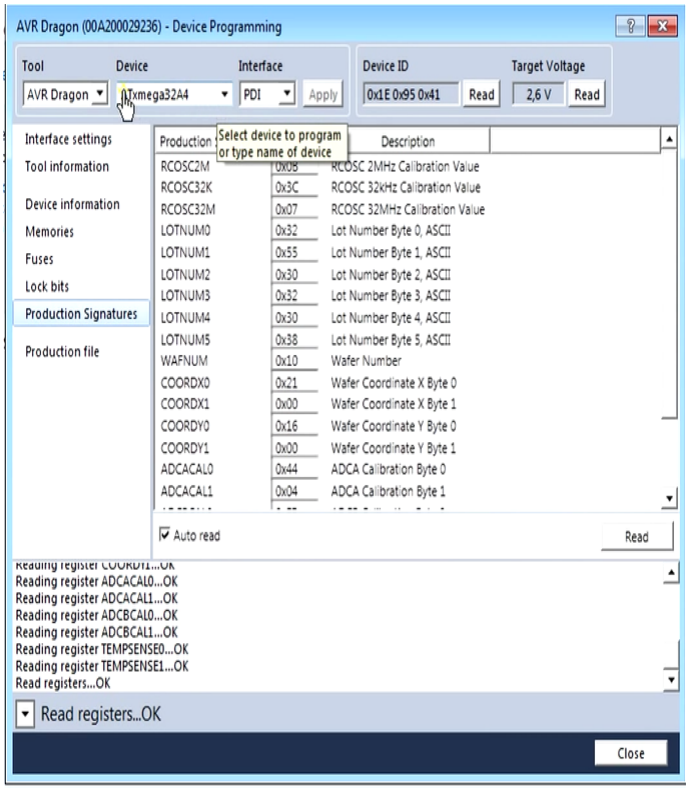


Device information – інформативна вкладка про процесор;

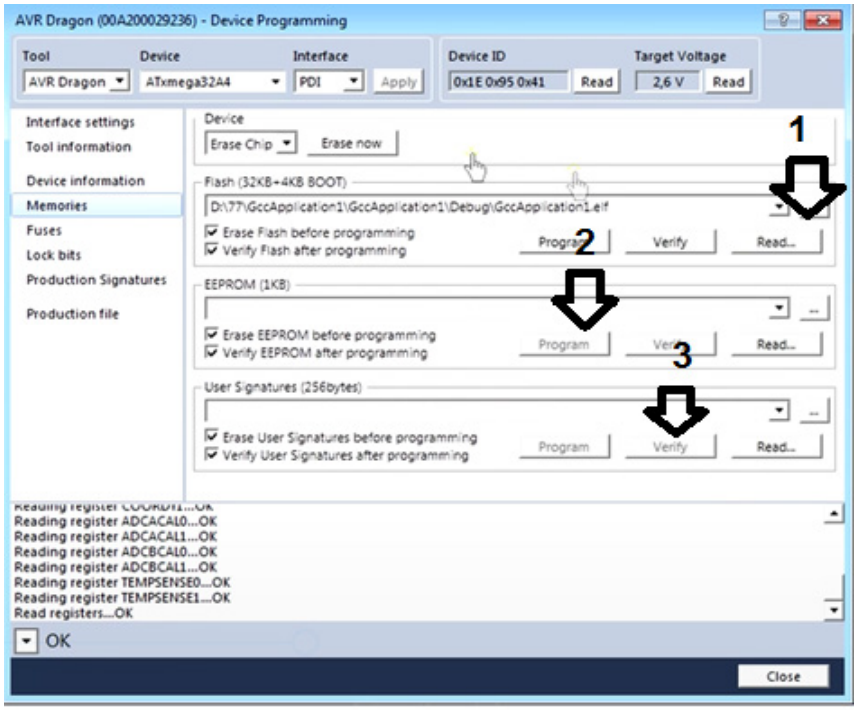
Fuses – це біти, які налаштовують МК при прошиванні, програмно змінити не можна;



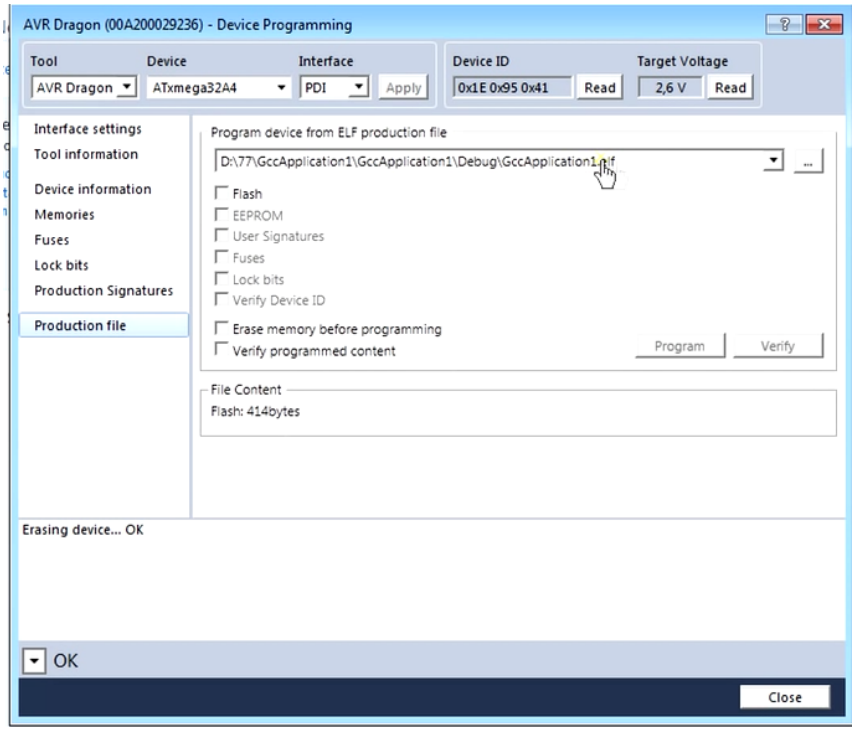
Production Signatures – сигнатура про вид процесора;



Memories – працює з пам’яттю: Flash, EEPROM, Signature. Дає можливості: читати (1), програмувати (2), звіряти дані з даними в процесорі на помилки (3).



Production file – робота з файлом розширення .elf.



6.3 Приклад запису програми в Atmel Studio

Програма мерехтіння двома світлодіодами порту B

```
# define F_CPU 8000000UL // Вибираємо частоту МК
#include <avr / io.h>                          // Для роботи з портами
#include <util / delay.h>                      // для затримки
int main(void)
{
                              // налаштування 2 и 3 виводів порту B як вихід
DDRB = 0b00001100;       // занесення «1» в 2-ий і 3-ій виводи порту B
DDRB = 0x0C;              // 16-ий код
DDRB = 12;                 // 10-ий код
DDRB |= ( 1 << 2 ) | ( 1 << 3 );            // розряди 2 та 3 порту B на вихід
                            // на 2-ий и 3-ій виводи подаємо логічну 1 (+ 5B )
PORTB |= ( 1 << 2 ) | ( 1 << 3 );
 while(1)
  {
                                 // мерехтять 2-ий та 3-ій виводи
PORTB ^ = 0b00001100;
_delay_ms(200);
  }
}
```

Для затримки використовуємо функцію _delay_ms ( ). Функція _delay_ms ( ) формує затримку залежно від переданого їй аргументу,

вираженого в мілісекундах (в одній секунді 1000 мілісекунд). Максимальна затримка може сягати 262.14 мілісекунд. Якщо користувач передасть функції значення більші 262.14, то відбудеться автоматичне зменшення дозволу до 1/10 мілісекунди, що забезпечує затримки до 6.5535 секунд.

Функція _delay_ms () міститься в файлі delay.h, тому нам буде потрібно внести цей файл до програми. Крім того, для нормальної роботи цієї функції необхідно вказати частоту, на якій працює мікроконтролер, в герцах.

В наведеній нижче програмі організований нескінченний цикл за допомогою оператора безумовного переходу «goto».

```
#define F_CPU 8000000UL          // вказуємо частоту в герцах
#include <avr / io.h>            // для роботи с портами
#include <util / delay.h>        // для затримки
int main ( void )
{
                                 // початок основної програми
DDRD = 0xff;                     // всі виводи порту D конфігурувати як виходи
start:                           // мітка для команди goto start
PORTD | = _BV (PD1);     // встановити «1» (високий рівень) на вивід PD1,
                                 // Засвітити світлодіод
_delay_ms (250);                 // чекаємо 0.25 сек.
PORTD & = ~ _BV (PD1);  // встановити «0» (низький рівень) на вивід PD1,
                                 // Погасити світлодіод
_delay_ms (250);                 // чекаємо 0.25 сек.
goto start;                      // перейти до мітки start
}                                // Закривна дужка основної програми
```

У бібліотеці avr/io.h знаходяться деякі функції введення/виведення та описи регістрів МК. Без цієї бібліотеки не обходиться жодна програма. У бібліотеці util/delay.h розміщені функції затримки (паузи). Головна функція – main. Мікроконтролер виконує її після скидання або ввімкнення живлення. Вона закінчується нескінченним циклом та поверненням нуля. Перш за все відбувається налаштування портів. Команда DDRx задає напрям роботи порту на введення чи виведення. При передачі команді «1» відповідний пін порту налаштовується на «вихід». При передачі «0» відповідний пін порту налаштовується на «вхід». Команда PORTx налаштовує стан порту на початок програми. Далі йде нескінченний цикл for(;;). На відміну від комп'ютерної програми, програма мікроконтролера працює у нескінченному циклі до вимкнення живлення або скидання МК. Функція _delay_ms(10) з бібліотеки util/delay.h організує затримку 10

мілісекунд. Написати _delay_ms(500) не можна, бо максимальне значення аргументу 262.14 мс, поділене на тактову частоту в 8 МГц, дасть число, більше 2^{16}, яке не може бути підраховане 16-розрядним таймером/лічильником.

6.4 Контрольні запитання та завдання

1. Розкажіть про програмне забезпечення Atmel Studio 6.
2. Опишіть основні елементи інтерфейсу Atmel Studio 6.
3. Розкажіть про порядок створення нового проекту в Atmel Studio 6.
4. Як перенести проект з Atmel Studio 4 в Atmel Studio 6?
5. Опишіть порядок запису, компіляції та налагоджування програми в Atmel Studio 6.
6. Розкажіть про корисні функції в Atmel Studio 6.
7. Розкажіть про порядок створення бібліотек в Atmel Studio 6 та суть їх застосування.
8. Опишіть порядок завантаження HEX.файлу та зчитування з мікроконтролера.
9. **Завдання на лабораторну роботу.** Створити програму для свого типу мікроконтролера (за ркомендацією викладача), яка виконує такі функції: світлодіоди мерехтять згідно з таблицею 6.1.

Таблиця 6.1 – Варіанти завдань

№ варіанта	Ввімкнені світлодіоди	Вимкнені світлодіоди
1	3, 4, 7, 8	1, 2, 5, 6
2	1, 2, 7, 6	3, 4, 5, 8
3	3, 4, 5, 8	1, 2, 6, 7
4	1, 2, 3, 4	5, 6, 7, 8
5	4, 5, 6, 7	1, 2, 3, 8
6	2, 3, 5, 7	1, 4, 6, 8
7	1, 2, 3, 7	4, 5, 6, 8

Зміст звіту

- Мета роботи.
- Стислий виклад теоретичних відомостей.
- Лістинг власної програми з детальним поясненням кожного рядка та зображення екрана.
- Відповіді на контрольні запитання.
- Висновки.

ЛАБОРАТОРНА РОБОТА № 7
Інтегроване середовище для проектування та моделювання електронних схем Proteus VSM

Мета роботи – вивчення принципів використання пакета моделювання електронних схем Proteus VSM

7.1 Порядок виконання роботи

7.1.1 Ознайомитися з можливостями та інтерфейсом середовища.

7.1.2 Завантажити середовище та вивчити принципи його роботи.

7.1.3 Спроектувати віртуальні моделі електронних схем та симуляція їх роботи в Proteus VSM.

7.1.4 Розробити проект, вказаний в завданнях на лабораторну роботу.

7.1.5 Оформити звіт.

7.2 Теоретичні відомості

7.2.1 Загальна характеристика середовища Proteus VSM

Proteus VSM – середовище для проектування та симуляції роботи електронних схем і мікропроцесорних пристроїв. Відмінністю пакета Proteus VSM є можливість моделювання роботи програмованих пристроїв: мікроконтролерів (МК PIC, 8051, AVR, HC11, ARM7/LPC2000 та ін.), мікропроцесорів, DSP.

PROTEUS VSM дозволяє дуже достовірно моделювати і налагоджувати досить складні пристрої, в яких може міститися кілька МК одночасно і навіть різних сімейств МК в одному пристрої.

PROTEUS складається з двох основних модулів:

- ISIS – графічний редактор принципових схем служить для введення розроблених проектів з подальшою імітацією і передачею для розробки друкованих плат в ARES. До того ж після налагоджування пристрою можна відразу розвести друковану плату в ARES яка підтримує авторозміщення і трасування по вже існуючій схемі;
- ARES – графічний редактор друкованих плат з вбудованим менеджером бібліотек і автотрасувальник ELECTRA з автоматичним розставленням компонентів на друкованій платі.

Для інсталяції Proteus VSM необхідно:

1. Запустити інсталяційний файл;
2. У відкритому вікні вибрати компоненти, які необхідно інсталювати;
3. Виконати налаштування, необхідні для інсталяції.

Основні пункти меню Proteus ISIS

1. Меню File.
2. Меню View.
3. Меню Edit.
4. Меню Library.
5. Меню Tools.
6. Меню Design.
7. Меню Graph.
8. Меню Source.
9. Меню Debug.
10. Меню Template.
11. Меню System.
12. Меню Help.

На рисунку 7.1 зображене головне вікно програми. Весь робочий простір програми розділено на декілька основних частин:

- вікно редактора схем (виконується синтез окремих компонентів);
- вікно вибору об’єктів (доступні різні елементи залежно від вибраного режиму);
- панель керування симуляцією (знаходиться у лівому нижньому кутку, містить такі команди: пуск; виконання одного такту, що вмикає симуляцію на час Single Step Time, який задається у розділі головного меню System>Set Animation Options; пауза; стоп).

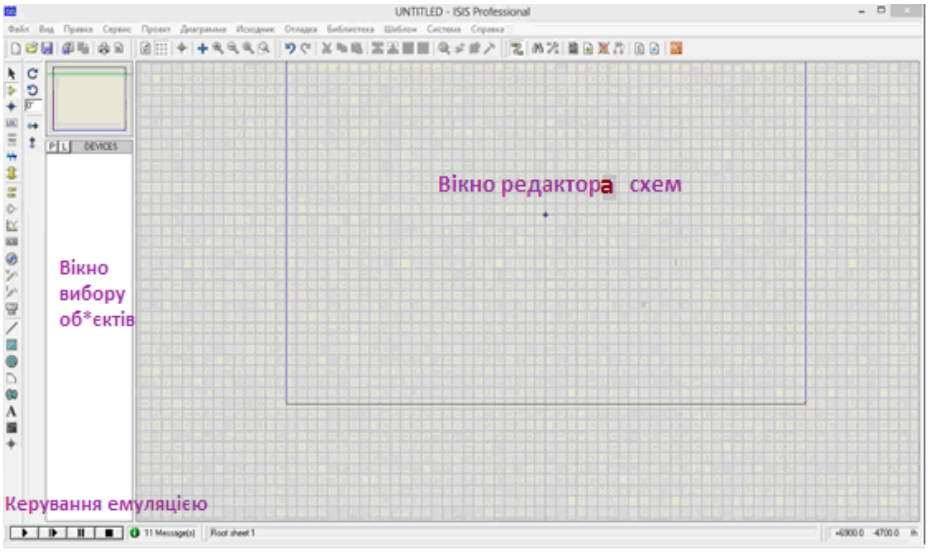


Рисунок 7.1 – Головне вікно Proteus VSM (ISIS)

7.2.2 Проектування віртуальної моделі електронної схеми та симуляція її роботи в Proteus VSM

Для того, щоб зібрати схему будь-якого пристрою, необхідно підготувати набір елементів, з яких буде складатися ця схема. Для цього

переходимо в режим компонентів (рис. 7.2, а) і натискаємо клавішу P, яка знаходиться під вікном перегляду поряд з клавішею L (рис. 7.2, а, б)

а) б)

Рисунок 7.2 – Панель інструментів

Перед нами з’являється менеджер компонентів, який пропонує на наш вибір всі елементи, що містяться в бібліотеці програми (рис 7.3).

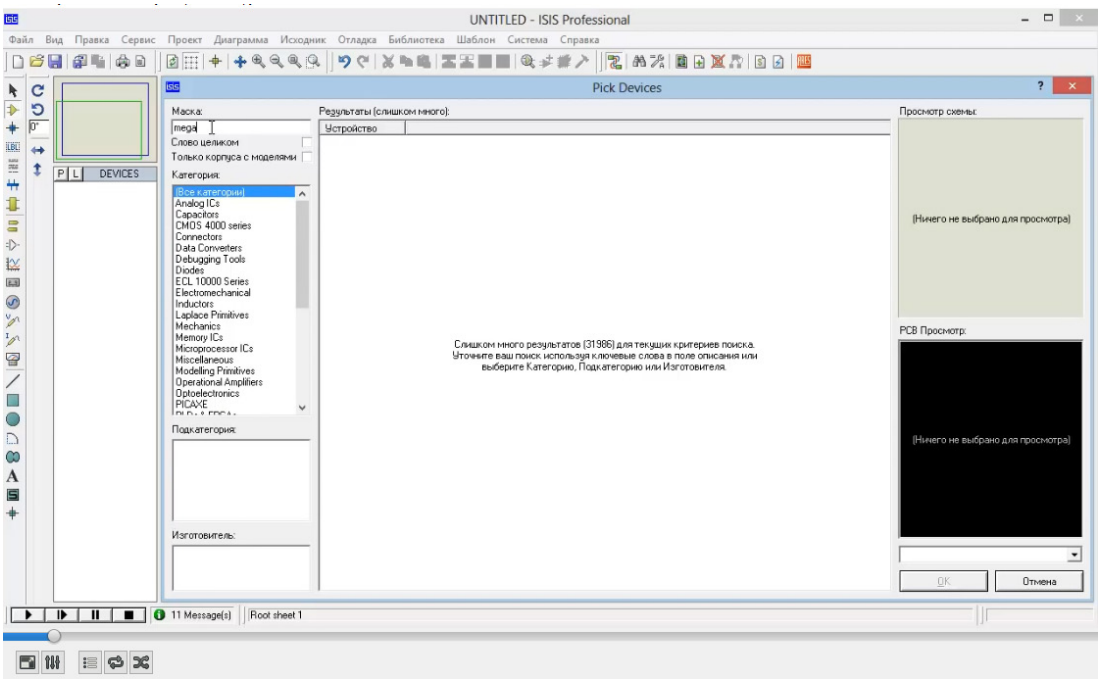


Рисунок 7.3 – Вікно менеджера компонентів

Потрібно користуватися рядком пошуку, який знаходиться у верхньому лівому кутку. Коли потрібний компонент знайдений, подвійне натискання лівої кнопки миші по його назві додасть його до переліку використовуваних компонентів. Наприклад, створимо електронну схему, яка буде складатися зі: світлодіода, резистора, кнопки і блока живлення (батареї). У рядку пошуку (маска) вводимо перший елемент, який будемо додавати до схеми: led-green. У списку елементів з’являється світлодіод з такою назвою (рис.7.4), двічі «клацаємо» по ньому лівою кнопкою миші (ЛКМ).

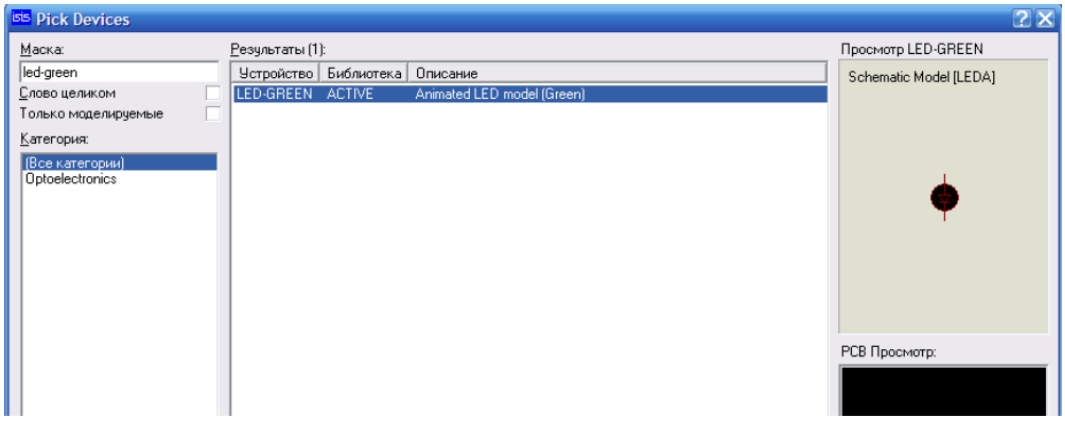


Рисунок 7.4 – Пошук компонентів

Далі так само робимо пошук за такими назвами: resistor, push button, battery (бібліотека – Active), потім по черзі подвійним натисканням по цих елементах у списку додаємо їх до нашого проекту. Після того як ми вибрали всі компоненти, натискаємо на кнопку ОК. Тепер наше вікно вибору об’єктів буде містити всі чотири компоненти, що ми їх вибрали, і виглядати, як зображено на рисунку 7.5.

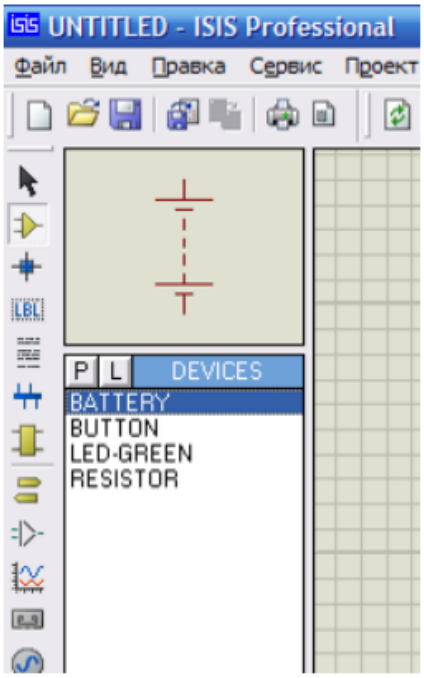
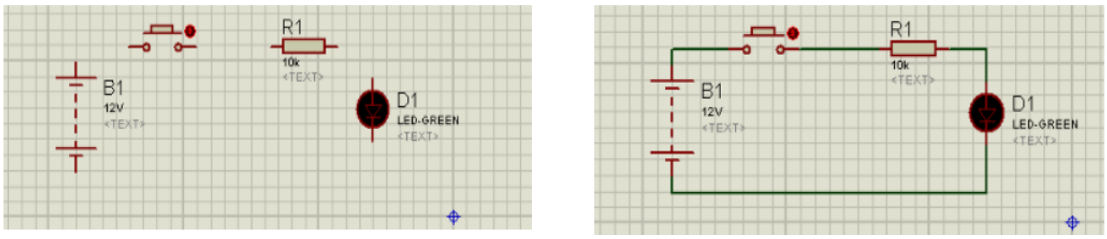


Рисунок 7.5 – Вікно вибору об’єктів

Для встановлення компонента у вікні редактора схеми його потрібно вибрати зі списку і подвійним натисканням лівої кнопки миші (ЛКМ) встановити у потрібному місці. До того, як встановити компонент на схемі, його можна попередньо розвернути у потрібне положення, що можна

контролювати у вікні перегляду. Наприклад, потрібно всі компоненти, які зображено на рис. 7.6, а), з'єднати, як показано на рис. 7.6, б).

Для того, щоб з'єднати два компоненти між собою, для початку потрібно вибрати інструмент **Стрілка**, потім на самій схемі навести курсор миші на кінець елемента, має з'явитися квадратний червоний контур, натиснути і відпустити ЛКМ, вести курсор до контакту з іншим елементом, на кінці іншого елемента також натиснути ЛКМ, щоб завершити побудову з'єднання між двома елементами (рис. 7.6, б).



а) б)

Рисунок 7.6 – Побудова електричної схеми

Після того як електричне коло створено, змінюємо характеристики наших компонентів. На початку батарея живить схему 12 В, а нам потрібно змінити живлення схеми на 5 В. Для цього подвійно натискаємо ЛКМ на батареї і у полі Voltage встановлюємо 5V.

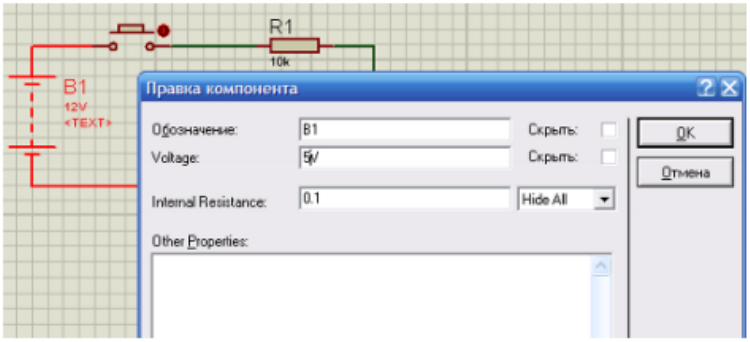


Рисунок 7.7 –Зміна параметрів живлення

Так само змінюємо опір резистора на 680 Ом.

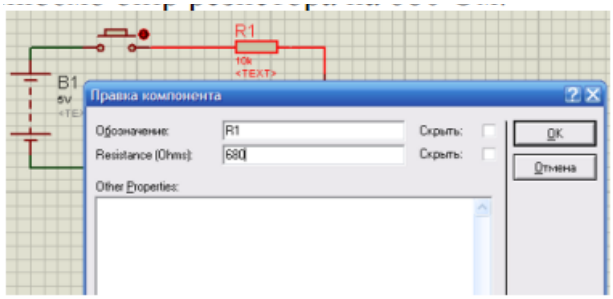


Рисунок 7.8 – Зміна параметрів опору R1

Потім натискаємо в панелі керування симуляцією кнопку «**Воспроизвести**».

Рисунок 7.9 – Панель керування симуляцією

Тепер наша схема може віртуально відтворити вмикання та вимикання світлодіода при натисканні на кнопку (рис 7.10). Для цього потрібно просто навести курсор миші на елемент кнопка у схемі та натиснути ЛКМ.

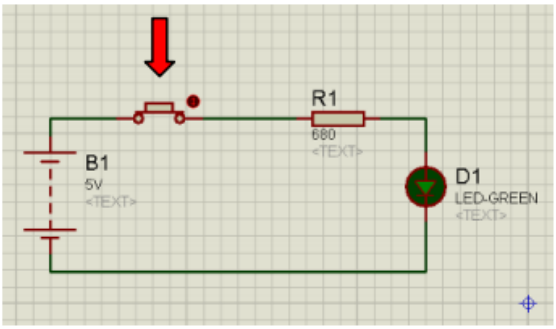


Рисунок 7.10 – Симуляція роботи світлодіода

7.3 Створення проекту програми для мікроконтролера в середовищах Atmel Studio 6.2 і Proteus

7.3.1 Опис роботи в середовищі Atmel Studio 6.2

Atmel Studio – інтегроване середовище розробки (IDE) для розробки восьми- та тридцятидвобітових додатків від компанії Atmel, що працює в операційних системах Windows NT/2000/XP/Vista/7. Atmel Studio містить асемблер і симулятор, що дозволяє відстежити виконання програми. Atmel Studio містить в собі менеджер проектів, редактор вихідного коду, інструменти віртуальної симуляції та внутрішньосхемного налагоджування, дозволяє писати програми мовами асемблер або C/C + +.

Після того, як ви встановите Atmel Studio 6.2 на свій комп'ютер, створіть новий проект.

1. Відкрийте Atmel Studio 6.2 та виберіть **New Project**. У діалоговому вікні New Project виберіть **GCC C Executable Project** як шаблон для створення С-коду (рис. 7.11). Введіть ім'я проекту і вкажіть місце, де він буде зберігатися. Назовемо наш проект «BlinkLED». Зніміть прапорець **Create directory for solution** для спрощення структури каталогів проекту. Натисніть **OK.**

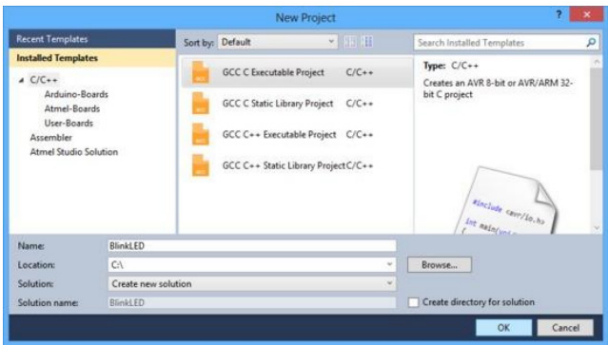


Рисунок 7.11 – Діалогове вікно New Project в Atmel Studio 6. 2

2. У вікні Device Selection (рис. 7.12), оберіть ім'я пристрою AVR, яке необхідно. Натисніть **OK** для створення нового проекту.

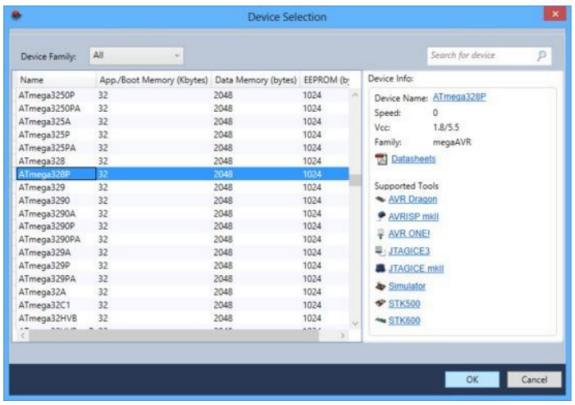


Рисунок 7.12 – Діалогове вікно Device Selection в Atmel Studio 6.2

3. Напишіть код у BlinkLED.cpp, наведений нижче:

```
#define F_CPU 8000000      // частота AVR в Гц, використовується для util/delay.h
#include <avr/io.h>
#include <unit/delay.h>
int main ( )  {
        DDRD |= (1<<DDD1);   // встановлення LED виходу PD1 на вихід
        while (1) {
                PORTD  |= (1<<PORT1);          // PD1 вимкнення
                _delay_ ms (50);                  // затримка на 50 мс
                PORTD & = ~(1<<PORTD1);    // PD1 вимкнення
                 _delay_ ms (100);                // затримка на 100 мс
                    }
                 }
```

4. На панелі інструментів натисніть кнопку **Build Solution** (або натисніть **F7**), щоб скомпілювати код (рис. 7.13).

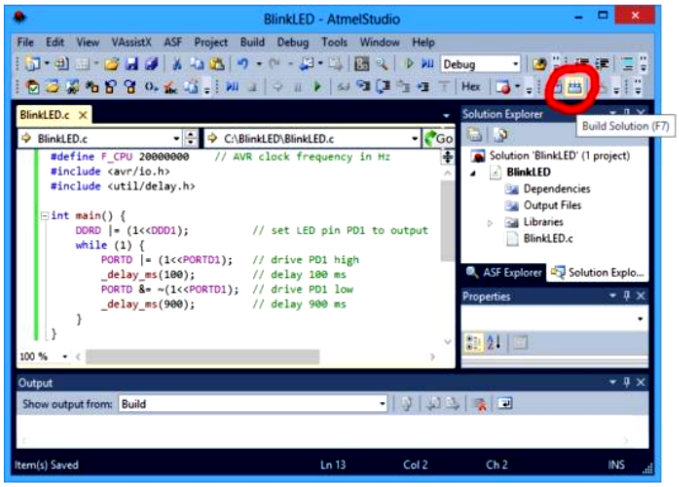


Рисунок 7.13 – Побудова проекту

Після компіляції перейдіть в директорію проекту і знайдіть файл BlinkLED.hex – це файл, який являє собою довільні двійкові дані у текстовому вигляді. З історичних причин є стандартом де-факто при прошиванні різноманітних мікросхем. Саме його необхідно буде завантажити у мікроконтролер схеми у Proteus VSM, яка буде побудована далі.

7.3.2 Симуляція роботи мікроконтролера за допомогою Proteus VSM

На рисунку 7.14 показано розведення виводів мікроконтролера ATMega328P. Світлодіод ми будемо під'єднувати до третього виводу мікроконтролера (PD1).

Мікроконтролер ATmega328P конструктивно виконаний у 32-вивідному корпусі типу TQFP і MLF (також випускається у 28-вивідному корпусі типу DIP) з максимальним числом контактів введення/виведення, що дорівнює 23: ATmega8, ATmega8L мають FLASH-пам'ять програм обсягом 8 Кбайтів, ОЗП обсягом 1 Кбайт та EEPROM-пам'ять даних обсягом 512 байтів.

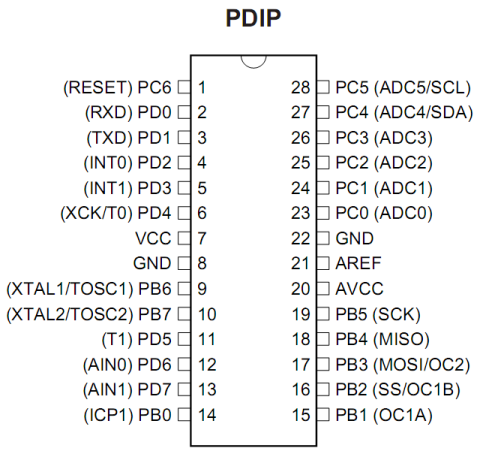


Рисунок 7.14 – Розташування виводів мікроконтролера ATMega328P

Таблиця 7.1 – Опис виводів мікроконтролера ATmega328P

Позначення	Номер		Тип	Опис
	DIP	TQFP		
Порт B. 8-бітовий двонаправлений порт введення/виведення з внутрішніми підтягувальними резисторами				
PB0 (ICP)	14	12	I/O	0-й біт порту B Вхід захоплення таймера/лічильника T1
PB1 (OC1A)	15	13	I/O	1-й біт порту B Вихід A таймера/лічильника T1
PB2 (SS/OC1B)	16	14	I/O	2-й біт порту B Вибір Slave-пристрою на шині SPI. Вихід B таймера/лічильника T1
PB3 (MOSI/OC2)	17	15	I/O	3-й біт порта B Вихід (Master) або вхід (Slave) даних модуля SPI. Вихід таймера/лічильника T2
PB4 (MISO)	18	16	I/O	4-й біт порту B Вхід (Master) або вихід (Slave) даних модуля SPI
PB5 (SCK)	19	17	I/O	5-й біт порту B Вихід (Master) або вхід (Slave) тактового сигналу модуля SPI
PB6 (XTAL1/TOSC1)	9	7	I/O	6-й біт порту B Вхід тактового генератора Вивід для під'єднання резонатора до таймера/лічильника T2
PB7 (XTAL2/TOSC2)	10	8	I/O	7-й біт порта B Вихід тактового генератора Вивід для підключення резонатора до таймера/лічильника T2
Порт C. 7-бітовий двонаправлений порт введення/виведення з внутрішніми підтягувальними резисторами				
PC0 (ADC0)	23	23	I/O	0-ий біт порту C Вхід АЦП
PC1 (ADC1)	24	24	I/O	1-ий біт порту C Вхід АЦП
PC2 (ADC2)	25	25	I/O	2-ий біт порту C Вхід АЦП
PC3 (ADC3)	26	26	I/O	3-ій біт порту C Вхід АЦП
PC4 (ADC4/SDA)	27	27	I/O	4-ий біт порту C Вхід АЦП Вхід/вихід даних модуля TWI
PC5 (ADC5/SCL)	28	28	I/O	5-ий біт порту C Вхід АЦП Вхід/вихід тактового сигналу модуля TWI
PC6 (RESET)	1	29	I/O	6-ий біт порту C Вхід скидання
ADC6	—	19	I	Вхід АЦП
ADC7	—	22	I	Вхід АЦП

Порт D. 8-бітовий двонаправлений порт введення/виведення з внутрішніми підтягувальними резисторами				
PDO (RXD)	2	30	I/O	0-ий біт порту D Вхід USART
PD1 (TXD)	3	31	I/O	1-ий біт порту D Вихід USART
PD2 (INTO)	4	32	I/O	2-ий біт порту D Вхід зовнішнього переривання
PD3 (INT1)	5	1	I/O	3-ій біт порту D Вхід зовнішнього переривання
PD4 (TE/XCK)	6	2	I/O	4-ий біт порту D Вхід зовнішнього тактового сигналу таймера/лічильника TE Вхід/вихід зовнішнього тактового сигналу USART
PD5 (T1)	11	9	I/O	5-ий біт порту D Вхід зовнішнього тактового сигналу таймера/лічильника T1
PD6 (AINO)	12	10	I/O	6-ий біт порту D Прямий вхід аналогового компаратора
PD7 (AIN1)	13	11	I/O	7-ий біт порту D Інверсний вхід аналогового компаратора
AREF	21	20	p	Вхід опорної напруги для АЦП
AVCC	20	18	p	Вивід джерела живлення АЦП
VCC	7	4,6	p	Вивід джерела живлення
GND	8,22	3,5,21	p	Загальний вивід

Створіть новий проект у Proteus VSM і накресліть схему, яку зображено на рисунку 7.15. У менеджері компонентів знайдіть нижченаведені елементи:

1 світлодіод;

2 мікроконтролер ATMega328P;

3 резистор;

4 заземлення.

Для того, щоб створити елемент «заземлення» (GROUND), перейдіть у панелі інструментів (знаходиться вертикально зліва) на вкладку Terminals Mode.

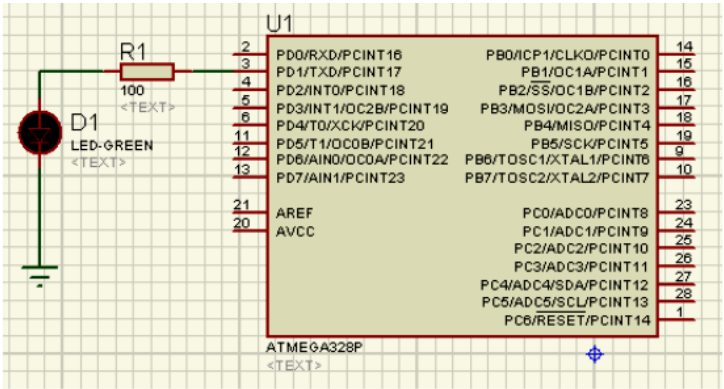


Рисунок 7.15 – Схема під’єднання

Після з’єднання елементів у електричне коло двічі натисніть на зображення мікроконтролера. Відкриється вікно Edit Component (рис.º7.16), в якому у полі Program File вкажіть шлях до файлу BlinkLED.hex. також перевірте, щоб була встановлена частота мікроконтролера така сама, як і у програмному коді, тобто 8 МГц.

Після цього на панелі керування симуляцією натискаємо на **Play**, і якщо все правильно було зроблено, спостерігаємо роботу схеми.

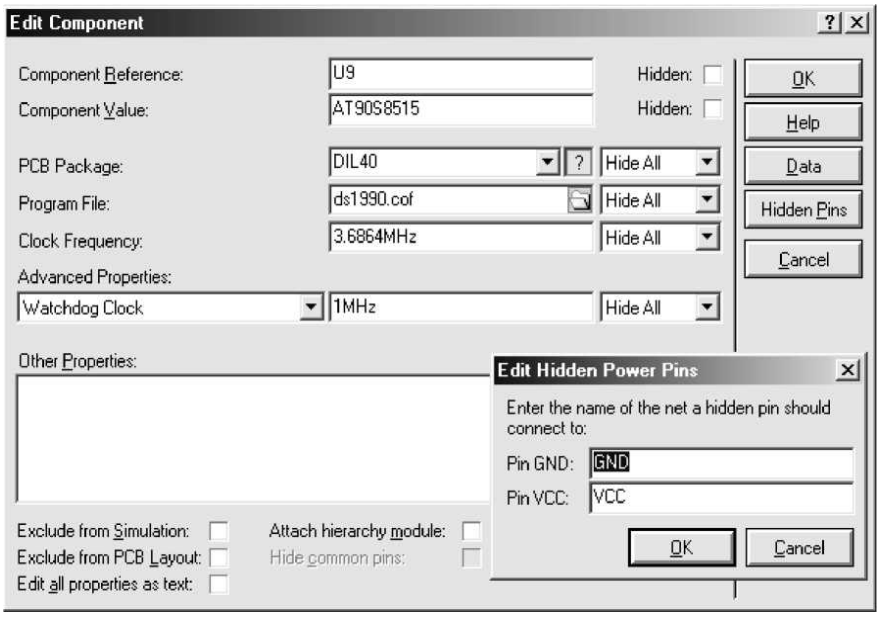


Рисунок 7.16 – Вікно Edit Component

7.4 Приклад розробки проекту в середовищах Atmel Studio 6.2 і Proteus

Завдання. Розробити програму і виконати симуляцію мерехтіння трьох різних світлодіодів, підключених до мікроконтролера ATMega8. Схема підключення у середовищі Proteus VSM наведена нижче на рис. 7.17.

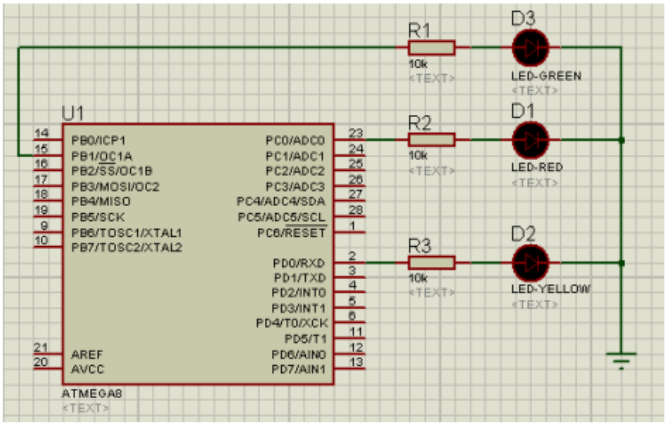


Рисунок 7.17 – Завдання до лабораторної роботи

Послідовність розробки

1. Запустіть Proteus ISIS. Найбільший простір відведено під вікно редагування EDIT WINDOW. В ньому відбуваються всі основні процеси створення, редагування та налагоджування схеми пристрою. Зліва вгорі знаходиться вікно попереднього перегляду Overview Window, за допомогою якого можна переміщуватись по вікні редагування. Під вікном попереднього перегляду знаходиться Object Selector – список обраних в даний момент компонентів, символів й інших елементів. Виділений в списку об'єкт відображається у вікні попереднього перегляду. Всі можливі функції та інструменти Proteus ISIS доступні через меню, розташоване вгорі основного вікна програми, через піктограми, що знаходяться під меню і в лівому кутку основного вікна та через «гарячі» клавіші. Внизу головного вікна розташовані: панель управління інтерактивною симуляцією (кнопки ПУСК-ПОКРОКОВИЙ РЕЖИМ-ПАУЗА-СТОП), рядок статусу (у ньому відображаються помилки, підказки, поточний стан процесу симуляції і т. ін.) і координати курсора, що відображаються в дюймах. Для маніпулювання об'єктами їх потрібно виділити за допомогою правої кнопки миші. Для виділення групи можна, утримуючи CTRL, послідовно натискати праву кнопку на всіх об'єктах або, утримуючи праву кнопку, протягнути область виділення по необхідних об'єктах. Повторне натискання правої кнопки миші по виділеному об'єкту видаляє його (ще можна видалити об'єкти, натиснувши на кнопку DELETE).

2. Відкрийте бібліотеку компонентів. Для цього натисніть кнопкою P (на клавіатурі) по піктограмі Pick Devices, виберіть елемент Pick Devices в меню Library, або двічі натисніть ліву кнопку у полі вибору компонентів Object Selector. Компоненти можна вибирати за категоріями Category, підкатегоріями Sub-categories, за виробником Manufacturer або ж шукати за ключовими словами Keywords. Для підтвердження вибору компонента натисніть лівою кнопкою по рядку з назвою компонента.

Натисніть кнопку OK і помістіть компонент на схему, натиснувши два рази ліву кнопку.

3. Зберіть схему, що зображена на рис. 7.15. З'єднуйте компоненти за допомогою лівої кнопки миші. Якщо необхідно, розгорніть їх за допомогою правої кнопки миші. Елементи типу «земля» вибираються в режимі INTER SHEET TERMINAL в лівій частині екрана.

4. Додайте hex-файл з програмою в проект. Для цього натисніть правою кнопкою мишки на контролері. Оберіть пункт Edit Properties. У меню Program File виберіть файл з розширенням *.hex у папці, яка створювалась для проекту в AVR Studio. У рядку PROCESSOR CLOCK FREQUENCY виставіть тактову частоту процесора. Збережіть проект.

5. Натисніть кнопку ПУСК на панелі управління інтерактивною симуляцією. Подивіться на результат роботи схеми.

6. Оформіть звіт з лабораторної роботи згідно з ЄСКД. Висновки до роботи мають стисло та зрозуміло пояснювати результати, отримані на кожному кроці виконання роботи згідно з наведеним вище порядком.

7.5 Контрольні запитання та завдання

1. Які мікроконтролери підтримує емулятор електронних пристроїв PROTEUS?
2. Що відображає панель DEVICES?
3. Де потрібно шукати необхідні елементи, для побудови схеми?
4. Як задати частоту тактування МК при емуляції?
5. Як задати необхідний час емуляції?
6. Як створити новий проект в Atmel Studio 6.2?
7. На яких мовах можна писати програми в Atmel Studio 6.2?
8. На які сегменти ділиться пам’ять мікроконтролерів AVR?
9. Завдання на лабораторну роботу.

Скласти програму, яка з використанням вказаного в таблиці 7.2 часу затримки, забезпечує перемикання світлодіодів, підключених до вказаного в таблиці порта, відповідно з заданим алгоритмом перемикання. Тактова частота мікроконтролера – 1024 кГц.

Таблиця 7.2 – Варіанти завдань

№ варіанта	Алгоритм роботи	Порт	Час затримки, мс
1	Послідовно, починаючи зі старшого розряду	A	65
2	Послідовно, починаючи з молодшого розряду	B	130
3	Послідовно, з країв до середини	C	195
4	Послідовно, з середини до країв	A	32,5
5	Послідовно, парні розряди порту	D	325
6	Послідовно, непарні розряди порту	C	260
7	Послідовно, 1, 2, 3, 6, 7, 8 розряди	B	390

Література

1. Олссон Г. Цифровые системы автоматизации и управления / Г.ºОлссон, Д. Пиани. – СПб. : Невский диалект, 2001. – 557 с.
2. Гребнев В. В. Микроконтроллеры семейства AVR фирмы Atmel / Гребнев В. В. – М. : ИП Радио Софт, 2002.
3. Биков М. М. Основи МП техніки : лабораторний практикум / М.ºМ.ºБиков, Г. Л. Лисенко, В. А. Лужецький, Т. Б. Мартинюк. – Вінниця : ВДТУ, 2003. – 64 с.
4. Баранов В. Н. Применение микроконтроллеров AVR: схемы, алгоритмы, программы / Баранов В. Н. – М. : Издательский дом «Додэка-XXI», 2004. – 288 с.
5. Голубцов М. С. Микроконтроллеры AVR: от простого к сложному / М. С. Голубцов, А. В. Кириченкова. – [2-е изд. испр. и доп.]. – М. : СОЛОН-Пресс, 2004.
6. Евстифеев А. В. Микроконтроллеры AVR семейства Tiny и Mega фирмы «ATMEL» / Евстифеев А. В. – М. : Издательский дом «Додэка-XXI», 2004. – 560 с.
7. Иванов Ю. И. Микропроцессорные устройства систем управления : учебное пособие / Ю. И. Иванов, В. Л. Югай. – Таганрог : Изд-во ТРТУ, 2005. – 133 с.
8. Программирование на языке С для AVR и PIC микроконтроллеров / Сост. Ю. А. Шпак – К. : «МК-Пресс», 2006. – 400 с.
9. Хартов В. Я. Микроконтроллеры AVR. Практикум для начинающих / Хартов В. Я. – М. : Изд-во МГТУ им. Н. Э. Баумана, 2007. – 240 с.
10. Методичні вказівки до виконання лабораторних робіт з дисципліни «Обчислювальні та мікропроцесорні засоби електронних апаратів» для студентів напрямів підготовки «Телекомунікації», «Радіотехніка» / Уклад. Дементьєв Ю. В. – Вінниця : ВНТУ, 2009. – 49 с.
11. Цирульник С. М. Проектування мікропроцесорних систем : навчальний посібник / С. М. Цирульник, Г. Л. Лисенко. – Вінниця : ВНТУ, 2010. – 201 с.
12. Биков М. М. Дискретний аналіз і теорія автоматів / М. М. Биков, В. Д. Червяков. – Суми : СУМДУ, 2016. – 354 с.
13. Proteus VSM. Пошаговая отладка. [Електронный ресурс]. – Режим доступа : http://easyelectronics.ru/sistema-modelirovaniya-isis-proteus-bystryj-start.html
14. Proteus VSM. Система. Руководство по интерактивному моделированию. [Електронный ресурс]. – Режим доступа : https://radioprog.ru/post/6
15. Proteus VSM. Система. Руководство по интерактивному моделированию. [Електронный ресурс]. – Режим доступа : http://cxem.net/software/proteus.php

Навчальне видання

Биков Микола Максимович
Грищук Тетяна Вікторівна
Ковтун Вячеслав Васильович

МІКРОПРОЦЕСОРНІ ЗАСОБИ СИСТЕМ УПРАВЛІННЯ

Лабораторний практикум

Рукопис оформив *М. Биков*

Редактор *В. Дружиніна*

Оригінал-макет виготовив *О. Ткачук*

Підписано до друку 11.04.2019.
Формат 29,7×42¼. Папір офсетний.
Гарнітура Times New Roman.
Друк різографічний. Ум. друк. арк. 7,38.
Наклад 50 (1-й запуск 1–21) пр. Зам. № 2019-055.

Видавець та виготовлювач
Вінницький національний технічний університет,
інформаційний редакційно-видавничий центр.
ВНТУ, ГНК, к. 114.
Хмельницьке шосе, 95,
м. Вінниця, 21021.
Тел. (0432) 65-18-06.
press.vntu.edu.ua;
E-mail: kivc.vntu@gmail.com.
Свідоцтво суб'єкта видавничої справи
серія ДК № 3516 від 01.07.2009 р.