Міністерство освіти і науки України
Чорноморський національний університет імені Петра Могили

Сідєлєв М. І., Запальський В. М., Бєліков О. Є.

ПРОГРАМУВАННЯ СПЕЦІАЛІЗОВАНИХ МІКРОКОНТРОЛЕРНИХ ТА ВБУДОВАНИХ КОМП'ЮТЕРНИХ СИСТЕМ ДЛЯ ЗАСОБІВ АВТОМАТИЗАЦІЇ

Навчальний посібник

Миколаїв – 2023

УДК 621.38.049.77
С 34

Рекомендовано до друку вченою радою Чорноморського національного університету імені Петра Могили (протокол № 5 від 10 червня 2021 р.)

Рецензенти:

Жук О. К., канд. техн. наук, доцент, професор кафедри програмованої електроніки, електротехніки і телекомунікацій Національного університету кораблебудування ім. адм. Макарова.

Козел В. М., канд. техн. наук, доцент кафедри інформаційних технологій Херсонського національного технічного університету.

С 34

Сідєлєв М. І. Програмування спеціалізованих мікроконтролерних та вбудованих комп'ютерних систем для засобів автоматизації : навчальний посібник / М. І. Сідєлєв, В. М. Запальський, О. Є. Бєліков. – Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2023. – 96 с.

ISBN 978-966-336-434-6

У навчальному посібнику наведені основи конструювання та програмування мікроконтролерних схем керування, що можуть використовуватись в автоматизованих системах різного призначення, наприклад, у виробничих та переробних технологіях, охорони підприємств, мобільних об'єктах. Курс оснований на вивченні архітектури та програмуванні мікроконтролерів сімейства AVR компанії ATMEL. Програми для прошивки мікроконтролерів складаються в оболонці ATMEL STUDIO, а перевірка схем на функціональність виконується в пакеті прикладних програм PROTEUS VSM. Курс представлений 8-а лабораторними роботами, орієнтованими на початківця з подальшими ускладненнями.

Навчальний посібник призначений для студентів спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології», аспірантів, молодих вчених та інженерів різних технічних спеціальностей.

УДК 621.38.049.77

ISBN 978-966-336-434-6

Зміст

Вступ

Метою цього посібника поставлено навчання студентів основам програмування мікропроцесорної техніки з точки зору використання одержаних знань та навичок для будування систем цифрової обробки інформації на базі стандартних промислових контролерів.

Основними завданнями вивчення дисципліни «Програмування спеціалізованих мікроконтролерних та вбудованих комп'ютерних систем для засобів автоматизації» є: формування поняття про можливості пристосування сучасних мікроконтролерів у побудові приладів і вузлів систем автоматизації; закладання основ побудування алгоритмів управління та обробки інформації в мікропроцесорних системах; знайомство з основними пакетами прикладних програм, що використовуються під час проєктування мікроконтролерних приладів та систем. Вивчення цієї дисципліни базується на початкових знаннях, отриманих під час засвоєння загальних курсів дисциплін: «Комп'ютерні технології та програмування» (алгоритмізація та мови програмування C#, C++, Assembler), «Електротехніка та електроніка» (склад, структура персональних ЕОМ та мікроконтролерів) тощо.

У результаті вивчення курсу «Програмування спеціалізованих мікроконтролерних та вбудованих комп'ютерних систем для засобів автоматизації» студент повинен:

знати: предмет і завдання дисципліни як основи програмування мікроконтролерних систем, принципи програмування на мовах C++ та Асемблері для мікроконтролерів; алгоритми збору, обробки та передачі сигнальних даних із застосуванням мікроконтролерів, особливості роботи з середовищем програмування ATMEL STUDIO та програмним симулятором PROTEUS;

уміти: працювати з промисловими мікроконтролерами та налагоджуючими пристроями; складати та налаштовувати програми; формулювати принципи та алгоритми роботи пристроїв.

Дисципліна «Програмування спеціалізованих мікроконтролерних та вбудованих комп'ютерних систем для засобів автоматизації» пропонується ЗВО за вибором студентів циклу професійної та практичної підготовки для навчання магістрів зі спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології».

Розділ 1.

Системи моделювання мікроконтролерних систем

Напевно багато початківців радіоаматорів стикалися з ситуацією коли, вирішивши зібрати вподобаний і, безсумнівно, потрібний пристрій, через недосвідченість, а може, через помилки в схемі, спалювали насилу придбані дорогі радіодеталі. І скоріше за все більшість, обпікшись на перших невдачах, закидали заняття радіоелектронікою назавжди. Але в роки повальної комп'ютеризації знайшовся вихід з цього глухого кута.

З'явилося безліч програм симуляторів, котрі використовують символи реальних радіодеталей та приладів для створення віртуальних моделей. Симулятори дозволяють, без складання реального пристрою, налагодити роботу схеми, знайти помилки, отримані на стадії проєктування, зняти необхідні характеристики і багато іншого.

Одна з таких програм – PROTEUS VSM. Але симуляція радіоелементів це не єдина здатність програми. PROTEUS VSM, що створена фірмою Labcenter Electronics на основі ядра SPICE 3F5 університету Berkeley, є так званим середовищем наскрізного проєктування. Це означає створення пристрою, починаючи з його графічного зображення (принципової схеми) і закінчуючи виготовленням друкованої плати пристрою, з можливістю контролю на кожному етапі виробництва. Але, незважаючи на згадану складність програми, користуватися нею зможуть не тільки професіонали у світі радіоелектроніки, але й новачки.

У склад PROTEUS VSM входять як найпростіші аналогові пристрої так і складні системи, що створені на популярних нині мікроконтролерах. Доступна величезна бібліотека моделей елементів, поповнювати котру може сам користувач, природньо для цього потрібно досконально знати роботу елемента чи схеми і вміти програмувати. Можливість анімації схем дозволяє програмі стати прекрасним навчальним

посібником на уроках у закладі вищої освіти (ЗВО). Достатній набір інструментів і функцій, серед яких вольтметр, амперметр, осцилограф, всілякі генератори, інструменти налагодження програмного забезпечення мікроконтролерів, роблять PROTEUS VSM хорошим помічником розробнику електронних пристроїв.

Сайт розробників програми розміщений на http://www.labcenter.co.uk.

PROTEUS VSM за замовчуванням встановлюється в папку C:\Program Files\Labcenter Electronics.

PROTEUS VSM складається з двох самостійних програм ISIS и ARES. ARES – це трасовик друкованих плат з можливістю створення своїх бібліотек корпусів. Основною програмою є ISIS, в ній передбачений гарячий зв'язок з ARES для передачі проєкту для розведення плати.

1.1. Програма синтезу та моделювання електронних схем

Під час запуску програми ISIS з'являється головне вікно (див. рис. 1.1).

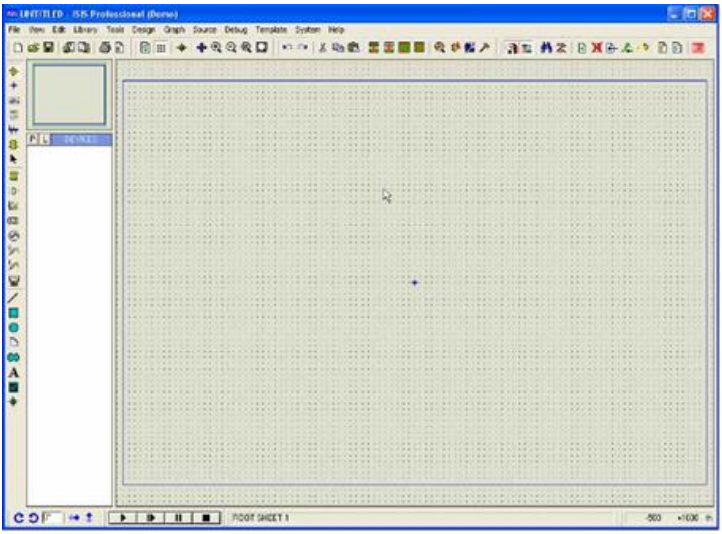


Рисунок 1.1. – Головне вікно ISIS

Найбільший простір відведений під вікно редагування Edit Window. Саме в ньому відбуваються всі основні процеси створення, редагування та налагодження схеми пристрою.

Ліворуч угорі маленьке вікно попереднього перегляду Overview Window (див. рис. 1.2). За його допомогою можна переміщатися вікном редагування (натискаючи ліву кнопку миші у вікні перегляду, ми переміщуємо вікно редагування за схемою, якщо звичайно схема не вміщаються у вікно).

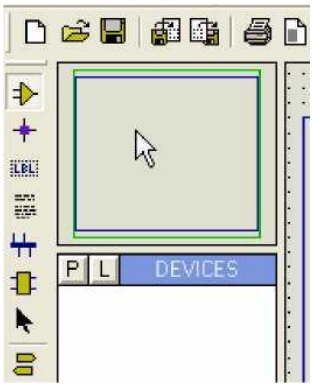


Рисунок 1.2. – Вікно попереднього перегляду Overview Window

Переміщувати вікно редагування за схемою можна ще так: утримуючи кнопку SHIFT, рухати курсор миші, не натискаючи її кнопок, у вікні редагування.

Наближати і віддаляти схему у вікні можна відповідно кнопками F6 і F7 або ж колесом миші, F5 центрує схему у вікні, а натискання F8 підлаштовує розмір схеми під вікно редагування.

Під вікном попереднього перегляду знаходиться Object Selector – список обраних на цей момент компонентів, символів та інших елементів. Виділений у списку об'єкт відображається у вікні попереднього перегляду.

Усі можливі функції та інструменти PROTEUS VSM доступні через меню, що розташоване зверху основного вікна програми, через піктограми, що знаходяться під меню і в лівому куті основного вікна і через «гарячі» клавіші, які можуть перепризначатися користувачем.

Унизу основного вікна розташовані: з лівої сторони на праву кнопки обертання та розвороту об'єкта навколо своєї осі (див. рис. 1.3, а) і функції: пуск, покроковий режим, пауза, стоп (див. рис. 1.3, б).

а)

б)

Рисунок 1.3. – Кнопки управління процесом симуляції

Для освоєння основних функцій програми використаємо один з наявних прикладів: виберемо в меню FILE опцію LOAD DESIGN та завантажимо файл SAMPLE / ANIMATION CIRCUIT / AC01DSN (див. рис. 4).

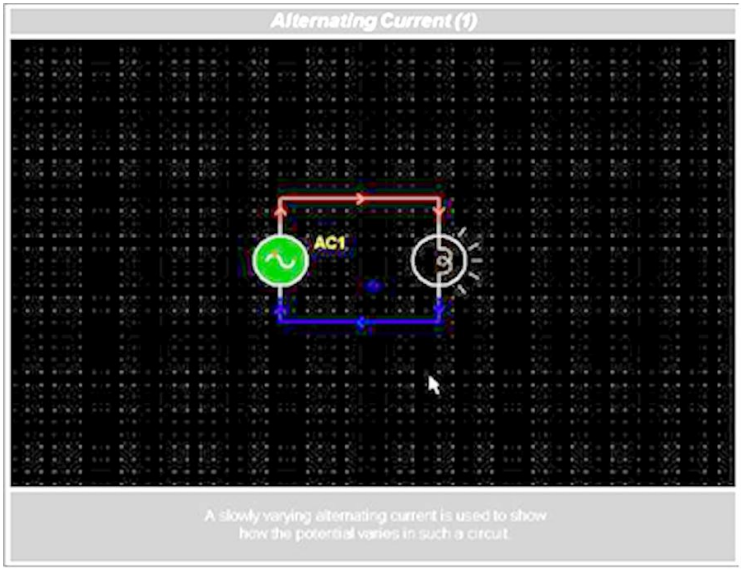


Рисунок 1.4. – Приклад AC01.DSN

Запустимо проєкт, натиснувши на панелі кнопку ПУСК (див. рис. 5).

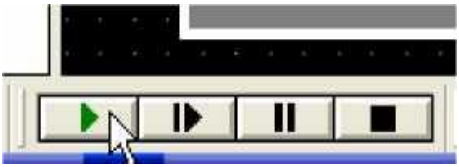


Рисунок 1.5. – Запуск проєкту AC01.DSN

Ця схема демонструє дію змінного струму в ланцюзі. Частота генератора занижена до 0.5 Гц для наочності.

Колір і яскравість проводів визначають полярність і рівень напруги, стрілки – напрямок струму. Червона точка на зображенні генератора показує поточне «положення» синусоїди.

Для того, щоб маніпулювати об'єктами, їх потрібно спочатку виділити (це можна зробити тільки на зупиненому проєкті). Для виділення одного об'єкта треба натиснути на ньому правою кнопкою миші. Для виділення групи можна, або утримуючи CTRL послідовно натискати правою кнопкою всі об'єкти або утримуючи праву кнопку протягнути область виділення необхідних об'єктів. Виділяти об'єкти треба обережно, повторне натискання правої кнопки миші на виділеному об'єкті видалить його (видалити виділені об'єкти можна кнопкою DELETE). Скасувати останні і всі попередні дії послідовно можна за допомогою кнопок скасування (UNDO, REDO) (див. рис. 1.6). Кнопки скасування діють як назад, за хронологією, так і вперед.

Рисунок 1.6. – Відміна або повторення останніх дій конструктора

Виділені об'єкти можна переміщувати схемою, схопивши їх лівою кнопкою миші, перетягнути в потрібне місце та відпустити кнопку. А за допомогою цих кнопок виконуються групові операції з виділеними об'єктами. Один за одним: копіювання, переміщення, поворот і видалення (див. рис. 1.7).

Рисунок 1.7 – Кнопки копіювання, переміщення, повороту і видалення

Відкриємо наступний проєкт Diode07.DSN, що знаходиться в тій же папці, що й попередній, попередній закриється, автоматично запитавши вас «Не бажаєте зберегти зміни?». Дайте відмову і запустіть проєкт (див. рис. 1.8).

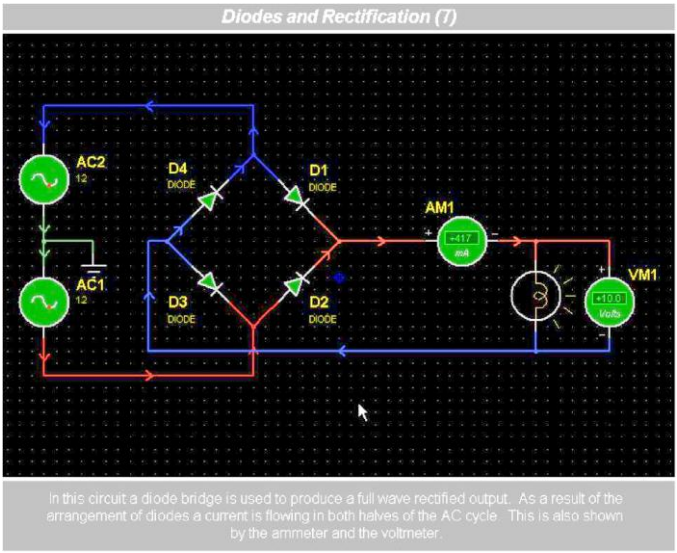


Рисунок 1.8. – Приклад Diode07.DSN

Проєкт ілюструє роботу двонапівперіодного випрямляча, тобто діодного моста. Добре видно всі процеси, що проходять у схемі. Так само, як і в попередньому проєкті, частота генераторів знижена. Переробимо схему в реальну. Нам потрібна частота 50 Гц. Для цього доведеться відредагувати властивості генераторів. Щоб відкрити вікно редагування компонента, потрібно або виділити компонент, натиснув-

ши на ньому лівою кнопкою миші, або помістивши на нього курсор, не натискаючи кнопок миші, натиснути CTRL + E. Відкриємо вікно редагування (див. рис. 1.9).

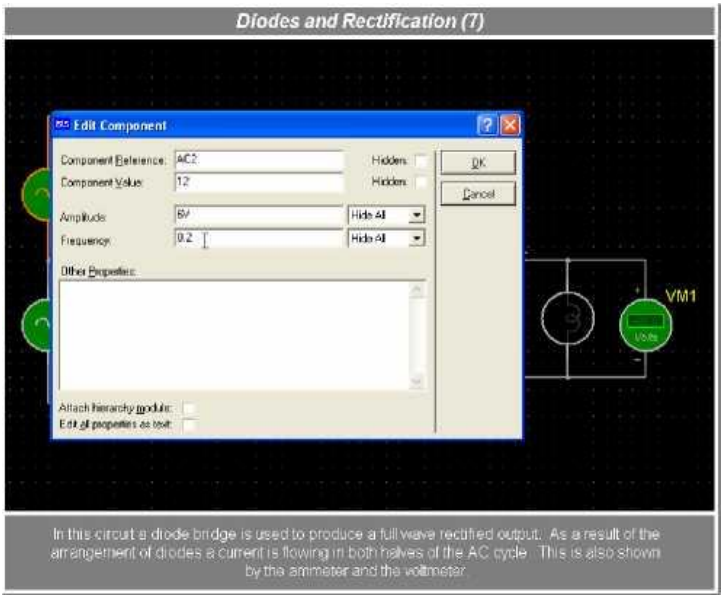


Рисунок 1.9. – Вікно редагування компонента

Внесемо в поле Frequency (частота) число 50. Закриємо вікно, натиснувши OK. Змінимо частоту і для другого генератора. Додамо в схему конденсатор, але той, що знаходиться в списку, CAPACITOR доведеться замінити. Справа в тому, що проєкт існує з найпершої версії програми, модель розробники змінили, а ввести зміни в схему не завдали собі клопоту. Такого роду незбігів в старих проєктах досить.

Усі елементи знаходяться, як на складі, в бібліотеці компонентів. Щоб потрапити на цей «склад» перейдемо в режим COMPONENT (компоненти), натиснувши відповідну піктограму (див. рис. 1.10).

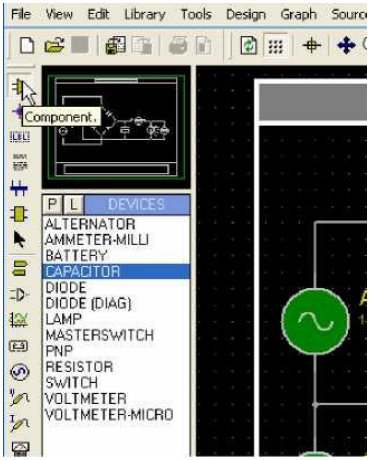


Рисунок 1.10. – Перехід в режим COMPONENT

Тепер або натиснувши на піктограму P (Pick devices), або двічі натиснувши лівою кнопкою у полі вибору компонентів Object Selector, ми потрапимо на «склад» (див. рис. 1.11).

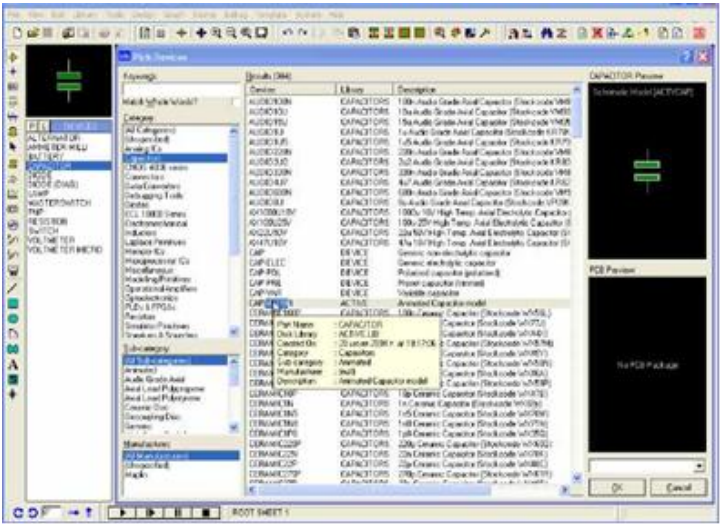


Рисунок 1.11. – «Склад» компонентів

Компоненти можна вибирати за категоріями Category, підкатегоріями – Sub category, чи виробникам Manufacturer або ж шукати за ключовими словами Keywords. Виберемо CAPACITOR бібліотеки ACTIV. Двічі натиснувши у рядку з назвою об'єкта, підтверджуючи вибір компонента.

Програма запитає, чи бажаєте ви замінити існуючий у списку схеми компонент попутно повідомивши, що компонент бібліотеки і схеми мають різний час створення (бібліотечний свіже, див. рис. 1.12).

Рисунок 1.12. – Попередження про оновлення екземпляра елемента

Дайте відповідь «Так». Закрийте «склад», натиснувши ОК чи ж закривши вікно. Виберіть компонент, у переліку, натиснувши на нього лівою кнопкою. Зображення конденсатора з'явитися у вікні попереднього перегляду (див. рис. 1.13). Якщо необхідно, розгорніть його як вам потрібно.

Рисунок 1.13. – Зображення конденсатора
у вікні попереднього перегляду

Помістимо конденсатор після діодного моста, натиснувши там лівою кнопкою (див. рис. 1.14).

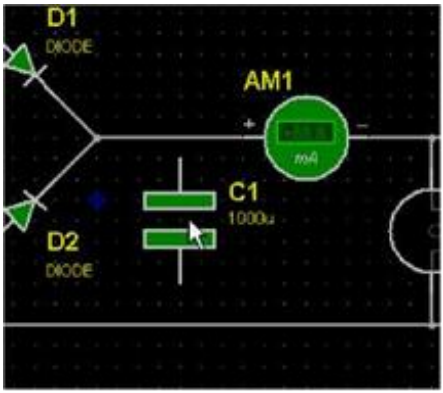


Рисунок 1.14. – Конденсатор, розміщений у схемі

Тепер нам треба приєднати його до схеми. Помістіть курсор на верхній вивід конденсатора, на кінці курсора з'явиться хрестик, котрий показує, що з'єднання можливе. Натисніть ліву кнопку, перемістіть курсор на дріт, вище конденсатора, з'явиться тонка лінія, що показує можливі з'єднання. Коли курсор буде над дротом, знову з'явиться хрестик. Натисніть ліву кнопку ще раз.

З'єднайте нижній вивід самі. Змініть ємність конденсатора на 500 мкФ. Запустіть симуляцію. Кількість плюсів і мінусів на обкладинках конденсатора вказує на рівень заряду. Змініть назад частоту обох генераторів на 0,2 Гц. У якості роздільника десяткового дробу PROTEUS VSM використовує точку (англійська розкладка клавіатури). Запустимо проєкт і побачимо процес зарядки-розрядки конденсатора в динаміці.

Ми вже навчилися відкривати проєкт, запускати його, переміщатися схемою, маніпулювати об'єктами, редагувати їх властивості та додавати елементи в схему.

Тепер навчимося застосовувати органи управління схеми, котрі називали в PROTEUS-активаторами. Відкриємо проєкт Basic07.DSN (див. рис. 1.18).

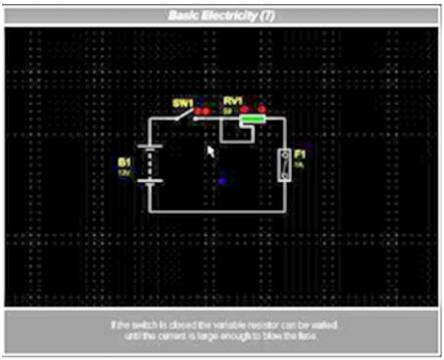


Рисунок 1.18. – Проєкт Basic07.DSN

Найпростіша схема. Запустіть проєкт. У тумблера і реостата є червоні стрілки в кутах. Це і є активатори. Натискаючи на них лівою кнопкою можна перемикати тумблер або ж переміщати движок реостата, змінюючи, таким чином, його опір. Увімкніть тумблер і перемістіть движок реостата в крайнє праве положення (див. рис. 1.19).

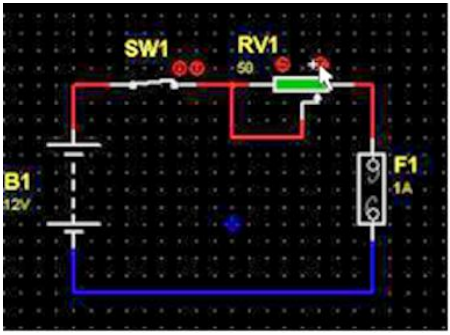


Рисунок 1.19. – Експеримент із запобіжником

1.2. Моделювання мікроконтролерних схем

Мікроконтролерні схеми відрізняються від аналогових та «чисто» логічних схем тим, що вони потребують прошивки спеціальної програми у Flash-пам'ять мікроконтролера.

Встановимо для зручності свої розміри листа схеми. Відкриємо меню SYSTEM> SET SHEET SIZE (встановити розміри листа). Виберемо варіант USER – призначений для користувача, у віконцях введемо 6 in 4 in (висота і ширина в дюймах). Після цього натисніть F8, щоб наблизити розмір аркуша схеми під вікно редагування. Зберемо схему згідно з рис. 1.20.

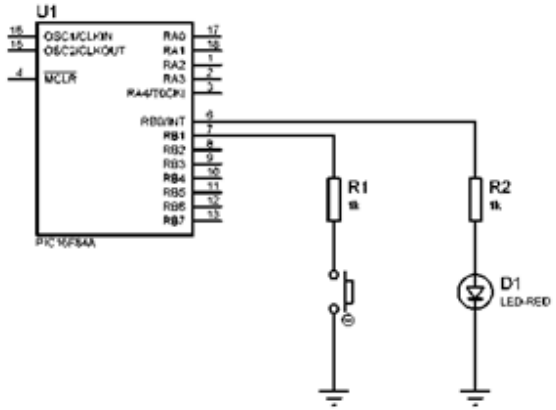


Рисунок 1.20. – Схема з кнопкою та світлодіодом

Для початку визначимося зі списком деталей.

Отже, нам потрібні: мікроконтролер PIC16E84Л 1 шт., світлодіод червоний 1 шт., кнопка і два резистори по 1 кОм кожен.

Кварц, конденсатори та батареї емулюються програмно, і немає необхідності додавати їх у схему. Хоча якщо ви будете розробляти проєкт до його логічного кінця, тобто до виготовлення друкованої плати, елементи доведеться додати.

Відкриваємо бібліотеку компонентів. Щоб довго не шукати, наберіть у вікні KEYWORDS pic16f84a. Тепер або двічі натиснувши ENTER, але тоді закриється бібліотека і доведеться заново її відкривати, або двічі натисніть ліву кнопку у рядку з описом компонента, що з’явився у вікні RESULTS (результат), перемістіть обраний вами компонент в список Object Selector. Виберіть резистор, набравши RES, кнопку BUTTON і світлодіод LED-RED.

Компоненти вибираються по одному примірнику, розмножити їх можна вже потім обравши в списку Object Selector. Закриємо бібліотеку, натиснувши OK або ж вікно. З часом до вас прийде досвід,

і ви будете самі визначати, які компоненти і де вони знаходяться. Якщо, все пройшло без помилок, то у вікні Object Selector з'явиться список вибраних нами компонентів. Якщо так, то розмістимо їх на схемі, натиснувши ліву кнопку спочатку нв назві компонента в списку, а потім у потрібному нам місці на порожній поки що схемі. Додайте і розгорніть, якщо це необхідно всі компоненти. У підсумку отримаємо щось на зразок цього (див. рис. 1.21).

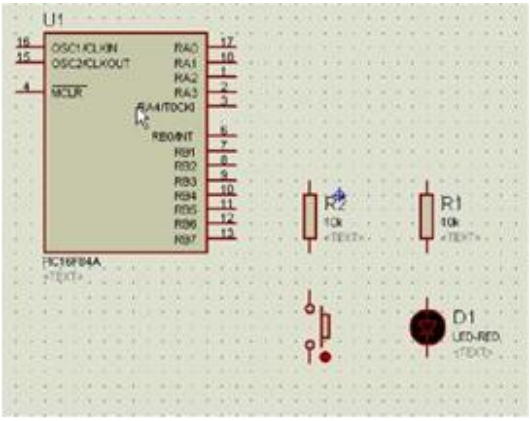


Рисунок 1.21. – Розміщені компоненти для схеми

Не вистачає ще одного важливого елемента – «землі» або «корпусу». Елементи такого типу (термінали) вибираються в режимі INTER SHEET TERMINAL (див. рис.1.22).

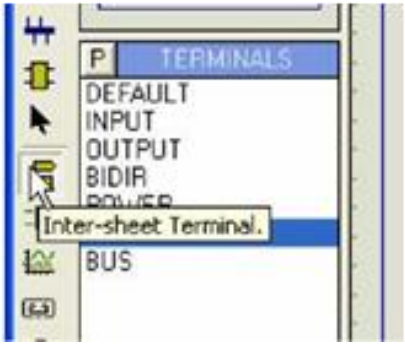


Рисунок 1.22. – Вибирання GROUND

Виберіть елемент GROUND (земля) і розмістіть його на схемі під кнопкою і світлодіодом. Тепер з'єднайте компоненти між собою як показано на схемі. Як проводити з'єднання показувалося вище на прикладі конденсатора.

Змініть опір резисторів на 1 кОм. Також змініть тип моделі резисторів на digital (цифровий), це необхідно, щоб симулятор не витрачав час на обрахування аналогових властивостей резисторів (див. рис. 1.23). Нам потрібно тільки, горить чи ні світлодіод і натиснута кнопка чи ні, тобто тільки логічні рівні.

Рисунок 1.23. – Налаштування резистора як цифрового

Схема готова. Збережемо проєкт у своїй папці, щоб не плутатися, під ім'ям LED.DSN. Зробимо невеликий відступ. Якщо ви збираєте схему на мікроконтролерах і хочете тільки налагодити програму, не намагайтеся точно відтворити схему з джерела. За можливості менше застосовуйте аналогові пристрої або ж намагайтеся замінити їх цифровими примітивами. Багато моделей мають два варіанти – аналоговий і цифровий, наприклад резистор. Транзистори, що працюють ключами, можна замінити або інверторами, або буферами, зважаючи на їх провідність.

Це допоможе розвантажити процесор. Зазвичай, PROTEUS має більш просунуті засоби вирішення цієї проблеми, наприклад, «магнітофон» TAPE, який дозволяє розбити схему на кілька частин, записати сигнали з однієї частини в проміжний файл, потім заморозити цю частину і використовувати записані сигнали для інших частин. Але цей варіант, якщо і буде розглянуто, то тільки не зараз. Так

що намагайтеся максимально спростити схему, тим більше, якщо у вас не дуже швидкодіючий комп'ютер. Для того, щоб оживити схему, нам необхідний файл з кодом програми для мікроконтролера. Яким чином його отримати, буде розглянуто у наступному розділі.

Розглянемо один із методів спрощення схем. Створимо новий проєкт з тим самим розміром схеми, як на попередньому аркуші. Наберемо такі елементи: PIC16F84, 7SEG-MPX4CC-BLUE – це семисегментний чотирьохрозрядний індикатор із синім світінням, знаходиться в категорії OPTOELECTRONICS. Підключимо PORTA до катодів, а PORTB до сегментів (див. рис. 1.24).

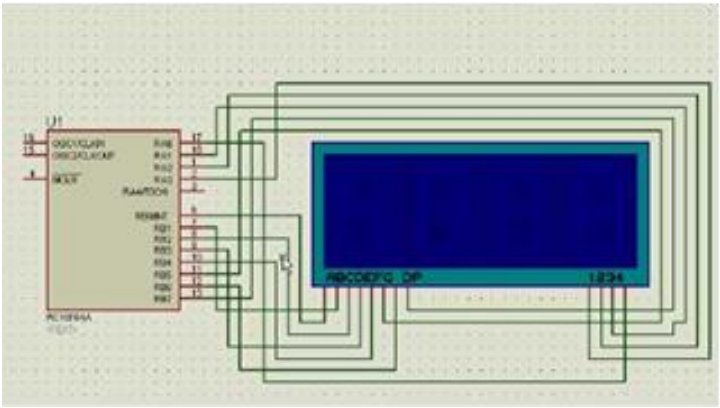


Рисунок 1.24. – Використання семисегментного індикатора

Ми отримали велике нагромадження дротів і це лише на прикладі двох компонентів схеми. От якби всі дроти зв'язати в «пучок»... Видалимо всі з'єднання, виберемо режим BUS (див. рис. 1.25).

Рисунок 1.25. – Вибирання шини (BUS)

Тепер, відзначаючи лівою кнопкою точки, проведемо шину як на малюнку. Кінець шини відзначається тисненням правої кнопки (див. рис. 1.26).

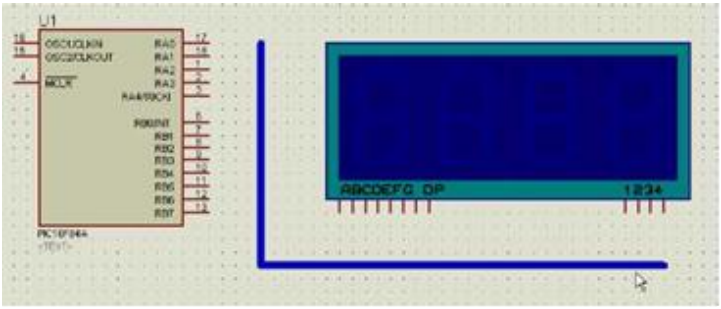


Рисунок 1.26. – Розміщення шини (BUS)

З'єднаємо дроти мікроконтролера і індикатора безпосередньо з шиною, та визначимо, який куди підключений. Для цього існують мітки. Перейдемо в режим WIRE LABLE (мітки дроти), натиснувши на піктограмі з написом LBL. Починаємо з індикатора, натискаючи на дріт, ми відкриваємо вікно введення і редагування мітки. Що б не заплутатися, даємо імена міткам відповідно до імен дротів індикатора (A-A B-B 1-1 тощо).

Якщо не «попадаємо» по дроту, збільшуємо схему, покрутивши колесо миші або натиснувши F6. Закінчивши з індикатором переходимо до мікроконтролера. Тут вже немає необхідності вводити з клавіатури символи. Усі ті символи, які ми ввели, вже присутні в списку. Натисніть в полі введення кнопку вибору і ми завжди побачимо: PA0-4 RA3-1, RB0-A ..RB7-H (DP).

Дещо про інструменти. Спочатку виберемо осцилограф. Включимо режим VIRTUAL INSTRUMENTS і в списку, що з'явився виберемо OSCILLOSCOPE. Осцилограф двоканальний. Підключимо канал A до катода 1, канал B до катода 2. Запустимо проєкт. Якщо вийшла інша картинка, встановіть ручки і перемикачі відповідно до рисунка (див. рис. 1.27). Експериментуйте з органами управління осцилографом. Управління таке ж, як і у реального. У разі виникнення питань, звертайтеся до відповідної літератури.

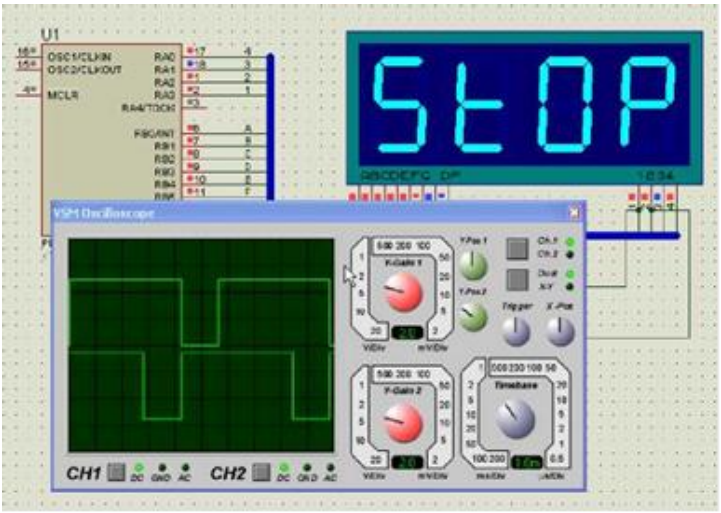


Рисунок 1.27. – Симуляція з осцилографом

Аналізатор логічних рівнів. Виберемо режим інструментів. У вікні вибору натиснемо LOGIC ANALISER. Розмістимо та під'єднаємо його, як показано на рис. 1.28.

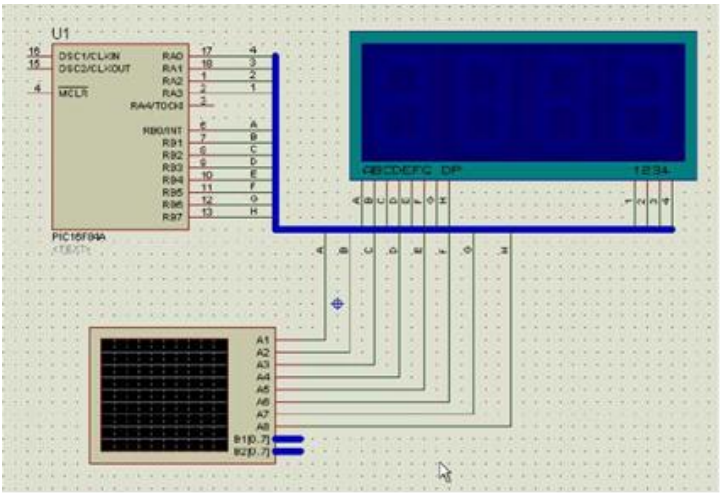


Рисунок 1.28. – Використання логічного аналізатора

Поставимо проєкт на паузу, натиснувши відповідну кнопку. Налаштуємо аналізатор, як показано на рис. 1.29 і запускаємо проєкт, натиснувши ПУСК.

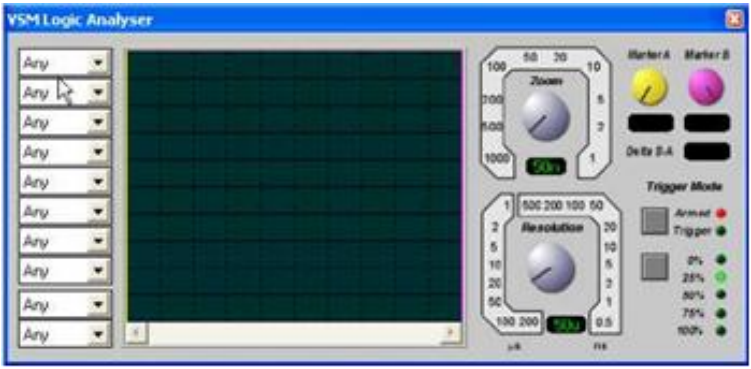


Рисунок 1.29. – Налаштування логічного аналізатора

Тригер спрацює, і на екрані аналізатора з'явиться таке (див. рис. 1.30):

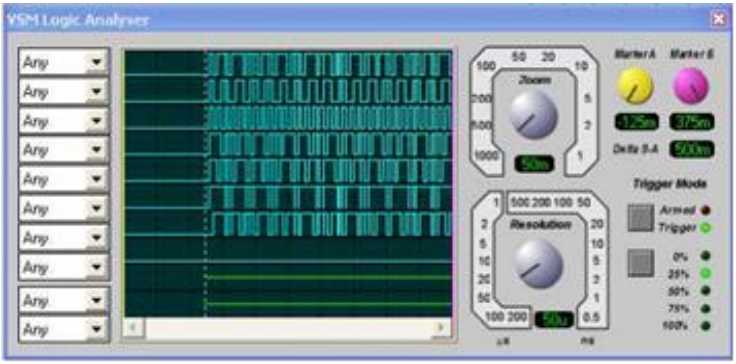


Рисунок 1.30. – Індикація логічного аналізатора

До аналізатора можна підключати вісім одиночних каналів і дві шини. Ліворуч на екрані знаходяться вікна вибору умов для спрацювання тригера. Вісім – для каналів і два – для шин. Для каналів умови визначаються логічними рівнями, а для шин шістнадцятковим числом від 0 до FF. Тригер спрацює, якщо всі умови співпадуть. Регулятори праворуч екрана: нижній – це вибір дозволу, верхній – збільшення

екрана перегляду. Кнопка з написом Armed Trigger взведений тригер: під нею кнопка вибору заповнення буфера аналізатора до і після точки спрацьовування тригера, яка позначена на екрані пунктирною вертикальною лінією. Два регулятори зверху переміщують тимчасові маркери, обидва рівнозначні, але мають різні кольори. У віконцях під регуляторами відображається час від початку буфера до маркера. У віконці з написом Delta B–A показана різниця значень верхніх вікон, тобто «відстань» від одного маркера до іншого.

Є більш потужні засоби аналізу, ніж описані вище осцилограф і логічний аналізатор. Зібрані вони в групу SIMULATION GRAPH. Опис кожного з них, їх можливостей займе багато часу. Тому наведемо лише невеликий приклад, який допоможе нам розібратися в принципі використання аналізаторів.

SIMULATION GRAPH (див. рис. 1.31). У списку натиснемо DIGITAL (цифровий). Тепер, в порожньому місці схеми, не відпускаючи лівої кнопки, розтягнемо прямокутник. Це вікно аналізатора. Зараз нам потрібно додати пробники. Включимо режим VOLTAGE PROBE. Розмістимо пробники на дротах, що йдуть до катодів індикатора, просто натискаючи на них лівою кнопкою. Приєднуючись, пробники, автоматично приймають мітку дрота. Потім перетягуємо одним за одним пробники у вікно аналізатора, спочатку виділивши його правою кнопкою, а потім, схопивши лівою відправити у вікно.

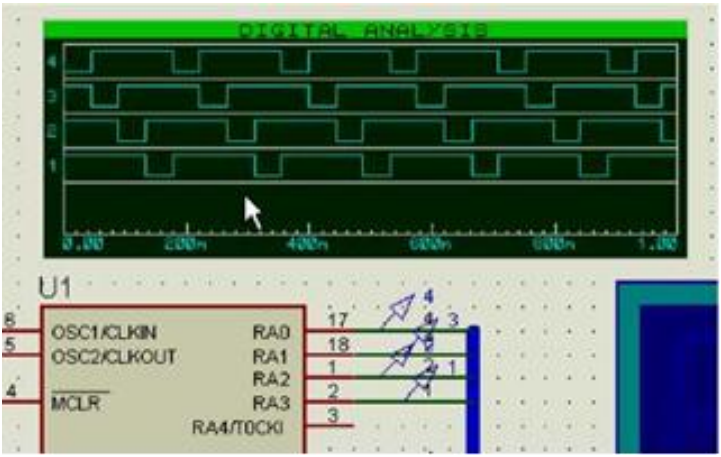


Рисунок 1.31. – Цифровий аналізатор

Якщо не виходить, спробуйте збільшити схему. Тепер помістивши курсор у вікні аналізатора, не натискаючи кнопок миші, натисніть пробіл. Щоб розгорнути або навпаки згорнути вікно аналізатора натисніть лівою кнопкою на зелену смугу вгорі вікна.

Розділ 2.

Системи програмування мікроконтролерних систем

2.1. Загальні відомості про інтегроване середовище розробки Atmel Studio

Atmel Studio 6 – це інтегроване середовище розробки (IDE, Integrated Development Envішnment), котре надає інструменти для розробки та відлагодження програмного забезпечення для мікроконтролерів (МК) фірми Atmel.

Це середовище дозволяє створювати програмні додатки на мові Assembler та C/C++. Atmel Studio 6 має вбудовану бібліотеку прикладів Atmel Software Framework (ASF). Вона дозволяє створювати програмне забезпечення для двох архітектур – восьми розрядних МК з RISC архітектурою та тридцяти двох розрядної архітектури ARM.

Крім того, Atmel Studio підтримує набір розробника STK500, що дозволяє програмувати AVR-пристрої, а також новий вбудований емулятор JTAG. Емулятор не є складовою частиною Atmel-Studio, тому розглядається тільки програмний симулятор.

Після встановлення Atmel-Studio за замовчуванням, на робочому столі Windows створюється відповідний ярлик. Для запуску програми Atmel-Studio можна скористатись ярликом на робочому столі, або виконати команду ***Пуск►Все программы ►Atmel ►Atmel Studio 6.0.***

Під час запуску цієї програми з'являється стартове вікно, зображене на рис. 2.1

Це вікно умовно поділено на три частини. Ліва – відповідає за керування та створення проєктів. У верхній частині цієї панелі доступні пункти створення та відкриття проєктів. Докладніше про кожен з пунктів:

New Project – під час натиснення на цей напис відкривається вікно створення нового проєкту.

New Example Project from ASF – використовується для створення нового проєкту для відлагоджувальних комплектів AVR (наприклад STK 500). Використовується бібліотека програмного коду Atmel Software Framework, що значно спрощує написання програмного коду.

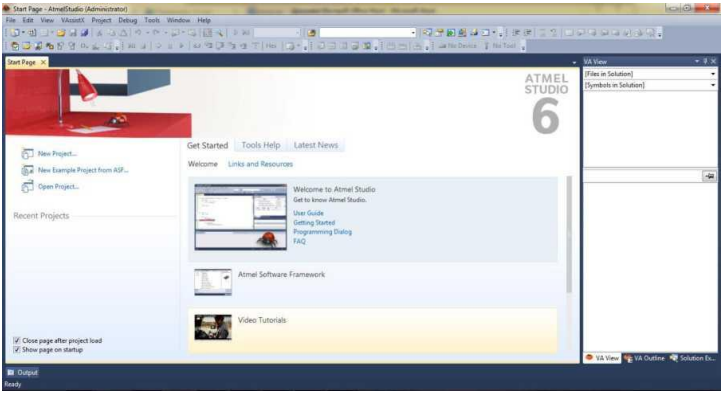


Рисунок 2.1. – Стартове вікно Atmel Studio 6

Open Project – відкриття вже існуючого проєкту збереженого на диску.

Recent Projects – у цій частині панелі відображаються раніше відкриті проєкти, тому досить легко повернутися до проєкту, уникаючи використання ***Open Project.***

У середній частині вікна відображені різні інформаційні ресурси, які необхідні для досить легкого освоєння середовища користувачем.

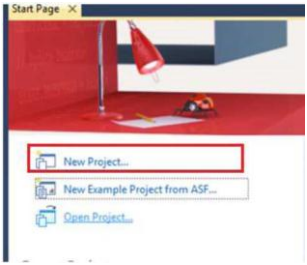


Рисунок 2.2. – Панель створення нового проєкту

Як приклад створимо новий проєкт. Для цього в стартовому вікні треба натиснути ***New Project*** (ця дія зображена на рис. 2.2).

У результаті виконання цієї дії відкривається вікно створення нового проєкту.

Вікно дозволяє обрати мову програмування, що використовується для написання програми для МК, тип проєкту, назву та розташування його на диску.

Для створення нового проєкту необхідно насамперед обрати мову на якій буде створюватися ПО МК. Для цього потрібно на лівій панелі відкрити вкладку Installed Templates і обрати мову програмування (в нашому випадку C/C++). Далі на середній панелі обрати необхідний тип проєкту (на правій панелі виводяться підказки які детальніше описують обраний тип).

Оскільки у цьому курсі для написання програм буде використано мову програмування С, варто обрати **GCC Executable Project.**

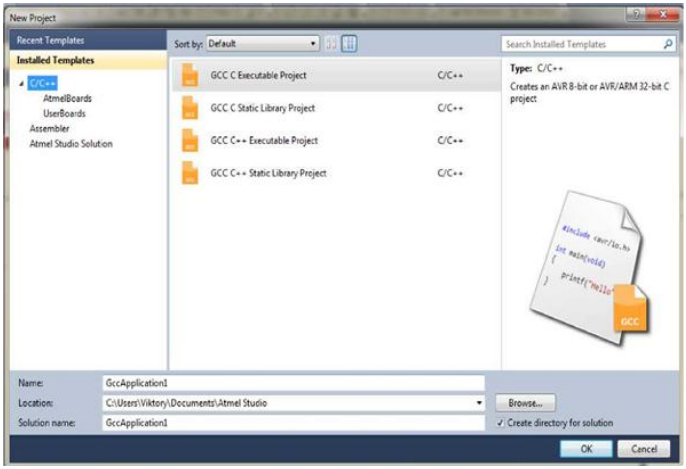


Рисунок 2.3. – Вікно створення проєкту

Для завершення створення проєкту необхідно вказати його ім'я в полі ***Name***, та за допомогою кнопки ***Browse*** обрати папку, де він буде збережений. Для завершення створення проєкту натиснути кнопку ***OK.***

Після цього відкривається вікно вибору мікроконтролера зображене на рис. 2.4. У вікні вибору МК є випадаюче меню ***Device Family.*** Завдяки меню можна обрати сімейство МК, щоб спростити пошук потрібного мікроконтролера. Також у цьому вікні є рядок пошуку, за допомогою якого можна знайти необхідний МК.

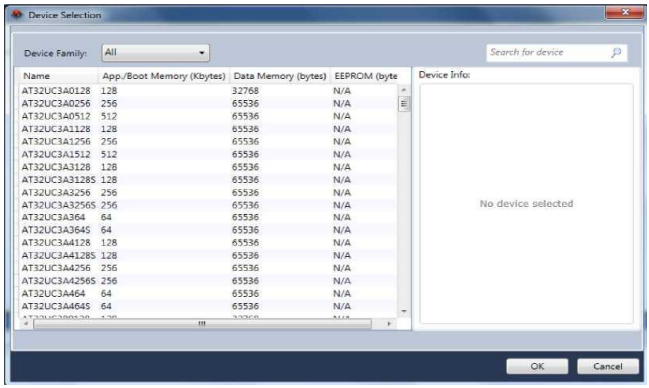


Рисунок 2.4. – Вікно вибору МК

В цьому курсі у якості мікроконтролеру буде використовуватись Atmega8. Цей МК має більшість периферії притаманної сімейству Mega AVR. Тому він досить зручний «навчальний снаряд» для освоєння мікроконтролерів.

Обравши необхідний МК на правій панелі вікна вибору мікроконтролера відображається посилання на datasheet та підтримувані відлагоджувальні засоби (див. рис. 2.5) .

Після натиснення кнопки ОК створюється новий проєкт та відкривається головне вікно з основними панелями та редактором коду (див. рис. 2.6).

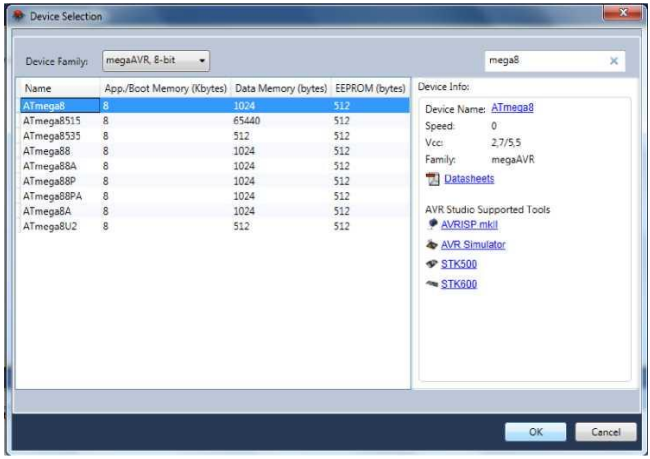


Рисунок 2.5. – Додаткова інформація

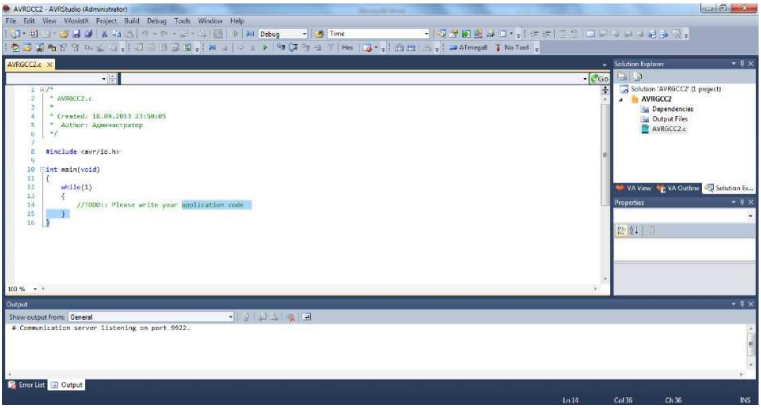


Рисунок 2.6. – Головне вікно

2.2. Робота з редактором коду

Під час створення нового проєкту редактор стартує з написаним у ньому стандартним кодом-заготівкою:

```
/*
* GccApplication1.c
*
* Created: 28/05/2021 22:36:01
* Author: Nickolay Siddelev
*/

#include <avr/io.h>
int main(void)
{
	while(1)
	{
		//TODO:: Please write your application code
	}
}
```

У коді автоматично підключається бібліотека *<avr/io.h>* у якій описані всі регістри мікроконтролера (це поняття варто розуміти як співвідношення імені та його адреси, необхідне для спрощення написання програми).

Також цей код містить головну функцію *int main(void)*. У мові програмування C/C++ саме з цієї функції починається виконання коду мікроконтролером.

Зазвичай мікроконтролерні програми зациклюються за допомогою циклу:

```
while(1)
{
      //TODO:: Please write your application code
}
```

Це дозволяє програмі, розташованій у цьому циклі, виконуватися допоки на мікроконтролер подається живлення.

Редактор коду Atmel studio має автодоповнення коду. Це дуже корисна функція, особливо для роботи з ARM мікроконтролерами. Вона сприяє більшій зручності у поцесі набору коду. Під час набору коду випадає меню з варіантами зображене на рис. 2.7.

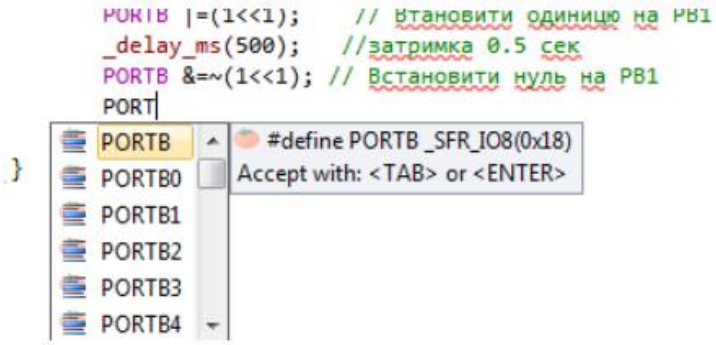


Рисунок 2.7. – Меню автодоповнювання коду, що випадає

У меню, що випадає, навігація здійснюється завдяки клавішам вверх/вниз, або за допомогою колеса миші. Вибір необхідного елемента здійснюється клавішею TAB або ENTER.

2.3. Створення нової програми. Компіляція

1. Набрати код (коментарі набирати не обов'язково):

```
/*
* GccApplication1.c
*
* Created: 28/05/2021 22:36:01
* Author: Nickolay
*/
#define F_CPU 8000000UL      // Частота роботи мікроконтролера
#include <util/delay.h>      //Бібліотека порграмних заримок
#include <avr/io.h>
    int main(void)
    {
    DDRB |=(1<<1);           // PB1 На вихід
    while(1)
    {
           PORTB |=(1<<1);  // Встановити одиницю на PB1
           _delay_ms(500);  // затримка 0.5 сек
           PORTB &=~(1<<1); // Встановити нуль на PB1
           _delay_ms(500);  // затримка 0.5 сек
    }
    }
```

Програма являє собою «класичний» приклад з якого починається вивчення нового сімейства мікроконтролерів. Дані змінюють логічні рівні на виводі PB1 порту B. Якщо до цього виводу підключити світлодіод, то можна спостерігати мигання з частотою 1 Гц.

Мікроконтролер напряму не розуміє ні одну з підтримуваних мов програмування Atmel Studio. Він працює з машинними кодами.

Тому треба виконати компіляцію. Компілятор (англ. Compiler від англ. to compile – збирати в ціле) – комп'ютерна програма (або набір комп'ютерних програм), що перетворює (компілює) програмний код, написаний певною мовою програмування (мова джерела, англ. source language), на семантично еквівалентний код в іншій мові програмування (мова цілі, англ. target language), котрий, як правило, необхідний для виконання програми машиною, наприклад, комп'ютером.

Компілятор можна визначити, як програму або технічний засіб, що виконує компіляцію. Історично компілятором називалася програма що зв'язувала підпрограми, чим й зумовлено походження слова. Сьогодні це завдання виконує компонувальник.

В Atmel Studio за функції компіляції та компоновки відповідає меню ***Build.*** Воно зображене на рис. 2.8.

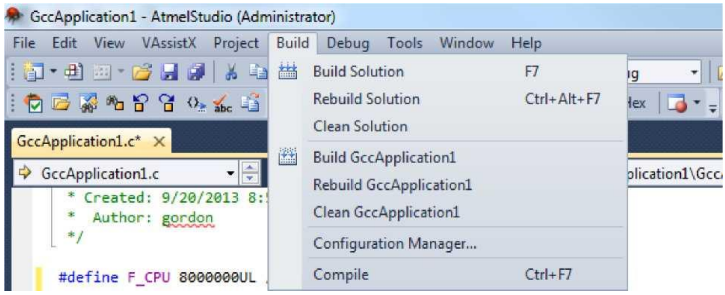


Рисунок 2.8. – Меню Build

Для того, щоб скомпілювати проєкт, необхідно натиснути ***Build Solution*** або натиснути клавішу ***F7.***

Якщо код написано без помилок, у вікні ***Output*** з'явиться лог компіляції і відомості про розмір скомпільованого коду (Program Memory Usage) (див. рис. 2.9).

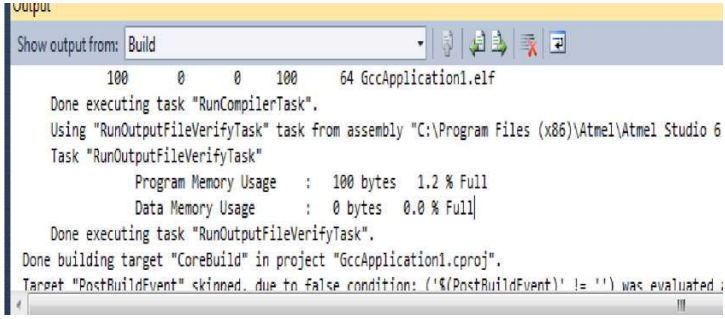


Рисунок 2.9. – Вікно Output

У результаті компіляції формуються файли програмного забезпечення (ПЗ) мікроконтролера, котрі безпосередньо використовують для «прошивки» мікроконтролера. Цей файл зберігається за замовчуванням у папці ***Debug***, яка знаходиться у папці з проєктом. Файл має розширення ***.hex.*** Інколи для прошивки мікроконтролера також потрібен файл з розширенням ***.eep***.

Якщо в коді виявлені помилки в нижній частині вікна з'явиться ***Error List*** зображений на рис. 2.10.

У панелі відображаються помилки в програмі. Для того, щоб перейти до місця виникнення помилки, необхідно подвійним кліком натиснути на неї.

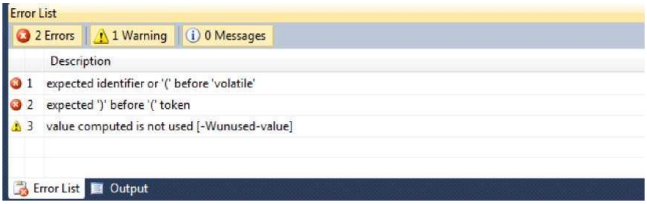


Рисунок 2.10. – Error List

2.4. Відлагодження програми на програмному симуляторі

В Atmel Studio існує симулятор роботи мікроконтролера. Усі інструменти відлагодження програми згруповані в меню ***Debug.*** Воно зображено на рис. 2.11.

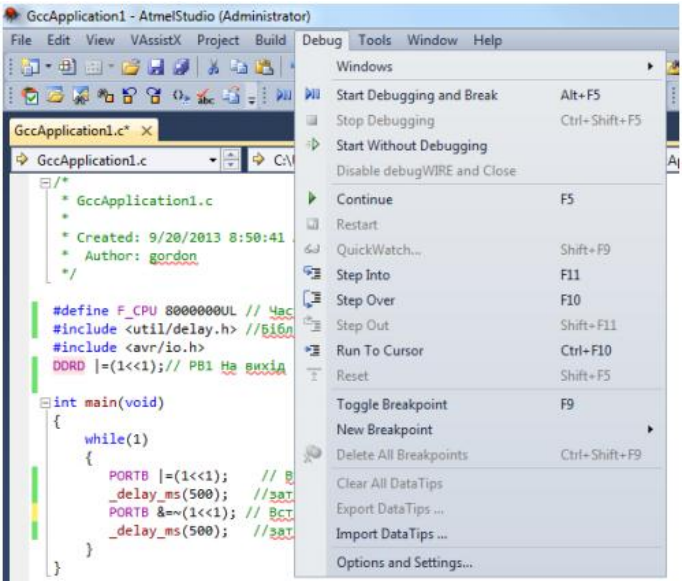


Рисунок 2.11. – Меню Debug

Під час першого запуску Atmel Studio необхідно додати вікно відображення периферії мікроконтролера ***I/OView*** (див. рис. 2.12).

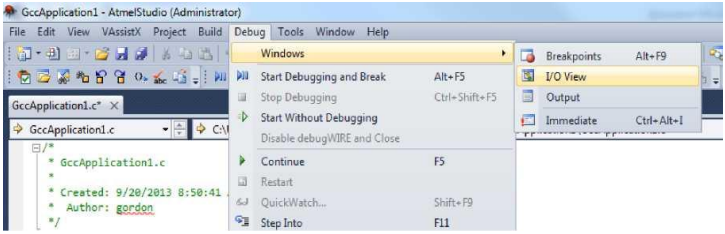


Рисунок 2.12. – Додавання вікна I/O View

Після виконання цієї дії з’явиться вікно I/O View (див. рис. 2.13).

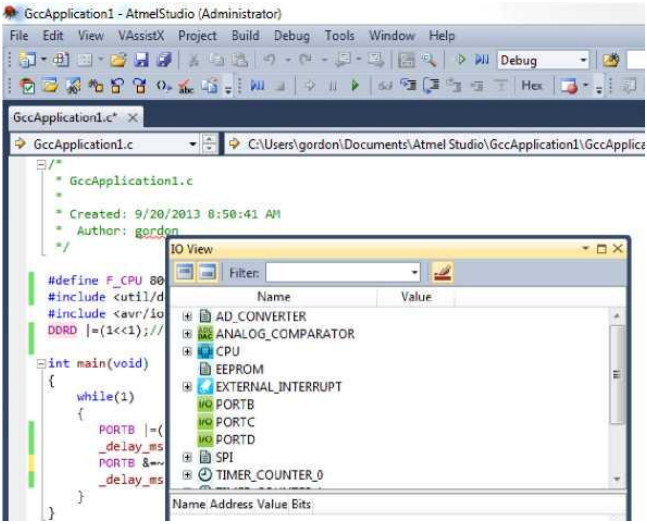


Рисунок 2.13. – Вікно I/O View

Для того, щоб перенести вікно в інше місце, необхідно навести курсор миші на верхню частину панелі та натиснути ліву кнопку миші і перемістити мишу в будь-якому напрямку до моменту появи панелі вибору розташування (див. рис. 14).

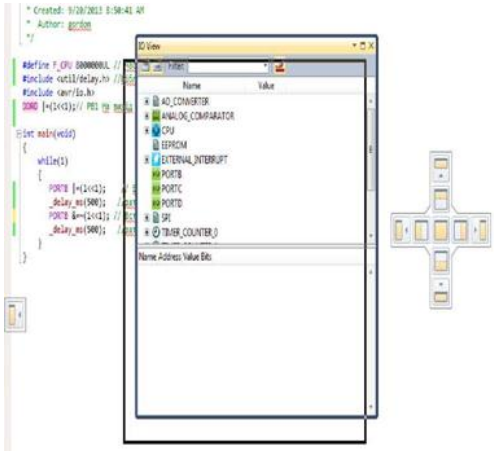


Рисунок 2.14. – Переміщення панелі

Перемісити панель в одне із зручних місць (див. рис. 2.15).

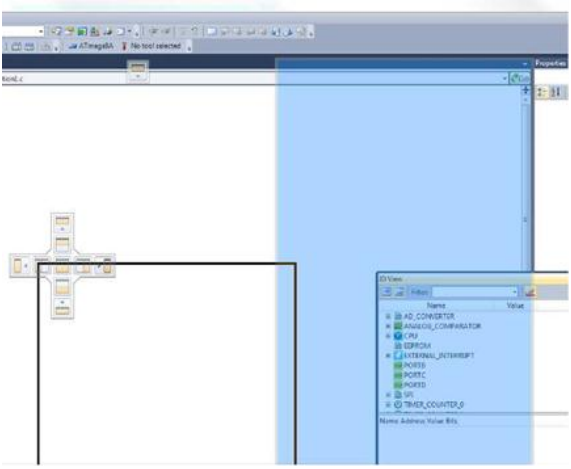


Рисунок 2.15. – Переміщення панелі

Більш докладно про переміщення панелей можна дізнатися на http://msdn.microsoft.com/ru-ru/librarv/windows/apps/ii155138.aspx.

Для запуску покрокового відлагодження програми необхідно перейти у вкадку ***Debug ►Step Into*** або натиснути ***F11.*** Після запуску симуля-

тора з'являється стрілка, яка інформує, котрий рядок на цей момент буде виконуватися (див. рис. 16).

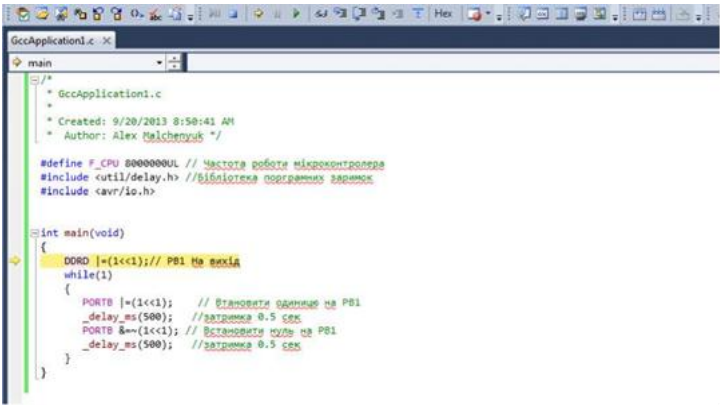


Рисунок 2.16. – Відлагодження коду

Якщо код, котрий виконується, змінює певні регістри периферії, то ці зміни можна спостерігати у вікні I/O View. Код прикладу змінює біти порту B. Тому для їх відображення, необхідно натиснути (після запуску відладки) у вікні I/O View на кнопку PORTB. Після цього в нижній частині вікна I/O View відображатимуться регістри порту вводу-виводу (див. рис. 2.17).

Кожен порт МК сімейства Mega має три регістри керування:

1. DDRx – відповідає за напрямок роботи кожного виводу (дроту) порту вводу-виводу котрий налаштований як вхід або вихід.
2. PINx – поточний стан кожного вивода порту, з котрого можна зчитати поточний логічний рівень на виводі порту вводу-виводу (нуль або одиниця).
3. PORTx – якщо вивід порту вводу-виводу зконфігурований на вихід запис нуля або одиниці призведе до зміни рівня на фізичному виводі порту. А якщо порт зконфігуровано на вхід, то запис одиниці призводить до підтягування виводу порту вводу-виводу до живлення.

Більш детально порти вводу-виводу будуть розглянуті в наступній лабораторній роботі.

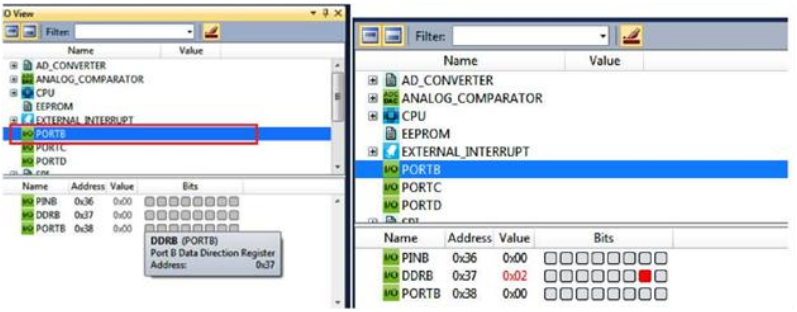


Рисунок 2.17. – Вікно I/O View. Зміна бітів порту вводу-виводу

Для того, щоб зупинити роботу програмного стимулятора необхідно перейти в ***Debug ►Sop Debugging*** або натиснути ***Ctrl+Shift+F5.*** Під час відлагодження ПЗ мікроконтролерів часто виникає необхідність зупинити виконання програми в певний момент часу (наприклад, входження в обробник переривання). Для таких цілей в меню відладки Atmel Studio призначений інструмент ***Breakpoint.*** Якщо стимулятор доходить до точки зупинки програми, то виконання програми тимчасово припиняється, що дає можливість програмісту уважніше розглянути результати роботи програми. Зазначений інструмент часто використовують для перевірки коректності роботи програми на певних контрольних точках, наприклад коректність реакції на певну умову.

Для того щоб встановити точку зупинки (breakpoint) в певному рядку програми необхідно натиснути мишею у будь-якому місці цього рядка та клавішу ***F9.*** Для вимкнення точки зупинки необхідно повторити вищенаведені дії. Розставляти точки зупинки в коді програми можна незалежно від того запущений налагоджував чи ні (див. рис. 2.18).

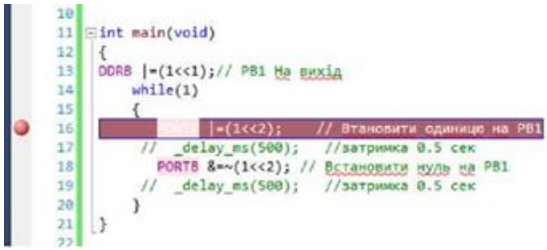


Рисунок 2.18. – Встановлення точки зупинки (breakpoint)

Після встановлення breakpoint запустити виконання програми за допомогою меню Debug ► Continue або натиснути F5. Як тільки симулятор дійде до точки зупинки, виконання програми призупиниться (див. рис. 19).

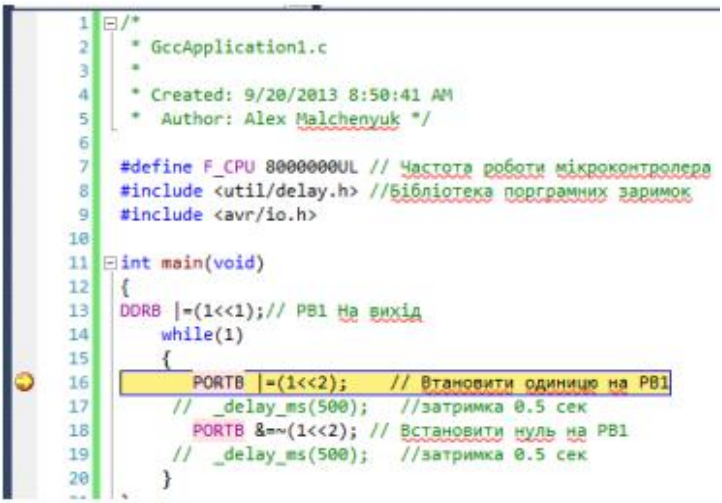


Рисунок 2.19. – Зупинка програми в breakpoint

Продовжити виконання програми можна за допомогою клавіш F5 (запуск виконання програми в безперервному режимі) або за допомогою клавіш F10, F11 (запуск у покроковому режимі).

2.5. Лабораторна робота № 1

Лабораторну роботу виконати у наступному порядку:

1. Створити новий проєкт в Atmel Studio. У якості ім’я проєкту вказати своє прізвище, ім’я (ім’я та прізвище вводити латиницею) та номер групи.

У якості папки для збереження використати мережевий диск N.

2. Обрати в якості мікроконтролера – ATmega8.

3. Набрати наведений нижче код у редакторі.

Код програми:

```
/*
* LR1.c
*
* Created: 30.05.2021 21:36:50
* Author: Nickolay
*/

#define F_CPU 8000000UL // Частота роботи мікроконтролера
#include <util/delay.h> // Бібліотека програмних затримок
#include <avr/io.h>
#define No_Delay 0
int main(void)
{
      DDRB |=(1<<1);// PB1 На виведення
      while(1)
      {
            PORTB |=(1<<1); // Встановити одиницю на PB1
            #if No_Delay
            _delay_ms(500);
            #endif
            PORTB &=~(1<<1); // Встановити нуль на PB1
            #if No_Delay
            _delay_ms(500);
            #endif
      }
}
```

4. Виконати компіляцію коду. Перевірити його роботу в симуляторі Atmel Studio. Встановити в коді декілька точок зупинки (їх розташування обрати на свій розсуд). Перевірити стан порту B в обраних точках зупинки. Результати внести до звіту.

5. Перекомпілювати код, замінивши рядок в коді **#define No_Delay 0** на рядок **#define No_Delay 1.** Перевірити роботу коду в середовищі Proteus. Для цього зібрати нижченаведену електричну принципову схему (див. рис. 2.20) та налаштувати модель (подвійним натисканням на елементі моделі Atmega 8 визвати панель edit component, обрати скомпільований hex-файл та встановити тактову частоту: в графі CKSEL fuses обрати Int. RC 8Mhz).

6. Далі треба зкоригувати схему програмний код. Кожен студент використовує номер PIN для вмикання-вимикання світлодіода, котрий вираховується за наступною формулою: <номер варіанта за журналом> % 8. Наприклад, для студента з номером 14 порядковий номер світлодіода:

$$14 \% 8 = 6.$$

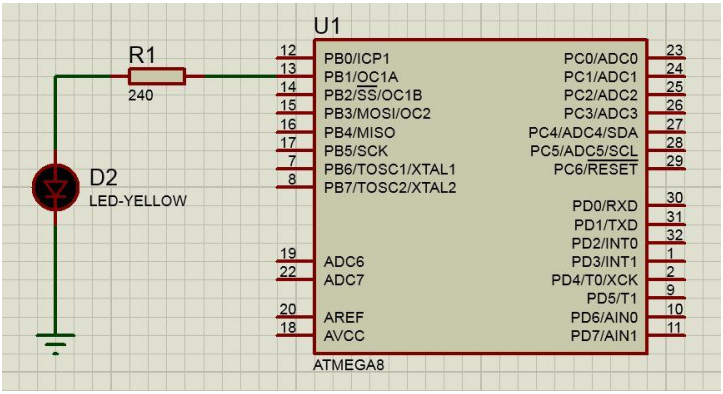


Рисунок 2.20. – Схема складання для виконання лабораторної роботи

Розділ 3.

Програмування вузлів засобів автоматизації

3.1. Основні конструкції мови програмування на C++

C (англ. C) – універсальна, процедурна, імперативна мова програмування загального призначення, розроблена у 1972 році Денісом Рітчі у Bell Telephone Laboratories з метою написання нею операційної системи UNIX.

Хоча **C** і було розроблено для написання системного програмного забезпечення, наразі вона досить часто використовується для написання прикладного програмного забезпечення.

C, ймовірно, є найпопулярнішою у світі мовою програмування за кількістю вже написаного нею програмного забезпечення, доступного під вільними ліцензіями коду та кількості програмістів, котрі її знають. Версії компіляторів для мови **C** існують для багатьох операційних систем та апаратних архітектур. **C** здійснила великий вплив на інші мови програмування, особливо на **C++**, яка спочатку проєктувалася, як розширення для **C**, а також на **Java** та **C#**, які запозичили у **C** синтаксис.

C – мінімалістична мова програмування. Серед її головних цілей: можливість прямолінійної реалізації компіляції, використовуючи відносно простий компілятор, забезпечити низькорівневий доступ до оперативної пам'яті, формувати лише декілька інструкцій машинної мови для кожного елементу мови, і не вимагати обширної динамічної підтримки. У результаті, код **C** придатний для більшості системного програмного забезпечення, яке традиційно писалося асемблером.

Незважаючи на її низькорівневі можливості, мова проєктувалася для машинно-незалежного програмування. Сумісна зі стандартами та машиннонезалежно написана мовою C програма, може легко

компілюватися на великій кількості апаратних платформ та операційних систем з мінімальними змінами. Мова стала доступною для великої кількості платформ, від вбудованих мікроконтролерів до суперкомп'ютерів.

Більшість мов програмування є абстрактними. Тому мову С абсолютно не хвилює, яка інформація буде оброблюватися.

Основне поняття в будь-якій мові програмуваня – це змінна. Без розуміння цього поняття неможливо вивчити будь-яку мову програмування. Змінна – це певний «контейнер» призначений для збереження певного значення.

Цей «контейнер» характерезується зазвичай трьома параметрами:

- розміром;
- адресою;
- іменем.

Існують певні домовленості, щодо розміру «контейнера» для того щоб ефективно використовувати пам'ять МК та ввести певну стандартизацію розміру змінної (тобто контейнера). Зазвичай розмір типу даних вказується в бітах або байтах. Оскільки робота процесора і мікроконтролера є процесом цифрової обробки даних, тому так чи інакше все розмаїття відомостей зводиться до чисел. Навіть літери є не що інше, як числа, співставленні їм (це номер букви за порядком в алфавіті). З цих причин у мові С основним видом даних є числа, які для зручності розділені на ряд стандартних типів. Atmel Studio 6 за замовчуванням підтримує такі типи змінних:

Вид	Назва	Діапазон	Обсяг байтів
Символьний	unsigned char	0 .. 255	1
	char	–128 .. 127	1
Перелічувальний	enum	–32,768 .. 32,767	2
Цілі числа	unsigned int	0 .. 65,535	2
	short int	–32,768 .. 32,767	2
	int	–32,768 .. 32,767	2
	unsigned long	0 .. 4,294,967,295	4
	long	–2,147,483,648 .. 2,147,483,647	4
	long long або int64	–9 223 372 036 854 775 808 ... 9 223 372 036 854 775 807	8
	unsigned ___ int64	0 ... 18446744073709551615	8
Дійсні числа	float	3.4 * (10–38) .. 3.4 * (10+38	4
	double	1.7 * (10–308) .. 1.7 * (10+308)	8
	long double	3.4 * (10–4932) .. 1.1 * (10+4932)	10

Довідка про типи змінних (даних):

char – символ, тобто число, що займає один байт (8-бітове число);

int – ціле число, що займає 2 байти (16-бітове число);

long – довге ціле число, що займає 4 байти (32-бітове число);

long long – подвійне довге ціле число, що займає 8 байт (64-бітове число);

float – дійсне число, тобто число з плаваючою крапкою (використовується длязбереження чисел з дробовою частиною);

double – дійсне число подвоєної точності (використовується для збереження чисел з дробовою частиною).

За допомогою додаткових службових слів визначаються різновиди типів цілих чисел:

signed – означає, що ціле число має знак (тобто може приймати і негативні значення);

unsigned – означає, що ціле число не має знаку, тобто може бути тільки нулем або більше;

short – означає, що фактичний розмір цілого числа вдвічі менше.

Рекомендація: для уникнення проблем бажано чітко вказувати тип даних знаковий чи беззнаковий за допомогою службових слів ***signed та unsigned.***

Наступним параметром, котрим характеризується змінна – це її адреса. Позаяк С абстрактна мова, такий параметр зазвичай користувачу не потрібен.

Мікроконтролери AVR мають три типи пам'яті. Щоб компілятор знав, у якій з трьох областей пам'яті знаходиться змінна – йому необхідна адреса комірки.

Останній і найголовніший параметр – ім'я змінної. Цей параметр повністю залежить від користувача. Звичайно можна записувати інформацію в змінні за допомогою адрес, але тоді код програми дуже важко модифікувати та відлагожувати. Оскільки С абстрактна мова – то назва змінної повинна відображати призначення інформації, яка в ній зберігається.

Ідентифікатори записуються латинськими буквами, цифрами, знаком підкреслення. Розпочинаються ідентифікатори з латинських літер та знаку підкреслення.

Приклади:

A, a, max, Max, MAX, max, maxi, max_znach *– правильно записані ідентифікатори.* ***imax, max-znach, max znach, a..b*** – неправильно записані ідентифікатори.

Під час написання імені ідентифікатора враховується регістр. (MAX, Max, max – три різні ідентифікатори).

Крім вищенаведених прикладів неправильних ідентифікаторів не можна також використовувати у якості назв змінних ключових слів. Ключове або службове слово – це ідентифікатор, зарезервований для особливого використання. У С стандартно визначені наступні службові слова:

auto	rnt if	double	struct	break	else	long
swtich	register	tupedef	char	extern	return	void
case	float	unsigned	default	for	signed	urnon
do	sizeof	volatile	continue	enum	short	white

Змінна зазвичай об'являється за таким принципом:

<Тип даних><Назва змінної >

Приклад:

unsigned char count;

unsigned char count, temp, i, water;

int wariator=0.

Іноді необхідно реалізувати лічильник імпульсів. Максимальна кількість імпульсів, яка може бути отримана за період виміру 254. Щоб назва змінної відображала своє призначення – її логічно назвати ***counter*** (тобто лічильник). Оскільки максимальне число імпульсів – 254, то для раціонального використання пам'яті мікроконтролера необхідно обрати тип даних ***unsigned char.***

Unsigned char counter=0; // об'явлення змінної.

3.2. Регістри мікроконтролера

Регістри мікроконтролера (МК) подібні коміркам пам'яті зберігають певні числові значення. Вони поділяються на регістри загального призначення (РЗП) та спеціальні регістри конфігурування периферії мікроконтролера. Регістри загального призначення використовуються без обмежень для виконання арифметичних операцій. Оскільки у цьому посібнику програми пишуться мовою програмування **С**, то ці регістри нам цікаві лише з теоретичної сторони, адже робота з ними захована від користувача компілятором (на відміну від мови програмування ASM).

Більш цікавими для нас є регістри налаштування периферії. Ці регістри використовуються для роботи з периферією. Зазвичай вони мають розрядність відповідну розрядності МК, у цьому випадку 8 біт.

Для кращого розуміння, представлення регістр, варто уявляти його залежно від його призначення коміркою пам'яті або набором з восьми вимикачів (див. рис. 3.1), за допомогою котрих налаштовується периферія МК.

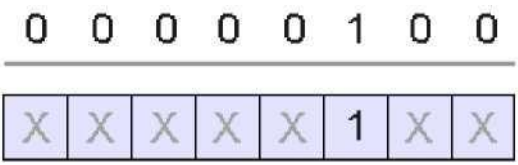


Рисунок 3.1. – Структура регістру

Регістри так само, як і змінні мають власні імена (ідентифікатори). Вони зазначені виробником МК і зазвичай являють собою скорочення від функціонального призначення цього регістру. Зручно асоціювати біти регістру з вимикачами, адже кожен біт регістру може приймати два логічні значення «нуль» та «один» тобто ввімкнено та вимкнено. Регістри периферії будуть розглянуті нижче.

На зразок до всіх комп'ютерних систем, мікроконтролери мають безліч регістрів, які використовуються для управління різними пристроями, підключеними до процесора. Це можуть бути регістри процесора (акумулятор, регістри стану, індексні регістри), управління (регістри управління переривань, управління таймером) або регістри, що забезпечують введення-виведення даних (регістри інформації і управління паралельним, послідовним або аналоговим вводом – виводом). Звернення до них може виконуватися різними способами.

Одним з важливих питань є розміщення регістрів в адресному просторі. У деяких процесорах всі регістри і RAM розташовуються в одному адресному просторі. Це означає, що пам'ять суміщена з регістрами. Такий підхід називається «відображенням пристроїв введення-виведення на пам'ять».

В інших процесорах адресний простір для пристроїв вводу-виводу відділений від загального простору пам'яті. Основна перевага розміщення регістрів введення-виведення в окремому просторі адресів полягає у спрощенні схеми підключення пам'яті програм і даних до загальної шини. Пристрої вводу-виводу зазвичай займають маленький

блок адрес, що робить незручним декодування їх адреси спільно з великими блоками основної пам'яті. Окремий простір вводу-виводу дає деяку перевагу процесорам з Гарвардською архітектурою, забезпечуючи можливість зчитування команди під час звернення до регістру введення-виведення.

Зважаючи на вищезазначене, ви, ймовірно, вважаєте, що Гарвардська архітектура з регістрами і змінними, розташованими в різних окремих адресних просторах, є найбільш ефективною. Проте є низка причин, через які використання мікроконтролерів Прінстонської архітектури з відображенням пристроїв введення-виведення на пам'ять може виявитися для деяких випадків кращим.

Приклади роботи з регістрами:

Необхідно записати в регістр даних UART-а число 255:

```
UDR =255 ;
UDR=0xff .
```

Необхідно встановити всі виводи порту B на вихід:

```
DDRB=0xff .
```

3.3. Бітові операції

Зазвичай для конфігурації периферії необхідно змінити певний біт у регістрі не чіпаючи інші. Для таких маніпуляцій необхідно використовувати бітові операції та маски.

	Установка біту	**Скидання біту**	**Інверсія байту**	**Інверсія окремих біт**
Початкове значення	10001000	10001000	10001000	10001000
Операція	**OR (|)**	**AND (&)**	**NOT (~)**	**XOR (^)**
Бітова маска	00010000	01111111	Не використовується	11000000
Результат	10011000	00001000	01110111	01001000

Нумерація бітів проводиться з правої сторони на ліву: 7 6 5 4 3 2 1 0.

Для встановлення біту (встановлення біту розуміється як зміна його значення на логічну одиницю) необхідно виконати операцію ***OR.*** У мові програмування С вона позначається | .

Приклади:

Встановити перший біт порту B:

```
PORTB =PORTB | 0b01000000;
PORTB |= 0b01000000;
PORTB |= (1<<6).
```

Символ **b** використовується для визначення числа у бітовій системі числення. Для обнуління певного біту необхідно скористатись операцією **AND.** У мові програмування **C** вона позначається **&.**

Приклади:

Обнулити п'ятий біт порту B:

```
PORTB = PORTB & 0b11011111;
PORTB &= 0b11011111;
PORTB &= ~ (1<<5).
```

Для того щоб змінити значення бітів у байті на протилежні (наприклад з «нуля» на «одиницю») необхідно використовувати операцію **NOT.** У мові C ця операція позначається **~** .

Приклад:

Інвертувати біти порту B.

```
PORTB= ~ PORTB.
```

Для інверсії окремих бітів у байті використовується операція **XOR.** Вона позначається за допомогою символу ^.

Приклади:

Виконати інверсію 4-го біту порту B:

```
PORTB =PORTB ^ 0b00010000;
PORTB ^= 0b00010000;
PORTB ^= (1<<4).
```

3.4. Бітовий зсув

Бітовий зсув – зміна позицій бітів у слові на одну і ту саму величину. Більшість комп'ютерів не можуть напряму адресувати біти, які містяться групами по 8, 16, 32 або 64 бітів у словах. Для забезпечення роботи з бітами існує безліч машинних інструкцій, що включають різні типи зрушень. Усі зрушення схожі одні на одних поведінкою середніх бітів, які просто зсуваються ліворуч або праворуч на певну величину. Однак, поведінка крайніх бітів, які йдуть з слова та які з'являються в слові, залежить від типу зсуву. В електроніці бітові зрушення здійснюються в регістрах зсуву.

Це дуже зручний інструмент для формування бітових масок які використовуються у вищенаведених операціях.

Наприклад, побітний зсув у ліву сторону зрушує число на **n** розрядів ліворуч. Старші **n** розрядів під час цього зникають, а молодші **n** розрядів заповнюються нулями. Операція зсуву ліворуч еквівалентна множенню на **2n**.

Приклади:

```
unsigned char tmp = 1;
tmp = tmp << 1;
// тепер в змінній tmp число 2 або 0b00000010
tmp = tmp << 3;
// тепер в змінній tmp число 16 або 0b00010000
tmp = 255;
tmp <= 2; //скорочений варіант запису
// тепер в змінній tmp число 252 або 0b11111100.
```

3.5. Регістри порту вводу-виводу

Через порти вводу-виводу МК спілкується із зовнішнім світом. У мікроконтролерах AVR ці порти двонаправлені, тобто будь-який вивід порту в будь-який момент часу може бути зконфігурований як на вхід так на вихід. Порти нумеруються латинськими літерами (A, B, C, D). Позаяк AVR Mega мікроконтролери є 8-розрядними то порти вводу-виводу теж є 8-розрядними. Схематичне спрощене зображення одного виводу порту зображено на рис. 3.2.

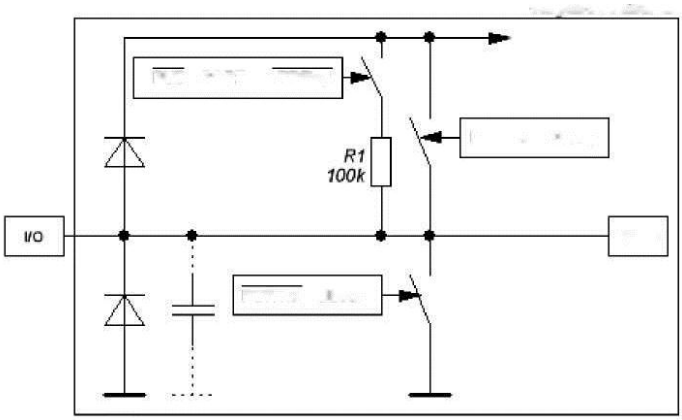


Рисунок 3.2. – Спрощене зображення одного порту вводу-виводу

Конфігураційні біти регістру керування зображені у якості вимикачів. Кожен порт вводу-виводу має три регістри для налаштування та зчитування поточного стану порту вводу-виводу.

Перший регістр **DDRx** (**x** – будь-який порт) використовується для встановлення напрямку роботи порту. Порт в конкретний момент часу може бути або входом, або виходом залежно від конфігурації.

– DDRxy=0 – вивід працює як ВХІД;
– DDRxy=1 – вивід працює як ВИХІД.

Примітка : x – назва порту (A, B, C, D), **y** – номер виводу порту.

Наступний регістр це **PORTx**. Коли ніжка порту налаштована на ВИХІД, то запис в біт регістру PORTx визначає стан виводу. Якщо PORTxy=1, то на фізичному виході встановлюється логічна одиниця. Якщо PORTxy=0, то логічний нуль. Якщо вихід налаштований як ВХІД (DDRxy=0), то встановлення PORTxy=1 в логічну одиницю призводить до підключення підтягувального резистора. Якщо PORTxy=0, то вихід встановлюється в третій стан (HI–Z).

Останній регістр який використовується для зчитування поточного логічного значення на фізичному виводі порту вводу-виводу **PINx**. Його можливо тільки зчитати, на відміну від вищезазначених.

Приклади ініціалізації порту вводу-виводу:

1. Ініціалізувати порт **B** на вихід. Встановити всі виводи порту в стан логічної одиниці.

```
DDRB=0xff; // Всі виводи порту B на вихід;
PORTB=0xff; // Встановити логічну одиницю на всіх виводах.
```

2. Ініціалізувати 2, 3, 7 виводи на вихід. Встановити ці виводи в стан логічної одиниці.

DDRB |=(1<<2) | (1<<3) | (1<<7); // Всі виводи 2,3,7 порту В на вихід;

PORTB |=(1<<2) | (1<<3) | (1<<7); // Встановити логічну одиницю на 2,3,7 виводах.

3. Встановити виводи 3, 4, 5 у логічний нуль.

PORTB &= ~((1<<3) | (1<<4) | (1<<5)); // Встановити логічний нуль на 3,4,5 виводах.

3.6. Лабораторна робота № 2

Завдання:

1. Створити новий проєкт в Atmel Studio. Світлодіоди встановити на всі «піни» МК ATmega8.

2. Створити світловий ефект згідно з варіантом завдання користуючись нижченаведеною заготовкою. Після кожного світлового ефекту встановлювати програмну затримку _delay_ms(200).

3. Кожен студент використовує послідовність цифр для вмикання-вимикання світлодіодів, котра складається із цифр наступних чисел: номер варіанта за журналом, день, місяць та рік народження самого студента. Якщо трапляються цифри 8 і 9, то їх ігнорувати. Якщо цифри повторюються, то використовувати один раз.

4. Перевірити коректність роботи на моделі в середовищі Proteus.

Заготовка програми:

```
#include <avr/io.h>
#define F_CPU 8000000UL
#include <util/delay.h>
int main(void)
   {
      DDRB =0xff; // Усі розряди порту налаштовані на виведення
      while(1)
      {
            PORTB=0x00;
            _delay_ms(400);
            // Вводити власний код сюди!
            // Кінець власного коду
      }
   }
```

Приклад виконання роботи:

Реалізувати нижченаведений світловий ефект (вмикання в певній послідовності світлодіодів). Не забуваємо ідентифікувати власність програми своїм ім'ям.

Реалізувати таку послідовність вмиканням: 0, 2, 3, 1, 5.

```
/*
* LR2.c
*
* Created: 30.05.2021 21:36:50
* Author: Nickolay
*/

#include <avr/io.h>
#define F_CPU 8000000UL
#include <util/delay.h>
int main(void)
{
	DDRB =0xff;
	while(1)
	{
		PORTB=0x00;
		_delay_ms(400);
		// Вводити власний код сюди!
		PORTB |=(1<<0);
		_delay_ms(200);
		PORTB |=(1<<2);
		_delay_ms(200);
		PORTB |=(1<<3);
		_delay_ms(200);
		PORTB |=(1<<1);
		_delay_ms(200);
		PORTB |=(1<<5);
		_delay_ms(200);
		// Кінець власного коду
	}
}
```

3.7. Робота з бітовими масками та портами вводу-виводу AVR-мікроконтролерів. Динамічна індикація

3.7.1. Бітові маски

Для створення бітових масок можна скористатись операціями зрушення праворуч-ліворуч, а також логічними операціями. Якщо нам необхідно одночасно обнулити або встановити окремі біти, то досить зручно створити бітову маску.

Приклад:

Необхідно встановити 0, 1, 4, 5 біти в одиницю. Для встановлення конкретного біту в одиницю скористаємося простою формулою:

$$Rez = \mathbf{2^n},$$

де **n** – номер біту який потрібно встановити.

Для нашого прикладу бітова маска буде дорівнювати,

$$Rez = 2^\circ + 2^1 + 2^4 + 2^5 = 51.$$

3.7.2. Динамічна індикація

Динамічна індикація – це метод відображення цілісного зображення шляхом швидкого послідовного відображення його частин. Видимість цілісності у цьому разі досягається завдяки інерційності людського зору. Як правило, зображення на електронно-променевій трубці формується саме таким чином (покроково), та й на сучасних ЖК-матрицях використовується такий самий метод. Сутність методу полягає в послідовному засвічуванні рядів точок-пікселів на екрані. Адже, якби всі пікселі керувалися одночасно, це вимагало б (для екрана з роздільною здатністю 800x600 точок) 800x600=480000 контактів для підєднання матриці і стільки ж ліній портів керуючого контролера. І це для монохромного дисплея, для кольорового – в 3 рази більше. Ми ж розглянемо дещо простіший варіант ніж матричний вивід – вивід на стандартні семисегментні світлодіодні індикатори.

Звичайно, якщо нам потрібно виводити число в 1–2 розряди, то можна просто приєднати індикатори до портів МК посегментно. Проте, якщо розрядів, скажімо 6 (годинник наприклад), то необхідно використати 6×7=42 виводи портів. Таку кількість мають далеко не всі МК серії AVR. Тут і приходить на допомогу динамічна індикація. Їх сутність у тому, що ми з'єднуємо, припустимо усі одноіменні аноди розрядів (для індикаторів зі спільним катодом), і заводимо на виводи

одного порта, а катоди кожного з розрядів – на інший порт (якщо індикатори великого розміру і споживають великий струм, доведеться використати додаткові ключі на транзисторах). Таким чином, використовується всього 8+6=14 виводів портів МК. Ще більше зменшити число задіяних виводів можна за допомогою спеціальних мікросхем-дешитфраторів, так часто роблять у разі великої кількості розрядів. Індикація відбувається шляхом швидкого циклічного показу кожного розряду числа з відповідно включеним на той момент катодом.

Тобто, припустимо, що нам треба вивести число **12**. Алгоритм виглядає наступним чином:

1. Подаємо сигнал лог.0 (підключаємо до земляного проводу) на перший катод (на інших лог. 1).

1.2. Подаємо на аноди комбінацію, що відповідає числу **1.** На першому індикаторі засвітиться **1**, решта будуть погашені.

1.3. Утримуємо вказаний стан протягом певного часу, наприклад 0,01 с.

2. Подаємо сигнал лог.0 (підключаємо до земляного проводу) на другий катод (на інших лог. 1).

2.2. Подаємо на аноди комбінацію, що відповідає числу **2.** На другому індикаторі засвітиться **2**, решта будуть погашені.

2.3. Утримуємо вказаний стан на протязі певного часу, наприклад 0,01 с.

3. Переходимо до п. 1.

Як ми бачимо, насправді індикатори висвічують числа почергово. Але, оскільки час переключення менший, ніж час інерції людського зору (біля 0,04 с), то ми побачимо, що усі індикатори показують одночасно. Навіть, якщо зробити більші затримки між переключеннями, то будемо бачити суцільне зображення, але може бути помітне неприємне мерехтіння розрядів.

3.7.3. Апаратне забезпечення експерименту

Для експерименту зберемо схему на основі трьохрозрядного світлодіодного індикатора.

На малюнку показана схема одного розряду індикатора, позначення сегментів буквами **a-h** у всіх стандартизоване. До речі, насправді сегментів у більшості індикаторів 8, але восьмий сегмент (К) – це крапка і вона не приймає участь у формуванні цифри.

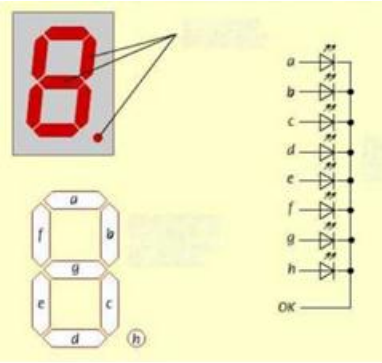


Рисунок 3.4. – Світлодіодний семисегментний індикатор

3.7.4. Кодування цифр для семисегментного індикатора

Тепер розглянемо принцип формування цифр семисегментним індикатором. Схематичне зображення цифр утворюється відповідною комбінацією включених сегментів. Отже, кожній цифрі від 0 до 9 відповідатиме певна комбінація лог.0 і 1 на виводах порта А, а отже, якесь число, записане у цей порт. Назвемо це число кодуванням (див. табл. 3.1).

Таблиця 3.1

Кодування цифр на семисегментному індикаторі

ЦИФРА	0	1	2	3	4	5	6	7	8	9
ЗОБРАЖЕННЯ НА ІНДИКАТОРІ	0	1	2	3	4	5	6	7	8	9
Сег. A(PA0)	1	0	1	1	0	1	1	1	1	1
Сег. B(PA1)	1	1	1	1	1	0	0	1	1	1
Сег. C(PA2)	1	1	0	1	1	1	1	1	1	1
Сег. D(PA3)	1	0	1	1	0	1	1	0	1	1
Сег. E(PA4)	1	0	1	0	0	0	1	0	1	0
Сег. F(PA5)	1	0	0	0	1	1	1	0	1	1
Сег. G(PA6)	0	0	1	1	1	1	1	0	1	1

3.8. Лабораторна робота № 3

Завдання для виконання роботи:

Вивести на семисегментний індикатор номер за загальним списком в журналі. Якщо номер варіанта менше 10 то виводити у форматі 0x, де x – номер варіанта.

Для виконання цього завдання необхідно зібрати модель в середовищі Proteus у відповідності з завданням. Якщо в завданні вказано, що потрібно ввімкнути точку, то ця операція має бути виконана за допомогою встановлення відповідного біту в числі. У якості індикатора використати 7SEG- MPX2-CC.

Таблиця 3.2

Варіанти завдань

Вар. №	Порт анодів індикатора	Порт катодів індикатора	Номери виводів катодів		Точка ввімкнена
			Катод 1	Катод 2	
1.	PORTB	PORTC	0	1	Так, 1 інд.
2.	PORTC	PORTB	1	2	Так, 2 інд.
3.	PORTB	PORTC	4	5	Ні
4.	PORTC	PORTA	7	0	Так, 1 інд.
5.	PORTB	PORTC	2	3	Так, 2 інд.
6.	PORTC	PORTA	1	0	Так, 1 інд.
7.	PORTA	PORTB	5	4	Ні
8.	PORTC	PORTB	0	2	Так, 1 інд.
9.	PORTA	PORTB	5	6	Ні
10.	PORTB	PORTA	4	3	Так, 2 інд.
11.	PORTC	PORTA	1	2	Ні
12.	PORTC	PORTB	7	1	Так, 1 інд.
13.	PORTA	PORTC	1	6	Ні
14.	PORTA	PORTB	3	5	Ні
15.	PORTB	PORTA	5	4	Так, 1 інд.

Примітка: модель мікроконтролера налаштувати так само як і в попередніх роботах.

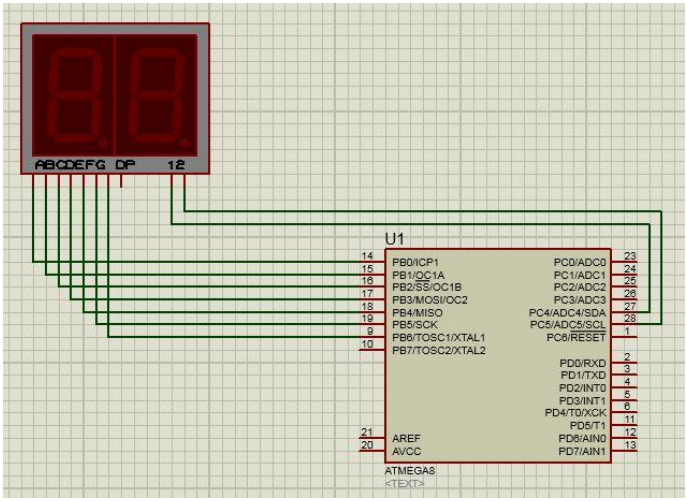


Рисунок 3.5. – Схема моделі

Код програми:

```
/*
* LR3.c
*
* Created: 30.05.2021 21:36:50
* Author: Nickolay
*/
#include <avr/io.h>
#define F_CU 8000000UL
#include <util/delay.h>

#define INDICATOR_1 4
#define INDICATOR_2 5
#define CATOD_PORT PORTC
#define CATOD_DDR DDRC
#define PIN_PORT PORTB
#define PIN_DDR DDRB

int main(void)
```

```
{
        PIN_DDR|=0xff;
        CATOD_DDR|=((1<<INDICATOR_1)|(1<<INDICATOR_2));
        CATOD_PORT|=((1<INDICATOR_1)|(1<<INDICATOR_2));
        while(1)
        {
              CATOD_PORT &=~(1<<INDICATOR_1);

              PIN_PORT=0b0111111;
              _delay_ms(50);
              CATOD_PORT|=(1<<INDICATOR_1);
              CATOD_PORT &=~(1<<INDICATOR_2);
              PIN_PORT=0b1001111;
              _delay_ms(50);
              CATOD_PORT|=(1<<INDICATOR_2);
         }
}
```

3.9. Робота з портами вводу-виводу в режимі вводу даних

3.9.1. Кнопки

У попередніх лабораторних роботах було розглянуто способи виведення інформації з мікроконтролера: підключення світлодіодів та семисегментного індикатора. Але як, же вводити інформацію у мікроконтролер? Існує безліч варіантів і пристроїв для цього. Але поки що розглянемо найпростіший варіант – це звичайна кнопка. Кнопки бувають двох видів: тактові і фіксуючі. Тактові, працюють за таким принципом: натиснув – контакти замкнулися, відпустив – контакти розімкнулися. Варто також зауважити, що існують тактові кнопки, котрі спочатку замкнуті, а з натисканням розмикаються. Фіксуючі, (їх іноді називають: тумблери, вимикачі, перемикачі) на відміну від тактових фіксують своє положення під час натискання, тобто: натиснув – контакти замкнулися, ще раз натиснув – контакти розімкнулися.

Найпростіший спосіб підключення кнопки до МК наведений на рис. 3.6.

Рисунок 3.6. – Схема підключення кнопки

У цій схемі кнопка підключена одним виводом до мінуса джерела живлення (на «сленговій мові» підключена на корпус). Можливо, ви запитаєте: навіщо резистор R1? Він необхідний для «підтяжки» порту вводу-виводу до живлення. Це необхідно для уникнення помилкових спрацювань кнопки.

У цій схемі підключення під час замикання кнопки на порті вводу-виводу встановлюється логічний «нуль», а розмиканні логічна «одиниця». Опір резистора R1 може бути в інтервалі від 4,7 кОм до 10кОм.

Приклад:

Зконфігурувати 1,4,5,7 виводи порту **B** для підключення кнопок.

```
DDRB &=~((1<<1)|(1<<4)|(1<<5)|(1<<7)); // Input.
```

3.9.2. Брязкіт контактів

Брязкіт контактів (англ. *Contact bounce*) – небажане замикання й розмикання контактів у момент комутації, що виникає в електричних і електронних перемикачах і котре не передбачене заданою дією пристрою (процес триває приблизно від декількох до десятків мілісекунд).

З цим явищем можна боротися двома способами: апаратним та програмним.

Перший нас не цікавить, адже веде до додаткового ускладнення схеми та збільшення її кінцевої вартості.

Зазвичай для опитування кнопок використовується багаторазове опитування стану кнопки упродовж певного часу, або затримки виконання програми. Зазвичай час брязкоту контактів не перевищує 50 мілісекунд. Тому час затримки зазвичай обирають рівним 50 мілісекундам.

Приклад:

Необхідно під час натиснення кнопки, котра підключена до PB1 увімкнути світлодіод, що підключений до PB2.

Програмний код з антибрязкітовою затримкою:

```
/*
* LR4.c
*
* Created: 30.05.2021 21:36:50
* Author: Nickolay
*/

#define F_CPU 8000000UL // Частота роботи мікроконтролера
#include <util/delay.h> // Бібліотека програмних затримок
#include <avr/io.h>

int main(void)
{
	DDRB &=~(1<<1); // Input botton
	DDRB |=(1<<2); // LED output
	while (1) // Безкінечний цикл
	{
		if ((PINB & (1<<1))==0)	// Якщо на PB1 лог.0, тоді
		{
			_delay_ms(50);	// Антибрязкітова затримка
			if ((PINB & (1<<1))==0)
			// Якщо стан кнопки не змінився ввімкнути світлодіод
			{
				PORTB |= (1<<2); // Ввімкнути світлодіод
			}
			else // Якщо ні, то вимкнути
			{
				PORTB &=~(1<<2); // Вимкнути світлодіод
			}
		}
	}
}
```

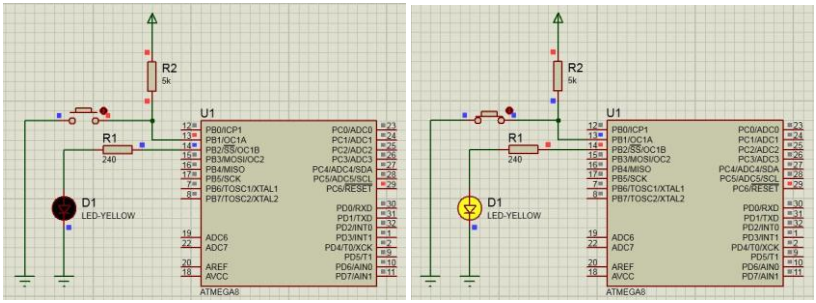


Рисунок 3.7. – Моделювання вищенаведеного коду
в середовищі Proteus

Розглянемо вищенаведений код більш детально:

DDRB &=~(1<<1); // Input botton;

DDRB |=(1<<2); // LED output.

Ця частина відповідає за ініціалізацію виводів порту вводу-виводу, до котрих підключено кнопку та світлодіод.

if ((PINB & (1<<1))==0) <набір операторів програми> // Якщо на PB1 лог.0, тоді виконується <набір операторів програми>

Така конструкція перевіряє логічний рівень на виводі, до якого підключена кнопка. Оператор if використовується для перевірки виконання умови, котра вказана в дужках:

(PINB & (1<<1))==0.

Конструкція використовується для перевірки значення, котре встановлено на PB1.

_delay_ms(50); // Антибрязкітова затримка.

Така затримка використовується для усунення брязкіту контактів.

У програмах, котрі будуть виконуватися в середовищі Proteus, така затримка не обов’язкова.

3.9.3. Опитування декількох кнопок

Для вивчення алгоритму опитування декількох кнопок скористаємося моделлю наведеною на рис. 3.8. У ній підключено три кнопки до PB0-PB2 та три світлодіоди до PB3-PB5.

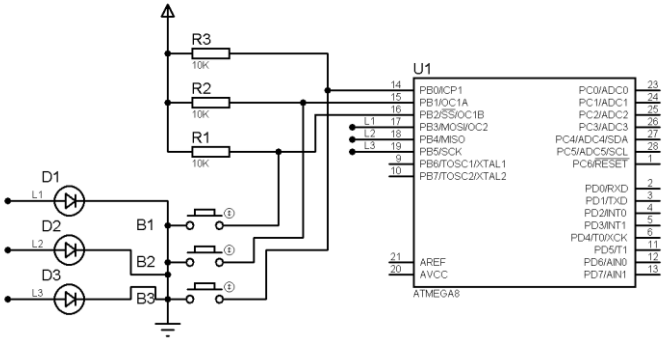


Рисунок 3.8. – Модель для середовища Proteus

Код програми:

```
/*
* LR5.c
*
* Created: 30.05.2021 21:36:50
* Author: Nickolay
*/

#define F_CPU 8000000UL // Частота роботи мікроконтролера
#include <util/delay.h> // Бібліотека програмних затримок
#include <avr/io.h>
int main(void)
{
	DDRB &=~((1<<0)|(1<<1)|(1<<2)); // Input botton
	DDRB |=((1<<3)|(1<<4)|(1<<5)); // LED output
	unsigned char button =0; //змінна для збереження номерів кнопок
	while (1) // Безкінечний цикл
	{
		//************ Початок опитування ***************
		if ((PINB &(1<<0))==0) //B1 on
		{
			button |=(1<<0);
			// Встановити біт відповідаючий за першу кнопку
		}
		if ((PINB &(1<<1))==0) //B2 on
		{
			button |=(1<<1);
			// Встановити біт відповідаючий за другу кнопку
		}
		if ((PINB &(1<<2))==0) //B3 on
```

```
            {
                    button |=(1<<2);
                    // Встановити біт відповідаючий за другу кнопку
            }
            _delay_ms(50); //Затримка
            //*********** Кінець опитування ***************

        if (((PINB &(1<<0))==0)&&((button & (1<<0))!=0))
        //Якщо кнопка 1 натиснута
            {
                    PORTB |=(1<<3);// D1 on
            }
            else
            {
                    PORTB &=~(1<<3);// D1 off
            }
            if (((PINB &(1<<1))==0)&&((button & (1<<1))!=0))
            //Якщо кнопка 2 натиснута
            {
                    PORTB |=(1<<4);// D2 on
            }
            else
            {
                    PORTB &=~(1<<4);// D2 off
            }
            if (((PINB &(1<<2))==0)&&((button & (1<<2))!=0))
            //Якщо кнопка 3 натиснута
                 {
                    PORTB |=(1<<5);// D3 on
            }
            else
            {
                    PORTB &=~(1<<5);// D3 off
                }
    }
}
```

3.10. Лабораторна робота № 4

Завдання:

Зібрати модель наведену на рис. 3.9. Написати ПЗ для мікроконтролера, яке буде реалізовувати наступний алгоритм:

Під час натиснення на кнопку B1 на світлодіодах D0-D4 буде відображатися номер варіанта студента в загальному списку, закодований у двійковому вигляді. З натисненням на кнопку B2 загорятимуться всі світлодіоди. Модель налаштовуватиметься так, як в попередніх роботах.

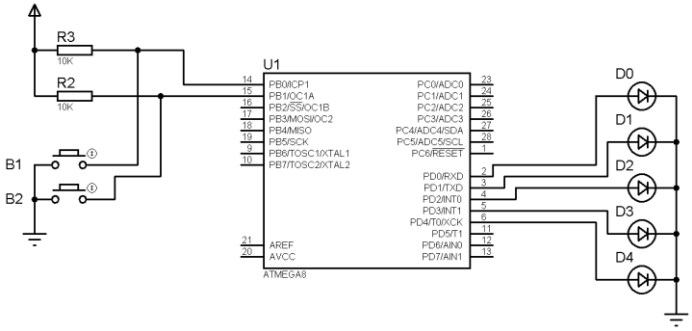


Рисунок 3.9. – Модель для середовища Proteus

3.11. Інтерфейс USART мікроконтролерів AVR

Налаштування моделі. Для виконання зазначеної лабораторної роботи необхідно зібрати модель в середовищі Proteus наведену на рис. 3.10.

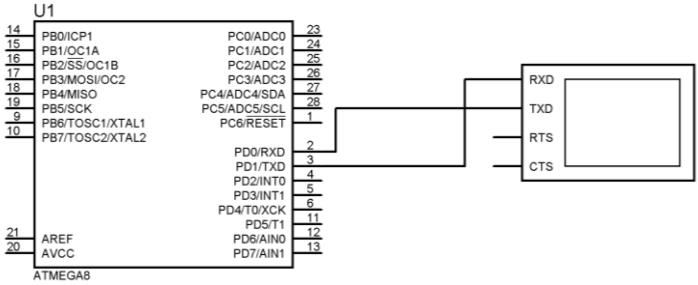


Рисунок 3.10. – Модель з використанням USART

У цій моделі використаний інструмент середовища Proteus Virtual Terminal. Для його додавання на робочий лист необхідно скористатися іконкою virtual instruments та додати інструмент Virtual terminal.

Також для коректної роботи модуля USART необхідно налаштувати модель МК ATmega8 відповідно до рис. 3.11.

Варто помітити що у якості джерела тактового сигналу використовується кварцовий резонатор частотою 7.3728 МГц. Тому необхідно налаштувати МК за допомогою ф'юзів CKSEL на роботу з зовнішнім кварцовим резонатором.

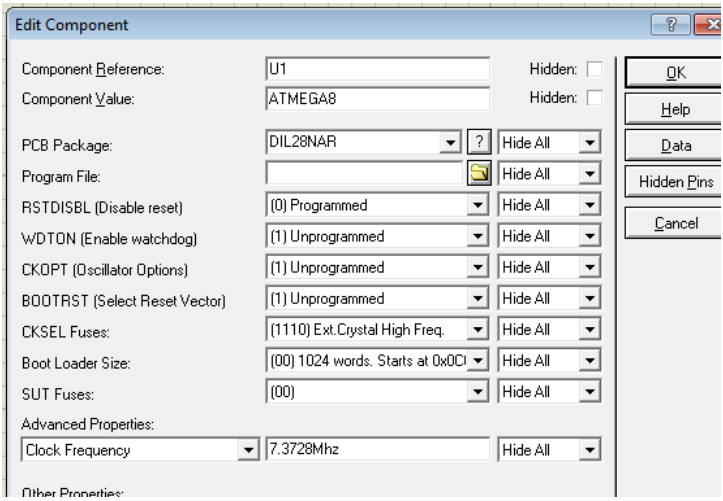


Рисунок 3.11. – Налаштування моделі МК

Наступним кроком є налаштування бітрейту інструменту Virtual terminal у відповідності з власним завданням (рис. 3.12).

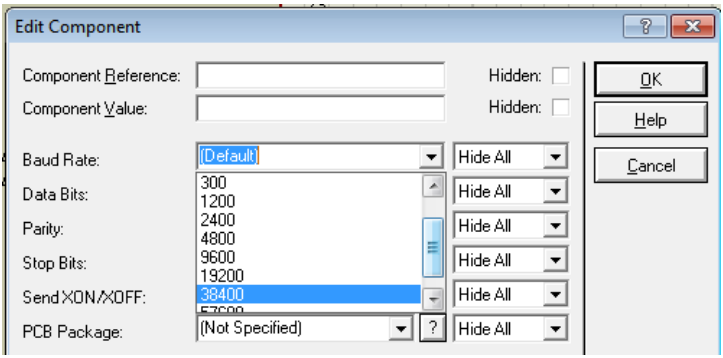


Рисунок 3.12. – Налаштування бітрейту

3.12. Лабораторна робота № 5

Завдання:

Вивести за допомогою інтерфейсу USART повідомлення такого змісту: номер варіанта, прізвище та ім'я. Налаштувати бітрейт модуля USART та Virtual terminal відповідно до табл. 3.3.

Таблиця 3.3

Варіанти завдань

№ Вар.	Baud Rate
1.	9600
2.	38400
3.	19200
4.	2400
5.	4800
6.	9600
7.	19200
8.	38400
9.	57600

Таблиця 3.4

Значення регістру UBRR для відповідної частоти

7.3728 Mhz		
Baud	**UBRR**	**% of error**
300	1535	0.0
600	767	0.0
1200	383	0.0
2400	191	0.0
4800	95	0.0
9600	47	0.0
14400	31	0.0
19200	23	0.0
28800	15	0.0
38400	11	0.0
57600	7	0.0
76800	5	0.0
115200	3	0.0

Приклад:

Для налаштування USART можна скористатись таким кодом:

```
#include <avr/io.h>
#include <avr/interrupt.h>
#define USART_BAUDRATE 23 // UBBR з таблиці 2
void uart_init(void)
{
	// Set baud rate
	UBRRL =USART_BAUDRATE;
	UCSRB |=(1<<TXEN); //Transmit on
	UCSRC = (1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0);
}
int main(void)
{
	uart_init(); //init uart
	uart_transmit_message("Group 571 \n\r");
	uart_transmit_message("Laboratory work N5 \r\n");
	uart_transmit_message("Semen Stepanenko \r\n");
}
// Для передачі рядка можна скористатись наступною функцією:
void uart_transmit_message(char* msg)
{ unsigned char i;
	i=0;
	// Цикл перебору елементів рядка
	while ((i<256)&(msg[i]!=0x00) )
	{
		// Відправлення поелементно символів рядка
		while ( !( UCSRA & (1<<UDRE))); // Перевірка
		UDR = msg[i]; // Запис символу в регістр передавача
		i++; // Збільшуємо номер елемента рядка
	}
}
```

Така функція приймає в якості параметра рядок символів та передає його за допомогою USART. Рядок повинен закінчуватися символом **\r\n,** інакше функція зациклиться.

3.13. Використання переривань

Налаштування моделі

Для виконання цієї лабораторної роботи необхідно зібрати модель у середовищі Proteus наведену на рис. 3.13.

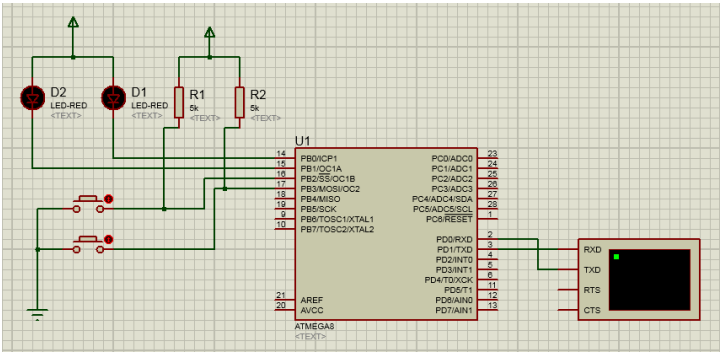


Рисунок 3.13. – Модель схеми для індикації датчиків та увімкнення сигнальних світлодіодів

Ця модель створена інструментами середовища Proteus Virtual Terminal. Для його додавання на робочий лист необхідно скористатися іконкою virtual instruments та додати інструмент Virtual terminal.

Також для коректної роботи модуля USART необхідно налаштувати модель МК ATmega8 у відповідності з рис. 3.14.

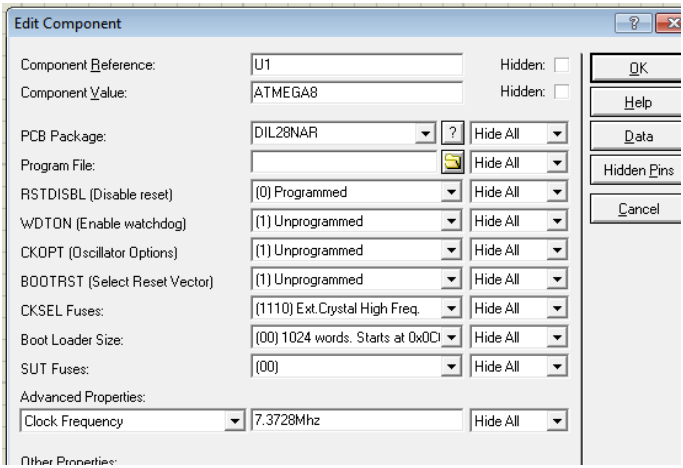


Рисунок 3.14. – Налаштування моделі МК

Варто помітити, що у якості джерела тактового сигналу використовується кварцовий резонатор частотою 7.3728 МГц. Тому необхідно

налаштувати МК за допомогою ф'юзів CKSEL на роботу із зовнішнім кварцовим резонатором.

Наступним кроком є налаштування бітрейту інструменту Virtual terminal (рис. 3.15).

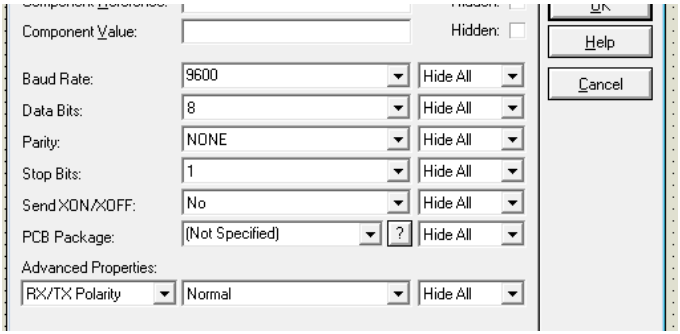


Рисунок 3.15. – Налаштування бітрейту

Після налаштування моделі можна приступати до написання коду. Прийомо-передатчик USART може генерувати переривання за такими подіями:

– переривання після завершення прийому байту;
– переривання після завершення передачі байту;
– переривання після спустошення регістра даних.

Використання переривань дозволяє збільшити ефективність роботи МК, адже під час очікування байту даних він може виконувати корисну роботу, а не опитувати прапор прийому байту.

Зупинимося більш детально на перериванні після завершення прийому байту. За увімкнення цього переривання відповідає біт **RXCIE** регістру **UCSRB** (для більш детальної інформації див. datasheet ATmega8). Вмикається це переривання за допомогою нижченаведеної конструкції:

```
UCSRB|=(1<<RXCIE); // Ввімкнення переривання по прийому байту.
```

Необхідно також пам'ятати, що його робота буде заборонена, якщо не буде встановлено глобальний прапор дозволу переривань за допомогою конструкції:

```
asm("sei");
```

Для роботи з перериваннями необхідно підключти бібліотеку,

```
#include <avr/interrupt.h>.
```

Для виконання зазначеної лабораторної роботи необхідно передавати команди на МК за допомогою USART. Для виконання цього завдання скористаємося інструментом Virtual Terminal. Для введення символів необхідно просто встановити курсор у вікні Virtual Terminal та ввести набір символів як це зображено на рис. 3.16.

Рисунок 3.16. – Введення символів

Зважаючи на те що за замовчуванням у цього інструменту вимкнений показ символів, котрі друкуються, необхідно натиснувши правою кнопкою у вікні Virtual terminal та встановити прапорець на Echo Typed Characters, як це зображено на рис. 3.17.

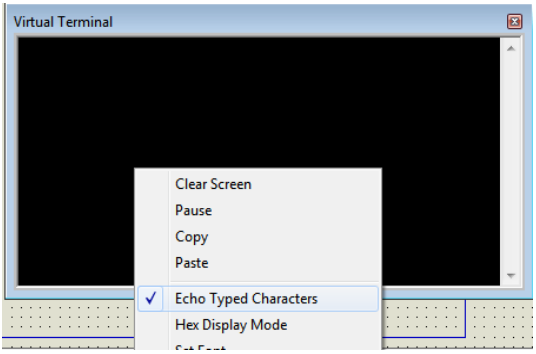


Рисунок 3.17. – Увімкнення відображення символів

3.14. Лабораторна робота № 6

Завдання

1. Зібрати схему у відповідності завданню варіанту. Світлодіоди та кнопки підключити в довільному порядку (до будь-яких не зайнятих портів).
2. Реалізувати програмне забезпечення МК, котре буде реалізовувати роботу згідно з вищенаведеним алгоритмом:

- Під час натиснення на будь-яку кнопку буде виводитися повідомлення, наприклад, такого формату: «Спрацював датчик № 1».
- Також реалізувати реакцію МК на певний символ, наприклад при надходженні команди «n» увімкнути перший світлодіод.

Таблиця 3.5

Варіанти завдань

	Кількість світлодіодів	**Кількість кнопок**
1.	4	1
2.	3	2
3.	2	3
4.	5	2
5.	2	4
6.	7	1
7.	3	4
8.	4	2
9.	1	5
10.	6	3

Приклад коду для вищенаведеної моделі

```
/*
* LR6.c
*
* Created: 30.05.2021 21:36:50
* Author: Nickolay
*/
#include <avr/io.h>
#include <avr/interrupt.h>
#define F_CPU 737280UL
#include <util/delay.h>
#define USART_BAUDRATE 47 // UBBR з таблиці 2
```

```
ISR(USART_RXC_vect)// Підпрограмма обробки переривання
{
   switch(UDR)
     {
        case '1': // Led_1 on
          {
               PORTB &=~(1<<0); // Ввімкнути 1-й світлодіод
               PORTB |= (1<<1); // Вимкнути 2-й світлодіод
           break;
          }
        case '2': // Led_2 on
{
               PORTB &=~(1<<1);
               PORTB |= (1<<0);
           break;
          }
     }
}

void uart_init(void)
{
     // Set baud rate
     UBRRL =USART_BAUDRATE
     UCSRB |=((1<<TXEN)|(1<<RXEN)); //Transmit on
     UCSRC = (1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0);
     UCSRB|=(1<<RXCIE); // Ввімкнення переривання по прийому байту
}
void uart_transmit_message(char* msg)
{ unsigned char i;
i=0;

//Цикл перебору елементів рядка
while ((i<256)&(msg[i]!=0x00) )
   {
     //відправлення поелементно символів рядка
     while ( !( UCSRA & (1<<UDRE))); // Перевірка
       UDR = msg[i]; //Запис символу в регістр передавача
     i++; //Збільшуємо номер елемента рядка
     }
}
int main(void)
{
        uart_init(); //init uart
        uart_transmit_message("Labwork 6 \r\n");
        uart_transmit_message("Author: Stepanenko Semen \r\n");
     asm("sei");
        DDRB |=((1<<1)|(1<<0));
        PORTB|=((1<<1)|(1<<0));
```

```
        unsigned char but_flag=0x00;
        while(1)
        {
             if ((PINB & 0x04)) //Якщо НЕ натиснута перша кнопка
             {
                  but_flag &=~(0x01);

             }
             if ((PINB & 0x08)) //Якщо НЕ натиснута друга кнопка
             {
                  but_flag &=~(0x02);

              }
             if (!(but_flag & 0x01)) // захист від повторення повідомлення
             {
               if (!(PINB & 0x04)) //Якщо натиснута перша кнопка
                {
                     uart_transmit_message("Datchik 1 ON\r\n");
                     but_flag |=(0x01);// Встанолення прапору натиснення кнопки
                 }
                }
              if (!(but_flag & 0x02)) // захист від повторення повідомлення
              {
                if (!(PINB & 0x08)) //Якщо натиснута друга кнопка
                {
                        uart_transmit_message("Datchik 2 ON\r\n");
                    but_flag |=(0x02);// Встанолення прапору натиснення кнопки
                  }
           }
      }
}
```

3.15. Аналогово-цифровий перетворювач мікроконтролерів AVR

Загальні відомості

АЦП, **аналого-цифровий перетворювач** (англ. *analog-to-digital converter (скорочено ADC)*) – пристрій, що перетворює вхідний аналоговий сигнал в дискретний код (цифровий сигнал). **На вхід АЦП подається безперервний аналоговий сигнал, а на виході виходить послідовність цифрових значень.**

Перше про що потрібно пам’ятати – АЦП мікроконтролера вміє вимірювати тільки напругу. Вимірюваний діапазон напруг розбивається

на частини: нуль відповідає мінімальному значенню, а максимальному – відповідає напруга джерела опорної напруги (Aref) .

Гранична частота дискретизації визначає швидкодію АЦП і вимірюється в герцах або кількості вибірок в секунду (SPS – вибірок в секунду). Для мікроконтролерів AVR ця величина дорівнює 15 тис. перетворень в секунду (кіло семплів в секунду). Практично АЦП AVR працює швидше, але його точність погіршується.

Теорема Котельникова (теорема Найквіста – Шеннона, теорема про вибірку) говорить, що аналоговий сигнал має обмежений спектр, може бути відновлений однозначно і без втрат за своїми дискретними відліками, якщо частота вибірки (дискретизації) перевищує максимальну частоту спектру сигналу більш ніж в 2 рази. Тобто, якщо вам потрібно оцифрувати аналоговий сигнал зі смугою спектра 0–7 КГц , то в ідеальному випадку частота дискретизації повинна бути вдвічі більшою від максимальної частоти спектру цього сигналу, тобто > 14 КГц. На практиці все набагато складніше. Перед входом АЦП майже завжди ставлять НЧ фільтр, щоб обмежити спектр сигналу, а частота дискретизації вибирається ще більш високою.

АЦП вимірює напругу з певною частотою, кратною основній частоті мікроконтролера. Щоб досягти цієї точності, виробник рекомендує вимірювати з частотою 50–200 кГц (чим вище частота роботи АЦП – тим більше похибка).

Кількість частин, на котрі буде розбитий вимірюваний сигнал, має назву «роздільна здатність». Для 8-ми бітного АЦП це 28 = 256 значень, для 10-ти бітного 2^{10}= 1024 значення. Припустимо, що необхідно виміряти сигнал від 0 до 10В. Використаємо мікроконтролер Atmega8, з 10-ти бітним АЦП. Це означає що діапазон 10В буде розділений на 1024 значення. 10В/1024 = 0,0097В – з таким кроком ми зможемо вимірювати напругу. Насправді ми не можемо подавати на вхід АЦП напругу більше 5В без використання дільника напруги.

Як джерело опорної напруги можна використовувати як внутрішнє джерело напруги так і зовнішнє. Напруга внутрішнього джерела опорної напруги 2,3–2,7В, його не рекомендується використовути, якщо необхідно досягти високої точності. Зовнішнє джерело підключається до ніжки AVCC або ARef, залежно від налаштувань програми.

Під час використання АЦП ніжка AVCC повинна бути підключена. Напруга на ніжці AVCC не повинна відрізнятися від напруги живлення мікроконтролера більш ніж на 0,3В. Як було зазначено, максимальна вимірювана напруга дорівнює значенню опорної напруги Vref , воно

знаходиться в діапазоні 2В-AVCC. Таким чином, мікроконтролер не може виміряти більш 5В.

Щоб розширити діапазон вимірювання, потрібно вимірювати сигнал через дільник напруги. Наприклад, максимальне значення вимірювальної напруги 10В, опорна напруга 5В. Щоб розширити діапазон вимірювання, потрібно зменшити вимірювальний сигнал в 2 рази.

Формула для розрахунку дільника виглядає так:

$$U_{Вих} = \frac{U_{Вх} R_2}{(R_1 + R_2)};$$

$$5 = \frac{10 R_2}{(R_1 + R_2)};$$

$$(R_1+R_2)=2R_2;$$

$$R_1=R_2.$$

тобто можна взяти будь-які два однакових резистора і підключити їх за схемою зображеною на рис. 3.18.

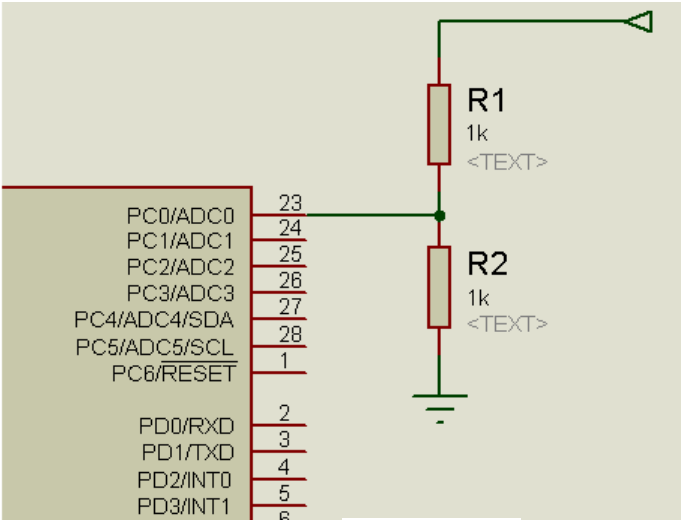


Рисунок 3.18. – Схема дільника

Отже, коли ми вимірюємо напругу через дільник, потрібно отримане значення АЦП помножити на коефіцієнт $U_{Вих}$ /$U_{Вх}$.

Повна формула обчислення вимірювальної напруги має вигляд:

U = (опорна напруга × значення АЦП × коефіцієнт дільника)/число розрядів АЦП.

Приклад: опорна напруга 5В, виміряне значення АЦП = 512, коефіцієнт дільника = 2, АЦП – 10-ти розрядний.

(5 × 512 × 2) / 1024 = 5В – реальне виміряне значення напруги.

Підключення 4-х кнопок до одного виводу МК

Для знайомства з модулем АЦП мікроконтролера AVR розглянемо просту, але цікаву схему (див. рис. 3.19). Чотири кнопки підключені до дільника напруги (резистори R1–R4). Під час натискання вони комутують на вхід нульового каналу АЦП різні напруги. Вимірюючи ці напруги за допомогою АЦП і, визначаючи в який діапазон вони потрапляють, мікроконтролер буде розпізнавати номер натиснутої кнопки.

Резистор R4 потрібен, щоб вхід АЦП не «бовтався в повітрі», коли жодна кнопка не натиснута. Якщо цього не зробити АЦП буде давати хибні значення в результаті наводок. Номінал резистора R4 обраний великим, щоб не впливати на дільник напруги.

R5 і C1 складають низькочастотний фільтр для захисту від брязкоту кнопок і перешкод. Додатково резистор R6 відіграє функцію струмообмежувача, без нього при натисканні кнопки S4 вивід мікроконтролера безпосередньо з'єднувався би з плюсом джерела живлення. Для індикації номера натиснутої кнопки в схемі використовуються 4 світлодіода.

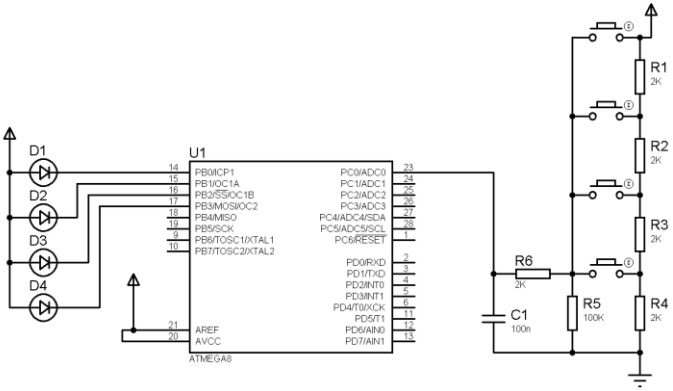


Рисунок 3.19. – Схема підключення

Для перевірки моделі використаємо такий код:

```
#include <avr/io.h>
#define F_CPU 8000000UL
#include <util/delay.h>
#define StartConvAdc() ADCSRA |= (1<<ADSC) //Макрос запуску конвертації ADC
int main(void)
{
  ADCSRA = ((1<<ADEN)|(0<<ADSC)|(0<<ADIF)|(1<<ADPS2)|(0<<ADPS1)|(1<<ADPS0));
  ADMUX = (0<<REFS1)|(1<<REFS0)|(1<<ADLAR)|(0<<MUX3)|(0<<MUX2)|(0<<MUX1)
|(0<<MUX0);
  StartConvAdc();
  DDRB=15;// LED 1-4
  PORTB=15;
  while(1)
  {
         while((ADCSRA & ADIF)==0);
         unsigned char data=ADCH;
         ADCSRA &=~(1<<ADIF);
         unsigned char flag=0x00;
         if (data > 240) // if B1 pressed
       PORTB &=~(1<<0);
  else
     if (data > 180)//if B2 pressed
              PORTB &=~(1<<1);
     else
          if (data > 120) //if B3 pressed
                    PORTB &=~(1<<2);
          else
                if (data > 50)//if B4 pressed
                      PORTB &=~(1<<3);
                else
                     PORTB=15;// all leds off
                StartConvAdc();
         _delay_ms(200);
  }
}
```

Алгоритм цього коду такий:

1. Ініціалізація модуля ADC.
2. Ініціалізація портів вводу виводу, до яких підключено світлодіоди.
3. Зчитування значення напруги з модуля ADC.
4. Визначення яка з кнопок натиснута, якщо виявлено натиснення кнопки, запалити відповідний світлодіод.

Регістри модуля АЦП

Для керування модулем АЦП передбачено 4 регістри: ***ADCSRA, ADMUX, ADCL, ADCH.*** Останні два регістри використовуються для зберігання результату аналогово-цифрового перетворення. В Atmel Studio ця пара регістрів називається ADC. Але ніяких даних ми не отримаємо без належного налаштування модуля АЦП за допомогою регістру ***ADCSRA.*** На нижченаведеному рисунку зображено функції окремих біт цього регістру (див. рис. 3.20).

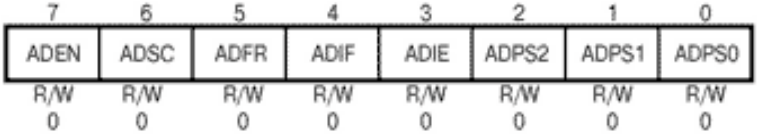


Рисунок 3.20. – Регістр ADCSRA

Перший біт який потребує нашої особливої уваги – ***ADEN***. Встановлення цього біту в одиницю вмикає модуль АЦП. Без його встановлення даного біту АЦП працювати не буде.

ADSC – відповідає за запуск аналогово-цифрової конвертації. Під час встановлення цього біту відбувається запуск аналогово-цифрової конвертації.

АЦП мікроконтолерів ATmega8 підтримує два режими роботи:

1. Одиничне перетворення (кожен раз конвертація відбувається встановленням біту ADSC).

2. Режим безперервних вимірювань активізується встановленням біту ***ADFR*** в одиницю.

Біт ***ADIF*** встановлюється в одиницю у разі завершення аналогово-цифрового перетворення. Використовується для інформування ЦП про завершення конвертації. Під час встановленого в одиницю ***ADIE*** генерується переривання.

Біт ***ADIE*** використовується для дозволу генерування переривання. У разі завершення конвертації модулем АЦП при встановленому біті ***ADIE*** генерується переривання.

Біти ***ADPS2:ADPS0*** використовуються для встановлення частоти роботи модуля АЦП.

Таблиця 3.6

Параметри ADPS

ADPS2	ADPS1	ADPS0	Кофіцієнт ділення частоти
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Частота роботи модуля АЦП отримується діленням основної тактової частоти на коефіцієнт ділення, який встановлюється за допомогою біт ***ADPS2:ADPS0.***

Наступним регістром є регістр ADC Multiplexer Selection Register (***ADMUX***). Багатоканальність АЦП МК AVR досягається за рахунок використання мультиплексора. Мультиплексори відносяться до пристроїв комутування цифрової інформації. Вони здійснюють комутацію одного з декількох інформаційних входів $\mathbf{x_i}$ до одного виходу ***y***.

Функції окремих біт регістру ***ADMUX*** наведені на рис. 3.21.

7	6	5	4	3	2	1	0	
REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

Рисунок 3.21. – Регістр ADMUX

REFS1:REFS0 – біти вибору джерела опорної напруги.
00 AREF
01 AVcc, із зовнішнім конденсатором на AREF
10 Резерв;
11 Внутрішнє 2.56В джерело, з зовнішнім конденсатором на AREF.

Біт ***ADLAR*** визначає вирівнювання результату в парі регістрів ***ADCL:ADCH.***

Якщо : ***ADLAR =0***

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	

Якщо : ***ADLAR =1***

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	

Бітове поле ***MUX3:MUX0*** використовується для вибору каналу мультиплексора. Більш детально табл. 3.7.

Таблиця 3.7

Параметри MUX3:MUX0

MUX3	***MUX2***	***MUX1***	***MUX0***	**АЦП канал**
0	0	0	0	ADC0
0	0	0	1	ADC1
0	0	1	0	ADC2
0	0	1	1	ADC3
0	1	0	0	ADC4
0	1	0	1	ADC5
0	1	1	0	ADC6
0	1	1	1	ADC7

3.16. Лабораторна робота № 7

Завдання

1. Дати відповідь на питання:

Яким чином зконфігуровано АЦП за допомогою таких команд:

```
ADCSRA = ((1<<ADEN)|(0<<ADSC)|(0<<ADIF)|(1<<ADPS2)|(0<<ADPS1)|(1<<ADPS0));

ADMUX=(0<<REFS1)|(1<<REFS0)|(1<<ADLAR)|(0<<MUX3)|(0<<MUX2)|(0<<MUX1)
                                        (0<<MUX0);
```

Яким чином запустити режим аналогово-цифрової конвертації?

Відповіді написати у звіті.

2. Розробити вольтметр згідно з власним завданням. Якщо діапазон вимірювальних напруг більше за 5В необхідно використати дільник напруги. АЦП використовувати у 8-ми бітному режимі (див. приклад). Результат перетворення вивести без змін за допомогою інтерфейсу, вказаного у завданні. Результат виводиться на інтерфейс під час виконання певної умови.

Таблиця 3.8

Варіанти завдань

№ вар.	Діапазон напруг	Канал АЦП	Інтерфейс виводу	Умова виводу
1.	0..5В	0	PORTB	Нат. кнопки
2.	0..10В	0	PORTB	Delay(1000)
3.	0..5В	2	UART	Нат. кнопки
4.	0..15В	3	UART	Нат. кнопки
5.	0..20В	1	PORTD	Delay(500)
6.	0..5В	2	UART	Delay(500)
7.	0..25В	4	PORTD	Нат. кнопки
8.	0..30В	1	PORTB	Delay(500)
9.	0..5В	0	UART,PORTB	Delay(500)
10.	0..5В	0	UART,PORTB	Нат. кнопки

Пояснення до завдання: Створити вольтметр на основі вищенаведеного завдання означає: відповідно до вказаної напруги розрахувати дільник напруги (за необхідності). У якості каналу АЦП обрати той, який вказано у завданні. Для виводу інформації обрати інтерфейс який вказано у завданні. Під виводом результату мається на увазі вивід 8-ми біт результату конвертації АЦП (регістр ADCH або ADCL залежно від налаштувань вирівнювання) на запропонований інтерфейс. Якщо вказано два інтерфейси, то вивід результату здійснити одночасно на два інтерфейси. У якості умов для запуску виводу інформації можуть бути такі події:

1. Натиснення кнопки. (до якого порту та виводу МК підключити кнопку обирає студент.)

2. Delay (час в мілісукендах) виводити результат перетворення на інтерфейс з періодичністю, вказаною в дужках (час вказано в мілісекундах). Для формування затримок необхідно використати бібліотеку Delay (#include <util/delay.h>), функція – _delay_ms (час в м. сек). Якщо у якості інтерфейсу виводу вказано порт вводу-виводу, то для відображення інформації до нього підключити світлодіоди за схемою із загальним катодом. Наприклад:

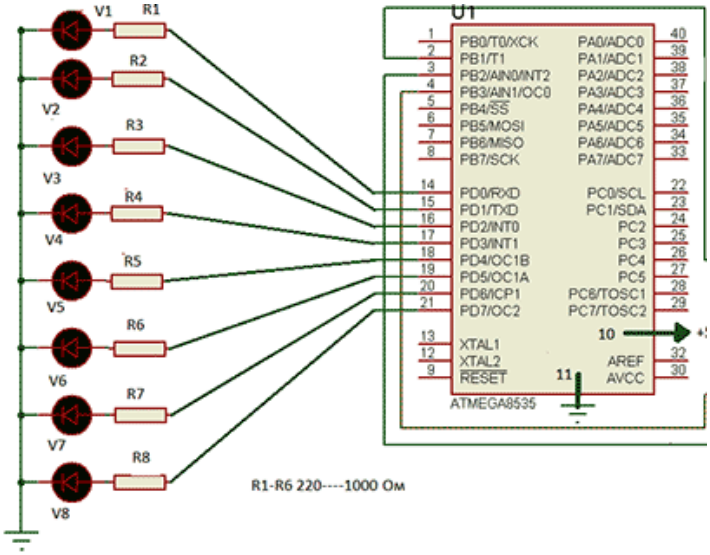


Рисунок 3.22. – Підключення світлодіодів

Якщо у якості інтерфейсу виводу вказано UART, то використати для відображення інформації інструмент Virtual Terminal (див. попередню роботу). Швидкість обміну 9600 Kb/s.

3.17. Інтерфейс SPI мікроконтролерів AVR. Робота з SD-карткою

Загальні відомості про інтерфейс SPI

SPI – популярний інтерфейс для послідовного обміну інформацією між мікросхемами. Інтерфейс SPI, разом з I^2C, належить до найбільш широко – використовуваних інтерфейсів для з'єднання мікросхем. Він був запроваджений компанією Motorola для своїх пристроїв, а в цей час використовується в продукції багатьох виробників. Його найменування є абревіатурою від «Serial Peripheral Bus», що відображає його призначення – шина для підключення зовнішніх пристроїв. Шина SPI організована за принципом «ведуча-підлегла». У якості ведучої шини зазвичай виступає мікроконтролер, але нею також може бути програмована логіка, DSP-контролер або спеціалізована мікросхема.

Підключенні до ведучої шини зовнішні пристрої утворюють підлеглі шини. В ролі підлеглих виступають різного роду мікросхеми, у тому числі запам'ятовуючі пристрої (EEPROM, Flash-пам'ять, SRAM), годинники реального часу (RTC), АЦП/ЦАП, цифрові потенціометри, спеціалізовані контролери тощо.

Електричне підключення

Існує два типи підключення до шини SPI, у кожному з яких беруть участь чотири сигнали (їх основні та альтернативні позначення див. у табл. 3.9). Найпростіше підключення, у якому беруть участь тільки дві мікросхеми, показано на рис. 3.23. Тут ведуча шина передає дані лінією MOSI синхронно зі згенерованим нею сигналом SCLK, а підлегла захоплює передані біти даних на певних фронтах прийнятого сигналу синхронізації.

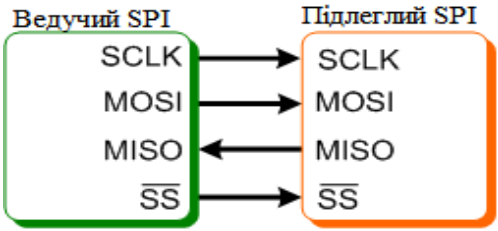


Рисунок 3.23. – Найпростіша схема підключення

Одночасно з цим підлегла виконує свою посилку даних. Представлену схему можна спростити виключенням лінії MISO, якщо використовувана підпорядкована мікросхема не передбачає відповідну передачу даних або в ній немає потреби.

Односторонню передачу даних можна зустріти у таких мікросхемах як ЦАП, цифрові потенціометри, програмовані підсилювачі і драйвери. Таким чином, розглянутий варіант підключення підпорядкованої мікросхеми вимагає 3 або 4 лінії зв'язку. Щоб підпорядкована мікросхема приймала і передавала інформацію, крім наявності сигналу синхронізації, необхідно також, щоб лінія SS була переведена в низький стан. В іншому випадку, підпорядкована мікросхема буде неактивна. Вхід (SS) вибору мікросхеми служить для переведення підлеглої мікросхеми в її початковий стан, а іноді ініціює виставлення першого біта даних на виводі MISO.

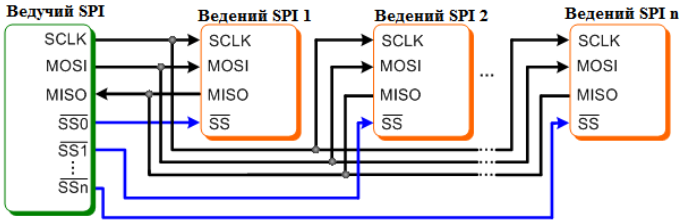


Рисунок 3.24. – Підключення до шини SPI декількох ведених

За необхідності підключення до шини SPI декількох мікросхем використовується незалежне (паралельне) підключення (див. рис. 3.24).

У разі незалежного підключення всі сигнали, окрім вибору мікросхем, з'єднані паралельно, а ведуча шина, переводом того чи іншого сигналу SS в низький стан, задає, з якою з ведених мікросхем вона буде обмінюватися даними. Головним недоліком такого підключення є необхідність у додаткових лініях для адресації підлеглих мікросхем (загальне число ліній зв'язку 3 + n, де n – кількість підлеглих мікросхем).

Таблиця 3.9

Електричні сигнали шини SPI

Ведуча шина SPI			**Введена шина**		
Основне позначення	Альтернативне позначення	Опис	Основне позначення	Альтернативне позначення	Опис
MOSI	DO, SDO, DOUT	Вихід послідовної передачі даних	MOSI	DI, SDI, DIN	Вхід послідовного прийому даних
MISO	DI, SDI, DIN	Вхід послідовного прийому даних	MISO	DO, SDO, DOUT	Вихід послідовної передачі даних
SCLK	DCLOCK, CLK, SCK	Вихід синхронізації передачі даних	SCLK	DCLOCK, CLK, SCK	Вхід синхронізації прийому даних
SS	CS	Вихід вибору веденого	SS	CS	Вхід вибору веденого (вибір мікросхеми)

Протокол передачі

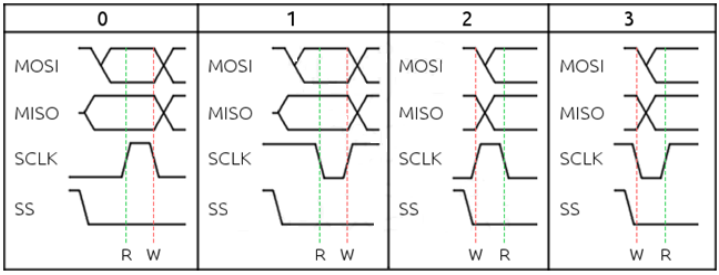
Протокол передачі інтерфейсом SPI гранично простий і, практично, ідентичний логіці роботи зсувного регістру, яка полягає у виконанні операції зсуву і, відповідно, побітного введення і виведення інформації за певними фронтами сигналу синхронізації. Установка даних під час передачі і вибірка у процесі прийому завжди виконується протилежними фронтами синхронізації. Це необхідно для гарантування вибірки даних після надійного їх встановлення. Якщо до цього врахувати, що в якості першого фронту в циклі передачі може виступати наростаючий або падаючий фронт, то можливі всього чотири варіанти логіки роботи інтерфейсу SPI. Ці варіанти отримали назву режимів SPI і описуються двома параметрами:

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

– CPHA – фаза синхронізації; від цього параметра залежить, у якій послідовності виконується установка і вибірка даних (якщо CPHA=0, то переднім фронтом в циклі синхронізації буде виконуватися вибірка даних, а потім, заднім фронтом – установка даних; якщо ж CPHA=1, то установка даних виконуватиметься переднім фронтом у циклі синхронізації, а вибірка – заднім). Інформація щодо режимів SPI узагальнена в табл. 3.10.

Таблиця 3.10

Режими SPI



Регістри модулю SPI мікроконтролера AVR ATmega8

Модуль SPI в мікроконтролері ATmega8 представлений трьома регістрами:

– SPCR – регістр керування модулем SPI;
– SPSR – регістр статусу модуля SPI;
– SPDR – регістр даних.

Кофігурація модуля SPI встановлюється за допомогою регістра **SPCR** (SPI Control Register).

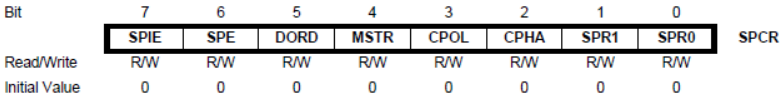


Рисунок 3.25. – Регістр SPCR

SPIE – дозволяє/забороняє переривання від модуля SPI. Якщо біт встановлений в 1, переривання від SPI дозволені.

SPE – вмикає / вимикає модуль SPI. Якщо біт встановлений в 1, модуль SPI ввімкнений.

DORD – визначає порядок передачі даних. Коли біт встановлений в 1, вміст регістра даних передається молодшим бітом уперед. Коли біт скинутий, то старшим бітом уперед.

MTSR – визначає режим роботи мікроконтролера. Якщо біт встановлений в 1, мікроконтролер працює в режимі Master (ведучий). Якщо біт скинутий – у режимі Slave (підпорядкований). Зазвичай мікроконтролер працює у режимі Master.

CPOL і **CPHA** – визначають у якому режимі працює SPI модуль. Необхідний режим роботи залежить від використовуваного периферійного пристрою.

Таблиця 3.11

Режим роботи SPI

Mode	CPOL	CPHA
SPI Mode 0	0	0
SPI Mode 1	0	1
SPI Mode 2	1	0
SPI Mode 3	1	1

SPR1 і **SPR0** – визначають частоту тактового сигналу SPI модуля, тобто швидкість обміну. Максимально можлива швидкість обміну завжди вказується в специфікації периферійного пристрою.

Статусний регістр SPSR (SPI Status Register) призначений для контролю стану SPI модуля, крім того він містить додатковий біт керування швидкістю обміну.

Bit	7	6	5	4	3	2	1	0	
	SPIF	WCOL	–	–	–	–	–	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 3.26. – Регістр SPSR

SPIF – прапор переривання від SPI. Він встановлюється в 1 після закінчення передачі байта даних. Якщо дозволені переривання модуля, одночасно з установкою цього прапора генерується переривання від SPI. Скидання прапора відбувається апаратно, під час виклику підпрограми обробки переривання або після читання регістра SPSR з наступним зверненням до регістру даних SPDR.

WCOL – прапор конфлікту запису. Прапор встановлюється в 1, якщо під час передачі даних виконується спроба запису в регістр даних SPDR. Прапор скидається апаратно після читання регістра SPSR з наступним зверненням до регістру даних SPDR.

SPI2X – біт подвоєння швидкості обміну. Установка цього розряду в 1 подвоює частоту тактового сигналу SCK. Мікроконтролер у цьому разі повинен працювати в режимі Master.

Взаємозв'язок між бітами SPR0, SPR1, SPI2X і частотою тактового сигналу SCK показаний в таблиці 3.12.

Таблиця 3.12

Швидкість роботи модуля SPI в режимі Master (ведучий)

SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

Робота з SD карткою

У якості веденого пристрою в зазначеній роботі використаємо SD картку. SD картка являє собою мікросхему флеш-пам'яті. Найменша одиниця до якої може адресуватися ведучий пристрій – це сектор розміром 512 байт.

Для ініціалізації картки в режим SPI необхідно подати не менше 74 імпульсів на вивід SCLK при встановлених високих рівнях на виводах MOSI та CS. Тобто, передати за допомогою інтерфейсу SPI 8 байтів 0xFF.

Після цього необхідно виставити CS в нульовий рівень для переходу картки в режим SPI. Команд для роботи з карткою, доволі багато, для виконання цієї роботи буде достатньо команд наведених у табл. 3.13.

Таблиця 3.13

Основні команди SD-Card

Індекс команди	Аргумент	Дані	Опис команди
CMD0	немає	Ні	Програмне зкидання
CMD1	немає	Ні	Ініціалізація SD
CMD17	Адреса блоку	Так	Читати блок
CMD24	Адреса блоку	Так	Записати блок

Для виконання зазначеної роботи в середовищі Proteus нам необхідна модель SD картки, в бібліотеці компонентів вона називається MMC.

Для роботи цієї моделі необхідний файл з розширенням .mmc, котрий являє собою образ флеш накопичувача. Його можна створити за допомогою програми WinImage. Для цього необхідно скопіювати файл SD.mmc з теки із зазначеною роботою, та вмонтувати його в модель MMC за допомогою кнопки вибору файлу Card Image File, як це зображено на рис. 3.27.

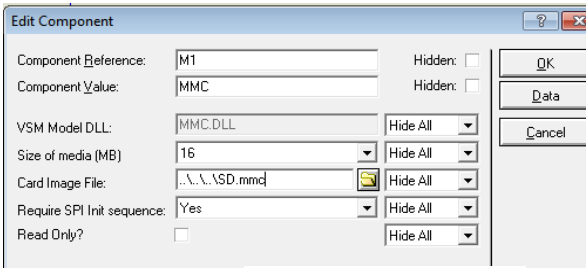


Рисунок 3.27. – Монтування файлу **mmc**

3.18. Лабораторна робота № 8

Завдання

1. Зібрати модель в середовищі Proteus зображену на рис. 3.28.
2. Записати в блок пам'яті номер який відповідає власному номеру наступну інформацію: номер варіанта прізвище та ім'я студента, будь-яку іншу за бажанням студента.

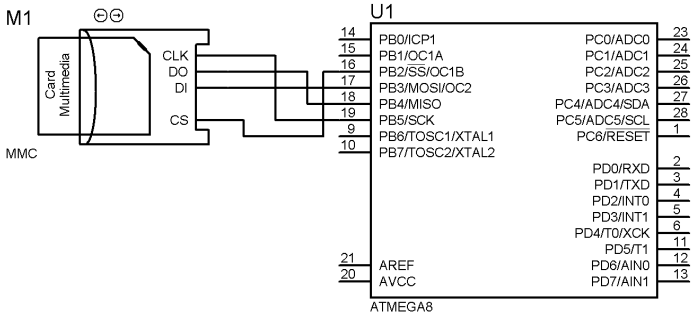


Рисунок 3.28. – Модель

Приклад виконання роботи

```
#include <avr/io.h>
#define F_CPU 8000000UL
#define SPI_CS 2
#define NO_DEVICE               0x01    // Немає зкиду
#define NO_DISK                 0x02    // Даний пристрій не є флеш-карткою
#define SECSESSFUL              0x00    // Успішна ініціалізація
unsigned int count=0;                   // Кількість записаних байт в сектор
void InterfaceInit(void)                // Ініціалізація SPI
{
       DDRB |= (1<<PB5) | (1<<PB3)|(1<<SPI_CS);
       PORTB &= ~(1<<PB4);
       SPCR = ((1<<MSTR)|(1<<SPE)|(1<<SPR1)); //MSB first
       PORTB|=(1<<SPI_CS); //CS High level
}
unsigned char SPI_RW(unsigned char arg) // Запис читання даних з інтерфейсу SPI
{
     SPDR =arg;
        while(!(SPSR & (1<<SPIF)));
        return SPDR;
}
unsigned char send_cmd (unsigned char cmd,unsigned long arg)
// процедура подачі комадни для флешки
{
unsigned char temp,count;
```

```
SPI_RW(0xff);
SPI_RW(cmd);                                // Start + Command index
SPI_RW((unsigned char)(arg >>24));          // Argument[31..24]
SPI_RW((unsigned char)(arg >> 16));         // Argument[23..16]
SPI_RW((unsigned char)(arg >> 8));          // Argument[15..8]
SPI_RW((unsigned char)arg);                 // Argument[7..0]
SPI_RW(0x95);                 //CRC
      count=0;
do {                          //Почекати R1 відповіді
     temp=SPI_RW(0xff);
       count++;
    } while ( ((temp&0x80)!=0x00)&&(count<0xff) );
return temp;
}

unsigned char card_init(void)        // Ініціалізація картки
  {
     unsigned char temp;
     unsigned int count=10;
     while (count) //80 імпільсів на лінії CLK
      {
        SPI_RW(0xff);
        count--;
        }
         PORTB &=~(1<<SPI_CS);        //CS Low
         temp=send_cmd(0x40,0x00);    //CMD0
         if (temp!=0x01) return NO_DEVICE;     //Вийти, карта не знайдена
         SPI_RW(0xff);
         count=0;
         do{
             temp=send_cmd(0x41,0x00);    //CMD1
             SPI_RW(0xff);
                 count++;
             } while ( (temp!=0x00)&&(count<0xffff) ); //Чекаємо 0x01
             if (count>=0xffff) return NO_DISK;
             return SECSESSFUL;           // Успіх
  }

unsigned char SD_Card_write(char msg[],unsigned long Block_NO)
   {
         unsigned char tmp;
         if (count==0)                    //Якщо старт запису
         {
                tmp=send_cmd(0x58,Block_NO);
              if (tmp)                    //command not successful
              {
                return tmp;
                }
                 SPI_RW(0xff);
                 SPI_RW(0xfe);
           }
           unsigned char i=0;
           while ((i<256)&(msg[i]!=0x00) )
   {
```

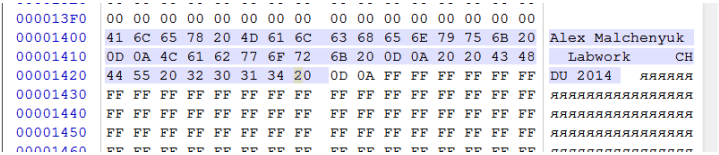
```
// Відправлення поелементно символів рядка
SPI_RW(msg[i]);
       i++;                          //Збільшуємо номер елемента рядка

       if (count==511)               //Якщо блок заповнено
       {
              SPI_RW(0x95);          // CRC
              SPI_RW(0x95);          // CRC
            tmp= SPI_RW(0xff);
          if ((tmp&0x05)!=0x05)      //Error write
          {
           return tmp;
            }
         count=0;
       do {                           //Чекаємо закінчення busy-стану
           tmp= SPI_RW(0xff);
           count++;
         }while ( (tmp!=0xff)&&(count<0xffff) );
         if (count>=0xffff) return 0x05; //Busy Error
         count=0;
                return SECSESSFUL;
       }
       else
       {
            count++;
       }

     }
  return SECSESSFUL;
}
void finalize(void)
{
       unsigned char temp;
       if (count!=511)               //Якщо блок не завершено
       {
              while(count<511)
              {
                       SPI_RW(0xff);
                       count++;
              }
       }
       SPI_RW(0x95); // CRC
              SPI_RW(0x95);// CRC
            temp= SPI_RW(0xff);
           count=0;
       do {                           //Чекаємо закінчення busy-стану
           temp= SPI_RW(0xff);
           count++;
             }while ( (temp!=0xff)&&(count<0xffff) );
             if (count>=0xffff)
                      {
                               DDRB|=(1<<0);
                           PORTB|=(1<<0);
                      }
             count=0;
```

```
    }
  int main(void)
  {
        unsigned char i=0;
        InterfaceInit();
        unsigned long Variant=10;
        Variant=0x0200*Variant; //Перетворення в номер блоку номера варіанта
        i=card_init();
        SD_Card_write("Alex Malchenyuk \r\n",Variant); //Передача даних на картку
        SD_Card_write("Labwork \r\n",Variant);
        SD_Card_write(" CHDU 2014 \r\n",Variant);
        finalize();
}
```

Для перевірки результатів роботи необхідно відкрити програмою WinHex файл SD.mmc. За допомогою комбінації ctr + F знайти частину з введеного тексту. Результат внести до звіту.



000013F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00001400	41	6C	65	78	20	4D	61	6C	63	68	65	6E	79	75	6B	20	Alex Malchenyuk
00001410	0D	0A	4C	61	62	77	6F	72	6B	20	0D	0A	20	20	43	48	Labwork CH
00001420	44	55	20	32	30	31	34	20	0D	0A	FF	FF	FF	FF	FF	FF	DU 2014 яяяяяя
00001430	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	яяяяяяяяяяяяяяяя
00001440	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	яяяяяяяяяяяяяяяя
00001450	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	яяяяяяяяяяяяяяяя

Рисунок 3.29. – Вікно програми WinHex

Список використаних джерел

1. PCB Design & Simulation Made Easy [Електронний ресурс] – Режим доступу : http://www.labcenter.co.uk. 2021. – Назва з екрана [Сайт розробників програми Proteus].
2. Кушнір І. С. Методичні вказівки до лабораторних робіт з дисципліни «Промислова електроніка» / І. С. Кушнір, С. Ю. Манаков. – Одеса : ОНАЗ ім. О. С. Попова, 2017 – 51 с.
3. Цирульник С. М. Програмування мікроконтролерів AVR : [навчальний посібник] / С. М. Цирульник, О. Д. Азаров, Л. В. Крупельницький, Т. І. Трояновська. – Вінниця : ВНТУ, 2018. – 111 с.
4. Мікропроцесорні та мікроконтролерні системи. Лабораторний практикум [Електронний ресурс] : навчальний посібник для студентів освітньої програми «Інтегровані інформаційні системи» спеціальності 126 «Інформаційні системи та технології» / КПІ ім. Ігоря Сікорського ; уклад. А. О. Новацький. – Електронні текстові дані (1 файл: 13,14 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2018. – 247 с. – Назва з екрана.
5. Новацький А. О. Проєктування та програмування мікропроцесорних систем і мереж : Проєктування мережі 1-WIRE [Електронний ресурс] : навчальний посібник для студентів спеціальностей 7.05020101, 8.05020101 «Комп'ютеризовані системи управління та автоматика» / [А. О. Новацький] ; НТУУ «КПІ». – Електронні текстові дані (1 файл : 3,43 Мбайт). – Київ : НТУУ «КПІ», 2014. – 141 с. – Назва з екрана.
6. Павельчак А. Г. Програмування мікроконтролерів систем автоматики : конспект лекцій для студентів базового напряму 050201 «Системна інженерія» / укл. : А. Г. Павельчак, В. В. Самотий, Ю. В. Яцук – Львів : Львівська політехніка. – 2012. – 143 с.
7. Поджаренко В. О., Кучерук В. Ю., Севастьянов В. М. Основи мікропроцесорної техніки. Навчальний посібник. – Вінниця : ВНТУ, 2006. – 226 с.

8. Проєктування мікропроцесорних систем : Розд. «Програмування мікроконтролерів родини AVR» [Електронний ресурс] : навчальний посібник для студентів напряму підготовки 6.050201 «Системна інженерія» / НТУУ «КПІ» ; уклад. А. О. Новацький, Є. В. Глушко. – Електронні текстові дані (1 файл : 7,36 Мбайт). – Київ : НТУУ «КПІ», 2010. – Назва з екрана.
9. Цирульник С. М. Проєктування мікропроцесорних систем : навчальний посібник / С. М. Цирульник, Г. Л. Лисенко. – Вінниця : ВНТУ, 2010. – 201 с.
10. Шпак Ю. А. Программирование на языке С для AVR и PIC микроконтроллеров. – К. : МК-Пресс, 2006. – 400с.

Навчальне видання

Микола Іванович
СІДЄЛЄВ

Володимир Миколайович
ЗАПАЛЬСЬКИЙ

Олександр Євгенович
БЄЛІКОВ

ПРОГРАМУВАННЯ СПЕЦІАЛІЗОВАНИХ МІКРОКОНТРОЛЕРНИХ ТА ВБУДОВАНИХ КОМП'ЮТЕРНИХ СИСТЕМ ДЛЯ ЗАСОБІВ АВТОМАТИЗАЦІЇ

Навчальний посібник

Редактор *Р. Грубкіна*.
Технічний редактор *О. Петроченко*.
Комп'ютерна верстка *Н. Кардаш.*
Друк *С. Волинець*. Фальцювально-палітурні роботи *О. Мішалкіна.*

Підп. до друку 23.09.2021.
Формат 60x84$^{1}/_{16}$. Папір офсет.
Гарнітура «Times New Roman». Друк ризограф.
Ум. друк. арк. 5,58. Обл.-вид. арк. 2,48.
Тираж 300 пр. Зам. № 6390.

Видавець і виготовлювач: ЧНУ ім. Петра Могили.
54003, м. Миколаїв, вул. 68 Десантників, 10.
Тел.: 8 (0512) 50–03–32, 8 (0512) 76–55–81, e-mail: rector@chmnu.edu.ua.
Свідоцтво суб'єкта видавничої справи ДК № 6124 від 05.04.2018.