МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Мікропроцесорні та мікроконтролерні системи

Лабораторний практикум

*Рекомендовано Методичною радою КПІ ім.Ігоря Сікорського
як навчальний посібник для студентів, які навчаються за освітньою програмою «Інтегровані інформаційні системи» за спеціальністю 126 «Інформаційні системи та технології»*

Київ
КПІ ім. Ігоря Сікорського
2018

Мікропроцесорні та мікроконтролерні системи : Лабораторний практикум [Електронний ресурс] : навч. посіб. для студ. освітньої програми «Інтегровані інформаційні системи» спеціальності 126 «Інформаційні системи та технології», / КПІ ім. Ігоря Сікорського; уклад.: А.О. Новацький. – Електронні текстові дані (1 файл: 18.983 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2018. – 247 с.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 5 від 24.01. 2019 р.) за поданням Вченої ради факультету Інформатики та обчислювальної техніки (протокол № 4 від 29.11. 2018 р.)

Електронне мережне навчальне видання

Мікропроцесорні та мікроконтролерні системи

Лабораторний практикум

Укладач: Новацький Анатолій Олександрович, к.т.н., доцент

Відповідальний редактор: Долина Віктор Георгійович, к.т.н., доцент, КПІ, ФІОТ, кафедра автоматики та управління в технічних системах

Рецензент: Ткач Михайло Мартинович, к.т.н., доцент, КПІ, ФІОТ, кафедра технічної кібернетики

Навчальний посібник охоплює теоретичний матеріал та практичні завдання, які необхідні для виконання лабораторного практикуму з дисципліни «Мікропроцесорні та мікроконтролерні системи». Практикум виконується в комп'ютерному класі кафедри з використанням моделюючого пакету PROTEUS 8.5/8.6. В посібнику наводяться рекомендації по використанню цього пакету та виконанню робіт з наступної тематики: моделювання модуля універсального асинхронного приймача–передавача, моделювання модуля аналого–цифрового перетворювача, моделювання пристроїв керування двигуном постійного струму, уніполярним та біполярним кроковим двигуном та моделювання цифрового вольтметру. В роботі наводяться приклади схем моделювання, алгоритми та керуючі програми мовою С та Асемблер для мікроконтролерів сім`ї МК51 та AVR. Тематика посібника відповідає робочій програмі дисципліни «Мікропроцесорні та мікроконтролерні системи», яка є важливою дисципліною у навчальному плані підготовки бакалаврів зі спеціальності 126 «Інформаційні системи та технології». Робота може бути корисною студентам відповідних спеціальностей при вивченні дисциплін, пов'язаних із використанням мікропроцесорних та мікроконтролерних пристроїв та систем, а також при виконанні бакалаврських робіт, курсових проектів, магістерських робіт, в яких використовуються відповідні пристрої. Останнє було враховано при оформленні роботи, яке виконано згідно вимог до конструкторської документації.

Зміст

ВСТУП

Напевно багато радіоаматорів–початківців стикалися з ситуацією коли, вирішивши зібрати вподобаний і, безсумнівно, потрібний пристрій, через недосвідченість, а може через помилки в схемі або ж за іншими обставинами, просто–напросто спалювали насилу придбані дорогі радіодеталі. І скоріше за все більшість, обпікшись на перших невдачах, закидали заняття радіоелектронікою назавжди.

У наше століття повальної комп'ютеризації, знайшовся вихід з цього глухого кута. З'явилася величезна кількість програм симуляторів, які заміняють реальні радіодеталі і прилади віртуальними моделями. Симулятори дозволяють, без складання реального пристрою, налагодити роботу схеми, знайти помилки, отримані на стадії проектування, зняти необхідні характеристики і багато іншого.

Одна з таких програм PROTEUS VSM. Але симуляція радіоелементів це не єдина здатність програми. Proteus VSM, яку створено фірмою Labcenter Electronics на основі ядра SPICE3F5 університету Berkeley, є так званим середовищем наскрізного проектування Це означає створення пристрою, починаючи з його графічного зображення (принципової схеми) і закінчуючи виготовлення друкованої плати пристрою, з можливістю контролю на кожному етапі виробництва.

Але, не дивлячись на уявну складність програми, користуватися нею зможуть не тільки професіонали в світі радіоелектроніки, а й новачки, які навчилися, а може поки що і ні, відрізняти резистор від транзистора.

PROTEUS VSM складається з двох самостійних програм ISIS і ARES. ARES це програма для трасування друкованих плат з можливістю створення своїх бібліотек корпусів і в даній роботі розглядатися не буде. Основною програмою є ISIS, в якій передбачено гарячий зв'язок з ARES для передачі проекту для розведення друкованої плати.

У «сферу–впливів» PROTEUS VSM входять як найпростіші аналогові пристрої, так і складні системи, які створено на популярних нині мікроконтролерах. Доступна величезна бібліотека моделей елементів, поповнювати яку може сам користувач. Природно для цього потрібно досконально знати роботу елемента і вміти програмувати. Можливість анімації схем дозволяє програмі стати прекрасним навчальним посібником на уроках в школі і ВНЗ. Достатній набір інструментів і функцій, серед яких вольтметр, амперметр, осцилограф, всілякі генератори, здатність налагоджувати програмне забезпечення мікроконтролерів, роблять PROTEUS VSM хорошим помічником розробнику електронних пристроїв.

1 ОПИС НАЛАГОДЖУВАЧА PROTEUS 8.6

1.1 Інсталяція

Завантажити Proteus останньої версії можна з файлообмінних мереж або придбати на сайті виробника. В даних методичних вказівках була використана програма Proteus версії 8.6. Після запуску файлу інсталятора з’явиться вікно привітання (рисунок 1.1). Далі необхідно виконати приведену на рисунках нижче послідовність дій. Перехід між екранами встановлення програми здійснюється по натисненню кнопки “Далее”.

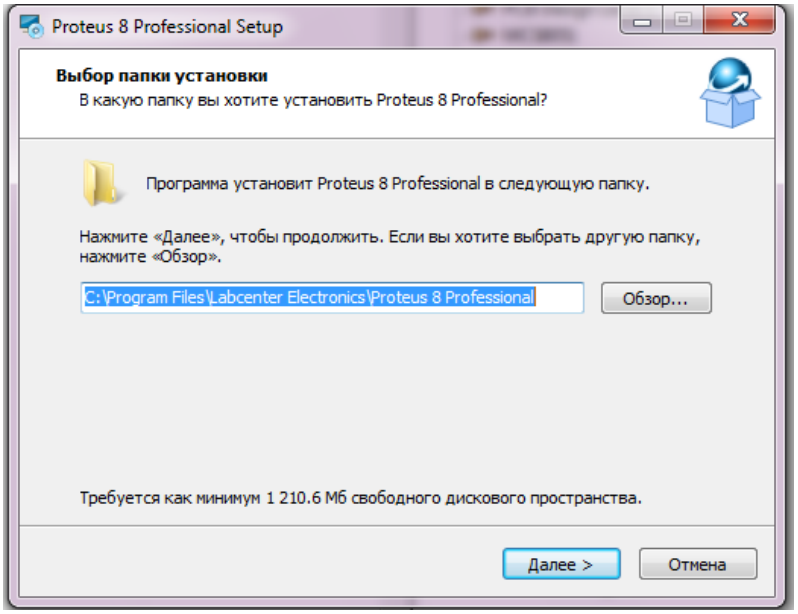


Рисунок 1.1 – Вікно привітання та вибір папки для програми

Слідуйте рекомендаціям, які дає постачальник вашої версії системи Proteus (рисунок 1.2).

Екран пропонує почати встановлення програми. Якщо всі налаштування обрані правильно, то встановлення Proteus 8.6 почнеться без помилок. Зазвичай встановлення триває близько 5 хвилин (рисунок 1.3).

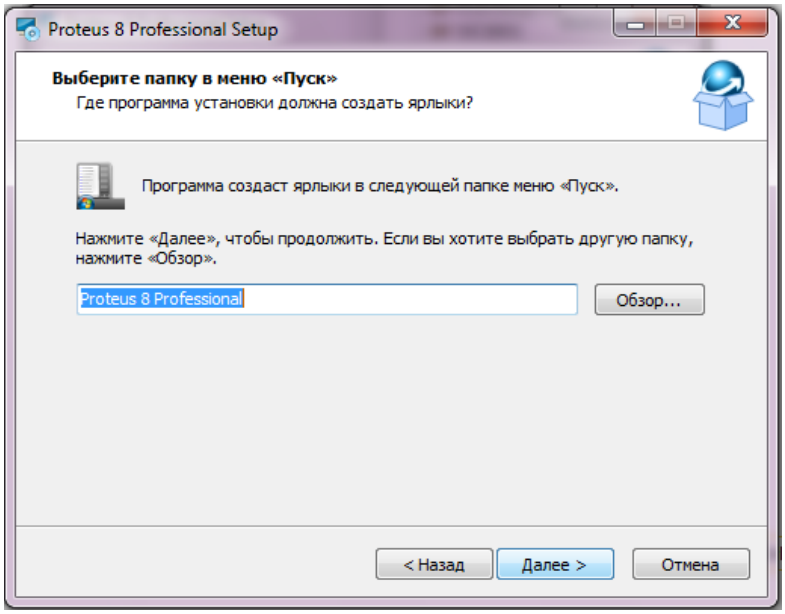


Рисунок 1.2 – Вибір локації у меню «Пуск»

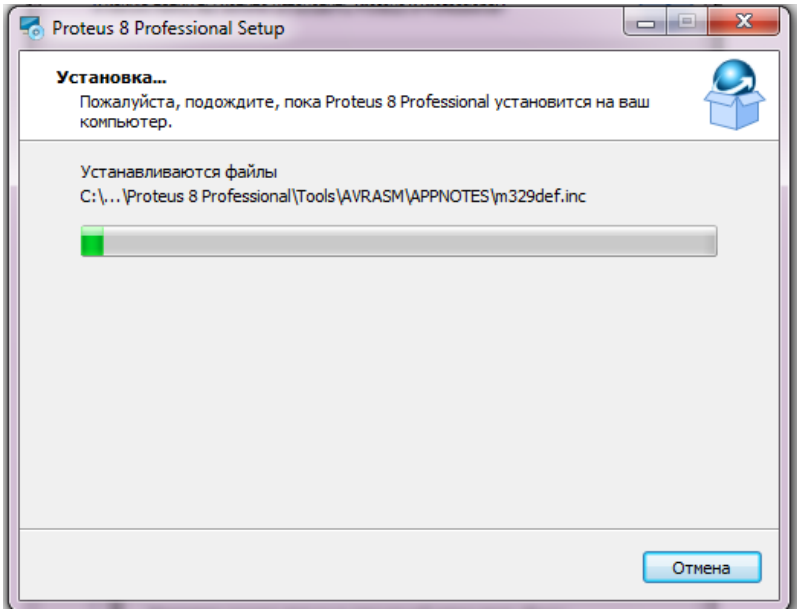


Рисунок 1.3 – Екран процесу встановлення

В кінці встановлення з’явиться вікно (рисунок 1.4) з кнопкою “Завершить”. Запустити програму можна з робочого стола або по кнопці “ПУСК” (рисунок 1.5) і вивід на головний екран програми PROTEUS 8.6 (рисунок 1.6).

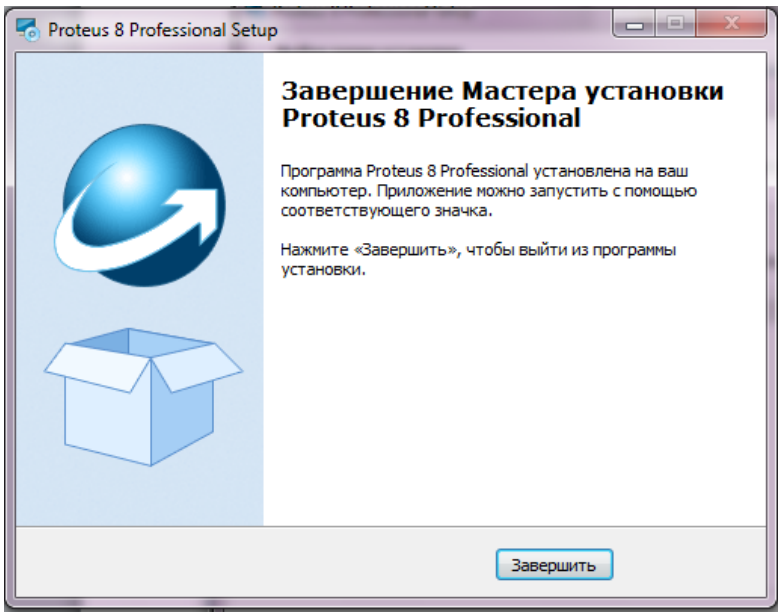


Рисунок 1.4 – Кінець інсталяції програми

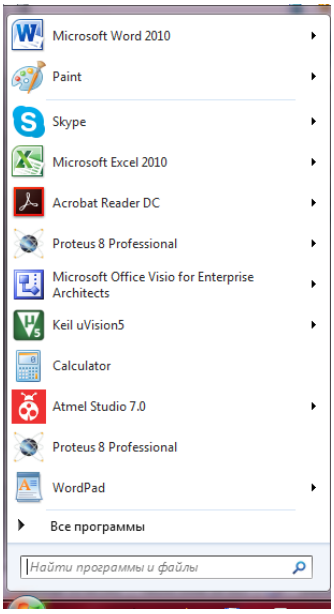


Рисунок 1.5 – Запуск програми по кнопці “ПУСК”

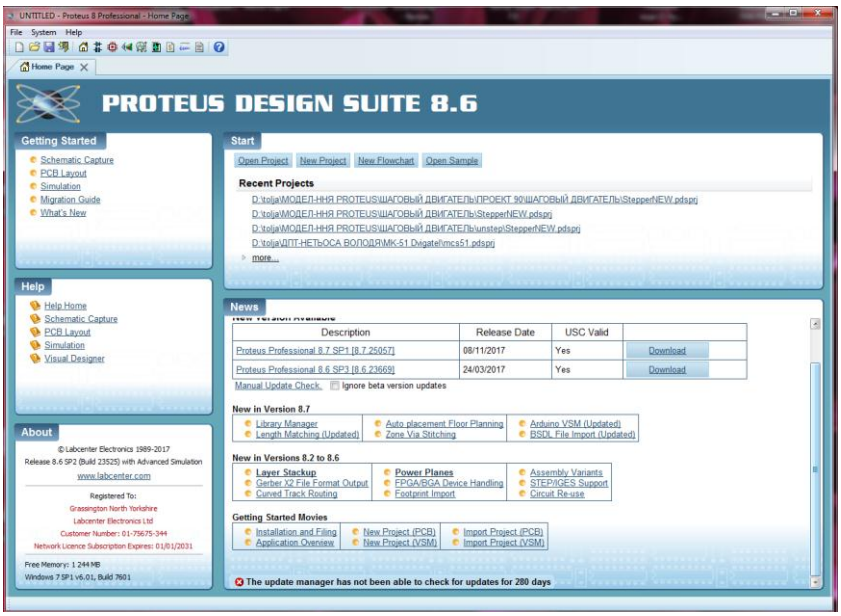


Рисунок 1.6 – Головний екран програми

1.2 Створення нового проекту

Щоб створити новий проект у Proteus 8.6 потрібно у стартовому вікні програми (рисунок 1.7) у лівому верхньому кутку вибрати пункт меню File – > New Project або застосувати скорочення клавіш (ctrl+N). Після чого з’явиться вікно налаштувань нового проекту, де можна вибрати назву нового проекту та локацію для збереження на комп’ютері.

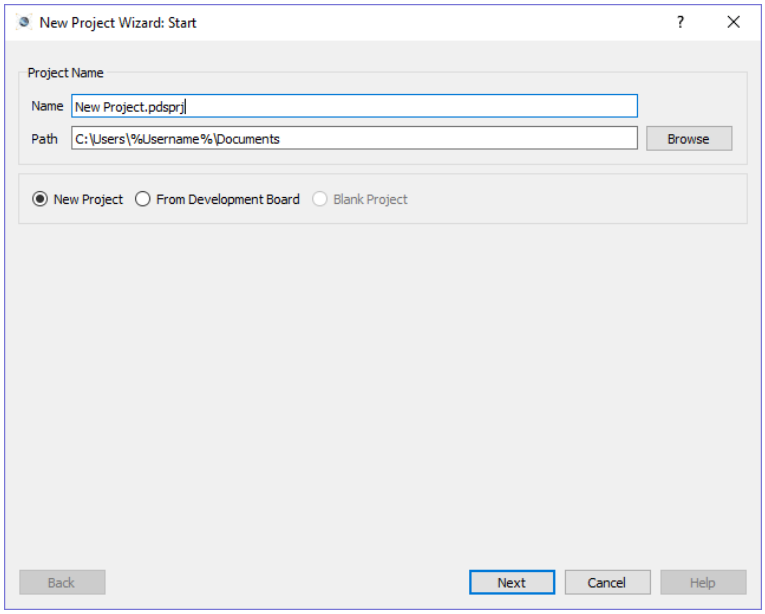


Рисунок 1.7 – Створення нового проекту у Proteus

Можна також обрати з уже запропонованих проектів, для цього треба обрати пункт From Development Board (рисунок 1.8).

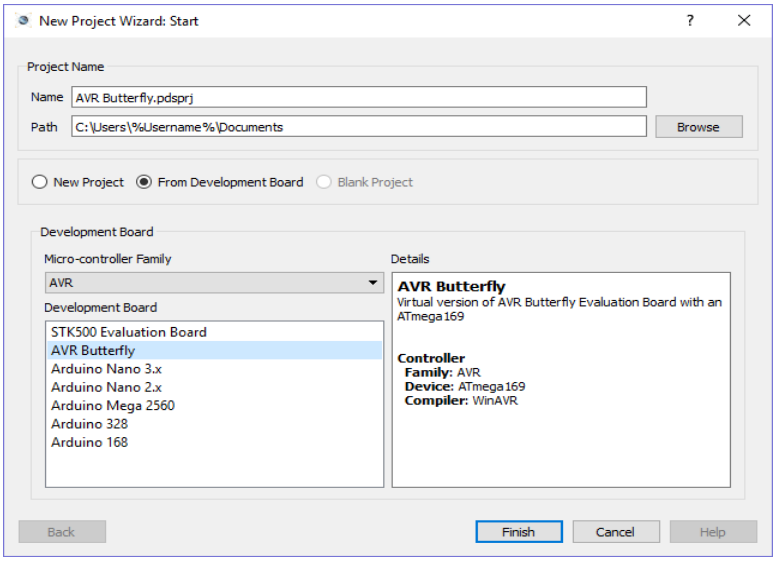


Рисунок1.8 – Заздалегідь налаштовані проекти

Обравши налаштування вручну ідемо на наступний крок – вибір розміру полотна для схеми (рисунок 1.9). Будемо обирати тип DEFAULT – за замовчуванням.

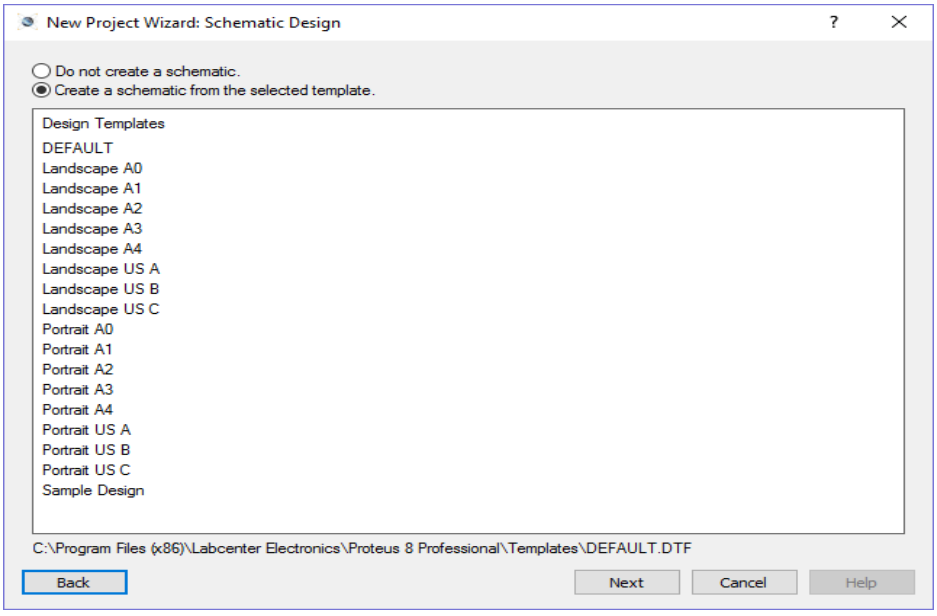


Рисунок 1.9 – Налаштування розмірів полотна схеми

В наступному вікні (рисунок 1.10) також оберемо тип DEFAULT.

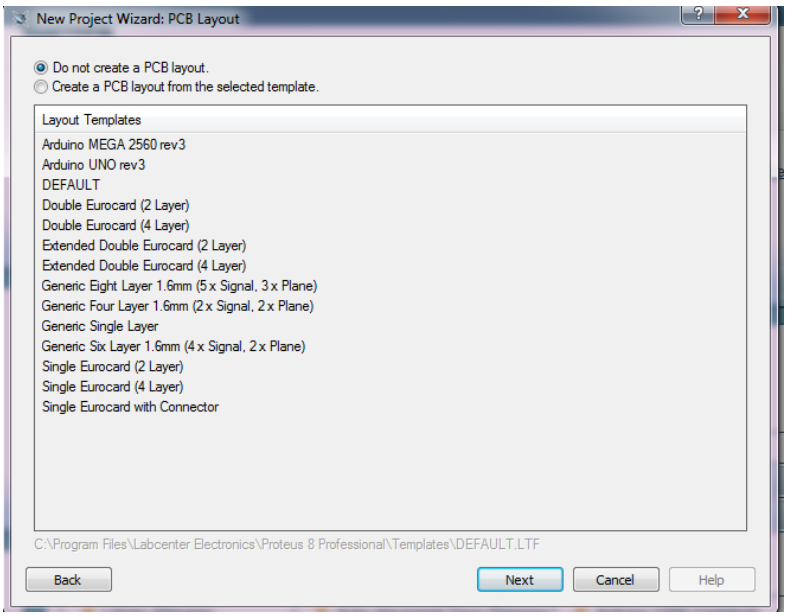


Рисунок 1.10– Вибір схеми

У наступному вікні (рисунок 1.11) можна обрати налаштування для створення проекту прошивки. А далі закінчуємо процес попереднього налаштування проекту.

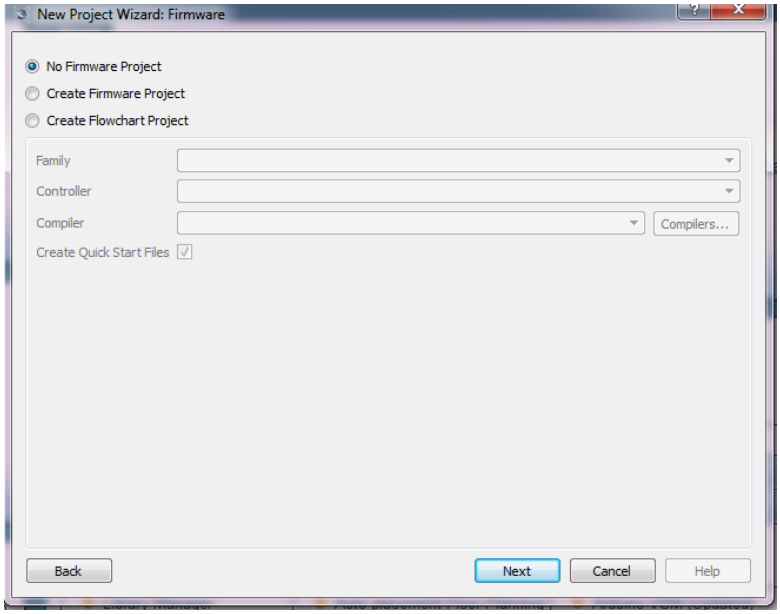


Рисунок 1.11 – Вибір налаштувань проекту прошивки

Оберемо перший пункт (без прошивки) та натиснемо кнопку NEXT (далі), після чого відкриється підсумкове вікно створення проекту (рисунок 1.12).

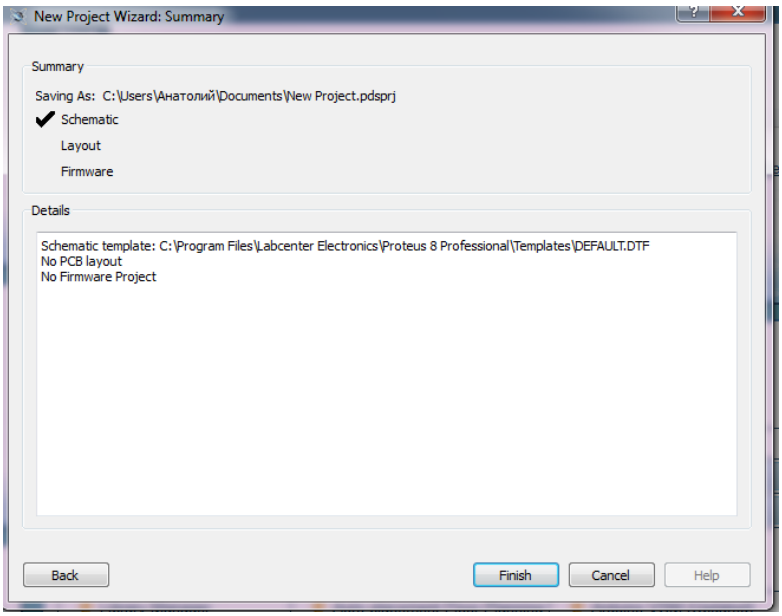


Рисунок 1.12 – Вікно з підсумками процесу налаштування

Врешті відкриється головний екран середовища розробки ISIS (рисунок 1.13)

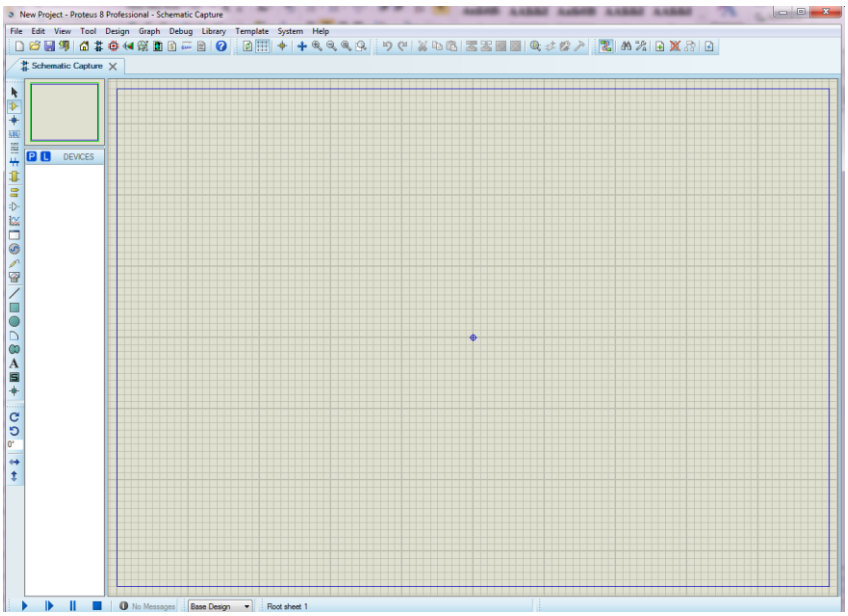


Рисунок 1.13 – Вигляд середовища ISIS після налаштувань

1.3 Відкриття існуючого проекту

Щоб відкрити вже існуючий ISIS проект для Proteus треба обрати пункт у меню зліва зверху File – > Open project або сполучення клавіш (ctrl+o).

Відобразиться вікно вибору файлу проекту з диска (рисунок 1.14). Файл має бути попередньо збережений у форматі .pdsprj.

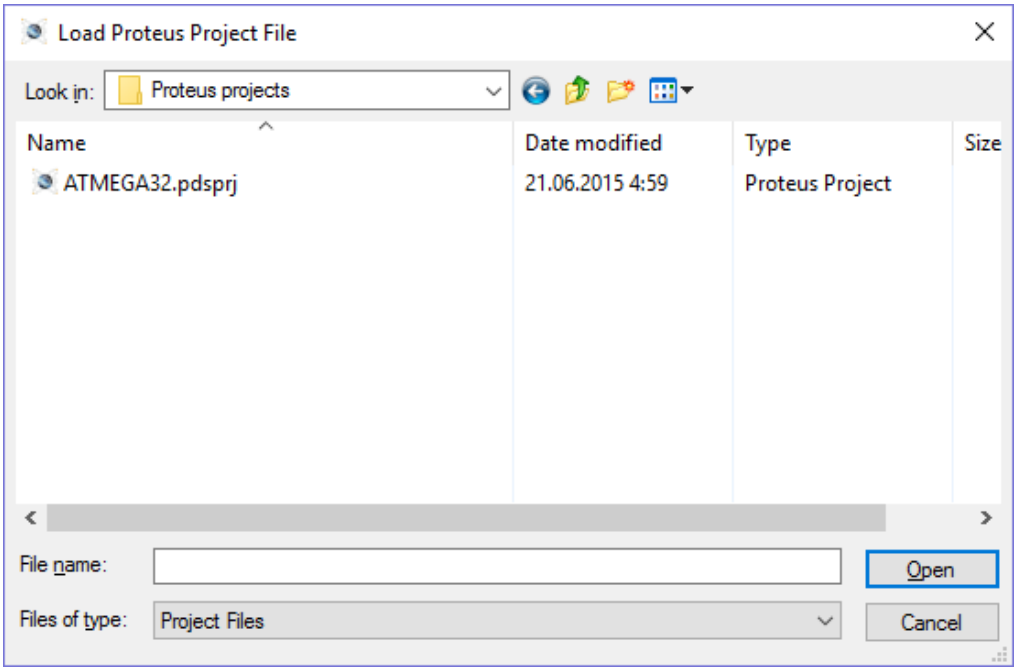


Рисунок 1.14 – Вікно вибору файлу

Приклад відкритого проекту наведено на рисунку 1.15.

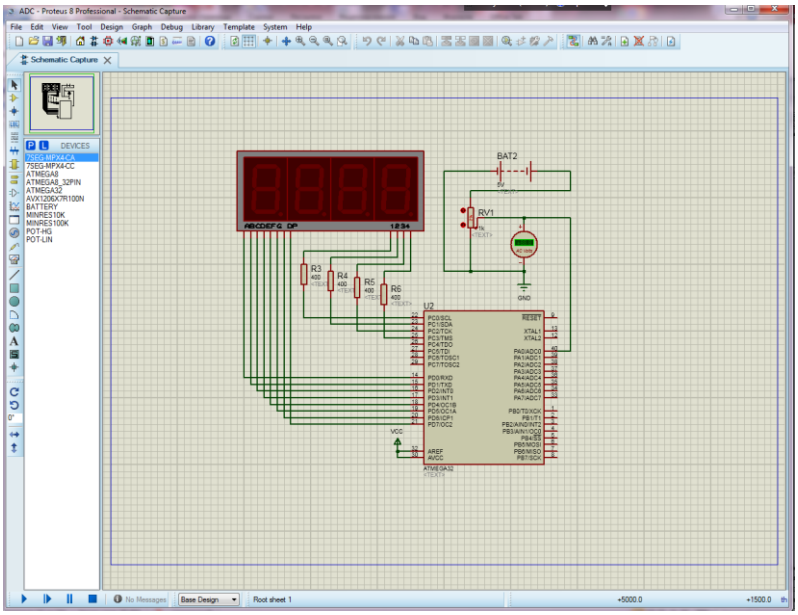


Рисунок 1.15 – Приклад відкритого проекту

1.4 Знайомство з інтерфейсом середовища ISIS

Це питання розглянемо на прикладі завантаженої моделі готової принципової схеми (рисунок 1.16).

Можна одразу побачити список компонентів, що були використані для побудови даної схеми, стрілки повороту та стрілки напрямів у меню посередині (рисунок 1.17) дозволяють більш точно рухати елемент на схемі.

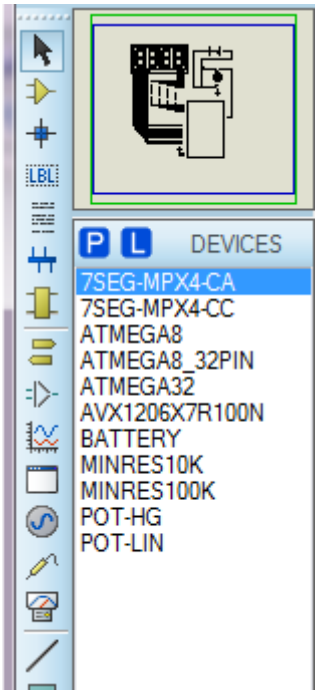


Рисунок 1.16 – Вид інтерфейсу бокового меню

Для запуску схеми використовується меню, що знаходиться внизу вікна ISIS (рисунок 1.17). Щоб розпочати виконання програми треба натиснути на синій трикутник, а щоб завершити – на квадрат.

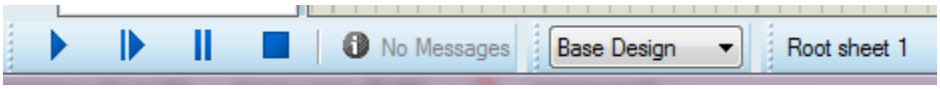


Рисунок 1.17 – Меню запуску моделювання

Меню на верхній панелі надає можливість обрати вигляд макету. Наприклад, обравши перший зліва на рисунку 1.18 пункт меню Schematic capture (схематичне зображення) ми перейдемо до стандартного вигляду схеми, який ми частіше за все використовуємо.

Рисунок 1.18 – Меню вибору вигляду

Наступний пункт PCB Layout – показує вигляд схеми з дотриманням масштабу, розмірів деталей та їх фізичних параметрів, а також показує розташуваня кріплень до мікросхеми.

1.5 Додавання моделей

Щоб додати будь–яку модель на схему потрібно слідувати крокам, що будуть описані далі (рисунок 1.19).

1. У головному вікні схематичного представлення мікросхеми треба натиснути правою кнопкою миші по вільній ділянці полотна. Відкриється невелике меню, з якого треба буде обрати перший пункт – Place (помістити).
2. У наступному меню потрібно обрати, який саме елемент ми хочемо помістити на схему. Обираємо варіант Component (компонент).
3. Третє меню дозволяє обрати недавно використані елементи. Ми ж оберемо останній пункт меню – From libraries (з бібліотек), щоб самостійно обрати потрібний елемент з бібліотеки.

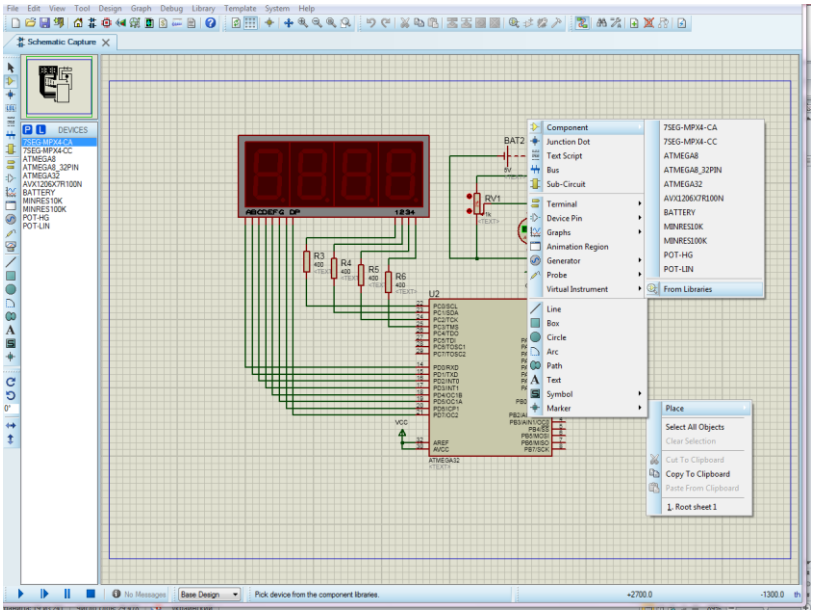


Рисунок 1.19 – Додавання моделі

4. У вікні, що з’явилося, впишемо у строку пошуку в лівому верхньому кутку вікна ключові слова: motor. Після чого буде знайдено усі елементи, що використовують це слово у своїй назві.

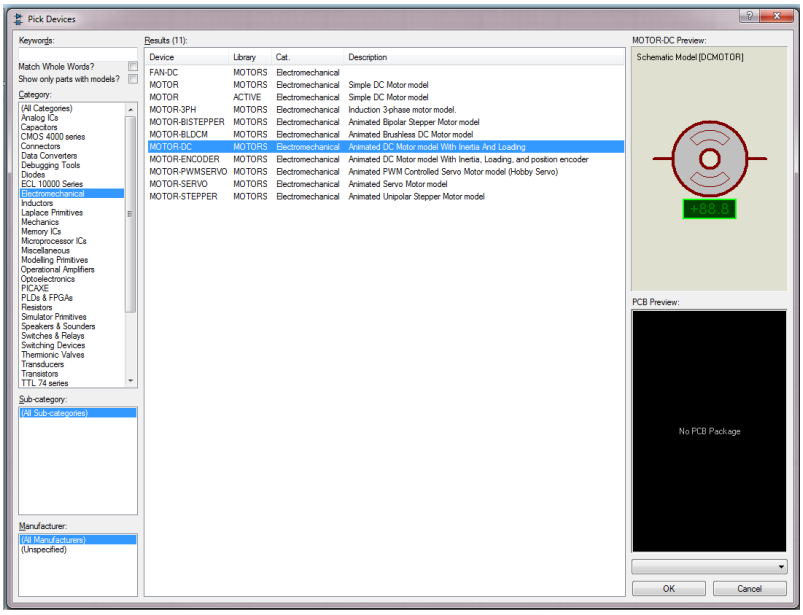


Рисунок 1.20 – Вікно вибору потрібного елемента

5. Обравши потрібний елемент (Motor–DC з бібліотеки Motors), (рисунок 1.20) натиснемо кнопку ОК та розмістимо елемент на схемі клацнувши по ній лівою кнопкою миші.

Нижче наведено приклади додавання світлодіода, двигуна постійного струму, осцилографа, вольтметра та крокового двигуна.

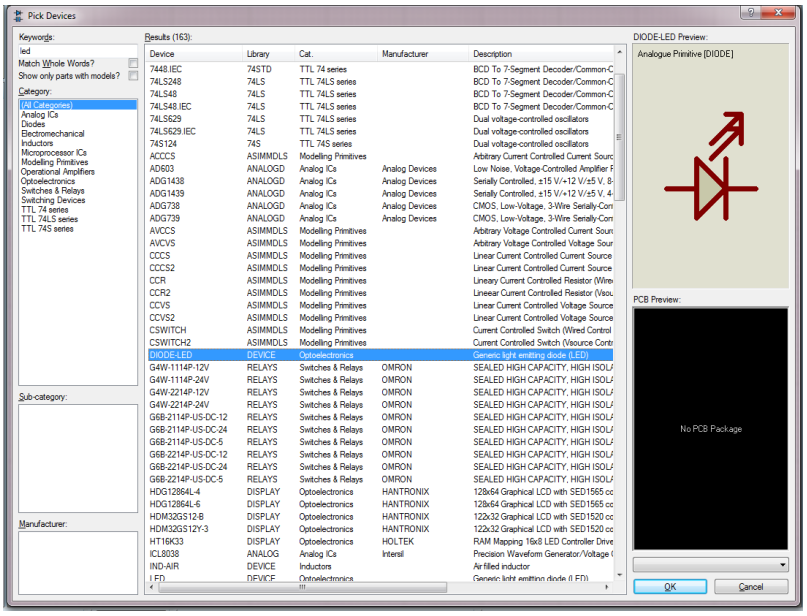


Рисунок 1.21 – Вибір потрібного елемента зі списку

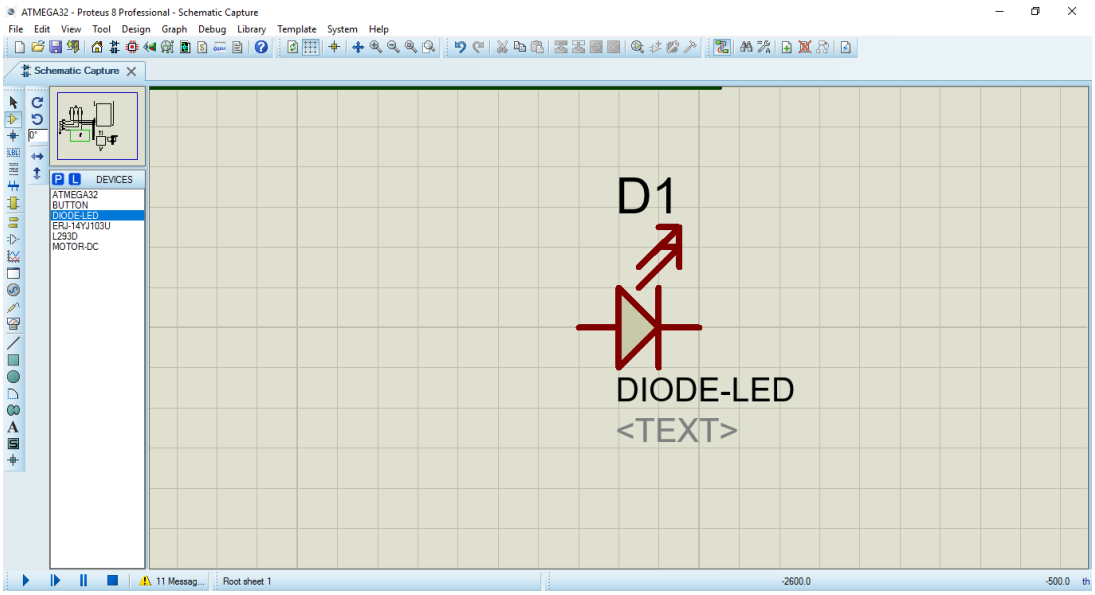


Рисунок 1.22 – Розміщення елемента на полотні

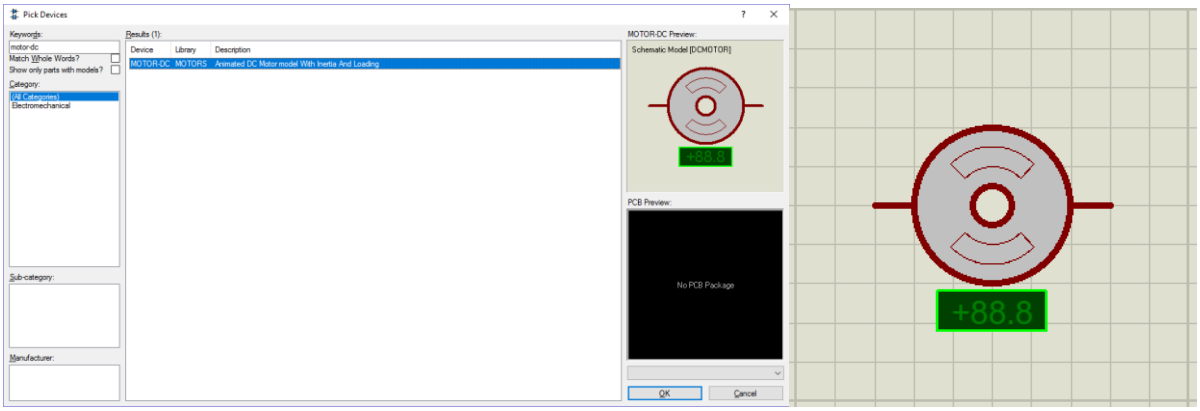


Рисунок 1.23 – Розміщення двигуна постійного струму

Щоб додати осцилограф та вольтметр потрібно переключитися у режим Instruments в меню на рисунок 1.24. Відкриється панель інструментів, з якої можна обрати потрібний нам.

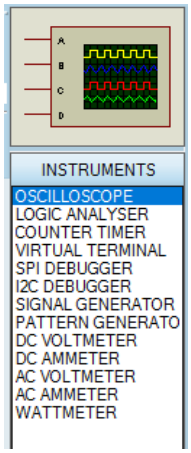


Рисунок 1.24 – Меню вибору інструментів

Додамо осцилограф з меню компонентів на схему цифрового вольтметра, яку виконано на мікроконтролері ATMEGA32 (рисунки 1.25, 1.38). Для цього після вибору осцилографа в меню натиснемо двічі по порожній ділянці на схемі, в цьому місці одразу з'явиться осцилограф (рисунок 1.38).

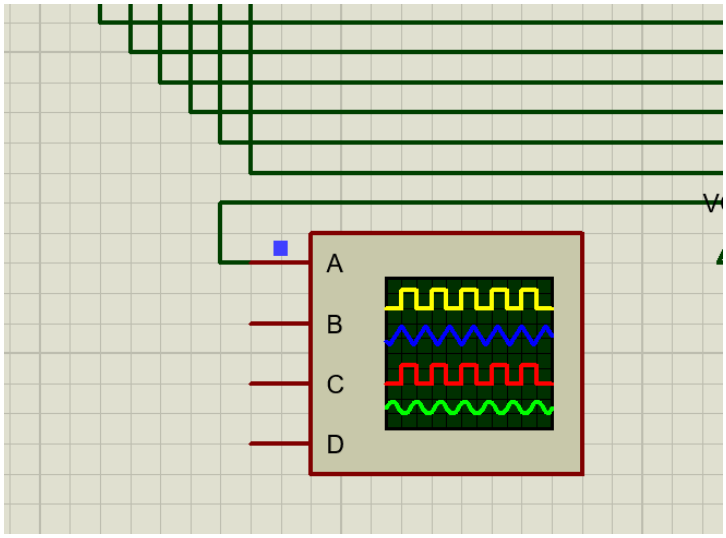


Рисунок 1.25 – Осцилограф на схемі

Під'єднаємо осцилограф до схеми. Для цього треба клікнути лівою кнопкою миші по одному з виходів осцилографа, має з'явитися лінія, що слідує за курсором миші (рисунок 1.26). Під'єднаємо інший кінець цього дроту до ділянки кола, яку нам треба прослухати. Для цього клацнемо ще раз по ділянці, до якої потрібно під'єднатися. З'явиться лінія між осцилографом та ділянкою кола.

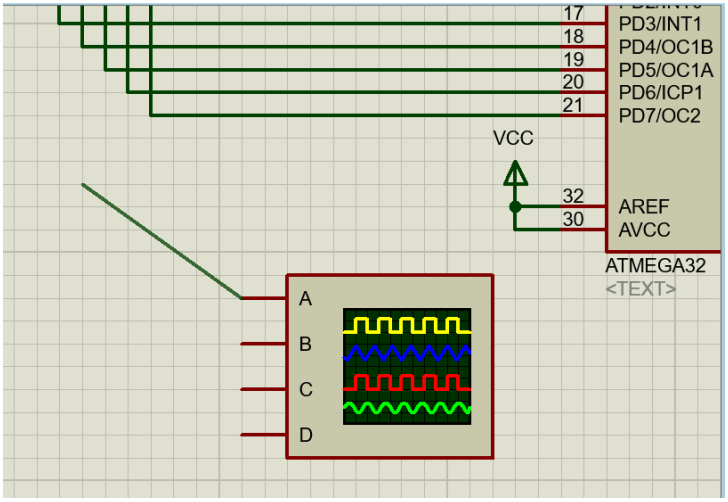


Рисунок 1.26 – Приклад приєднання осцилографа

Щоб відкрити вікно осцилографа, треба запустити схему. Для цього треба натиснути на синій трикутник в нижній частині екрану (рисунок 1.27). Схема запуститься, і трикутник змінить колір на зелений.

Рисунок 1.27 – Увімкнення системи

Далі треба у пункті меня Debug на верхній панелі у головному вікні програми обрати зі списку компонентів системи осцилограф і натиснути на його назву (рисунок 1.28). Одразу ж відкриється вікно осцилографа з даними (рисунок 1.29).

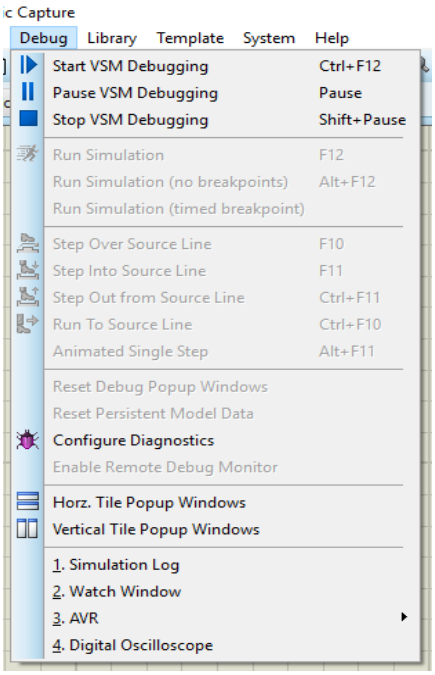


Рисунок 1.28 – Список компонентів у системі

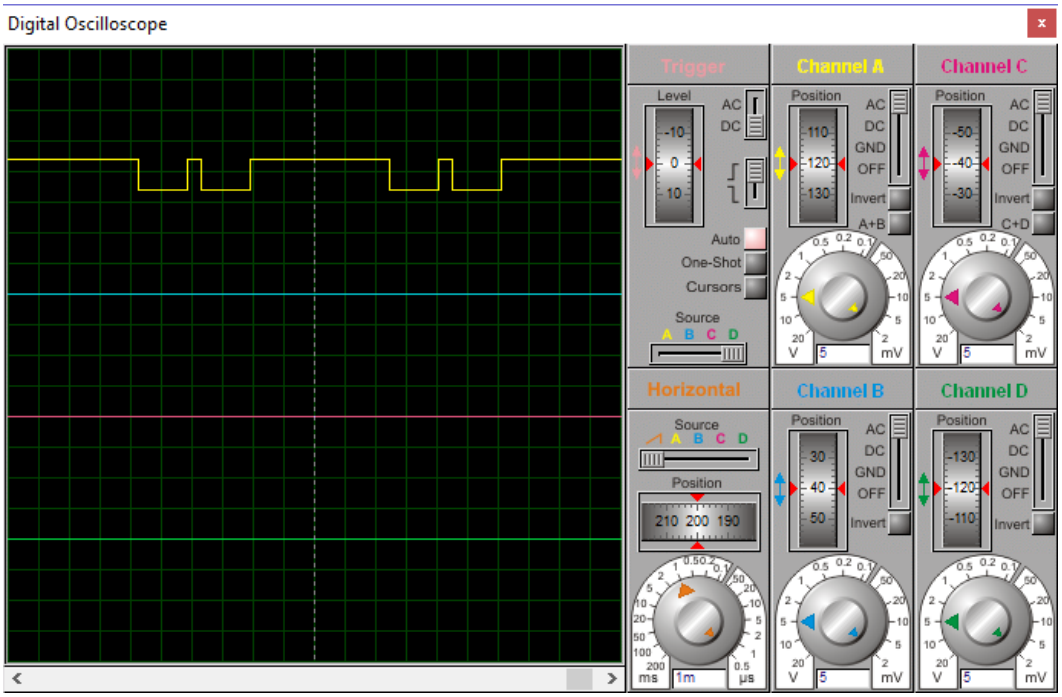


Рисунок1.29 – Вікно осцилографа

Аналогічно додамо на схему вольтметр. Оберемо у меню зліва компонент AC VOLTMETER і розмістимо його на схемі (рисунок 1.30).

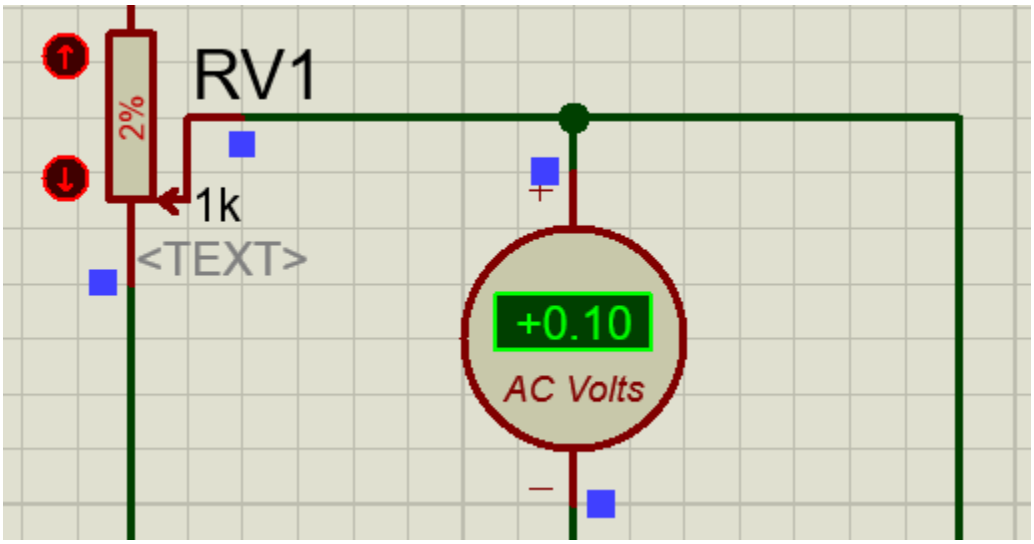


Рисунок 1.30– Вольтметр, який розміщено на схемі

Додамо на схему семисегментний катодний дисплей з маркуванням 7SEG–MPX1–CA. Для цього повторимо процес, який описано в пункті 1.5, але на цей раз записавши у строку пошуку назву даного пристрою по маркуванню.

Для додаванная семисегментного індикатора на схему потрібно виконати аналогічні кроки, як і при доданні попередніх компонентів. Знайдемо у бібліотеці компонентів червоний чотирьохцифровий індикатор за ключовим словом 7SEG–MPX4–CA (рисунок 1.31).

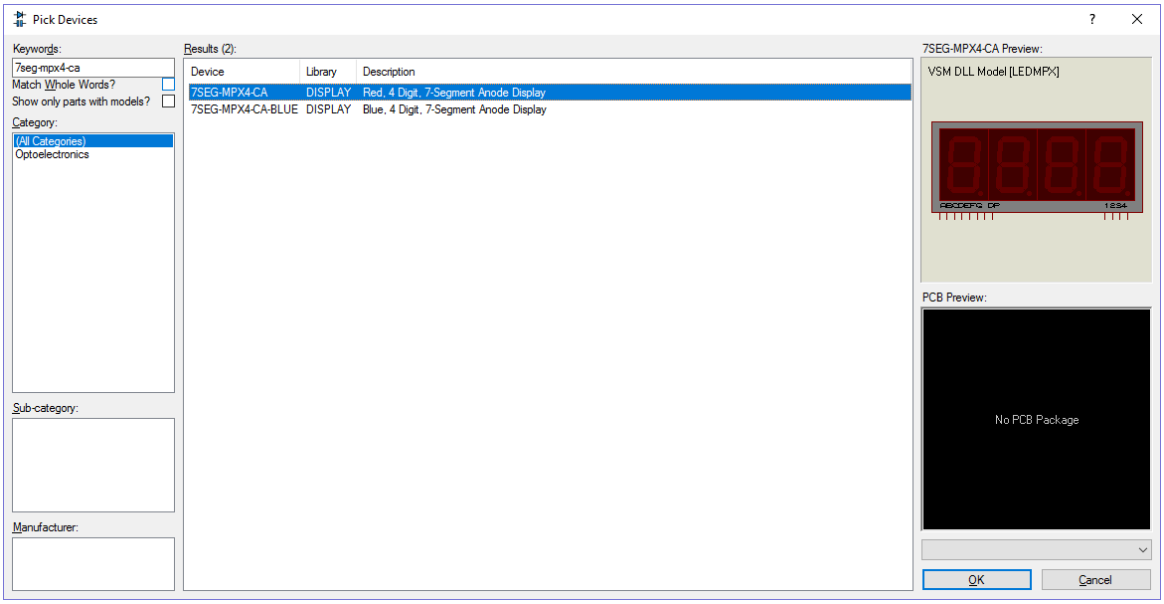


Рисунок 1.31 – Вікно додавання семисегментного індикатора

Виберемо місце на схемі та розмістимо індикатор, натиснувши ліву кнопку миші (рисунок 1.32).

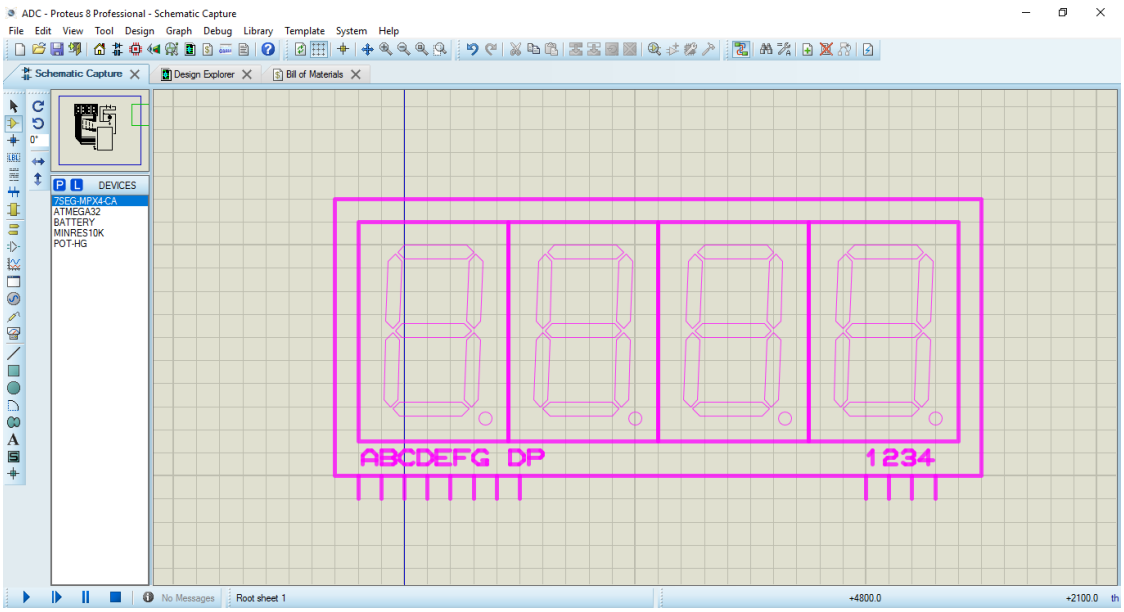


Рисунок 1.32 – Розміщення індикатора на схемі

Після такої операції на полотні схеми з’явиться новий елемент, який згодом можна буде під’єднати до схеми.

Додавання резисторів на схему відбувається за аналогічні кроки: спочатку ми шукаємо потрібний елемент у бібліотеці компонентів у бібліотеці RESISTORS. В даній бібліотеці за ключовим словом MINRES присутні різноманітні резистори з різними характеристиками (рисунок 1.33).

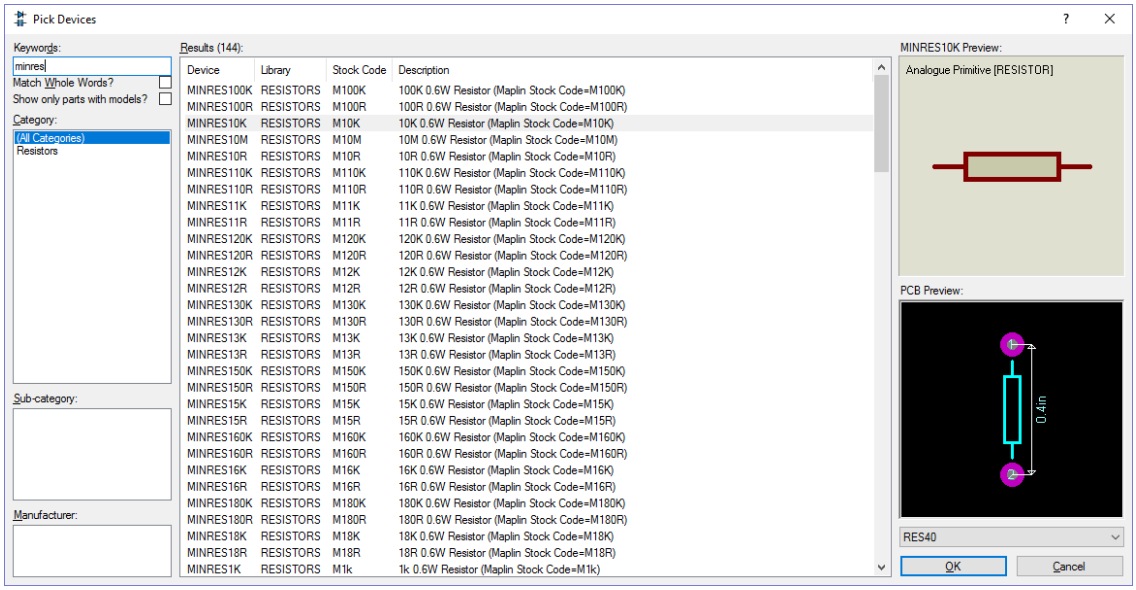


Рисунок 1.33 – Вікно вибору потрібного резистора

А далі потрібний резистор розміщається на схему. За бажанням можна змінити опір резистора. Для цього по елементу потрібно двічі натиснути лівою кнопкою миші, відкриється вікно як на рисунок 1.34. Після зміни опору у полі Resistance тиснимо клавішу Enter, після чого у резистора зміниться опір.

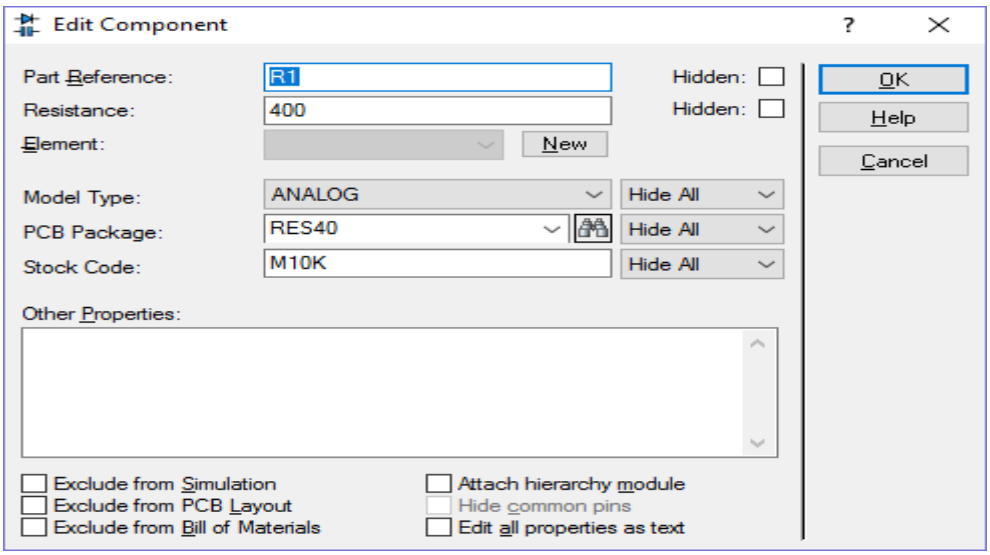


Рисунок 1.34 – Вікно налаштувань резистора

Додамо до схеми кроковий двигун. Вибираємо “Electromechanical” категорію. Для створення уніполярного крокового двигуна обираємо MOTOR–STEPPER (рисунок 1.35), для біполярного крокового двигуна – MOTOR–BISTEPPER (рисунок 1.36).

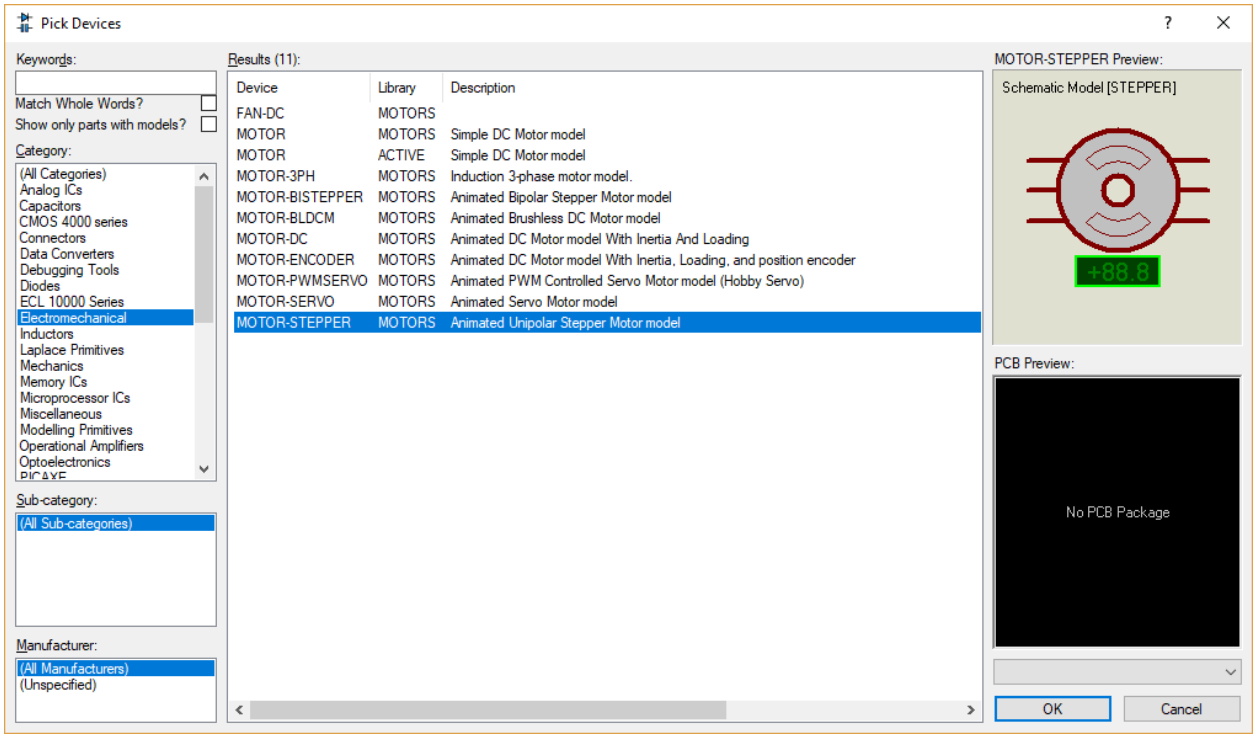


Рисунок 1.35 – Уніполярний кроковий двигун (STEPPER)

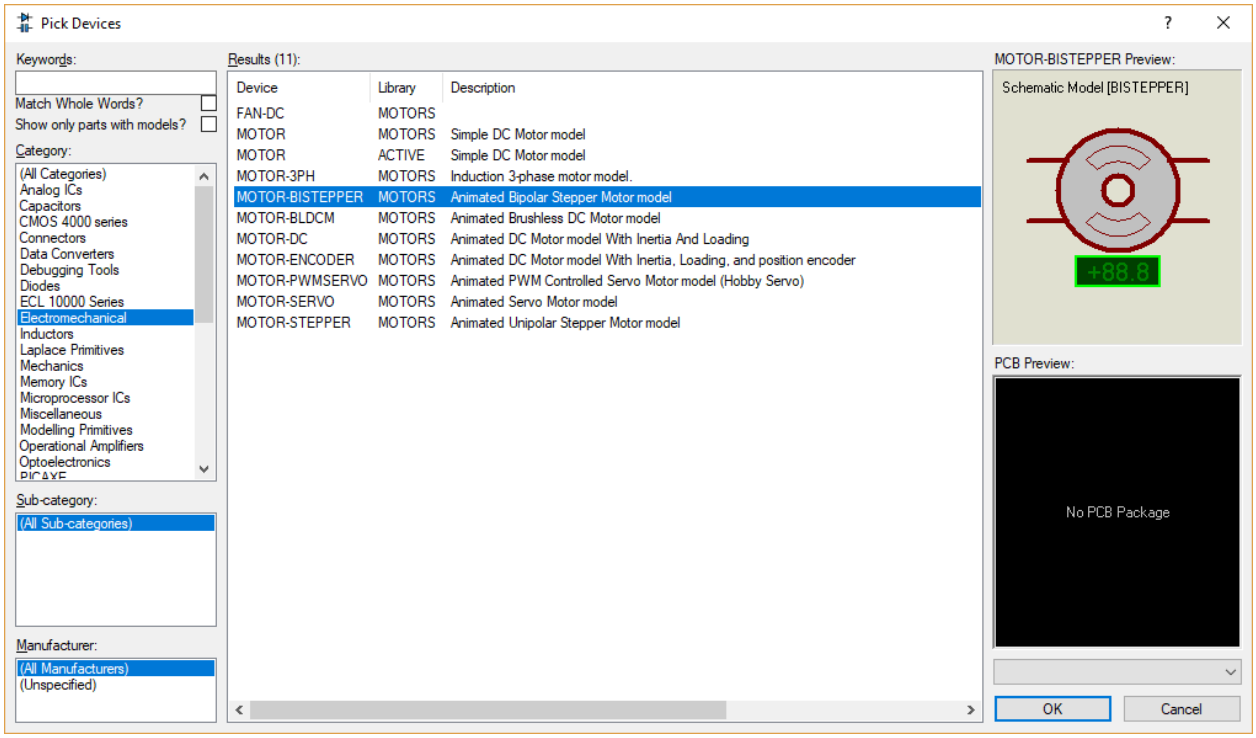


Рисунок 1.36 – Біполярний кроковий двигун (BISTEPPER)

1.6 Додавання мікроконтролера

Щоб додати у проект мікроконтролер з архітектурою AVR, який планується програмувати, наприклад, ATMEGA32, оберіть режим роботи “Component mode” (рисунок 1.37) у меню, що знаходиться зліва на головному екрані середовища ISIS, та натисніть правою кнопкою миші на вільній частині робочого простору проекту. У меню, що з’явилося після натискання на кнопку миші, оберіть Place –> Component –> From Libraries (рисунок 1.38).

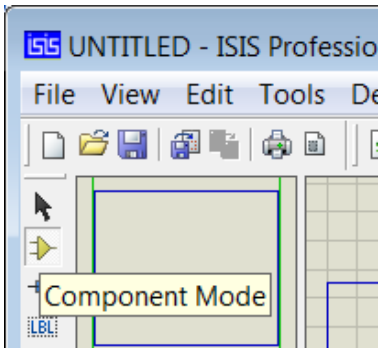


Рисунок 1.37 – Перехід у режим Component Mode

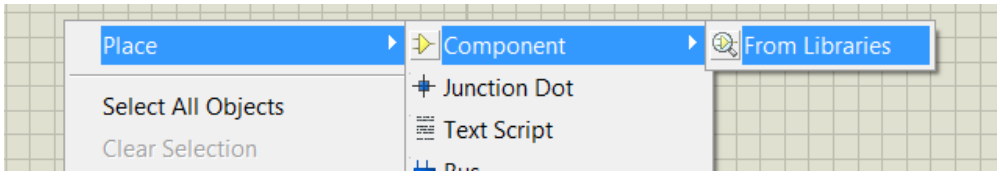


Рисунок 1.38 – Меню додавання нового компоненту

У меню, що відкрилося, введіть ключове слово пошуку “AVR” у верхньому лівому кутку вікна, оберіть бажаний AVR–сумісний МК та натисніть “OK” (рисунок1.39). Далі треба лівою кнопкою миші розташувати мікроконтролер на полотні, двічі натиснувши на ліву кнопку.

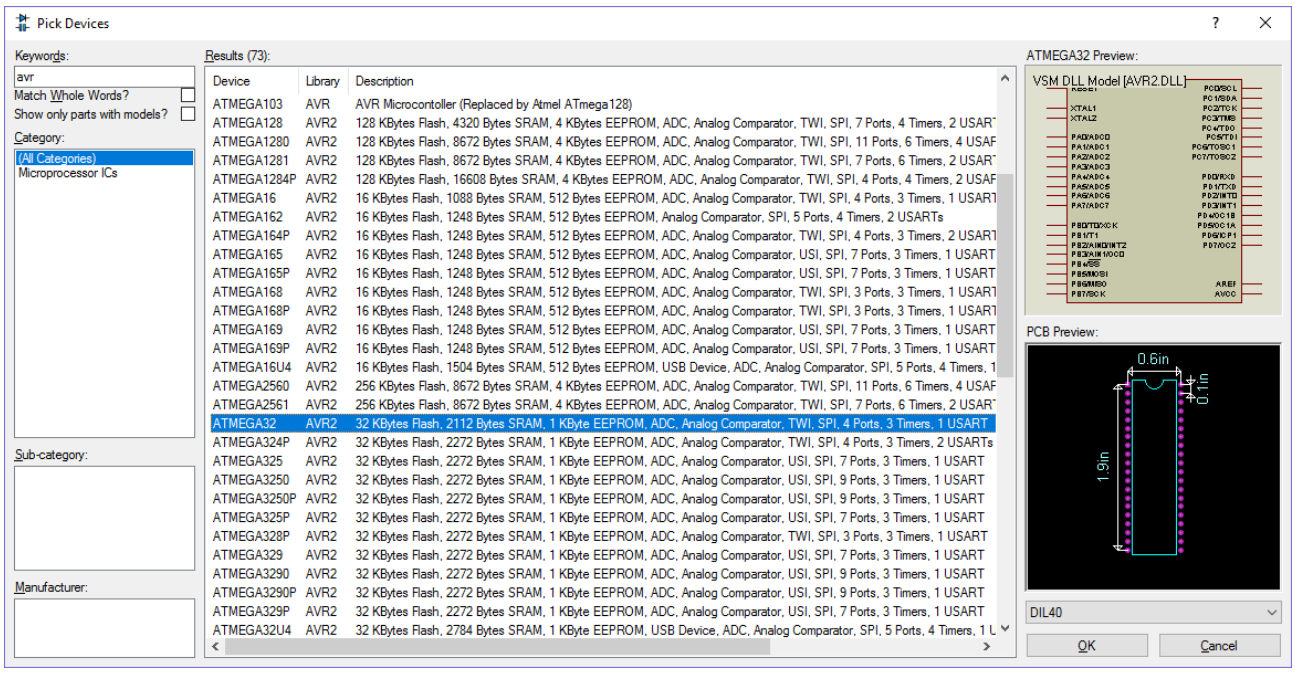


Рисунок 1.39 – Вибір мікроконтролера

1.7 Приклад схеми цифрового вольтметра та його побудова

З уже наведених раніше у прикладах компонентів пропонується побудувати схему моделі цифрового вольтметра, яку зображено на рисунок 1.40.

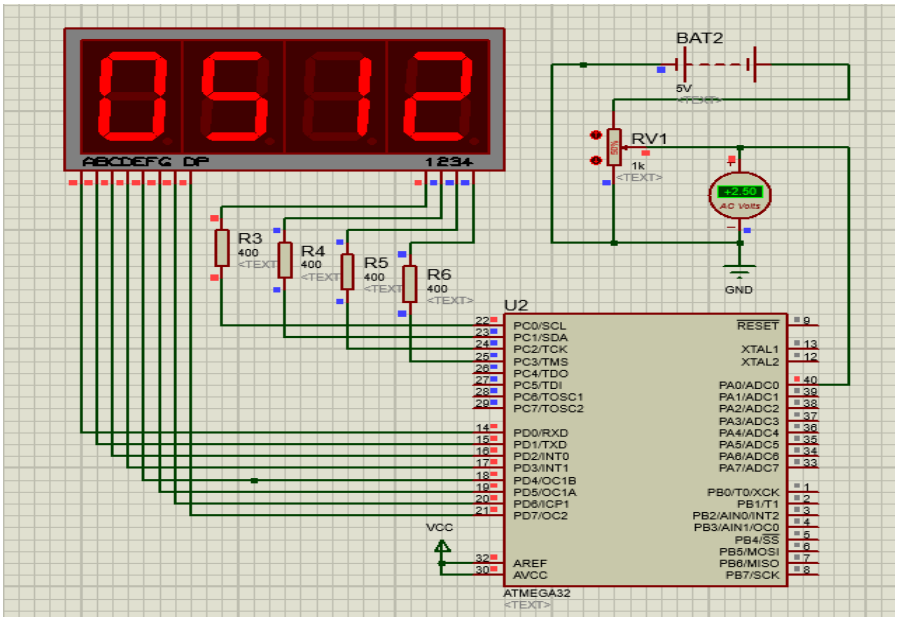


Рисунок 1.40 – Приклад схеми цифрового вольтметра

Роботу цієї схеми буде розглянуто нижче у розділах 3 та 7. Нижче розглядається побудова цієї схеми для її моделювання у PROTEUS.

Почнемо побудову схеми з додаваня до неї мікроконтролера. Для цього оберемо в бібліотеці компонентів МК ATMEGA32. Перший у списку знайдених контролерів нам підходить – це AVR–контролер, що знаходиться у бібліотеці AVR2 (рисунок 1.41).

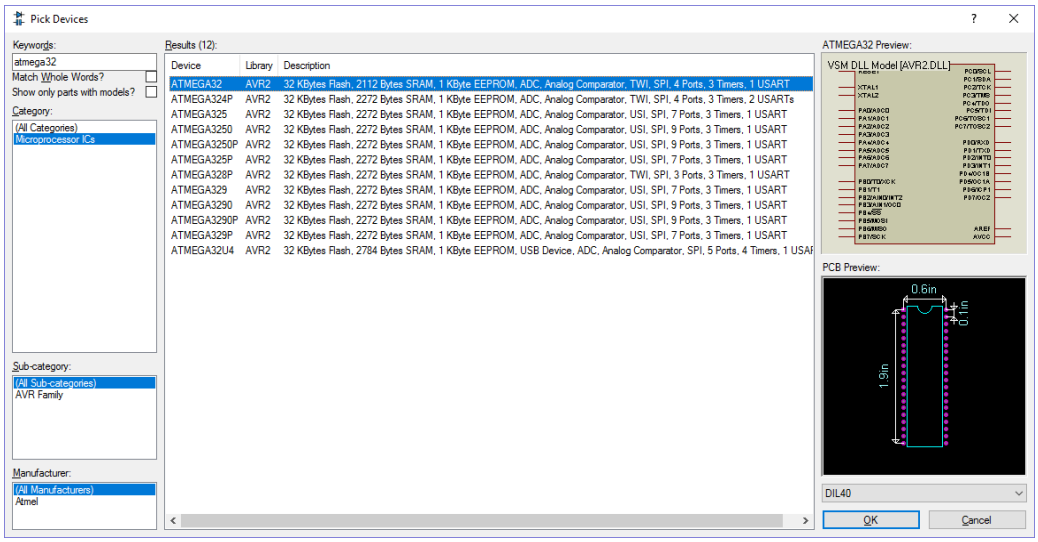


Рисунок 1.41 – Пошук МК у бібліотеці компонентів

Щойно доданий елемент схеми матиме вигляд як на рисунку 1.42.

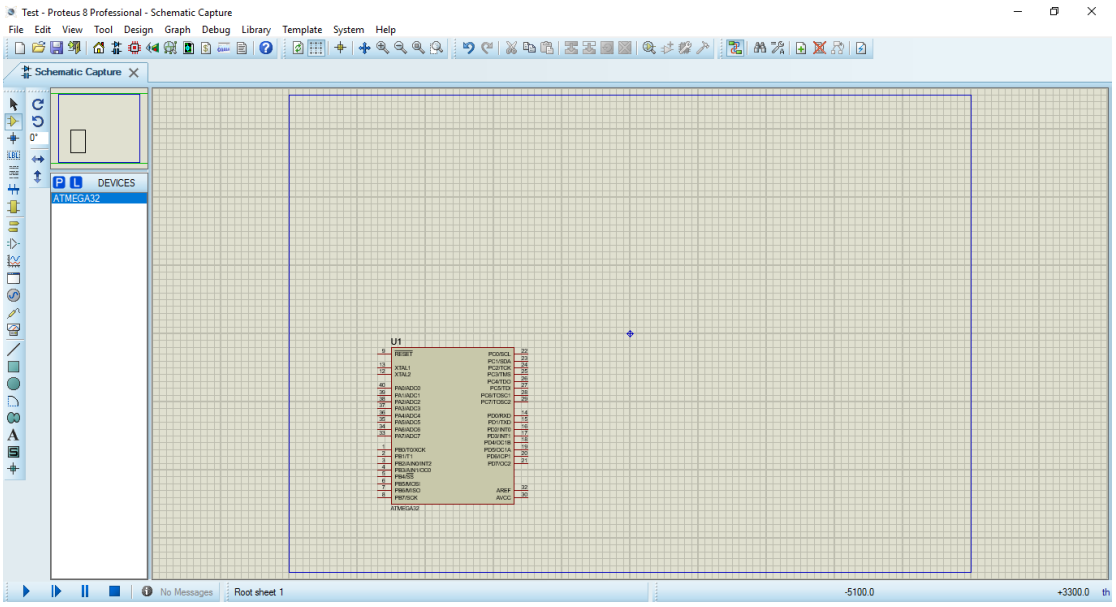


Рисунок 1.42 – Щойно доданий МК на схему

Оберемо зі списку семисегментний індикатор. Детальніше цей процес був розглянутий у попередньому підрозділі. Додавши індиатор на схему,

з’єднаємо його з мікроконтролером, що вже міститься на схемі (рисунок 1.43).

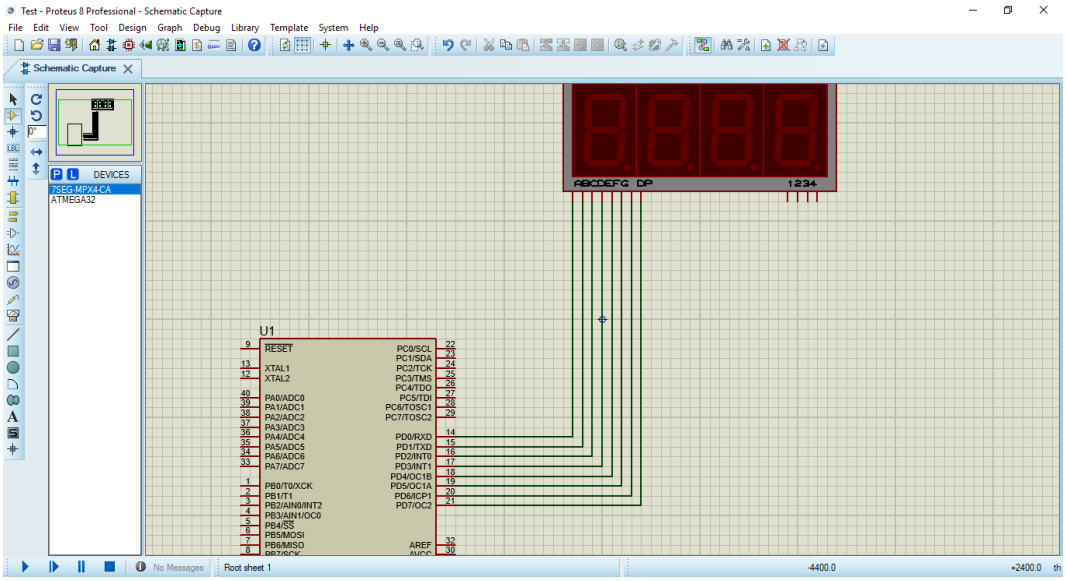


Рисунок 1.43 – З’єднання МК з індикатором

Далі на схему потрібно додати чотири резистори, як це вказано в попередньому підрозділі. Оберемо з бібліотеки резистори MINRES10K та змінимо їх опір на 400 Ом кожен. Далі з’єднаємо їх з семисегментним індикатором наступним чином, як це показано на рисунок 1.44.

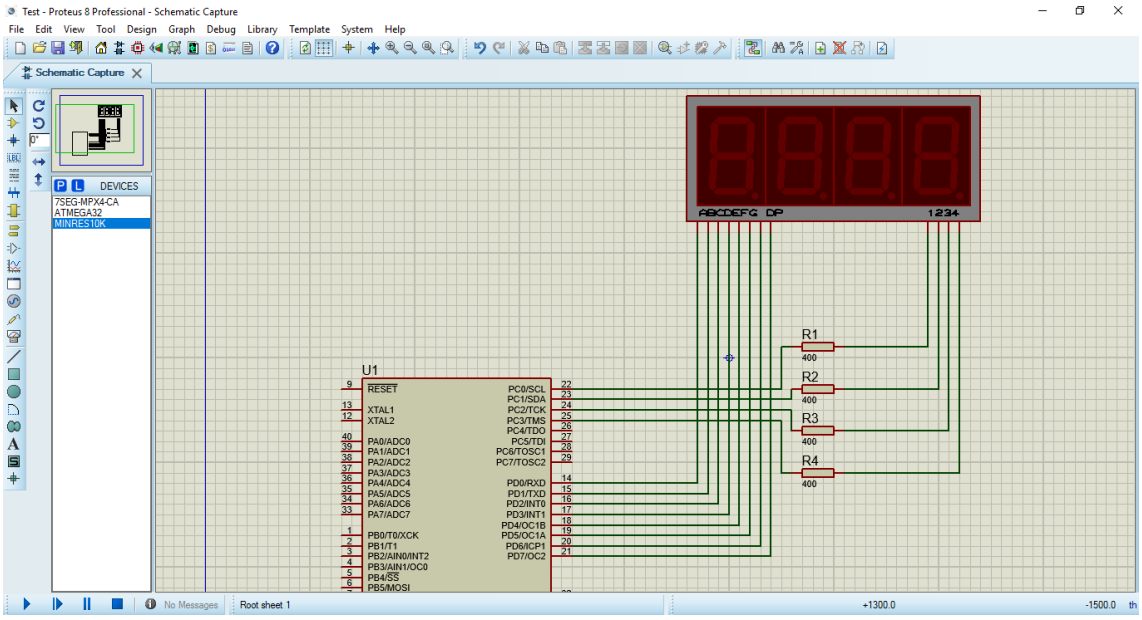


Рисунок 1.44 – Під’єднання резисторів

Додамо на схему потенціометр, знайшовши його у бібліотеці компонентів за ключовим словом POT–HG (рисунок 1.45).

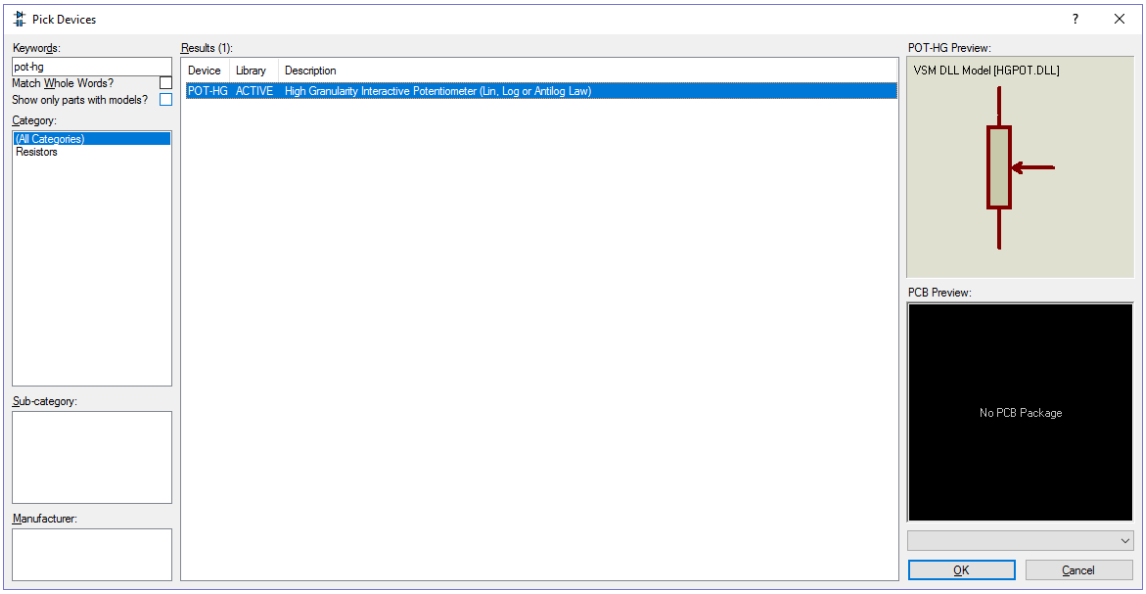


Рисунок 1.45 – Додавання потенціометра

За ключовим словом BATTERY знайдемо блок батарей і додамо їх на схему. Натиснемо двічі по даному блоку та у полі Voltage змінимо напругу елемента живлення до 5V. Розмістимо елементи на схемі, як показано на рисунок 1.46.

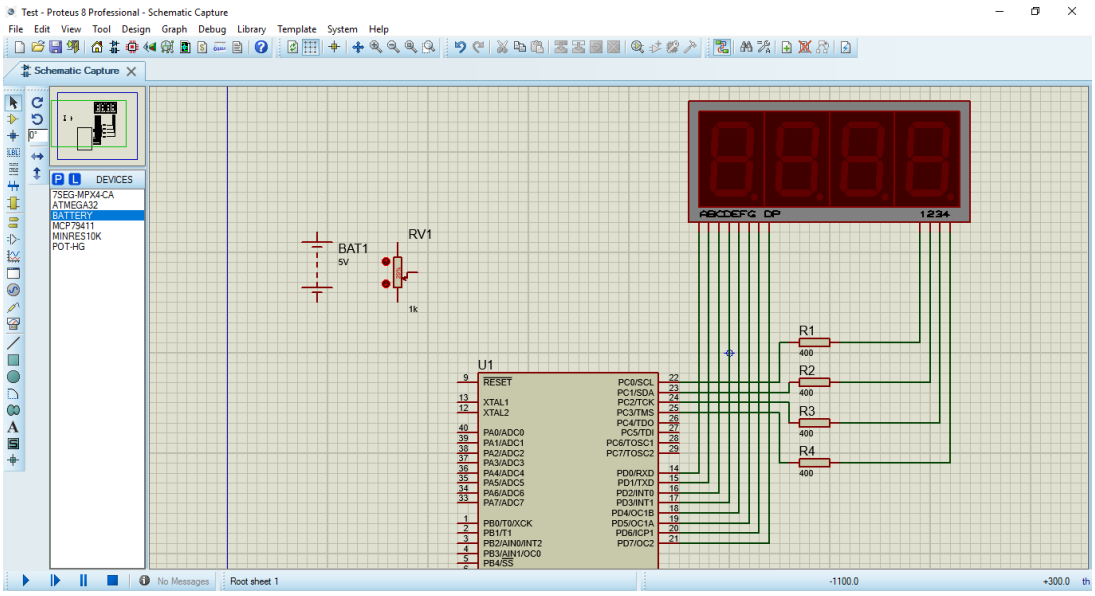


Рисунок 1.46 – Розташування батарей та потенціометра

Нарешті додамо на схему звичайний вольтметр, виконавши кроки, як в попередньому підрозділі. З’єднаємо елементи та мікроконтролер (рисунок 1.47).

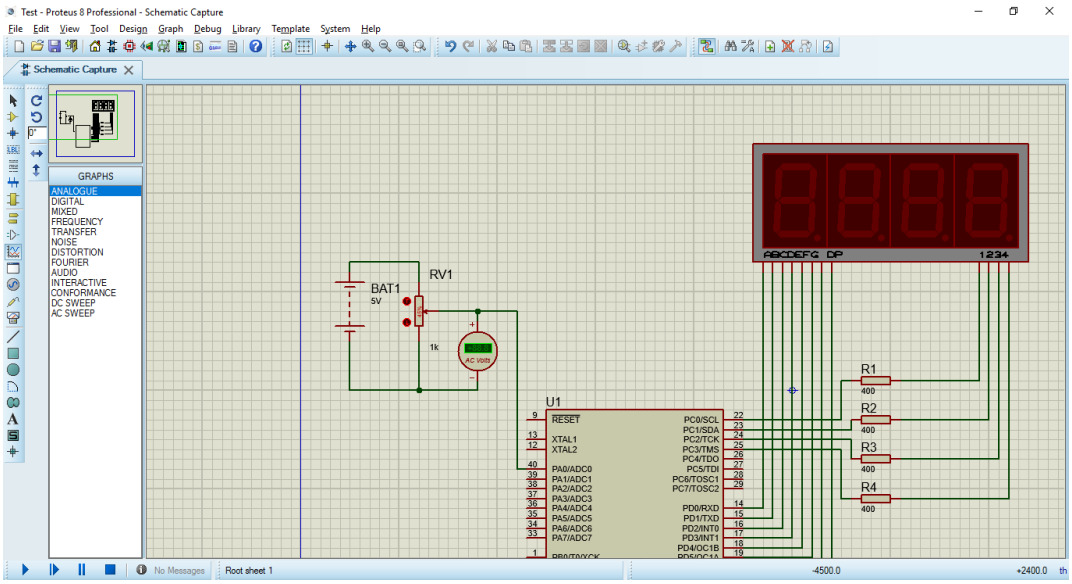


Рисунок 1.47 – Схема з’єднання компонентів

Важливо тепер підключити заземлення до схеми. Для цього у боковому меню вікна розробки потрібно обрати підпункт Terminals mode. У цьому меню обираємо компонент GROUND, натискаємо лівою кнопкою миші двічі по місцю схеми, яке потрібно заземлити. В тому місці з’явиться значок заземлення (рисунок 1.48).

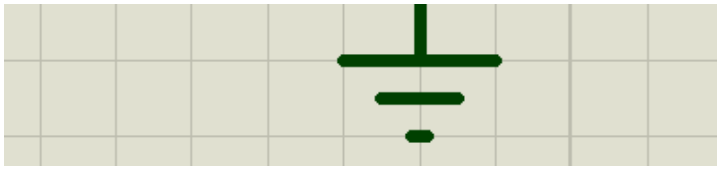


Рисунок 1.48 – Значок заземлення на схемі

З того ж меню потрібно обрати у меню зліва пункт POWER та додати даний елемент у місце, показане на рисунок 1.49.

Заключним кроком буде процес завантаження .hex файла у пам’ять мікроконтролера для його подальшого виконання. Даний процес описано у наступному підрозділі.

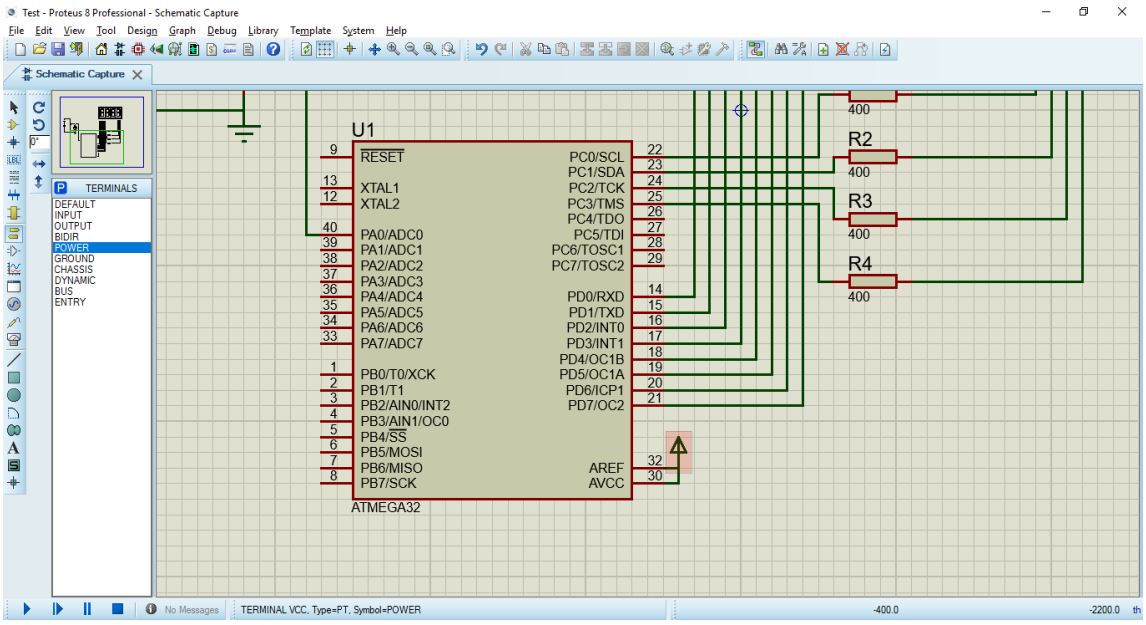


Рисунок 1.49 – Місце під’єднання живлення

1.8 Завантаження коду в пам’ять мікроконтролера

Наступна послідовність дій та ілюстрації демонструють процес завантаження вихідного коду в пам’ять мікроконтролера.

1. Створити новий проект у середовищі ISIS та зберегти його.
2. Додати у проект мікроконтролер з архітектурою AVR, який планується програмувати, наприклад, ATMEGA32.
3. Щоб завантажити вже скомпільований файл з програмою в пам’ять мікроконтролера, потрібно двічі натиснути лівою кнопкою миші по МК, після чого відкриється вікно властивостей (рисунок 1.50).
4. Далі потрібно натиснути на зображення файлів (виділено червоним на рисунку 1.50), після чого відкриється вікно вибору файлу з комп’ютера (рисунок1.51). В даному вікні можна знайти та обрати файл у форматі HEX, OBJ або ELF.
5. Після вибору файлу треба зберегти налаштування. Після цього мікроконтролер буде в змозі виконувати завантажену програму.

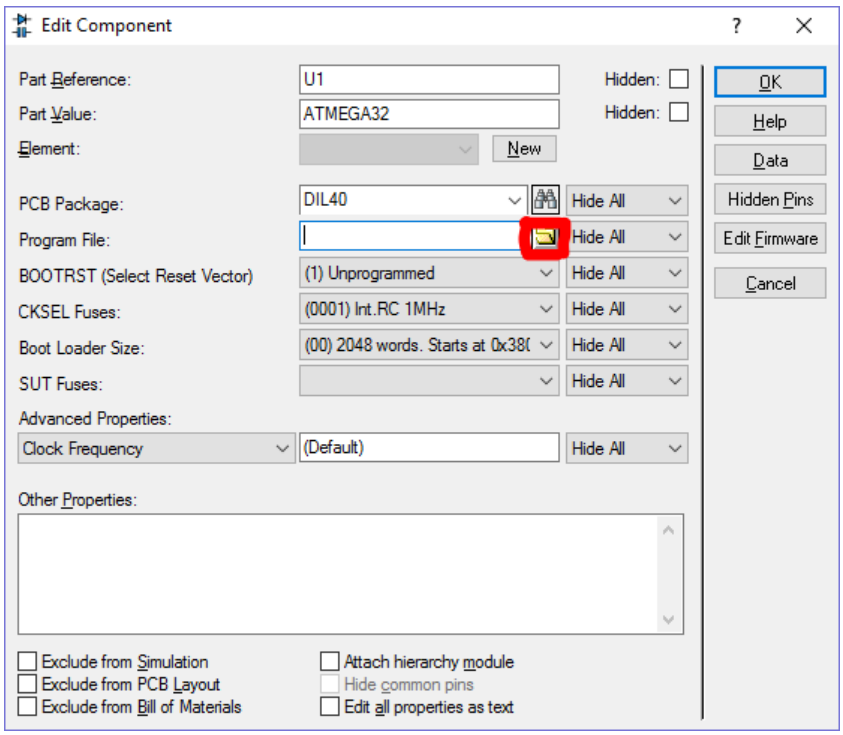


Рисунок 1.50 – Вікно налаштувань мікроконтролера

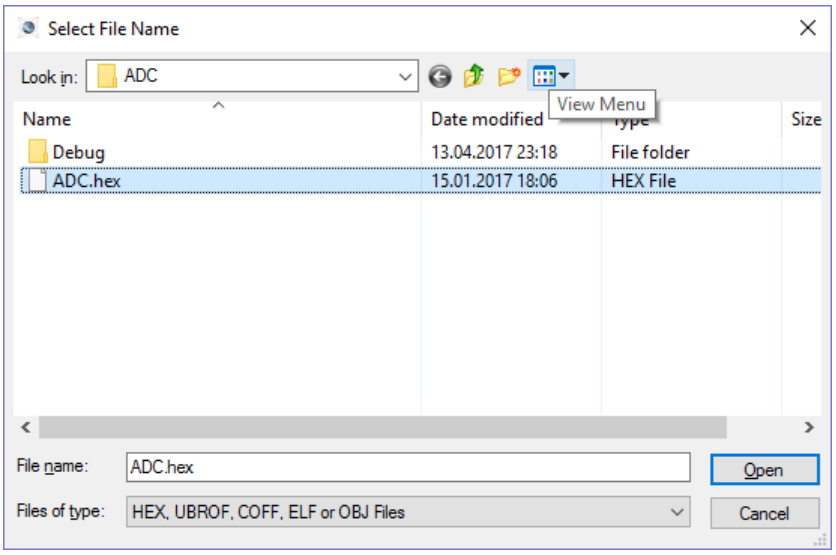


Рисунок 1.51– Вибір файлу програми

1.9 Створення проекту та отримання hex–файла у Atmel Studio

Для програмування мікроконтролерів AVR може використовуватися Atmel Studio 7 – інтегроване середовище розробки для програмування та відлагодження програм для мікроконтролерів AVR та AVR32 в операційних системах Windows. Після шостої версії середовище може працювати як і з AVR– мікроконтролерами, так і з системами з ARM– архітектурою.

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

Atmel Studio містить в собі такі інструмени, як вбудований C/C++–компілятор, симулятор мікропроцесорної системи для відлагодження програм, менеджер проектів, редактор коду, модуль внутрішньосхемного відлагодження, а також інтерфейс командного рядка. Крім стандартних елементів, середовище підтримує ряд інших інструментів, таких як компілятор GCC та плагін AVR RTOS (операційної системи реального часу). Крім C/C++, середовище дозволяє програмувати також на асемблері. Завдяки зв'язці програмних пакетів Atmel Studio и Proteus от фирмы Labcenter Electronics у нас є можливість програмування мікроконтролерів без наявності будь–якої матеріальної бази.

Остання версія Atmel Studio підтримує всі існуючі на сьогоднішній момент 8–бітові, 32–бітові AVR, SAM3 і SAM4 мікроконтролери і включає в себе понад 1100 проектів з прикладами. Також доступні старі версії програми.

Програма останньої версії розроблялась за підтримки Microsoft Visual Studio тому і дизайн Atmel Studio схожий на їх продукт.

Дана програма являється безкоштовною та доступна для завантаження з офіційного сайту виробника (рисунок 1.52).

В даних методичних вказівках була використана програма Atmel Studio 7. Після запуску файлу інсталятора з'явиться вікно привітання. Перед початком інсталяції з'явиться вікно ліцензійного погодження (рисунок 1.53). Після цього погодження з правилами треба переходити між екранами встановлення програми по натисненню кнопки "Next".

Рисунок1.52 – Зовнішній вигляд програми

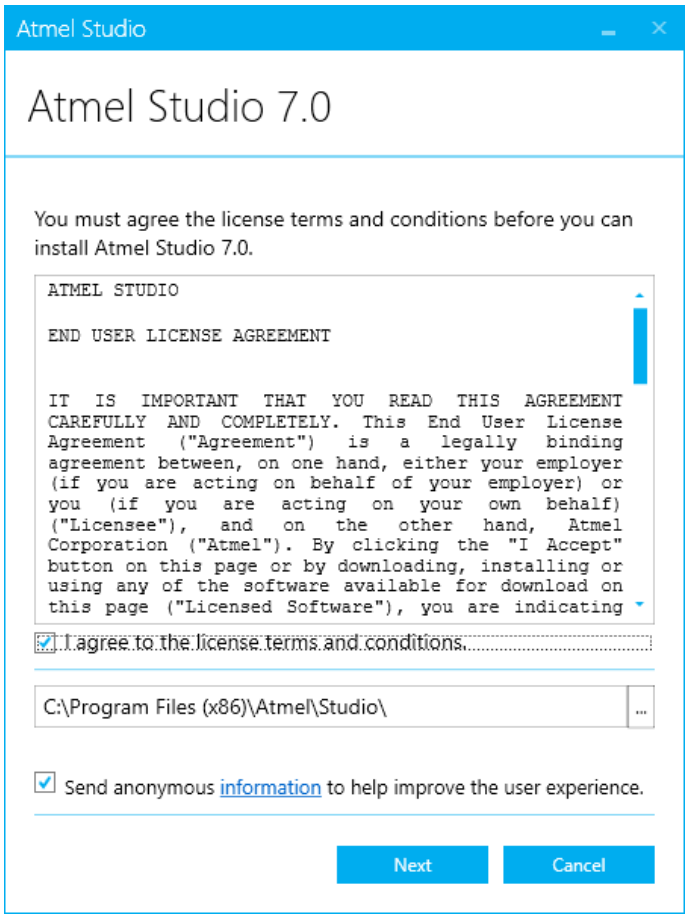


Рисунок 1.53 – Вікно погодження з правилами

Після запуску студії користувач потрапляє на стартову форму. На ній користувач може почати роботу із створення нового проекту або відкрити існуючий (рисунок 1.54).

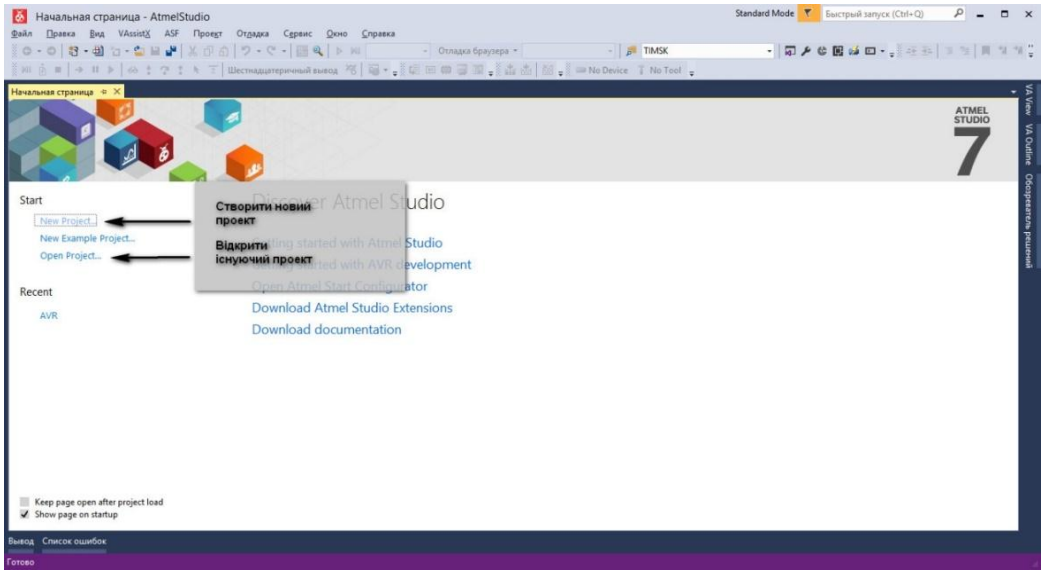


Рисунок 1.54 – Початкове вікно Atmel studio

Створимо новий проект. Для цього виберемо New Project, після чого потрапимо на форму параметрів (рисунок 1.55).

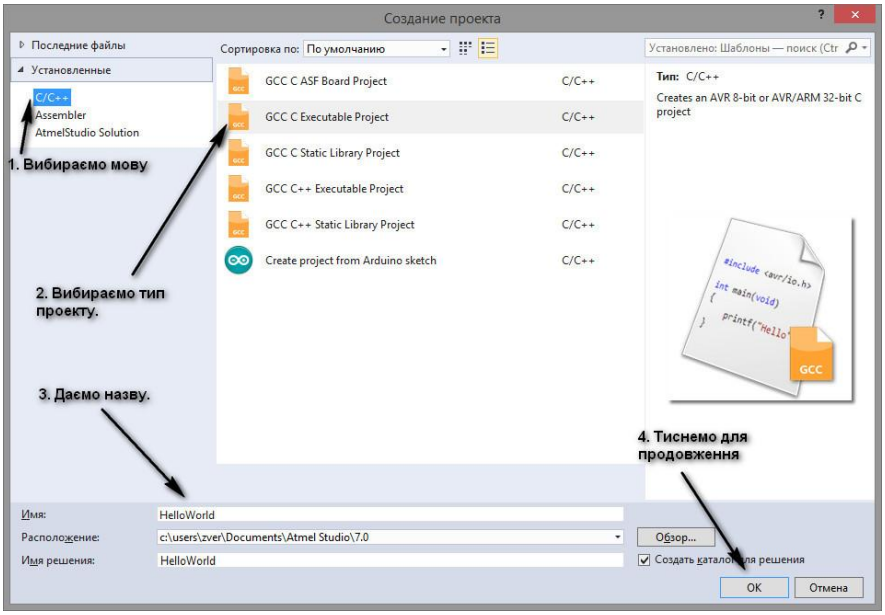


Рисунок 1.55 – Меню вибору мови, типу проекту та назви

З області шаблонів (вкладки встановлені) необхідно вибрати C/C ++ в якості нашого шаблону. В області з типами проектів вибираємо «Виконуваний проект». Після цього в нижній частині вікна даємо назву для нашого рішення та вибираємо його місце розташування. Тиснемо «ОК» та переходимо до вибору віртуального виконавчого пристрою (рисунок 1.56).

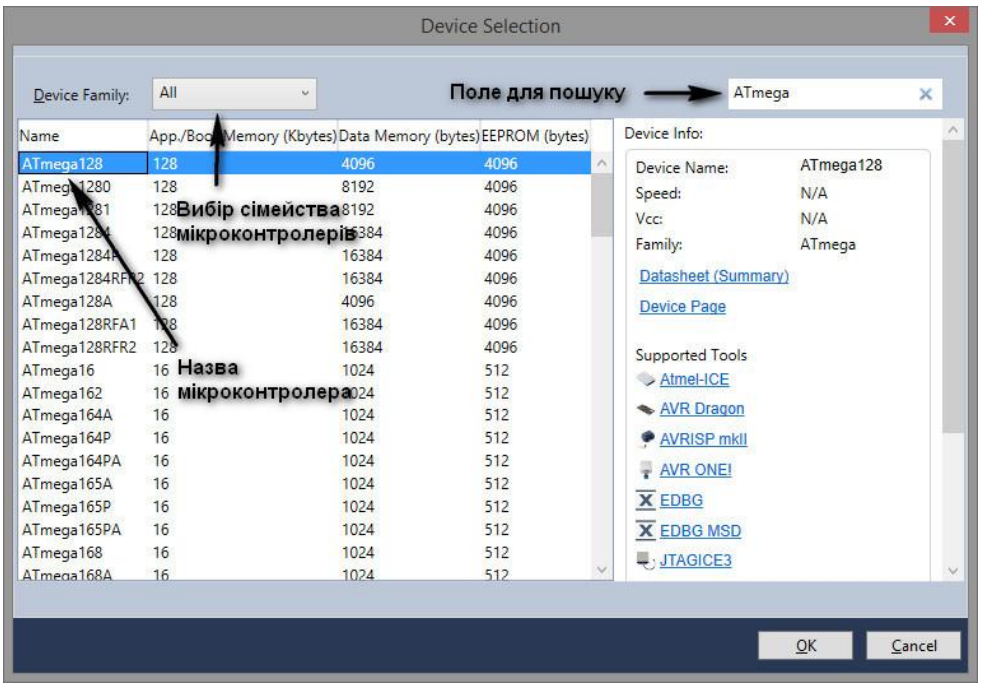


Рисунок1.56 – Меню вибору мікроконтролера

В даному випадку вибираємо ATmega16 та будемо використовувати даний мікроконтролер як цільовий пристрій, для якого буде створюватись виконувана програма. Після цього необхідно натиснути «ОК». Atmel Studio створить проект та покаже вікно середовища розробки (рисунок 1.57).

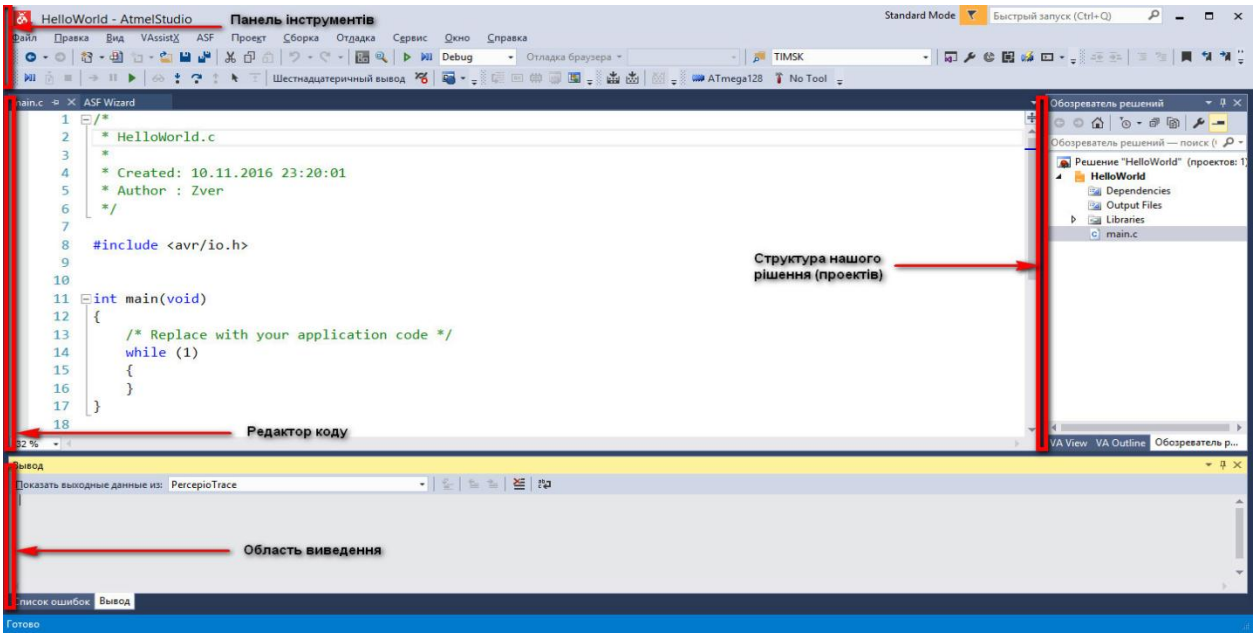


Рисунок 1.57 – Головне вікно середовища розробки

Як бачимо, середовище можна умовно поділити на зони. Зверху розташовується панель інструментів, посередині головна область коду, знизу – область виводу результату, справа – вікно перегляду структури проекту. Звісно, середовище можна налаштувати по–іншому, вибравши необхідні вікна для відображення.

Щоб створити hex файл у Atmel studio 7 (рисунок 1.58) з проекту на мові С по–перше потрібно упевнитися, що в проекті існує файл з функцією main(), а також, що всі бібліотеки підключені.

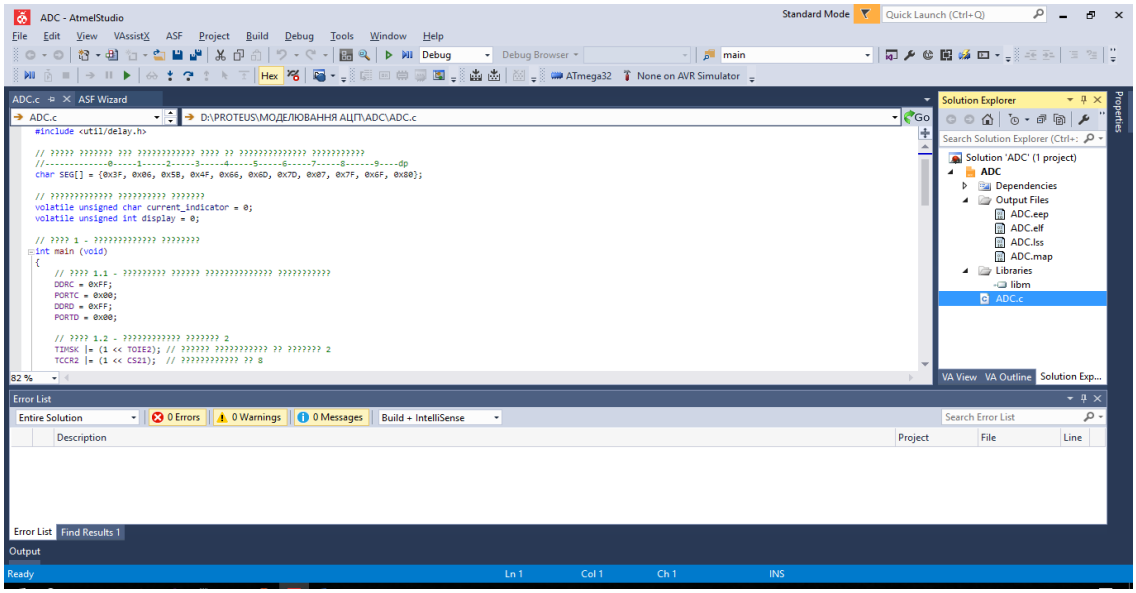


Рисунок 1.58 – Загальний вигляд середовища розробки

Після цього потрібно натиснути правою кнопкою миші на даний проект у меню Solution explorer, що знаходиться в основному вікні справа за замовчуванням (рисунок 1.59). В меню потрібно обрати пункт Build (збудувати) та зачекати декілька секунд.

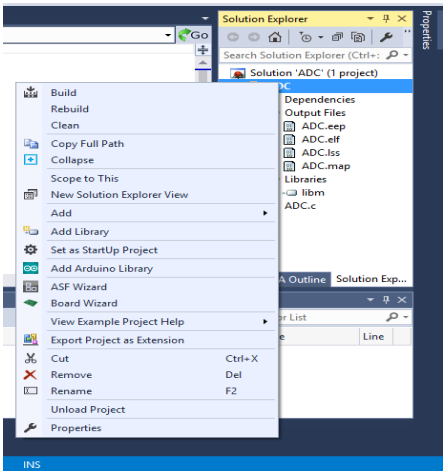


Рисунок 1.59 – Меню налаштувань проекту

Компілятор може вивести декілька зауважень (рисунок 1.60), які, проте, не заважають використовувати програму та її файли.

Рисунок 1.60 – Попередження від компілятора

Також буде автоматично згенеровано hex–файл для проекту, який збережеться у папку з проектом, як показано на рисунок 1.61.

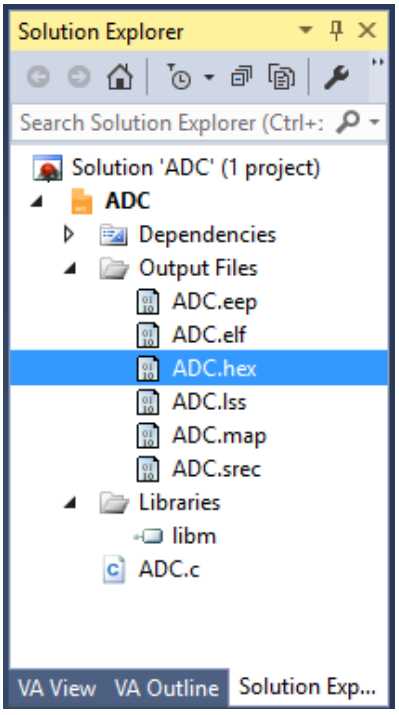


Рисунок 1.61 – Новий файл

Дані з файла одразу можна продивитися, якщо двічі натиснути лівою кнопкою по файлу у списку (рисунок 1.62).

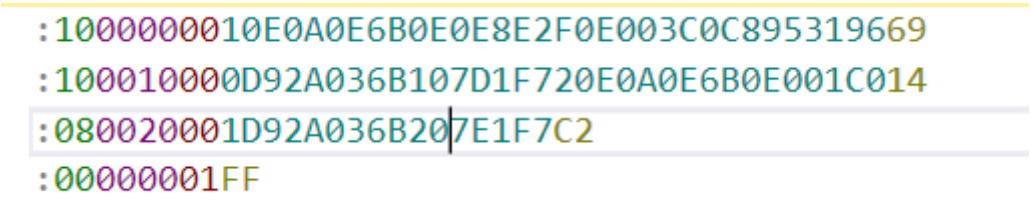


Рисунок 1.62 – Дані з файлу

1.10 Створення проекту та отримання hex–файла у Keil uVision

Для програмування мікроконтролерів МК–51 може використовуватися Keil – інтегроване середовище розробки для програмування та відлагодження програм для мікроконтролерів в операційній системі Windows. Середовище також може працювати також як з AVR– контролерами, так і з системами з ARM– архітектурою.

Keil uVision містить в собі такі інструмени, як вбудований C/C++– компілятор, симулятор мікропроцесорної системи для відлагодження програм, менеджер проектів, редактор коду, модуль внутрішньосхемного відлагодження, а також інтерфейс командного рядка. Крім C/C++, середовище дозволяє програмувати також на асемблері. Завдяки зв’язці програмних пакетів Keil та Proteus от фирмы Labcenter Electronics у нас є можливість програмування мікроконтролерів без наявності будь– якої матеріальної бази.

Дана програма являється безкоштовною та доступна для завантаження з офіційного сайту виробника (рисунок 1.63).

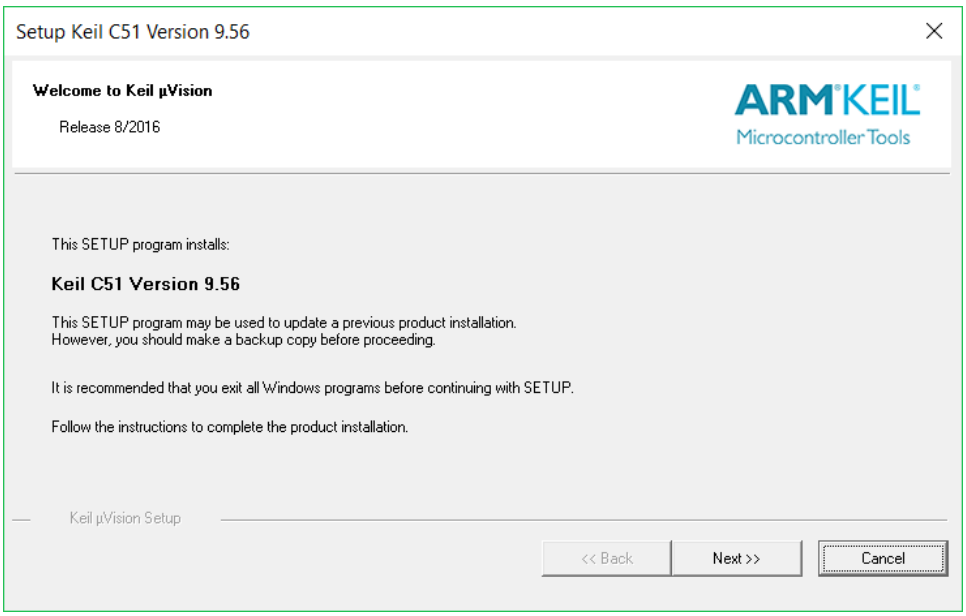


Рисунок 1.63 – Інсталяція продукту Keil

Для продовження потрібно вказати ім’я, назву закладу та електронну адресу (рисунок 1.64).

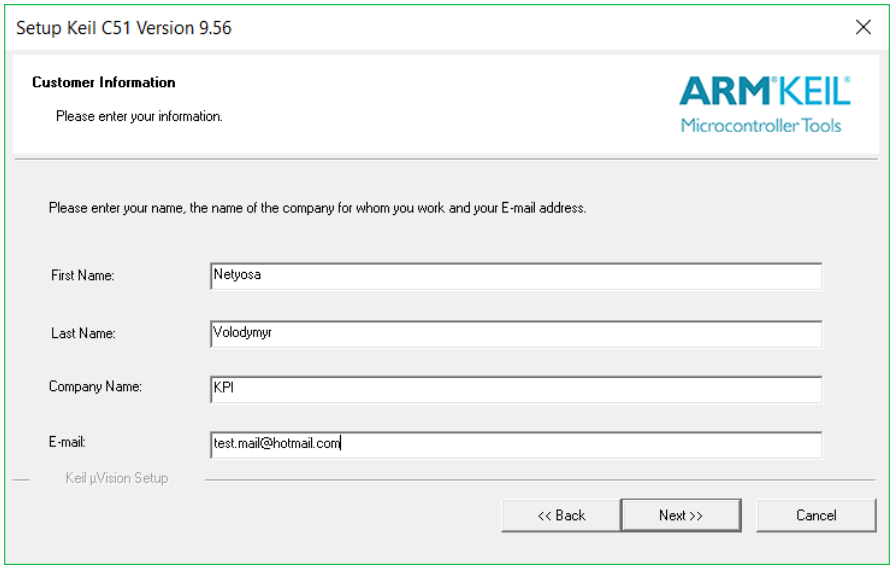


Рисунок 1.64 – Продовження інсталяції продукту Keil

Далі з`явиться вікно Keil uVision (рисунок 1.65).

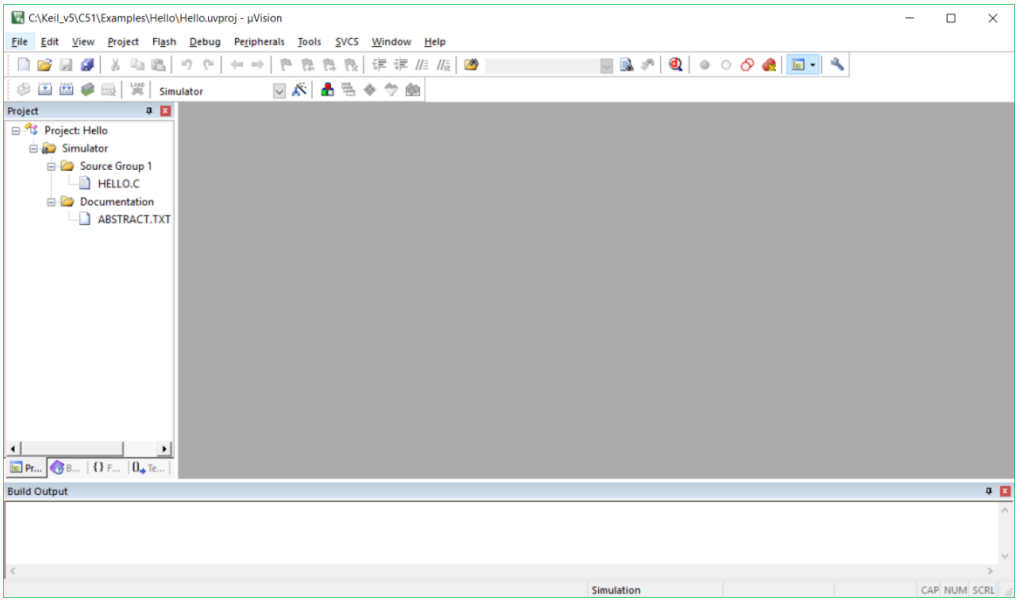


Рисунок1.65 – Початкове вікно Keil uVision

Створимо новий проект. Для цього виберемо New Project, після чого потрапимо на форму параметрів (рисунок 1.66).

Після цього потрібно вибрати директорію, у яку потрібно зберегти цей проект та вибрати необхідну архітектуру (наприклад, AT89C51) (рисунок 1.67)

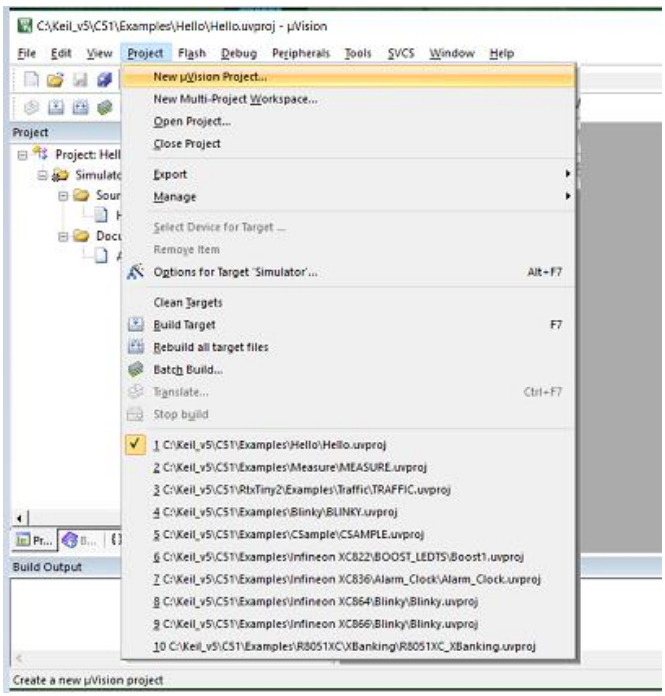


Рисунок 1.66 – Створення нового проекту

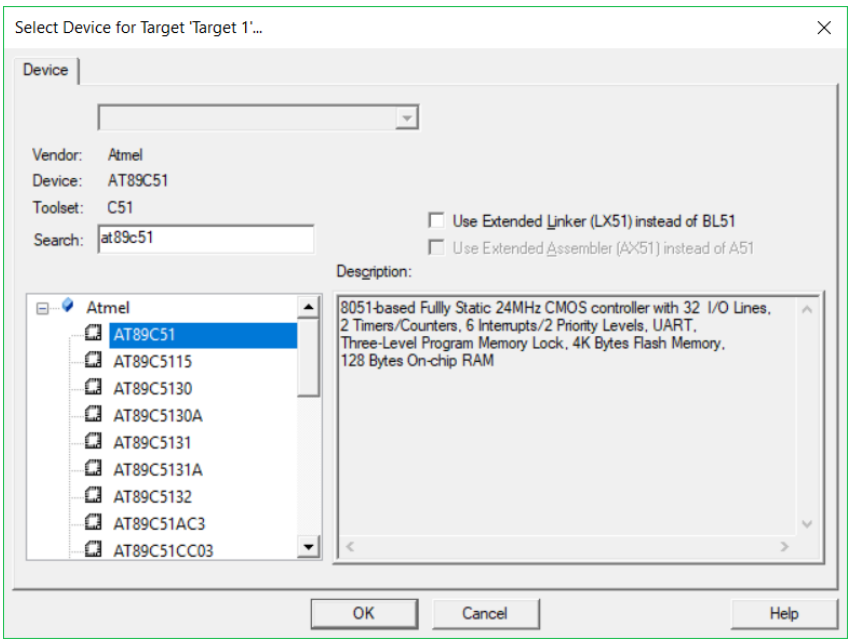


Рисунок 1.67 – Вибір потрібної архітектури

Щоб додати новий *.c файл до проекту, потрібно натиснути на «Add new item to group» (рисунок 1.68, 1.69)

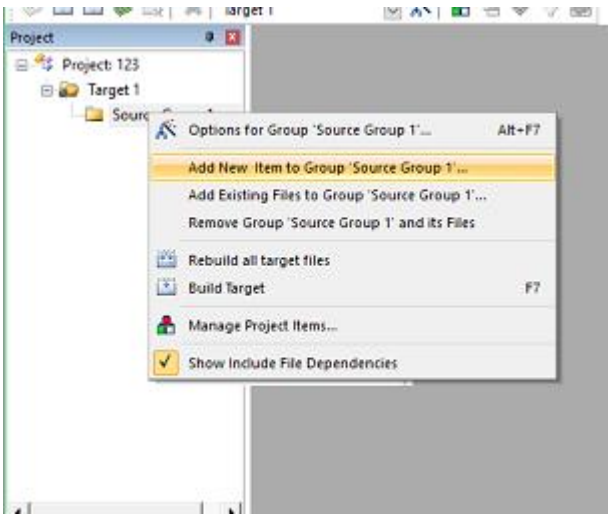


Рисунок 1.68 – Створення нового файлу в проекті

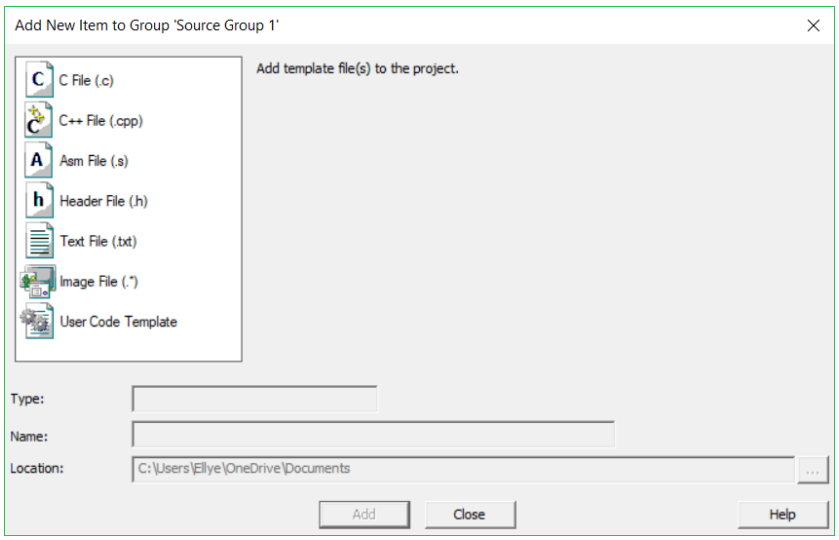


Рисунок1.69 – Вибір типу файлу, що створюється

Після цього потрібно вказати шлях генерації hex–файлу (рисунки 1.70, 1.71)

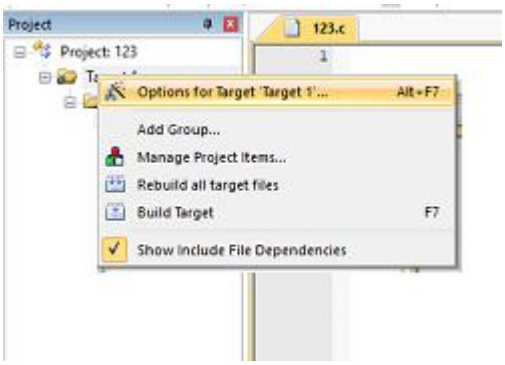


Рисунок 1.70 – Налаштування проекту

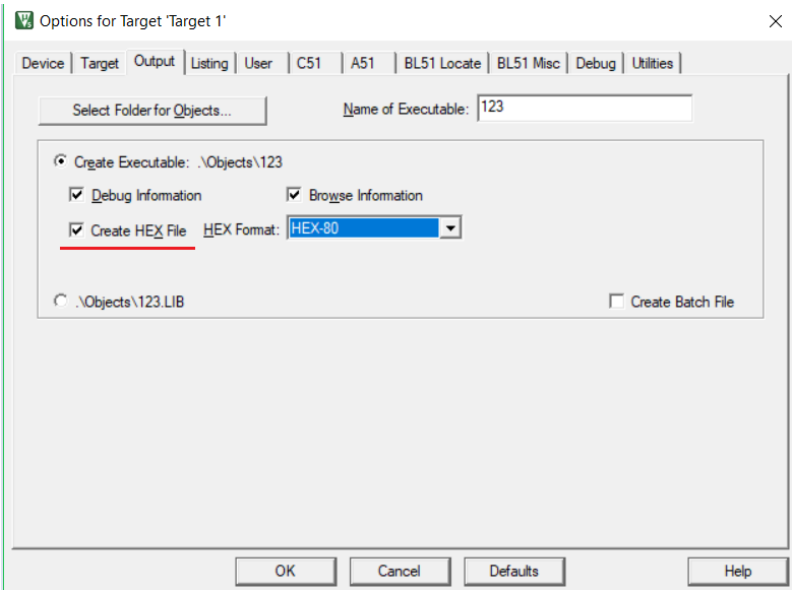


Рисунок 1.71 – Генерація hex – файлу

Для генерації hex–файлу потрібно ввімкнути прапорець «Create HEX file» та вибрати формат «Hex–80». Після цього потрібно виконати побудову проекту та отримати hex файл за шляхом у папці Objects, який був вказаний як головний для цього проекту (рисунки 1.72, 1.73).

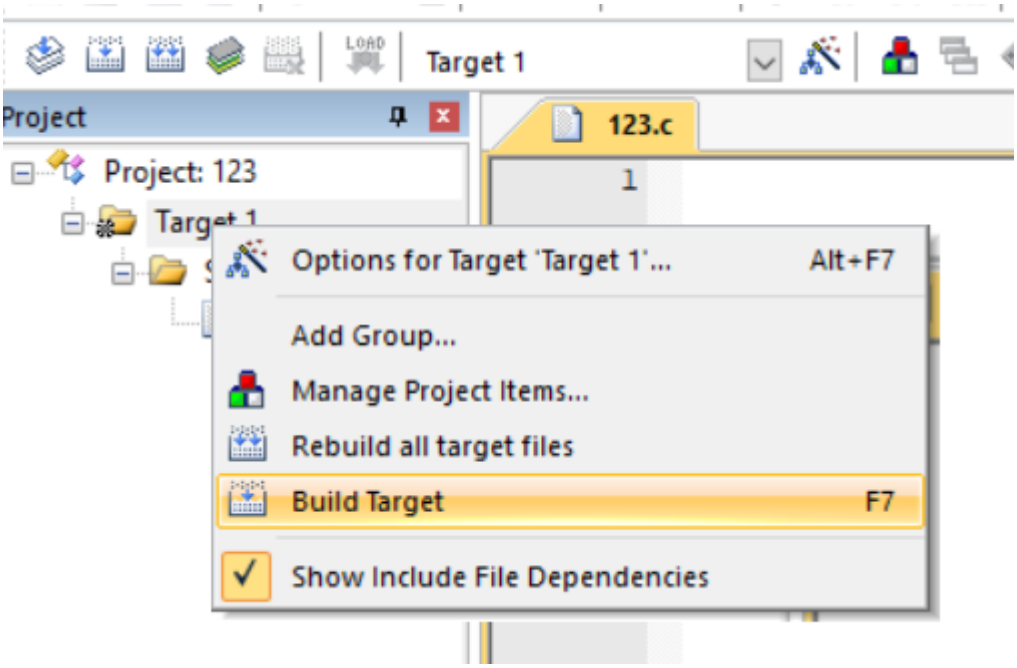


Рисунок1.72 – Побудова проекту

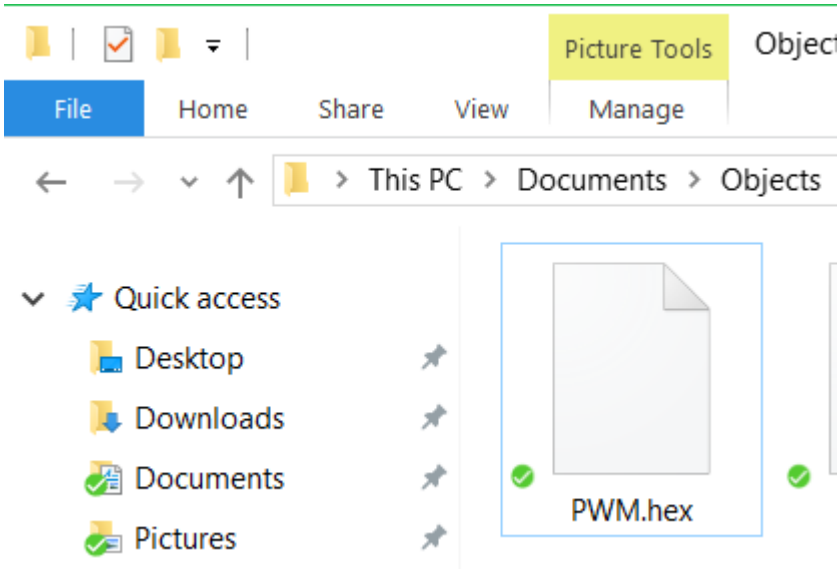


Рисунок 1.73 – hex – файл

2 МОДЕЛЮВАННЯ ПРИСТРОЮ КЕРУВАННЯ ДВИГУНОМ ПОСТІЙНОГО СТРУМУ З ВИКОРИСТАННЯМ МІКРОКОНТРОЛЕРА СІМ`Ї AVR

2.1 Опис моделі

Робочу модель пристрою керування двигуном постійного струму показано на рисунку 2.1.

Розберемо цю модель докладніше. З лівого боку рисунку 2.1 ми бачимо кнопки керування: K1, K2, K3, K4 на які, через резистори R1, R2, R3, R4 відповідно, подається напруга. У правому нижньому куті ми бачимо двигун постійного струму DC Motor, а трохи лівіше мікросхему L293D, через яку мікроконтролер і керує двигуном за допомогою ШІМ – сигналу. Сам мікроконтролер ATmega128 знаходиться у верхньому правому куті. Зовнішні резистори ми використовуємо через те, що не використовуємо внутрішні підтягуючи резистори на вхідних лініях мікроконтролера. На рисунку 2.2 наведено позначення виводів мікроконтролера. Нижче більш докладніше пояснимо, що і куди підключаємо в моделі.

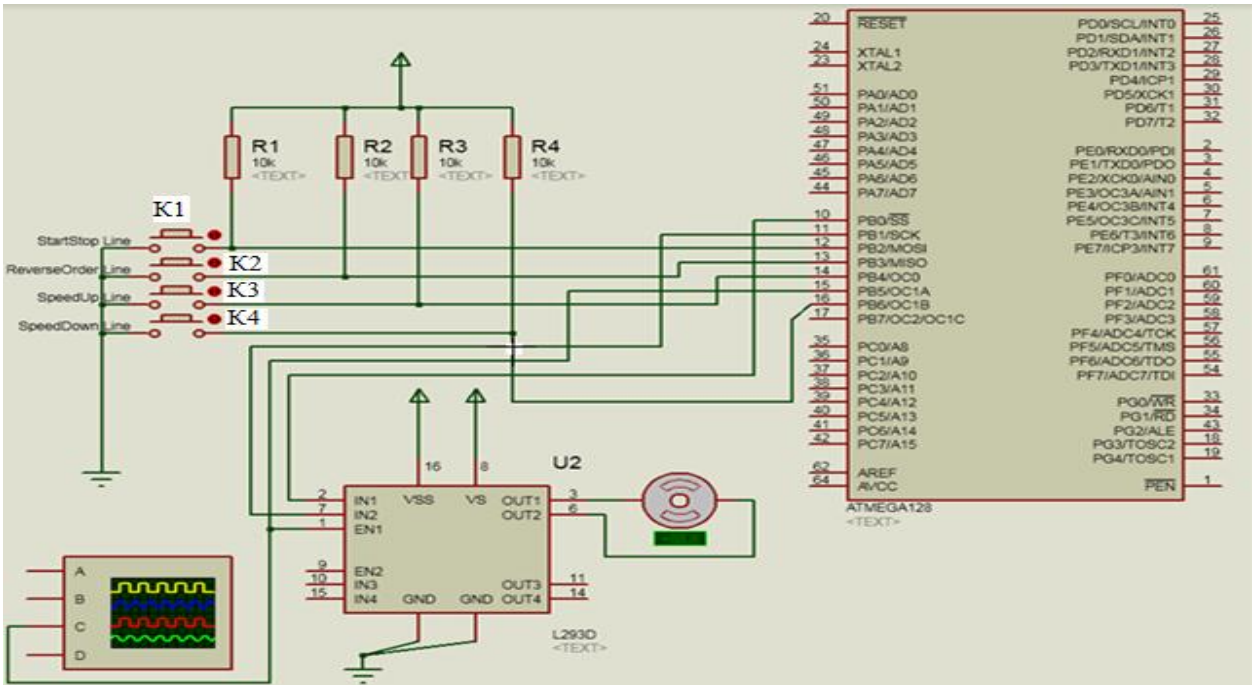


Рисунок 2.1– Схема моделі пристрою керування двигуном постійного струму

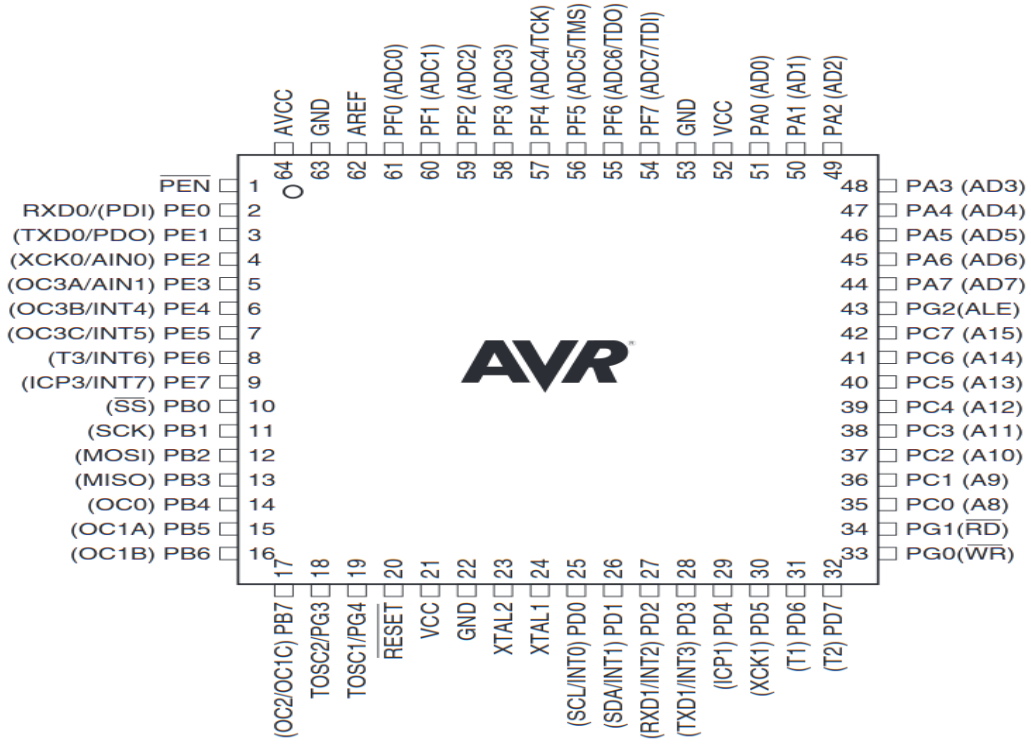


Рисунок 2.2 – Виводи мікроконтролера ATmega128

Почнемо з кнопок, за допомогою яких ми будемо здійснювати керування двигуном постійного струму DC Motor на моделі. Збільшений їх вигляд наведено на рисунку 2.3. Кнопки є нормально розімкненими та не фіксованими, тобто після зняття прикладеного зусилля вони повертаються у вихідний стан. Це у нашому випадку означає наступне: у вихідному стані на входи мікроконтролера ATmega128 через резистори R1…R4 подається високий рівень напруги, тобто логічна 1. А коли кнопки натискаються, на входах мікроконтролера спостерігається низький рівень напруги, тобто логічний 0.

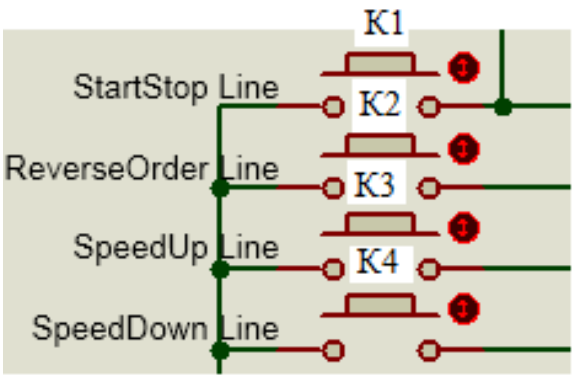


Рисунок 2.3 – Порядок розташування елементів керування

Кнопка K1 (StartStop Line) відповідає за вмикання та вимикання двигуна постійного струму DC Motor. Тобто коли двигун вимкнено, то вона його ввімкне. Коли двигун ввімкнений та обертається, то вона його вимкне. Кнопку K1 підключено до 12–го виводу мікроконтролера – PB2, який повинен бути запрограмований як вхід.

Кнопка K2 (ReverseOrder Line) відповідає за зміну напрямку обертання двигуна постійного струму DC Motor. Одне натискання – одна зміна напрямку. Проте потрібно мати на увазі, що двигун DC Motor інерційний, і він не зупиняється миттєво і одразу починає обертатися в інший бік. Після зміни напрямку програмно, у нього витрачається певний період часу, щоб фізично зупинитись, та почати обертатись у іншу сторону. Кнопку K2 підключено до 13–го виводу ATmega128, а саме до PB3, який попередньо запрограмовано як вхід.

Кнопка K3 (SpeedUp Line) відповідає за фізичне збільшення швидкості обертання нашого двигуна постійного струму DC Motor шляхом поступового збільшення постійної еквівалентної напруги імпульсного ШІМ – сигнала до максимуму, який дорівнює повному значенню напруги живлення на виводі VS мікросхеми L293D (рисунок 2.4). Знову ж, двигун інерційний, тому швидкість обертання збільшується не стрибкоподібно, а поступово, і потребує певного проміжку часу, щоб вийти на новий програмно встановлений рівень.

Кнопку K3 підключаємо до 14–го виводу (PB4) нашого мікроконтролера. Перед цим цей вивід також потрібно запрограмувати як вхід.

Кнопка K4 (SpeedDown Line) відповідає за зменшення швидкості обертання двигуна постійного струму DC Motor, для чого вона поступово зменшує значення постійної еквівалентної напруги імпульсного ШІМ – сигнала.

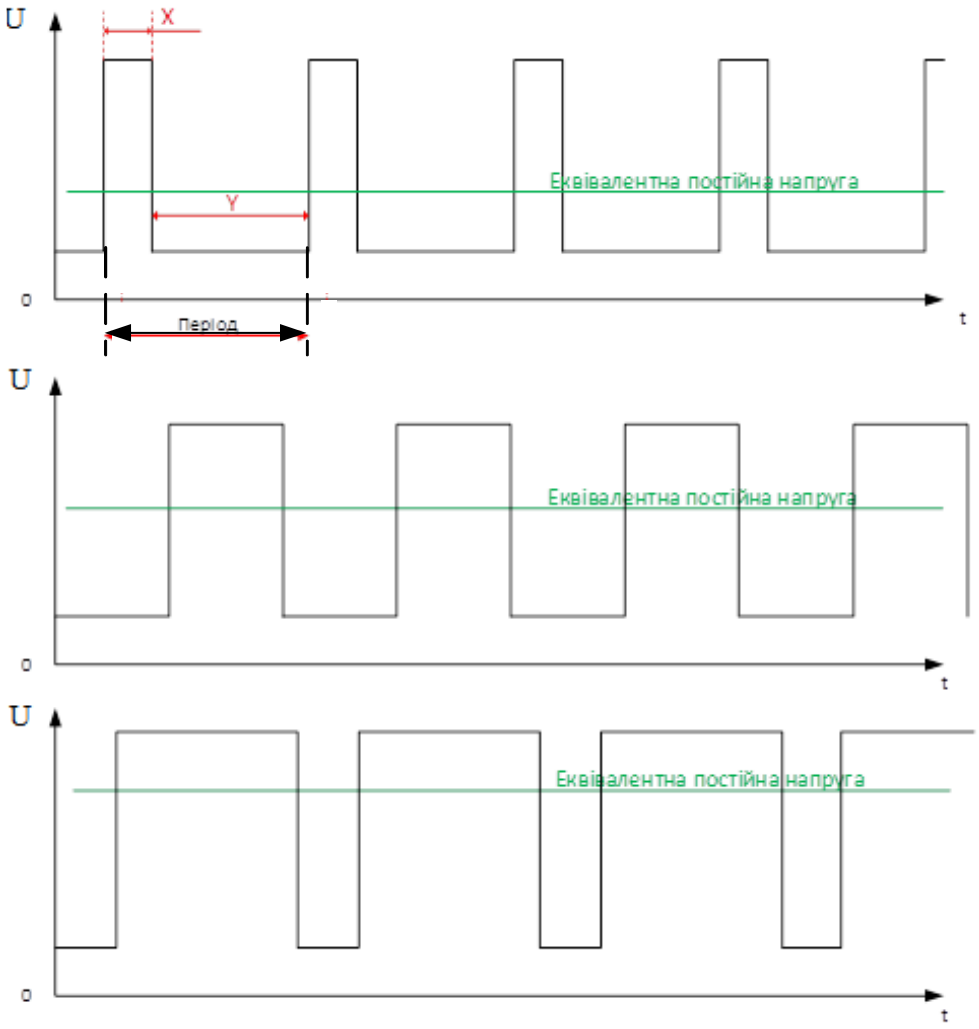


Рисунок 2.4 – Еквівалентна постійна напруга при ШІМ– сигналі

Через інерційність двигуна швидкість обертання зменшується не стрибками, а поступово і вимагає для свого зменшення певного проміжку часу. Кнопку K4 підключаємо до 16–го виводу ATmega128 (PB6), перед цим запрограмувавши його на вхід.

Тепер розглянемо детальніше підключення мікросхеми L293D до мікроконтролера (рисунок 2.5).

Виводи 10 та 11 мікроконтролера, тобто PB0 та PB1 запрограмовані як виходи і підключені, відповідно, до входів IN1 та IN2 мікросхеми L293D (U2).

В залежності від керування, на виходах PB0 та PB1 буде або високий рівень напруги, або низький, і в залежності від цього відповідний ключ у

мосту мікросхеми буде або замкнений, або розімкнений. Це дозволить змінювати напрямок обертання ДПС.

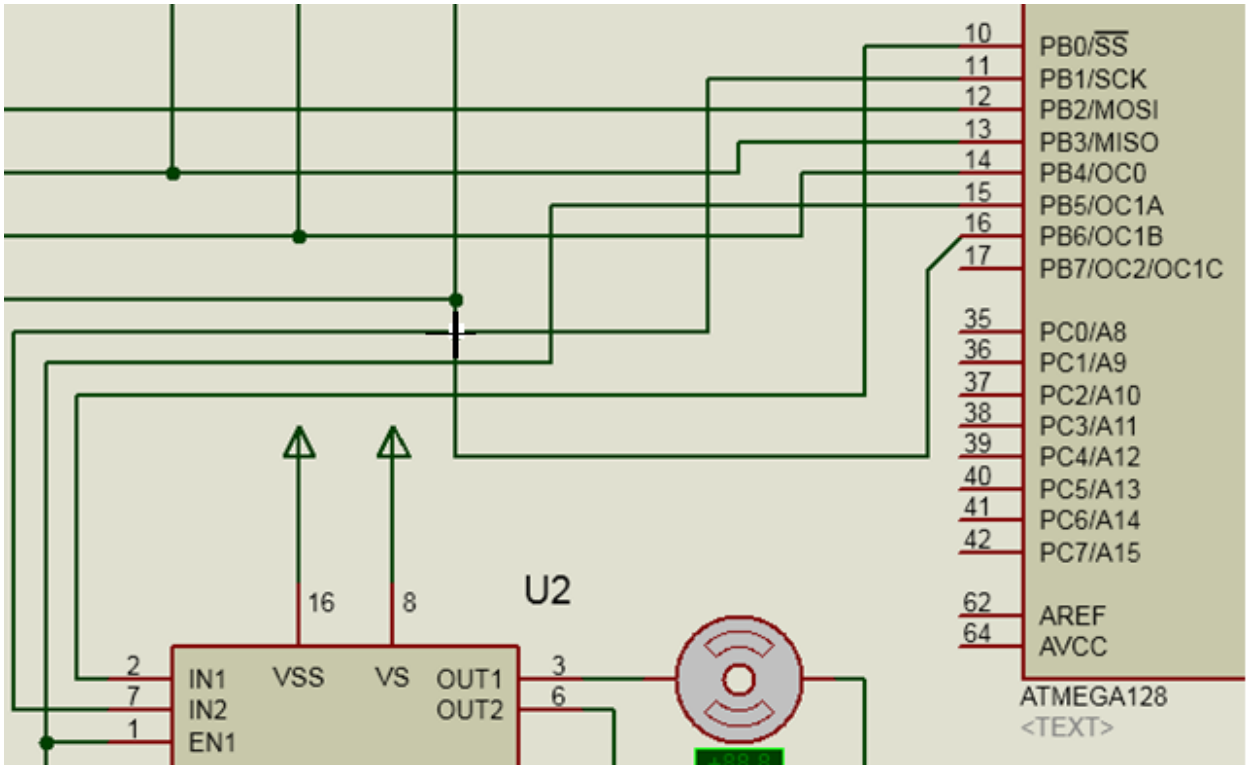


Рисунок 2.5 – Підключення мікросхеми L293D та мікроконтролера ATmega128

15–й вивід мікроконтролера, PB5, запрограмовано як вихід і підключено до входу EN1 мікросхеми L293D. З цього виходу мікроконтролера знімається ШІМ – сигнал. У свою чергу вхід EN1 відповідає за роботу першої мостової схеми у складі мікросхеми (див. 2.4). Коли на ньому високий рівень, на перший міст подається живлення, яке підведено до виводу VS мікросхеми L293D. Коли на вході EN1 низький рівень, на перший міст напруга живлення не подається.

Саме завдяки цьому входу, на який подається ШІМ – сигнал, ми можемо керувати двигуном постійного струму.

Розглянемо використання інших виводів мікросхеми L293D, які зображені на рисунку 2.1. Два виходи GND заземлені. вхід VSS подається напруга живлення самої мікросхеми L293D. На вхід VS подається напруга живлення двигуна постійного струму DC Motor.

Після цього нам залишилось ознайомитись з підключенням двигуна постійного струму DC Motor та мікросхеми L293D. Це підключення зображено на рисунку 2.6.

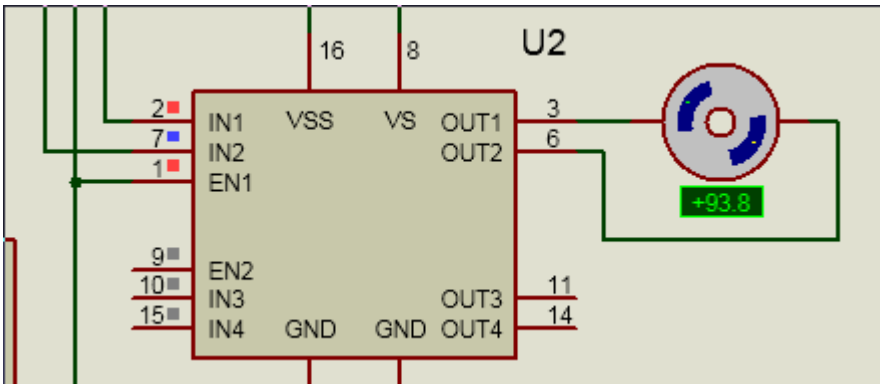


Рисунок 2.6 – Підключення двигуна постійного струму до мікросхеми L293D

Як ми бачимо з рисунку 2.6, наш двигун постійного струму DC Motor підключено до 3 та 6 виходів мікросхеми L293D, а саме OUT1 та OUT2.

На ці виходи, в залежності від того, чи ввімкнене живлення першої мостової схеми, що керується входом EN1, та в залежності від стану ключів у першій мостовій схемі, які керуються входами IN1 та IN2, або подається напруга живлення, яку ми подали на вхід VS, або напруга не подається.

У випадку, коли на один з виводів, OUT1 чи OUT2, подається напруга живлення, а на інший не подається, виникне різниця потенціалів, що змусить потекти струм по обмотці нашого двигуна постійного струму DC Motor, внаслідок чого він почне обертатися.

Деякі фрагменти, які демонструють роботу системи, наведено на рисунках 2.7…2.11.

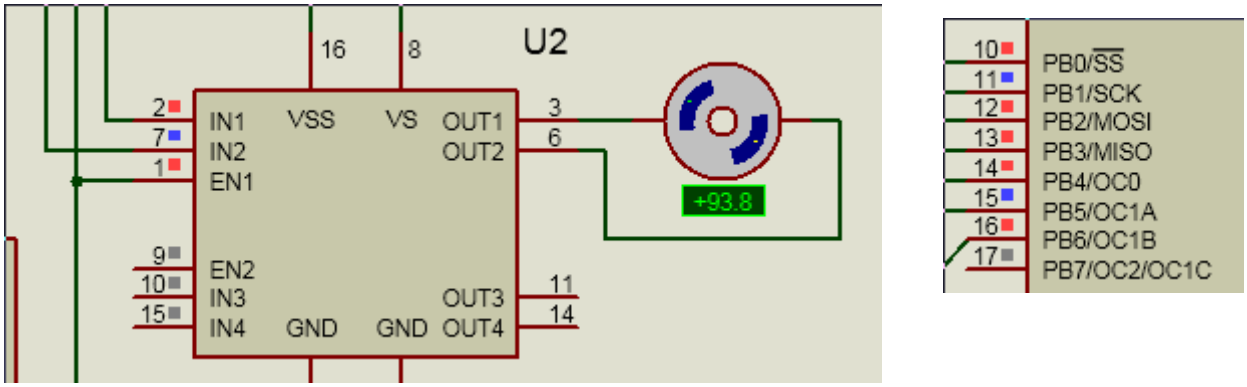


Рисунок 2.7 – Стан моделі після натискання кнопки «старт/стоп»

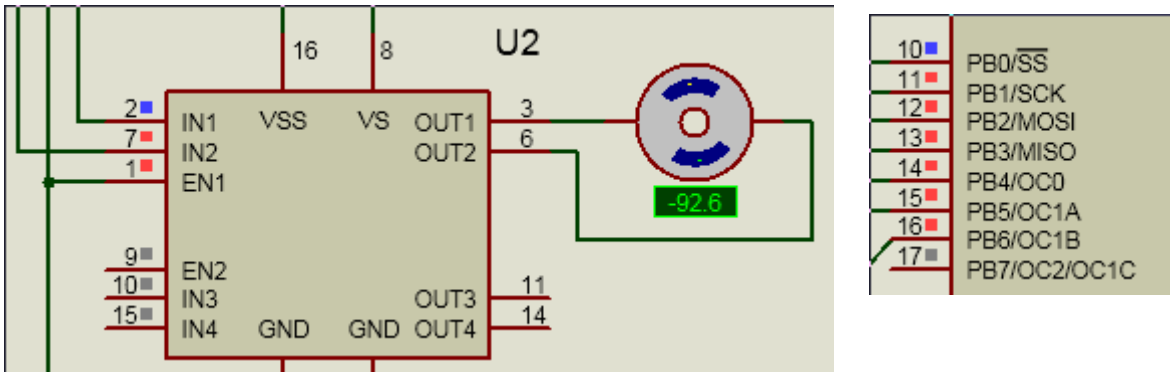


Рисунок 2.8 – Стан моделі після активації режиму реверсу

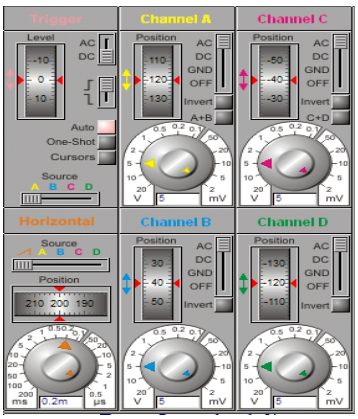


Рисунок 2.9 – Налаштування осцилографа

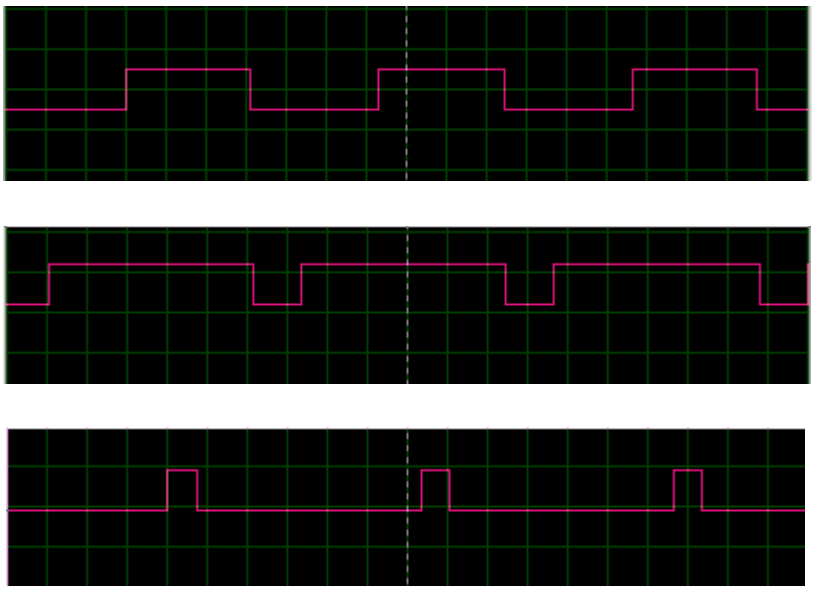


Рисунок 2.10 –ШІМ– сигнал, що подається на двигун (шпаруватість q=2, q > 2, q <2)

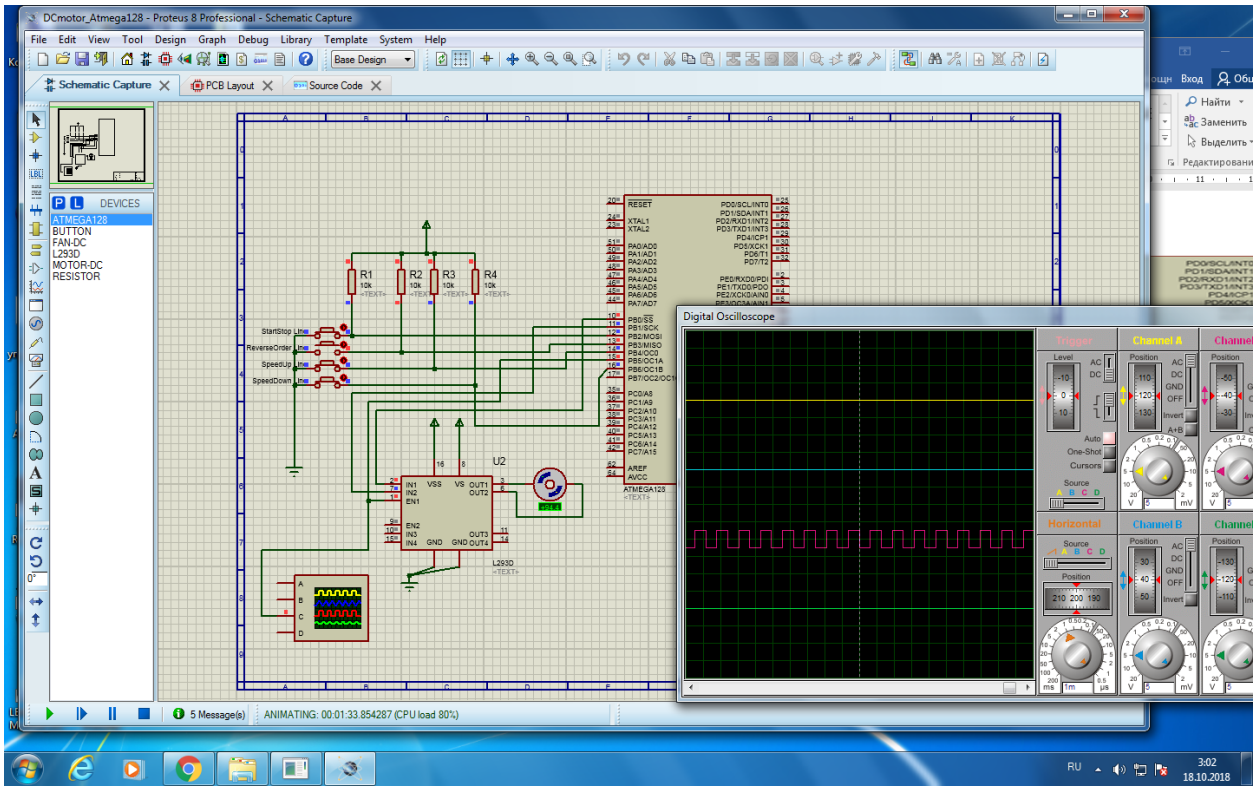


Рисунок 2.11 – Стан усієї системи після запуску сценарію

2.2 Мікроконтролер

Для контролю та виконання всіх функцій, які були покладені на систему потрібен мікроконтролер. Для цієї цілі був вибраний AVR – ATmega 128.

ATmega128 – малопотужний 8 – розрядний КМОН – мікроконтролер, який оснований на розширеній AVR RISC архітектурі. ATmega128 досягає 1 млн. операцій в секунду, оскільки більшість команд виконується за один машинний такт.

AVR ATmega 128 має 64 виводи типу вхід – вихід. В мікроконтролер вмонтовано FLASH – пам’ять, яку є можливість програмувати та перепрограмувати.

Основні характеристики контролера:

1) Високопродуктивний, споживає мало енергії;
2) Розвинена RISC архітектура:
   а) 131 команда, більшість виконується за один машинний такт;
   б) 32 робочих регістри загального призначення;
   в) повністю статичний режим роботи;
3) Енергонезалежна пам’ять програм и даних;
4) 128 КБ внутрішньо системної самопрограмуваної FLASH пам’яті з кількістю циклів перепрограмування до 10 000;
5) 4096 байт EEPROM із припустимою кількістю циклів стирання та запису до 100 000;
6) 64 виводи типу вхід – вихід;
7) JTAG (IEEE1149.1 сумісний) інтерфейс;
8) Сканування пам’яті відповідно до JTAG стандарту;
9) Периферійні функції:
   а) два 8– бітних таймери/лічильники із програмованим режимом порівняння;

б) один 16– бітний таймер/лічильник із програмованим режимом порівняння й захоплення;

в) лічильник реального часу із програмованим генератором;

г) чотири ШІМ– генератори;

д) 8 вхідних каналів 10 – бітного АЦП;

10) SPI та I^2C синхронні послідовні інтерфейси;

11) Вбудований аналоговий компаратор;

12) Спеціальні функції:

а) функція Reset для включення живлення і функція вимикання для зниження напруги живлення;

б) внутрішній калібрований RC – генератор;

в) зовнішні й внутрішні джерела переривання;

13) 40 вивідний корпус PDIP, 44 вивідний корпус TQFP, і 44 контактний MLF;

14) Напруга живлення – від 4.5 В до 5.5 В;

15) Тактова частота – від 0 до 16 МГЦ.

Опис виводів мікроконтролера:

- VCC – напруга живлення;
- GND – спільна земля;
- порт A (PA7..PA0) – 8 – розрядний порт двонаправленого введення – виведення із внутрішніми підтягуючими резисторами, які вибираються окремо для кожного розряду. При введенні лінії порту A будуть діяти як джерело струму, якщо зовні діє низький рівень і включені резистори, що підтягують;
- порт B (PB7..PB0) – 8 – розрядний порт двонаправленого введення – виведення із внутрішніми підтягуючими резисторами, які вибираються окремо для кожного розряду;
- порт C (PC7..PC0) – 8 – розрядний порт двонаправленого введення – виведення із внутрішніми підтягуючими резисторами, які вибираються окремо для кожного розряду;

– порт D (PD7..PD0) – 8 – розрядний порт двонаправленого введення – виведення із внутрішніми підтягуючими резисторами, які вибираються окремо для кожного розряду;
– порт E (PE7..PE0) – 8 – розрядний порт двонаправленого введення – виведення із внутрішніми підтягуючими резисторами, які вибираються окремо для кожного розряду;
– порт F (PF7..PF0) – 8 – розрядний порт двонаправленого введення – виведення із внутрішніми підтягуючими резисторами, які вибираються окремо для кожного розряду. Якщо активізований інтерфейс JTAG, то резистори на лініях PF7(TDI), PF5(TMS) і PF4(TCK) будуть підключені, навіть якщо виконується скидання. Порт F також виконує функції інтерфейсу JTAG;
– порт G (PG4..PG0) – 5 – розрядний порт двонаправленого введення – виведення із внутрішніми підтягуючими резисторами, які вибираються окремо для кожного розряду.
– порт G також виконує деякі спеціальні функції Atmega128;
– RESET–вхід скидання. Якщо на цей вхід прикласти низький рівень тривалістю більш мінімально необхідної, то буде згенеровано скидання незалежно від роботи синхронізації. Дія імпульсу меншої тривалості не гарантує генерацію скидання;
– XTAL1–вхід інвертуючого підсилювача генератора та вхід зовнішньої синхронізації;
– XTAL2– вихід інвертуючого підсилювача генератора;
– AVCC– вивід для подачі живлення порту F і аналогово–цифрового перетворювача. Він повинен бути зовні пов’язаний з VCC, навіть якщо АЦП не використовується. При використанні АЦП цей вивід пов’язаний з VCC через фільтр низьких частот;
– AREF– вхід підключення джерела опорної напруги АЦП;

– PEN–вхід дозволу програмування для режиму послідовного програмування через інтерфейс SPI. Якщо під час дії скидання при подачі живлення на цей вхід подати низький рівень, то мікроконтролер переходить у режим послідовного програмування через SPI.

2.3 Двигун

В моделі використовується електродвигун постійного струму. Двигуном можна керувати за допомогою широтно–імпульсної модуляції. Реалізувати широтну–імпульсну модуляцію можна з допомогою вбудованого в мікроконтроллер таймера. Вибір двигуна є важливим, оскільки від цього вибору буде залежати стабільність роботи системи. Реверсивний режим двигуна постійного струму здійснюється зміною полярності.

Двигунами постійного струму можна керувати за допомогою реле або транзисторів. Одиночне перемикати реле вмикає і вимикає двигун, а парне відповідає за напрямок обертання.

Регулювання швидкості обертання двигуна постійного струму зазвичай забезпечується за допомогою формування керуючих імпульсів у вигляді сигналів з широтно–імпульсною модуляцією. Подібний підхід дозволяє регулювати середню величину потужності, що надходить на двигун. В даному випадку, чим більше величина відношення «тривалість імпульсу / період», тим більше потужність надходить на двигун.

Частота сигналу з широтно–імпульсною модуляцією повинна перевищувати 20 кГц, що дозволяє виключити звукові ефекти, пов’язані з формуванням звукових сигналів самим двигуном при зміні магнітного поля, яке в ньому утворюється.

2.4 Драйвер ШІМ

Для керування двигуном постійного струму нам потрібно підключати його за допомогою відповідної мостової схеми, яку зображено на рисунку 2.12.

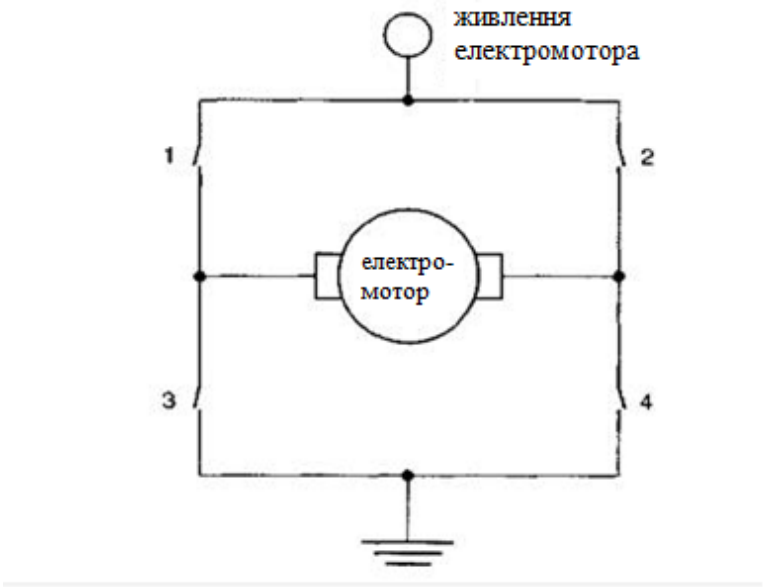


Рисунок 2.12– Мостова схема підключення двигуна постійного струму

Також, нам бажано знайти пристрій, який би перетворював керуючі сигнали малої потужності у струми, достатні для керування двигунами. Такі пристрої називаються драйверами двигунів.

Існує достатньо багато самих різних схем для керування двигунами постійного струму. Вони відрізняються як потужністю, так і елементною базою, на якій вони виконані. Нижче зупинимось на простому драйвері керування двигунами, який виконано у вигляді повністю готової до роботи мікросхеми. Назва цієї мікросхеми L293D і вона є одною з самих розповсюджених мікросхем, призначених для цих цілей. Зовнішній вигляд мікросхеми L293D показано на рисунку 2.13.

Рисунок 2.13 – Мікросхема L293D

L293D включає одразу два драйвери для керування двома електродвигунами відносно невеликої потужності. Це відбувається через чотири незалежних канали, які об’єднано у дві пари. Мікросхема також має дві пари входів для сигналів керування і дві пари виходів для підключення електродвигунів. Крім того, у L293D є два входи для вмикання кожного з драйверів.

Ці входи використовуються для керування швидкістю обертання електродвигунів за допомогою широтно–імпульсної модуляції сигналів. Саме через це ця мікросхема нам добре підходить.

Також, L293D забезпечує розділене живлення для мікросхеми і для керованих нею двигунів, що дозволяє підключати електродвигуни з більшою напругою живлення, чим у мікросхеми. Розділення живлення мікросхеми і електродвигунів нам також буде необхідним для зменшення завад, які викликано стрибками напруги, що пов’язані з роботою двигунів.

Обидва драйвери, що входять у склад мікросхеми мають ідентичні принципи роботи, тому розглянемо принцип роботи лише одного з них. Спрощений вид драйвера наведено на рисунку 2.14.

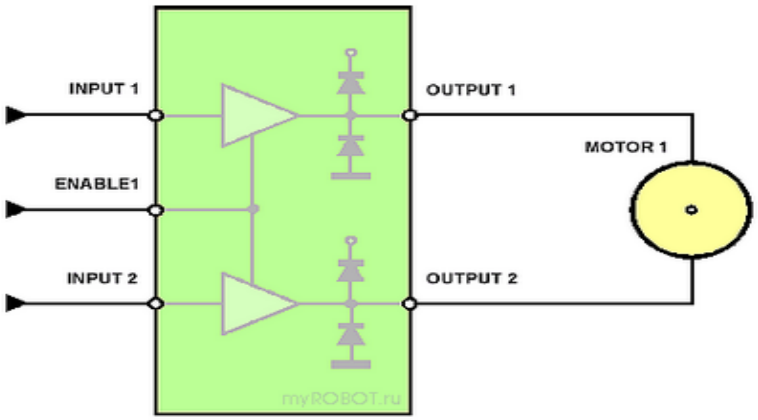


Рисунок 2.14 – Спрощений вид драйвера мікросхеми L293D

До виходів OUTPUT1 та OUTPUT2 підключається двигун постійного струму MOTOR1. На вхід ENABLE1, який відповідає за ввімкнення драйвера, подаємо керуючий сигнал, наприклад, під’єднаємо його до додатного полюсу джерела живлення +5V. Якщо при цьому на входи

INPUT1 та INPUT2 не подаються відповідні керуючі сигнали, то двигун обертатися не буде.

Якщо вхід INPUT1 з'єднати з додатним полюсом джерела живлення, а вхід INPUT2 – з від`ємним, то двигун почне обертатися.

Тепер з'єднаємо вхід INPUT1 з від`ємним полюсом джерела струму, а вхід INPUT2 – з додатним. Двигун почне обертатися в іншу сторону.

Спробуємо подати сигнали одного рівня одразу на два керуючих входи INPUT1 та INPUT2, тобто під'єднати обидва входи до додатного полюсу джерела живлення або до від`ємного. Як результат – двигун обертатися не буде.

Якщо ми приберемо керуючий сигнал з входу ENABLE1, то при будь яких варіантах наявності сигналів на двох входах INPUT1 та INPUT2 двигун обертатися не буде. На вхід ENABLE1 подається ШІМ – сигнал, який формується ШІМ–модулем мікроконтролера. Змінюючи шпаруватість цього сигналу ми змінюємо постійну складову імпульсного сигналу, що в свою чергу змінює швидкість обертання двигуна.

На рисунку 2.15 показано нумерацію та позначення виводів мікросхеми.

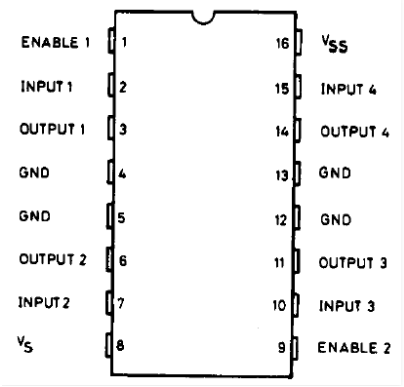


Рисунок 2.15 – Нумерація та позначення виводів мікросхеми L293D

Призначення виводів:

– входи ENABLE1 та ENABLE2 відповідають за вмикання відповідного драйвера, що входить у склад мікросхеми. Це ті самі входи для ШІМ – сигналу, які ми будемо використовувати;

- входи INPUT1 та INPUT2 керують двигуном, який підключено до виходів OUTPUT1 та OUTPUT2;
- входи INPUT3 та INPUT4 керують другим двигуном, який підключено до виходів OUTPUT3 та OUTPUT4;
- контакт Vs з'єднують з додатним полюсом джерела живлення двигунів або просто з додатним полюсом джерела живлення, якщо схема і двигуни живляться від одного джерела. Простіше кажучи, цей контакт відповідає за живлення електродвигунів;
- контакт Vss з'єднують з додатним полюсом джерела живлення. Цей контакт забезпечує живлення самої мікросхеми;
- чотири контакти GND з'єднують з "землею", спільним дротом або від`ємним полюсом джерела живлення. Крім того, за допомогою цих контактів зазвичай забезпечують тепловідвід від мікросхеми, тому їх краще всього паяти на достатньо широку контактну поверхню.

Дана мікросхема має наступні технічні характеристики:

- напруга живлення двигунів (Vs) – 4,5...36 В;
- напруга живлення мікросхеми (Vss) – 5 В;
- допустимий струм навантаження – 600 мА (на кожен канал);
- піковий (максимальний) струм на виході – 1,2 А (на кожен канал);
- логічний "0" вхідної напруги – до 1,5 В;
- логічна "1" вхідної напруги – 2,3...7 В;
- швидкість перемикань до 5 кГц;
- захист від перегріву.

Інший спосіб керування двигунами постійного струму заснований на використанні мостових схем типу L298N (SGS– Thomson, RS636– 384). Це двоканальний пристрій, який працює від рівнів ТТЛ, з потужною напругою до 46 В та струмом до 2 А на кожен канал. Спрощену структуру мікросхеми зображено на рисунку 2.16.

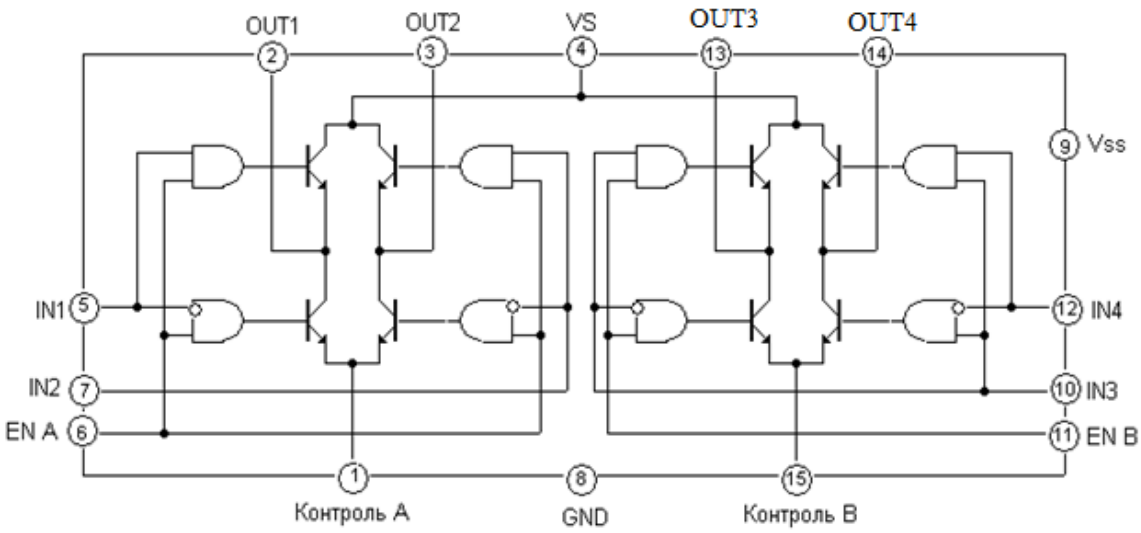


Рисунок 2.16 – Розташування видів і внутрішня структура мостового пристрою керування L298N

Через вивід Vs (контакт 4) надходить напруга живлення для двигуна. На вивід Vss (контакт 9) подається напруга живлення мікросхеми схеми (+5 В). На входи ENA і ENB (контакти 6 і 11) подаються керуючі ШІМ – сигнали. Входи IN1 і IN2 (контакти 5 і 7) керують напрямком обертання двигуна для першого каналу, а IN3 і IN4 – другого. Емітери транзисторів з'єднано для підключення зовнішніх контролюючих датчиків.

Коли на вході ENA низький рівень, входи заблоковані і двигун не обертається. Якщо на цей вхід подати високий рівень, входи відкриваються. Входи IN1 і IN2 керують режимами роботи двигуна наступним чином:

- IN1 – l, IN2 – 0 – двигун обертається за годинниковою стрілкою;
- IN1 – 0, IN2 – 1 – двигун обертається проти годинникової стрілки;
- INI = IN2= 0 двигун не обертається.

Якщо, наприклад, вихідний струм, що надходить безпосередньо на обмотку двигуна постійного струму досягає величини до 5 А, то на вихід схеми ми ставимо додаткові пари транзисторів, шунтуючи їх також захисними діодами. Таким чином остаточна схема буде вигляд, показаний на рисунку 2.17.

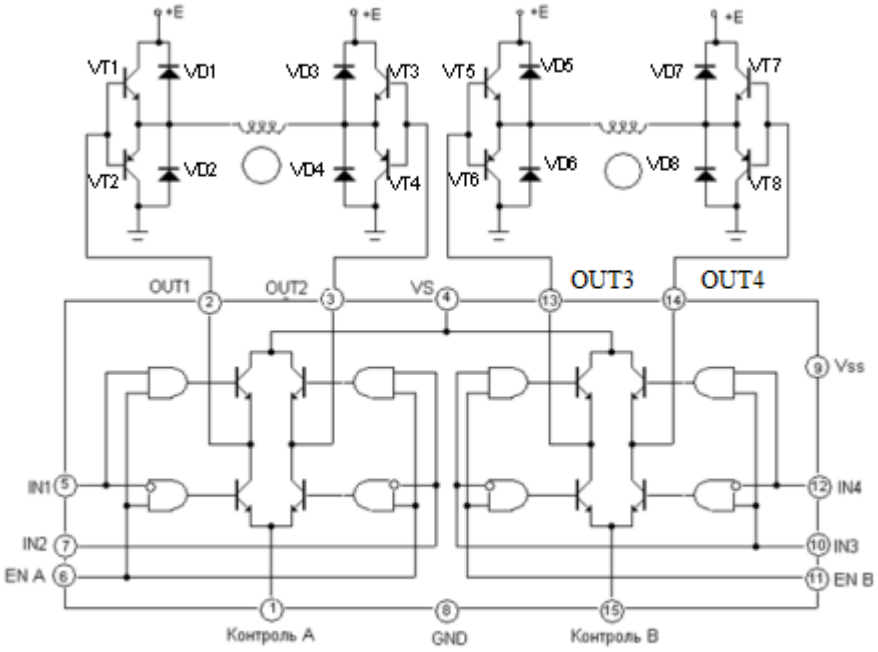


Рисунок 2.17 – Остаточна схема драйвера ШІМ

2.5 Захисні діоди

Як видно з рисунка 2.17 схема містить 8 захисних діодів VD1… VD8. Ці діоди призначені для захисту транзисторних ключів (VT1...VT8) від додатних (VT1, VT3, VT5, VT7) і від`ємних (VT2, VT4, VT6, VT8) паразитних імпульсів досить високої амплітуди, які з’являються на обмотках двигуна при комутації обмоток. Додатні імпульси виникають при запиранні (вимиканні) ключів, а від`ємні – при включенні. Механізм виникнення цих імпульсів описаний нижче.

Коли сила струму в котушці змінюється, змінюється і магнітний потік всередині котушки, який збуджує у ній електрорушійну силу (ЕРС) індукції (ЕІ). Ця ЕРС протидіє зміні магнітного потоку (правило Ленца). Якщо, наприклад, струм у котушці різко збільшується (включення ключа), то наростаючий магнітний потік індукує в котушці ЕРС: ЕІ, під дією якої виникає струм, протилежний первісному струму і прагне загальмувати його. При цьому на ключі виникає від`ємний імпульс (рисунок 2.18, а) достатньо великої амплітуди, який може вивести ключ з ладу.

Якщо ж струм в котушці різко зменшується (при виключенні ключа), то убиваючий магнітний потік збуджує ЕРС індукції: ЕІ, яка створює струм, спрямований аналогічно вихідного струму, що підтримує в котушці початковий струм. При цьому на ключі виникає додатний імпульс (рисунок 2.18, б) достатньо великої амплітуди, який може вивести ключ з ладу.

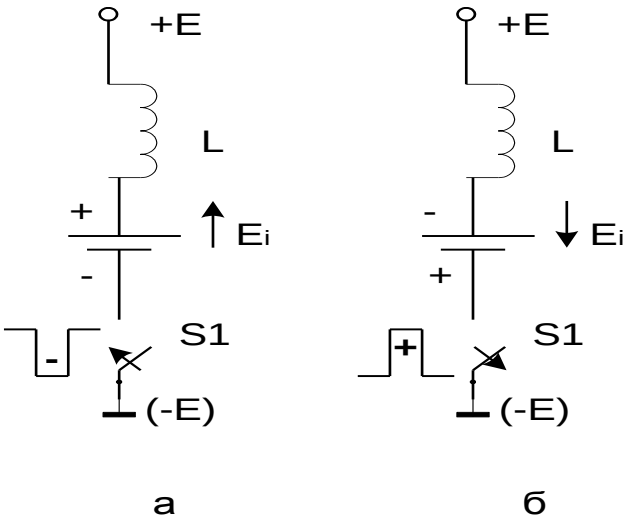


Рисунок 2.18 – Виникнення паразитних імпульсів при комутації ключів: а – від`ємний імпульс при включенні ключа; б – додатний імпульс при відключенні ключа

2.6 Приклад програмування ШІМ–модуля

2.6.1 Вхідні дані

Програмування ШІМ–модуля треба виконати та основі таких вхідних даних:

- тип МК– ра: АТ mega 128;
- номер таймера: Т/С 1;
- інвертований ШІМ– сигнал сформувати на виході РВ5;
- частота ШІМ– сигналу: f_{PB5} =25 кГц;
- частота тактового сигналу підсистеми введення/виведення: $f_{CLKI/0}$= 16 МГц ;

– режим роботи модуля ШІМ: Phase and Frequency Correct PWM (PFCPWM);
– шпаруватість ШІМ– сигналу: Q_{PB5}=2.

2.6.2 Завдання

Розрахувати:
– коефіцієнт ділення переддільника: Кділ = N;
– модуль лічби: TOP;
– період ШІМ– сигналу: T_{PB5};
– тривалість імпульсу ШІМ– сигналу: $t_{імпPB5}$.

Написати мовою Асемблер фрагмент програми, який забезпечує формування ШІМ– сигналу згідно вхідних даних.

На рисунку 2.19 наведено формування ШІМ– сигналу в режимі Phase and Frequency Correct PWM.

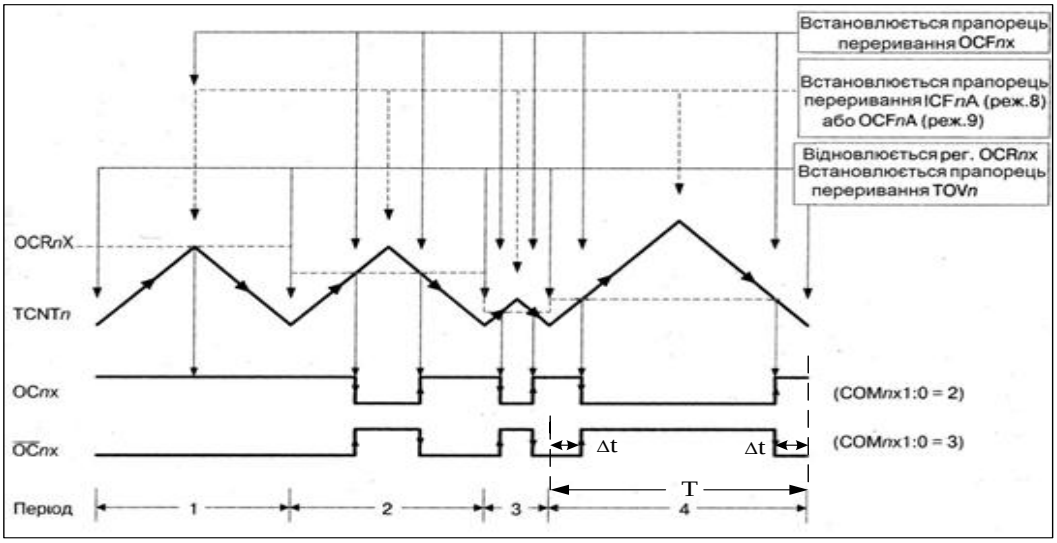


Рисунок 2.19 – Формування ШІМ– сигналу в режимі Phase and Frequency Correct PWM

2.6.3 Рішення завдання

Величина частоти ШІМ– сигналу f_{PL3} визначається виразом [2]:

$$f_{PB5} = \frac{f_{CLK\ I/O}}{N(TOP+1)}. \quad (2.1)$$

Згідно таблиці 2.1 обираємо режим роботи номер 8, тоді значення модуля лічби TOP визначається вмістом регістра ICR1. Значення f_{PB5} задано в умові і дорівнює 25кГц. Звідси шукаємо значення ICR1=TOP, згідно формули (1) при N=8:

$$ICR1 = \frac{f_{CLK\,I/O}}{2*N*f_{PB5}} = \frac{16*10^6}{2*8*25*10^3} = 40\ .$$

Обираємо значення ICR1=40=00101000B=$0028.

Період ШІМ– сигналу:

$$T_{PB5} = \frac{1}{f_{PB5}} = \frac{1}{25000} = 40 \text{ мкс}.$$

Потрібна тривалість імпульсу при шпаруватості ШІМ – сигналу Q_{PB5}=2:

$$t_{імпPB5} = \frac{T_{PB5}}{Q} = \frac{40 \text{ мкс}}{2} = 20 \text{ мкс}.$$

Згідно з рисунком 2.18

$$t_{імпPB5} = T_{PB5} - 2\,\Delta\,t. \qquad (2.2)$$

$$\Delta\,t = \frac{T_{PB5} - t_{імпPB5}}{2} = \frac{40-20}{2} = 10 \text{ мкс}.$$

$$\Delta\,t = T_{CLKI/O} * N * OCR1A, \qquad (2.3)$$

де OCR1A – регістр порівняння таймера 1.

$$OCR1A = \frac{\Delta\,t}{T_{CLK\,I/O}*N} = \frac{10*10^6}{\frac{1}{16}*10^{-6}*8} = 20 = 00010100B = \$0014.$$

Шпаруватість:

$$Q = \frac{T_{PB5}}{T_{PB5} - 2\,\Delta\,t} = \frac{T_{PB5}}{T_{PB5} - 2\,T_{CLKI/O} * N * OCR1A} = \frac{1}{1 - \frac{2\,T_{CLKI/O} * N * OCR1A}{T_{PB5}}} =$$

$$= \frac{1}{1-2*\frac{1}{f_{CLKI/O}}*N*OCR1A*f_{PB5}}. \qquad (2.4)$$

Таблиця 2.1 – Режими роботи 16– розрядних таймерів/лічильників

Номер режиму	WGMn3	WGMn2	WGMn1	WGMn0	Режим роботи таймера/лічильника Tn	Модуль лічби (TOP)	Оновлення регістрів TOVn	Момент установки прапорця TOVn
0	0	0	0	0	Normal	$FFFF	Негайно	$FFFF
1	0	0	0	1	Phase correct PWM, 8– розрядний	$00FF	При TOP	$0000
2	0	0	1	0	Phase correct PWM, 9– розрядний	$01FF	При TOP	$0000
3	0	0	1	1	Phase correct PWM, 10– розрядний	$03FF	При TOP	$0000
4	0	1	0	0	CTC (скидання при збігу)	OCRnA	Негайно	$FFFF
5	0	1	0	1	Fast PWM, 8– розрядний	$00FF	При TOP	При TOP
6	0	1	1	0	Fast PWM, 9– розрядний	$01FF	При TOP	При TOP
7	0	1	1	1	Fast PWM, 10– розрядний	$03FF	При TOP	При TOP
8	1	0	0	0	Phase and Frequency Correct PWM	ICRn	$0000	$0000
9	1	0	0	1	Phase and Frequency Correct PWM	OCRnA	$0000	$0000
10	1	0	1	0	Phase correct PWM	ICRn	При TOP	$0000
11	1	0	1	1	Phase correct PWM	OCRnA	При TOP	$0000
12	1	1	0	0	CTC (скидання при збігу)	ICRn	Негайно	$FFFF
13	1	1	0	1	Зарезервовано	_	_	_
14	1	1	1	0	Fast PWM	ICRn	При TOP	При TOP
15	1	1	1	1	Fast PWM	OCRnA	При TOP	При TOP

Примітка. n = 1, 3, 4 або 5.

Для перевірки використаємо OCR1A=20, Q повинне бути рівним 2:

$$Q = \frac{1}{1-2*\frac{1}{f_{CLKI/O}}*N*OCR1A*f_{PB5}} = \frac{1}{1-2*\frac{1}{16*10^6}*8*20*25*10^3} = 2.$$

Тепер дослідимо характер зміни Q при збільшенні OCR1A на 5 і при зменшенні на 5.

$$Q_{(OCR1A=25)} = \frac{1}{1-2*\frac{1}{16*10^6}*8*25*25*10^3} = 2{,}667.$$

$$Q_{(OCR1A=15)} = \frac{1}{1-2*\frac{1}{16*10^6}*8*15*25*10^3} = 1{,}6.$$

2.6.4 Розробка фрагменту програми

2.6.4.1 Зупинка таймера

Для зупинки таймера треба в регістр керування TCCR1B (рисунок 2.20), адреса якого згідно таблиці 2.2 дорівнює $004E, записати керуюче слово КС1:

Тоді програма має вид:

7 р.	6р.	5р.	4р.	3р.	2р.	1р.	0р.
ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10
0	0	0	0	0	0	0	0В=$00=КС1

```
LDI R18, $00; R18 ←КС1=$00
```

LDI R27, $00; R27 ← $01
LDI R26, $4E; R26 ← $21 } X(R27, R26) ← $004E;

```
ST X, R18; TCCR1B ← R18 = $00, зупинка таймера.
```

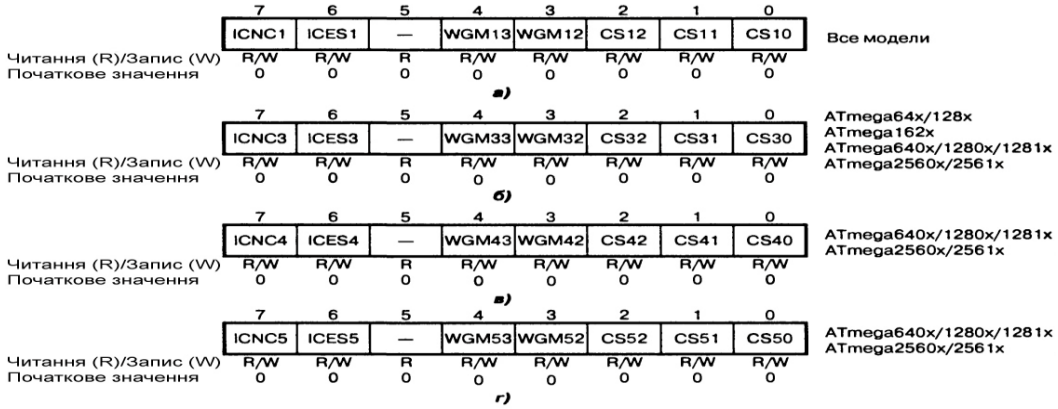


Рисунок 2.20 – Формат регістрів TCCR1B (а), TCCR3B (б), TCCR4B (в), TCCR5B (г)

Таблиця 2.2– Регістри 16– розрядних таймерів/лічильників

Регістр	Адреса	ATmega8515x	ATmega8535x	ATmega8x	ATmega16x/32x	ATmega64x/128x	ATmega48x/88x/168x	ATmega162x	ATmega164x/324x/644x	ATmega165x	ATmega325x/3250x, ATmega645x/6450x	ATmega640x, ATmega1280x/1281x, ATmega2560x/2561x
TCCR1A	$2F($4F)	•	•	•	•	•		•				
	($80)						•		•	•	•	•
TCCR1B	$2E ($4E)	•	•	•	•	•		•				
	($81)						•		•	•	•	•
TCCR1C	($7A)					•						
	($82)						•		•	•	•	•
TCNT1	$2D:$2C ($4D:$4C)	•	•	•	•	•		•				
	($85:$84)						•		•	•	•	•
OCR1A	$2B:$2A ($4B:$4A)	•	•	•	•	•		•				
	($89:$88)						•		•	•	•	•
OCR1B	$29:$28 ($49:$48)	•	•	•	•	•		•				
	($8B:$8A)						•		•	•	•	•
OCR1C	($79:$78)					•						
	($8D:$8C)											•
ICR1	$27:$26 ($47:$46)		•	•	•	•						
	$25:$24 ($45:$44)	•						•				
	($87:$86)						•		•	•	•	•
TCCR3A	($8B)					•		•				
	($90)											•
TCCR3B	($8A)					•		•				
	($91)											•

Продовження таблиці 2.2

Регістр	Адреса	ATmega8515x	ATmega8535x	ATmega8x	ATmega16x/32x	ATmega64x/128x	ATmega48x/88x/168x	ATmega162x	ATmega164x/324x/644x	ATmega165x	ATmega325x/3250x, ATmega645x/6450x	ATmega640x, ATmega1280x/1281x, ATmega2560x/2561x
TCCR3C	($8C)					•						
	($92)											•
TCNT3	($89:$88)					•		•				
	($95:$94)											•
OCR3A	($87:$86)					•		•				
	($99:$98)											•
OCR3B	($85:$84)					•		•				
	($9B:$9A)											•
OCR3C	($83:$82)					•						
	($9D:$9C)											•
ICR3	($81:$80)					•		•				
	($97:$96)											•
TCCR4A	($A0)											•
TCCR4B	($A1)											•
TCCR4C	($A2)											•
TCNT4	($A5:$A4)											•
OCR4A	($A9:$A8)											•
OCR4B	($AB:$AA)											•
OCR4C	($AD:$AC)											•
ICR4	($A7:$A6)											•
TCCR5A	($120)											•
TCCR5B	($121)											•
TCCR5C	($122)											•
TCNT5	($125:$124)											•
OCR5A	($129:$128)											•
OCR5B	($12B:$12A)											•
OCR5C	($12D:$12C)											•
ICR5	($127:$126)											•

2.6.4.2 Завантаження регістра TCCR1A

Згідно рисунку 2.21 формат регістра TCCR1A МК– ра AT Mega 128 має вид:

7 р.	6р.	5р.	4р.	3р.	2р.	1р.	0р.	
COM1A1	COM1A0	COB1B1	COM1BO	COM1C1	COM1C0	WGM11	WGM10	KC2
1	1	0	0	0	0	0	0 B=$C0	

Для нашої задачі для формування інвертованого ШІМ– сигналу згідно таблиці 2.3 необхідно встановити біти COM1A1=COM1A0=1.

Таблиця 2.3 – Поведінка виводу OCnA/OCnB/OCnC в режимі Phase and Frequency Correct PWM

COMnx1	COMnx0	Опис
0	0	Таймер/лічильник Tn відключено від виводу OCnA/OCnB/OCnC
0	1	WGMn3 = «0»: таймер/лічильник Tn відключено від виводу OCnA/OCnB/OCnC; WGMn3 = «1»: стан виводу OCnA змінюється на протилежний
1	0	Скидається в "0" при прямій лічбі і встановлюється в "1" при зворотній лічбі (неінвертований ШІМ – сигнал)
1	1	Встановлюється в "1" при прямій лічбі і скидається в "0" при зворотній лічбі (інвертований ШІМ – сигнал)

Примітка. n = 1, 3, 4 або 5; x= A, B або C.

Для програмування режиму роботи 8 згідно таблиці 2.1 необхідно запрограмувати біти: WGM11=0; WGM10=0.

Інші біти регістра TCCR1A у нашому прикладі не використовуються. Тому запишемо в них нулі.

Тоді керуюче слово КС2=11000000B=$C0.

Згідно таблиці 2.2 адреса регістра TCCR1A=$004F.

Тоді програма має вид:

```
LDI R17, $C0; R17←KC2=$C0
```

$$\left.\begin{array}{l}\text{LDI R29, \$00; R27} \leftarrow \text{\$00}\\ \text{LDI R28, \$4F; \$26} \leftarrow \text{\$4F}\end{array}\right\} \text{Y(R29, R28)} \leftarrow \text{\$004F;}$$

```
ST Y, R17; TCCR5A←R17=$C0.
```

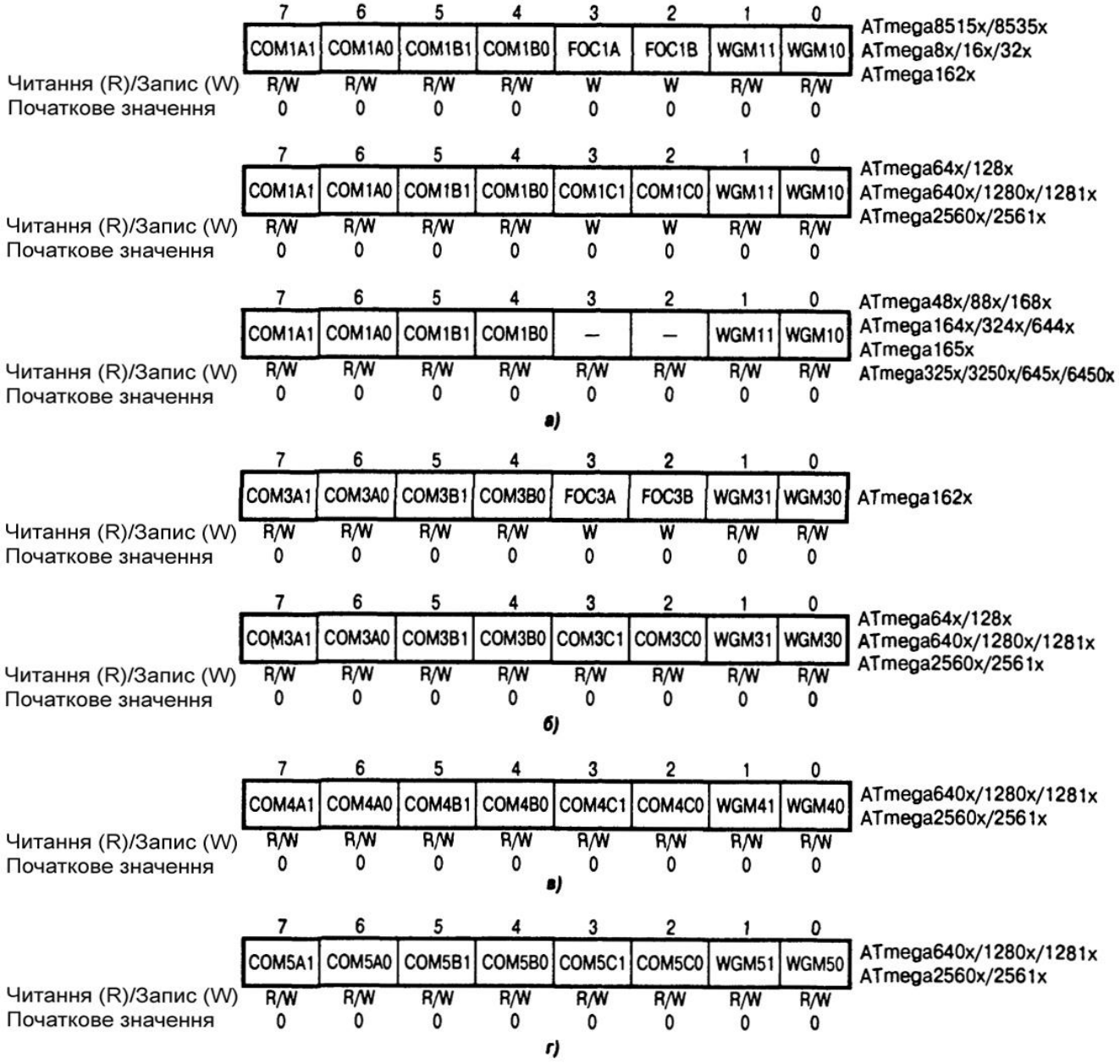


Рисунок 2.21 – Формат регістрів TCCR1A (а), TCCR3A (б), TCCR4A (в), TCCR5A (г)

2.6.4.3 Програмування лінії PB5 на виведення

Для програмування лінії PB5 на виведення необхідно встановити 5– й розряд регістра DDRB в одиницю. Згідно таблиці 2.4 адреса регістра: $0037.

Таблиця 2.4 – Регістри портів введення/виведення

Порт	Регістр	ATmega8515x	ATmega8535x	ATmega8x	ATmega16x	ATmega162x	ATmega64x, ATmega128x	ATmega48x/88x/168x	ATmega164x/324x/644x	ATmega165x, ATmega325x/645x	ATmega3250x/6450x	ATmega1281x/2561x	ATmega640x, ATmega1280x/2560x
A	PORTA	$1B ($3B)		–	$1B ($3B)			–	$02 ($22)				
	DDRA	$1A ($3A)		–	$1A ($3A)			–	$01 ($21)				
	PINA	$19 ($39)		–	$19 ($39)			–	$00 ($20)				
B	PORTB	$18 ($38)						$05 ($25)					
	DDRB	$17 ($37)						$04 ($24)					
	PINB	$16 ($36)						$03 ($23)					
C	PORTC	$15 ($35)						$08 ($28)					
	DDRC	$14 ($34)						$07 ($27)					
	PINC	$13 ($33)						$06 ($26)					
D	PORTD	$12 ($32)						$0B ($2B)					
	DDRD	$11 ($31)						$0A ($2A)					
	PIND	$10 ($30)						$09 ($29)					
E	PORTE	$07 ($27)		–	–	$07 ($27)	$03 ($23)	–	–	$0E ($2E)			
	DDRE	$06 ($26)		–	–	$06 ($26)	$02 ($22)	–	–	$0D ($2D)			
	PINE	$05 ($25)		–	–	$05 ($25)	$01 ($21)	–	–	$0C ($2C)			

Продовження таблиці 2.4

Порт	Регістр	ATmega8515x	ATmega8535x	ATmega8x	ATmega16x	ATmega162x	ATmega64x, ATmega128x	ATmega48x/88x/168x	ATmega164x/324x/644x	ATmega165x, ATmega325x/645x	ATmega3250x/6450x	ATmega1281x/2561x	ATmega640x, ATmega1280x/2560x
F	PORTF	$03 ($23)	–	–	–	–	($62)	–	–	$11 ($31)			
	DDRF	$02 ($22)	–	–	–	–	($61)	–	–	$10 ($30)			
	PINF	$01 ($21)	–	–	–	–	$00 ($20)	–	–	$0F ($2F)			
G	PORTG	–	–	–	–	–	($65)	–	–	$14 ($34)			
	DDRG	–	–	–	–	–	($64)	–	–	$13 ($33)			
	PING	–	–	–	–	–	($63)	–	–	$12 ($32)			
H	PORTH	–	–	–	–	–	–	–	–	–	($DA)	–	($102)
	DDRH	–	–	–	–	–	–	–	–	–	($D9)	–	($101)
	PINH	–	–	–	–	–	–	–	–	–	($D8)	–	($100)
J	PORTJ	–	–	–	–	–	–	–	–	–	($DD)	–	($105)
	DDRJ	–	–	–	–	–	–	–	–	–	($DC)	–	($104)
	PINJ	–	–	–	–	–	–	–	–	–	($DB)	–	($103)
K	PORTK	–	–	–	–	–	–	–	–	–	–	–	($108)
	DDRK	–	–	–	–	–	–	–	–	–	–	–	($107)
	PINK	–	–	–	–	–	–	–	–	–	–	–	($106)
L	PORTL	–	–	–	–	–	–	–	–	–	–	–	($10B)
	DDRL	–	–	–	–	–	–	–	–	–	–	–	($10A)
	PINL	–	–	–	–	–	–	–	–	–	–	–	($109)

Тоді програма має вид:

```
LDI R31, $00; R31← $00; R31, R30 (Z)- адреса статичної пам`яті
даних- регістра DDRB: $ 0037;
LDI R30, $37; R30 ← $37 ;
LD R19, Z; R19← DDRB
ORI R19, $08; R19←$08=00100000B+R19, R19.5←1
```

```
ST Z, R19; DDRB←R19, DDRB. 5 ←1.
```

2.6.4.4 Завантаження регістра ICR1

Згідно таблиці 2.2 16– розрядний регістр ICR1 має адресу:

$0047(СБ) : $0046(МБ). В ICR1 треба завантажити: 40 = 00101000B = $0028 (див. вище).

Тоді програма має вид:

```
LDI R23, $00; R23← $00;
LDI R27, $00; R27 ← $00;
LDI R26, $47; R26 ← $27 } X(R27, R26) ← $0047;
ST X, R23; СБ ICR1←R23 = $00.
LDIR24, $28; R24← $28
LDI R29, $00; R29 ← $00
LDI R28, $46; R28 ← $46 } Y(R29, R28) ← $0046;
ST Y, R24; МБ ICR1←R24 = $28.
```

2.6.4.5 Завантаження регістра OCR1A

Згідно таблиці 2.2, 16– розрядний регістр OCR1A має адресу: $004B(СБ): $004A(МБ). В OCR1A треба завантажити: 20 = 00010100B = $0014 (див. вище).

Тоді програма має вид:

```
LDI R24, $00; R24← $00;
LDI R31, $00; R31 ← $00;
LDI R30, $4B; R30 ← $4B } Z(R31, R30) ← $004B;
ST Z, R24; СБ OCR1A←R24 = $00.
LDI R25, $14; R25 ← $14;
LDI R31, $00; R31 ← $00;
LDI R30, $4A; R30 ← $4A } Z(R31, R30) ← $004A;
ST Z, R25; МБ OCR1A ← R25 = $14.
```

2.6.4.6 Програмування Кділ=N=8, режиму роботи номер 8 та запуск таймера

Вище при завантаженні регістра TCCR1A було записано WGM11=0 та WGN10=0, що разом з двома бітами WGM12=0 та WGM13=1 регістра

TCCRB програмують режим роботи №8 (таблиця 2.1).

Для програмування Кділ=8 (табл. 2.5) та запуску таймера також використовують регістр TCCR1B (рисунок 2.20).

7 р.	6р.	5р.	4р.	3р.	2р.	1р.	0р.
ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10
0	0	0	1	0	0	1	0B = $12

(режим 8) (Кділ = 8)

Таблиця 2.5 – Вибір джерела тактового сигналу таймерів/лічильників Tn

CSn2	CSn1	CSn0	Джерело тактового сигналу	
			Т3 в моделях ATmega162x	Інші
0	0	0	Таймер/лічильник зупинений	Таймер/лічильник зупинений
0	0	1	$f_{clkI/0}$	$f_{clkI/0}$
0	1	0	$f_{clkI/0}/8$	$f_{clkI/0}/8$
0	1	1	$f_{clkI/0}/64$	$f_{clkI/0}/64$
1	0	0	$f_{clkI/0}/256$	$f_{clkI/0}/256$
1	0	1	$f_{clkI/0}/1024$	$f_{clkI/0}/1024$
1	1	0	$f_{clkI/0}/16$	Вивід Tn, лічба виконується за спадаючим фронтом
1	1	1	$f_{clkI/0}/32$	Вивід Tn, лічба виконується за наростаючим фронтом

Примітка. n = 1, 3, 4 або 5.

2.7 Схема алгоритму роботи та керуюча програма
2.7.1 Схема алгоритму роботи

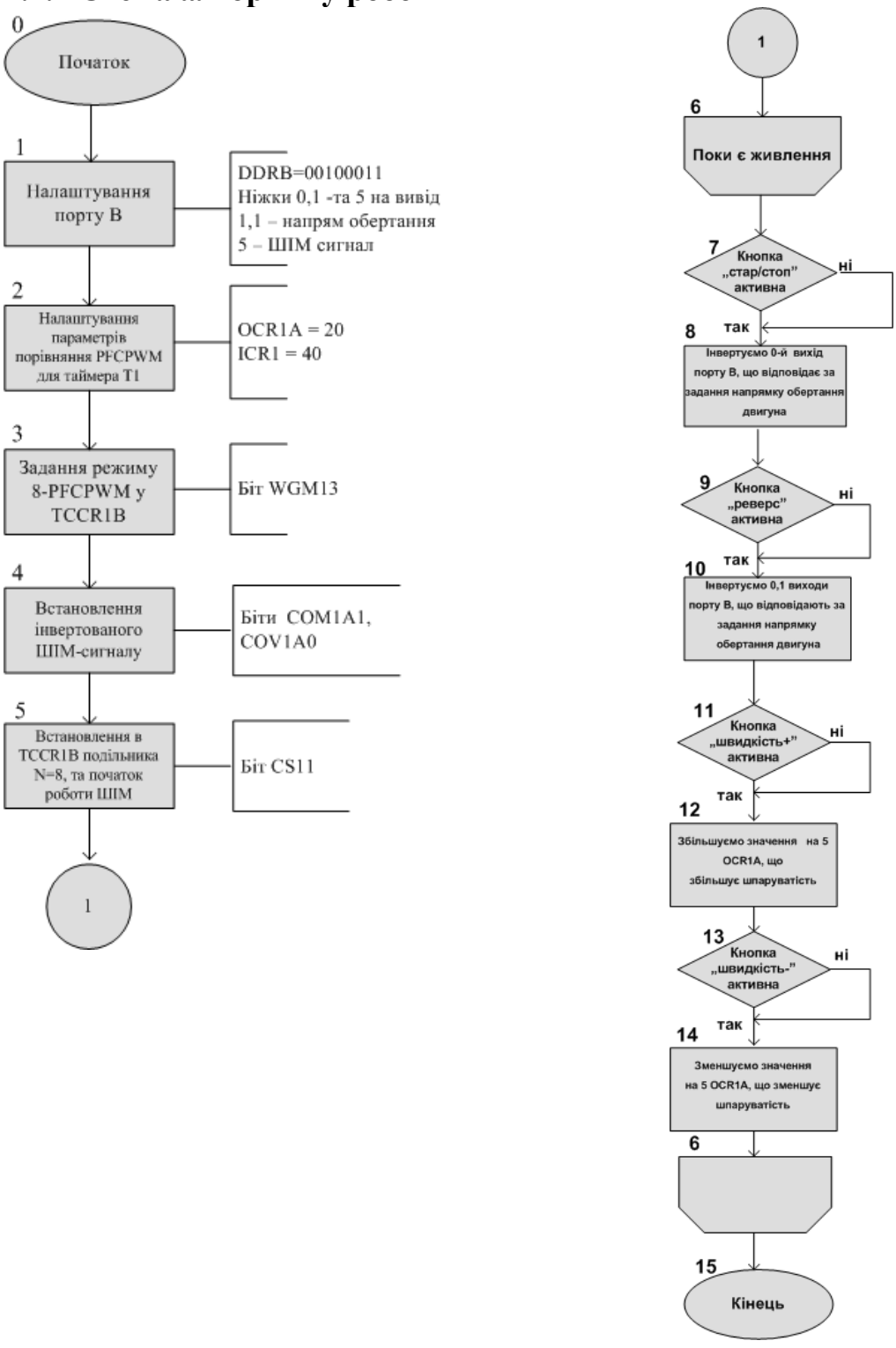


Рисунок 2.22 – Схема алгоритму роботи

2.7.2 Керуюча програма мовою С

```
1  #include <inttypes.h>
2  #include <avr/io.h>
3  #include <avr/interrupt.h>
4  #include <avr/sleep.h>
5  #include <util/delay.h>
6  #include F_CPU 16000000L
7  int main(void) // 0
8  {
9  // 1: Init port B
10 DDRB = 0b00100011;
11 PORTB= 0b00000000;
12 // 2: Init 16- bit timer 1 values
13 OCR1A = 20;
14 ICR1 = 40;
15 int delay = 1;
16 // Init Phase and frequency correct PWM
17 TCCR1B = _BV(WGM13);  // 3
18 TCCR1A = 0;
19 TCCR1A |= _BV(COM1A0);// 4
20 TCCR1A |= _BV(COM1A1);
21 TCCR1B |= _BV(CS11);  // 5
22 char
   flag_start_stop=0,flag_reverse=0,flag_speedup=0,flag_speeddown=0;

23 while(1) // 6
24 {
25 // 7: Start- Stop button action
26 if(!(PINB&4))
27 { flag_start_stop=1; _delay_ms(10); }
28 if(( flag_start_stop==1 )&&(PINB&4))
29 {PORTB^=1; flag_start_stop=0; } // 8
30 // 9: Reverse button action
31 if(!(PINB&8))
32 { flag_reverse=1; _delay_ms(10); }
33 if(( flag_reverse==1 )&&(PINB&8))
34 {PORTB^=3; flag_reverse=0; }    // 10
35 // 11: Speed -  button action
36 if(!(PINB&16))
37 { flag_speeddown=1; _delay_ms(10); }
38 if(( flag_speeddown==1 )&&(PINB&16))
39 { if (OCR1A!=40) OCR1A+=delay; flag_speeddown=0; } // 12
40 // 13: Speed + button action
41 if(!(PINB&64))
42 { flag_speedup=1; _delay_ms(10); }
43 if(( flag_speedup==1 )&&(PINB&64))
44 {
45 if (OCR1A!=0)OCR1A- =delay; // 14
46 flag_speedup=0;
47 }
48 } // 6: End of loop
49 } // 15: End of proram
```

Нижче наведено деякі пояснення окремих фрагментів програми, яку наведено вище.

2.7.2.1 Ініціалізація порту PB

Біт DDxn регістра DDRB визначає напрям передачі даних через контакт введення/виведення. Якщо цей біт встановлено в 1, то n– й вивід порту являє собою вихід, якщо ж цей біт скинутий у 0, то цей вивід функціонує як вхід. Загальний вигляд регістра DDRx показано на рисунку 2.23.

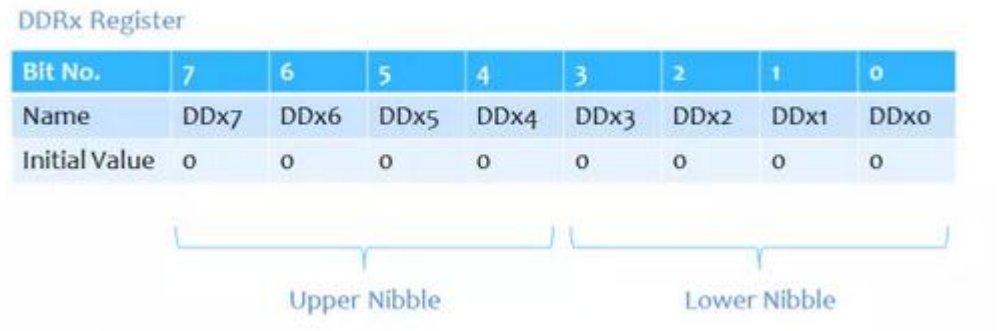


Рисунок 2.23– Вигляд регістра DDRx

Ініціалізуємо виводи 0,1 на виведення для передачі сигналів на входи драйвера ШІМ, за допомогою яких будемо керувати напрямом обертання двигуном постійного струму. Вивід 5 ініціалізуємо також на вихід. З нього будемо знімати ШІМ– сигнал, який буде керувати драйвером.

Тоді DDRB = 0b00100011.

Через можливе близьке знаходження ДПС до контролера і можливість наводок вбудовані підтягуючі резистори ємністю 100кОм використовувати не будемо, а використаємо зовнішні, ємністю по 10кОм.

Біт PORTBn регістра PORTB виконує подвійну функцію. Якщо вивід функціонує як вихід, то цей біт визначає стан виходу порту. Коли він встановлений в 1, на виході встановлюється напруга високого рівня, коли ж він скинутий в 0, на виході встановлюється напруга низького рівня.

Якщо ж вивід функціонує як вхід, то біт PORTBn визначає стан внутрішнього підтягуючого резистора для даного виводу. При встановленні

біта PORTBn в 1 підтягуючий резистор підключається між виводом мікроконтролера та лінією живлення. Загальний вигляд регістра PORTx показано на рисунку 2.24

Проініціалізуємо регістр PORTB нулями:

PORTB=0b00000000.

PORTx Register

Bit No.	7	6	5	4	3	2	1	0
Name	Px7	Px6	Px5	Px4	Px3	Px2	Px1	Px0
Initial Value	0	0	0	0	0	0	0	0

Upper Nibble Lower Nibble

Рисунок 2.24 – Вигляд регістра PORTx

2.7.2.2 Запуск та зупинка двигуна постійного струму

Для керування запуском/зупинкою двигуна постійного струму з допомогою драйвера та мікроконтролера, нам потрібно читати вхід з защіпки порту, до якого підключена відповідна кнопка. Для цього будемо перевіряти біт PX2 регістра PINB. Загальний вигляд регістра PINx показано на рисунку 2.25

PINx Register

Bit No.	7	6	5	4	3	2	1	0
Name	Px7	Px6	Px5	Px4	Px3	Px2	Px1	Px0
Initial Value	0	0	0	0	0	0	0	0

Upper Nibble Lower Nibble

Рисунок 2.25 – Вигляд регістра PINx

Якщо на вхід PB2 порту буде подаватись високий рівень напруги, то біт PX2 прийматиме значення 1, якщо низький рівень напруги то біт PX3 прийматиме значення 0.

Щоб запобігти зайвим спрацюванням при натисканні кнопки керування, введемо індикатор натискання та затримку, і будемо міняти

керуючу програму вже після відпускання кнопки. Одразу після ініціалізації таймера, ШІМ– сигнал вже поступає на схему драйвера, проте на портах, через які живиться двигун низький рівень напруги. Після натискання на кнопку ми встановлюємо індикатор і робимо паузу на 10 мс. Наступна умова спрацює тоді, коли прапорець встановлений, а кнопку ми вже відпустили. Після спрацювання цієї умови ми скидатимемо прапорець, та будемо інвертувати перший біт регістра PORTB, що дасть нам змогу почергово встановлювати високий та низький рівні напруги на виході PB0 який виступає у нас одним з виходів, які ми використовуватимемо для живлення двигуна постійного струму. Нижче приведено фрагмент коду на С, який все це реалізовує.

```
//Start- Stop button action
 if(!(PINB&4))
 { flag_start_stop=1; _delay_ms(10); }
 if(( flag_start_stop==1 )&&(PINB&4))
 {PORTB^=1; flag_start_stop=0; }
```

2.7.2.3 Зміна напрямку обертання двигуна постійного струму

Для зміни напрямку обертання двигуна постійного струму, або ввімкнення реверсу, потрібно слідкувати за станом кнопки керування, та за її натисканням, відповідно, змінювати напрям руху. Слідкувати за натисканням кнопки будемо за допомогою зчитування рівня напруги з защіпки порту, до якого підключена ця кнопка. Власне, дану кнопку було підключено до виводу PB3 мікроконтролера ATmega128, який налаштовано як вхід. Будемо читати рівень напруги на защіпці порту з біта PX3 регістра PINB. При натисненні на кнопку рівень напруги буде низький, тобто зніматимемо 0, а коли кнопка не натиснена то рівень напруги високий і зніматимемо 1.

Логіка керування буде наступною. Сторона, у яку обертається двигун залежить від потенціалів на входах IN1 та IN2 на драйвері, яким виступає схема з двома мостами L293D. Змінювати потенціали ми будемо два одразу,

лінію з низьким потенціалом будемо робити лінією з високим, а лінію з високим – лінією з низьким. Тут ми говоримо про лінії PB0 та PB1 мікроконтролера, які налаштовані як виходи. Як ми бачимо вище, при вмиканні було запрограмовано лінію порту PB0 як вихід з високим рівнем напруги. Після натискання кнопки, за допомогою операції побітового XOR ми будемо інвертувати перші два розряди регістра PORTB, який у випадку ліній виведення відповідає за рівень напруги на виході. Нижче наводимо фрагмент коду на C, який виконує вказані операції.

```
  //Reverse button action
if(!(PINB&8))
{ flag_reverse=1; _delay_ms(10); }
if(( flag_reverse==1 )&&(PINB&8))
{PORTB^=3; flag_reverse=0; }
```

2.7.2.4 Зміна швидкості обертання двигуна постійного струму

Для зміни швидкості обертання двигуна постійного струму у нас буде дві кнопки, перша буде збільшувати швидкість, а друга – зменшувати. Принцип, за яким ми будемо змінювати швидкість, ґрунтується на властивості ШІМ– сигналу. Двигун до джерела живлення підключаємо через ключі, мостовою схемою, як показано на рисунку 2.12.

В даному випадку для нас важливі комбінації, коли є різниця потенціалів. Це наступні комбінації ключів: 1 відкритий, 2 закритий, 3 закритий, 4 відкритий; 1 закритий, 2 відкритий, 3 відкритий, 4 закритий.

Також важливими є комбінації, коли немає різниці потенціалів, і ротор електродвигуна не обертається. вони наступні: 1 відкритий, 2 відкритий, 3 закритий, 4 закритий; 1 закритий, 2 закритий, 3 відкритий, 4 відкритий.

Тобто, в загальному випадку, коли ми відкриваємо ключі за діагоналлю – наш двигун починає обертатись.

Для того, щоб регулювати швидкість обертання ми будемо вмикати ці ключі на час X та вимикати на час Y з великою швидкістю. Як це буде виглядати, представлено на рисунку 2.4

Швидкість обертання двигуна залежить від напруги, яка на нього подається. На рисунку 2.4 ця напруга позначається, як еквівалентна постійна напруга. З цього рисунка ми також бачимо, що використовуючи цифрові сигнали логічного “0” та логічної “1” ми можемо отримувати своєрідну еквівалентну напругу, яка відрізняється від напруги логічних рівнів.

Регулюючи шпаруватість – відношення періоду слідування імпульсів до їх довжини, ми змінюємо еквівалентну постійну напругу та регулюємо швидкість обертання нашого двигуна постійного струму.

Тепер власне перейдемо до кнопок. Спочатку розберемо схему збільшення швидкості. Кнопка підключена до виводу PB4 мікроконтролера, який в свою чергу запрограмовано як вхід. Знімати значення цього входу ми будемо за допомогою 8 бітного регістра PINB, а точніше – за допомогою біта PB4. Коли кнопка не натиснута цей біт має значення логічної 1. Коли кнопка натискається, цей біт приймає значення логічного 0. Саме це ми будемо перевіряти у програмі керування мікроконтролером ATmega128. При виявленні логічного нуля ми будемо встановлювати індикатор натискання та очікувати 10 мс. Наступна умова у нас виконається, якщо індикатор натискання встановлений в 1, і на защіпці PB4 порту логічна одиниця, тобто кнопка вже відпущена. У цьому випадку ми збільшуватимемо значення 8 бітного регістра OCR1A на величену, яку записано у змінну delay, але не більше, ніж 20.

При цій постійній еквівалентній напрузі наш двигун постійного струму обертається з максимальною швидкістю, яку ми можемо досягти у нашому моделюванні. Нижче наведемо код на C, який відповідає за збільшення швидкості обертання двигуна постійного струму при натисканні на відповідну кнопку.

```
//Speed + button action
 if(!(PINB&16))
 { flag_speedup=1; _delay_ms(10); }
 if(( flag_speedup==1 )&&(PINB&16))
 { if (OCR1A!=40) OCR1A+=delay; flag_speedup=0; }
```

Зменшення швидкості обертання двигуна постійного струму відбувається за схожим принципом. Ми перевіряємо рівень напруги на виводі PB6 який запрограмовано на вхід. Програмно читаємо цей рівень напруги з біта PX6 регістра PINB. Коли кнопка не натиснена, біт дорівнює 1. Коли кнопка натискається, значення біта змінюється з 1 на 0, і залишається таким до того часу, доки кнопка натиснена. Саме для того, щоб не відбувались багаторазові зміни при натисканні на кнопку, у програмі ми використовуємо ключі, які скидаються лише коли кнопки відпускаються.

Програмно читаючи значення біта PX6 регістра PINB, ми змінюємо вміст регістра OCR1A, зменшуючи його на величену, яку записано у змінну delay, але не менше 0. Код, який це реалізовує приведено нижче.

```
//Speed -  button action
   if(!(PINB&64))
   { flag_speeddown=1; _delay_ms(10); }
   if(( flag_speeddown==1 )&&(PINB&64))
   {
      if (OCR1A!=0)OCR1A- =delay;
      flag_speeddown=0;
   }
```

3 МОДЕЛЮВАННЯ МОДУЛЯ АЦП МІКРОКОНТРОЛЕРІВ СІМ`Ї AVR

3.1 Особливості модуля АЦП у складі мікроконтролера ATmega32

Опис архітектури модуля АЦП у складі мікроконтролерів AVR наведено у [1…3]. Нижче наведено скорочений опис архітектури модуля АЦП у складі мікроконтролера ATmega32, який використано у моделі.

Модуль 10–розрядного АЦП послідовного наближення входить до складу моделей сімейства Mega, за винятком ATmega8515x/162x. Основні параметри модуля АЦП наступні:

- абсолютна похибка: ±2 МЗР (молодшого значущого розряду);
- інтегральна нелінійність: ±0.5 МЗР[1];
- швидкодія: до 15 тис. вибірок/с [1].

На вході модуля АЦП моделей є 8–канальний аналоговий мультиплексор, що надає в розпорядження користувача 8 каналів з однополярними (несиметричними) входами.

У більшості моделей входи АЦП можуть також поєднуватися попарно для формування різного числа каналів з диференціальним входом.

При цьому в деяких каналах є можливість 10– і 200–кратного попереднього підсилення вхідного сигналу. При коефіцієнтах підсилення 1x та 10x діюча роздільна здатність АЦП складає 8 розрядів, а при коефіцієнті підсилення 200x – 7 розрядів.

Як джерело опорної напруги для АЦП може використовуватись як напруга живлення мікроконтролера, так і внутрішнє або зовнішнє джерело опорної напруги.

Модуль АЦП може функціонувати у двох режимах:

- режим одиночного перетворення, коли запуск кожного перетворення ініціюється користувачем;

[1] МЗР в зарубіжній літературі позначається LSB (Least Significant Bit – молодший значущий розряд), аналогічно, MSB (Most Significant Bit – старший значущий розряд) відповідає позначенню СЗР.

– режим безперервного перетворення, коли запуск перетворень виконується безперервно через певні інтервали часу.

Модуль АЦП містить пристрій вибірки–зберігання (ПВЗ), що підтримує під час перетворення напругу безпосередньо на вході АЦП на постійному рівні.

3.2 Опис функціональної схеми модуля АЦП

Функціональна схема модуля АЦП наведено на рисунку 3.1. У моделях ATmega8x і ATmega48x/88x/168x елементи і пов’язані з ними сигнали, які виділено на рисунку сірим кольором, відсутні, а неінвертуючий вхід компаратора з пристроєм вибірки–збереження підключений безпосередньо до виходу мультиплексора (показано пунктирною лінією).

Вхідний аналоговий сигнал зберігається в ПВЗ. За командою «Старт» за допомогою логіки перетворення (регістра зсуву) послідовно у часі кожен розряд ЦАП, починаючи зі старшого розряду, переводиться у положення 1. Компаратор зрівнює напругу на виході ЦАП з вхідною напругою: якщо напруга на вході більша, ніж напруга на виході ЦАП, то на виході компаратора високий рівень, а в тригер регістра зсуву записується 1, якщо напруга на вході менша, ніж напруга на виході ЦАП, то тригер переводиться у положення 0. В кінці перетворення у регістрі даних АЦП (ADCH/ADCL) буде двійковий код, який еквівалентний вхідній напрузі.

АЦП працює у двох режимах: однополярному та диференціальному. При однополярному режимі вхідний сигнал поступає на один із входів ADC0–ADC17. При диференціальному режимі використовуються шість входів мультиплексора (ADC0, ADC1, ADC2, ADC3). Різниця сигналу між цими входами підсилюється за допомогою диференціального підсилювача і поступає на вхід компаратора.

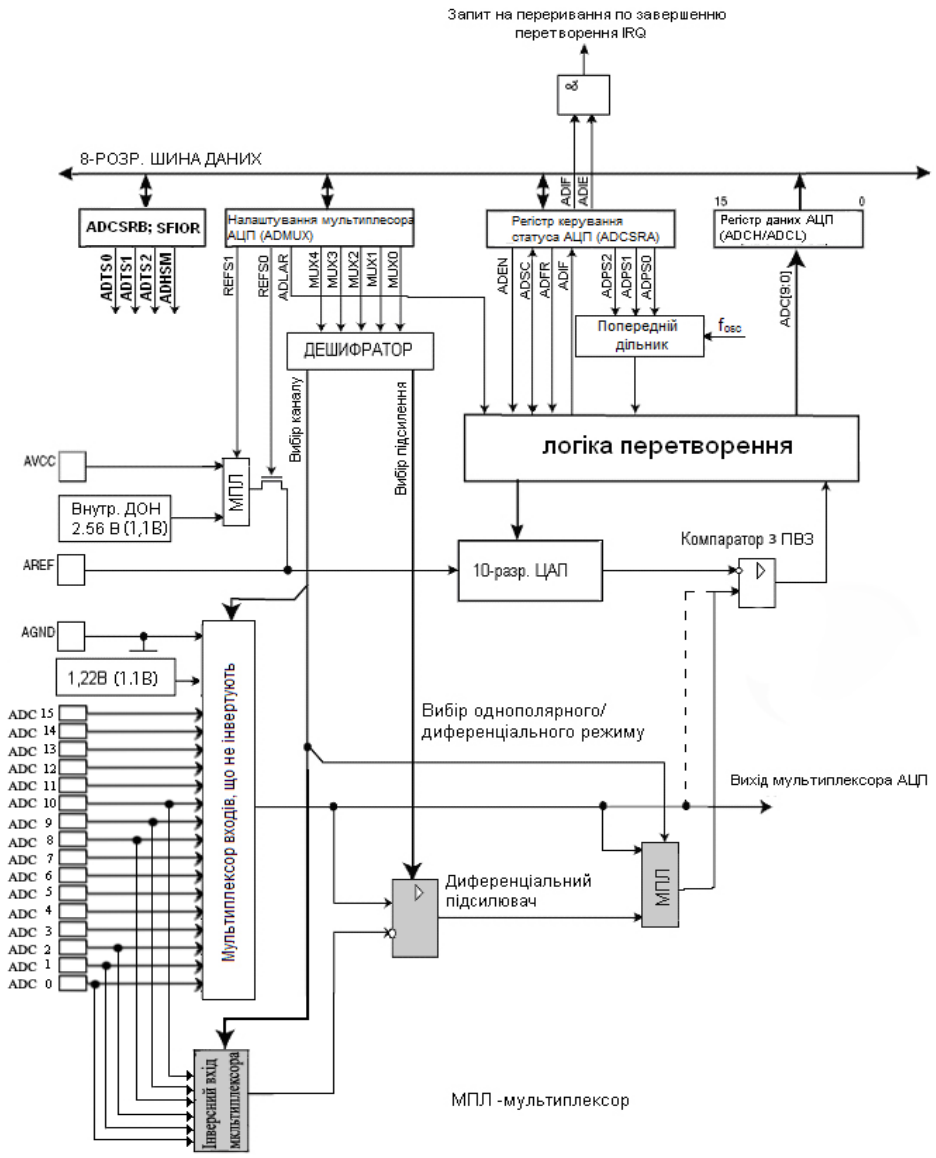


Рисунок 3.1 – Функціональна схема аналого–цифрового перетворювача

3.3 Програмне керування модулем АЦП

Як приклад для деяких моделей сімейства AVR у таблиці 3.1 наведено регістри, що використовуються для керування модулем АЦП. Формат регістрів ADCSRA (ADCSR) і ADMUX наведено на рисунках 3.2 і 3.3, а короткий опис функцій їх розрядів наведено у таблицях 3.2 і 3.3 відповідно.

Таблиця 3.1– Регістри керування модулем АЦП

Регістр	Адреса	ATmega8535x	ATmega8x	ATmega16x\32x	ATmega163x	ATmega323x	ATmega48x\88x\168x	ATmega64x	ATmega164x\324x\644x	ATmega165x	ATmega325x/3250x, ATmega645x/6450x	ATmega640x, ATmegal280x/1281x, ATmega2560x/2561x	ATmega128x	Опис
ADCSR	$06($26)		♦		♦	♦								Регістр керування і стану
ADCSRA	$06 ($26)	♦		♦				♦					♦	Регістр керування і стану А
	($7A)						♦		♦	♦	♦	♦		
ADCSRB	($8E)							♦						Регістр керування і стану В
	($7B)						♦		♦	♦	♦	♦		
ADMUX	$07 ($27)	♦	♦	♦	♦	♦		♦					♦	Регістр керування мультиплексором
	(7C)						♦		♦	♦	♦	♦		
SFIOR	$30 ($50)	♦	♦	♦										Регістр спеціальних функцій
	$20 ($40)												♦	

Таблиця 3.2 – Розряди регістра ADCSRA (ADCSR*)

Розряд	Назва	Опис
7	ADEN	Дозвіл АЦП (1 – увімкнено, 0 – вимкнено)
6	ADSC	Запуск перетворення (1 – почати перетворення)
5	ADATE (ADFR **)	Вибір режиму роботи АЦП (0 – одиночне, 1 – безперервне перетворення)
4	ADIF	Прапорець переривання завершення АЦП
3	ADIE	Дозвіл переривання від завершення АЦП
2..0	ADPS2:ADPS0	Вибір частоти перетворення (див. таблицю 6)
* В моделі ATmega8x ** В моделях ATmega8x, ATmega128x		

Таблиця 3.3– Розряди регістра ADMUX

Розряд	Назва	Опис	Модель
7..6	REFS1:REFS0	Вибір джерела опорної напруги (див. таблицю 5)	Всі моделі
5	ADLAR	Ліве вирівнювання результату (див. таблицю 11)	Всі моделі
4	–	Зарезервовано	ATmega8x
	MUX4	Вибір вхідних каналів (див. таблицю 9)	Всі моделі крім ATmega8x
3..0	MUX3..MUX0	Вибір вхідних каналів і частоти перетворення (див. таблиці 8, 9)	Всі моделі

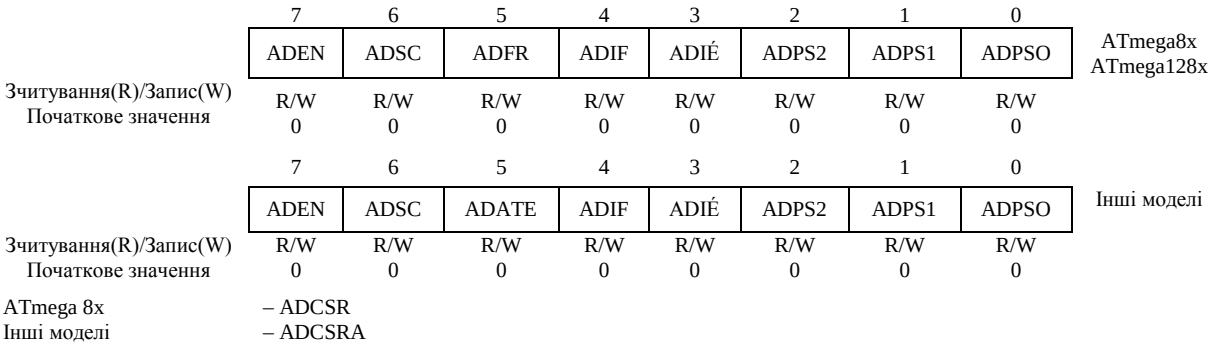


Рисунок 3.2 – Формат регістра ADCSRA (ADCSR)

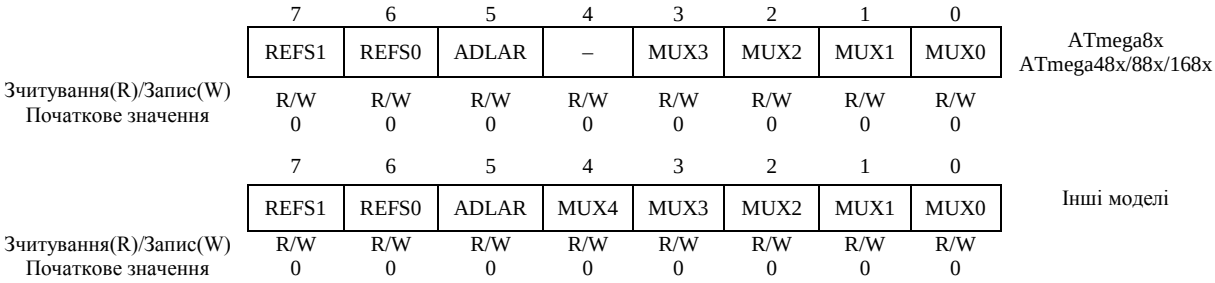


Рисунок 3.3 – Формат регістра ADMUX

Формат регістрів ADCSRB і SFIOR наведено рисунках 3.4 і 3.5 відповідно (не використовувані розряди регістра SFIOR позначені як «–»).

Для дозволу робити АЦП необхідно записати лог. 1 в розряд ADEN регістра ADCSRA (ADCSR), а для заборони– відповідно лог. 0. Якщо АЦП буде вимкнено під час циклу перетворення, то перетворення не буде завершено (в регістрі даних АЦП залишиться результат попереднього перетворення).

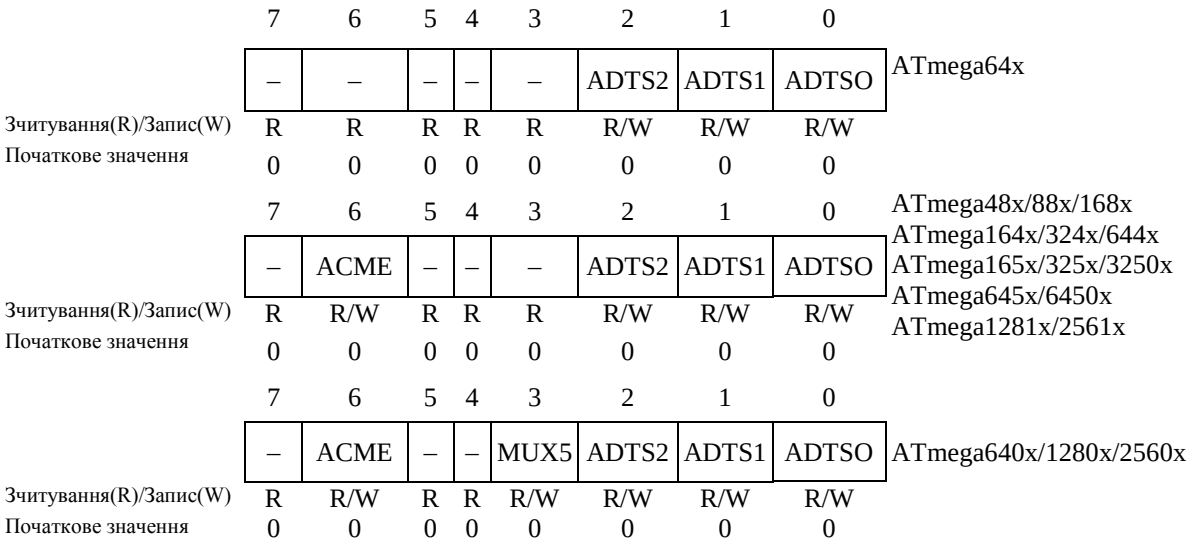


Рисунок 3.4 – Формат регістра ADCSRB

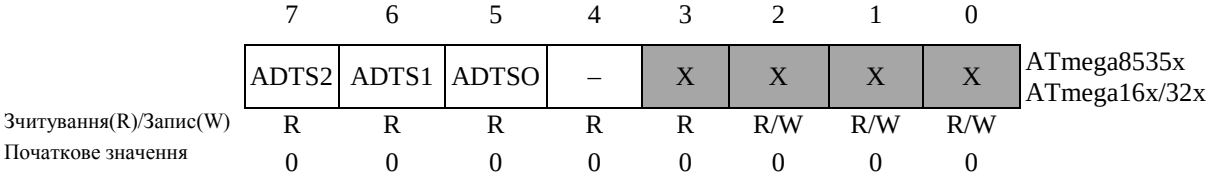


Рисунок 3.5 – Формат регістра SFIOR

У всіх моделях, окрім ATmega8x та ATmega128x, запуск АЦП можливий не тільки командою користувача, але й перериванням від деяких периферійних пристроїв, наявних у складі мікроконтролера. Для вибору режиму роботи в цих моделях використовується розряд ADATE регістра ADCSRA і розряди ADTS2...0 регістра SFIOR або ADCSRB.

Якщо розряд ADATE скинуто в "0", АЦП працює в режимі одиночного перетворення. Якщо ж розряд ADATE встановлено в "1", функціонування АЦП визначається вмістом розрядів ADTS2...0 відповідно до таблиці 3.4.

Запуск кожного перетворення в режимі одиночного перетворення, а також запуск першого перетворення в режимі безперервного перетворення здійснюється встановленням в "1" розряду ADSC регістра ADCSRA (ADCSR).

Запуск перетворення за перериванням здійснюється при встановлені в "1" прапорця обраного переривання. Розряд ADSC регістра ADCSRA при цьому апаратно встановлюється в "1". Запуск перетворення в цих режимах

також може бути здійснений встановленням в "1" розряду ADSC регістра ADCSRA.

Таблиця 3.4 – Джерело сигналу для запуску перетворення

ADTS2	ADTS1	ADTS0	Джерело стартового сигналу
0	0	0	Режим безперервного перетворення
0	0	1	Переривання від аналогового компаратора
0	1	0	Зовнішнє переривання INT0
0	1	1	Переривання за подією "Збіг А" таймера/лічильника Т0
1	0	0	Переривання за переповненням таймера/лічильника Т0
1	0	1	Переривання за подією "Збіг В" таймера/лічильника Т1
1	1	0	Переривання за переповненням таймера/лічильника Т1
1	1	1	Переривання за подією "Захоплення" таймера/лічильника Т1

Модуль АЦП може використовувати різні джерела опорної напруги (ДОН). Вибір конкретного джерела опорної напруги здійснюється за допомогою розрядів REFS1:REFS0 регістра ADMUX (таблиця 3.5).

Як зазначено в таблиці 3.5, внутрішнє ДОН може бути підключене до виводу AREF мікроконтролера. Тому при його використанні для підвищення завадозахищеності до виводу AREF можна підключити зовнішній фільтруючий конденсатор.

Таблиця 3.5– Вибір джерела опорної напруги

REFS1	REFS0	Джерело опорної напруги[1)]	Модель
0	0	Зовнішнє ДОН, підключено до виводу AREF; внутрішнє ДОН відключено	Всі моделі
0	1	Напруга живлення AVcc[2)]	Всі моделі
1	0	Внутрішнє ДОН напругою 1.1 В[2)]	ATmega164x/324x/644x ATmega640x/1280x/1281x ATmega2560x/2561x
		Зарезервовано	Решта моделей
1	1	Внутрішнє ДОН напругою 2.56 В[2]	ATmega48x/88x/168x, mega32 ATmega165/325x/3250x ATmega645x/6450x
		Внутрішнє ДОН напругою 1.1 В[2)]	Решта моделей

[1)] При роботі з підсиленням 10x чи 200x в якості внутрішнього ДОН можна використовувати тільки ДОН яке, підключено при REFS1:0=11.

[2)] Якщо до виводу AREF підключено зовнішнє джерело напруги, дані варіанти використовуватись не можуть.

АЦП перетворює вхідну аналогову напругу в 10–розрядний код методом послідовного наближення. Мінімальне значення відповідає рівню GND, а максимальне рівню AREF мінус 1 молодшого розряду.

Канал однополярного або диференціального аналогового введення та каскад диференціального підсилення обираються шляхом програмування розрядів MUXn (n=0,1…4) у регістрі ADMUX. У якості однополярного аналогового входу АЦП в залежності від моделі мікроконтролера може бути обраний один із входів ADC0...ADC7. У режимі диференціального введення передбачена можливість вибору входів, що інвертують і не інвертують диференціального підсилювача.

Якщо обрано диференціальний режим аналогового введення, то диференціальний підсилювач буде помножувати різницю напруг між обраною парою входів на заданий коефіцієнт підсилення. Підсилене в такий спосіб значення надходить на аналоговий вхід АЦП. Якщо обирається однополярний режим аналогового введення, то каскад підсилення пропускається.

Робота АЦП дозволяється шляхом встановлення розряду ADEN у регістрі ADCSRA. Вибір опорного джерела та каналу перетворення неможливо виконати до встановлення ADEN. Якщо ADEN = 0, то АЦП не споживає струм, тому при переході в економічні режими сну рекомендується попередньо відключити АЦП.

АЦП генерує 10–розрядний результат, що міститься в парі регістрів даних АЦП: ADCH і ADCL. У початковому стані результат перетворення розміщується в молодших 10–ти розрядах 16–розрядного слова (вирівнювання вправо), але може бути розміщений у старших 10–ти розрядах (вирівнювання вліво) шляхом встановлення розряду ADLAR у регістрі ADMUX (таблиці 3.3, 3.9).

Практична корисність подання результату з вирівнюванням вліво існує, коли досить точності 8–розрядного значення. В цьому випадку необхідно читати тільки регістр ADCH. В іншому ж випадку необхідно

першим читати вміст регістра ADCL, а потім ADCH, чим гарантується, що обидва байти є результатом того самого перетворення. Як тільки виконано читання ADCL блокується доступ до регістрів даних з боку АЦП. Це означає, що якщо зчитаний ADCL і перетворення завершується перед читанням регістра ADCH, то жоден з регістрів не може модифікуватися й результат перетворення губиться. Після читання ADCH доступ до регістрів ADCH і ADCL з боку АЦП знову дозволяється.

АЦП генерує власний запит на переривання за завершенням перетворення. Якщо між читанням регістрів ADCH і ADCL доступ до даних для АЦП заборонено, то переривання виникне, навіть якщо результат перетворення буде загублено.

Одиночне перетворення запускається шляхом запису лог. 1 у розряд запуску перетворення АЦП ADSC. Даний розряд залишається у високому стані в процесі перетворення й скидається за завершенням перетворення. Якщо в процесі перетворення перемикається канал аналогового введення, то АЦП автоматично завершить поточне перетворення, перш ніж перемкне канал.

У режимі автоматичного перезапуску АЦП безупинно оцифровує аналоговий сигнал і оновлює регістр даних АЦП. Даний режим задається шляхом запису лог. 1 у розряд ADFR (ADATE) регістра ADCSR (ADCSRA). Перше перетворення ініціюється шляхом запису лог. 1 у розряд ADSC регістра ADCSR (ADCSRA). У даному режимі АЦП виконує послідовні перетворення, незалежно від того скидається прапорець переривання АЦП ADIF чи ні.

3.4 Формування тактової частоти АЦП

На рисунку 3.6 наведено схему попереднього дільника, що формує тактовий сигнал для АЦП. Попередній дільник формує похідні частоти відносно частоти синхронізації мікроконтролера. Коефіцієнт ділення

встановлюється за допомогою розрядів ADPSn у регістрі ADCSRA (таблиця 3.6).

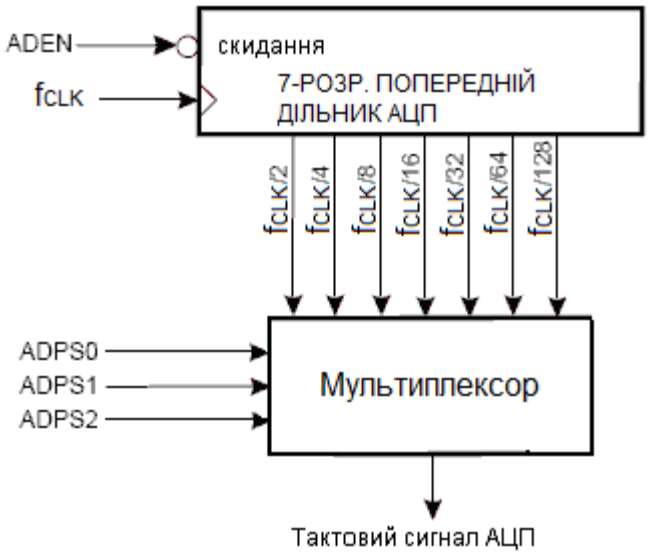


Рисунок 3.6 – Схема попереднього дільника АЦП

Попередній дільник починає лічбу з моменту включення АЦП встановленням розряду ADEN у регістрі ADCSRA. Попередній дільник працює доки розряд ADEN = 1 і скинутий, коли ADEN = 0.

Якщо потрібно забезпечити максимальну роздільну здатність (10 розрядів), то частота на вході схеми послідовного наближення повинна бути в діапазоні 50...200 кГц [1]. Якщо достатньо точності менше 10 розрядів, але потрібна більш висока частота перетворення, то частота на вході АЦП може бути встановлена понад 200 кГц.

Таблиця 3.6 – Задання коефіцієнта ділення попереднього дільника АЦП

ADPS2	ADPS1	ADPS0	Коефіцієнт ділення
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

3.5 Часові діаграми роботи АЦП

На рисунках 3.7…3.10 наведено часові діаграми роботи АЦП у різних режимах.

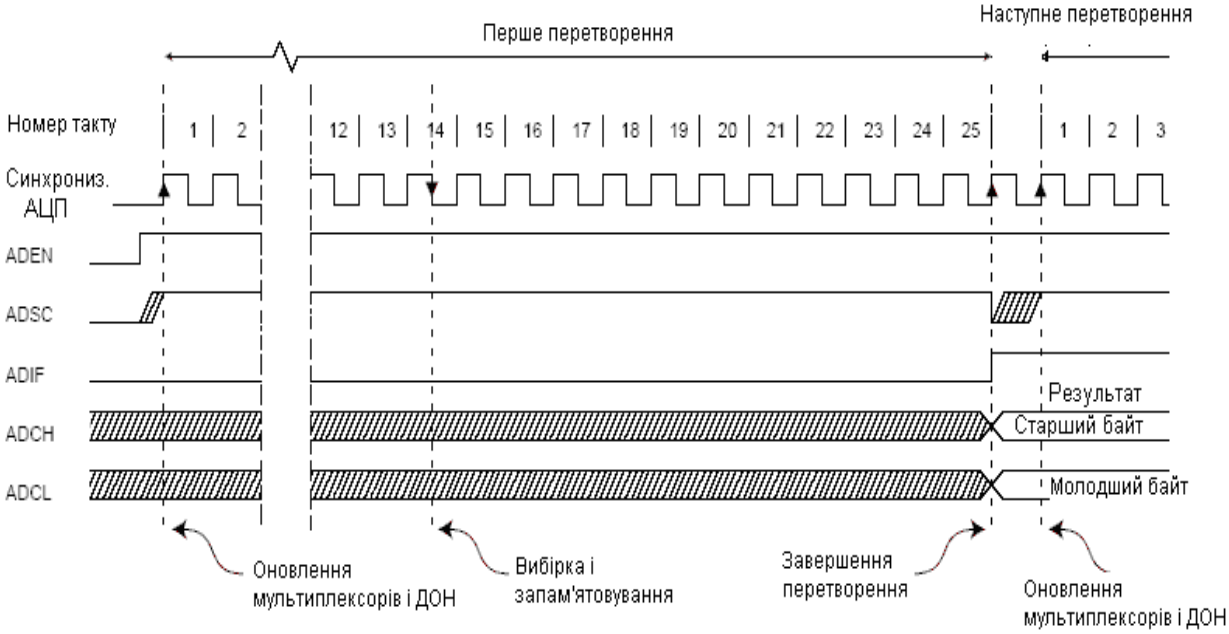


Рисунок 3.7 – Часові діаграми роботи АЦП при першому перетворенні в режимі одиночного перетворення

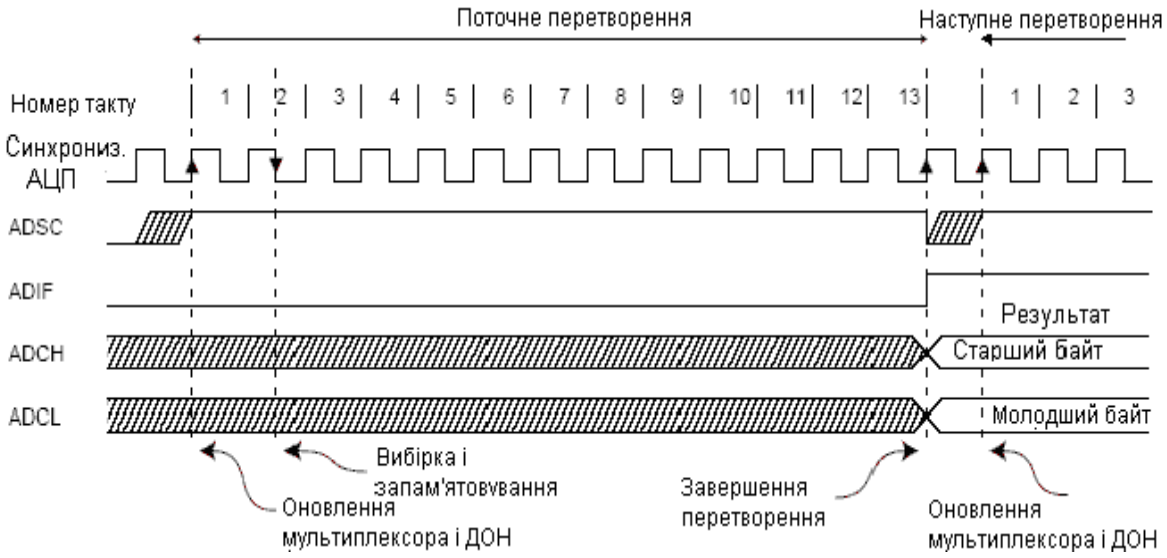


Рисунок 3.8 – Часові діаграми роботи АЦП у режимі одиночного перетворення

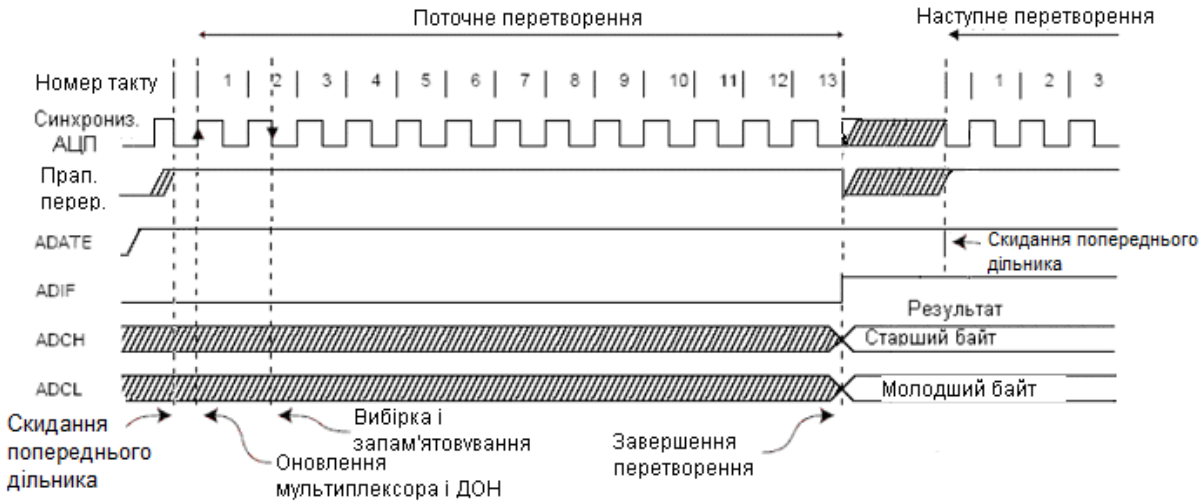


Рисунок 3.9 – Часові діаграми роботи АЦП у режимі запуску за перериванням

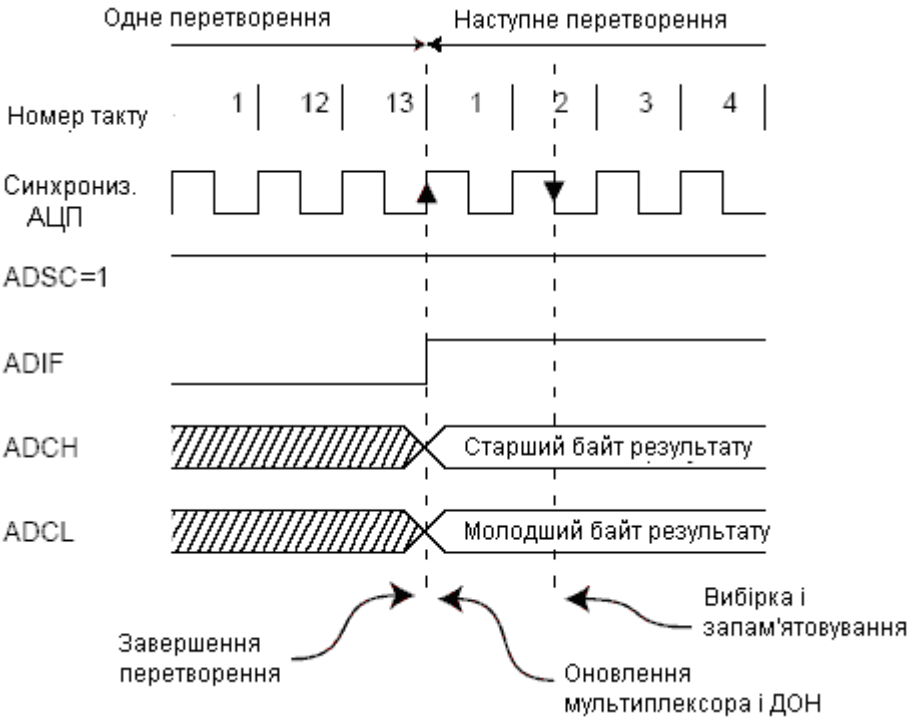


Рисунок 3.10 – Часові діаграми роботи АЦП у режимі автоматичного перезапуску для однополярного перетворення

Якщо ініціюється одиночне однополярне перетворення встановленням розряду ADSC у регістрі ADCSRA, то перетворення починається з наступного наростаючого фронту тактового (синхро) сигналу АЦП.

У режимі одиночного перетворення нове перетворення може бути запущено відразу ж після скидання розряду ADSC (до збереження результату поточного перетворення). Однак реально цикл перетворення

почнеться не раніше ніж через один такт після закінчення поточного перетворення.

Нормальне перетворення вимагає 13 тактів синхронізації АЦП. Перше перетворення після включення АЦП (встановлення ADEN в регістрі ADCSRA) вимагає 25 тактів синхронізації АЦП за рахунок необхідності ініціалізації модуля.

Після початку нормального перетворення на вибірку–зберігання затрачується 1.5 такти синхронізації АЦП, а після початку першого перетворення – 13,5 тактів. По завершенні перетворення результат зберігається в регістрах даних АЦП і встановлюється прапорець ADIF. У режимі одиночного перетворення одночасно скидається розряд ADSC. Програмно розряд ADSC може бути знову встановлено і нове перетворення буде ініційовано першим наростаючим фронтом тактового сигналу АЦП.

У режимі безперервного перетворення (автоматичного перезапуску) нове перетворення починається відразу по завершенні попереднього, при цьому ADSC залишається у високому стані. Час перетворення для різних режимів перетворення представлено в таблиці 3.7.

Таблиця 3.7 –Час перетворення АЦП

Тип перетворення	Тривалість вибірки–зберігання (у тактах з моменту початку перетворення)	Час перетворення (у тактах)
Перше перетворення	14.5	25
Нормальне однополярне перетворення	1.5	13
Нормальне диференціальне перетворення	1.5/2.5	13/14

3.6 Керування вхідним мультиплексором

Виводи мікроконтролера, які підключені до входу АЦП, визначаються станом розрядів MUX4...MUX0 регістра ADMUX (рисунок 3.3), та регістра ADCSRB (рисунок 3.4). Для каналів з диференціальним входом зазначені

розряди визначають також коефіцієнт попереднього підсилення вхідного сигналу. Для мікроконтролера ATmega32 керування вхідним мультиплексором відображає таблиця 3.8.

Таблиця 3.8 – Керування вхідним мультиплексором у моделях ATmega16x/164x/32x/8535x/64x/128x/164x/165x/325x/3250x/645x/6450x/ 1281x/2561x

MUX4…MUX0	Однополярний вхід	Диференціальний вхід		Попереднє підсилення
		(додатний)	(від'ємний)	
00000	ADC0	Не застосовується		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000[1]	Не застосовується	ADC0	ADC0	10x
01001[1]		ADC1	ADC0	10x
01010[1]		ADC0	ADC0	200x
01011[1]		ADC1	ADC0	200x
01100[1]		ADC2	ADC2	10x
01101[1]		ADC3	ADC2	10x
01110[1]		ADC2	ADC2	200x
01111[1]		ADC3	ADC2	200x
10000[1]		ADC0	ADC1	1x
10001[1]		ADC1	ADC1	1x
10010[1]		ADC2	ADC1	1x
10011[1]		ADC3	ADC1	1x
10100[1]		ADC4	ADC1	1x
10101[1]		ADC5	ADC1	1x
10110[1]		ADC6	ADC1	1x
10111[1]		ADC7	ADC1	1x
11000[1]		ADC0	ADC2	1x
11001[1]		ADC1	ADC2	1x
11010[1]		ADC2	ADC2	1x
11011[1]		ADC3	ADC2	1x
11100[1]	Не застосовується	ADC4	ADC2	1x
11101[1]		ADC5	ADC2	1x
11110	1.22 В (1,1 В[1])	Не застосовується		
11111	0В (GND)			
[1] У моделях ATmega165x/325x/3250x/645x/6450x/1251x/2561x				

Програмування розрядів MUXn (n=0,1,…,4) і REFS1:0 у регістрах ADMUX та ADCSRB підтримується буферизацією через тимчасовий регістр. Цим гарантується, що нові налаштування каналу перетворення й опорного джерела набудуть чинності в безпечний момент для перетворення.

До початку перетворення будь–які зміни каналу та опорного джерела набувають чинності відразу після їхньої модифікації. Як тільки починається процес перетворення доступ до зміни каналу та опорного джерела блокується, чим гарантується достатність часу на перетворення для АЦП. Безперервність модифікації повертається на останньому такті АЦП перед завершенням перетворення (перед встановленням прапорця ADIF у регістрі ADCSRA). Зверніть увагу, що перетворення починається наступним наростаючим фронтом тактового сигналу АЦП після встановлення ADSC. Таким чином, користувачеві не рекомендовано записувати нове значення каналу або опорного джерела в ADMUX до 1–го такту синхронізації АЦП після встановлення ADSC.

3.7 Збереження результату перетворення

Після завершення перетворення (при встановленні в "1" прапорця ADIF регістра ADCSR) його результат зберігається в регістрі даних АЦП. Оскільки АЦП має 10 розрядів, цей регістр фізично розміщено у двох регістрах введення/виведення ADCH:ADCL, доступних тільки для читання. Ці регістри розташовані за адресами $05:$04 і при включенні мікроконтролера містять значення "$0000".

У початковому стані результат перетворення вирівнюється вправо (старші 6 розрядів регістра ADCH – не є значущими).

Однак він може вирівнюватися також вліво (молодші 6 розрядів регістра ADCL – не є значущими). Для керування вирівнюванням результату перетворення призначено розряд ADLAR регістра ADMUX. Якщо цей розряд встановлено в "1", результат перетворення вирівнюється

за лівою границею 16–розрядного слова, якщо скинутий в "0" – за правою границею.

При використанні диференціального режиму перетворення результат представляється в коді двійкового доповнення до двох (в додатковому коді).

У таблиці 3.9 наведено приклади вирівнювання результату вліво та вправо.

Таблиця 3.9 – Вирівнювання результату АЦП

ADLAR	Розряд	15	14	13	12	11	10	9	8	ADCH
0		–	–	–	–	–	–	ADC9	ADC8	
		ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	Розряд	7	6	5	4	3	2	1	0	
	R/$\overline{W}$	R	R	R	R	R	R	R	R	ADCH
		R	R	R	R	R	R	R	R	ADCL
	Поч. зн.	0	0	0	0	0	0	0	0	ADCH
		0	0	0	0	0	0	0	0	ADCL
1	Розряд	15	14	13	12	11	10	9	8	
		ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
		ADC1	ADC0	–	–	–	–	–	–	ADCL
	Розряд	7	6	5	4	3	2	1	0	
	R/$\overline{W}$	R	R	R	R	R	R	R	R	ADCH
		R	R	R	R	R	R	R	R	ADCL
	Поч. зн.	0	0	0	0	0	0	0	0	ADCH
		0	0	0	0	0	0	0	0	ADCL

Звернення до регістрів ADCH і ADCL для отримання результату перетворення повинно виконуватися в певній послідовності: спочатку необхідно прочитати регістр ADCL, а потім ADCH. Ця вимога пов’язана з тим, що після звернення до регістра ADCL процесор блокує доступ до регістрів даних з боку АЦП доти, поки не буде прочитано регістр ADCH. Завдяки цьому можна бути впевненим, що при читанні регістрів ADCH, ADCL у них будуть перебувати складові того самого результату. Відповідно, якщо чергове перетворення завершиться до звернення до регістра ADCH, результат перетворення буде загублено. З іншого боку, якщо результат перетворення вирівнюється вліво й досить точності 8–розрядного значення, для отримання результату можна прочитати тільки вміст регістра ADCH.

3.8 Особливості підключення джерела опорної напруги

Джерело опорної напруги (ДОН) для АЦП ($U_{ДОН}$) визначає діапазон перетворення АЦП. Якщо рівень однополярного сигналу понад $U_{ДОН}$, то результатом перетворення буде 0x3FF. У якості $U_{ДОН}$ можуть виступати AVCC, внутрішнє ДОН 2,56 В (1,1 В) або зовнішнє ДОН, що підключено до виводу AREF. AVCC підключається до АЦП через пасивний ключ. Внутрішня опорна напруга 2,56 В (1,1 В) генерується внутрішнім еталонним джерелом VBG, що буферизовано внутрішнім підсилювачем. У кожному разі зовнішній вивід AREF зв'язаний безпосередньо з АЦП і, тому, можна знизити вплив шумів на опорне джерело за рахунок підключення конденсатора між виводом AREF і спільним виводом. Напруга $U_{ДОН}$ також може бути виміряна на виводі AREF вольтметром з високим вхідним опором. Зверніть увагу, що $U_{ДОН}$ є високоомним джерелом і, тому, зовні до нього може бути підключене тільки ємнісне навантаження.

Якщо користувач використовує зовнішнє опорне джерело, що підключено до виводу AREF, то не допускається використання іншої опції опорного джерела, тому що це приведе до шунтування зовнішньої опорної напруги. Якщо до виводу AREF не прикладена напруга, то користувач може обрати AVCC і 2,56 В (1,1 В) як опорне джерело. Результат першого перетворення після перемикання опорного джерела може характеризуватися низькою точністю, тому користувачеві рекомендується його ігнорувати.

3.9 Результат перетворення АЦП

Для каналів з однополярним (несиметричним) входом результат перетворення визначається виразом: $ADC = \frac{1023 U_{IN}}{U_{REF}}$, де U_{IN} – значення вхідної напруги, U_{REF} – величина опорної напруги, ADC – десятковий еквівалент двійкового коду на виході АЦП. На рисунку 3.13 представлено функцію перетворення АЦП в однополярному режимі. Код 0x000 відповідає

рівню аналогової землі, а 0x3FF – рівню напруги ДОН мінус 1 крок квантування за напругою.

Таблиця 3.10 відображає зв’язок між вхідним сигналом й вихідними кодами для однополярного режиму.

Приклад: Нехай ADMUX = 0x00…0x07 (будь–який однополярний вхід), напруга на одному з входів 1000 мВ, напруга ДОН рівна 2,56 В, тоді:

КодАЦП = 1024 * 1000 / 2560 = 400 = 0x190.

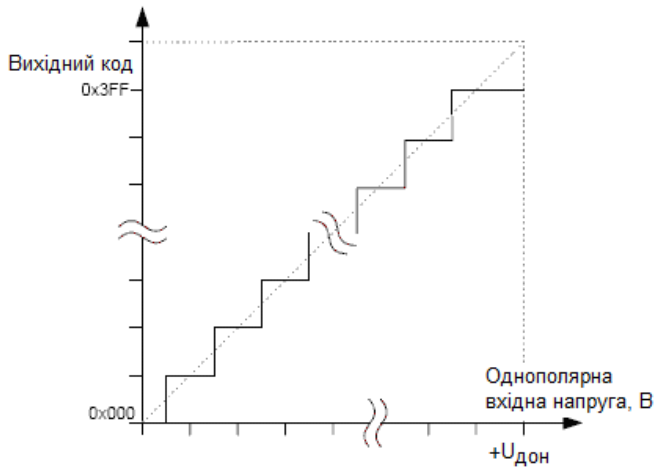


Рисунок 3.13 – Функція перетворення АЦП при зміні однополярного сигналу

Таблиця 3.10 – Зв’язок між вхідним и вихідним кодами

$U_{АЦПn}$*	Зчитаний код	Відповідне десяткове значення
$U_{АЦПm} + U_{ДОН}$	0x3FF	1023
$U_{АЦПm} + 0.999\ U_{ДОН}$	0x3FF	1023
$U_{АЦПm} + 0.998\ U_{ДОН}$	0x3FE	1022
…	...	...
$U_{АЦПm} + 0.001\ U_{ДОН}$	0x001	1
$U_{АЦПm}$	0x000	0

$U_{АЦПm}$ – вхідна напруга, яка дорівнює нулю, $U_{АЦПn}$ – поточне значення вхідної напруги.

Для каналів з диференціальним входом результат перетворення описується в [2].

3.10 Розробка схеми алгоритму роботи

Схему алгоритму роботи модуля АЦП наведено на рисунках 3.14…3.17.

10–розрядний АЦП працює у режимі безперервного перетворення з використанням каналів з однополярним (несиметричним) входом.

Скориставшись прикладом наведеним у 3.9, при вхідній напрузі $U_{вх} = 5В$ та величині опорної напруги U_{REF}=5В в ADC буде зберігатися значення: $2^{10} - 1 = 1023$.

З числа ADC виділяємо необхідні частини значення, шляхом ділення на 10000, 1000, 100, 10 та отримуємо тисячі, сотні, десятки та одиниці відповідно.

Для налаштування параметрів АЦП виконуються такі дії: в регістрі ADCSRA встановлюємо в лог.1 біти ADEN – дозвіл АЦП, ADSC – запуск перетворення, ADATE – безперервний режим роботи АЦП, ADPS2 і ADPS1 – переддільник на 64, ADIE – дозвіл переривань від АЦП; в регістрі ADMUX скидаємо в лог. нуль біти REFS1 і REFS0 щоб обрати зовнішнє джерело опорної напруги.

Таймер/лічильник у даній програмі використовується виключно для відображення отриманого десяткового коду перетворення АЦП на семисегментних індикаторах. При переповненні таймера 2 програма відображає одну із цифр на чотирьохпозиційному семисегментному індикаторі. Для цього на порт С подається число, в якому в лог. одиницю встановлені біти активного індикатора. Потім, за допомогою математичної операції отримання остачі, визначається відповідна цифра, яка є індексом у масиві SEG та відповідає своєму семисегментному аналогу.

Схему загального алгоритму роботи модуля наведено на рисунку 3.17.

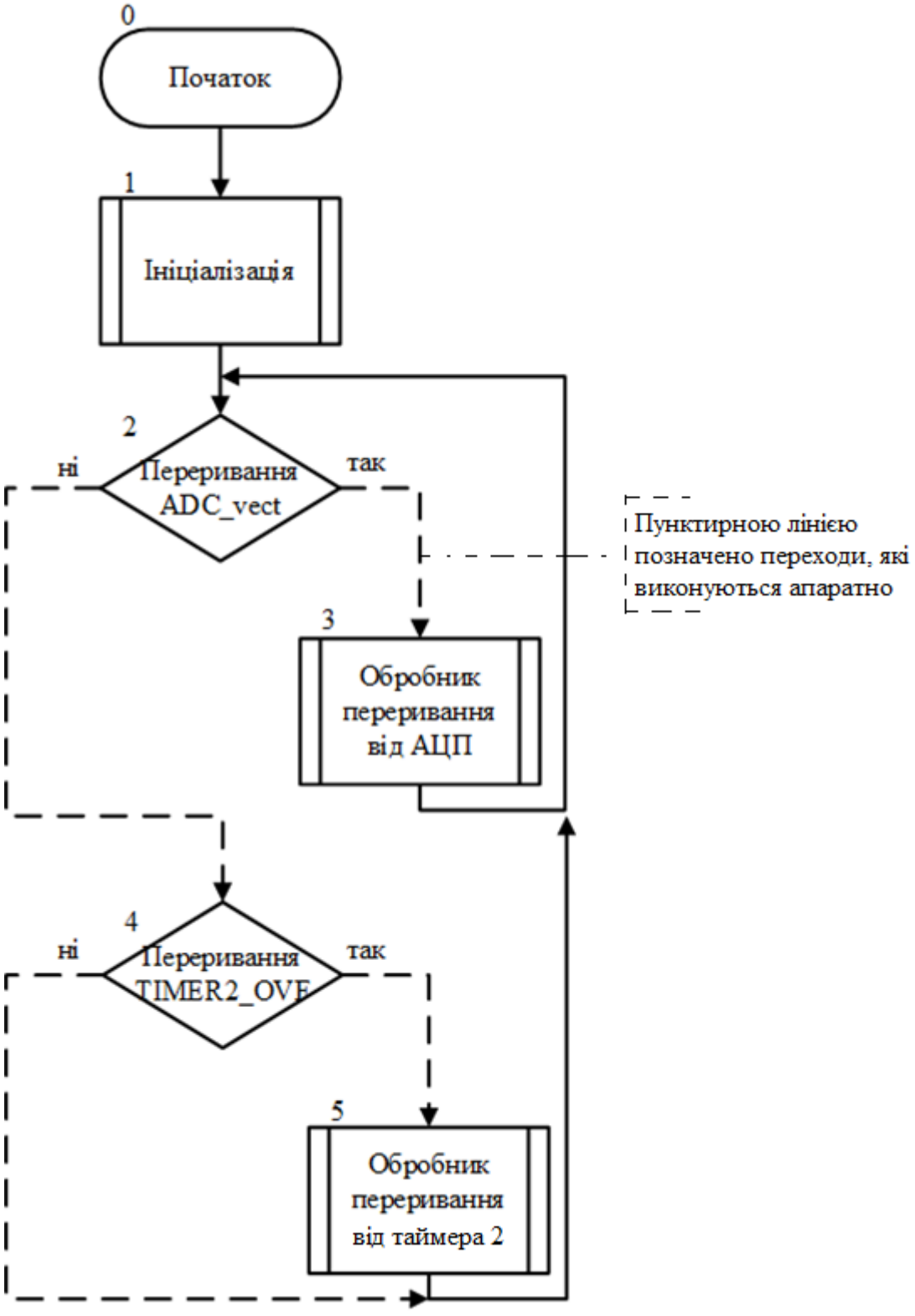


Рисунок 3.14 – Схема загального алгоритму роботи модуля

Схему алгоритму ініціалізації наведено на рисунку 3.18.

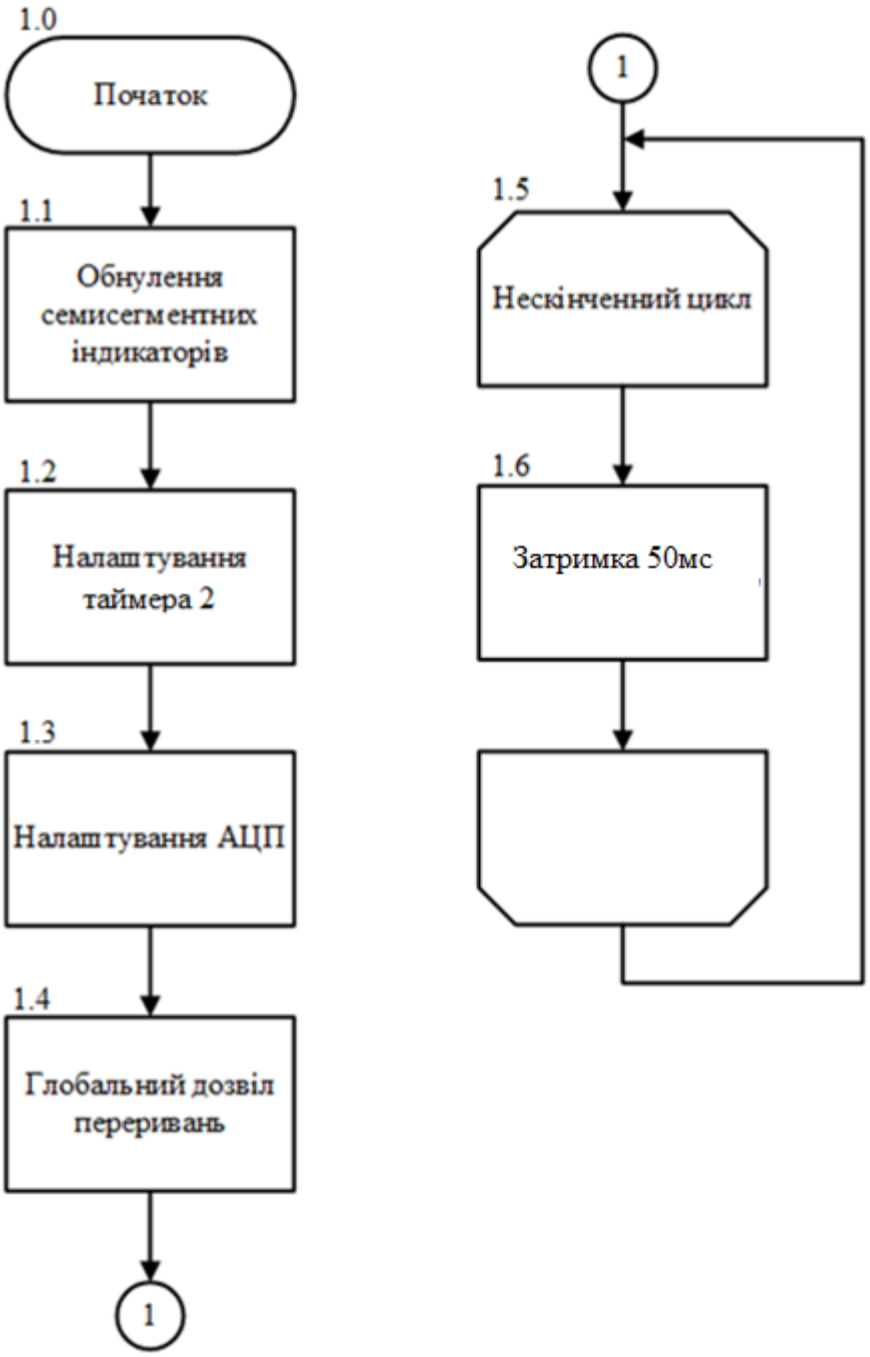


Рисунок 3.15 – Схема алгоритму ініціалізації

Схему алгоритму обробки переривання від таймера 2 наведено на рисунку 3.16.

Рисунок 3.16 – Схема алгоритму обробки переривання від таймера 2

Схему алгоритму обробки переривання завершення перетворення АЦП наведено на рисунку 3.17.

Рисунок 3.17 – Схема алгоритму обробки переривання завершення перетворення АЦП

3.11 Робоча програма мовою програмування С

```
// Підключення заголовних файлів бібліотек
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

// Масив значень для відображення цифр на семисегментних
індикаторах
//-------------0-----1-----2-----3-----4-----5-----6-----7-----
8------9----dp
char SEG[] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07,
0x7F, 0x6F, 0x80};

// Ініціалізації глобальних змінних
volatile unsigned char current_indicator = 0;
volatile unsigned int display = 0;

// Блок 1 - Ініціалізація програми
int main (void)
{
    // Блок 1.1 - Обнулення портів семисегментних індикаторів
    DDRC = 0xFF;
    PORTC = 0x00;
```

```
    DDRD = 0xFF;
    PORTD = 0x00;

    // Блок 1.2 - Налаштування Таймера 2
    TIMSK |= (1 << TOIE2); // Дозвіл переривання по таймеру 2
    TCCR2 |= (1 << CS21);  // Переддільник на 8

    // Блок 1.3 - Налаштування АЦП
    ADCSRA |= (1 << ADEN) // Дозвіл АЦП
    |(1 << ADSC) // Запуск перетворення
    |(1 << ADATE) // Безперервний режим роботи АЦП
    |(1 << ADPS2)|(1 << ADPS1) // Переддільник на 64
    |(1 << ADIE); // Дозвіл переривань від АЦП

    ADMUX &= (~(1 << REFS1))&(~(1 << REFS0)); // Зовнішнє ДОН

    // Блок 1.4 - Глобальний дозвіл переривань
    sei();

    // Блок 1.5 - Основний цикл
    while(1)
    {
        _delay_ms(50); // Блок 1.6   Затримка 50 мс
    }
}

// Блок 2 - Умова переривання від АЦП
ISR (ADC_vect)
{// Блок 3 - Обробник перериваннь від АЦП
    Display = ADC; // Блок 3.1 - Присвоюємо глобальній змінній
поточне значення АЦП
}

// Блок 4 - Умова переривання від таймера Т2
ISR (TIMER2_OVF_vect)
{// Блок 5 - Обробник переривань від таймера Т2
    PORTD = 0xFF; // Блок 5.1 - Вимикаємо всі сегменти
    PORTC = (1 << current_indicator); // Блок 5.2 - Обираємо
поточний індикатор

    //Блок 5.3
    switch (current_indicator)
    {
        case 0:
        PORTD = ~(SEG[display % 10000 / 1000]); // Вмикаємо
цифру одиниць
        break;
        case 1:
        PORTD = ~((SEG[display % 1000 / 100])); // Вмикаємо
цифру десятків
        break;
        case 2:
```

```
        PORTD = ~(SEG[display % 100 / 10]); // Вмикаємо цифру
сотень
        break;
        case 3:
        PORTD = ~(SEG[display % 10 / 1]); // Вмикаємо цифру
тисяч
        break;
    }
    if ((current_indicator++) > 2) // Блок 5.4 – Переходимо на
наступний індикатор
        current_indicator = 0; // Блок 5.5 – Обнуляємо, якщо він
за межами
}
```

Нижче наведено пояснення фрагменту програми виведення на чотири семисігментних індикатора результату перетворення АЦП: ADC який надходить у десятковому коді згідно з формулою, яку наведено у підрозділі 3.9.

Наприклад, при $U_{вхАЦП}$=5В значення ADC згідно з роботою моделі в Proteus 8.6 дорівнює 1023. Це значення програма обробляє наступним чином:

1023%10000/1000=1023/1000=1,023. Цифра 1 виводиться на перший індикатор.

Далі відбуваються наступні дії:

1023%1000/100=23/100=0,23. Цифра 0 виводиться на другий індикатор;

1023%100/10=23/10=2,3. Цифра 2 виводиться на третій індикатор;

1023%10/1=3/1=3. Цифра 3 виводиться на четвертий індикатор.

Наприклад, ADC=686, тоді:

686%10000/1000=686/1000=0,686. Цифра 0 виводиться на перший індикатор.

686%1000/100=686/100=6,86. Цифра 6 виводиться на другий індикатор;

686%100/10=86/10=8,6. Цифра 8 виводиться на третій індикатор;

686%10/1=6/1=6. Цифра 6 виводиться на четвертий індикатор.

3.12 Опис моделі

Схему моделювання модуля АЦП в PROTEUS зображено на рисунку 3.18.

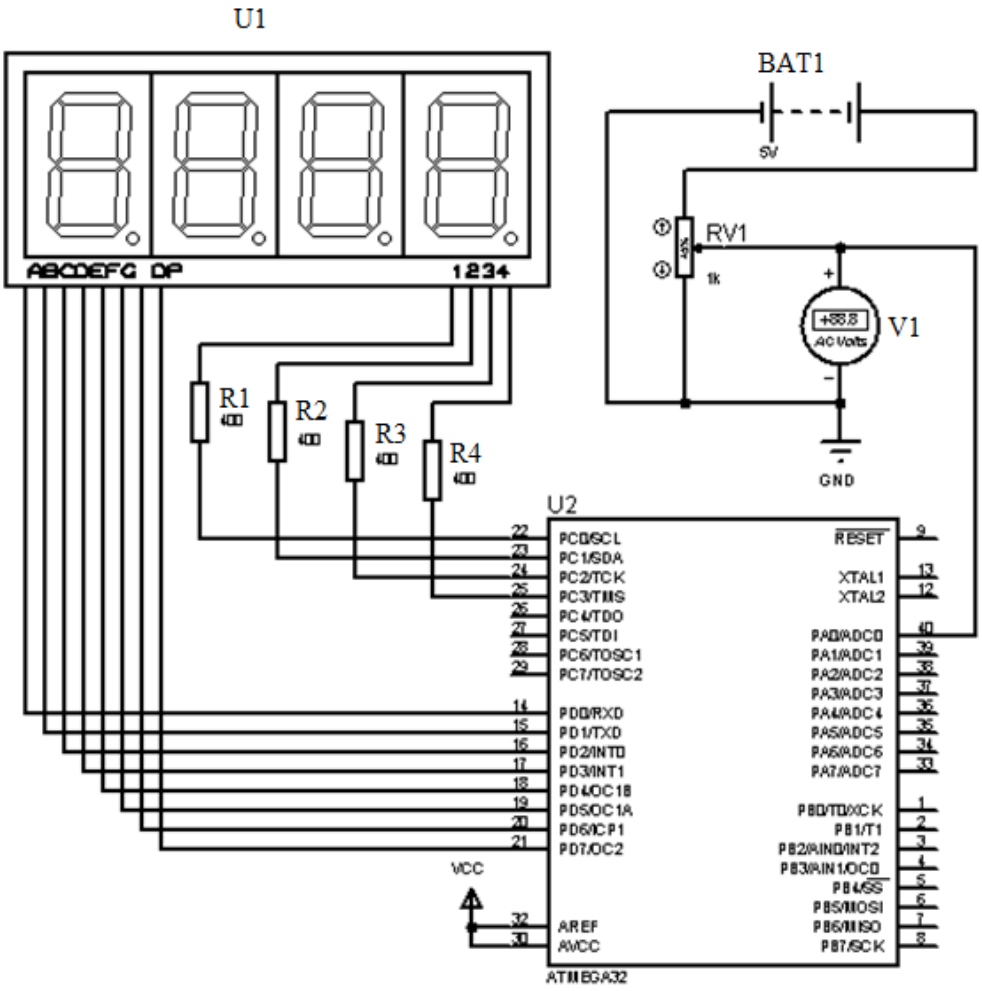


Рисунок 3.18 – Схема моделювання модуля АЦП в PROTEUS

Дана схема складається з наступних елементів:

- U2 – AVR мікропроцесор ATMEGA32;
- R1…R4 – резистори опором 400Ом;
- RV1 – резистор змінного опору на 10кОм;
- BAT1 – джерело напруги 5В;
- VCC – джерело живлення мікроконтролера;
- U1 – семисегментний чотирипозиційний індикатор;

– V1 – вольтметр для вимірювання вхідної напруги.

На початковому етапі, вхідна напруга для АЦП подається на лінію PA0 порту A з виходу дільника напруги +5В (RV1). Після перетворення її значення у цифровий еквівалент, розрахована величина виводиться у порти мікроконтролера для відображення на індикатори. Індикатори підключені до порту C (PC0…PC3) та порту D (PD0…PD7).

За допомогою резистора RV1, можна змінювати величину вхідної напруги у режимі реального часу. Значення вхідної напруги може змінюватися від 0В до 5В.

Для каналів з однополярним (несиметричним) входом результат перетворення визначається виразом: $ADC = \frac{1023 U_{IN}}{U_{REF}}$, де U_{IN} – значення вхідної напруги, U_{REF} – величина опорної напруги, ADC – десятковий еквівалент двійкового коду на виході АЦП. На рисунку 3.13 представлено функцію перетворення АЦП в однополярному режимі. Код 0x000 відповідає рівню аналогової землі, а 0x3FF – рівню напруги 5В.

Таблиця 3.10 відображає зв’язок між вхідним сигналом й вихідними кодами для однополярного режиму.

Коефіцієнт передачі АЦП визначається формулою

$$К_{пер} = \frac{1023}{U_{REF}} \left[\frac{МЗР}{mB}\right].$$

При U_{REF} =5В $К_{пер} = \frac{1023}{5000} = 0{,}2046.$

Тоді, наприклад, якщо $U_{вх}$ =5В=5000мВ, то десятковий код на виході буде дорівнювати

$$ADC = 5000 \cdot 0{,}2046 = 1023 .$$

В цьому можна переконатися під час роботи моделі. Для цього треба за допомогою дільника RV1 встановити на вольтметрі V1 напругу 5В. На індикаторі U1 побачимо число 1023, що відповідає наведеному вище розрахунку.

4 МОДЕЛЮВАННЯ МОДУЛЯ УНІВЕРСАЛЬНОГО АСИНХРОННОГО ПРИЙМАЧА–ПЕРЕДАВАЧА СІМ`Ї МК51

4.1 Опис моделі

Робочу модель дослідження універсального асинхронного приймача–передавача (УАПП) наведено на рисунку 4.1.

Розберемо цю модель докладніше. Зліва ми бачимо вісім кнопок (K1…K8), за допомогою яких можна задавати байт (8 біт) даних, який ми будемо передавати через модуль УАПП.

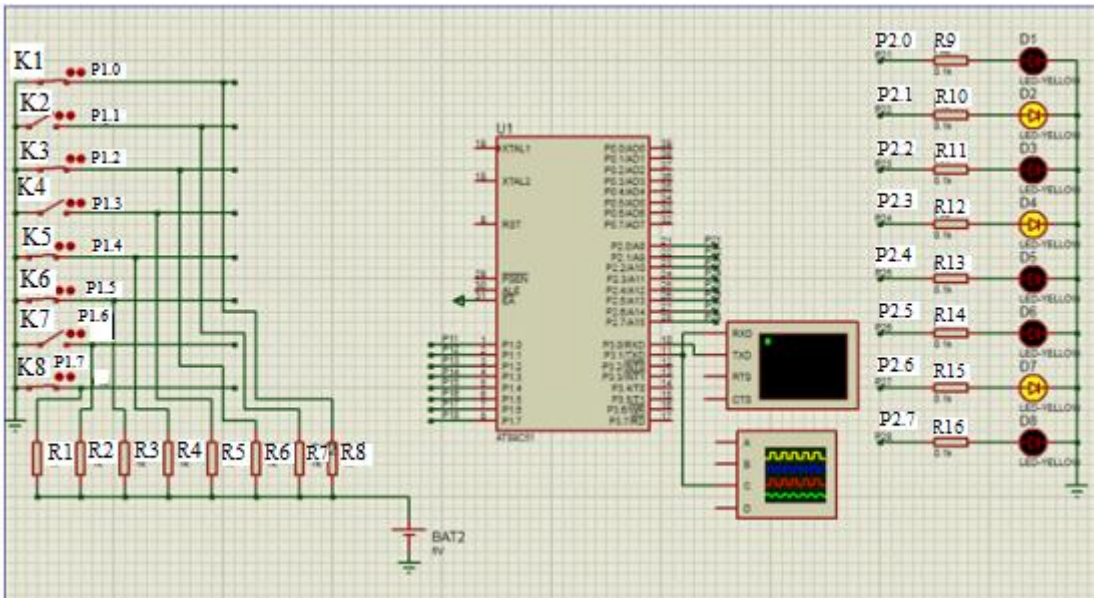


Рисунок 4.1 – Схема робочої моделі

Кнопки підключено до восьми ліній порту P1 мікроконтролера: P1.0, P1.1, … , P1.7. Якщо якась із кнопок відпущена, то через резистори R1...R8 на відповідну лінію порту P1 подається логічна 1. Якщо кнопка нажата, то вводиться логічний нуль. Праворуч зображено вісім світлодіодів, які підключено до ліній порту P2: P2.0, P2.1, … , P2.7. Катоди світлодіодів (D1…D8) підключено до спільного провода (землі), а на аноди через резистори R9…R16, які обмежують струм, із виходів порту P2 подаються логічні одиниці, коли треба засвітити відповідний світлодіод.

До ліній TxD–вихід передавача УАПП та RxD–вхід приймача підключено осцилограф, та Virtual Terminal, який вибрано з віртуальних

інструментів програми Proteus. На них ми будемо відображати повідомлення, які вводиться та виводяться через модуль УАПП в/з мікроконтролера. Додавання в модель осцилографа описано вище в 1.5.

До виводів XTAL1 та XTAL2 підключають кварцовий резонатор (на рисунку 4.1 не показано) частотою, яка визначає тактову частоту генератора мікроконтролера. В нашому прикладі вона дорівнює f_{BQ}=6МГц. Також підключають конденсатори С1 та С2, ємністю 30пФ (на рисунку 4.1 також не показано), які призначені для підвищення стабільності роботи системного генератора [2]. Резонатор та конденсатори потрібні в практичній схемі. В модель Proteus їх можна не вводити. Для задання частоти треба двічі лівою кнопкою миші клацнути на мікроконтролері та в опції Clock Frequency вказати значення частоти.

В якості мікроконтролера використано мікросхему AT89C51. Мікроконтролер побудований за процесорною архітектурою МК51 (MCS–51), тобто він вміє виконувати асемблерні команди, які описано цим стандартом. Стандарт був розроблений фірмою INTEL і в подальшому став основою для створення сучасних INTEL–подібних мікроконтролерів, але проблема створення маленьких пристроїв (мікроконтролерних систем) залишилася актуальною і до цього дня. Перші мініатюрні мікроконтролери, наприклад i8031, експлуатуються до сих пір (наприклад в телефонах АОН). Мікроконтролер випускають у трьох корпусах: PDIP, PQFP/TFQP, PLCC (рисунок 4.2).

4.2 Скорочений опис архітектури модуля УАПП

4.2.1 Програмування модуля УАПП

Для програмування модуля УАПП (послідовного порту) призначені:

– прегістр керування приймачем–передавачем (РКПП): SCON;
– старший біт регістра керування потужністю (РКП): PCON.7 (SMOD).

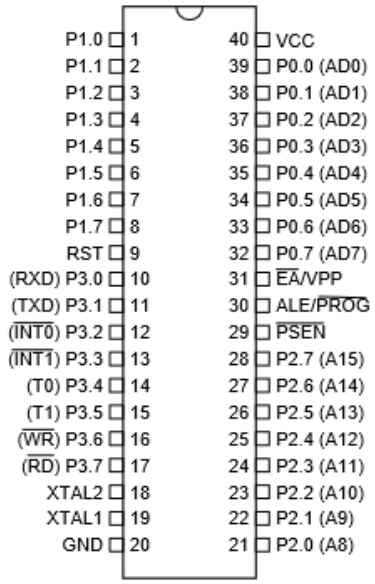


Рисунок 4.2– Виводи мікроконтролера AT89C51 корпусу типу PDIP

Регістр керування (SCON) призначений для прийому і зберігання коду восьмибітового слова, яке керує послідовним інтерфейсом. Позначення розрядів регістра SCON наведено в таблиці 4.1. Всі розряди регістра SCON програмно доступні для запису і читання.

Таблиця 4.1 – Позначення розрядів регістра SCON

Біти	7	6	5	4	3	2	1	0
Позначення	SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Розряди SM0, SM1 визначають режим роботи інтерфейсу, як зазначено в таблиці 4.2.

Інші біти регістра мають наступне призначення:

–SM2 – дозвіл багатопроцесорної роботи. У режимах 2 і 3 при SM2 = 1 прапорець RI не активізується, якщо дев’ятий прийнятий біт даних дорівнює "0". У режимі 1 при SM2 = 1 прапорець RI не активізується, якщо не прийнятий стоп–біт, що дорівнює "1". В режимі 0 біт SM2 повинен бути встановлений у "0";

–REN – дозвіл прийому послідовних даних. Встановлюється і скидається програмою відповідно для дозволу і заборони прийому;

–TB8 – дев’ятий біт даних, які передаються, у режимах 2 і 3. Встановлюється і скидається програмою;

–RB8 – дев’ятий біт прийнятих даних у режимах 2 і 3. У режимі 1, якщо SM2 = 0, RB8 є прийнятим стоп–бітом. У режимі 0 біт RB8 не використовується;

–TI – прапорець переривання передавача. Встановлюється апаратно наприкінці видачі 8–го біта в режимі 0 або на початку стоп–біта в інших режимах. Скидається програмно;

–RI – прапорець переривання приймача. Встановлюється апаратно наприкінці прийому 8–го біта в режимі 0 або через половину інтервалу стоп–біта в режимах 1, 2, 3 при SM2 = 0. При SM2 = 1 див. опис для біта SM2.

Таблиця 4.2 – Вплив розрядів SM0, SM1 SCON на режим роботи інтерфейсу

SM0	SM1	Режим	Найменування	Швидкість передачі
0	0	0	Регістр зсуву	$f_{BQ}/12$
0	1	1	8–бітовий універсальний асинхронний приймач–передавач (УАПП)	змінна, задається Т/Л1
1	0	2	9–бітовий УАПП	$f_{BQ}/64$ або $f_{BQ}/32$
1	1	3	9–бітовий УАПП	змінна, задається Т/Л1

Для програмування послідовного порту призначений також старший біт регістра PCON: PCON.7 (SMOD), за допомогою якого можна змінювати в 2 рази швидкість передачі даних послідовним портом (таблиці 4.3, 4.4).

Таблиця 4.3 – Позначення розрядів регістра PCON

Біти	7	6	5	4	3	2	1	0
Позначення	SMOD	–	–	–	GF1	GF0	PD	IDL

Таблиця 4.4 – Призначення розрядів регістра PCON

Біти	Найменування	Призначення бітів	Примітка
7	SMOD	Біт подвоєння швидкості передачі: при встановленні у стан «лог. 1» – швидкість передачі подвоюється	При роботі послідовного порту
6	–	Резервний	
5	–	Резервний	
4	–	Резервний	
3	GF1	Прапорець загального призначення	
2	GF0	Прапорець загального призначення	
1	PD	Біт включення режиму мікроспоживання: при встановленні у стан «лог. 1» – режим мікроспоживання	Якщо в PD і IDL одночасно записаний сигнал «лог. 1» – перевагу має PD
0	IDL	Біт холостого ходу: при встановленні у стан «лог. 1» – режим холостого ходу	

4.2.2 Робота послідовного порту в режимах 1, 2 і 3

4.2.2.1 Формування синхрочастот

У режимі 1 прийом/передача даних здійснюється у форматі восьмирозрядного УАПП. Через TxD передаються, а через RxD приймаються 10 біт: старт–біт (лог. 0), 8 біт даних і стоп–біт (лог. 1). Під час прийому стоп–біт заноситься у біт RB8 регістра SCON. Швидкість (частота) прийому/передачі визначається частотою переповнень таймера/лічильника1:

FOV T/C1. Ця частота усередині УАПП формується за схемою, наведеною на рисунку 4.3.

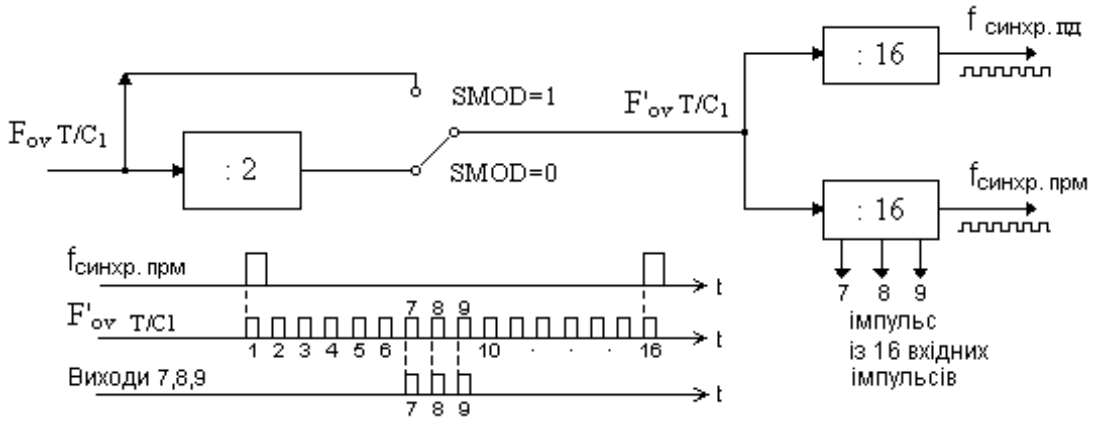


Рисунок 4.3 – Схема формування синхрочастот передачі і прийому всередині МК–51 для послідовного порту, що працює в режимах 1, 3

Режими 2 і 3 – це режими 9–розрядного УАПП з постійною (режим 2) та змінною (режим 3) швидкостями обміну. У цих режимах 11 біт передаються/приймаються відповідно через виводи TxD/RxD у наступній послідовності: старт–біт, 9 біт даних, стоп–біт. 9–й біт даних при передачі визначаються вмістом розряду TB8 регістра SCON. При прийомі 9–й біт даних заноситься у біт RB8 регістра SCON.

Швидкість (частота) прийому/передачі у режимі 2 (рисунок 4.4) програмно налаштовується на одну з двох можливих величин: $f_{BQ}/32$ і $f_{BQ}/64$, де f_{BQ} – частота синхронізації мікроконтролера.

У режимі 3 швидкість (частота) прийому/передачі визначається частотою переповнень таймера/лічильника1: FOV T/C1.

Розходження у швидкості (частоті) прийому/передачі є єдиною відмінністю між режимом 2 і режимом 3. В усьому іншому ці два режими є цілком ідентичними.

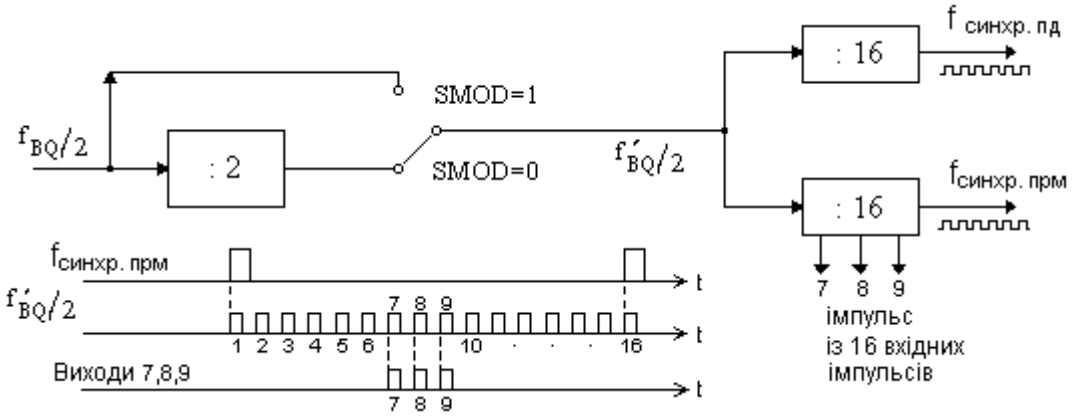


Рисунок 4.4 – Схема формування синхрочастот передачі і прийому всередині МК–51 для послідовного порту, що працює у режимі 2

Часові діаграми, що ілюструють роботу послідовного порту в режимах 2 і 3, наведені у [1…3].

Робота УАПП у режимах 2 і 3 дуже схожа на режим 1. Але при цьому існує ряд відмінностей:

–у форматі даних, якими відбувається обмін. Після восьми інформаційних, перед стоп–бітом, присутній 9–й біт, що програмується. При передачі значення 9–го біта визначається значенням розряду TB8 регістра SCON. При прийомі 9–й біт фіксується у розряді RB8 регістра SCON;

–якщо біт SM2 регістра SCON встановлений в одиницю, то повідомлення, у якому 9–й біт дорівнює нулю, бракується (втрачається). Тобто прапорець RI не встановлюється, і переривання основної програми при прийомі не відбувається;

–на часових діаграмах роботи передавача у режимах 2 і 3 у порівнянні з режимом 1 (рисунки 4.5, 4.6) буде доданий ще один біт (TB8) і цикл передачі подовжується на один такт (період частоти синхронізації передавача);

–на часових діаграмах роботи приймача у режимах 2, 3 буде доданий ще один прийнятий біт (RB8) перед стоп–бітом. Крім того,

детектування стоп–біта не відбувається, і прапорець RI встановлюється після 10–го зсуву, тобто після фіксації RB8.

4.2.2.2 Передача в режимі 1

Передача (рисунок 4.5) ініціюється будь–якою командою, що використовує SBUF в якості регістра призначення, у який виконується запис.

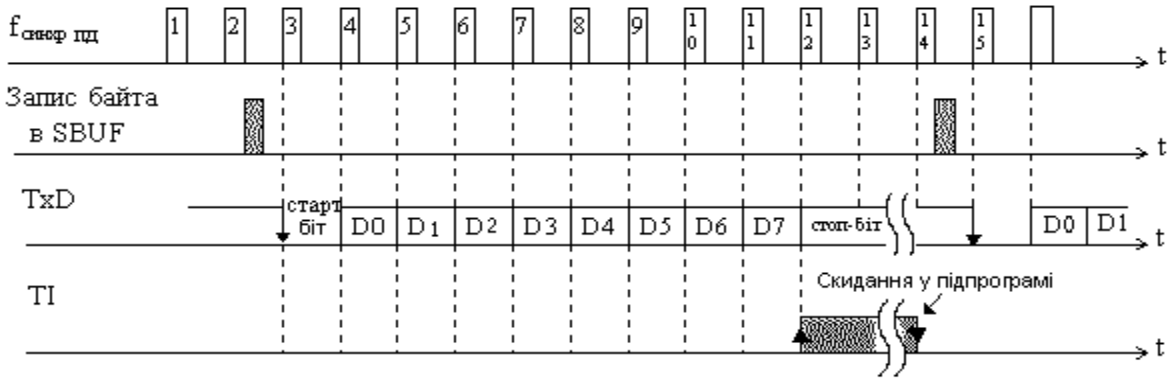


Рисунок 4.5 – Передача в режимі 1

Внутрішній імпульс мікроконтролера ЗАПИС У SBUF, що виробляється при цьому, завантажує призначений до передачі байт у молодші 8 розрядів регістра зсуву передавача та ініціює початок роботи блока керування передачею.

В режимі 1 регістр зсуву передавача має 9 розрядів і в його 9–й розряд за імпульсом ЗАПИС У SBUF заноситься "1" (стоп–біт).

Реально передача починається у фазі S1 P1 машинного циклу, яка слідує за найближчим після ЗАПИС У SBUF переповненням дільника на 16 у ланцюзі формування сигналу $f_{СИНХР.\ ПД}$ (рисунок 4.3). Таким чином, початок передачі синхронізований дільником на 16, а не імпульсом ЗАПИС У SBUF. Період сигналу $f_{СИНХР.\ ПД}$ (синхронізація передачі) визначає час, протягом якого біт, що видається, присутній на виході TxD (час передачі біта).

Передача починається встановленням активного рівня внутрішнього сигналу мікроконтролера ПОСИЛКА, поява якого викликає видачу на вихід TxD рівня старт–біта (нуль). Після цього через час передачі одного біта стає

активним внутрішній сигнал мікроконтролера ДАНІ, що дозволяє видачу вмісту регістра зсуву передавача на вихід TxD (вивід P3.0 мікроконтролера).

При появі активного сигналу ДАНІ старт–біт на виході TxD замінюється бітом D0 регістра зсуву передавача. По закінченні часу передачі біта D0 формується перший внутрішній імпульс мікроконтролера ЗСУВ, за яким вміст регістра зсуву передавача зсувається на один розряд, і біт D0 на виході TxD замінюється бітом D1. Усього формується 9 імпульсів ЗСУВ, у результаті чого на вихід TxD видаються 8 біт даних і стоп–біт. По закінченні видачі всіх біт посилки блок керування передачею встановлює прапорець переривання передавача TI і знімає сигнали ПОСИЛКА і ДАНІ. За перериванням відбувається скидання прапорця TI, запис нового байта тощо.

4.2.2.3 Прийом у режимі 1

Прийом (рисунок 4.6) починається при виявленні переходу сигналу на вході RxD з "1" в "0".

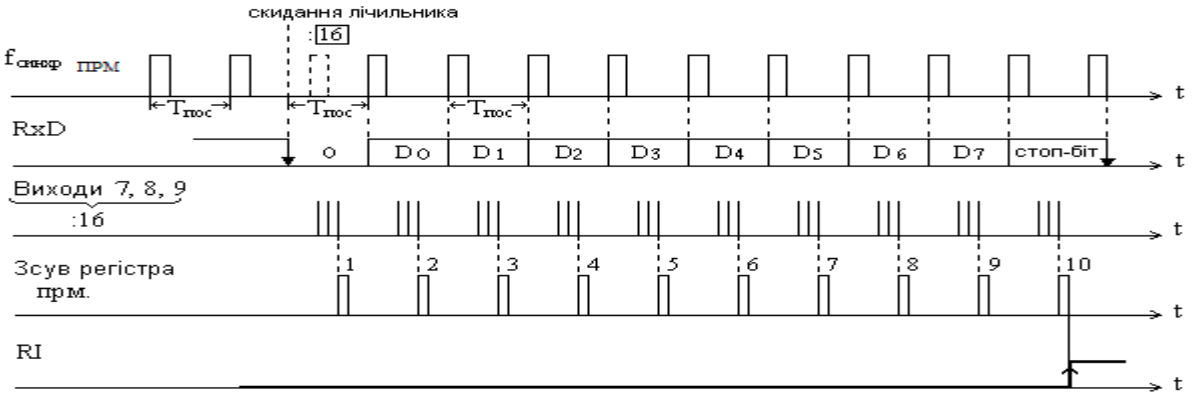


Рисунок 4.6 – Прийом у режимі 1

Для того, щоб виявити такий перехід, вхід RxD апаратно опитується з частотою $F'_{OV\ T/C1}$ (рисунок 4.3), тобто 16 разів на одну посилку. Коли перехід сигналу на вході RxD із "1" в "0" виявлений, негайно скидається лічильник (дільник на 16) у ланцюзі формування сигналу $f_{СИНХР.\ ПРМ}$ (рисунок 4.3), у результаті чого відбувається поєднання моментів переповнення цього лічильника (дільника на 16) (імпульси $f_{СИНХР.\ ПРМ}$ на рисунку 4.6) із

границями зміни бітів прийнятої посилки на вході RxD.

Шістнадцять станів лічильника (дільника на 16) поділяють час, протягом якого кожний біт прийнятої посилки присутній на вході RxD, на 16 фаз, з 1–ї по 16–у для кожного біта (див. $F_{OV\ T/C1}$ на рисунку 4.3). У фазах 7, 8 і 9 спеціальний пристрій мікроконтролера: біт–детектор, зчитує із входу RxD три значення прийнятих біт, за мажоритарним принципом "2 або 3 із 3–х", вибирає одне з них і подає його на вхід регістра зсуву приймача. Блок керування прийомом при цьому формує внутрішній імпульс мікроконтролера ЗСУВ, у результаті чого вміст регістра зсуву приймача зсувається на один розряд і прийнятий біт заноситься у регістр зсуву приймача. Усього формується 10 імпульсів ЗСУВ, а регістр зсуву приймача у режимі 1 є 9–розрядним. Тому після 10–го імпульсу ЗСУВ у регістрі зсуву приймача знаходяться біти даних D0…D7 і стоп–біт. Після 10–го імпульсу ЗСУВ блок керування прийомом завантажує дані з регістра зсуву приймача у регістр SBUF, завантажує стоп–біт з регістра зсуву приймача у розряд RB8 регістра SCON і встановлює прапорець переривання приймача RI. Сигнал завантаження SBUF, RB8 і встановлення RI виробляється блоком керування прийомом тільки в тому випадку, якщо в момент генерації останнього імпульсу ЗСУВ виконуються наступні умови:

– RI = 0 і

– або SM2 = 0, або (SM2 = 1) прийнятий стоп–біт дорівнює "1".

Якщо хоча б одна з цих умов не виконується, прийнята посилка безповоротно втрачається, а прапорець RI не встановлюється. Якщо обидві наведені умови виконані, стоп–біт надходить у RB8, вісім біт даних надходять у SBUF і встановлюється прапорець RI. У цей час, незалежно від виконання наведених вище умов, послідовний порт знову починає відслідковувати наявність переходу сигналу з "1" у "0" на вході RxD і прийом нового байта. До закінчення цього процесу попередній байт повинний бути прочитаний із буфера ПРМ (регістр SBUF), інакше буде накладення нового прийнятого байта на старий. У цьому випадку старий байт буде втрачено.

Якщо мажоритарний відбір при прийомі першого біта посилки (старт–біт) показує ненульове значення біта, всі пристрої блоку прийому скидаються, і починається відслідковування наступного переходу сигналу з "1" в "0" на вході RxD. Таким чином, забезпечується захист від помилкових старт–бітів.

Відмінність роботи інтерфейсу в режимі 3 в порівнянні з розглянутим вище режимом 1 полягає в тому, що перед стоп–бітом передається або приймається 9–й біт: TB8 або RB8.

4.2.2.4 Швидкість передачі–прийому даних через послідовний порт

Швидкість (частота передачі бітів) послідовного обміну Vпд в залежності від режиму роботи послідовного порту визначається або частотою синхронізації мікроконтролера f_{BQ} (режими 0 і 2), або частотою переповнення таймера/лічильника1 Fov T/C1 (режими 1 і 3).

У режимі 0 швидкість послідовного обміну максимальна. Вона постійна і складає:

$$V_{пд} = f_{BQ}/12 \ \ [біт/с]. \qquad (4.1)$$

При необхідності працювати зі зміненою у 2 рази швидкістю використовується режим 2 послідовного порту. У цьому режимі швидкість послідовної передачі залежить від стану біта SMOD регістра SCON і частоти f_{BQ}:

$$V_{пд} = (2^{SMOD}/64) * f_{BQ} \ \ [біт/с]. \qquad (4.2)$$

Тобто при SMOD = 0, Vпд = f_{BQ}/64, а при SMOD = 1, Vпд = f_{BQ}/32. За сигналом “скидання” біт SMOD встановлюється в нуль. Для встановлення біта SMOD використовуються команди з адресацією байтів, наприклад, команда MOV 87H, #80H.

У режимах 1, 3 є можливість змінити швидкість послідовної передачі у більш широкому діапазоні:

$$V_{пд} = (2^{SMOD} /32) * F_{OV\ T/C1}\ \ [біт/с], \qquad (4.3)$$

де $F_{OV\ T/C1}$ – частота переповнення таймера/лічильника1 (Т/Л1).

Для використання Т/Л1 в якості джерела для задання швидкості обміну необхідно:

- заборонити переривання від Т/Л1;
- запрограмувати роботу Т/Л1 в якості таймера або в якості лічильника, встановивши при цьому для нього один з режимів 0, 1 або 2;
- увімкнути Т/Л1 на лічбу.

Звичайно для синхронізації послідовного порту таймер Т/Л1 програмується в якості таймера у режим автозавантаження (режим 2). У цьому випадку швидкість послідовного обміну визначається за формулою:

$$V_{пд} = (2^{SMOD} * f_{BQ})/(32 * 12 * [256–(TH1)])\ \ [біт/с], \qquad (4.4)$$

де (TH1) – десятковий код вмісту TH1.

Якщо необхідний послідовний обмін з дуже низькою швидкістю, то можна використовувати Т/Л1 у режимі 16–розрядного таймера (режим 1), дозволивши при цьому переривання від Т/Л1 з метою перезавантаження TL1/TH1 у підпрограмі обслуговування переривання.

У таблиці 4.5 наведений ряд стандартних швидкостей послідовного обміну і те, як вони можуть бути реалізовані в МК–рі.

У таблиці 4.6 наведена зведена інформація з усіх чотирьох режимів роботи послідовного порту МК–ра сімейства МК–51.

Таблиця 4.5 – Формування стандартних швидкостей обміну через послідовний порт

Режим роботи послідовного порту	Швидкість прийому/ передачі, Кбод	f_{BQ}, МГц	SMOD	Розряди TMOD			TH1	Примітка
				C/T	M1	M0		
Режим 0	Макс.: 1000	12	X	X	X	X	X	
Режим 2	Макс.: 375	12	1	X	X	X	X	
Режим 1, 3	62,5	12	1	0	1	0	FFH	
	19,2	11,059	1	0	1	0	FDH	
	9,6	11,059	0	0	1	0	FDH	
	4,8	11,059	0	0	1	0	FAH	
	2,4	11,059	0	0	1	0	F4H	
	1,2	11,059	0	0	1	0	E8H	
	0,1375	11,986	0	0	1	0	1DH	
	0,110	6	0	0	1	0	72H	
	0,110	12	0	0	0	1	FEH	TL1 = E4H

Таблиця 4.6 – Зведена інформація з усіх режимів роботи послідовного порту

Режим обміну	Вид обміну	Розряди регістра SCON							Швидкість передачі	Примітка
		SM0	**SM1**	**SM2**	**REN**	TB8	RB8	**ПРАПОРЕЦЬ**		
0	ПД	0	0	0	–	–	–	TI	$f_{BQ}/12$	
	ПРМ				1			RI		Для ініціалізації прийому встановити: RI=0
1	ПД	0	1	–	–	–	–	TI	$\frac{2^{SMOD} \cdot F_{OV}}{32}$	
	ПРМ			0	1		стоп–біт	RI		
				1			1			
							0	–		Прийнятий байт губиться
2	ПД	1	0	–	–	9–й біт даних	–	TI	$\frac{2^{SMOD} \cdot f_{BQ}}{32}$	
	ПРМ			0	1		9–й біт даних	RI		
				1			1			
							0	–		Прийнятий байт губиться
3	ПРД	1	1	–	–	9–й біт даних	–	TI	$\frac{2^{SMOD} \cdot F_{OV}}{32}$	
	ПРМ			0	1		9–й біт даних	RI		
				1			1			
							0	–		Прийнятий байт губиться

4.3 Програмування модуля таймерів/лічильників

Для програмування модуля таймерів/лічильників (Т/Л) призначені:

–восьмирозрядний регістр режимів (TMOD);

–чотири старших біти восьмирозрядного регістра керування–статусу (TCON);

Регістр режимів (TMOD) призначений для прийому і збереження 8–бітного коду, що визначає:

–один із 4–х можливих режимів роботи кожного Т/Л;

–роботу в якості таймерів або лічильників;

–керування Т/Л від зовнішнього виводу.

Позначення розрядів регістра TMOD наведене в таблиці 4.7, а призначення розрядів – у таблиці 4.8.

При роботі в якості таймера вміст регістра Т/Л інкрементується в кожному машинному циклі, тобто Т/Л являється лічильником машинних циклів МК. Оскільки машинний цикл складається з 12 періодів частоти синхронізації МК f_{BQ}, то частота лічби в даному випадку дорівнює $f_{BQ}/12$.

Регістр керування–статусу (TCON) призначений для прийому і збереження 8–бітного коду керуючого слова. Позначення розрядів регістра TCON наведене в таблиці 4.9, а призначення розрядів – у таблиці 4.10.

Прапорці переповнення TF0 і TF1 встановлюються апаратно при переповненні відповідних Т/Л (перехід Т/Л із стану "всі одиниці" у стан "всі нулі"). Якщо при цьому переривання від відповідного Т/Л дозволено, то встановлення прапорця TF викликає переривання. Прапорці TF0 і TF1 скидаються апаратно при передачі керування програмі обробки відповідного переривання.

Таблиця 4.7 – Позначення розрядів регістра TMOD

Біти	7	6	5	4	3	2	1	0
Позн.	GATE1	$C/\overline{T}1$	M1.1	M0.1	GATE0	$C/\overline{T}0$	M1.0	M0.0

Таблиця 4.8 – Призначення розрядів регістра TMOD

<table>
<tr><th>Біти</th><th>Найменув.</th><th colspan="3">Призначення бітів</th><th>Примітка</th></tr>
<tr><td rowspan="6">0–1
4–5</td><td rowspan="6">M0–M1</td><td colspan="3">Визначають один із 4–х режимів роботи, окремо для Т/Л1 і Т/Л0.</td><td rowspan="6">Усі біти встановлюються програмно;
біти 0–3 визначають режим роботи Т/Л0,
біти 4–7 визначають режим роботи Т/Л1.</td></tr>
<tr><td>M1</td><td>M0</td><td>Режим</td></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>2</td></tr>
<tr><td>1</td><td>1</td><td>3</td></tr>
<tr><td>2, 6</td><td>$C/\overline{T}0$
$C/\overline{T}1$</td><td colspan="3">Визначають роботу в якості:
C/T0, C/T1 = 0 – таймера
C/T0, C/T1 = 1 – лічильника</td></tr>
<tr><td>3, 7</td><td>GATE</td><td colspan="3">Дозволяє керувати таймером від зовнішнього виводу ($\overline{INT0}$ – для Т/Л0, $\overline{INT1}$ – для Т/Л1).
GATE = 0 – керування заборонено
GATE = 1 – керування дозволено</td></tr>
</table>

Таблиця 4.9 – Позначення розрядів регістра TCON

Біти	7	6	5	4	3	2	1	0
Позначення	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Таблиця 4.10 – Призначення розрядів регістра TCON

Біти	Найменув.	Призначення бітів	Примітка
6 4	TR1 TR0	Біти включення Т/Л, окремо для Т/Л0 і Т/Л1. TR = 0 – вимкнений, TR = 1 – увімкнений.	Біти встановлюються і скидаються програмно. Доступні за читанням.
7 5	TF1 TF0	Прапорці переповнення Т/Л.	Біти скидаються і встановлюються апаратно і програмно. Доступні за читанням.
2 0	IT1 IT0	Біти, що визначають вид переривання за входами INT1, INT0. IT = 0 – переривання за рівнем (низьким), IT = 1 – переривання за фронтом (перехід із "1" в "0")	Біти встановлюються і скидаються програмно. Доступні за читанням.
3 1	IE1 IE0	Прапорці запиту зовнішніх переривань за входами INT1, INT0.	Біти скидаються і встановлюються апаратно і програмно. Доступні за читанням.
			Біти 4, 5 відносяться до Т/Л0; біти 6, 7 – до Т/Л1. Біти 0, 1 визначають зовнішні переривання за входом INT0, біти 2, 3 – за входом INT1.

Прапорці TF0 і TF1 програмно доступні і можуть бути встановлені/скинуті програмою. Використовуючи цей механізм, переривання по TF0 і TF1 можуть бути викликані (встановлення TF) і скасовані (скидання TF) програмою.

Біти 0…3 регістра TCON використовуються при програмуванні системи переривань мікроконтролера і до програмування модуля таймерів/лічильників не мають відношення.

4.4 Створення нової програми мовою Асемблер

Для того, щоб створити нову програму на мові Асемблер у середовищі Proteus необхідно на вкладці Source Code, де відображається структура проекту натиснути праву клавішу та обрати “Add new file” (рисунок 4.7).

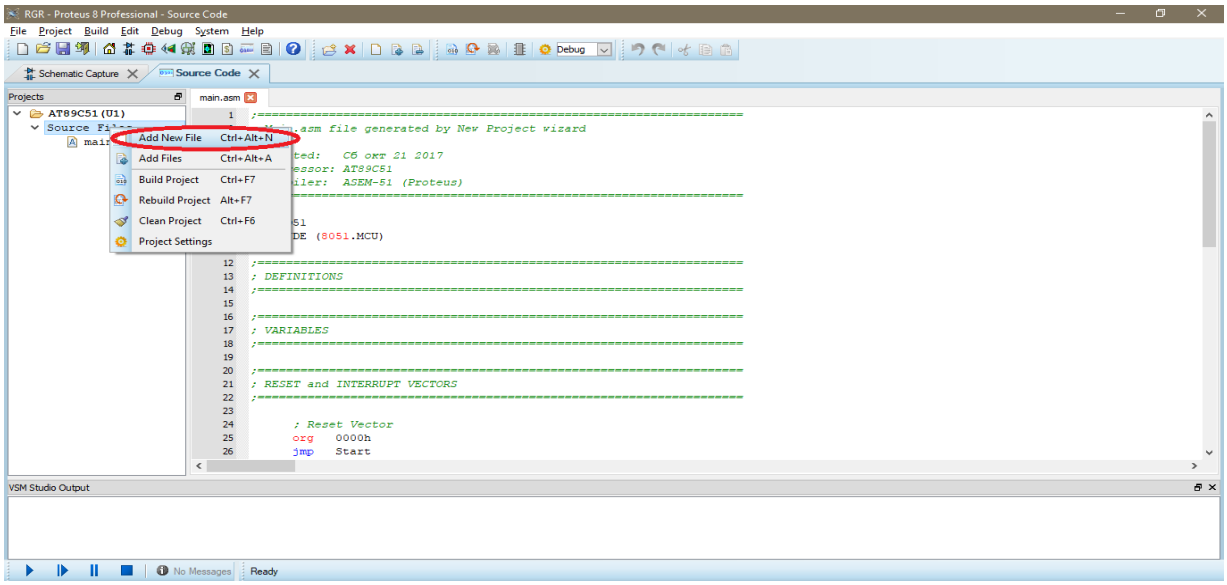


Рисунок 4.7–Створення нового файлу на мові Асемблер

Далі обрати місце збереження та формат файлу (рисунок 4.8). Для програми на мові Асемблер обираємо тип файлу ASM і натискаємо на кнопку збереження.

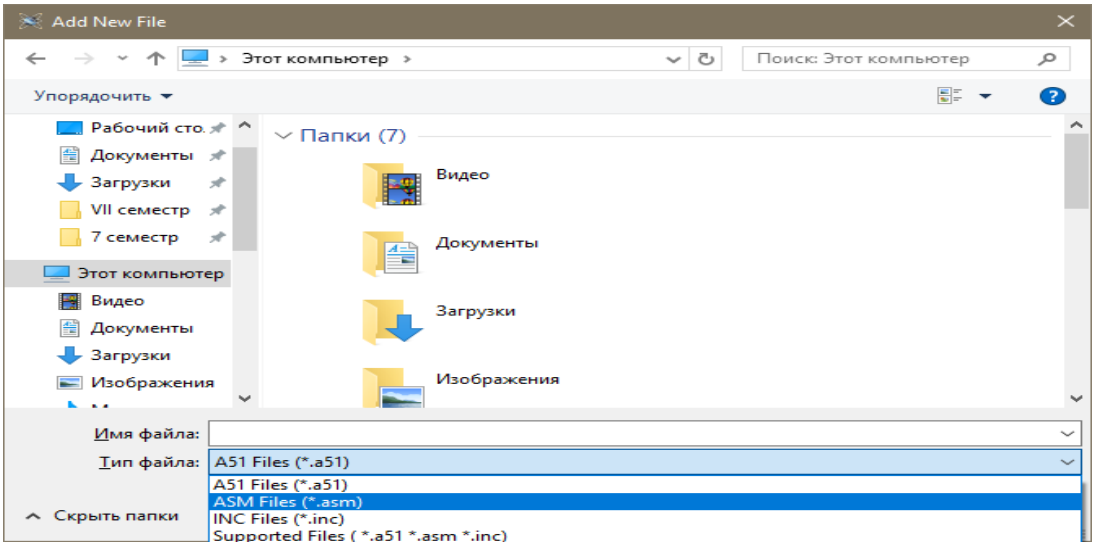


Рисунок 4.8–Обрання місця збереження файлу

Після цього, з лівої сторони нам буде доступна нова вкладка з нашим файлом, натискаємо на неї лівою клавішею миші і перед нами буде вікно для створення та редагування нашої програми (рисунок 4.9):

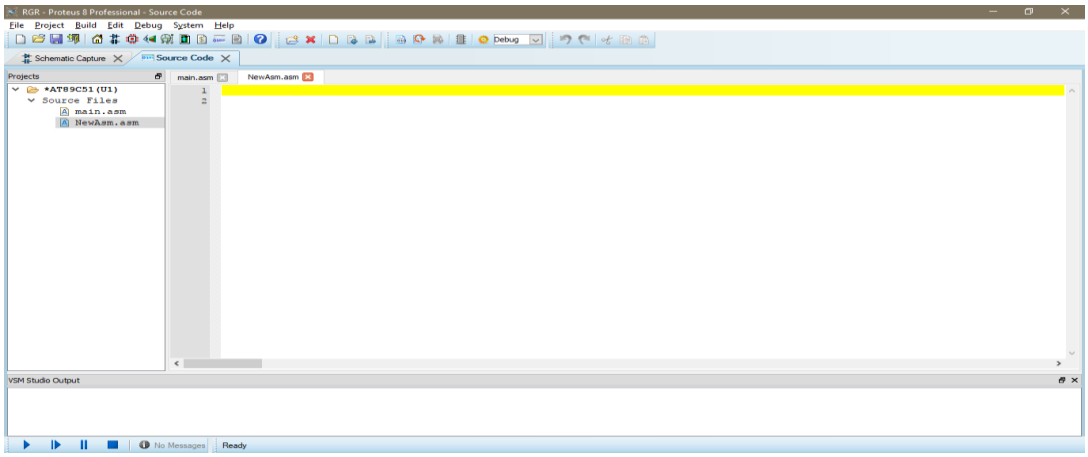


Рисунок 4.9–Вікно редагування файла

Після цього натиснути кнопку START та почати процес моделювання.

4.5 Внесення змін в програмі в процесі моделювання

Для того щоб внести зміни в процесі моделювання необхідно зупинити будь–які процеси відлагодження та симуляції, натиснувши на кнопку STOP.

Перейти на вкладку Source Code, де нам буде доступна опція редагування тексту програми (рисунок 4.10).

Після редагування натиснути кнопку START, та почати процес моделювання.

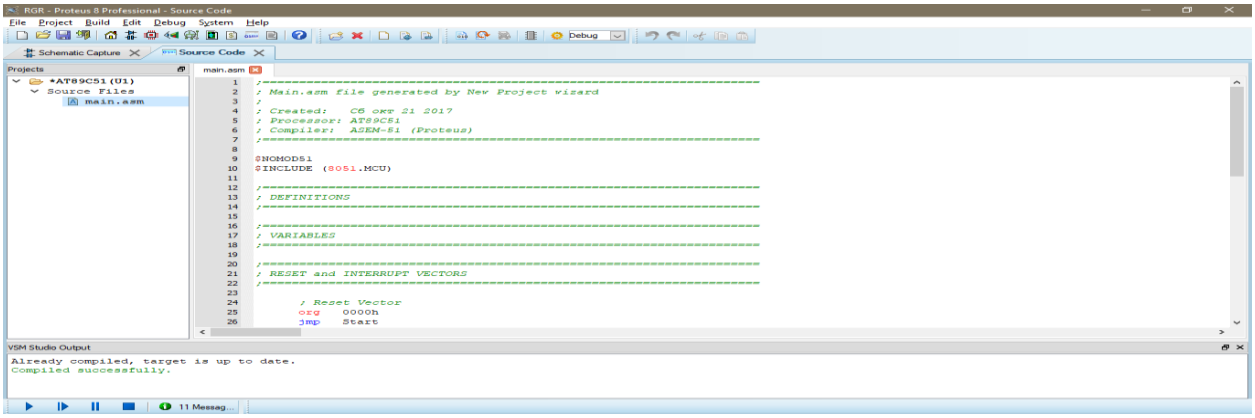


Рисунок 4.10

4.6 Порядок моделювання

4.6.1 Перш за все, потрібно пересвідчитись, що Terminal правильно налаштовано. Для цього треба двічі клацнути на нього лівою кнопкою миші, та виставити відповідні налаштування. Нижче наведено приклад для випадку, коли потрібно передавати зі швидкістю 110 біт/сек: 8 біт даних та один стоп–біт з перевіркою на непарність (рисунок 4.11). Якщо треба використовувати перевірку на парність, то це треба вказати в опції Parity.

4.6.2 Далі натисніть Start VSM Debugging (рисунок 4.12).

4.6.3 Потім натисніть на кнопку Pause VSM Debugging

4.6.4 Перейти на вкладку Source Code та поставити дві точки зупинки. Для цього потрібно лівою кнопкою миші двічі клацнути з лівої сторони двох команд, як показано на рисунку 4.13.

4.6.5 Натисніть Run Simulation (F12).

4.6.6 Перейти до вікна Virtual Terminal, натисніть правою кнопкою миші на вікні, та оберіть Hex Display Mode та Echo Typed Characters (рисунок 4.14).

Якщо вікно Virtual Terminal закрито, то його можна відкрити із вкладки Debug.

4.6.7 Подивіться на вікно осцилографа. Для того, щоб побачити переданий сигнал на осцилографі, натисніть кнопку на опції One –Shot (один кадр), як показано на рисунку 4.15.

4.6.8 Далі введіть символ, який хочемо надіслати від віртуального терміналу до мікроконтролера через УАПП. При введенні символу у вікні терміналу повинен стояти курсор. Символ, який ми вводимо з клавіатури, віртуальний термінал перетворює в ASCII–код згідно таблиці 4.11.

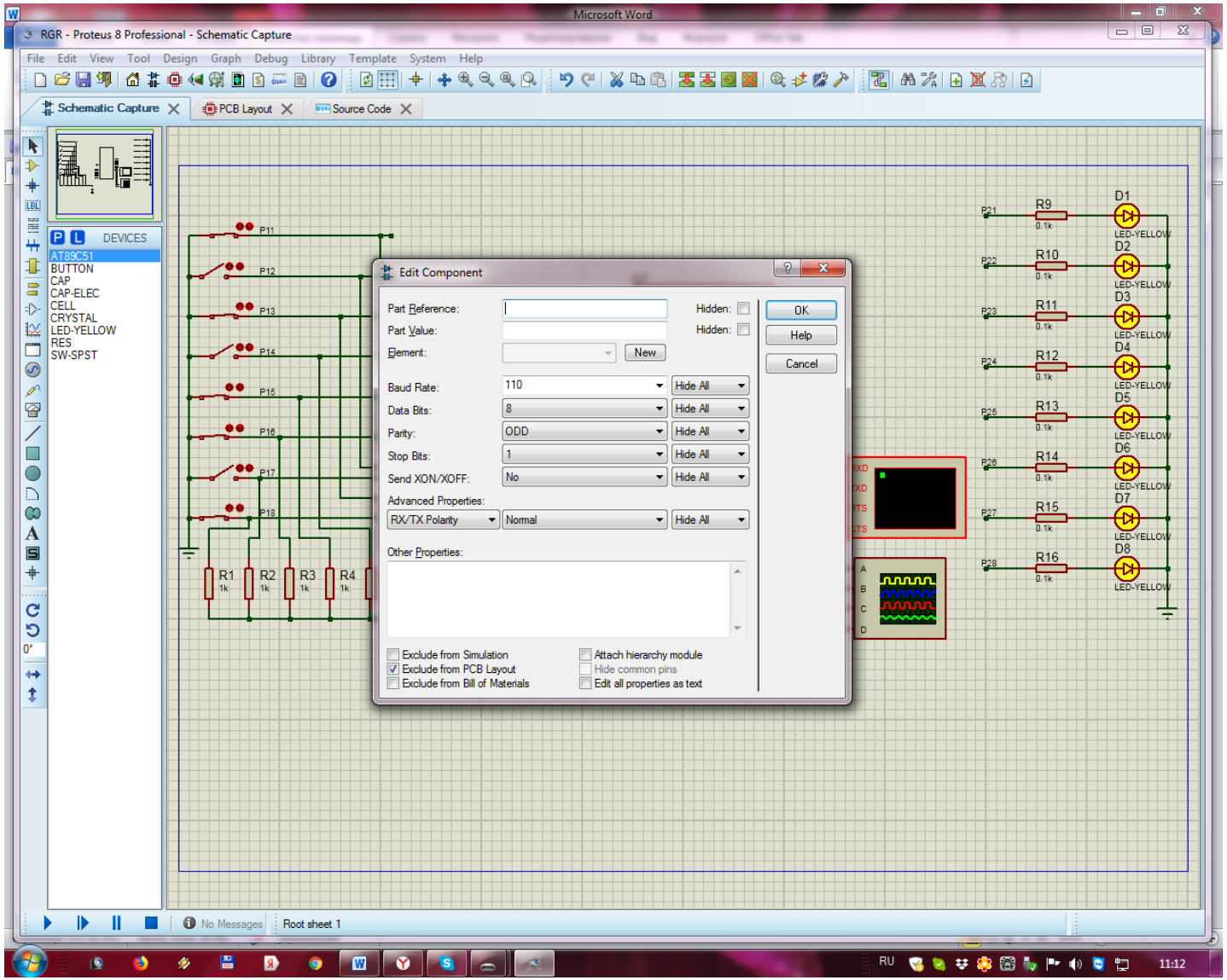

Microsoft Word
RGR - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
Schematic Capture
PCB Layout
Source Code
DEVICES
AT89C51
BUTTON
CAP
CAP-ELEC
CELL
CRYSTAL
LED-YELLOW
RES
SW-SPST
Edit Component
Part Reference:
Part Value:
Element:
New
Hidden:
OK
Help
Cancel
Baud Rate: 110
Data Bits: 8
Parity: ODD
Stop Bits: 1
Send XON/XOFF: No
Hide All
Advanced Properties:
RX/TX Polarity
Normal
Other Properties:
Exclude from Simulation
Exclude from PCB Layout
Exclude from Bill of Materials
Attach hierarchy module
Hide common pins
Edit all properties as text
R1
R2
R3
R4
R9
R10
R11
R12
R13
R14
R15
R16
D1
D2
D3
D4
D5
D6
D7
D8
LED-YELLOW
No Messages
Root sheet 1
RU
11:12

Рисунок 4.11

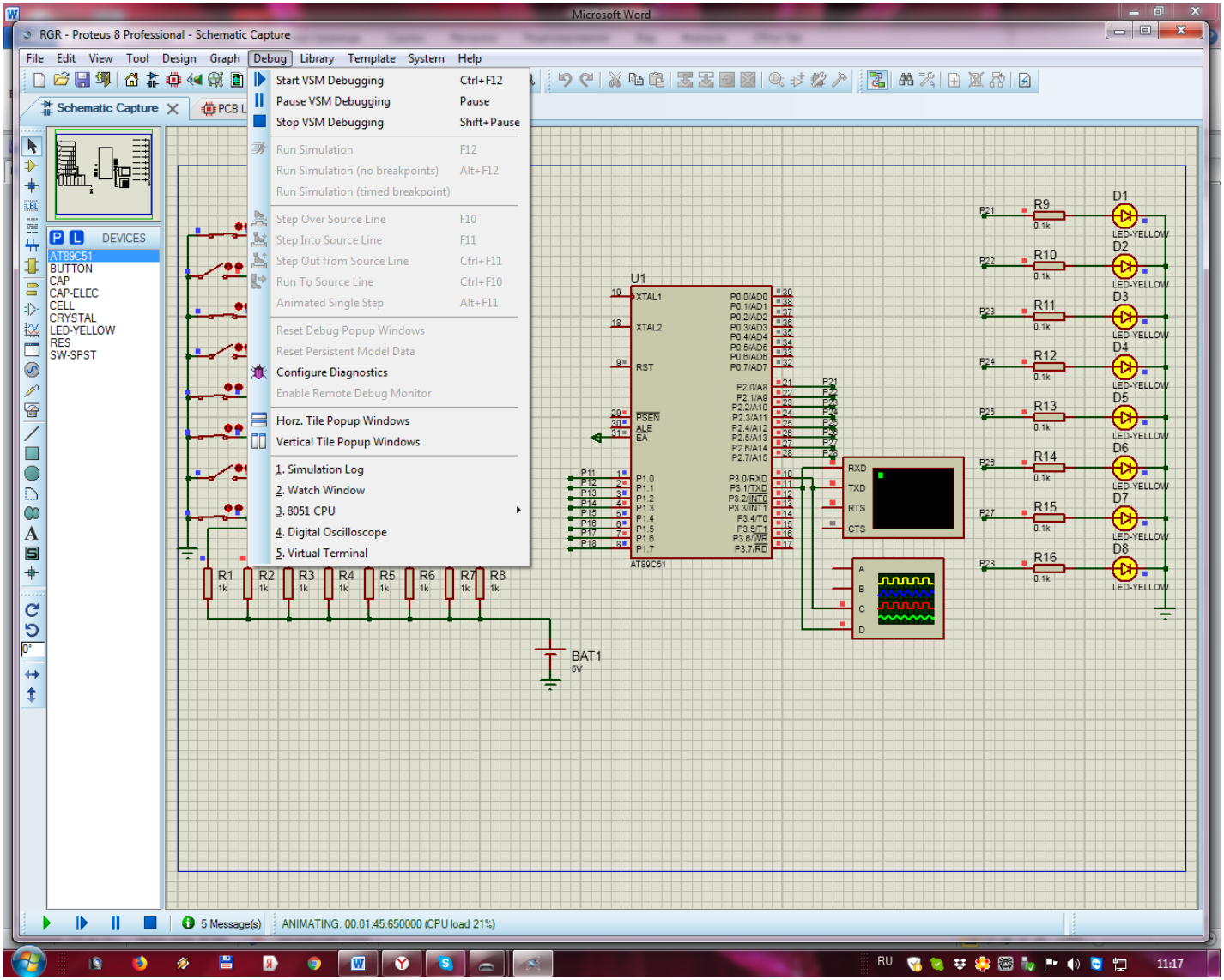

Microsoft Word
RGR - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
Start VSM Debugging Ctrl+F12
Pause VSM Debugging Pause
Stop VSM Debugging Shift+Pause
Run Simulation F12
Run Simulation (no breakpoints) Alt+F12
Run Simulation (timed breakpoint)
Step Over Source Line F10
Step Into Source Line F11
Step Out from Source Line Ctrl+F11
Run To Source Line Ctrl+F10
Animated Single Step Alt+F11
Reset Debug Popup Windows
Reset Persistent Model Data
Configure Diagnostics
Enable Remote Debug Monitor
Horz. Tile Popup Windows
Vertical Tile Popup Windows
1. Simulation Log
2. Watch Window
3. 8051 CPU
4. Digital Oscilloscope
5. Virtual Terminal
DEVICES
AT89C51
BUTTON
CAP
CAP-ELEC
CELL
CRYSTAL
LED-YELLOW
RES
SW-SPST
U1
AT89C51
R1 R2 R3 R4 R5 R6 R7 R8
R9 R10 R11 R12 R13 R14 R15 R16
D1 D2 D3 D4 D5 D6 D7 D8
BAT1
5V
RXD TXD RTS CTS
5 Message(s)
ANIMATING: 00:01:45.650000 (CPU load 21%)
11:17

Рисунок 4.12

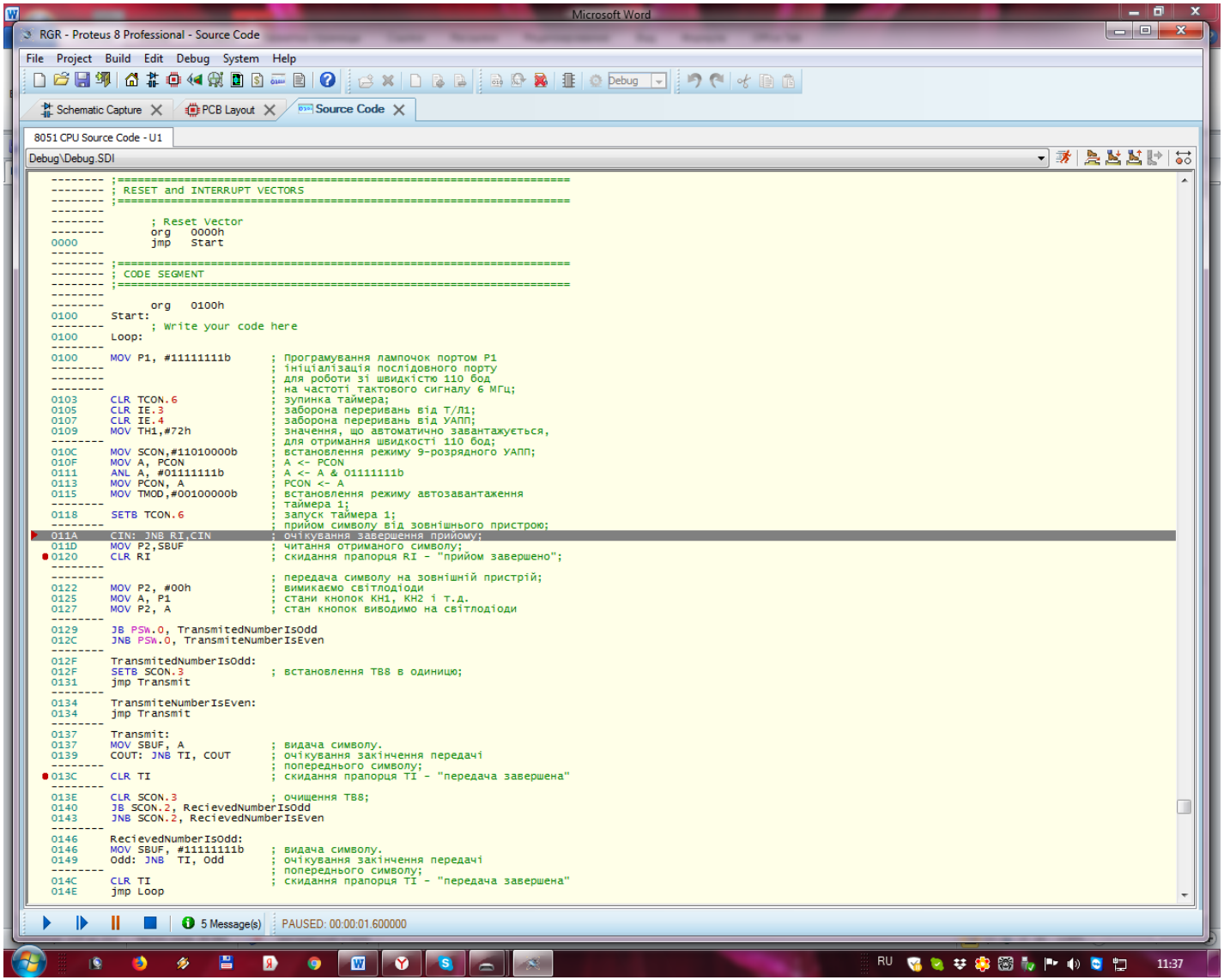

Microsoft Word
RGR - Proteus 8 Professional - Source Code
File Project Build Edit Debug System Help
Schematic Capture
PCB Layout
Source Code
8051 CPU Source Code - U1
Debug\Debug.SDI
; RESET and INTERRUPT VECTORS
; Reset Vector
org 0000h
jmp Start
; CODE SEGMENT
org 0100h
Start:
; Write your code here
Loop:
MOV P1, #11111111b ; Програмування лампочок портом P1
; ініціалізація послідовного порту
; для роботи зі швидкістю 110 бод
; на частоті тактового сигналу 6 МГц;
CLR TCON.6 ; зупинка таймера;
CLR IE.3 ; заборона переривань від Т/Л1;
CLR IE.4 ; заборона переривань від УАПП;
MOV TH1,#72h ; значення, що автоматично завантажується,
; для отримання швидкості 110 бод;
MOV SCON,#11010000b ; встановлення режиму 9-розрядного УАПП;
MOV A, PCON ; A <- PCON
ANL A, #01111111b ; A <- A & 01111111b
MOV PCON, A ; PCON <- A
MOV TMOD,#00100000b ; встановлення режиму автозавантаження
; таймера 1;
SETB TCON.6 ; запуск таймера 1;
; прийом символу від зовнішнього пристрою;
CIN: JNB RI,CIN ; очікування завершення прийому;
MOV P2,SBUF ; читання отриманого символу;
CLR RI ; скидання прапорця RI - "прийом завершено";
; передача символу на зовнішній пристрій;
MOV P2, #00h ; вимикаємо світлодіоди
MOV A, P1 ; стани кнопок KH1, KH2 і т.д.
MOV P2, A ; стан кнопок виводимо на світлодіоди
JB PSW.0, TransmitedNumberIsOdd
JNB PSW.0, TransmiteNumberIsEven
TransmitedNumberIsOdd:
SETB SCON.3 ; встановлення TB8 в одиницю;
jmp Transmit
TransmiteNumberIsEven:
jmp Transmit
Transmit:
MOV SBUF, A ; видача символу.
COUT: JNB TI, COUT ; очікування закінчення передачі
; попереднього символу;
CLR TI ; скидання прапорця TI - "передача завершена"
CLR SCON.3 ; очищення TB8;
JB SCON.2, RecievedNumberIsOdd
JNB SCON.2, RecievedNumberIsEven
RecievedNumberIsOdd:
MOV SBUF, #11111111b ; видача символу.
Odd: JNB TI, Odd ; очікування закінчення передачі
; попереднього символу;
CLR TI ; скидання прапорця TI - "передача завершена"
jmp Loop
5 Message(s)
PAUSED: 00:00:01.600000
RU
11:37

Рисунок 4.13

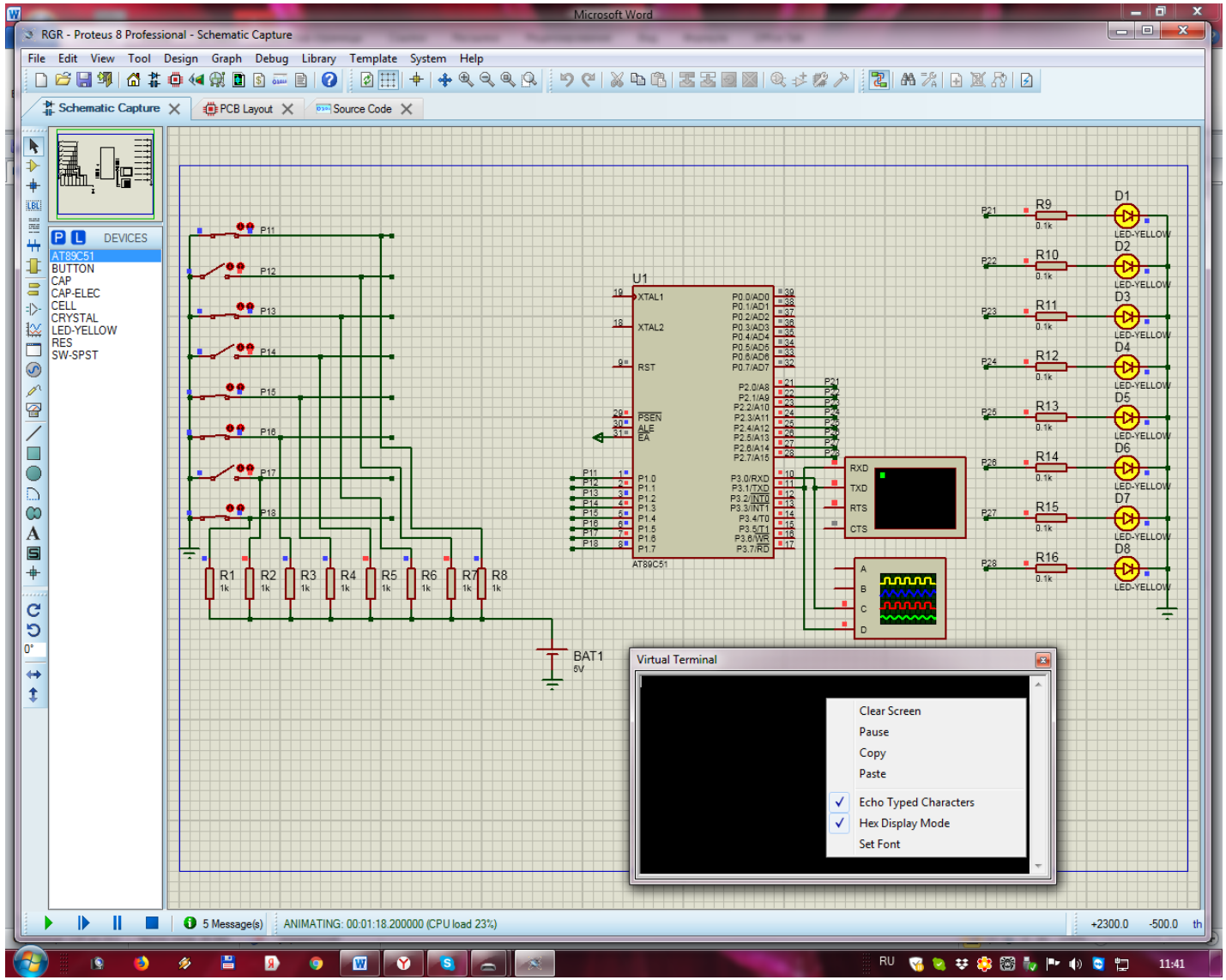

Microsoft Word
RGR - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
Schematic Capture
PCB Layout
Source Code
DEVICES
AT89C51
BUTTON
CAP
CAP-ELEC
CELL
CRYSTAL
LED-YELLOW
RES
SW-SPST
U1
XTAL1
XTAL2
RST
PSEN
ALE
EA
AT89C51
R1
R2
R3
R4
R5
R6
R7
R8
BAT1
5V
R9
R10
R11
R12
R13
R14
R15
R16
D1
D2
D3
D4
D5
D6
D7
D8
LED-YELLOW
RXD
TXD
RTS
CTS
Virtual Terminal
Clear Screen
Pause
Copy
Paste
Echo Typed Characters
Hex Display Mode
Set Font
5 Message(s)
ANIMATING: 00:01:18.200000 (CPU load 23%)
+2300.0
-500.0
RU
11:41

Рисунок 4.14

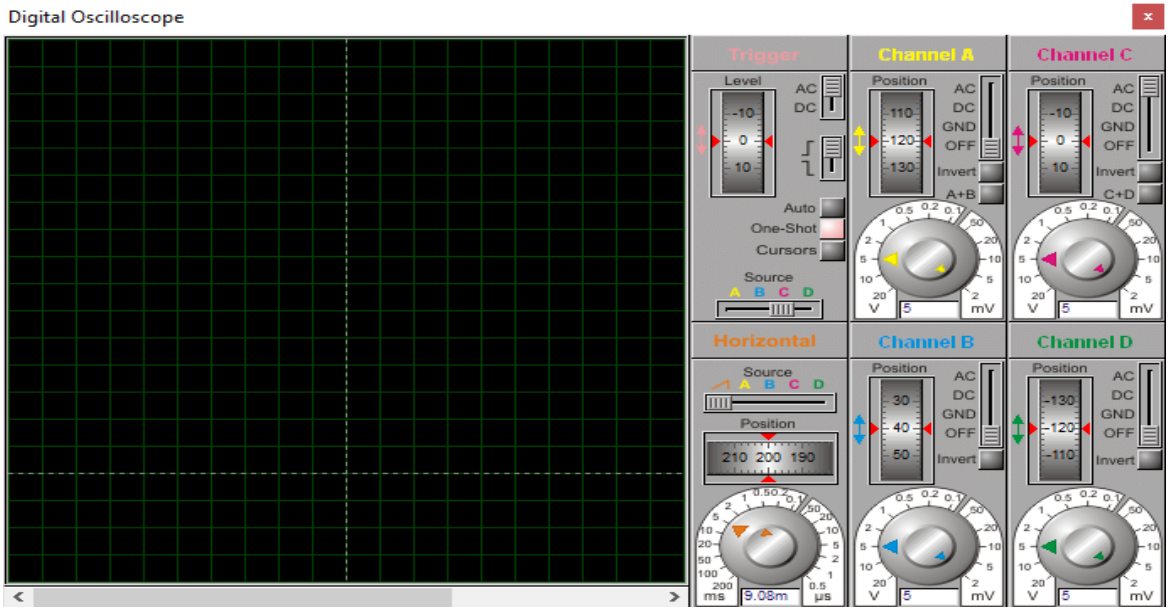


Рисунок 4.15

Введіть, наприклад, цифру 7, ASCII код якої 37 (hex) = 0011 0111 (bin). Програма перейде на рядок, де ми поставили першу точку зупину. Перейдіть на вкладку зі схемою Schematic Capture. Зверніть увагу на лампочки (рисунок 4.16) та на комбінацію 8 розрядів на виході порту P2, до якого підключено світлодіоди. Ця комбінація дорівнює: 0011 0111 (bin).

Лампочки (світлодіоди) точно відображають бінарний код символу, який ми відправили (якщо дивитися знизу уверх, а бінарне число читати зліва направо), нуль – лампочка вимкнена, один – лампочка увімкнена (прийнято символ 00110111b = 37h). Подивіться на вікно віртуального терміналу (рисунок 4.16) та побачте шістнадцятковий код 37, що відповідає ASCII–коду цифри 7, яку було натиснуто. Подивіться на екран осцилографа (лінія червоного кольору, рисунок 4.16) та побачте передачу через послідовний порт віртуального терміналу (лінія TxD) у старт-стопному режимі символу 37h=00110111b. Зверніть увагу, що передача ведеться починаючи з молодшого розряду: спочатку іде нульовий старт-біт, далі передаються: 111, 0, 11 та 00. Оскільки згідно з рисунком 4.11 виконується перевірка на непарність, то далі передано одиничний контрольний біт. Закінчується передача від терміналу одиничним стоп-бітом.

Таблиця 4.11 – ASCII–коди

DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR
0	0		53	35	5	106	6A	j	159	9F	Я
1	1		54	36	6	107	6B	k	160	A0	а
2	2		55	37	7	108	6C	l	161	A1	б
3	3		56	38	8	109	6D	m	162	A2	в
4	4		57	39	9	110	6E	n	163	A3	г
5	5		58	3A	:	111	6F	o	164	A4	д
6	6		59	3B	;	112	70	p	165	A5	е
7	7		60	3C	<	113	71	q	166	A6	ж
8	8		61	3D	=	114	72	r	167	A7	з
9	9		62	3E	>	115	73	s	168	A8	и
10	A		63	3F	?	116	74	t	169	A9	й
11	B		64	40	@	117	75	u	170	AA	к
12	C		65	41	A	118	76	v	171	AB	л
13	D		66	42	B	119	77	w	172	AC	м
14	E		67	43	C	120	78	x	173	AD	н
15	F		68	44	D	121	79	y	174	AE	о
16	10		69	45	E	122	7A	z	175	AF	п
17	11		70	46	F	123	7B	{	176	B0	░
18	12		71	47	G	124	7C	|	177	B1	▒
19	13		72	48	H	125	7D	}	178	B2	▓
20	14		73	49	I	126	7E	~	179	B3	│
21	15		74	4A	J	127	7F		180	B4	┤
22	16		75	4B	K	128	80	А	181	B5	╡
23	17		76	4C	L	129	81	Б	182	B6	╢
24	18		77	4D	M	130	82	В	183	B7	╖
25	19		78	4E	N	131	83	Г	184	B8	╕
26	1A		79	4F	O	132	84	Д	185	B9	╣
27	1B		80	50	P	133	85	Е	186	BA	║
28	1C		81	51	Q	134	86	Ж	187	BB	╗
29	1D		82	52	R	135	87	З	188	BC	╝
30	1E		83	53	S	136	88	И	189	BD	╜
31	1F		84	54	T	137	89	Й	190	BE	╛

Продовження таблиці 4.11

32	20		85	55	U	138	8A	К	191	BF	┐
33	21	!	86	56	V	139	8B	Л	192	C0	└
34	22	“	87	57	W	140	8C	М	193	C1	┴
35	23	#	88	58	X	141	8D	Н	194	C2	┬
36	24	$	89	59	Y	142	8E	О	195	C3	├
37	25	%	90	5A	Z	143	8F	П	196	C4	─
38	26	&	91	5B	[	144	90	Р	197	C5	┼
39	27	′	92	5C	\	145	91	С	198	C6	╞
40	28	(	93	5D	]	146	92	Т	199	C7	╟
41	29	)	94	5E	^	147	93	У	200	C8	╚
42	2A	*	95	5F	–	148	94	Ф	201	C9	╔
43	2B	+	96	60	‘	149	95	Х	202	CA	╩
44	2C	′	97	61	a	150	96	Ц	203	CB	╦
45	2D	D	98	62	b	151	97	Ч	204	CC	╠
46	2E	.	99	63	c	152	98	Ш	205	CD	═
47	2F	/	100	64	d	153	99	Щ	206	CE	╬
48	30	0	101	65	e	154	9A	Ъ	207	CF	╧
49	31	1	102	66	f	155	9B	Ы	208	D0	╨
50	32	2	103	67	g	156	9C	Ь	209	D1	╤
51	33	3	104	68	h	157	9D	Э	210	D2	╥
52	34	4	105	69	i	158	9E	Ю	211	D3	╙
212	D4	╙	224	E0	p	236	EC	ь	248	F8	Ї
213	D5	╒	225	E1	c	237	ED	э	249	F9	ї
214	D6	╓	226	E2	т	238	EE	ю	250	FA	Ў
215	D7	╫	227	E3	y	239	EF	я	251	FB	ў
216	D8	╪	228	E4	ф	240	F0	Ё	252	FC	ⁿ
217	D9	┘	229	E5	x	241	F1	ё	253	FD	²
218	DA	┌	230	E6	ц	242	F2	Ґ	254	FE	■
219	DB		231	E7	ч	243	F3	ґ	255	FF	
219	DB		231	E7	ч	243	F3	ґ	255	FF	

Продовження таблиці 4.11

220	DC	▄	232	E8	Ш	244	F4	Є			
221	DD	▌	233	E9	Щ	245	F5	є			
222	DE	▐	234	EA	Ъ	246	F6	І			
223	DF	▀	235	EB	Ы	247	F7	і			

4.6.9 Перевірте як мікроконтролер через УАПП відправить символ, який закодували кнопками: замкнута кнопка – нуль, а розімкнута – одиниця (рисунок 4.16).

Якщо дивитися на кнопки знизу уверх, то на рисунку 4.16 набрано число 0100 1010 (bin) = 4A (hex).

Для того, щоб побачити переданий сигнал на осцилографі, потрібно натиснути кнопку на опції One –Shot (один кадр), як показано на рисунку 4.15.

4.6.10 Натисіть Run Simulation для продовження виконання програми Програма дійде до другої точки зупину. Подивіться на лампочки та кнопки (рисунок 4.17). Лампочки повністю відповідають тому, що було закодовано кнопками, а саме: 0100 1010 (bin) = 4A (hex).

4.6.11 Розберіть більш детально сигнал, що знімається із лінії Tx мікроконтролера та відображається на осцилографі (рисунок 4.17, зелений колір).

Перший перепад із високого рівня у низький – старт–біт. Потім ідуть вісім інформаційних бітів, починаючи з молодшого розряду, – 0101 0010, які відображаються на віртуальному терміналі як 0100 1010 (bin) = 4A (hex). Далі іде 9–й біт TB8, який дорівнює 1, тому що число одиниць в байті, який було передано – непарне (розряд P в регістрі прапорців дорівнює 1). Останній 10–й біт, який дорівнює 1, це стоп–біт. Щоб його побачити інколи треба виконати останню частину програми (див. 4.6.13). Лінія червоного кольору на рисунку 4.17 відображає передачу ASCII коду цифри 7: 37 (hex) = 0011 0111 (bin).

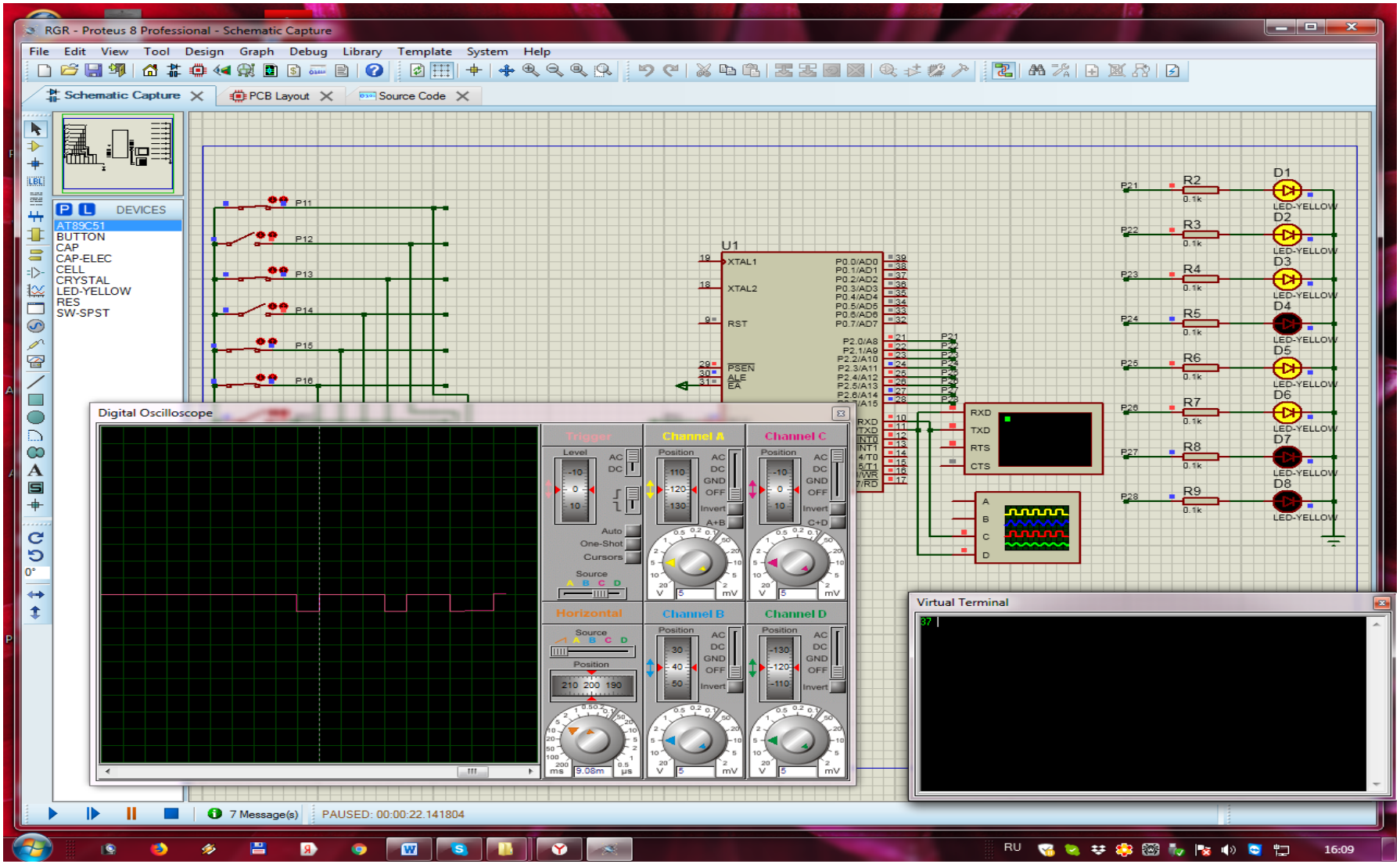

RGR - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
Schematic Capture
PCB Layout
Source Code
DEVICES
AT89C51
BUTTON
CAP
CAP-ELEC
CELL
CRYSTAL
LED-YELLOW
RES
SW-SPST
U1
XTAL1
XTAL2
RST
PSEN
ALE
EA
Digital Oscilloscope
Trigger
Channel A
Channel C
Channel B
Channel D
Horizontal
Virtual Terminal
RXD
TXD
RTS
CTS
7 Message(s)
PAUSED: 00:00:22.141804
16:09

Рисунок 4.16

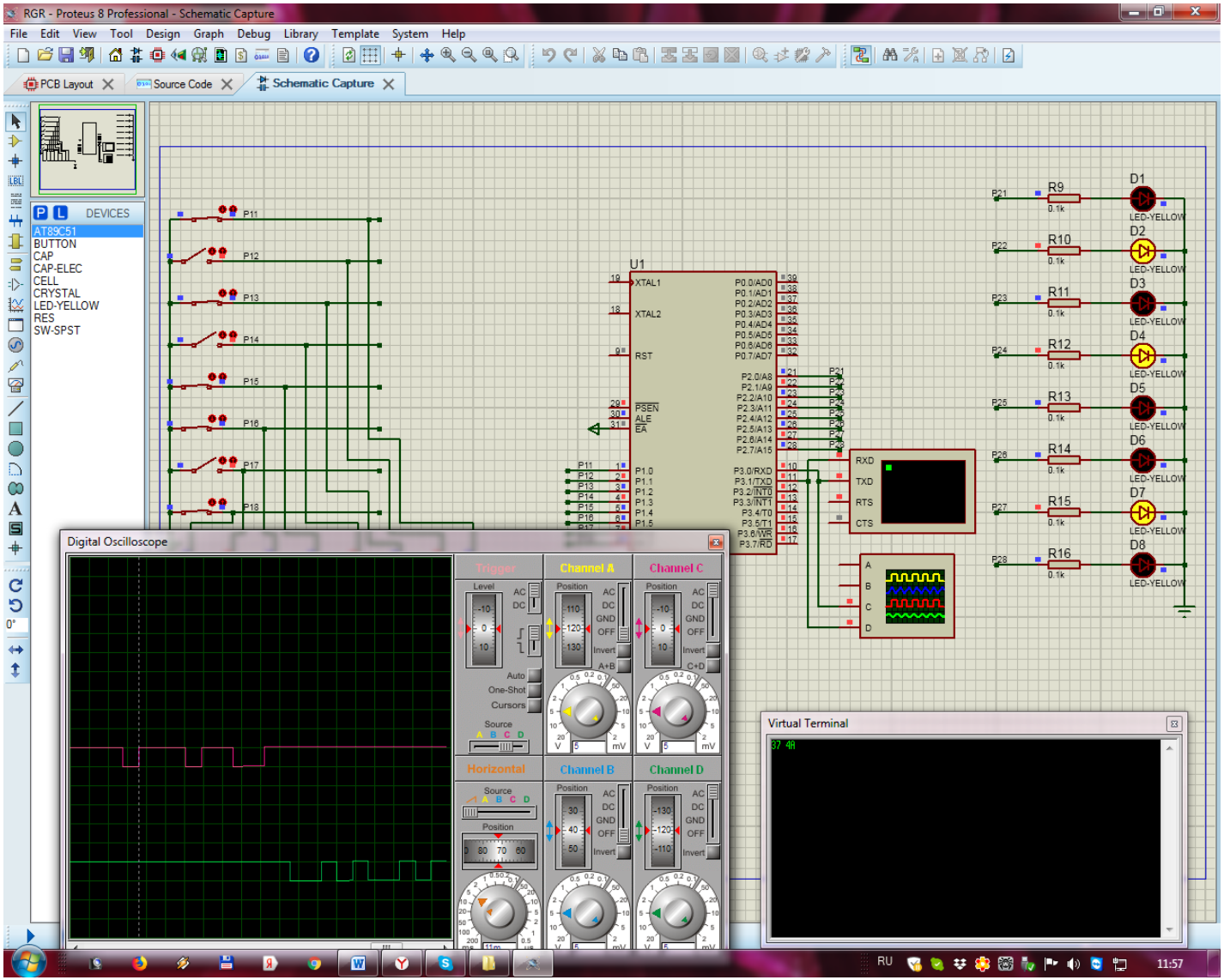

RGR - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
PCB Layout
Source Code
Schematic Capture
DEVICES
AT89C51
BUTTON
CAP
CAP-ELEC
CELL
CRYSTAL
LED-YELLOW
RES
SW-SPST
U1
XTAL1
XTAL2
RST
PSEN
ALE
EA
RXD
TXD
RTS
CTS
R9
R10
R11
R12
R13
R14
R15
R16
D1
D2
D3
D4
D5
D6
D7
D8
LED-YELLOW
Digital Oscilloscope
Trigger
Channel A
Channel C
Horizontal
Channel B
Channel D
Virtual Terminal
RU
11:57

Рисунок 4.17

На рисунку 4.18 наведено приклад передачі від терміналу на осцилограф ASCII-коду цифри 5: 35 (hex) = 0011 0101 (bin).

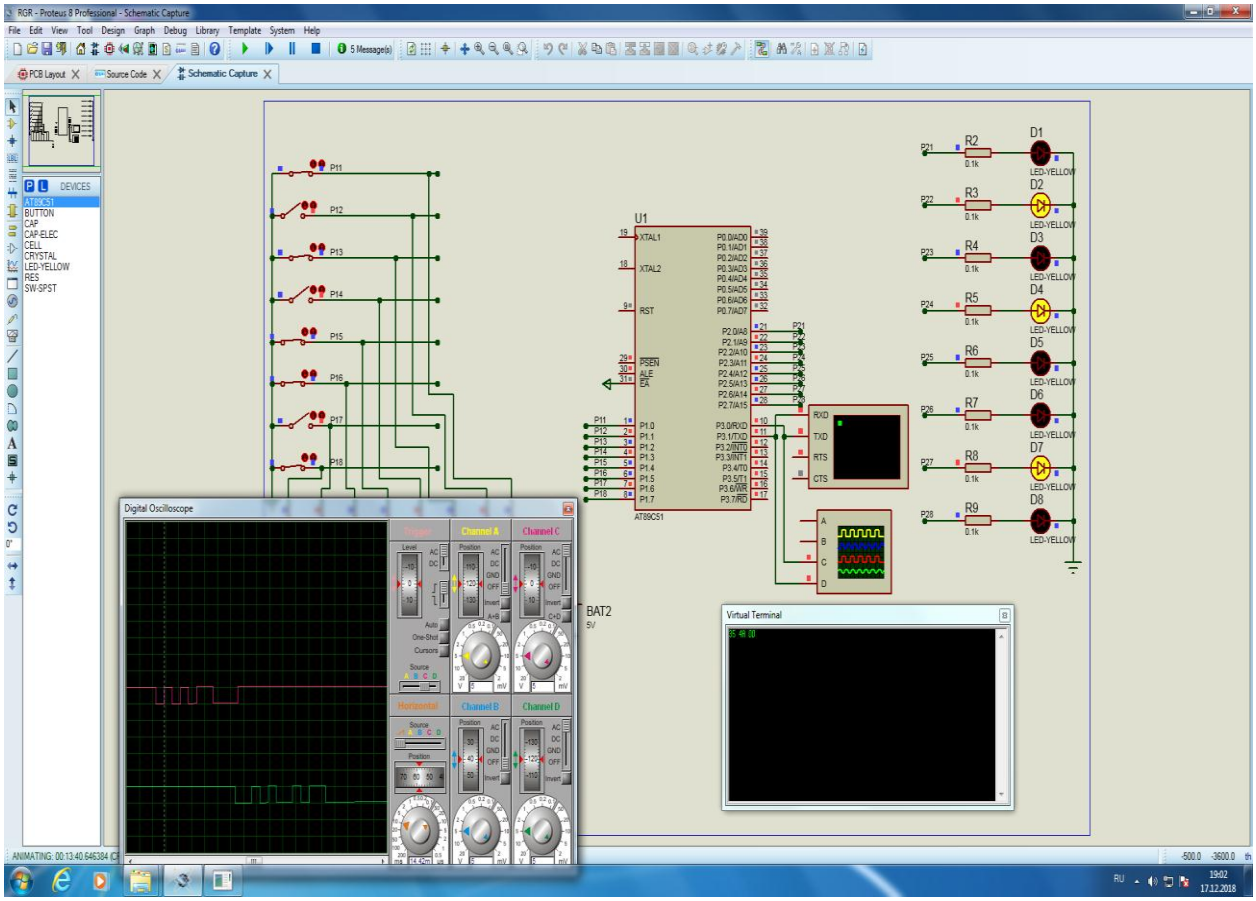


Рисунок 4.18

Подивіться на екран осцилографа (лінія червоного кольору) та побачте передачу через послідовний порт віртуального терміналу (лінія TxD) у старт-стопному режимі символу 35h=00110101b. Зверніть увагу, що передача ведеться починаючи з молодшого розряду: спочатку іде нульовий старт-біт, далі передаються: 1, 0, 1, 0, 1, 1 та 0, 0. Оскільки згідно з рисунком 4.11 виконується перевірка на непарність, то далі передано нульовий контрольний біт. Закінчується передача від терміналу одиничним стоп-бітом.

4.6.12 Визначте отриману швидкість передачі. Для знаходження швидкості передачі (в даному прикладі було обрано швидкість 110 біт/сек = =110 пос/сек = 110 бод) підберіть таку розгортку сигналу на осцилографі, щоб сумістити бічні сторони імпульсу із вертикальними лініями сітки осцилографа.

Для швидкості передачі 110 біт/сек отримано наступну картину (рисунок 4.19).

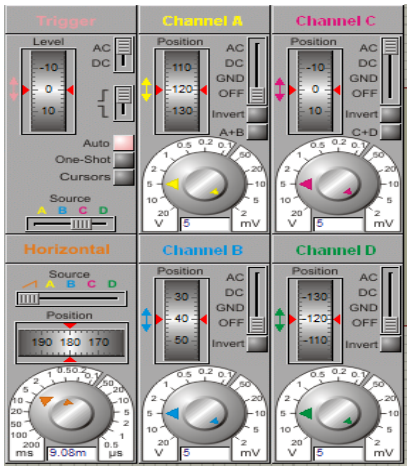


Рисунок 4.19

Тривалість одного імпульсу дорівнює 9,08 мс. Якщо розділити один біт на цей час, та помножити на 1000, буде отримано задану швидкість:

$$V = \frac{1}{9{,}08} * 1000 = 110 \ \frac{\text{біт}}{\text{сек}}.$$

Після завершення виконання цієї частини програми у вікні віртуального терміналу буде відображено символи, а саме їхні hex–коди, один з яких був прийнятий від терміналу до мікроконтролера, а другий передавався від мікроконтролера через УАПП (рисунки 4.17, 4.18).

4.6.13 Для виконання останньої частини програми знов натисніть Run Simulation. Програма дійде до кінця. При цьому на осцилографі ми побачимо байт, який було передано через УАПП: 00000000b або 11111111b. Це залежить від значення біта RB8, яке було отримано після прийому відповідного ASCII–символу (див. 4.6.8) та опції Parity (рисунок 4.11). Якщо в цій опції було вказано ODD то біт RB8 встановиться в 1, якщо число одиниць в ASCII–коді–непарне. В іншому випадку RB8 скидається в нуль. Якщо в опції Parity було вказано EVEN, то біт RB8 встановиться в 1, якщо число одиниць в ASCII–символі–парне. В іншому випадку RB8 скидається в нуль.

Згідно з алгоритмом роботи (рисунок 4.22) та робочою програмою, якщо RB8=1, то на осцилографі ми побачимо байт, який було передано через УАПП: 11111111b. Якщо RB8=0, то на осцилографі ми побачимо байт, який було передано через УАПП: 000000000b. В прикладі, який розглядається, ми вводили символ 7 (ASCII–код: 00110111b=37h). Оскільки в опції Parity було вказано ODD та число одиниць в ASCII–коді–непарне, то після закінчення прийому біт RB8 встановиться в 1. Після закінчення виконання програми через УАПП було передано: 11111111b=FFh (рисунок 4.20).

Після завершення виконання цієї частини програми у вікні віртуального терміналу буде відображено три символи (рисунок 4.20), а саме їх hex–коди: 37h, який був прийнятий мікроконтролером від терміналу, 4A та FFh, які передавалися від мікроконтролера до терміналу через УАПП.

Якщо подивитись на вікно осцилографа (рисунок 4.20, зелений колір), то після передачі від МК–ра до віртуального терміналу символу 4A, перед стоп–бітом стоїть одиничний біт, тому що біт TB8=1 та доповнює символ 4A до парного числа одиниць.

Якщо при передачі символу від МК–ра до терміналу змінити комбінацію, наприклад, на CAh, то перед стоп–бітом буде передано логічний нуль, тому що число одиниць у символі CAh парне (рисунок 4.21).

Якщо від терміналу було передано цифру 6 (ASCII–код: 00110110b=36h), то оскільки в опції Parity було вказано ODD та число одиниць в ASCII–коді–парне, то після закінчення прийому біт RB8 скинеться в нуль. Після закінчення виконання програми через УАПП було передано: 00000000b=00h (рисунок 4.21).

Подивіться на екран осцилографа (лінія червоного кольору) та побачте передачу через послідовний порт віртуального терміналу (лінія TxD) у старт-стопному режимі символу 36h=00110110b. Зверніть увагу, що передача ведеться починаючи з молодшого розряду: спочатку іде нульовий старт-біт, далі передаються: 0, 1, 1, 0, 1, 1 та 0, 0. Оскільки згідно з рисунком 4.11

виконується перевірка на непарність, то далі передано нульовий контрольний біт. Закінчується передача від терміналу одиничним стоп-бітом.

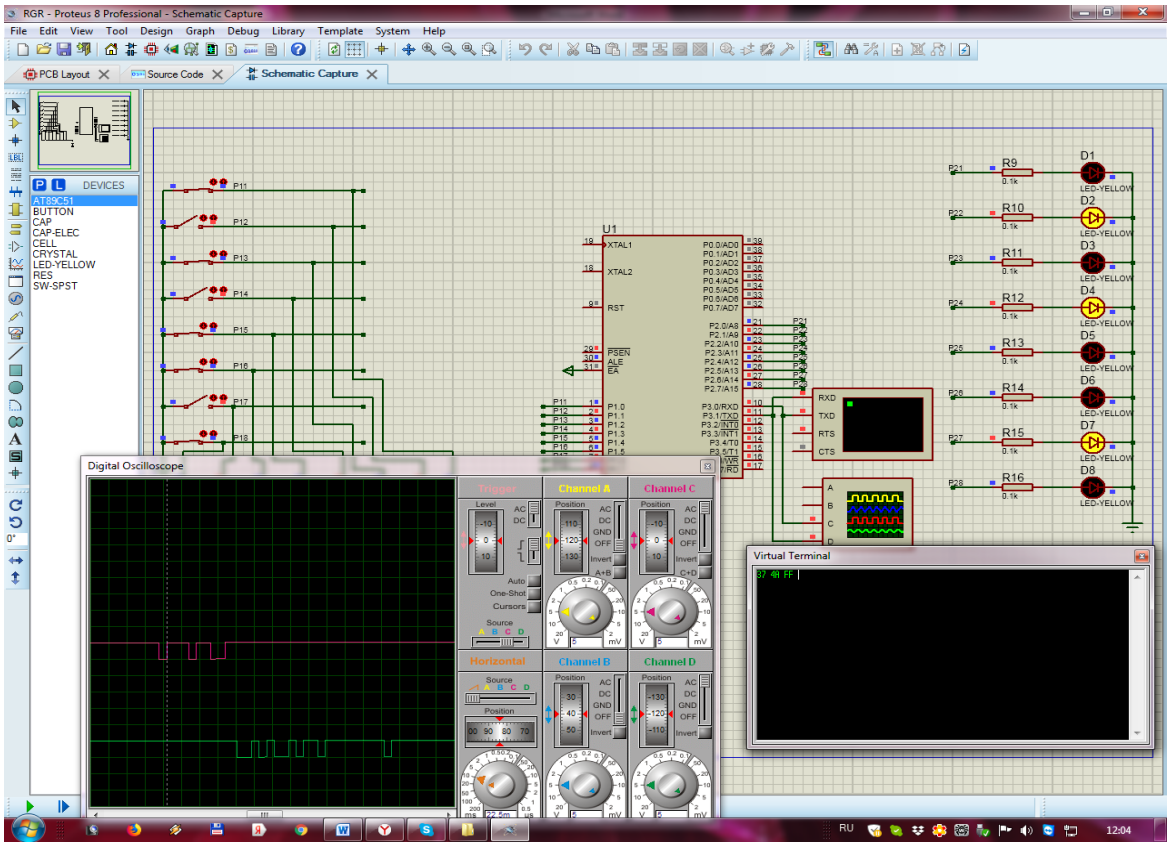


Рисунок 4.20

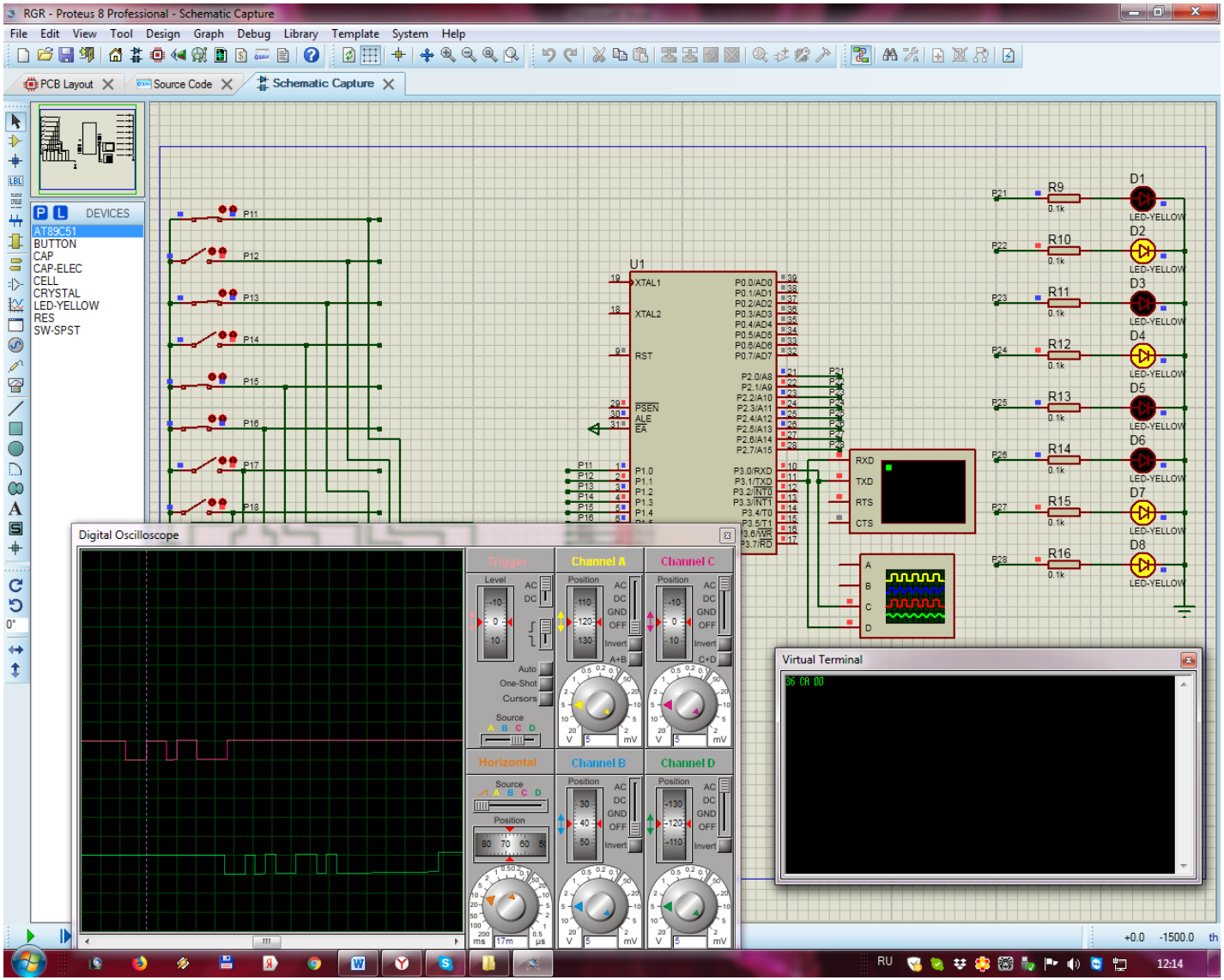

RGR - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
PCB Layout
Source Code
Schematic Capture
DEVICES
AT89C51
BUTTON
CAP
CAP-ELEC
CELL
CRYSTAL
LED-YELLOW
RES
SW-SPST
U1
XTAL1
XTAL2
RST
PSEN
ALE
EA
P0.0/AD0
P0.1/AD1
P0.2/AD2
P0.3/AD3
P0.4/AD4
P0.5/AD5
P0.6/AD6
P0.7/AD7
P2.0/A8
P2.1/A9
P2.2/A10
P2.3/A11
P2.4/A12
P2.5/A13
P2.6/A14
P2.7/A15
P3.0/RXD
P3.1/TXD
P3.2/INT0
P3.3/INT1
P3.4/T0
P3.5/T1
P3.6/WR
P3.7/RD
P1.0
P1.1
P1.2
P1.3
P1.4
RXD
TXD
RTS
CTS
R9
R10
R11
R12
R13
R14
R15
R16
0.1k
D1
D2
D3
D4
D5
D6
D7
D8
LED-YELLOW
Digital Oscilloscope
Trigger
Channel A
Channel B
Channel C
Channel D
Horizontal
Virtual Terminal
36 CR 00
+0.0
-1500.0
RU
12:14

Рисунок 4.21

4.7 Схема алгоритму роботи моделі

На рисунку 4.22 наведено схему алгоритму роботи моделі.

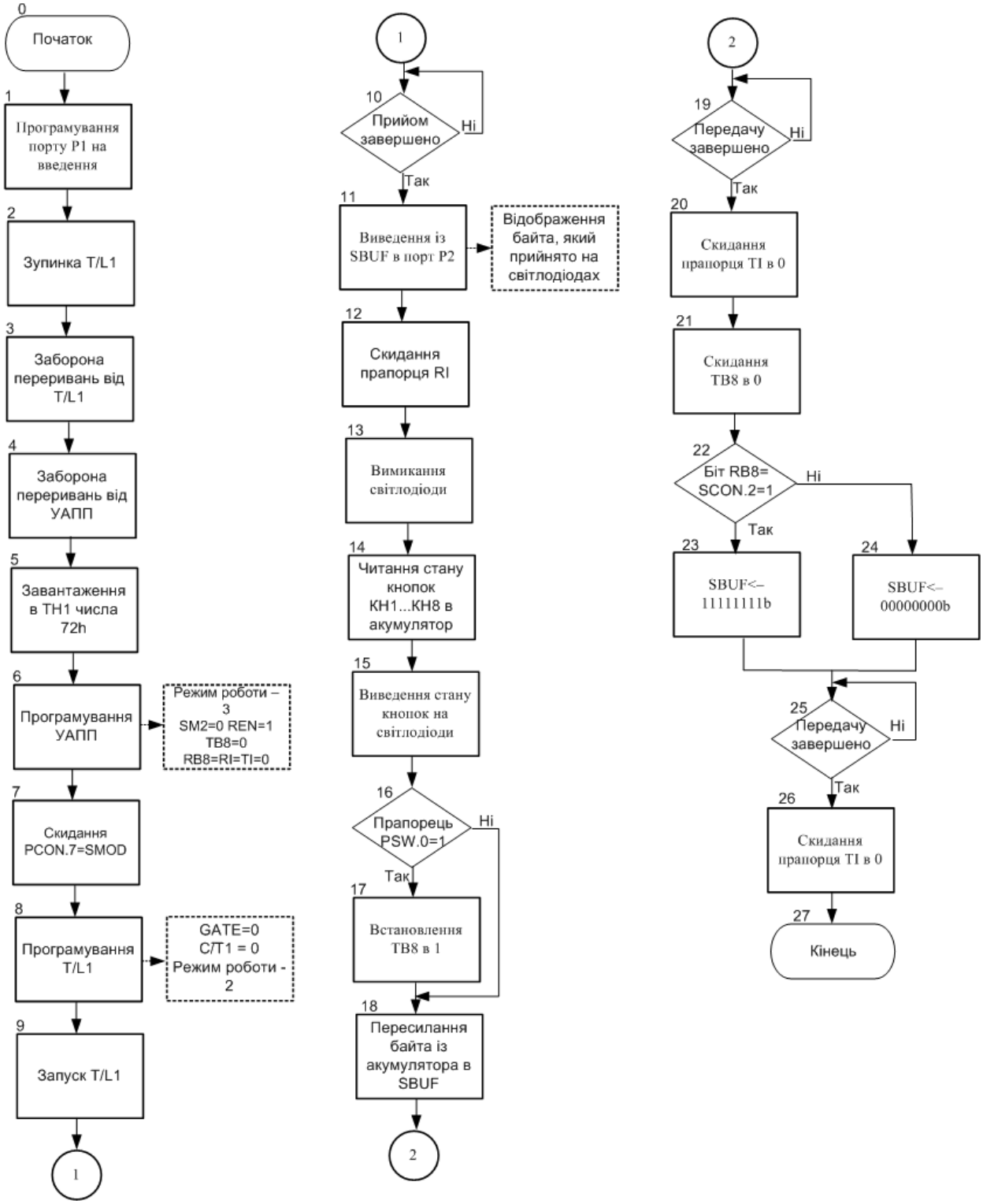


Рисунок 4.22–Схема алгоритму роботи моделі

4.8 Робоча програма мовою Асемблер

Нижче наведено лістинг робочої програми, яку розроблено згідно з алгоритмом, який наведено на рисунку 4.22.

```
; Processor: AT89C51
; Compiler:  ASEM-51 (Proteus)
;=======================================================
$NOMOD51
$INCLUDE (8051.MCU)
;=======================================================
; DEFINITIONS
;=======================================================
;=======================================================;
VARIABLES
;=======================================================
;=======================================================
; RESET and INTERRUPT VECTORS
;=======================================================
; Reset Vector
org   0000h
jmp   Start
;=======================================================
; CODE SEGMENT
;=======================================================
org   0100h
Start:              ; (блок 0)
; Write your code here
;=========================
Loop:
MOV P1, #11111111b  ; Програмування порту P1 на введення (блок 1)
CLR TCON.6          ; зупинка таймера 1 (блок 2);
CLR IE.3            ; заборона переривань від Т/Л1 (блок 3);
CLR IE.4            ; заборона переривань від УАПП (блок 4);
MOV TH1,#72h        ; значення, що автоматично завантажується в
```

```
                    ; TH1 для отримання швидкості 110 бод
                    ;(блок 5)
MOV SCON,#11010000b ; програмування УАПП (блок 6)
MOV A, PCON         ; A <- PCON (блок 7)
ANL A, #01111111b   ; A <- A & 01111111b (блок 7)
MOV PCON, A         ; PCON <- A (блок 7)
MOV TMOD,#00100000b ; програмування      таймера 1 (блок 8)
SETB TCON.6         ; запуск таймера 1 (блок 9)
```

; прийом символу від зовнішнього пристрою

```
;=============================================
CIN: JNB  RI,CIN    ; очікування завершення прийому (блок 10)
MOV P2,SBUF         ; передача отриманого символу в порт P2
                    ;(блок 11)
CLR RI              ; скидання прапорця RI – "прийом завершено"
                    ;(блок 12)
```

; передача символу на зовнішній пристрій

```
;=============================================
MOV P2, #00h        ; вимикаємо світлодіоди (блок 13)
MOV A, P1           ; читаємо стан кнопок КН1...КН8 в акумулятор
                    ;(блок 14)
MOV P2, A           ; стан кнопок виводимо на світлодіоди
                    ;(блок 15)
JB PSW.0, TransmitedNumberIsOdd ;  якщо біт PSW.0=P=1, то
перехід на мітку TransmitedNumberIsOdd (блок 16)
JNB PSW.0, TransmiteNumberIsEven ; якщо біт PSW.0=P=0,
          ; то перехід на мітку TransmiteNumberIsEven (блок 16)
TransmitedNumberIsOdd:
SETB SCON.3        ; встановлення біта TB8 в одиницю (блок 17)
jmp Transmit        ; безумовний перехід на мітку Transmit
                    ;(блок 18);
TransmiteNumberIsEven:
jmp Transmit        ; безумовний перехід на мітку Transmit
                    ;(блок 18)
Transmit:
MOV SBUF, A         ; передача символу в регістр SBUF
```

```
                    ;(початок передачі) (блок 18)
COUT: JNB TI, COUT  ; очікування закінчення передачі
; попереднього символу (блок 19)
CLR TI              ; скидання прапорця TI – "передачу завершено"
                    ; (блок 20)
CLR SCON.3          ; очищення біта TB8 (блок 21)
JB SCON.2, RecievedNumberIsOdd ;  якщо біт SCON.2=RB8=1,
                    ; то перехід на мітку RecievedNumberIsOdd
                    ; (блок 22)
JNB SCON.2, RecievedNumberIsEven ;  якщо біт SCON.2=RB8=0, то
                         ; перехід на мітку  RecievedNumberIsEven
                         ; (блок 22)
RecievedNumberIsOdd:
MOV SBUF, #11111111b     ; передача через УАПП символу 11111111b
                         ;(RB8=1), блок 23
Odd: JNB  TI, Odd        ; очікування закінчення передачі
                         ; попереднього символу (блок 25)
CLR TI                   ; скидання прапорця TI – "передачу
                         ; завершено"(блок 26)
jmp Loop                 ; безумовний перехід на мітку Loop
                         ; (блок 27);
RecievedNumberIsEven:
MOV SBUF, #00000000b     ; передача через УАПП символу 00000000b
                         ;(RB8=0), блок 24
Even: JNB  TI, Even      ; очікування закінчення передачі
                         ; попереднього символу (блок 25)
CLR TI                   ; скидання прапорця TI – "передачу
                         ; завершено" (блок 26)
jmp Loop                 ; безумовний перехід на мітку Loop
                         ;(блок 27)
;=============================================================
      END (блок 27)
```

5 МОДЕЛЮВАННЯ ПРИСТРОЮ КЕРУВАННЯ КРОКОВИМ ДВИГУНОМ

5.1 Моделювання уніполярного крокового двигуна

5.1.1 Опис моделі

Крім відмінностей в загальній конструкції, крокові двигуни відрізняються ще й схемою включення обмоток. Є декілька варіантів їх конфігурацій, залежно від якої двигуни поділяються на уніполярні (англ. «Unipolar») і біполярні (англ. «Bipolar»).

Уніполярний двигун, має ввімкнену одну обмотку в кожній фазі, з відведенням від середини кожної обмотки. Це дозволяє змінювати напрямок магнітного поля, створюваного обмоткою, перемиканням її половинок. Як правило, уніполярний двигун має 6 виводів, але середні виводи обмоток можуть бути об’єднані всередині самого двигуна, тому такий двигун може мати й 5 виводів. Таким чином, якщо вам в руки потрапив невідомий двигун з шість чи п’ятьма виводами – це гарантовано уніполярний кроковий двигун.

Робочу модель пристрою керування уніполярним кроковим двигуном показано на рисунку 5.1.

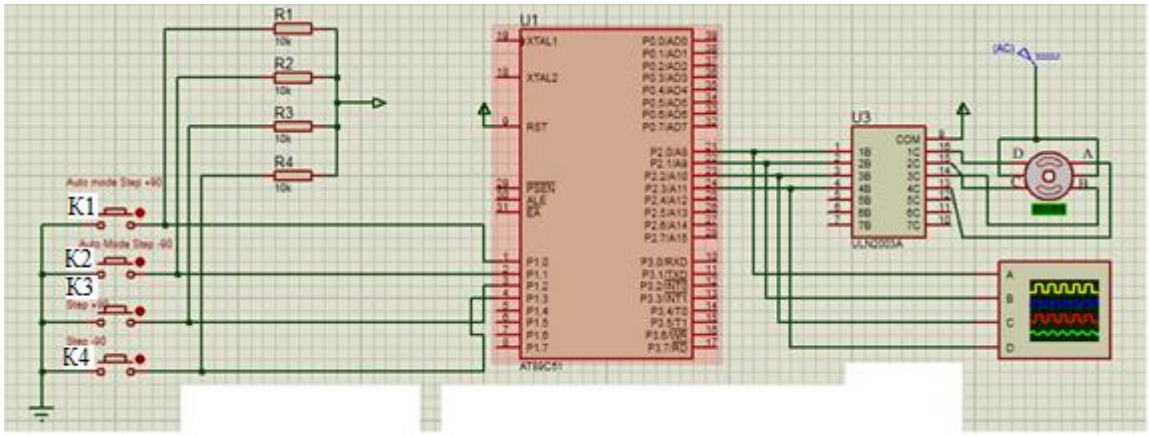


Рисунок 5.1 – Схема моделі пристрою керування уніполярним кроковим двигуном

Розберемо цю модель докладніше. З лівого боку рисунок 5.1 ми бачимо кнопки керування: K1, K2, K3, K4 на які, через резистори R1, R2, R3, R4

подається напруга: +5В. У правому верхньому куті ми бачимо уніполярний кроковий двигун, а трохи лівіше мікросхему ULN2003A, через яку мікроконтролер і керує двигуном. По центру знаходиться сам мікроконтролер сім`ї MCS–51–AT89C51. Зовнішні резистори ми використовуємо через те, що не використовуємо внутрішні резистори, що підтягують, на вхідних лініях мікроконтролера. На рисунку 5.2 наведено позначення виводів мікроконтролера. Нижче більш докладніше пояснюється, що і куди підключаємо в моделі.

Почнемо з кнопок, за допомогою яких ми будемо здійснювати керування уніполярним кроковим двигуном в моделі. Їх збільшений вигляд наведено на рисунку 5.3.

Кнопки є нормально розімкненими та не фіксованими, тобто після зняття прикладеного зусилля вони повертаються у вихідний стан.

Це у нашому випадку означає наступне: у вихідному стані на входи мікроконтролера від кнопок подається високий рівень напруги, тобто логічна 1. А коли кнопки короткочасно натискаються, на входах мікроконтролера спостерігається низький рівень напруги, тобто логічний 0.

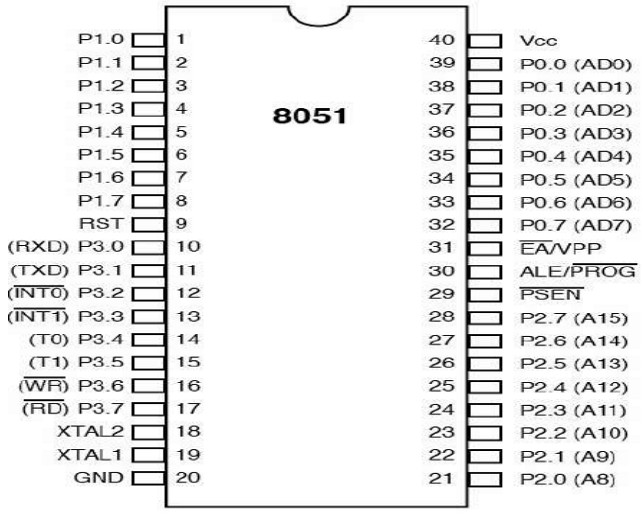


Рисунок 5.2 – Позначення виводів мікроконтролера AT89C51

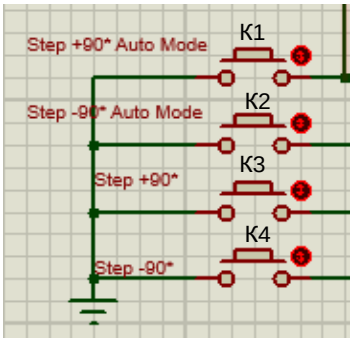


Рисунок 5.3 – Порядок розташування елементів керування

Кнопка K1 (Step +90 Auto Mode) відповідає за вмикання та вимикання двигуна. Тобто коли двигун вимкнено, то вона його ввімкне. Якщо натиснути на кнопку, то наш двигун виконає цикл кроків від 0 до 360 градусів та повернеться в початковий стан: 0 градусів. Кнопку підключено до виводу P1.0 мікроконтролера, який програмується, як вхід.

Кнопка K2 (Step –90 Auto Mode) працює так само як і K1 , але при натисканні, відбуватиметься цикл кроків в зворотному порядку до циклу кроків кнопки K1. Кнопку підключено до виводу P1.1 мікроконтролера, який програмується, як вхід .

Кнопка K3 (Step +90) відповідає за виконання одного кроку двигуна за годинниковою стрілкою. Крок двигуна у кроковому режимі дорівнює 90 градусів. Кнопку підключено до виводу P1.2 мікроконтролера, який програмується, як вхід .

Кнопка K4 (Step –90) відповідає за виконання одного кроку двигуна проти годинникової стрілки. Крок двигуна дорівнює 90 градусів. Кнопку підключено до виводу P1.3 мікроконтролера, який програмується, як вхід.

Тепер розглянемо детальніше підключення мікросхеми ULN2003A до мікроконтролера (рисунок 5.4).

Виводи 21, 22, 23 та 24 мікроконтролера, тобто P2.0, P2.1, P2.2 та P2.3 запрограмовані як виходи і підключені, відповідно, до входів 1B, 2B, 3B та 4B мікросхеми ULN2003A (U2).

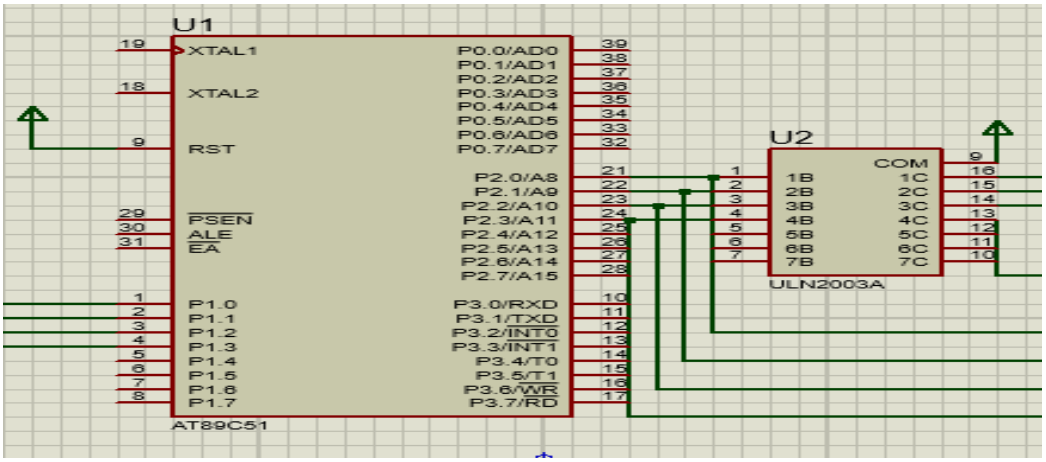


Рисунок 5.4 – Підключення мікросхеми ULN2003A та мікроконтролера

В залежності від керування, на виходах P2.0, P2.1, P2.2 та P2.3 буде або високий рівень напруги, або низький, і в залежності від цього мікросхема буде керувати кроковим двигуном.

Розглянемо використання інших виводів мікросхеми, які зображено на рисунку 5.4. Вихід GND підключено до землі. На вхід COM подається напруга живлення самої мікросхеми ULN2003A: +5В.

На вхід (АС) подається напруга живлення уніполярного крокового двигуна: +12В.

Після цього нам залишилось ознайомитись з підключенням крокового двигуна та мікросхеми ULN2003A. Це підключення зображено на рисунку 5.5.

Як ми бачимо з рисунку 5.5, наш уніполярний кроковий двигун підключено до 13, 14, 15 та 16 виходів мікросхеми ULN2003A, а саме 4С, 3С, 2С та 1С.

На ці виходи за допомогою мікросхеми ULN2003A подаються керуючі сигнали, що і повертають наш кроковий двигун в положення, яке визначається заданим кроком.

Деякі фрагменти, які демонструють роботу системи, наведено на рисунках 5.6…5.8.

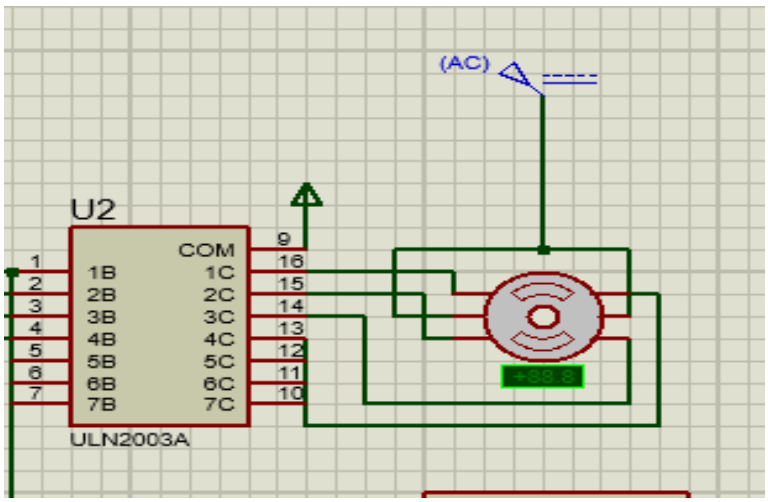


Рисунок 5.5 – Підключення крокового двигуна до мікросхеми ULN2003A

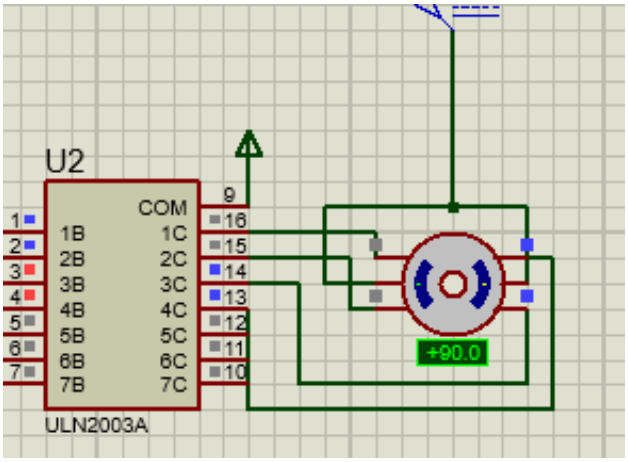


Рисунок 5.6 – Стан моделі після натискання кнопки «Step +90»

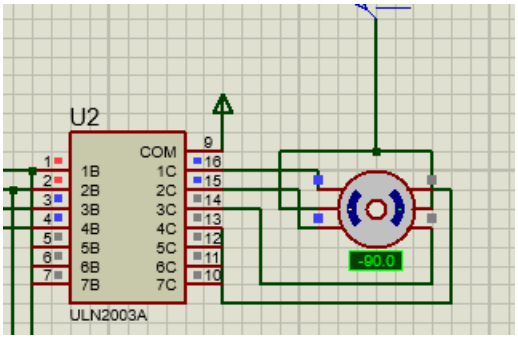


Рисунок 5.7 – Стан моделі після активації режиму реверсу, відповідна кнопка «Step –90»

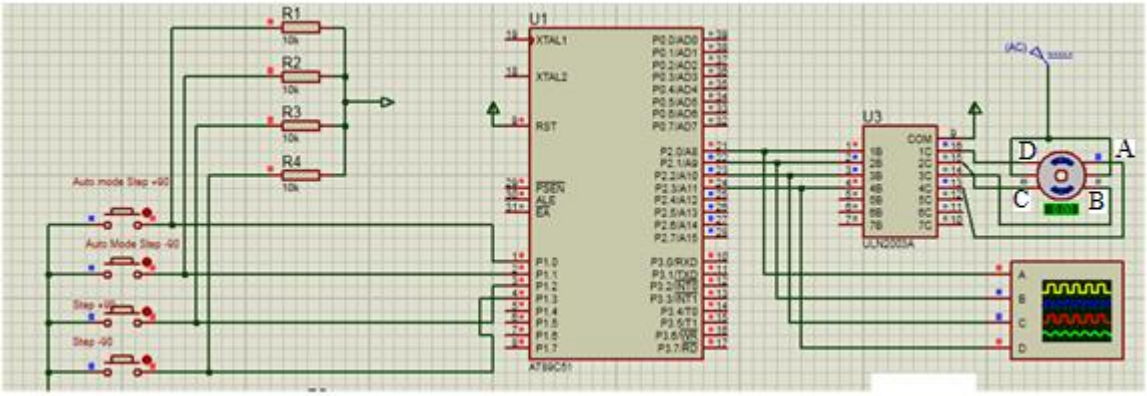


Рисунок 5.8 – Стан усієї системи після запуску сценарію

5.1.2 Мікроконтролер

Для контролю та виконання всіх функцій, які були покладені на систему, обрано мікроконтролер AT89C51 сім`ї MCS51. AT89C51 – малопотужний 8 – розрядний КМОН – мікроконтролер. Мікроконтролер має 32 виводи типу вхід – вихід. В AT89C51 вмонтовано FLASH – пам’ять, яку є можливість програмувати та перепрограмувати.

Характеристики контролера:

Ядро:

– 8–бітовий АЛП, 8–бітові регістри;
– побітова адресація частини ОЗП;
– система команд із 111 інструкцій;
– архітектура системи команд: акумулятор.

Пам’ять:

– гарвардська архітектура пам’яті;
– 8–бітова шина даних;
– 16–бітна шина адреси. Можливість адресації до 64 Кб пам’яті програм і до 64 Кб пам’яті даних;
– 4096 байт вбудованої пам’яті програм (додаткові 60К досягаються за рахунок зовнішніх мікросхем пам’яті);

– 128 байт вбудованої пам’яті даних (додаткові 64К досягаються за рахунок зовнішніх мікросхем пам’яті).

Периферія:

– 32 двосторонні бітові лінії введення/виведення;
– двосторонній послідовний УАПП;
– два 16–бітні таймери/лічильники;
– система з 5 переривань з 2 рівнями пріоритетів;
– вбудований тактовий генератор
– енергозберігаючий режим (тільки у версіях КМОН–технологій).

5.1.3 Уніполярний кроковий двигун

Уніполярні КД (Unipolar Stepper Motors) аналогічні біполярним, але мають відвід від середини кожної з обмоток Це дозволяє змінювати напрямок магнітного поля простими ключовими каскадами, а не мостовими схемами У уніполярному КД середні виводи обмоток можуть електрично зєднуватися всередині корпусу, тому в такому двигуні назовні виходять не 6, а 5 відводів Іноді буває й 8 відводів, якщо половина кожної обмотки зроблено окремою секцією. В останньому випадку уніполярний КД легко перетворити на біполярний

5.1.4 Драйвер уніполярного крокового двигуна ULN2003A

В якості драйвера двигуна обрано мікросхему ULN2003A.

Мікросхема ULN2003A (рисунок 5.9) – це транзисторна схема Дарлінгтона з вихідними ключами підвищеної потужності з відкритим колектором, що має на виходах діоди, які захищають, котрі потрібні для захисту керуючих електричних ланцюгів від оберненого викиду напруги від індуктивного навантаження.

Кожен канал в ULN2003A розрахований на навантаження 500мА і витримує максимальний струм до 600мА. Входи і виходи розміщені в

корпусі мікросхеми один напроти другого, що сильно облегшує розводку друкованої плати. ULN2003A відносяться до сім`ї мікросхем ULN200X.

Рисунок 5.9 –Зовнішній вигляд мікросхеми ULN2003A

Різні версії цієї мікросхеми призначені для певної логіки, а саме мікросхема ULN2003A пристосована до роботи з ТТЛШ та КМОН–логікою (5В). Мікросхема може бути застосована для керування навантаженням значної потужності, включаючи електромагнітні реле, двигуни постійного струму, електромагнітні клапани, в схемах керування різними кроковими двигунами і т. ін. На рисунку 5.10 наведено схему підключення драйвера до крокового двигуна.

Керування кроковими двигунами можна забезпечити різними способами, але найчастіше використовуються такі крокові послідовності:

– хвильова;
– крокова;
– напівкрокова.

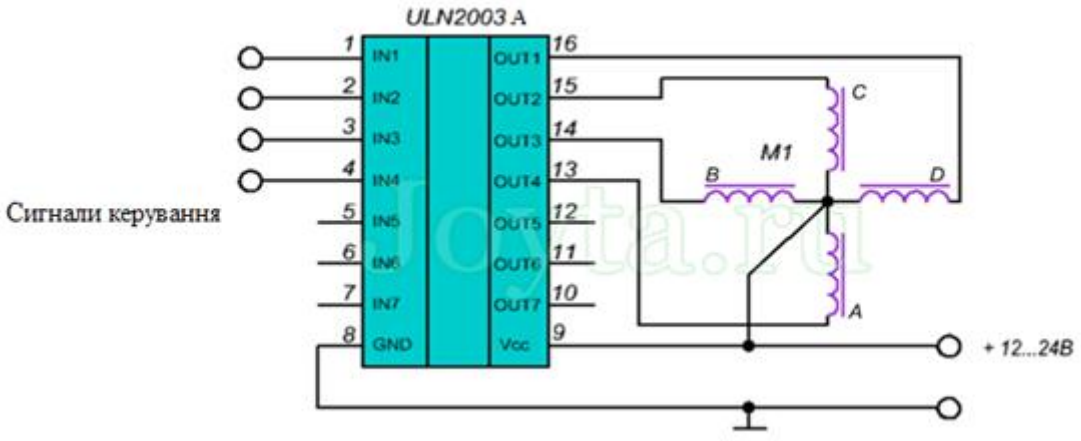


Рисунок 5.10 – Схема підключення драйвера до крокового двигуна

Порядок подачі напруги на обмотки КД наведено на рисунку 5.11.

Крок	A	B	C	D
1	ON	OFF	OFF	OFF
2	OFF	ON	OFF	OFF
3	OFF	OFF	ON	OFF
4	OFF	OFF	OFF	ON

а)

Крок	A	B	C	D
1	ON	OFF	OFF	ON
2	ON	ON	OFF	OFF
3	OFF	ON	ON	OFF
4	OFF	OFF	ON	ON

б)

Крок	A	B	C	D
1	ON	OFF	OFF	OFF
2	ON	ON	OFF	OFF
3	OFF	ON	OFF	OFF
4	OFF	ON	ON	OFF
5	OFF	OFF	ON	OFF
6	OFF	OFF	ON	ON
7	OFF	OFF	OFF	ON
8	ON	OFF	OFF	ON

в)

Рисунок 5.11 – Послідовності збудження обмоток КД : а) хвильова, б) крокова, в) напівкрокова

Хвильова послідовність збудження – найпростіший спосіб керування кроковими двигунами: обмотки збуджуються послідовно одна за одною. При цьому двигун починає обертатися в сторону, протилежну порядку збудження обмоток. Оскільки в свій час збуджується тільки одна обмотка, то момент двигуна, що обертає, є невеликим. Для його збільшення використовується крокова послідовність.

Крокова послідовність – аналогічна попередній, але тут одночасно збуджуються дві обмотки, завдяки чому збільшується обертаючий момент двигуна.

Напівкрокова послідовність збудження – це комбінація перших двох. Під час одного обороту ротора кількість циклів збудження подвоюється. При цьому режимі в 2 рази зменшується величина кроку і двигун працює рівніше.

Нижче наведено схему алгоритму роботи моделі (рисунки 5.1, 5.8) уніполярного крокового двигуна. На цих рисунках обмотки двигуна позначено: A, B, C та D.

Відповідно схемі алгоритму (рисунок 5.12) розроблено керуючу програму мовою С, яку наведено нижче у 5.1.6.

В цих алгоритмі та програмі реалізовано крокову послідовність збудження обмоток.

Уніполярний кроковий двигун, який використовується в моделі, за один крок у кроковому режимі обертається на кут у 90 градусів. Згідно схемі моделювання, яку наведено на рисунку 5.1, керуюча послідовність імпульсів передається через лінії мікроконтролера: P2.0 (обмотка D), P2.1 (обмотка C), P2.2 (обмотка B) та P2.3 (обмотка A). Ця послідовність у чотирьохрозрядному двійковому коді для крокового режиму (рисунок 5.11) виглядає наступним чином: 1001 для кута 0 градусів; 0011 для кута 90 градусів; 0110 для кута 180 градусів, 1100 для кута 270 градусів та 1001 для повернення у початковий стан (кут дорівнює 0 градусів). З виходів 1С, 2С, 3С та 4С драйвера ULN2003A імпульси керування подаються безпосередньо на обмотки двигуна, які на рисунку 5.10 позначено як D, C, B та A.

Якщо перепрограмувати роботу моделі у напівкроковий режим згідно з рисунком 5.11, в, то за один крок двигун буде обертатися на кут у 45 градусів.

4.1.5 Схема алгоритму роботи моделі уніполярного крокового двигуна для крокового режиму роботи

Схему алгоритму роботи моделі уніполярного крокового двигуна для крокового режиму роботи наведено на рисунку 5.12.

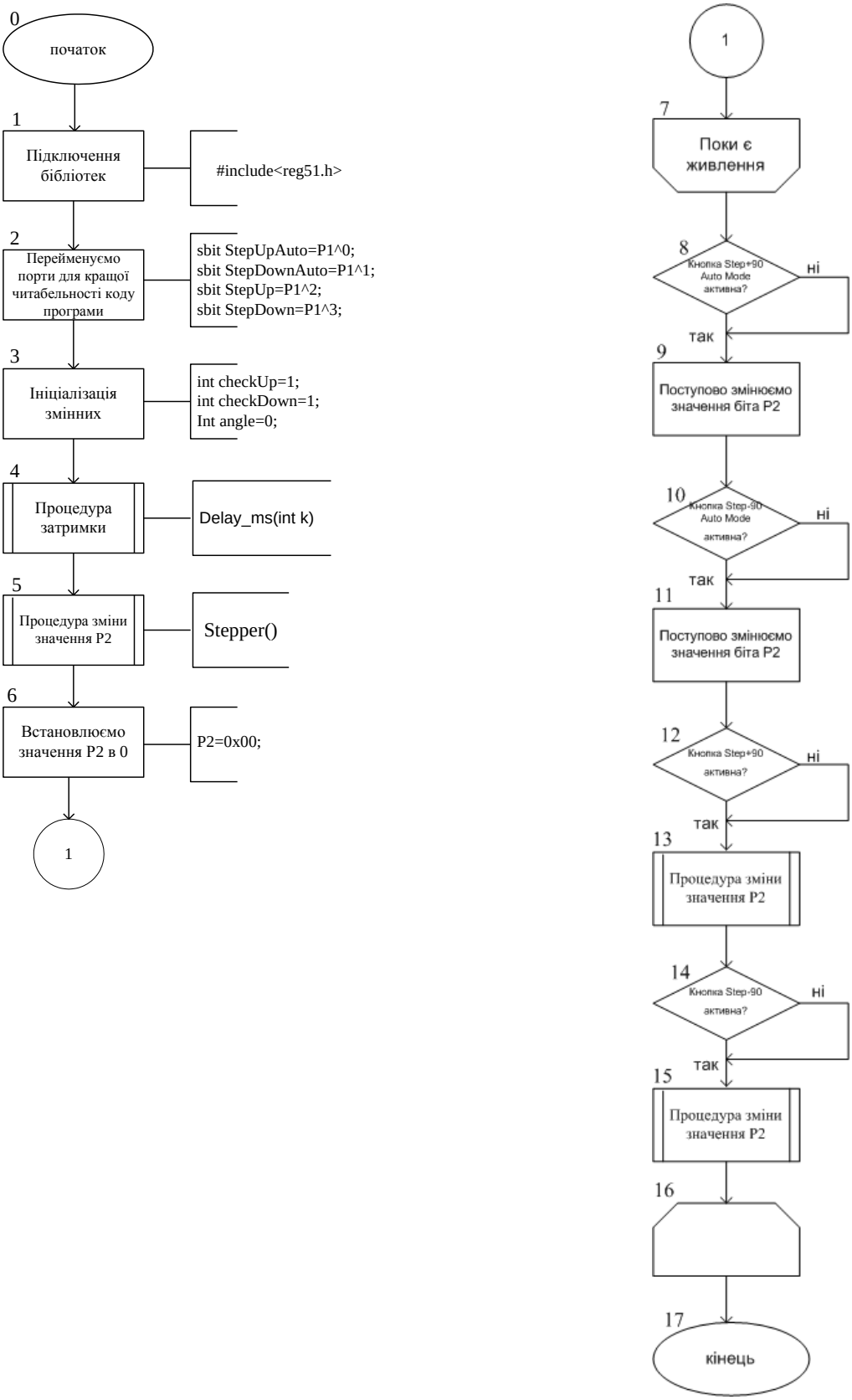
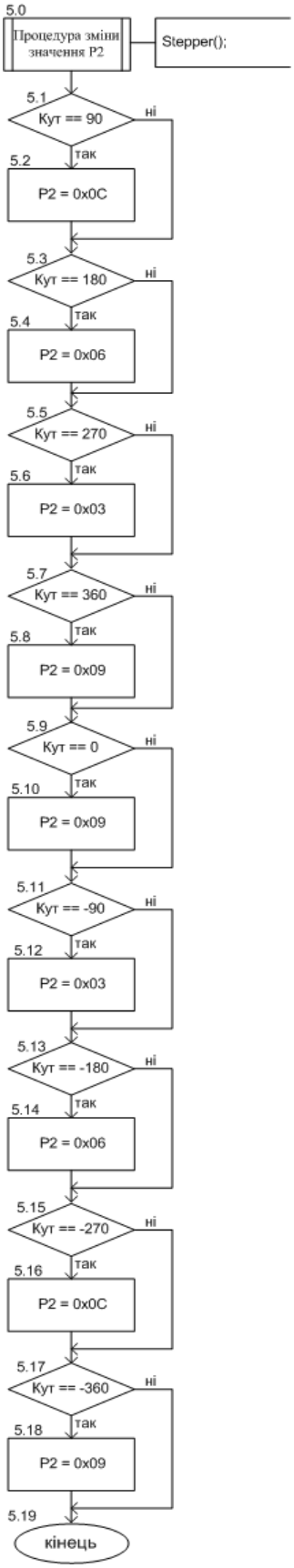


Рисунок 5.12 – Схема алгоритму роботи моделі уніполярного крокового двигуна для крокового режиму роботи



Продовження рисунку 5.12

5.1.6 Керуюча програма моделі уніполярного двигуна для крокового режиму роботи

```
#include <reg51.h>  //підключення бібліотеки reg51.h

sbit StepUpAuto = P1^0;  //2       перейменування портів
sbit StepDownAuto = P1^1;
sbit StepUp = P1^2;
sbit StepDown = P1^3;
int checkUp=1;           //3  ініціалізація змінних
int checkDown=1;
int angle =0;

void Stepper();          //об'ява процедури змінення P2 від кута
void delay_ms( unsigned int k ) //процедура «затримки»
{
unsigned int i,j;
for (i=0;i<=k;i++){ //цикл від параметру (наприклад 1000)
     for(j=0;j<110;j++){}}
}

void main(void)          //основна програма
{
     P2=0x00;       //4

     while(1){      //5 цикл поки є живлення

if (!StepUpAuto) {       //6 кнопка Step +90 Auto Mode
     P2=0x0C;       //7 поступово змінюємо значення біта P2
     delay_ms(1000);
     P2=0x06;
     delay_ms(1000);
     P2=0x03;
     delay_ms(1000);
     P2=0x09;
```

```
     delay_ms(1000);
}

     if (!StepDownAuto) {     //8 кнопка Step -90 Auto Mode
     P2=0x03;            //9 поступово змінюємо значення біта P2
     delay_ms(1000);
     P2=0x06;
     delay_ms(1000);
     P2=0x0C;
     delay_ms(1000);
     P2=0x09;
     delay_ms(1000);
     }

     if (!StepUp && checkUp==1)         //10 кнопка Step +90
     {
       if(angle==360)    angle=0;  //11 якщо кут == 360, то кут
= 0
                              //
          angle+=90;
          checkUp=0;
     }

     if (StepUp)
          checkUp=1;

     if(!StepDown && checkDown==1)      // 12     кнопка Step -90
     {
if(angle==-360)     angle=0;
          angle-=90;
          checkDown=0;
     }

     if (StepDown)
          checkDown=1;
```

```
     Stepper();                    //13 зміна P2

     }                   //кінець циклу 2  «поки є»живлення
}         //кінець основної програми main

     void Stepper(){     //процедура зміни P2 від значення кута
     if(angle==90)
          P2=0x0C;
     if(angle==180)
          P2=0x06;
     if(angle==270)
          P2=0x03;
     if(angle==360)
          P2=0x09;
     if (angle==0)
          P2=0x09;
     if(angle==-90)
          P2=0x03;
     if(angle==-180)
          P2=0x06;
     if(angle==-270)
          P2=0x0C;
     if(angle==-360)
          P2=0x09;
}
```

На рисунку 5.13 наведено керуючі послідовності, які відображає осцилограф після натискання кнопки «Step +90 Auto Mode», для уніполярного двигуна, який працює у кроковому режимі.

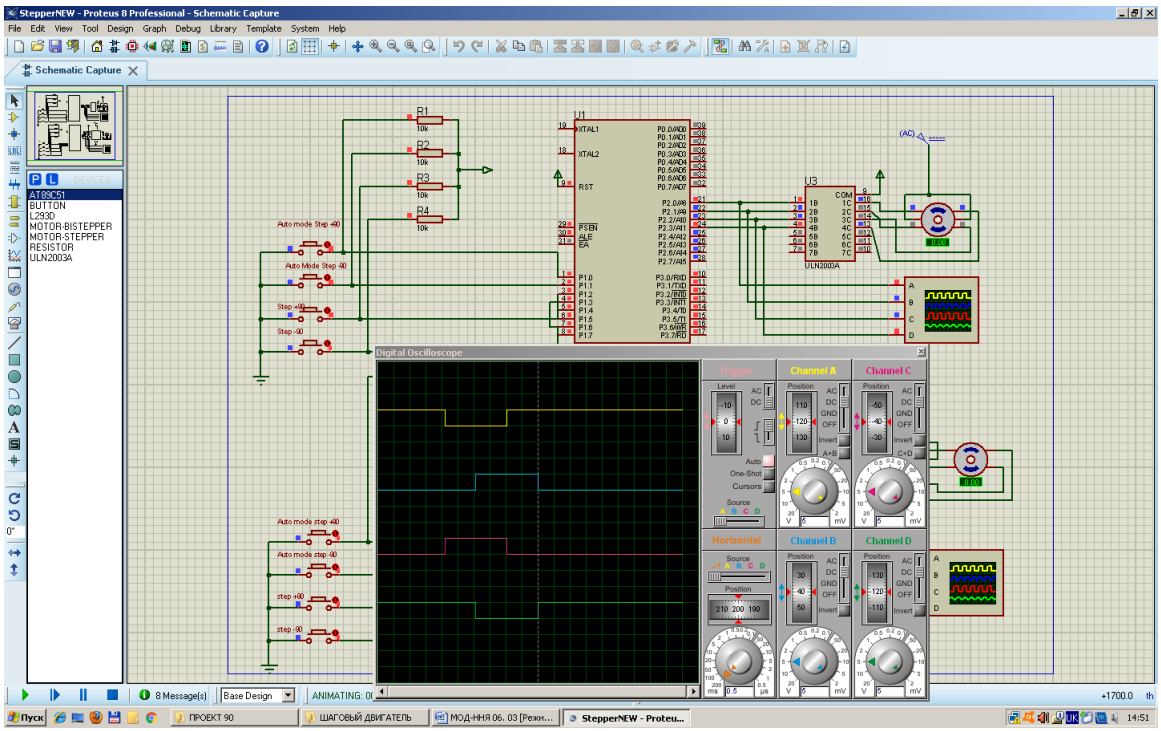


Рисунок 5.13 – Керуючі послідовності для уніполярного двигуна для крокового режиму роботи, які відображає осцилограф після натискання кнопки «Step +90 Auto Mode»

5.1.7 Керуюча програма моделі уніполярного крокового двигуна для напівкрокового режиму

```
#include <reg51.h>  // reg51.h
void Stepper();           //
void delay_ms( unsigned int k ) //
{
unsigned int i,j;
for (i=0;i<=k;i++){ // (затримка на 1000)
     for(j=0;j<110;j++){}}
}
sbit StepUpAuto = P1^0;  //2
sbit StepDownAuto = P1^1;
sbit StepUp = P1^2;
sbit StepDown = P1^3;
int checkUp=1;            //3
int checkDown=1;
```

```
int angle =0;

void main(void)          //

{
     P2=0x00;       //4

     while(1)  {         //5

if (!StepUpAuto) {       //6  Step +45 Auto Mode
     delay_ms(1000);
P2=0x08;
     delay_ms(1000);
P2=0x0C;
     delay_ms(1000);
     P2=0x04;
     delay_ms(1000);
     P2=0x06;
     delay_ms(1000);
     P2=0x02;
     delay_ms(1000);
     P2=0x03;
     delay_ms(1000);
     P2=0x01;
     delay_ms(1000);
     P2=0x09;
     delay_ms(1000);

     }

     if (!StepDownAuto) {     //8 Step -45 Auto Mode
     P2=0x01;
     delay_ms(1000);
     P2=0x03;
     delay_ms(1000);
```

```
      P2=0x02;
      delay_ms(1000);
      P2=0x06;
      delay_ms(1000);
      P2=0x04;
      delay_ms(1000);
      P2=0x0C;
      delay_ms(1000);
      P2=0x08;
      delay_ms(1000);
      P2=0x09;
      delay_ms(1000);

      }

      if (!StepUp && checkUp==1)         //10  Step +45
      {
        if(angle==360)    angle=0;  //11

           angle+=45;
           checkUp=0;
      }

      if (StepUp)
           checkUp=1;

      if(!StepDown && checkDown==1)      // 12     Step -45
      {
if(angle == -360)   angle=0;
      angle = angle - 45;
           checkDown=0;
      }
           if (StepDown)
           checkDown=1;
```

```
      Stepper();                    //13

      }
}          // main

      void Stepper()
           {
      if(angle==0)
           P2=0x09;
      if(angle==45)
           P2=0x08;
      if(angle==90)
           P2=0x0C;
      if(angle==135)
           P2=0x04;
      if(angle==180)
           P2=0x06;
      if (angle==225)
           P2=0x02;
      if(angle== 270)
           P2=0x03;
      if(angle== 315)
           P2=0x01;
      if(angle==360)
           P2=0x09;
           if(angle==-45)
           P2=0x01;
      if(angle==-90)
           P2=0x03;
      if(angle==-135)
           P2=0x02;
      if(angle==-180)
           P2=0x06;
      if (angle==-225)
           P2=0x04;
```

```
        if(angle== -270)
              P2=0x0C;
        if(angle==- 315)
              P2=0x08;
        if(angle==-360)
              P2=0x09;
}
```

На рисунку 5.14 наведено керуючі послідовності для уніполярного крокового двигуна для напівкрокового режиму, які відображає осцилограф після натискання кнопки «Step +45 Auto Mode».

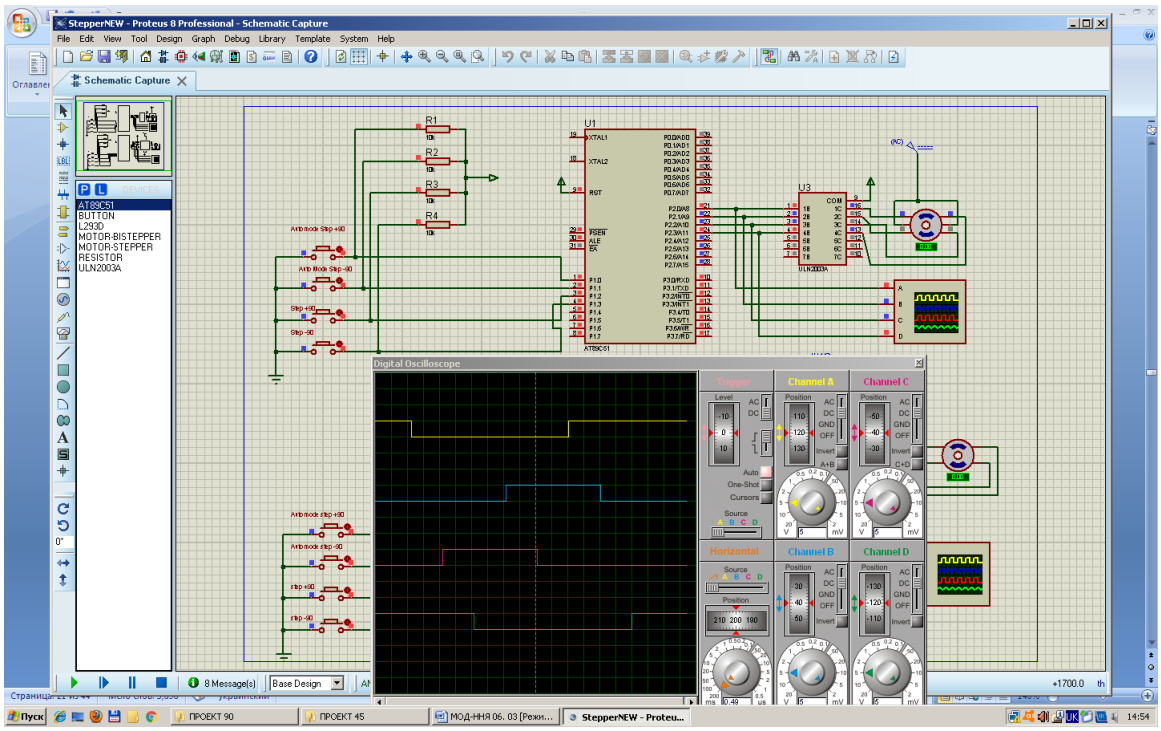


Рисунок 5.14 – Керуючі послідовності для уніполярного крокового двигуна для напівкрокового режиму, які відображає осцилограф після натискання кнопки «Step +45 Auto Mode»

5.2 Моделювання біполярного крокового двигуна

5.2.1 Опис моделі

Робочу модель пристрою керування біполярним крокових двигуном показано на рисунку 5.15.

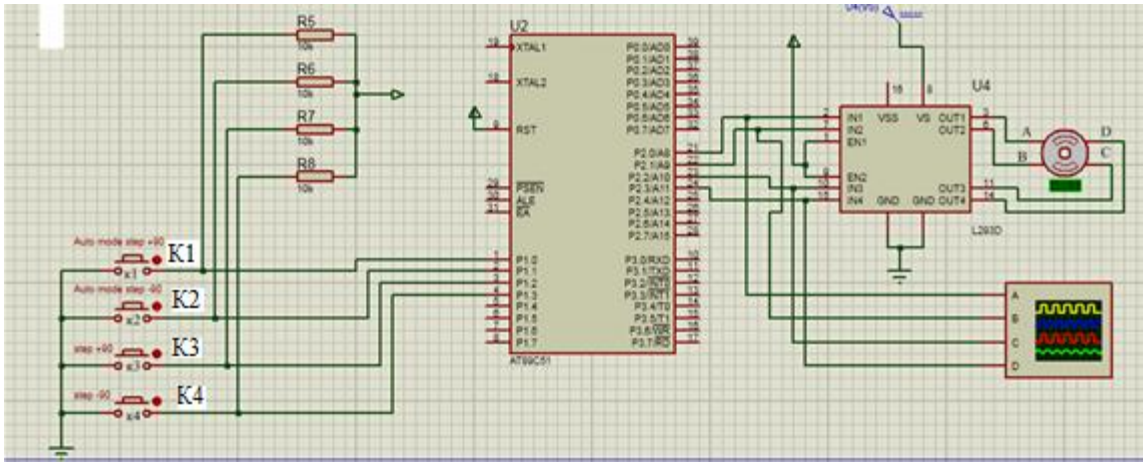


Рисунок 5.15 – Схема моделі пристрою керування біполярним кроковим двигуном

Розберемо цю модель докладніше. З лівого боку рисунка 5.15 ми бачимо кнопки керування: K1, K2, K3, K4 на які через резистори R5, R6, R7, R8 відповідно, подається напруга: +5В. У правому куті ми бачимо біполярний кроковий двигун, а трохи лівіше мікросхему L293D, через яку мікроконтролер і керує двигуном. Сам мікроконтролер AT89C51 знаходиться по центру. Зовнішні резистори ми використовуємо через те, що не використовуємо внутрішні резистори, що підтягують, на вхідних лініях мікроконтролера. Вище на рисунку 5.2 наведено позначення виводів мікроконтролера.

Нижче більш докладніше пояснимо, що і куди підключаємо в моделі. Почнемо з кнопок, за допомогою яких ми будемо здійснювати керування біполярним кроковим двигуном на моделі. Їх збільшений вигляд наведено на рисунку 5.16.

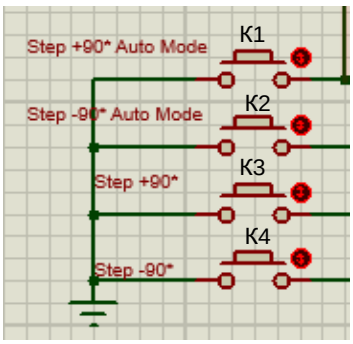


Рисунок 5.16 – Порядок розташування елементів керування

Кнопки є нормально розімкненими та не фіксованими, тобто після зняття прикладеного зусилля вони повертаються у вихідний стан. Це у нашому випадку означає наступне: у вихідному стані на входи мікроконтролера MCS–51 від кнопок подається високий рівень напруги, тобто логічна 1. А коли кнопки натискаються, на входах мікроконтролера спостерігається низький рівень напруги, тобто логічний 0.

Кнопка K1 (Step +90 Auto Mode) відповідає за вмикання та вимикання двигуна. Тобто коли двигун вимкнено, то вона його ввімкне. Якшо натиснути на кнопку, то наш двигун виконає цикл кроків від 0 до 360 градусів та повернеться в початковий стан: 0 градусів. У однофазному режимі з цілим кроком крок двигуна дорівнює 90 градусів. Кнопку підключено до виводу P1.0 мікроконтролера, який програмується, як вхід.

Кнопка K2 (Step –90 Auto Mode) працює так само як і K1 , але при натисканні, відбуватиметься цикл кроків в зворотному порядку до циклу кроків кнопки K1. У однофазному режимі з цілим кроком крок двигуна дорівнює 90 градусів. Кнопку підключено до виводу P1.1 мікроконтролера, який програмується, як вхід .

Кнопка K3 (Step +90) відповідає за виконання одного кроку двигуна за годинниковою стрілкою. У однофазному режимі з цілим кроком крок двигуна дорівнює 90 градусів. Кнопку підключено до виводу P1.2 мікроконтролера, який програмується, як вхід .

Кнопка K4 (Step –90) відповідає за виконання одного кроку двигуна проти годинникової стрілки. У однофазному режимі з цілим кроком крок двигуна дорівнює 90 градусів. Кнопку підключено до виводу P1.3 мікроконтролера, який програмується, як вхід .

Тепер розглянемо детальніше підключення мікросхеми L293D до мікроконтролера (рисунок 5.17).

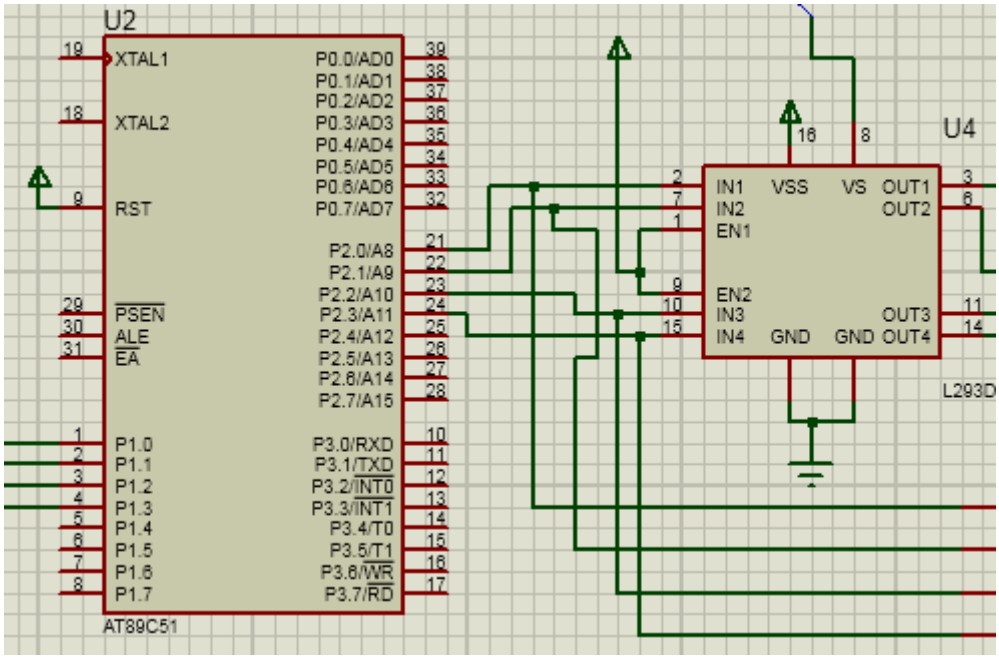


Рисунок 5.17 – Підключення мікросхеми L293D до мікроконтролера

Виводи 21, 22, 23 та 24 мікроконтролера, тобто P2.0, P2.1, P2.2 та P2.3 запрограмовані як виходи і підключені, відповідно, до входів IN1, IN2, IN3 та IN4 мікросхеми L293D (U4).

В залежності від керування, на виходах P2.0, P2.1, P2.2 та P2.3 буде або високий рівень напруги, або низький, і в залежності від цього мікросхема буде керувати кроковим двигуном.

Розглянемо використання інших виводів мікросхеми, які зображено на рисунку 5.17. Виводи GND підключено до землі. Виводи EN1, EN2 підключено до напруги живлення: +5В. На вхід VSS подається напруга живлення самої мікросхеми L293D: 5В. На вхід VS подається напруга живлення біполярного крокового двигуна: 12В.

Після цього нам залишилось ознайомитись з підключенням крокового двигуна до мікросхеми L293D. Це підключення зображено на рисунку 5.18.

Як ми бачимо з рисунок 5.18, наш біполярний кроковий двигун підключено до виходів 3, 6, 11 та 14 мікросхеми L293D, а саме OUT1, OUT2, OUT3 та OUT4.

На ці виходи за допомогою мікросхеми L293D подається керуючі сигнали, що і повертають наш кроковий двигун в положення, яке відповідає заданому кроку.

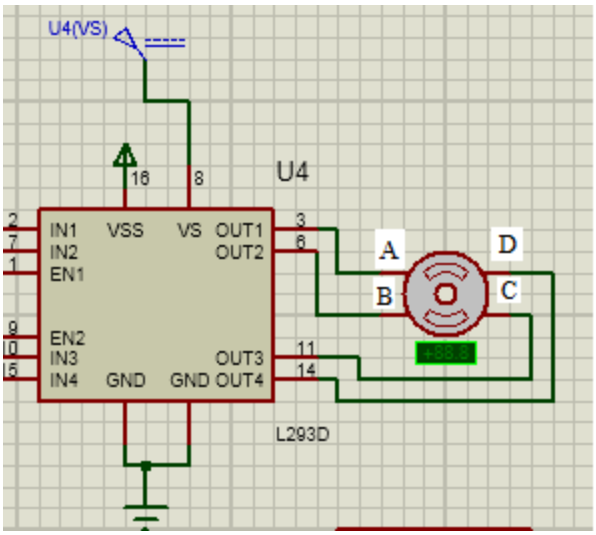


Рисунок 5.18 – Схема підключення драйвера до крокового двигуна

5.2.2 Драйвер біполярного крокового двигуна

В якості драйвера біполярного крокового двигуна обрано мікросхему L293D (рисунок 5.19).

Рисунок 5.19 – Зовнішній вигляд мікросхеми L293D

L293D включає два драйвери для керування електродвигунами відносно невеликої потужності. Мікросхема має дві пари входів для керування напрямом обертання двигунів і дві пари виходів для підключення електродвигунів. Крім того, у L293D є входи для вмикання кожного з драйверів.

Також, L293D забезпечує розділене живлення для мікросхеми і для керованих нею двигунів, що дозволяє підключати електродвигуни з

більшою напругою живлення, ніж у мікросхеми. Розділення живлення мікросхеми і електродвигунів нам також буде необхідним для зменшення завад, які викликано стрибками напруги, що пов’язані з роботою двигунів.

Обидва драйвери, що входять у склад мікросхеми мають ідентичний принцип роботи, тому розглянемо принцип роботи лише одного з них. Схематичний вигляд драйвера мікросхеми L293D наведено на рисунку 5.20.

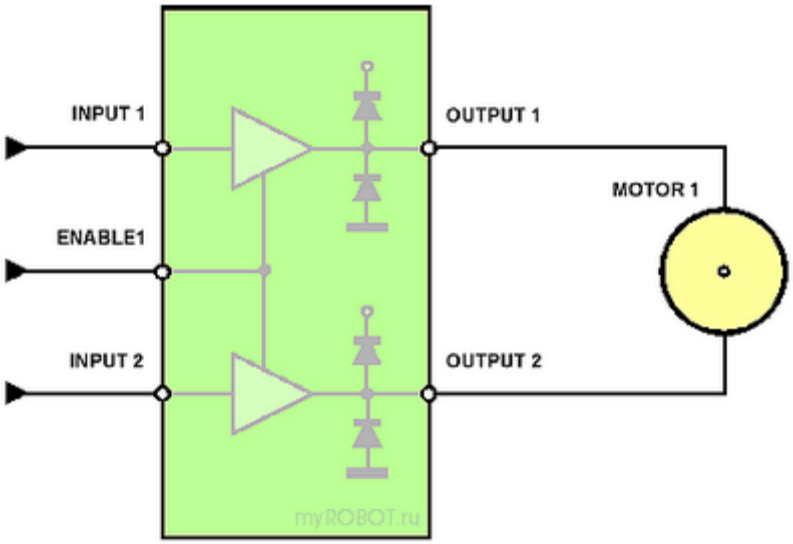


Рисунок 5.20 – Схематичний вигляд драйвера мікросхеми L293D

До виходів OUTPUT1 та OUTPUT2 підключається двигун постійного струму (MOTOR1). На вхід ENABLE1, який відповідає за ввімкнення драйвера, подаємо керуючий сигнал, наприклад, підключаємо його до додатного полюсу джерела живлення: +5V. Якщо при цьому на входи INPUT1 та INPUT2 не подаються відповідні керуючі сигнали, то двигун обертатися не буде.

Якщо вхід INPUT1 з’єднати з додатним полюсом джерела живлення, а вхід INPUT2 – з від`ємним, то двигун почне обертатися за годинниковою стрілкою. Якщо вхід INPUT1 з’єднаємо з від`ємним полюсом джерела живлення, а вхід INPUT2 – з додатним, то двигун почне обертатися в іншу сторону.

Якщо подати сигнали одного рівня одразу на два керуючих входи INPUT1 та INPUT2, тобто підключити обидва входи до додатного полюсу джерела живлення або до від`ємного, то двигун обертатися не буде.

Якщо ми приберемо керуючий сигнал з входу ENABLE1, то при будь яких варіантах сигналів на двох входах INPUT1 та INPUT2 двигун обертатися не буде. В нашій моделі входи ENABLE1 та ENABLE2 підключаються до напруги: +5В. Змінюючи сигнали керування на входах INPUT1 та INPUT2 ми будемо подавати відповідні сигнали керування на обмотки біполярного двигуна.

На рисунку 5.21 наведено нумерацію та позначення виводів мікросхеми.

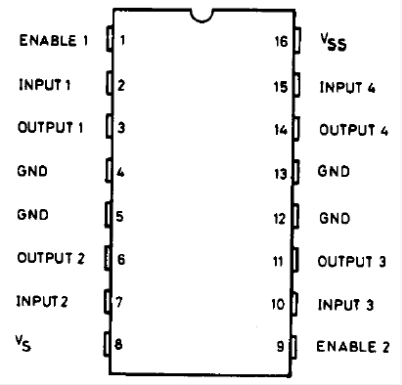


Рисунок 5.21 – Нумерація та позначення виводів мікросхеми L293D

Призначення виводів:

– входи ENABLE1 та ENABLE2 відповідають за вмикання відповідного драйвера, що входить у склад мікросхеми. В нашій моделі ці входи підключаються до напруги: +5В;

– входи INPUT1 та INPUT2 керують формуванням відповідних сигналів керування на обмотці OUTPUT1, OUTPUT2 біполярного двигуна;

– входи INPUT3 та INPUT4 керують формуванням відповідних сигналів керування на обмотці OUTPUT3, OUTPUT4 біполярного двигуна;

– контакт Vs з’єднують з додатним полюсом джерела живлення двигунів або просто з додатним полюсом джерела живлення, якщо схема і

двигуни живляться від одного джерела. Простіше кажучи, цей контакт відповідає за живлення електродвигунів;

– контакт Vss з'єднують з додатним полюсом джерела живлення. Цей контакт забезпечує живлення самої мікросхеми;

– чотири контакти GND з'єднують з "землею", спільним дротом або від`ємним полюсом джерела живлення. Крім того, за допомогою цих контактів зазвичай забезпечують тепловідвід від мікросхеми, тому їх краще всього паяти на достатньо широку контактну поверхню.

Дана мікросхема має наступні основні технічні характеристики:

- напруга живлення двигунів (Vs) – 4,5...36 В;
- напруга живлення мікросхеми (Vss) – 5 В;
- допустимий струм навантаження – 600 мА (на кожен канал);
- піковий (максимальний) струм на виході – 1,2 А (на кожен канал);
- логічний "0" вхідної напруги – до 1,5 В;
- логічна "1" вхідної напруги – 2,3...7 В;
- вбудовано захист від перегріву.

5.2.3 Біполярний кроковий двигуна

Біполярний кроковий двигун (Bipolar Stepper Motors) – це електричний двигун з двома обмотками, які на рисунку 5.15 позначено як A, B та C, D. Подача електричного струму на обмотки двигуна у відповідному напрямку та послідовності приводить до того, що його ротор фіксується в строго певному положенні та обертається на заданий кут. Струм в обмотках повинен переполюсовиватися драйвером мостового типу, наприклад, L293D. Завдяки цьому кут повороту ротора залежить від кількості послідовних перемикань обмоток, а швидкість обертання ротора дорівнює частоті перемикання обмоток, яку помножено на кут повороту ротора за одне перемикання.

Існують три можливі послідовності включення обмоток. Перша послідовність – це включення обмоток у порядку: AB, CD, BA та DC (BA та

DC означає зворотне включення обмоток, коли струм протікає у протилежному напрямку). Ця послідовність зветься як однофазний режим з цілим кроком. У кожен фіксований момент часу включено тільки одну обмотку.

Наступний режим зветься двохфазним з цілим кроком, коли обидві фази включаються одночасно та ротор завжди центрує себе між двома полюсами. Друга послідовність має такий вигляд: AB та CD; BA та CD; BA та DC та AB та DC.

Ще один варіант – це включення фаз у послідовності: AB; AB та CD; CD; BA та CD; BA; BA та DC; DC; AB та DC. Ця послідовність зветься як режим з напівкроком. В цьому режимі біполярний двигун, який використовується в нашій моделі, за один крок буде обертатися на кут у 45 градусів.

Згідно схемі моделювання, яку наведено на рисунку 5.15, керуюча послідовність імпульсів передається через лінії P2.0, P2.1, P2.2 та P2.3 мікроконтролера. Ця послідовність у чотирьохрозрядному двійковому коді виглядає наступним чином: 0001 для кута 0 градусів (IN1=1, IN2=0, IN3=0, IN4=0); 1000 для кута 90 градусів (IN1=0, IN2=0, IN3=0, IN4=1); 0010 для кута 180 градусів (IN1=0, IN2=1, IN3=0, IN4=0); 0100 для кута 270 градусів (IN1=0, IN2=0, IN3=1, IN4=0) та 0001 для повернення у початковий стан (кут дорівнює 0 градусів, IN1=1, IN2=0, IN3=0, IN4=0).

Нижче наведено схему алгоритму роботи біполярного двигуна у однофазному режимі з цілим кроком та керуючі програми для однофазного режиму з цілим кроком та напівкрокового режиму.

5.2.4 Схема алгоритму роботи біполярного крокового двигуна у однофазному режимі з цілим кроком

Схему алгоритму роботи біполярного крокового двигуна у однофазному режимі з цілим кроком наведено на рисунку 5.22.

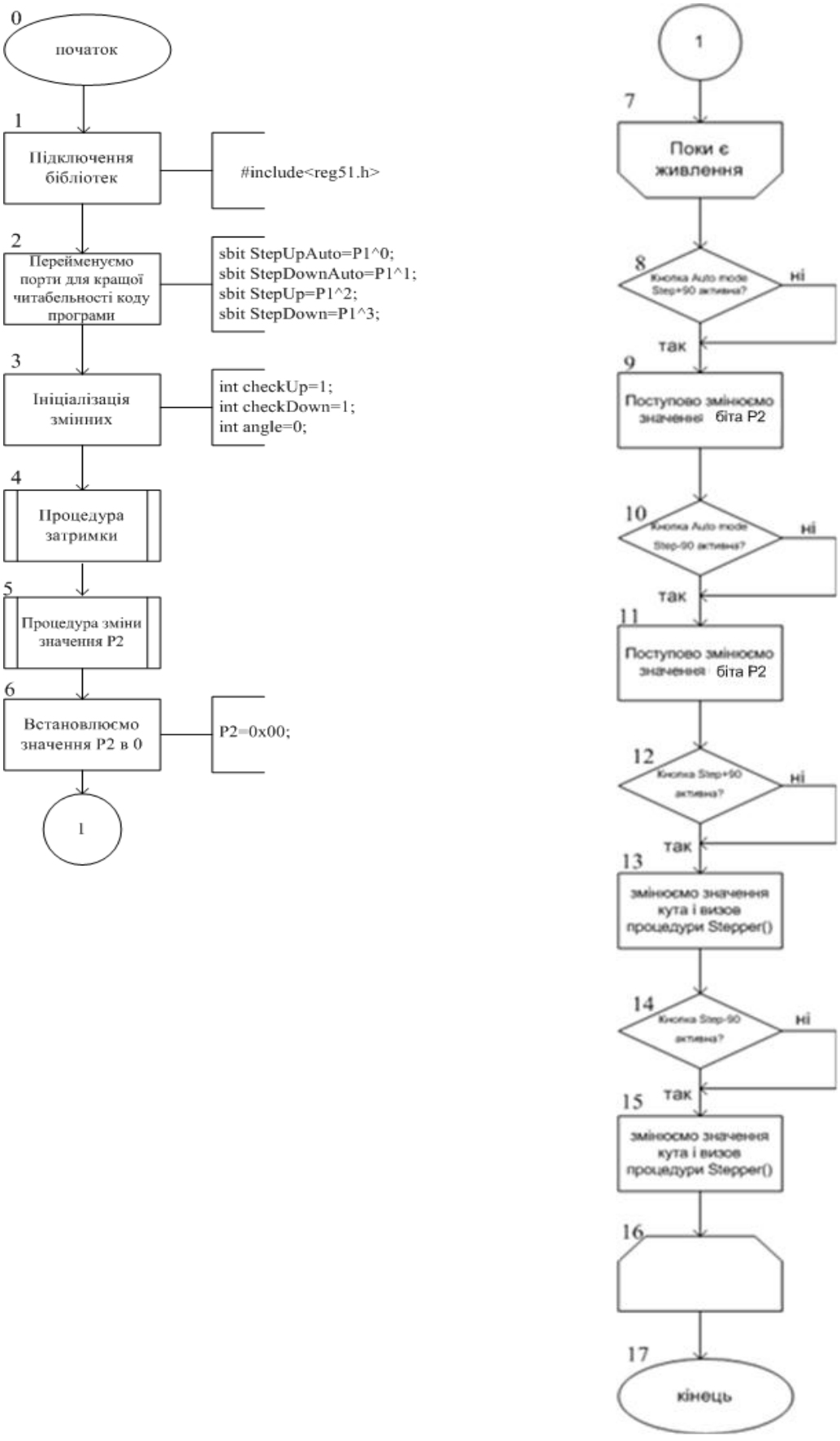
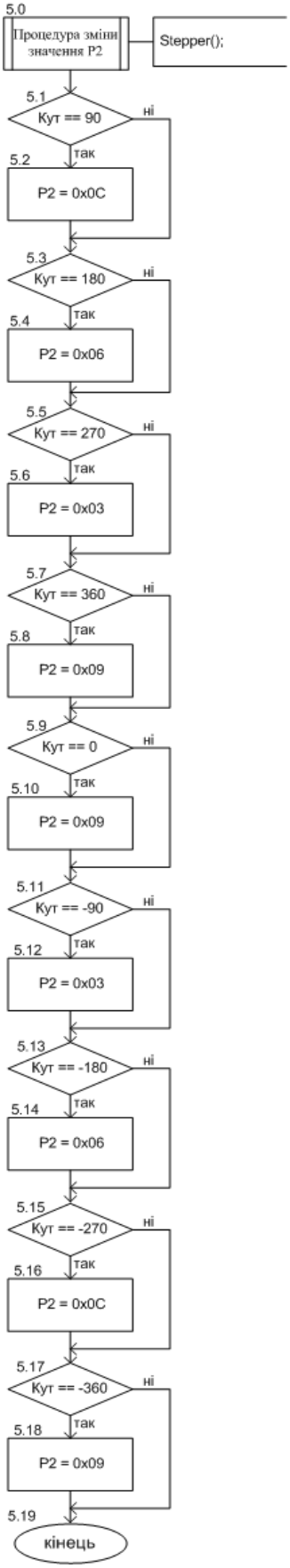


Рисунок 5.22 – Схема алгоритму роботи біполярного крокового двигуна у однофазному режимі з цілим кроком



Продовження рисунка 5.22

5.2.5 Керуюча програма роботи біполярного крокового двигуна у однофазному режимі з цілим кроком

```
#include <reg51.h>
#define true 1
#define false 0
sbit stepUpAuto = P1^0;
sbit stepDownAuto = P1^1;
sbit stepUp = P1^2;
sbit stepDown = P1^3;
unsigned char stepUP_=true;
int checkUp = 1;
int checkDown = 1;
int angle = 0;

void delay_ms(unsigned int);
void Stepper();
void delay_ms(unsigned int k){
unsigned int i,j;

for (i=0;i<k;i++){
     for(j=0;j<110;j++){}}
}

void main(void){
     P2=0x00;

     while (1){

     if(!stepUpAuto){

                P2=0x08;
                delay_ms(1000);

                P2=0x02;
```

```
          delay_ms(1000);

          P2=0x04;
          delay_ms(1000);

          P2=0x01;
          delay_ms(1000);
}
if(!stepDownAuto){
          P2=0x04;
          delay_ms(1000);

          P2=0x02;
          delay_ms(1000);

          P2=0x08;
          delay_ms(1000);

          P2=0x01;
          delay_ms(1000);
     }
if(!stepUp && checkUp==1){
          if(angle==360){
               angle=0;
          }
          angle+=90;
          checkUp=0;
}
if(stepUp)
{    checkUp=1; }

if(!stepDown && checkDown==1){
     if(angle==-360){
               angle=0;
          }
```

```
                angle-=90;
                checkDown=0;
      }
      if(stepDown)
      {    checkDown=1; }
      Stepper();
      }
}

void Stepper(){
      if(angle==90){
            P2=0x08;  }
      if(angle==180){
            P2=0x02;}
      if(angle==270){
            P2=0x04;}
      if(angle==360){
            P2=0x01;}
      if (angle==0)
            P2=0x01;
      if(angle==-90){
            P2=0x04;  }
      if(angle==-180){
            P2=0x02;}
      if(angle==-270){
            P2=0x08;}
      if(angle==-360){
            P2=0x01;}
}
```

На рисунку 5.23 наведено керуючі послідовності, які відображає осцилограф після натискання кнопки «Step +90 Auto Mode», для біполярного крокового двигуна, який працює у однофазному режимі з цілим кроком.

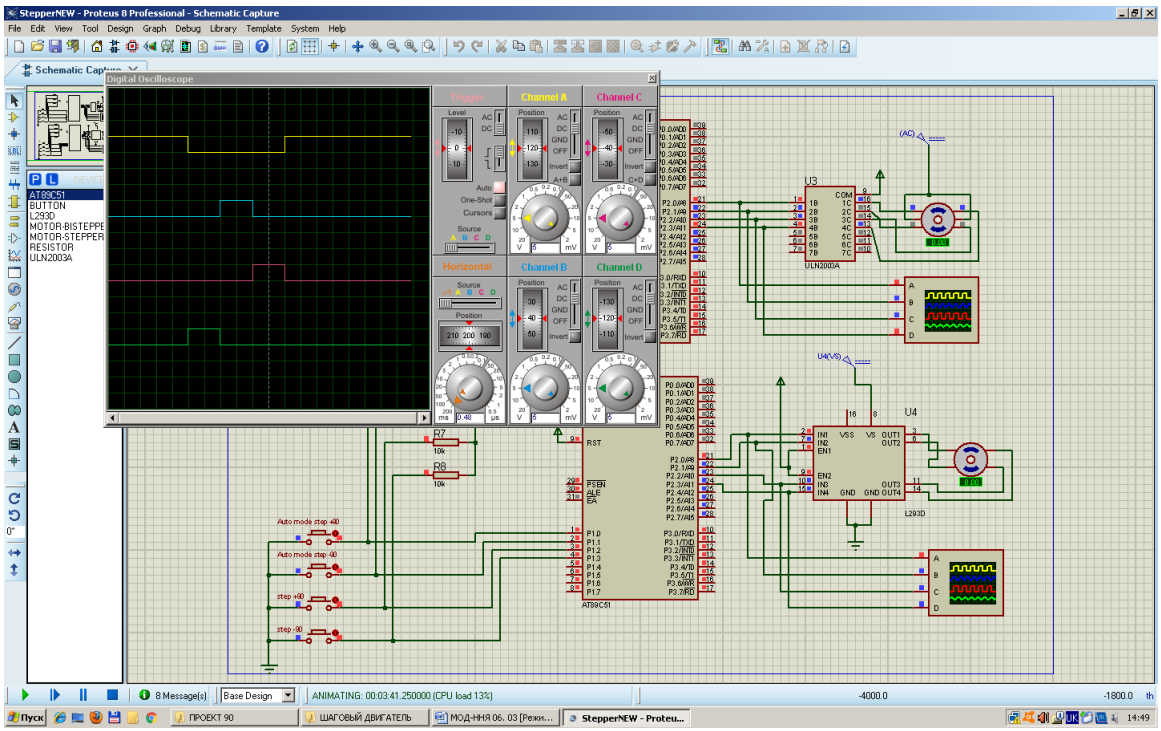


Рисунок 5.23 – Керуючі послідовності для біполярного крокового двигуна для однофазного режиму з цілим кроком, які відображає осцилограф після натискання кнопки «Step +90 Auto Mode»

5.2.6 Керуюча програма роботи біполярного крокового двигуна у напівкроковому режимі

```
#include <reg51.h>
#define true 1
#define false 0
void Stepper();
void delay_ms(unsigned int k){
unsigned int i,j;
for (i=0;i<k;i++){
     for(j=0;j<110;j++){}}
}
sbit stepUpAuto = P1^0;
sbit stepDownAuto = P1^1;
sbit stepUp = P1^2;
sbit stepDown = P1^3;
int checkUp = 1;
int checkDown = 1;
```

```
int angle = 0;
void main(void){
     P2=0x00;
     while (1)  {
     if(!stepUpAuto)  {
          P2=0x01;
               delay_ms(1000);
          P2=0x09;
               delay_ms(1000);
          P2=0x08;
               delay_ms(1000);
          P2=0x0A;
               delay_ms(1000);
          P2=0x02;
               delay_ms(1000);
          P2=0x06;
               delay_ms(1000);
          P2=0x04;
               delay_ms(1000);
          P2=0x05;
               delay_ms(1000);
          P2=0x01;
               delay_ms(1000);
     }
     if(!stepDownAuto){
          P2=0x01;
               delay_ms(1000);
          P2=0x05;
               delay_ms(1000);
          P2=0x04;
               delay_ms(1000);
          P2=0x06;
               delay_ms(1000);
          P2=0x02;
               delay_ms(1000);
```

```
            P2=0x0A;
                  delay_ms(1000);
            P2=0x08;
                  delay_ms(1000);
            P2=0x09;
                  delay_ms(1000);
            P2=0x01;
                  delay_ms(1000);
            }
      if(!stepUp && checkUp==1)     {
      if(angle==360)      angle=0;
                  angle+=45;
                  checkUp=0;
      }
      if(stepUp)     checkUp=1;
      if(!stepDown && checkDown==1)
      {
      if(angle==-360)     angle=0;
                  angle-= 45;
                  checkDown=0;
      }
      if(stepDown)   checkDown=1;
      Stepper();
      }
}
void Stepper(){
      if(angle==0)
            P2=0x01;
      if(angle==45)
            P2=0x09;
      if(angle==90)
            P2=0x08;
      if(angle==135)
            P2=0x0A;
      if(angle==180)
```

```
        P2=0x02;
    if (angle==225)
        P2=0x06;
    if(angle== 270)
        P2=0x04;
    if(angle== 315)
        P2=0x05;
    if(angle==360)
        P2=0x01;
    if(angle==-45)
        P2=0x05;
    if(angle==-90)
        P2=0x04;
    if(angle==-135)
        P2=0x06;
    if(angle==-180)
        P2=0x02;
    if (angle==-225)
        P2=0x0A;
    if(angle== -270)
        P2=0x08;
    if(angle==- 315)
        P2=0x09;
    if(angle==-360)
        P2=0x01;
}
```

На рисунку 5.24 наведено керуючі послідовності, які відображає осцилограф після натискання кнопки «Step +45 Auto Mode», для біполярного крокового двигуна, який працює у напівкроковому режимі.

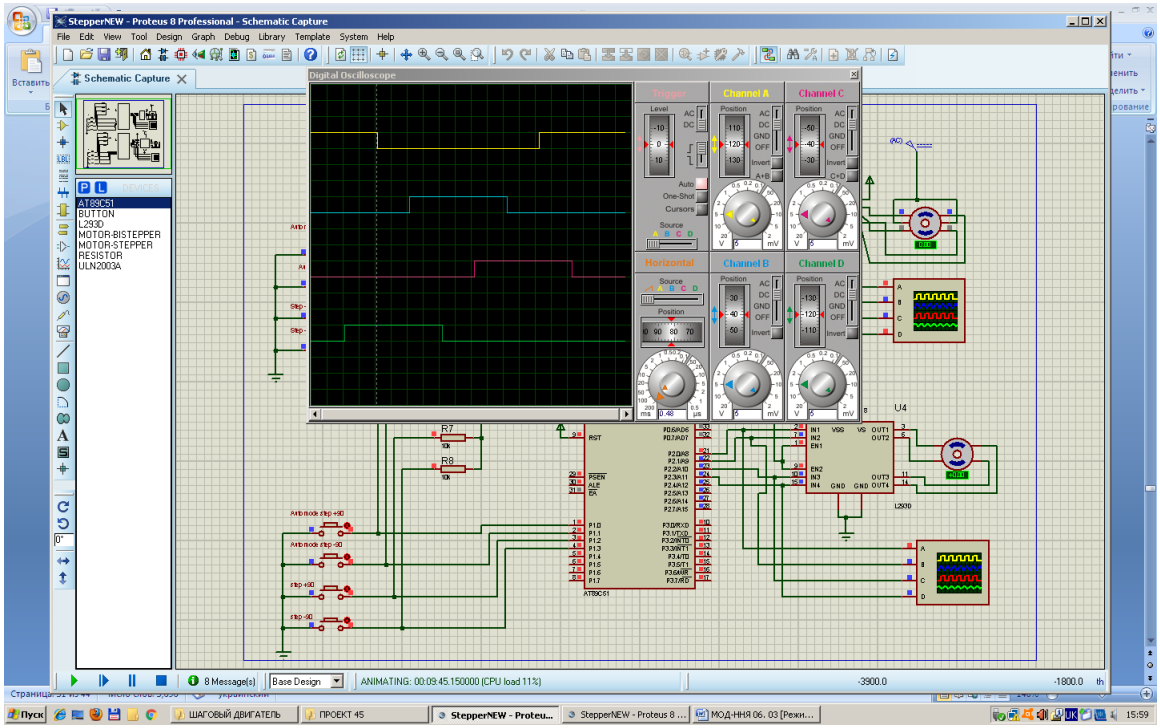


Рисунок 5.24 – Керуючі послідовності для біполярного крокового двигуна для напівкрокового режиму, які відображає осцилограф після натискання кнопки «Step +45 Auto Mode»

6 МОДЕЛЮВАННЯ ПРИСТРОЮ КЕРУВАННЯ ДВИГУНОМ ПОСТІЙНОГО СТРУМУ НА БАЗІ МІКРОКОНТРОЛЕРА AT89C51RD2

6.1 Опис моделі

Робочу модель пристрою керування двигуном постійного струму показано на рисунку 6.1.

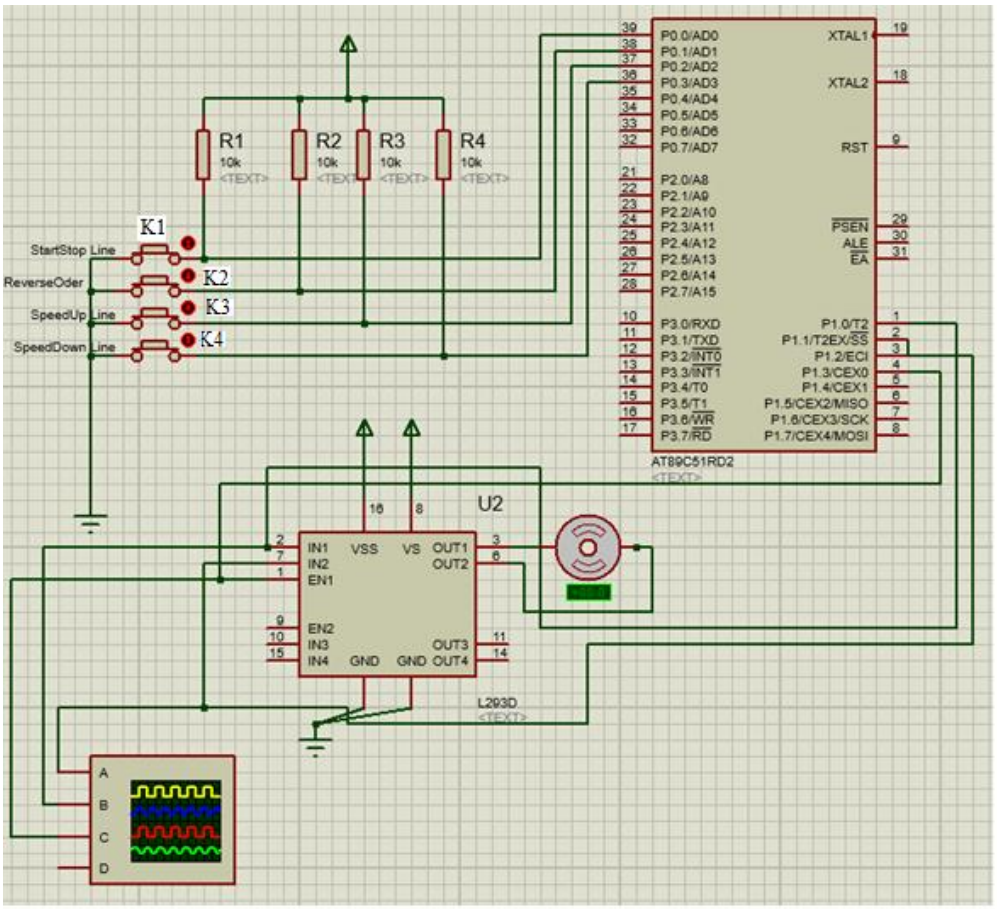


Рисунок 6.1– Схема моделі пристрою керування двигуном постійного струму

Розберемо цю модель докладніше. З лівого боку рисунка 6.1 ми бачимо кнопки керування: K1, K2, K3, K4 на які, через резистори R1, R2, R3, R4 відповідно, подається напруга +5В. Знизу праворуч ми бачимо двигун постійного струму DC Motor, а трохи лівіше мікросхему L293D, через яку мікроконтролер і керує двигуном за допомогою ШІМ – сигналу, який формує відповідний модуль МК–ра. Сам мікроконтролер AT89C51RD2 знаходиться у

верхньому правому куті. Зовнішні резистори ми використовуємо через те, що не використовуємо внутрішні підтягуючі резистори на вхідних лініях мікроконтролера. Нижче наведено рисунок 6.2 з виводами мікроконтролера.

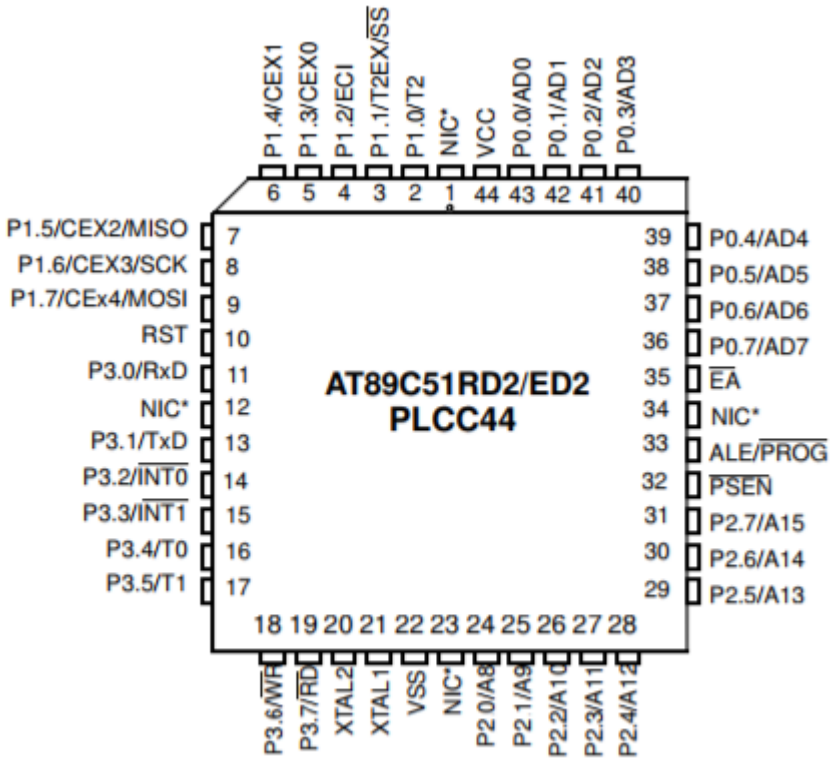


Рисунок 6.2 – Зовнішній вид мікросхеми AT89C51RD2

Далі ми більш докладніше пояснимо, що і куди підключаємо в моделі.

Почнемо з кнопок, за допомогою яких ми будемо здійснювати керування двигуном постійного струму DC Motor на моделі. Збільшений їх вигляд ми можемо бачити на рисунку 6.3.

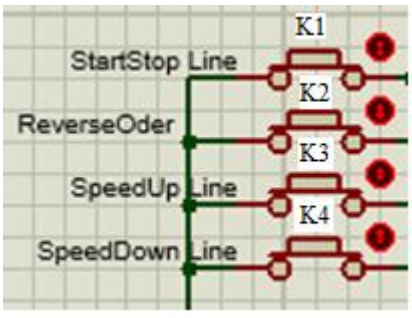


Рисунок 6.3 – Порядок розташування елементів керування

Кнопки є нормально замкненими та не фіксованими, тобто після зняття прикладеного зусилля повертаються у вихідний стан. В нашому випадку це означає наступне: у вихідному стані на входи мікроконтролера AT89C51RD2 по цих лініях напруга іде на землю, а на входах мікроконтролера спостерігається низький рівень напруги, тобто логічний 0. А коли кнопки розмикаються, на входи мікроконтролера подається високий рівень напруги, тобто логічна 1.

Кнопка K1 (StartStop Line) відповідає за вмикання та вимикання двигуна постійного струму DC Motor. Тобто коли двигун вимкнено, то вона його ввімкне. Коли двигун ввімкнений та обертається, то вона його вимкне. Кнопку K1 підключено до виводу мікроконтролера P0.0.

Кнопка K2 (ReverseOrder Line) відповідає за зміну напрямку обертання двигуна постійного струму DC Motor. Одне короткочасне розмикання – одна зміна напрямку. Проте потрібно мати на увазі, що двигун DC Motor інерційний, і він не зупиняється миттєво і одразу починає обертатися в інший бік. Після зміни напрямку програмно, у нього витрачається певний період часу, щоб фізично зупинитись, та почати обертатись у іншу сторону. Кнопку K2 підключено до виводу P0.1 AT89C51RD2.

Кнопка K3 (SpeedUp Line) відповідає за фізичне збільшення швидкості обертання нашого двигуна постійного струму DC Motor шляхом поступового збільшення постійної еквівалентної напруги ШІМ – сигналу до максимума, який дорівнює повному значенню напруги живлення на виводі VS мікросхеми L293D. Знову ж, двигун інерційний, тому швидкість обертання збільшується не стрибкоподібно, а поступово, і потребує певного проміжку часу, щоб вийти на новий встановлений програмно рівень.

Кнопку K3 підключаємо до виводу P0.2 нашого мікроконтролера.

Кнопка K4 (SpeedDown Line) відповідає за зменшення швидкості обертання двигуна постійного струму DC Motor, для чого вона зменшує значення постійної еквівалентної напруги ШІМ – сигналу до мінімума. Через інерційність двигуна швидкість обертання зменшується не стрибками, а

поступово і вимагає для свого зменшення певного проміжку часу. Кнопку K4 підключаємо до виводу P0.3.

Тепер розглянемо детальніше підключення мікросхеми L293D до мікроконтролера. Детально роздивитись це можна на рисунку 6.4.

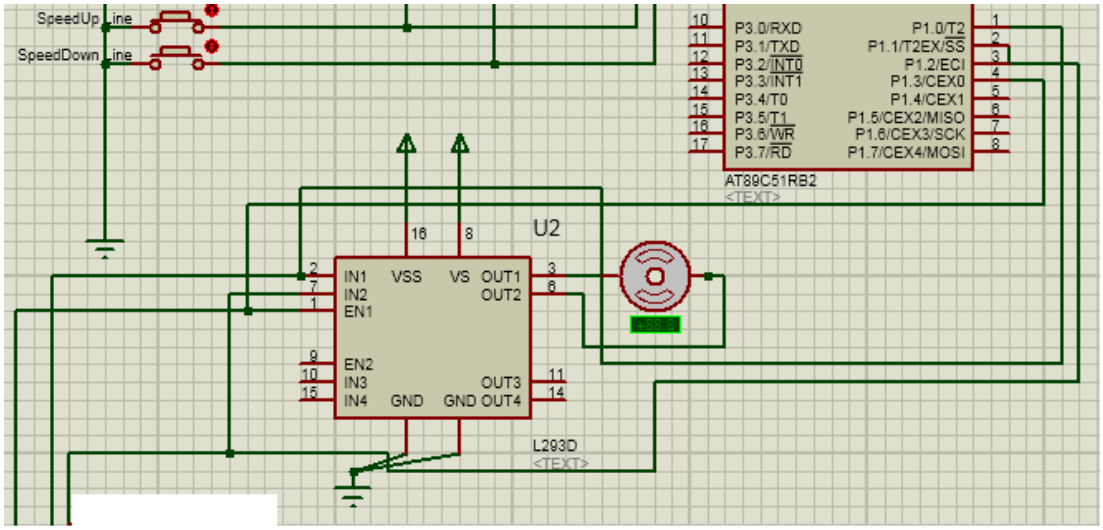


Рисунок 6.4 – Підключення мікросхеми L293D до мікроконтролера AT89C51RB2

Виводи мікроконтролера P1.0 та P1.1 запрограмовані як виходи і підключені, відповідно, до входів IN1 та IN2 мікросхеми L293D.

В залежності від керуючої програми на виходах P1.0 та P1.1 буде або високий рівень напруги, або низький, і в залежності від цього відповідний ключ у мосту мікросхеми буде або замкнений, або розімкнений. У початковому стані після запуска моделі на входи IN1 та IN2 від мікроконтролера подаються логічні одиниці та двигун стоїть. При IN1=1 та IN2=0 двигун буде обертатись за годинниковою стрілкою, а при IN1=0 та IN2=1 двигун буде обертатись у протилежному напрямку.

ШІМ – сигнал знімається з виводу P1.3/CEX0 мікроконтролера і подається на вхід EN1 мікросхеми L293D. У свою чергу вхід EN1 відповідає за роботу першої мостової схеми у складі мікросхеми (рисунок 6.13). Коли на ньому високий рівень, на перший міст подається живлення, яке підведено до виводу VS мікросхеми L293D. Коли на вході EN1 низький рівень, на перший міст напруга живлення не подається.

Саме завдяки цьому входові, на який подається ШІМ – сигнал, ми можемо змінювати швидкість обертання двигуна постійного струму.

Розглянемо використання інших виводів мікросхеми. Два виводи GND заземлені. На вхід VSS подається напруга живлення самої мікросхеми L293D: 5В. На вхід VS подається напруга живлення двигуна постійного струму: 5В.

Після цього нам залишилось ознайомитись з підключенням двигуна постійного струму та мікросхеми L293D. Це підключення зображено на рисунку 6.5.

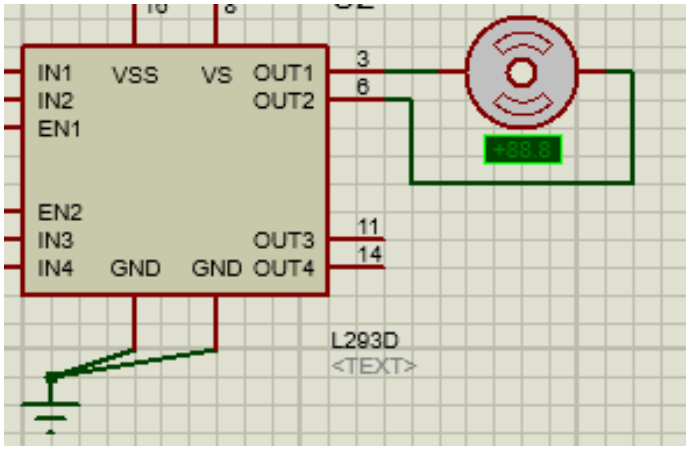


Рисунок 6.5 – Підключення двигуна постійного струму до мікросхеми L293D

Як ми бачимо з рисунку 6.5, наш двигун постійного струму підключено до 3 та 6 виходів мікросхеми L293D, а саме OUT1 та OUT2.

На ці виходи, в залежності від того, чи ввімкнене живлення першої мостової схеми, що керується входом EN1, та в залежності від стану ключів у першій мостовій схемі, які керуються входами IN1 та IN2, або подається напруга живлення, яку ми подали на вхід VS, або напруга не подається.

У випадку, коли на один з виводів, OUT1 чи OUT2, подається напруга живлення, а на інший не подається, виникне різниця потенціалів, що змусить потекти струм по обмотці нашого двигуна постійного струму, внаслідок чого він почне обертатися.

Деякі фрагменти, які демонструють роботу системи, наведено на рисунках 6.6…6.10.

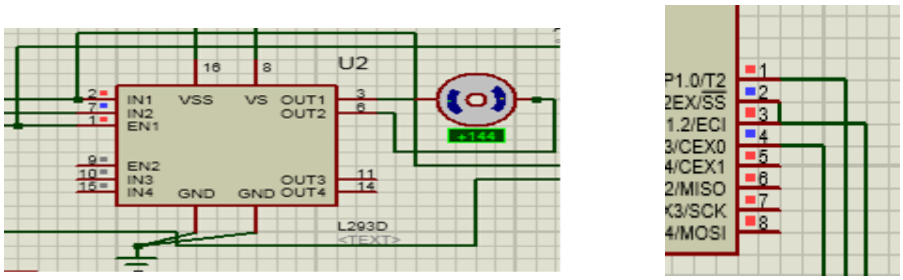


Рисунок 6.6 – Стан моделі після натискання кнопки «старт/стоп»

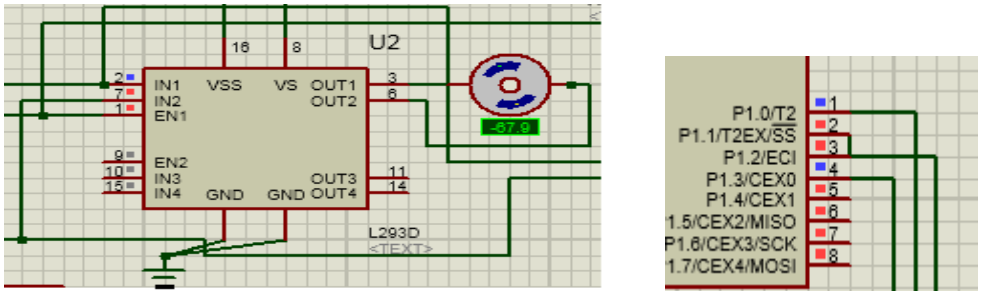


Рисунок 6.7 – Стан моделі після активації режиму реверсу

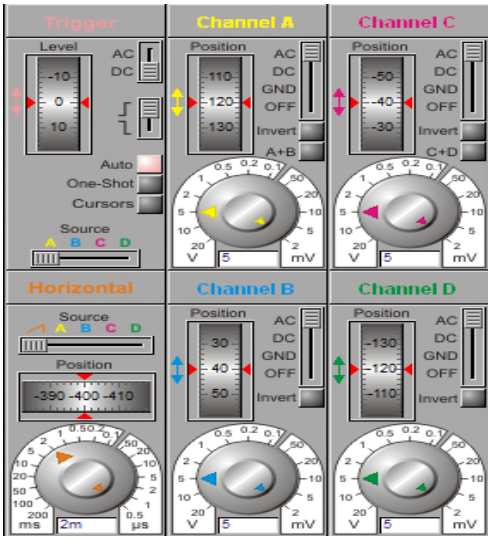


Рисунок 6.8 – Налаштування осцилографа

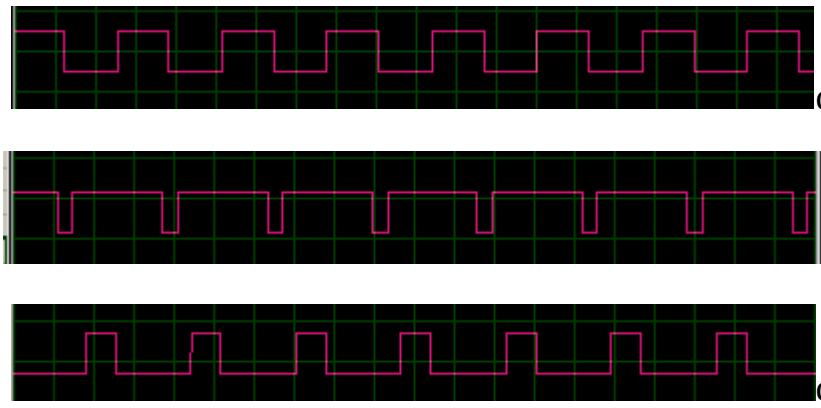


Рисунок 6.9 –ШІМ– сигнал, що подається на двигун (шпаруватість q=2, q < 2, q >2)

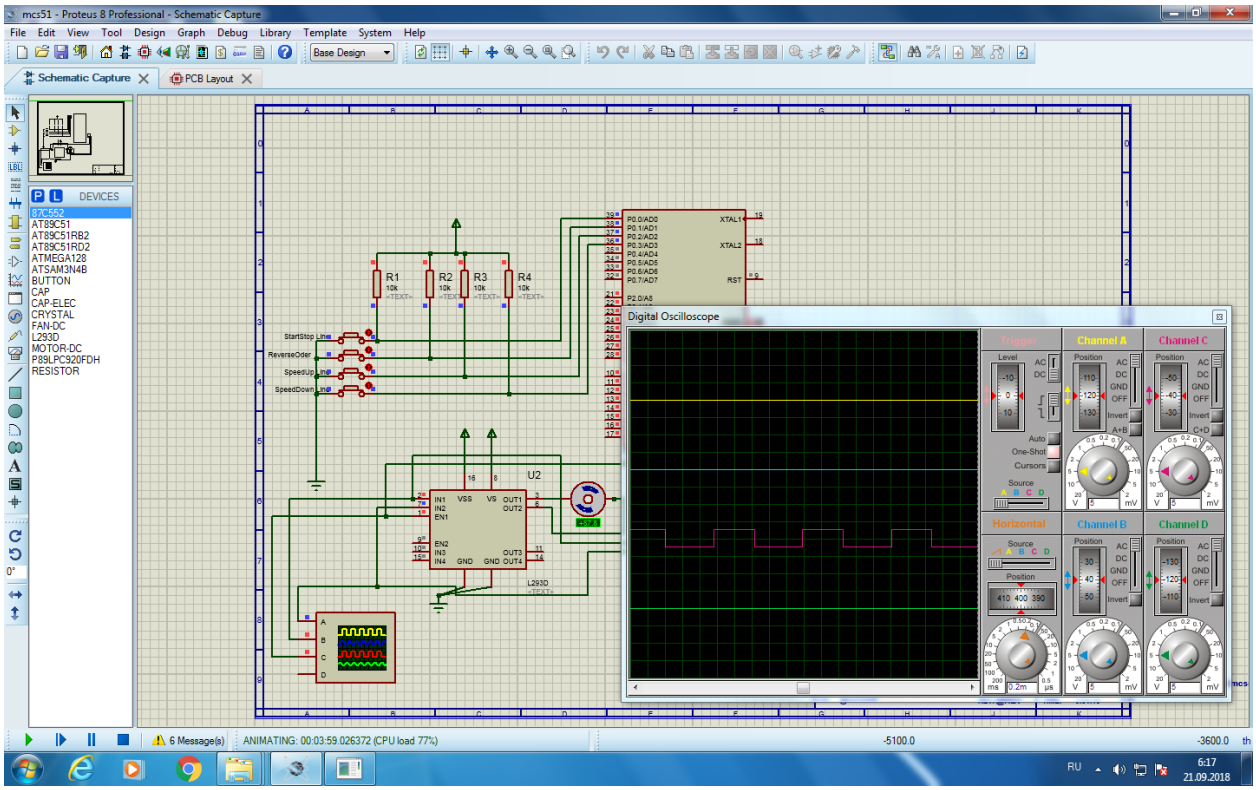


Рисунок 6.10 – Стан усієї системи після запуску сценарію

6.2 Мікроконтролер

Для контролю та виконання всіх функцій, які були покладені на систему було використано мікроконтролер AT89C51RD2 сім`ї MK51 (MCS51).

AT89C51RD2–досить економічний 8 – розрядний

КМОН–мікроконтролер. Його швидкодія досягає 1 млн. операцій в секунду, оскільки більшість команд виконується за один машинний цикл.

Мікроконтролер має 32 виводи типу вхід – вихід. В AT89C51RD2 вмонтовано FLASH–пам'ять, яку є можливість програмувати та перепрограмувати та масив програмованих лічильників.

Також мікроконтролер може формувати ШІМ – сигнал для керування ДПС.

6.3 Двигун постійного струму

В моделі використовується електродвигун постійного струму, для керування яким використовується широтно–імпульсна модуляція. Реалізувати широтну–імпульсну модуляцію можна за допомогою вбудованого в мікроконтроллер таймера, який може формувати ШІМ – сигнали. Вибір двигуна є важливим, оскільки від цього вибору буде залежати стабільність роботи системи. Реверсивний режим двигуна постійного струму здійснюється зміною полярності.

Чим більше величина відношення «тривалість імпульсу/період» ШІМ – сигналу, тим швидше буде обертатися двигун.

6.4 Драйвер ШІМ

Для керування двигуном постійного струму нам потрібно підключати його за допомогою відповідної мостової схеми, яку зображено на рисунку 6.11.

Також, нам бажано знайти пристрій, який би перетворював керуючі сигнали малої потужності у струми, достатні для керування двигунами. Такі пристрої називаються драйверами двигунів.

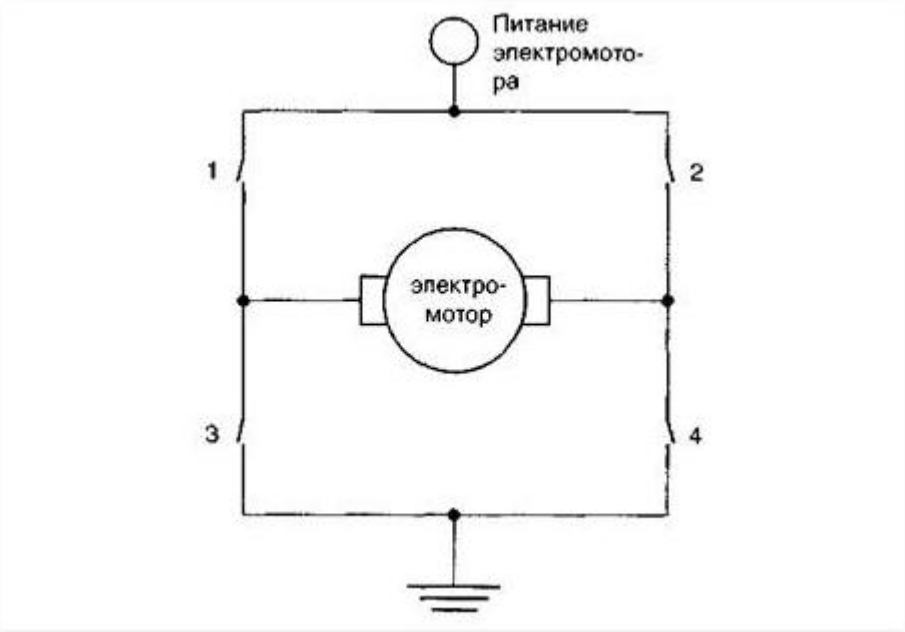


Рисунок 6.11– Мостова схема підключення двигуна постійного струму

Існує достатньо багато самих різних схем для керування двигунами постійного струму. Вони відрізняються як потужністю, так і елементною базою, на якій вони виконані. Нижче зупинимось на простому драйвері керування двигунами, який виконано у вигляді повністю готової до роботи мікросхеми. Назва цієї мікросхеми L293D і вона є одною з самих розповсюджених мікросхем, призначених для цих цілей. Зовнішній вигляд мікросхеми L293D показано на рисунку 6.12.

Рисунок 6.12 – Мікросхема L293D

L293D включає одразу два драйвери для керування електродвигунами відносно невеликої потужності. Це відбувається через чотири незалежних канали, які об’єднано у дві пари. Мікросхема також має дві пари входів для сигналів керування і дві пари виходів для підключення електродвигунів. Крім того, у L293D є два входи для вмикання кожного з драйверів.

Ці входи використовуються для керування швидкістю обертання електродвигунів за допомогою широтно–імпульсної модуляції сигналів. Саме через це ця мікросхема нам добре підходить.

Також, L293D забезпечує розділене живлення для мікросхеми і для керованих нею двигунів, що дозволяє підключати електродвигуни з більшою напругою живлення, чим у мікросхеми. Розділення живлення мікросхеми і електродвигунів нам також буде необхідним для зменшення завад, які викликано стрибками напруги, що пов’язані з роботою двигунів.

Обидва драйвери, що входять у склад мікросхеми мають ідентичні принципи роботи, тому розглянемо принцип роботи лише одного з них. Він наведений на рисунку 6.13.

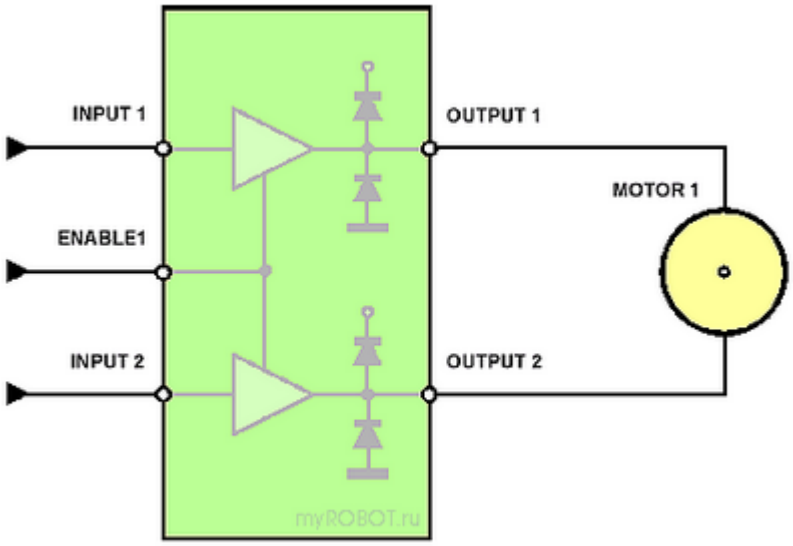


Рисунок 6.13 – Схематичний вигляд драйвера з мікросхеми L293D

До виходів OUTPUT1 та OUTPUT2 підключається двигун постійного струму MOTOR1. На вхід ENABLE1, який відповідає за ввімкнення драйвера, подаємо керуючий сигнал, наприклад, під’єднаємо його до додатного полюсу джерела живлення +5V. Якщо при цьому на входи INPUT1 та INPUT2 не подаються відповідні керуючі сигнали, то двигун обертатися не буде.

Якщо вхід INPUT1 з’єднати з додатним полюсом джерела живлення, а вхід INPUT2 – з від`ємним, то двигун почне обертатися.

Тепер з’єднаємо вхід INPUT1 з від`ємним полюсом джерела струму, а вхід INPUT2 – з додатним. Двигун почне обертатися в іншу сторону.

Спробуємо подати сигнали одного рівня одразу на два керуючих входи INPUT1 та INPUT2, тобто під’єднати обидва входи до додатного полюсу джерела живлення або до від`ємного. Як результат – двигун обертатися не буде.

Якщо ми приберемо керуючий сигнал з входу ENABLE1, то при будь яких варіантах наявності сигналів на двох входах INPUT1 та INPUT2 двигун обертатися не буде. На вхід ENABLE1 подається ШІМ – сигнал, який формується модулем ШІМ мікроконтролера. Змінючи шпаруватість цього сигналу ми змінюємо постійну складову імпульсного сигналу, що в свою чергу змінює швидкість обертання двигуна.

На рисунку 6.14 показано нумерацію та позначення виводів мікросхеми.

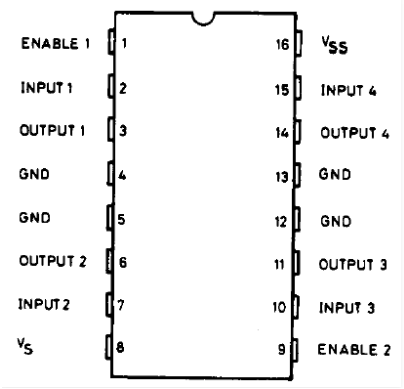


Рисунок 6.14 – Нумерація та позначення виводів мікросхеми L293D

Призначення виводів:

- входи ENABLE1 та ENABLE2 відповідають за вмикання відповідного драйвера, що входить у склад мікросхеми. Це ті самі входи для ШІМ – сигналу, які ми будемо використовувати;
- входи INPUT1 та INPUT2 керують двигуном, який підключено до виходів OUTPUT1 та OUTPUT2;

– входи INPUT3 та INPUT4 керують другим двигуном, який підключено до виходів OUTPUT3 та OUTPUT4;
– контакт Vs з'єднують з додатним полюсом джерела живлення двигунів або просто з додатним полюсом джерела живлення, якщо схема і двигуни живляться від одного джерела. Простіше кажучи, цей контакт відповідає за живлення електродвигунів;
– контакт Vss з'єднують з додатним полюсом джерела живлення. Цей контакт забезпечує живлення самої мікросхеми;
– чотири контакти GND з'єднують з "землею", спільним дротом або від`ємним полюсом джерела живлення. Крім того, за допомогою цих контактів зазвичай забезпечують тепловідвід від мікросхеми, тому їх краще всього паяти на достатньо широку контактну поверхню.

Інший спосіб керування двигунами постійного струму заснований на використанні мостових схем типу L298N (SGS– Thomson, RS636– 384) (див. рисунки 2.16, 2.17).

6.5 Захисні діоди

Як видно з рисунка 2.17 схема містить 8 захисних діодів: VD1…VD8. Ці діоди призначені для захисту транзисторних ключів у складі драйвера від додатних і від`ємних паразитних імпульсів досить високої амплітуди, які з'являються на обмотках двигуна при комутації обмоток. Додатні імпульси виникають при запиранні (вимиканні) ключів, а від`ємні – при включенні. Механізм виникнення цих імпульсів описано у (2.5).

6.6 Модуль PCA мікроконтролера

6.6.1 Загальна характеристика

Мікроконтролер AT89C51RD2, який використовується в роботі, має модуль PCA–Programmable Counter Array (група програмованих лічильників).

Цей модуль може програмуватися в режим широтно–імпульсного модулятора та формувати ШІМ – сигнал. Модуль PCA має 16–бітний таймер/лічильник, який складається із регістрів CH і CL (відповідно старший і молодший байти), а також групу 16–бітних програмованих модулів порівняння/захоплення. В таблиці 6.1 приведено зв'язок таймера та модулів порівняння/захоплення PCA із зовнішніми виводами мікроконтролерів.

Спрощену структуру 16–бітного таймера/лічильника PCA представлено на рисунку 6.15.

Таблиця 6.1 – Зв'язок модулів PCA і зовнішніх виводів мікроконтролерів

Модулі PCA	Зовнішній вивід
16–бітний лічильник	P1.2/ECI
16–бітний модуль 0 порівняння/захоплення	P1.3/CEX0
16–бітний модуль 1 порівняння/захоплення	P1.3/CEX1
16–бітний модуль 2 порівняння/захоплення	P1.3/CEX2
16–бітний модуль 3 порівняння/захоплення	P1.3/CEX3
16–бітний модуль 4 порівняння/захоплення	P1.3/CEX4

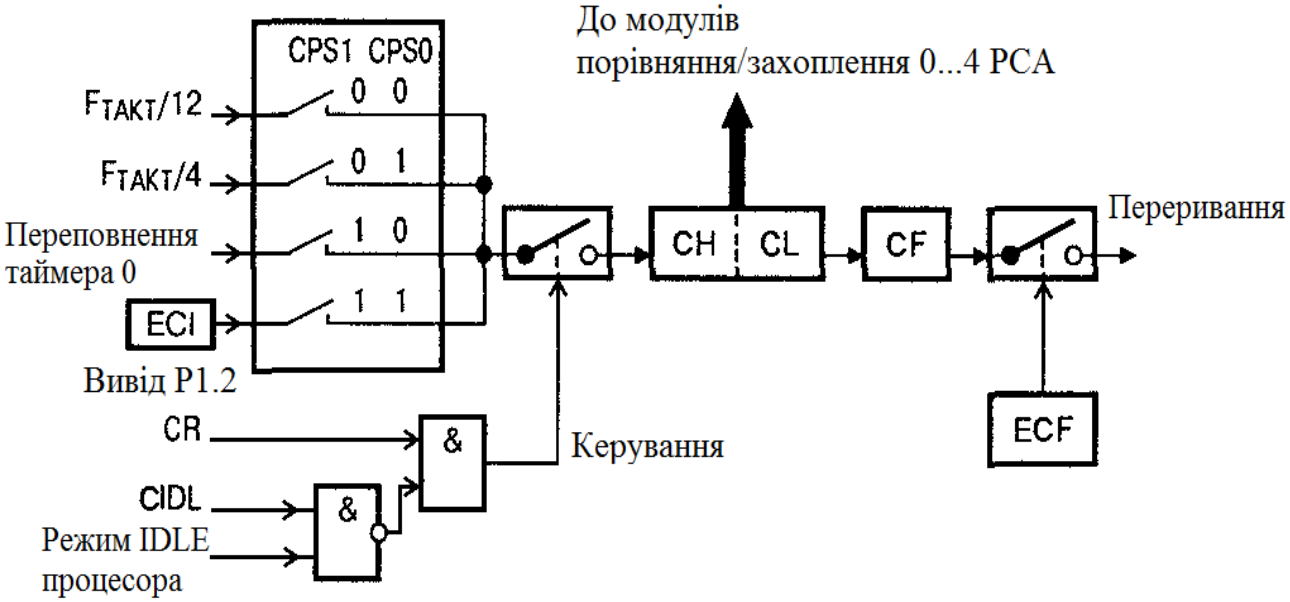


Рисунок 6.15 – Спрощена структура 16–бітного таймера/лічильника PCA

Користувач може обрати один з чотирьох варіантів вхідних сигналів для таймера/лічильника (Т/Л) PCA:

- частоту тактового генератора, яку поділено на 12 (Fтакт/12). При цьому таймер PCA інкрементується один раз в кожному машинному циклі. Якщо тактова частота дорівнює 16 МГц – таймер інкрементується кожні 750 нс;
- частоту тактового генератора, яку поділено на 4 (Fтакт/4). Т/Л PCA інкрементується тричі в кожному машинному циклі, тобто кожні 250 нс при тактовій частоті генератора 16 МГц;
- переповнення таймера/лічильника Т/С0. Т/Л PCA інкрементується кожен раз, коли відбувається переповнення Т/С0. Цей режим дає можливість програмно задавати частоту вхідного сигналу Т/Л PCA;
- вхідний сигнал на лінії P1.2 (ECI). Т/Л PCA інкрементується з кожним переходом із 1 в 0 на вході ECI (вивід P1.2 мікроконтролера). Максимально допустима частота сигналу на цьому вході дорівнює частоті тактового генератора, яку поділено на 8.

Вибір вхідного сигналу здійснюється бітами CPS0 і CPS1 в регістрі CMOD.

В цьому регістрі також знаходиться біт ECF, який дозволяє переривання при переповненні Т/Л PCA. Крім того, встановлення в 1 біта CIDL дає можливість відключати Т/Л PCA в режимі Idle, що дає зниження енергоспоживання в згаданому режимі на 30%. В таблиці 6.2 приведено імена і призначення бітів регістра CMOD.

Ще два біти, що наведено у структурі таймера/лічильнику PCA (рисунок 6.15), знаходяться в регістрі CCON (таблиця 6.3). Біт CF встановлюється апаратно при переповненні таймера/лічильника. Встановлення або скидання біта CR відповідно вмикає або вимикає лічильник при умові, що біт CIDL=0 або відсутній режим IDLE.

Таблиця 6.2 – Регістр режиму таймера/лічильника PCA–CMOD

Символ	Позиція	Ім’я та призначення
CIDL	CMOD.7	Встановлення цього біта в 1 завершує лічбу Т/Л PCA в режимі Idle. При нульовому значенні CIDL лічба в режимі Idle не завершується
WDTE	CMOD.6	Біт керування вартовим таймером. Встановлення його в 1 дозволяє функціонування вартового таймера, скидання в 0 – забороняє
–	CMOD.5	Зарезервовано для подальшого використання
–	CMOD.4	Зарезервовано для подальшого використання
–	CMOD.3	Зарезервовано для подальшого використання
CPS1	CMOD.2	Вибір джерела тактування Т/Л PCA, старший біт (рисунок 6.15)
CPS0	CMOD.1	Вибір джерела тактування Т/Л PCA, молодший біт (рисунок 6.15)
ECF	CMOD.0	Біт дозволу переривання за переповненням таймера/лічильника PCA. При ECF=1 дозволено переривання при встановленні біта CF регістра CCON, при ECF=0 – заборонено переривання.
Адреса регістра CMOD – 0D9H, значення при скиданні – 00xxx000b		

Таблиця 6.3 – Регістр керування PCA–CCON

Символ	Позиція	Ім’я та призначення
CF	CMOD.7	Прапорець переповнення Т/Л PCA. Встановлюється апаратно і викликає переривання при ECF=1. Може бути встановлено також і програмно. Скидається тільки програмний шляхом
CR	CMOD.6	Біт керування ввімкненням лічильника PCA. Встановлення його в 1 дозволяє функціонування лічильника, скидання в 0 – забороняє при умові, що біт CIDL=0 або відсутній режим IDLE
–	CMOD.5	Зарезервовано для подальшого використання.
CCF4	CMOD.4	Прапорець порівняння/захоплення модуля 4 PCA. Очищується програмно
CCF3	CMOD.3	Прапорець порівняння/захоплення модуля 3 PCA
CCF2	CMOD.2	Прапорець порівняння/захоплення модуля 2 PCA
CCF1	CMOD.1	Прапорець порівняння/захоплення модуля 1 PCA
CCF0	CMOD.0	Прапорець порівняння/захоплення модуля 0 PCA
Адреса регістра CCON – 0D8H, значення при скиданні – 00x00000b		

6.6.2 Модуль порівняння/захоплення

Кожній із п’яти модулів порівняння/захоплення може функціонувати в наступних режимах:

16–бітний регістр–защіпка з керуванням за фронтом вхідного імпульсу;

16–бітний регістр–защіпка з керуванням за спадом вхідного імпульсу;

16–бітний програмований таймер/лічильник;

16–бітний високошвидкісний вихід;

8–бітний широтно–імпульсній модулятор (ШІМ).

Додатково модуль 4 може використовуватися як вартовий таймер.

Кожен модуль можна запрограмувати на виконання будь–якої із функцій незалежно від інших. Кожен із модулів має регістр CCAPMn (n=0…4), за допомогою якого відбувається вибір режиму його функціонування (таблиця 6.4).

Таблиця 6.4 – Регістр модуля порівняння–защіпки n: CCAPMn

Символ	Позиція	Ім’я і призначення
–	CCAPMn.7	Зарезервовано для подальшого використання
ECOMn	CCAPMn.6	Прапорець дозволу компаратора PCA (функція порівняння)
CAPPn	CCAPMn.5	Біт дозволу захоплення за додатним фронтом (перепад з 0 в 1)
CAPNn	CCAPMn.4	Біт дозволу захоплення за від’ємним фронтом (за спадом, перепадом з 1 в 0)
MATn	CCAPMn.3	При встановлені цього біта в 1 при рівності лічильника PCA і відповідного регістра порівняння/защіпки встановлюється прапорець переривання CCFn
TOGn	CCAPMn.2	При встановленні цього біта в 1 при рівності лічильника PCA і відповідного регістра порівняння/защіпки змінюється рівень (з 0 на 1 або навпаки) на відповідному виводі CEXn
PWMn	CCAPMn.1	При встановленні цього біта на вивід CEXn видається широтно–імпульсно–модульований сигнал (ШІМ– сигнал)
ECCFn	CCAPMn.0	Дозвіл переривання за встановленням прапорця CCFn регістра CCON
Адреса регістра з 0DAH (CCAPM0) по 0DEH (CCAPM4), значення при скиданні – x0000000b		

Біт ECCFn регістра CCAPMn дозволяє переривання від модуля PCA, якщо його прапорець переривання (CCFn) було встановлено. Ці прапорці (CCF0…CCF4) знаходяться в регістрі CCON і встановлюються при описаних вище умовах функціонування модуля в режимах захоплення, програмованого таймера або в режимі високошвидкісного виходу. Крім того, кожен модуль має пару власних 8–бітних регістрів порівняння/защіпки (CCAPnH і CCAPnL). В цих регістрах запам’ятовуються значення таймера PCA в момент приходу зовнішнього сигналу на захоплення інформації або містяться дані для порівняння з показами таймера/лічильника. В ШІМ–режимі старший байт CCAPnH керує шпаруватістю вихідних імпульсів.

6.6.3 Режим широтно–імпульсного модулятора

Будь–який із п’яти модулів PCA може бути запрограмовано в режим широтно–імпульсного модулятора (ШІМ). Режим ШІМ може бути використано для перетворення цифрових даних в аналоговий сигнал з мінімальними апаратними витратами. Частота модуляції залежить від швидкості лічби таймера PCA. З 16–ти мегагерцовим кварцовим резонатором максимальна частота сигналу ШІМ не перевищує 15,6 кГц.

На рисунку 6.16 показано спрощену структуру таймера PCA в режимі ШІМ.

Для роботи в режимі ШІМ біти ECOMn і PWMn в регістрі CCAPMn повинно бути встановлено в одиницю. PCA виробляє 8–бітний ШІМ – сигнал шляхом порівняння вмісту CCAPnL і CL. Якщо CL<CCAPnL, то на зовнішньому виводі відповідного модуля порівняння/захоплення буде сигнал з нульовим рівнем, якщо CL>CCAPnL, то з одиничним.

Значення, яке записано в CCAPnL, задає шпаруватість вихідного сигналу. Для зміни значення CCAPnL без збоїв користувач повинен попередньо занести потрібне значення в CCAPnH. Це значення апаратно заноситься в CCAPnL в момент, коли CL міняє своє значення з FFH на 0, що відповідає початку нового циклу формування вихідного сигналу.

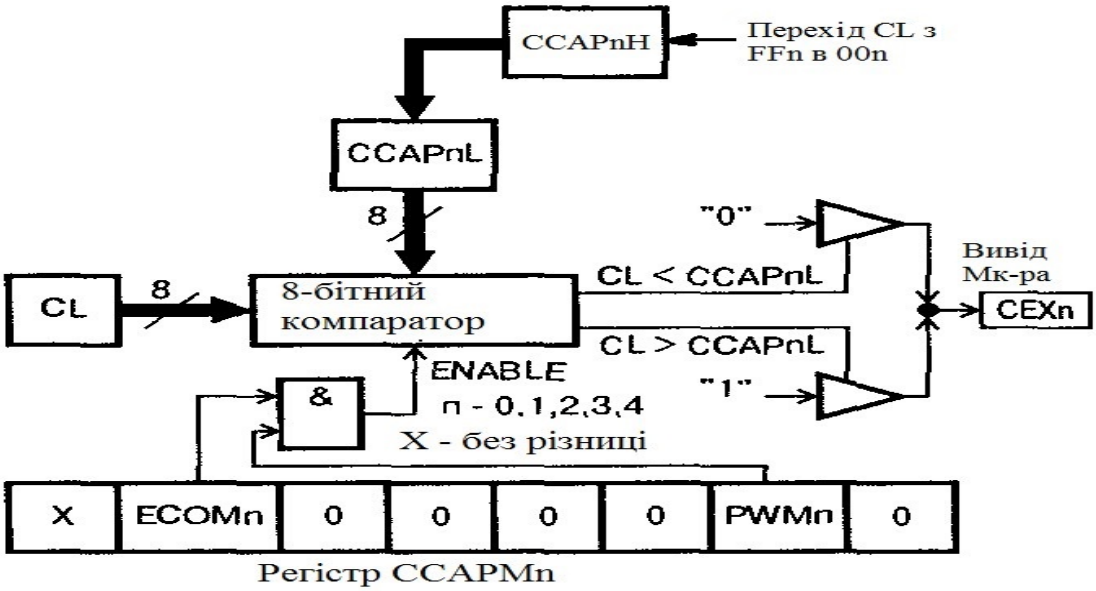


Рисунок 6.16 – PCA лічильник/таймер у режимі ШІМ

В CCAPnH можна занести будь–яке ціле число від 0 до 255, при цьому коефіцієнт заповнення змінюється від 100 до 0,4 %. Коефіцієнт заповнення розраховується за формулою:

$$K_{зап} = \frac{t_{імп}}{T} \cdot 100\% \quad , \qquad (6.1)$$

де tімп – тривалість вихідного імпульсу; T – період слідування вихідних імпульсів.

Нульове значення Кзап забезпечується шляхом прямого запису інформації на вивід відповідного порту. Значення CCAPnH, при якому досягається потрібний Кзап, визначається відношенням

$$CCAPnH=256\ (1 – Кзап), \qquad (6.2)$$

де CCAPnH – найближче ціле число до значення в правій частині виразу, Кзап представляє собою величину в діапазоні від 0,004 до 1.

В нашому випадку ми використовуємо регістр CCAP0H для регулювання шпаруватості сигналу (у межах від 0 до 255, що видно у приведеному нижче коді програми) та регістри CCON і CMOD для встановлення масиву лічильників у формат генерації ШІМ – сигналу на відповідному виході.

6.6.4 Розрахунок характеристик ШІМ – сигналу

Режим ШІМ – сигналу генерує імпульси за допомогою порівняння молодшого байта таймера масиву лічильників CL з молодшим байтом регістра CCAPnL (рисунок 6.16). Якщо значення CL менше за значення CCAPnL, то генерується низький рівень сигналу, у іншому випадку високий. У цьому режимі частота ШІМ – сигналу залежить від частоти оновлення масиву лічильників, яка задається частотой мікроконтролера у середовищі Proteus. Частота ШІМ – сигналу дорівнює

$$f_{\text{шім}} = f_{PCA}/256,$$

де $f_{PCA} -$ частота оновлення масиву лічильників.

Для генерації сигналу частотою в 10кГц ми встановлюємо частоту мікроконтролера рівною частоті оновлення масиву лічильників

$$f_{PCA} = 10000 * 256 = 2.56\text{МГц}.$$

Для задання значення шпаруватості сигналу ми записуємо в регістр:

$$\text{CCAPnH} = 256(1 - \text{К}_{\text{зап}}).$$

В CCAPnH можна занести будь–яке ціле число від 0 до 255, при цьому коефіцієнт заповнення змінюється від 100 до 0,4 %. Коефіцієнт заповнення розраховується за формулою:

$$K_{\text{зап}} = \frac{t_{\text{імп}}}{T} \cdot 100\%.$$

Таким чином, для отримання, наприклад, значення коефіцієнта заповнення в 75% потрібно записати значення

$$\text{CCAPnH} = 256(1 - 0.75) = 64 = 0\text{x}40.$$

При цьому шпаруватість дорівнює 1.33.

6.7 Схема алгоритму роботи та керуюча програма

В роботі розроблено схему алгоритму роботи (рисунок 6.17) та керуючу програму, які наведено нижче.

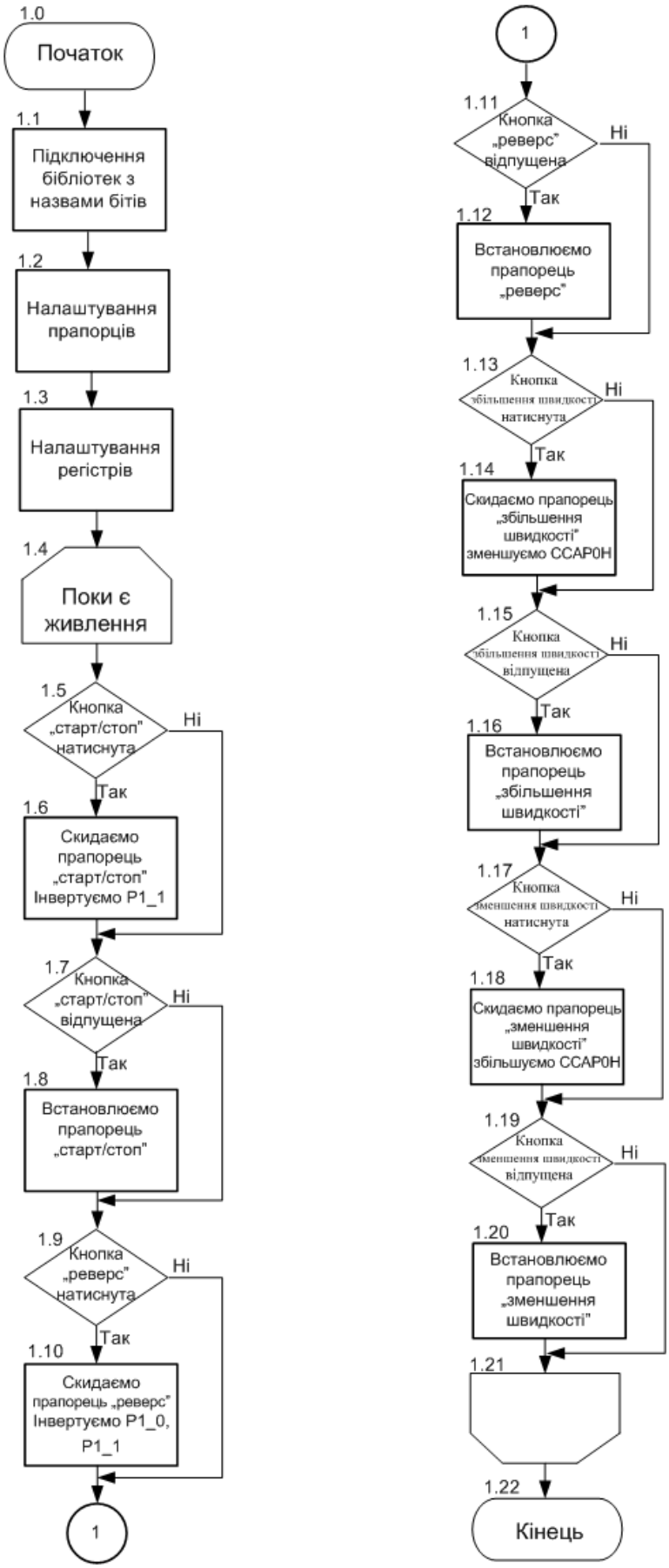


Рисунок 6.17 – Схема алгоритму роботи

Керуюча програма мовою С.

```
#include<reg51rd2.h>  //1.1

typedef unsigned char bool;
#define true 1
#define false 0

// Main Function
int main(void)

{
     //1.2
     bool start_released = true,
     reverse_released = true,
     speedup_released = true,
     speeddown_released = true;

     CMOD = 0x02; //1.3 Ініціалізаіця PCA
     CL = 0x00;
     CCAP0L = 0x40;
     CCAP0H = 0x40; // 75% заповненості – початковий рівен
     CCAPM0 = 0x42; // Установка PCA у ШІМ – режим
     CR = 1;

     while (1)  //1.4
     {
          if (P0_0 && start_released) {  //1.5
               start_released = false;
               P1_1 ^= 1;                //1.6
          }
          if (!P0_0){                    //1.7
               start_released = true;     //1.8
          }

          if (P0_1 && reverse_released) { //1.9
               reverse_released = false;
               P1_0 ^= 1;                 //1.10
               P1_1 ^= 1;
          }
          if (!P0_1){                     //1.11
               reverse_released = true;   //1.12
          }

          if (P0_2 && speedup_released) { //1.13
               speedup_released = false;
               if (CCAP0H > 0x0F) {       //1.14
                    CCAP0H – =15;
               }
          }
          if (!P0_2){                      //1.15
               speedup_released = true;    //1.16
          }
```

```
            if (P0_3 && speeddown_released) { //1.17
                 speeddown_released = false;
                 if (CCAP0H < 0xF0) {          //1.18
                      CCAP0H+=15;
                 }
            }
            if (!P0_3){                        //1.19
                 speeddown_released = true;    //1.20
            }
}
}
```

На рисунку 6.18 наведено стан моделі після запуску процесу моделювання.

ДПС починає обертатись за годинниковою стрілкою, тому що на вхід IN1 подається логічна одиниця, а на вхід IN1-логічний нуль. Керуючий ШІМ-сигнал, який відображає осцилограф, має коефіцієнт заповнення 75%, відповідає робочій програмі.

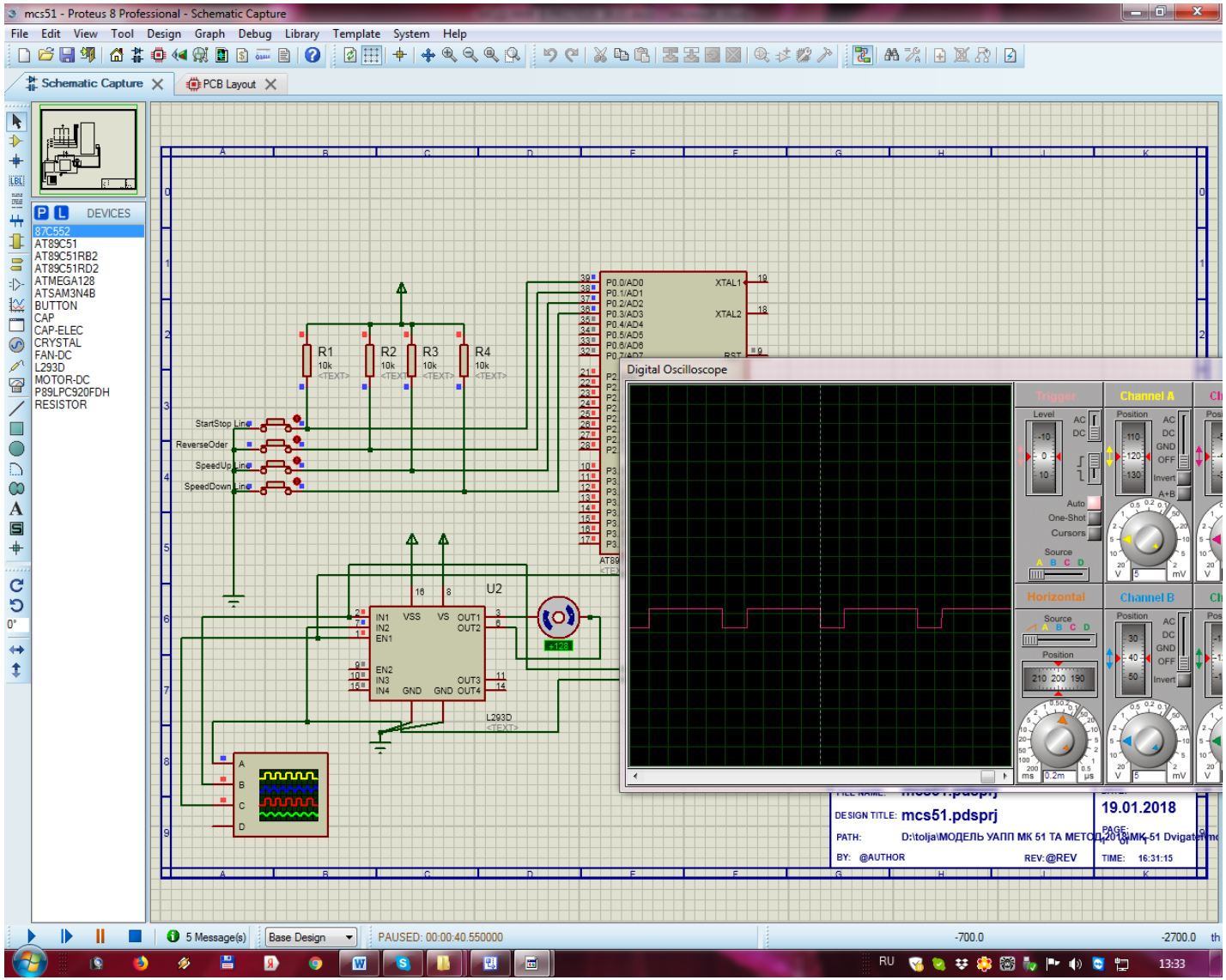

mcs51 - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
Schematic Capture
PCB Layout
DEVICES
87C552
AT89C51
AT89C51RB2
AT89C51RD2
ATMEGA128
ATSAM3N4B
BUTTON
CAP
CAP-ELEC
CRYSTAL
FAN-DC
L293D
MOTOR-DC
P89LPC920FDH
RESISTOR
R1
R2
R3
R4
10k
StartStop Line
ReverseOder
SpeedUp Line
SpeedDown Line
U2
L293D
Digital Oscilloscope
Trigger
Channel A
Channel B
Horizontal
DESIGN TITLE: mcs51.pdsprj
PATH: D:\tolja\МОДЕЛЬ УАПП МК 51 ТА МЕТОД
BY: @AUTHOR
REV:@REV
19.01.2018
TIME: 16:31:15
5 Message(s)
Base Design
PAUSED: 00:00:40.550000
-700.0
-2700.0
13:33

Рисунок 6.18

6.7.1 Запуск та зупинка двигуна

Щоб запустити двигун треба відпусти кнопку K1 (StartStop Line), яка відповідає за вмикання та вимикання двигуна постійного струму (рисунок 6.3). Щоб запобігти зайвим спрацюванням при відпусканні кнопки керування, введемо індикатор відпускання і будемо міняти керуючу програму вже після відпускання кнопки. Одразу після ініціалізації таймера, ШІМ – сигнал вже поступає на схему драйвера, проте на лініях IN1, IN2 драйвера (рисунок 6.4) високий рівень напруги, тому двигун стоїть. Після відпускання кнопки ми встановлюємо індикатор відпускання. Попередня умова спрацює тоді, коли прапорець `start_released` встановлений, а кнопку ми вже відпустили (на вхід P0.0 подається високий рівень напруги, рисунок 6.1). Після спрацювання цієї умови ми скидатимемо прапорець `start_released` та будемо інвертувати сигнал на лінії P1.1 мікроконтролера. Після цього двигун починає обертатися за годинниковою стрілкою. Коли кнопка K1 (StartStop Line) знов буде натиснута, на вхід P0.0 подається низький рівень напруги та прапорець `start_released` знов буде встановлюватись у одиницю.

Нижче приведено фрагмент коду на C, який все це реалізовує.

//Start– Stop button action

```
if (P0_0 && start_released) {  //1.5
     start_released = false;
     P1_1 ^= 1;                 //1.6
}
if (!P0_0){                    //1.7
     start_released = true;     //1.8
```

6.7.2 Зміна напрямку обертання двигуна постійного струму

Для зміни напрямку обертання двигуна постійного струму, або ввімкнення реверсу, потрібно слідкувати за станом кнопки керування K2 (ReverseOrder Line) та за її відпусканням (рисунок 6.3). Слідкувати за відпусканням кнопки будемо за допомогою зчитування рівня напруги з лінії порту, до якого підключена ця кнопка.

Власне, дану кнопку підключено до виводу P0.1 мікроконтролера AT89C51RD2, який використовуємо як вхід. При відпусканні кнопки рівень напруги буде високий, тобто зніматимемо 1, а коли кнопка натиснена у вихідному стані, то рівень напруги низький і зніматимемо 0.

Логіка керування буде наступною. Сторона, у яку обертається двигун залежить від потенціалів на входах IN1 та IN2 на драйвері, яким виступає схема з двома мостами L293D. Для зміни напрямку обертання двигуна будемо змінювати одразу потенціали, лінію з низьким потенціалом будемо робити лінією з високим, а лінію з високим – лінією з низьким. Як ми бачимо вище, при вмиканні моделі лінію порту P1.0 (IN1) було запрограмовано як вихід з високим рівнем напруги, а лінію P1.2 (IN2) як вихід з низьким рівнем напруги. Після натискання кнопки K2, за допомогою операції побітового XOR ми будемо інвертувати P1.0 та P1.1, які у якості ліній виведення мікроконтролера відповідають за рівень напруги на виході. Нижче наводимо фрагмент коду на C, який виконує вказані операції.

//Reverse button action

```
if (P0_1 && reverse_released) { //1.9
     reverse_released = false;
     P1_0 ^= 1;                  //1.10
     P1_1 ^= 1;
}
if (!P0_1){                      //1.11
     reverse_released = true;    //1.12
}
```

6.7.3 Зміна швидкості обертання двигуна постійного струму

Для зміни швидкості обертання двигуна постійного струму у нас буде дві кнопки, перша буде збільшувати швидкість, а друга – зменшувати. Принцип, за яким ми будемо змінювати швидкість, ґрунтується на властивості ШІМ– сигналу. Двигун до джерела живлення підключаємо через ключі, мостовою схемою, як показано на рисунку 6.11.

В даному випадку для нас важливі комбінації, коли є різниця потенціалів. Це наступні комбінації ключів: 1 відкритий, 2 закритий, 3 закритий, 4 відкритий; 1 закритий, 2 відкритий, 3 відкритий, 4 закритий.

Також важливими є комбінації, коли немає різниці потенціалів, і ротор електродвигуна не обертається. вони наступні: 1 відкритий, 2 відкритий, 3 закритий, 4 закритий; 1 закритий, 2 закритий, 3 відкритий, 4 відкритий.

Тобто, в загальному випадку, коли ми відкриваємо ключі за діагоналлю – наш двигун починає обертатись.

Для того, щоб регулювати швидкість обертання ми будемо вмикати ці ключі на час X та вимикати на час Y з великою швидкістю. Як це буде виглядати, представлено на рисунку 6.18

Швидкість обертання двигуна залежить від напруги, яка на нього подається. На рисунку 6.18 ця напруга позначається, як еквівалентна постійна напруга. З цього рисунка ми також бачимо, що використовуючи цифрові сигнали логічного “0” та логічної “1” ми можемо отримувати своєрідну еквівалентну напругу, яка відрізняється від напруги логічних рівнів.

Змінюючі шпаруватість – відношення періоду слідування імпульсів до їх довжини, ми регулюємо швидкість обертання нашого двигуна постійного струму.

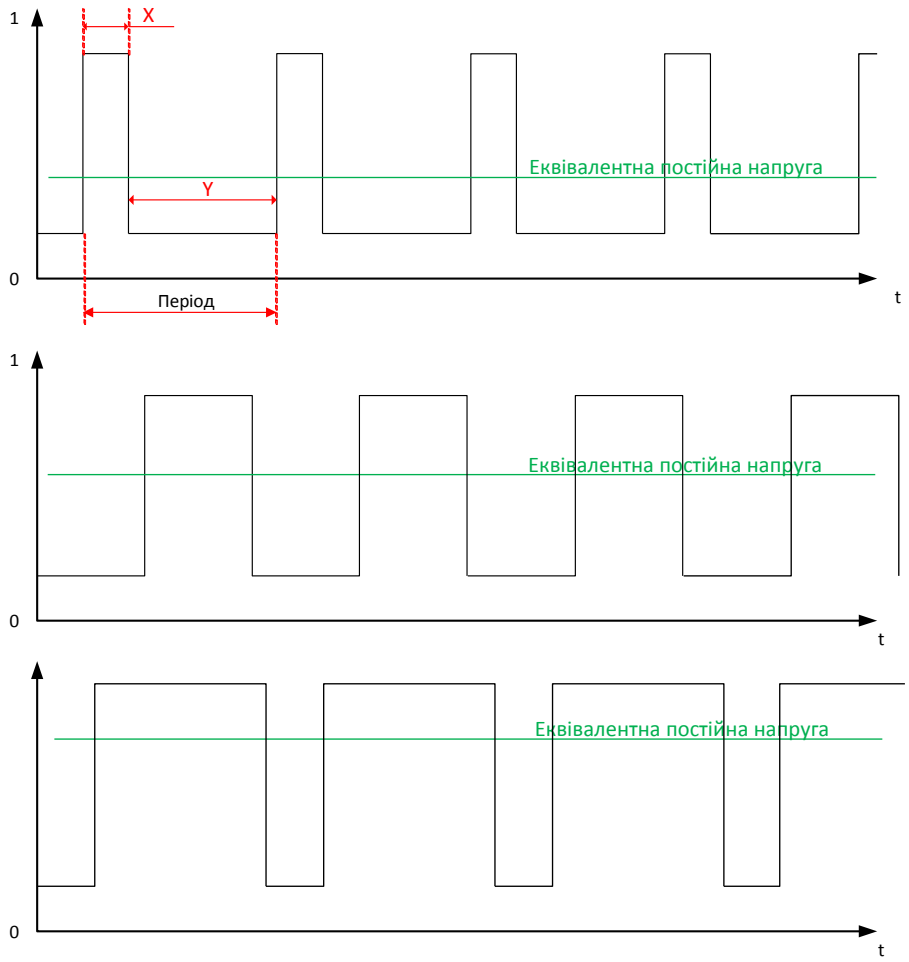


Рисунок 6.18 – Еквівалентна постійна напруга при ШІМ– сигналі

Тепер власне перейдемо до кнопок. Спочатку розберемо схему збільшення швидкості. Кнопка підключена до виводу PB4 мікроконтролера, який в свою чергу запрограмовано як вхід. Знімати значення цього входу ми будемо за допомогою 8 бітного регістра PINB, а точніше – за допомогою біта PB4. Коли кнопка не натиснута цей біт має значення логічної 1. Коли кнопка натискається, цей біт приймає значення логічного 0. Саме це ми будемо перевіряти у програмі керування мікроконтролером ATmega128. При виявленні логічного нуля ми будемо встановлювати індикатор натискання та очікувати 10 мс. Наступна умова у нас виконається, якщо індикатор натискання встановлений в 1, і на защіпці PB4 порту логічна одиниця, тобто кнопка вже відпущена. У цьому випадку ми збільшуватимемо значення 8 бітного регістра OCR1A на величену, яку записано у змінну delay, але не більше, ніж 20.

При цій постійній еквівалентній напрузі наш двигун постійного струму обертається з максимальною швидкістю, яку ми можемо досягти у нашому моделюванні. Нижче наведемо код на С, який відповідає за збільшення швидкості обертання двигуна постійного струму при натисканні на відповідну кнопку.

```
//Speed + button action
 if(!(PINB&16))
 { flag_speedup=1; _delay_ms(10); }
 if(( flag_speedup==1 )&&(PINB&16))
 { if (OCR1A!=40) OCR1A+=delay; flag_speedup=0; }
```

Зменшення швидкості обертання двигуна постійного струму відбувається за схожим принципом. Ми перевіряємо рівень напруги на виводі PB6 який запрограмовано на вхід. Програмно читаємо цей рівень напруги з біта PX6 регістра PINB. Коли кнопка не натиснена, біт дорівнює 1. Коли кнопка натискається, значення біта змінюється з 1 на 0, і залишається таким до того часу, доки кнопка натиснена. Саме для того, щоб не відбувались багаторазові зміни при натисканні на кнопку, у програмі ми використовуємо ключі, які скидаються лише коли кнопки відпускаються.

Програмно читаючи значення біта PX6 регістра PINB, ми змінюємо вміст регістра OCR1A, зменшуючи його на величену, яку записано у змінну delay, але не менше 0. Код, який це реалізовує приведено нижче.

```
//Speed -  button action
   if(!(PINB&64))
   { flag_speeddown=1; _delay_ms(10); }
   if(( flag_speeddown==1 )&&(PINB&64))
   {
      if (OCR1A!=0)OCR1A- =delay;
      flag_speeddown=0;
   }
```

7 МОДЕЛЮВАННЯ ЦИФРОВОГО ВОЛЬТМЕТРА

7.1 Особливості архітектури модуля АЦП у складі мікроконтролера ATmega32

Основним елементом цифрового вольтметра є аналого–цифровий перетворювач, в якості якого використовується відповідний модуль мікроконтролера ATmega32. Архітектуру цього модуля цього модуля розглянуто у підрозділах 3.1…3.9.

7.2 Опис моделі

Робочу модель цифрового вольтметра показано на рисунку 7.1.

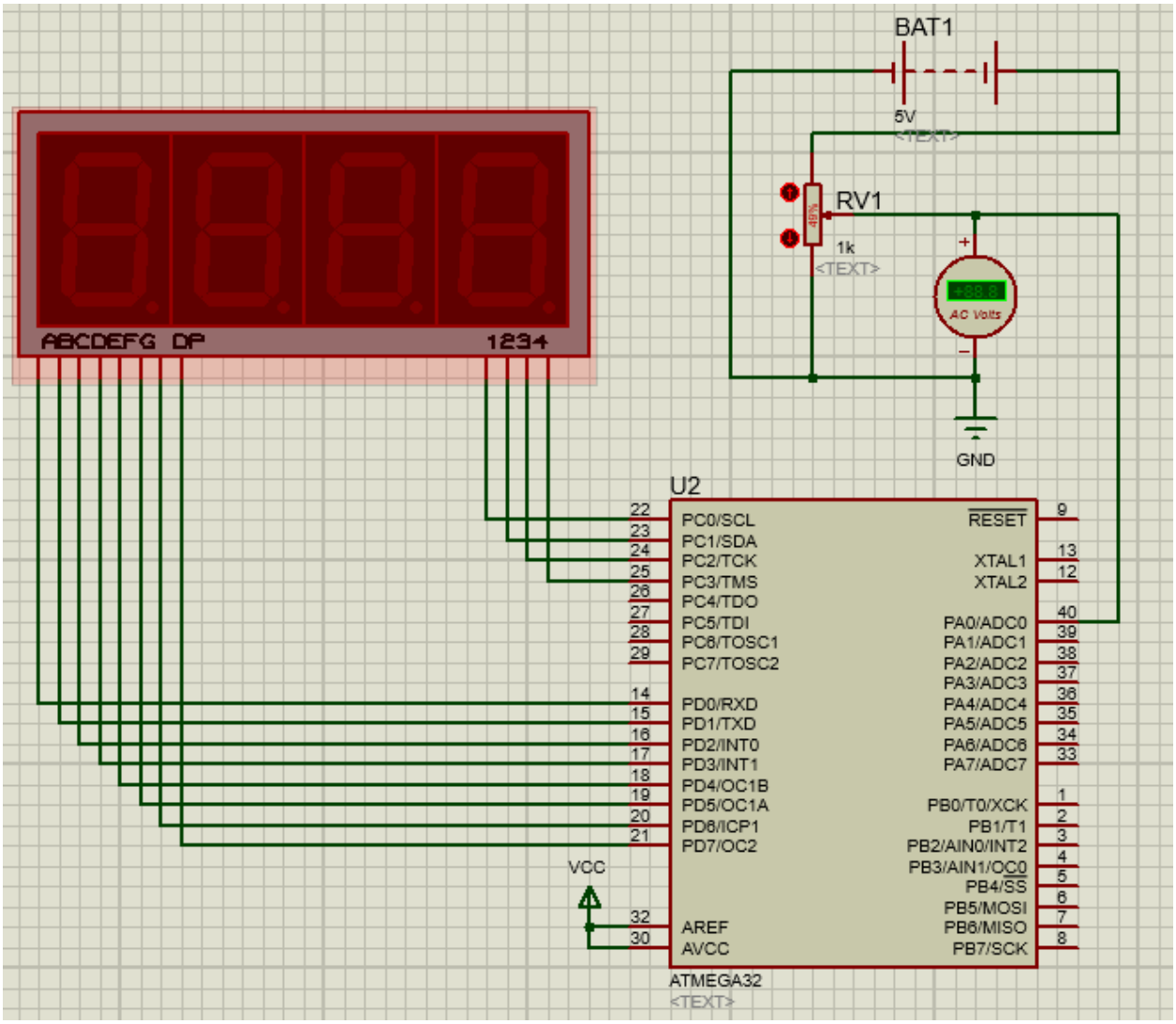


Рисунок 7.1 – Схема моделі цифрового вольтметра

З лівого боку рисунка 7.1 ми бачимо 7–сегментний чотирьохпозиційний цифровий дисплей, який з'єднаний з портами C та D мікроконтролера ATMega32 відповідними лініями зв'язку. У верхній частині рисунку знаходиться батарея BAT1 та резистор RV1, напругу з виходу якого можна змінювати у режимі реального часу. На початковому етапі вхідна напруга подається на порт PA0 мікроконтролера ATMega32. Після перетворення її значення у цифровий еквівалент, розрахована величина подається у порти мікроконтролера у відповідному цифровому коді. Побудована схема реалізує аналого–цифрове перетворення з точністю до сотої частини значення вхідної напруги за допомогою АЦП, що є у складі ATmega32.

Оскільки ми використовуємо 7–сегментний дисплей, то для отримання на ньому чисел нам потрібно керувати сегментами індикатора A, B, C, D, E, F, G та точкою Dp згідно рисунка 7.2 та таблиць 7.1, 7.2.

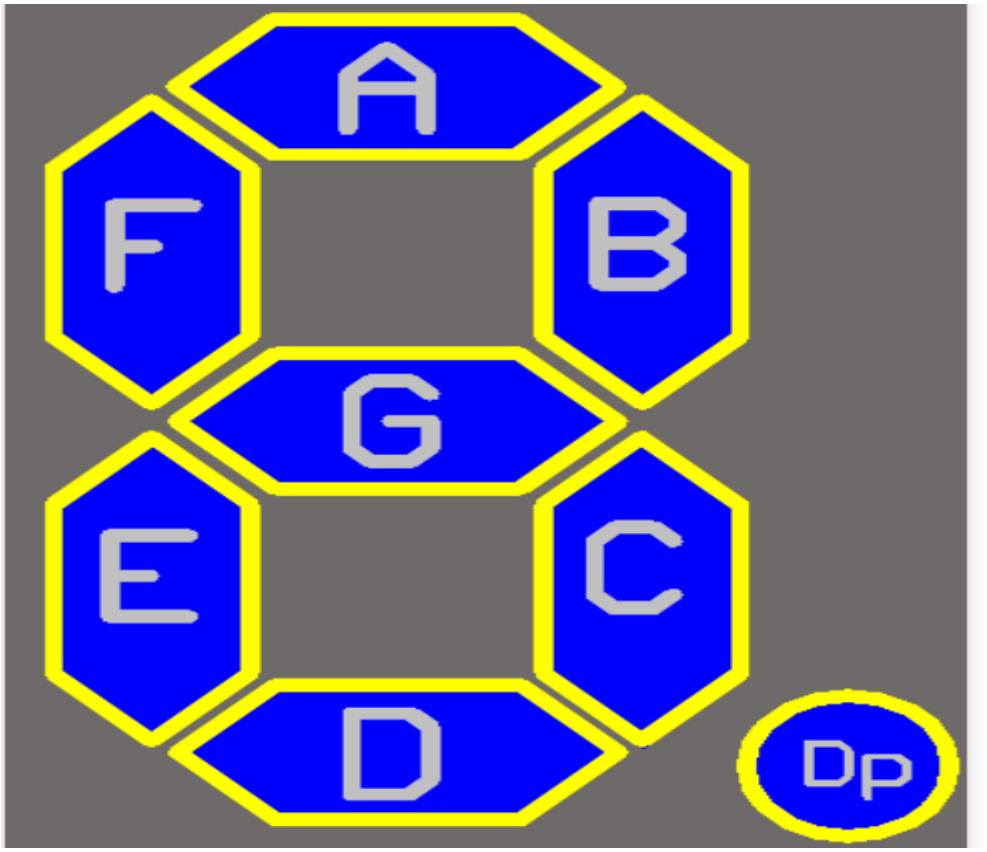


Рисунок 7.2 – 7–сегментний дисплей

Таблиця 7.1 – Зв’язок між цифрами на дисплеї та значенням керуючих сигналів

Цифра на дисплеї	Dp	G	F	E	D	C	B	A	Значення керуючих сигналів
0	0	0	1	1	1	1	1	1	0x3F
1	0	0	0	0	0	1	1	0	0x06
2	0	1	0	1	1	0	1	1	0x5B
3	0	1	0	0	1	1	1	1	0x4F
4	0	1	1	0	0	1	1	0	0x66
5	0	1	1	0	1	1	0	1	0x6D
6	0	1	1	1	1	1	0	1	0x7D
7	0	0	0	0	0	1	1	1	0x07
8	0	1	1	1	1	1	1	1	0x7F
9	0	1	1	0	0	1	1	1	0x6F

Таблиця 7.2 – Зв’язок між цифрами з точкою на дисплеї та значенням керуючих сигналів

Цифра на дисплеї	Dp	G	F	E	D	C	B	A	Значення керуючих сигналів
0	1	0	1	1	1	1	1	1	0xBF
1	1	0	0	0	0	1	1	0	0x86
2	1	1	0	1	1	0	1	1	0xDB
3	1	1	0	0	1	1	1	1	0xCF
4	1	1	1	0	0	1	1	0	0xE6
5	1	1	1	0	1	1	0	1	0xED
6	1	1	1	1	1	1	0	1	0xFD
7	1	0	0	0	0	1	1	1	0x87
8	1	1	1	1	1	1	1	1	0xFF
9	1	1	1	0	0	1	1	1	0xEF

Робота моделі цифрового вольтметра в більшості співпадає з роботою моделі АЦП (див. 3.14). Відмінність полягає в тому, що модель АЦП значення вхідної напруги відображає на індикаторі у десятковому коді, а модель цифрового вольтметра відображає результат перетворення у вольтах з точністю до сотих долей вольта. Нижче на рисунках 7.3, 7,4 наведено декілька прикладів роботи моделі.

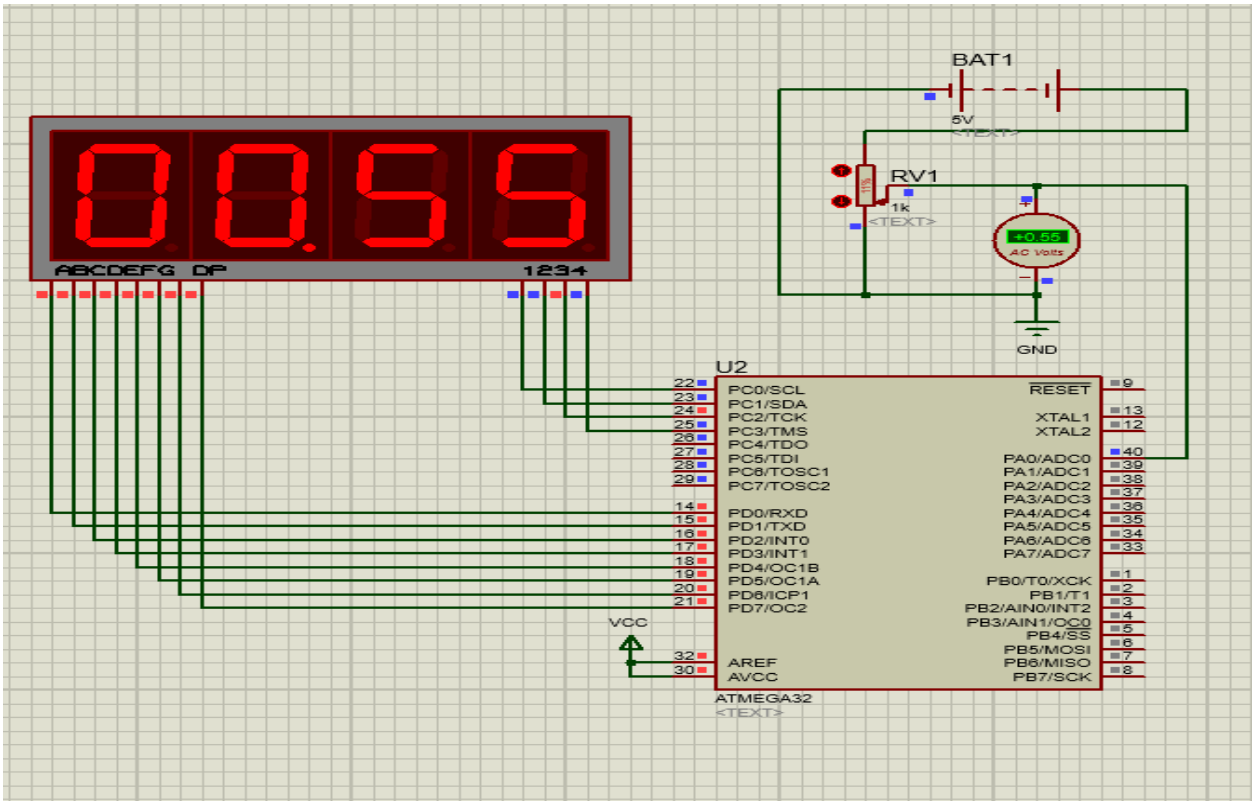


Рисунок 7.3 – Робота цифрового вольтметра при U_{BX}=0,55В

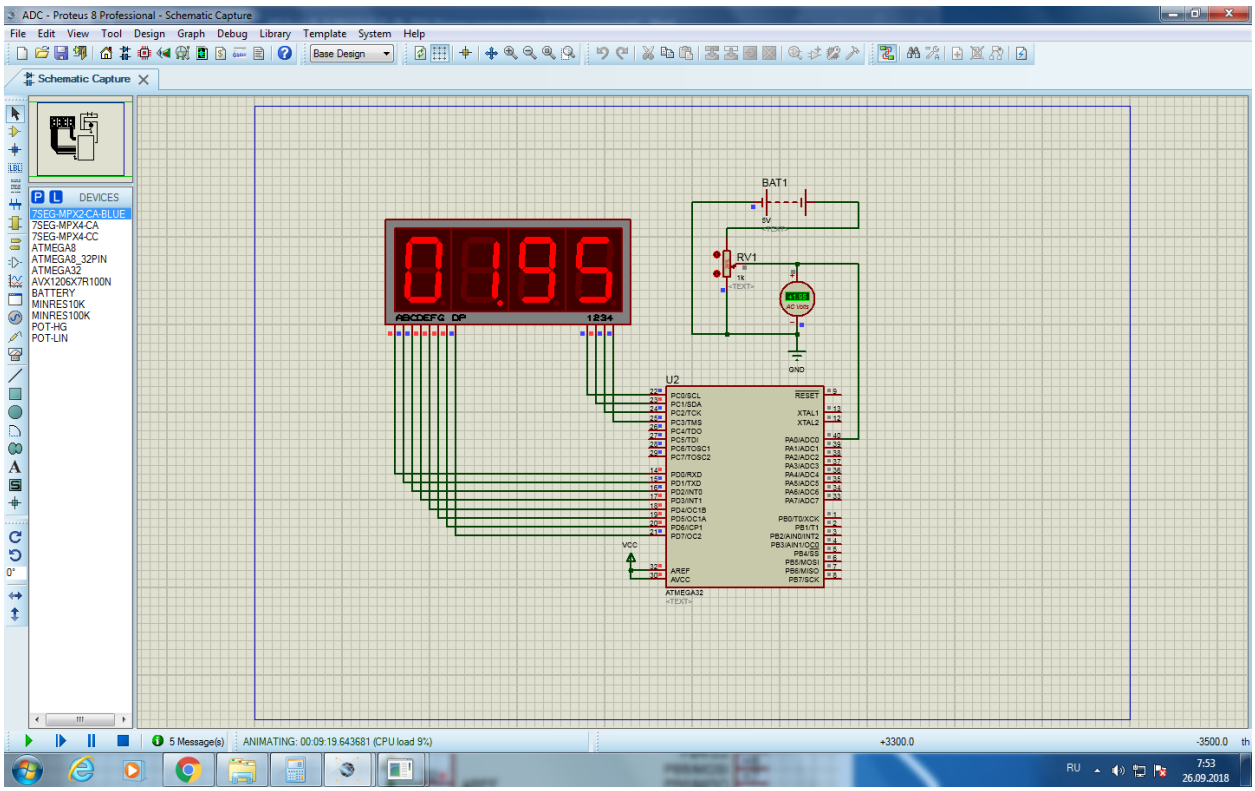


Рисунок 7.4 – Робота цифрового вольтметра при U_{BX}=1,95В

7.3 Програмна реалізація моделі мовою програмування С

7.3.1 Схема алгоритму роботи модуля

Схему алгоритму роботи модуля наведено на рисунках 7.5…7.8.

Для налаштування параметрів АЦП виконуються такі дії: в регістрі ADCSRA встановлюємо в лог.1 біти ADEN – дозвіл АЦП, ADSC – запуск перетворення, ADATE – безерервний режим роботи АЦП, ADPS2 і ADPS1 – переддільник на 64, ADIE – дозвіл переривань від АЦП; в регістрі ADMUX скидаємо в лог. нуль біти REFS1 і REFS0 щоб обрати зовнішнє джерело опорної напруги.

На рисунку 7.5 наведено схему загального алгоритму роботи модуля.

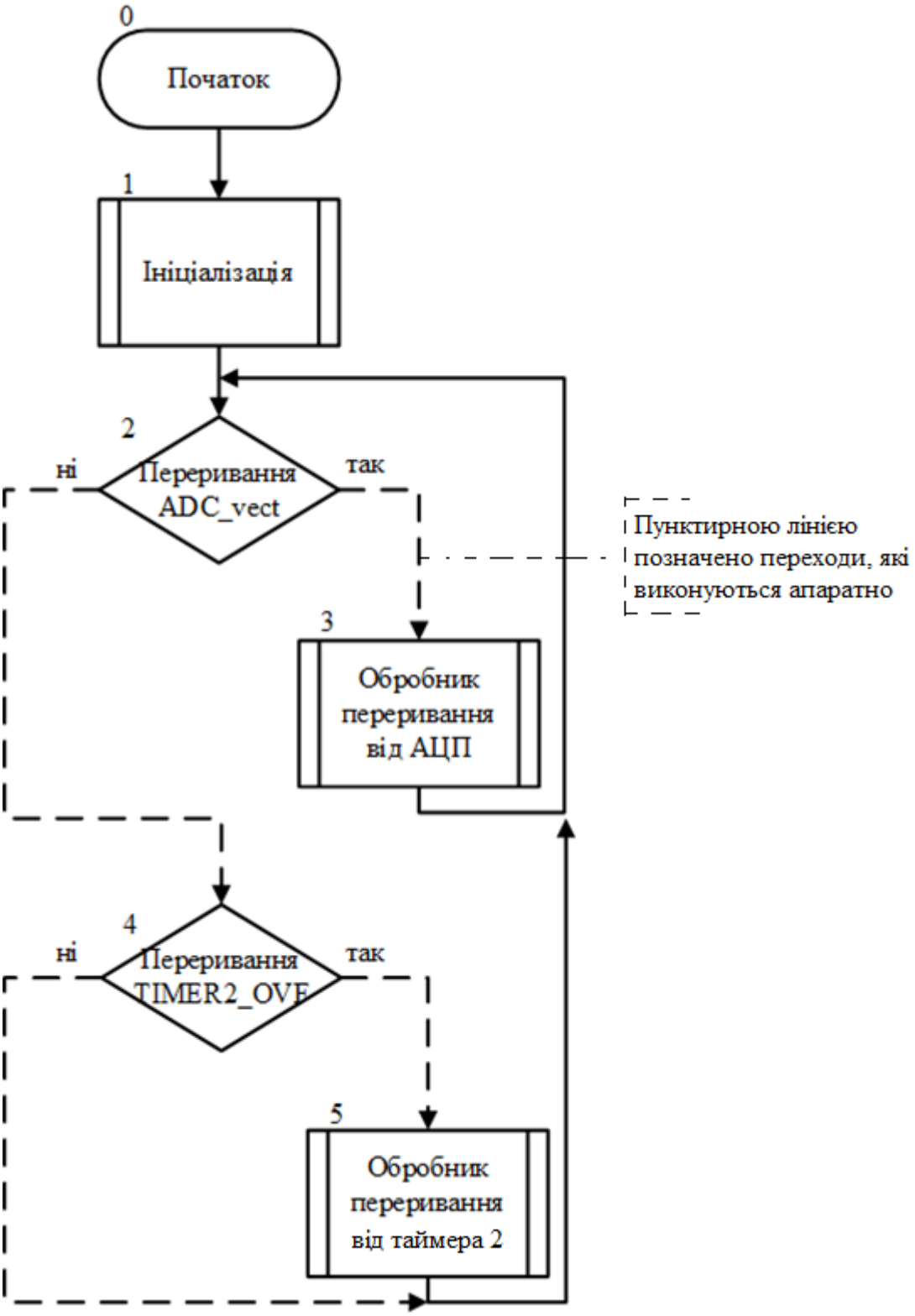


Рисунок 7.5 – Схема загального алгоритму роботи модуля

На рисунку 7.6 наведено схему алгоритму ініціалізації.

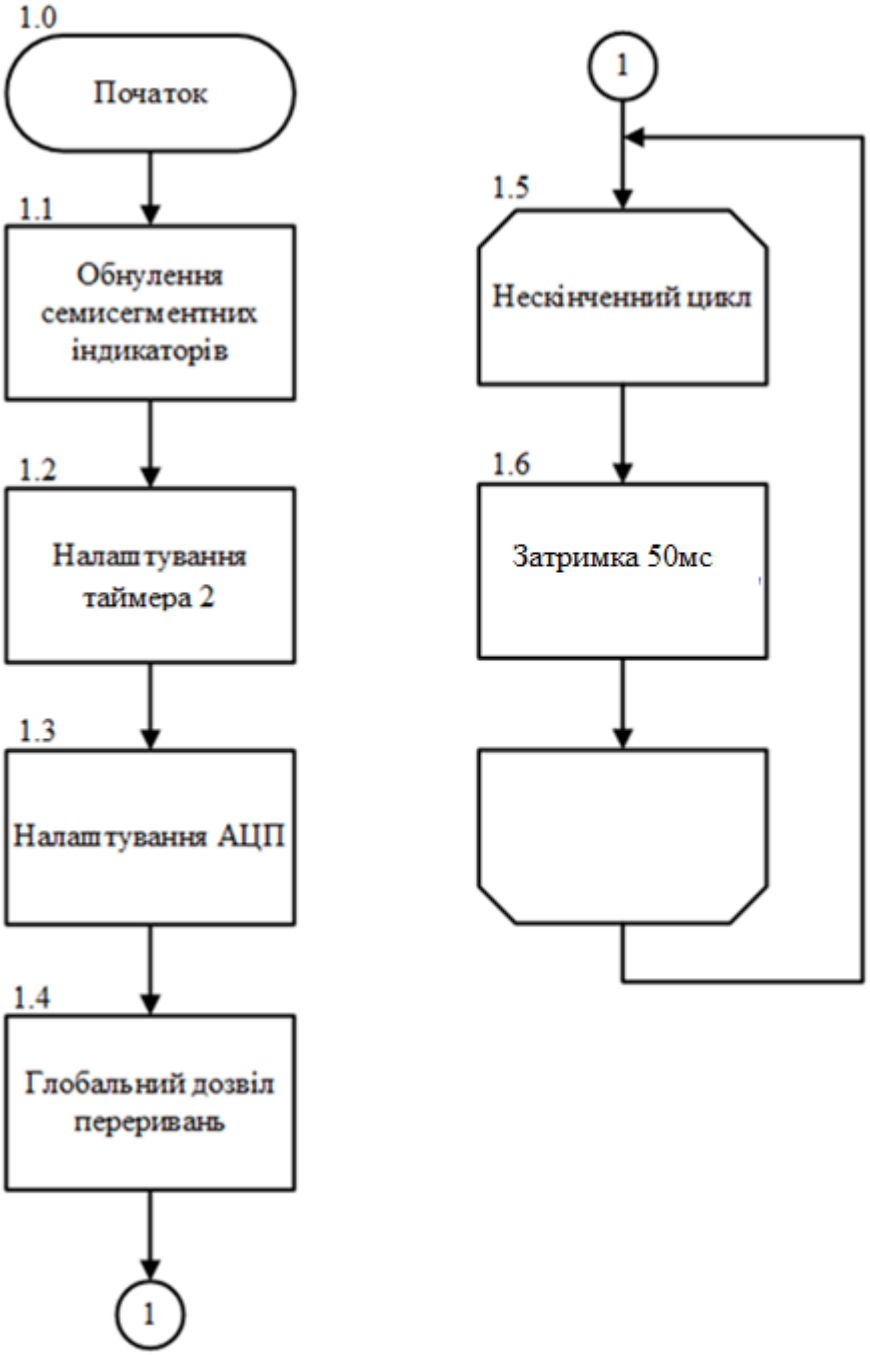


Рисунок 7.6 – Схема алгоритму ініціалізації

На рисунку 7.7 наведено схему алгоритму обробки переривання таймера 2.

Рисунок 7.7 – Схема алгоритму обробки переривання таймера 2

На рисунку 7.8 наведено схему алгоритму обробки переривання завершення перетворення АЦП.

Рисунок 7.8 – Схема алгоритму обробки переривання АЦП

7.3.2 Робоча програма мовою С

Нижче наведено програму для розглянутого у роботі прикладу: 10–розрядний АЦП, максимальна вхідна напруга якого $U_{\text{вх. } max} = 5\text{В}$.

```
// Підключення заголовних файлів бібліотек
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

// Масив значень для відображення цифр на семисегментних індикаторах

char SEG[] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F,
0x6F};
char SEG_with_dot[] = {0xBF, 0x86, 0xDB, 0xCF, 0xE6, 0xED, 0xFD, 0x87,
0xFF, 0xEF};


// Ініціалізації глобальних змінних
volatile unsigned char current_indicator = 0;
volatile unsigned int display = 0;
volatile unsigned int ADC_value;

// Блок 1 – Ініціалізація програми
int main (void)
{
    // Блок 1.1 – Обнулення портів семисегментних індикаторів
    DDRC = 0xFF;
    PORTC = 0x00;
    DDRD = 0xFF;
    PORTD = 0x00;

    // Блок 1.2 – Налаштування Таймера 2
    TIMSK |= (1 << TOIE2); // Дозвіл переривання по таймеру 2
    TCCR2 |= (1 << CS21);  // Переддільник на 8

    // Блок 1.3 – Налаштування АЦП
    ADCSRA |= (1 << ADEN) // Дозвіл АЦП
    |(1 << ADSC) // Запуск перетворення
    |(1 << ADATE) // Безперервний режим роботи АЦП
    |(1 << ADPS2)|(1 << ADPS1) // Переддільник на 64
    |(1 << ADIE); // Дозвіл переривань від АЦП

    ADMUX &= (~(1 << REFS1))&(~(1 << REFS0)); // Зовнішнє ДОН

    // Блок 1.4 – Глобальний дозвіл переривань
    sei();

    // Блок 1.5 – Основний цикл
    while(1)
```

```
    {
        _delay_ms(50); // Затримка 50 мс
    }
}

// Блок 2 - Умова переривання від АЦП
ISR (ADC_vect)
{// Блок 3 - Обробник перериванння від АЦП
    ADC_value = ADC; // Блок 3.1 - Присвоюємо глобальній змінній
поточне значення АЦП
}

// Блок 4 - Умова переривання від таймера Т2
ISR (TIMER2_OVF_vect)
{// Блок 5 - Обробник переривань від таймера Т2
    PORTD = 0xFF; // Блок 5.1 - Вимикаємо всі сегменти
    PORTC = (1 << current_indicator); // Блок 5.2 - Обираємо поточний
індикатор
    display = (ADC_value+1)*(5/1024.0)*100; // Обраховуємо значення
напруги
    //Блок 5.3
    switch (current_indicator)
    {
        case 0:
        PORTD = ~(SEG[display % 10000 / 1000]); // Вмикаємо цифру
десятків
        break;
        case 1:
        PORTD = ~(SEG_with_dot[display % 1000 / 100]); // Вмикаємо
цифру одиниць
        break;
        case 2:
        PORTD = ~(SEG[display % 100 / 10]); // Вмикаємо цифру десятих
        break;
        case 3:
        PORTD = ~(SEG[display % 10 / 1]); // Вмикаємо цифру сотих
        break;
    }
    if ((current_indicator++) > 2) // Блок 5.4 - Переходимо на
наступний індикатор
        current_indicator = 0; // Блок 5.5 - Обнуляємо, якщо він за
межами
}
```

Нижче наведено пояснення перетворення значення ADC, у десятковому коді яке отримається після завершення аналого-цифрового перетворення, у значення вхідної напруги у вольтах.

Як відмічено у підрозділі 3.9 для каналів з однополярним (несиметричним) входом результат АЦП перетворення визначається виразом: ADC = 1023•U_{IN}/U_{REF},

де U_{IN} – значення вхідної напруги, U_{REF} – величина опорної напруги, ADC – десятковий еквівалент двійкового коду на виході АЦП.

Коефіцієнт передачі АЦП

$$K_{\text{пер}} = \frac{\text{ADC}}{U_{IN}} = \frac{1023 \cdot U_{IN}}{U_{REF} \cdot U_{IN}} = \frac{1023}{U_{REF}}.$$

При U_{REF} = 5В

$$K_{\text{пер}} = \frac{1023}{5} = 0.2046 \left[\frac{\text{МЗР}}{\text{мВ}}\right],$$

Тоді значення вхідної напруги

$$U_{\text{IN}} = \frac{ADC}{K_{\text{пер}}} = \frac{ADC \cdot 5}{1023}.$$

Наприклад, якщо на вхід АЦП було подано напругу 0,55В, тоді згідно з моделюванням АЦП (див. розділ 3) ADC=0.

Тоді після перетворювання ADC в U_{IN} отримаємо

$$U_{\text{IN}} = \frac{0113 \cdot 5}{1023} = 0{,}55\text{В}.$$

Для того, щоб використовувати лише цілі значення для обчислення, значення напруги, отримане з формули вище, множимо на 100:

```
display = ADC_value*(5/1023)*100.
```

Наприклад, при $U_{\text{вхАЦП}}$=5В значення ADC згідно з роботою моделі в Proteus 8.6 дорівнює U_{IN} =0,55*100=55. Це значення програма обробляє наступним чином:

1) 55%10000/1000=55/1000=0,055. Цифра *0* виводиться на перший індикатор.
2) Далі відбуваються наступні дії:
   на другий індикатор завжди виводиться число з крапкою
   55%1000/100=55/100=0,55. Цифра *0 десятковою крапкою* виводиться на другий індикатор;
3) 55%100/10=55/10=5,5. Цифра *5* виводиться на третій індикатор;
4) 55%10/1=5/1=5. Цифра *5* виводиться на четвертий індикатор.

Наприклад, ADC=1,95*100=195, тоді:

195%10000/1000=195/1000=0,195. Цифра 0 виводиться на перший індикатор.

195%1000/100=195/100=1,95. Цифра 1 десятковою крапкою виводиться на другий індикатор;

195%100/10=95/10=9,5. Цифра 9 виводиться на третій індикатор;

195%10/1=5/1=5. Цифра 5 виводиться на четвертий індикатор.

8 МОДЕЛЮВАННЯ МОДУЛЯ УНІВЕРСАЛЬНОГО АСИНХРОННОГО ПРИЙМАЧА–ПЕРЕДАВАЧА СІМ`Ї AVR

8.1 Опис моделі

Робочу модель дослідження універсального асинхронного приймача–передавача (УАПП) наведено на рисунку 8.1.

Розберемо цю модель докладніше. Зліва ми бачимо вісім кнопок, за допомогою яких можна задавати байт (8 біт) даних, який ми будемо передавати через модуль УАПП.

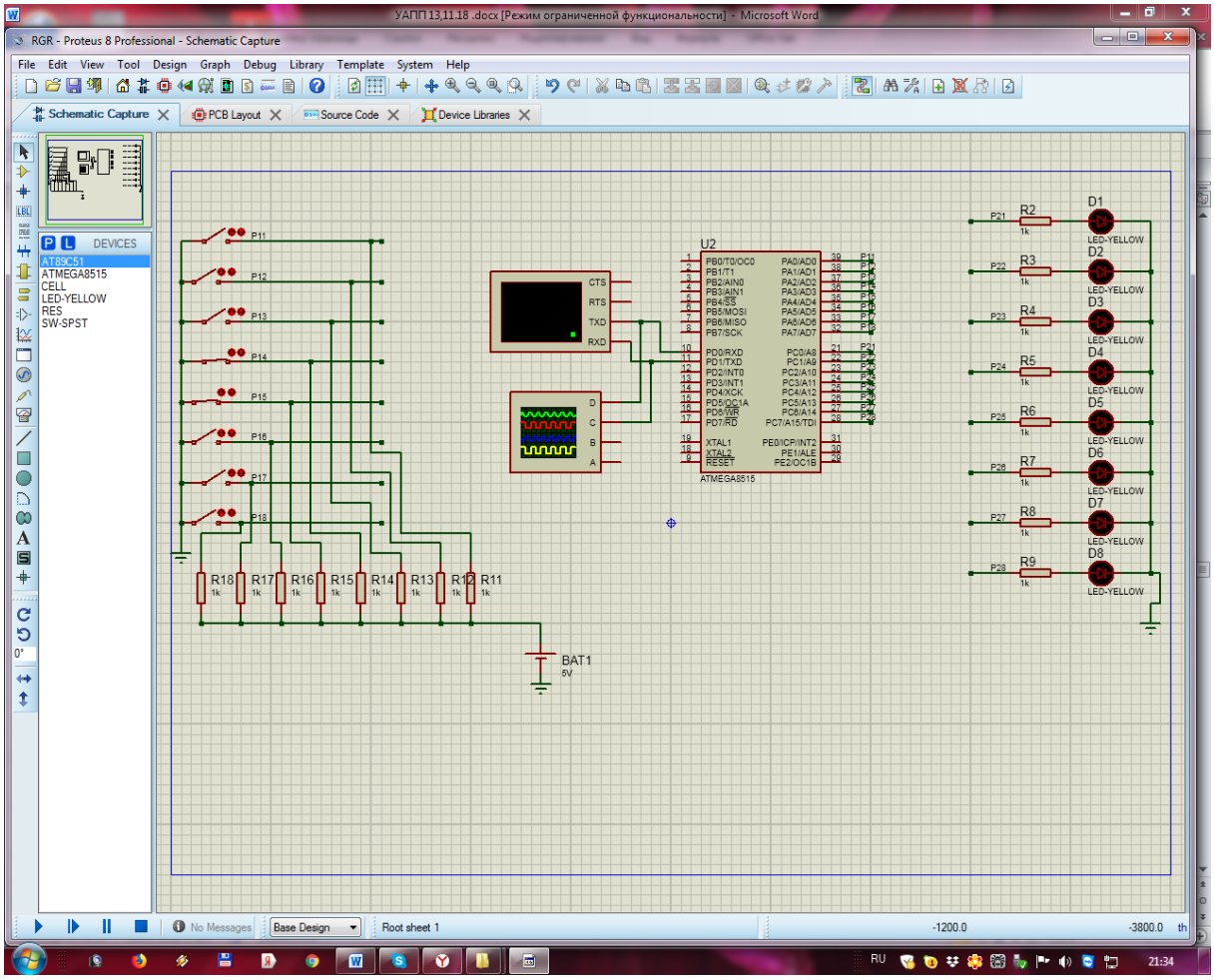


Рисунок 8.1 – Схема робочої моделі

Кнопки підключено до восьми ліній порту PA мікроконтролера: PA.0, PA.1, … , PA.7. Якщо якась із кнопок відпущена, то через резистори R11...R18 на відповідну лінію порта PA подається логічна 1. Якщо кнопка нажата, то вводиться

логічний нуль. Праворуч зображено вісім світлодіодів, які підключено до ліній порта PC: PC.0, PC.1, … , PC.7. Катоди світлодіодів підключено до спільного провода (землі), а на аноди через резистори R2 … R9, які обмежують струм, із виходів порту PC подаються логічні одиниці, коли треба засвітити відповідний світлодіод.

До ліній TxD–вихід передавача УАПП та RxD–вхід приймача підключено осцилограф, та Virtual Terminal, який вибрано з віртуальних інструментів програми Proteus. На них ми будемо відображати повідомлення, які вводиться та виводяться через модуль УАПП в/з мікроконтролера.

До виводів XTAL1 та XTAL2 підключають кварцовий резонатор частотою, яка визначає тактову частоту генератора мікроконтролера. В нашому прикладі вона дорівнює f_{BQ}=8МГц. Також підключають конденсатори C1 та C2, ємністю 30пФ, які призначені для підвищення стабільності роботи системного генератора. Резонатор та конденсатори потрібні в практичній схемі. В модель Proteus їх можна не вводити. Для задання частоти треба двічі лівою кнопкою миші клацнути на мікроконтролері та в опції Clock Frequency вказати значення частоти.

В якості мікроконтролера використано мікросхему ATmega8515. Мікроконтролер побудований за процесорною архітектурою AVR, тобто він вміє виконувати асемблерні команди, які описано цим стандартом.

8.2 Порядок моделювання

8.2.1 Перш за все, потрібно пересвідчитись, що Terminal правильно налаштовано. Для цього треба двічі клацнути на нього лівою кнопкою миші, та виставити відповідні налаштування. Нижче наведено приклад для випадку, коли потрібно передавати зі швидкістю 9600 біт/сек: 8 біт даних та один стоп–біт з перевірки на не парність (рисунок 8.2).

Якщо треба використовувати перевірку на парність або непарність, то це треба вказати в опції Parity (рисунок 8.2).

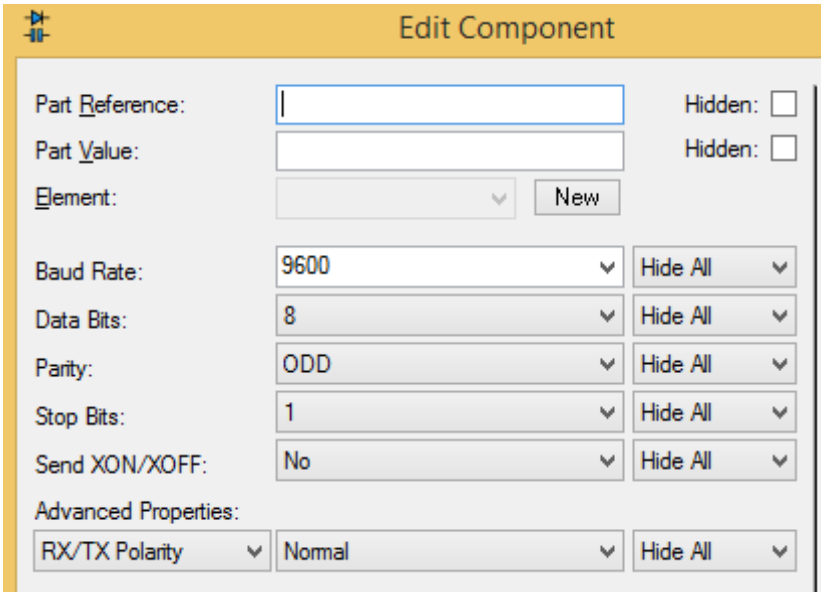


Рисунок 8.2

8.2.2 Далі натискаємо Start VSM Debugging (рисунок 8.3).

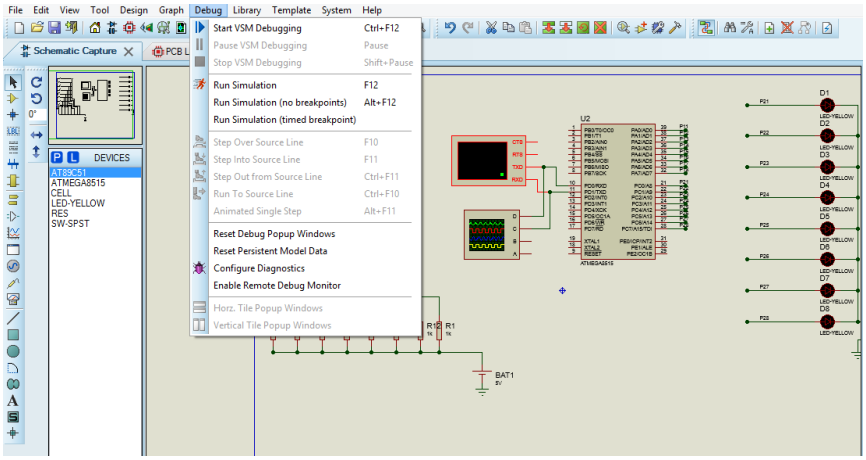


Рисунок 8.3

8.2.3 Потім натискаємо на кнопку Pause VSM Debugging

8.2.4 Переходимо на вкладку Source Code та ставимо дві точки зупинки. Для цього потрібно лівою кнопкою миші двічі клацнути з лівої сторони двох команд, як показано на рисунку 8.4.

8.2.5 Натиснемо Run Simulation (F12).

8.2.6 Перейдемо до вікна Virtual Terminal, натиснемо правою кнопкою миші на вікні, та виберемо Hex Display Mode та Echo Typed Characters (рисунок 8.5).

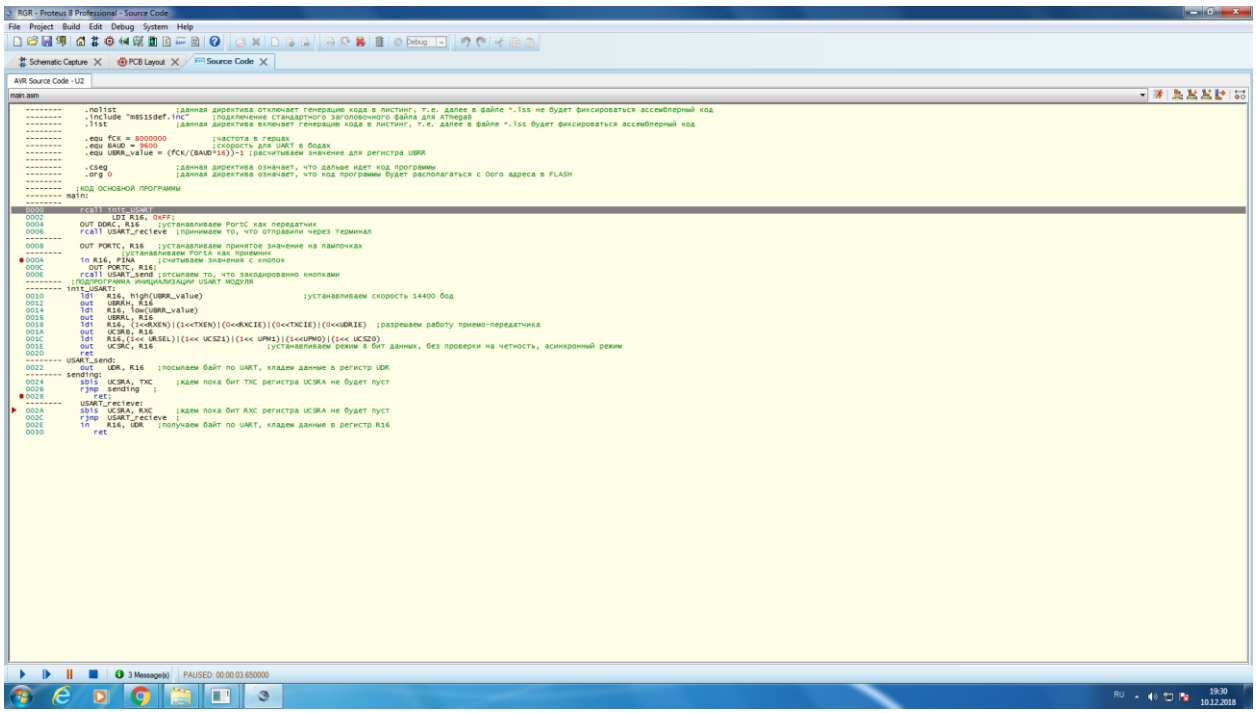


Рисунок 8.4

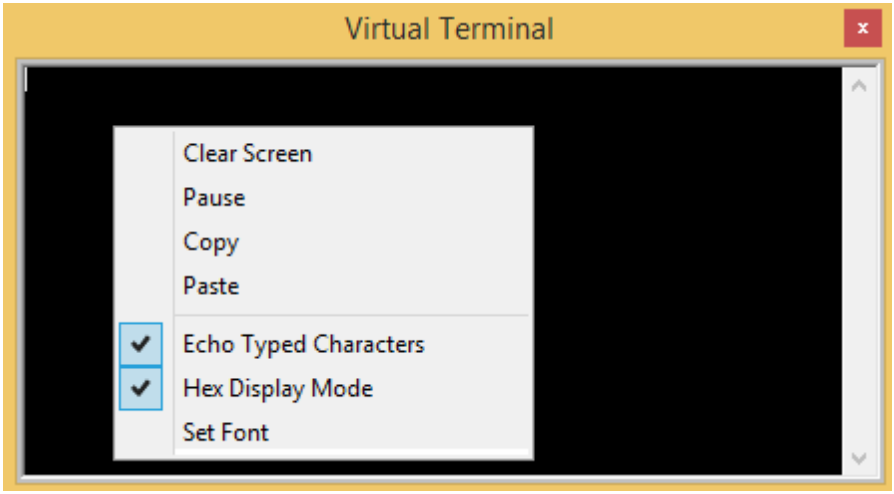


Рисунок 8.5

Якщо вікно Virtual Terminal закрито, то його можна відкрити із вкладки Debug.

8.2.7 Далі введемо якийсь символ, котрий хочемо надіслати від віртуального терміналу до мікроконтролера через УАПП. При введенні символу у вікні терміналу повинен стояти курсор. Символ, який ми вводимо з клавіатури, віртуальний термінал перетворює в ASCII–код згідно таблиці, яку наведено у [4].

Введемо, наприклад, цифру 7, ASCII код якої 37 (hex) = 0011 0111 (bin). Програма перейде на рядок, де ми поставили першу точку зупину. Натиснемо F10,

зробимо один наступний крок програми, та перейдемо на вкладку зі схемою Schematic Capture. Звернемо увагу на світлодіоди та на індикатори, які на виході порту С (рисунок 8.6).

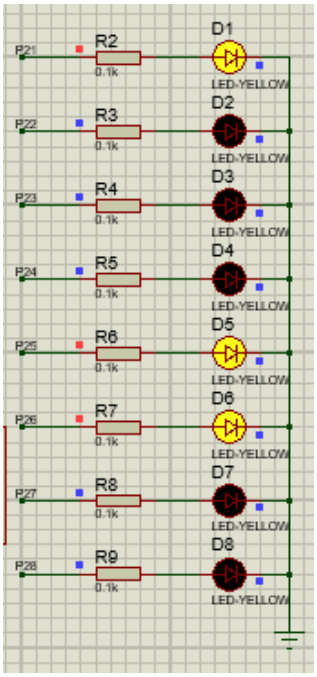


Рисунок 8.6

Світлодіоди та індикатори точно відображають бінарний код символу, який ми відправили (якщо дивитися знизу уверх, а бінарне число читати зліва на право), нуль – лампочка вимкнена, один – лампочка увімкнена (прийнято символ 00110111в = 37h).

8.2.8 Далі перевіримо як мікроконтролер через УАПП відправить символ, який ми закодували кнопками: замкнута кнопка – нуль, а розімкнута – одиниця (рисунок 8.7).

Якщо дивитися на кнопки знизу уверх, то на рисунку 8.7 набрано число 1110 0111 (bin) = E7 (hex).

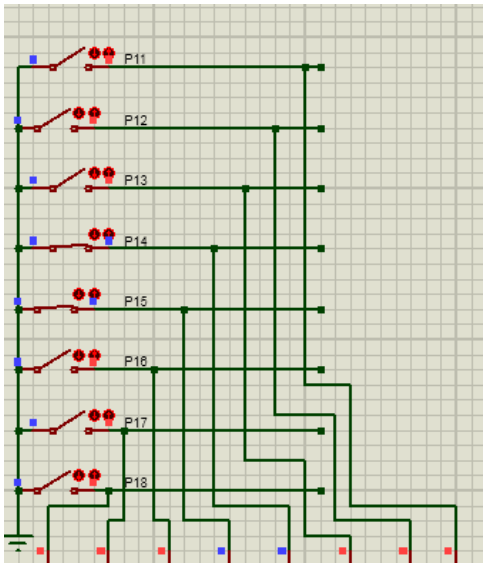


Рисунок 8.7

8.2.9 Тепер подивимося на вікно осцилографа. Для того, щоб побачити переданий сигнал на осцилографі, потрібно натиснути кнопку на опції One –Shot (один кадр), як показано на рисунку 8.8.

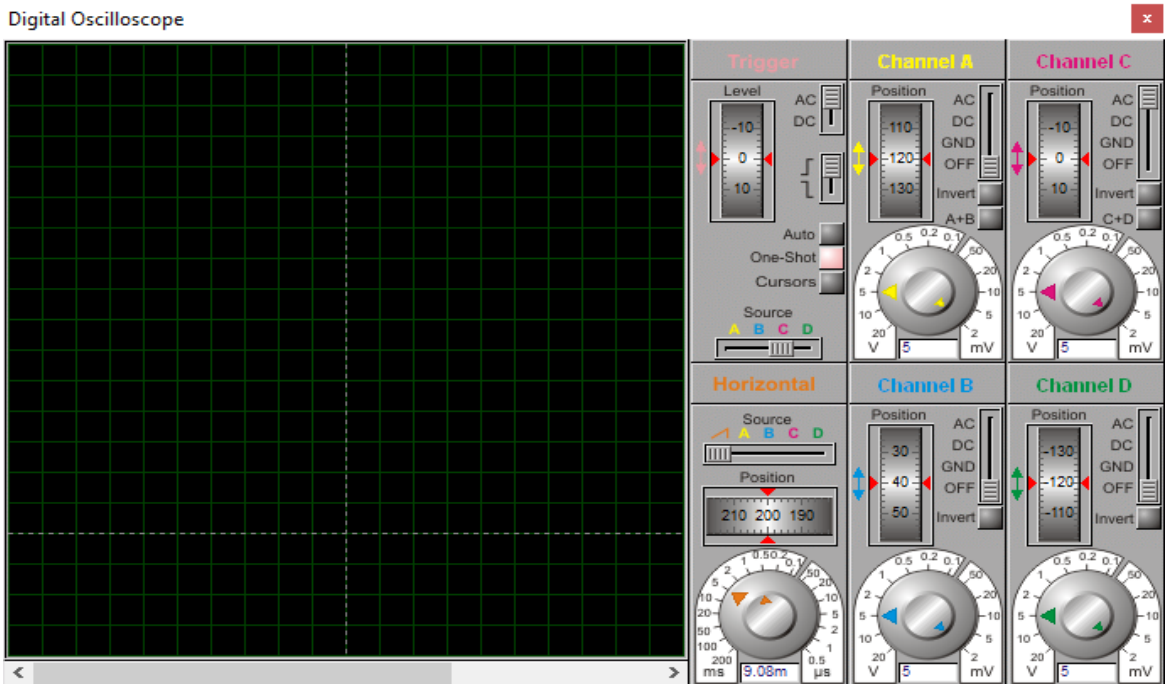


Рисунок 8.8

8.2.10 Натиснемо Run Simulation для продовження виконання програми (рисунок 8.9). Програма дійде до другої точки зупину.

8.2.11 Розберемо більш детально сигнали, що відображаються на осцилографі (рисунок 8.9).

Осцилограф відображає сигнали за двома промінями: верхній (червоний) – сигнал, який передається від кнопок, нижній (зелений) – сигнал, який приймається від терміналу: 37(hex).

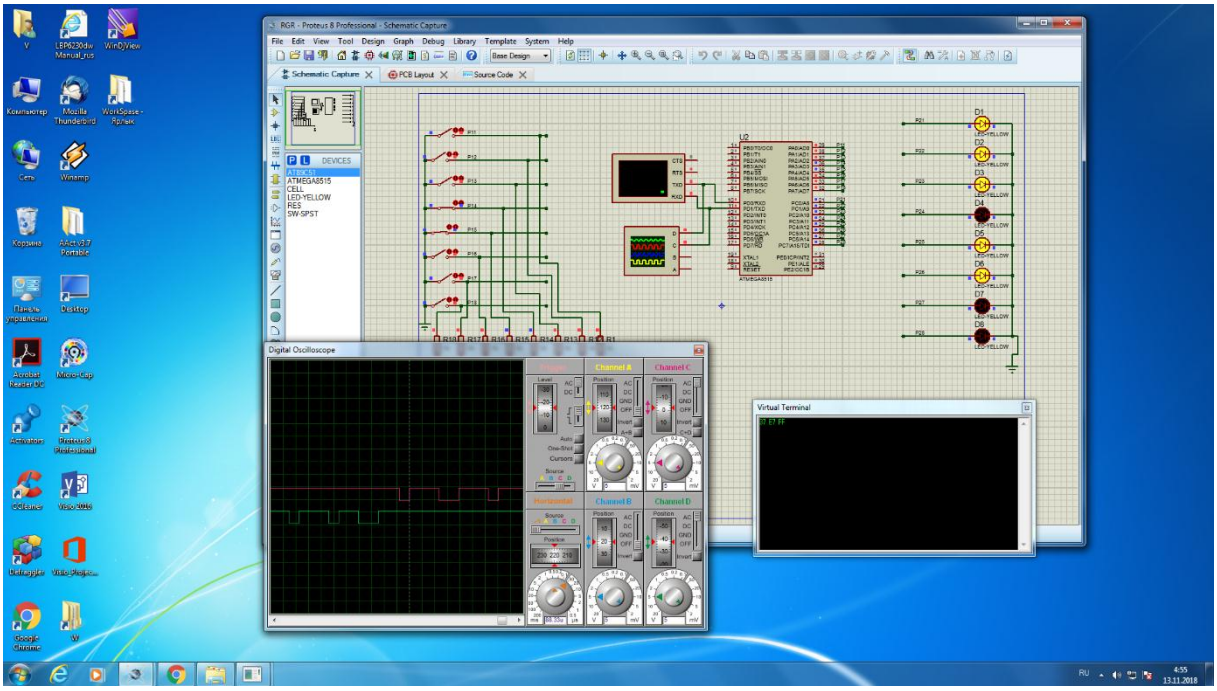


Рисунок 8.9

Розглянемо сигнал, який передається від кнопок.

Перший перепад із високого рівня у низький – старт–біт.

Потім ідуть вісім інформаційних бітів, починаючи з молодшого розряду, які будуть відображені на віртуальному терміналі та осцилографі, як 1110 0111 (bin) = E7 (hex). Далі іде 9–й біт , який дорівнює 0, тому що число одиниць в байті, який було передано – парне (розряд P в регістрі прапорців дорівнює 0). Останній 10–й біт, який дорівнює 1, це стоп–біт.

8.2.13 Тепер визначимо отриману швидкість передачі. Для знаходження швидкості передачі (в даному прикладі було обрано швидкість 9600 біт/сек = =9600 пос/сек = 9600 бод) підберемо таку розгортку сигналу на осцилографі, щоб сумістити бічні сторони імпульсу із вертикальними лініями сітки осцилографа.

Для швидкості передачі 9600 біт/сек отримаємо наступну картину (рисунок 8.10).

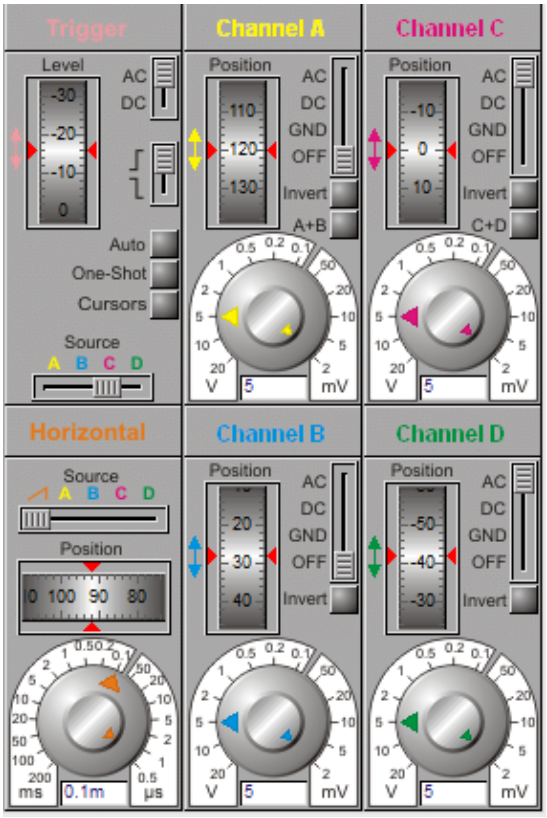


Рисунок 8.10

Як ми бачимо, тривалість одного імпульсу дорівнює 0,104 мс. Отже, якщо розділити один біт на цей час, та помножити на 1000, отримаємо задану швидкість:

$$V = \frac{1}{0.104} * 1000 = 9600 \ \frac{\text{біт}}{\text{сек}}.$$

Нижче розглянуто ще один приклад (рисунок 8.11, 8.12), коли на кнопках було набрано число 67(hex)=01100111(bin). На верхньому промені осцилографа побачимо: перший нуль – старт- біт, потім 3 одиниці та один 0 – це число 7 на кнопках, далі йде нуль, дві одиниці та нуль – це число 6 на кнопках, а потім на осцилографі бачимо 1 – це біт доповнення до парності, та наступна одиниця – стоп-біт.

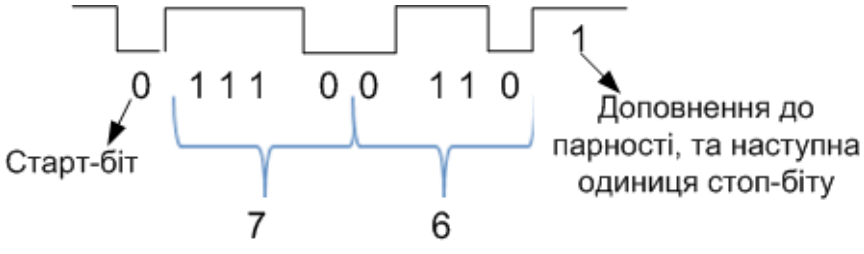


Рисунок 8.11

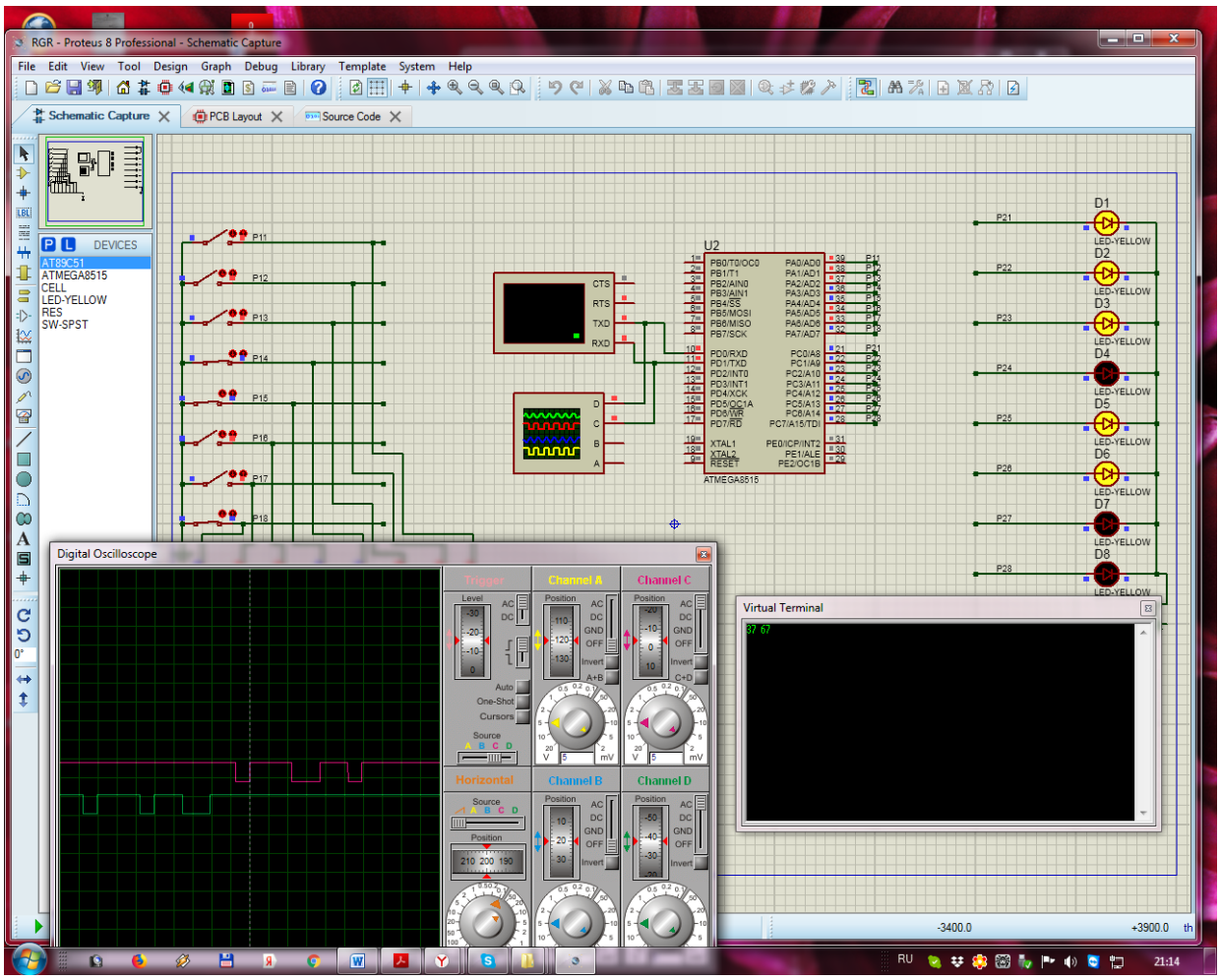


Рисунок 8.12

8.3 Схема алгоритму роботи

На рисунку 8.13 наведено схему алгоритму роботи.

8.4 Лістинг робочої програми

Лістинг робочої програми, яку розроблено згідно з алгоритмом, який наведено на рисунку 8.13, представлено надалі. В дужках праворуч біля кожної команди цифрою відмічено номер блоку схеми алгоритму, який реалізує ця команда.

Рисунок 8.13 - Схема алгоритму роботи

Робоча програма мовою Асемблер

```
.nolist                    ; директива відключає генерацію коду у лістинг,
        ;тобто далі у файлі *.lss не буде фіксуватися асемблерний код
.include "m8515def.inc"    ;підключення стандартного заголовочного
                           ;файлу для ATmega8
.list                      ;директива включає генерацію коду у лістинг,
        ;тобто далі у файлі *.lss буде фіксуватися асемблерний код
.equ f_CK = 8000000                ;частота в герцах
.equ BAUD = 9600                   ;швидкість для UART у бодах
.equ UBRR_value = (f_CK/(BAUD*16))-1   ;розраховуємо значення для
                                       регістра  ;UBRR
.cseg                      ;директива означає, що далі іде код програми
.org 0                     ;директива означає, що  код програми у FLASH
буде
                           ;розміщено з нульової адреси
; КОД ОСНОВНОЇ ПРОГРАМИ
main:
;ініціалізації з мітки init_USART
rcall init_USART           ; (блок 2) відносний виклик підпрограми
;налаштування порту С на передачу
LDI R16, 0xFF              ; (блок 2) R16← 0xFF
OUT DDRC, R16              ; (блок 2) DDRC← R16
; (блок 3) отримуємо те, що відправив віртуальний термінал
rcall USART_recieve        ; (блок 3) відносний виклик підпрограми з мітки
                                   ; USART_recieve
; (блок 4) виводимо отримане від терміналу на світлодіоди
OUT PORTC, R16             ; (блок 4) PORTC←R16
; (блок 5) налаштовуємо порт А як передавач
LDI R16, 0x00              ; (блок 5) R16←0x00
OUT DDRA, R16              ; (блок 5) DDRA← R16
```

```
                                ; (блок 6) читаємо стан кнопок
        in R16, PINA            ; (блок 6) R16← PINA
        ; (блок 7) виводимо отримане від кнопок на світлодіоди
        OUT PORTC, R16          ; (блок 7) PORTC←R16
        ; (блок 8)  підпрограма з міткою USART_send відправляє стан кнопок
        на осцилограф та термінал
        rcall USART_send        ; (блок 8) відносний виклик підпрограми з мітки
                                ; USART_ send
        ; (блок 1) ПІДПРОГРАМА ІНІЦІАЛІЗАЦІЇ USART – МОДУЛЯ
        init_USART:
        ; програмуємо швидкість обміну 9600бод
    ldi   R16, high(UBRR_value)      ; (блок 1) R16←старший байт
UBRR_value
    out   UBRRH, R16                 ; (блок 1) UBRRH←R16
    ldi   R16, low(UBRR_value)       ; (блок 1) R16←молодший байт
                                       UBRR_value
    out   UBRRL, R16                 ; (блок 1) UBRRL←R16
      ; дозволяємо роботу приймача-передавача модуля USART
    ldi   R16, (1<<RXEN)|(1<<TXEN)|(0<<RXCIE)|(0<<TXCIE)|(0<<UDRIE)
    out   UCSRB, R16                       ; (блок 1) UCSRB←R16
      ; програмуємо передачу 8 біт з перевіркою на парність
      ; у асинхроному режимі
    ldi   R16, (1<< URSEL)|(1<<UPM1)|(1<<UPM0)|(1<< UCSZ1)|(1<< UCSZ0)
      out   UCSRC, R16               ; (блок 1) UCSRC← R16
      ret                           ; (блок 1) повернення з підпрограми
ініціалізації
      ; передача через модуль USART
      USART_send:
      out   UDR, R16                  ; (блок 8) регістр даних UDR← R16
      sending:
```

```
; чекаємо завершення передачі
sbis  UCSRA, TXC        ; (блок 8) якщо біт TXC =1, то наступна
;команда пропускається, інакше  - наступна команда
rjmp  sending           ; блок 8) безумовний перехід на мітку sending
ret                     ; (блок 8) повернення з підпрограми
USART_recieve:
sbis  UCSRA, RXC        ; (блок 3), якщо біт RXC =1 (в буфері ПРМ є
; отриманий байт) , то пропустити наступну команду,
; інакше наступна команда
rjmp  USART_recieve     ; (блок 3) безумовний перехід на мітку
                        ;USART_recieve
in      R16, UDR        ; (блок 3) R16←UDR (завантажуємо в
                        ; регістр  R16 отриманий байт від термінала

ret                     ; (блок 3) повернення з підпрограми
```

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Евстифеев А. В. Микроконтроллеры AVR семейства Mega. Руководство пользователя. – М.:Издательский дом «Додэка–XXI», 2007.
2. Проектування мікропроцесорних систем: Проектування мікропроцесорних систем на базі AVR–мікроконтролерів: Периферійні модулі AVR–мікроконтролерів: Навчальний посібник для студентів напряму підготовки 6.050201 «Системна інженерія» кафедри Автоматики та управління у технічних системах / Укл.: А.О. Новацький– К: НТУУ „КПІ”, 2012– 470с.
3. Навчальний посібник з дисципліни «Проектування мікропроцесорних систем», розділ «Програмування мікроконтролерів родини AVR» для студентів напряму підготовки 6.050201 «Системна інженерія» кафедри Автоматики та управління у технічних системах / Укл.: А.О. Новацький, Є.В. Глушко – К: НТУУ „КПІ”, 2013 – 109 с.
4. Комп’ютерна електроніка: Мікропроцесорні системи: Програмування мікропроцесорних систем: Навчальний посібник для студентів напряму підготовки 6.050201 «Системна інженерія» кафедри Автоматики та управління у технічних системах /Автор.: А.О. Новацький– К: НТУУ „КПІ”, 2014– 307с.
5. Проектування мікропроцесорних систем: Кредитний модуль «Проектування мікропроцесорних систем на базі мікроконтролерів сімейства MCS-51»: Програмування мікроконтролерів сімейства MCS–51: Навчальний посібник для студентів напряму підготовки 6.050201 «Системна інженерія» кафедри Автоматики та управління у технічних системах / Автор: А.О. Новацький– К: НТУУ „КПІ”, 2015– 156с.

6. Проектування мікропроцесорних систем: Проектування мікропроцесорних систем на базі мікроконтролерів сімейства MCS–51: Периферійні модулі мікроконтролерів сімейства MCS–51:
Навчальний посібник для студентів напряму підготовки 6.050201 «Системна інженерія» кафедри Автоматики та управління у технічних системах / Автор: А.О. Новацький– К: НТУУ „КПІ”, 2016– 391с.