ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«МЕТІНВЕСТ ПОЛІТЕХНІКА»

ЕЛЕКТРОНІКА ТА МІКРОПРОЦЕСОРНА ТЕХНІКА

**методичні вказівки
до виконання практичних робіт**

Запоріжжя 2024

metinvest polytechnic

УДК 621.3(072)
С56

Рекомендовано Науково-методичною радою
ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА»
(протокол № 8 від 12.07.2024 р)

Автори:
Сокол Сергій Петрович, старший викладач; Койфман Олексій Олександрович, доцент, канд. тех. наук.

Рецензенти:
Разживін Олексій Валерійович – кандидат технічних наук, доцент кафедри автоматизації виробничих процесів Донбаської державної машинобудівної академії
Марков Олег Євгенійович – доктор технічних наук, професор, завідувач кафедри автоматизації виробничих процесів Донбаської державної машинобудівної академії

С59 Сокол С. П., Койфман О. О. Електроніка та мікропроцесорна техніка : методичні вказівки до виконання практичних робіт з дисципліни «Електроніка та мікропроцесорна техніка». Запоріжжя : ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА», 2024. 142 с.

Методичні вказівки включають стислі теоретичні відомості, методичні поради та рекомендації щодо порядку виконання практичних робіт з дисципліни «Електроніка та мікропроцесорна техніка», приклади, критерії оцінювання звітів з практичних робіт.

УДК 621.3(072)

ЗМІСТ

ВСТУП

Метою виконання практичних робіт з дисципліни «Електроніка та мікропроцесорна техніка» є закріплення здобувачами знань щодо розробки принципових електричних схем різноманітних приладів та систем та програмного забезпечення для мікроконтролерів. В результаті виконання практичних робіт передбачається набуття здобувачами навичок у конструюванні схем цифрової електроніки на базі мікроконтролерів, використанні булевої алгебри та комбінаторної логіки, а також програмування мікроконтролерів мовами високого рівня.

Практичні роботи здобувачами виконуються за допомогою обчислювальної техніки, в середовищі SimulIDE. Результатом виконання практичної роботи є оформлений за вимогами та зданий звіт.

Звіт по кожній роботі має бути оформлений відповідно до вимог щодо оформлення технічної документації на аркушах формату А4 у вигляді документу у форматі *.doc, *.docx, або *.pdf. До складу звіту повинні входити:

- титульна сторінка;
- назва та мета практичної роботи;
- постановка завдання відповідно варіанту;
- лістінг програми;
- результати роботи програми;
- висновки.

Критерії оцінювання наведені в таблиці.

Таблиця – Критерії оцінювання

Кількість балів	Критерій оцінювання
8	Здобувач(ка) працював(ла) на практичних заняттях, виконав(ла) роботу в повному обсязі відповідно до свого варіанту завдання та завантажив(ла) правильно оформлений звіт у Moodle.
7-6	Здобувач(ка) працював(ла) на практичних заняттях, виконав(ла) роботу в повному обсязі відповідно до свого варіанту завдання та завантажив(ла) правильно оформлений звіт у Moodle, але зробив незначні помилки у схемі або програмі.
5-4	Здобувач(ка) працював(ла) на практичних заняттях, виконав(ла) роботу в повному обсязі відповідно до свого варіанту завдання та завантажив(ла) звіт у Moodle, але зробив суттєві помилки у схемі або програмі, або виконав(ла) роботу не в повному обсязі.
3-1	Здобувач(ка) працював(ла) на практичних заняттях, виконав(ла) роботу частково або з критичними помилками та завантажив(ла) звіт у Moodle.
0	Здобувач(ка) не завантажив(ла) звіт в Moodle.

1 ПРАКТИЧНА РОБОТА №1 «СИМУЛЯЦІЯ СХЕМ З МІКРОКОНТРОЛЕРАМИ З ВИКОРИСТАННЯМ ПРОГРАМИ SIMULIDE»

1.1 Завдання

Завдання практичної роботи:

– встановлення програми SimulIDE та компілятора AVR GCC на комп'ютер;

– ознайомлення з інтерфейсом програми SimulIDE;

– створення електричної схеми у програмі SimulIDE та запуск її симуляції;

– написання програми для мікроконтролеру у програмі SimulIDE та створення файлу прошивки мікроконтролера.

Посилання на сторінку завантаження програми SimulIDE
https://www.simulide.com/p/downloads.html
Посилання на сторінку завантаження компілятора AVR GCC
https://www.microchip.com/en-us/tools-resources/develop/microchip-studio/gcc-compilers

1.2 Теоретичні дані

1.2.1 Призначення програми SimulIDE

Програма SimulIDE, так само, як і розглянута у минулому семестрі LTspice, дозволяє креслити на екрані комп'ютера принципові електричні схеми електричних пристроїв та виконувати імітацію чи симуляцію роботи всієї схеми. Набір інструментів дозволяє контролювати роботу схеми та графіки зміни сигналів.

На відміну від LTSpice, SimulIDE виконує симуляцію у реальному часі, дозволяючи впливати на схему (натиснути кнопку, змінити опір змінного резистора, ввімкнути/вимкнути напругу та інше) і одразу бачити результат цього впливу.

SimulIDE базується на спрощених моделях електронних компонентів, то ж її не можна використовувати для серйозних досліджень, але для того, щоб мати уявлення про роботу тієї чи іншої схеми, вона підходить дуже добре.

Головною відмінністю SimulIDE від LTSpice, через яку ми і будемо її використовувати в цьому семестрі, є можливість симулювати роботу мікроконтролерів, завантажуючи в них файл прошивки, який можна зробити прямо в цій програмі, що є дуже зручним.

Головним конкурентом програми SimulIDE є Proteus, який вже став де-факто стандартом, якщо йде мова про симуляцію схем на базі мікроконтролерів. Але Proteus є платною програмою, що в

максимальному варіанті коштує декілька тисяч доларів, в той час як SimulIDE є безкоштовною і з відкритим кодом.

Звичайно, можливості SimulIDE поступаються можливостям Proteus, але для навчальних цілей першої більш ніж достатньо.

1.2.2 Встановлення необхідних програм та засобів

SimulIDE – це вільно-розповсюджувана програма, яку можна безкоштовно завантажити з офіційного сайту її однойменного розробника за посиланням https://www.simulide.com/p/downloads.html. На сторінці загрузки ви можете обрати варіант дистрибутиву для саме вашої операційної системи (рис. 1.1).

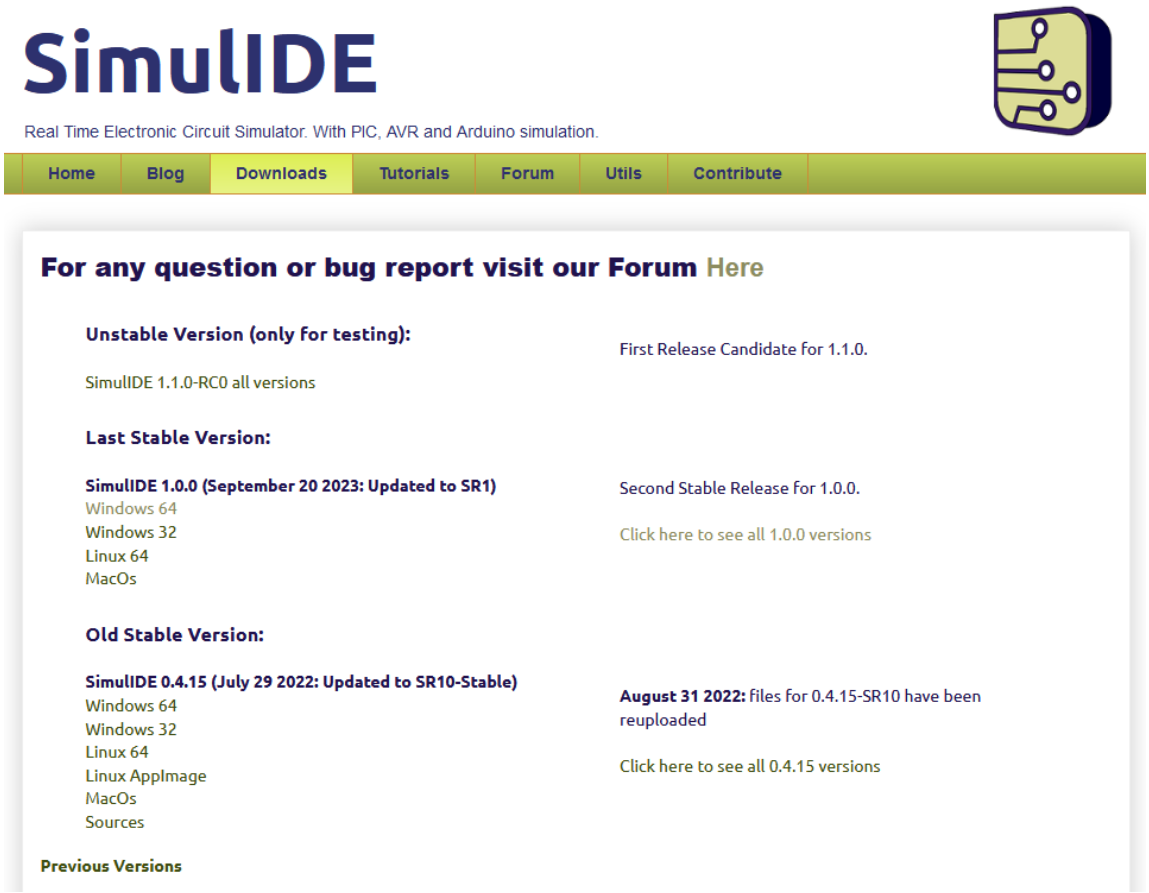


Рисунок 1.1 – Завантаження SimulIDE

Після вибору версії (для прикладу була обрана версія для Windows 64), відкривається наступне вікно (рис. 1.2).

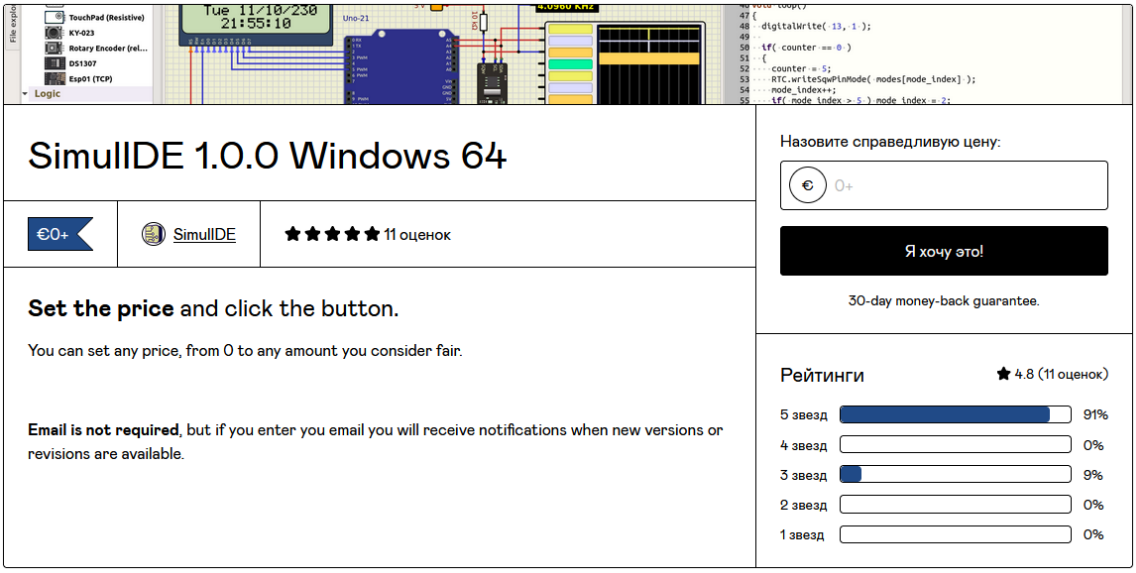


Рисунок 1.2 – Придбання SimulIDE

Як вже було сказано раніше, програма SimulIDE є безкоштовною, то ж у полі «Назовите справедливую цену» ви можете вказати 0 і натиснути на кнопку «Я хочу это!». Звичайно, ви можете вказати і більшу суму, якщо хочете задонатити авторам програми, але це не обов'язково.

У наступному вікні (рис. 1.3) треба ввести вашу електронну адресу і натиснути на кнопку «Получить».

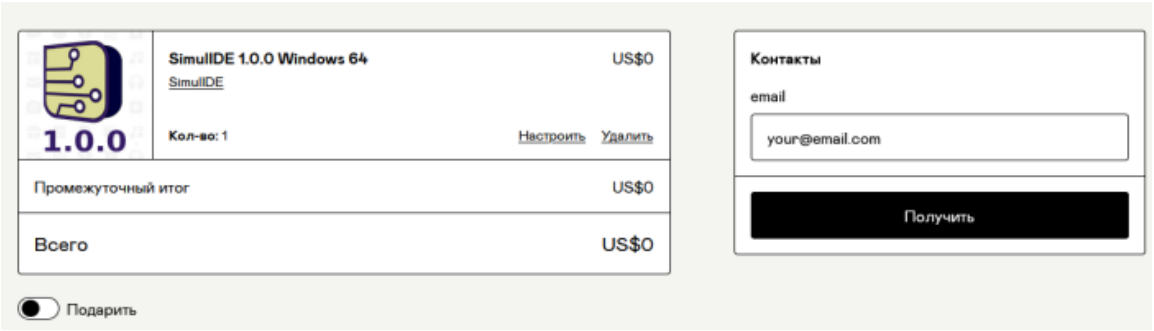


Рисунок 1.3 – Отримання SimulIDE

Після цього почнеться завантаження архіву з програмою, який на момент написання цього керівництва має назву «SimulIDE_1.0.0-SR1_Win64.zip» і розмір близько 18 МБ.

Програма не потребує встановлення. Треба лише розпакувати отриманий архів у будь-яку теку, і програмою вже можна користуватися.

Для запуску програми треба зайти у теку «SimulIDE_1.0.0-SR1_Win64» та запустити файл «simulide.exe». Після чого відкриється вікно, показане на рис. 1.4.

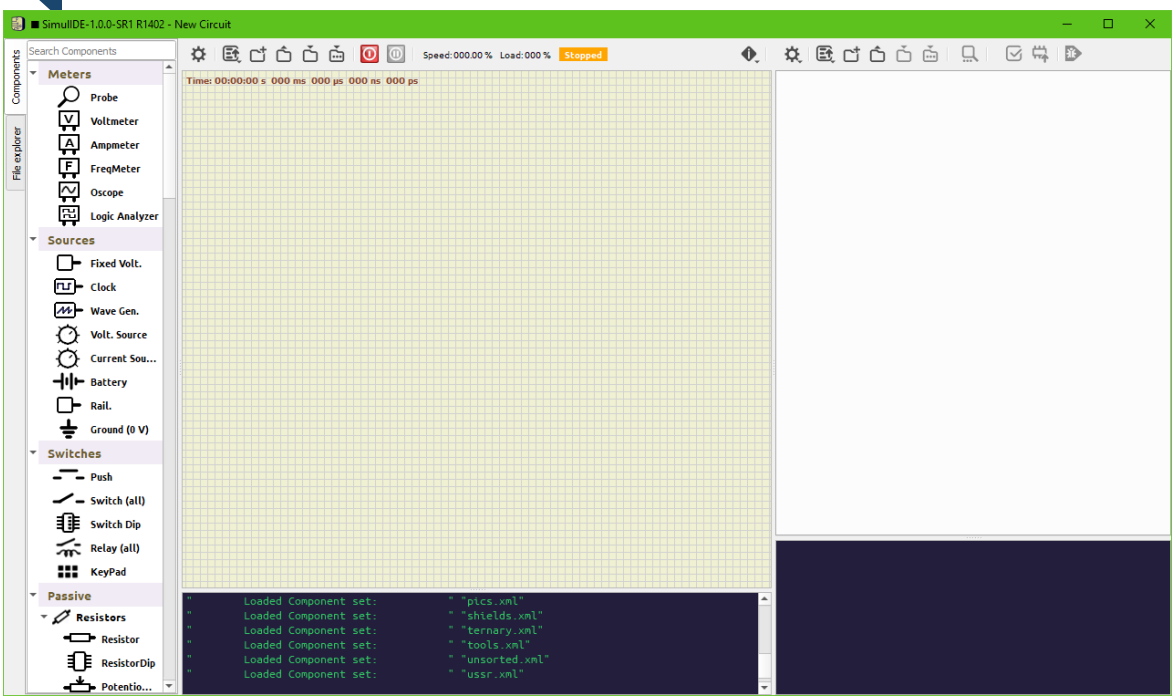


Рисунок 1.4 – Інтерфейс програми SimulIDE

Більш детально ми розглянемо цю програму у наступному розділі, а поки нам ще потрібно встановити компілятор AVR GCC, призначений для створення файлів прошивки для мікроконтролерів AVR.

Краще за все завантажити компілятор з офіційного сайту виробника цих мікроконтролерів – компанії Microchip – за наступним посиланням: https://www.microchip.com/en-us/tools-resources/develop/microchip-studio/gcc-compilers. На цій сторінці треба доскролити до кінця, та завантажити файл, що відповідає вашій операційній системі (рис. 1.5).

GCC Compilers for AVR® and Arm®-Based Devices

Title	Version Number	Date	
AVR 8-Bit Toolchain (Windows)	3.7.0	12 May 2022	Download
AVR 8-Bit Toolchain (Linux)	3.7.0	12 May 2022	Download
AVR 8-Bit Toolchain (OSX)	3.7.0	12 May 2022	Download
AVR 8-Bit Toolchain- Release Notes	3.7.0	12 May 2022	Download
Arm 32-bit GNU Toolchain (Linux)	6.3.1	06 Jun 2019	Download
Arm 32-bit GNU Toolchain (Windows)	6.3.1	06 Jun 2019	Download

Рисунок 1.5 – Завантаження компілятора AVR GCC

Будьте уважні, на цій сторінці є також посилання на завантаження компілятора для 32-бітних мікроконтролерів ARM, який нам не потрібний. То ж обирайте один з компіляторів «AVR 8-bit Toolchain» і натискайте на кнопку «Download» у правому стовпчику таблиці.

Після цього завантажиться архів, який для Windows має назву «avr8-gnu-toolchain-3.7.0.1796-win32.any.x86_64.zip» і розмір приблизно 51 МБ. Його також треба розархівувати у будь-яку теку, і поки що залишити, ми повернемося до нього пізніше.

1.2.3 Подання на екрані та основні елементи керування

Інтерфейс програми SimulIDE більш інтуїтивно-зрозумілий, ніж у LTspice (рис. 1.6). У ньому є такі основні елементи.

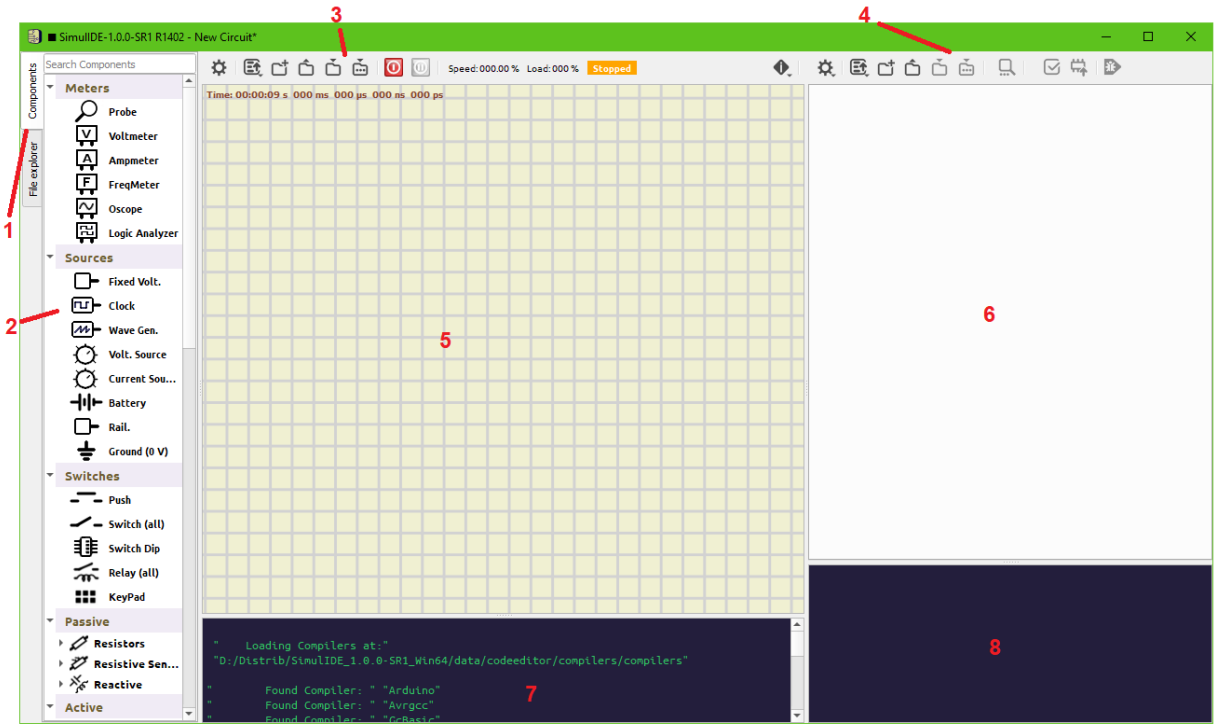


Рисунок 1.6 – Інтерфейс програми SimulIDE

1 – Вибір лівої панелі (2). Тут є два варіанти:

- Components – у лівій панелі (2) показуються всі компоненти, що підтримуються програмою на даний момент;
- File Explorer – у лівій панелі (2) замість компонентів показується поточний каталог та файлова система комп'ютера, так що прямо звідти можна відкрити файл схеми або програми;

2 – Ліва панель. Як вже було зазначено, в ній може відображатися або список компонентів, або файловий менеджер. Компонентів у SimulIDE досить багато, і деякі з них є доволі просунутими (наприклад, датчики, дисплеї, тощо), що дозволяє симулювати відносно складні пристрої. Зараз ми не будемо розглядати тут всі існуючі компоненти, лише коротенько пройдемося по підзаголовках:

- Meters – вимірювальні прилади: вольтметр, амперметр, частотомер, осцилоскоп та логічний аналізатор;

- Sources – джерела живлення (як напруги, так і струму);
- Switches – кнопки, вимикачі, реле, клавіатури;
- Passive – пасивні компоненти: резистори, конденсатори, котушки індуктивності тощо;
- Active – активні компоненти: діоди, транзистори, тиристори, операційні підсилювачі та інше;
- Outputs – різні компоненти, якими можна керувати: світлодіоди, світлодіодні індикатори та матриці, дисплеї, двигуни, динамік;
- Micro – це власне мікроконтролери та додаткові компоненти та плати. SimulIDE підтримує декілька видів мікроконтролерів: AVR (які ми і будемо використовувати в цьому курсі), PIC, I51 та Arduino. Останній, до речі, не є окремим типом мікроконтролера, бо базується на базі AVR, але має власну мову програмування та власні бібліотеки, то ж його винесено окремо;
- Logic – різноманітні логічні компоненти, тригери, лічильники, регістри, АЦП, ЦАП та інше;
- Connectors – роз'єми різних типів;
- Graphical – графічні примітиви, які також можна додати до схеми;
- Other – компоненти, що не підійшли до жодної з перелічених категорій.

3 – Панель керування принциповою електричною схемою, на якій знаходяться наступні кнопки (табл. 1.1).

Таблиця 1.1 – Кнопки панелі керування принциповою електричною схемою програми SimulIDE

Кнопка	Призначення
	Налаштування симуляції
	Список останніх відкритих файлів схем
	Створити нову схему
	Відкрити існуючу схему
	Зберегти поточну схему з поточним ім'ям
	Зберегти поточну схему з новим ім'ям
	Запустити або зупинити симуляцію схеми
	Призупинити або знову запустити симуляцію

4 – Панель керування програмним кодом, на якій знаходяться наступні кнопки (табл. 1.2).

5 – Робоче поле для створення принципової електричної схеми пристрою.

6 – Поле для написання програмного коду.

7 – Вікно інформації про стан роботи симулятора.

8 – Вікно інформації про процес компіляції та про знайдені у тексті програми помилки.

Таблиця 1.2 – Кнопки панелі керування програмним кодом програми SimulIDE

Кнопка	Призначення
	Інформація про SimulIDE
	Налаштування редактора коду або компілятора
	Список останніх відкритих файлів програм
	Створити новий файл програми
	Відкрити існуючу програму
	Зберегти поточну програму з поточним ім’ям
	Зберегти поточну програму з новим ім’ям
	Знайти або замінити текст у програмі
	Створити файл прошивки
	Завантажити файл прошивки у мікроконтролер
	Запустити налагодження програми

1.2.4 Основні прийоми створення та коригування схеми

Процес створення схеми у SimulIDE максимально інтуїтивно зрозумілий. Для того, щоб додати якийсь компонент на робоче поле (5), треба його знайти у бібліотеці компонентів (2) та перетягнути лівою кнопкою миші.

Створимо просту схему, в яку буде входити мікроконтролер, світлодіод та резистор, що обмежує струм через цей світлодіод.

Щоб додати до схеми мікроконтролер, треба знайти підзаголовок «Micro», всередині нього розгорнути підзаголовок «AVR», натиснувши на трикутник зліва від назви. Всередині ми побачимо ще два підзаголовка: «attiny» та «atmega», які відповідають мікроконтролерам сімейств ATTiny та ATMega. Перше сімейство включає в себе більш прості мікроконтролери з невеликим об’ємом пам’яті (0,5 – 32 кБ), невеликою кількістю виводів (8 – 32) та скороченим набором периферійних модулів, в той час, як друге сімейство є більш просунутим, має більше пам’яті (4 – 256 кБ), більше виводів (28 – 100), та більшу кількість периферійних модулів.

Розгорнемо підзаголовок «atmega», оберемо мікроконтролер «mega8» та за допомогою миші перетягнемо його на робоче поле (рис. 1.7).

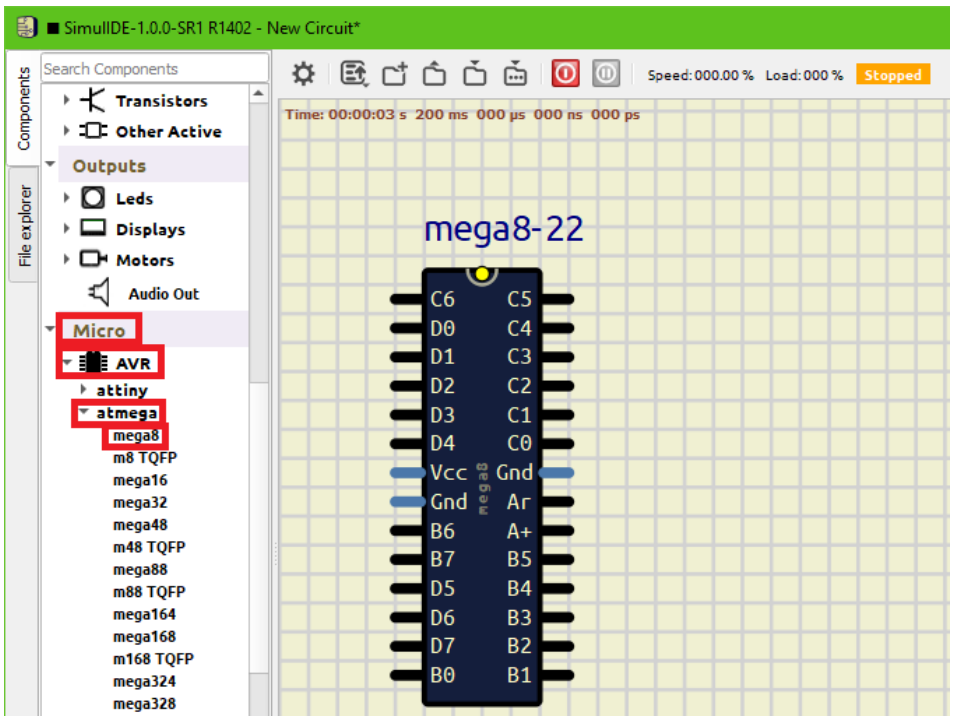


Рисунок 1.7 – Додавання мікроконтролера ATMega8 до схеми

Решту виводів, що мають чорний колір, можна довільно підключати до будь-яких компонентів (або навіть з'єднувати їх з іншими виводами того ж самого компоненту).

Тепер таким самим чином розкриємо підзаголовок «Passive» - «Resistors» та витягнемо до схеми компонент «Resistor» (рис. 1.8).

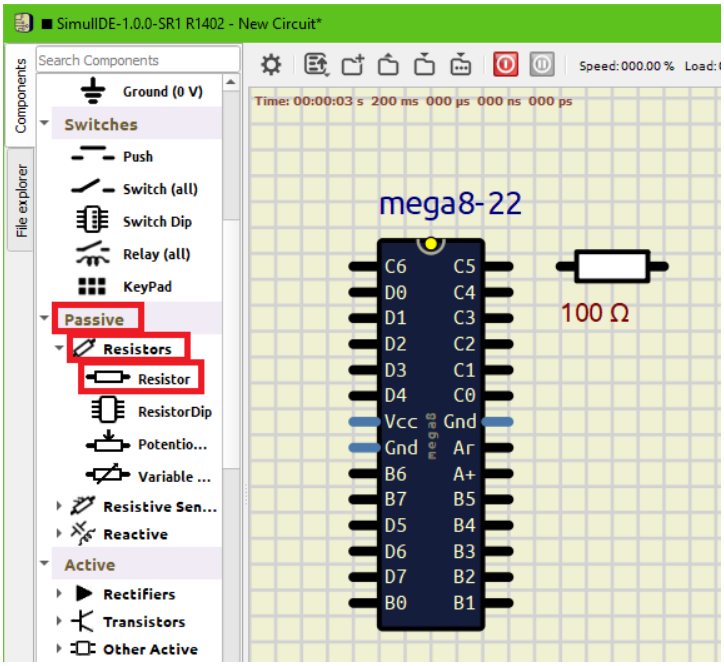


Рисунок 1.8 – Додавання резистора до схеми

Аналогічно, розкриємо підзаголовок «Outputs» - «LEDs», та витягнемо до схеми компонент «LED» (рис. 1.9).

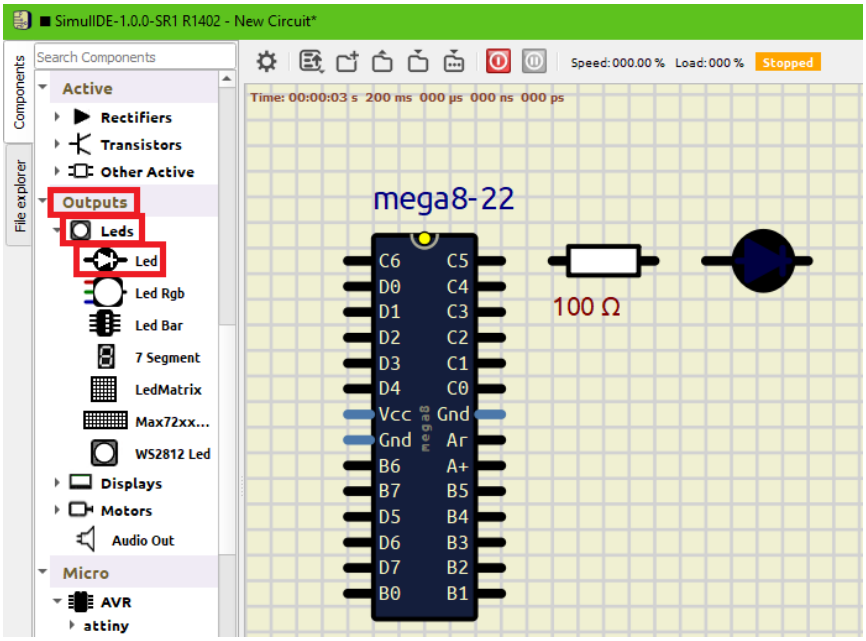


Рисунок 1.9 – Додавання світлодіода до схеми

Нарешті, до схеми треба додати землю, як і в усіх інших подібних програмах, щоб симулятор знав, де точка з нульовим потенціалом. Для цього розгорнемо підзаголовок «Sources» та витягнемо компонент «Ground (0V)» (рис. 1.10).

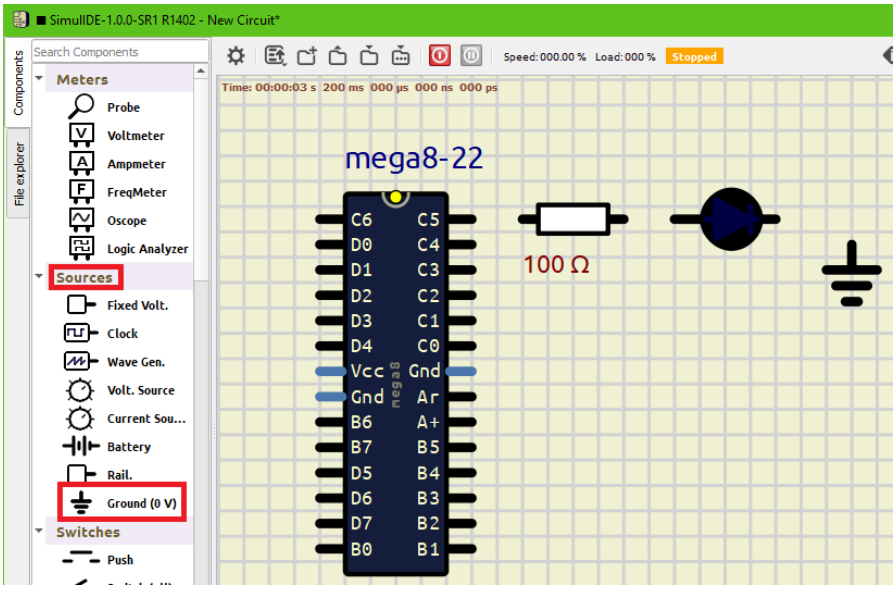


Рисунок 1.10 – Додавання землі до схеми

Нарешті, всі необхідні компоненти знаходяться на місці, тепер необхідно їх з’єднати між собою.

Для того, щоб це зробити, треба натиснути лівою кнопкою миші на виводі, який ми хочемо під'єднати, після чого кнопку можна відпустити, і помітити, що тепер за курсором миші тягнеться чорний провід. Треба підвести його до того виводу, до якого ми хочемо підключитися, і знову натиснути на ньому лівою кнопкою миші. Тепер два виводи з'єднані між собою. Аналогічним чином з'єднуються всі інші компоненти. В результаті маємо наступну схему (рис. 1.11).

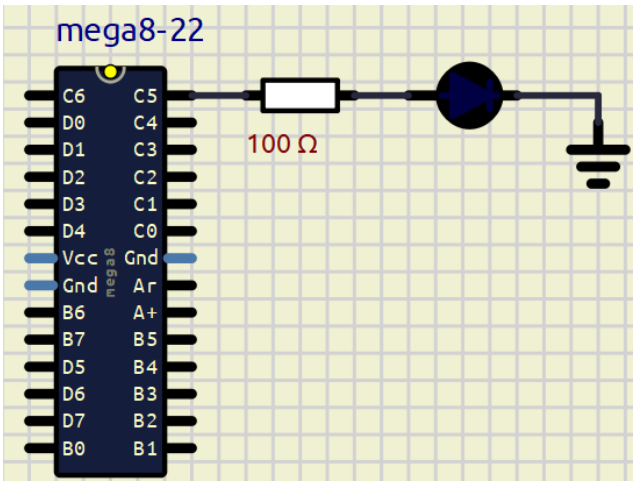


Рисунок 1.11 – З'єднана схема

Тепер розглянемо, що саме ми зробили, і як воно має працювати.

Мікроконтролер спілкується з навколишнім світом через порти вводу-виводу. У мікроконтролерів AVR такі порти є восьмибітними, тобто кожний порт може мати не більше восьми виводів. Всі порти позначаються великою латинською літерою від A і далі по алфавіту. Деякі літери можуть бути відсутні.

У мікроконтролера ATMega8 23 лінії вводу-виводу: 8 ліній порту B (B0-B7), 7 ліній порту C (C0-C6) та 8 ліній порту D (D0-D7), див. рис. 1.11.

У мікроконтролерів AVR всі виводи є рівнозначними, це означає, що будь який вивід може бути сконфігурованим або як вхід (тобто приймати вхідний сигнал від інших приладів), або як вихід (тобто подавати на цей вивід або високий логічний рівень, або низький).

В нашій схемі (рис. 1.11) ми підключили світлодіод до виводу C5, але таким самим чином ми могли б використати будь-який вивід. Далі ми напишемо програму, яка буде періодично переключати стан виводу C5, подаючи та прибираючи напругу на ньому. В момент подачі на вивід C5 високого рівня світлодіод ввімкнеться, а в момент подачі низького рівня напруги він вимкнеться. Таким чином ми реалізуємо блимання світлодіодом із заданою частотою.

Перед тим, як переходити до написання програми, треба встановити правильні параметри для всіх компонентів.

Для того, щоб налаштувати параметри компонента, треба двічі клацнути на ньому лівою кнопкою миші.

Вікно налаштувань мікроконтролера виглядає наступним чином (рис. 1.12).

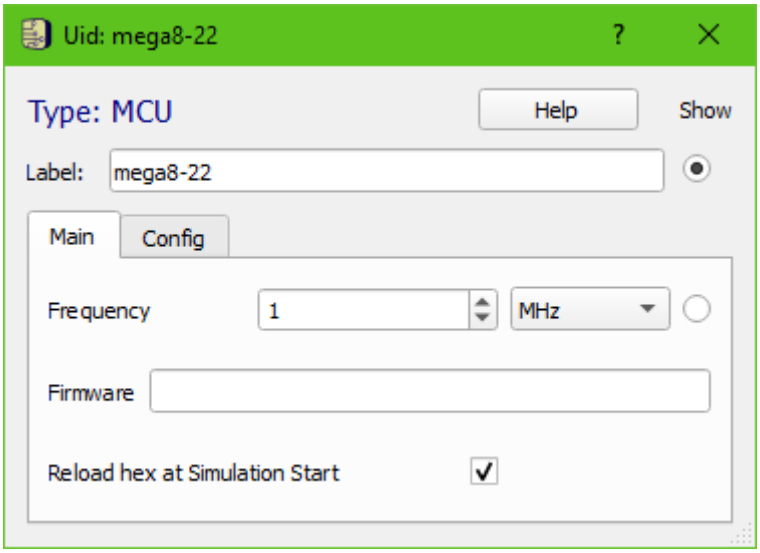


Рисунок 1.12 – Вікно налаштувань мікроконтролера

Як можна бачити, самих налаштувань не так багато.

- Якщо натиснути на кнопку «Help», то праворуч відкриється детальна інформація про даний компонент.

- Поле «Label» дозволяє задати ім'я компонента на схемі. Можна його залишити без змін.

- Власне налаштування розбиті на дві вкладки: «Main» та «Config». На вкладці «Main» розташовані такі параметри:

  ○ «Frequency» - робоча частота мікроконтролера. За замовчанням вона встановлена, як 16 МГц, але ми зменшимо її до 1 МГц. Для чого це потрібно, буде пояснено пізніше.

  ○ «Firmware» - файл прошивки мікроконтролера. Можна його прописати прямо тут, а можна вибрати у спеціальному пункті меню, про який також буде сказано пізніше.

  ○ «Reload hex at simulation start» - досить корисна опція, що дозволяє автоматично завантажувати прошивку мікроконтролера перед стартом симуляції. Таким чином ми будемо впевнені, що симуляція відбувається з найновішою версією прошивки. За замовчанням, ця опція відключена, то ж її рекомендується включити.

- На вкладці «Config» (рис. 1.13) розташовані наступні налаштування:

  ○ «Enable Reset Pin» - ввімкнути вхід скиду

  ○ «External Oscillator» - ввімкнути зовнішнє джерело тактових імпульсів для мікроконтролера

  ○ «Enable Watchdog» - ввімкнути вбудований сторожовий таймер мікроконтролера.

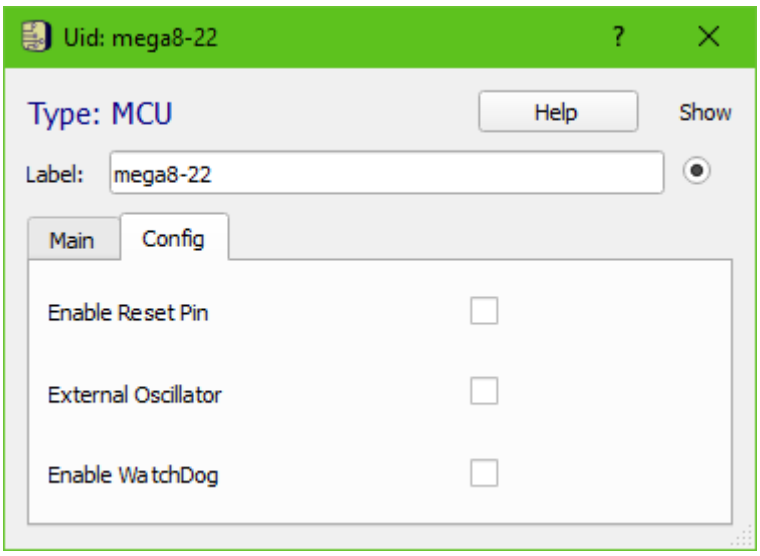


Рисунок 1.13 – Вікно налаштувань мікроконтролера

Всі налаштування з вкладки «Config» треба залишити вимкненими. Поки що ми не будемо зупинятися на них, а розглянемо пізніше у подальших лекціях та практичних роботах. Єдине, що тут треба зазначити, що **деякі мікроконтролери мають окремий вивід скиду, який не можна відключити. В такому разі його треба під'єднати до джерела з напругою +5 В інакше симуляція не зможе запуститися через помилку.**

Вікно налаштувань світлодіода виглядає наступним чином (рис. 1.14).

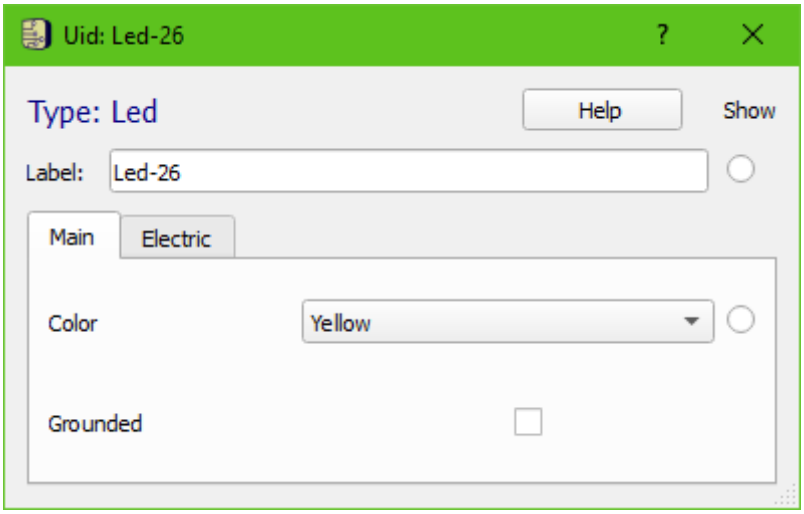


Рисунок 1.14 – Налаштування світлодіода

Структура вікна налаштувань така ж сама, як і для мікроконтролера.

На вкладці «Main» розташовані такі опції:

- «Color» - колір світлодіода. Можна обрати один з шести запропонованих варіантів. За замовчанням стоїть жовтий колір.

- «Grounded» - якщо встановити цю опцію, то катод світлодіода автоматично буде під’єднано до землі, тобто до нульового потенціалу. Таким чином можна не використовувати окремий елемент «Ground», до якого підключено катод, як це зробили ми.

На вкладці «Electric» розташовані такі опції (рис 1.15):

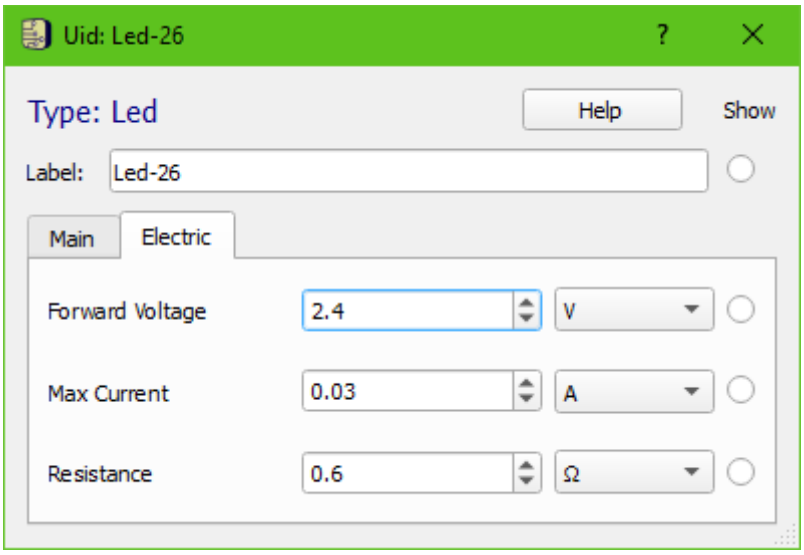


Рисунок 1.15 – Налаштування світлодіода

- «Forward Voltage» - падіння напруги на світлодіоді у відкритому стані.
- «Max Current» - максимальний струм через світлодіод.
- «Resistance» - опір світлодіода у відкритому стані.

Ці налаштування можна не змінювати, але їх треба запам’ятати, адже вони нам знадобляться для розрахунку опору резистору, що обмежує струм через світлодіод.

Нарешті, налаштування резистора представлені на рис. 1.16.

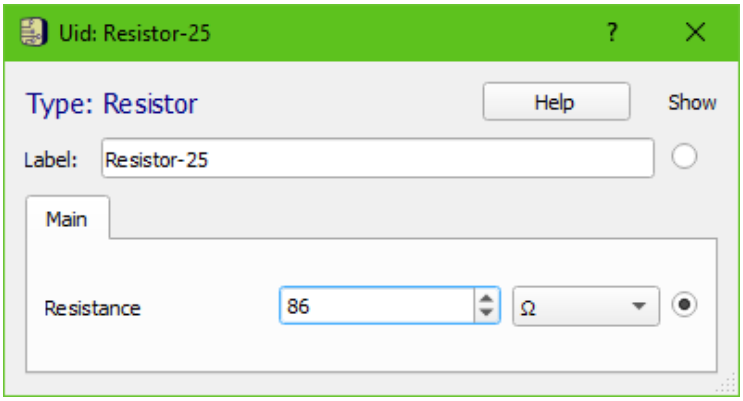


Рисунок 1.16 – Налаштування резистора

Як бачимо, у нього є лише один параметр «Resistance», який задає опір резистора.

Давайте порахуємо цей опір базуючись на знаннях параметрів світлодіода (рис. 1.15).

Мікроконтролер видає на своїх виводах близько 5 В, що відповідає логічній одиниці. Падіння напруги на світлодіоді складає 2.4 В (рис. 1.15). То ж на резистор припадає напруга 5 – 2.4 = 2.6 В.

Струм через світлодіод (він же і струм через резистор) складає 0.03 А, тому величина опору ланцюга повинна бути 2.6 / 0.03 = 86.6 Ом. Оскільки сам світлодіод має опір 0.6 Ом (рис. 1.15), то опір резистора повинен бути 86.6 – 0.6 = 86 Ом, що ми й встановили на рис. 1.16.

Тепер всі параметри задані, можна зберегти схему і переходити до написання програми.

1.2.5 Написання програмного коду

Щоб почати писати програму, треба у панелі задач (4 на рис. 1.6) натиснути на кнопку  , а потім у полі 6 (рис. 1.6) написати наступний код (рис. 1.17).

Перед тим, як розглянути сам код, нагадаємо, що керування всіма модулями мікроконтролера відбувається за допомогою спеціальних комірок пам'яті, які називаються регістрами спеціального призначення. Кожен з цих регістрів має власне ім'я і розташований у закріпленій за ним адресі.

Порти вводу-виводу також конфігуруються за допомогою регістрів, яких у мікроконтролерах AVR є три типи:

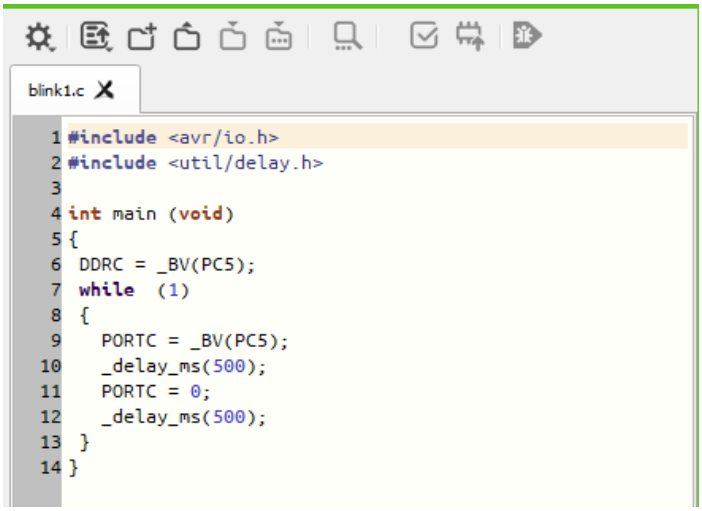


Рисунок 1.17 – Програмний код для блимання світлодіодом з частотою 1 Гц

- DDRx – регістр напрямку передачі даних. Замість літери «x» підставляється назва порту. Як вже було зазначено, порти називаються великими латинськими літерами. То ж, наприклад, для мікроконтролера

ATMega8, у якого є порти B, C та D, є, відповідно, три регістри DDR: DDRB, DDRC та DDRD.

Кожен з регістрів є 8-бітним (бо AVR є 8-бітною архітектурою). І кожен біт регістру конфігурує відповідний вивід мікроконтролеру. Наприклад, біт №0 регістру DDRB відповідає за вивід B0, біт №1 цього ж регістру відповідає за вивід B1 і т.д.

То ж, якщо якийсь біт регістру DDR дорівнює 0, то відповідний йому вивід працює як вхід, тобто на нього можна подавати сигнали з якихось зовнішніх пристроїв, а мікроконтролер буде їх зчитувати і якимось чином обробляти.

А якщо якийсь біт регістру DDR дорівнює 1, то відповідний вивід працює як вихід, тобто контролер сам встановлює логічний рівень, який буде на цьому виводі, що дозволяє передавати сигнал на якісь зовнішні пристрої.

- PORTx – має два значення, але ми в цій роботі поки що познайомимося лише з одним з них. Замість літери «x» тут так само підставляється ім'я порту. і так само кожен біт цього регістру відповідає за однойменний вивід.

Якщо якийсь вивід сконфігурований, як вихід (тобто значення біту в регістрі DDR для нього дорівнює 1), то значення будь-якого біту регістру PORTx задає вихідний рівень напруги на відповідному виводі: 1 відповідає високому рівню напруги (5 В), а 0 відповідає низькому рівню напруги (0 В).

- PINx – це регістр стану порту вводу-виводу. Значення бітів цього регістру відповідають логічному рівню на відповідному виводі. Наприклад, якщо на вході B3 присутній високий логічний рівень, то біт №3 регістру PINB буде дорівнювати 1, а якщо на цьому вході буде низький логічний рівень, то і значення біта №3 буде 0.

Тепер можна детально розглянути програмний код (рис. 1.17). Програма написана на мові C, то ж тут ми не будемо вдаватися в основи власне мови, бо вважається, що ви з ними вже знайомі з попередніх курсів. У цій програмі ми сконфігуруємо вивід C5 як вихід і будемо поперемінно видавати на нього високий і низький логічний рівні, вмикаючи та вимикаючи світлодіод з частотою 1 Гц.

Розглянемо програму порядково.

1. *#include <avr/io.h>* – тут ми підключаємо заголовний файл «avr/io.h». В цьому файлі описані відповідності між усіма регістрами мікроконтролера та їх реальними адресами, то ж додавати цей файл в програму обов'язково, якщо ви хочете використовувати назви регістрів у вашій програмі, а не просто їх адреси, які ще треба знайти в документації.
2. *#include <util/delay.h>* – цей заголовний файл потрібен у разі, якщо ми хочемо використовувати функції затримки роботи мікроконтролера на деякий час. Ці функції також використовуються досить часто, то ж рекомендується додавати і цей файл до програми, але це не обов'язково.
3. *Порожній рядок*

4. *int main (void)* – початок головної функції програми main. Зазвичай головна функція при програмуванні мікроконтролерів розділяється на дві частини: ініціалізація та нескінченний цикл. Частина ініціалізації виконується один раз при запуску мікроконтролера, і в ній ініціалізуються всі необхідні периферійні модулі за допомогою відповідних регістрів. У нескінченному циклі пишеться та частина програми, яка повинна виконуватися весь час, поки подана напруга живлення на мікроконтролер. Відповідно до нашого випадку, в ініціалізаційній частині ми повинні сконфігурувати вивід C5 як вихід. А в нескінченному циклі переключати стан цього виводу.

5. *{* – Відкриття головної функції main.

6. *DDRC = _BV(PC5);* – цей рядок заслуговує більш детального пояснення. В ньому ми зустрічаємо вже знайомий нам регістр DDRC, який відповідає за напрямок передачі інформації виводів порту C. Макрос PC5 відповідає виводу C5. Його значення насправді дорівнює просто 5, то ж можна було б замість PC5 написати просто 5, але PC5 виглядає більш наочно. Макрос _BV(x) встановлює 1 у біті, чий номер заданий параметром «x». Тобто, результатом запису _BV(PC5) стане число, у якому буде 1 в біті №5 і нулі в усіх інших бітах: 00100000_2. Таким чином, всі виводи порту C будуть сконфігуровані, як входи, окрім виводу C5, який сконфігурований, як вихід. До речі, цей рядок можна було б записати і таким чином: *DDRC = 0b00100000;* Префікс 0b перед числом означає, що воно подається у двійковому форматі.

7. *while (1)* – початок нескінченного циклу програми. При програмуванні на комп'ютері ми звикли, що такі записи, що утворюють нескінченні цикли, є вкрай небажаними, бо вони «підвішують» програму. У мікроконтролерах, навпаки, це – цілком нормальна практика, що використовується для того, щоб уникнути повторної ініціалізації програми. Якщо не використовувати нескінчений цикл, покажчик адресу програми пройде по всім адресам флеш-пам'яті та почне виконання з самого початку, що є небажаним.

8. *{* – Відкриття нескінченного циклу.

9. PORTC = _BV(PC5); – цей запис дуже схожий на рядок 6, тільки замість регістру DDRC тут записаний регістр PORTC. Результатом виконання цього рядку стане поява високого логічного рівня на виводі C5, що призведе до ввімкнення світлодіода.

10. *_delay_ms(500);* – затримка виконання програми на 500 мс. Функція _delay_ms описана у файлі «*util/delay.h*», який ми підключили у рядку 2, і вона виконує затримку на задану кількість мілісекунд. Для того, щоб отримати частоту блимання світлодіода в 1 Гц, треба на половину періоду, тобто на 500 мс, його ввімкнути, потім на 500 мс вимкнути, тому ми й задали таке значення затримки. Якщо не використовувати затримку, то світлодіод буде перемикатися з дуже високою частотою (приблизно 1/10 від тактової частоти, тобто десь 100 кГц).

11. *PORTC = 0;* – тут ми встановлюємо всі біти регістру PORTC у нулі, таким чином, встановлюючи на всіх виводах порту C (включно з C5) низький логічний рівень, що призведе до вимкнення світлодіода.

12. _delay_ms(500); – знову затримка на виконання програми на 500 мс, тепер для того, щоб утримати світлодіод вимкненим. Після цього рядку виконання програми знову повернеться до початку нескінченного циклу (до рядку 9), і знов ввімкне світлодіод, і так далі.

13. *}* – Закриття нескінченного циклу.

14. *}* – Закриття головної функції програми.

Далі треба зберегти написаний програмний код у файл. Цей файл повинен мати розширення «.c», яке треба записати власноруч, встановивши тип файлів, як «All files (*.*)» (рис. 1.18).

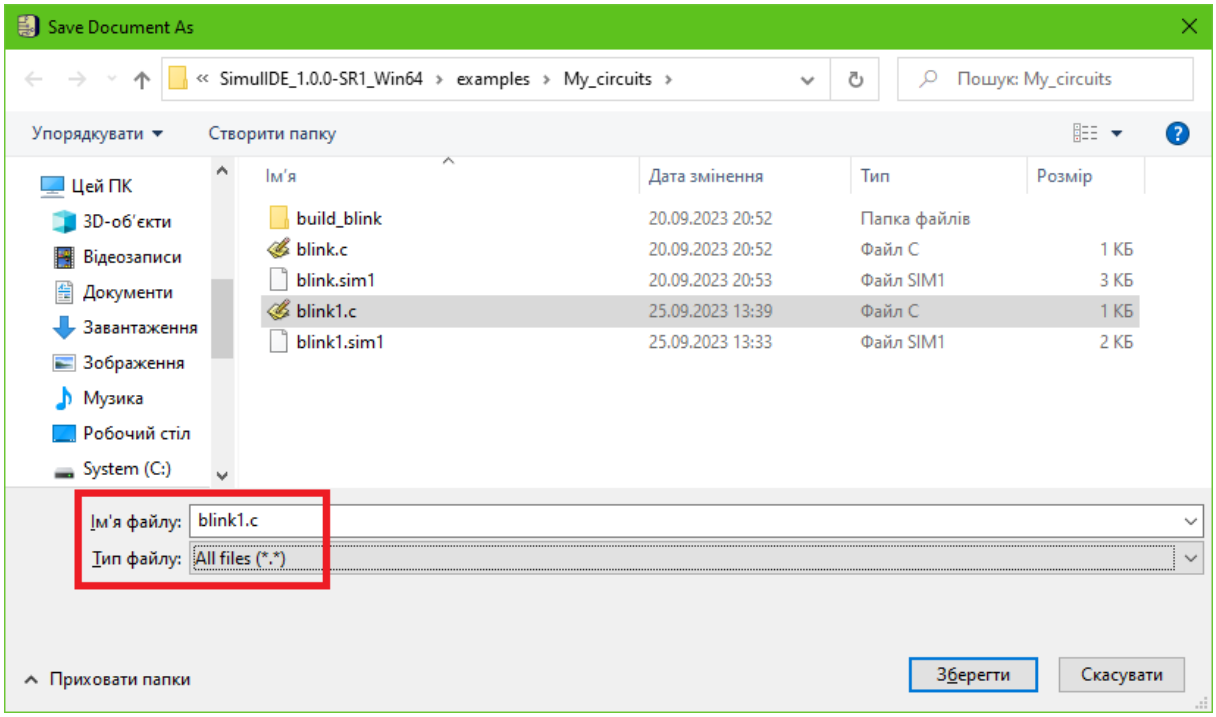


Рисунок 1.18 – Збереження файлу програми

1.2.6 Компіляція файлу прошивки та запуск симуляції

Тепер треба з програмного коду створити файл прошивки. Для цього нам нарешті знадобиться компілятор AVR GCC, який ми завантажили та розпакували раніше.

Спочатку треба показати програмі SimulIDE, де саме знаходиться цей компілятор. Для цього натискаємо кнопку  та у випадаючому меню обираємо пункт «Compiler Settings» (рис. 1.19).

У відкритому вікні треба обрати зі списку компілятор («Compiler») типу «Avrgcc» (рис. 1.20).

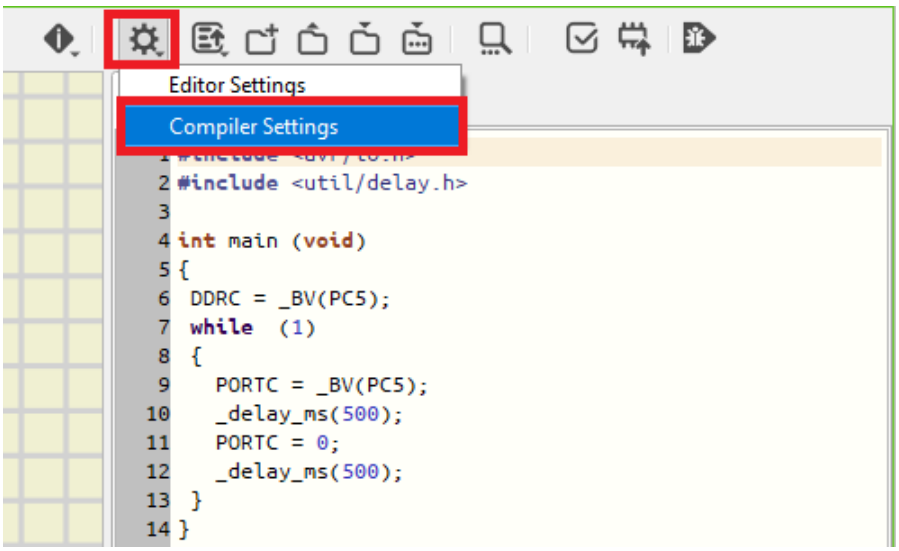


Рисунок 1.19 – Відкриття вікна налаштувань компілятора

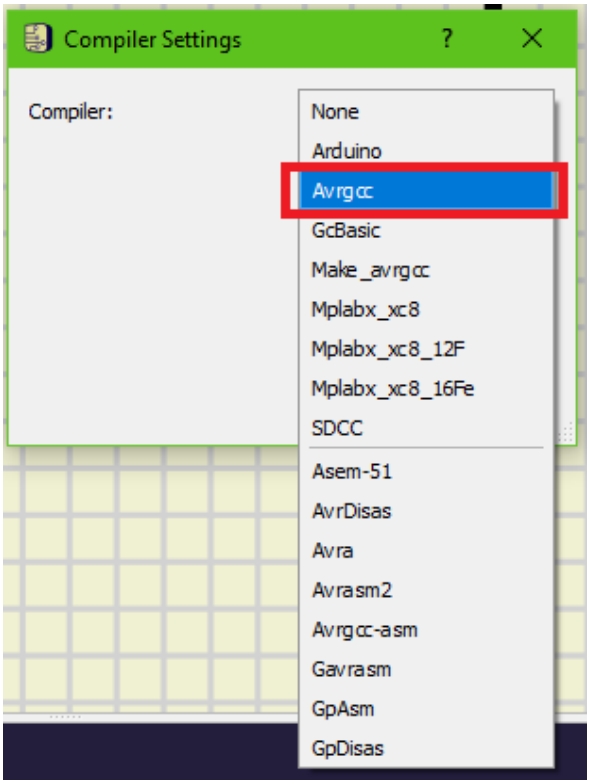


Рисунок 1.20 – Вибір компілятора

Тепер треба вказати шляхи, де встановлений власне компілятор «Tool Path» та де знаходяться заголовні файли «Include Path» (рис. 1.21).

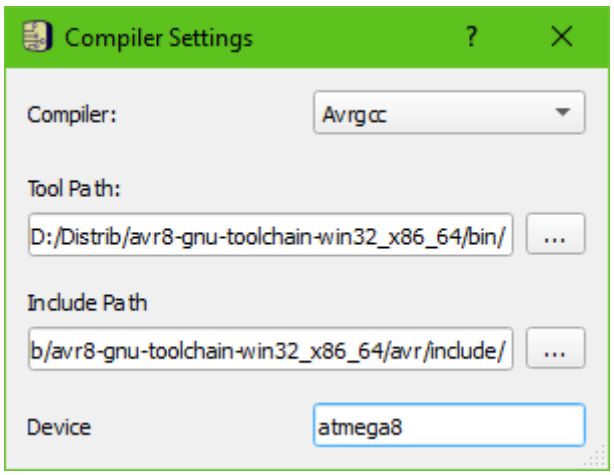


Рисунок 1.21 – Налаштування компілятора

Компілятор знаходиться у теці «avr8-gnu-toolchain-win32_x86_64/bin/», де «avr8-gnu-toolchain-win32_x86_64» – це назва теки, в яку розпакувався архів «avr8-gnu-toolchain-3.7.0.1796-win32.any.x86_64.zip». У вас ця назва може дещо відрізнятися, то ж в загальному вигляді цей шлях знаходиться за таким шляхом «<шлях до теки, куди розпакувався завантажений архів>**/bin/**».

Заголовні файли знаходяться за шляхом «<шлях до теки, куди розпакувався завантажений архів>**/avr/include/**».

У полі «Device» треба задати тип мікроконтролера, для якого буде створюватися прошивка. Тобто, в нашому випадку це буде «atmega8» (рис. 1.21).

Тепер можна повернутися на головний екран і спробувати скомпілювати програму, натиснувши на кнопку  . Якщо у програмі немає помилок, і якщо всі шляхи вказані вірно, то у полі 8 (рис. 6) ви повинні побачити інформацію про успішне створення файлу «.hex» (рис. 1.22).

```
In file included from D:/Distrib/SimulIDE_1.0.0-SR1_Win64/examples/My_circuits/blink1.c:
2:0:
d:\distrib\avr8-gnu-toolchain-win32_x86_64\avr\include\util\delay.h:92:3: warning:
#warning "F_CPU not defined for <util/delay.h>" [-Wcpp]
 # warning "F_CPU not defined for <util/delay.h>"
   ^~~~~~~

Executing:
"D:/Distrib/avr8-gnu-toolchain-win32_x86_64/bin/avr-objcopy" -j .text -j .data -O ihex
D:/Distrib/SimulIDE_1.0.0-SR1_Win64/examples/My_circuits/build_blink1/blink1.elf D:/
Distrib/SimulIDE_1.0.0-SR1_Win64/examples/My_circuits/build_blink1/blink1.hex
```

Рисунок 1.22 – Інформація про успішне створення файлу прошивки

Також на рис. 1.22 можна побачити попередження, що «F_CPU not defined for <util/delay.h>«. Це означає, що ми не задали значення макросу F_CPU, який використовується файлом «delay.h», щоб правильно розраховувати затримку. За замовчанням ця частота встановлена в самому цьому файлі, як 1 МГц. Саме тому в налаштуваннях мікроконтролера ми встановили саме цю частоту (рис. 1.12).

Тепер натискаємо на кнопку  , щоб загрузити створений файл у мікроконтролер. У вікні 8 бачимо інформацію, що файл завантажено успішно (рис. 1.23).

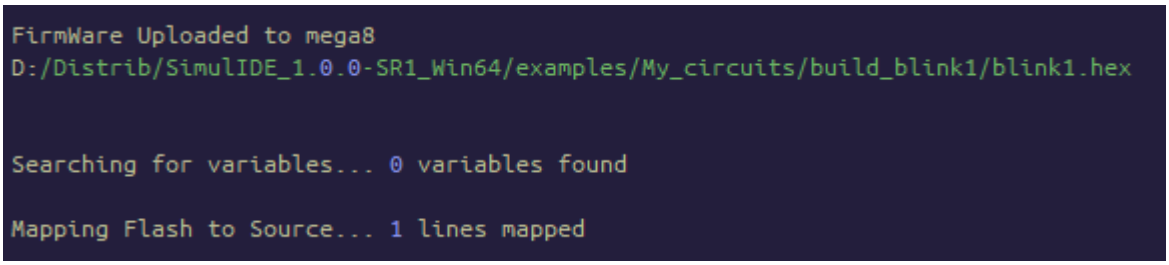


Рисунок 1.23 – Інформація про успішне завантаження файлу прошивки у мікроконтролер

Тепер треба натиснути на кнопку  , щоб запустити симуляцію. Якщо все зроблено вірно, то можна побачити, що світлодіод блимає жовтим кольором з частотою 1 Гц (рис. 1.24).

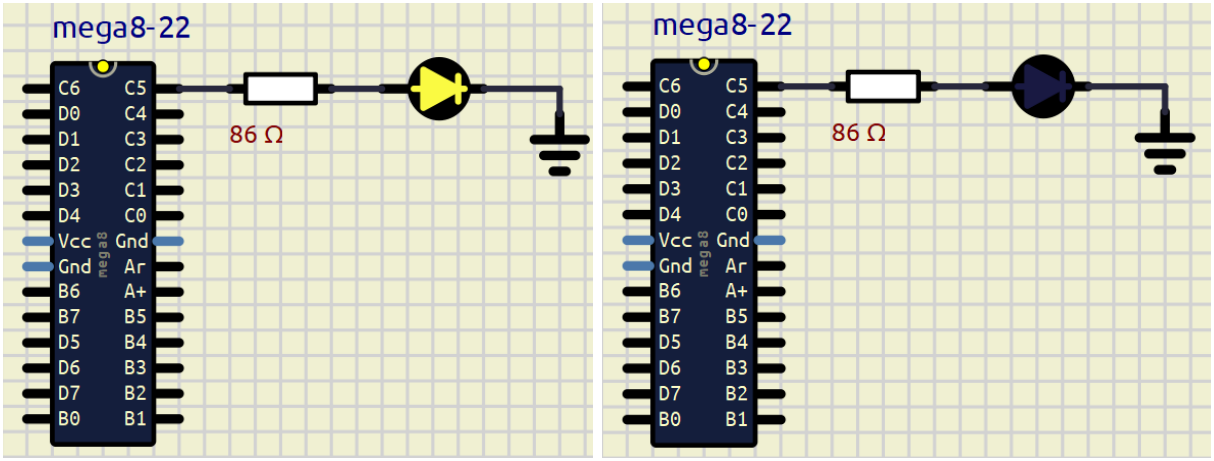


Рисунок 1.24 – Світлодіод у ввімкненому та вимкненому стані

На цьому завдання поточної практичної роботи можна вважати виконаним.

1.3 Завдання до практичної роботи

1. Завантажити та розпакувати програму SimulIDE та компілятор AVR GCC.

2. Ознайомитись з поданням програми SimulIDE та її інтерфейсом.

3. Створити принципову електричну схему, схожу на рис. 1.11, яка включає мікроконтролер, обраний згідно з варіантом завдань (табл. 1.3), резистор та світлодіод. Останній підключити до виводу, також вказаному в таблиці 1.3. Змінити колір світлодіода відповідно до таблиці 3. У другому стовпчику таблиці 1.3 у дужках вказано справжню назву мікроконтролеру, яку треба вводити у налаштуваннях компілятора (рис. 1.21).

4. Написати програму блимання світлодіодом з частотою, вказаною в таблиці 1.3.

5. Створити файл прошивки, загрузити його у мікроконтролер та переконатися, що все працює, як треба.

Таблиця 1.3 – Варіанти завдань до практичної роботи №1

Номер варіанту	**Мікроконтролер**	**Вивід для підключення світлодіоду**	**Частота блимання, Гц**	**Колір світлодіоду**
1	tiny44 (attiny44)	A0	2	Жовтий
2	mega48 (atmega48)	C4	4	Червоний
3	tiny45 (attiny45)	B1	0.5	Зелений
4	mega88 (atmega88)	D6	0.25	Синій
5	tiny2313 (attiny2313)	D3	2.5	Помаранчевий
6	mega168 (atmega168)	C1	5	Бузковий
7	tiny84 (attiny84)	B2	0.2	Жовтий
8	mega328 (atmega328)	D7	0.4	Червоний
9	tiny85 (attiny85)	B5	0.8	Зелений
10	m48 TQFP (atmega48)	C2	6	Синій
11	tiny24 (attiny24)	A7	0.3	Помаранчевий
12	m88 TQFP (atmega88)	B0	3	Бузковий

1.4 Питання для самоперевірки

1. Призначення елементів інтерфейсу програми SimulIDE.
2. Додавання та налаштування електронних компонентів.
3. Написання програм у програмі SimulIDE.
4. Завантаження файлу прошивку у мікроконтролер у програмі SimulIDE.
5. Симуляція електричних схем у програмі SimulIDE.

1.5 Перелік рекомендованих джерел

1. SimulIDE Tutorial. URL: https://simulide.com/p/simulidekb/ (дата звернення: 02.07.2024).

2. ATMega8 : технічна документація на мікроконтролер. URL:: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2486-8-bit-AVR-microcontroller-ATmega8_L_datasheet.pdf (дата звернення: 02.07.2024).

3. 8-bit AVR® MCUs : інформація про мікроконтролери. URL: https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/8-bit-mcus/avr-mcus (дата звернення: 02.07.2024).

2 ПРАКТИЧНА РОБОТА №2 «ПРОГРАМУВАННЯ ЦИФРОВИХ ПОРТІВ ВВОДУ-ВИВОДУ МІКРОКОНТРОЛЕРА AVR»

2.1 Завдання

Освоєння прийомів програмування мікроконтролерів на прикладі портів вводу-виводу мікроконтролерів AVR.

Завдання практичної роботи:

– Створення електричної схеми у програмі SimulIDE відповідно до завдання на практичну роботу.

– Написання програми для мікроконтролеру відповідно до завдання на практичну роботу.

2.2 Теоретичні дані

2.2.1 Робота з портами вводу-виводу мікроконтролерів AVR[1]

Порти вводу-виводу є основним засобом зв'язку мікроконтролерів AVR з навколишнім світом. Спрощена схема порту вводу-виводу показана на рисунку 2.1.

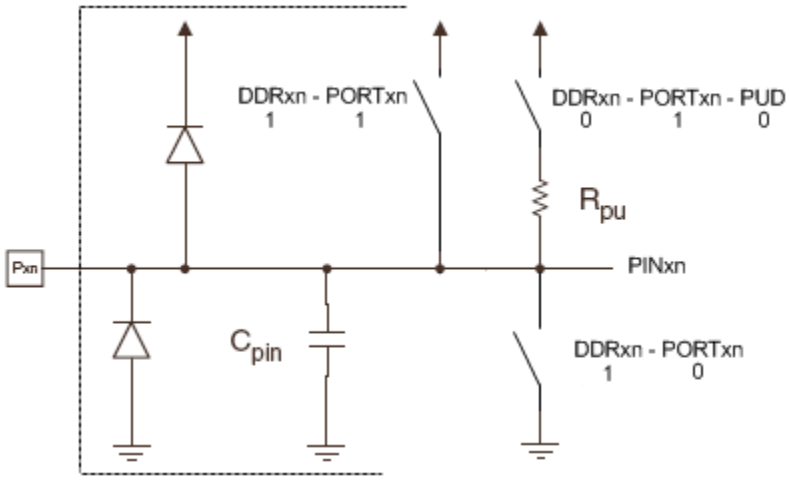


Рисунок 2.1 – Схема порту вводу-виводу

На кожній ніжці МК стоять захисні діоди. На них сильно не розраховують, якщо напруга на вході перевищить 5,5 В, вони напевно не витримають. Ніжка МК здатна пропустити через себе струм не більше 20 мА. Кожний вхід мікроконтролера має паразитну ємність C_{pin}. Далі йдуть перемикачі (у МК замість перемикачів стоять польові транзистори). Кожний перемикач замикається при певній конфігурації регістрів

[1] Даний підрозділ викладений на основі матеріалів [1]

керування портів вводу-виводу - DDRxn, PORTxn, PINxn і біта 2 (PUD) регістру SFIOR. x - ім'я порту (наприклад «B»), n - номер біта порту (0-7). У різних мікроконтролерів різна кількість портів. Наприклад, розглянемо МК Atmega8.

З рисунку 2.2 видно, що в МК Mega8 три порти вводу-виводу. Два повні порти (по 8 біт) - «PB» і «PD» і один неповний (7 біт) - «PC».

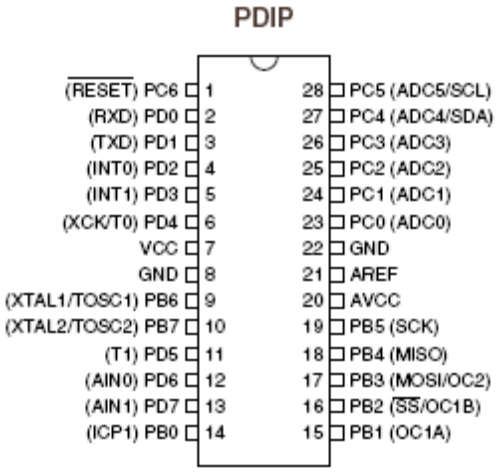


Рисунок 2.2 – Мікроконтролер Atmega8

PINxn – Регістр читання стану порту. Із цього регістру можна тільки читати. Цей регістр містить інформацію про логічний рівень на виводах МК і це не залежить від налаштувань порту.

DDRxn – регістр напрямку порту.

PORTxn – регістр керування станом виводу.

Біт 2 SFIOR: PUD(pullup disable) – забороняє резистор, що підтягує, незалежно від того дозволений він регістром керування чи ні («0» - дозволений pullup, «1» - заборонений pullup).

Варіанти установок бітів регістрів показано в таблиці 2.1.

Таблиця 2.1 – Варіанти установок бітів регістрів вводу-виводу

DDxn	PORTxn	PUD (у SFIOR)	Стан виводу	Підтягу-вання	Коментар
0	0	X	Вхід	Ні	Високоімпедансний стан
0	1	0	Вхід	Так	Pxn буде джерелом струму, якщо зовнішня схема підключена до нижчої напруги
0	1	1	Вхід	Ні	Високоімпедансний стан
1	0	X	Вихід	Ні	Вихід з низьким рівнем напруги
1	1	X	Вихід	Ні	Вихід з високим рівнем напруги

Вихід (DDRxn = 1). Тут якщо в PORTxn записати «1», на виході й буде логічна одиниця, якщо записати «0» - логічний нуль.

Вхід (DDRxn = 0). Якщо в PORTxn записати «0», це буде режим високоімпедансного входу (DDRxn=0, PORTxn=0 – включений за замовчуванням). Якщо подивитися на рисунок вище, цей режим відповідає ситуації коли всі перемикачі розімкнуті, при цьому вхідний опір входу можна вважати рівним нескінченності.

Якщо в PORTxn записати «1», це буде режим з резистором, що підтягує.

Також кожна ніжка МК має альтернативні функції. За замовчуванням вони відключені. Якщо альтернативна функція включена, то ніжка управляється периферійним обладнанням і тоді запис в DDRxn і PORTxn нічого не дає.

Якщо ніжка МК не використовується, тоді рекомендується забезпечити на ній певний рівень. Це потрібно щоб зменшити енергоспоживання мікроконтролера через наведення виникаючих на виводах. Найпростіший спосіб забезпечити рівень - включити резистор, що підтягує. Підключати невикористовувані ніжки безпосередньо до шини живлення або до шини заземлення не рекомендується. Це може привести до надмірного струму, якщо вивід випадково налаштований на вихід.

2.2.2 Бітові операції на мові С[2]

Біт може приймати одне з двох можливих значень: 1 або 0 (аналогічно: увімк/вимк, встановлено/очищено, високий/низький логічний рівень). Декілька бітів утворюють число в двійковому вигляді, де кожен біт є одним розрядом даного числа. В мікроконтролерах AVR 8 бітів з'єднані разом, формують один **байт**, в якому молодший біт (LSB — **L**east **S**ignificant **B**it) знаходиться справа. Нумерація бітів починається з нуля від молодшого біта. Для прикладу розглянемо десяткове число 15. Його можна відобразити у двійковій 8-бітній формі:

00001111

Чотири молодші біти встановлені.

Якщо є потреба більш докладно зупинитися на темі перетворень між десятковими, двійковими та шістнадцятковими числами, то можна пошукати статті до цієї теми, або скористатись цією таблицею. Інший приклад, десяткове число 40, відображене у двійковій формі:

00101000

Біти 3 та 5 встановленні.

У програмах мовою С для мікроконтролерів AVR числові значення можуть виражатися у трьох формах, залежно від контексту або вибору програміста. Шістнадцяткові числа визначаються з використанням 0x-префіксу, двійкові — 0b-префіксу. Наступний С код демонструє три варіанти ініціалізації змінних десятковим значенням 15.

```
uint8_t a = 15; // десяткове значення
```

[2] Даний підрозділ викладений на основі матеріалів [2]

uint8_t b = 0x0F; // шістнадцяткове
uint8_t c = 0b00001111; // двійкове
uint8_t — один з типів даних рівноширокого цілого типу, зі стандарту C99. Стандарт передбачає 8-бітний беззнаковий цілий тип. Типи даних стандарту C99 будуть і надалі використовуватись у цьому курсі.

2.2.3 Бітові операції[3]

Виходячи з того, що кожен окремий біт — носій важливої інформації при програмуванні для AVR мікроконтролерів, бітові операції є значною складовою цього процесу.

У результаті виконання побітової операції **AND**, вихідні біти буде встановлено лише у випадку, коли у обох операндах вони дорівнюють одиниці. Іншими словами, біт n буде встановлено, якщо у першому операнді та (**AND**) у другому операнді біт n також встановлено.

У мові програмування C побітовий оператор AND позначається одинарним знаком амперсанд.

uint8_t a = 0xAA; // 10101010
uint8_t b = 0x0F; // 00001111
uint8_t c = a & b; // 00001010

У результаті виконання побітової операції **OR**, вихідні біти буде встановлено у випадку, якщо хоча б в одному з операндів вони також були встановлені. Іншими словами, біт n буде встановлено, якщо у першому операнді або (**OR**) у другому операнді біт n також встановлено.

У мові програмування C для позначення побітової операції OR використовується одинарна вертикальна риска (|).

uint8_t a = 0xAA; // 10101010
uint8_t b = 0x0F; // 00001111
uint8_t c = a | b; // 10101111

У результаті виконання побітової операції **XOR**(виключне OR) вихідні біти буде встановлено тільки в тому разі, якщо в одному з операндів вони були встановлені, а в іншому ні. Іншими словами, біт n буде встановлено, якщо **виключно **в одному з операндів біт n також встановлено.

Оператор XOR у мові C позначається символом каретки (^).

uint8_t a = 0xAA; // 10101010
uint8_t b = 0x0F; // 00001111
uint8_t c = a ^ b; // 10100101

Операція **NOT**, відома як порозрядне доповнення, є унарною операцією. Це означає, що така операція потребує лише одного операнда, а не двох, як інші. NOT просто перетворює кожен біт на протилежний.

[3] Даний підрозділ викладений на основі матеріалів [2]

Тобто, кожен біт, що дорівнює 1, перетворюється на 0, а кожен, що дорівнює 0, перетворюється на 1.

У мові програмування С операція NOT позначається знаком тильда (~).

```
uint8_t a = 0xAA; // 10101010
uint8_t b = ~a; // 01010101
```

Операція зсуву (**shift**), переміщує всі біти вліво або вправо. При зсуві вліво, біти зсуваються «назовні» зліва, та нульові біти зсуваються «всередину» справа.

В мові програмування С два знаки «менше ніж» () позначають операцію зсуву вправо. З правої сторони від оператора вказується числове значення — кількість розрядів для зсуву.

```
uint8_t a = 0x99; // 10011001
uint8_t b = a<<1; // 00110010
uint8_t c = a>>3; // 00010011
```

2.2.4 Очищення та встановлення бітів[4]

Встановлення та очищення окремого біту, без зміни всіх інших бітів, одна з найважливіших задач при програмуванні мікроконтролерів AVR. Ви будете користуватись цією схемою знов і знов.

Отже, загалом для керування окремо взятим бітом, зазвичай, потрібен байт в якому нас цікавить лише один встановлений біт. Цей байт надалі, за допомогою побітових операцій, можна використовувати для керування потрібним бітом. Такий принцип керування бітами називається **бітова маска**. Для прикладу, розглянемо бітову маску для біта номер 2: 00000100, та бітову маску для біту номер 6: 01000000.

Якщо взяти число 1, то маємо двійкове число в якому встановлено лише нульовий розряд, але за допомогою зсуву вліво на деяку кількість розрядів, можна отримати потрібну маску. Для прикладу наведена бітова маска для біту номер 2, яку отримали з числа 1 за допомогою зсуву вліво на два розряди.

Для встановлення потрібного біту у мові С, застосовується операція OR до потрібного байту разом з бітовою маскою.

```
uint8_t a = 0x08; // 00001000
// встановлення біту 2
a |= (1<<2); // 00001100
```

Для встановлення більше ніж одного біту використовується декілька операторів OR.

```
uint8_t a = 0x08; // 00001000
// встановлення бітів 1 та 2
a |= (1<<2)|(1<<1); // 00001110
```

[4] Даний підрозділ викладений на основі матеріалів [2]

Для очищення біту застосовується операція NOT до бітової маски, у результаті тільки потрібний біт буде не встановлено, після цього потрібно застосувати операцію AND до потрібного байту разом з бітовою маскою.

```
uint8_t a = 0x0F; // 00001111
// очищення біту 2
a &= ~(1<<2); // 00001011
```

Так само — для очищення більше ніж одного біту використовується декілька операторів OR.

```
uint8_t a = 0x0F; // 00001111
// очищення бітів 1 та 2
a &= ~((1<<2)|(1<<1)); // 00001001
```

Для того, щоб, так би мовити, перемикати потрібний біт, встановлюючи його, або в 1, або в 0, можна скористатись операцією XOR та, знов ж таки, — бітовою маскою.

```
uint8_t a = 0x0F; // 00001111
// змінення значення біту 2
a ^= (1<<2); // 00001011
a ^= (1<<2); // 00001111
```

2.2.5 Макрос керування значенням біту _BV()[5]

В AVR Libc визначено макрос _BV() для керування значенням біту. Цей макрос зручно використовувати для отримання потрібної бітової маски. Головна ідея в тому, щоб зробити код більш читабельним використовуючи побітовий зсув вліво. Застосування _BV(n) еквівалентно вживанню (1<<n).

```
// встановлення біту 0, використовуючи _BV()
a |= _BV(0);
// встановлення біту 0, використовуючи зсув
a |= (1<<0);
```

Який метод використовувати — залежить від вас. Ви побачили обидва методи в дії і тепер зможете вибрати, що для вас зручніше. Також, треба зазначити, що оскільки макрос _BV() є унікальним для GCC, то такий метод, на відміну від (1<<n), не сумісний з іншими компіляторами. Але цим зазвичай не дуже переймаються аматори, та й нерідко _BV() виявляється зручнішим для початківців.

2.2.6 Перевірка значення біту[5]

В операторах умовного переходу або в циклах іноді треба перевіряти значення окремого біту у байті. Для цього використовується операція побітового AND.

[5] Даний підрозділ викладений на основі матеріалів [2]

Перевірка на те, чи є значення біту логічною одиницею записується наступним чином:

```
uint8_t a = 0x0F; // 00001111
// Перевірка, чи дорівнює одиниці біт №5 у числі a
if (a & (1 << 5))
{
//Зробити щось
}
```

Вираз (a & (1 << 5)) буде істинним тільки тоді, коли біт №5 у числі a дорівнює 1, у всіх інших випадках цей вираз буде хибним.

У цих перевірках можна також використовувати макрос _BV():

```
if (a & _BV(5))
{
//Зробити щось
}
```

Перевірка на те, чи є значення біту логічним нулем виконується аналогічним чином, тільки результат перевірки інвертується:

```
uint8_t a = 0x0F; // 00001111
// Перевірка, чи дорівнює нулю біт №5 у числі a
if (!(a & (1 << 5)))
{
//Зробити щось
}
```

Можна робити інверсію не всього результату, а тільки числа, яке перевіряється. Результат перевірки буде таким самим

```
if (~a & (1 << 5))
{
//Зробити щось
}
```

З точки зору процесору, перший варіант запису є кращим, оскільки в ньому є команди умовного переходу і якщо результат операції дорівнює нулю, і якщо він відрізняється від нуля. У другому випадку треба виконати додаткову операцію інверсії, що віднімає процесорний час та займає додаткову пам’ять.

2.3 Приклад програми

Виконаємо наступне завдання на базі мікроконтролера ATMega8. Підключити два світлодіоди LED1 та LED2 до виводів PC5 та PC2, відповідно, та дві кнопки S1 та S2 до виводів PB5 та PB1, відповідно. При утриманні натиснутою кнопки S1 блимати світлодіодом LED1 з частотою 2 Гц, а при відпусканні кнопки світлодіод погасити. При натисканні кнопки S2 перемикати світлодіод LED2 у протилежний стан один раз на кожне натискання.

2.3.1 Принципова електрична схема

Принципова електрична схема, що реалізує поставлену задачу, показана на рис. 2.3.

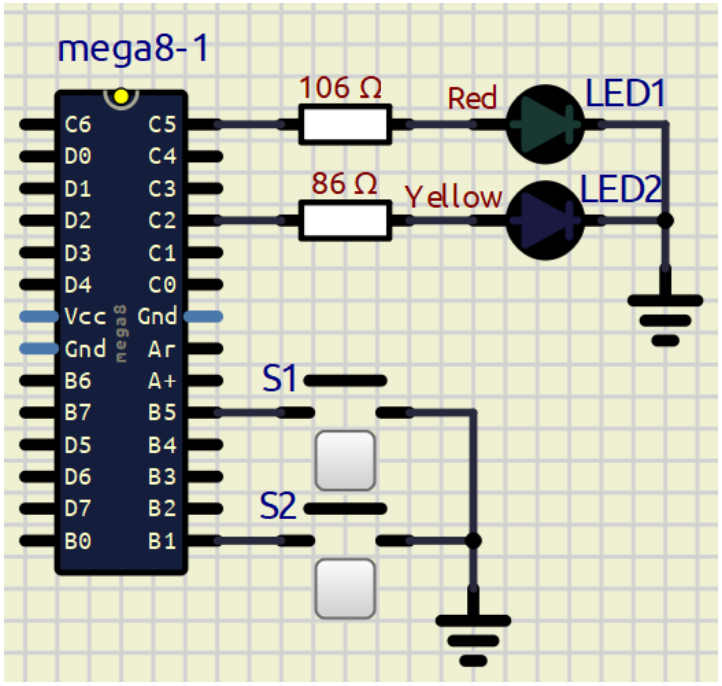


Рисунок 2.3 – Принципова електрична схема

Окрім вже відомих нам компонентів з попередньої практичної роботи, у цій схемі присутні дві кнопки, які у SimulIDE знаходяться у підзаголовку Switches і називаються Push (рис. 2.4).

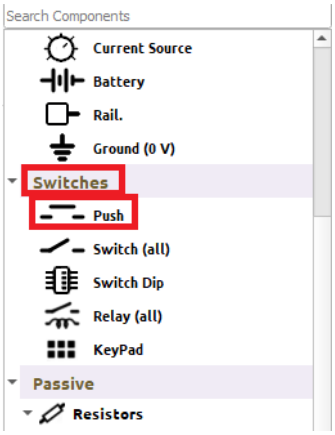


Рисунок 2.4 – Кнопка у бібліотеці компонентів

Світлодіоди LED1 та LED2 підключаються так само, як і в попередній практичній роботі. Кнопки S1 та S2 підключаються так, що один їх кінець під'єднаний до виводу мікроконтролера, а інший – до нульового проводу. Таким чином, коли кнопка натиснута, на вході мікроконтролера, до якого

вона підключена, напруга дорівнює 0, що відповідає логічному 0 у відповідному біті регістру PINx.

Але тоді, коли кнопка не натиснута, на вході повинна бути висока напруга, щоб можна було чітко розрізнити стани кнопки. Для того, щоб забезпечити цю високу напругу, як раз і використовуються підтягувальні резистори (рис. 2.1). Їхній опір складає кілька десятків кілоом, що набагато менше опору кнопки у ненатиснутому стані, тому на вході мікроконтролера встановлюється високий логічний рівень. Коли кнопку натискають, її опір різко знижується майже до 0 Ом, і тепер на вході з'являється низький логічний рівень.

У мікроконтролерах AVR внутрішні резистори можуть підтягувати потенціал тільки до напруги живлення (у той час, як деякі більш просунуті мікроконтролери мають внутрішні резистори, що можуть підтягувати потенціал, як до напруги живлення, так і до 0), тому зазвичай кнопки до них підключаються саме так, як вказано на рис. 2.3.

2.3.2 Програмний код

Розглянемо тепер програму, що реалізує поставлену задачу (рис. 2.5).

Рядки 1-4 вже зустрічалися у попередній практичній роботі, то ж ми на них не будемо зупинятися тут.

У рядку 6 виводи PC5 та PC2 конфігуруються, як виходи (оскільки до них під'єднані світлодіоди), шляхом встановлення відповідних бітів у регістрі DDRC. Зверніть увагу, що тут замість простого присвоєння ми використовуємо операцію побітове АБО для того, щоб змінити тільки біти PC5 та PC2, залишаючи всі інші біти у тому стані, в якому вони були до виконання цієї операції. В конкретному даному випадку можна було б використати й операцію звичайного присвоєння (=), але **в загальному випадку краще користуватися побітовим АБО для встановлення бітів та побітовим І з інверсією для їх скидання, як показано в п. 2.2.4**.

У рядку 7 ми скидаємо біти PB1 та PB5 у регістрі DDRB, таким чином конфігуруючи їх як входи, оскільки до них під'єднані кнопки S2 та S1, відповідно. Знову ж таки, цей рядок не є обов'язковим, адже після скидання всі виводи сконфігуровані, як входи. Але правилом гарного тону у програмуванні мікроконтролерів є явна конфігурація усіх використаних виводів (а у деяких випадках, і невикористаних).

У рядку 8 ми встановлюємо біти PB5 та PB1 у регістрі PORTB. Оскільки відповідні виводи сконфігуровані як входи, то встановлення цих бітів вмикає внутрішні підтягувальні резистори на цих виводах. Для чого вони потрібні на входах, до яких підключені кнопки, було написано у попередньому розділі.

На цьому конфігураційна частина програми закінчується, і з 9 рядку починається головний безкінечний цикл програми. У ньому обробляються

натискання на кнопки: S2 (рядки 11-23) та S1 (рядки 24-40). Як можна побачити, обробка кнопок є досить нелегкою задачею.

```
 1 #include <avr/io.h>
 2 #include <util/delay.h>
 3
 4 int main (void)
 5 {
 6   DDRC |= _BV(PC2) | _BV(PC5);
 7   DDRB &= ~(_BV(PB1) | _BV(PB5));
 8   PORTB |= _BV(PB1) | _BV(PB5);
 9   while  (1)
10   {
11     if (!(PINB & _BV(PB1)))
12     {
13       _delay_ms(30);
14       if (!(PINB & _BV(PB1)))
15       {
16         while (!(PINB & _BV(PB1)));
17         _delay_ms(30);
18         if (PINB & _BV(PB1))
19         {
20           PORTC ^= _BV(PC2);
21         }
22       }
23     }
24     if (!(PINB & _BV(PB5)))
25     {
26       _delay_ms(30);
27       if (!(PINB & _BV(PB5)))
28       {
29         while (!(PINB & _BV(PB5)))
30         {
31           PORTC ^= _BV(PC5);
32           _delay_ms(250);
33         }
34         _delay_ms(30);
35         if (PINB & _BV(PB1))
36         {
37           PORTC &= ~_BV(PC5);
38         }
39       }
40     }
41   }
42 }
```

Рисунок 2.5 – Програмний код

Спочатку завжди перевіряється, чи дорівнює нулю біт регістру PINx, що відповідає виводу, до якого підключена кнопка. У рядку 11 як раз і

відбувається це: ми перевіряємо, чи дорівнює нулю біт PB1 регістру PINB. Якщо це так, це значить, що кнопка натиснута. Але якщо бути більш точним, це означає, що кнопка тільки у процесі натискання. Контакти реальних кнопок є неідеальними, і в момент натискання може відбуватися так званий брязкіт контактів. Це явище полягає у зміні стану кнопки під час натискання або відпускання через те, що контакти то замикаються, то розмикаються з високою частотою. Таким чином, навіть одне натискання кнопки може бути зчитане мікроконтролером, як декілька послідовних натискань через те, що він опитує стан виводів з дуже високою частотою. Для програмної боротьби з брязкотом контактів вводять затримку величиною 20-30 мс. Це час, протягом якого відбувається перехідний процес натискання кнопки. По закінченню цього часу вважається, що контакти вже щільно прилягають один до одного (або повністю розімкнені), і стан кнопки є визначеним.

То ж у рядку 13 ми виконуємо цю затримку на брязкіт контактів. Після цього ми ще раз перевіряємо стан біту PB1 регістру PINB (рядок 14). Якщо він все ще 0, це означає, що кнопка таки була натиснута. Після цього ми просто чекаємо, поки кнопка знаходиться у натиснутому стані, і нічого не робимо, використовуючи цикл типу while (рядок 16). Це також один із стандартних методів обробки натискання кнопки. Зазвичай, якусь однократну дію, яку повинна робити кнопка, виконують після її відпускання, щоб уникнути повторного виконання. Якщо ж потрібно зробити щось, поки кнопка утримується натиснутою, це якраз і робиться всередині цього циклу (це ми побачимо при розгляданні другої кнопки).

З циклу while ми можемо вийти тільки при умові, що аргумент циклу стає хибним, тобто біт PB1 регістру PINB перестає бути рівним 0. Це означає, що почався процес відпускання кнопки. Цей процес також супроводжується брязкотом контактів. То ж ми робимо ще одну затримку величиною в 30 мс (рядок 17), після чого перевіряємо, чи дійсно біт PB1 регістру PINB став рівним 1 (рядок 18). І лише в тому випадку, коли виконується ця умова, ми переходимо до власне дії, яка повинна відбуватися при натисканні кнопки. Тобто усі рядки від 11 до 19 – це просто обробка натискання кнопки, а корисна дія, що виконується при цьому, займає лише рядок 20.

В ньому відбувається перемикання біту PC2 регістру PORTC у протилежний стан за допомогою операції «Побітове виключне АБО», або XOR (див. п. 1.2.2). При цьому світлодіод LED2 перемкнеться у стан, протилежний тому, у якому він був до того. Це як раз те, що і потрібно виконати, відповідно до завдання.

Рядки 24-29 аналогічні рядкам 11-16, тільки відносяться до виводу (і, відповідно, біту) PB5, до якого підключена кнопка S1. Відміна наступає у рядку 30, коли ми замість того, щоб просто очікувати, поки кнопка буде утримуватися натиснутою, виконуємо якусь повторну дію. В даному випадку це перемикання світлодіоду LED1 у протилежний стан (рядок 31)

з наступною затримкою у 250 мс (рядок 32), для того, щоб блимання відбувалося з частотою 2 Гц.

Частоті 2 Гц відповідає період 1/2 = 0.5 с = 500 мс. Світлодіод повинен половину періоду бути ввімкненим, а половину періоду – вимкненим, тому затримка між переключеннями стану світлодіоду повинна бути 500/2 = 250 мс.

Після відпускання кнопки S1 (коли умова виконання циклу while у рядку 29 стає хибною), ми знов виконуємо затримку на брязкіт контактів (рядок 34), після чого перевіряємо, чи кнопка дійсно відпущена (рядок 35), і якщо так, то вимикаємо світлодіод LED1, скидаючи біт PC5 у регістрі PORTC (рядок 37).

Ось, в принципі, і вся програма. Тепер можна скомпілювати файл .hex, завантажити його у мікроконтролер та запустити симуляцію. Для «натискання» на кнопку треба натиснути на сірий квадрат під позначенням кнопки за допомогою лівої кнопки миші. Якщо все написано вірно, робота програми буде відповідати завданню.

2.4 Завдання до практичної роботи

1. Створити принципову електричну схему, згідно з варіантом завдань (таблиця 2.1).
2. Написати програму, що виконує поставлене завдання (табл. 2.1).
3. Створити файл прошивки, загрузити його у мікроконтролер та переконатися, що все працює, як треба.

Таблиця 2.1 – Варіанти завдань до практичної роботи №2

Варіант	Мікроконтролер	Завдання
1	tiny44 (attiny44)	Підключити кнопку до виводу PA6, а світлодіод до виводу PB2. При натисканні на кнопку протягом 1 секунди, перемикати світлодіод у протилежний стан. Якщо тривалість натискання менша за 1 секунду, нічого не здійснювати.
2	mega48 (atmega48)	Підключити кнопку до виводу PD0, а світлодіод до виводу PC3. При першому натисканні на кнопку починати блимання світлодіодом з частотою 2 Гц. При наступному натисканні кнопки блимання заборонити, а світлодіод погасити. Процес повторювати циклічно.
3	tiny45 (attiny45)	Підключити кнопку до виводу PB2, а світлодіод до виводу PB1. При запуску контролера здійснювати блимання світлодіодом із частотою 1 Гц. При натисканні на кнопку цю частоту змінювати циклічно таким чином: 2 – 4 – 8 – 1 Гц. Одному натисканню кнопки відповідає одна зміна частоти.
4	mega88 (atmega88)	Підключити кнопку до виводу PC1, а світлодіод до виводу PD5. При запуску контролера чекати натискання кнопки. При її натисканні рахувати тривалість утримання кнопки в натиснутому стані.

Варіант	Мікроконтролер	Завдання
		При відпусканні кнопки увімкнути світлодіод на час, що дорівнює часу натискання кнопки, після чого погасити світлодіод і знову чекати натискання кнопки.
5	tiny2313 (attiny2313)	Підключити кнопку до виводу PB3, світлодіод LED1 до виводу PA0, а світлодіод LED2 до виводу PA1. При запуску контролера здійснювати блимання світлодіодом LED1 із частотою 1 Гц. При натисканні на кнопку світлодіод LED1 погасити, і почати блимання світлодіодом LED2 із частотою 2 Гц. При повторному натисканні на кнопку погасити 6світлодіод LED2 і знову почати блимати світлодіодом LED1. Процес повторювати циклічно.
6	mega168 (atmega168)	Підключити кнопку до виводу PB1, а світлодіод до виводу PC2. При запуску контролера чекати натискання кнопки. При її натисканні видати серію із п'яти світлових імпульсів за допомогою світлодіода наступної тривалості: 0,25 с – 0,5 с – 1 с – 0,5 с – 0,25 с. Пауза між усіма імпульсами постійна та дорівнює 0,5 с. Після видачі серії імпульсів знову чекати натискання кнопки.
7	tiny84 (attiny84)	Підключити кнопку до виводу PB2, світлодіод LED1 до виводу PA4, а світлодіод LED2 до виводу PA6. Організувати генератор випадкових чисел в такий спосіб. При натисканні на кнопку рахувати число проходів циклу, поки кнопка натиснута. Якщо під час відпускання кнопки це число парне, увімкнути світлодіод LED1, якщо непарне – LED2. Через 1 секунду обидва світлодіоди погасити. Процес повторювати циклічно.
8	mega328 (atmega328)	Підключити кнопку до виводу PD7, світлодіод LED1 до виводу PB0, а світлодіод LED2 до виводу PB1. Організувати генератор випадкових чисел в такий спосіб. При запуску контролера рахувати кількість проходів головного циклу програми. При натисканні на кнопку перевіряти число, що вийшло. Якщо воно парне, увімкнути світлодіод LED2, а якщо непарне – LED1. При відпусканні кнопки обидва світлодіоди погасити, а лічильну змінну обнулити. Процес повторювати циклічно.
9	tiny85 (attiny85)	Підключити кнопку до виводу PB4, світлодіод LED1 до виводу PB3, а світлодіод LED2 до виводу PB2. Рахувати кількість натискань кнопки та індикувати її за допомогою світлодіодів LED1 (молодший розряд) та LED2 (старший розряд) у двійковій системі відліку в діапазоні від 0 (обидва світлодіоди погашені) до 3 (обидва світлодіоди включені).
10	m48 TQFP (atmega48)	Підключити кнопку S1 до виводу PD6, кнопку S2 до виводу PD3, а світлодіод до виводу PB7. При

Варіант	Мікроконтролер	Завдання
		натисканні на кнопку S1 починати блимання світлодіодном з частотою 4 Гц. При натисканні на кнопку S2 блимання закінчити, а світлодіод погасити.
11	tiny24 (attiny24)	Підключити кнопку до виводу PA2, світлодіод до виводу PB3. При натисканні на кнопку протягом довше 2 секунд ввімкнути світлодіод, а при натисканні коротше 1 секунди вимкнути його. Якщо час натискання знаходиться у діапазоні від 1 до 2 секунд, не робити нічого.
12	m88 TQFP (atmega88)	Підключити кнопку до виводу PD0, а світлодіод до виводу PC3. При запуску контролера чекати натискання кнопки. При її натисканні видати сигнал SOS за допомогою світлодіоду: три коротких імпульси, пауза, три довгих імпульси, пауза, три коротких імпульси. Довжина короткого імпульсу і паузи всередині літери 0.5 с, довжина довгого імпульсу та паузи між літерами 1.5 с. Після видачі сигналу знову чекати натискання кнопки.

2.5 Питання для самоперевірки

1. Призначення регістрів DDRx, PORTx і PINx.
2. Як встановити та скинути окремий біт у байті?
3. Як перевірити, чи встановлений чи скинутий окремий біт у байті.?
4. Як записуються побітові операції на мові С?

2.6 Перелік рекомендованих джерел

1. Конспект лекцій з дисципліни «Мікропроцесорна техніка» для здобувачів вищої освіти першого (бакалаврського) рівня зі спеціальності 153 «Мікро-та наносистемна техніка» за освітньо-професійною програмою «Мікро-та наносистемна техніка» та зі спеціальності 171 «Електроніка» за освітньо-професійною програмою «Електроніка» / уклад. О. М. Гулєша. Кам'янське : ДДТУ, 2020 р. 57 с.

2. Основи Програмування AVR C. DevZone. URL: https://devzone.org.ua/post/osnovi-programuvannia-avr-c (дата звернення: 02.07.2024).

3. ATMega8 : технічна документація на мікроконтролер. URL:: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2486-8-bit-AVR-microcontroller-ATmega8_L_datasheet.pdf (дата звернення: 02.07.2024).

4. 8-bit AVR® MCUs : інформація про мікроконтролери. URL: https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/8-bit-mcus/avr-mcus (дата звернення: 02.07.2024).

3 ПРАКТИЧНА РОБОТА №3 «ПРОГРАМУВАННЯ ТАЙМЕРІВ МІКРОКОНТРОЛЕРА AVR»

3.1 Завдання

Освоєння прийомів програмування таймерів мікроконтролерів AVR.

Завдання практичної роботи:

– Створення електричної схеми у програмі SimulIDE відповідно до завдання на практичну роботу.

– Написання програми для мікроконтролеру відповідно до завдання на практичну роботу.

3.2 Теоретичні дані

3.2.1 Переривання

AVR забезпечує кілька різних джерел переривань. Ці переривання та окремий вектор скиду мають окремий програмний вектор в просторі пам’яті програм. Усім перериванням призначаються окремі біти дозволу, які повинні бути встановлені, як логічні одиниці разом із бітом дозволу глобального переривання в регістрі стану, щоб дозволити переривання. Залежно від значення програмного лічильника, переривання можуть бути автоматично вимкнені, коли запрограмовані біти блокування завантажувача BLB02 або BLB12. Ця функція покращує безпеку програмного забезпечення.

Найнижчі адреси в просторі пам’яті програм за замовчуванням визначені як вектори скидання та переривання. Повний список векторів показано далі. Список також визначає рівні пріоритету різних переривань. Чим нижче адреса, тим вищий рівень пріоритету. RESET має найвищий пріоритет, а наступним є INT0 – запит зовнішнього переривання 0. Вектори переривань можна перемістити на початок секції завантажувача флеш-пам’яті, встановивши біт вибору вектора переривань (IVSEL) у загальному регістрі керування перериваннями (GICR).

Коли виникає переривання, біт I «Глобальний дозвіл переривань» очищується, і всі переривання вимикаються. Програмне забезпечення користувача може записати логічну одиницю в біт I, щоб увімкнути вкладені переривання. Тоді всі дозволені переривання можуть переривати поточну процедуру переривання. I-біт встановлюється автоматично, коли виконується інструкція повернення з переривання – RETI.

В основному існує два типи переривань.

Перший тип ініціюється подією, яка встановлює прапор переривання. Для цих переривань програмний лічильник переходить до фактичного вектора переривань, щоб виконати процедуру обробки переривань, а апаратне забезпечення очищає відповідний прапор

переривання. Прапори переривання можна також очистити шляхом запису логічної одиниці до біта прапору, який потрібно очистити. Якщо умова переривання виникає, коли відповідний біт дозволу переривання скинутий, прапор переривання буде встановлено та запам’ятовано, доки переривання не буде дозволено, або прапор не буде очищений програмним забезпеченням. Подібним чином, якщо одне або більше умов переривань виникають, коли біт дозволу глобального переривання очищено, відповідні прапори переривань будуть встановлені та запам’ятовані, доки не буде встановлено біт дозволу глобального переривання, а потім виконуватимуться за порядком пріоритету.

Другий тип переривань запускатиметься, доки існує умова переривання. Ці переривання не обов'язково мають прапори переривань. Якщо умова переривання зникає до того, як переривання буде дозволено, переривання не буде ініційовано.

Коли AVR виходить із переривання, він завжди повертається до основної програми та виконує ще одну інструкцію, перш ніж буде обслуговано будь-яке очікуване переривання.

Зауважте, що регістр стану не зберігається автоматично під час входу в програму переривання та не відновлюється під час повернення з процедури переривання. Це повинно оброблятися програмним забезпеченням.

Якщо використовувати інструкцію CLI для вимкнення переривань, переривання буде негайно вимкнено. Жодне переривання не буде виконано після інструкції CLI, навіть якщо воно відбувається одночасно з інструкцією CLI.

У разі використання інструкції SEI для ввімкнення переривань, інструкція, що слідує після SEI, буде виконана перед будь-якими незавершеними перериваннями.

Час від настання переривання до початку його виконання для всіх увімкнених переривань мікроконтролерів AVR становить мінімум чотири такти. Після чотирьох тактів виконується фактична процедура обробки переривання за адресою програмного вектора. Протягом цього 4-тактового періоду лічильник програм зберігається в стеку. Вектор зазвичай є переходом до процедури переривання, і цей перехід займає три такти. Якщо під час виконання багатотактової інструкції виникає переривання, ця інструкція завершується до того, як переривання обслуговується. Якщо переривання виникає, коли мікроконтролер знаходиться в режимі сну, час відповіді на виконання переривання збільшується на чотири такти. Це збільшення відбувається на додаток до часу запуску з вибраного режиму сну. Повернення з процедури обробки переривань займає чотири такти. Протягом цих чотирьох тактових циклів програмний лічильник (2 байти) повертається зі стеку, вказівник стека збільшується на 2 і встановлюється біт I у регістрі SREG.

№ вектору	Адреса	Джерело	Опис переривання	№ вектору	Адреса	Джерело	Опис переривання
1	0x0000	RESET	Зовнішній скид, скид під час увімкнення живлення, скид при зниженні напруги та скид від сторожового таймера	11	0x000A	SPI, STC	Послідовне передавання завершено
2	0x0001	INT0	Запит зовнішнього переривання 0	12	0x000B	USART, RXC	USART, прийом завершено
3	0x0002	INT1	Запит зовнішнього переривання 1	13	0x000C	USART, UDRE	USART, регістр даних порожній
4	0x0003	TIMER2 COMP	Збіг при порівнянні таймера/лічильника 2	14	0x000D	USART, TXC	USART, передачу завершено
5	0x0004	TIMER2 OVF	Переповнення таймера/лічильника 2	15	0x000E	ADC	Перетворення АЦП завершено
6	0x0005	TIMER1 CAPT	Захоплення таймера/лічильника 1	16	0x000F	EE_RDY	Пам'ять EEPROM готова
7	0x0006	TIMER1 COMPA	Збіг при порівнянні А таймера/лічильника 1	17	0x0010	ANA_COMP	Аналоговий компаратор
8	0x0007	TIMER1 COMPB	Збіг при порівнянні В таймера/лічильника 1	18	0x0011	TWI	Двопровідний послідовний інтерфейс TWI
9	0x0008	TIMER1 OVF	Переповнення таймера/лічильника 1	19	0x0012	SPM_RDY	Пам'ять зберігання програм готова
10	0x0009	TIMER0 OVF	Переповнення таймера/лічильника 0				

Рисунок 3.1 – Вектори переривань ATMega8

Зовнішні переривання. Зовнішні переривання викликаються виводами INT0 і INT1. Зауважте, що якщо їх ввімкнено, переривання запускатимуться, навіть якщо контакти INT0..1 налаштовані як виходи. Ця функція забезпечує спосіб генерації програмного переривання. Зовнішні переривання можуть бути викликані спадаючим та/або наростаючим фронтом або низьким рівнем. Це налаштовується за допомогою регістру керування мікроконтролером – MCUCR. Якщо зовнішнє переривання ввімкнено та налаштоване для спрацьовування по рівню, воно запускатиметься, доки контакт утримується на низькому логічному рівні. Зауважте, що розпізнавання переривань по спадаючому або наростаючому фронту на INT0 і INT1 вимагає наявності тактового сигналу вводу/виводу. Переривання по низькому рівню на INT0/INT1 виявляються асинхронно. Це означає, що ці переривання можна використовувати для виведення мікроконтролера із режимів сну, відмінних від неактивного режиму. Тактовий сигнал вводу/виводу зупиняється в усіх режимах сну, крім неактивного.

Зауважте, що якщо переривання, викликане рівнем, використовується для пробудження з режиму вимкнення живлення, змінений рівень потрібно утримувати деякий час, щоб розбудити MCU. Це робить MCU менш чутливим до шуму. Змінений рівень двічі перевіряється тактовим сигналом сторожового таймера. Номінальний період генератора сторожового таймера становить 1 мкс при 5,0 В і 25 °C. Мікроконтролер вийде з режиму сну, якщо вхідний сигнал має необхідний рівень протягом цієї вибірки або якщо цей рівень утримується до кінця часу запуску. Якщо низький рівень виявляється протягом двох тактових імпульсів сторожового таймера, але зникає до закінчення часу запуску, мікроконтролер все одно вийде з режиму сну, але переривання не буде створено. Необхідний рівень має утримуватись достатньо довго, щоб мікроконтролер завершив пробудження, щоб ініціювати переривання по рівню.

Регістр керування мікроконтролером – MCUCR містить біти для керування зовнішніми перериваннями і загальними функціями мікроконтролера (рис. 3.2).

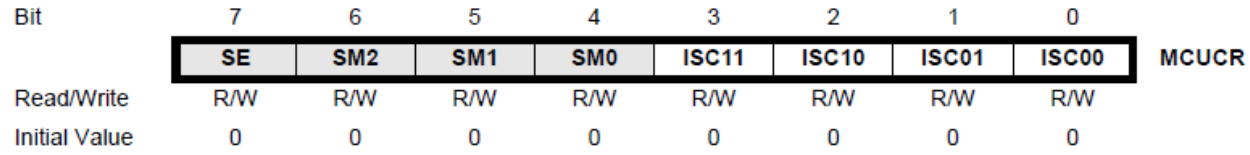


Рисунок 3.2 - Регістр керування мікроконтролером – MCUCR

Біти 3, 2 – ISC11, ISC10: управління умовою настання зовнішнього переривання 1, Біт 1 і Біт 0. Зовнішнє переривання 1 активується зовнішнім виводом INT1, якщо встановлено біт I у регістрі SREG і відповідну маску переривання в регістрі GICR. Рівень і фронти на зовнішньому контакті INT1, які активують переривання, визначені в таблиці 3.1. Значення на контакті INT1 зчитується перед виявленням фронтів. Якщо вибрано переривання по фронту або перемикання, імпульси, які тривають довше одного тактового періоду, генеруватимуть переривання. Коротші імпульси не гарантують створення переривання. Якщо вибрано переривання по низькому рівню, то він має утримуватися до завершення поточної інструкції, щоб створити переривання.

Таблиця 3.1 – Рівень і фронти на зовнішньому контакті INT1

ISC11	ISC10	Опис
0	0	Низький рівень на виводі INT1 генерує запит на переривання
0	1	Будь-яка зміна рівня на виводі INT1 генерує запит на переривання
1	0	Спадаючий фронт на виводі INT1 генерує запит на переривання
1	1	Наростаючий фронт на виводі INT1 генерує запит на переривання

Біти 1, 0 – ISC01, ISC00: управління умовою настання зовнішнього переривання 0, Біт 1 і Біт 0. Зовнішнє переривання 0 активується зовнішнім виводом INT0, якщо встановлено біт I у регістрі SREG і відповідну маску переривання в регістрі GICR. Рівень і фронти на зовнішньому контакті INT0, які активують переривання, визначені в таблиці 4.3. Значення на контакті INT0 зчитується перед виявленням фронтів. Якщо вибрано переривання по фронту або перемикання, імпульси, які тривають довше одного тактового періоду, генеруватимуть переривання. Коротші імпульси не гарантують створення переривання.

Якщо вибрано переривання по низькому рівню, то він має утримуватися до завершення поточної інструкції, щоб створити переривання.

Загальний регістр керування перериваннями – GICR (рис. 3.3).

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	–	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 3.3 - Загальний регістр керування перериваннями – GICR

Біт 7 – INT1: дозвіл запиту зовнішнього переривання 1. Коли біт INT1 встановлено (один) і біт I в регістрі стану SREG встановлено (один), зовнішнє переривання ввімкнено. Біти 1/0 керування умовою настання переривання (ISC11 та ISC10) у загальному регістрі керування мікроконтролера MCUCR визначають, чи активується зовнішнє переривання наростаючим та/або спадаючим фронтом на виводі INT1, або низьким рівнем. Зміна рівня виводу спричинить запит на переривання, навіть якщо INT1 налаштовано як вихід. Відповідне переривання «Запит зовнішнього переривання 1» виконується з вектора переривання INT1.

Біт 6 – INT0: дозвіл запиту зовнішнього переривання 0. Коли біт INT0 встановлено (один) і біт I в регістрі стану SREG встановлено (один), зовнішнє переривання ввімкнено. Біти 1/0 керування умовою настання переривання (ISC01 та ISC00) у загальному регістрі керування мікроконтролера MCUCR визначають, чи активується зовнішнє переривання наростаючим та/або спадаючим фронтом на виводі INT0, або низьким рівнем. Зміна рівня виводу спричинить запит на переривання, навіть якщо INT0 налаштовано як вихід. Відповідне переривання «Запит зовнішнього переривання 0» виконується з вектора переривання INT0.

Загальний регістр прапорів переривань – GIFR (рис. 3.4).

Bit	7	6	5	4	3	2	1	0	
	INTF1	INTF0	–	–	–	–	–	–	GIFR
Read/Write	R/W	R/W	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 3.4 - Загальний регістр прапорів переривань – GIFR

Біт 7 – INTF1: прапор зовнішнього переривання 1. Коли подія на виводі INT1 викликає запит на переривання, біт INTF1 встановлюється (стає одиницею). Якщо біт I в регістрі SREG і біт INT1 біт в регістрі GICR встановлені (один), мікроконтролер перейде до відповідного вектору переривань. Прапор очищується, коли виконується підпрограма переривання. Крім того, прапор можна очистити, записавши в нього

логічну одиницю. Цей прапор завжди скинутий, коли INT1 налаштовано як переривання по рівню.

Біт 6 – INTF0: прапор зовнішнього переривання 0. Коли подія на виводі INT0 викликає запит на переривання, біт INTF0 встановлюється (стає одиницею). Якщо біт I в регістрі SREG і біт INT0 біт в регістрі GICR встановлені (один), мікроконтролер перейде до відповідного вектору переривань. Прапор очищується, коли виконується підпрограма переривання. Крім того, прапор можна очистити, записавши в нього логічну одиницю. Цей прапор завжди скинутий, коли INT0 налаштовано як переривання по рівню.

3.2.2 8-бітний таймер/лічильник 0 мікроконтролера ATMega8

Таймер/лічильник0 — одноканальний 8-розрядний модуль таймера/лічильника загального призначення. Основні особливості:

- Одноканальний лічильник
- Генератор частоти
- Лічильник зовнішніх подій
- 10-бітовий попередній дільник тактового сигналу

Спрощена блок-схема 8-розрядного таймера/лічильника показана на рисунку 3.5.

Регістри. Таймер/лічильник (TCNT0) є 8-розрядним регістром. Усі сигнали запиту на переривання (скорочено Int. Req. на рисунку 3.5) відображаються в регістрі прапорів переривання таймера (TIFR). Всі переривання індивідуально маскуються за допомогою регістра маски переривання таймера (TIMSK). TIFR і TIMSK не показані на рисунку 3.5, оскільки ці регістри спільно використовуються іншими блоками таймерів.

Таймер/лічильник може тактуватися внутрішньо або через попередній дільник, або зовнішнім джерелом імпульсів на контакті T0. Логічний блок вибору тактового сигналу контролює, яке джерело та фронт тактового сигналу таймер/лічильник використовує для збільшення свого значення. Таймер/лічильник неактивний, якщо не вибрано джерело тактового сигналу. Вихід блоку вибору тактового сигналу називається тактовим сигналом таймера (clk_{T0}).

Визначення. Багато посилань на регістри та біти в цій лекції написані у загальній формі. Літера «n» замінює номер таймера/лічильника, у цьому випадку, 0. Однак, коли в програмі використовується регістр або біт, необхідно використовувати точну форму, тобто TCNT0 для доступу до значення лічильника таймера/лічильника 0 і так далі. Визначення в таблиці 3.2 також широко використовуються далі.

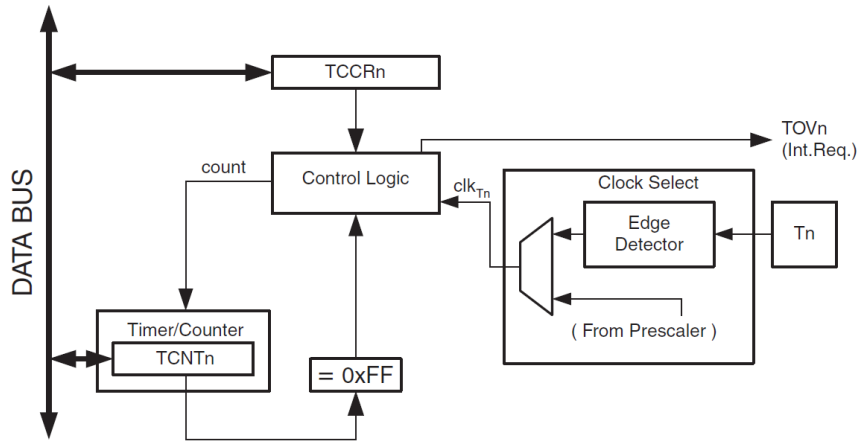


Рисунок 3.5 – Спрощена блок-схема 8-розрядного таймера/лічильника

Джерела тактового сигналу таймера/лічильника. Таймер/лічильник може тактуватися внутрішнім або зовнішнім джерелом тактового сигналу. Джерело вибирається блоком вибору тактового сигналу, який керується бітами вибору тактового сигналу (CS02:0), розташованими в регістрі керування таймером/лічильником (TCCR0).

Таблиця 3.2 – Визначення

Визначення	Значення
Нижнє значення	Лічильник досягає **нижнього значення**, коли стає 0x00
Максимум	Лічильник досягає свого **максимуму**, коли він стає 0xFF (десяткове число 255)

Лічильник. Основною частиною 8-розрядного таймера/лічильника є блок програмованого лічильника. На рисунку 3.6 показана блок-схема лічильника та його оточення.

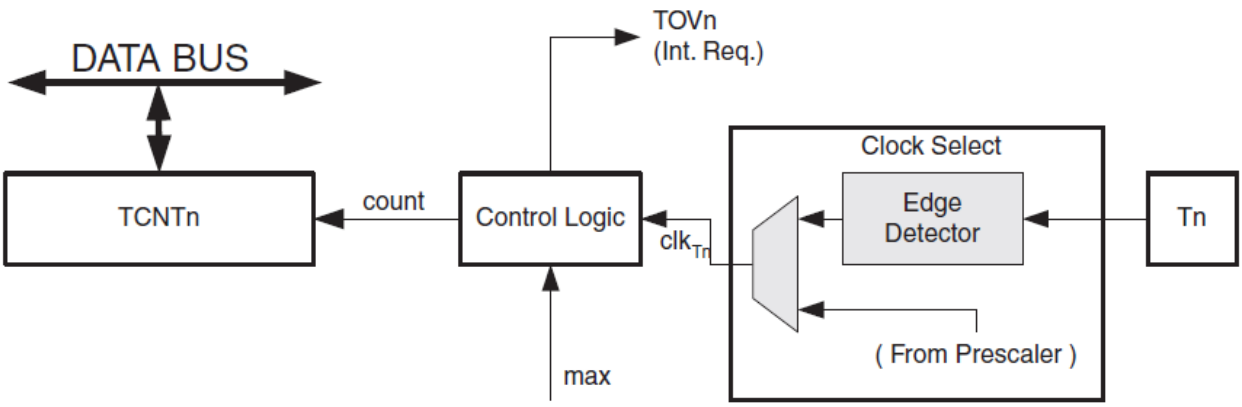


Рисунок 3.6 - Блок-схема лічильника та його оточення

Опис сигналів (внутрішні сигнали):

- **count** Збільшує TCNT0 на 1
- **clk_{Tn}** Тактовий сигнал таймера/лічильника, далі згадується як clk_{T0}
- **max** Сигналізує, що TCNT0 досяг максимального значення

Лічильник збільшується на кожному такті таймера (clk_{T0}). clk_{T0} може генеруватися зовнішнім або внутрішнім джерелом тактового сигналу за допомогою бітів вибору тактового сигналу (CS02:0). Якщо джерело тактового сигналу не вибрано (CS02:0 = 0), таймер зупиняється. Однак ЦП може отримати доступ до значення TCNT0, незалежно від того, присутній clk_{T0} чи ні. Значення, записане за допомогою ЦП, перевизначає значення таймера, отримане при операціях очищення чи рахунку лічильника.

Принцип дії. Напрямок рахунку лічильника завжди вгору (збільшення), а очищення лічильника не виконується. Лічильник просто переповнюється, коли він проходить максимальне 8-бітне значення (MAX = 0xFF), а потім перезапускається з нижнього значення (0x00). У нормальній роботі прапор переповнення таймера/лічильника (TOV0) буде встановлено в той самий цикл таймера, коли TCNT0 стає нульовим. Прапор TOV0 у цьому випадку поводиться як дев'ятий біт, за винятком того, що він може лише бути встановленим, а не скинутим. Однак у поєднанні з перериванням переповнення таймера, яке автоматично очищає прапор TOV0, роздільну здатність таймера можна збільшити за допомогою програмного забезпечення. Нове значення лічильника можна записати будь-коли.

Часові діаграми таймера/лічильника. Таймер/лічильник є синхронним модулем, тому на рисунках тактовий сигнал таймера (clk_{T0}) показаний як сигнал увімкнення тактового сигналу. Рисунки містять інформацію про те, коли встановлюються прапори переривання. Рисунок 3.7 містить дані тактування при нормальній роботі таймера/лічильника. Показана послідовність рахунку, близька до максимального значення.

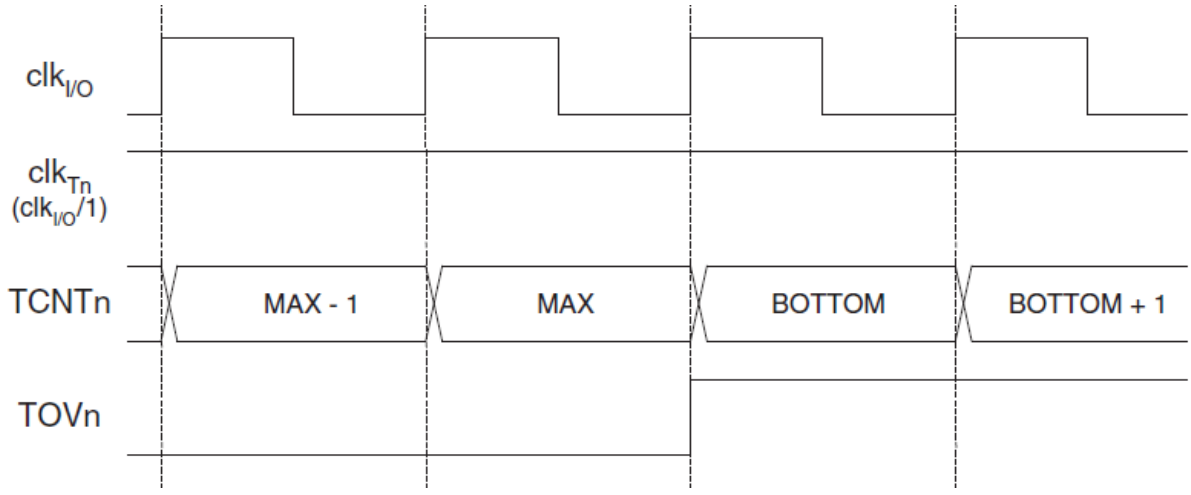


Рисунок 3.7 – Дані тактування при нормальній роботі таймера/лічильника

На рисунку 3.8 показані ті самі дані про тактування, але з увімкненим попереднім дільником ($f_{clkI/O}/8$).

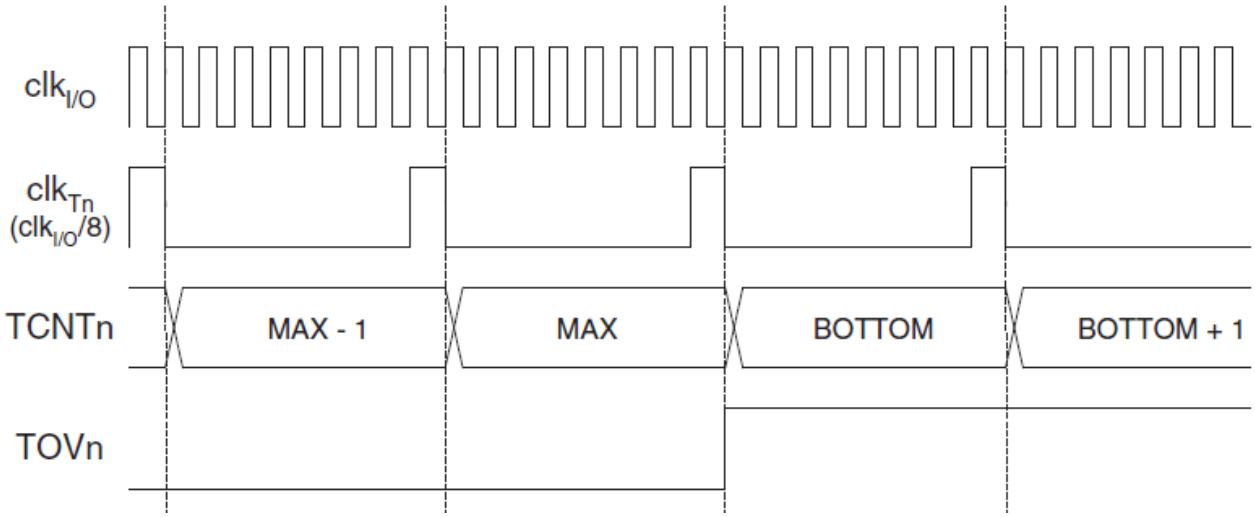


Рисунок 3.8 - Дані про тактування з увімкненим попереднім дільником

Опис регістрів 8-бітного таймера/лічильника 0 мікроконтролера ATMega8:

Регістр керування таймером/лічильником – TCCR0 зображено на рис. 3.9.

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	–	CS02	CS01	CS00	TCCR0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 3.9 - Регістр керування таймером/лічильником – TCCR0

Біти 2:0 – CS02:0: Вибір тактового сигналу. Три біти вибору тактового сигналу вибирають джерело тактового сигналу, яке буде використовуватися таймером/лічильником.

Якщо для таймера/лічильника0 використовуються режими тактування від зовнішніх сигналів, зміни на виводі T0 змінюватимуть лічильник, навіть якщо цей вивід налаштовано як вихід. Ця функція дозволяє програмно контролювати рахунок таймера.

Налаштування таймера за бітами тактового сигналу представлено у таблиці 3.3.

Таблиця 3.3 – Налаштування таймера за бітами тактового сигналу

CS02	CS01	CS00	Опис
0	0	0	Немає тактового сигналу (таймер/лічильник зупинений)
0	0	1	$clk_{I/O}$ (без попереднього дільника)
0	1	0	$clk_{I/O}/8$ (від попереднього дільника)
0	1	1	$clk_{I/O}/64$ (від попереднього дільника)
1	0	0	$clk_{I/O}/256$ (від попереднього дільника)
1	0	1	$clk_{I/O}/1024$ (від попереднього дільника)
1	1	0	Зовнішній тактовий сигнал на виводі T0. Синхронізація по спадаючому фронту
1	1	1	Зовнішній тактовий сигнал на виводі T0. Синхронізація по зростаючому фронту

Регістр таймера/лічильника – TCNT0 зображено на рис. 3.10.

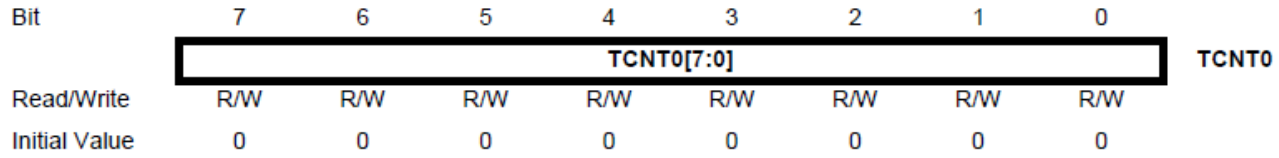


Рисунок 3.10 – Регістр таймера/лічильника – TCNT0

Регістр таймера/лічильника надає прямий доступ як для операцій читання, так і для запису до 8-розрядного лічильника модуля таймера/лічильника.

Регістр маски переривання таймерів/лічильників – TIMSK зображено на рис 3.11**.**

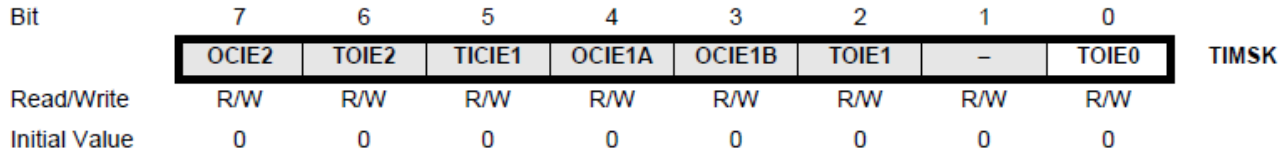


Рисунок 3.11 – Регістр маски переривання таймерів/лічильників – TIMSK

Біт 0 – TOIE0: Ввімкнення переривання по переповненню таймера/лічильника 0. Коли біт TOIE0 встановлено як 1, і біт I в регістрі стану також встановлений як 1, переривання по переповненню таймера/лічильника 0 увімкнено. Відповідне переривання генерується, якщо відбувається переповнення таймера/лічильника 0, тобто коли

встановлюється біт TOV0 в регістрі прапорів переривання таймерів/лічильників TIFR.

Регістр прапорів переривання таймерів/лічильників – TIFR (рис. 3.12)

Біт 0 – TOV0: прапор переповнення таймера/лічильника 0. Біт TOV0 встановлюється в 1, коли відбувається переповнення таймера/лічильника 0. TOV0 очищується апаратним забезпеченням під час виконання відповідного вектора обробки переривань. Крім того, TOV0 очищується шляхом запису логічної одиниці у біт прапора. Коли встановлено біт I регістру SREG, TOIE0 регістру TIMSK і TOV0 у регістрі TIFR, генерується переривання по переповненню таймера/лічильника 0.

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	–	TOV0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 3.12 - Регістр прапорів переривання таймерів/лічильників – TIFR

3.2.3 16-бітний таймер/лічильник 1 мікроконтролера ATMega8

16-розрядний блок таймера/лічильника дозволяє точно визначати час виконання програми (керування подіями), генерувати прямокутний сигнал та вимірювати тривалість сигналів. Основні особливості:

- Справжній 16-бітний дизайн (тобто дозволяє 16-бітний ШІМ)
- Два незалежних блоки порівняння
- Подвійно буферизовані регістри порівняння
- Один блок захоплення
- Фільтр шуму на вході захоплення
- Очистка таймеру під час порівняння (автоматичне перезавантаження)
- Широтно-імпульсний модулятор (ШІМ) без збоїв, і з корекцією фази
- Змінний період ШІМ
- Генератор частоти
- Лічильник зовнішніх подій
- Чотири незалежних джерела переривань (TOV1, OCF1A, OCF1B і ICF1)

Більшість посилань на регістри та біти в цьому розділі написані у загальній формі. Літера «n» замінює номер таймера/лічильника, а літера «x» замінює вихідний канал порівняння. Однак, коли в програмі використовується регістр або біт, необхідно використовувати точну форму, тобто TCNT1 для доступу до значення лічильника таймера/лічильника тощо.

Спрощена блок-схема 16-розрядного таймера/лічильника 1 показана на рисунку 3.13.

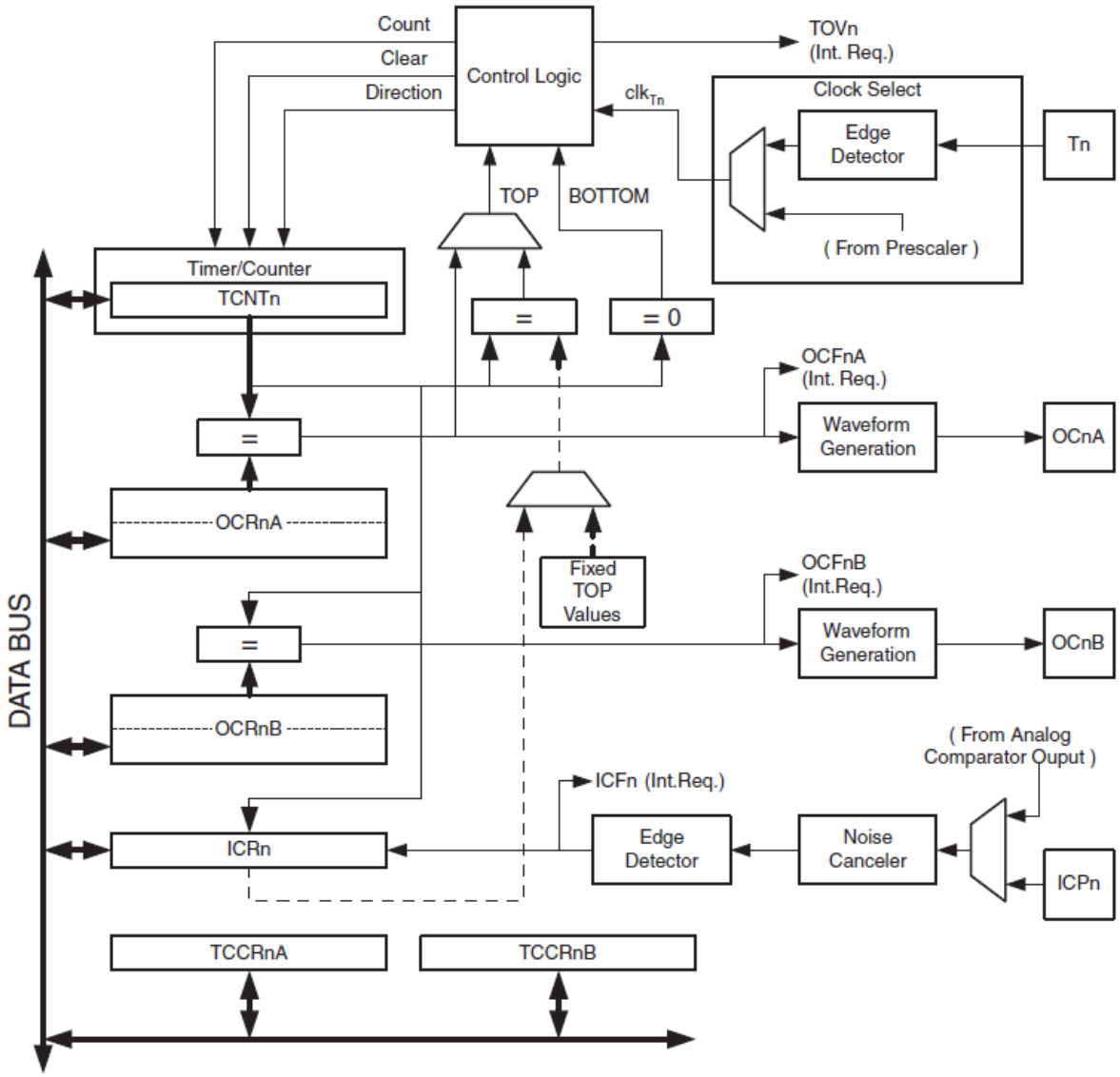


Рисунок 3.13 – Спрощена блок-схема 16-розрядного таймера/лічильника 1

Регістри. Таймер/лічильник (TCNT1), регістри порівняння (OCR1A/B) і регістр захоплення (ICR1) є 16-розрядними регістрами. Під час доступу до 16-розрядних регістрів необхідно дотримуватися спеціальних процедур, що описані далі. Регістри керування таймером/лічильником (TCCR1A/B) є 8-розрядними регістрами і не мають обмежень доступу до ЦП. Усі сигнали запитів на переривання (скорочено Int. Req. на рисунку 3.13) відображаються в реєстрі прапорів переривання таймера (TIFR). Усі переривання індивідуально маскуються за допомогою регістра маски переривань таймерів (TIMSK). TIFR і TIMSK не показані на рисунку 3.13, оскільки ці регістри спільно використовуються іншими блоками таймерів.

Таймер/лічильник можна тактувати внутрішньо, через попередній дільник, або зовнішнім джерелом тактування на виводі T1. Логічний блок вибору тактового сигналу контролює джерело тактового сигналу та фронт, які таймер/лічильник використовує для збільшення (або зменшення) свого значення. Таймер/лічильник неактивний, якщо не вибрано джерело тактування. Вихід блоку вибору тактового сигналу називається тактовим сигналом таймера (clk_{T1}).

Подвійно буферизовані регістри порівняння (OCR1A/B) постійно порівнюються зі значенням таймера/лічильника. Результат порівняння може бути використаний генератором сигналів для генерації вихідного сигналу ШІМ або змінної частоти на виводах порівняння (OC1A/B). Подія «Збіг при порівнянні» також встановлює прапор збігу при порівнянні (OCF1A/B), який можна використовувати для створення запиту на відповідне переривання.

Регістр захоплення може фіксувати значення таймера/лічильника при заданій зовнішній події на виводі захоплення (ICP1) або на виводах аналогового компаратора. Блок захоплення вхідного сигналу містить блок цифрової фільтрації для зменшення ймовірності захоплення шумових змін на вході.

Верхнє або максимальне значення таймера/лічильника в деяких режимах роботи може бути визначено регістром OCR1A, регістром ICR1 або набором фіксованих значень. Якщо OCR1A використовується як верхнє значення у режимі ШІМ, регістр OCR1A не може використовуватися для генерації виходу ШІМ. Однак у цьому випадку верхнє значення буде подвійно буферизовано, що дозволяє змінювати його під час виконання. Якщо потрібне фіксоване верхнє значення, регістр ICR1 можна використовувати як альтернативу, звільняючи OCR1A для використання як виходу ШІМ.

Визначення. У цій практичній роботі широко використовуються такі визначення (табл 3.4).

Таблиця 3.4 – Визначення

Визначення	Значення
Нижнє значення	Лічильник досягає **нижнього значення**, коли стає 0x0000
Максимум	Лічильник досягає свого **максимуму**, коли він стає 0xFFFF (десяткове число 65535)
Верхнє значення	Лічильник досягає **верхнього значення**, коли стає рівним найвищому значенню в послідовності підрахунку. **Верхнє значення** можна встановити рівним одному з фіксованих значень: 0x00FF, 0x01FF або 0x03FF, або значенню, що зберігається в регістрі OCR1A або ICR1.

Джерела тактового сигналу таймера/лічильника 1. Таймер/лічильник може тактуватися внутрішнім або зовнішнім джерелом

тактового сигналу. Джерело тактового сигналу вибирається блоком вибору тактового сигналу, який керується бітами вибору тактового сигналу (CS12:0), розташованими в регістрі керування таймером/лічильником B (TCCR1B).

Модуль лічильника. Основною частиною 16-бітного таймера/лічильника 1 є програмований 16-бітний двонаправлений лічильник. На рисунку 3.14 показана блок-схема лічильника та його оточення.

Опис сигналів (внутрішні сигнали):

count - Збільшення або зменшення TCNT1 на 1

direction - Вибір напрямку рахунку таймера між збільшенням і зменшенням

clear - Очистити TCNT1 (встановити всі біти як 0)

clk_{T1} - Тактовий сигнал таймера/лічильника

TOP - Сигналізує, що TCNT1 досяг верхнього значення

BOTTOM - Сигналізує, що TCNT1 досяг нижнього значення (нуль)

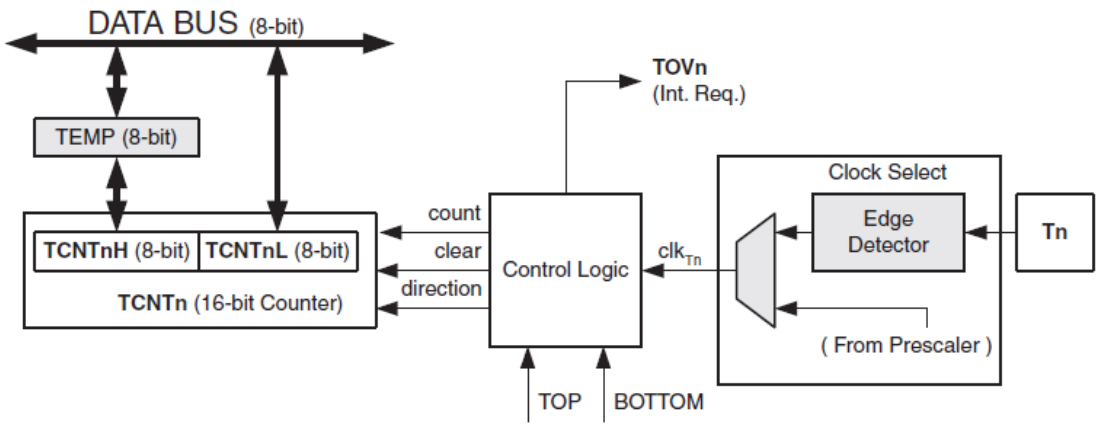


Рисунок 3.14 – Блок-схема лічильника та його оточення

16-розрядний лічильник розташований в двох 8-розрядних комірках пам'яті вводу/виводу: старшому байті лічильника (TCNT1H), що містить вісім старших бітів лічильника, і нижчому байті лічильника (TCNT1L), що містить нижні вісім бітів. ЦП може отримати доступ до регістру TCNT1H лише опосередковано. Коли ЦП здійснює доступ до регістру вводу/виводу TCNT1H, він отримує доступ до тимчасового регістра старшого байта (TEMP). Тимчасовий регістр оновлюється значенням TCNT1H під час зчитування TCNT1L, а TCNT1H оновлюється значенням тимчасового регістру під час запису TCNT1L. Це дозволяє процесору читати або записувати все 16-бітне значення лічильника протягом одного такту через 8-бітну шину даних. Важливо зауважити, що існують особливі випадки запису в регістр TCNT1 під час рахунку лічильника, які дадуть непередбачувані результати.

Залежно від використовуваного режиму роботи лічильник очищується, збільшується або зменшується на кожному такті таймера (clk_{T1}). Clk_{T1} може бути згенерований зовнішнім або внутрішнім джерелом тактового сигналу, вибраним бітами вибору тактового сигналу (CS12:0). Якщо джерело тактування не вибрано (CS12:0 = 0), таймер зупиняється. Однак ЦП може отримати доступ до значення TCNT1 незалежно від того, присутній clk_{T1} чи ні. Запис за допомогою ЦП перевизначає значення отримані в результаті усіх операцій очищення чи рахунку лічильника.

Послідовність рахунку визначається налаштуванням бітів режиму генерації сигналу (WGM13:0), розташованих у регістрах керування таймером/лічильником A та B (TCCR1A та TCCR1B). Існують тісні зв'язки між тим, як лічильник поводиться (рахує), і як сигнали генеруються на виходах порівняння OC1x.

Прапор переповнення таймера/лічильника (TOV1) встановлюється відповідно до режиму роботи, вибраного бітами WGM13:0. TOV1 можна використовувати для генерації переривання.

Блоки порівняння. 16-розрядний компаратор безперервно порівнює TCNT1 з регістром порівняння (OCR1x). Якщо TCNT дорівнює OCR1x, компаратор сигналізує про збіг. При цьому встановлюється прапор порівняння (OCF1x) на наступному такті таймера. Прапор порівняння генерує відповідне переривання, якщо воно ввімкнене (OCIE1x = 1). Прапор OCF1x автоматично очищується, коли виконується підпрограма обробки переривання. Крім того, прапор OCF1x може бути очищений програмним забезпеченням, при записі логічної одиниці у нього. Генератор сигналів використовує сигнал збігу для генерування вихідних даних відповідно до режиму роботи, встановленого бітами режиму генерації сигналу (WGM13:0) і режиму порівняння (COM1x1:0). Сигнали нижнього і верхнього значень використовуються генератором сигналів для обробки особливих випадків граничних значень у деяких режимах роботи.

Спеціальна функція блоку порівняння A дозволяє йому визначати верхнє значення таймера/лічильника (тобто роздільну здатність лічильника). Окрім роздільної здатності лічильника, верхнє значення визначає тривалість періоду для сигналів, створених генератором сигналів.

На рисунку 3.15 показана блок-схема блоку порівняння. Маленьке «n» в іменах регістрів і бітів вказує на номер таймера (n=1 для таймера/лічильника 1), а «x» вказує на назву блока порівняння вихідних даних (A/B). Елементи блок-схеми, які безпосередньо не є частиною блоку порівняння вихідних даних, виділені сірим кольором.

Регістр OCR1x подвійно буферизується при використанні будь-якого з дванадцяти режимів широтно-імпульсної модуляції (ШІМ). Для нормального режиму роботи та режиму «Очищення таймера при збігу» подвійна буферизація вимкнена. Подвійна буферизація синхронізує оновлення регістра порівняння OCR1x до верхнього або нижнього

значення рахунку. Синхронізація запобігає появі непарної довжини несиметричних ШІМ-імпульсів, тим самим роблячи вихідний сигнал без збоїв.

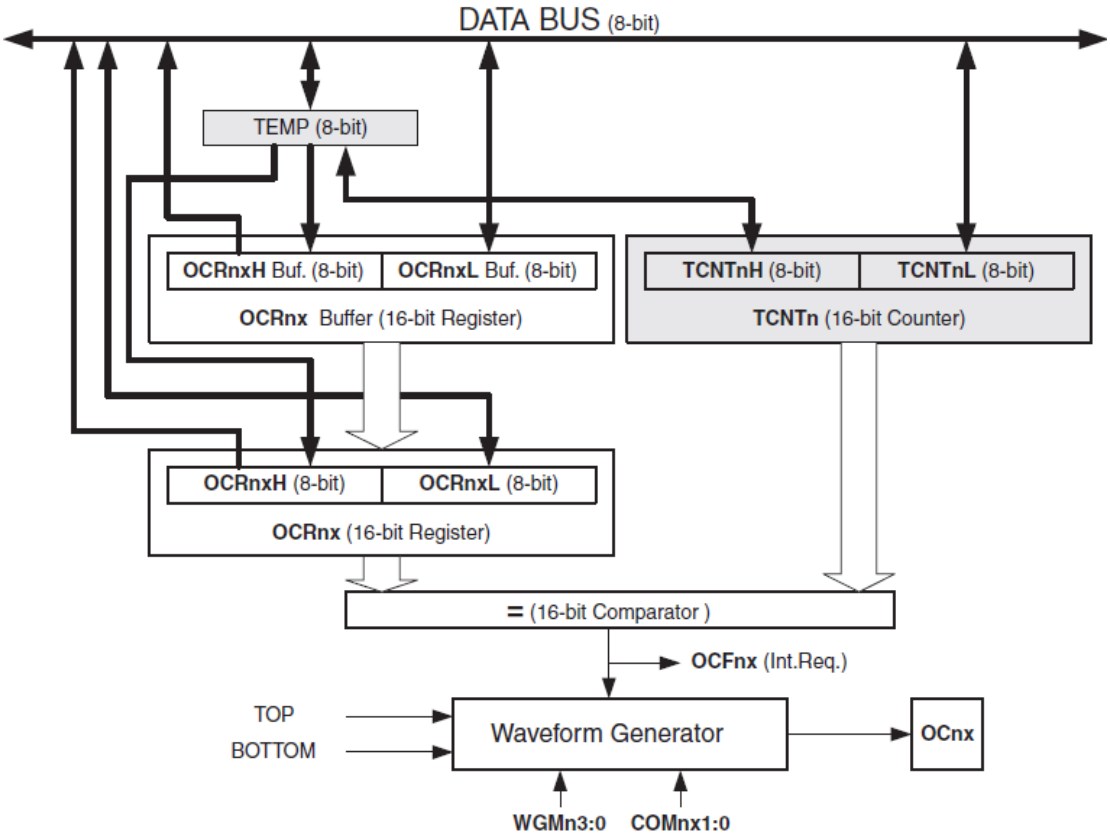


Рисунок 3.15 – Блок-схема блоку порівняння

Доступ до регістру OCR1x може здатися складним, але це не так. Коли подвійну буферизацію ввімкнено, ЦП має доступ до регістру буфера OCR1x, і якщо подвійну буферизацію вимкнено, ЦП має доступ до OCR1x безпосередньо. Вміст регістра OCR1x (буфер або порівняння) змінюється лише операцією запису (таймер/лічильник не оновлює цей регістр автоматично, як регістри TCNT1 та ICR1). Тому OCR1x не читається через тимчасовий регістр старшого байта (TEMP). Однак, як під час доступу до інших 16-бітних регістрів, рекомендується спочатку читати молодший байт. Запис регістрів OCR1x має здійснюватися через регістр TEMP, оскільки порівняння всіх 16-розрядних даних виконується постійно. Першим потрібно записати старший байт (OCR1xH). Коли значення старшого байта записується центральним процесором, регістр TEMP буде оновлено записаним значенням. Потім, коли записується молодший байт (OCR1xL), старший байт буде скопійовано у старші 8 бітів або буфера OCR1x, або регістра порівняння OCR1x у тому самому тактовому циклі системи.

Примусове порівняння. У режимах генерації сигналу без ШІМ вихідний сигнал компаратора можна примусово встановити, записавши одиницю в біт «Примусове порівняння» (FOC1x). Примусове порівняння не встановлює прапор OCF1x і не перезавантажує/скидає таймер, але вивід OC1x буде оновлено так, ніби відбулося справжнє порівняння (параметри бітів COM1x1:0 визначають, чи вивід OC1x буде встановлено, скинуто або перемкнуто).

Блокування збігу при порівнянні шляхом запису у TCNT1. Усі записи центральним процесором в регістр TCNT1 блокуватимуть будь-який збіг при порівнянні, який відбувається в наступному такті таймера, навіть якщо таймер зупинено. Ця функція дозволяє встановити у OCR1x таке саме значення, що й TCNT1, не запускаючи переривання, коли ввімкнено тактування таймера/лічильника.

Використання блоку порівняння. Оскільки запис у TCNT1 у будь-якому режимі роботи блокуватиме всі збіги при порівнянні протягом одного тактового циклу таймера, є ризики, пов'язані зі зміною TCNT1 під час використання будь-якого з вихідних каналів порівняння, незалежно від того, працює таймер/лічильник чи ні. Якщо значення, записане в TCNT1, дорівнює значенню OCR1x, збіг при порівнянні буде пропущено, що призведе до неправильної генерації сигналу. Не записуйте TCNT1 рівним верхньому значенню у режимах ШІМ зі змінними верхніми значеннями. Збіг при порівняльний для верхнього значення буде проігноровано, а лічильник продовжить рахувати до 0xFFFF. Так само не записуйте значення TCNT1, що дорівнює нижньому значенню, коли лічильник рахує вниз.

Налаштування OC1x слід виконати перед налаштуванням регістра напрямку даних для порту вводу/виводу. Найпростішим способом встановлення значення OC1x є використання біту «Примусове порівняння» (FOC1x) у нормальному режимі. Регістр OC1x зберігає своє значення навіть при зміні режимів генерації сигналу.

Майте на увазі, що біти COM1x1:0 не буферизуються подвійно, як регістри порівняння. Зміна бітів COM1x1:0 набуде чинності негайно.

Блок збігу при порівнянні. Біти режиму виводів порівняння (COM1x1:0) мають дві функції. Генератор сигналу використовує біти COM1x1:0 для визначення стану порівняння (OC1x) під час наступного збігу при порівнянні. По-друге, біти COM1x1:0 керують джерелом вихідного сигналу OC1x. На рисунку 3.16 показано спрощену схему логіки, на яку впливає налаштування бітів COM1x1:0. Регістри вводу-виводу, біти вводу-виводу та контакти вводу-виводу на рисунку виділені жирним шрифтом. Показано лише частини загальних регістрів керування портом вводу/виводу (DDR і PORT), на які впливають біти COM1x1:0. Коли йдеться про стан OC1x, посилання стосується внутрішнього регістру OC1x, а не контакту OC1x. Якщо відбувається скидання системи, регістр OC1x скидається у «0».

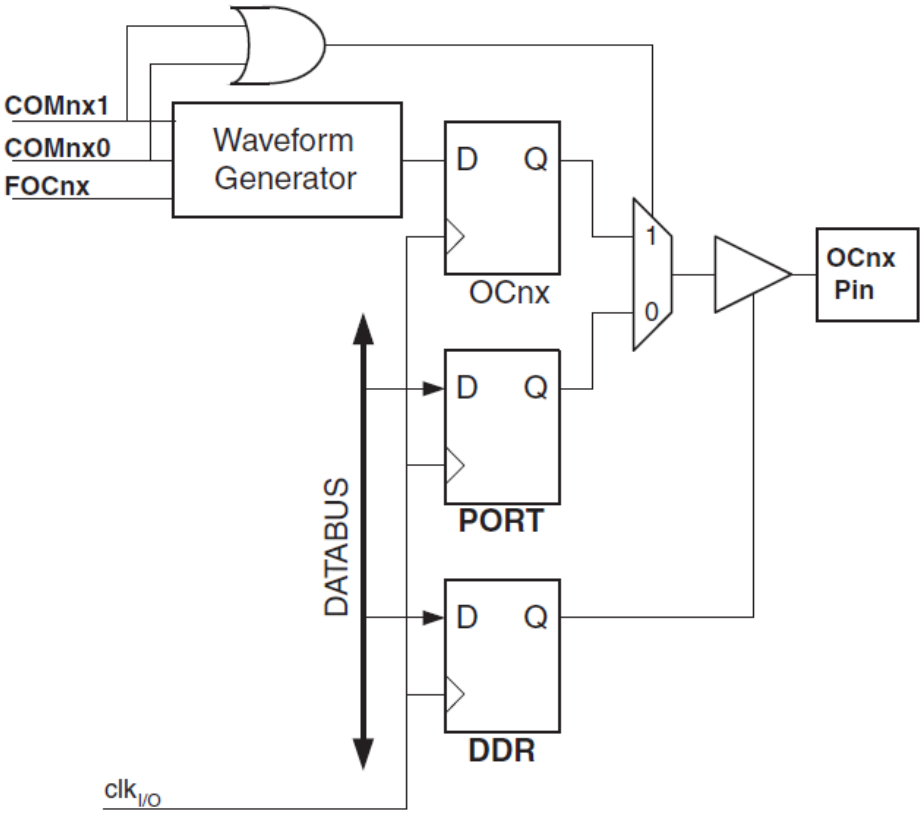


Рисунок 3.16 – Спрощена схему логіки блоку збігу

Функція загального порту вводу/виводу перевизначається на вихід порівняння (OC1x) від генератора сигналу, якщо встановлено один із бітів COM1x1:0. Однак напрямок виводу OC1x (вхід або вихід) усе ще контролюється регістром напряму даних (DDR) для виводу порту. Біт регістра направлення даних для виводу OC1x (DDR_OC1x) має бути встановлений як вихідний, перш ніж значення OC1x буде видно на виводі. Функція перевизначення порту зазвичай не залежить від режиму генерації сигналу, але є деякі винятки.

Конструкція логіки виводу порівняльного дозволяє ініціалізувати стан OC1x до ввімкнення виводу. Зауважте, що деякі налаштування бітів COM1x1:0 зарезервовані для певних режимів роботи.

Біти COM1x1:0 не впливають на блок захоплення.

Режим порівняння та генерація сигналу. Генератор сигналів по-різному використовує біти COM1x1:0 у звичайному режимі, режимі очищення таймера при збігу і ШІМ. Для всіх режимів встановлення COM1x1:0 = 0 повідомляє генератору сигналу, що жодних дій у регістрі OC1x не потрібно виконувати під час наступного порівняння.

Зміна стану бітів COM1x1:0 матиме ефект під час першого збігу при порівнянні після запису бітів. Для режимів без ШІМ дію можна примусово виконати за допомогою бітів FOC1x.

Режими роботи. Режим роботи (тобто поведінка таймера/лічильника та виводів порівняння) визначається комбінацією бітів

режиму генерації сигналу (WGM13:0) і режиму порівняння (COM1x1:0). Біти режиму порівняння не впливають на послідовність рахунку, тоді як біти режиму генератора сигналу впливають. Біти COM1x1:0 визначають, чи слід інвертувати згенерований вихід ШІМ чи ні (інвертований чи неінвертований ШІМ). Для режимів без ШІМ біти COM1x1:0 контролюють, чи вихідний сигнал під час збігу при порівнянні має бути встановлений, очищений або перемкнутий.

Звичайний режим. Найпростішим режимом роботи є звичайний режим (WGM13:0 = 0). У цьому режимі напрямок рахунку завжди спрямований вгору (збільшується), а очищення лічильника не виконується. Лічильник просто переповнюється, коли він проходить максимальне 16-бітне значення (MAX = 0xFFFF), а потім перезапускається з нижнього значення (0x0000). У звичайній роботі прапор переповнення таймера/лічильника (TOV1) буде встановлено в той самий цикл таймера, коли TCNT1 стає рівним 0. Прапор TOV1 у цьому випадку поводиться як 17-й біт, за винятком того, що він може бути тільки встановленим, а не скинутим. Однак у поєднанні з перериванням переповнення таймера, яке автоматично очищає прапор TOV1, роздільну здатність таймера можна збільшити за допомогою програмного забезпечення. У звичайному режимі немає особливих випадків, нове значення лічильника можна записати будь-коли.

Блок захоплення просто використовувати в звичайному режимі. Однак зауважте, що максимальний інтервал між зовнішніми подіями не повинен перевищувати роздільну здатність лічильника. Якщо інтервал між подіями занадто тривалий, для збільшення роздільної здатності блоку захоплення слід використовувати переривання переповнення таймера або попередній дільник.

Блоки порівняння можна використовувати для генерації переривань у певний момент часу. Використання порівняння для генерування сигналів у нормальному режимі не рекомендується, оскільки це займе надто багато часу ЦП.

Очищення таймеру при збігу (OT3). У режимі очищення таймеру при збігу (WGM13:0 = 4 або 12) регістри OCR1A або ICR1 використовуються для керування роздільною здатністю лічильника. У режимі OT3 лічильник обнулюється, коли значення лічильника (TCNT1) збігається з OCR1A (WGM13:0 = 4) або ICR1 (WGM13:0 = 12). OCR1A або ICR1 визначають верхнє значення для лічильника, отже, також його роздільну здатність. Цей режим дозволяє краще контролювати вихідну частоту таймера. Це також спрощує роботу рахунку зовнішніх подій. Часова діаграма для режиму OT3 показана на рисунку 3.17. Значення лічильника (TCNT1) збільшується, доки не відбудеться збіг зі значенням OCR1A або ICR1, а потім лічильник (TCNT1) очищується.

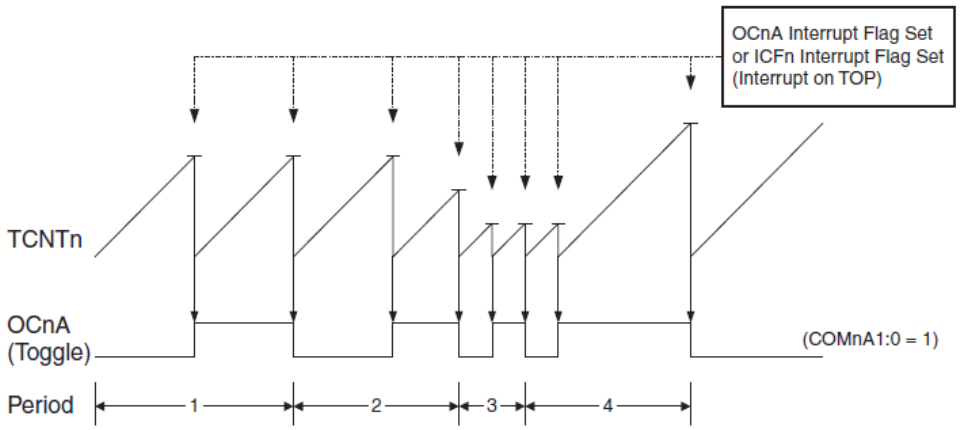


Рисунок 3.17 – Часова діаграма для режиму OT3

Кожного разу, коли значення лічильника досягає верхнього значення, може бути згенероване переривання, використовуючи прапор OCF1A або ICF1 відповідно до регістру, який використовується для визначення верхнього значення. Якщо переривання ввімкнено, програму обробки переривань можна використовувати для оновлення верхнього значення. Однак, змінюючи верхнє значення на значення, близьке до нижнього, коли лічильник працює без попереднього дільника або з низьким значенням попереднього дільника, слід виконувати обережно, оскільки режим OT3 не має функції подвійної буферизації. Якщо нове значення, записане в OCR1A або ICR1, є нижчим за поточне значення TCNT1, лічильник пропустить збіг при порівнянні. Тоді лічильнику доведеться рахувати до свого максимального значення (0xFFFF) і починати з 0x0000, перш ніж відбудеться порівняння. У багатьох випадках ця функція є небажаною. Альтернативою буде використання режиму швидкої ШІМ з використанням OCR1A для визначення верхнього значення (WGM13:0 = 15), оскільки тоді OCR1A буде подвійно буферизований.

Для генерування вихідного сигналу в режимі OT3 вихід OC1A може бути налаштований на перемикання свого логічного рівня при кожному збігу при порівнянні, установивши біти режиму виводу порівняння на режим перемикання (COM1A1:0 = 1). Значення OC1A не з'явиться на виводі, якщо напрям даних для нього не налаштовано як вихід (DDR_OC1A = 1). Згенерований сигнал матиме максимальну частоту $f_{OC1A} = f_{clk\_I/O}/2$, коли OCR1A встановлено як нуль (0x0000). Частота вихідного сигналу визначається наступним рівнянням:

$$f_{OCnA} = \frac{f_{CLK\_I/O}}{2 \cdot N \cdot (1 + OCRnA)} \qquad (3.1)$$

Змінна N представляє коефіцієнт попереднього поділення (1, 8, 64, 256 або 1024). Що стосується звичайного режиму роботи, прапор TOV1

встановлюється в той самий тактовий цикл таймера, в який лічильник рахує від максимального значення до 0x0000.

Режим швидкого ШІМ. Режим швидкої широтно-імпульсної модуляції або режим швидкого ШІМ (WGM13:0 = 5, 6, 7, 14 або 15) забезпечує можливість генерації високочастотного сигналу ШІМ. Швидкий ШІМ відрізняється від інших варіантів ШІМ однонахилою роботою. Лічильник веде відлік від нижнього до верхнього значення, а потім перезапускається з нижнього. У неінвертованому режимі порівняння вихід порівняння (OC1x) очищується при збігу між TCNT1 і OCR1x і встановлюється при нижньому значенні. В режимі інвертування вихідний сигнал встановлюється при збігу і очищується при нижньому значенні. Завдяки роботі з одним нахилом робоча частота швидкого ШІМ-режиму може бути вдвічі вищою, ніж при фазо-коректному режимі, і фазо- і частото-коректному режимі ШІМ, які використовують подвійний нахил. Ця висока частота робить режим швидкого ШІМ добре придатним для регулювання потужності, випрямлення та додатків ЦАП. Висока частота дозволяє використовувати зовнішні компоненти фізично невеликого розміру (котушки, конденсатори), що знижує загальну вартість системи.

Роздільна здатність ШІМ для швидкого ШІМ може бути встановлена як 8-біт, 9-біт або 10-біт або визначена за допомогою ICR1 або OCR1A. Мінімальна дозволена роздільна здатність — 2 біти (ICR1 або OCR1A встановлено як 0x0003), а максимальна — 16 бітів (ICR1 або OCR1A встановлено на максимальне значення). Роздільну здатність ШІМ у бітах можна обчислити за допомогою такого рівняння:

$$R_{ШШІМ} = \frac{\log(\text{верхнє значення}+1)}{\log(2)} \qquad (3.2)$$

У режимі швидкого ШІМ лічильник збільшується, доки значення лічильника не збігається з одним із фіксованих значень 0x00FF, 0x01FF або 0x03FF (WGM13:0 = 5, 6 або 7), значенням у ICR1 (WGM13:0 = 14), або значенням в OCR1A (WGM13:0 = 15). Потім лічильник обнулюється в наступному такті таймера. Часова діаграма для режиму швидкої ШІМ показана на рисунку 3.18. Тут показано режим швидкої ШІМ, коли OCR1A або ICR1 використовуються для визначення верхнього значення. Значення TCNT1 на часовій діаграмі показано як гістограму для ілюстрації роботи з одним нахилом. На схемі представлені неінвертований і інвертований ШІМ-виходи. Маленькі горизонтальні лінії на схилах TCNT1 представляють збіги при порівнянні між OCR1x і TCNT1. Прапор переривання OC1x буде встановлено, коли відбувається збіг при порівнянні.

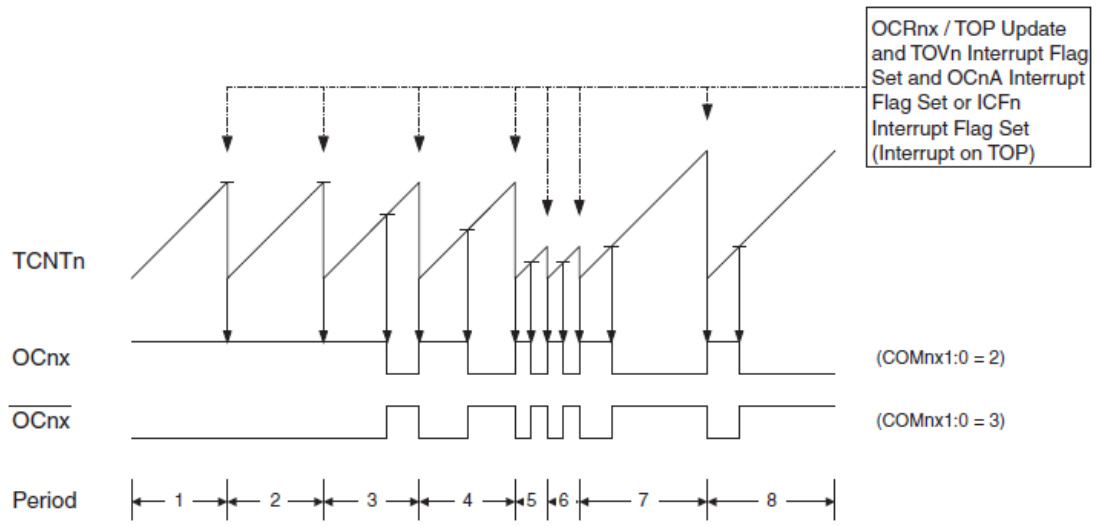


Рисунок 3.18 – Часова діаграма для режиму швидкої ШІМ

Прапор переповнення таймера/лічильника (TOV1) встановлюється щоразу, коли лічильник досягає верхнього значення. Крім того, прапор OCF1A або ICF1 встановлюється на той самий такт таймера, що й TOV1, коли OCR1A або ICR1 використовуються для визначення верхнього значення Якщо одне з переривань увімкнено, підпрограма обробки переривань може бути використана для оновлення верхнього значення і порівняння значень.

Під час зміни верхнього значення програма повинна переконатися, що нове верхнє значення вище або дорівнює значенню всіх регістрів порівняння. Якщо верхнє значення нижче за будь-який з регістрів порівняння, порівняння ніколи не відбудеться між TCNT1 і OCR1x. Зауважте, що при використанні фіксованих верхніх значень невикористані біти маскуються як нулі під час запису будь-якого з регістрів OCR1x.

Процедура оновлення ICR1 відрізняється від оновлення OCR1A, коли він використовується для визначення верхнього значення. Регістр ICR1 не має подвійної буферизації. Це означає, що якщо ICR1 змінено на низьке значення, коли лічильник працює без попереднього дільника або з низьким значенням попереднього дільника, існує ризик того, що нове записане значення ICR1 буде нижчим за поточне значення TCNT1. Тоді результатом буде те, що лічильник пропустить збіг при порівнянні із верхнім значенням. Потім лічильник буде рахувати до максимального значення (0xFFFF) і починати з 0x0000, перш ніж відбудеться порівняння. Регістр OCR1A, однак, має подвійну буферизацію. Ця функція дозволяє будь-коли записувати будь-яке значення у OCR1A. Після запису у OCR1A записане значення буде поміщено в буферний регістр OCR1A. Потім регістр порівняння OCR1A буде оновлено значенням у регістрі буфера під час наступного такту таймера, коли TCNT1 відповідає верхньому значенню. Оновлення виконується у той самий цикл таймера, коли TCNT1 очищується та встановлюється прапор TOV1. Застосування регістра ICR1

для визначення верхнього значення добре працює при використанні фіксованих верхніх значень. Використовуючи ICR1, регістр OCR1A можна вільно застосовувати для генерації виходу ШІМ на OC1A. Однак, якщо базова частота ШІМ активно змінюється (шляхом зміни верхнього значення), використання OCR1A як верхнього значення є кращим вибором через його функцію подвійної буферизації.

У режимі швидкого ШІМ блоки порівняння дозволяють генерувати сигнали ШІМ на виводах OC1x. Встановлення бітів COM1x1:0 як 2 створить неінвертований ШІМ, а інвертований вихід ШІМ можна отримати, встановивши COM1x1:0 як 3. Фактичне значення OC1x буде видно на виводі порту, якщо напрям даних для цього виводу встановлено як вихід (DDR_OC1x). ШІМ-сигнал генерується встановленням (або очищенням) регістра OC1x при збігу між OCR1x і TCNT1, а також очищенням (або встановленням) регістра OC1x під час очищення лічильника таймера (коли його значення змінюється з верхнього на нижнє).

Частоту ШІМ можна розрахувати за таким рівнянням:

$$f_{OCnxШІМ} = \frac{f_{CLK\_I/O}}{N \cdot (1 + \text{верхнє значення})} \quad (3.3)$$

Змінна N представляє собою значення попереднього дільника (1, 8, 64, 256 або 1024).

Крайні значення регістра OCR1x уявляють собою особливі випадки під час генерації сигналу ШІМ у режимі швидкого ШІМ. Якщо OCR1x встановлено рівним нижньому значенню (0x0000), результатом буде вузький сплеск для кожного такту таймера TOP+1. Встановлення OCR1x рівним верхньому значенню призведе до постійного високого або низького рівня вихідного сигналу (залежно від полярності вихідного сигналу, встановленого бітами COM1x1:0).

Прямокутний вихідний сигнал (з робочим циклом 50%) у швидкому ШІМ-режимі може бути досягнутий шляхом налаштування OC1A на перемикання логічного рівня під час кожного збігу при порівнянні (COM1A1:0 = 1). Це стосується лише того випадку, коли OCR1A використовується для визначення верхнього значення (WGM13:0 = 15). Згенерований сигнал матиме максимальну частоту $f_{OC1A} = f_{clk\_I/O}/2$, коли OCR1A встановлено як нуль (0x0000). Ця функція подібна до перемикання OC1A в режимі OT3, за винятком того, що функція подвійної буферизації блоку порівняння увімкнена в режимі швидкого ШІМ.

Режим фазо-коректної широтно-імпульсної модуляції. Режим фазо-коректної широтно-імпульсної модуляції або фазо-коректного ШІМ (WGM13:0 = 1, 2, 3, 10 або 11) забезпечує генерацію сигналу ШІМ з високою роздільною здатністю. Режим фазо-коректного ШІМ , як і фазо- та частото-коректного ШІМ, заснований на роботі з подвійним нахилом. Лічильник спочатку веде рахунок від нижнього значення (0x0000) до

верхнього, а потім від верхнього до нижнього. У неінвертованому режимі порівняння вихідний вивід (OC1x) очищується при збігу між TCNT1 і OCR1x під час підрахунку вгору та встановлюється при збігу під час зворотного підрахунку. У режимі інвертування результатів порівняння операція інвертується. Робота з подвійним нахилом має нижчу максимальну робочу частоту, ніж робота з одним нахилом. Однак через симетричну особливість режимів ШІМ з подвійним нахилом ці режими є кращими для додатків керування двигуном.

У фазо-коректному режимі ШІМ лічильник збільшується, доки його значення не збігається з одним із фіксованих значень 0x00FF, 0x01FF або 0x03FF (WGM13:0 = 1, 2 або 3), значенням у ICR1 (WGM13:0 = 10) або значенням в OCR1A (WGM13:0 = 11). Після цього лічильник досягає верхнього значення та змінює напрямок підрахунку. Значення TCNT1 дорівнюватиме верхньому значенню протягом одного такту таймера. Часова діаграма для фазо-коректного режиму ШІМ показана на рисунку 3.19. Тут показано фазо-коректний режим ШІМ, коли OCR1A або ICR1 використовуються для визначення верхнього значення. Значення TCNT1 на часовій діаграмі показано як гістограма для ілюстрації роботи подвійного нахилу. На схемі представлені неінвертований і інвертований ШІМ-виходи. Маленькі горизонтальні лінії на схилах TCNT1 представляють порівняльні збіги між OCR1x і TCNT1. Прапор переривання OC1x буде встановлено, коли відбувається збіг при порівнянні.

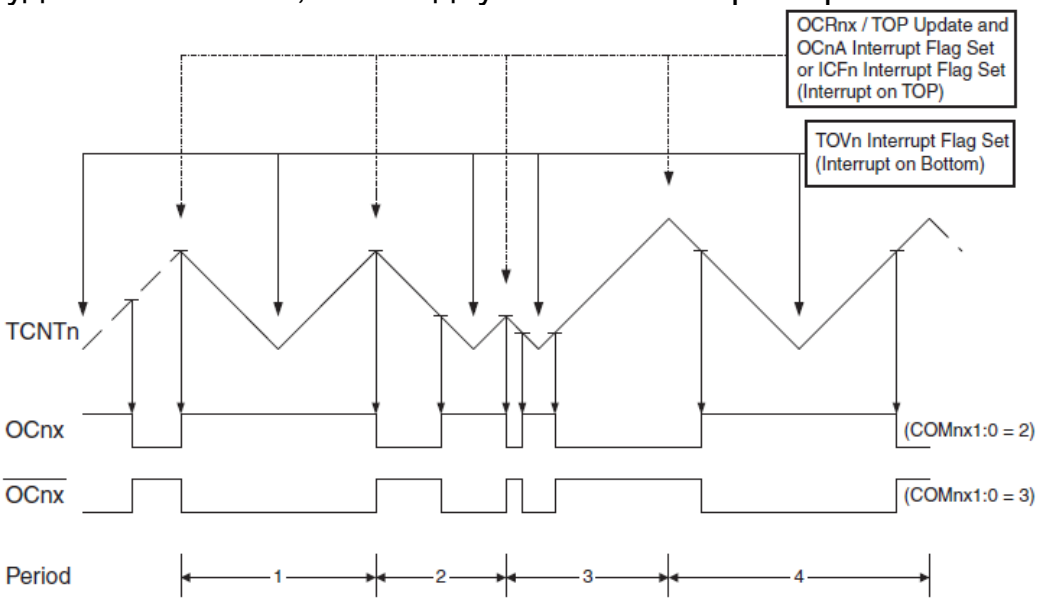


Рисунок 3.19 – Часова діаграма для фазо-коректного режиму ШІМ

Режим фазо- та частото-коректного ШІМ. Режим фазо- та частото-коректної широтно-імпульсної модуляції або фазо- та частото-коректного ШІМ (WGM13:0 = 8 або 9) забезпечує генерацію ШІМ-сигналу високої роздільної здатності з правильною фазою та частотою. Режим фазо- та частото-коректного ШІМ, як і режим фазо-коректного ШІМ, заснований на

роботі з подвійним нахилом. Лічильник багаторазово веде підрахунок від нижнього значення (0x0000) до верхнього, а потім від верхнього до нижнього. У неінвертованому режимі порівняння вихід порівняння (OC1x) очищується при збігу між TCNT1 і OCR1x під час рахунку вгору та встановлюється при збігу при порівнянні під час зворотного рахунку. У режимі інвертування виходу порівняння операція інвертується. Робота з подвійним нахилом дає нижчу максимальну робочу частоту порівняно з роботою з одним нахилом. Однак через симетричну особливість режимів ШІМ з подвійним нахилом ці режими є кращими для додатків керування двигуном.

Основна відмінність між режимом фазо-коректного ШІМ та фазо- та частото-коректного ШІМ полягає в часі оновлення регістра OCR1x буферним регістром OCR1x (див. рисунок 3.20).

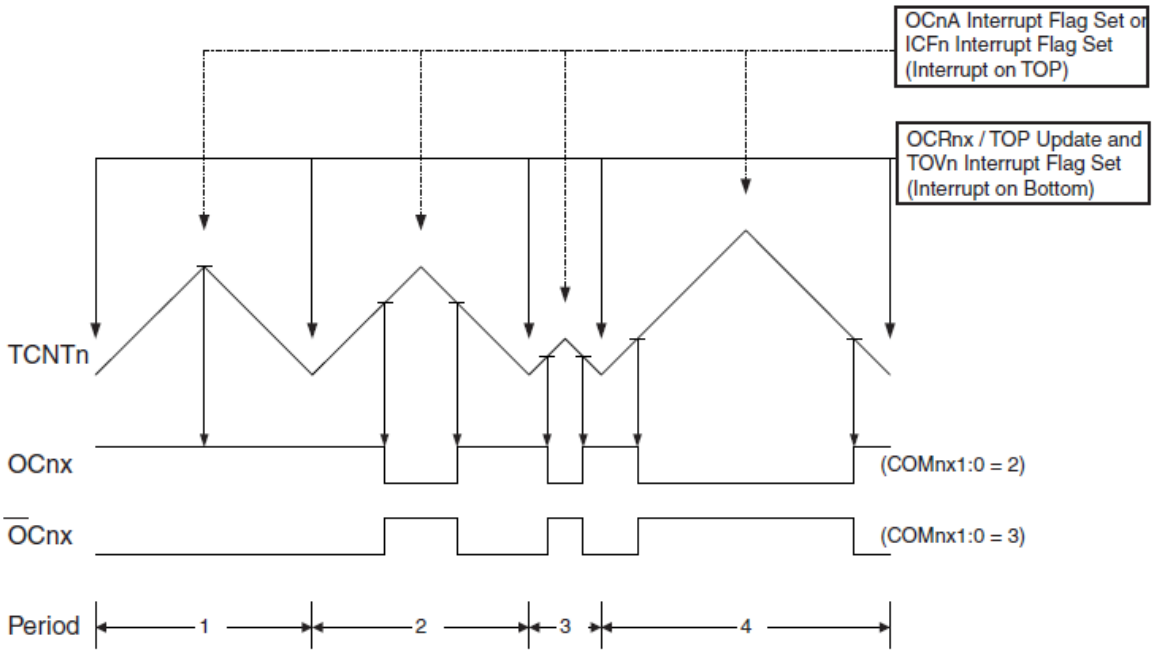


Рисунок 3.20 – Часова діаграма фазо- та частото-коректного ШІМ

Опис **регістрів 16-бітного таймера/лічильника 1 мікроконтролера ATMega8:**

Регістр керування А таймера/лічильника 1 – TCCR1A зображено на рисунку 3.21

Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	W	W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 3.21 – Регістр керування А таймера/лічильника 1

• Біт 7:6 – COM1A1:0: режим порівняння для каналу A.

• Біт 5:4 – COM1B1:0: режим порівняння для каналу B.

COM1A1:0 і COM1B1:0 керують поведінкою виводів порівняння (OC1A і OC1B відповідно). Якщо один або обидва біти COM1A1:0 записані як 1, вихід OC1A перекриває нормальну роботу виводу, до якого він підключений. Якщо один або обидва біти COM1B1:0 записуються як 1, вихід OC1B перекриває нормальну роботу виводу, до якого він підключений. Однак зауважте, що біт регістра направлення даних (DDR), який відповідає виводу OC1A або OC1B, має бути встановлено окремо, щоб увімкнути вихідний драйвер. Коли OC1A або OC1B підключено до виводу, функція бітів COM1x1:0 залежить від налаштування бітів WGM13:0. Таблиця 3.5 показує функціональні можливості бітів COM1x1:0, коли біти WGM13:0 встановлено, як звичайний режим або режим OT3 (не ШІМ). Таблиця 3.6 показує функціональність бітів COM1x1:0, коли біти WGM13:0 встановлено як режим швидкого ШІМ (особливий випадок виникає, коли OCR1A/OCR1B дорівнює верхньому значенню і встановлено COM1A1/COM1B1. У цьому випадку збіг при порівнянні ігнорується, але встановлення або очищення виконується при нижньому значенні). Таблиця 3.7 показує функціональність бітів COM1x1:0, коли біти WGM13:0 встановлені, як фазо-коректний, або фазо- та частото-коректний ШІМ.

Таблиця 3.5 – Функціональні можливості бітів COM1x1:0, коли біти WGM13:0 встановлено, як звичайний режим або режим OT3 (не ШІМ)

COM1A1/ COM1B1	COM1A0/ COM1B0	Опис
0	0	Нормальна робота виводу, OC1A/OC1B відключено
0	1	Перемикання OC1A/OC1B при збігу
1	0	Очищення OC1A/OC1B при збігу (встановити вихід на низький рівень)
1	1	Встановлення OC1A/OC1B при збігу (встановити вихід на високий рівень)

Таблиця 3.6 – Функціональність бітів COM1x1:0, коли біти WGM13:0 встановлено як режим швидкого ШІМ

COM1A1/ COM1B1	COM1A0/ COM1B0	Опис
0	0	Нормальна робота виводу, OC1A/OC1B відключено
0	1	При WGM13:0 = 15: перемикання OC1A при збігу, OC1B відключено (нормальна робота порту). Для всіх інших налаштувань WGM1 - нормальна робота порту, OC1A/OC1B відключено.

COM1A1/ COM1B1	COM1A0/ COM1B0	Опис
1	0	Очищення OC1A/OC1B при збігу, встановлення OC1A/OC1B при нижньому значенні (режим без інвертування)
1	1	Встановлення OC1A/OC1B при збігу, очищення OC1A/OC1B при верхньому значенні (режим з інвертуванням)

Таблиця 3.7 – Функціональність бітів COM1x1:0, коли біти WGM13:0 встановлені, як фазо-коректний, або фазо- та частото-коректний ШІМ.

COM1A1/ COM1B1	COM1A0/ COM1B0	Опис
0	0	Нормальна робота виводу, OC1A/OC1B відключено
0	1	При WGM13:0 = 9 або 14: перемикання OC1A при порівнянні, OC1B відключено (нормальна робота порту). Для всіх інших налаштувань WGM1 - нормальна робота порту, OC1A/OC1B відключено.
1	0	Очищення OC1A/OC1B при збігу при рахунку вгору. Встановлення OC1A/OC1B при збігу при рахунку вниз
1	1	Встановлення OC1A/OC1B при збігу при рахунку вгору. Очищення OC1A/OC1B при збігу при рахунку вниз

• Біт 3 – FOC1A: Примусове порівняння для каналу A.

• Біт 2 – FOC1B: Примусове порівняння для каналу B.

Біти FOC1A/FOC1B активні лише тоді, коли біти WGM13:0 встановлюють режим без ШІМ. Однак для забезпечення сумісності з майбутніми пристроями ці біти повинні бути встановлені як 0, при запису TCCR1A під час роботи в режимі ШІМ. Під час запису логічної одиниці в біт FOC1A/FOC1B негайний збіг при порівняні примусово виконується у модулі генерації сигналу. Вихід OC1A/OC1B змінюється відповідно до налаштування бітів COM1x1:0.

Встановлення бітів FOC1A/FOC1B не генеруватиме жодних переривань і не очищатиме таймер у режимі ОТ3, використовуючи OCR1A як верхнє значення.

Біти FOC1A/FOC1B завжди читаються як нульові.

• Біт 1:0 – WGM11:0: режим генерації сигналу У поєднанні з бітами WGM13:2, що знаходяться в регістрі TCCR1B, ці біти керують послідовністю рахунку лічильника, джерелом верхнього значення лічильника та видом генерації сигналу. Режими роботи, що підтримує блок таймера/лічильника: звичайний режим (лічильник), режим очищення таймера при збігу (ОТ3) і три типи режимів широтно-імпульсної модуляції (ШІМ). Відповідність між значеннями бітів WGM13:0 та режимами роботи показана у таблиці 3.8.

Таблиця 3.8 – Відповідність між значеннями бітів WGM13:0 та режимами роботи

Режим	WGM13	WGM12	WGM11	WGM10	Режим роботи таймера/лічильника	Верхнє значення	Оновлення OCR1x	Встановлення прапору TOV1 при
0	0	0	0	0	Звичайний	0xFFFF	Одразу	максимальному зн.
1	0	0	0	1	Фазо-коректний ШІМ (8-бит)	0x00FF	Верхнє зн.	нижньому зн.
2	0	0	1	0	Фазо-коректний ШІМ (9-бит)	0x01FF	Верхнє зн.	нижньому зн.
3	0	0	1	1	Фазо-коректний ШІМ (10-бит)	0x03FF	Верхнє зн.	нижньому зн.
4	0	1	0	0	ОТЗ	OCR1A	Одразу	максимальному зн.
5	0	1	0	1	Швидкий ШІМ (8-бит)	0x00FF	Нижнє зн.	верхньому зн.
6	0	1	1	0	Швидкий ШІМ (9-бит)	0x01FF	Нижнє зн.	верхньому зн.
7	0	1	1	1	Швидкий ШІМ (10-бит)	0x03FF	Нижнє зн.	верхньому зн.
8	1	0	0	0	Фазо- та частото-коректний ШІМ	ICR1	Нижнє зн.	нижньому зн.
9	1	0	0	1	Фазо- та частото-коректний ШІМ	OCR1A	Нижнє зн.	нижньому зн.
10	1	0	1	0	Фазо-коректний ШІМ	ICR1	Верхнє зн.	нижньому зн.
11	1	0	1	1	Фазо-коректний ШІМ	OCR1A	Верхнє зн.	нижньому зн.
12	1	1	0	0	ОТЗ	ICR1	Одразу	максимальному зн.
13	1	1	0	1	Зарезервовано	-	-	-
14	1	1	1	0	Швидкий ШІМ	ICR1	Нижнє зн.	верхньому зн.
15	1	1	1	1	Швидкий ШІМ	OCR1A	Нижнє зн.	верхньому зн.

Регістр керування B таймера/лічильника 1 – TCCR1B зображено на рисунку 3.22.

Bit	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 3.22 – Регістр керування B таймера/лічильника 1

• **Біт 7 – ICNC1: фільтр шуму блока захоплення.** Встановлення цього біта (в одиницю) активує фільтр шуму блока захоплення. Коли фільтр шуму активовано, вхідний сигнал із входу захоплення (ICP1) фільтрується. Для роботи фільтра потрібні чотири послідовні однакові значення виводу ICP1 для зміни вихідного сигналу. Тому захоплення вхідного сигналу затримується на чотири цикли осцилятора, коли фільтр шуму увімкнено.

• **Біт 6 – ICES1: вибір фронту захоплення.** Цей біт вибирає фронт на вході захоплення (ICP1), який використовується для запуску події захоплення. Коли біт ICES1 записаний як нуль, спадаючий (негативний) фронт використовується як тригер, а коли біт ICES1 записаний як одиниця, наростаючий (позитивний) фронт ініціює захоплення. Коли захоплення запускається відповідно до налаштування біту ICES1, значення лічильника копіюється у вхідний регістр захоплення (ICR1). Подія також встановлює прапор захоплення (ICF1), і його можна використовувати для виклику переривання захоплення, якщо це переривання ввімкнено. Коли ICR1 використовується як верхнє значення (див. опис бітів WGM13:0, розташованих у TCCR1A та регістрі TCCR1B), вивід ICR1 від'єднується, і, отже, функція захоплення вимикається.

• **Біт 5 – зарезервований біт.** Цей біт зарезервовано для майбутнього використання. Для забезпечення сумісності з майбутніми пристроями цей біт має бути записаний як нуль під час запису TCCR1B.

• **Біти 4:3 – WGM13:2: Режим генерації сигналу.** Див. опис реєстру TCCR1A.

• **Біт 2:0 – CS12:0: Вибір тактового сигналу.** Три біти вибору тактового сигналу обирають джерело тактового сигналу, який буде використовуватися таймером/лічильником. Якщо для таймера/лічильника 1 використовуються режими зовнішніх імпульсів, зміни рівня на виводі T1 будуть тактувати лічильник, навіть якщо цей контакт налаштовано як вихід (табл 3.9). Ця функція дозволяє програмно контролювати рахунок.

Таблиця 3.9 – Налаштування таймера/лічильника в залежності від бітів

CS12	CS11	CS10	Опис
0	0	0	Немає тактового сигналу (таймер/лічильник зупинений)
0	0	1	$clk_{I/O}$ (без попереднього дільника)
0	1	0	$clk_{I/O}/8$ (від попереднього дільника)
0	1	1	$clk_{I/O}/64$ (від попереднього дільника)
1	0	0	$clk_{I/O}/256$ (від попереднього дільника)
1	0	1	$clk_{I/O}/1024$ (від попереднього дільника)
1	1	0	Зовнішній тактовий сигнал на виводі T1. Синхронізація по спадаючому фронту
1	1	1	Зовнішній тактовий сигнал на виводі T1. Синхронізація по зростаючому фронту

Таймер/Лічильник 1 – TCNT1H і TCNT1L (рис. 3.23).

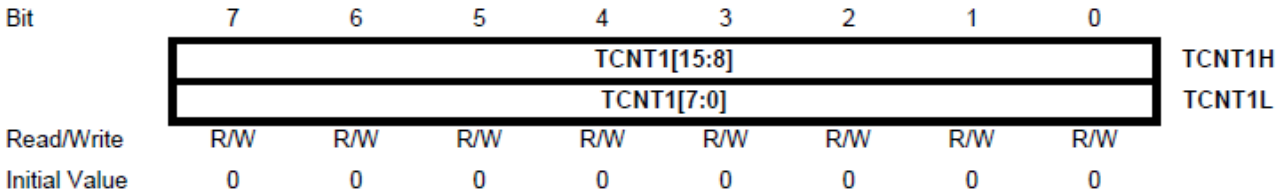


Рисунок 3.23 – Таймер/Лічильник 1 – TCNT1H і TCNT1L

Два регістри вводу/виводу таймера/лічильника (TCNT1H і TCNT1L, комбінований TCNT1) забезпечують прямий доступ до 16-розрядного лічильника блоку таймера/лічильника як для операцій читання, так і для запису. Щоб переконатися, що старший і молодший байти читаються і записуються одночасно, коли ЦП отримує доступ до цих регістрів, доступ виконується за допомогою 8-розрядного тимчасового регістра старшого байта (TEMP). Цей тимчасовий регістр спільно використовується всіма іншими 16-розрядними регістрами.

Зміна значення лічильника (TCNT1) під час його роботи створює ризик втрати події збігу при порівнянні між TCNT1 і одним із регістрів OCR1x.

Запис у регістр TCNT1 блокує (видаляє) збіг при порівнянні на наступному тактовому імпульсі таймера для всіх модулів порівняння.

Регістр порівняння 1 A – OCR1AH і OCR1AL (рис 3.24).

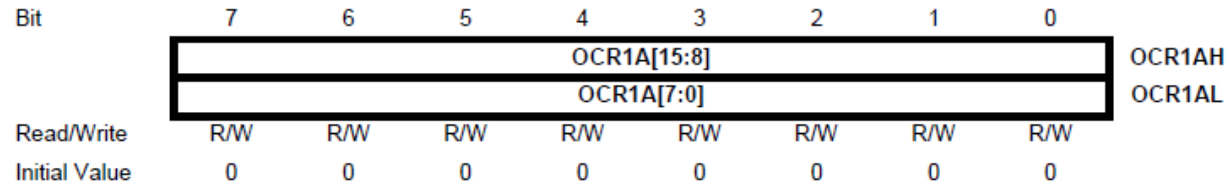


Рисунок 3.24 – Регістр порівняння 1 A – OCR1AH і OCR1AL

Регістр порівняння 1 B – OCR1BH і OCR1BL (рис. 3.25).

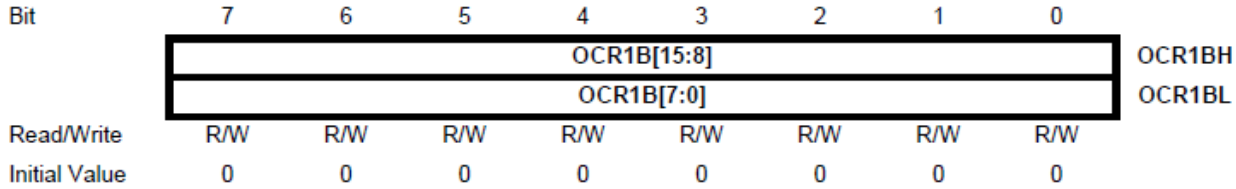


Рисунок 3.25 – Регістр порівняння 1 B – OCR1BH і OCR1BL

Регістри порівняння містять 16-бітне значення, яке постійно порівнюється зі значенням лічильника (TCNT1). Збіг може бути використаний для генерації переривання при збігу при порівнянні або для генерації вихідного сигналу на виводі OC1x.

Регістри порівняння мають 16-бітний розмір. Щоб переконатися, що старший і молодший байти записуються одночасно, коли ЦП записує в ці регістри, доступ виконується за допомогою 8-розрядного тимчасового регістра старшого байта (TEMP). Цей тимчасовий регістр спільно використовується всіма іншими 16-розрядними регістрами.

Регістр захоплення 1 – ICR1H і ICR1L (рис. 3.26).

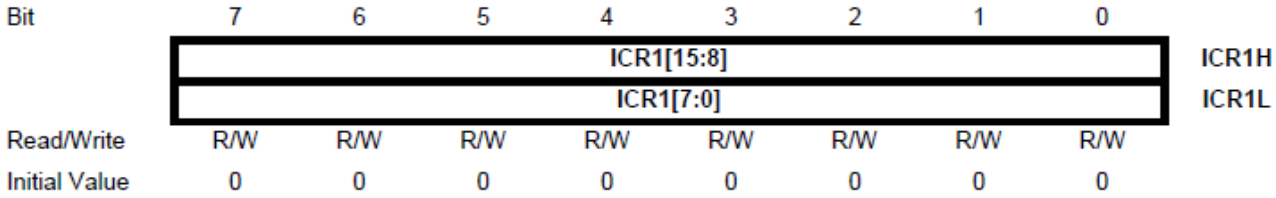


Рисунок 3.26 – Регістр захоплення 1 – ICR1H і ICR1L

Регістр захоплення оновлюється значенням лічильника (TCNT1) кожного разу, коли відбувається подія на виводі ICP1 (або за бажанням на виході аналогового компаратора для таймера/лічильника1). Регістр захоплення можна використовувати для встановлення верхнього значення лічильника.

Регістр захоплення має 16-бітний розмір. Щоб забезпечити одночасне зчитування як старшого, так і молодшого байтів, коли ЦП звертається до цих регістрів, доступ виконується за допомогою 8-розрядного тимчасового регістра старшого байта (TEMP). Цей тимчасовий регістр спільно використовується всіма іншими 16-розрядними регістрами.

Регістр маски переривання таймерів/лічильників – TIMSK (рис. 3.27)

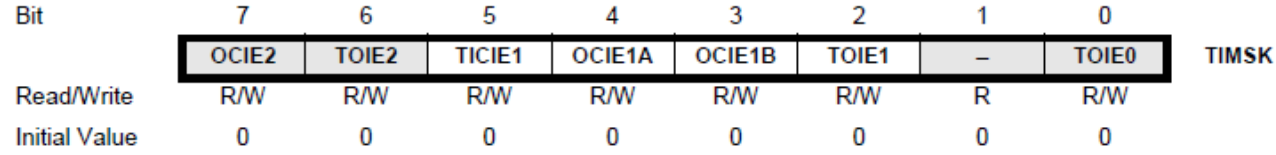


Рисунок 3.27 – Регістр маски переривання таймерів/лічильників – TIMSK

• Біт 5 – TICIE1: таймер/лічильник1, увімкнення переривання при захопленні. Коли цей біт встановлено як 1, і прапор I у регістрі стану встановлено (глобальні переривання ввімкнено), переривання при захопленні для таймера/лічильника1 увімкнено. Відповідний вектор

переривання активується, коли встановлюється прапор ICF1, розташований у TIFR.

• Біт 4 – OCIE1A: Таймер/Лічильник1, увімкнення переривання при збігу при порівнянні A. Коли цей біт встановлено як 1, і прапор I у регістрі стану встановлено, переривання при збігу при порівнянні A для таймера/лічильника1 увімкнено. Відповідний вектор переривання активується, коли встановлюється прапор OCF1A, розташований у TIFR.

• Біт 3 – OCIE1B: Таймер/Лічильник1, увімкнення переривання при збігу при порівнянні B. Коли цей біт встановлено як 1, і прапор I у регістрі стану встановлено, переривання при збігу при порівнянні B для таймера/лічильника1 увімкнено. Відповідний вектор переривання активується, коли встановлюється прапор OCF1B, розташований у TIFR.

• Біт 2 – TOIE1: таймер/лічильник1, увімкнення переривання при переповненні. Коли цей біт встановлено як 1, і прапор I у регістрі стану встановлено, переривання при переповненні таймера/лічильника1 увімкнено. Відповідний вектор переривання активується, коли встановлюється прапор TOV1, розташований у TIFR.

Регістр прапорів переривання таймерів/лічильників – TIFR (рис. 3.28).

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	–	TOV0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 3.28 – Регістр прапорів переривання таймерів/лічильників – TIFR

• Біт 5 – ICF1: таймер/лічильник1, прапор захоплення. Цей прапор встановлюється, коли подія захоплення відбувається на виводі ICP1. Коли регістр захоплення вхідних даних (ICR1) встановлений бітами WGM13:0 для використання як верхнє значення, прапор ICF1 встановлюється, коли лічильник досягає значення верхнього значення. ICF1 автоматично очищується, коли виконується підпрограма обробки переривання захоплення. Крім того, ICF1 можна очистити, записавши логічну одиницю у нього.

• Біт 4 – OCF1A: Таймер/Лічильник 1, прапор збігу при порівнянні A. Цей прапор встановлюється в наступному тактовому циклі таймера після того, як значення лічильника (TCNT1) збігається з вихідним регістром порівняння A (OCR1A). Зверніть увагу, що строб примусового порівняння (FOC1A) не встановлює прапор OCF1A. OCF1A автоматично очищується, коли виконується підпрограма обробки переривання збігу при порівнянні A. Крім того, OCF1A можна очистити, записавши логічну одиницю в нього.

• **Біт 4 – OCF1A: Таймер/Лічильник 1, прапор збігу при порівнянні B.** Цей прапор встановлюється в наступному тактовому циклі таймера після того, як значення лічильника (TCNT1) збігається з вихідним регістром порівняння B (OCR1B). Зверніть увагу, що строб примусового порівняння (FOC1B) не встановлює прапор OCF1B. OCF1B автоматично очищується, коли виконується підпрограма обробки переривання збігу при порівнянні B. Крім того, OCF1B можна очистити, записавши логічну одиницю в нього.

• **Біт 2 – TOV1: таймер/лічильник1, прапор переповнення.** Налаштування цього прапора залежить від налаштування бітів WGM13:0. У звичайному режимі та режимі OT3 прапор TOV1 встановлюється, коли таймер переповнюється. Зверніться до таблиці 3.8 щодо поведінки прапора TOV1 під час використання інших налаштувань бітів WGM13:0. TOV1 автоматично очищується, коли виконується підпрограма обробки переривання переповнення таймера/лічильника 1. Альтернативно, TOV1 можна очистити, записавши логічну одиницю в нього.

3.3 Приклад програми

Виконаємо наступне завдання на базі мікроконтролера ATMega8. Підключити два світлодіоди LED1 та LED2 до виводів PB4 та PB1 (OC1A).

Встановити тактову частоту таймера 0 як $f_{CLK\_I/O}/1024$. При переповненні таймера 0 перемикати світлодіод LED1 у протилежний стан.

Встановити тактову частоту таймера 1 як $f_{CLK\_I/O}/8$, та режим як швидкий ШІМ (10 біт). Світлодіод LED2 повинен керуватися виводом OC1A модуля порівняння. Яскравість світлодіода підвищувати на 1 при кожному переповненні таймера 1. При досяганні максимальної яскравості, встановити її рівною 0.

3.3.1 Принципова електрична схема

Принципова електрична схема, що реалізує поставлену задачу, показана на рис. 3.29.

Ніяких нових компонентів тут немає, єдина відмінність від попередніх схем полягає в тому, що тут замість зовнішнього підключення катодів світлодіодів до землі, вони підключені до неї за допомогою опції «Grounded» у налаштуваннях світлодіодів. Але на функціональність це ніяк не впливає, то ж можно підключати їх як завгодно.

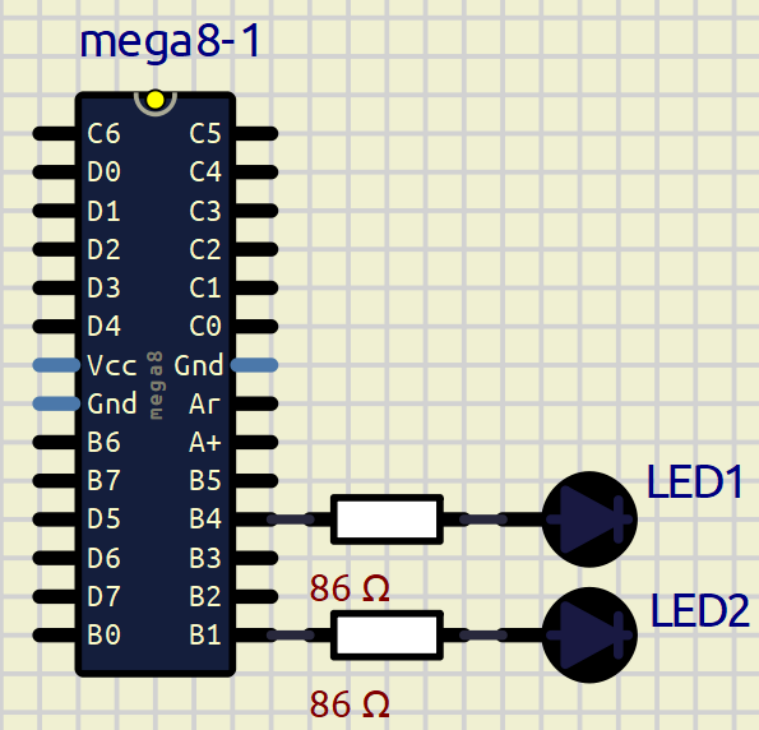


Рисунок 3.29 – Принципова електрична схема

3.3.2 Програмний код

Розглянемо тепер програму, що реалізує поставлену задачу (рис. 3.30).

У рядку 1 підключається заголовковий файл «io.h», який містить назви всіх бітів та регістрів, що використовуватимуться далі у програмі.

У рядку 2 підключається заголовковий файл «interrupt.h», який потрібно додавати, якщо в програмі планується використовувати переривання. Оскільки ми тут будемо мати справу з перериваннями по переповненню таймерів, то цей файл необхідно додати.

Рядки 4-16 поки пропустимо, і перейдемо до головної функції програми, що розташована у рядках 18-28. Її відмінність від всіх інших головних функцій, що ми розглядали раніше, полягає в тому, що в ній основний цикл програми є порожнім (рядок 27). Тобто процесор більшу частину часу не робить нічого, а тільки чекає на переривання. У такому випадку було б логічніше перевести його у неактивний режим сну, і у реальних пристроях так і рекомендовано робити для зменшення енергоспоживання, але оскільки у програмі SimulIDE режими сну не працюють коректно, ми тут цього не робимо. Не дивлячись на те, що процесор весь час тільки очікує на переривання, не можна не створювати основний безкінечний цикл, оскільки без нього програма буде періодично перезавантажуватися з початку й ініціалізувати модулі мікроконтролера знову і знову, що є небажаним.

```
 1 #include <avr/io.h>
 2 #include <avr/interrupt.h>
 3
 4 ISR(TIMER0_OVF_vect)
 5 {
 6   PORTB ^= _BV(PB4);
 7 }
 8
 9 ISR(TIMER1_OVF_vect)
10 {
11   OCR1A ++;
12   if (OCR1A > 0x03FF)
13   {
14     OCR1A = 0;
15   }
16 }
17
18 int main (void)
19 {
20   DDRB = _BV(PB1) | _BV(PB4);
21   TCCR0 = _BV(CS00) | _BV(CS02);
22   TCCR1A = _BV(COM1A1) | _BV(WGM10) | _BV(WGM11);
23   TCCR1B = _BV(WGM12) | _BV(CS11);
24   OCR1A = 0x00;
25   TIMSK |= _BV(TOIE0) | _BV(TOIE1);
26   sei();
27   while(1);
28 }
```

Рисунок 3.30 – Програмний код

Тепер розглянемо ініціалізаційну частину головної функції (рядки 20-26).

У рядку 20 виводи PB4 та PB1 конфігуруються, як виходи (оскільки до них під’єднані світлодіоди), шляхом встановлення відповідних бітів у регістрі DDRB.

У рядку 21 ми конфігуруємо таймер 0. У нього насправді не так і багато налаштувань. Все, що ми можемо зробити – це обрати тактовий сигнал таймера, після чого він одразу почне рахунок. За завданням нам треба встановити частоту тактування таймера як $f_{CLK\_I/O}/1024$. Це відповідає наступній комбінації бітів CS02...CS00 (відповідно до таблиці на 3.3): CS02 = 1, CS01 = 0, CS00 = 1. То ж у рядку 21 ми встановлюємо біти CS02 та CS00 у регістрі TCCR0. Оскільки ми використовуємо просту операцію присвоєння «=«, біт CS01 автоматично стає рівним 0.

Порахуємо, з якою частотою буде блимати світлодіод у даному випадку. Якщо тактова частота процесора дорівнює 1 МГц, то тактова частота таймера 0 буде дорівнювати 1000000 Гц / 1024 ≈ 977 Гц. Таймер 0 є 8-розрядним. Це значить, що він може рахувати від 0 до 255, після чого станеться переповнення і згенерується переривання. То ж частота між перериваннями буде дорівнювати 977 / 256 = 3,8 Гц. Оскільки при кожному перериванні по переповненню таймера стан світлодіода буде змінено на протилежний, частота його блимання буде дорівнювати 3.8 / 2 = 1.9 Гц.

У рядках 22-23 ми конфігуруємо таймер 1 за допомогою регістрів TCCR1A та TCCR1B. За завданням треба встановити частоту таймера 1 як $f_{CLK\_I/O}/8$. Ця частота задається бітами CS12…CS10 регістру TCCR1B відповідно до таблиці 3.9. Для частоти $f_{CLK\_I/O}/8$: CS12 = 0, CS11 = 1, CS10 = 0. То ж у рядку 23 ми встановлюємо лише біт CS11, залишаючи CS12 і CS10 рівними 0.

Режим роботи таймера 1 задається бітами WGM13…WGM10, причому перша половина з них знаходиться у регістрі TCCR1B, а друга – у регістрі TCCR1A. За завданням треба встановити режим роботи «Швидкий ШІМ (10 біт)». З таблиці 3.8 знаходимо, що цьому режиму відповідає така комбінація цих бітів: WGM13 = 0, WGM12 = 1, WGM11 = 1, WGM10 = 1.

Порахуємо частоту між перериваннями таймера 1. Його тактова частота дорівнює 1000000 Гц / 8 = 125000 Гц. Період таймера становить 1024 тактів (це відповідає 10-бітній розрядності 2^{10}=1024), тому повний цикл таймера відбувається з частотою 125000 / 1024 ≈ 122 Гц. Щоб не було помітно блимання світлодіода, частота повного циклу таймера повинна бути не менше 50 Гц.

Далі треба налаштувати функціональність виводу OC1A блоку порівняння таймера 1. У мікроконтролера ATMega8 цей вивід суміщений з виводом PB1. У завданні не сказано явно, але яскравість світлодіода можна змінювати у режимі ШІМ. Оберемо неінвертований режим роботи виводу OC1A за допомогою бітів COM1A1 та COM1A0, що розташовані у регістрі TCCR1A, відповідно до верхньої таблиці 3.6. Режим без інвертування задається такою комбінацією цих бітів: COM1A1 = 1, COM1A0 = 0.

То ж, підсумовуючи все вище сказане, треба встановити біти COM1A1, WGM11 та WGM10 у регістрі TCCR1A, та біти WGM12 та CS11 у регістрі TCCR1B, залишивши всі інші біти цих регістрів як 0. Як раз це ми й робимо у рядках 22, 23.

У рядку 24 ми записуємо 0 у регістр OCR1A. Це регістр блока порівняння A таймера 1, відповідно до таблиці 3.6. Чим більше значення, записане у цей регістр, тим пізніше буде встановлюватися низький рівень на виводі OC1A, і тим більша буде середня напруга на ньому, і тим більша буде яскравість світлодіода. Таким чином, змінюючи лише значення, записане у цей регістр, можна керувати величиною вихідної напруги за

допомогою ШІМ. Оскільки ми записали 0 у цей регістр, спочатку яскравість світлодіода буде мінімальною, а точніше, він буде вимкнений.

Вже після цього рядку обидва таймери починають працювати, і ШІМ починає генеруватися на виводі OC1A без будь-якої участі з боку процесору.

У рядку 25 ми демаскуємо (дозволяємо) переривання по переповненню таймера 0 та таймера 1, встановлюючи біти TOIE0 та TOIE1 у регістрі TIMSK. Тепер лишилося тільки дозволити глобальні переривання за допомогою функції sei (рядок 26). Ця функція виконує команду процесора SEI, що встановлює біт I у регістрі статусу, і вона описана у файлі «interrupt.h», що ми підключили у другому рядку нашої програми.

У рядку 27, як вже було сказано, знаходиться порожній безкінечний цикл, у якому процесор знаходиться в очікуванні спрацювання переривань від переповнення таймерів.

Тепер розглянемо безпосередньо підпрограми обробки переривань. Вони уявляють собою просто функції, але на відміну від звичайних функцій, вони не викликаються явно з програми, а викликаються апаратно при настанні умови генерації переривання: встановлений біт I у регістрі статусу, переривання дозволено за допомогою регістру маскування, встановився прапор переривання.

Всі функції обробки переривань мають назву ISR, що як раз і означає підпрограма обробки переривання (Interrupt SubRoutine). В якості аргументу цієї функції передається назва переривання. Назви всіх переривань описані у заголовковому файлі для кожного конкретного мікроконтролера. Але їх можна взяти також з таблиці на рис. 3.1, замінивши пробіли на знак підкреслення, і додавши в кінці «_vect». Наприклад, для переривання по переповненню таймера 0, яке у таблиці називається «TIMER0 OVF» відповідна назва переривання, що передається у функцію ISR, буде «TIMER0_OVF_vect». Цю назву ми і записуємо у рядку 4, де починається функція обробки переривання по переповненню таймера 0. Відповідно до завдання, кожен раз при переповненні таймера 0, ми повинні перемикати стан світлодіода LED1 у протилежний. Це ми робимо у рядку 6 за допомогою оператора «виключне АБО», що перемикає біт PB4 регістра PORTB.

У рядку 9 починається нова функція обробки переривання, але цього разу по переповненню таймера 1, тому в якості аргументу у функцію ISR передається TIMER1_OVF_vect. У цій функції, відповідно до завдання, ми повинні збільшувати яскравість світлодіода на 1. Раніше ми вже з'ясували, що яскравість задається значенням, записаним у регістр OCR1A. То ж всередині цієї функції ми спочатку збільшуємо цей регістр на 1 (рядок 11). Якщо його значення стає більшим 0x03FF (рядок 12), що відповідає десятковому числу 1023 (тобто 2^{10}-1), то регістр OCR1A треба обнулити

(рядок 14). Таким чином, яскравість буде поступово збільшуватися, і при досягнені максимальної величини скинеться у нуль.

Ось, в принципі, і вся програма. Тепер можна скомпілювати файл .hex, завантажити його у мікроконтролер та запустити симуляцію. Для того, щоб подивитися форму сигналу, програма SimulIDE пропонує прилад, який називається осцилограф (у програмі – “Oscope”), який знаходиться у підзаголовку “Meters” (рис. 3.31).

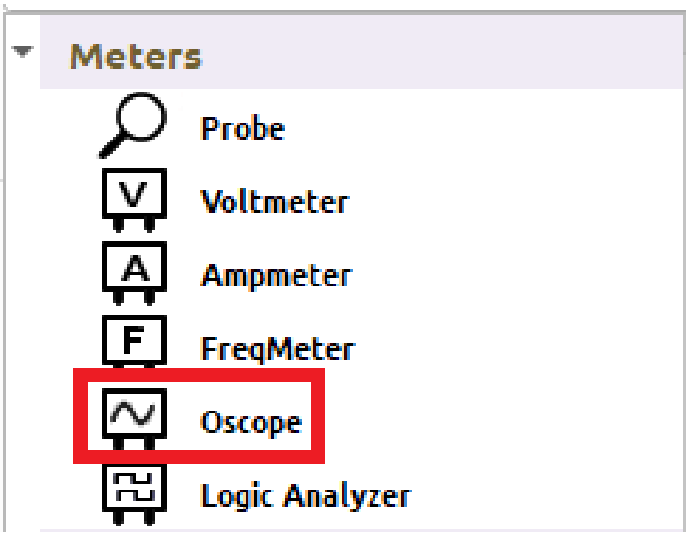


Рисунок 3.31 – Знаходження осцилографа у бібліотеці компонентів

Цей осцилограф є 4-канальним, тобто дозволяє одночасно дивитися до 4 сигналів. Схему з підключеним осцилографом показано на рис. 3.32.

Як бачите, його перший канал підключений до аноду світлодіода LED2, щоб побачити сигнал ШІМ, що приходить на нього. Щоб змінити налаштування осцилографа, треба натиснути на кнопку «Expand» на ньому. Після чого він розгорнеться у окреме вікно, показане на рис. 3.33.

Тут можна встановити роздільну здатність по осі часу «Time Div» та по осі напруги «Volt Div», зміщення графіку по осі абсцис «Time Pos» та по осі ординат «Volt Pos», задати тригер, по якому буде синхронізуватися графік «Trigger», задати режим роботи «Auto», приховати графіки «Hide» та встановити кількість осей абсцис «Tracks».

Після запуску симуляції можна побачити, що при малій ширині імпульсів ШІМ яскравість світлодіода LED2 невелика (рис. 3.34), а при її збільшенні вона також збільшується (рис. 3.35).

При цьому світлодіод LED1 блимає з постійною частотою незалежно від яскравості світлодіода LED2.

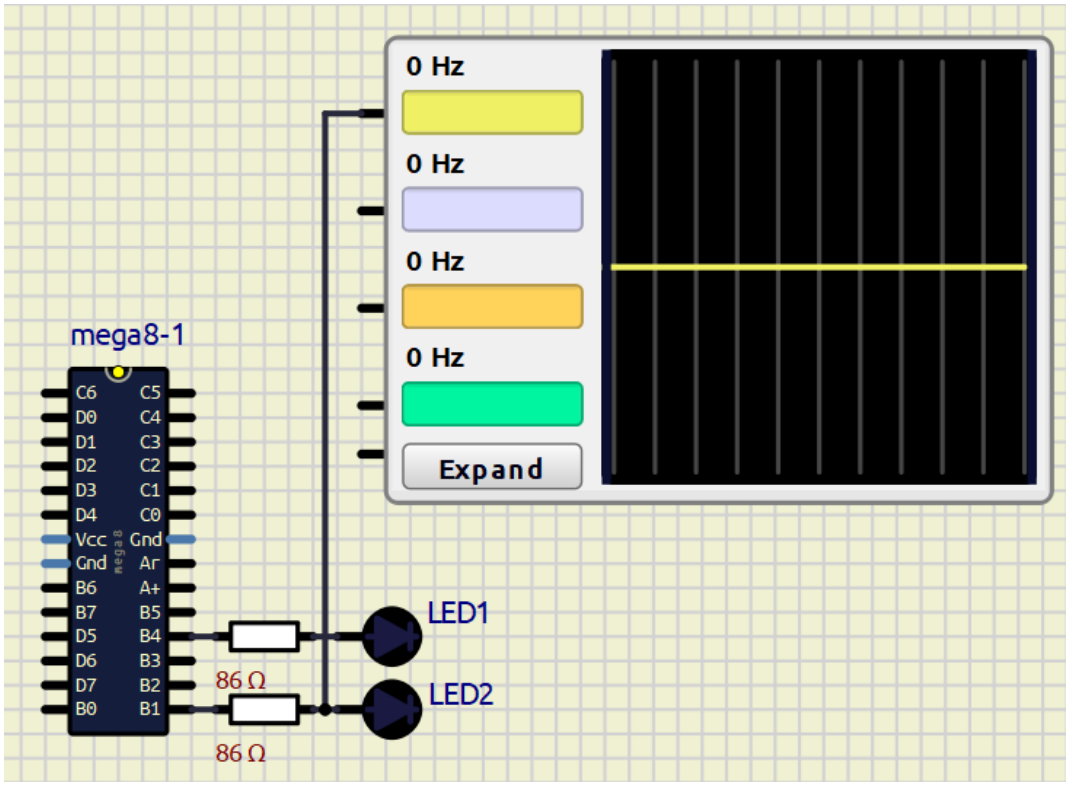


Рисунок 3.32 – Принципова електрична схема з осцилографом

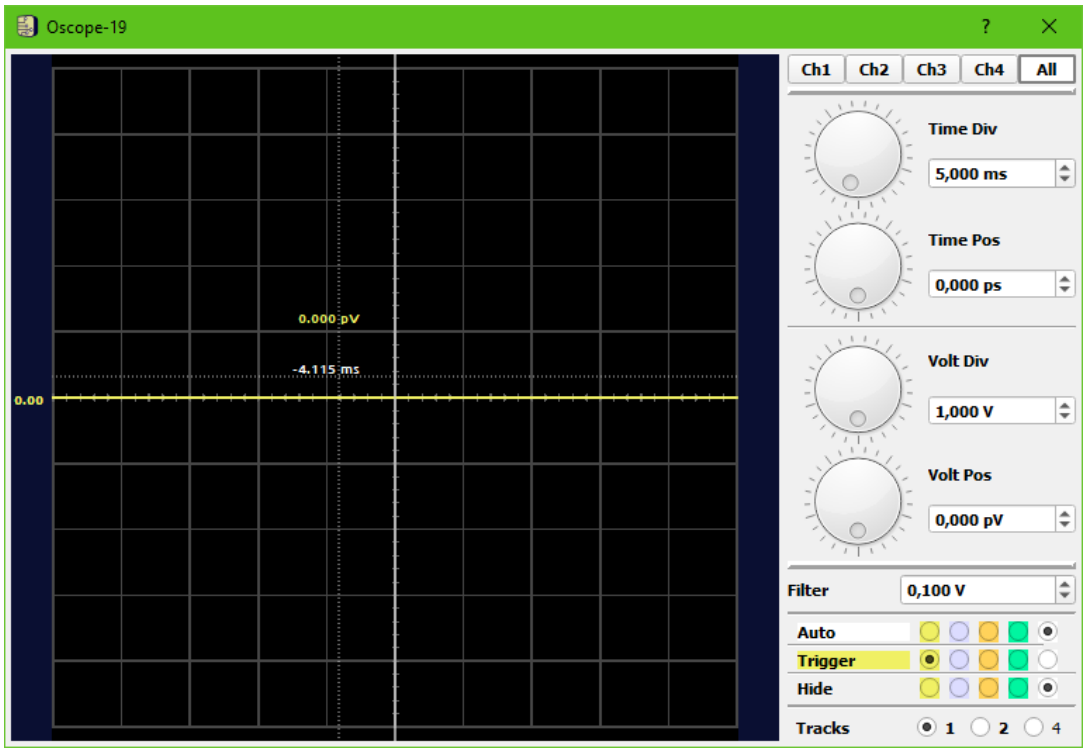


Рисунок 3.33 – Вікно осцилографа

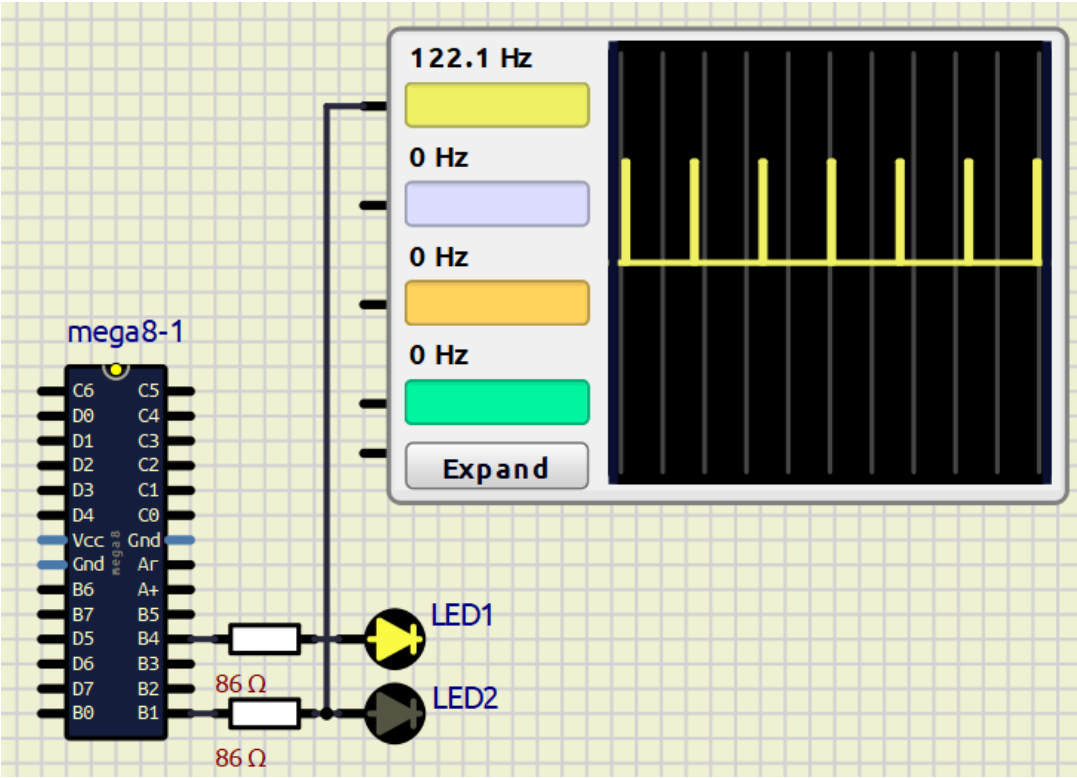


Рисунок 3.34 – Яскравість світлодіода LED2 при малій ширині імпульсів

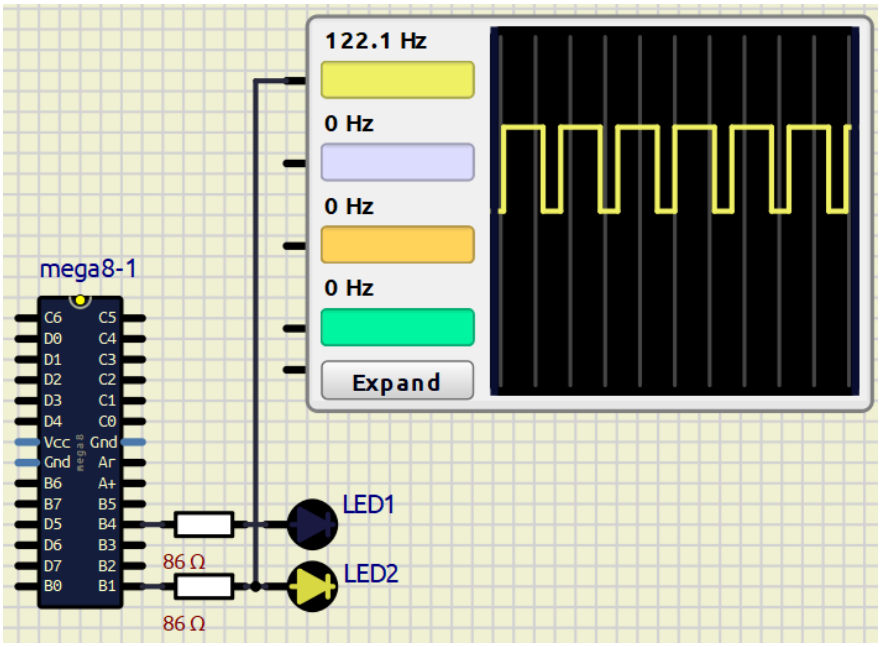


Рисунок 3.35 – Яскравість світлодіода LED2 при великій ширині імпульсів

3.4 Завдання до практичної роботи

1. Створити принципову електричну схему, згідно з варіантом завдань (таблиця 3.10).
2. Написати програму, що виконує поставлене завдання.
3. Створити файл прошивки, загрузити його у мікроконтролер та переконатися, що все працює, як треба.

Таблиця 3.10 – Варіанти завдань до практичної роботи №3

Варіант	Завдання
1	Підключити кнопку до виводу PD6, а світлодіод до виводу PB2 (OC1B). Встановити тактову частоту таймера 1 як $f_{CLK\_I/O}$, та режим як швидкий ШІМ з верхнім значенням, що задається регістром ICR1. Встановити частоту повного циклу таймера 1, як 256 Гц. Яскравість світлодіода повинна керуватися виводом OC1B модуля порівняння за допомогою ШІМ. При запуску програми встановити яскравість рівною 0. При кожному натисканні на кнопку збільшувати яскравість на 20% від максимальної величини. При досяганні максимального значення при наступному натисканні кнопки встановити яскравість рівною 0.
2	Підключити світлодіод LED1 до виводу PB1 (OC1A), а світлодіод LED2 до виводу PB2 (OC1B). Встановити тактову частоту таймера 1 як $f_{CLK\_I/O}/8$, та режим як швидкий ШІМ (8 біт). Яскравість світлодіодів повинна керуватися виводами OC1A та OC1B модуля порівняння за допомогою ШІМ. При запуску програми встановити яскравість LED1 рівною 0, а яскравість LED2 максимальною. При кожному переповненні таймера 1 змінювати яскравість світлодіодів у протифазі. При збільшенні яскравості світлодіода LED1, яскравість світлодіода LED2 зменшувати. При досягненні світлодіодом LED1 максимальної яскравості, її стрибком зменшити до 0, а яскравість світлодіода LED2 встановити максимальною.
3	Підключити світлодіод LED1 до виводу PB2 (OC1B). Встановити тактову частоту таймера 1 як $f_{CLK\_I/O}$, та режим як швидкий ШІМ з верхнім значенням, що задається регістром OCR1A. Встановити частоту повного циклу таймера 1, як 128 Гц. Яскравість світлодіоду повинна керуватися виводом OC1B модуля порівняння за допомогою ШІМ. При запуску програми встановити яскравість LED1 рівною 0. При кожному переповненні таймера 1 змінювати яскравість світлодіода, спочатку збільшуючи її на 1, а при досяганні максимального значення – зменшуючи на 1 до досягання 0, потім знову збільшуючи і т.д.
4	Підключити світлодіод LED1 до виводу PB2 (OC1B), а кнопку до виводу PD2. Встановити тактову частоту таймера 1 як $f_{CLK\_I/O}/64$, та режим як швидкий ШІМ з верхнім значенням, що задається регістром OCR1A. Встановити частоту повного циклу таймера 1, як 32 Гц. Яскравість світлодіоду повинна керуватися виводом OC1B модуля порівняння за допомогою ШІМ. При запуску програми встановити яскравість LED1 рівною 0.

Варіант	Завдання
	Натискання на кнопку обробляти за допомогою переривання INT0. При натисканні на кнопку, копіювати значення лічильника таймера 1 в регістр OCR1B, тим самим випадковим чином змінюючи яскравість світлодіода.
5	Підключити світлодіод LED1 до виводу PB1 (OC1A), а кнопку до виводу PD3. Встановити тактову частоту таймера 1 як $f_{CLK\_I/O}$, та режим як швидкий ШІМ з верхнім значенням, що задається регістром ICR1. Встановити частоту повного циклу таймера 1, як 100 Гц. Яскравість світлодіоду повинна керуватися виводом OC1A модуля порівняння за допомогою ШІМ. При запуску програми встановити яскравість LED1 максимальною. Натискання на кнопку обробляти за допомогою переривання INT1. При натисканні на кнопку, записувати в регістр OCR1A випадкове число в діапазоні від 0 до ICR1, тим самим випадковим чином змінюючи яскравість світлодіода.
6	Підключити світлодіод LED1 до виводу PB1 (OC1A). Встановити тактову частоту таймера 1 як $f_{CLK\_I/O}$, та режим як фазо-коректний ШІМ (8 біт). Встановити тактову частоту таймера 0 як $f_{CLK\_I/O}/256$. Яскравість світлодіоду повинна керуватися виводом OC1A модуля порівняння за допомогою ШІМ. При запуску програми встановити яскравість LED1 рівною половині максимальної. При кожному переповненні таймера 0 змінювати яскравість світлодіода, спочатку збільшуючи її на 1, а при досяганні максимального значення – зменшуючи на 1 до досягання 0, потім знову збільшуючи і т.д.
7	Підключити світлодіод LED1 до виводу PB2 (OC1B), а кнопку до виводу PD0. Встановити тактову частоту таймера 1 як $f_{CLK\_I/O}/64$, та режим як ОТП з верхнім значенням, що задається регістром OCR1A. Встановити частоту повного циклу таймера 1, як 1 Гц. Стан світлодіоду повинен керуватися виводом OC1B модуля порівняння, перемикаючись при кожному збігу при порівнянні. При натисканні на кнопку змінювати частоту повного циклу таймера у такій послідовності: 1 Гц – 2 Гц – 4 Гц – 8 Гц – 1 Гц…
8	Підключити світлодіод LED1 до виводу PB2 (OC1B). Встановити тактову частоту таймера 1 як $f_{CLK\_I/O}$, та режим як фазо-коректний ШІМ з верхнім значенням, що задається регістром OCR1A. Встановити частоту повного циклу таймера 1, як 300 Гц. Встановити тактову частоту таймера 0 як $f_{CLK\_I/O}/1024$. Яскравість світлодіоду повинна керуватися виводом OC1B модуля порівняння за допомогою ШІМ. При запуску програми встановити яскравість LED1 рівною максимальній. При кожному переповненні таймера 0 копіювати значення лічильника таймера 1 в регістр OCR1B, тим самим випадковим чином змінюючи яскравість світлодіода.
9	Підключити світлодіод LED1 до виводу PB1 (OC1A), світлодіод LED2 до виводу PB2 (OC1B), а кнопку до виводу PD2. Встановити тактову частоту таймера 1 як $f_{CLK\_I/O}$, та режим як фазо-коректний ШІМ (10 біт). Яскравість світлодіодів повинна керуватися виводами OC1A та OC1B модуля порівняння за допомогою ШІМ. При запуску програми встановити

Варіант	Завдання
	яскравість LED1 рівною 20%, а яскравість LED2 рівною 80% від максимальної. Натискання на кнопку обробляти за допомогою переривання INT0. При кожному натисканні на кнопку міняти місцями яскравості світлодіодів.
10	Підключити світлодіод LED1 до виводу PB1 (OC1A), світлодіод LED2 до виводу PB2 (OC1B). Встановити тактову частоту таймера 1 як $f_{CLK\_I/O}/8$, та режим як фазо-коректний ШІМ (9 біт). Встановити тактову частоту таймера 0 як $f_{CLK\_I/O}/1024$. Яскравість світлодіодів повинна керуватися виводами OC1A та OC1B модуля порівняння за допомогою ШІМ. При запуску програми встановити яскравість LED1 рівною 30%, а яскравість LED2 рівною 70% від максимальної. При кожному переповненні таймера 0 міняти місцями яскравості світлодіодів.
11	Підключити світлодіод LED1 до виводу PB1 (OC1A), а кнопку до виводу PD5. Встановити тактову частоту таймера 1 як $f_{CLK\_I/O}/64$, та режим як ОТП з верхнім значенням, що задається регістром ICR1. Встановити частоту повного циклу таймера 1, як 2 Гц. Встановити тактову частоту таймера 0 як $f_{CLK\_I/O}/1024$. Стан світлодіоду повинен керуватися виводом OC1A модуля порівняння, перемикаючись при кожному збігу при порівнянні. При кожному десятому переповненню таймера 0 змінювати частоту повного циклу таймера 1 у такій послідовності: 2 Гц – 3 Гц – 4 Гц – 5 Гц – 1 Гц…
12	Підключити світлодіод LED1 до виводу PB2 (OC1A), а кнопку до виводу PD2. Встановити тактову частоту таймера 1 як $f_{CLK\_I/O}/8$, та режим як швидкий ШІМ (8 біт). Встановити тактову частоту таймера 0 як $f_{CLK\_I/O}$. Яскравість світлодіоду повинна керуватися виводом OC1A модуля порівняння за допомогою ШІМ. При запуску програми встановити яскравість LED1 рівною половині максимальної. Натискання на кнопку обробляти за допомогою переривання INT1. При натисканні на кнопку, копіювати значення лічильника таймера 0 в регістр OCR1A, тим самим випадковим чином змінюючи яскравість світлодіода.

3.5 Питання для самоперевірки

1. Принцип роботи таймера 0 і таймера 1 мікроконтролера ATMega8.
2. Як налаштувати тактову частоту таймера?
3. Які є режими роботи модуля порівняння таймера 1?
4. Які переривання можуть генерувати таймери мікроконтролера ATMega8?

3.6 Перелік рекомендованих джерел

1. ATMega8 : технічна документація на мікроконтролер. URL:: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2486-8-bit-AVR-microcontroller-ATmega8_L_datasheet.pdf (дата звернення: 02.07.2024).
2. 8-bit AVR® MCUs : інформація про мікроконтролери. URL: https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/8-bit-mcus/avr-mcus (дата звернення: 02.07.2024).
3. Конспект лекцій з дисципліни «Мікропроцесорна техніка» для здобувачів вищої освіти першого (бакалаврського) рівня зі спеціальності 153 «Мікро-та наносистемна техніка» за освітньо-професійною програмою «Мікро-та наносистемна техніка» та зі спеціальності 171 «Електроніка» за освітньо-професійною програмою «Електроніка» / уклад. О. М. Гулєша. Кам‘янське : ДДТУ, 2020 р. 57 с.
4. Основи Програмування AVR C. DevZone. URL: https://devzone.org.ua/post/osnovi-programuvannia-avr-c (дата звернення: 02.07.2024).

4 ПРАКТИЧНА РОБОТА №4 «ПРОГРАМУВАННЯ ЦИФРОВИХ ІНТЕРФЕЙСІВ МІКРОКОНТРОЛЕРА AVR»

4.1 Завдання

Освоєння прийомів програмування цифрових інтерфейсів мікроконтролерів AVR.

Завдання практичної роботи:

– Створення електричної схеми у програмі SimulIDE відповідно до завдання на практичну роботу.

– Написання програми для мікроконтролеру відповідно до завдання на практичну роботу.

4.2 Теоретичні дані

4.2.1 Загальні відомості про інтерфейс UART[6]

Універсальний синхронний/асинхронний послідовний прийомопередавач (USART) забезпечує обмін даними МК AVR з зовнішніми пристроями по послідовному каналу в повнодуплексному режимі. При цьому передача даних може бути як асинхронна так і синхронна.

При **синхронному** послідовному вводі/виводі передача окремих бітів даних синхронізується за допомогою тактового сигналу, який передається одночасно з даними. Синхронна послідовна передача даних використовується, основним чином, на рівні друкованих плат, у тому числі, для обміну даними між різними інтегрованими блоками у складі схеми МК та різними периферійними схемами.

При **асинхронній** передачі даних синхронізація виконується у часі за допомогою стартових та стопових бітів, що визначають початок та кінець передачі слова даних. Асинхронна передача даних використовується для комунікації блоків, розділених у просторі та які мають певну автономність один від одного. Наприклад, між ПК та принтером, між пристроєм на базі МК та комп’ютером.

Модуль USART підтримує як синхронний, так і асинхронний режими роботи. Однак на практиці його найчастіше використовують саме в асинхронному режимі, а синхронний режим реалізують за допомогою модуля SPI. Тому в курсі лекцій ми розглядатимемо лише асинхронний режим, і надалі модуль називатимемо UART.

Формат передачі кадру даних UART. За своєю структурою він ідентичний інтерфейсу RS-232, з тією лиш відмінністю, що в інтерфейсі

[6] Даний підрозділ викладений на основі матеріалів [1]

RS-232 логічні рівні формуються напругами від ±3 до ±12В, а в модулі UART логічні рівні відповідають TTL-рівням (0 та 5 В).

Початок кадру даних завжди фіксується низьким рівнем стартового біту (рисунок 4.1). Після цього йде байт даних (5-9 бітів) з молодшими розрядами спереду. Якщо дозволена перевірка на парність, то далі йде біт парності, що доповнює байт даних «1» чи «0» так, щоб кількість «1» байту даних була парною (при опції «Парність») чи непарною (при опції «Непарність»). Цей простий засіб дає можливість виявляти непарну кількість спотворених бітів. Останніми передаються стопові (1 чи 2) біти, що представлені високим рівнем. Якщо на лінії передача даних відсутня, тоді на ній завжди присутній високий рівень. Швидкість передачі вимірюється у бодах (baud), бітів за секунду (bps), і на рисунку 4.1 кадр даних складається з 12 бодів.

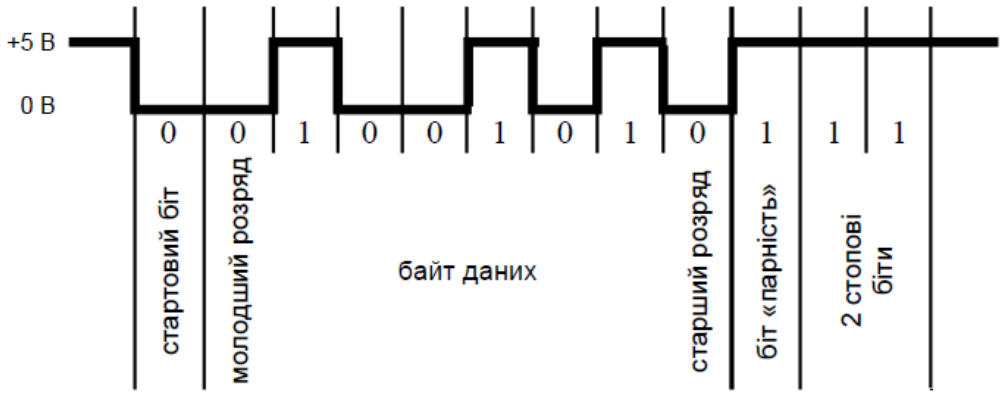


Рисунок 4.1 – Діаграма швидкості передачі

Формат кадру задається відповідними бітами регістрів керування UCSRB та UCSRC (див. рис. 4.2).

Визначення розміру байту даних

	Кількість біт у байті даних				
	5 біт	6 біт	7 біт	8 біт	9 біт
UCSZ0	0	1	0	1	1
UCSZ1	0	0	1	1	1
UCSZ2	0	0	0	0	1

Керування контролем парності

	немає	парність	непарність
UPM0	0	0	1
UPM1	0	1	1

Рисунок 4.2 – Визначення розміру байту даних та керування контролем парності

Вибір кількості стоп-бітів здійснюється за допомогою розряду USBS у регістрі керування UCSRC. Якщо цей розряд скинутий в «0», тоді передавач формує 1 стоп-біт у кінці посилки. Якщо ж встановлений в «1», тоді – 2 стоп-біти. Варто зазначити, що приймачем другий стоп-біт ігнорується, і відповідно, помилки кадрування виявляються лише для першого стоп-біта. Найбільш популярним є формат кадру 8n1 (1 старт, 8 біт даних, 1 стоп) без контролю парності.

Підключення UART. У МК AVR протокол передачі даних UART реалізований апаратно. На деяких моделях навіть є реалізовано декілька модулів UART. Приймач даних під'єднаний до виводу з надписом RxD, а передавач до виводу TxD. При з'єднанні між собою двох модулів UART для передачі даних, необхідно з'єднати навхрест між собою виводи модулів передачі та приймання, як на рисунку 4.3.

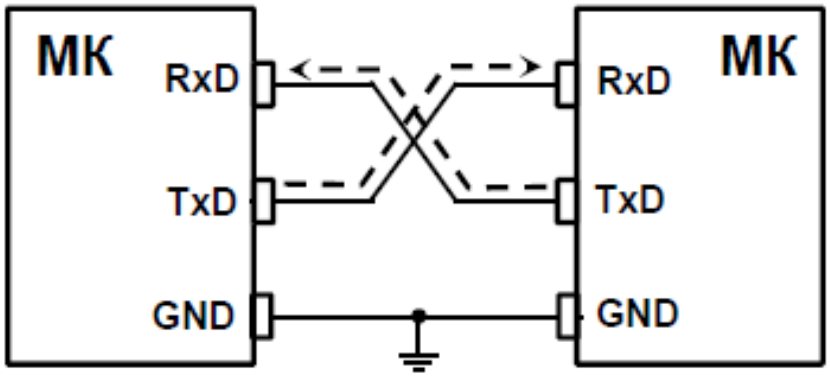


Рисунок 4.3 – З'єднання двох модулів UART для передачі даних

4.2.2 Апаратна частина UART мікроконтролерів AVR[7]

Модуль складається з 3-х основних частин (рисунок 4.4): тактового генератора (контролера) швидкості передачі, блоку приймача та блоку передавача.

Блок передавача містить однорівневий буфер, зсувний регістр, схему формування біта парності та схему керування. Блок приймача містить схеми відновлення тактового сигналу та даних, схему контролю парності, дворівневий буфер, зсувний регістр та схему керування.

Буферні регістри приймача та передавача розміщуються за єдиним адресом простору вводу/виводу та позначаються як регістр даних UDR. У цьому регістрі зберігаються молодші 8 розрядів даних, що приймаються чи передаються. При читанні UDR виконується звертання до буферного регістра приймача, а при записі – до буферного регістра передавача.

У модулях USART буфер приймача дворівневий (FIFO-буфер). При будь-якому звертанні до регістра UDR цей буфер змінює свій стан. Тому

[7] Даний підрозділ викладений на основі матеріалів [1]

необхідно спершу зчитати дані з цього регістра, а потім вже виконувати необхідні маніпуляції над ним.

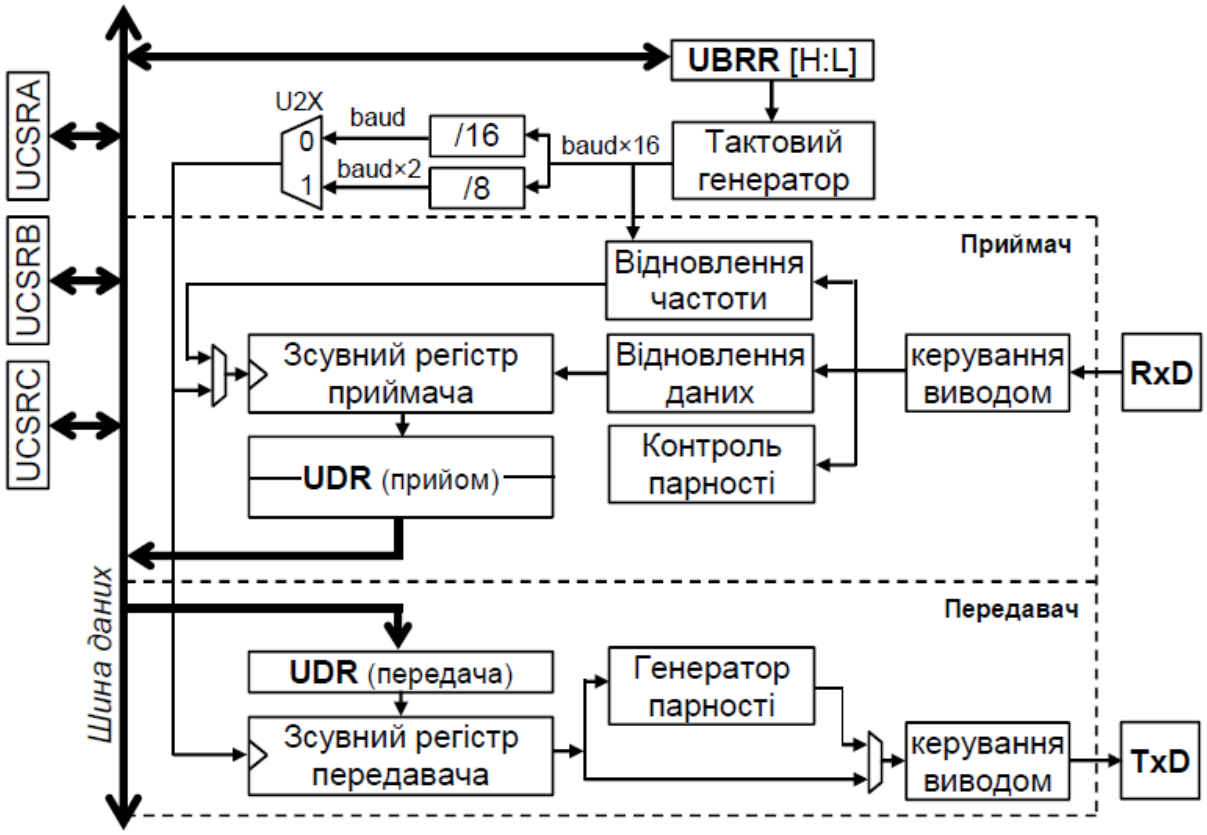


Рисунок 4.4 – Схема модуля UART

Регістр контролера швидкості UBRR задає необхідний коефіцієнт поділу для системного тактового сигналу, після чого цей сигнал ще поступає на додаткові дільники, вибір яких здійснюється за допомогою додаткового біта U2X.

Схема відновлення тактового сигналу (у приймачі) призначена для синхронізації внутрішнього тактового сигналу, що формується контролером швидкості передачі, та пакетів з даними, що поступають на вивід RxD. Схема відновлення даних виконує зчитування та фільтрацію кожного розряду для отримуваного пакету.

Швидкість прийому/передачі. Швидкість обміну задається контролером швидкості передачі, що функціонує як подільник системного тактового сигналу з програмованим коефіцієнтом поділу, значення якого знаходиться у регістрі UBRR. Регістр UBRR є 12-розрядним та фізично розміщується у 2-х регістрах UBRRH та UBRRL.

В асинхронному режимі швидкість обміну визначається не лише значенням регістра UBRR, але і станом розряду U2X у регістрі керування UCSRA. Якщо цей біт встановлений в «1», то коефіцієнт поділу подільника

зменшується у 2 рази, а швидкість, відповідно, подвоюється. Швидкість обміну в асинхронному режимі визначається за такими формулами:

$$\text{при U2X} = 0: \mathbf{BAUD} = \frac{\text{XTAL}}{16(\text{UBBR}+1)};\ \mathbf{UBRR} = \frac{\text{XTAL}}{16\cdot\text{BAUD}} - 1 \qquad (4.1)$$

$$\text{при U2X} = 1: \mathbf{BAUD} = \frac{\text{XTAL}}{8(\text{UBBR}+1)};\ \mathbf{UBRR} = \frac{\text{XTAL}}{8\cdot\text{BAUD}} - 1 \qquad (4.2)$$

Прийняті такі стандартні швидкості обміну даними: 1200, 1800, 2400, 4800, 7200, 9600, 14400, 19200, 28800, 38400, 57600, 76800, 115200, 230400 бод.

Для уникнення виникнення помилок передачі рекомендується використовувати стабілізований кварцовий тактовий генератор. Також має значення і величина частоти, на якій працює кварцовий кристал. На деяких частотах можна отримати нульову похибку при передачі даних відносно ряду стандартних швидкостей. Похибка передачі обчислюється за такою формулою:

$$\text{Error}[\%] = \left(\frac{\text{BAUD}_{\text{розрах.}}}{\text{BAUD}} - 1\right)\cdot 100\% \qquad (4.3)$$

Розрахуємо похибку для стандартної швидкості 9600 бод при частоті тактового генератора 8МГц.

$$\text{UBRR} = \frac{8\cdot 10^6}{16\cdot 9600} - 1 = 51.083 = 51\ ;$$

$$\text{BAUD}_{\text{розрах.}} = \frac{8\cdot 10^6}{16(51+1)} = 9615{,}38\ \text{Бод};$$

$$\text{Error}[\%] = \left(\frac{9615.38}{9600} - 1\right)\cdot 100\% = 0.16\%.$$

Рекомендується використовувати значення регістра UBRR, при яких отримана швидкість передачі відрізняється від необхідного значення менше, аніж на 0,5%.

Передача та прийом даних, переривання модуля UART. Для активації прийому/передачі модуля UART необхідно надати дозволи на роботу передавача та приймача, встановивши відповідні біти TXEN та RXEN у регістрі керування UCSRB. Тоді відповідні виводи МК, позначені як TxD та RxD, підключаються до модуля UART та працюють на прийом і передачу, незалежно від налаштувань регістрів керування портом, до якого вони належать.

Для відправки байту даних необхідно записати його значення у регістр даних UDR. Після цього ці дані пересилаються із UDR у зсувний

регістр передавача. Якщо в регістр UDR відправити одразу ще один байт даних, то ці дані будуть відправлені у зсувний регістр лише після того, як у зсувному регістрі буде відправлений останній біт з кадру. Отже, частота запису даних в UDR визначається швидкістю обміну даними модуля UART.

Прийом даних починається з моменту виявлення приймачем коректного старт-біту. Далі, кожен наступний біт кадру зчитується зі швидкістю, заданою для модуля UART, та розміщується у зсувному регістрі, аж поки не буде виявлений перший стоп-біт. Після цього вміст зсувного регістра пересилається у буфер приймача UDR, звідки прийняте значення має бути зчитаним.

Якщо формат кадру передбачає 9 біт даних, тоді перед записом в регістр UDR молодших 8 біт необхідно виставити у потрібне значення біт TXB8 (регістр UCSRB). Аналогічно і при прийомі даних, спершу необхідно прочитати значення біту RXB8 (регістр UCSRB), а потім вже читати значення молодших 8-ми бітів у регістрі UDR.

При прийомі даних також можемо виконати перевірку прапорів помилок (регістр UCSRA), які мають бути перевіреними ще перед читанням регістру даних UDR:

UPE – прапор помилки контролю парності, який виставляється при виявлені помилки парності у прийнятих даних.

DOR – прапор переповнення, який виставляється при виявленні нового старт-біта у зсувному регістрі, а буфер приймача у цей момент є заповнений (2 значення).

FE – прапор помилки кадрування, який виставляється при виявленні у прийнятому кадрі «0» на місці першого стоп-біта.

Для сповіщення про події: прийнято новий байт даних, завершено передачу даних, регістр даних UDR порожній - передбачені відповідні прапори RXC, TXC, UDRE (регістр UCSRA).

На основі цих прапорів також можуть бути згенеровані переривання для обробки цих подій. Дозвіл на переривання визначаються відповідними прапорами дозволів (регістр UCSRB):

RXCIE – дозвіл на переривання по завершенню прийому;

TXCIE – дозвіл на переривання по завершенню передачі;

UDRIE – дозвіл на переривання при спорожненні регістра UDR.

Переривання по завершенню передачі даних використовуються лише в окремих випадках. Наприклад, для переключення кінцевого пристрою у режим прийому по завершенню передачі даних в протоколі передачі даних RS-485.

4.2.3 Регістри вводу-виводу модуля USART

Регістр даних USART – UDR (рис. 4.5)

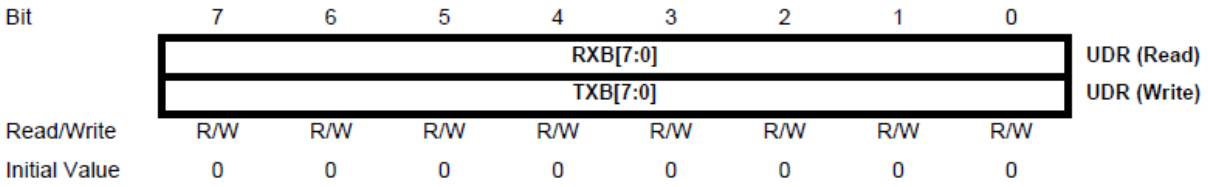


Рисунок 4.5 – Регістр даних USART – UDR

Регістр буфера передачі даних USART і регістр буфера прийому даних USART спільно використовують ту саму адресу вводу/виводу, що називається регістром даних USART або UDR. Регістр буфера передачі даних (TXB) буде місцем призначення для даних, записаних до регістру UDR. Зчитування регістру UDR повертає вміст регістру буфера прийому даних (RXB).

Для 5-бітних, 6-бітових або 7-бітових символів передавач ігноруватиме верхні невикористані біти, а приймач встановлюватиме їх як нуль.

Буфер передачі може бути записаний лише тоді, коли встановлено прапор UDRE у регістрі UCSRA. Дані, записані в UDR, коли прапор UDRE не встановлено, передавач USART ігноруватиме. Коли дані записані в буфер передачі, і передавач увімкнено, він завантажуватиме дані в регістр зсуву передачі, коли той порожній. Потім дані будуть послідовно передаватися на вивід TxD.

Буфер прийому складається з дворівневого FIFO. FIFO змінює свій стан кожного разу, коли здійснюється доступ до буфера прийому. Через таку поведінку буфера прийому не використовуйте інструкції читання-зміни-запису (SBI та CBI) для цього регістру. Будьте також обережні, використовуючи інструкції перевірки бітів (SBIC і SBIS), оскільки вони також змінять стан FIFO.

Регістр контролю та статусу USART A – UCSRA (рис. 4.6)

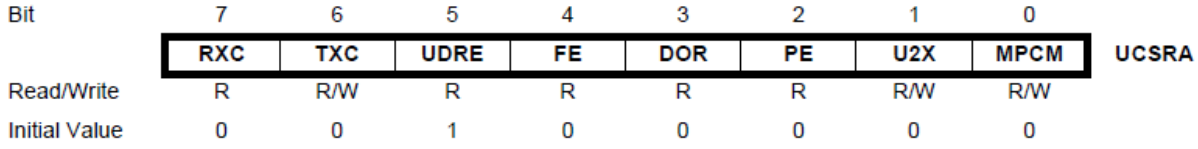


Рисунок 4.6 – Регістр контролю та статусу USART A – UCSRA

• **Біт 7 – RXC: прийом USART завершено.** Цей прапор встановлюється, коли в буфері прийому є непрочитані дані, і очищається,

коли буфер прийому порожній (тобто не містить непрочитаних даних). Якщо приймач вимкнено, буфер прийому буде очищено, і, отже, біт RXC стане нульовим. Прапор RXC можна використовувати для генерації переривання по завершенню прийому даних.

• **Біт 6 – TXC: передача USART завершена.** Цей прапор встановлюється, коли весь кадр у регістрі зсуву передачі було передано, а в буфері передачі (UDR) наразі немає нових даних. Біт прапора TXC автоматично очищається, коли виконується переривання по завершенню передачі, або його можна скинути, записавши в нього одиницю. Прапор TXC може генерувати переривання по завершенню передачі даних.

• **Біт 5 – UDRE: Регістр даних USART порожній.** Прапор UDRE вказує, чи буфер передачі (UDR) готовий приймати нові дані. Якщо UDRE дорівнює 1, буфер порожній і, отже, готовий до запису. Прапор UDRE може генерувати переривання порожнього регістру даних. UDRE дорівнює 1 після скидання, щоб вказати, що передавач готовий.

• **Біт 4 – FE: Помилка кадру.** Цей біт встановлюється, якщо наступний символ у приймальному буфері мав помилку кадру під час отримання (тобто, коли перший стоп-біт наступного символу в приймальному буфері дорівнював нулю). Цей біт дійсний, доки не буде зчитано буфер прийому (UDR). Біт FE дорівнює нулю, коли стоп-біт отриманих даних дорівнює одиниці. Завжди встановлюйте цей біт як нуль під час запису в UCSRA.

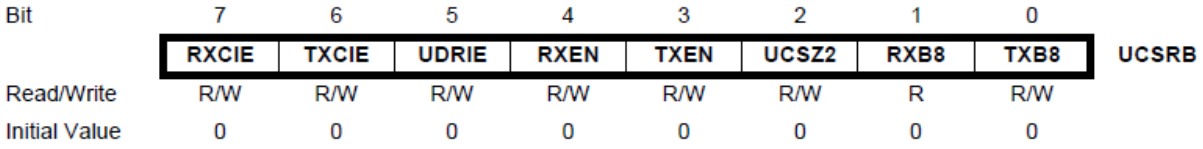
• **Біт 3 – DOR: Переповнення даних.** Цей біт встановлюється, якщо виявлено стан переповнення даних. Це відбувається, коли буфер прийому заповнений (два символи), присутній новий символ, який очікує в регістрі зсуву прийому, і виявлено новий початковий біт. Цей біт дійсний, доки не буде зчитано буфер прийому (UDR). Завжди встановлюйте цей біт як нуль під час запису в UCSRA.

• **Біт 2 – PE: помилка парності.** Цей біт встановлюється, якщо наступний символ у буфері прийому мав помилку парності під час отримання, та перевірка парності була ввімкнена в цей момент (UPM1 = 1). Цей біт дійсний, доки не буде зчитано буфер прийому (UDR). Завжди встановлюйте цей біт як нуль під час запису в UCSRA.

• **Біт 1 – U2X: подвоєна швидкість передачі USART.** Цей біт працює лише для асинхронного режиму. Встановіть цей біт як нуль при використанні синхронного режиму. Запис одиниці у цей біт зменшить значення дільника швидкості передачі з 16 до 8, фактично подвоюючи швидкість передачі для асинхронного зв’язку.

• **Біт 0 – MPCM: багатопроцесорний режим зв’язку.** Цей біт вмикає багатопроцесорний режим зв’язку. Коли біт MPCM записаний як 1, усі вхідні кадри, отримані приймачем USART, які не містять інформації про адресу, ігноруватимуться. Налаштування MPCM не впливає на передавач.

Регістр контролю та статусу USART B – UCSRB (рис. 4.7).



Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 4.7 – Регістр контролю та статусу USART B – UCSRB

• **Біт 7 – RXCIE: дозвіл переривання по завершенню прийому даних.** Запис одиниці в цей дозволяє переривання при встановленні прапору RXC. Переривання по завершенню прийому даних буде створено, лише якщо біт RXCIE встановлений як 1, глобальний прапор переривання в SREG встановлений як 1 і біт RXC в регістрі UCSRA встановлено.

• **Біт 6 – TXCIE: дозвіл переривання по завершенню передачі даних.** Запис одиниці у цей біт дозволяє переривання при встановленні прапору TXC. Переривання по завершенню передачі даних буде створено, лише якщо біт TXCIE встановлений як 1, глобальний прапор переривання в SREG встановлений як 1 і біт TXC в регістрі UCSRA встановлено.

• **Біт 5 – UDRIE: дозволено переривання порожнього регістру даних USART.** Запис цього біта в одиницю дозволяє переривання при встановленні прапору UDRE. Переривання порожнього регістру даних буде згенеровано, лише якщо біт UDRIE встановлений як 1, глобальний прапор переривання в SREG встановлений як 1 і біт UDRE в регістрі UCSRA встановлено.

• **Біт 4 – RXEN: увімкнення приймача.** Запис одиниці в цей біт вмикає приймач USART. Приймач замінює звичайний режим порту для виводу RxD, якщо він ввімкнений. Вимкнення приймача очистить буфер прийому, зробивши прапори FE, DOR і PE недійсними.

• **Біт 3 – TXEN: увімкнення передавача.** Запис одиниці в цей біт вмикає передавач USART. Передавач замінює звичайний режим порту для виводу TxD, якщо він ввімкнений. Вимкнення передавача (записування 0 у TXEN) не набуде чинності, доки поточні та незавершені передачі не будуть завершені (тобто, коли регістр зсуву передачі та регістр буфера передачі не будуть містити даних для передачі). Якщо передавач вимкнений, він більше не займає вивід TxD.

• **Біт 2 – UCSZ2: довжина символу.** Біт UCSZ2 у поєднанні з бітами UCSZ1:0 в UCSRC встановлюють кількість бітів даних (довжину символу) у кадрі, який використовують приймач і передавач.

• **Біт 1 – RXB8: Біт 8 прийнятих даних.** RXB8 є дев’ятим бітом даних отриманого символу під час роботи з послідовними кадрами з дев’ятьма бітами даних. Його необхідно прочитати перед читанням молодших бітів з UDR.

• **Біт 0 – TXB8: Біт 8 даних для передачі.** TXB8 є дев'ятим бітом даних у символі, що передається під час роботи з послідовними кадрами з дев'ятьма бітами даних. Його потрібно записати перед записом молодших бітів до UDR.

Регістр контролю та статусу USART C – UCSRC (рис. 4.8).

Bit	7	6	5	4	3	2	1	0	
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	0	0	0	0	1	1	0	

Рисунок 4.8 – Регістр контролю та статусу USART C – UCSRC

• **Біт 7 – URSEL: вибір регістра.** Регістр UCSRC має ту саму адресу, що й регістр UBRRH. Тому під час доступу до нього необхідно звернути особливу увагу.

Під час запису у цю адресу старший біт записаного значення (біт вибору реєстру USART (URSEL)) контролює, який із двох регістрів буде записано. Якщо URSEL дорівнює нулю під час операції запису, буде оновлено значення UBRRH. Якщо URSEL дорівнює одиниці, буде оновлено регістр UCSRC.

Здійснення читання регістру UBRRH чи UCSRC є більш складною операцією. Однак у більшості програм рідко потрібно читати будь-який із цих регістрів. Доступ для читання контролюється певною послідовністю. Одноразове зчитування з цієї адреси повертає вміст регістру UBRRH. Якщо значення регістра було прочитано в попередньому системному такті, повторне читання регістру в поточному такті поверне вміст UCSRC. Зауважте, що часова послідовність для читання UCSRC є неподільною операцією. Таким чином, переривання повинні контролюватися (наприклад, шляхом глобального вимкнення переривань) під час операції читання. Читання вмісту UBRRH не є неподільною операцією, тому його можна читати як звичайний регістр, якщо попередня інструкція не зчитувала значення з цього регістру.

• **Біт 6 – UMSEL: вибір режиму USART.** Цей біт вибирає між асинхронним (коли він дорівнює 0) і синхронним (коли він дорівнює 1) режимами роботи.

• **Біти 5:4 – UPM1:0: режим парності.** Ці біти вмикають і задають тип генерації та перевірки парності. Якщо перевірку парності ввімкнено, передавач автоматично генеруватиме та надсилатиме біт парності переданих бітів даних у кожному кадрі. Приймач генеруватиме значення біту парності для вхідних даних і порівнюватиме його з налаштуванням UPM0. Якщо буде виявлено невідповідність, буде встановлено прапор PE в UCSRA.

Таблиця 4.1 – Режими контролю парності у відповідності до налаштувань бітів

UPM1	UPM0	Режим контролю парності
0	0	Вимкнено
0	1	Зарезервовано
1	0	Ввімкнено, контроль парності
1	1	Ввімкнено, контроль непарності

• **Біт 3 – USBS: вибір стопових бітів.** Цей біт вибирає кількість стоп-бітів, які вставляє передавач. Одержувач ігнорує це налаштування. Якщо цей біт дорівнює 0, то передається один стоп-біт, а якщо дорівнює 1, то передається два стоп-біти.

• **Біти 2:1 – UCSZ1:0: довжина символу.** Біти UCSZ1:0 у поєднанні з бітом UCSZ2 в регістрі UCSRB встановлюють кількість бітів даних (довжина символу) у кадрі, який використовують приймач і передавач (табл. 4.2).

• **Біт 0 – UCPOL: полярність тактового сигналу.** Цей біт використовується лише для синхронного режиму. Записуйте цей біт як 0, коли використовуєте асинхронний режим. Біт UCPOL встановлює співвідношення між зміною вихідних даних і вибіркою вхідних даних, а також синхронним тактовим сигналом (XCK).

Таблиця 4.2 – Довжина символи в залежності від налаштувань

UCSZ2	UCSZ1	UCSZ0	Довжина символу
0	0	0	5 біт
0	0	1	6 біт
0	1	0	7 біт
0	1	1	8 біт
1	0	0	Зарезервовано
1	0	1	Зарезервовано
1	1	0	Зарезервовано
1	1	1	9 біт

Регістри швидкості передачі даних USART – UBRRL і UBRRH (рис. 4.9).

Bit	15	14	13	12	11	10	9	8	
	URSEL	–	–	–	UBRR[11:8]				UBRRH
	UBRR[7:0]								UBRRL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Рисунок 4.9 – Регістри швидкості передачі даних USART – UBRRL і UBRRH

• **Біт 15 – URSEL: вибір регістра.** Цей біт вибирає між доступом до UBRRH або UCSRC. Читається як нуль при читанні UBRRH. URSEL має дорівнювати нулю під час запису у UBRRH.

• **Біти 14:12 – зарезервовані біти.** Ці біти зарезервовано для використання в майбутньому. Для сумісності з майбутніми пристроями ці біти повинні бути записані як 0 під час запису UBRRH.

• **Біти 11:0 – UBRR11:0: Регістр швидкості передачі даних USART.** Це 12-бітний регістр, який містить швидкість передачі USART. UBRRH містить чотири старші біти, а UBRRL містить вісім молодших бітів швидкості передачі USART. Поточні передачі передавача та приймача будуть пошкоджені, при зміні швидкості передачі даних. Запис UBRRL ініціює негайне оновлення попереднього дільника швидкості передачі.

Для стандартних частот кварцових резонаторів найбільш часто використовувані швидкості передачі даних для асинхронної роботи можна створити за допомогою налаштувань UBRR, що показані у розділі 4.2.4. Значення UBRR, які дають фактичну швидкість передачі даних, що відрізняється від цільової швидкості передачі даних менш ніж на 0,5%, виділені жирним шрифтом у таблиці. Вищі значення помилок прийнятні, але приймач матиме меншу завадостійкість, коли значення помилок високі, особливо для великих довжин символів.

4.2.4 Приклади налаштування швидкості передачі даних

На рис. 4.10-4.13 представлені розраховані значення регістрів UBRRH та UBRRL для різних швидкостей передачі UART та для різних тактових частот мікропроцесора за формулами (4.1) та (4.2).

Значення регістрів UBRRH та UBRRL представлені для двох варіантів – коли біт U2X регістра UCSRA встановлено та очищено.

При використанні тієї чи іншої швидкості передачі треба спочатку впевнитися, що відхилення частоти (рядок Error) не перебільшує 0.5%. Такі значення на рис. 4.10 – 4.13 виділені жирним шрифтом.

Baud Rate (bps)	f_{osc} = 1.0000MHz				f_{osc} = 1.8432MHz				f_{osc} = 2.0000MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	25	0.2%	51	0.2%	47	0.0%	95	0.0%	51	0.2%	103	0.2%
4800	12	0.2%	25	0.2%	23	0.0%	47	0.0%	25	0.2%	51	0.2%
9600	6	-7.0%	12	0.2%	11	0.0%	23	0.0%	12	0.2%	25	0.2%
14.4k	3	8.5%	8	-3.5%	7	0.0%	15	0.0%	8	-3.5%	16	2.1%
19.2k	2	8.5%	6	-7.0%	5	0.0%	11	0.0%	6	-7.0%	12	0.2%
28.8k	1	8.5%	3	8.5%	3	0.0%	7	0.0%	3	8.5%	8	-3.5%
38.4k	1	-18.6%	2	8.5%	2	0.0%	5	0.0%	2	8.5%	6	-7.0%
57.6k	0	8.5%	1	8.5%	1	0.0%	3	0.0%	1	8.5%	3	8.5%
76.8k	–	–	1	-18.6%	1	-25.0%	2	0.0%	1	-18.6%	2	8.5%
115.2k	–	–	0	8.5%	0	0.0%	1	0.0%	0	8.5%	1	8.5%
230.4k	–	–	–	–	–	–	0	0.0%	–	-	–	–
250k	–	–	–	–	–	–	–	–	–	–	0	0.0%
Max (1)	62.5kbps		125kbps		115.2kbps		230.4kbps		125kbps		250kbps	

1. UBRR = 0, Error = 0.0%

Рисунок 4.10 – Налаштування регістрів UBRRH та UBRRL при тактових частотах процесора від 1 МГц до 2 МГц

Baud Rate (bps)	f_{osc} = 3.6864MHz				f_{osc} = 4.0000MHz				f_{osc} = 7.3728MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	95	0.0%	191	0.0%	103	0.2%	207	0.2%	191	0.0%	383	0.0%
4800	47	0.0%	95	0.0%	51	0.2%	103	0.2%	95	0.0%	191	0.0%
9600	23	0.0%	47	0.0%	25	0.2%	51	0.2%	47	0.0%	95	0.0%
14.4k	15	0.0%	31	0.0%	16	2.1%	34	-0.8%	31	0.0%	63	0.0%
19.2k	11	0.0%	23	0.0%	12	0.2%	25	0.2%	23	0.0%	47	0.0%
28.8k	7	0.0%	15	0.0%	8	-3.5%	16	2.1%	15	0.0%	31	0.0%
38.4k	5	0.0%	11	0.0%	6	-7.0%	12	0.2%	11	0.0%	23	0.0%
57.6k	3	0.0%	7	0.0%	3	8.5%	8	-3.5%	7	0.0%	15	0.0%
76.8k	2	0.0%	5	0.0%	2	8.5%	6	-7.0%	5	0.0%	11	0.0%
115.2k	1	0.0%	3	0.0%	1	8.5%	3	8.5%	3	0.0%	7	0.0%
230.4k	0	0.0%	1	0.0%	0	8.5%	1	8.5%	1	0.0%	3	0.0%
250k	0	-7.8%	1	-7.8%	0	0.0%	1	0.0%	1	-7.8%	3	-7.8%
0.5M	–	–	0	-7.8%	–	–	0	0.0%	0	-7.8%	1	-7.8%
1M	–	–	–	–	–	–	–	–	–	–	0	-7.8%
Max (1)	230.4kbps		460.8kbps		250kbps		0.5Mbps		460.8kbps		921.6kbps	

1. UBRR = 0, Error = 0.0%

Рисунок 4.11 – Налаштування регістрів UBRRH та UBRRL при тактових частотах процесора від 3,6864 МГц до 7,3728 МГц

Baud Rate (bps)	f_{osc} = 8.0000MHz				f_{osc} = 11.0592MHz				f_{osc} = 14.7456MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	207	0.2%	416	-0.1%	287	0.0%	575	0.0%	383	0.0%	767	0.0%
4800	103	0.2%	207	0.2%	143	0.0%	287	0.0%	191	0.0%	383	0.0%
9600	51	0.2%	103	0.2%	71	0.0%	143	0.0%	95	0.0%	191	0.0%
14.4k	34	-0.8%	68	0.6%	47	0.0%	95	0.0%	63	0.0%	127	0.0%
19.2k	25	0.2%	51	0.2%	35	0.0%	71	0.0%	47	0.0%	95	0.0%
28.8k	16	2.1%	34	-0.8%	23	0.0%	47	0.0%	31	0.0%	63	0.0%
38.4k	12	0.2%	25	0.2%	17	0.0%	35	0.0%	23	0.0%	47	0.0%
57.6k	8	-3.5%	16	2.1%	11	0.0%	23	0.0%	15	0.0%	31	0.0%
76.8k	6	-7.0%	12	0.2%	8	0.0%	17	0.0%	11	0.0%	23	0.0%
115.2k	3	8.5%	8	-3.5%	5	0.0%	11	0.0%	7	0.0%	15	0.0%
230.4k	1	8.5%	3	8.5%	2	0.0%	5	0.0%	3	0.0%	7	0.0%
250k	1	0.0%	3	0.0%	2	-7.8%	5	-7.8%	3	-7.8%	6	5.3%
0.5M	0	0.0%	1	0.0%	–	–	2	-7.8%	1	-7.8%	3	-7.8%
1M	–	–	0	0.0%	–	–	–	–	0	-7.8%	1	-7.8%
Max (1)	0.5Mbps		1Mbps		691.2kbps		1.3824Mbps		921.6kbps		1.8432Mbps	

1. UBRR = 0, Error = 0.0%

Рисунок 4.12 – Налаштування регістрів UBRRH та UBRRL при тактових частотах процесора від 8 МГц до 14,7456 МГц

Baud Rate (bps)	f_{osc} = 16.0000MHz				f_{osc} = 18.4320MHz				f_{osc} = 20.0000MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	416	-0.1%	832	0.0%	479	0.0%	959	0.0%	520	0.0%	1041	0.0%
4800	207	0.2%	416	-0.1%	239	0.0%	479	0.0%	259	0.2%	520	0.0%
9600	103	0.2%	207	0.2%	119	0.0%	239	0.0%	129	0.2%	259	0.2%
14.4k	68	0.6%	138	-0.1%	79	0.0%	159	0.0%	86	-0.2%	173	-0.2%
19.2k	51	0.2%	103	0.2%	59	0.0%	119	0.0%	64	0.2%	129	0.2%
28.8k	34	-0.8%	68	0.6%	39	0.0%	79	0.0%	42	0.9%	86	-0.2%
38.4k	25	0.2%	51	0.2%	29	0.0%	59	0.0%	32	-1.4%	64	0.2%
57.6k	16	2.1%	34	-0.8%	19	0.0%	39	0.0%	21	-1.4%	42	0.9%
76.8k	12	0.2%	25	0.2%	14	0.0%	29	0.0%	15	1.7%	32	-1.4%
115.2k	8	-3.5%	16	2.1%	9	0.0%	19	0.0%	10	-1.4%	21	-1.4%
230.4k	3	8.5%	8	-3.5%	4	0.0%	9	0.0%	4	8.5%	10	-1.4%
250k	3	0.0%	7	0.0%	4	-7.8%	8	2.4%	4	0.0%	9	0.0%
0.5M	1	0.0%	3	0.0%	–	–	4	-7.8%	–	–	4	0.0%
1M	0	0.0%	1	0.0%	–	–	–	–	–	–	–	–
Max (1)	1Mbps		2Mbps		1.152Mbps		2.304Mbps		1.25Mbps		2.5Mbps	

1. UBRR = 0, Error = 0.0%

Рисунок 4.13 – Налаштування регістрів UBRRH та UBRRL при тактових частотах процесора від 16 МГц до 20 МГц

4.3 Приклад програми

Виконаємо наступне завдання на базі мікроконтролера ATMega8. Підключити трирозрядний семисегментний індикатор зі спільним катодом наступним чином: A – PB5, B – PB4, C – PB3, D – PB2, E – PB1, F – PB0, G – PB7, перший розряд – PC1, другий розряд – PC2, третій розряд – PC3. Підключити дві кнопки до виводів PD2 (INT0) та PD3 (INT1). Налаштувати модуль USART для роботи в асинхронному режимі з такими параметрами: швидкість 9600 бод, кількість бітів даних – 8, кількість стопових бітів – 1, перевірка парності відсутня.

Натискання кнопок обробляти за допомогою відповідних переривань.

При натисканні на кнопку, підключену до виводу PD2, збільшувати значення деякої змінної на 1.

При натисканні на кнопку, підключену до виводу PD3, зменшувати значення тієї самої змінної на 1.

При будь-якій зміні змінної відправляти її значення через UART у вигляді текстового рядку з символами повернення каретки (‘\r’) та переводу рядку (‘\n’) наприкінці.

При прийомі через UART текстового рядка, перетворювати його на число та відображати на семисегментному індикаторі. Символом закінчення передачі рядка вважати повернення каретки. Якщо передане число більше 999, відображати 999 замість нього.

4.3.1 Принципова електрична схема

Принципова електрична схема, що реалізує поставлену задачу, показана на рис. 4.14.

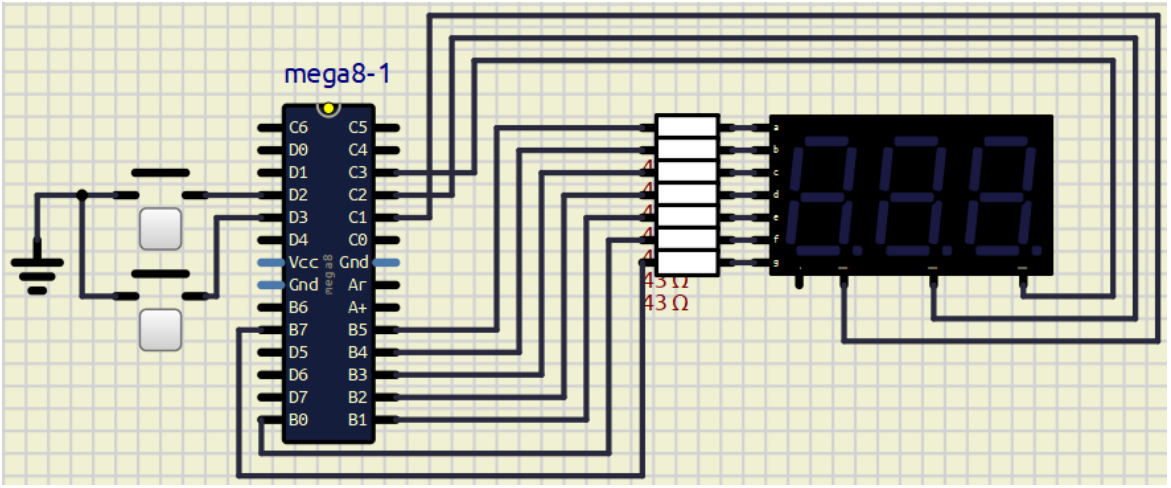


Рисунок 4.14 – Принципова електрична схема

У схемі є новий елемент, який ще не траплявся раніше – це семисегментний світлодіодний індикатор.

Всі бачили такі індикатори у різних побутових приладах – пральних машинах, мультварках, кухонних таймерах та ін.

Принцип їх дії заснований на відображенні цифр за допомогою лише 7 світлодіодів (тому вони і називаються семисегментними), що розташовані у спеціальній формі, показаній на рис. 4.15.

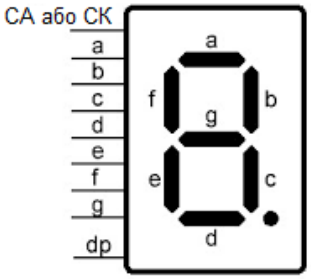


Рисунок 4.15 – Семисегментний індикатор

Сегменти уявляють собою пласкі світлодіоди і мають назви у вигляді латинських літер від A до G. Окремо стоїть сегмент, що відображає десяткову крапку, це також світлодіод, але круглої форми, який позначається як DP (decimal point).

Світлодіодні індикатори можуть мати або спільний анод (СА), або спільний катод (СК), відповідно до того, який вивід є спільним у всіх світлодіодів, що утворюють один розряд (рис. 4.16).

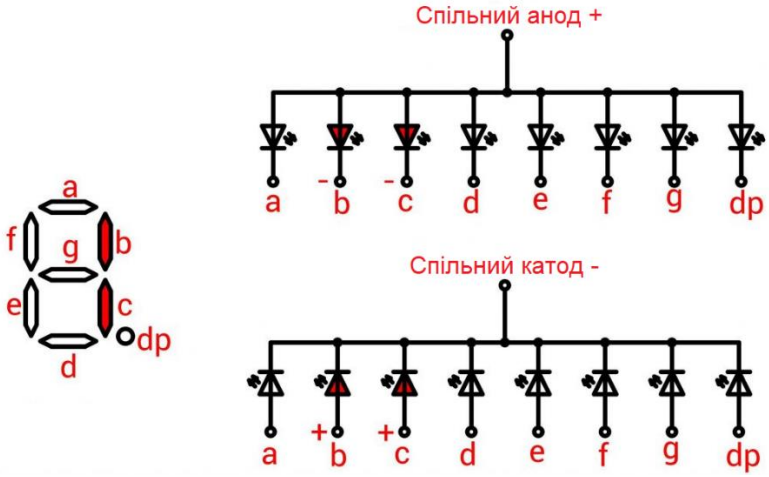


Рисунок 4.16 – Структура семисегментного індикатора

Якщо використовуються багаторазрядні індикатори (в нашому випадку – трирозрядний), то всі однойменні виводи сегментів об’єднані між собою (рис. 4.17). Це зроблено для того, щоб зменшити кількість виводів, необхідних для підключення індикатора. Але при такому підключенні не можна одночасно виводити значення на всі розряди, бо

тоді зображення просто будуть накладатися одне на одне. У такому випадку використовують так звану динамічну індикацію, при якій розряди переключаються по черзі з великою частотою, що непомітна оку.

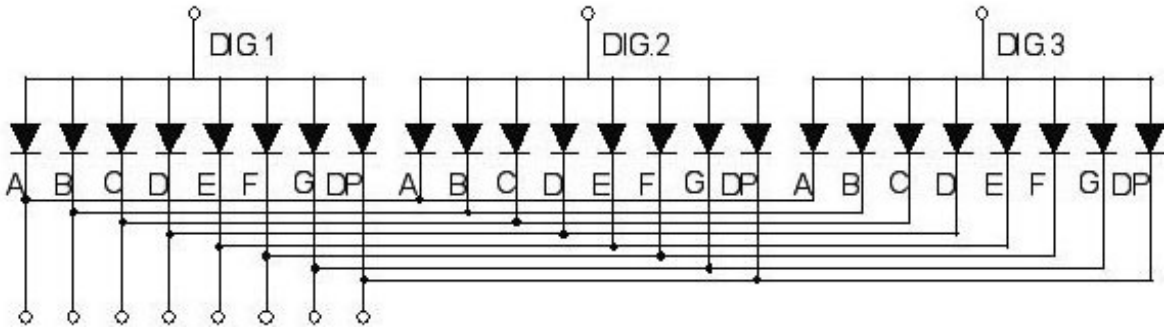


Рисунок 4.17 – Принципова електрична схема трирозрядного семисегментного індикатора зі спільним анодом

У програмі SimulIDE семисегментний індикатор знаходиться у підзаголовку «Outputs» - «Leds», і називається «7 Segment» (рис. 4.18)

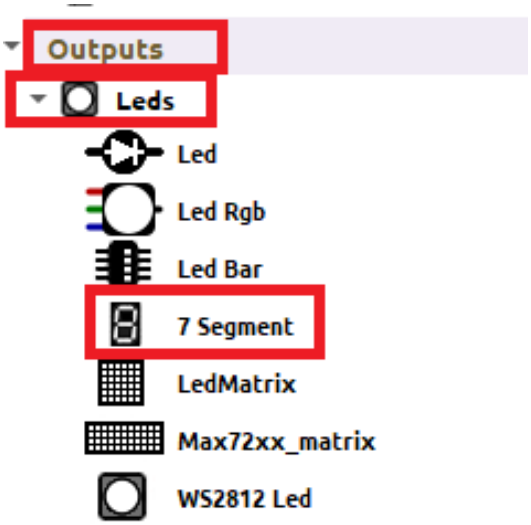


Рисунок 4.18 – Розташування семисегментного індикатора у бібліотеці компонентів

Він має досить багато налаштувань (рис. 4.19).

Більшість параметрів у нього такі самі, як і у світлодіода (що недивно): колір (Color), пряме падіння напруги (Forward Voltage), максимальний прямий струм (Max Current) та власний опір (Resistance), то ж ми не будемо на них тут зупинятися.

Параметр «Size» - це кількість розрядів індикатора. Може бути від 1 до теоретично безкінечності. Нам, відповідно до завдання треба встановити три розряди.

Параметр «Vertical Pins» переносить виводи сегментів з лівої частини у верхню і нижню частини. Це зроблено просто для зручності створення схеми у деяких випадках.

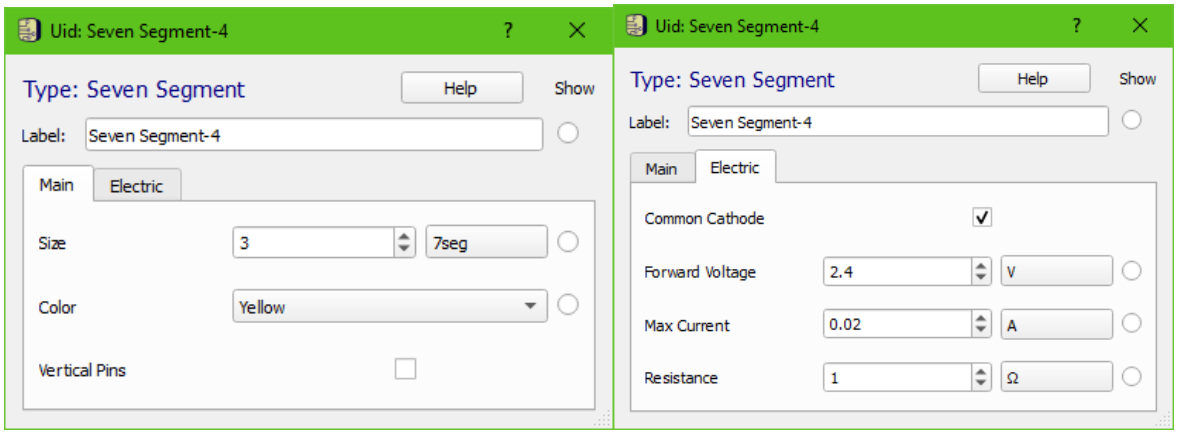


Рисунок 4.19 – Налаштування семисегментного індикатора

Параметр «Common Cathode» задає, який вивід у індикатора буде спільним. Якщо галочка встановлена, то спільним буде катод, а якщо ні, то анод. Відповідно до завдання, нам потрібно використовувати індикатор зі спільним катодом, то ж ми ставимо цю галочку.

Струмообмежуючі резистори повинні підключатися до кожного сегменту. Їх величина розраховується таким самим чином, як і для звичайних світлодіодів. Але тут треба враховувати той факт, що кожен розряд буде світитися не весь час, а лише частину, що дорівнює одиниці, поділеній на кількість розрядів.

То ж у нашому випадку:

R = ((5 В – 2.4 В) / 0.02 А) / 3 – 1 Ом = 43 Ом.

Чим більша кількість розрядів, тим менший повинен бути опір, щоб забезпечити необхідну яскравість індикації.

У реальному пристрої вивід RxD інтерфейсу UART суміщений з виводом PD0, а вивід TxD – з виводом PD1, то ж для роботи з цим інтерфейсом ми повинні були б підключити їх до передавача та приймача сигналу, відповідно. Але у програмі SimulIDE є вбудований термінал для роботи з UART, який підключений як би всередині мікроконтролера, то ж ніяких додаткових під’єднань робити не треба.

Все інше на схемі вже зустрічалося нам раніше, то ж ми не будемо на ній зупинятися далі, а перейдемо до розглядання програмного коду.

4.3.2 Програмний код

Розглянемо тепер програму, що реалізує поставлену задачу (рис. 4.20).

```c
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define A _BV(PB5)
#define B _BV(PB4)
#define C _BV(PB3)
#define D _BV(PB2)
#define E _BV(PB1)
#define F _BV(PB0)
#define G _BV(PB7)
#define D1 _BV(PC1)
#define D2 _BV(PC2)
#define D3 _BV(PC3)

uint8_t digit = 1;
int16_t number_in = 0;
int16_t number_out = 0;
char rx_buf[9];
uint8_t rx_count = 0;
uint8_t rx_complete = 0;
uint8_t tx_count;
uint8_t tx_len;
char tx_buf[9];
const uint8_t gen[10] =
{
  A | B | C | D | E | F, //0
  B | C, //1
  A | B | G | D | E, //2
  A | B | C | D | G, //3
  B | C | G | F, //4
  A | F | G | C | D, //5
  A | G | C | D | E | F, //6
  A | B | C, //7
  A | B | C | D | E | F | G, //8
  A | B | C | D | G | F, //9
};

void send_number (uint16_t num)
{
  sprintf(tx_buf, "%d\r\n", num);
  tx_len = strlen(tx_buf);
  tx_count = 0;
  UCSRB |= _BV(UDRIE);
}

void parse_data (void)
{
  number_in = atoi(rx_buf);
  rx_count = 0;
  for (uint8_t i = 0; i < sizeof(rx_buf); i ++)
    rx_buf[i] = 0;
  if (number_in > 999)
    number_in = 999;
}

ISR(INT0_vect)
{
  number_out ++;
  send_number(number_out);
}

ISR(INT1_vect)
{
  number_out --;
  send_number(number_out);
}

ISR (USART_RXC_vect)
{
  rx_buf[rx_count] = UDR;
  if (rx_buf[rx_count] == '\r')
  {
    rx_complete = 1;
  }
  else
    rx_count ++;
}

ISR (USART_UDRE_vect)
{
  if (tx_count < tx_len)
  {
    UDR = tx_buf[tx_count];
    tx_count ++;
  }
  else
  {
    UCSRB &= ~_BV(UDRIE);
  }
}

int main (void)
{
  DDRB |= A | B | C | D | E | F | G;
  DDRC |= D1 | D2 | D3;
  DDRD &= ~(_BV(PD2) | _BV(PD3));
  PORTD |= _BV(PD2) | _BV(PD3);
  PORTB = 0;

  UCSRA = _BV(U2X);
  UCSRB = _BV(RXCIE) | _BV(RXEN) | _BV(TXEN);
  UCSRC = _BV(URSEL) | _BV(UCSZ1) | _BV(UCSZ0);
  UBRRH = 0;
  UBRRL = 12;

  MCUCR |= _BV(ISC01) | _BV(ISC00) | _BV(ISC11) | _BV(ISC10);
  GICR |= _BV(INT0) | _BV(INT1);
  GIFR |= _BV(INTF0) | _BV(INTF1);
  sei();

  while(1)
  {
    PORTC |= D1 | D2 | D3;
    switch (digit)
    {
      case 1:
        PORTB = gen[number_in / 100];
        PORTC &= ~D1;
        break;
      case 2:
        PORTB = gen[(number_in % 100) / 10];
        PORTC &= ~D2;
        break;
      case 3:
        PORTB = gen[number_in % 10];
        PORTC &= ~D3;
        break;
    }
    _delay_ms(5);
    digit ++;
    if (digit > 3)
      digit = 1;
    if (rx_complete)
    {
      rx_complete = 0;
      parse_data();
    }
  }
}
```

Рисунок 4.20 – Програмний код

У рядках 1-6 ми підключаємо необхідні заголовкові файли. Файли «io.h» та «delay.h» ми вже використовували у минулих програмах.

У рядку 3 підключається заголовковий файл «interrupt.h», який потрібно додавати, якщо в програмі планується використовувати переривання. Оскільки ми тут будемо мати справу з зовнішніми перериваннями INT0 та INT1, то цей файл необхідно додати.

Зверніть увагу, що у рядках 4-6 ми підключаємо стандартні заголовкові файли компілятора С, які досить широко використовуються і для звичайного програмування: «stdio.h», «stdlib.h» та «string.h». Це значить, що компілятор AVR GCC підтримує і деякі стандартні функції мови С, що є загальноприйнятими. Про кожну з них буде написано окремо, коли вони зустрінуться у тексті програми.

У рядках 8-17 ми визначаємо макровизначення, або просто макроси за допомогою директиви препроцесора define. Це зроблено просто для зручності подальшого написання і читання коду і не є обов'язковим.

То ж, ми визначаємо макроси для кожного сегменту та розряду індикатора, як назву виводу, до якого він підключений (рис. 4.14). Наприклад, розряд A підключений до виводу PB5, тому ми визначаємо макрос з назвою «A», який тотожно дорівнює _BV(PB5), тобто номеру біта, що відповідає виводу PB5. Таким самим чином ми визначаємо макроси для всіх інших сегментів (B-G), а також для розрядів індикатора (D1-D3).

У рядку 19 ми визначаємо змінну digit і одразу ініціалізуємо її значенням 1. Ця змінна буде відповідати номеру розряду семисегментного індикатора, який в даний момент світиться.

У рядку 20 ми визначаємо змінну number_in, яка уявляє собою число, що потрібно приймати через інтерфейс UART і виводити на семисегментний індикатор.

У рядку 21 ми визначаємо змінну number_out, яка уявляє собою число, що потрібно змінювати за допомогою кнопок та виводити на через інтерфейс UART.

У рядку 22 визначається масив rx_buf, який буде використовуватися для того, щоб зберігати строку, що надходить через інтерфейс UART.

У рядку 23 визначається змінна rx_count, що уявляє собою лічильник прийнятих символів.

У рядку 24 визначається змінна rx_complete, що є прапором, який індикує про те, що коректна строка прийнята.

У рядку 25 визначається змінна tx_count, що уявляє собою лічильник переданих символів.

У рядку 26 визначається змінна tx_len, яка містить довжину строки, яку потрібно передати.

У рядку 27 визначається масив tx_buf, що буде використовуватися для зберігання строки, яку потрібно передати через інтерфейс UART.

У рядках 28-40 знаходиться знакогенератор для семисегментного індикатора. Це масив з десяти констант, кожна з яких містить число, яке

потрібно записати у PORTB, щоб відобразити ту чи іншу цифру. Номер елемента масиву відповідає цифрі, що повинна відображатись.

Наприклад, нульовий елемент масиву записаний, як «A | B | C | D | E | F». Макроси A-F в нас вже визначені раніше, і вони відповідають тим бітам в регістри PORTB, які треба встановити, щоб засвітити відповідний сегмент. Для того, щоб відобразити цифру 0, треба засвітити сегменти A, B, C, D, E та F (див. рис. 2), тому ми їх і записуємо сюди. Всі біти, що не вказані тут, будуть встановлені, як 0, а відповідні сегменти не будуть світитися. Аналогічним чином визначаються сегменти для всіх інших цифр. Тут треба зазначити, що все, описане тут, стосується індикатора зі спільним катодом. Для індикатора зі спільним анодом всі елементи масиву повинні бути інвертованими, оскільки для ввімкнення світлодіода сегменту на його вивід треба подати низький рівень, а на спільний анод – високий. То ж, для індикатора зі спільним анодом елементи масиву будуть такими:

```
const uint8_t gen[10] =
{
        ~(A | B | C | D | E | F), //0
        ~(B | C), //1
        …
        …
}
```

Рядки 42-95 поки пропустимо, і перейдемо до головної функції програми, що розташована у рядках 96-143.

Спочатку треба налаштувати всі необхідні порти вводу/виводу. Виводи, до яких підключені і аноди, і катоди індикаторів, треба налаштувати як виходи. У рядку 98 ми записуємо одиниці у біти, які відповідають сегментам світлодіодного індикатора, у регістр DDRB. А у рядку 99 ми записуємо одиниці у біти, що відповідають розрядам світлодіодного індикатора, у регістр DDRC. Зверніть увагу, що тут так само використовуються описані раніше макроси.

У рядку 100 ми скидаємо біти PD2 та PD3 у регістрі DDRD, налаштовуючи відповідні виводи, до яких підключені кнопки, як входи. А у рядку 101 ми вмикаємо підтягувальні резистори на цих виводах, встановлюючи ті ж самі біти у регістрі PORTD.

У рядку 102 ми записуємо 0 у регістр PORTB, тим самим вимикаючи одразу всі сегменти індикатора (для індикатора зі спільним анодом треба записати в усі біти цього регістру одиниці).

У рядках 104-108 ми конфігуруємо модуль USART. Інформація щодо регістрів цього модуля приведена у пункті 4.2.3.

Оскільки ми хочемо встановити швидкість передачі як 9600 бод при частоті процесора 1 МГц, нам треба встановити біт U2X у регістрі UCSRA,

оскільки в іншому випадку ми не зможемо досягти потрібної похибки (див. рис. 4.10). При скинутому біті U2X та значенні регістру UBRR = 6 похибка складає -7%, що є неприпустимим. А при встановленому біти U2X та UBRR = 12 похибка зменшується до 0,2%, що вкладається у допустимі межі ±0,5%. То ж ми встановлюємо біт U2X у регістрі UCSRA (рядок 104), залишаючи всі інші біти як 0, оскільки більшість із них уявляє собою прапори переривань або помилок і доступні лише для читання.

У регістрі UCSRB ми встановлюємо наступні біти (рядок 105): RXCIE, щоб дозволити переривання при прийомі символу, RXEN та TEXEN, щоб дозволити роботу прийому і передачі, відповідно. Після встановлення двох останніх біт виводи PD0 та PD1 будуть керуватися модулем USART, і їхній стан не буде залежати від значень регістрів DDRD та PORTD. Оскільки за завданням кількість байтів даних у пакеті дорівнює 8, ми залишаємо біт UCSZ2 рівним 0. Також ми залишаємо рівним нулю біт UDRIE, який дозволяє переривання при спорожненні регістру UDR. Це зроблено через те, що прапор UDRE завжди дорівнює 1, якщо регістр UDR порожній, а це відбувається більшу кількість часу. При цьому постійно буде генеруватися відповідно переривання, заважаючи нормальній роботі програми. То ж цей біт будемо встановлювати лише тоді, коли ми збираємося передавати якісь дані.

У регістрі UCSRC ми встановлюємо такі біти (рядок 106): URSEL, тому що його потрібно встановлювати завжди, коли ми хочемо щось записати у регістр UCSRC, UCSZ1 та UCSZ0, щоб задати довжину символу 8 біт (див. табл. 4.2). Біт UMSEL залишаємо рівним 0, щоб модуль працював у асинхронному режимі. Біти UPM1 та UPM0 залишаємо рівними 0, щоб вимкнути режим контролю парності. Біт USBS також залишається рівним 0, щоб встановити один стоповий біт. Значення біту UCPOL не має значення у синхронному режимі, то ж ми залишимо його рівним 0 також.

У рядках 107, 108 ми встановлюємо старші 4 біти регістру UBRR рівними 0, а молодші 8 біт рівними 12, відповідно до таблиці, показаній на рис. 4.10, щоб отримати швидкість передачі даних 9600 бод.

У рядках 110-113 налаштовуються переривання. Спочатку треба сконфігурувати фронти на виводах INT0 та INT1, по яким будуть генеруватися переривання. Ми будемо генерувати переривання в момент відпускання кнопки, при якому відбувається зміна рівня на вході від 0 до 1, тобто наростаючий фронт. Біти, що керують цією логікою, знаходяться у регістрі MCUCR (табл. 3.1). Відповідно до цієї таблиці, для того, щоб встановити наростаючий фронт як джерело запиту на переривання, треба встановити обидва біти ISCn0 та ISCn1, що ми і робимо у рядку 110.

У рядку 111 ми демаскуємо (дозволяємо) зовнішні переривання INT0 та INT1, встановлюючи біти INT0 та INT1 у регістрі GICR її. Також ми очищуємо прапори відповідних переривань у рядку 112, записуючи одиниці у біти INTF0 та INTF0 регістру GIFR. Це зроблено для того, щоб

не генерувати переривання, що можуть виникнути при конфігурації портів вводу/виводу.

Тепер лишилося тільки дозволити глобальні переривання за допомогою функції sei (рядок 113).

Тепер переходимо до основного циклу програми, що знаходиться у рядках 115-142. У ньому спочатку реалізований алгоритм динамічної індикації (рядки 117-136), який детально описаний в індивідуальному завданні №1, то ж ми не будемо тут його розглядати ще раз. Рядки 137-141 будуть пояснені пізніше, після розглядання функцій обробки переривань і функцій обробки даних.

Всі функції обробки переривань, як вже було зазначено у попередніх практичних роботах, мають назву ISR, а в якості їх аргументу передається назва переривання, яку можна знайти у заголовковому файлі для кожного конкретного мікроконтролера або у таблиці 3.1, замінивши пробіли на знак підкреслення, і додавши в кінці «_vect».

У рядках 60-64 знаходиться функція обробки зовнішнього переривання INT0. Відповідно до завдання, «При натисканні на кнопку, підключену до виводу PD2, збільшувати значення деякої змінної на 1.». Це ми і робимо у рядку 62. Окрім того, ще потрібно «При будь-якій зміні змінної відправляти її значення через UART у вигляді строки з символами повернення каретки (‘\r’) та переводу строки (‘\n’) наприкінці». Це відбувається у функції send_number, що викликається у рядку 63, і яку ми розглянемо дуже скоро.

У рядках 66-70 описана аналогічна підпрограма обробки переривання, тільки на цей раз вона викликається при генерації зовнішнього переривання INT1 (рядок 66). Дія при настанні цього переривання протилежна попередній – значення змінної number_out зменшується на 1 (рядок 68). І, як і в попередній функції, тут також відбувається викликається функція send_number (рядок 69).

Розглянемо цю функцію більш детально. Вона знаходиться у рядках 42-48 і приймає один аргумент – число num, що потрібно передати через UART. У рядку 44 викликається стандартна функція sprint, яка описана у файлі «stdio.h», що ми підключили у рядку 4. Ця функція дозволяє сформувати символьну строку з аргументів різного типу. В нашому випадку ми формуємо строку tx_buf, в яку ми записуємо спочатку число num, як звичайне ціле число (параметр «%d»), а потім додаємо символи повернення каретки (‘\r’) та переводу рядка (‘\n’) для того, щоб після передачі цього рядку новий текстовий рядок починався з нового рядку у вікні терміналу.

Далі ми знаходимо довжину рядка tx_buf, використовуючи іншу стандартну функцію strlen (рядок 45). Ця функція описана у заголовковому файлі «string.h», що підключений у рядку 5. Потім ми обнулюємо лічильник переданих символів tx_count (рядок 46), і нарешті дозволяємо переривання по спорожненню регістру даних UDR, встановлюючи біт

UDRIE у регістрі UCSRB (рядок 47). Оскільки ми ще нічого не записували у регістр UDR, прапор UDRE у регістрі UCSRA встановлений, тому одразу буде згенеровано переривання порожнього регістру даних. В обробнику цього переривання ми й будемо власне передавати данні.

Цей обробник знаходиться у рядках 183-194. Відповідний вектор цього переривання називається USART_UDRE_vect (рядок 183). Всередині цього обробника ми спочатку перевіряємо, чи лічильник переданих символів менше, ніж довжина рядку, що ми хочемо передати (рядок 85), тобто, чи є ще дані, які нам потрібно передати. Якщо є, то ми завантажуємо поточний елемент масиву tx_buf у регістр UDR (рядок 87) і збільшуємо лічильник переданих символів tx_count (рядок 88). Після завантаження даних у регістр UDR, вони передаються у регістр зсуву, якщо він вже порожній, з якого ці дані видаються на вивід TxD. Якщо ж регістр зсуву ще містить попередні дані, регістр UDR залишається повним, біт UDRE не встановлюється, і нове переривання не генерується. Як тільки цей регістр спорожніє, одразу встановиться прапор UDRE, згенерується відповідне переривання, і ми зможемо завантажити новий символ. Таким чином, ми не витрачаємо процесорний час, опитуючи значення біту UDRE, щоб дізнатися, коли можна буде завантажити наступний символ, все це робиться автоматично за допомогою переривання.

Коли всі символи вже передані (рядок 90), треба вимкнути це переривання, скинувши біт UDRIE (рядок 92), щоб не відбувалося постійного викликання цієї підпрограми, коли нам нема чого передавати.

Тепер розглянемо, як відбувається прийом даних. Для цього використовується ще один обробник переривання, тільки цього разу, він спрацьовує при прийомі символу у регістр даних (рядки 72-81). Відповідний вектор переривання називається USART_RXC_vect (рядок 72). Всередині цього обробника ми першим чином копіюємо прийняті дані з регістру UDR у масив rx_buf (рядок 74). Після цього ми перевіряємо, чи не дорівнює прийнятий символ значенню ‘\r’ (повернення каретки) (рядок 75). Цей символ є останнім у прийнятому пакеті даних, відповідно до завдання. Тому, коли ми його прийняли, ми припиняємо прийом наступних даних і встановлюємо прапор rx_complete (рядок 77), що індикує про те, що дані прийняті, і можна тепер їх обробляти. Якщо ж ми прийняли якийсь інший символ (рядок 79), ми просто збільшуємо лічильник прийнятих символів (рядок 80) і чекаємо на нові дані.

Обробник цього переривання повинен бути якомога коротшим, щоб не втратити дані, бо якщо новий символ прийде, поки відбуватиметься обробка попереднього, то він просто буде втрачений. Саме через це ми не виводимо нове число на індикатор всередині цієї підпрограми, а просто встановлюємо прапор rx_complete, який перевіряється в основному циклі програми (рядок 137). Коли цей прапор встановлено, його першим чином потрібно скинути (рядок 138), щоб коли прийде наступний пакет даних, він

би знову встановив цей прапор, і його також було б оброблено. А далі викликається функція parse_data (рядок 149), у якій власне і оновлюється значення, що показується на семисегментному індикаторі.

Ця функція знаходиться у рядках 50-58. В ній вхідний масив символів rx_buf перетворюється на ціле число number_in за допомогою стандартної функції atoi, що описана у файлі «stdlib.h», який ми підключили у рядку 6. Після цього нам потрібно підготувати масив rx_buf до прийому нових даних. То ж ми обнулюємо лічильник прийнятих символів rx_count (рядок 53), а також обнулюємо всі елементи самого масиву (рядки 54-55). Ну і наостанок, відповідно до завдання, якщо прийняте число більше 999 (рядок 56), ми встановлюємо його як 999 (рядок 57).

На цьому опис програми можна вважати завершеним. Тепер треба перевірити, як вона працює.

4.3.3 Симуляція роботи програми

Після компіляції програми та завантаження її у мікроконтролер треба ввімкнути монітор послідовного порту. Для того, щоб це зробити, треба натиснути правою кнопкою миші на мікроконтролері і з випадаючого списку обрати пункт «Open Serial Monitor» і далі «USART1» (рис. 4.21).

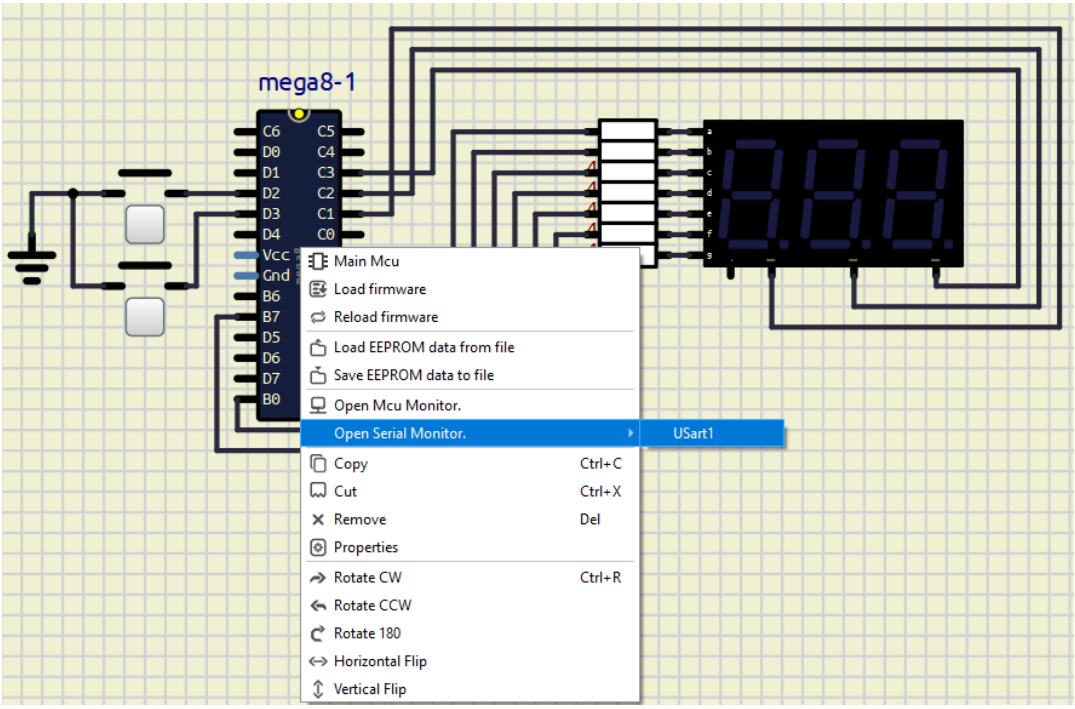


Рисунок 4.21 – Відкриття монітору послідовного порту

Після цього відкриється вікно, показане на рис. 4.22. Розберемо його основні елементи більш детально:

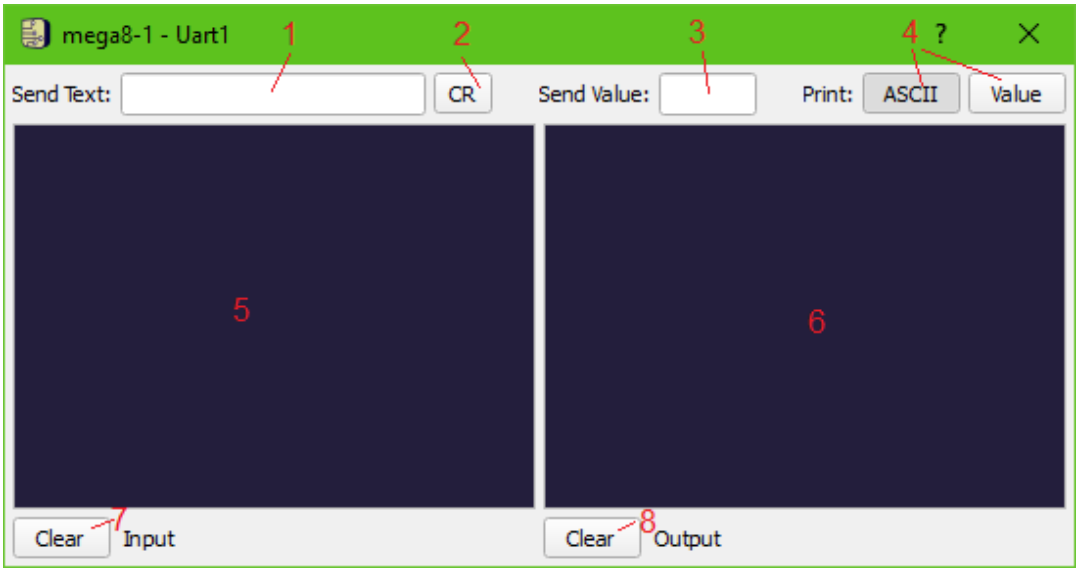


Рисунок 4.22 – Вікно монітору послідовного порту

1 – Поле для вводу строки, яка буде відправлена у мікроконтролер. Для того, щоб надіслати строку, треба набрати її на клавіатурі та натиснути кнопку Enter. Після цього передана строка з’явиться у полі вхідних даних 5.

2 – Кнопка, що дозволяє додати символ повернення каретки ‘\r’ в кінці строки. Ми її як раз і будемо використовувати при відправці числа у мікроконтролер як ознаку закінчення пакету даних.

3 – Відправка одного символу в ASCII-коді.

4 – Вигляд поля вхідних і вихідних даних: або у вигляді звичайної строки (варіант ASCII), або у вигляді набору чисел, що відповідають ASCII кодам переданих символів (варіант Value).

5 – Поле вхідних даних. Тут будуть відображатися дані, що надходять у мікроконтролер.

6 – Поле вихідних даних. Тут відображаються дані, що передаються мікроконтролером.

7 – Кнопка очищення поля вхідних даних 5.

8 – Кнопка очищення поля вихідних даних 6.

То ж тепер запустимо симуляцію і будемо натискати кнопки, що підключені до мікроконтролеру. При цьому у полі вихідних даних 6 повинні з’являтися числа, що або збільшуються, або зменшуються, відповідно до того, яка кнопка натиснута (рис. 4.23).

Тепер перевіримо, як працює прийом даних. Для цього у полі вводу строки 1 введемо будь-яке число, наприклад, 236. І натиснемо кнопку CR (2), щоб символ повернення каретки автоматично додавався до кінця строки. Якщо цього не зробити, но на індикаторі не буде нічого відображатися відповідно до логіки роботи нашої програми.

Тепер якщо натиснути на клавіатурі клавішу Enter, введене число повинно передатися до мікроконтролера і відобразитися на

семисегментному індикаторі. Одночасно воно повинно відобразитися у полі вхідних даних 5 (рис. 4.24).

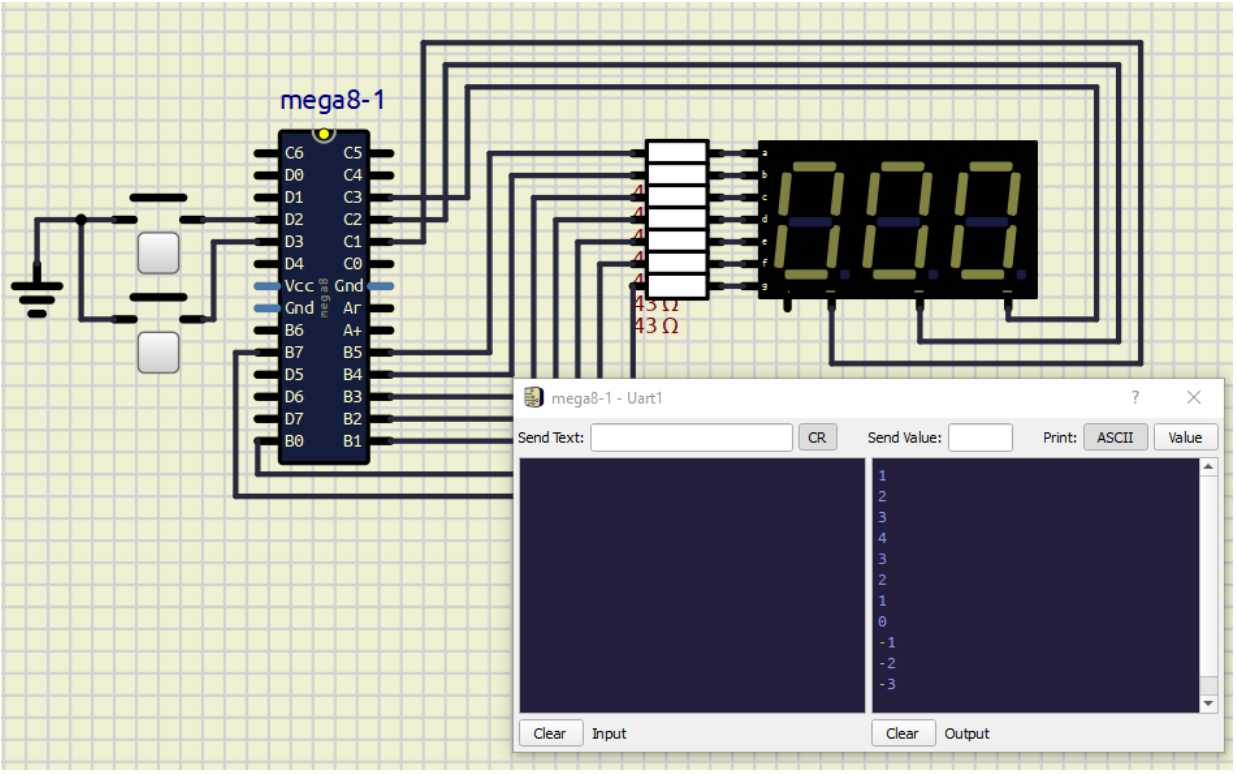


Рисунок 4.23 – Результат передачі даних мікроконтролером

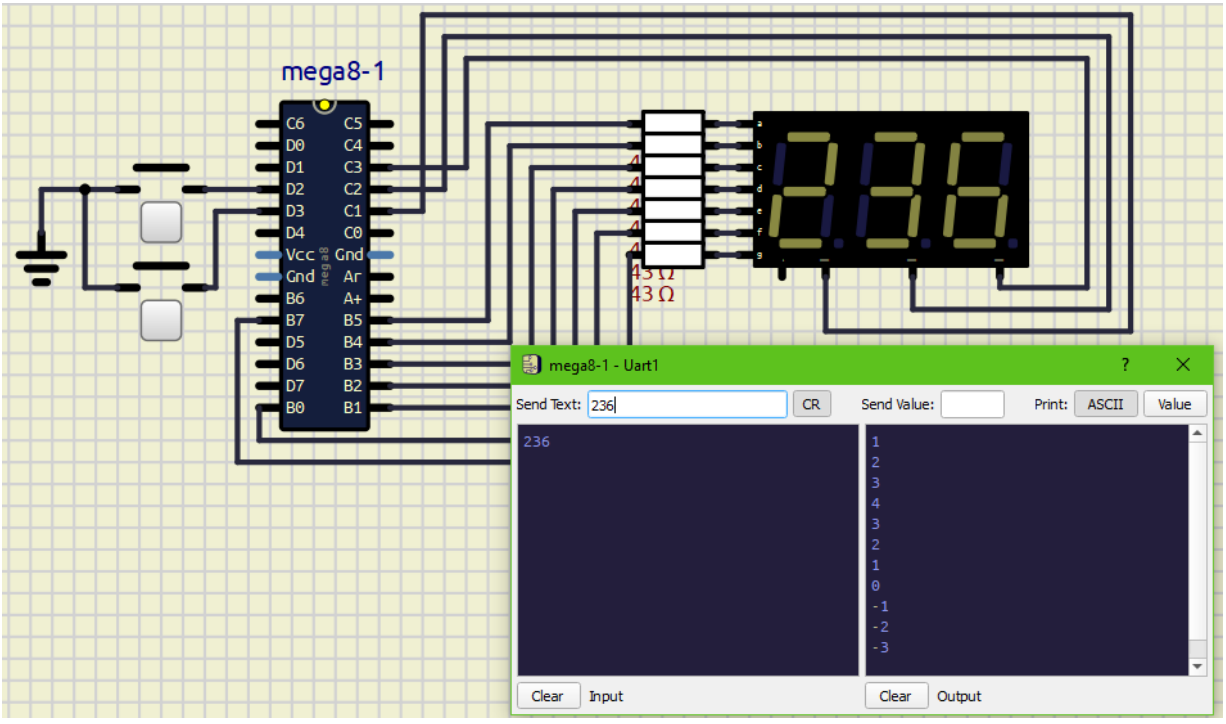


Рисунок 4.24 – Результат прийому даних мікроконтролером

Як бачимо, програма працює, як потрібно. Можна спробувати вводити різні числа і переконатися, що вони відображаються коректно. При вводі числа, більшого за 999, на індикаторі відобразиться 999. У програмі ніяк не опрацьовуються від'ємні числа, то ж при їх вводі результат буде невірним.

4.4 Завдання до практичної роботи

1. Створити принципову електричну схему, згідно з варіантом завдань (табл. 4.3).
2. Написати програму, що виконує поставлене завдання.
3. Створити файл прошивки, загрузити його у мікроконтролер та переконатися, що все працює, як треба.

Таблиця 4.3 – Варіанти завдань до практичної роботи №4

Варіант	Завдання
1	Підключити дворозрядний семисегментний індикатор зі спільним анодом наступним чином: A – PB0, B – PB1, C – PB2, D – PB3, E – PB4, F – PB5, G – PB6, перший розряд – PC0, другий розряд – PC1. Налаштувати модуль USART для роботи в асинхронному режимі з такими параметрами: швидкість 2400 бод, кількість бітів даних – 8, кількість стопових бітів – 2, перевірка парності відсутня. При прийомі числа у діапазоні від 0 до 255 через USART виводити його на індикатор у шістнадцятковому форматі. У моніторі послідовного порту число передавати саме як число за допомогою поля Send Value.
2	Підключити трирозрядний семисегментний індикатор зі спільним катодом наступним чином: A – PC0, B – PC1, C – PC2, D – PC3, E – PC4, F – PC5, G – PC6, перший розряд – PD0, другий розряд – PD1, третій розряд – PD4. Налаштувати модуль USART для роботи в асинхронному режимі з такими параметрами: швидкість 4800 бод, кількість бітів даних – 8, кількість стопових бітів – 1, перевірка парності - непарність. При прийомі через UART строки, перетворювати її на число зі знаком та відображати його на семисегментному індикаторі. Символом закінчення передачі строки вважати повернення каретки. Якщо передане число більше 999, відображати 999 замість нього. Якщо передане число менше -99, відображати -99.
3	Підключити до порту PB мікроконтролера вісім світлодіодів. Налаштувати модуль USART для роботи в асинхронному режимі з такими параметрами: швидкість 9600 бод, кількість бітів даних – 8, кількість стопових бітів – 1, перевірка парності - парність. При прийомі числа у діапазоні від 0 до 255 через USART виводити його за допомогою світлодіодів у двійковому форматі. У моніторі послідовного порту число передавати саме як число за допомогою поля Send Value.
4	Підключити до портів PB та PC мікроконтролера десять світлодіодів. Налаштувати модуль USART для роботи в асинхронному режимі з такими параметрами: швидкість 2400 бод, кількість бітів даних – 5, кількість стопових бітів – 1, перевірка парності - непарність. При прийомі через USART символу від 0 до 9 вмикати відповідний світлодіод. При прийомі будь-якого іншого символу всі світлодіоди

Варіант	Завдання
	вимкнути. У моніторі послідовного порту символ передавати за допомогою поля Send Text.
5	Налаштувати модуль USART для роботи в асинхронному режимі з такими параметрами: швидкість 4800 бод, кількість бітів даних – 8, кількість стопових бітів – 2, перевірка парності - парність. Налаштувати таймер 1 в режимі скиду при збігу з періодом спрацювання 1 секунда. При перериванні таймеру збільшувати якусь змінну на 1 та відправляти її значення через UART у вигляді строки з символами повернення каретки (‘\r’) та переводу строки (‘\n’) наприкінці.
6	Налаштувати модуль USART для роботи в асинхронному режимі з такими параметрами: швидкість 9600 бод, кількість бітів даних – 7, кількість стопових бітів – 1, перевірка парності – непарність. Налаштувати таймер 1 в режимі скиду при збігу з періодом спрацювання 2 секунди. При перериванні таймеру генерувати випадкове число у диапазоні від 0 до 100 та відправляти його через UART у вигляді власне числа.
7	Підключити кнопку до виводу PD2. Налаштувати модуль USART для роботи в асинхронному режимі з такими параметрами: швидкість 2400 бод, кількість бітів даних – 8, кількість стопових бітів – 2, перевірка парності – немає. Ввімкнути таймер 0 без попереднього дільника частоти. При натисканні на кнопку зчитувати поточне значення лічильника таймера та передавати його через UART у вигляді строки з символами повернення каретки (‘\r’) та переводу строки (‘\n’) наприкінці. Обробку натискання кнопки робити за допомогою переривання INT0.
8	Підключити кнопку до виводу PD3. Налаштувати модуль USART для роботи в асинхронному режимі з такими параметрами: швидкість 4800 бод, кількість бітів даних – 6, кількість стопових бітів – 1, перевірка парності – непарність. При натисканні на кнопку генерувати випадкове число у діапазоні від 0 до 63 та передавати його через UART у вигляді строки з символами повернення каретки (‘\r’) та переводу строки (‘\n’) наприкінці. Обробку натискання кнопки робити за допомогою переривання INT1.
9	Підключити кнопку до виводу PD2. Налаштувати модуль USART для роботи в асинхронному режимі з такими параметрами: швидкість 9600 бод, кількість бітів даних – 8, кількість стопових бітів – 2, перевірка парності – парність. При натисканні на кнопку, починати кожної секунди генерувати випадкове число у діапазоні від 0 до 255 та передавати його через UART у вигляді власне числа. При повторному натисканні на кнопку генерацію зупинити. Обробку натискання кнопки робити за допомогою переривання INT0.
10	Підключити кнопку до виводу PD3. Налаштувати модуль USART для роботи в асинхронному режимі з такими параметрами: швидкість 2400 бод, кількість бітів даних – 8, кількість стопових бітів – 2, перевірка парності – парність. Налаштувати таймер 1 в звичайному режимі з тактовою частотою, рівною частоті $clk_{I/O}$. При натисканні на кнопку зчитувати поточне значення лічильника таймера та передавати його через UART у вигляді двох восьмибітних чисел: старшого і молодшого байтів. Обробку натискання кнопки робити за допомогою переривання INT1.

Варіант	Завдання
11	Налаштувати модуль USART для роботи в асинхронному режимі з такими параметрами: швидкість 4800 бод, кількість бітів даних – 8, кількість стопових бітів – 1, перевірка парності – парність. При прийомі через UART строки довжиною до 64 символів, що закінчується символом повернення каретки, рахувати кількість цифр у ній і передавати це значення через UART як число.
12	Налаштувати модуль USART для роботи в асинхронному режимі з такими параметрами: швидкість 9600 бод, кількість бітів даних – 8, кількість стопових бітів – 2, перевірка парності – непарність. При прийомі через UART строки довжиною до 100 символів, що закінчується символом повернення каретки, замінити всі малі літери на великі і передати нову строку назад через UART.

4.5 Питання для самоперевірки

1. Принцип роботи модуля USART мікроконтролера ATMega8.
2. Як налаштувати швидкість передачі даних UART?
3. Які є налаштування пакету даних UART?
4. Які переривання може генерувати модуль USART мікроконтролера ATMega8?

4.6 Перелік рекомендованих джерел

1. Організація передачі даних через UART/USART : лекція 11. URL: https://learn.ztu.edu.ua/pluginfile.php/311730/mod_folder/content/0/%D0%9B%D0%B5%D0%BA%D1%86%D1%96%D1%8F%2011.pdf (дата звернення: 02.07.2024).
2. ATMega8 : технічна документація на мікроконтролер. URL:: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2486-8-bit-AVR-microcontroller-ATmega8_L_datasheet.pdf (дата звернення: 02.07.2024).
3. 8-bit AVR® MCUs : інформація про мікроконтролери. URL: https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/8-bit-mcus/avr-mcus (дата звернення: 02.07.2024).
4. Конспект лекцій з дисципліни «Мікропроцесорна техніка» для здобувачів вищої освіти першого (бакалаврського) рівня зі спеціальності 153 «Мікро-та наносистемна техніка» за освітньо-професійною програмою «Мікро-та наносистемна техніка» та зі спеціальності 171 «Електроніка» за освітньо-професійною програмою «Електроніка» / уклад. О. М. Гулєша. Кам‘янське : ДДТУ, 2020 р. 57 с.
5. Основи Програмування AVR C. DevZone. URL: https://devzone.org.ua/post/osnovi-programuvannia-avr-c (дата звернення: 02.07.2024).

5 ПРАКТИЧНА РОБОТА №5 «ПРОГРАМУВАННЯ АНАЛОГОВИХ МОДУЛІВ МІКРОКОНТРОЛЕРА AVR»

5.1 Завдання

Освоєння прийомів програмування аналогових модулів мікроконтролерів AVR.

Завдання практичної роботи:

– Створення електричної схеми у програмі SimulIDE відповідно до завдання на практичну роботу.

– Написання програми для мікроконтролеру відповідно до завдання на практичну роботу.

5.2 Теоретичні дані

5.2.1 Аналоговий компаратор

Аналоговий компаратор порівнює вхідні значення на позитивному вході AIN0 і негативному вході AIN1. Коли напруга на позитивному вході AIN0 перевищує напругу на негативному вході AIN1, вихід аналогового компаратора ACO стає рівним логічній 1. Вихід компаратора можна налаштувати для запуску функції захоплення таймера/лічильника1. Крім того, компаратор може ініціювати окреме переривання, виділене для аналогового компаратора. Користувач може вибрати тип спрацьовування переривання при зростанні, падінні або перемиканні вихідного сигналу компаратора. Блок-схема компаратора та його оточуюча логіка показані на рисунку 5.1.

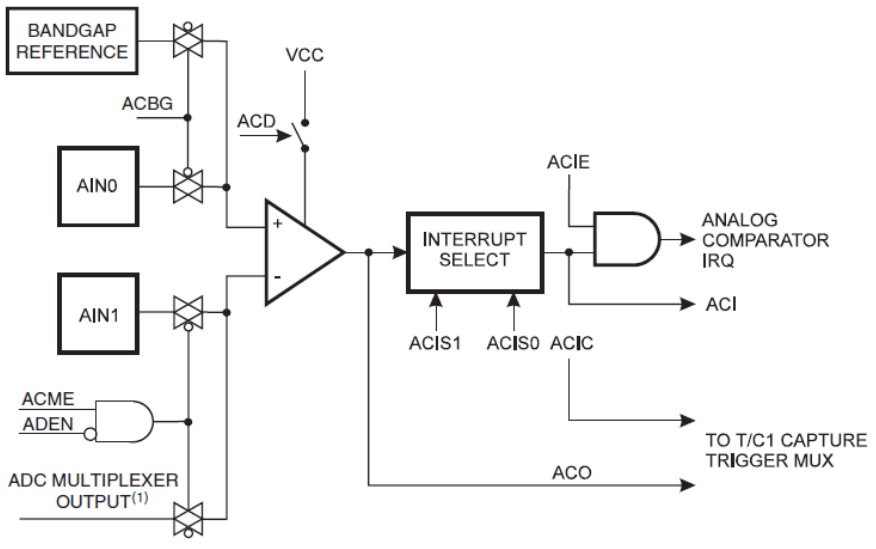


Рисунок 5.1 – Блок-схема компаратора та його оточуюча логіка

Регістри аналогового компаратору мікроконтролера ATMega8. Регістр спеціальних функцій вводу-виводу – SFIOR (рис. 5.2).

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	ACME	PUD	PSR2	PSR10	SFIOR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 5.2 – Регістр спеціальних функцій вводу-виводу – SFIOR

• **Біт 3 – ACME: увімкнення мультиплексора аналогового компаратора.** Коли в цей біт записується логічна одиниця, і АЦП вимкнено (ADEN в ADCSRA дорівнює нулю), мультиплексор АЦП вибирає негативний вхід для аналогового компаратора. Коли в цей біт записується логічний нуль, вхід AIN1 під'єднується до негативного входу аналогового компаратора. Детальний опис цього біта буде розглянуто далі.

Регістр управління та стану аналогового компаратора – ACSR (рис. 5.3).

Bit	7	6	5	4	3	2	1	0	
	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	ACSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	N/A	0	0	0	0	0	

Рисунок 5.3 – Регістр управління та стану аналогового компаратора – ACSR

• **Біт 7 – ACD: вимкнення аналогового компаратора.** Коли в цей біт записується логічна одиниця, живлення аналогового компаратора вимикається. Цей біт можна встановити в будь-який час, щоб вимкнути аналоговий компаратор. Це зменшить енергоспоживання в активному режимі та режимі очікування. При зміні біта ACD необхідно вимкнути переривання аналогового компаратора шляхом очищення біта ACIE в ACSR. Інакше при зміні біта може виникнути переривання.

• **Біт 6 – ACBG: вибір забороненої зони аналогового компаратора.** Коли цей біт установлено, опорна напруга фіксованої забороненої зони замінює позитивний вхід аналогового компаратора. Коли цей біт очищено, вхід AIN0 під'єднується до позитивного входу аналогового компаратора.

ATmega8 має внутрішнє джерело опорної напруги забороненої зони. Цей еталонний сигнал використовується для виявлення зниження напруги живлення, і його можна використовувати як вхідний сигнал для аналогового компаратора або АЦП. Величина напруги цього джерела дорівнює 2,56 В для АЦП та аналогового компаратору.

• **Біт 5 – ACO: вихід аналогового компаратора.** Вихід аналогового компаратора синхронізується, а потім безпосередньо підключається до ACO. Синхронізація вносить затримку в 1 - 2 такти.

• **Біт 4 – ACI: прапор переривання аналогового компаратора.** Цей біт встановлюється апаратним забезпеченням, коли вихідна подія компаратора, визначена бітами ACIS1 і ACIS0**,** генерує переривання. Процедура переривання аналогового компаратора виконується, якщо встановлено біт ACIE та встановлено біт I у SREG. ACI очищається апаратним забезпеченням під час виконання відповідного вектора обробки переривань. Крім того, ACI очищається шляхом запису логічної одиниці до прапора.

• **Біт 3 – ACIE: увімкнення переривання аналогового компаратора.** Коли в біт ACIE записується логічна одиниця, і біт I у регістрі стану встановлений, активується переривання аналогового компаратора. Коли в нього записується логічний нуль, переривання відключається.

• **Біт 2 – ACIC: увімкнення захоплення від аналогового компаратора.** Коли в цей біт записана логічна одиниця, він дозволяє аналоговому компаратору запускати функцію захоплення у таймері/лічильнику 1. Вихід компаратора в цьому випадку безпосередньо підключений до зовнішньої логіки захоплення, завдяки чому компаратор використовує придушник шуму і функції вибору фронту переривання по захопленню таймера/лічильника 1. Коли в цей біт записаний логічний нуль, зв’язок між аналоговим компаратором і функцією захоплення розривається. Щоб змусити компаратор запускати переривання по захопленню таймера/лічильника 1, необхідно встановити біт TICIE1 у регістрі маски переривання таймерів (TIMSK).

• **Біти 1,0 – ACIS1, ACIS0: вибір режиму переривання від аналогового компаратора.** Ці біти визначають, які події компаратора викликають переривання аналогового компаратора. Різні налаштування наведено в таблиці 5.1.

Таблиця 5.1 – Вибір режиму переривання від аналогового компаратора

ACIS1	ACIS0	Режим переривання
0	0	Переривання компаратора при будь-якій зміні стану виходу
0	1	Зарезервовано
1	0	Переривання компаратора по спадаючому вихідному фронту
1	1	Переривання компаратора по наростаючому вихідному фронту

Під час зміни бітів ACIS1/ACIS0 переривання від аналогового компаратора має бути відключено шляхом очищення його біта дозволу переривання в регістрі ACSR. Інакше при зміні бітів може виникнути переривання.

Мультиплексований вхід аналогового компаратора. Можна вибрати будь-який з входів ADC7..0 для підключення до негативного входу аналогового компаратора. Мультиплексор АЦП використовується для вибору цього входу, отже, АЦП має бути вимкнено, щоб використовувати цю функцію. Якщо встановлено біт увімкнення мультиплексору аналогового компаратора (ACME у SFIOR), а АЦП вимкнено (ADEN у ADCSRA дорівнює нулю), MUX2..0 у ADMUX вибирає вхідний контакт для підключення до негативного входу аналогового компаратора, як показано у таблиці 5.2. Якщо ACME скинуто або встановлено ADEN, вхід AIN1 використовується як негативний вхід аналогового компаратора.

Таблиця 5.2 – Вибір вхідного контакту для підключення до негативного входу аналогового компаратора

ACME	ADEN	MUX2..0	Негативний вхід аналогового компаратора
0	x	xxx	AIN1
1	1	xxx	AIN1
1	0	000	ADC0
1	0	001	ADC1
1	0	010	ADC2
1	0	011	ADC3
1	0	100	ADC4
1	0	101	ADC5
1	0	110	ADC6
1	0	111	ADC7

5.2.2 Модуль АЦП мікроконтролерів AVR

АЦП мікроконтролера ATMega8.

Особливості:

• 10-бітна роздільна здатність
• Інтегральна нелінійність 0,5 LSB
• Абсолютна точність ±2 LSB
• Час перетворення 13 мкс - 260 мкс
• До 15 kSPS при максимальній роздільній здатності
• 6 мультиплексованих односторонніх вхідних каналів
• 2 додаткові мультиплексовані односторонні вхідні канали (тільки для корпусів TQFP і QFN/MLF)
• Опціональне вирівнювання результату перетворення ліворуч
• діапазон вхідної напруги АЦП 0 - VCC
• Вибір опорної напруги АЦП 2,56 В
• Режим безперервного або одноразового перетворення
• Переривання після завершення перетворення АЦП
• Пригнічувач шуму в режимі сну

ATmega8 має 10-бітний АЦП послідовного наближення. АЦП підключений до 8-канального аналогового мультиплексора, який дозволяє

підключати вісім односторонніх входів напруги, підключених до виводів порту С. Вхідна напруга вимірюється відносно 0 В (GND).

АЦП містить схему вибірки та утримання, яка гарантує, що вхідна напруга АЦП утримується на постійному рівні під час перетворення. Блок-схема АЦП показана на рисунку 5.4.

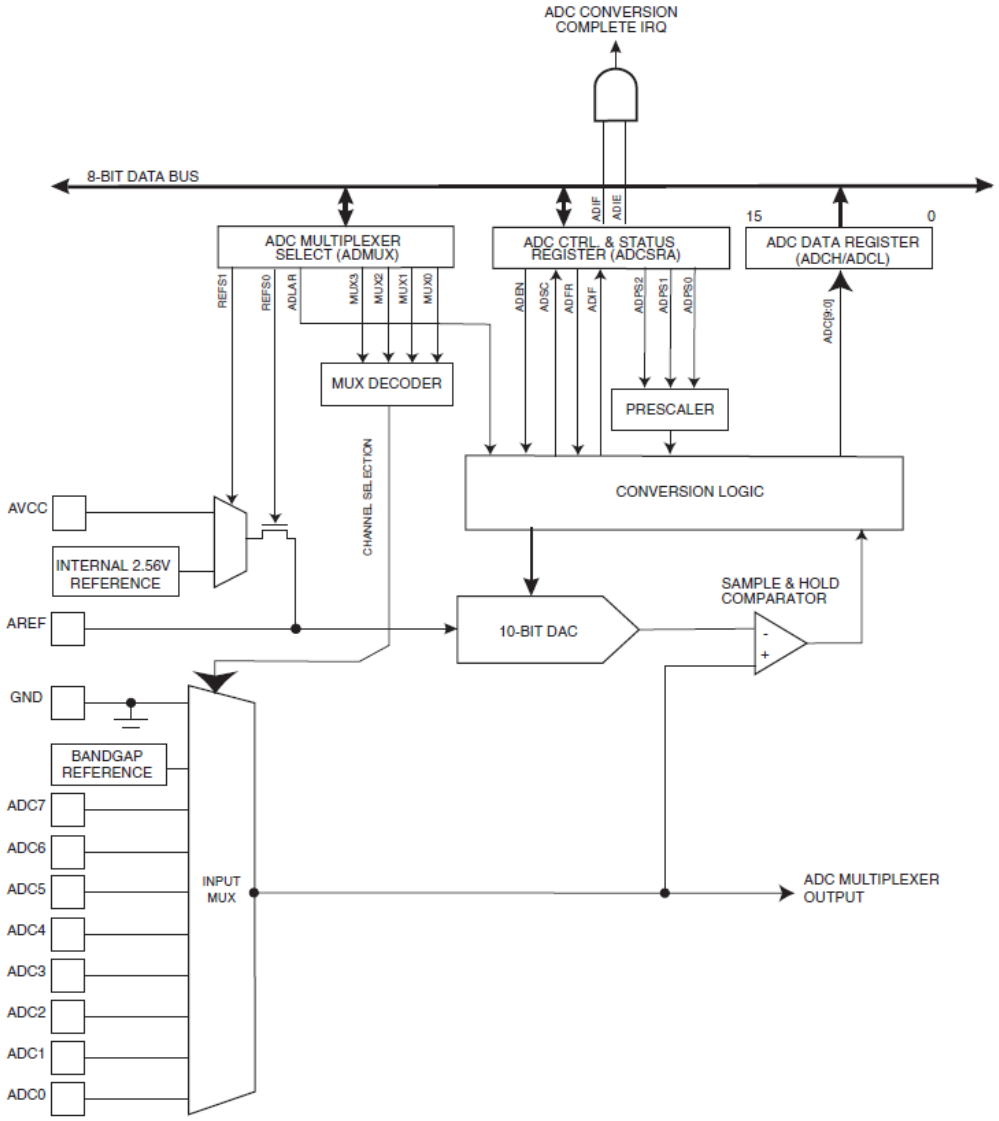


Рисунок 5.4 – Блок-схема АЦП

АЦП має окремий аналоговий вхід напруги живлення, AVCC. AVCC не має відрізнятися більше ніж на ±0,3 В від VCC.

АЦП перетворює аналогову вхідну напругу в 10-бітне цифрове значення шляхом послідовного наближення. Мінімальне значення являє собою GND, а максимальне значення являє собою напругу на виводі AREF мінус 1 LSB. За бажанням, AVCC або внутрішня опорна напруга 2,56 В може бути підключена до виводу AREF шляхом запису в біти REFSn у регістрі ADMUX. Таким чином, внутрішня опорна напруга може бути

розв’язана зовнішнім конденсатором на виводі AREF для підвищення завадостійкості.

Аналоговий вхідний канал вибирається шляхом запису в біти MUX у регістрі ADMUX. Будь-який із вхідних контактів АЦП, а також GND і опорну напругу з фіксованою забороненою зоною можна вибрати як входи АЦП. АЦП вмикається встановленням біта ввімкнення АЦП ADEN у регістрі ADCSRA. Вибір опорної напруги та вхідного каналу не набуде чинності, доки не буде встановлено ADEN. АЦП не споживає електроенергію, коли ADEN очищено, тому рекомендується вимкнути АЦП перед переходом у енергозберігаючі режими сну.

АЦП генерує 10-бітний результат, який представляється в регістрах даних АЦП ADCH і ADCL. За замовчуванням результат представлено з вирівнюванням праворуч, але за бажанням його можна представити вирівняним ліворуч шляхом встановлення біта ADLAR у ADMUX.

Якщо результат вирівняний ліворуч, і не потрібна точність більше 8 біт, достатньо прочитати ADCH. В іншому випадку спочатку потрібно прочитати ADCL, а потім ADCH, щоб переконатися, що вміст регістрів даних належить до того самого перетворення. Після зчитування ADCL доступ ADC до регістрів даних блокується. Це означає, що якщо ADCL було прочитано, і перетворення завершується до того, як буде зчитано ADCH, жоден регістр не оновлюється, і результат перетворення втрачається. Коли зчитується ADCH, доступ ADC до регістрів ADCH і ADCL знову вмикається.

АЦП має власне переривання, яке може бути викликане після завершення перетворення. Коли доступ ADC до регістрів даних заборонено між читанням ADCH і ADCL, переривання спрацює, навіть якщо результат буде втрачено.

Початок перетворення. Одиночне перетворення починається записом логічної одиниці в біт початку перетворення АЦП, ADSC. Цей біт залишається високим, доки триває перетворення, і буде очищено апаратним забезпеченням після завершення перетворення. Якщо під час перетворення вибрано інший канал даних, АЦП завершить поточне перетворення перед виконанням зміни каналу.

У режимі безперервного перетворення АЦП постійно виконує вибірку та оновлює регістр даних АЦП. Режим безперервного перетворення вибирається записом одиниці у біт ADFR в регістрі ADCSRA. Перше перетворення має бути розпочато із запису логічної одиниці до біта ADSC у регістрі ADCSRA. У цьому режимі АЦП виконуватиме послідовні перетворення незалежно від того, чи скинуто прапор переривання АЦП, ADIF чи ні.

Попереднє поділення та час перетворення. За замовчуванням схема послідовного наближення вимагає вхідної тактової частоти від 50 до 200 кГц для отримання максимальної роздільної здатності. Якщо

необхідна роздільна здатність нижче 10 біт, вхідна тактова частота АЦП може бути вищою за 200 кГц, щоб отримати вищу частоту дискретизації.

Модуль АЦП містить попередній дільник, який генерує прийнятну тактову частоту АЦП на будь-якій частоті ЦП вище 100 кГц (рис. 5.5). Попереднє поділення встановлюється бітами ADPS в ADCSRA. Попередній дільник починає відлік з моменту ввімкнення АЦП установкою біта ADEN в ADCSRA. Попередній дільник продовжує працювати до тих пір, поки встановлено біт ADEN, і постійно скидається, коли ADEN низький.

Якщо ініціювати перетворення шляхом встановлення біта ADSC у ADCSRA, воно починається з наступного наростаючого фронту тактового циклу АЦП. Нормальне перетворення займає 13 тактів АЦП. Перше перетворення після ввімкнення АЦП (встановлено ADEN в ADCSRA) потребує 25 тактових циклів АЦП для ініціалізації аналогової схеми.

Фактична вибірка й утримання займає 1,5 тактів АЦП після початку звичайного перетворення та 13,5 такту АЦП після початку першого перетворення. Після завершення перетворення результат записується в регістри даних АЦП і встановлюється ADIF. У режимі одиночного перетворення одночасно очищується біт ADSC. Після цього програмне забезпечення може знову встановити ADSC, і нове перетворення буде ініційовано на першому наростаючому фронті синхронізації АЦП.

У безперервному режимі нове перетворення розпочнеться відразу після завершення перетворення, поки біт ADSC залишається високим.

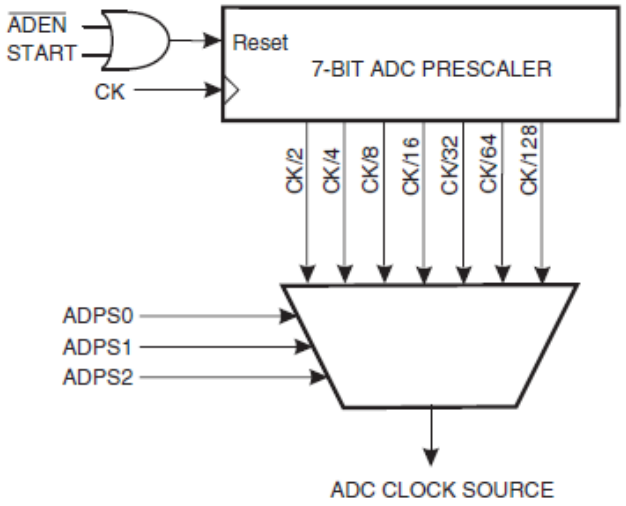


Рисунок 5.5 – Попереднє поділення АЦП

Зміна вхідного каналу або джерела опорної напруги. Біти MUXn і REFS1:0 у регістрі ADMUX буферизуються через тимчасовий регістр, до якого ЦП має довільний доступ. Це гарантує, що під час перетворення вибір каналів і опорної напруги відбуватиметься лише в безпечному місці протягом перетворення. Вибір каналу та опорної напруги може постійно

оновлюватися, доки не розпочнеться перетворення. Після початку перетворення вибір каналу та опорної напруги блокується, щоб забезпечити достатній час вибірки для АЦП. Безперервне оновлення відновлюється в останньому такті АЦП перед завершенням перетворення (встановлено ADIF в ADCSRA). Зверніть увагу, що перетворення починається на наступному наростаючому фронті синхронізації АЦП після запису ADSC. Таким чином, користувачеві рекомендується не записувати нові значення вибору каналу або опорної напруги в ADMUX до тих пір, поки не пройде хоча б один такт АЦП після запису ADSC.

Якщо і ADFR, і ADEN записані як одиниці, переривання може відбутися в будь-який час. Якщо реєстр ADMUX змінено протягом цього періоду, користувач не зможе визначити, чи базується наступне перетворення на старих чи нових налаштуваннях. ADMUX можна безпечно оновити такими способами:

1. Коли ADFR або ADEN очищено
2. Під час перетворення, через мінімум один такт АЦП після старту перетворення
3. Після перетворення, перш ніж прапор переривання, який використовується як джерело запуску, буде очищено.

Під час оновлення ADMUX при одній із цих умов нові параметри вплинуть на наступне перетворення АЦП.

Вхідні канали АЦП. Змінюючи вхідний канал, користувач повинен дотримуватися наведених нижче вказівок, щоб переконатися, що вибрано правильний канал:

У режимі одиночного перетворення завжди вибирайте канал перед початком перетворення. Вибір каналу можна змінити через один такт АЦП після запису в біт ADSC. Однак найпростіший спосіб — дочекатися завершення перетворення перед зміною каналу.

У режимі безперервного перетворення завжди вибирайте канал перед початком першого перетворення. Вибір каналу можна змінити через один такт АЦП після запису в ADSC. Однак найпростіший спосіб — дочекатися завершення першого перетворення, а потім змінити канал. Оскільки наступне перетворення вже почалося автоматично, наступний результат відображатиме попередній вибір каналу. Подальші перетворення відображатимуть новий вибраний канал.

Опорна напруга АЦП. Опорна напруга для АЦП (VREF) задає діапазон перетворення для АЦП. Значення каналів, які перевищують VREF, призведуть до отримання кодів, близьких до 0x3FF. VREF можна вибрати як AVCC, внутрішній опорний сигнал 2,56 В або зовнішня напруга AREF.

AVCC підключається до АЦП через пасивний комутатор. Внутрішній опорний сигнал 2,56 В подається з внутрішнього опорного джерела забороненої зони (VBG) через внутрішній підсилювач. У будь-якому випадку зовнішній вивід AREF підключається безпосередньо до АЦП, і

опорну напругу можна зробити більш стійкою до шуму, підключивши конденсатор між виводом AREF і землею. VREF також можна виміряти на виводі AREF за допомогою вольтметра з високим опором. Зауважте, що VREF є джерелом високого імпедансу, тому до системи слід підключати лише ємнісне навантаження.

Якщо користувач має фіксоване джерело напруги, підключене до виводу AREF, він не може використовувати інші варіанти опорної напруги в програмі, оскільки вони будуть закорочені з зовнішньою напругою. Якщо до виводу AREF не подається зовнішня напруга, користувач може перемикатися між AVCC і 2,56 В в якості опорного вибору. Перший результат перетворення АЦП після перемикання джерела еталонної напруги може бути неточним, тому користувачеві рекомендується відхилити цей результат.

Пригнічувач шуму АЦП. АЦП має пригнічувач шуму, який дозволяє робити перетворення під час режиму сну, щоб зменшити шум, викликаний ядром процесора та іншими периферійними пристроями вводу/виводу. Пригнічувач шуму можна використовувати в режимі зменшення шуму АЦП і в неактивному режимі. Щоб скористатися цією функцією, необхідно виконати таку процедуру:

1. Переконайтеся, що АЦП увімкнений та не зайнятий перетворенням. Необхідно вибрати режим одиночного перетворення та ввімкнути переривання по завершенню перетворення АЦП.

2. Увійдіть у режим зменшення шуму АЦП (або неактивний режим). АЦП розпочне перетворення після зупинки ЦП

3. Якщо до завершення перетворення АЦП не виникає жодних інших переривань, переривання АЦП виведе з режиму сну ЦП і виконає процедуру переривання по завершенню перетворення АЦП. Якщо інше переривання активує ЦП до завершення перетворення АЦП, це переривання буде виконано, і після завершення перетворення АЦП буде згенеровано запит на переривання по завершенню перетворення АЦП. ЦП залишатиметься в активному режимі, доки не буде виконано нову команду сну. Зауважте, що АЦП не вимикатиметься автоматично під час входу в інші режими сну, крім неактивного режиму та режиму зменшення шуму АЦП. Користувачеві рекомендується записати нуль в ADEN перед входом у такі режими сну, щоб уникнути надмірного споживання енергії.

Аналогова вхідна схема. Аналогова вхідна схема показана на рисунку 5.6. Аналогове джерело, що подається до АЦП, піддається впливу ємності на виводі та вхідного струму цього виводу, незалежно від того, чи обрано цей канал як вхід для АЦП. Коли канал вибрано, джерело має зарядити конденсаторо S/H через послідовний опір (комбінований опір у вхідному тракті).

АЦП оптимізовано для аналогових сигналів з вихідним опором приблизно 10 кОм або менше. Якщо використовується таке джерело, час вибірки буде незначним. Якщо використовується джерело з вищим

імпедансом, час вибірки залежатиме від того, скільки часу потрібно джерелу для заряджання конденсатора S/H, і може значно відрізнятися. Користувачеві рекомендується використовувати лише джерела з низьким імпедансом із повільно змінними сигналами, оскільки це мінімізує необхідну передачу заряду до конденсатора S/H.

Компоненти сигналу, вищі за частоту Найквіста ($f_{ADC}/2$), не повинні бути присутніми для жодного типу каналів, щоб уникнути спотворень через непередбачувану згортку сигналу. Користувачеві рекомендується видалити високочастотні компоненти за допомогою фільтра низьких частот перед подачею сигналів на входи АЦП.

Методи пригнічування аналогового шуму. Цифрові схеми всередині та зовні пристрою створюють електромагнітні перешкоди, які можуть вплинути на точність аналогових вимірювань. Якщо точність перетворення має вирішальне значення, рівень шуму можна зменшити за допомогою наступних методів:

1. Тримайте шляхи аналогового сигналу якомога коротшими. Переконайтеся, що аналогові доріжки проходять по площині заземлення, і тримайте їх подалі від високошвидкісних комутаційних цифрових доріжок.

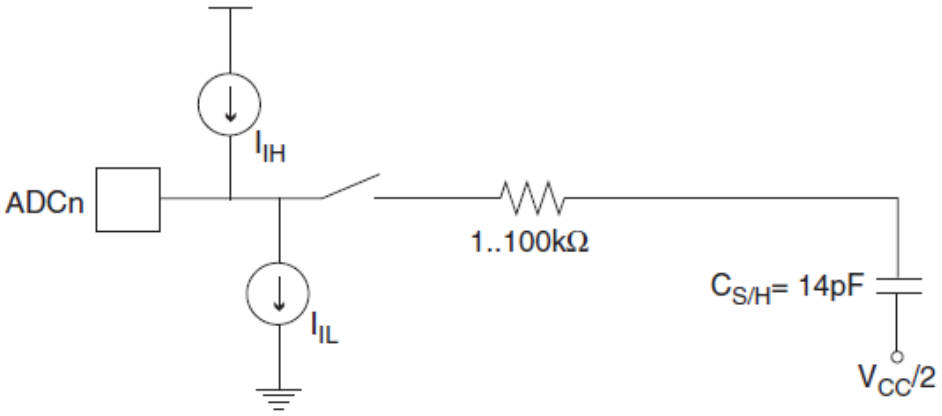


Рисунок 5.6 – Аналогова вхідна схема

2. Вивід AVCC на пристрої має бути підключений до цифрової напруги живлення VCC через LC-ланцюжок

3. Використовуйте функцію пригнічення шуму АЦП, щоб зменшити індукований шум від ЦП.

4. Якщо будь-які контакти порту ADC [3..0] використовуються як цифрові виходи, важливо, щоб вони не перемикалися під час перетворення. Однак використання двопровідного інтерфейсу (ADC4 і ADC5) вплине лише на перетворення на ADC4 і ADC5, а не інших каналах ADC.

Результат перетворення АЦП. Після завершення перетворення (ADIF дорівнює 1), результат перетворення можна зчитати з регістрів результатів АЦП (ADCL, ADCH).

Результат визначається за наступною формулою:

$$ADC = \frac{V_{IN} \cdot 1023}{V_{REF}} \quad (5.1)$$

де V_{IN} — напруга на вибраному вхідному контакті, а V_{REF} — вибрана опорна напруга. 0x000 представляє напругу землі, а 0x3FF представляє вибрану опорну напругу мінус один LSB.

Регістри АЦП:

Регістр мультиплексора АЦП – ADMUX (рис. 5.7):

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

Рисунок 5.7 – Регістр мультиплексора АЦП – ADMUX

• **Біти 7:6 – REFS1:0: біти вибору опорної напруги.** Ці біти вибирають опорну напругу для АЦП, як показано в таблиці. Якщо ці біти змінено під час перетворення, зміна не вступить в силу, доки це перетворення не буде завершено (ADIF в регістрі ADCSRA не буде встановлено). Внутрішню опорну напругу не можна використовувати, якщо зовнішня опорна напруга подається на вивід AREF.

Таблиця 5.3 – Біти вибору опорної напруги

REFS1	REFS0	Опорна напруга
0	0	AREF, внутрішній Vref вимкнено
0	1	AVCC із зовнішнім конденсатором на виводі AREF
1	0	Зарезервовано
1	1	Внутрішня опорна напруга 2,56 В із зовнішнім конденсатором на виводі AREF

• **Біт 5 – ADLAR: вирівнювання результату АЦП ліворуч.** Біт ADLAR впливає на представлення результату перетворення АЦП у регістрі даних АЦП. Запис одиниці у ADLAR вирівнює результат перетворення ліворуч. В іншому випадку результат вирівнюється праворуч. Зміна біта ADLAR негайно вплине на регістр даних АЦП, незалежно від будь-яких поточних перетворень. Повний опис цього біта буде наведено далі.

• **Біти 3:0 – MUX3:0: біти вибору аналогового каналу.** Значення цих бітів визначає, який аналоговий вхід підключаються до АЦП (див.

таблицю 5.4). Якщо ці біти змінено під час перетворення, зміна не набуде чинності, доки це перетворення не буде завершено (біт ADIF у регістрі ADCSRA не буде встановлено).

Таблиця 5.4 – Біти вибору аналогового каналу

MUX3..0	Вхід
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	Зарезервовано
1001	Зарезервовано
1010	Зарезервовано
1011	Зарезервовано
1100	Зарезервовано
1101	Зарезервовано
1110	1,3 В (V_{BG})
1111	0 В (земля)

Регістр управління та стану АЦП А – ADCSRA (рис. 5.8):

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 5.9 – Регістр управління та стану АЦП А – ADCSRA

• **Біт 7 – ADEN: увімкнення АЦП.** Запис одиниці в цей біт вмикає АЦП. При запису в нього нуля АЦП вимикається. Вимкнення АЦП під час перетворення призведе до припинення цього перетворення.

• **Біт 6 – ADSC: початок перетворення АЦП.** У режимі одиночного перетворення записуйте в цей біт одиницю, щоб почати кожне перетворення. У безперервному режимі запишіть в цей біт одиницю, щоб розпочати перше перетворення. Перше перетворення після запису ADSC після ввімкнення АЦП, або якщо ADSC записується одночасно з увімкненням АЦП, займе 25 тактів АЦП замість звичайних 13. Це перше перетворення виконує ініціалізацію АЦП. ADSC читатиметься як один, поки триває перетворення. Після завершення перетворення він повертається до нуля. Запис нуля в цей біт не має ефекту.

• **Біт 5 – ADFR: вибір режиму безперервного перетворення АЦП.** Коли цей біт встановлено (один), АЦП працює у режимі безперервного перетворення. У цьому режимі АЦП безперервно збирає та оновлює

регістри даних. Очищення цього біта (нуль) призведе до завершення режиму безперервного перетворення.

• **Біт 4 – ADIF: прапор переривання АЦП.** Цей біт встановлюється після завершення перетворення АЦП і оновлення регістрів даних. Переривання по завершенню перетворення АЦП виконується, якщо встановлено біт ADIE та біт I у регістрі SREG. ADIF очищується апаратним забезпеченням під час виконання відповідного вектору обробки переривань. Крім того, ADIF очищується записом логічної одиниці до нього.

• **Біт 3 – ADIE: дозвіл переривання АЦП.** Коли в цей біт записано одиницю, і встановлено біт I у регістрі SREG, активується переривання по завершенню перетворення АЦП.

• **Біти 2:0 – ADPS2:0: біти вибору попереднього дільника АЦП.** Ці біти визначають коефіцієнт ділення між частотою XTAL і тактовою частотою на вході АЦП (див. табл. 5.5).

Таблиця 5.5 – Біти вибору 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

Регістр даних АЦП – ADCL і ADCH (рис. 5.9).

Після завершення перетворення АЦП результат знаходиться в цих двох регістрах.

Коли ADCL зчитується, регістр даних ADC не оновлюється, доки ADCH не буде зчитано. Отже, якщо результат вирівняний ліворуч і не потрібна точність більше 8 біт, достатньо прочитати ADCH. В іншому випадку спочатку потрібно прочитати ADCL, а потім ADCH.

Біт ADLAR в ADMUX і біти MUXn в ADMUX впливають на спосіб зчитування результату з регістрів. Якщо встановлено ADLAR, результат вирівняно ліворуч. Якщо ADLAR очищено (за замовчуванням), результат вирівняно праворуч.

ADLAR = 0

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

ADLAR = 1

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

Рисунок 5.9 – Регістр даних АЦП – ADCL і ADCH

• **ADC9:0: результат перетворення АЦП.** Ці біти представляють результат перетворення.

5.3 Приклад програми з використанням аналогового компаратору та аналого-цифрового перетворювача

Модифікуємо приклад, описаний в минулій практичній роботі.

Виконаємо наступне завдання на базі мікроконтролера ATMega8. Підключити трирозрядний семисегментний індикатор зі спільним катодом наступним чином: A – PB5, B – PB4, C – PB3, D – PB2, E – PB1, F – PB0, G – PB7, перший розряд – PD4, другий розряд – PD3, третій розряд – PD2. Підключити два регульованих джерела постійної напруги до входів аналогового компаратору AIN0 та AIN1 і одночасно до входів АЦП ADC0 та ADC1. Виводити на екран напругу того джерела, в якого вона в даний момент більша, з роздільною здатністю 10 мВ.

5.4 Принципова електрична схема

Принципова електрична схема, що реалізує поставлену задачу, показана на рис. 5.10.

Вона схожа на ту, що використовувалася у прикладі до попередньої практичної роботи, але в ній є нові компоненти, які нам ще не зустрічалися раніше. По-перше, це джерело напруги живлення мікроконтролера, яке у програмі називається Rail та знаходиться у підзаголовку Sources (рис. 5.11).

Воно має лише один параметр – власне величину напруги (рис. 5.12), яку ми залишимо за замовчанням рівною 5 В.

Це джерело треба підключити до входу AVCC мікроконтролера, який у програмі SimulIDE називається “A+”. Цей вхід буде використовуватися як джерело опорної напруги для АЦП, я якщо на нього не подати ніякої напруги, то на виході АЦП завжди буде 0.

Ще нам потрібні два регульованих джерела постійної напруги. Вони знаходяться у тому самому підзаголовку Sources і називаються Volt. Source (рис. 5.11). На схемі вони виглядають як прямокутники з круглим регулятором всередині (рис. 5.10). Якщо покрутити регулятор мишкою, то вихідна напруга джерела буде змінюватися від 0 (коли регулятор знаходиться у крайньому лівому положенні) до максимальної напруги (коли регулятор знаходиться у крайньому правому положенні). Також вихідну напругу можна задавати в налаштуваннях компоненту (рис. 5.13).

Як бачимо, цей компонент має лише два параметри: поточна напруга (Current Value) та максимальна напруга (Max. Voltage). Зверніть увагу, що цей компонент треба ввімкнути, бо за замовчанням він вимкнений, і виглядає наступним чином (рис. 5.14)

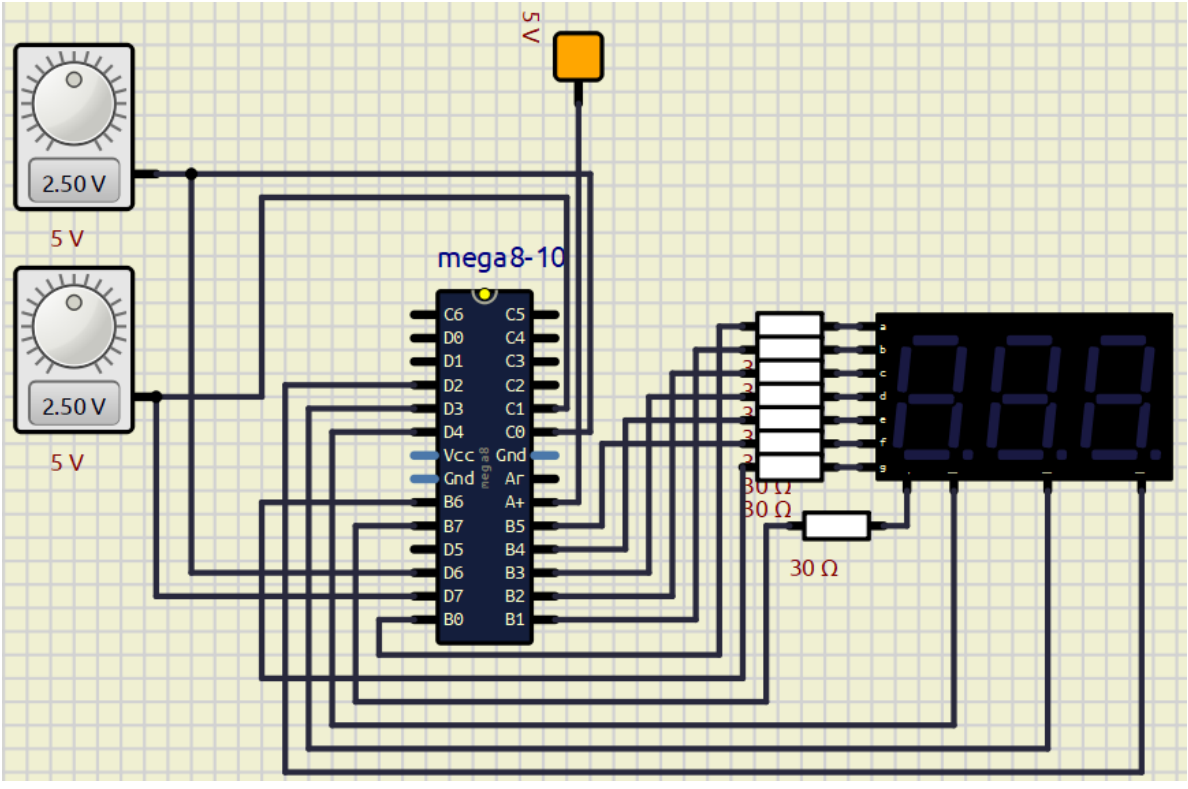


Рисунок 5.10 – Принципова електрична схема

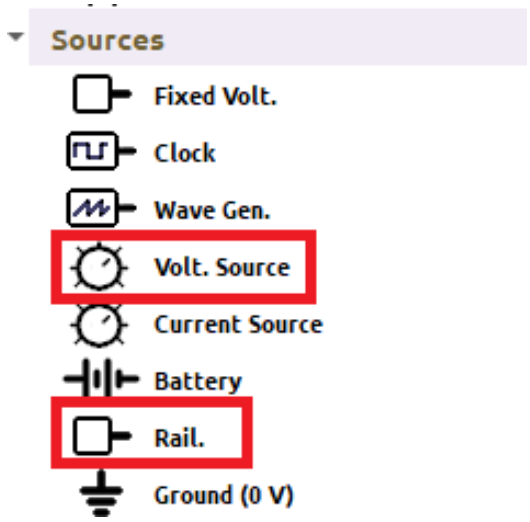


Рисунок 5.11 – Знаходження компонентів Rail та Volt. Source

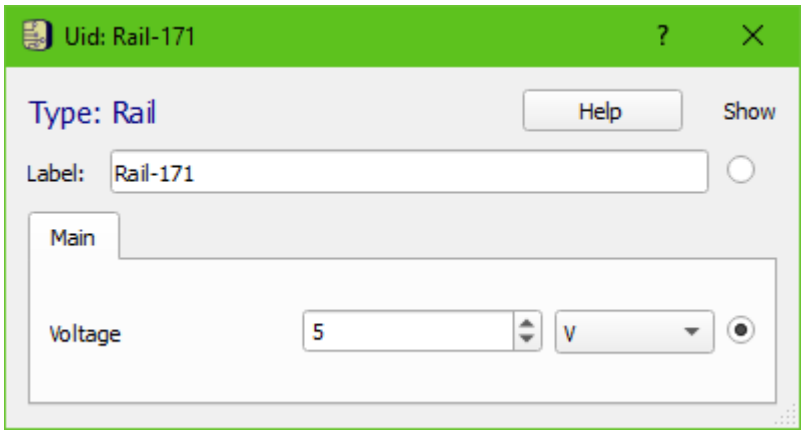


Рисунок 5.12 – Параметри компонента Rail

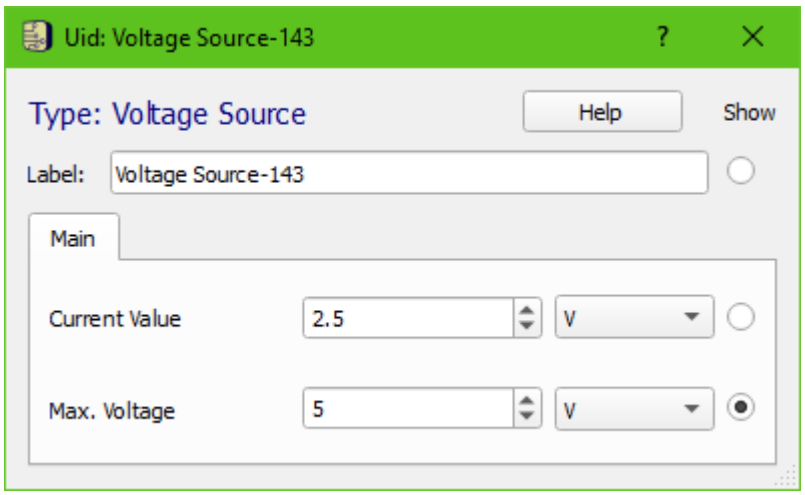


Рисунок 5.13 – Параметри компонента Volt. Source

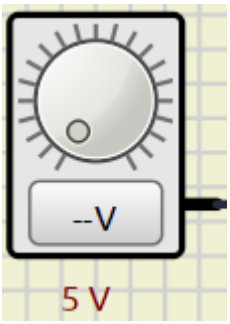


Рисунок 5.14 – Початковий вигляд регульованого джерела напруги

Для того, щоб ввімкнути це джерело, треба натиснути на надпис “--V” всередині нього, і тоді замість прочерку у надпису з’явиться величина поточної напруги, як показано на рис. 5.10.

Входи компаратора AIN0 та AIN1 мікроконтролера ATMega8 суміщені з виводами PD6 та PD7, відповідно. А входи АЦП ADC0-ADC6 сумішені з виводами PC0-PC6, відповідно.

Як бачимо на рис. 5.10, регульовані джерела напруги одночасно підключені до входів компаратора та АЦП. За допомогою компаратору ми будемо визначати, яка з двох напруг більша, і перемикати вхідний мультиплексор АЦП, щоб виміряти цю напругу.

5.4.1 Програмний код

Розглянемо тепер програму, що реалізує поставлену задачу (рис. 5.15).

Вона досить велика, але частину її ми вже розглядали в попередній практичній роботі. То ж тут ті рядки, які були вже розглянути раніше, ми детально описувати не будемо.

У рядках 1, 2 ми підключаємо необхідні заголовкові файли. Зверніть увагу, що в цій програмі ми не підключаємо файл delay.h, бо ми не будемо використовувати функції затримки для роботи з семисегментним індикатором, а замість того скористуємося таймером 0.

У рядках 4-15 ми визначаємо макроси для роботи з семисегментним індикатором.

У рядку 17 ми визначаємо змінну digit, яка буде відповідати номеру розряду семисегментного індикатора, який в даний момент світиться.

У рядку 18 ми визначаємо змінну number, яка уявляє собою число, що відповідає виміряній напрузі і виводиться на семисегментний індикатор.

У рядках 19-31 знаходиться знакогенератор для семисегментного індикатора.

Рядки 33-70 поки пропустимо, і перейдемо до головної функції програми, що розташована у рядках 72-91.

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

#define A _BV(PB0)
#define B _BV(PB1)
#define C _BV(PB2)
#define D _BV(PB3)
#define E _BV(PB4)
#define F _BV(PB5)
#define G _BV(PB6)
#define DP _BV(PB7)

#define D1 _BV(PD4)
#define D2 _BV(PD3)
#define D3 _BV(PD2)

uint8_t digit;
uint16_t number;
const uint8_t numbers[10] =
  {
    A | B | C | D | E | F,    //0
    B | C,                    //1
    A | B | D | E | G,        //2
    A | B | C | D | G ,       //3
    B | C | F | G,            //4
    A | C | D | F | G,        //5
    A | C | D | E | F | G,    //6
    A | B | C,                //7
    A | B | C | D | E | F | G,//8
    A | B | C | D | F | G     //9
  };

ISR (TIMER0_OVF_vect)
{
  ADCSRA |= _BV(ADSC);
  PORTD |= D1 | D2 | D3;
  switch (digit)
  {
    case 0:
      PORTB = numbers[number / 100] | DP;
      PORTD &= ~D1;
      break;
    case 1:
      PORTB = numbers[(number % 100) / 10];
      PORTD &= ~D2;
      break;
    case 2:
      PORTB = numbers[number % 10];
      PORTD &= ~D3;
      break;
  }
  digit ++;
  if (digit > 2)
    digit = 0;
}

ISR (ADC_vect)
{
  uint32_t res = ADCL;
  res += (ADCH << 8);
  number = (5 * 100 * res) / 1023;
}

ISR(ANA_COMP_vect)
{
  if(ACSR & _BV(ACO))
    ADMUX = _BV(REFS0);
  else
    ADMUX = _BV(REFS0) | _BV(MUX0);
}

int main (void)
{
  DDRB = 0xFF;
  DDRD = D1 | D2 | D3;

  TCCR0 = _BV(CS01) | _BV(CS00);
  TIMSK = _BV(TOIE0);

  SFIOR &= ~_BV(ACME);
  ACSR = _BV(ACIE);

  ADMUX = _BV(REFS0);
  ADCSRA = _BV(ADEN) | _BV(ADIE) | _BV(ADPS1) | _BV(ADPS0);

  digit = 0;

  sei();

  while (1);
}
```

Рисунок 5.15 – Програмний код

Спочатку треба налаштувати всі необхідні порти вводу/виводу. Виводи, до яких підключені і аноди, і катоди індикаторів, треба налаштувати як виходи. У рядку 74 ми записуємо одиниці у біти, які відповідають сегментам світлодіодного індикатора, у регістр DDRB. А у рядку 75 ми записуємо одиниці у біти, що відповідають розрядам світлодіодного індикатора, у регістр DDRD. Більше ніякі виводи налаштовувати не потрібно, адже вони будуть працювати в аналоговому режимі.

У рядку 77 ми вмикаємо таймер 0 з тактовою частотою $clk_{I/O}/64$ (див. табл. 3.9), що в нашому випадку дорівнює 1000000 / 64 = 15625 Гц. Оскільки таймер 0 є восьмибітним, то період між його переповненнями буде дорівнювати 1/15625 * 256 ≈ 16 мс. Цей таймер в нашій програмі буде виконувати дві функції – запускати вимірювання АЦП та перемикати розряди семисегментного індикатора для реалізації динамічної індикації.

Для трирозрядного індикатора повний цикл буде дорівнювати 3 * 16 = 48 мс, що відповідає частоті приблизно 20 Гц. Це частота, що знаходиться на межі сприйняття, і в реальному пристрої вже може бути помітно мерехтіння розрядів, але в симуляторі все працює рівно, то ж ми залишимо це значення.

У рядках 80, 81 ми налаштовуємо аналоговий компаратор. Спочатку ми відключаємо мультиплексор аналогового компаратора (рядок 80), оскільки в цій програмі ми також будемо використовувати і АЦП. Це робиться шляхом запису 0 у біт ACME регістра SFIOR. Насправді, можна було б цього і не робити, оскільки при вмиканні АЦП мультиплексор автоматично підключається до нього.

У рядку 81 ми встановлюємо тільки біт ACIE у регістрі ACSR, залишаючи всі інші біти нулями, тому налаштування будуть наступні:

- ACD = 0, компаратор ввімкнений.
- ACBG = 0, в якості позитивного сигналу компаратора використовується вхід AIN0.
- ACIE = 1, переривання від аналогового компаратора ввімкнено.
- ACIC = 0, захоплення таймера/лічильника 1 від аналогового компаратора вимкнено.
- ACIS1 = 0, ACIS0 = 1, переривання від аналогового компаратора відбувається при будь-якій зміні стану виходу.

У рядках 83, 84 ми налаштовуємо АЦП. Спочатку ми обираємо вхідний канал та опорну напругу за допомогою регістру ADMUX. В якості опорної напруги задамо AVCC, оскільки ми підключали напругу до цього виводу (рис. 1), для цього треба записати 1 у біт REFS0. Далі обираємо в якості початкового каналу ADC0, до якого підключене одне з джерел напруги (рис. 1). Тут одразу можна було б перевірити вихід компаратора та обрати або канал ADC0, або ADC1. Біт ADLAR цього регістру залишимо рівним 0, то ж результат буде вирівняно праворуч.

У рядку 84 ми задаємо параметри власне модуля АЦП за допомогою регістру ADCSRA:

- ADEN = 1, вмикаємо модуль АЦП.
- ADSC = 0, поки не починаємо перетворення АЦП.
- ADFR = 0, режим одноразового перетворення.
- ADIE = 1, дозволяємо переривання по закінченню перетворення АЦП.
- ADPS2 = 0, APDS1 = 1, ADPS0 = 1, задаємо дільник частоти АЦП рівним 8. При цьому тактова частота АЦП буде дорівнювати 1000000 Гц / 8 = 125 кГц, що входить у межі дозволених частот АЦП 50 – 200 кГц.

У рядку 86 ми встановлюємо значення змінної digit рівним 0, щоб почати індикацію з першої цифри.

Тепер лишилося тільки дозволити глобальні переривання за допомогою функції sei (рядок 88).

Головний цикл програми цього разу буде порожній (рядок 90), оскільки всі дії будуть виконуватися у підпрограмах обробки переривань.

Всі ці підпрограми, як вже було зазначено у попередніх практичних роботах, мають назву ISR, а в якості їх аргументу передається назва переривання, яку можна знайти у заголовковому файлі для кожного конкретного мікроконтролера або у таблиці 3.1, замінивши пробіли на знак підкреслення, і додавши в кінці «_vect».

У рядках 33-55 знаходиться функція обробки переривання по переповненню таймера 0. Як вже було зазначено раніше, в цьому перериванні ми будемо запускати перетворення АЦП та виконувати процедуру динамічної індикації. Для початку перетворення АЦП треба записати 1 у біт ADSC регістра ADCSRA (рядок 35). Процедура динамічної індикації майже не відрізняється від тих, що ми вже використовували у минулих роботах за декількома винятками. По-перше, тепер нам треба вказувати напругу з точністю 2 знаки після коми, то ж після першої цифри треба ввімкнути десяткову крапку. Це ми робимо додаванням сегменту DP до регістру PORTB при вмиканні першого розряду (рядок 40). По-друге, тепер не потрібно додавати затримку після вмикання кожного розряду, оскільки ця затримка тепер задається таймером. Все інше залишається таким самим, то ж ми не будемо розглядати цей алгоритм більш детально тут.

У рядках 57-62 знаходиться функція обробки переривання по завершенню перетворення АЦП, вектор якого називається просто ADC_vect. Після закінчення перетворення треба зберегти його результат. Оскільки результат вирівняний праворуч, треба зчитати обидва його регістри – ADCH та ADCL. Починати зчитувати треба завжди з регістра ADCL, щоб зафіксувати регістр ADCH і запобігти його перезапису під час зчитування.

То ж у рядку 59 ми декларуємо локальну змінну res і спочатку присвоюємо їй значення регістру ADCL. А у рядку 60 ми додаємо до цієї змінної значення регістру ADCH, зсунуте вліво на 8 біт. Таким чином, у змінній res після цих двох рядків буде збережено результат перетворення у 10-бітному форматі, тобто число від 0 до 1023. Тепер ми повинні перетворити це число на напругу у діапазоні від 0 до 5 В з роздільною здатністю 0,01 В. Оскільки десяткову крапку у числі, що виводиться, ми вже поставили (рядок 40), напруга буде уявляти собою значення від 0 до 500. Згідно з (5.1), результат перетворення визначається за наступною формулою:

$$ADC = \frac{V_{IN} \cdot 1023}{V_{REF}}$$

де V_{IN} — напруга на вибраному вхідному контакті, а V_{REF} — вибрана опорна напруга. 0x000 представляє напругу землі, а 0x3FF представляє вибрану опорну напругу мінус один LSB.

Звідси

$$V_{IN} = \frac{V_{REF} \cdot ADC}{1023} \quad (5.2)$$

Тобто для отримання вхідної напруги треба взяти результат перетворення, помножити його на опорну напругу, яка в нашому випадку дорівнює 5 В, або 500 х 0,01 В, і поділити на 1023. Це ми й робимо у рядку 61. Перетворене значення зберігаємо у змінній number, яка виводиться на індикатор. Зверніть увагу, що у рядку 61 спочатку виконується операція множення, а потім поділення. Це важливо, оскільки, якщо взяти і одразу поділити результат перетворення на 1023, то ми завжди отримуватимемо 0, оскільки це є цілочисельне ділення.

У рядках 64-70 знаходиться функція обробки переривання по зміні виходу компаратора, вектор якого називається ANA_COMP_vect. У цій функції ми спочатку перевіряємо, чому дорівнює значення біту ACO регістру ACSR (рядок 64). Цей біт уявляє собою вихід аналогового компаратора. Якщо він дорівнює 1, це значить, що напруга на позитивному вході AIN0 більша за напругу на негативному вході AIN1, і тому ми повинні зчитувати значення з джерела напруги, що підключене до входу AIN0 та ADC0, тобто встановити значення бітів MUX3…0 мультиплексора АЦП ADMUX рівними 0, що ми і робимо у рядку 67. Якщо ж значення ACO дорівнює 0, це значить, що більша напруга на негативному вході AIN1, і тому ми повинні зчитувати значення з джерела напруги, що підключене до входу AIN1 та ADC1. Для цього треба встановити біт MUX0 у 1 відповідно до таблиці 5.4 (рядок 69).

На цьому опис програми можна вважати завершеним. Тепер треба перевірити, як вона працює.

5.4.2 Симуляція роботи програми

Після компіляції програми та завантаження її у мікроконтролер на індикаторі з’явиться напруга джерела, що підключене до входу ADC0 (рис. 5.16).

Значення на індикаторі може бути на 0,01 В менше, ніж напруга, яку ми задаємо у регульованому джерелі напруги. Якщо тепер покрутити регулятори обох джерел, можна побачити, що на індикаторі відображається більша з двох напруг (рис. 5.17, 5.18), таким чином, наша програма працює вірно.

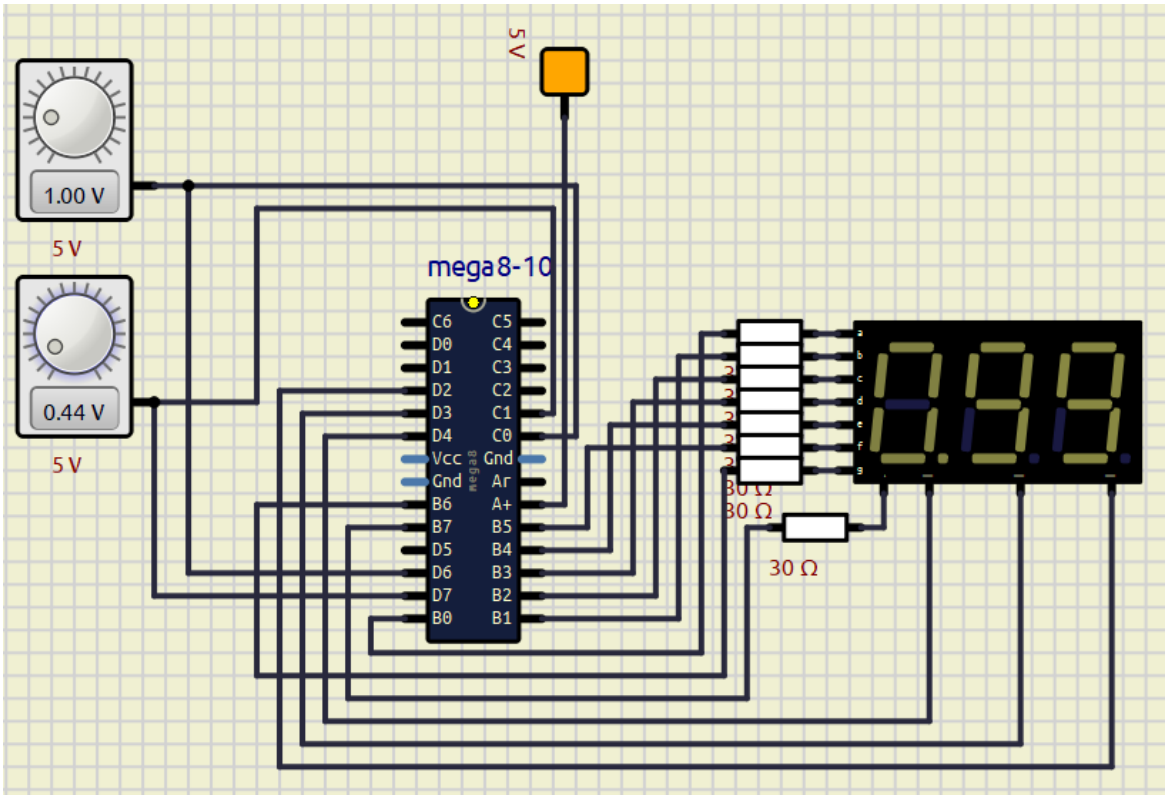


Рисунок 5.16 – Результат роботи програми

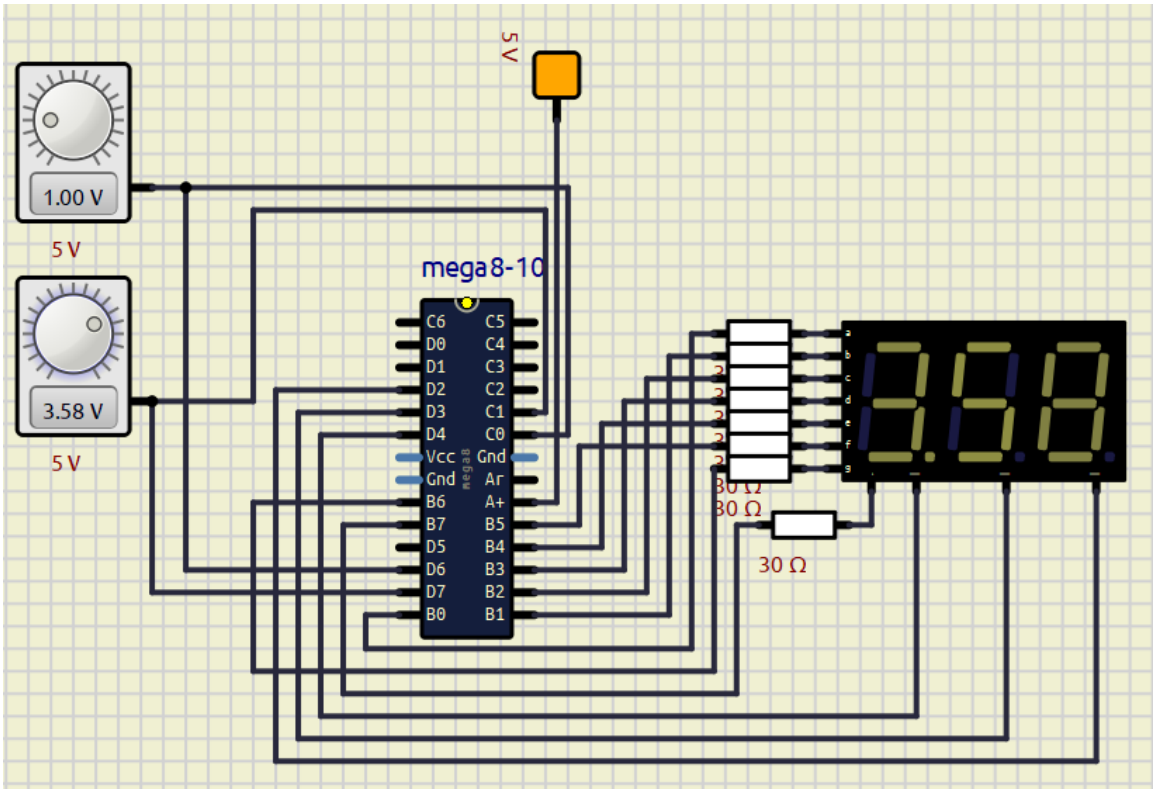


Рисунок 5.17 – Друга напруга більша за першу

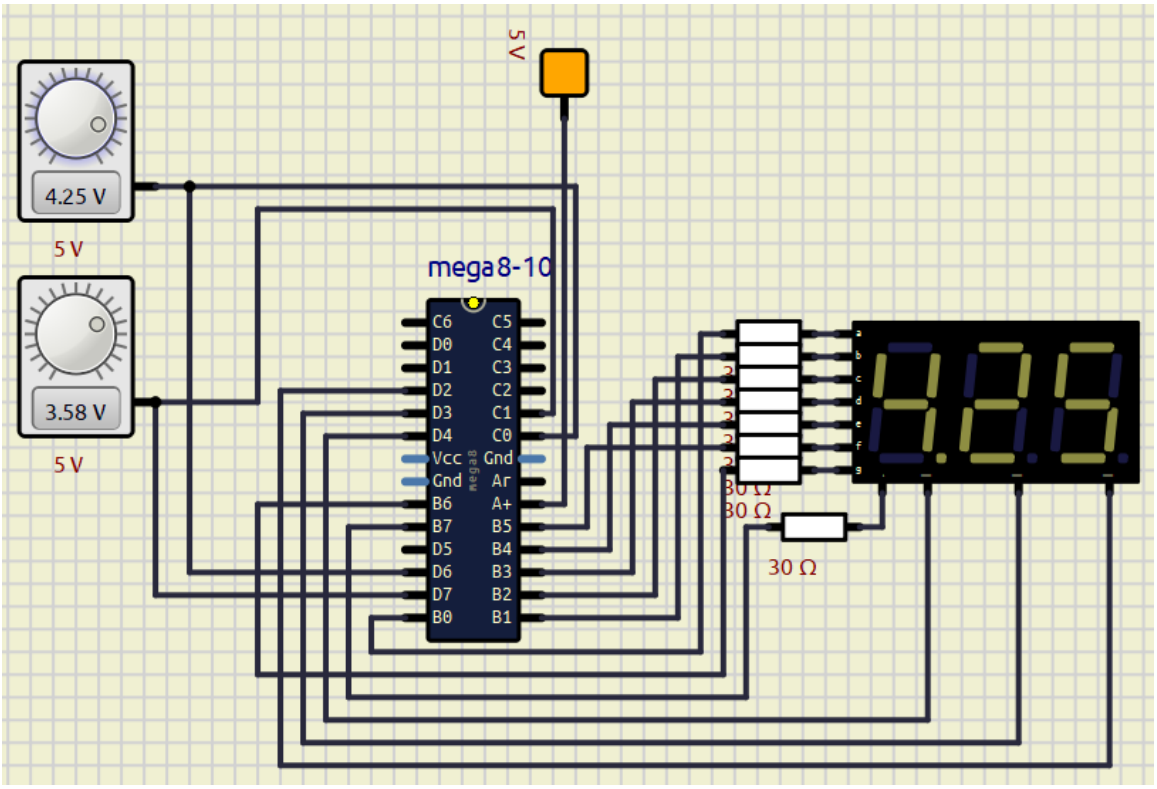


Рисунок 5.18 – Перша напруга більша за другу

5.4.3 Завдання до практичної роботи

1. Створити принципову електричну схему, згідно з варіантом завдань (табл. 5.6).
2. Написати програму, що виконує поставлене завдання.
3. Створити файл прошивки, загрузити його у мікроконтролер та переконатися, що все працює, як треба.

Таблиця 5.6 – Варіанти завдань до практичної роботи №5

Варіант	Завдання
1	Підключити дворозрядний семисегментний індикатор зі спільним анодом наступним чином: A – PB0, B – PB1, C – PB2, D – PB3, E – PB4, F – PB5, G – PB6, перший розряд – PC0, другий розряд – PC1. Налаштувати аналоговий компаратор, щоб позитивний вхід був підключений до внутрішнього джерела опорної напруги забороненої зони (2.56 В), а негативний вхід – до виводу AIN1. Якщо напруга на вході AIN1 більша за опорну, видавати на індикатор слово HI, в протилежному випадку видавати слово LO.
2	Підключити світлодіод до виводу PB2. Налаштувати аналоговий компаратор, щоб позитивний вхід був підключений до виводу AIN0, а негативний – до виводу ADC1 через мультиплексор АЦП. Якщо вихід компаратора дорівнює 1, то плавно (протягом 3-4 секунд) підвищувати яскравість світлодіода від 0 до максимального значення. У протилежному випадку так само плавно знижувати яскравість від максимуму до 0. Керувати яскравістю за допомогою модуля порівняння

Варіант	Завдання
	таймера/лічильника 1. Режими та частоту роботи таймера обрати самостійно.
3	Підключити світлодіод до виводу PB1. Налаштувати аналоговий компаратор, щоб позитивний вхід був підключений до внутрішнього джерела опорної напруги забороненої зони (2.56 В), а негативний – до виводу ADC4 через мультиплексор АЦП. Якщо напруга на вході AIN1 менша за опорну, встановити яскравість світлодіода на рівні 70%, а якщо більша, то на рівні 20%. Керувати яскравістю за допомогою модуля порівняння таймера/лічильника 1. Режими та частоту роботи таймера обрати самостійно.
4	Підключити світлодіод до виводу PB1. Налаштувати аналоговий компаратор, щоб позитивний вхід був підключений до виводу AIN0, а негативний – до виводу ADC3 через мультиплексор АЦП. Якщо вихід компаратора дорівнює 1, миготіти світлодіодом з частотою 4 Гц. У протилежному випадку частоту миготіння знизити до 1 Гц. Керувати частотою миготіння світлодіода за допомогою модуля порівняння таймера/лічильника 1. Режими та частоту роботи таймера обрати самостійно.
5	Підключити однорозрядний семисегментний індикатор зі спільним катодом наступним чином: A – PB7, B – PB6, C – PB5, D – PB4, E – PB3, F – PB2, G – PB1. Катод підключити просто на землю. Налаштувати аналоговий компаратор, щоб позитивний вхід був підключений до виводу AIN0, а негативний – до виводу ADC0 через мультиплексор АЦП. Якщо вихід компаратора дорівнює 1, виводити на індикатор цифру 1, а в протилежному випадку виводити цифру 0.
6	Підключити трирозрядний семисегментний індикатор зі спільним катодом наступним чином: A – PB0, B – PB1, C – PB2, D – PB6, E – PB5, F – PB4, G – PB3, перший розряд – PD0, другий розряд – PD1, третій розряд – PD2. Сконфігурувати таймер 1, щоб він генерував переривання кожну секунду. Сконфігурувати АЦП наступним чином: вхід – ADC3, вирівнювання результату – ліворуч, опорна напруга – внутрішня величиною 2,56 В, одиночне перетворення. Підключити до входу АЦП регульоване джерело напруги з максимальною напругою 2,56 В. При спрацюванні таймера запускати перетворення АЦП. На індикатор виводити результат перетворення з точністю 8 біт, тобто число в діапазоні від 0 до 255.
7	Підключити трирозрядний семисегментний індикатор зі спільним анодом наступним чином: A – PB0, B – PB7, C – PB1, D – PB6, E – PB2, F – PB5 G – PB3, перший розряд – PD6, другий розряд – PD4, третій розряд – PD0. Підключити кнопки до виводів PD2 (INT0) та PD3 (INT1). Натискання кнопок обробляти за допомогою переривання. Сконфігурувати АЦП наступним чином: два входи – ADC1 або ADC2, вирівнювання результату – праворуч, опорна напруга – AVCC величиною 5 В, одиночне перетворення. Підключити до входів АЦП регульовані джерела напруги з максимальною напругою 5 В. При натисканні на кнопку INT0 виводити на індикатор напругу джерела, підключеного до виводу ADC1, а при натисканні на кнопку INT1 – напругу джерела, підключеного до виводу ADC2. Результат виводити у вольтах з двома знаками після коми.

Варіант	Завдання
8	Підключити світлодіод до виводу PB2. Сконфігурувати АЦП наступним чином: вхід – ADC6, вирівнювання результату – ліворуч, опорна напруга – AVCC величиною 5 В, безперервне перетворення. Підключити до входу АЦП регульоване джерело напруги з максимальною напругою 5 В. Встановлювати яскравість світлодіода пропорційною напрузі на вході АЦП, тобто при 0 В на вході світлодіод не горить, а при максимальній напрузі – горить з повною потужністю. Керувати яскравістю за допомогою модуля порівняння таймера/лічильника 1. Режими та частоту роботи таймера обрати самостійно.
9	Підключити світлодіоди до виводів PB0 та PB1. Сконфігурувати АЦП наступним чином: два входи – ADC0 або ADC1, вирівнювання результату – ліворуч, опорна напруга – внутрішня величиною 2,56 В, одиночне перетворення. Підключити до входів АЦП регульовані джерела напруги з максимальною напругою 2,56 В. Вимірювати по черзі напруги від обох джерел з частотою 10 вимірювань на секунду. Якщо напруга на вході ADC0 більша за напругу на вході ADC1, вмикати перший світлодіод і вимикати другий, якщо напруга на вході ADC0 менша за напругу на вході ADC1, вмикати другий світлодіод і вимикати перший, якщо напруга на обох входах однакова, вмикати обидва світлодіода. Для порівняння використовувати тільки старші 8 біт результату.
10	Підключити дворозрядний семисегментний індикатор зі спільним анодом наступним чином: A – PD5, B – PD6, C – PD0, D – PD1, E – PD3, F – PD4 G – PD2, перший розряд – PB5, другий розряд – PB7. Сконфігурувати таймер 1, щоб він генерував переривання два рази на секунду. Сконфігурувати АЦП наступним чином: вхід – ADC5, вирівнювання результату – ліворуч, опорна напруга – AVCC величиною 5 В, одиночне перетворення. Підключити до входу АЦП регульоване джерело напруги з максимальною напругою 5 В. При спрацюванні таймера запускати перетворення АЦП. На індикатор виводити результат перетворення з точністю 8 біт у шістнадцятковій формі, тобто число в діапазоні від 00 до FF.
11	Підключити вісім світлодіодів до виводів PB0-PB7. Сконфігурувати АЦП наступним чином: вхід – ADC4, вирівнювання результату – ліворуч, опорна напруга – AVCC величиною 5 В, безперервне перетворення. Підключити до входу АЦП регульоване джерело напруги з максимальною напругою 5 В. За допомогою світлодіодів виводити результат перетворення з точністю 8 біт у двійковій формі, тобто число в діапазоні від 00000000 до 11111111, де 1 відповідає ввімкненому світлодіоду, а 0 – вимкненому.
12	Підключити вісім світлодіодів до виводів PD0-PD7. Сконфігурувати таймер 0, щоб він генерував 10-20 переривань на секунду. Сконфігурувати АЦП наступним чином: вхід – ADC3, вирівнювання результату – праворуч, опорна напруга – внутрішня величиною 2,56 В, одиночне перетворення. Підключити до входу АЦП регульоване джерело напруги з максимальною напругою 2,56 В. При спрацюванні таймера запускати перетворення АЦП. За допомогою світлодіодів індикувати результат перетворення у вигляді гістограми, тобто при напрузі 0 В всі світлодіоди погашені, при напрузі від 0 до 1/8 від максимальної ввімкнути один нижній світлодіод, при напрузі від 1/8 до 2/8 від максимальної ввімкнути два світлодіоди і т.д.

5.5 Питання для самоперевірки

1. Принцип роботи аналогового компаратора мікроконтролера ATMega8.
2. До чого можна підключати входи и вихід аналогового компаратора?
3. Принцип роботи АЦП мікроконтролера ATMega8.
4. Які є режими роботи АЦП?
5. Як можна виводити результат перетворення АЦП?

5.6 Перелік рекомендованих джерел

1. ATMega8 : технічна документація на мікроконтролер. URL:: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2486-8-bit-AVR-microcontroller-ATmega8_L_datasheet.pdf (дата звернення: 02.07.2024).
2. 8-bit AVR® MCUs : інформація про мікроконтролери. URL: https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/8-bit-mcus/avr-mcus (дата звернення: 02.07.2024).
3. Конспект лекцій з дисципліни «Мікропроцесорна техніка» для здобувачів вищої освіти першого (бакалаврського) рівня зі спеціальності 153 «Мікро-та наносистемна техніка» за освітньо-професійною програмою «Мікро-та наносистемна техніка» та зі спеціальності 171 «Електроніка» за освітньо-професійною програмою «Електроніка» / уклад. О. М. Гулєша. Кам‘янське : ДДТУ, 2020 р. 57 с.
4. Основи Програмування AVR C. DevZone. URL: https://devzone.org.ua/post/osnovi-programuvannia-avr-c (дата звернення: 02.07.2024).

Навчально-методичне видання

Сокол Сергій Петрович
Койфман Олексій Олександрович

ЕЛЕКТРОНІКА ТА МІКРОПРОЦЕСОРНА ТЕХНІКА

методичні вказівки до виконання практичних робіт

Самостійне електронне мережеве видання

Публікується в авторський редакції