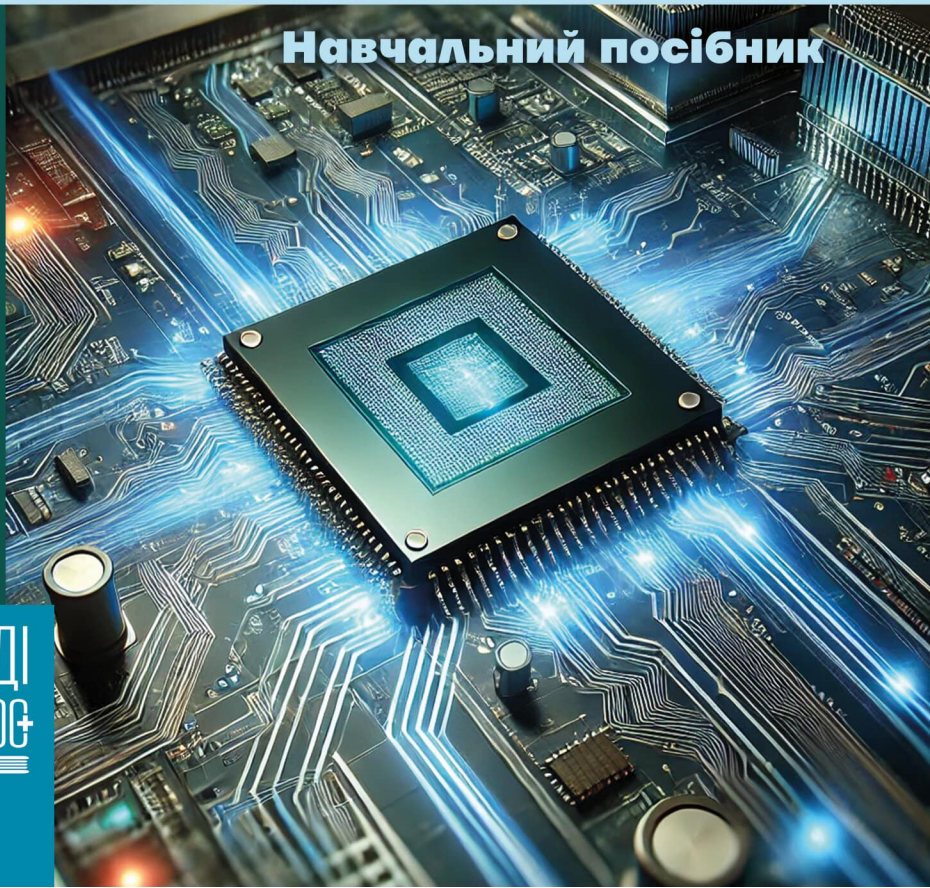
metinvest polytechnic

С.П. Сокол, О.О. Койфман, А.Б. Ісаєв, В.І. Мірошниченко

ЕЛЕКТРОНІКА ТА МІКРОПРОЦЕСОРНА ТЕХНІКА: ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ AVR

Навчальний посібник



ОЛДІ ПЛЮС

ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«МЕТІНВЕСТ ПОЛІТЕХНІКА»

ЕЛЕКТРОНІКА ТА МІКРОПРОЦЕСОРНА ТЕХНІКА: ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ AVR

Навчальний посібник

Одеса • 2025 • Олді+

УДК 004.896(075.8)
Е50

Автори:
С. П. Сокол, О. О. Койфман, А. Б. Ісаєв, В. І. Мірошниченко

Рецензенти:

О. В. Бондар, начальник відділу автоматизації Управління проєктуванням ТОВ «МЕТІНВЕСТ СІЧСТАЛЬ»;

О. В. Разживін, кандидат технічних наук, доцент, доцент кафедри автоматизації, електро- та робототехнічних систем, ТОВ «Технічний університет «Метінвест Політехніка»;

О. Є. Марков, доктор технічних наук, професор, завідувач кафедри автоматизації виробничих процесів, Донбаська державна машинобудівна академія

Робота друкується за рішенням Вченої ради
ТОВ «Технічний університет «Метінвест Політехніка»
(протокол № 5 від 30.01.2025 р.)

Е50 **Електроніка** та мікропроцесорна техніка: програмування мікроконтролерів AVR : навчальний посібник / С. П. Сокол, О. О. Койфман, А. Б. Ісаєв, В. І. Мірошниченко ; ТОВ «Технічний університет «Метінвест Політехніка». – Одеса : Олді+, 2025. – 428 с.

ISBN 978-966-289-960-3

У посібнику розглянуто загальні відомості про мікропроцесорні системи, архітектуру мікроконтролерів сімейства AVR, їх цифрові порти вводу-виводу, системний скид, переривання, а також робочі режими таймерів. Наведено загальну інформацію про асинхронно-синхронний приймач-передавач USART, послідовні синхронні інтерфейси SPI та I^2C, аналогові модулі, а також наведено формули для розрахунку швидкості обміну даними та похибки передачі, приклади налаштування мікроконтролерів сімейства AVR, програмування пам'яті за допомогою послідовного інтерфейсу. Надано методичні поради та рекомендації щодо порядку виконання практичних робіт з дисципліни «Електроніка та мікропроцесорна техніка» на прикладі мікроконтролерів AVR, наведено приклади програм та варіанти завдань для самостійного виконання. Розглянуто використання програми SimulIDE для програмування та симуляції роботи мікроконтролерів. У практичних роботах показано використання більшості можливостей мікроконтролерів AVR таких як: цифрові порти вводу-виводу, переривання, таймери, послідовні цифрові інтерфейси, аналогові модулі. Набуті навички можуть бути використані при програмуванні мікроконтролерів інших моделей і виробників.

Посібник призначений для здобувачів вищої освіти всіх форм навчання, зокрема за спеціальністю G7 (174) «Автоматизація, комп'ютерно-інтегровані технології та робототехніка», та може використовуватись інженерно-технічним персоналом при розробці та обслуговуванні автоматизованих систем керування.

УДК 004.896(075.8)

ISBN 978-966-289-960-3

ЗМІСТ

ПЕРЕДМОВА 8

1 ЗАГАЛЬНІ ВІДОМОСТІ ПРО МІКРОПРОЦЕСОРНІ СИСТЕМИ 10

1.1 Конструкція та принцип роботи мікроконтролерів 10
1.2 Архітектура процесорів 15
1.3 Типи пам'яті мікроконтролерів 21
Контрольні запитання до теми 1 26
Використана література 27

2 АРХІТЕКТУРА МІКРОКОНТРОЛЕРІВ СІМЕЙСТВА AVR 28

2.1 Основні характеристики мікроконтролерів AVR на прикладі МК ATMega8 28
2.2 Коротка характеристика архітектури 32
2.3 Регістровий файл 36
2.4 Арифметико-логічний пристрій (АЛУ) 37
2.5 Доступ до пам'яті та виконання інструкцій 37
2.6 Пам'ять вводу-виводу 38
2.7 EEPROM 40
2.8 Порти вводу-виводу 41
2.9 Таймер 42
2.10 Система переривань 43
2.11 Вбудований сторожовий таймер (Watchdog) 46
2.12 Режими роботи з низьким споживанням енергії 47
Контрольні запитання до теми 2 48
Використана література 49

3 ЦИФРОВІ ПОРТИ ВВОДУ-ВИВОДУ МІКРОКОНТРОЛЕРІВ СІМЕЙСТВА AVR 51

3.1 Конфігурація виводів 51
3.2 Читання стану виводу 55

3.3 Альтернативні функції портів вводу-виводу . . . 58
3.4 Опис регістрів вводу-виводу . . . 65
3.5 Бітові операції . . . 66
Контрольні запитання до теми 3 . . . 72
Використана література . . . 73

4 СИСТЕМНИЙ СКИД, ПЕРЕРИВАННЯ ТА РОБОЧІ РЕЖИМИ МІКРОКОНТРОЛЕРІВ СІМЕЙСТВА AVR . . . 74

4.1 Система тактування . . . 74
4.2 Режими роботи мікроконтролера . . . 77
4.3 Системний скид . . . 82
4.4 Сторожовий таймер . . . 84
4.5 Переривання . . . 87
Контрольні запитання до теми 4 . . . 94
Використана література . . . 95

5 ТАЙМЕРИ МІКРОКОНТРОЛЕРІВ СІМЕЙСТВА AVR . . . 96

5.1 8-бітний таймер-лічильник 0 мікроконтролера ATMega8 . . . 96
5.2 16-бітний таймер-лічильник 1 мікроконтролера ATMega8 . . . 103
Контрольні запитання до теми 5 . . . 133
Використана література . . . 134

6 АСИНХРОННО-СИНХРОННИЙ ПРИЙМАЧ-ПЕРЕДАВАЧ USART . . . 135

6.1 Загальні відомості про інтерфейс UART . . . 135
6.2 Апаратна частина UART мікроконтролерів AVR . . . 139
6.3 Регістри вводу-виводу модуля USART . . . 143
6.4 Приклади налаштування швидкості передачі даних . . . 150
Контрольні запитання до теми 6 . . . 152
Використана література . . . 153

7 ПОСЛІДОВНІ СИНХРОННІ ІНТЕРФЕЙСИ SPI ТА I^2C . . . 154

7.1 Загальні відомості про інтерфейс SPI . . . 154
7.2 Регістри вводу-виводу модуля SPI . . . 159

7.3 Загальні відомості про інтерфейс I^2C . . . 164
7.4 Модуль TWI мікроконтролерів AVR . . . 170
7.5 Регістри вводу-виводу модуля TWI . . . 175
Контрольні запитання до теми 7 . . . 202
Використана література . . . 203

8 АНАЛОГОВІ МОДУЛІ МІКРОКОНТРОЛЕРІВ СІМЕЙСТВА AVR . . . 204

8.1 Аналоговий компаратор . . . 204
8.2 Загальні відомості про АЦП . . . 208
8.3 Модуль АЦП мікроконтролерів AVR . . . 211
Контрольні запитання до теми 8 . . . 226
Використана література . . . 227

9 ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ СІМЕЙСТВА AVR . . . 228

9.1 Пам'ять мікроконтролерів AVR . . . 228
9.2 Самопрограмування пам'яті . . . 236
9.3 Біти фьюзів та блокування пам'яті . . . 252
9.4 Програмування пам'яті за допомогою послідовного інтерфейсу . . . 254
Контрольні запитання до теми 9 . . . 258
Використана література . . . 259

10 ПРАКТИЧНА РОБОТА № 1 «СИМУЛЯЦІЯ СХЕМ З МІКРОКОНТРОЛЕРАМИ З ВИКОРИСТАННЯМ ПРОГРАМИ SIMULIDE» . . . 260

10.1 Завдання . . . 260
10.2 Теоретичні дані . . . 261
10.2.1 Призначення програми SimulIDE . . . *261*
10.2.2 Встановлення необхідних програм та засобів . . . *262*
10.2.3 Подання на екрані та основні елементи керування . . . *265*
10.2.4 Основні прийоми створення та коригування схеми . . . *268*
10.2.5 Написання програмного коду . . . *276*
10.2.6 Компіляція файлу прошивки та запуск симуляції . . . *280*
10.3 Завдання до практичної роботи . . . 284
Питання для самоперевірки . . . 285
Перелік рекомендованих джерел . . . 285

11 ПРАКТИЧНА РОБОТА № 2 «ПРОГРАМУВАННЯ ЦИФРОВИХ ПОРТІВ ВВОДУ-ВИВОДУ МІКРОКОНТРОЛЕРА AVR» 286
11.1 Завдання 286
11.2 Теоретичні дані 287
11.2.1 Робота з портами вводу-виводу мікроконтролерів AVR 287
11.2.2 Бітові операції на мові С 289
11.2.3 Бітові операції 290
11.2.4 Очищення та встановлення бітів 292
11.2.5 Макрос керування значенням біту _BV() 293
11.2.6 Перевірка значення біту 294
11.2.7 Приклад програми 295
11.2.8 Принципова електрична схема 296
11.2.9 Програмний код 297
11.3 Завдання до практичної роботи 301
Питання для самоперевірки 304
Перелік рекомендованих джерел 304
12 ПРАКТИЧНА РОБОТА № 3 «ПРОГРАМУВАННЯ ТАЙМЕРІВ МІКРОКОНТРОЛЕРА AVR» 306
12.1 Завдання 306
12.2 Теоретичні дані 306
12.2.1 Переривання 306
12.2.2 8-бітний таймер-лічильник 0 мікроконтролера ATMega8 313
12.3 Приклад програми 346
12.3.1 Принципова електрична схема 346
12.3.2 Програмний код 347
12.4 Завдання до практичної роботи 355
Питання для самоперевірки 358
Перелік рекомендованих джерел 359

13 ПРАКТИЧНА РОБОТА № 4 «ПРОГРАМУВАННЯ ЦИФРОВИХ ІНТЕРФЕЙСІВ МІКРОКОНТРОЛЕРА AVR» 360
13.1 Завдання 360
13.2 Теоретичні дані 361
13.2.1 Загальні відомості про інтерфейс UART 361
13.2.2 Апаратна частина UART мікроконтролерів AVR 363
13.2.3 Регістри вводу-виводу модуля USART 367
13.2.4 Приклади налаштування швидкості передачі даних 374
13.3 Приклад програми 377
13.3.1 Принципова електрична схема 377
13.3.2 Програмний код 381
13.3.3 Симуляція роботи програми 389
13.4 Завдання до практичної роботи 392
Питання для самоперевірки 395
Перелік рекомендованих джерел 395
14 ПРАКТИЧНА РОБОТА № 5 «ПРОГРАМУВАННЯ АНАЛОГОВИХ МОДУЛІВ МІКРОКОНТРОЛЕРА AVR» 397
14.1 Завдання 397
14.2 Теоретичні дані 398
14.2.1 Аналоговий компаратор 398
14.2.2 Модуль АЦП мікроконтролерів AVR 402
14.3 Приклад програми з використанням аналогового компаратора та аналого-цифрового перетворювача 414
14.4 Принципова електрична схема 415
14.4.1 Програмний код 417
14.4.2 Симуляція роботи програми 422
14.5 Завдання до практичної роботи 424
Питання для самоперевірки 427
Перелік рекомендованих джерел 427

ПЕРЕДМОВА

Електроніка та мікропроцесорна техніка – фундаментальна навчальна дисципліна, яка забезпечить наявність необхідних знань для вирішення практичних задач у процесі інженерної діяльності, яка пов'язана з розробкою принципових електричних схем різноманітних приладів та систем, а також програмного забезпечення для мікроконтролерів. Посібник призначений для набуття студентами знань та вмінь щодо конструювання та принципів дії сучасних електронних компонентів, базових схем аналогової та цифрової електроніки, основ булевої алгебри та комбінаторної логіки, сучасних підходів до аналізу і синтезу електронних пристроїв, програмування мікроконтролерів з використанням мов високого рівня. Особливістю посібника є акцент на саме практичному використанні сучасних програмних засобів створення та моделювання електронних схем, проте будуть надані й необхідні теоретичні знання, що дозволять самостійно конструювати різноманітні електронні пристрої, синтезувати та розробляти системи автоматичного управління. Отримані знання будуть корисними для проектування систем автоматизації як побутового, так і промислового рівня.

Основні результати навчання, які передбачаються:

– здатність застосовувати знання електроніки і мікропроцесорної техніки, в обсязі, необхідному для розуміння процесів в системах автоматизації та комп'ютерно-інтегрованих технологіях;

– здатність обґрунтовувати вибір технічної структури мікропроцесорних систем;

– спроможність за допомогою сучасних інформаційних технологій до самостійного пошуку, аналізу та вибору необхідної інформації для оптимального розв'язання інженерних завдань, програмувати та використовувати прикладні та спеціалізовані комп'ютерно-інтегровані середовища для вирішення задач автоматизації;

– спроможність застосовувати сучасні інформаційні технології для розробки електричних схем та програмування мікроконтролерів з використанням мов високого рівня.

В посібнику розглянуто загальні відомості про мікропроцесорні системи, архітектуру мікроконтролерів сімейства AVR, їхні цифрові порти вводу-виводу, системний скид, переривання, робочі режимів таймерів. Надано загальну інформацію про асинхронно-синхронний приймач-передавач USART, послідовні синхронні інтерфейси SPI та I^2C, аналогові модулі, а також формули для розрахунку швидкості обміну даними та похибки передачі, приклади налаштування мікроконтролерів сімейства AVR, програмування пам'яті за допомогою послідовного інтерфейсу.

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.

1 ЗАГАЛЬНІ ВІДОМОСТІ ПРО МІКРОПРОЦЕСОРНІ СИСТЕМИ

Метою вивчення теми є ознайомлення з мікроконтролерами, їх типами пам'яті та архітектурою процесорів.

Завдання вивчення теми збігаються з переліком питань для розгляду, що наведений нижче.

Перелік питань до розділу:

1.1. Конструкція та принцип роботи мікроконтролерів.
1.2. Архітектура процесорів.
1.3. Типи пам'яті мікроконтролерів.

Мікроконтролер – це компактна інтегральна схема, призначена для керування певною операцією у вбудованій системі. Типовий мікроконтролер включає в себе процесор, пам'ять і периферійні пристрої вводу-виводу (I/O) на одній мікросхемі.

Мікроконтролери, які іноді називають вбудованим контролером або просто мікроконтролером (MCU), використовуються серед інших пристроїв у системах керування автомобільними двигунами, роботах, офісних машинах, медичних пристроях, мобільних радіостанціях, торгових автоматах і побутовій техніці. Це прості мініатюрні ПК, призначені для керування невеликими функціями більшого компонента без складної зовнішньої операційної системи.

1.1 Конструкція та принцип роботи мікроконтролерів

Мікроконтролер вбудовано в систему для виконання однієї функції в пристрої. Він використовує свій центральний процесор для інтерпретації даних, які він отримує від периферійних пристроїв введення-виведення. Інформація, яку отримує мікроконтролер,

тимчасово зберігається в його пам'яті даних, де процесор отримує до неї доступ і використовує інструкції, що зберігаються в пам'яті програм, щоб розшифрувати та застосувати вхідні дані. Потім він використовує свої периферійні пристрої вводу-виводу для зв'язку та виконання відповідних дій.

Мікроконтролери використовуються в ряді систем і пристроїв. Пристрої часто використовують кілька мікроконтролерів, які працюють разом для виконання відповідних завдань.

Наприклад, автомобіль має багато мікроконтролерів, які керують різними окремими системами, такими як антиблокувальна система гальм, контроль тяги, упорскування палива та керування підвіскою. Кожен мікроконтролер спілкується з іншими, щоб повідомити їм про правильні дії. Деякі можуть спілкуватися з більш складним центральним комп'ютером у автомобілі, а інші можуть спілкуватися лише з іншими мікроконтролерами. Вони надсилають і отримують дані за допомогою своїх периферійних пристроїв вводу-виводу та обробляють ці дані для виконання призначених завдань.

Основними елементами, з яких складається мікроконтролер, є центральний процесор (CPU, ЦП), пам'ять і периферійні пристрої вводу-виводу.

ЦП. Також відомий як процесор. Центральний процесор є мозком пристрою. Він обробляє та відповідає на різні інструкції, які керують функціями мікроконтролера. Це передбачає виконання основних арифметичних, логічних операцій і операцій вводу-виводу. Він також виконує операції передачі даних, які передають команди іншим компонентам у більшій вбудованій системі.

Пам'ять. Пам'ять мікроконтролера зберігає дані, які отримує процесор і використовує для відповіді на інструкції, на які він запрограмований. Мікроконтролер має два основних типи пам'яті:

1. Програмна пам'ять. Вона зберігає довготривалу інформацію про інструкції, які виконує ЦП. Пам'ять програм є енергонезалежною пам'яттю, тобто вона зберігає інформацію протягом тривалого часу без потреби в джерелі живлення.

2. Пам'ять даних. Це тимчасове сховище даних використовується під час виконання інструкцій. Пам'ять даних є енергонезалежною, тобто дані, які в ній зберігаються, є тимчасовими

та зберігаються, лише якщо пристрій підключено до джерела живлення.

Периферійні пристрої вводу-виводу. Пристрої вводу-виводу є для процесора інтерфейсом із зовнішнім світом. Вхідні порти отримують інформацію і передають її процесору у вигляді двійкових даних. Процесор отримує ці дані та надсилає необхідні інструкції до пристроїв виведення, які виконують зовнішні завдання щодо мікроконтролера.

Інші елементи. Хоча процесор, пам'ять і периферійні пристрої вводу-виводу є визначальними елементами мікропроцесора, є й інші елементи, які часто включають в мікроконтролери. Термін «периферійний пристрій вводу-виводу» означає допоміжний компонент, який взаємодіє з пам'яттю та процесором. Існує багато допоміжних компонентів, які можна віднести до периферійних пристроїв. Наявність певного прояву периферійного пристрою введення-виведення є елементарним для мікропроцесора, оскільки це механізм, за допомогою якого функціонує процесор.

Інші допоміжні елементи мікроконтролера включають наступне:

– **Аналого-цифровий перетворювач.** АЦП – це схема, яка перетворює аналогові сигнали в цифрові. Це дозволяє процесору всередині мікроконтролера взаємодіяти з зовнішніми аналоговими пристроями, такими як датчики.

– **Цифро-аналоговий перетворювач.** ЦАП виконує зворотну функцію АЦП, дозволяючи процесору мікроконтролера передавати свої вихідні сигнали зовнішнім аналоговим компонентам.

– **Системна шина.** Системна шина – це сполучний провід, який з'єднує всі компоненти мікроконтролера.

– **Послідовний порт.** Послідовний порт є одним із прикладів порту вводу-виводу, який дозволяє мікроконтролеру підключатися до зовнішніх компонентів. Він має подібну функцію до USB або паралельного порту, але відрізняється способом обміну бітами.

Особливості мікроконтролерів. Процесори мікроконтролерів відрізняються залежно від застосування. Варіанти варіюються від простих 4-розрядних, 8- або 16-розрядних процесорів до складніших 32- або 64-розрядних процесорів. Мікроконтролери можуть використовувати енергозалежну пам'ять, таку як RAM,

і типи енергонезалежної пам'яті, включаючи флеш-пам'ять, програмовану постійну пам'ять, що стирається, і програмовану ПЗУ, що електрично стирається.

Як правило, мікроконтролери можна використовувати без додаткових обчислювальних компонентів. Вони розроблені з достатньою кількістю вбудованої пам'яті, а також пропонують контакти для загальних операцій вводу-виводу, тому вони можуть безпосередньо взаємодіяти з датчиками та іншими компонентами.

Архітектура мікроконтролера базується на архітектурі Гарвардського університету або архітектурі фон Неймана. Вони пропонують різні методи обміну даними між процесором і пам'яттю. У гарвардській архітектурі шина даних і інструкції є роздільними, що забезпечує одночасну передачу. В архітектурі фон Неймана одна шина використовується як для даних, так і для інструкцій.

Процесори мікроконтролерів засновані на комп'ютері зі складним набором команд (CISC) або комп'ютері зі скороченим набором команд (RISC). CISC зазвичай має близько 80 інструкцій, тоді як RISC має близько 30. CISC також має більше режимів адресації, від 12 до 24 у порівнянні з RISC від трьох до п'яти.

Приклади моделей мікроконтролерів:

– **MCS-51.** Intel розробила цей тип однокристального мікроконтролера в 1980 році. Його також називають мікроконтролером 8051. Він використовував CISC і архітектуру Гарвардського університету та мав 8-, 16- та 32-розрядні розміри даних. Intel припинила виробництво MCS-51 на початку 2000-х, хоча інші виробники мікросхем пропонують його вдосконалені версії.

– **AVR.** Компанія Atmel розробила цей 8-розрядний однокристальний мікроконтролер RISC у 1996 році, використовуючи модифіковану гарвардську архітектуру. Це сімейство мікроконтролерів було одним із перших, хто використовував флеш-пам'ять комп'ютера для зберігання програм. Microchip Technology придбала Atmel у 2016 році та продовжує виробляти мікроконтролери AVR.

– **Програмований інтелектуальний комп'ютер (PIC).** Компанія General Instrument розробила мікроконтролер PIC у 1976 році під назвою Контролер програмованого інтерфейсу. Це сімейство мікроконтролерів

можна запрограмувати для виконання різних завдань, таких як керування електричними процесами в будинках, транспортних засобах і медичних установах.

– **ARM**. Мікроконтролери Arm також відомі як мікроконтролери Arm Cortex-M. Ці легкі мікроконтролери використовуються в мобільних електронних пристроях, а також у виробничих установках. Вони розроблені таким чином, щоб бути енергоефективними та придатними для ряду вбудованих систем. Ці мікроконтролери є частиною сімейства процесорів Arm, розроблених Acorn Computers на початку 1980-х років.

Застосування мікроконтролерів. Мікроконтролери використовуються в багатьох галузях промисловості, зокрема вдома та на підприємствах, автоматизації будівель, виробництві, робототехніці, автомобільній промисловості, освітленні, інтелектуальній енергетиці, промисловій автоматизації, зв'язку та додатках Інтернету речей у бізнес-налаштуваннях. Основні області, де використовуються мікроконтролери, включають наступні:

– **Цифрові сигнальні процесори (DSP).** Одним із застосувань мікроконтролера є його використання як DSP. Часто вхідні аналогові сигнали мають певний рівень шуму. Шум у цьому контексті означає неоднозначні значення, які не можна легко перевести в стандартні цифрові значення. Мікроконтролер може використовувати свій АЦП і ЦАП для перетворення вхідного шумового аналогового сигналу в рівномірний вихідний цифровий сигнал.

– **Побутова техніка.** Найпростіші мікроконтролери полегшують роботу електромеханічних систем, які є в побутових предметах, таких як печі, холодильники, тостери, мобільні пристрої, брелоки, системи відеоігор, телевізори та системи поливу газонів.

– **Офісні машини.** Мікроконтролери також поширені в офісних машинах, таких як фотокопіювальні апарати, сканери, факсимільні апарати та принтери, а також в розумних лічильниках, банкоматах і системах безпеки.

– **Більш складні програми.** Мікроконтролери виконують важливі функції в літаках, космічних кораблях, океанських суднах і роботах.

– **Медичні застосування.** У медичних сценаріях мікроконтролери можуть регулювати роботу штучного серця, нирок або інших органів. Вони також можуть сприяти функціонуванню протезів.

Мікроконтролери проти мікропроцесорів. Основна відмінність мікроконтролерів від мікропроцесорів полягає в рівні функціональності. Мікроконтролери функціонують самостійно, безпосередньо підключаючись до датчиків і виконавчих механізмів. Мікропроцесори розроблені для максимізації обчислювальної потужності чіпу за допомогою внутрішніх шинних з'єднань, а не для прямого вводу-виводу на допоміжне обладнання, таке як оперативна пам'ять і послідовні порти. Простіше кажучи, в кавоварках використовуються мікроконтролери; настільні комп'ютери використовують мікропроцесори.

Різниця між мікроконтролерами та мікропроцесорами стала менш чіткою, оскільки щільніші та складніші мікросхеми стали відносно дешевими у виробництві. Ця тенденція дозволила мікроконтролерам використовувати комп'ютерні функції загального призначення.

Мікроконтролери дешевші та споживають менше енергії, ніж мікропроцесори. Мікропроцесори не мають вбудованої ОЗП, ПЗУ чи інших периферійних пристроїв на чіпі, а приєднуються до них за допомогою контактів. Мікропроцесор вважається серцем комп'ютерної системи, тоді як мікроконтролер є серцем вбудованої системи.

1.2 Архітектура процесорів

CISC проти RISC. Комп'ютер зі скороченим набором інструкцій (RISC) – це тип або категорія процесора або архітектура набору інструкцій (ISA). Якщо говорити в широкому сенсі, то ISA – це середовище, за допомогою якого процесор спілкується з людиною-програмістом (хоча між процесором і програмістом є кілька інших формально ідентифікованих рівнів). Інструкція – це команда, яка дається процесору для виконання певної дії. Набір інструкцій – це повний набір інструкцій для певного процесора, а термін «архітектура» означає певний спосіб побудови системи, яка створює процесор.

RISC зазвичай відноситься до спрощеної версії свого попередника, CISC. На зорі процесорів не існувало офіційної ідентифікації, відомої як CISC, але з тих пір цей термін був придуманий, щоб ідентифікувати їх як відмінні від архітектури RISC. Деякі приклади архітектур набору інструкцій мікропроцесора CISC (ISA) включають Motorola 68000 (68K), DEC VAX, PDP-11, кілька поколінь Intel x86 і 8051.

Приклади процесорів з RISC-архітектурою включають MIPS, PowerPC, Atmel AVR, процесори Microchip PIC, процесори Arm, RISC-V, і всі сучасні мікропроцесори мають принаймні деякі елементи RISC. Перехід від 8- і 16-розрядних до 32-розрядних архітектур по суті викликав потребу в архітектурах RISC. Тим не менш, знадобилося десятиліття, перш ніж архітектури RISC почали закріплюватися, в основному через відсутність програмного забезпечення, яке б працювало на архітектурах RISC. Корпорація Intel також вплинула на неї, оскільки вона мала засоби продовжувати використовувати архітектуру CISC і не виявила потреби переробляти її з нуля. Архітектура MIPS була однією з перших RISC ISA і широко використовувалася для навчання архітектурі RISC.

Які відмінності між RISC і CISC? Коротка відповідь полягає в тому, що багато хто сприймає RISC як покращення порівняно з CISC. Не існує найкращої архітектури, оскільки різні архітектури можуть просто бути кращими в одних сценаріях, але менш ідеальними в інших. Машини на основі RISC виконують одну інструкцію за такт. Машини CISC можуть мати спеціальні інструкції, а також інструкції, для виконання яких потрібно більше одного такту. Це означає, що одна і та сама інструкція, що виконується на архітектурі CISC, може потребувати кількох інструкцій для виконання на машині RISC. Архітектура RISC потребуватиме більше робочої (RAM) пам'яті, ніж CISC, щоб зберігати значення, коли вона завантажує кожну інструкцію, виконує її, а потім завантажує наступну.

Архітектура CISC може виконувати одну, хоча й більш складну інструкцію, яка виконує ті самі операції, усі одночасно, безпосередньо в пам'яті. Таким чином, архітектура RISC вимагає більше оперативної пам'яті, але завжди виконує одну інструкцію за такт для передбачуваної обробки, що добре для конвеєрної обробки. Однією з головних відмінностей між RISC і CISC є те, що RISC підкреслює

ефективність в циклах на інструкцію, а CISC підкреслює ефективність в інструкціях на програму. Швидкість процесора залежить від того, скільки часу потрібно для виконання кожного такту, скільки циклів потрібно для виконання інструкцій і кількості інструкцій у кожній програмі. RISC потребує більших розмірів програмного коду (через менший набір інструкцій, тому кілька послідовних кроків можуть прирівнюватися до одного кроку в CISC).

RISC ISA наголошує на програмному забезпеченні, а не на апаратному забезпеченні. Набір інструкцій RISC вимагає написання більш ефективного програмного забезпечення (наприклад, компіляторів або коду) з меншою кількістю інструкцій. CISC ISA використовують більше транзисторів в апаратному забезпеченні для реалізації більшої кількості інструкцій, а також більш складних інструкцій.

RISC потребує більше оперативної пам'яті, тоді як CISC наголошує на меншому розмірі коду та використовує менше оперативної пам'яті загалом, ніж RISC. Проте багато мікропроцесорів сьогодні містять суміш RISC- і CISC-подібних атрибутів, наприклад CISC-подібний ISA, який розглядає інструкції так, ніби вони є рядком інструкцій типу RISC. Деякі основні відмінності між архітектурами CISC і RISC наведено в таблиці 1.1.

Таблиця 1.1 – Порівняння архітектур CISC і RISC

RISC	CISC
RISC – це скорочений набір інструкцій.	CISC – це складний набір інструкцій.
Кількість інструкцій менша порівняно з CISC.	Кількість інструкцій більше в порівнянні з RISC.
Менше режимів адресації.	Режимів адресації більше.
Працює у фіксованому форматі інструкцій.	Працює у форматі змінних інструкцій.
RISC споживає мало енергії.	CISC споживає багато енергії.
Процесори RISC високо конвеєрні.	Процесори CISC менш конвеєрні.
Оптимізує продуктивність, зосереджуючись на програмному забезпеченні.	Оптимізує продуктивність, зосереджуючись на апаратному забезпеченні.
Потрібно більше оперативної пам'яті.	Вимагає менше оперативної пам'яті.

Гарвардська архітектура проти фон-Нейманівської. Архітектури фон Неймана та Гарвардська є основними в області комп'ютерної архітектури, які пояснюють спосіб взаємодії пам'яті та блоків процесів у комп'ютерній системі. У той час як перша займає домінуюче положення, її принципові відмінності від останньої, переваги та недоліки будуть обговорюватися далі, щоб надати вам інформацію про те, яка структура більше підходить для певної програми.

Архітектура фон Неймана. Архітектура фон Неймана – це цифрова комп'ютерна архітектура, дизайн якої базується на концепції комп'ютерів із збереженими програмами, де дані програми та дані інструкцій зберігаються в одній пам'яті. Цю архітектуру спроектував відомий математик і фізик Джон фон Нейман у 1945 році (рис. 1.1).

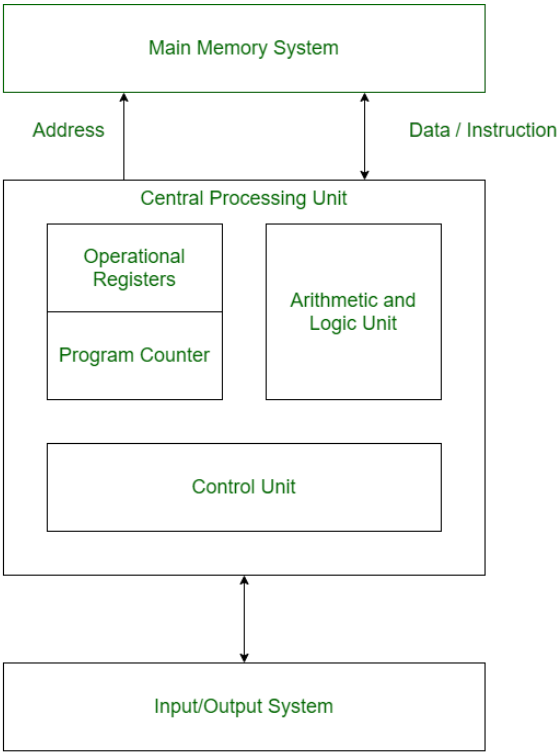


Рисунок 1.1 – Архітектура фон Неймана

Переваги архітектури фон Неймана

– **Простота:** Той факт, що всі дані та інструкції зберігаються в одному просторі пам'яті, допомагає процесу проектування комп'ютерної системи, оскільки немає необхідності створювати складні системи маршрутизації, оскільки шляхи можуть збігатися.

– **Рентабельність:** потрібна менша кількість компонентів порівняно з іншими архітектурними проектами, отже, вона є економічнішою.

– **Гнучкість:** програму можна завжди змінювати, не роблячи істотних змін у деяких основних фізичних аспектах, таких як схема.

Недоліки архітектури фон Неймана

– **Проблеми з вузькими місцями:** спільна шина може бути проблемою, оскільки дані та керуючі інструкції не можуть бути отримані одночасно, і тому вона стає повільною.

– **Пошкодження пам'яті:** оскільки дані та інструкції зберігаються в одній пам'яті, виникає ризик, що одне стирає інше, що призводить до системних збоїв.

Гарвардська архітектура. Гарвардська архітектура – це цифрова комп'ютерна архітектура, дизайн якої базується на концепції, де є окреме сховище та окремі шини (шлях сигналу) для інструкцій і даних. В основному вона була розроблена, щоб подолати вузьке місце архітектури фон Неймана (рис. 1.2).

Особливості:

– Окремі простори пам'яті.
– Фіксована довжина інструкції.
– Паралельні інструкції та доступ до даних.
– Більш ефективне використання пам'яті.
– Підходить для вбудованих систем.
– Обмежена гнучкість.

Переваги Гарвардської архітектури:

– **Швидша обробка:** наявність двох шин для даних і інструкцій дозволяє уникнути проблеми суперечок, коли використовується лише одна шина, і це підвищує швидкість роботи системи.

– **Покращена безпека:** у такий спосіб ймовірність пошкодження пам'яті зменшується принаймні вдвічі, оскільки дані зберігаються не в тих самих місцях, що й інструкції.

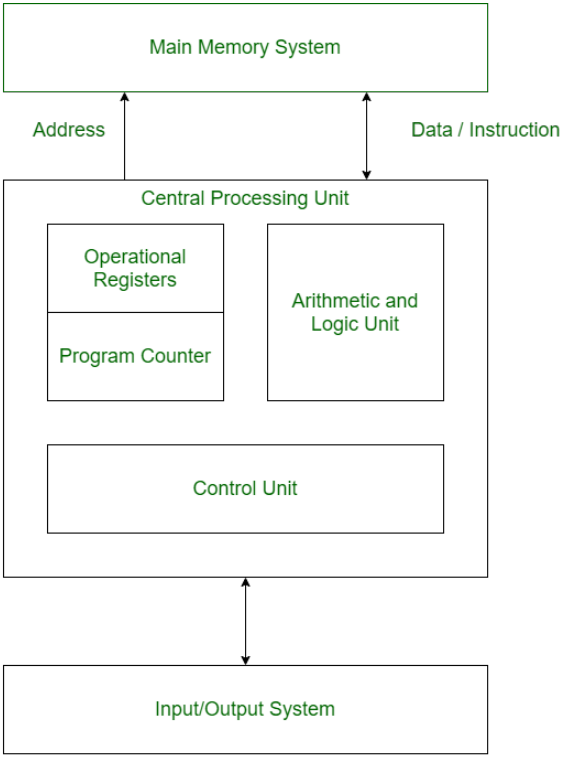


Рисунок 1.2 – Гарвардська архітектура

– **Ефективне використання ресурсів:** дозволяє використовувати різну пам'ять для даних і інструкцій різного розміру, оскільки це сприяє оптимальному використанню шин та інших ресурсів.

Недоліки Гарвардської архітектури:

– **Складність:** конструкція та реалізація цього типу є більш складними, тому потрібні інші апаратні засоби.

– **Вища вартість:** оскільки концепція гарвардської архітектури передбачає два набори пам'яті та дві окремі шини, вартість їх реалізації порівняно висока, ніж архітектури фон Неймана.

– **Менша гнучкість:** зміна або навіть покращення системи також може бути дещо складними через різні області пам'яті.

Гарвардська і фон-Нейманівська архітектури мають певні переваги і недоліки, і можна вибрати ту, що залежить від подальшого застосування пристрою. Для обчислень загального призначення здебільшого використовується архітектура фон Неймана, оскільки її легше та дешевше реалізувати, тоді як гарвардська архітектура використовується широко, особливо коли швидкість є основним фактором, як у використанні вбудованих систем.

1.3 Типи пам'яті мікроконтролерів

Кількість внутрішньої пам'яті в MCU залежить від того, як пам'ять категоризовано. На найвищому рівні є два типи:

– оперативна пам'ять (RAM);
– постійна пам'ять (ROM).

Але, якщо нас цікавить продуктивність пам'яті, існують різні типи RAM і ROM. І ці різні типи пам'яті можуть використовуватися для різних функцій, таких як кеш-пам'ять, основна пам'ять, пам'ять програм і так далі. З іншого боку, існує питання визначення віртуальної та фізичної пам'яті.

Двома основними типами оперативної пам'яті є статична оперативна пам'ять (SRAM) і динамічна оперативна пам'ять (DRAM). Обом потрібна напруга, щоб зберегти інформацію. DRAM є простою і вимагає лише одного транзистора та одного конденсатора для базової реалізації. DRAM є найпоширенішою з усіх технологій пам'яті. Якщо вона інтегрована в MCU, то називається вбудованою DRAM (eDRAM). Ціна за біт для eDRAM вища порівняно з еквівалентними автономними мікросхемами DRAM, які використовуються як зовнішня пам'ять. Тим не менш, переваги продуктивності розміщення eDRAM на тому ж чіпі, що й процесор, переважують недоліки у вартості високопродуктивних програм.

SRAM є складнішим, ніж eDRAM, і зазвичай реалізується за допомогою шістьох транзисторів типу CMOS. SRAM також швидше, ніж DRAM, що робить її дуже придатною для інтеграції в мікроконтролери. Це одна з найбільш використовуваних

технологій внутрішньої пам'яті MCU. SRAM зазвичай використовується як кеш-пам'ять і для регістрів процесора.

Енергонезалежна пам'ять у мікроконтролерах включає флеш-пам'ять і та програмоване ПЗУ з електричним стиранням (EEPROM). Флеш-пам'ять є формою EEPROM. Основна відмінність між ними полягає в тому, як ними керують; Flash управляється (записується або стирається) на рівні блоку, а EEPROM можна керувати на рівні байтів. Флеш-пам'ять доступна в архітектурах NAND і NOR. NAND Flash обробляє дані блоками та читає швидше, ніж записує. Вона може швидко передавати сторінки даних. Вона забезпечує більшу місткість на одиницю площі, ніж NOR, і використовується для зберігання з високою щільністю. NOR Flash підтримує більш детальну роботу та забезпечує високошвидкісний довільний доступ. NOR Flash може читати та записувати дані певним чином.

Фудзіо Масуока винайшов флеш-пам'ять, коли працював у Toshiba у 1980-х роках. Його колега Шодзі Аріідзумі використовував термін Flash для опису нової технології, оскільки стирання всіх даних нагадувало йому спалах камери. Технології енергозалежної та енергонезалежної пам'яті можна порівняти за кількома критеріями продуктивності:

– Швидкість: Енергозалежна пам'ять зазвичай швидша.

– Вартість: Енергозалежна пам'ять коштує дешевше.

– Термін служби: енергозалежна пам'ять має довший термін служби. Енергонезалежна пам'ять має обмежений термін служби через її можливості повторного запису.

– Енергоспоживання. Енергозалежна пам'ять, наприклад DRAM, потребує повторного оновлення даних, що споживає додаткову енергію. Енергонезалежна пам'ять зазвичай споживає менше енергії.

Ієрархії пам'яті. Кеш-пам'ять є критично важливою системою в MCU. Є два способи класифікації кеш-пам'яті: ієрархія або функціональність. При описі в термінах ієрархії може бути до 4 рівнів кеш-пам'яті. Кеш складається зі швидкої пам'яті, такої як SRAM та eDRAM, щоб компенсувати повільніший час доступу до основної флеш-пам'яті. Кеш рівня 1 – це невеликий блок пам'яті, який може працювати так само швидко, як ЦП, щоб підтримувати

максимальну швидкість обробки. Кеші рівня 2 і рівня 3 підтримують кеш рівня 1. Вони більші та повільніші, ніж кеш рівня 1, але мають швидший час доступу, ніж основна пам'ять (рис. 1.3).

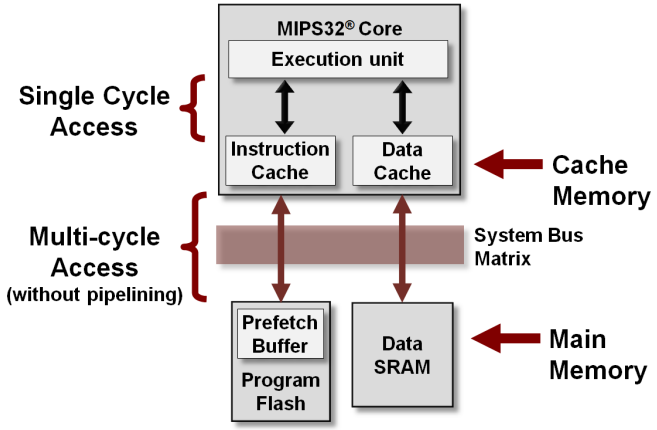


Рисунок 1.3 – Кеш-пам'ять, що встановлюється між процесором і основною пам'яттю, щоб забезпечити швидший час доступу, необхідний для підтримки ефективної роботи процесора

З точки зору продуктивності та розміру, eDRAM знаходиться між кеш-пам'яттю рівня 3 і звичайною DRAM на шині пам'яті та функціонує як кеш рівня 4. Вища щільність eDRAM порівняно з SRAM може підтримувати набагато ширші шини та вищі робочі швидкості. І більші обсяги eDRAM можна інтегрувати в меншу область, ніж SRAM. Виготовити eDRAM складніше, ніж SRAM, але 3-кратна економія площі за допомогою eDRAM може компенсувати витрати на виготовлення, коли потрібні великі обсяги пам'яті.

Для будь-якого рівня кеш-пам'яті всі блоки мають однаковий розмір і асоціативність. Нижчі рівні, такі як кеш рівня 1, мають менше блоків, менші розміри блоків і менше блоків у наборі, але вони забезпечують дуже швидкий час доступу. Коли рівень підвищується до 2 і 3, кеш-пам'ять має все більшу кількість блоків, більший розмір блоків і більше блоків у наборі. Але кожен рівень

кеш-пам'яті набагато швидший, ніж основна пам'ять. На додаток до основної кеш-пам'яті для певних функцій використовуються спеціалізовані типи. Приклади:

– **Кеш конвеєра.** Конвеєрні процесори, наприклад, у RISC MCU, отримують доступ до пам'яті з кількох точок конвеєра, включаючи вибірку інструкцій вибірки даних і трансляцію адреси з віртуальної у фізичну. Конвеєр використовує три спеціалізовані кеші: дані, інструкції та буфер перегляду трансляції (TLB).

– **Кеш жертовних даних.** Блоки даних, які були замінені та видалені з кешу процесора, зберігаються в кеші жертовних даних. Він встановлюється між основним кеш-пам'яттю та шляхом поповнення. Він повністю асоціативний і призначений для зменшення кількості промахів конфліктів у програмах, які можуть отримати вигоду від високоасоціативного відображення. У деяких реалізаціях кеш рівня 4 може функціонувати як кеш жертовних даних.

– **Кеш мікрооперацій (μop).** Цей кеш зберігає мікрооперації декодованих інструкцій, отриманих з кешу інструкцій або декодера інструкцій. Це може прискорити обробку. Кеш μop перевіряється, коли інструкція потребує декодування, щоб побачити, чи є її декодована форма вже доступною. Якщо вона недоступна в кеші μop, інструкція декодується та кешується для майбутнього використання.

Організація пам'яті та архітектура МК. Різні типи МК, такі як архітектура AVR і ARM, використовують різні способи організації пам'яті. Гарвардська архітектура AVR організовує пам'ять як флеш-пам'ять, внутрішню та зовнішню DRAM та EEPROM (рис. 1.4).

У результаті системи, які використовують ці MCU, організовують пам'ять як певні розділи, зокрема:

– Текст (text).
– Дані (data).
– Символ початку блоку (BSS).
– Стек (Stack).
– Купа (Heap).

Текстовий розділ містить інструкції, завантажені у флеш-пам'ять; розділ даних містить змінні, які були ініціалізовані, BSS містить неініціалізовані дані, стек містить дані функцій і переривань, а купа містить змінні, створені під час виконання.

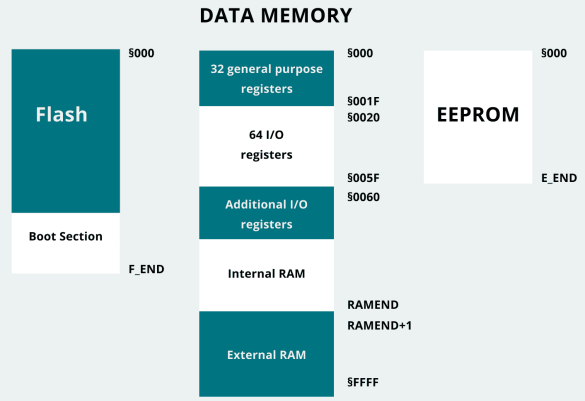


Рисунок 1.4 – Організація пам’яті у мікроконтролері AVR гарвардської архітектури

У ARM MCU карти пам’яті використовуються з іншою конфігурацією адреси 32-, 36- та 40-біт, що залежить від вимог системного адресного простору та додаткової DRAM. Блок керування пам’яттю (MMU) керує інструкціями доступу до пам’яті, які можна використовувати в коді високого рівня для керування модулями переривань та вбудованими периферійними пристроями.

Основне призначення MMU – дозволити процесору незалежно виконувати кілька завдань у просторі віртуальної пам’яті. MMU використовує таблиці перекладу для з’єднання адрес віртуальної та фізичної пам’яті. Віртуальними адресами керують за допомогою програмного забезпечення з інструкціями пам’яті. Фізичні адреси контролюються на основі вхідних даних таблиці перекладу, заданих віртуальною адресою (рис. 1.5).

MMU – це спеціалізований блок пам’яті, який включає блок обходу таблиці, який зчитує таблиці перекладу з пам’яті та TLB, які кешують нещодавно використані переклади. Усі адреси пам’яті з програмного забезпечення ЦП є віртуальними. MMU перевіряє TLB на наявність нещодавно кешованого перекладу. Якщо його не існує, блок обходу таблиці зчитує відповідний запис таблиці або записи з пам’яті.

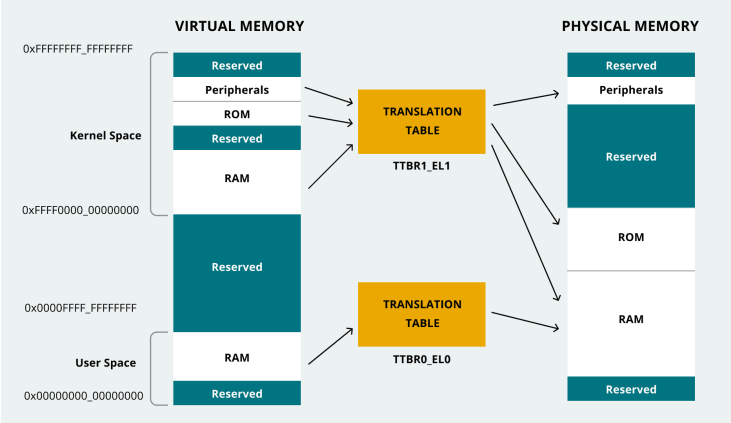


Рисунок 1.5 – У гібридному процесорі ARM таблиці трансляції відображаються між віртуальною та фізичною пам'яттю

Висновки. MCU включають різні форми RAM і ROM для підтримки конкретних вимог до продуктивності. Найпоширеніші форми пам'яті в мікроконтролерах включають енергозалежну пам'ять eDRAM і SRAM, а також енергонезалежну пам'ять Flash і EEPROM. Flash і EEPROM зазвичай використовуються для основної пам'яті, тоді як eDRAM і SRAM використовуються для різних функцій кеш-пам'яті. Крім того, пам'ять MCU організована на основі віртуальних і фізичних адрес і функціональних можливостей і управляється за допомогою MMU.

Контрольні запитання до теми 1

1. Які основні типи мікроконтролерів?
2. Що таке вбудовані мікроконтролери і які їхні основні характеристики?
3. Які функції можуть додатково реалізувати більш складні вбудовані мікроконтролери?
4. Чим відрізняються мікроконтролери з зовнішньою пам'яттю від вбудованих мікроконтролерів?
5. Що таке цифрові сигнальні процесори (DSP) і для чого вони призначені?

6. Яка технологія використовується для виготовлення сучасних мікроконтролерів і які її переваги?
7. Які типові значення максимальної частоти тактових сигналів для різних мікроконтролерів?
8. Для яких завдань найкраще підходять мікроконтролери з зовнішньою пам'яттю?
9. Який класичний приклад мікроконтролера з зовнішньою пам'яттю наводиться в тексті, і які його основні характеристики?
10. Які основні функції виконує ALU в цифрових сигнальних процесорах?
11. Що означають абревіатури CISC та RISC?
12. Чому вважається, що RISC-процесори швидші за CISC-процесори?
13. Чим відрізняється набір команд у CISC- та RISC-процесорах?
14. Як умовні переходи реалізуються у CISC- та RISC-процесорах?
15. Яка основна перевага RISC-процесорів?
16. Що являє собою архітектура фон Неймана?
17. Яка проблема може виникнути у процесорів з Прінстонською архітектурою і як її вирішують?
18. Яка головна перевага Гарвардської архітектури порівняно з архітектурою фон Неймана?
19. Які три основні види пам'яті використовуються в мікроконтролерах?
20. Для чого призначена пам'ять програм в мікроконтролерах?
21. Які види постійної пам'яті використовуються для зберігання програм в мікроконтролерах?
22. Які недоліки масочної пам'яті ROM обмежують її використання?
23. Як програмується і стирається пам'ять EPROM?
24. Які переваги має пам'ять EEPROM перед EPROM?
25. Чим функціонально відрізняється Flash-пам'ять від EEPROM?

Використана література

1. What is a microcontroller (MCU)? URL: https://www.techtarget.com/iotagenda/definition/microcontroller
2. RISC vs. CISC Architectures: Which one is better? URL: https://www.microcontrollertips.com/risc-vs-cisc-architectures-one-better/
3. Difference between Von Neumann and Harvard Architecture. URL: https://www.geeksforgeeks.org/difference-between-von-neumann-and-harvard-architecture/
4. How many internal memories does an MCU have? URL: https://www.microcontrollertips.com/how-many-internal-memories-does-an-mcu-have-faq/

2 АРХІТЕКТУРА МІКРОКОНТРОЛЕРІВ СІМЕЙСТВА AVR

Метою вивчення теми є ознайомлення з основними характеристиками мікроконтролерів AVR на прикладі МК ATMega8.

Завдання вивчення теми збігаються з переліком питань для розгляду, що наведений нижче.

Перелік питань до розділу:

2.1. Основні характеристики мікроконтролерів AVR на прикладі МК ATMega8.
2.2. Коротка характеристика архітектури.
2.3. Регістровий файл.
2.4. Арифметико-логічний пристрій (АЛУ).
2.5. Доступ до пам'яті та виконання інструкцій.
2.6. Пам'ять вводу-виводу.
2.7. EEPROM.
2.8. Порти вводу-виводу.
2.9. Таймер.
2.10. Система переривань.
2.11. Вбудований сторожовий таймер (Watchdog).
2.12. Режими роботи з низьким споживанням енергії.

2.1 Основні характеристики мікроконтролерів AVR на прикладі МК ATMega8

Мікроконтролер AVR був виготовлений корпорацією Atmel у 1996 році, а його архітектуру розробили Альф-Егіл Боген і Вегард Воллан. Назву цього мікроконтролера було взято від його розробників, а саме RISC від Альф-Егіла Богена і Вегарда Воллана. Першим

мікроконтролером на основі архітектури AVR є AT90S8515, але перший комерційний мікроконтролер був AT90S1200.

Atmel AVR є одним із найпопулярніших сімейств мікроконтролерів сьогодні. Причиною такої величезної популярності є відносна простота використання та низька вартість мікроконтролерів, які можна придбати за ціною від 1 до 10 доларів.

Мікроконтролер AVR містить процесор, програмовані периферійні пристрої вводу-виводу та пам'ять. Мікроконтролер AVR забезпечує цифрове керування будь-якими електричними, автомобільними чи механічними системами, промисловими установками, різними пристроями, електронними гаджетами тощо.

Мікроконтролери AVR доступні в 3 серіях, як-от TinyAVR, MegaAVR і XmegaAVR.

Мікроконтролери TinyAVR доступні в невеликих розмірах, мають менше пам'яті та оптимізовані під простіші програми.

Мікроконтролери MegaAVR дуже поширені, оскільки вони мають до 256 Кб пам'яті, включають максимальну кількість периферійних пристроїв і використовуються в додатках середнього та складного рівня.

XmegaAVR часто використовується для складних програм, де потрібна висока швидкість і велика пам'ять програм.

Таблиця 2.1 – Порівняння серій мікроконтролерів AVR

Назва серії	**Кількість виводів**	**Об'єм флеш-пам'яті**	**Особливості**
TinyAVR	від 6 до 32	від 0,5 до 8 Кб	Маленький розмір
MegaAVR	від 28 до 100	від 4 до 256 КБ	Розширені периферійні пристрої
XmegaAVR	від 44 до 100	від 16 до 384 КБ	Включає DMA та систему подій

ATMega8 є восьмирозрядним мікроконтролером із внутрішньою програмованою Flash-пам'яттю розміром 8 Кбайт.

Загальні відомості:

• Високопродуктивний 8-розрядний мікроконтролер Atmel®AVR® з низьким енергоспоживанням.

• Розширена архітектура RISC:

– 130 потужних інструкцій – більшість з яких виконуються за один такт.

– 32 × 8 робочих регістрів загального призначення.

– Повністю статична архітектура.

– Продуктивність до 16 MIPS на 16 МГц.

– Вбудований двотактний множник.

• Сегменти енергонезалежної пам'яті високої витривалості:

– 8 Кбайт внутрішньосистемної самопрограмованої флеш-пам'яті програм.

– 512 байт EEPROM.

– 1 Кбайт внутрішньої SRAM.

– Цикли запису-стирання: 10 000 Flash/100 000 EEPROM.

– Зберігання даних: 20 років при 85 °C/100 років при 25 °C.

– Додатковий розділ коду завантаження з незалежними бітами блокування.

– Внутрішньосистемне програмування за допомогою вбудованої програми завантаження.

– Справжня операція читання під час запису.

– Блокування програмування для безпеки програмного забезпечення.

• Периферійні функції:

– Два 8-бітних таймера-лічильника з окремим попереднім дільником, один режим порівняння.

– Один 16-бітний таймер-лічильник з окремим попереднім дільником, режимом порівняння та захоплення.

– Лічильник реального часу з окремим осцилятором.

– Три канали ШІМ.

– 8-канальний АЦП в корпусі TQFP і QFN/MLF – вісім каналів 10-бітної точності.

– 6-канальний АЦП в корпусі PDIP – шість каналів 10-бітної точності.

– Байт-орієнтований двопровідний послідовний інтерфейс.

– Програмований послідовний USART.

– Послідовний інтерфейс SPI Master/Slave.

– Програмований сторожовий таймер з окремим вбудованим генератором.

– Вбудований аналоговий компаратор.

• Спеціальні функції мікроконтролера:

– Скидання під час увімкнення живлення та програмоване виявлення зниження напруги живлення.

– Внутрішній калібрований RC осцилятор.

– Зовнішні та внутрішні джерела переривань.

– П’ять режимів сну: неактивний режим, зменшення шуму АЦП, енергозбереження, вимкнення та Режим очікування.

• Введення-виведення та корпуси:

– 23 програмовані лінії вводу-виводу.

– 28-вивідний PDIP, 32-вивідний TQFP і 32-вивідний QFN/MLF (рис. 2.1).

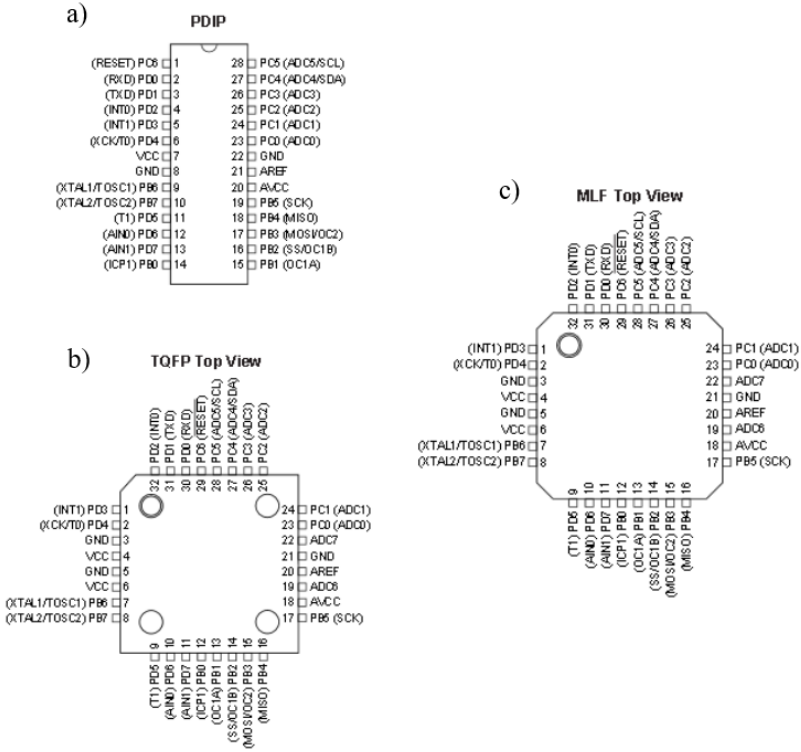


Рисунок 2.1 – Виводи вводу-виводу:
a) PDIP корпус, b) TQFP корпус, c) MLF корпус

- Робоча напруга:
– 2,7 В – 5,5 В (ATmega8L).
– 4,5 В – 5,5 В (ATmega8).
- Тактові частоти:
– 0–8 МГц (ATmega8L).
– 0–16 МГц (ATmega8).
- Енергоспоживання при 4 МГц, 3 В, 25 °С:
– Активний: 3,6 мА.
– Неактивний: 1,0 мА.
– Режим вимкнення живлення: 0,5 мкА.

2.2 Коротка характеристика архітектури

AVR використовує гарвардську архітектуру. Це передбачає окремі шини пам'яті даних і програм. На рис. 2.2 показано структурну схему контролера.

Шина даних пам'яті даних є 8-розрядною і з'єднує більшість периферійних компонентів з файлом регістру. Шина даних програмної пам'яті має ширину 16 біт і підключена лише до регістру інструкцій.

Незважаючи на те, що рис. 2.2 стосується контролера AVR ATMega8, він однаково добре підходить для всіх процесорів і відрізняється лише наявністю додаткових (або менших) периферійних компонентів, а також різницею в обсязі пам'яті програм і пам'яті даних.

Пам'ять програм – це безперервна частина флеш-пам'яті. Точна кількість залежить від процесора. ATTiny13, мікроконтролер базового рівня, має 1 Кбайт програмної пам'яті, організованої як 512-X-16 біт, тоді як ATMega128 має 128 Кбайт пам'яті, організованої як 64K-X-16 біт. K тут дорівнює 1024, а не 1000. Доступ до пам'яті програми здійснюється кожного такту, і команда завантажується в регістр інструкцій. Регістр інструкцій передає дані файлу регістрів, вибираючи, який із регістрів використовуватиметься ALU для виконання інструкцій.

Вихідний сигнал регістру команд також декодується декодером команд, щоб вирішити, які керуючі сигнали будуть активовані для завершення поточної команди. Пам'ять програми, крім зберігання

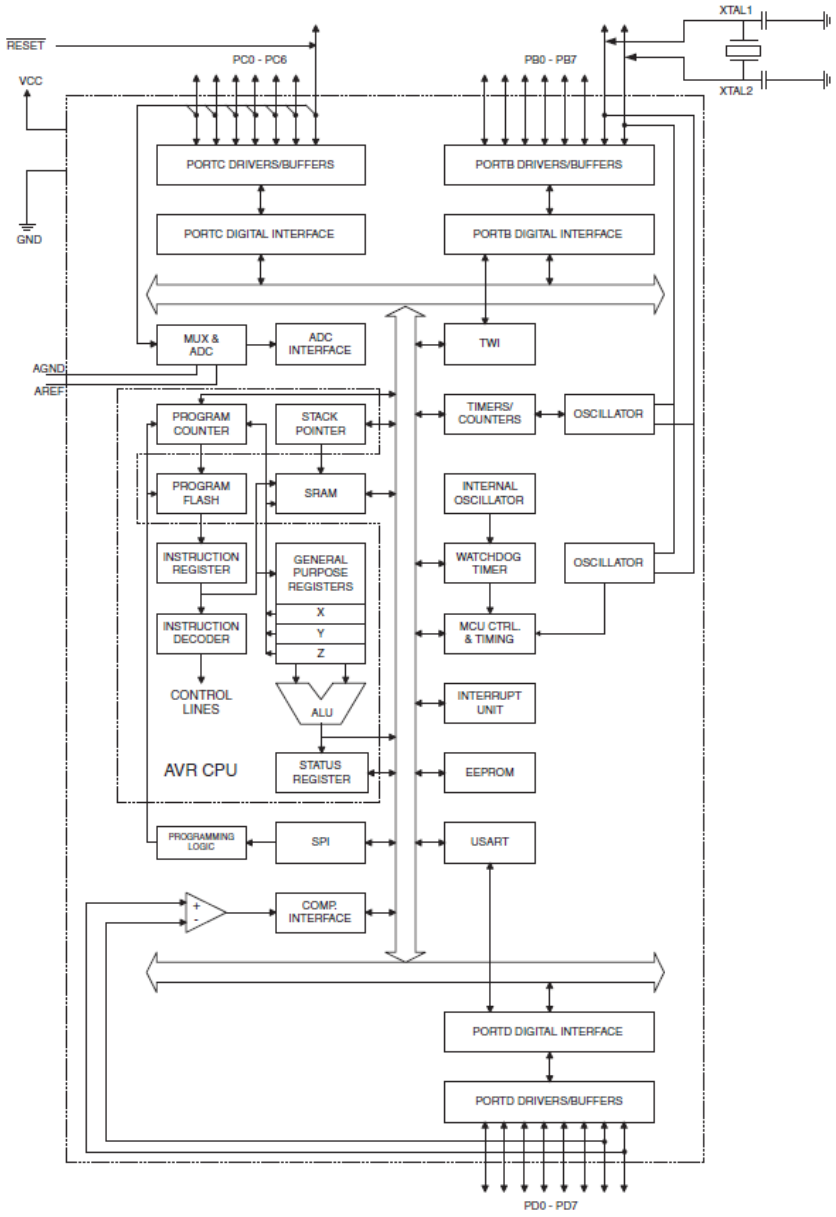


Рисунок 2.2 – Мікроконтролер ATMega8

інструкцій, також зберігає вектори переривань, починаючи з адреси $0000. Фактична програма повинна починатися з місця пам'яті за межами простору, призначеного для векторів. Кількість векторів залежить від процесора.

Пам'ять даних, з іншого боку, розділена на різні типи. На рис. 2.3 показано різні типи пам'яті, доступні для процесора AVR. Пам'ять даних складається з п'яти різних компонентів:

1. Регістровий файл із 32 регістрами шириною 8 біт. Всі процесори сімейства AVR мають цей реєстровий файл.

2. 64 регістри вводу-виводу по 8 біт кожен. Всі процесори не мають усіх 64 регістрів. Деякі мають більше, ніж інші, залежно від кількості периферійних компонентів на мікросхемі. Ці регістри насправді є частиною вбудованої SRAM, і до них можна отримати доступ як до SRAM з адресами від $20 до $5F, або як до регістрів вводу-виводу з адресами від $00 до $3F. Найчастіше до всіх цих регістрів звертаються як до регістрів вводу-виводу, а не як до SRAM.

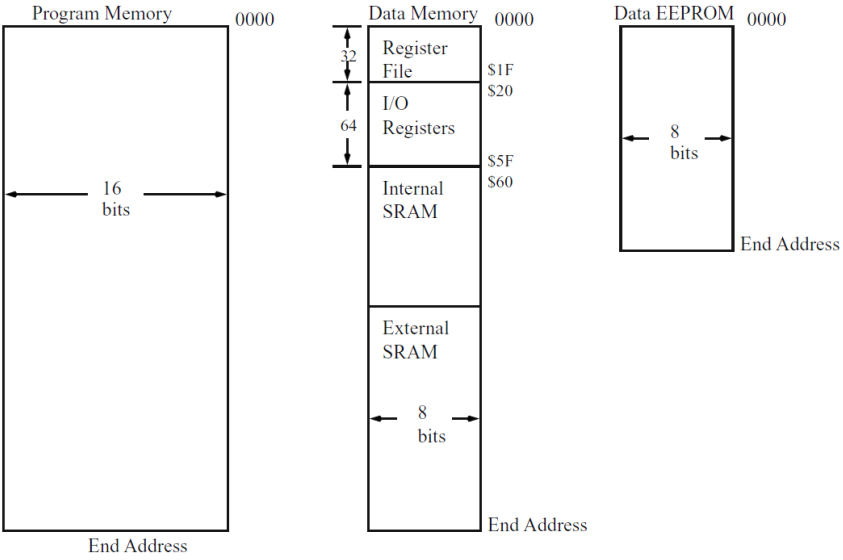


Рисунок 2.3 – Пам'ять мікроконтролерів AVR

3. Внутрішня SRAM. Вона доступна в більшості процесорів AVR, за винятком базових процесорів, таких як ATTiny12. Обсяг SRAM коливається від 32 байт до 8 Кбайт. SRAM використовується для стека, а також для зберігання змінних. Під час викликів переривань і підпрограм поточне значення лічильника програми зберігається в стеку. Розмір стека обмежений доступною SRAM на кристалі. Поточне розташування стека вказується вказівником стека. Покажчик стека має бути ініціалізований після скидання та перед використанням стека. Для тих процесорів, які не мають вбудованої SRAM, доступний апаратний стек для зберігання програмних адрес повернення. Цей апаратний стек може зберігати лише до 3 адрес.

4. Зовнішня SRAM. Вона доступна лише на більших процесорах сімейства AVR. Ті процесори, які мають зовнішні порти доступу до даних і пам'яті (такі як ATMega8535), можуть використовувати будь-яку доступну зовнішню SRAM, яку користувач може вирішити застосувати.

5. EEPROM. EEPROM доступний майже у всіх процесорах AVR і доступ до нього здійснюється в окремій області пам'яті. Початкова адреса EEPROM завжди $0000. Різні процесори мають від 64 байт до 8 Кбайт EEPROM. EEPROM може читатися та записуватися будь-якою програмою. Читання EEPROM відбувається швидше, ніж запис. EEPROM можна записувати приблизно 100 000 разів.

Більшість інструкцій AVR мають довжину в 1 слово (2 байти), тому займають 1 місце в пам'яті програми. Багато інструкцій виконуються за один такт, а деякі займають 2 або більше тактів. Таке одноциклове виконання досягається за рахунок використання 2-ступінчастого конвеєру. Конвеєр працює шляхом одночасного отримання нової інструкції з пам'яті програми, поки попередня інструкція виконується в іншій частині процесора. Таким чином, вибірка інструкцій, декодування та виконання є процесами, які виконуються процесором одночасно.

2.3 Регістровий файл

Усі процесори AVR мають 32 загальнопризначені регістри. Деякі з цих регістрів мають додаткові спеціальні функції. Регістри називаються від R0 до R31. Регістровий файл розділений на дві частини по 16 регістрів: R0–R15 і R16–R31. Усі інструкції, що працюють із регістрами, мають прямий доступ і виконуються за один цикл для всіх регістрів.

Регістри **R0** та **R26–R31** мають додаткові функції. **R0** використовується в інструкції **LPM** (завантаження пам'яті програми), тоді як **R26–R31** працюють як вказівні (pointer) регістри, як показано на рис. 2.4. Ці вказівні регістри широко використовуються в багатьох інструкціях непрямого доступу до регістрів.

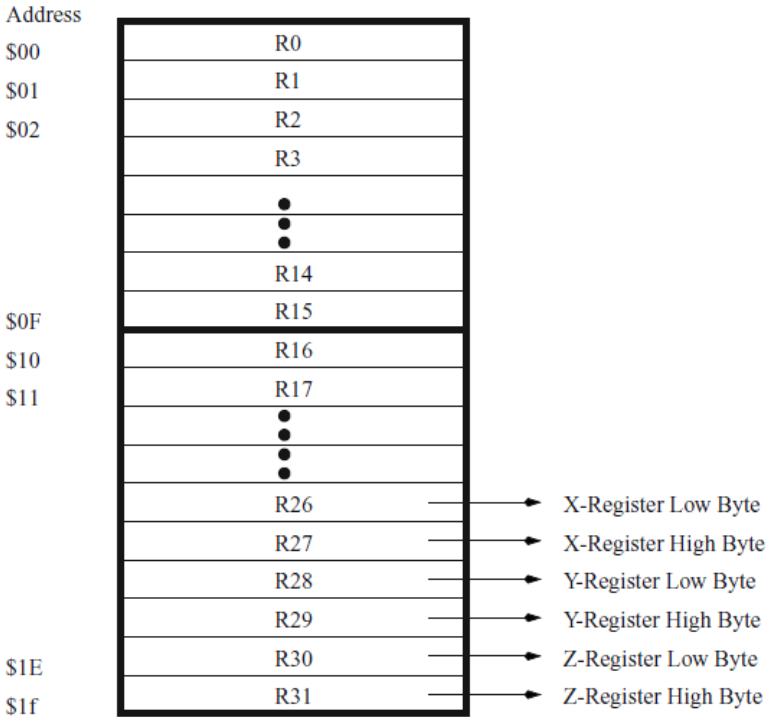


Рисунок 2.4 – Регістровий файл мікроконтролерів AVR

2.4 Арифметико-логічний пристрій (АЛУ)

Арифметико-логічний пристрій (АЛУ) виконує операції над вмістом регістрів, зокрема побітові, арифметичні та логічні операції, і записує результат назад у регістровий файл у визначений регістр.

Ці операції виконуються за один тактовий цикл. Кожна операція ALU впливає на прапори у STATUS-регістрі, залежно від виконуваної інструкції.

2.5 Доступ до пам'яті та виконання інструкцій

Процесор AVR працює від системного тактового сигналу, який може надходити ззовні або, якщо він доступний і активований, використовувати внутрішній RC-генератор. Цей тактовий сигнал без поділу використовується безпосередньо для всіх операцій усередині процесора.

Процесор має двоступеневий конвеєр, і вибірка-декодування інструкції виконується одночасно з виконанням попередньої інструкції.

Якщо вибрана інструкція пов'язана з АЛУ, вона може бути виконана за один тактовий цикл.

З іншого боку, доступ до SRAM займає два тактові цикли. Це пояснюється тим, що доступ до SRAM здійснюється за допомогою регістра-вказівника (X, Y або Z).

• Перший тактовий цикл використовується для доступу до файлу регістрів і виконання операцій над регістром-вказівником (інструкції доступу до SRAM можуть виконувати автоінкремент або автодекремент адреси вказівника).

• Наприкінці першого тактового циклу АЛУ виконує обчислення адреси.

• Другий тактовий цикл використовується для доступу до відповідної комірки SRAM і запису або зчитування даних у призначений регістр.

2.6 Пам'ять вводу-виводу

Пам'ять вводу-виводу є шлюзом до всіх периферійних компонентів процесора AVR. Вона реалізована як оперативна пам'ять (SRAM) і може бути доступна двома способами: як SRAM, а також як регістри вводу-виводу.

Як SRAM, адреси знаходяться в діапазоні від $20 до $5F, а як регістри вводу-виводу – від $00 до $3F.

Ми розглядатимемо регістри вводу-виводу саме як регістри, а не як SRAM.

Для доступу до регістрів вводу-виводу AVR надає інструкції **IN** і **OUT**. Ці інструкції можуть працювати з усіма регістрами вводу-виводу в діапазоні $00–$3F.

Крім **IN** і **OUT**, AVR також підтримує побітову адресацію для деяких регістрів у діапазоні $00–$1F. Завдяки інструкціям **SBI** (Set Bit in I/O Register) і **CBI** (Clear Bit in I/O Register) будь-який біт у регістрах цього діапазону можна встановити або скинути без необхідності читання регістра, зміни біта та запису зміненого значення назад у регістр. Це економить час порівнянні зі стандартним методом, який потребує приблизно втричі більше тактів процесора.

Для решти регістрів необхідно використовувати інший метод, який передбачає читання, зміну та запис значення, що займає більше тактів процесора.

Регістр стану (STATUS register, SREG). Регістр стану містить 8 бітів прапорців, які вказують на поточний стан процесора.

Після скидання всі біти встановлюються в нуль. Програма може як зчитувати, так і змінювати їх.

Адреса вводу-виводу регістра стану: $3F (Адреса в пам'яті: $5F)

Основні прапорці SREG і їхні функції (рис. 2.5):

Біт 7 (I): Глобальне увімкнення переривань

○ Якщо цей біт встановлений (1), усі переривання дозволені.

○ Якщо біт скинутий (0), усі переривання заборонені.

Біт 6 (T): Буферне збереження біта

○ Використовується разом з інструкціями BLD (Bit Load) і BST (Bit Store).

	7	6	5	4	3	2	1	0
I/O Address = $3F	I	T	H	S	V	N	Z	C
Initial Value	0	0	0	0	0	0	0	0

Рисунок 2.5 – Регістр стану SREG

○ Дозволяє завантаження та збереження окремих бітів між регістрами.

Біт 5 (H): Прапорець напівпереносу (Half Carry Flag)

○ Вказує на виникнення напівпереносу у деяких арифметичних операціях.

Біт 4 (S): Прапорець знаку (Sign Flag)

○ Обчислюється як виключне АБО (XOR) між прапорцями N (негативний результат) і V (переповнення).

Біт 3 (V): Прапорець переповнення у додатковому коді (Two's Complement Overflow Flag)

○ Вказує на переповнення при операціях у двійковому доповненні.

Біт 2 (N): Прапорець негативного результату (Negative Flag)

○ Встановлюється, якщо результат арифметичної або логічної операції має встановлений старший біт (означає від'ємне число у додатковому коді).

Біт 1 (Z): Прапорець нульового результату (Zero Flag)

○ Встановлюється, якщо після арифметичної або логічної операції отримано нульовий результат.

Біт 0 (C): Прапорець переносу (Carry Flag)

○ Вказує на перенесення (або позичку) при виконанні арифметичної або логічної операції.

Примітка: Регістр стану не зберігається автоматично під час обробки переривання. Оскільки інструкції в обробнику переривань можуть змінювати біти SREG, програміст повинен самостійно зберігати та відновлювати його значення під час роботи з перериваннями.

SP: РЕГІСТР УКАЗІВНИКА СТЕКУ. Цей регістр має ширину 1 байт для процесорів з обсягом SRAM до 256 байт і 2 байти (називаються SPH і SPL) для процесорів з обсягом SRAM понад 256 байт.

Цей регістр використовується для вказування області в SRAM, яка є вершиною стеку. Стек використовується процесором для збереження зворотних адрес під час виклику переривань і підпрограм. Оскільки SP ініціалізується значенням $00 (або $0000 для 2-байтового SP) під час скидання, користувацька програма повинна правильно ініціалізувати SP, оскільки початкова адреса SRAM не є $00. Початкова адреса SRAM – $60. Стек зростає вниз у пам'яті, тобто розміщення значення в стеку призводить до зменшення SP. Витягування значення зі стеку збільшує SP.

2.7 EEPROM

Усі контролери AVR мають вбудовану EEPROM. Кількість EEPROM варіюється від 0 байтів у ATTiny4 до 4 Кбайт у ATMega2560. EEPROM доступна через реєстри доступу до EEPROM, а саме: реєстр адреси EEPROM (EEAR), реєстр даних EEPROM (EEDR) і реєстр керування EEPROM (EECR).

Для пристроїв з EEPROM обсягом більше 256 байт, EEAR насправді є двома реєстрами, EEARL і EEARH. EEAR (як у вигляді одного реєстру, так і у вигляді двох реєстрів) використовується для встановлення адреси EEPROM, в яку буде записано дані або з якої дані будуть зчитані. EEAR – це реєстр для читання-запису, тобто реєстр можна читати, щоб побачити, яка адреса EEPROM була встановлена.

EEDR – це реєстр даних EEPROM і є реєстром для читання-запису. Коли потрібно записати дані в EEPROM, необхідно завантажити потрібні дані в EEDR. Коли потрібно зчитати дані з EEPROM, після завершення процесу зчитування потрібно прочитати EEDR для отримання даних. EECR містить необхідні контрольні біти для читання і запису в EEPROM. Запис в EEPROM не такий простий, як запис в SRAM, наприклад. Час доступу для запису в EEPROM

на контролерах AVR складає від 2,5 до 4,0 мс, в залежності від напруги живлення. Контрольний біт EEWE в EECR дозволяє користувачу виявити, коли раніше запитуване дані були записані в EEPROM і чи можна записати новий байт.

2.8 Порти вводу-виводу

Усі контролери AVR мають певну кількість портів вводу-виводу, що варіюється від 4 шт. у ATtiny4 до 86 шт. у ATMega2560. Усі вихідні порти контролерів AVR можуть споживати до 20 мА струму, що робить їх дуже зручними для прямого підключення світлодіодів без потреби у зовнішніх буферах.

Усі порти вводу-виводу мають три регістри вводу-виводу, пов'язані з ними. Три регістри необхідні для налаштування окремих бітів як вхід або вихід; інший регістр потрібна для виведення даних на ці (або всі) виводи, налаштовані як виходи, а третій регістр потрібен для зчитування даних з цих (або всіх) виводів, налаштованих як входи.

Ці регістри позначаються як DDRx, PORTx, PINx для відповідного порту x. DDRx – це реєстр напрямку даних. Запис «1» в біт DDR робить відповідний вивід вихідним на порті x. Після цього, щоб вивести «1» у вивід порту, відповідний біт можна встановити або скинути за допомогою інструкцій CBI чи SBI або інструкції OUT.

Аналогічно, для зчитування даних з вхідного виводу порту використовується реєстр PINx. Реєстр PINx безпосередньо підключений до виводу порту. Порти можна забезпечити внутрішнім підтягувальним резистором, записавши «1» в біт порту за адресою PORTx. Значення цього підтягувального резистора складає від 30 кОм до 150 кОм. Відповідне значення струму підтягування знаходиться в межах від 160 мкА до 33 мкА. Замість цього, якщо записати «0» в біт порту за адресою PORTx, підтягувальний резистор буде видалений, і вхідний пін залишиться в плаваючому стані з високим імпедансом.

2.9 Таймер

Таймер в контролері AVR може працювати як таймер або лічильник. Як таймер використовується внутрішній тактовий сигнал або його похідне для тактування таймера, а як лічильник – зовнішній сигнал на піні порту для тактування таймера-лічильника.

Блок-схема 8-бітного таймера-лічильника 0 зображена на рисунку 2.6.

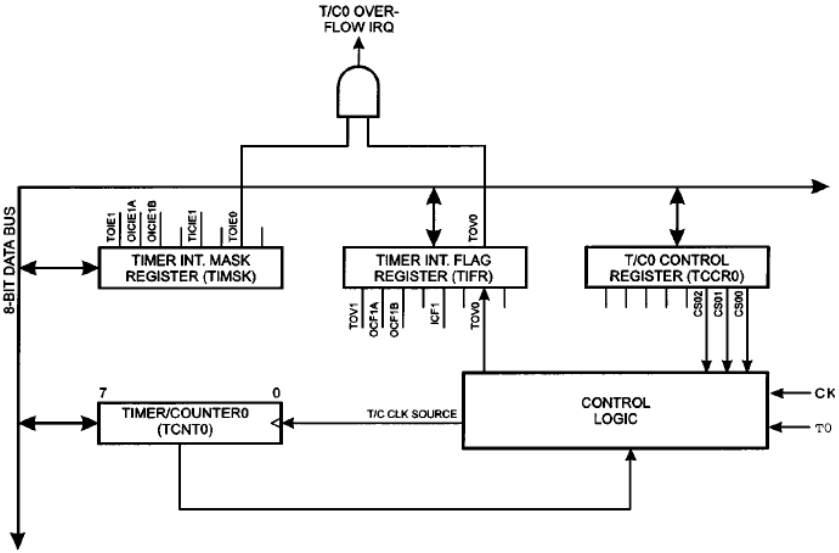


Рисунок 2.6 – Структурна схема 8-бітного таймера

8-бітний таймер-лічильник 0 може вибирати джерело тактового сигналу з CK, поділеного CK або зовнішнього піну. Крім того, його можна зупинити за допомогою керуючих бітів у реєстрі керування таймером-лічильником 0 (TCCR0).

Флаг переповнення знаходиться в регістрі флагів переривань таймера-лічильника TIFR. Керуючі сигнали знаходяться в регістрі керування таймером-лічильником 0 (TCCR0).

Налаштування увімкнення-вимкнення переривань для таймера-лічильника 0 знаходяться в регістрі маски переривань таймера-лічильника (TIMSK). Коли таймер-лічильник 0 тактується зовнішнім сигналом, зовнішній сигнал синхронізується з частотою осцилятора процесора. Для забезпечення правильного зчитування зовнішнього тактового сигналу мінімальний час між двома переходами зовнішнього тактового сигналу повинен бути не меншим за один внутрішній тактовий період процесора. Зовнішній тактовий сигнал зчитується на фронті підвищення внутрішнього тактового сигналу процесора. 8-бітний таймер-лічильник 0 має високу роздільну здатність і точність при використанні з нижчими можливостями переддільника.

2.10 Система переривань

Переривання – це механізм керування ходом програми, реалізований на більшості контролерів. У системі процесора, яка взаємодіє із зовнішнім світом, багато речей відбувається асинхронно, наприклад, користувач може натиснути перемикач, щоб виконати певну дію, тоді як на послідовний порт може надійти байт даних. Для процесора було б абсолютно неможливо відстежувати всі речі, просто запитуючи ці пристрої для отримання даних. Натомість було б краще, якби ці пристрої могли «оголошувати» про надходження даних. Це те, що робить механізм переривання. Периферійний пристрій може «перервати» виконання основної програми, а процесор бере час для нормального виконання програми, щоб перевірити джерело переривання та вжити необхідних заходів. Після виконання необхідної дії перерване виконання програми відновлюється. Програма переривання схожа на підпрограму, за винятком того, що виконання цієї підпрограми переривання не передбачається процесором у певний момент часу.

AVR має розвинену систему переривань. Для більшості периферійних пристроїв надано можливість переривання, тому головній програмі не потрібно постійно опитувати ці пристрої.

Послідовність подій при виникненні переривання така:

1. Периферійний пристрій перериває роботу процесора.
2. Виконання поточної інструкції завершується.
3. Адреса наступної інструкції зберігається в стеку (апаратному або програмному).
4. Адреса ISR (підпрограми переривання) завантажується в лічильник програми.
5. Процесор виконує ISR.
6. Про завершення виконання ISR вказує інструкція RETI (повернення з переривання).
7. Процесор завантажує лічильник програми зі значенням, що зберігається в стеку, і нормальне виконання програми відновлюється.

Оскільки переривання може відбутися в будь-який момент, стан процесора (прапори тощо) повинен бути збережений, щоб нормальне виконання програми могло відновитися після завершення ISR. Статус процесора міститься в регістрі SREG. ISR має зберегти SREG перед виконанням будь-якої іншої інструкції, а перед поверненням керування головній програмі має відновити регістр SREG. Це можна зробити двома способами: або SREG копіюється в інший регістр, скажімо, R1, який не можна використовувати для будь-яких інших цілей, і перед тим, як ISR виконає інструкцію RETI, R1 копіюється назад в SREG. Іншим способом збереження SREG є збереження його в стеку (за допомогою інструкції PUSH SREG), а потім перед виконанням інструкції RETI значення SREG копіюється зі стеку (за допомогою інструкції POP SREG). Цей спосіб можливий тільки для тих процесорів, які мають програмний стек.

На рис. 2.7 показано, як переривається основна програма. Також можна перервати ISR, якщо виникає інше переривання, а глобальний прапор переривання встановлено на «1» у ISR для interrupt1 (за допомогою інструкції SEI). У цьому випадку ISR1 переривається, і виконується інший ISR, ISR2. Виконання ISR1 відновлюється після завершення ISR2, а після завершення ISR1 основна програма відновлює виконання.

Зазвичай після того, як переривання виникає та обслуговується відповідним ISR, глобальні переривання вимикаються автоматично

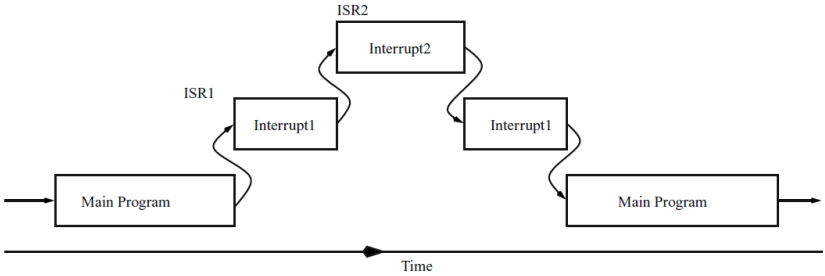


Рисунок 2.7 – Виконання вкладених переривань

(еквівалентно виконанню інструкції CLI); однак, можна ввімкнути переривання під час виконання ISR, виконавши інструкцію SEI в ISR. Якщо інше переривання відбувається протягом часу, коли ISR вже працює, тоді воно буде обслуговуватися шляхом переривання початкового ISR. Пріоритет переривань визначається способом призначення векторів переривань. Вектор переривання за нижчою адресою в пам'яті програми має вищий пріоритет.

Пріоритет переривання використовується для визначення того, яке переривання обслуговується першим, якщо в будь-який момент часу очікується більше одного переривання. Така ситуація може виникнути, якщо глобальні переривання були вимкнені в системі, щоб дозволити виконання деяких критичних розділів програми. Після завершення критичної секції програма вмикає глобальні переривання. Тепер, під час виконання критичної секції, виникає два переривання, зовнішнє Interrupt0 і UART Rx Complete. Тоді, оскільки зовнішнє Interrupt0 має вищий пріоритет, ніж переривання UART, буде виконано ISR, що відповідає зовнішньому Interrupt0, а після цього буде виконано ISR для переривання UART.

Дуже важливий фактор під час використання переривань – це те, наскільки швидко процесор може реагувати на переривання. Це багато в чому залежить від архітектури процесора. Для контролерів AVR відповідь на виконання переривання для всіх увімкнених переривань AVR становить мінімум чотири такти. Через чотири такти після встановлення прапора переривання виконується адреса програмного вектора для фактичної процедури обробки

переривання. Протягом цього чотиритактового періоду програмний лічильник (2 байти) надсилається в стек, а вказівник стека зменшується на 2. Вектор зазвичай є відносним переходом до процедури переривання, і цей стрибок займає два тактові цикли. Якщо переривання виникає під час виконання багатоциклової інструкції, ця інструкція завершується до того, як переривання обслуговується. Повернення з процедури обробки переривань займає чотири такти. Протягом цих чотирьох тактових циклів програмний лічильник (2 байти) повертається зі стеку, покажчик стека збільшується на 2, а прапор I у SREG встановлюється. Коли AVR виходить із переривання, він завжди повертатиметься до основної програми та виконуватиме ще одну інструкцію, перш ніж буде обслуговано будь-яке очікуване переривання.

2.11 Вбудований сторожовий таймер (Watchdog)

Сторожовий таймер – це керований таймер, який використовується як пристрій для скидання, якщо програмне забезпечення потрапило в нескінченний цикл або виникла помилка виконання програми. Сторожовий таймер має вихід, який може скинути контролер. На рисунку 2.8 показано блок-схему сторожового таймера.

Сторожовий таймер працює від окремого внутрішнього RC осцилятора. Регулюючи переддільник сторожового таймера, можна налаштувати інтервал скидання сторожового таймера. Інтервали скидання сторожового таймера також залежать від напруги живлення.

Інструкція скидання сторожового таймера (WDR) скидає сторожовий таймер. Можна вибрати вісім різних періодів тактових циклів для визначення періоду скидання. Якщо період скидання спливає без іншого скидання сторожовим таймером, контролер AVR скидається і починає виконувати програму з вектора скидання. Для запобігання ненавмисному вимкненню сторожового таймера потрібно дотримуватися спеціальної послідовності вимкнення, як описано в розділі про регістр керування сторожовим таймером.

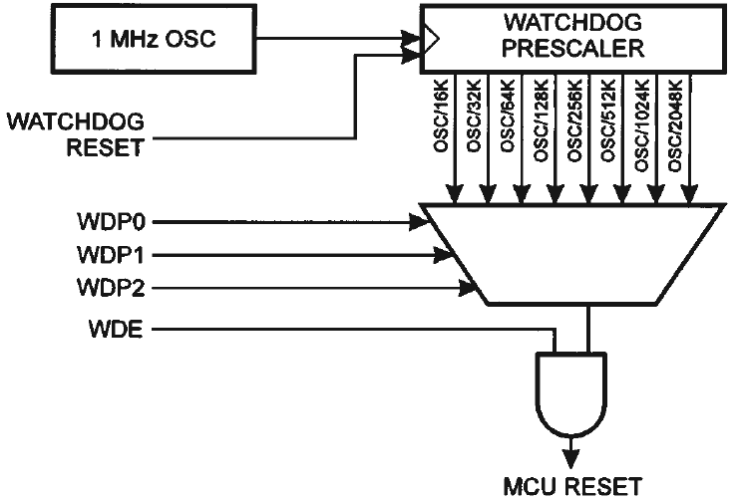


Рисунок 2.8 – Блок-схема сторожового таймера

2.12 Режими роботи з низьким споживанням енергії

Контролер AVR пропонує різноманітні схеми зниження споживаної потужності. Для того щоб увійти в режими сну, потрібно встановити біт SE в регістрі MCUCR (встановити в 1) і виконати інструкцію SLEEP. Якщо під час перебування в режимі сну виникає переривання, контролер AVR прокидається, виконує обробку переривання та продовжує виконання з інструкції, що йде після SLEEP. Вміст регістра файлу, SRAM і пам'яті I/O не змінюється. Якщо під час режиму сну відбувається скидання, контролер AVR прокидається і виконує програму з вектору скидання.

Коли біт SM очищено (0), інструкція SLEEP змушує MCU перейти в режим простою, зупиняючи процесор, але дозволяючи працювати таймерам-лічильникам, сторожовому таймеру та системі переривань. Це дозволяє MCU прокидатися від зовнішніх переривань, а також від внутрішніх, таких як переривання від

переповнення таймера та скидання сторожовим таймером. Якщо прокидання від переривання аналого-цифрового компаратора не потрібно, то компаратор можна вимкнути, встановивши біт ACD в регістрі керування і стану аналого-цифрового компаратора ACSR. Це дозволить знизити споживання енергії в режимі простою. Коли MCU прокидається з режиму простою, процесор починає виконувати програму без затримок.

Коли біт SM встановлено (1), інструкція SLEEP змушує MCU перейти в режим вимкнення. У цьому режимі зовнішній осцилятор вимикається, в той час як зовнішні переривання та сторожовий таймер (якщо він увімкнений) продовжують працювати. Прокинути MCU може тільки зовнішнє скидання, скидання сторожовим таймером (якщо увімкнений) або зовнішнє переривання рівня на INT0 чи INT1. Зауважте, що коли для прокидання з режиму вимкнення використовується переривання рівня, то низький рівень повинен триматися довше, ніж час затримки скидання t_{TOUT}. В іншому випадку пристрій не прокинеться.

Контрольні запитання до теми 2

1. Які основні характеристики архітектури AVR-RISC у мікроконтролері ATMega8?
2. Які типи пам'яті використовуються у мікроконтролері ATMega8 та їх основні характеристики?
3. Які типи таймерів-лічильників інтегровані у мікроконтролер ATMega8?
4. Які спеціальні мікроконтролерні функції підтримуються мікроконтролером ATMega8?
5. Які робочі частоти та напруги підтримує мікроконтролер ATMega8?
6. Які особливості Гарвардської архітектури використовує мікроконтролер ATMega8?
7. Як реалізується конвеєрна обробка команд у мікроконтролері ATMega8?
8. Які функції виконують регістри X, Y, Z у мікроконтролері ATMega8?
9. Як здійснюється робота з регістром стану під час виклику процедури обробки переривань?
10. Які способи адресації підтримує пам'ять даних SRAM у мікроконтролері ATMega8?

11. Які модулі ядра AVR синхронізуються за допомогою тактового сигналу clkcpu?
12. Як здійснюється робота зовнішніх переривань в умовах відсутності тактового сигналу clkI/O?
13. Які джерела тактового сигналу можуть бути використані у мікроконтролері, і як їх вибір здійснюється?
14. Які три спеціальні регістри має кожний порт вводу-виводу мікроконтролера ATMega8, і яке їх основне призначення?
15. Як впливає запис логічної одиниці у будь-який розряд регістра PINxn на відповідний розряд регістру даних PORTxn?
16. Що таке вектор переривань у мікроконтролері AVR, і які особливості його використання?
17. Які основні резервовані адреси програмної пам'яті в мікроконтролері ATMega8 і яке призначення у них?
18. Як відрізняються внутрішні та зовнішні переривання в мікроконтролерах?
19. Що відбувається, якщо в програмі для мікроконтролера не використовуються механізми переривань?
20. Яким чином встановлюються адреси векторів переривань у мікроконтролері ATMega8?
21. Які основні функції виконують конфігураційні комірки (Fuse Bits) у мікроконтролері ATMega8?
22. Які параметри можна змінювати за допомогою конфігураційних комірок мікроконтролера?
23. Які переваги має використання конфігураційних комірок Fuse Bits у порівнянні з програмованою пам'яттю EEPROM чи Flash?
24. Як впливає стан незапрограмованих Fuse-комірок (які містять одиницю) на роботу мікроконтролера ATMega8?
25. Як відбувається програмування та перепрограмування конфігураційних комірок у мікроконтролері?

Використана література

1. Розробка радіоелектронних схем на основі мікроконтролерів (на прикладі AVR мікроконтролерів фірми Atmel) : методичний посібник до курсу «Проектування радіоелектронних схем» для студентів радіофізичного факультету / уклад.: Д. А. Пархоменко, Є. М. Смирнов. Київ : Радіофізичний факультет Київського національного університету імені Тараса Шевченка, 2013. 74 с.
2. Методичний посібник для виконання лабораторних робіт з курсу «Мікропроцесорна техніка» для студентів факультету радіофізики,

електроніки та комп'ютерних систем / упоряд.: А. І. Білецький та ін.; за ред. А. М. Веклича. Київ : ВЦ «Київський університет», 2021. 40 с.
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. 57 с.

3 ЦИФРОВІ ПОРТИ ВВОДУ-ВИВОДУ МІКРОКОНТРОЛЕРІВ СІМЕЙСТВА AVR

Метою вивчення теми є ознайомлення з цифровими портами вводу-виводу мікроконтролерів сімейства AVR, а саме з їх конфігураціями виводів, альтернативними функціями портів, читанням стану виводу та бітовими операціями.

Завдання вивчення теми збігаються з переліком питань для розгляду, що наведений нижче.

Перелік питань до розділу:

3.1. Конфігурація виводів.
3.2. Читання стану виводу.
3.3. Альтернативні функції портів вводу-виводу.
3.4. Опис регістрів вводу-виводу.
3.5. Бітові операції.

3.1 Конфігурація виводів

Усі порти AVR мають функціональність читання – зміни – запису при використанні їх як загальних цифрових портів вводу-виводу. Це означає, що напрямок одного виводу порту можна змінити без випадкової зміни напрямку будь-якого іншого виводу за допомогою інструкцій SBI та CBI. Те саме стосується зміни значення вихідної напруги (якщо вивід налаштовано як вихід) або ввімкнення/вимкнення підтягувальних резисторів (якщо вивід налаштовано як вхід). Кожен вихідний буфер має симетричні характеристики виходу з високою навантажувальною здатністю. Драйвер порту достатньо потужний, щоб керувати світлодіодними індикаторами безпосередньо. Усі виводи порту мають індивідуальні

підтягуювальні резистори з опором, що не залежить від напруги живлення. Усі контакти вводу-виводу мають захисні діоди як для VCC, так і для заземлення, як показано на рисунку 3.1.

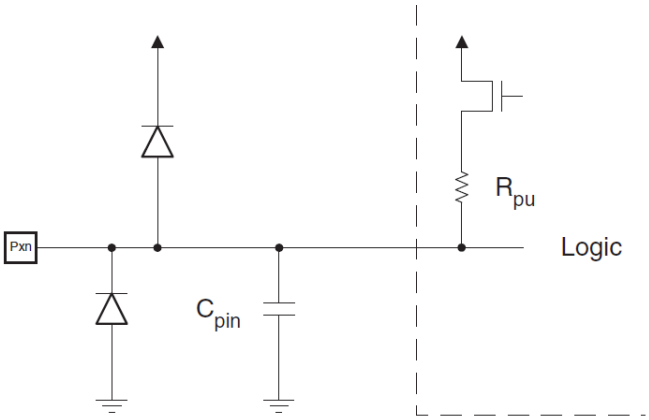


Рисунок 3.1 – Схема виводів портів вводу-виводу

Усі регістри та посилання на біти в цьому розділі записані в загальному вигляді. Мала літера "x" представляє номер порту, а мала літера "n" представляє номер біта. Однак, коли в програмі використовується визначення регістру або біту, необхідно використовувати точну форму (тобто PORTB3 для біта 3 у порту B, тут записано у загальному вигляді як PORTxn).

Для кожного порту виділено три адреси у пам'яті вводу-виводу, по одній для регістра даних – PORTx, регістра напрямку даних – DDRx і вхідного стану порту – PINx. Регістр вхідного стану порту призначений лише для читання, тоді як регістр даних і регістр напрямку даних доступні для читання / запису. Крім того, біт "Pull-up Disable" – PUD у регістрі SFIOR вимикає функцію підтягування для всіх контактів у всіх портах, якщо він встановлений, як 1.

Більшість контактів порту мультиплексовано з альтернативними функціями периферійних пристроїв мікроконтролера.

Порти є двонаправленими портами вводу-виводу з додатковими внутрішніми підтягувальними резисторами. На рисунку 3.2 показано функціональний опис одного контакту порту вводу-виводу, який тут узагальнено називається Pxn.

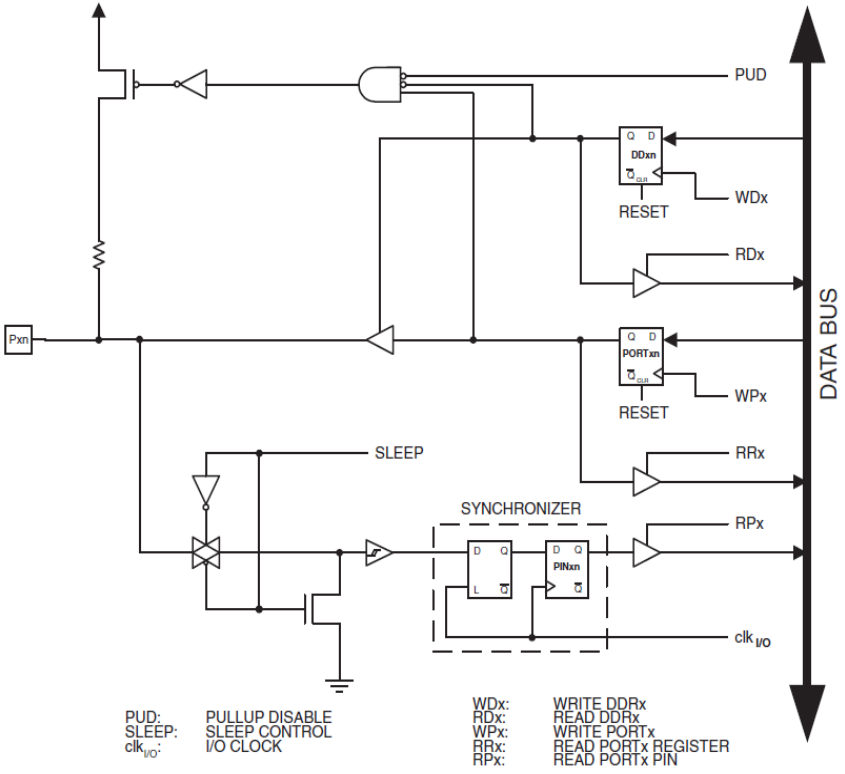


Рисунок 3.2 – Функціональний опис одного контакту порту вводу-виводу

Кожен вивід порту визначається бітами трьох регістрів: DDxn, PORTxn і PINxn. Доступ до бітів DDxn здійснюється за адресою регістру вводу-виводу DDRx, до бітів PORTxn – за адресою регістру вводу-виводу PORTx, а до бітів PINxn – за адресою регістру вводу-виводу PINx.

Біт DDxn у регістрі DDRx вибирає напрямок відповідного виводу. Якщо у DDxn записується логічна одиниця, Pxn налаштовується, як вихідний контакт. Якщо DDxn у записується логічний нуль, Pxn налаштовується, як вхідний контакт.

Якщо у PORTxn записується логічна одиниця, коли вивід налаштовано як вхідний, активується підтягувальний резистор. Щоб вимкнути підтягувальний резистор, у PORTxn має бути записаний логічний нуль або вивід має бути налаштований як вихід. Виводи порту переводяться у високоімпедансний стан в режимі скиду, навіть якщо тактовий генератор не працює.

Якщо у PORTxn записується логічна одиниця, коли вивід налаштовано як вихід, то вивід порту має високий рівень напруги (одиниця). Якщо у PORTxn записується логічний нуль, коли вивід налаштовано як вихід, на виводі порту встановлюється низький рівень напруги (нуль).

Під час перемикання між високоімпедансним станом ({DDxn, PORTxn} = 0b00) і виходом з високим рівнем напруги ({DDxn, PORTxn} = 0b11), проміжним станом є або стан з увімкненим підтягуванням ({DDxn, PORTxn} = 0b01), або вихід з низьким рівнем напруги ({DDxn, PORTxn} = 0b10). Зазвичай стан з активним підтягування є цілком прийнятним, оскільки схема з високим вхідним імпедансом не помітить різниці між виходом з високим рівнем напруги і підтягуванням. Якщо це не прийнято, біт PUD у регістрі SFIOR можна встановити так, щоб відключити всі підтягування у всіх портах.

Перемикання між входом із підтягуванням і виходом з низьким рівнем напруги створює ту саму проблему. Користувач повинен або використовувати високоімпедансний стан ({DDxn, PORTxn} = 0b00), або вихідний стан з високою напругою ({DDxn, PORTxn} = 0b11) як проміжний крок.

Таблиця 3.1 підсумовує контрольні сигнали для значення контакту.

Таблиця 3.1 – Контрольні сигнали для значення контакту

DDxn	PORTxn	PUD (у SFIOR)	Стан виводу	Підтягування	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

3.2 Читання стану виводу

Незалежно від налаштування біта напрямку даних DDxn, стан виводу порту можна прочитати через біт регістру PINxn. Як показано на рис. 3.3, біт регістра PINxn і попередній тригер утворюють синхронізатор. Це необхідно, щоб уникнути метастабільності, якщо вивід змінює фізичне значення біля зміни сигналу внутрішнього тактового генератора, але це також створює затримку.

На рис. 3.4 показано часову діаграму синхронізації під час зчитування зовнішнього значення виводу. Максимальна та мінімальна затримки поширення позначаються $t_{pd,max}$ та $t_{pd,min}$ відповідно.

Припустимо, що зміна стану виводу відбувається незабаром після першого спаду фронту системного тактового сигналу. Тригер закривається, коли тактовий сигнал має низький рівень, і відкривається, коли тактовий сигнал має високий рівень, на що вказує заштрихована область сигналу “SYNC LATCH”. Значення сигналу фіксується, коли системний тактовий сигнал стає низьким.

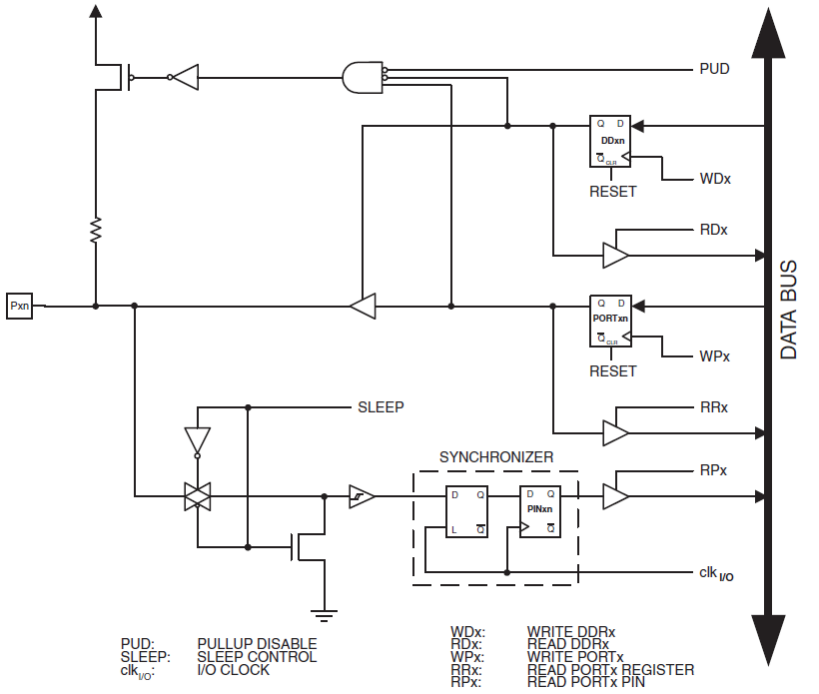


Рисунок 3.3 – Спрощена схема порту вводу-виводу

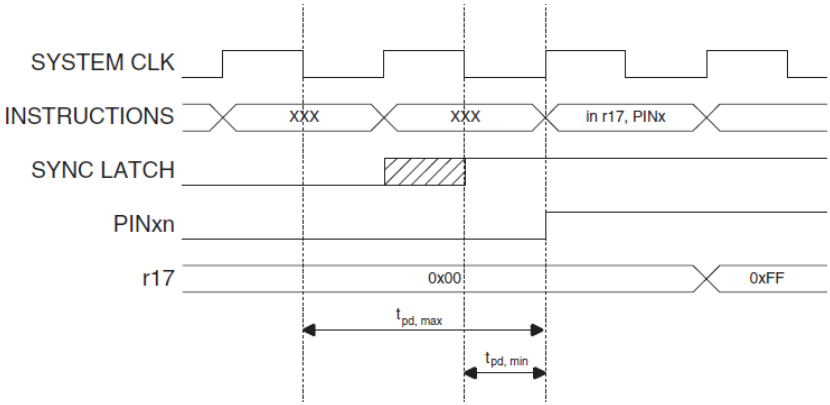


Рисунок 3.4 – Часова діаграма синхронізації під час зчитування зовнішнього значення виводу

Він синхронізується в регістрі PINxn на наступному позитивному фронті тактового сигналу. Як вказано двома стрілками $t_{pd,max}$ і $t_{pd,min}$, передача сигналу на виводі може бути затримана між ½ і 1½ системного тактового періоду залежно від часу виникнення.

Під час зчитування програмно встановленого значення піна необхідно вставити інструкцію nop, як показано на рис. 3.5. Інструкція out встановлює сигнал “SYNC LATCH” на позитивному фронті тактового сигналу. У цьому випадку затримка t_{pd} через синхронізатор становить 1 системний такт.

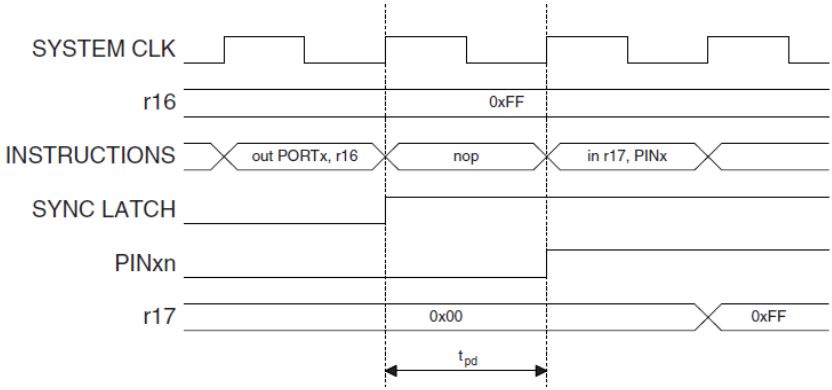


Рисунок 3.5 – Вставлення інструкції nop під час зчитування програмно встановленого значення піна

Робота цифрових входів у режимі сну. Як показано на рисунку 3.3, цифровий вхідний сигнал може бути заземлений на вході тригера Шмідта. Сигнал, позначений на рисунку 3.3 – SLEEP, встановлюється контролером сну MCU у режимі вимкнення живлення, режимі енергозбереження та режимі очікування, щоб уникнути високого енергоспоживання, якщо деякі вхідні сигнали залишаються плаваючими або мають рівень аналогового сигналу, близький до VCC/2.

Сигнал SLEEP нехтується для контактів порту, увімкнених як контакти зовнішнього переривання. Якщо запит зовнішнього

переривання не ввімкнено, сигнал SLEEP активний також і для цих контактів. SLEEP також перекривається різними іншими альтернативними функціями.

Непідключені виводи. Якщо деякі виводи не використовуються, рекомендується переконатися, що ці виводи мають певний рівень. Незважаючи на те, що більшість цифрових входів вимкнено в режимах глибокого сну, як описано вище, слід уникати плаваючих входів, щоб зменшити споживання струму в усіх інших режимах, де цифрові входи ввімкнено (скид, активний режим і режим очікування).

Найпростіший спосіб забезпечити певний рівень невикористаного виводу – увімкнути внутрішнє підтягування. Але у цьому випадку підтягування буде вимкнено під час скиду. Якщо важливе низьке енергоспоживання і під час скиду, рекомендується використовувати зовнішнє підтягування. Підключати невикористовувані контакти безпосередньо до VCC або GND не рекомендується, оскільки це може спричинити надмірні струми, якщо вивід буде випадково налаштовано як вихід.

3.3 Альтернативні функції портів вводу-виводу

Більшість виводів портів мають альтернативні функції на додаток до загальних цифрових входів/виходів. На рисунку 3.6 показано, як сигнали керування контактами порту можуть бути замінені альтернативними функціями. Альтернативні сигнали можуть бути присутні не на всіх контактах порту, але рисунок 3.6 служить загальним описом, застосовним до всіх контактів порту сімейства мікроконтролерів AVR.

Альтернативні функції порту B:
XTAL2/TOSC2 – порт B, біт 7

XTAL2: Вивід 2 генератора тактового сигналу мікросхеми. Використовується як вивід підключення кварцового або низькочастотного кварцового резонатора. Якщо використовується як вивід тактового сигналу, його не можна використовувати як контакт вводу-виводу.

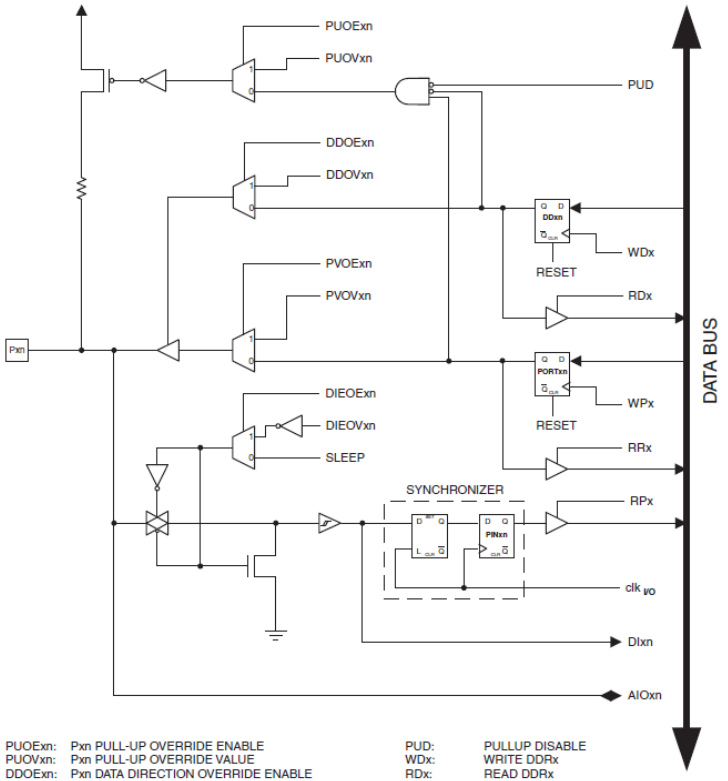


Рисунок 3.6 – Приклад заміщення сигналів керування контактами порту на альтернативні функції

TOSC2: Вивід 2 генератора таймера. Використовується, лише якщо внутрішній калібрований RC-генератор вибрано як джерело тактування мікросхеми, а асинхронний таймер увімкнено за допомогою правильного налаштування в ASSR. Коли біт AS2 в ASSR встановлено (один), щоб увімкнути асинхронне тактування таймеру Timer/Counter2, вивід PB7 від'єднується від

порту та стає інвертуючим виходом підсилювача генератора. У цьому режимі кварцовий генератор підключено до цього контакту, і цей контакт не можна використовувати як контакт введення/виведення.

Якщо PB7 використовується як вивід тактування, DDB7, PORTB7 і PINB7 читатимуться, як 0.

XTAL1/TOSC1 – порт B, біт 6

XTAL1: Вивід 1 генератора тактового сигналу мікросхеми. Використовується для всіх джерел тактового сигналу мікросхеми, крім внутрішнього каліброваного RC-генератора. Якщо використовується як вивід тактування, його не можна використовувати як вивід вводу-виводу.

TOSC1: Вивід 1 генератора таймера. Використовується, лише якщо внутрішній калібрований RC-генератор вибрано як джерело тактування мікросхеми, а асинхронний таймер увімкнено за допомогою правильного налаштування в ASSR. Коли біт AS2 в ASSR встановлено (один), щоб увімкнути асинхронне тактування Timer/Counter2, вивід PB6 від'єднується від порту та стає входом підсилювача інвертувального генератора. У цьому режимі до цього контакту підключається кварцовий генератор, і цей контакт не можна використовувати як контакт введення/виведення.

Якщо PB6 використовується як вивід тактування, DDB6, PORTB6 і PINB6 читатимуться як 0.

SCK – порт B, біт 5

SCK: Вихід тактового сигналу головного або вхід тактового сигналу підлеглого приладу шини SPI. Коли SPI увімкнено як підлеглий, цей контакт налаштовано як вхід незалежно від налаштування DDB5. Коли SPI увімкнено як головний, напрям даних цього контакту контролюється DDB5. Коли SPI змушує вивід бути входом, підтягування все ще керується бітом PORTB5.

MISO – порт B, біт 4

MISO: вхід даних головного або вихід даних підлеглого приладу шини SPI. Коли SPI увімкнено як головний, цей контакт налаштовано як вхід незалежно від налаштування DDB4. Коли SPI увімкнено як підлеглий, напрям даних цього контакту контролюється

DDB4. Коли SPI змушує вивід бути входом, підтягування все ще керується бітом PORTB4.

MOSI/OC2 – порт B, біт 3

MOSI: вихід даних головного або вхід даних підлеглого приладу шини SPI. Коли SPI увімкнено як підлеглий, цей контакт налаштовано як вхід незалежно від налаштування DDB3. Коли SPI увімкнено як головний, напрям даних цього контакту контролюється DDB3. Коли SPI змушує вивід бути входом, підтягування все ще керується бітом PORTB3.

OC2, вихід збігу при порівнянні: вивід PB3 може служити зовнішнім виходом для збігу при порівнянні таймера-лічильника 2. Вивід PB3 має бути налаштований як вихід (DDB3 встановлено (один)), щоб виконувати цю функцію. Вивід OC2 також є вихідним виводом для функції ШІМ таймера.

SS/OC1B – порт B, біт 2

SS: Вибір підлеглого приладу шини SPI. Коли SPI увімкнено як підлеглий, цей контакт налаштовано як вхід, незалежно від налаштування DDB2. Як підлеглий, SPI активується, коли на цьому виводі знаходиться низький рівень. Коли SPI увімкнено як головний, напрям даних цього контакту контролюється DDB2. Коли SPI змушує вивід бути входом, підтягуванням все ще можна керувати бітом PORTB2.

OC1B, вихід збігу при порівнянні: Вивід PB2 може служити зовнішнім виходом для збігу при порівнянні B таймера-лічильника 1. Вивід PB2 має бути налаштований як вихід (DDB2 встановлений (один)), щоб виконувати цю функцію. Вивід OC1B також є вихідним виводом для таймера в режимі ШІМ.

OC1A – порт B, біт 1

OC1A, вихід збігу при порівнянні: вивід PB1 може служити зовнішнім виходом для збігу при порівнянні B таймера-лічильника 1. Вивід PB1 має бути налаштований як вихід (DDB1 встановлений (один)), щоб виконувати цю функцію. Вивід OC1A також є вихідним виводом для таймера в режимі ШІМ.

ICP1 – порт B, біт 0

ICP1 – Вивід захоплення вхідного сигналу: вивід PB0 може діяти як вхід захоплення вхідного сигналу для таймера-лічильника 1.

Таблиця 3.2 – Альтернативні функції порту B

Вивід	Альтернативні функції
PB7	XTAL2 (вивід 2 тактового генератора мікросхеми) TOSC2 (вивід 2 генератора таймера)
PB6	XTAL1 (вивід 1 генератора тактового сигналу або вхід зовнішнього тактового сигналу) TOSC1 (вивід 1 генератора таймера)
PB5	SCK (вхід/вихід тактового сигналу шини SPI)
PB4	MISO (вхід головного/вихід підлеглого приладу шини SPI)
PB3	MOSI (вихід головного/вхід підлеглого приладу шини SPI) OC2 (вихід збігу при порівнянні таймера-лічильника 2)
PB2	SS (вхід/вихід вибору підлеглого приладу шини SPI) OC1B (вихід збігу при порівнянні B таймера-лічильника 1)
PB1	OC1A (вихід збігу при порівнянні A таймера-лічильника 1)
PB0	ICP1 (вхід захоплення сигналу таймера-лічильника 1)

Альтернативні функції порту C:

RESET – порт C, біт 6

RESET, вхід скиду: коли запобіжник RSTDISBL запрограмовано, цей контакт функціонує як звичайний контакт вводу-виводу, і мікроконтролер повинен покладатися на скид при включенні живлення та скид при зниженні напруги, як на джерела скиду. Коли запобіжник RSTDISBL не запрограмований, схема скиду підключається до виводу, і його не можна використовувати як контакт вводу-виводу. Якщо PC6 використовується як вхід скиду, DDC6, PORTC6 і PINC6 читатимуться як 0.

SCL/ADC5 – порт C, біт 5

SCL, тактовий сигнал двопровідного послідовного інтерфейсу TWI: коли біт TWEN у TWCR встановлено (один) для ввімкнення TWI, вивід PC5 від'єднується від порту та стає контактом вводу-виводу послідовного тактового сигналу для двопровідного інтерфейсу. У цьому режимі на виводі є фільтр для придушення викидів напруги, коротших за 50 нс, у вхідному сигналі, а вивід керується драйвером з відкритим стоком з обмеженням швидкості наростання.

PC5 також можна використовувати як вхідний канал АЦП 5. Зауважте, що вхідний канал АЦП 5 використовує цифрове живлення.

SDA/ADC4 – порт C, біт 4

SDA, дані двопровідного послідовного інтерфейсу TWI: коли біт TWEN у TWCR встановлено (один) для ввімкнення TWI, вивід PC4 від'єднується від порту та стає контактом вводу-виводу послідовних даних для двопровідного інтерфейсу. У цьому режимі на виводі є фільтр для придушення викидів напруги, коротших за 50 нс, у вхідному сигналі, а вивід керується драйвером відкритого стоку з обмеженням швидкості наростання.

PC4 також можна використовувати як вхідний канал АЦП 4. Зауважте, що вхідний канал 4 АЦП використовує цифрове живлення.

ADC3 – порт C, біт 3

PC3 також можна використовувати як вхідний канал АЦП 3. Зауважте, що вхідний канал 3 АЦП використовує аналогове живлення.

ADC2 – порт C, біт 2

PC2 також можна використовувати як вхідний канал АЦП 2. Зауважте, що вхідний канал 2 АЦП використовує аналогове живлення.

ADC1 – порт C, біт 1

PC1 також можна використовувати як вхідний канал АЦП 1. Зауважте, що вхідний канал АЦП 1 використовує аналогове живлення.

ADC0 – порт C, біт 0

PC0 також можна використовувати як вхідний канал АЦП 0. Зауважте, що вхідний канал АЦП 0 використовує аналогове живлення.

Таблиця 3.3 – Альтернативні функції порту C

Вивід	Альтернативні функції
PC6	RESET (вхід скиду)
PC5	ADC5 (вхідний канал АЦП 5) SCL (лінія тактування шини TWI)
PC4	ADC4 (вхідний канал АЦП 4) SDA (лінія даних шини TWI)
PC3	ADC3 (вхідний канал АЦП 3)
PC2	ADC2 (вхідний канал АЦП 2)
PC1	ADC1 (вхідний канал АЦП 1)
PC0	ADC0 (вхідний канал АЦП 0)

Альтернативні функції порту D:

AIN1 – порт D, біт 7

AIN1, негативний вхід аналогового компаратора. Налаштуйте контакт порту як вхід із вимкненим внутрішнім підтягуванням, щоб функція цифрового порту не перешкоджала роботі аналогового компаратора.

AIN0 – порт D, біт 6

AIN0, позитивний вхід аналогового компаратора. Налаштуйте контакт порту як вхід із вимкненим внутрішнім підтягуванням, щоб функція цифрового порту не перешкоджала роботі аналогового компаратора.

T1 – порт D, біт 5

T1, вхід зовнішнього лічильника таймера-лічильника 1.

XCK/T0 – порт D, біт 4

XCK, зовнішній сигнал тактування USART.

T0, вхід зовнішнього лічильника таймера-лічильника 0.

INT1 – порт D, біт 3

INT1, зовнішнє джерело переривання 1: вивід PD3 може служити зовнішнім джерелом переривання.

INT0 – порт D, біт 2

INT0, зовнішнє джерело переривання 0: вивід PD2 може служити зовнішнім джерелом переривання.

TXD – порт D, біт 1

TXD, передача даних (вихід даних для USART). Коли передавач USART увімкнено, цей контакт налаштовано як вихід незалежно від значення DDD1.

RXD – порт D, біт 0

RXD, прийом даних (контакт для вводу даних для USART). Коли приймач USART увімкнено, цей контакт налаштовано як вхід незалежно від значення DDD0. Коли USART змушує цей вивід бути входом, підтягування все ще може контролюватися бітом PORTD0.

Таблиця 3.4 – Альтернативні функції D

Вивід	Альтернативні функції
PD7	AIN1 (негативний вхід аналогового компаратора)
PD6	AIN0 (позитивний вхід аналогового компаратора)
PD5	T1 (вхід зовнішнього лічильника таймера-лічильника 1)
PD4	XCK (вхід/вихід зовнішнього сигналу тактування USART) T0 (вхід зовнішнього лічильника таймера-лічильника 0)
PD3	INT1 (вхід зовнішнього переривання 1)
PD2	INT0 (вхід зовнішнього переривання 0)
PD1	TXD (вихідний контакт USART)
PD0	RXD (вхідний контакт USART)

3.4 Опис регістрів вводу-виводу

На рисунках 3.7–3.9 показані регістри вводу-виводу, що використовуються для керуванням портами вводу-виводу мікроконтролера ATMega8. Як можна бачити, всі вони є однотипними і відрізняються лише останньою літерою, що відповідає назві порту.

У порті C старший біт регістрів DDRC, PORTC та PINC завжди читається як нуль, оскільки вивід PC7 відсутній у цьому мікроконтролері.

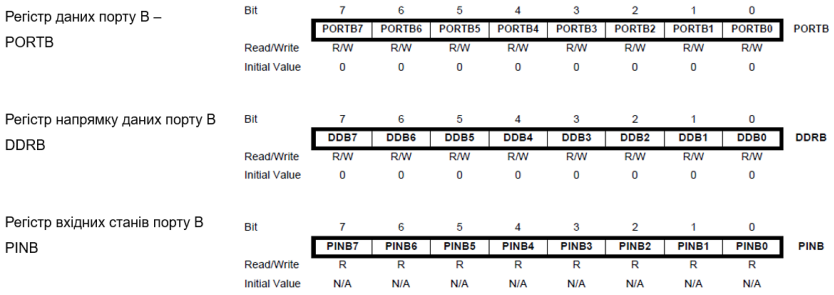


Рисунок 3.7 – Регістри вводу-виводу порту B

Регістр даних порту C – PORTC

Bit	7	6	5	4	3	2	1	0	
	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	PORTC
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Регістр напрямку даних порту C DDRC

Bit	7	6	5	4	3	2	1	0	
	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Регістр вхідних станів порту C PINC

Bit	7	6	5	4	3	2	1	0	
	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Рисунок 3.8 – Регістри вводу-виводу порту C

Регістр даних порту D – PORTD

Bit	7	6	5	4	3	2	1	0	
	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
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	

Регістр напрямку даних порту D DDRD

Bit	7	6	5	4	3	2	1	0	
	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
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	

Регістр вхідних станів порту D PIND

Bit	7	6	5	4	3	2	1	0	
	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Рисунок 3.9 – Регістри вводу-виводу порту D

3.5 Бітові операції

Біти та байти

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.

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
uint8_t b = 0x0F; // шістнадцяткове
uint8_t c = 0b00001111; // двійкове
```

uint8_t – один з типів даних рівноширокого цілого типу, зі стандарту С99. Стандарт передбачає 8-бітний беззнаковий цілий тип. Типи даних стандарту С99 будуть і надалі використовуватись у цьому посібнику.

Бітові операції. 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 С 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 С 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 С 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 операнду, а не двох, як інші. NOT просто перетворює кожен біт на протилежний. 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
```

Очищення та встановлення бітів. 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
```

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
```

Макрос керування значенням біту _BV(). 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 не зрідка _BV() виявляється зручнішим для початківців.

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

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.

Контрольні запитання до теми 3

1. Які основні функції виконують вихідні буфери портів вводу-виводу мікроконтролера AVR ATMega8?
2. Які переваги має можливість зміни напрямку та стану виводів порту без впливу на інші виводи?
3. Як відбувається активація підтягувальних резисторів для вхідних виводів у мікроконтролері ATMega8?
4. Які функції виконують регістри DDRx, PORTx і PINx у керуванні портами вводу-виводу мікроконтролера?
5. Яким чином можна відключити всі підтягувальні резистори у всіх портах мікроконтролера AVR?
6. Які функції виконує тригер синхронізації у системі читання стану виводу мікроконтролера AVR ATMega8?
7. Які проблеми вирішує використання синхронізатора для читання стану виводу мікроконтролера?
8. Як впливає час затримки та метастабільність на зчитування стану виводу через PIN регістри мікроконтролера?
9. Які інструкції необхідно використовувати для коректного зчитування програмно встановленого значення піна у мікроконтролері AVR?
10. Які стратегії рекомендується використовувати для мінімізації споживання енергії на невикористовуваних виводах мікроконтролера?
11. Які основні альтернативні функції порту B представлені в мікроконтролерах AVR?

12. Які альтернативні функції доступні для порту C, і як вони впливають на функції вводу-виводу та інші операції?
13. Які альтернативні функції доступні для порту D, і в яких випадках вони використовуються?
14. Яка роль виводів XTAL1 і TOSC1 на порту B, і чому їх не можна використовувати як вивід вводу-виводу у деяких випадках?
15. Як альтернативна функція SCL на порту C впливає на взаємодію з двопровідним послідовним інтерфейсом TWI, і які особливості її використання?
16. Які можливі значення може приймати один біт в мікроконтролерах AVR, і як це відображається у двійковій формі?
17. Які існують форми представлення числових значень в мові програмування C для AVR, зокрема для числа 15?
18. Які операції виконуються при побітовому AND, і як вони впливають на біти операндів?
19. Яким чином за допомогою побітового OR можна комбінувати біти двох чисел?
20. Як відбувається операція побітового XOR і в яких випадках вона корисна?
21. Як працює операція побітового NOT і що вона робить з кожним бітом операнду?
22. Як відбувається зсув бітів вліво та вправо, і як це використовується у програмуванні мікроконтролерів?
23. Яким чином можна встановити окремий біт у визначеному байті за допомогою бітової маски?
24. Як здійснити очищення конкретного біту у байті за допомогою побітової маски та операції AND?
25. Які практичні застосування можуть мати перевірки значення конкретного біту за допомогою побітового AND у програмуванні мікроконтролерів?

Використана література

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. 57 с.
2. Atmel: ATMega8, ATmega8L : технічна документація на мікроконтролер. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2486-8-bit-AVR-microcontroller-ATmega8_L_datasheet.pdf

4 СИСТЕМНИЙ СКИД, ПЕРЕРИВАННЯ ТА РОБОЧІ РЕЖИМИ МІКРОКОНТРОЛЕРІВ СІМЕЙСТВА AVR

Метою вивчення теми є ознайомлення з системами тактування, режимами роботи мікроконтролерів сімейства AVR та проведенням системного скиду та переривання.

Завдання вивчення теми збігаються з переліком питань для розгляду, що наведений нижче.

Перелік питань до розділу:

4.1. Система тактування.
4.2. Режими роботи мікроконтролера.
4.3. Системний скид.
4.4. Сторожовий таймер.
4.5. Переривання.

4.1 Система тактування

На рисунку 4.1 представлені основні системи тактування мікроконтролерів AVR та їх розподіл. Не всі тактові сигнали повинні бути активними в певний час. Щоб зменшити енергоспоживання, тактування модулів, які не використовуються, можна зупинити за допомогою різних режимів сну.

Тактовий сигнал ЦП – $\mathbf{clk_{CPU}}$ подається до частин системи, пов'язаних з роботою ядра AVR. Прикладами таких модулів є регістровий файл загального призначення, регістр стану та пам'ять даних, що містить покажчик стека. Зупинення тактового сигналу процесора зупиняє виконання ядром загальних операцій і обчислень.

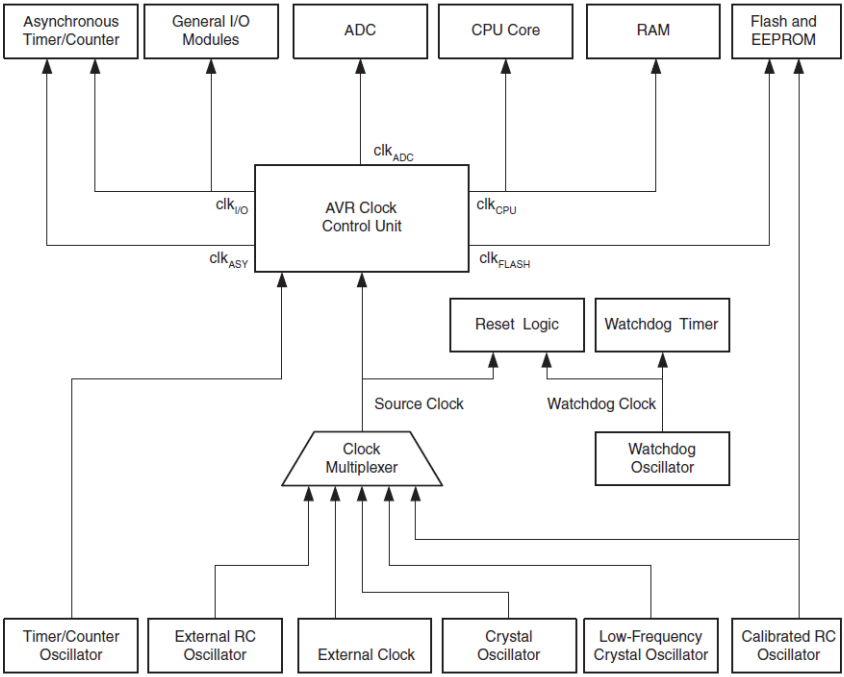


Рисунок 4.1 – Основні системи тактування мікроконтролерів AVR та їх розподіл

Тактовий сигнал вводу-виводу – $clk_{I/O}$ використовується більшістю модулів вводу-виводу, такими, як таймери/лічильники, SPI та USART. Тактовий сигнал вводу-виводу також використовується модулем зовнішнього переривання, але зауважте, що деякі зовнішні переривання виявляються асинхронною логікою, що дозволяє виявляти такі переривання, навіть якщо тактовий сигнал вводу-виводу зупинено. Також зауважте, що розпізнавання адреси в модулі TWI виконується асинхронно, навіть коли $clk_{I/O}$ зупиняється, що дозволяє приймати адресу TWI в усіх режимах сну.

Тактовий сигнал Flash пам'яті – clk_{FLASH} контролює роботу інтерфейсу Flash. Тактовий сигнал флеш-пам'яті зазвичай активний одночасно з тактовим сигналом ЦП.

Тактовий сигнал асинхронного таймера – clk_{ASY} дозволяє тактувати асинхронний таймер-лічильник безпосередньо від зовнішнього кристала синхронізації 32 кГц. Спеціальний домен тактування дозволяє використовувати цей таймер-лічильник як лічильник реального часу, навіть коли пристрій перебуває в режимі сну. Асинхронний таймер-лічильник використовує ті самі контакти XTAL, що й основний тактовий генератор процесора, але вимагає, щоб основна тактова частота ЦП хоча б у чотири рази перевищувала частоту генератора. Таким чином, асинхронна робота доступна лише тоді, коли чіп працює від внутрішнього тактового генератора.

Тактовий сигнал АЦП – clk_{ADC}. АЦП забезпечений спеціальним доменом тактування. Це дозволяє зупинити ЦП і тактовий сигнал вводу-виводу, щоб зменшити шум, створюваний цифровими схемами. Це дає більш точні результати перетворення АЦП.

Джерела тактування. Мікроконтролер має такі параметри джерел тактування, які можна вибрати за допомогою бітів фьюзів, як показано у таблиці 4.1. Тактовий сигнал з вибраного джерела вводиться в тактовий генератор AVR і направляється до відповідних модулів.

Таблиця 4.1 – Джерела тактування, відповідні бітам фьюзів

Джерело тактування	CKSEL3..0
Зовнішній кварцовий/керамічний резонатор	1111-1010
Зовнішній низькочастотний кварцовий резонатор	1001
Зовнішній RC-осцилятор	1000-0101
Калібрований внутрішній RC-осцилятор	0100-0001
Зовнішній тактовий сигнал	0000

Коли ЦП виходить із режиму вимкнення або енергозбереження, вибране джерело синхронізації використовується для визначення часу запуску, забезпечуючи стабільну роботу осцилятора перед початком виконання інструкцій. Коли ЦП запускається після скидання, існує додаткова затримка, яка дозволяє досягти стабільного рівня живлення перед початком нормальної роботи. Тактовий генератор сторожового таймера використовується для визначення часу цієї частини часу запуску в реальному часі.

Пристрій поставляється з CKSEL = «0001» і SUT = «10» (внутрішній RC-генератор 1 МГц, напруга живлення повільно зростає).

4.2 Режими роботи мікроконтролера

Управління живленням і режими сну. Режими сну дозволяють програмі вимикати невикористовувані модулі в мікроконтролері, заощаджуючи таким чином енергію. AVR забезпечує різні режими сну, що дозволяє користувачеві налаштовувати енергоспоживання відповідно до вимог програми.

Щоб увійти в будь-який з п'яти режимів сну, біт SE в регістрі MCUCR має бути встановлений, як логічна одиниця, після чого повинна бути виконана інструкція SLEEP. Біти SM2, SM1 і SM0 у регістрі MCUCR визначають, який режим сну (неактивний, зменшення шуму АЦП, вимкнення, енергозбереження або очікування) буде активовано інструкцією SLEEP (дивіться таблицю 4.2). Якщо дозволене переривання виникає, коли MCU перебуває в режимі сну, MCU прокидається. Після цього MCU зупиняється на чотири цикли на додаток до часу запуску, він виконує функцію обробки переривання та відновлює виконання з інструкції після SLEEP. Вміст реєстрового файлу та SRAM не змінюється, коли пристрій виходить із режиму сну. Якщо скидання відбувається під час режиму сну, MCU прокидається та починає виконання програми з вектору скиду.

Таблиця 4.2 – Визначення режиму сну

SM2	SM1	SM0	Режим сну
0	0	0	Неактивний
0	0	1	Зменшення шуму АЦП
0	1	0	Вимкнення
0	1	1	Енергозбереження
1	0	0	Зарезервоване
1	0	1	Зарезервоване
1	1	0	Очікування

Регістр керування мікроконтролером – MCUCR містить біти для керування живленням.

Bit	7	6	5	4	3	2	1	0	
	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
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 4.2 – Регістр керування мікроконтролером – MCUCR

Біт 7 – SE: увімкнення режиму сну

Біт SE повинен бути встановлений, як логічна одиниця, щоб MCU перейшов у режим сну, коли виконується інструкція SLEEP. Щоб уникнути переходу MCU в режим сну, якщо це не є метою програміста, рекомендується встановити біт Sleep Enable (SE) безпосередньо перед виконанням інструкції SLEEP.

Біти 6..4 – SM2..0: Біти вибору режиму сну 2, 1 і 0

Ці біти вибирають між п'ятьма доступними режимами сну, як показано в таблиці 4.2.

Неактивний режим. Коли біти SM2..0 встановлені як 000, інструкція SLEEP змушує MCU перейти в неактивний режим, зупиняючи ЦП, але дозволяючи SPI, USART, аналоговому компаратору, АЦП, TWI, таймерам/лічильникам, сторожовому таймеру і системі переривання продовжувати роботу. Цей режим сну фактично зупиняє clk_{CPU} та clk_{FLASH}, дозволяючи іншим тактовим сигналам працювати. Неактивний режим дозволяє мікроконтролеру виходити з режиму сну через зовнішні переривання, а також внутрішні, такі як переповнення таймера та переривання USART. Якщо пробудження від переривання аналогового компаратора не потрібне, аналоговий компаратор можна вимкнути, встановивши біт ACD у регістрі керування та стану аналогового компаратора – ACSR. Це зменшить споживання енергії в режимі очікування. Якщо АЦП увімкнено, перетворення починається автоматично при вході в цей режим.

Режим зменшення шуму АЦП. Коли біти SM2..0 встановлені як 001, інструкція SLEEP змушує MCU перейти в режим зменшення шуму АЦП, зупиняючи ЦП, але дозволяючи АЦП, зовнішнім

перериванням, контролю адреси TWI, таймеру/лічильнику 2 і сторожовому таймеру продовжувати роботу (якщо вони ввімкнені). Цей режим сну фактично зупиняє $clk_{I/O}$, clk_{CPU} та clk_{FLASH}, дозволяючи іншим тактовим сигналам працювати.

Це покращує шумове середовище для АЦП, що дозволяє проводити вимірювання з вищою роздільною здатністю. Якщо АЦП увімкнено, перетворення починається автоматично при вході в цей режим. Крім переривання по завершенню перетворення АЦП, тільки зовнішнє скидання, скидання від сторожового таймера, скидання від детектору зниженої напруги, переривання по збігу адреси TWI, переривання таймера-лічильника 2, переривання готовності SPM/EEPROM або зовнішнє переривання INT0 або INT1, може розбудити MCU з режиму зменшення шуму АЦП.

Режим вимкнення. Коли біти SM2..0 встановлені як 010, інструкція SLEEP змушує MCU перейти в режим вимкнення живлення. У цьому режимі зовнішній генератор зупиняється, тоді як зовнішні переривання, контроль адреси TWI та сторожовий таймер продовжують працювати (якщо вони ввімкнені). Лише зовнішнє скидання, скидання від сторожового таймера, скидання при зниженні напруги живлення, переривання збігу адреси TWI або зовнішнє переривання INT0 або INT1 можуть розбудити MCU. Цей режим сну фактично зупиняє всі джерела тактування, дозволяючи працювати лише асинхронним модулям.

Зауважте, що якщо зовнішнє переривання, викликане рівнем, використовується для пробудження з режиму вимкнення живлення, змінений рівень потрібно утримувати деякий час, щоб розбудити MCU.

Під час пробудження з режиму вимкнення існує затримка від моменту початку стану пробудження до моменту, коли пробудження дійсно настає. Це дозволяє перезапустити та стабілізувати тактові сигнали після зупинки. Період пробудження визначається тими самими фьюзами CKSEL, які визначають період очікування скидання.

Режим енергозбереження. Коли біти SM2..0 встановлені як 011, інструкція SLEEP змушує MCU перейти в режим енергозбереження. Цей режим ідентичний режиму вимкнення, за одним винятком:

Якщо таймер-лічильник 2 тактується асинхронно, тобто встановлено біт AS2 у ASSR, таймер-лічильник 2 працюватиме під час сну. Пристрій може вийти з режиму сну через подію «переповнення таймера» або «збіг при порівнянні» від таймера-лічильника 2, якщо відповідні біти дозволу переривань встановлені в TIMSK, а біт дозволу глобального переривання встановлений в SREG.

Якщо асинхронний таймер НЕ працює асинхронно, рекомендується використовувати режим вимкнення живлення замість режиму енергозбереження, оскільки вміст регістрів в асинхронному таймері слід вважати невизначеним після пробудження в режимі енергозбереження, якщо AS2 дорівнює 0. Цей режим сну фактично зупиняє всі годинники, крім clk_{ASY}, дозволяючи працювати лише асинхронним модулям, включаючи таймер-лічильник 2, якщо він тактується асинхронно.

Режим очікування. Коли біти SM2..0 дорівнюють 110 і вибрано опцію тактування від зовнішнього резонатора, інструкція SLEEP змушує MCU перейти в режим очікування. Цей режим ідентичний режиму вимкнення, за винятком того, що осцилятор продовжує працювати. В режимі очікування пристрій виходить з режиму сну за 6 тактів.

Мінімізація енергоспоживання. Є кілька питань, які слід враховувати, намагаючись мінімізувати споживання електроенергії в системі, керованій AVR. Загалом, режими сну слід використовувати якомога частіше, а тип режиму сну слід вибирати таким чином, щоб працювало якомога менше функцій пристрою. Усі непотрібні функції слід відключити. Зокрема, наступні модулі можуть потребувати особливої уваги, якщо ви намагаєтесь досягти найменшого енергоспоживання.

Аналого-цифровий перетворювач (АЦП). Якщо він ввімкнений, АЦП буде залишатися ввімкненим в усіх режимах сну. Щоб заощадити енергію, АЦП слід вимкнути перед переходом у режим сну. Коли АЦП вимкнеться та знову ввімкнеться, наступне перетворення буде розширеним.

Аналоговий компаратор. Під час переходу в неактивний режим аналоговий компаратор слід вимкнути, якщо він не використовується. Під час входу в режим зменшення шуму АЦП аналоговий компаратор має бути вимкнено. В інших режимах сну аналоговий

компаратор автоматично вимикається. Проте, якщо аналоговий компаратор налаштовано на використання внутрішньої опорної напруги як вхідної, аналоговий компаратор слід вимкнути в усіх режимах сну. Інакше внутрішня опорна напруга буде ввімкнена незалежно від режиму сну.

Детектор зниження напруги живлення. Якщо детектор зниження напруги живлення не потрібен у програмі, цей модуль слід вимкнути. Якщо детектор зниження напруги ввімкнений фьюзом BODEN, він залишиться ввімкненим в усіх режимах сну, а отже, завжди споживатиме електроенергію. У режимах глибшого сну це значно впливає на загальне споживання струму.

Внутрішня опорна напруга. Внутрішня опорна напруга буде ввімкнена, коли це необхідно для детектора зниження напруги живлення, аналогового компаратора або АЦП. Якщо ці модулі вимкнено, як описано в розділах вище, внутрішня опорна напруга буде вимкнена, і вона не споживатиме електроенергію. Після повторного ввімкнення користувач повинен дозволити модулю опорної напруги запуститися перед його використанням. Якщо опорний сигнал увімкнено в сплячому режимі, його можна використовувати одразу.

Сторожовий таймер. Якщо сторожовий таймер не потрібен у програмі, цей модуль слід вимкнути. Якщо ввімкнути сторожовий таймер, він буде ввімкненим в усіх режимах сну, а отже, завжди споживатиме енергію. У режимах глибшого сну це значно вплине на загальне споживання струму.

Порти вводу-виводу. Під час переходу в режим сну усі порти мають бути налаштовані на використання мінімальної потужності. Найважливіше переконатися, що ніякі виводи не підключені до резистивних навантажень. У режимах сну, коли тактовий сигнал вводу-виводу ($clk_{I/O}$) і тактовий сигнал АЦП (clk_{ADC}) зупиняються, вхідні буфери пристрою будуть вимкнені. Це гарантує, що вхідна логіка не споживає електроенергію, коли вона не потрібна. У деяких випадках вхідна логіка потрібна для виявлення умов пробудження, і тоді її буде ввімкнено. Якщо вхідний буфер увімкнено, а вхідний сигнал залишається плаваючим або має рівень аналогового сигналу, близький до VCC/2, вхідний буфер споживатиме надмірну потужність.

4.3 Системний скид

Під час скиду всі регістри вводу-виводу встановлюються на їхні початкові значення, і програма починає виконання з вектору скиду. Якщо програма не використовує переривання, вектори переривань не використовуються, і звичайний код програми можна розмістити в цих місцях. На електричній схемі на рисунку 4.3 показано логіку скидання.

Порти вводу-виводу AVR одразу скидаються до початкового стану, коли джерело скиду стає активним. Для цього не потрібно, щоб будь-який тактовий сигнал був запущений.

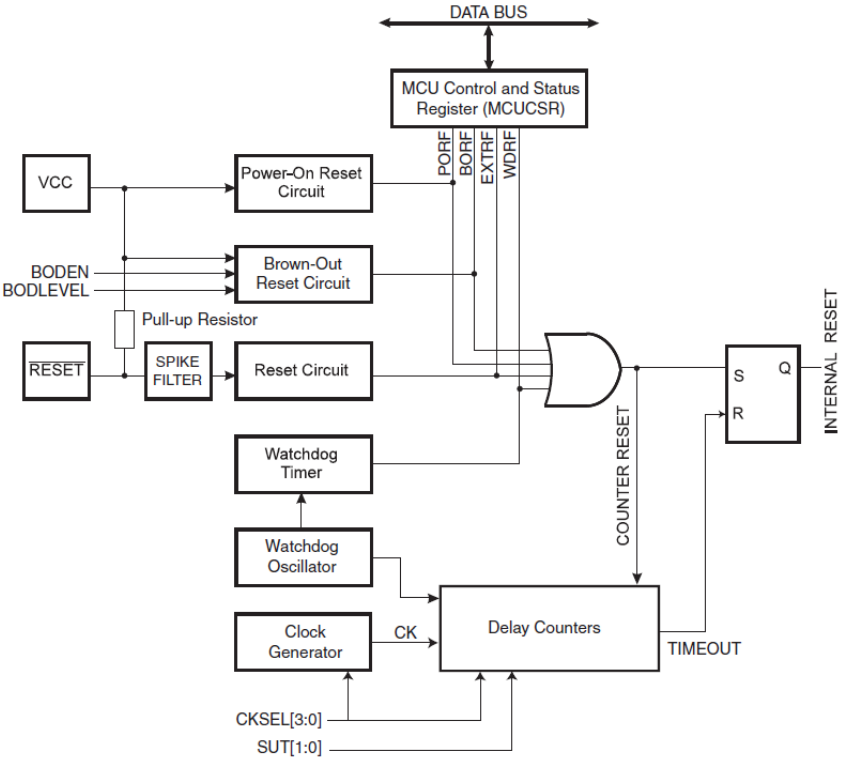


Рисунок 4.3 – Логіка скидання

Після того, як усі джерела скиду стають неактивними, запускається лічильник затримки, розтягуючи внутрішній скид. Це дозволяє досягти стабільного рівня напруги перед початком нормальної роботи. Період очікування лічильника затримки визначається користувачем за допомогою фьюзів CKSEL.

ATmega8 має чотири джерела скиду:

– скид під час увімкнення. MCU скидається, коли напруга живлення нижча за порогове значення скидання при включенні (VPOT);

– зовнішній скид. MCU скидається, коли низький рівень присутній на виводі RESET довше, ніж мінімальна довжина імпульсу;

– скид від сторожового таймера. MCU перезавантажується, коли період сторожового таймера закінчується при умові, що сторожовий таймер увімкнений;

– скид при зниженні напруги живлення. MCU скидається, коли напруга живлення VCC нижча за порогове значення перезапуску (VBOT), а детектор зниження напруги увімкнено.

Регістр керування та стану мікроконтролера – MCUCSR. Регістр керування та стану мікроконтролера надає інформацію про те, яке джерело викликало скид мікроконтролера (рис. 4.4).

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	WDRF	BORF	EXTRF	PORF	MCUCSR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	See Bit Description				

Рисунок 4.4 – Регістр керування та стану мікроконтролера – MCUCSR

Біти 7..4 – Res: зарезервовані біти

Ці біти є зарезервованими бітами в ATmega8 і завжди читаються як нуль.

Біт 3 – WDRF: прапор скиду від сторожового таймера

Цей біт встановлюється, якщо відбувається скид від сторожового таймера. Біт скидається при скиді під час увімкнення живлення або записом логічного нуля у цей прапор.

Біт 2 – BORF: прапор скиду при зниженні напруги живлення

Цей біт встановлюється, якщо відбувається скидання при зниженні напруги живлення. Біт скидається при скиді під час увімкнення живлення або записом логічного нуля у цей прапор.

Біт 1 – EXTRF: прапор зовнішнього скиду

Цей біт встановлюється, якщо відбувається зовнішній скид. Біт скидається при скиді під час увімкнення живлення або записом логічного нуля у цей прапор.

Біт 0 – PORF: прапор скиду під час увімкнення

Цей біт встановлюється, якщо відбувається скид під час увімкнення живлення. Біт скидається тільки шляхом запису логічного нуля у цей прапор.

Щоб використовувати прапор скидання для визначення стану скидання, користувач повинен прочитати та скинути MCUCSR якомога раніше в програмі.

4.4 Сторожовий таймер

Сторожовий таймер (рис. 4.5) тактується від окремого вбудованого генератора, який працює на частоті 1 МГц. Це типове значення при VCC = 5 В. Керуючи попереднім дільником сторожового таймера, інтервал скиду можна налаштувати, як показано на рисунку 4.6.

WDR – Watchdog Reset – інструкція, що скидає сторожовий таймер. Сторожовий таймер також скидається, якщо його вимкнено та коли відбувається скид мікроконтролера. Для визначення періоду скиду можна вибрати вісім різних періодів тактового генератора.

Якщо період скидання сторожового таймера закінчується без повторного скиду самого таймера, ATmega8 скидається, а програма починає виконуватися з вектору скиду. Щоб запобігти випадковому вимкненню Watchdog, слід дотримуватися спеціальної послідовності вимкнення, коли Watchdog вимкнено.

Регістр керування сторожовим таймером – WDTCR (рис. 4.7).

Біти 7..5 – Res: зарезервовані біти

Ці біти є зарезервованими бітами в ATmega8 і завжди читаються як нуль.

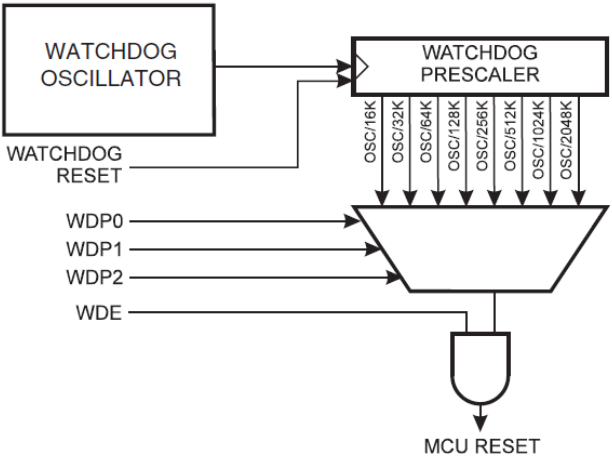


Рисунок 4.5 – Функціональна схема сторожового таймера

WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles	Typical Time-out at V_{CC} = 3.0V	Typical Time-out at V_{CC} = 5.0V
0	0	0	16K (16,384)	17.1ms	16.3ms
0	0	1	32K (32,768)	34.3ms	32.5ms
0	1	0	64K (65,536)	68.5ms	65ms
0	1	1	128K (131,072)	0.14s	0.13s
1	0	0	256K (262,144)	0.27s	0.26s
1	0	1	512K (524,288)	0.55s	0.52s
1	1	0	1,024K (1,048,576)	1.1s	1.0s
1	1	1	2,048K (2,097,152)	2.2s	2.1s

Рисунок 4.6 – Значення попереднього дільника та відповідні їм періоди очікування

Bit	7	6	5	4	3	2	1	0	
	–	–	–	WDCE	WDE	WDP2	WDP1	WDP0	WDTCR
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 4.7 – Регістр керування сторожовим таймером – WDTCR

Біт 4 – WDCE: дозвіл зміни сторожового таймера

Цей біт потрібно встановити, коли в біт WDE записаний логічний нуль. В іншому випадку сторожовий таймер не буде вимкнено. Після запису одиниці в цей біт, він буде апаратно скинутий в 0 після чотирьох тактів. На рівнях безпеки 1 і 2 цей біт також потрібно встановлювати під час зміни бітів попереднього дільника.

Біт 3 – WDE: ввімкнення сторожового таймера

Коли у WDE записується логічна одиниця, сторожовий таймер увімкнено, а якщо у WDE записується логічний нуль, функція сторожового таймера вимкнена. WDE можна очистити, лише якщо біт WDCE має логічний рівень один. Щоб вимкнути ввімкнений сторожовий таймер, необхідно виконати наступну процедуру:

1. У тій самій операції запишіть логічну одиницю в WDCE і WDE. Логічну одиницю потрібно записати в WDE, навіть якщо він вже встановлений перед початком операції вимкнення.

2. Протягом наступних чотирьох тактів запишіть логічний нуль у WDE. Це вимикає сторожовий таймер.

Біти 2..0 – WDP2, WDP1, WDP0: Попередній дільник сторожового таймера 2, 1 і 0

Біти WDP2, WDP1 і WDP0 визначають попередній дільник сторожового таймера, коли він увімкнений.

У наступному прикладі коду показано функцію на мові С для вимкнення WDT. У прикладі припускається, що переривання контролюються програмою (наприклад, шляхом глобального вимкнення переривань), щоб жодних переривань не виникало під час виконання цих функцій.

```
void WDT_off(void)
{
/* Скид сторожового таймера*/
_WDR();
/* Запис логічної одиниці у WDCE та WDE */
WDTCR |= (1<<WDCE) | (1<<WDE);
/* Вимкнення сторожового таймера */
WDTCR = 0x00;
}
```

Послідовність зміни конфігурації сторожового таймера дещо відрізняється для різних рівнів безпеки. Для кожного рівня описані окремі процедури.

Рівень безпеки 1 (фьюз WDTON не запрограмований)

У цьому режимі сторожовий таймер спочатку вимкнено, але його можна ввімкнути записом 1 у біт WDE без будь-яких обмежень. Під час зміни періоду тайм-ауту сторожового таймера або вимкнення увімкненого сторожового таймера потрібна певна послідовність. Щоб вимкнути ввімкнений сторожовий таймер і/або змінити час очікування сторожового таймера, необхідно виконати таку процедуру:

1. В одній і тій самій операції запишіть логічну одиницю в WDCE і WDE. Логічна одиниця повинна бути записана в WDE незалежно від попереднього значення біта WDE.

2. Протягом наступних чотирьох тактів в одній і тій самій операції запишіть біти WDE та WDP за бажанням, але з очищеним бітом WDCE.

Рівень безпеки 2 (фьюз WDTON запрограмований)

У цьому режимі сторожовий таймер завжди ввімкнено, а біт WDE завжди читається як одиниця. Під час зміни періоду тайм-ауту сторожового таймера необхідна певна послідовність. Щоб змінити тайм-аут сторожового таймера, потрібно виконати таку процедуру:

1. В одній і тій самій операції запишіть логічну одиницю в WDCE і WDE. Незважаючи на те, що WDE завжди встановлено, в WDE потрібно записати одиницю, щоб розпочати часову послідовність.

2. Протягом наступних чотирьох тактів у тій самій операції запишіть біти WDP за бажанням, але з очищеним бітом WDCE. Значення, записане в біт WDE, не має значення.

4.5 Переривання

AVR забезпечує кілька різних джерел переривань. Ці переривання та окремий вектор скиду мають окремий програмний вектор у 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 вектору 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 вектору 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 A таймера/лічильника 1	17	0x0010	ANA_COMP	Аналоговий компаратор
8	0x0007	TIMER1 COMPB	Збіг при порівнянні B 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 4.8 – 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. 4.9).

Bit	7	6	5	4	3	2	1	0	
	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
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 4.9 – 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 4.3. Значення на контакті INT1 зчитується перед виявленням фронтів.

Таблиця 4.3 – 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

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.

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. 4.10).

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 4.10 – 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 вектору 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 вектору переривання INT0.

Загальний регістр прапорів переривань – GIFR (рис. 4.11).

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 4.11 – 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.

Контрольні запитання до теми 4

1. Яким чином можна зменшити енергоспоживання мікроконтролера AVR?
2. Які модулі використовують тактовий сигнал clkI/O і що відбувається з перериваннями, коли цей сигнал зупинено?
3. Яка основна функція тактового сигналу clkFLASH і коли він зазвичай активний?
4. Як тактовий сигнал clkASY дозволяє використовувати асинхронний таймер-лічильник в режимі реального часу?
5. Які джерела тактування можна вибрати для мікроконтролера AVR і як вони визначаються за допомогою бітів фьюзів?
6. Яким чином режим сну впливає на енергоспоживання мікроконтролера AVR?
7. Які функції можуть продовжувати працювати, коли мікроконтролер знаходиться в неактивному режимі сну?
8. Які події можуть пробудити MCU з режиму зменшення шуму АЦП?
9. Які модулі слід вимкнути для мінімізації енергоспоживання в режимах сну?
10. Які регістри встановлюються на початкові значення під час скиду мікроконтролера AVR?
11. Які джерела скиду має ATmega8 і які умови для кожного з них?
12. Як регістр MCUCSR інформує про джерело скиду мікроконтролера, і які біти він містить?
13. Яка роль лічильника затримки під час скиду мікроконтролера і як він налаштовується користувачем?
14. Як працюють біти в регістрі MCUCSR, що відповідають за скидання від сторожового таймера, зниження напруги живлення, зовнішнього скиду та скиду під час увімкнення живлення?
15. Чому важливо читати та скидувати регістр MCUCSR якомога раніше після скиду мікроконтролера?
16. Яка частота вбудованого генератора сторожового таймера в ATmega8?
17. Як можна налаштувати інтервал скиду сторожового таймера за допомогою попереднього дільника?
18. Які біти містить регістр керування сторожовим таймером (WDTCR) і які їх функції?
19. Яка процедура вимкнення сторожового таймера для рівня безпеки 1 (фьюз WDTON не запрограмований)?
20. Які кроки необхідно виконати для зміни часу очікування сторожового таймера при рівні безпеки 2 (фьюз WDTON запрограмований)?

21. Яким чином AVR забезпечує переривання та як вони пов'язані з програмним простором пам'яті?
22. Як визначаються пріоритети різних переривань у AVR, і яке переривання має найвищий пріоритет?
23. Що відбувається з глобальним дозволом переривань (біт I) при виникненні переривання, і як можна увімкнути вкладені переривання?
24. Як функціонують прапори переривань у AVR, і яким чином можна очистити відповідний прапор переривання?
25. Які умови необхідні для активації зовнішніх переривань через виводи INT0 і INT1, і як вони налаштовуються за допомогою регістру MCUCR?

Використана література

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. 57 с.
2. Atmel: ATMega8, ATmega8L : технічна документація на мікроконтролер. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2486-8-bit-AVR-microcontroller-ATmega8_L_datasheet.pdf

5 ТАЙМЕРИ МІКРОКОНТРОЛЕРІВ СІМЕЙСТВА AVR

Метою вивчення теми є ознайомлення з таймерами мікроконтролерів сімейства AVR, а саме 8-бітним таймер-лічильником 0 та 16-бітним таймер-лічильником 1 мікроконтролера ATMega8.

Завдання вивчення теми збігаються з переліком питань для розгляду, що наведений нижче.

Перелік питань до розділу:

5.1. 8-бітний таймер-лічильник 0 мікроконтролера ATMega8.

5.2. 16-бітний таймер-лічильник 1 мікроконтролера ATMega8.

5.1 8-бітний таймер-лічильник 0 мікроконтролера ATMega8

Таймер-лічильник 0 – 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 5.1.

Регістри. Таймер-лічильник (TCNT0) є 8-розрядним регістром. Усі сигнали запиту на переривання (скорочено Int. Req. на рисунку 5.1) 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 5.1, оскільки ці регістри спільно використовуються іншими блоками таймерів.

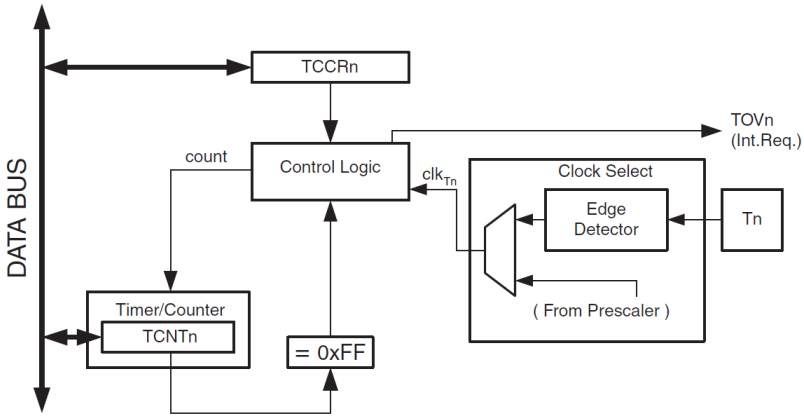


Рисунок 5.1 – Спрощена блок-схема
8-розрядного таймера-лічильника

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 цьому розділі 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 5.1 також широко використовуються далі.

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 5.1 – 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 5.2 показана блок-схема лічильника та його оточення.

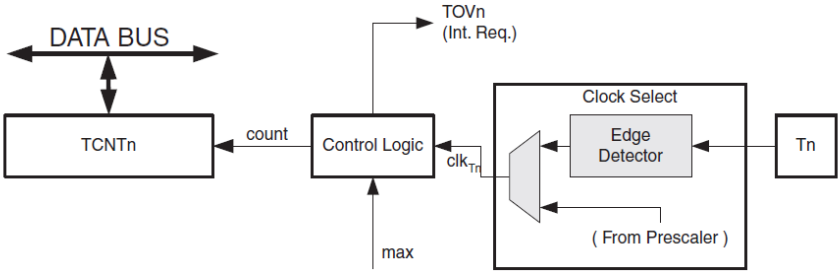


Рисунок 5.2 – Блок-схема лічильника та його оточення

Опис сигналів (внутрішні сигнали):

– **count** збільшує TCNT0 на 1;

– **clk$_{Tn}$** тактовий сигнал таймера-лічильника, далі згадується як clkT0;

– **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 5.3 містить дані тактування при нормальній роботі таймера-лічильника. Показана послідовність рахунку, близька до максимального значення.

На рисунку 5.4 показані ті самі дані про тактування, але з увімкненим попереднім дільником ($f_{clkI/O}/8$).

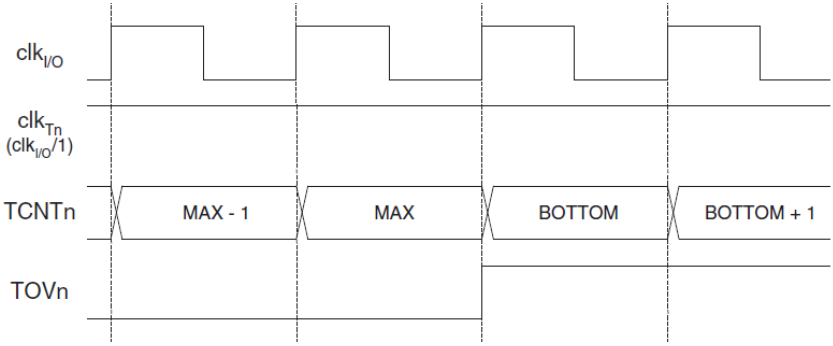


Рисунок 5.3 – Дані тактування при нормальній роботі таймера-лічильника

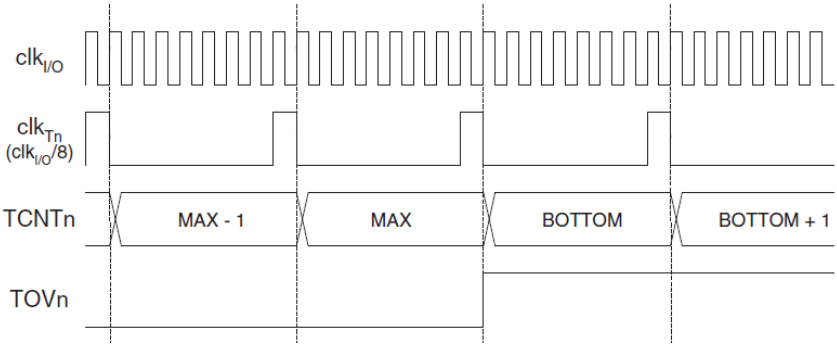


Рисунок 5.4 – Дані про тактування з увімкненим попереднім дільником

Опис регістрів 8-бітного таймера-лічильника 0 мікроконтролера ATMega8

Регістр керування таймером-лічильником – TCCR0 зображено на рис. 5.5.

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 5.5 – 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-лічильника 0 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 5.2.

Таблиця 5.2 – 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. 5.6.

Bit	7	6	5	4	3	2	1	0	
	TCNT0[7:0]								TCNT0
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 5.6 – 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 5.7.

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	–	TOIE0	TIMSK
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 5.7 – 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. 5.8)

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 5.8 – Регістр прапорів переривання таймерів-лічильників – TIFR

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 вектору 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.

5.2 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 5.9.

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 5.9) 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 5.9, оскільки ці регістри спільно використовуються іншими блоками таймерів.

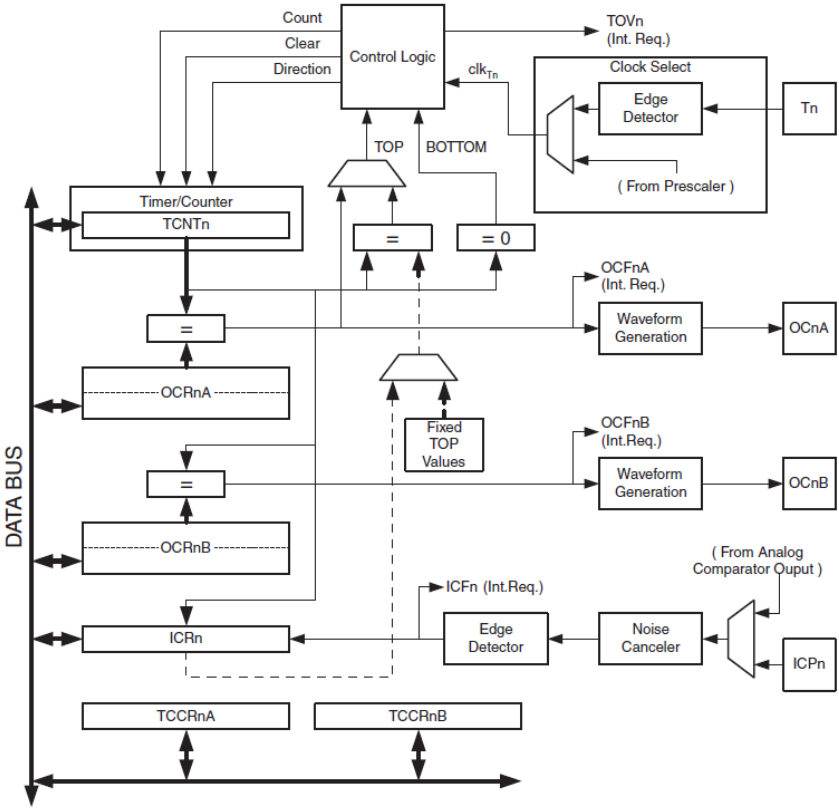


Рисунок 5.9 – Спрощена блок-схема 16-розрядного таймера-лічильника 1

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 цьому розділі широко використовуються такі визначення:

Таблиця 5.3 – 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 5.10 показана блок-схема лічильника та його оточення.

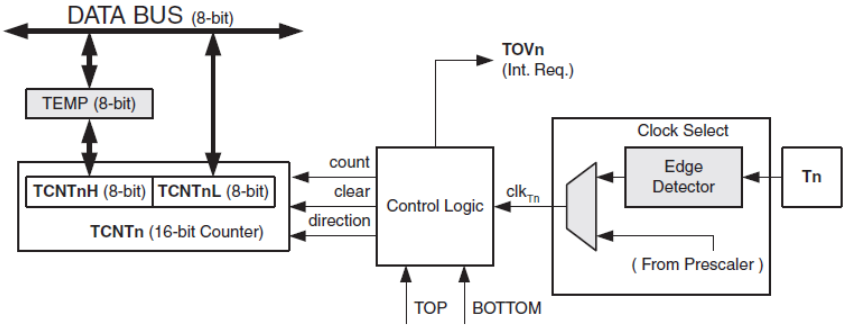


Рисунок 5.10 – Блок-схема лічильника та його оточення

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).

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 байту (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.

Блок захоплення. Таймер-лічильник містить блок захоплення лічильника, який може фіксувати зовнішні події та надавати їм мітку часу, що вказує на час їх виникнення. Зовнішній сигнал, що вказує на подію або кілька подій, може бути подано на вивід ICP1 або альтернативно через блок аналогового компаратора. Потім мітки часу можна використовувати для розрахунку частоти, робочого циклу та інших характеристик сигналу, що подається. Крім того, мітки часу можна використовувати для створення журналу подій.

Блок захоплення ілюструється блок-схемою, показаною на рисунку 5.11. Елементи блок-схеми, які безпосередньо не є частиною блоку захоплення, виділені сірим кольором. Маленька буква “n” у назвах регістрів і бітів вказує на номер таймера-лічильника.

Коли зміна логічного рівня (подія) відбувається на вхідному виводі захоплення (ICP1), або на виході аналогового компаратора (ACO), і ця зміна відповідає налаштуванням детектора фронту, запускається захоплення. При цьому, 16-бітне значення лічильника (TCNT1) записується у вхідний регістр захоплення (ICR1). Прапор захоплення (ICF1) встановлюється у той самий такт системного тактового сигналу, що й значення TCNT1 копіюється в регістр ICR1. Прапор захоплення генерує переривання захоплення, якщо воно ввімкнено (TICIE1 = 1). Прапор ICF1 автоматично очищується, коли виконується підпрограма обробки переривання. Крім того, прапор ICF1 можна скинути програмним забезпеченням, записавши логічну одиницю у відповідний біт.

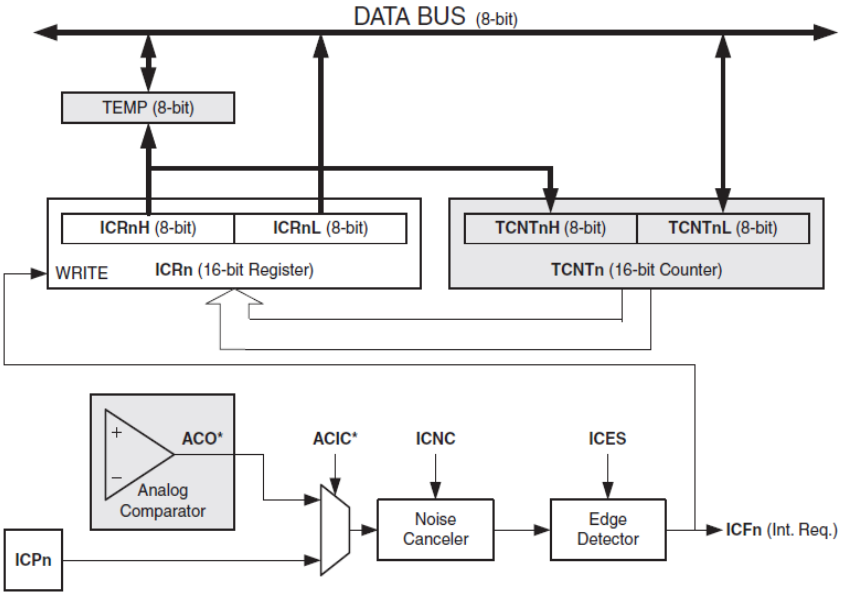


Рисунок 5.11 – Блок-схема блоку захоплення

Зчитування 16-бітного значення з регістру захоплення (ICR1) виконується шляхом спочатку зчитування молодшого байту (ICR1L), а потім старшого байту (ICR1H). Коли зчитується молодший байт, старший байт копіюється в тимчасовий регістр старшого байту (TEMP). Коли ЦП зчитує значення регістру вводу-виводу ICR1H, він отримує доступ до реєстру TEMP.

Регістр ICR1 можна записувати лише у режимі генерації сигналу, який використовує регістр ICR1 для визначення верхнього значення лічильника. У цих випадках біти режиму генерації сигналу (WGM13:0) повинні бути встановлені до того, як значення верхнього значення можна буде записати в регістр ICR1. Під час запису в регістр ICR1 старший байт має бути записаний до регістру ICR1H перед тим, як молодший байт буде записаний до ICR1L.

Джерело захоплення. Основним джерелом запуску для блоку захоплення є вивід захоплення вхідних даних (ICP1). Таймер/Лічильник 1 може альтернативно використовувати вихід аналогового компаратора як джерело запуску для блоку захоплення. Аналоговий компаратор вибирається як джерело запуску шляхом встановлення біта захоплення входу аналогового компаратора (ACIC) у регістрі керування та стану аналогового компаратора (ACSR). Майте на увазі, що зміна джерела тригера може ініціювати захоплення. Таким чином, прапор захоплення має бути очищеним після зміни.

Вхідні сигнали виводу захоплення (ICP1) і виходу аналогового компаратора (ACO) вибираються з використанням тієї ж методики, що й для контакту T1. Детектор фронтів також ідентичний. Однак, коли фільтр шуму увімкнено, перед детектором фронту вставляється додатковий блок, що збільшує затримку на чотири такти системи. Зауважте, що вхід фільтра шуму та детектора фронту завжди ввімкнено, якщо таймер-лічильник не налаштовано в режимі генерації сигналу, який використовує ICR1 для визначення верхнього значення. Захоплення може бути ініційовано програмним забезпеченням, керуючи портом контакту ICP1.

Фільтр шуму. Фільтр шуму покращує завадостійкість за допомогою простої цифрової схеми фільтрації. Вхідний сигнал зчитується фільтром шуму чотири рази, і всі чотири результати мають

бути однаковими для зміни вихідного сигналу, який, у свою чергу, використовується детектором фронту. Фільтр шуму вмикається встановленням біта фільтру шуму блока захоплення (ICNC1) у регістрі керування таймером-лічильником B (TCCR1B). Коли ввімкнуто фільтр шуму, до оновлення регістру ICR1 додається чотири цикли системного тактового сигналу затримки від моменту зміни сигналу на вході. Фільтр шуму використовує системний тактовий генератор і, отже, на нього не впливає попередній дільник.

Використання блоку захоплення. Основною складністю при використанні модуля захоплення є виділення достатнього процесорного часу для обробки вхідних подій. Час між двома подіями є критичним. Якщо процесор не прочитав захоплене значення в регістрі ICR1 до наступної події, ICR1 буде перезаписано новим значенням. У цьому випадку результат захоплення буде некоректним.

При використанні переривання захоплення регістр ICR1 слід зчитати якомога раніше в підпрограмі обробки переривань. Незважаючи на те, що переривання захоплення має відносно високий пріоритет, максимальний час відповіді на переривання залежить від максимальної кількості тактових циклів, необхідних для обробки будь-яких інших запитів на переривання.

Не рекомендується використовувати блок захоплення у будь-якому режимі роботи, коли верхнє значення (роздільна здатність) активно змінюється під час роботи.

Вимірювання робочого циклу зовнішнього сигналу вимагає зміни фронту запуску після кожного захоплення. Змінити визначення фронту слід якомога раніше після зчитування регістру ICR1. Після зміни фронту прапор захоплення вхідного сигналу (ICF1) має бути очищено програмним забезпеченням (записуючи логічну одиницю у відповідний біт). Для вимірювання лише частоти очищення прапора ICF1 не потрібно (якщо використовується обробник переривань).

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 5.12 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.

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 байту (TEMP). Однак, як під час доступу до інших 16-бітних регістрів,

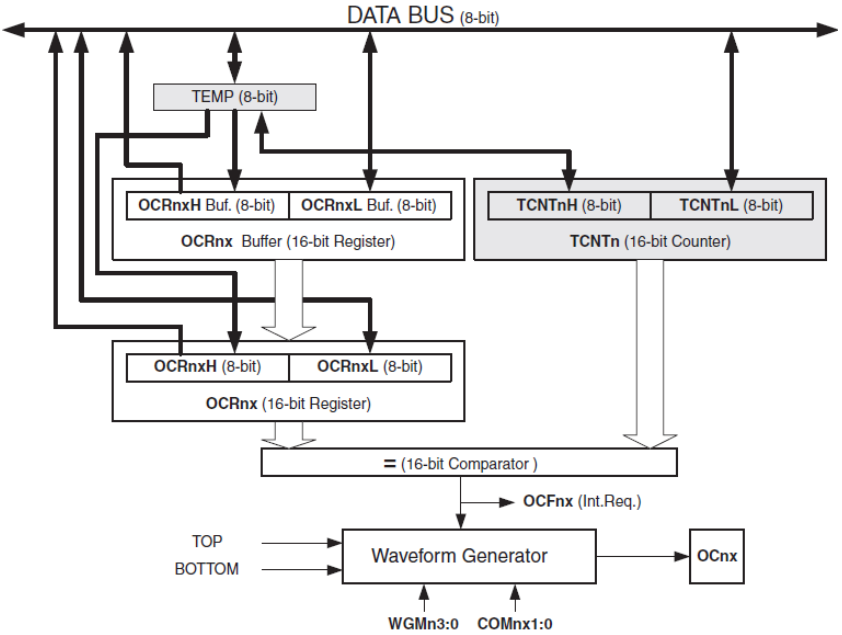


Рисунок 5.12 – Блок-схема блоку порівняння

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 байту 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 5.13 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.

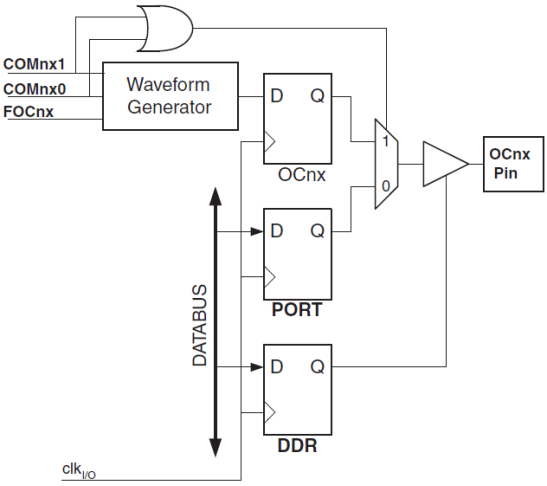


Рисунок 5.13 – Спрощена схема логіки блоку збігу

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».

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 (ОТЗ). У режимі очищення таймеру при збігу (WGM13:0 = 4 або 12) регістри OCR1A або ICR1 використовуються для керування роздільною здатністю лічильника. У режимі ОТЗ 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 ОТЗ показана на рисунку 5.14. Значення лічильника (TCNT1) збільшується, доки не відбудеться збіг зі значенням OCR1A або ICR1, а потім лічильник (TCNT1) очищується.

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

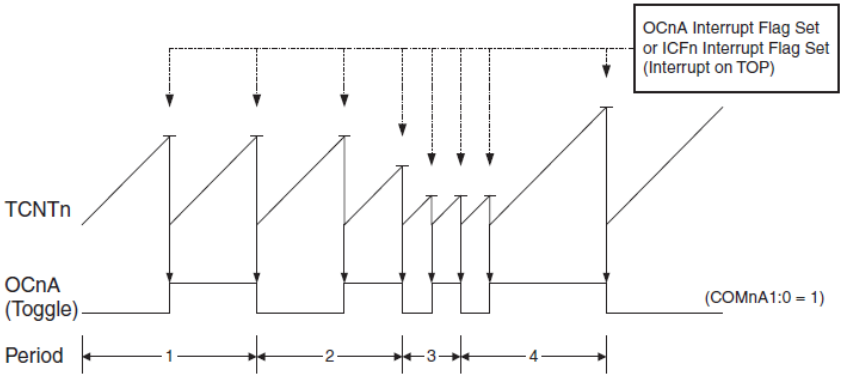


Рисунок 5.14 – Часова діаграма для режиму ОТЗ

або з низьким значенням попереднього дільника, слід виконувати обережно, оскільки режим ОТЗ 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 ОТЗ 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:

$$f_{OCnA} = \frac{f_{CLK\_I/O}}{2 \cdot N \cdot (1 + OCRnA)}. \qquad (5.1)$$

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:

$$R_{ШІМ} = \frac{log(\text{верхнє значення} + 1)}{log(2)}. \qquad (5.2)$$

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 5.15. 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.

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.

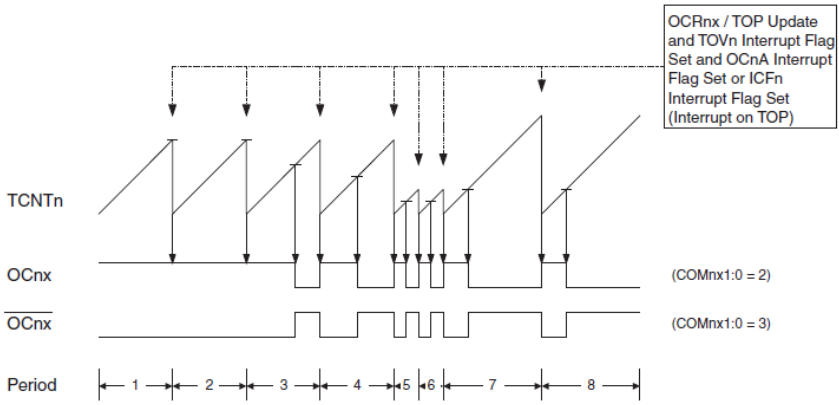


Рисунок 5.15 – Часова діаграма для режиму швидкої ШІМ

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:

$$f_{OCnx\text{ШІМ}} = \frac{f_{CLK\_I/O}}{N \cdot (1 + \text{верхнє значення})}. \qquad (5.3)$$

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 ОТЗ, 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 5.16. 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.

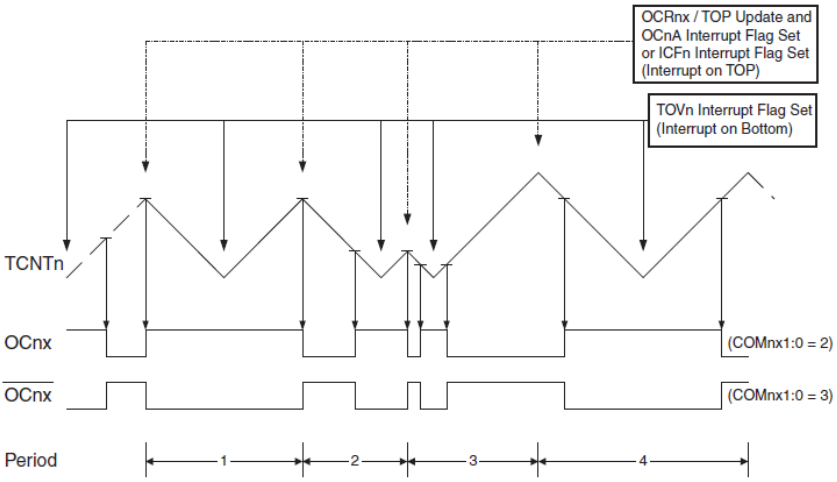


Рисунок 5.16 – Часова діаграма для фазо-коректного режиму ШІМ

Маленькі горизонтальні лінії на схилах TCNT1 представляють порівняльні збіги між OCR1x і TCNT1. Прапор переривання OC1x буде встановлено, коли відбувається збіг при порівнянні.

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 5.17).

Опис регістрів 16-бітного таймера-лічильника 1 мікроконтролера ATMega8

Регістр керування А таймера-лічильника 1 – TCCR1A зображено на рисунку 5.18.

Біт 7:6 – COM1A1:0: режим порівняння для каналу А

Біт 5:4 – COM1B1:0: режим порівняння для каналу В

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,

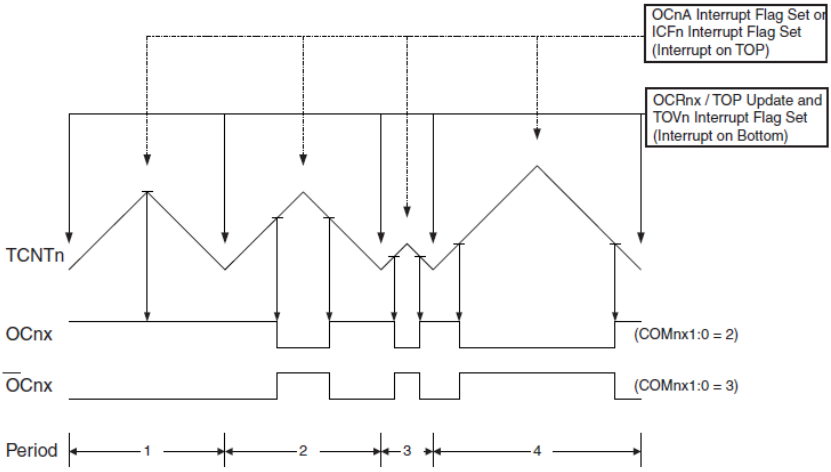


Рисунок 5.17 – Часова діаграма фазо- та частото-коректного ШІМ

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 5.18 – Регістр керування А таймера-лічильника 1

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 5.4 показує функціональні можливості бітів COM1x1:0, коли біти WGM13:0 встановлено, як звичайний режим або режим ОТЗ (не ШІМ).

Таблиця 5.4 – Функціональні можливості бітів COM1x1:0, коли біти WGM13:0 встановлено, як звичайний режим або режим ОТЗ (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 5.5 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 5.6 показує функціональність бітів COM1x1:0, коли біти WGM13:0 встановлені, як фазо-коректний, або фазо- та частото-коректний ШІМ.

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.

Таблиця 5.5 – 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
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 5.6 – 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

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 ОТЗ, 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 (ОТЗ) і три типи режимів широтно-імпульсної модуляції (ШІМ). Відповідність між значеннями бітів WGM13:0 та режимами роботи показана у таблиці 5.7.

Таблиця 5.7 – 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
1	2	3	4	5	6	7	8	9
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.

Продовження таблиці 5.7

1	2	3	4	5	6	7	8	9
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 5.19.

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 5.19 – 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.

Таблиця 5.8 – 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

Біт 2:0 – CS12:0: Вибір тактового сигналу

Три біти вибору тактового сигналу вибирають 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. Ця функція дозволяє програмно контролювати рахунок.

Таймер-Лічильник 1 – TCNT1H і TCNT1L (рис. 5.20).

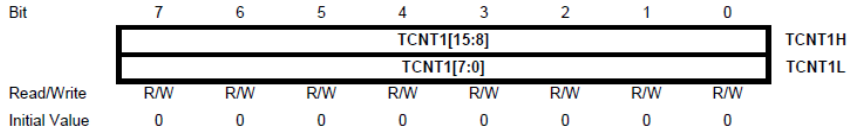


Рисунок 5.20 – Таймер-лічильник 1 – TCNT1H і TCNT1L

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 байту (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 5.21).

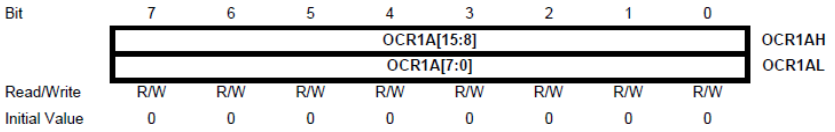


Рисунок 5.21 – Регістр порівняння 1 A – OCR1AH і OCR1AL

Регістр порівняння 1 B – OCR1BH і OCR1BL (рис. 5.22).

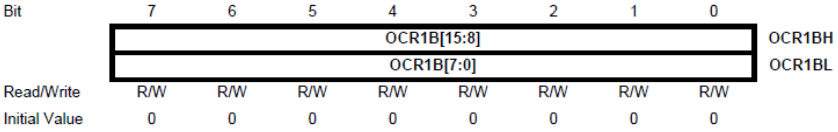


Рисунок 5.22 – Регістр порівняння 1 B – OCR1BH і OCR1BL

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 байту (TEMP). Цей тимчасовий регістр спільно використовується всіма іншими 16-розрядними регістрами.

Регістр захоплення 1 – ICR1H і ICR1L (рис. 5.23).

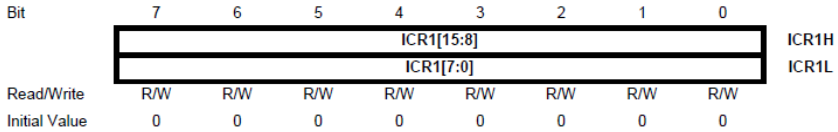


Рисунок 5.23 – Регістр захоплення 1 – ICR1H і ICR1L

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 байту (TEMP). Цей тимчасовий регістр спільно використовується всіма іншими 16-розрядними регістрами.

Регістр маски переривання таймерів-лічильників – TIMSK (рис. 5.24)

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	–	TOIE0	TIMSK
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 5.24 – Регістр маски переривання таймерів-лічильників – TIMSK

Біт 5 – TICIE1: таймер-лічильник 1, 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-лічильник 1, 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-лічильник 1, 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-лічильника 1 увімкнено. Відповідний вектор переривання активується, коли встановлюється прапор OCF1B, розташований у TIFR.

Біт 2 – TOIE1: таймер-лічильник 1, увімкнення переривання при переповненні

Коли цей біт встановлено як 1, і прапор I у регістрі стану встановлено, переривання при переповненні таймера-лічильника 1 увімкнено. Відповідний вектор переривання активується, коли встановлюється прапор TOV1, розташований у TIFR.

Регістр прапорів переривання таймерів-лічильників – TIFR (рис. 5.25).

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 5.25 – Регістр прапорів переривання таймерів-лічильників – TIFR

Біт 5 – ICF1: таймер-лічильник 1, 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-лічильник 1, прапор переповнення

Налаштування цього прапора залежить від налаштування бітів WGM13:0. У звичайному режимі та режимі OTЗ прапор TOV1 встановлюється, коли таймер переповнюється. Зверніться до таблиці 5.7 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.

Контрольні запитання до теми 5

1. Яка основна особливість таймера-лічильника 0 мікроконтролера ATMega8?
2. Що таке TCNT0 і яку функцію він виконує?
3. Які регістри використовуються для маскування переривань у таймері/лічильнику 0?
4. Як таймер-лічильник 0 може тактуватися?
5. Що відбувається, коли таймер-лічильник 0 досягає свого максимального значення?
6. Які біти у регістрі TCCR0 відповідають за вибір тактового сигналу таймера-лічильника 0?
7. Що трапляється з таймером-лічильником 0, якщо біти CS02:0 у регістрі TCCR0 встановлені в значення 0?
8. Як можна збільшити роздільну здатність таймера-лічильника за допомогою програмного забезпечення?
9. Що означає, коли встановлюється біт TOV0 у регістрі TIFR?
10. Які основні функції 16-розрядного таймера-лічильника мікроконтролера ATMega8?
11. Як використовуються подвійно буферизовані регістри порівняння (OCR1A/B) у 16-розрядному таймері/лічильнику?
12. Що таке регістр захоплення (ICR1) і як він працює в контексті 16-розрядного таймера-лічильника?
13. Які режими роботи доступні для 16-розрядного таймера-лічильника і як вони впливають на функціонування лічильника?
14. Які джерела тактового сигналу можуть використовуватися для таймера-лічильника 1 і як вибирається джерело тактового сигналу?
15. Які регістри керують роботою 16-розрядного таймера-лічильника і як здійснюється доступ до них?

16. Які основні джерела запуску для блоку захоплення даних?
17. Як вибрати аналоговий компаратор як джерело запуску для блоку захоплення?
18. Що відбувається, коли увімкнено фільтр шуму?
19. Чому важливо своєчасно читати регістр ICR1 при використанні переривання захоплення?
20. Як впливає запис значення в регістр TCNT1 на роботу блоку порівняння?
21. Які функції виконують біти COM1x1:0 при порівнянні і їх взаємозв'язок з генератором сигналу?
22. Як впливають біти COM1x1:0 на перевизначення порту виводу OC1x?
23. Яким чином біти COM1x1:0 впливають на вихідний сигнал в режимі очищення таймера при збігу (ОТЗ)?
24. Які режими роботи таймера визначаються комбінацією бітів WGM13:0 та COM1x1:0?
25. Як відрізняється робота таймера у режимі швидкого ШІМ від режиму ОТЗ з точки зору генерації сигналу?

Використана література

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. 57 с.
2. Atmel: ATMega8, ATmega8L : технічна документація на мікроконтролер. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2486-8-bit-AVR-microcontroller-ATmega8_L_datasheet.pdf

6 АСИНХРОННО-СИНХРОННИЙ ПРИЙМАЧ-ПЕРЕДАВАЧ USART

Метою вивчення теми є ознайомлення з загальними відомостями інтерфейсу UART, його апаратною частиною та регістрами вводу-виводу.

Завдання вивчення теми збігаються з переліком питань для розгляду, що наведений нижче.

Перелік питань до розділу:

6.1. Загальні відомості про інтерфейс UART.
6.2. Апаратна частина UART мікроконтролерів AVR.
6.3. Регістри вводу-виводу модуля USART.
6.4. Приклади налаштування швидкості передачі даних.

6.1 Загальні відомості про інтерфейс UART

Цифрові системи – це обмін і зберігання інформації у формі одиниць і 0. Щоб поділитися цією інформацією з кількома пристроями з різною архітектурою, нам потрібен універсальний підхід до обміну даними. Тут в дію вступають різноманітні протоколи зв'язку, одним із яких є універсальний асинхронний приймач-передавач (UART). Це один із найбільш поширених протоколів зв'язку у вбудованій електроніці. Це послідовний, повнодуплексний, асинхронний протокол зв'язку «плата-плата».

Оскільки UART є повнодуплексним і асинхронним протоколом зв'язку, передача та прийом даних можуть відбуватися одночасно. Тому нам потрібні окремі піни для передачі та отримання даних. У нас є контакт TX для передачі та контакт RX для прийому даних. Передані дані з порту TX одного пристрою приймаються портом

RX іншого пристрою і навпаки. Тому з'єднання між пристроями встановлюються таким чином, що TX1 під'єднується до RX2, а TX2 – до RX1 для здійснення зв'язку. На рис. 6.1 показано, як виконуються підключення для протоколу UART.

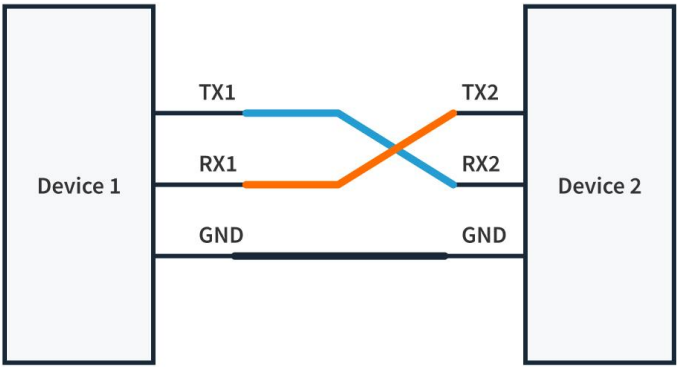


Рисунок 6.1 – Схема підключення пристроїв по інтерфейсу UART

Однією з найпоширеніших помилок під час першого налаштування апаратного забезпечення UART є підключення TX1 до TX2 і RX1 до RX2. Цю помилку важко помітити, і вона викликає розчарування. Тому завжди переконайтеся, що ви з'єднуєте TX одного пристрою з RX іншого. Ще одна важлива річ, на яку слід звернути увагу, це те, що GND пристроїв також має бути підключена. Це важливо, оскільки електронні пристрої зчитують або записують дані про різницю напруги (5 В або 3,3 В ВИСОКА або 1 і 0 В: НИЗЬКА).

Щоб виміряти різницю напруг, нам потрібна опорна напруга. Підключення землі забезпечує те посилання, яке запобігає пошкодженню даних під час зв'язку. Зробивши підключення, ми завершуємо налаштування апаратного забезпечення. Давайте перейдемо до розуміння того, що таке пакет даних UART і що в ньому є (рис. 6.2).

1. **Біти синхронізації.** Є 1 стартовий біт і 1 або 2 стоп-біти (рис. 6.3). Вони вказують на початок і кінець пакета. лінія підтягується до 0, коли передача ініціюється, що вказує на початок кадру, і знову підтягується до 1, коли передача завершується.

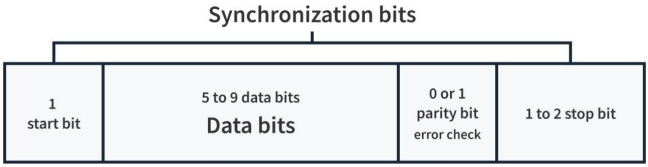


Рисунок 6.2 – Формат пакету даних UART

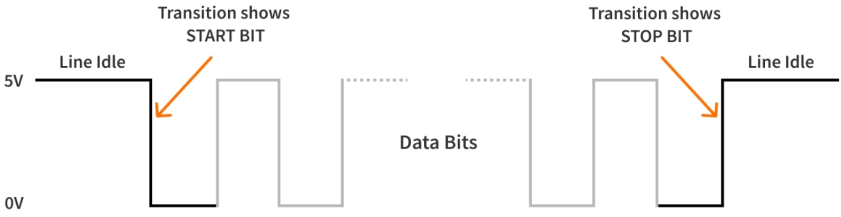


Рисунок 6.3 – Стартовий і стоповий біти для синхронізації в UART

Коли лінія має значення 1, це вважається неактивним станом і означає відсутність передачі даних. Існує лише 1 біт для представлення «початку», але стопові біти можна налаштувати на 1 або 2. Для деяких мікроконтролерів, наприклад серії STM32F4xx, стопові біти також можна налаштувати на 1,5.

2. **Біти даних.** Ця частина кадру даних є суттю питання, оскільки вона переносить інформацію, яку потрібно надіслати одержувачу. Довжина цього блоку може варіюватися від 5 до 9 біт, залежно від узгодженої конфігурації між пристроями. Стандартний розмір даних становить 8 біт або 1 байт. Відповідно до стандартів UART, протокол дотримується порядку від LSB (молодшого значущого біта) до MSB (старшого значущого біта), але його також можна налаштувати для дотримання порядку байтів від MSB до LSB.

3. **Біт парності.** UART постачається з механізмом перевірки помилок для перевірки цілісності отриманих фрагментів даних. Оскільки дані надсилаються байт за байтом в UART, перевірка помилок виконується для кожного отриманого байта. Для цього використовується біт парності; це код перевірки помилок, який надсилається після бітів даних.

Існує два режими біта парності: непарний або парний. Перевірка на непарність: щоб перевірити цілісність блоку, програма перевіряє

кількість одиниць у даних, включаючи біт парності. Передавач виконує наступні кроки для генерації біта парності для кадру даних.

1) Підраховує кількість одиниць у бітах даних.

2) Якщо кількість одиниць непарна, тоді біт парності встановлюється в 0, інакше він встановлюється в 1. Цей процес робить загальну кількість одиниць у фрагменті (біти даних + біт парності) непарною.

Одержувач перевіряє, чи число одиниць у отриманій послідовності (включаючи біт парності) є парним чи непарним. Якщо є непарна кількість 1, дані дійсні. Якщо є парна кількість 1, дані хибні.

Перевірка парності на парність: замість непарної кількості одиниць у даних парна перевірка перевіряє на парну кількість одиниць і дотримується тих самих методів на стороні передавача та приймача. Біт парності встановлюється в 1, якщо є непарна кількість одиниць, і в 0, якщо є парна кількість одиниць у фрагменті.

Кінцева мета полягає в тому, щоб кількість одиниць у передачі даних була рівною. Приймач перевіряє парність одиниць у фрагменті (біти даних + біт парності), щоб перевірити передачу даних. Це простий механізм перевірки помилок, і він не працює, якщо під час передачі пошкоджено більше одного біта. Він також не може визначити, який біт було пошкоджено під час передачі.

У нас готові дані для надсилання. Але як UART синхронізує передачу даних без загального годинника? Це робиться за допомогою конфігурації під назвою «Швидкість передачі даних».

4. **Швидкість передачі даних.** Вона визначає швидкість передачі даних у б/с (бітах за секунду). Щоб зв'язок UART мав місце, пристрої мають бути налаштовані на ідентичні швидкості передачі даних. Деякі стандартні швидкості передачі: 9600, 38400, 115200, 921600 тощо.

Інвертуйте швидкість передачі, і ви матимете час, який потрібен кожному біту для передачі. Наприклад,

$$\frac{1}{9600} = 104{,}16\,\text{мкс}.$$

Отже, зі швидкістю передачі даних 9600 бод кожен біт передається за 104,16 мікросекунди. Що стосується апаратного забезпечення, пристрій підтягне лінію вгору або вниз протягом 104,16 мікросекунди.

6.2 Апаратна частина UART мікроконтролерів AVR

Модуль складається з 3-х основних частин (рисунок 6.4): тактового генератора (контролера) швидкості передачі, блоку приймача та блоку передавача.

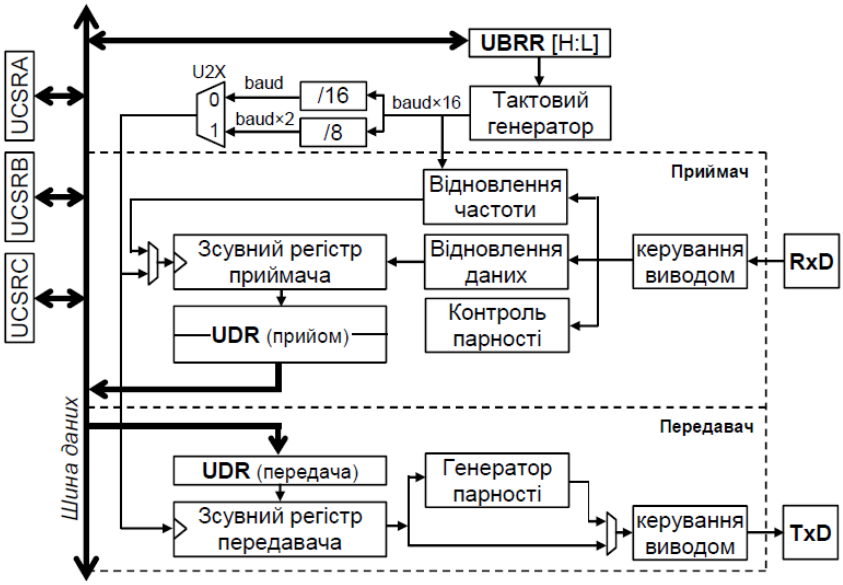


Рисунок 6.4 – Схема модуля UART

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

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:

$$\text{при } U2X = 0: \boldsymbol{BAUD} = \frac{XTAL}{16\left(UBBR + 1\right)};\ \boldsymbol{UBRR} = \frac{XTAL}{16 \cdot BAUD} - 1; \quad (6.1)$$

$$\text{при } U2X = 1: \boldsymbol{BAUD} = \frac{XTAL}{8\left(UBBR + 1\right)};\ \boldsymbol{UBRR} = \frac{XTAL}{8 \cdot BAUD} - 1. \quad (6.2)$$

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:

$$Error[\%] = \left(\frac{BAUD_{\text{розрах.}}}{BAUD} - 1 \right) \cdot 100\%. \quad (6.3)$$

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

$$UBRR = \frac{8 \cdot 10^6}{16 \cdot 9600} - 1 = 51.083 = 51; \quad (6.4)$$

$$BAUD_{\text{розрах.}} = \frac{8 \cdot 10^6}{16(51+1)} = 9615{,}38 \text{ Бод}; \quad (6.5)$$

$$Error[\%] = \left(\frac{9615.38}{9600} - 1 \right) \cdot 100\% = 0.16\%. \quad (6.6)$$

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.

6.3 Регістри вводу-виводу модуля USART

Регістр даних USART – UDR (рис. 6.5)

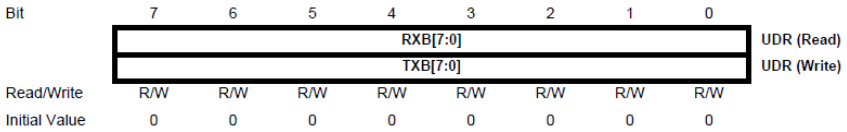


Рисунок 6.5 – Регістр даних USART – UDR

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. 6.6)

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

Рисунок 6.6 – 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. 6.7).

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 6.7 – 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. 6.8).

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 6.8 – 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 зрідка 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 6.1 – 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.

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 6.2 – Довжина символу в залежності від налаштувань

UCSZ2	UCSZ1	UCSZ0	Довжина символу
1	2	3	4
0	0	0	5 біт
0	0	1	6 біт

Продовження таблиці 6.2

1	2	3	4
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. 6.9).

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 6.9 – 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 6.4. 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.

6.4 Приклади налаштування швидкості передачі даних

На рис. 6.10–6.13 представлені розраховані значення регістрів UBRRH та UBRRL для різних швидкостей передачі UART та для різних тактових частот мікропроцесора за формулами (6.1) та (6.2).

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 6.10 – 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	

Рисунок 6.11 – 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	

Рисунок 6.12 – 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	

Рисунок 6.13 – Налаштування регістрів UBRRH та UBRRL при тактових частотах процесора від 16 МГц до 20 МГц

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. 6.10–6.13 виділені жирним шрифтом.

Контрольні запитання до теми 6

1. Що таке USART і які основні режими роботи він підтримує?
2. В чому полягає відмінність між синхронною та асинхронною передачею даних через UART?
3. Які основні функції виконує стартовий біт в кадрі даних UART?
4. Як визначається біт парності в кадрі даних UART і яка його роль?
5. Які параметри формату кадру даних UART визначаються за допомогою регістрів керування UCSRB та UCSRC?
6. Як підключаються два модулі UART для передачі даних між собою?
7. Які основні компоненти входять до складу модуля UART, які їх функції?
8. Які функції виконує блок передавача UART? Опишіть його основні компоненти.

9. Що включає в себе блок приймача UART? Які його основні функції?
10. Які регістри використовуються для зберігання даних в UART? Як вони організовані?
11. Які функції виконує регістр контролера швидкості UBRR? Яким чином він впливає на швидкість обміну даними?
12. Як визначається швидкість обміну даними в асинхронному режимі UART? Як впливає розряд U2X на цей процес?
13. Які основні прапори помилок та статусні біти використовуються для відстеження прийому даних в UART?
14. Який регістр використовується для передачі і прийому даних в USART?
15. Що відбувається, якщо спроба запису до UDR відбувається без встановлення прапорця UDRE?
16. Як визначається кількість бітів у символі даних для приймача і передавача?
17. Яка особливість FIFO буфера прийому USART?
18. Які події можуть призвести до встановлення біту FE (помилка кадру) в регістрі UCSRA?
19. Які можливості надає біт U2X у регістрі UCSRA?
20. Як активувати переривання по завершенню прийому даних (RXC) в USART?
21. Які дії потрібно виконати для вимкнення передавача в USART?
22. Які параметри визначають довжину символу в USART?
23. Як визначити кількість стопових бітів в передачі USART?
24. Що відбувається, якщо спроба зчитування UCSRC регістру не виконана правильно відповідно до послідовності?
25. Які переваги використання дворівневого FIFO буфера в реалізації USART?

Використана література

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. 57 с.
2. Atmel: ATMega8, ATmega8L : технічна документація на мікроконтролер. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2486-8-bit-AVR-microcontroller-ATmega8_L_datasheet.pdf
3. UART – Universal Asynchronous Receiver Transmitter. URL: https://www.circuitbread.com/tutorials/uart-universal-asynchronous-receiver-transmitter

7 ПОСЛІДОВНІ СИНХРОННІ ІНТЕРФЕЙСИ SPI ТА I^2C

Метою вивчення теми є ознайомлення з послідовними синхронними інтерфейсами SPI та I^2C та модулем TWI мікроконтролерів AVR.

Завдання вивчення теми збігаються з переліком питань для розгляду, що наведений нижче.

Перелік питань до розділу:

7.1. Загальні відомості про інтерфейс SPI.
7.2. Регістри вводу-виводу модуля SPI.
7.3. Загальні відомості про інтерфейс I^2C.
7.4. Модуль TWI мікроконтролерів AVR.
7.5. Регістри вводу-виводу модуля TWI.

7.1 Загальні відомості про інтерфейс SPI

Послідовний периферійний інтерфейс (Serial Peripheral Interface, SPI) – це один із найпоширеніших протоколів зв'язку, який мікроконтролери використовують для зв'язку з периферійними пристроями, такими як SRAM, SD-карти, регістри зсуву, датчики тощо.

Це синхронний, повнодуплексний протокол на основі головного та підлеглого. Він підтримує високошвидкісну передачу даних, і існує пряма залежність між швидкістю передачі даних (біт/с) і тактовою частотою (Гц) у протоколі SPI. Наприклад, якщо тактова частота SPI становить 36 МГц, швидкість передачі становитиме 36 Мбіт/с. Таким чином, швидкість передачі даних за протоколом SPI не обмежена. Це залежить виключно від тактової частоти, яку

підтримує пристрій. Якщо говорити про підключення, SPI – це 4-провідний інтерфейс із такими сигнальними лініями:

– Master Out Slave In (MOSI).
– Master In Slave Out (MISO).
– Clock (SCLK).
– Slave Select (SS).

Схема підключення пристроїв за інтерфейсом SPI показана на рис. 7.1.

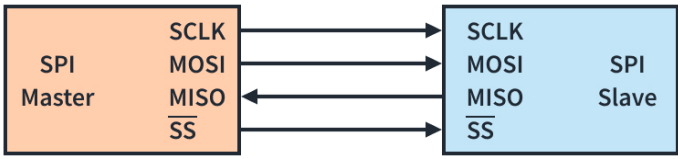


Рисунок 7.1 – Інтерфейс SPI

MISO та MOSI є основними лініями передачі даних. SCLK – це тактовий сигнал, який генерує головний пристрій для вибірки даних на шині. Нарешті, SS – це сигнальна лінія з активним низьким рівнем (високий рівень, коли неактивна, і низький, коли використовується) для вибору веденого на шині.

Підключення пристроїв за допомогою SPI просте та зрозуміле. Кожен контакт на головному з'єднується з тим самим контактом на підлеглому, не залишаючи місця для плутанини. MOSI головного підключається до MOSI підлеглого пристрою, а MISO головного – до MISO підлеглого. Те саме стосується SCLK і також SS.

Як згадувалося раніше, SPI є повнодуплексним інтерфейсом, тобто головний і підлеглий можуть одночасно передавати та отримувати за допомогою відповідних ліній MOSI та MISO. Однією з головних переваг SPI є те, що немає потреби в стартових і стопових бітах, оскільки лінія SCLK шини синхронізує дані. Лінія SCLK синхронізує зсув і дискретизацію бітів на лініях даних.

Щодо кількості бітів, які можна передати, підтримувана довжина слова для протоколу SPI коливається від 3 до 16 бітів, але стандартом є 8 бітів. Якщо головний і ведений хочуть поділитися 2 байтами одночасно, вони можуть вибрати довжину слова 16 біт і використовувати

16-бітний регістр даних (якщо доступний в MCU) або передати його у вигляді фрагмента з двох 8-бітних пакетів даних.

SPI також підтримує кілька підлеглих пристроїв, підключених до одного головного. Для кожного підлеглого має бути окрема лінія SS. Обмін даними відбувається тільки між провідним і обраним веденим; решта підлеглих пристроїв на шині залишаються неактивними і не можуть використовувати свої лінії MOSI та MISO.

Реалізація інтерфейсу SPI в мікроконтролері ATMega8 представлено на рисунку 7.2.

ATmega8 SPI містить такі функції:

– повнодуплексна трипровідна синхронна передача даних;
– головний або ведений режим;
– обмін даними старшим чи молодшим бітом вперед;

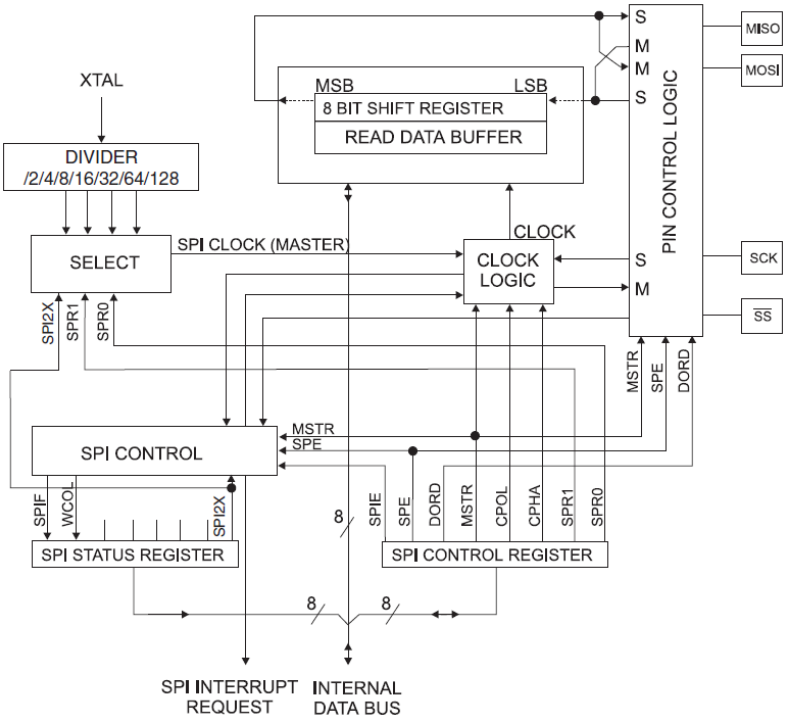


Рисунок 7.2 – Схема інтерфейсу SPI в мікроконтролері ATMega8

– сім програмованих швидкостей передачі;
– прапор переривання по закінченню передачі;
– прапор колізії при записі;
– виведення з режиму очікування;
– подвійна швидкість (CK/2) в головному режимі SPI.

Зв'язок між головним і підлеглим процесорами за допомогою SPI показано на рисунку 7.3. Система складається з двох регістрів зсуву та головного тактового генератора. Головний пристрій SPI (Master) ініціює цикл зв'язку, коли виставляє низький рівень на виводі Slave Select SS потрібного веденого пристрою (Slave). Головний і ведений пристрої готують дані для надсилання у відповідних регістрах зсуву, а головний пристрій генерує необхідні тактові імпульси на лінії SCK для обміну даними. Дані завжди переміщуються від Master до Slave на лінії Master Out – Slave In, MOSI, і від Slave до Master на лінії Master In – Slave Out, MISO. Після кожного пакета даних головний синхронізує веденого, підтягуючи до логічної 1 лінію вибору підлеглого, SS.

Коли інтерфейс SPI налаштований як головний, він не має автоматичного керування лінією SS. Це має бути оброблено програмним забезпеченням користувача перед початком зв'язку. Коли це зроблено, запис байту в регістр даних SPI запускає тактовий генератор SPI, а апаратне забезпечення зсуває вісім бітів у ведений. Після зсуву на один байт тактовий генератор SPI зупиняється, встановлюючи прапор кінця передачі (SPIF). Якщо встановлено біт дозволу переривання SPI (SPIE) у регістрі SPCR, генерується переривання. Головний

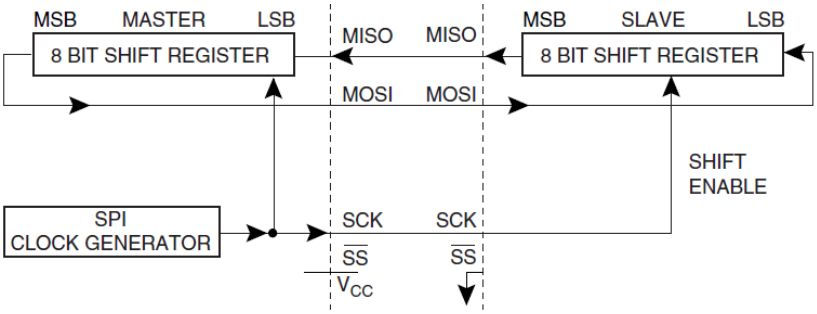


Рисунок 7.3 – Зв'язок між головним і підлеглим процесорами за допомогою SPI

пристрій може продовжувати передавати наступний байт, записуючи його в SPDR, або сигналізувати про кінець пакету, підтягуючи рівень SS до логічної 1. Останній прийнятий байт буде зберігатися в буферному регістрі для подальшого використання.

Якщо інтерфейс SPI налаштовано як ведений, він залишатиметься в режимі сну з високоімпедансним станом MISO, доки на виводі SS буде високий рівень. У цьому стані програмне забезпечення може оновлювати вміст регістра даних SPI, SPDR, але дані не будуть зміщені вхідними синхронізуючими імпульсами на виводі SCK, доки на виводі SS не буде встановлено низький рівень. Після того, як один байт повністю зсувається, встановлюється прапор кінця передачі, SPIF. Якщо встановлено біт дозволу переривання SPI, SPIE, у регістрі SPCR, генерується переривання. Підлеглий пристрій може продовжувати розміщувати нові дані для надсилання в SPDR перед читанням вхідних даних. Останній прийнятий байт буде зберігатися в буферному регістрі для подальшого використання.

Система має одинарну буферизацію в напрямку передачі та подвійну буферизацію в напрямку прийому. Це означає, що байти для передачі не можуть бути записані в регістр даних SPI до завершення всього циклу зсуву. Однак під час отримання даних отриманий символ має бути прочитаний із регістру даних SPI до того, як буде повністю прийнято наступний символ. Інакше перший байт буде втрачено.

У веденому режимі SPI логіка керування буде зчитувати вхідний сигнал з виводу SCK. Щоб забезпечити правильне зчитування тактового сигналу, мінімальний періоди низького і високого рівнів повинні бути довше 2 тактів ЦП.

Коли SPI увімкнено, напрямок даних контактів MOSI, MISO, SCK і SS змінюється відповідно до таблиці 7.1.

Таблиця 7.1 – Напрямок контактів MOSI, MISO, SCK і SS при ввімкненому SPI

Вивід	Напрямок, режим Master	Напрямок, режим Slave
MOSI	Визначається користувачем	Вхід
MISO	Вхід	Визначається користувачем
SCK	Визначається користувачем	Вхід
SS	Визначається користувачем	Вхід

7.2 Регістри вводу-виводу модуля SPI

Регістр управління SPI – SPCR зображено на рисунку 7.4.

Bit	7	6	5	4	3	2	1	0	
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
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 7.4 – Регістр управління SPI – SPCR

Біт 7 – SPIE: дозвіл переривання SPI

Цей біт викликає виконання переривання SPI, якщо встановлено біт SPIF у регістрі SPSR і встановлено біт дозволу глобального переривання в SREG.

Біт 6 – SPE: дозвіл роботи SPI

Коли біт SPE встановлено як одиниця, SPI вмикається. Цей біт має бути встановлено, щоб дозволити будь-які операції SPI.

Біт 5 – DORD: порядок даних

Коли біт DORD встановлено як одиниця, спочатку передається молодший біт байту даних. Коли біт DORD встановлено як 0, старший біт байту даних передається першим.

Біт 4 – MSTR: вибір головного/веденого

Цей біт вибирає режим Master, коли він встановлений як одиниця, і режим Slave, коли він встановлений як логічний нуль. Якщо SS налаштовано як вхід, і його рівень низький, коли встановлено MSTR, MSTR буде очищено, а SPIF у регістрі SPSR стане встановленим. Потім користувачеві доведеться налаштувати MSTR для повторного ввімкнення режиму SPI Master.

Біт 3 – CPOL: полярність тактового сигналу

Коли цей біт встановлений як одиниця, SCK має високий рівень під час простою. Коли CPOL встановлений як нуль, SCK має низький рівень під час простою. Короткий опис функціональності CPOL наведено в таблиці 7.2:

Таблиця 7.2 – Опис функціональності CPOL

CPOL	Передній фронт	Задній фронт
0	Наростаючий	Спадаючий
1	Спадаючий	Наростаючий

Біт 2 – CPHA: фаза тактового сигналу

Налаштування біта фази тактового сигналу (CPHA) визначають, чи вибірка даних здійснюється на передньому (першому) чи задньому (останньому) фронті сигналу SCK. Функціональність CPHA підсумована у таблиці 7.3.

Таблиця 7.3 – Функціональність CPHA

CPHA	Передній фронт	Задній фронт
0	Вибірка	Зсув
1	Зсув	Вибірка

Біти 1, 0 – SPR1, SPR0: вибір тактової частоти SPI 1 і 0

Ці два біти контролюють швидкість сигналу SCK пристрою, налаштованого як головний. SPR1 і SPR0 не впливають на ведений пристрій. Зв'язок між SCK і тактовою частотою осцилятора f_{osc} показано в наступній таблиці 7.4.

Таблиця 7.4 – Зв'язок між SCK і тактовою частотою осцилятора f_{osc}

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

Регістр статусу SPI – SPSR (рис. 7.5).

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

Рисунок 7.5 – Регістр статусу SPI – SPSR

Біт 7 – SPIF: прапор переривання SPI

Після завершення послідовної передачі встановлюється прапор SPIF. Переривання генерується, якщо встановлено біт SPIE у регістрі SPCR і ввімкнено глобальні переривання. Якщо SS є входом і має низький рівень, коли SPI перебуває в режимі Master, це також встановить прапор SPIF. SPIF очищується апаратним забезпеченням під час виконання відповідного вектору обробки переривань. Крім того, біт SPIF очищується, якщо спочатку зчитати регістр стану SPI із встановленим SPIF, а потім звернутися до регістру даних SPI (SPDR).

Біт 6 – WCOL: прапор колізії при запису

Біт WCOL встановлюється при запису в регістр даних SPI (SPDR) під час передачі даних. Біт WCOL (та біт SPIF) очищується, якщо спочатку зчитати регістр стану SPI із встановленим WCOL, а потім звернутися до регістру даних SPI.

Біти 5..1 – Res: зарезервовані біти

Ці біти є зарезервованими бітами в ATmega8 і завжди читатимуться як нуль.

Біт 0 – SPI2X: біт подвоєння швидкості SPI

Коли цей біт встановлено, як логічна одиниця, швидкість SPI (частота SCK) буде подвоєна, якщо SPI перебуває в режимі Master (див. таблицю 7.4). Це означає, що мінімальний період SCK становитиме 2 тактових періоди ЦП. Коли SPI налаштовано як Slave, SPI гарантовано працюватиме лише на fosc/4 або нижче.

Інтерфейс SPI у ATmega8 також використовується для завантаження чи зчитування програмної пам'яті та EEPROM.

Регістр даних SPI – SPDR (рис. 7.6).

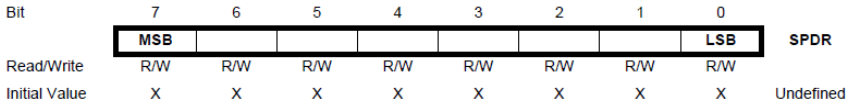


Рисунок 7.6 – Регістр даних SPI – SPDR

Регістр даних SPI – це регістр призначений для читання/запису. Він використовується для передачі даних між регістровим файлом та регістром зсуву SPI. Запис до регістру ініціює передачу даних. Читання регістру викликає зчитування буфера прийому регістра зсуву.

Режими роботи SPI. Існує чотири комбінації фази і полярності SCK відносно до даних, які визначаються керуючими бітами CPHA і CPOL. Формати передачі даних SPI показано на рисунку 7.7. Біти даних зсуваються та фіксуються на протилежних фронтах сигналу SCK, забезпечуючи достатній час для стабілізації сигналів даних. Це чітко видно з наступної таблиці 7.5:

Таблиця 7.5 – Комбінації фази і полярності SCK відносно до даних

Комбінація бітів	**Передній фронт**	**Задній фронт**	**Режим SPI**
CPOL = 0 CPHA = 0	Вибірка (наростаючий)	Зсув (спадаючий)	0
CPOL = 0 CPHA = 1	Зсув (наростаючий)	Вибірка (спадаючий)	1
CPOL = 1 CPHA = 0	Вибірка (спадаючий)	Зсув (наростаючий)	2
CPOL = 1 CPHA = 1	Зсув (спадаючий)	Вибірка (наростаючий)	3

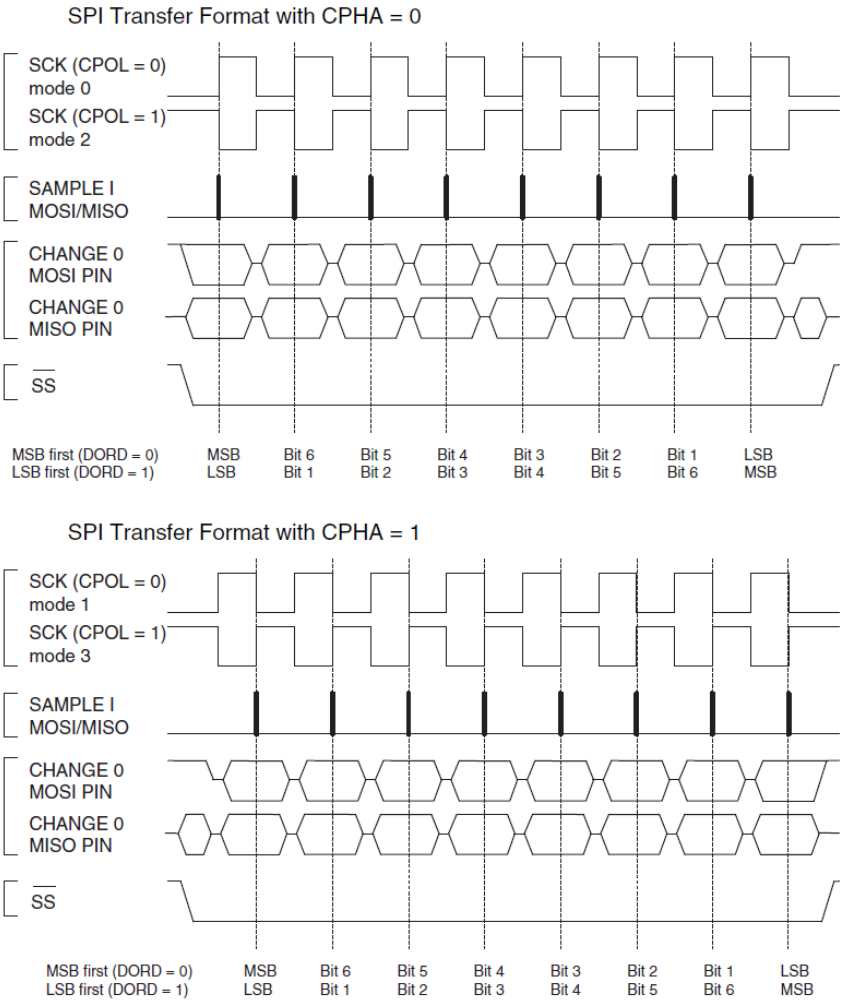


Рисунок 7.7 – Формати передачі даних SPI

7.3 Загальні відомості про інтерфейс I²C

Inter-Integrated Circuit Protocol (I²C або IIC) – це послідовний, синхронний, напівдуплексний протокол зв'язку між платами з декількома головними пристроями. Як випливає з назви, він в основному використовується для зв'язку всередині друкованих плат (PCB). Компанія Philips Semiconductors винайшла його в 1982 році з метою використання меншої кількості контактів мікроконтролера для роботи з іншими електронними пристроями (рис. 7.8). Він використовує лише дві лінії для зв'язку з підключеними пристроями; тому іноді його також називають двопровідним протоколом. I2C підтримує конфігурацію провідний-підлеглий, але термінологія тут змінюється з провідного-підлеглого на контролер-ціль або контролер-периферійний пристрій.

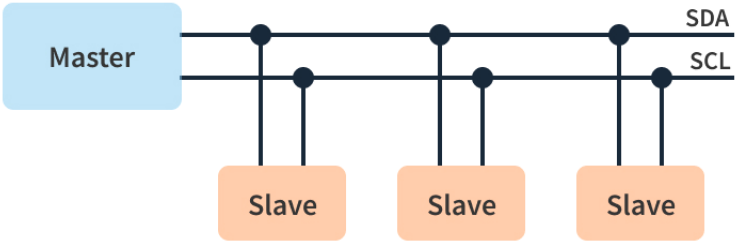


Рисунок 7.8 – Кілька підлеглих підключені до однієї шини з одним головним пристроєм

Апаратний інтерфейс. Фізичний інтерфейс I2C складається з двох ліній: SDA і SCL. SCL (Serial Clock) – це тактовий сигнал головного пристрою шини, а SDA (Serial Clock) – це сигнал даних. Драйвери I2C є «відкритими стоками», тобто пристрій може лише заземлити вихід або перевести свій вихід у стан високого опору, що означає, що він не може встановити свій вихід в стан логічної 1 (рис. 7.9). Стан високого опору означає, що вихід ні до чого не підключений, тобто він знаходиться в плаваючому стані. Пара зовнішніх резисторів, кожен на лініях SDA і SCA, підключається

для підтягування ліній, коли пристрої переводять свої виходи в стан високого опору. Ця конфігурація з відкритим стоком запобігає короткому замиканню в лініях, оскільки головний і всі підлеглі пристрої, з'єднані разом, ніколи не можуть подавати конфліктні напруги на одну лінію.

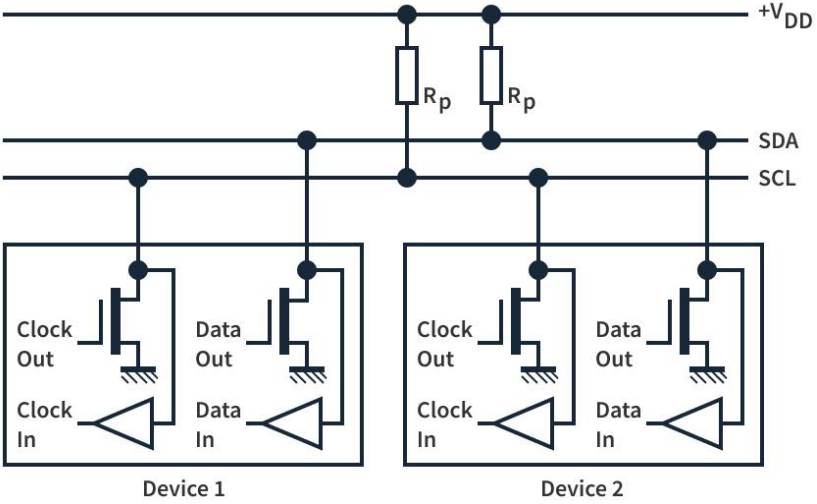


Рисунок 7.9 – Драйвери з відкритим стоком шини I²C

Буфери Clock In і Data In драйверів I²C служать вхідними даними для пристроїв I²C, оскільки вони використовуються для зчитування стану ліній. З іншого боку, MOSFET контролює вихід на лінії, коли пристрій записує на шину.

Шина I²C підтримує різні режими, які підтримують різні бітрейти для обміну даними (таблиця 7.6).

Таблиця 7.6 – Режими роботи шини I²C

Режим	Швидкість
Стандартний	100 кб/с
Швидкий	400 кб/с
Швидкий плюс	1 Мб/с
Високої швидкості	3,4 Мб/с

Слід зазначити, що ці бітрейти визначають швидкість передачі даних по шині, а не швидкість обробки пристрою.

Пакет даних. Протокол I²C працює в конфігурації головний-підлеглий (контролер-периферійний пристрій), тому привілеї читання-запису на шині має лише головний. Головний вирішує, який підлеглий пристрій буде отримувати або надсилати дані головному. I²C має спеціалізований пакет даних (рис. 7.10) і може спілкуватися з до 128 пристроями на шині.

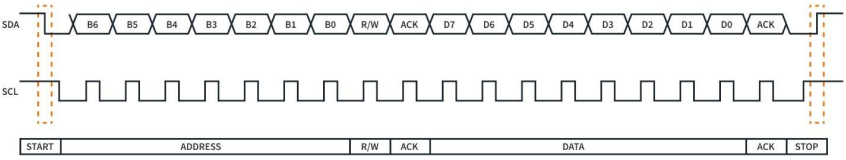


Рисунок 7.10 – Формат пакету даних шини I²C

Умова «СТАРТ» (START). У неактивному стані зв'язку лінії SDA і SCL підтягуються до напруги живлення за допомогою підтягувальних резисторів. Щоб ініціювати зв'язок, головний пристрій перемикає лінію SDA в низький рівень, поки лінія SCL залишається високою, і попереджає підлеглі пристрої на шині про готовність до зв'язку.

Адресація підлеглого пристрою. Після цієї початкової умови головний пристрій надсилає 7-бітну адресу для підлеглого пристрою, з яким він хоче спілкуватися. Адреса зчитується всіма пристроями, підключеними до шини, і пристрій з цією адресою відповідає на запити головного.

Операція читання-запису (R/W). За 7-бітовою адресою слід один біт R/W – «1» для операції читання, і «0» для операції запису. R/W вирішує, хто отримає лінію SDA для передачі даних: головний чи підлеглий. Операції читання та запису виконуються з точки зору головного пристрою та є такими:

– Операція зчитування – головний отримає дані, які передає підлеглий, тому це також називається зчитуванням із шини.

Операція запису – головний передає дані підлеглому. Тому це називається записом в шину.

Важливо відзначити, що саме головний (а не підлеглий) генерує тактовий сигнал на лінії SCL для вибірки даних на лінії SDA.

Підтвердження / не підтвердження (ACK/NACK). Щоб перевірити, чи підлеглий пристрій під'єднано до шини, чи зайнятий він чи ні, головний пристрій очікує підтвердження від підлеглого пристрою, тобто головний пристрій чекає, поки підлеглий пристрій підтягне лінію SDA до низького рівня на 9-му такті. Якщо така умова задовольняється, головний отримує позитивну відповідь від підлеглого, що називається ACK. Але якщо лінія SDA залишається високою на 9-му такті, це називається NACK. Цей механізм дозволяє головному перевірити, чи присутній ведений із заданою адресою на шині чи ні. Інше використання ACK/NACK полягає в тому, щоб визначити, чи підлеглий пристрій отримав передані біти без помилок чи ні.

Дані (DATA). Дані, надіслані або отримані майстром, є фактичними даними, для яких здійснюється зв'язок. Усі інші розділи кадру даних є допоміжними механізмами протоколу. Як згадувалося раніше, є дві операції: запис і читання.

Операція запису. Спочатку головний адресує ведений на шині, надсилаючи 7-бітну адресу. Якщо ведений присутній або активний на шині, він готується до обміну даними. Після цього головний пристрій надсилає біт W, щоб сповістити підлеглий пристрій, що він перейме на себе лінію SDA та надішле дані підлеглому. Головний чекає ACK від підлеглого під час наступного такту. Отримавши ACK від підлеглого пристрою, головний пристрій надсилає «дані» біт за бітом на кожному такті. Головний записує дані в шину, і тому це називається операцією «запису» в протоколі I^2C.

Операція читання. Якщо головний пристрій бажає отримати дані від підлеглого, він звертається до підлеглого на шині та надсилає біт R і очікує ACK від підлеглого. Після отримання ACK від підлеглого пристрою, головний пристрій дозволяє йому перейти на лінію SDA для передачі даних головному. Головний тут зчитує дані з шини, тому це називається операцією «читання» в протоколі I^2C.

Одна з переваг використання I^2C перед UART полягає в тому, що зв'язок підтримується головним, і немає необхідності повторно ініціалізувати зв'язок, доки не буде завершена передача всіх

даних, тобто немає обмежень на кількість бітів, які можна передати на кадр даних. Наприклад: якщо головний пристрій хоче записати 32 біти даних у підлеглий пристрій, немає потреби починати та завершувати зв'язок 4 рази (8 біт за раз). Після кожного ACK від підлеглого пристрою може бути надісланий ще один фрагмент із 8 біт. На відміну від UART, I2C зменшує накладні витрати, спричинені початковими та стоповими бітами для кожного кадру даних. Але нам потрібно завершити кадр, якщо ми хочемо змінити режим роботи з читання на запис або навпаки.

Окрім зменшення накладних витрат бітів, ACK/NACK також працює як механізм перевірки помилок у шині I^2C. Для операції запису, якщо головний отримує NACK від підлеглого, він повторно надсилає дані підлеглому. І якщо NACK отримано, коли головний зчитує з підлеглого, головний відкидає отримані біти.

Умова «СТОП» (STOP). Коли передача завершена, головний надсилає умову «СТОП», змінюючи лінію SDA з низького на високий, тоді як SCL є високим. Це завершує транзакцію та повертає зв'язок у стан очікування.

На цьому ми завершили пояснення кадру даних I^2C і всіх його розділів. Але це ще не все. I2C пропонує кілька додаткових функцій, які надзвичайно корисні для створення надійної системи. Ці функції підтримуються не всіма пристроями I^2C, але про них потрібно знати.

Розтягування тактового сигналу. Як згадувалося раніше, швидкість передачі даних, яку підтримує пристрій для зв'язку I^2C, не обов'язково відповідає швидкості обробки даних пристроєм. Тож що, якщо головний просить деякі дані від підлеглого, але він ще не готовий із запитаними даними? Оскільки головний пристрій очікує ACK від підлеглого під час наступного тактового циклу, і якщо підлеглому не вдасться підтягнути лінію SDA до низького рівня, головний пристрій вважатиме, що зв'язок перервався. Щоб подолати це обмеження зв'язку, підлеглий пристрій використовує «розтягнення тактового сигналу», щоб сповістити головний, що він зайнятий і потребує більше часу для обробки даних. Підлеглий пристрій досягає цього шляхом підтягування лінії SCL до низького рівня, що призупиняє зв'язок, оскільки вибірка даних в I^2C

виконується по передньому фронту лінії SCL (рис. 7.11). Не всі пристрої I^2C підтримують розтягування тактового сигналу; отже, для підтвердження цього необхідна специфікація пристрою. Іноді це також може спричинити проблеми, тому що якщо пристрій несправний і тримає лінію SCL у низькому стані, вся шина зупиняється, і головний не може спілкуватися з іншими підлеглими.

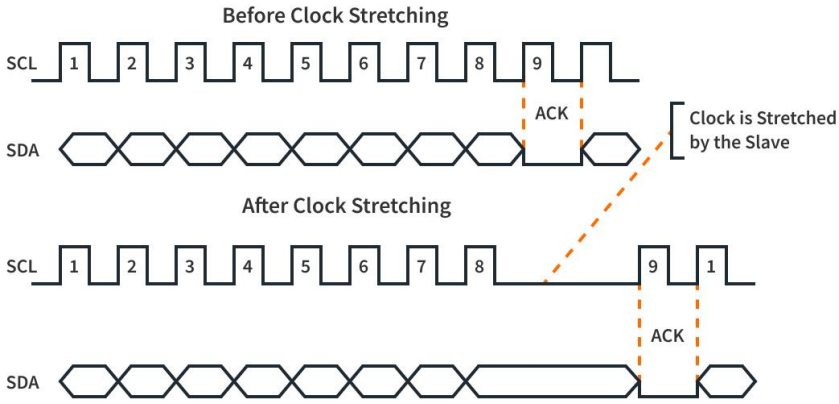


Рисунок 7.11 – Реалізація розтягування тактового сигналу

Арбітраж шини. Шина I2C підтримує кілька головних пристроїв і може спілкуватися з усіма пристроями, підключеними до шини. Головні, підключені до шини, постійно контролюють лінії SDA та SCL для умов запуску та зупинки та утримують незавершені передачі, доки шина знову не буде вільна. Таким чином вони працюють одночасно на одній шині, але може виникнути ситуація, коли передача ініціюється обома майстрами одночасно. Щоб уникнути цього, майстри на шині постійно контролюють лінію SDA, щоб перевірити, чи лінія SDA підтягнута вниз іншим головним чи ні. Якщо один із них виявляє, що SDA має низький рівень, тоді як він повинен бути високим, він робить висновок, що інший головний наразі активний, і негайно припиняє власну передачу. Ця процедура називається арбітражем шини (рис. 7.12).

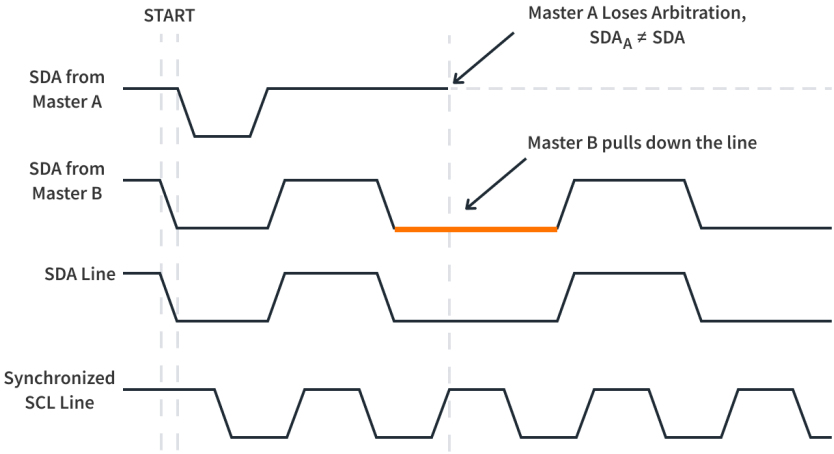


Рисунок 7.12 – Арбітраж шини I^2C

Як показано на рис. 7.12, і Master A, і Master B ініціюють передачу даних одночасно. Однак Майстру B вдається підтягнути лінію SDA вниз, тоді як Майстер A хоче, щоб лінія SDA мала високий рівень. Цей конфлікт виявляє Master A, і він програє арбітраж, тобто більше не контролює лінію SDA.

7.4 Модуль TWI мікроконтролерів AVR

Двопровідний послідовний інтерфейс TWI (Two Wire Interface)

Апаратний модуль TWI входить до складу багатьох мікроконтролерів AVR.

Основні характеристики інтерфейсу TWI наступні:

Простий, але потужний і гнучкий інтерфейс зв'язку, потрібно лише дві лінії шини.

Підтримка як головного (Master), так і веденого (Slave) режимів роботи.

Пристрій може працювати як передавач або приймач.

7-бітний адресний простір дозволяє підключити до 128 різних ведених пристроїв.

Підтримка арбітражу для конфігурації з багатьма головними пристроями (multimaster).

Швидкість передавання даних до 400 кГц.

Обмеження швидкості наростання вихідних сигналів.

Схема придушення шуму відфільтровує імпульсні завади на лініях шини.

Повністю програмована адреса веденого пристрою з підтримкою загального виклику (General Call).

Розпізнавання адреси дозволяє вихід із режиму сну для AVR.

Двопровідний послідовний інтерфейс (TWI) ідеально підходить для типових застосувань мікроконтролерів. Протокол TWI дозволяє системному розробнику з'єднувати до 128 різних пристроїв, використовуючи лише дві двонаправлені шини: одну для тактування (SCL) і одну для даних (SDA). Єдиним зовнішнім апаратним забезпеченням, необхідним для реалізації шини, є один підтягувальний резистор для кожної з ліній TWI (рис. 7.13). Усі пристрої, підключені до шини, мають унікальні адреси, а механізми розв'язання конфліктів на шині є невід'ємною частиною протоколу TWI.

Модуль TWI складається з кількох підмодулів, як показано на рисунку 7.14. Усі регістри, позначені жирною лінією, доступні через шину даних AVR.

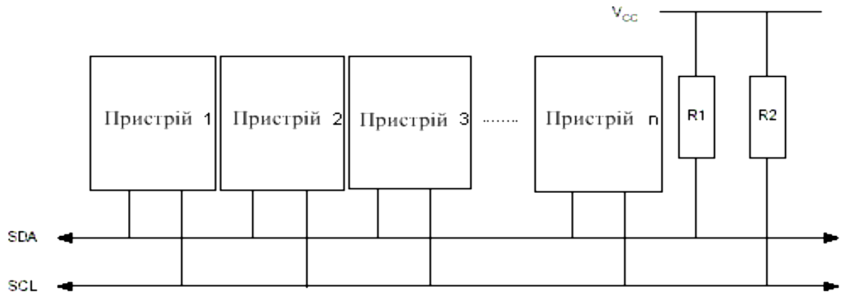


Рисунок 7.13 – Реалізація шини

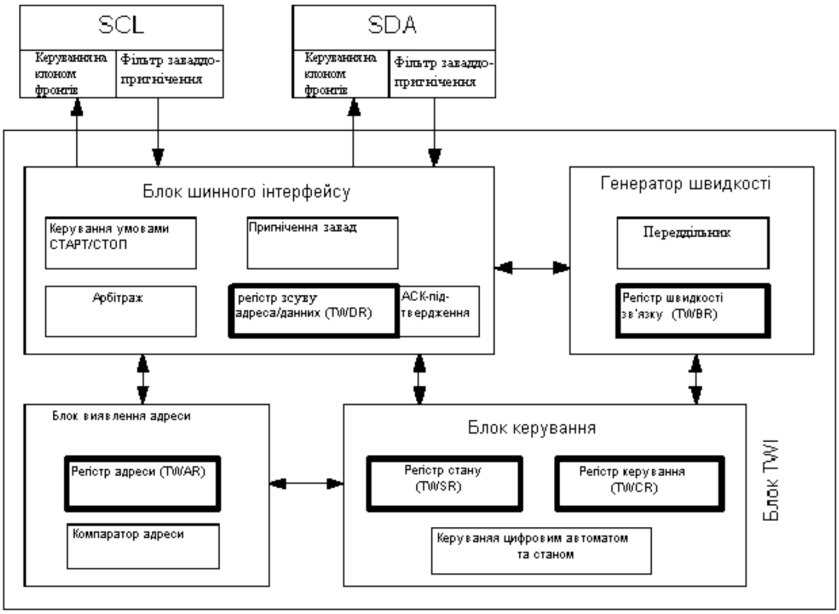


Рисунок 7.14 – Модуль TWI

Виводи SCL і SDA. Ці виводи забезпечують інтерфейс між модулем TWI AVR та рештою системи МК. Вихідні драйвери містять обмежувач швидкості наростання сигналу для відповідності специфікації TWI. Вхідні каскади оснащені схемою придушення імпульсних завад, що відфільтровує імпульси тривалістю менше ніж 50 нс.

Зверніть увагу, що внутрішні підтягувальні резистори у виводах AVR можна ввімкнути, встановивши відповідні біти PORT для ліній SCL і SDA, як описано в розділі про порти вводу-виводу. У деяких системах це дозволяє обійтися без зовнішніх підтягувальних резисторів.

Блок генератора швидкості передачі. Цей блок керує періодом сигналу SCL під час роботи в режимі головного (Master). Період SCL визначається налаштуваннями в регістрі швидкості передачі TWI (TWBR) та бітах попереднього дільника (Prescaler) у регістрі стану TWI (TWSR).

Режим веденого (Slave) не залежить від налаштувань швидкості передачі або дільника, але тактова частота процесора у веденому пристрої має бути принаймні в 16 разів вищою за частоту SCL. Зверніть увагу, що ведені пристрої можуть подовжувати низький рівень сигналу SCL, тим самим збільшуючи середній період тактування шини TWI.

Частота SCL розраховується за такою формулою:

$$f_{SCL} = \frac{f_Q}{16 + 2 \cdot TWBR \cdot 4^{TWPS}}, \qquad (7.1)$$

де f_Q – частота кварцового генератора МК;

TWBR – значення регістра швидкості передачі TWI (TWI Bit Rate Register);

TWPS – значення бітів попереднього дільника в регістрі стану TWI (TWI Status Register).

Блок шинного інтерфейсу. Цей блок містить регістр зсуву даних і адреси (TWDR), контролер START/STOP і апаратний модуль виявлення арбітражу. TWDR зберігає адресні або інформаційні байти, що передаються або приймаються. Окрім 8-бітного TWDR, блок інтерфейсу шини містить також регістр, що зберігає біти (N)ACK для передавання або приймання. Цей регістр (N)ACK недоступний безпосередньо для програмного забезпечення, але при прийманні його можна встановлювати або скидати, змінюючи значення у регістрі керування TWI (TWCR). У режимі передавача (Transmitter) значення прийнятого біта (N)ACK можна визначити за вмістом регістра стану TWI (TWSR).

Контролер START/STOP відповідає за генерацію та виявлення умов START, REPEATED START і STOP. Він може розпізнавати ці умови навіть тоді, коли мікроконтролер AVR перебуває в одному з режимів сну, що дозволяє пробудження МК у разі звернення до нього головного пристрою (Master).

Якщо TWI ініціює передавання в режимі головного (Master), цей модуль постійно контролює передавання, намагаючись визначити, чи відбувається арбітраж. Якщо пристрій втрачає арбітраж, блок керування отримує відповідне сповіщення. Після цього можуть бути виконані коригувальні дії та згенеровані відповідні коди стану.

Блок порівняння адреси. Блок порівняння адреси перевіряє, чи збігається отриманий адресний байт із семибітною адресою в регістрі адреси TWI (TWAR). Якщо біт TWI General Call Recognition Enable (TWGCE) у TWAR записано в одиницю, всі вхідні адресні біти також будуть порівнюватися з адресою загального виклику. При збігу адреси блок керування отримує повідомлення, що дозволяє виконати відповідні дії. TWI може або підтвердити свою адресу, або ні, залежно від налаштувань у TWCR. Блок порівняння адреси може порівнювати адреси навіть коли AVR MCU знаходиться в режимі сну, що дозволяє MCU прокинутися, якщо до нього звертається головний пристрій. Якщо під час порівняння адреси в режимі Power-down виникає інше переривання (наприклад, INT0), яке пробуджує процесор, TWI припиняє операцію і повертається в стан очікування. Якщо це викликає проблеми, переконайтеся, що порівняння адреси TWI є єдиним активним перериванням при вході в Power-down.

Блок керування. Блок керування моніторить шину TWI і генерує відповіді відповідно до налаштувань у регістрі керування TWI (TWCR). Коли на шині TWI відбувається подія, яка вимагає уваги програми, встановлюється прапор переривання TWI (TWINT). У наступному такті тактового сигналу регістр стану TWI (TWSR) оновлюється статусним кодом, що ідентифікує подію. TWSR містить релевантну інформацію про статус тільки коли прапор переривання TWI встановлений. В інші часи TWSR містить спеціальний код стану, що вказує на відсутність релевантної інформації про статус. Поки прапор TWINT встановлений, лінія SCL утримується в низькому стані. Це дозволяє програмному забезпеченню завершити свої завдання перед тим, як дозволити продовження передачі TWI.

Прапор TWINT встановлюється в таких ситуаціях:

Після того, як TWI передав умову START/REPEATED START.

Після того, як TWI передав SLA+R/W.

Після того, як TWI передав адресний байт.

Після того, як TWI втратив арбітраж.

Після того, як TWI отримав адресу свого веденого пристрою або загальний виклик.

Після того, як TWI отримав байт даних.

Після того, як було отримано умову STOP або REPEATED START, перебуваючи в режимі веденого.

Коли сталася помилка шини через нелегальну умову START або STOP.

7.5 Регістри вводу-виводу модуля TWI

Регістр швидкості зв'язку TWI – TWBR (рис. 7.15).

Bit	7	6	5	4	3	2	1	0	
	TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0	TWBR
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 7.15 – Регістр швидкості зв'язку TWI – TWBR

TWBR вибирає коефіцієнт поділу для генератора швидкості передачі. Генератор швидкості передачі є дільником частоти, який генерує частоту тактового сигналу SCL у режимах головного пристрою (Master).

Регістр управління TWI – TWCR (рис 7.16).

Bit	7	6	5	4	3	2	1	0	
	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	TWCR
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 7.16 – Регістр управління TWI – TWCR

TWCR використовується для керування роботою TWI. Він використовується для увімкнення TWI, ініціації доступу веденого шляхом подачі умови START на шину, генерації підтвердження приймача (Receiver acknowledge), генерації умови STOP і для керування зупинкою шини під час запису даних до шини через TWDR.

Він також вказує на зіткнення запису, якщо спробувати записати дані в TWDR, коли регістр недоступний.

Біт 7 – TWINT: Прапор переривання TWI

Цей біт встановлюється апаратно, коли TWI завершив свою поточну операцію і чекає відповіді від програмного забезпечення. Якщо біти I у SREG і TWIE у TWCR встановлені, МК перейде до вектору переривання TWI. Поки прапор TWINT встановлений, період низького рівня сигналу SCL подовжується. Прапор TWINT повинен бути очищений програмним забезпеченням шляхом запису логічної одиниці. Зверніть увагу, що цей прапор не очищається автоматично апаратно під час виконання обробника переривання. Також зверніть увагу, що очищення цього прапора ініціює роботу TWI, тому всі звернення до регістра адреси TWI (TWAR), регістра стану TWI (TWSR) і регістра даних TWI (TWDR) повинні бути завершені до очищення цього прапора.

Біт 6 – TWEA: Біт підтвердження TWI (TWI Enable Acknowledge Bit).

Біт TWEA контролює генерацію імпульсу підтвердження. Якщо біт TWEA записано в одиницю, імпульс ACK генерується на шині TWI, якщо виконуються такі умови:

1. Була отримана адреса власного веденого пристрою.
2. Була отримана адреса загального виклику, коли біт TWGCE у TWAR встановлено в одиницю.
3. Був отриманий байт даних у режимі приймача веденого або приймача головного пристрою.

Записавши біт TWEA в нуль, пристрій можна тимчасово відключити від шини TWI. Визнання адреси можна відновити, знову записавши біт TWEA в одиницю.

Біт 5 – TWSTA: Біт умови START TWI (TWI START Condition Bit). Програма записує біт TWSTA в одиницю, коли вона бажає стати головним пристроєм на шині TWI. Апаратне забезпечення TWI перевіряє, чи вільна шина, і генерує умову START на шині, якщо вона вільна. Якщо ж шина не вільна, TWI чекає, поки не буде виявлена умова STOP, а потім генерує нову умову START, щоб отримати статус головного пристрою. Біт TWSTA повинен бути очищений програмним забезпеченням після того, як умова START буде передана.

Біт 4 – TWSTO: Біт умови STOP TWI (TWI STOP Condition Bit).

Записавши біт TWSTO в одиницю в режимі головного пристрою, буде згенеровано умову STOP на шині TWI. Коли умова STOP виконується на шині, біт TWSTO автоматично очищається.

У режимі веденого пристрою, запис цього біта використовується для відновлення після помилкової умови. Це не генерує умову STOP, але TWI повертається до чітко визначеного стану веденого пристрою без адреси та звільняє лінії SCL і SDA до стану високого імпедансу.

Біт 3 – TWWC: Біт зіткнення запису TWI (TWI Write Collision Flag).

Біт TWWC встановлюється, коли спроба запису в реєстр даних TWI (TWDR) здійснюється, коли TWINT низький. Цей прапор очищується записом у реєстр TWDR, коли TWINT високий.

Біт 2 – TWEN: Біт увімкнення TWI (TWI Enable Bit).

Біт TWEN дозволяє роботу TWI та активує інтерфейс TWI. Коли TWEN записано в одиницю, TWI бере під контроль I/O піни, підключені до пінів SCL і SDA, увімкнувши обмежувачі швидкості наростання та фільтри імпульсів. Якщо цей біт записано в нуль, TWI вимикається, і всі передачі TWI припиняються, незважаючи на будь-які поточні операції.

Біт 1 – Res: Зарезервований біт.

Цей біт зарезервований і завжди читається як нуль.

Біт 0 – TWIE: Увімкнення переривання TWI (TWI Interrupt Enable).

Коли цей біт записано в одиницю, а біт I в SREG встановлений, запит на переривання TWI активується, поки біт TWINT високий.

Регістр стану TWI – TWSR (рис. 7.17).

Bit	7	6	5	4	3	2	1	0	
	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	TWSR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	1	1	1	1	1	0	0	0	

Рисунок 7.17 – Регістр стану TWI – TWSR

Біти 7..3 – TWS: Стан TWI

Ці 5 біт відображають стан логіки TWI та шини Two-wire Serial Bus. Різні коди стану описані далі. Зверніть увагу, що значення, яке читається з TWSR, містить як 5-бітне значення стану, так і 2-бітне значення переддільника. Розробнику програми слід маскувати біти переддільника в нуль під час перевірки стану. Це робить перевірку стану незалежною від налаштування переддільника.

Біт 2 – Res: Зарезервований біт.

Цей біт зарезервований і завжди читається як нуль.

Біти 1..0 – TWPS: Біти переддільника TWI (TWI Prescaler Bits).

Ці біти можуть бути прочитані та записані і контролюють переддільник швидкості передачі, як показано в табл. 7.7.

Таблиця 7.7 – Значення переддільника TWI за бітами

TWPS1	TWPS0	Значення переддільника
0	0	1
0	1	4
1	0	16
1	1	64

Регістр даних TWI – TWDR (рис. 7.18).

Bit	7	6	5	4	3	2	1	0	
	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0	TWDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	1	

Рисунок 7.18 – Регістр даних TWI – TWDR

У режимі передачі TWDR містить наступний байт для передачі. У режимі прийому TWDR містить останній отриманий байт. Він може бути записаний, коли TWI не знаходиться в процесі зміщення байта. Це відбувається, коли біт переривання TWI (TWINT) встановлено апаратно. Зверніть увагу, що реєстр даних не може бути ініціалізований користувачем до того, як відбудеться перше переривання. Дані в TWDR залишаються стабільними, поки TWINT встановлено. Поки дані передаються, дані на шині одночасно приймаються.

TWDR завжди містить останній байт, що присутній на шині, за винятком випадку пробудження з режиму сну через переривання TWI. У цьому випадку вміст TWDR не визначений. У разі втрати арбітражу на шині дані не втрачаються при переході від головного пристрою до веденого. Обробка біта ACK контролюється автоматично логікою TWI, ЦП не може безпосередньо отримувати доступ до біта ACK.

Біти 7..0 – TWD: Реєстр даних TWI (TWI Data Register).

Ці вісім біт складають наступний байт даних для передачі або останній байт даних, отриманий на шині TWI.

Регістр адреси веденого шини TWI – TWAR (рис. 7.19).

Bit	7	6	5	4	3	2	1	0	
	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	TWAR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	0	

Рисунок 7.19 – Регістр адреси веденого шини TWI – TWAR

TWAR має бути завантажений 7-бітною адресою веденого пристрою (у семи найбільш значущих бітах TWAR), на яку TWI відповідатиме при налаштуванні як ведений передавач або приймач. У головному режимі це не потрібно. У мультимастерних системах TWAR має бути налаштований у головних пристроях, які можуть бути адресовані іншими головними пристроями як ведені.

Молодший біт TWAR використовується для увімкнення розпізнавання загальної адреси виклику (0x00). Існує відповідний компаратор адреси, який перевіряє отриману серійну адресу на збіг із адресою веденого пристрою (або загальною адресою виклику, якщо вона увімкнена). Якщо знайдено збіг, генерується запит на переривання.

Біти 7..1 – TWA: Регістр адреси веденого пристрою TWI.

Ці сім біт утворюють адресу веденого пристрою у блоці TWI.

Біт 0 – TWGCE: Біт увімкнення розпізнавання загального виклику TWI.

Якщо встановлено, цей біт увімкне розпізнавання загального виклику, переданого через двопровідну послідовну шину.

Використання модулю TWI. TWI у мікроконтролерах AVR є байторієнтованим і працює на основі переривань. Переривання генеруються після всіх подій на шині, таких як отримання байта або передача умови START. Завдяки тому, що TWI працює на основі переривань, прикладне програмне забезпечення може виконувати інші операції під час передачі байта через TWI.

Зверніть увагу, що біт увімкнення переривань TWI (TWIE) у TWCR разом із бітом глобального увімкнення переривань у SREG дозволяють прикладному програмному забезпеченню визначати, чи повинно встановлення прапора TWINT спричиняти запит переривання. Якщо біт TWIE скинуто, прикладна програма має опитувати прапор TWINT, щоб визначити дії на шині TWI.

Коли прапор TWINT встановлено, TWI завершив операцію та очікує відповіді від прикладного програмного забезпечення. У цьому випадку регістр стану TWI (TWSR) містить значення, яке вказує поточний стан шини TWI. Потім прикладна програма може визначити, як TWI повинен поводитися в наступному циклі шини, змінюючи значення регістрів TWCR і TWDR.

На Рисунку 7.20 наведено простий приклад взаємодії прикладної програми з апаратною частиною TWI. У цьому прикладі головний пристрій бажає передати один байт даних веденому пристрою. Цей опис досить абстрактний, більш детальне пояснення наведено

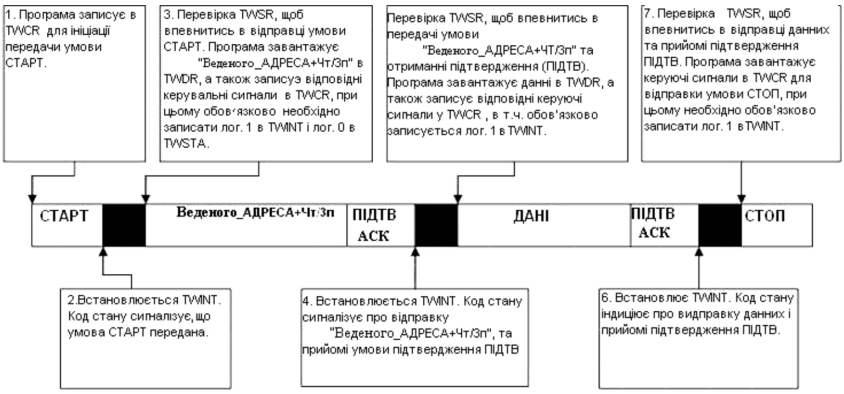


Рисунок 7.20 – Приклад підключення до шини TWI

далі в цьому розділі. Також представлений простий кодовий приклад, що реалізує бажану поведінку.

1. Першим кроком у передачі через TWI є передача умови START. Це виконується шляхом запису певного значення в TWCR, що наказує апаратному забезпеченню TWI передати умову START. Яке саме значення потрібно записати, буде описано далі. Однак важливо, щоб у записаному значенні було встановлено біт TWINT. Запис одиниці в TWINT очищає цей прапор. TWI не почне жодної операції, доки біт TWINT у TWCR встановлений. Одразу після того, як прикладна програма очистить TWINT, TWI розпочне передачу умови START.

2. Коли умова START буде передана, прапор TWINT у TWCR встановиться, а регістр стану TWSR оновиться кодом стану, що вказує на успішну передачу умови START.

3. Тепер прикладна програма повинна перевірити значення TWSR, щоб переконатися, що умову START було успішно передано. Якщо TWSR вказує інше, програма може виконати певні дії, наприклад, викликати процедуру обробки помилок.

Припустимо, що код стану відповідає очікуваному, тоді необхідно завантажити значення SLA+W у TWDR. Пам'ятайте, що TWDR використовується як для адреси, так і для даних. Після завантаження TWDR потрібним значенням SLA+W слід записати певне значення в TWCR, яке наказує апаратному забезпеченню TWI передати SLA+W, що міститься у TWDR. Яке саме значення слід записати, буде описано далі.

Однак важливо, щоб у записаному значенні було встановлено біт TWINT. Запис одиниці в TWINT очищає цей прапор. TWI не почне жодної операції, доки біт TWINT у TWCR встановлений.

Одразу після того, як прикладна програма очистить TWINT, TWI розпочне передачу пакета адреси.

4. Коли пакет адреси буде передано, прапор TWINT у TWCR встановиться, а регістр стану TWSR оновиться кодом стану, що вказує на успішну передачу пакета адреси. Код стану також покаже, чи ведений пристрій підтвердив отримання пакета.

5. Тепер прикладна програма повинна перевірити значення TWSR, щоб переконатися, що пакет адреси було успішно передано і що значення біта ACK відповідає очікуваному. Якщо TWSR вказує

інше, програма може виконати певні дії, наприклад, викликати процедуру обробки помилок.

Припустимо, що код стану відповідає очікуваному, тоді необхідно завантажити пакет даних у TWDR. Після цього слід записати певне значення в TWCR, яке наказує апаратному забезпеченню TWI передати пакет даних, що міститься у TWDR. Яке саме значення слід записати, буде описано далі.

Однак важливо, щоб у записаному значенні було встановлено біт TWINT. Запис одиниці в TWINT очищає цей прапор. TWI не почне жодної операції, доки біт TWINT у TWCR встановлений.

Одразу після того, як прикладна програма очистить TWINT, TWI розпочне передачу пакета даних.

6. Коли пакет даних буде передано, прапор TWINT у TWCR встановиться, а регістр стану TWSR оновиться кодом стану, що вказує на успішну передачу пакета даних. Код стану також покаже, чи ведений пристрій підтвердив отримання пакета.

7. Тепер прикладна програма повинна перевірити значення TWSR, щоб переконатися, що пакет даних було успішно передано і що значення біта ACK відповідає очікуваному. Якщо TWSR вказує інше, програма може виконати певні дії, наприклад, викликати процедуру обробки помилок.

Припустимо, що код стану відповідає очікуваному, тоді необхідно записати певне значення в TWCR, яке наказує апаратному забезпеченню TWI передати умову STOP. Яке саме значення слід записати, буде описано далі.

Однак важливо, щоб у записаному значенні було встановлено біт TWINT. Запис одиниці в TWINT очищає цей прапор. TWI не почне жодної операції, доки біт TWINT у TWCR встановлений.

Одразу після того, як прикладна програма очистить TWINT, TWI розпочне передачу умови STOP.

Зверніть увагу, що після передачі умови STOP прапор TWINT НЕ встановлюється.

Хоча цей приклад є простим, він демонструє основні принципи всіх передавань через TWI. Їх можна підсумувати так:

- Коли TWI завершує операцію і очікує відповіді від прикладної програми, прапор TWINT встановлюється. Лінія SCL утримується в низькому рівні, доки TWINT не буде очищено.

• Коли прапор TWINT встановлено, користувач повинен оновити всі регістри TWI відповідними значеннями для наступного циклу роботи шини TWI. Наприклад, у TWDR слід завантажити значення, яке буде передано в наступному циклі шини.

• Після оновлення всіх регістрів TWI та виконання інших очікуючих завдань прикладного програмного забезпечення слід записати значення в TWCR. При записі в TWCR біт TWINT повинен бути встановлений. Запис одиниці в TWINT очищає прапор. Після цього TWI розпочне виконання операції, заданої значенням TWCR.

Нижче, у табл. 7.8, наведено реалізацію цього прикладу на мові С. Зверніть увагу, що наведений код передбачає наявність певних визначень, наприклад, шляхом використання заголовкових файлів.

Таблиця 7.8 – Програма на С з коментарями

№	Програма на С	Коментар
1	2	3
1	TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN)	Надіслати умову START.
2	while (!(TWCR & (1<<TWINT)));	Очікувати встановлення прапора TWINT. Це вказує, що умова START була передана.
3	if ((TWSR & 0xF8) != START) ERROR();	Перевірити значення Регістра стану TWI. Замаскувати біти переддільника. Якщо стан відрізняється від START, перейти до ERROR.
	TWDR = SLA_W; TWCR = (1<<TWINT) | (1<<TWEN);	Завантажити SLA_W у регістр TWDR. Очистити біт TWINT у TWCR, щоб розпочати передавання адреси.
4	while (!(TWCR & (1<<TWINT)));	Очікувати встановлення біта TWINT. Це вказує на те, що SLA+W було передано, і отримано ACK/NACK.
5	if ((TWSR & 0xF8) != MT_SLA_ACK) ERROR();	Перевірити значення Регістра Стану TWI. Замаскувати біти переддільника. Якщо стан відрізняється від MT_SLA_ACK, перейти до ERROR.
	TWDR = DATA; TWCR = (1<<TWINT) | (1<<TWEN);	Завантажити DATA у регістр TWDR. Очистити біт TWINT у TWCR, щоб розпочати передавання даних.

Продовження таблиці 7.8

1	2	3
6	while (!(TWCR & (1<<TWINT)));	Чекати встановлення біта TWINT. Це вказує на те, що DATA було передано, і отримано ACK/NACK.
7	if ((TWSR & 0xF8) ! = MT_DATA_ACK) ERROR();	Перевірити значення реєстру стану TWI. Маскувати біти переддільника. Якщо статус відрізняється від MT_DATA_ACK, перейти до ERROR.
	TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);	Передати умову STOP.

Режими роботи. TWI може працювати в одному з чотирьох основних режимів: «Головний Передавач» (MT), «Головний Приймач» (MR), «Ведений Передавач» (ST) і «Ведений Приймач» (SR). У межах однієї програми можна використовувати кілька режимів. Наприклад, TWI може працювати в режимі MT для запису даних у TWI EEPROM, а в режимі MR – для зчитування даних із EEPROM. Якщо в системі є інші головні пристрої, деякі з них можуть передавати дані у TWI, у такому разі буде використовуватися режим SR. Вибір дозволених режимів визначається програмним забезпеченням.

Коли прапорець TWINT встановлено, код стану в TWSR використовується для визначення необхідних програмних дій. Для кожного коду стану у табл. 7.14, 7.19, 7.22, 7.25 наведено необхідні дії програмного забезпечення та деталі подальшого послідовного передавання. Зверніть увагу, що в цих таблицях біти переддільника замасковані як нуль.

Режим «Головний Передавач» (Master Transmitter Mode). У режимі «Головний Передавач» передається кілька байтів даних до Веденого Приймача.

Щоб увійти в режим Головного пристрою, необхідно передати умову START. Формат наступного пакета адреси визначає, який режим буде активовано:

• Якщо передається SLA+W, активується режим «Головний Передавач» (MT).

• Якщо передається SLA+R, активується режим «Головний Приймач» (MR).

Усі коди стану, згадані в цьому розділі, припускають, що біти переддільника мають значення нуль.

Умова START передається записом наступного значення в TWCR (табл. 7.9):

Таблиця 7.9 – Формування на шині умови START

TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
1	x	1	0	x	1	0	x
Скинути прапор TWINT		Сформувати умову START			Дозволити роботу TWI		

TWEN має бути встановлений для увімкнення інтерфейсу TWI, TWSTA має бути записаний у 1 для передавання START-умови, а TWINT має бути записаний у 1 для очищення прапора TWINT. Після цього TWI перевірить шину TWI і згенерує START-умову, щойно шина стане вільною. Після передавання START-умови прапор TWINT встановлюється апаратно, а код стану в TWSR буде 0x08.

Щоб увійти в режим «Головний Передавач», необхідно передати SLA+W. Це виконується шляхом запису SLA+W у TWDR. Після цього біт TWINT слід очистити (записавши в нього 1) для продовження передавання. Це виконується записом такого значення в TWCR (табл. 7.10):

Таблиця 7.10 – Значення, занесене у регістр TWCR

TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
1	x	0	0	x	1	0	x
Скинути прапор TWINT					Дозволити роботу TWI		

Коли SLA+W було передано і отримано біт підтвердження, TWINT встановлюється знову, і в TWSR можливі різні коди стану. У головному режимі можливі коди стану: 0x18, 0x20 або 0x38. Відповідні дії для кожного з цих кодів стану детально описані в табл. 7.14.

Коли SLA+W успішно передано, слід передати пакет даних. Це виконується шляхом запису байта даних у TWDR. TWDR можна записувати тільки тоді, коли TWINT встановлений. В іншому випадку доступ буде відхилено, а біт зіткнення запису (TWWC) буде встановлений у регістрі TWCR.

Після оновлення TWDR біт TWINT слід очистити (записавши в нього 1) для продовження передавання. Це виконується записом такого значення в TWCR:

Таблиця 7.11 – Значення, після якого почнеться передача пакету

TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
1	x	0	0	x	1	0	x
Скинути прапор TWINT					Дозволити роботу TWI		

Ця схема повторюється, доки не буде передано останній байт, після чого передавання завершується шляхом генерування умови STOP або повторної умови START. Умова STOP генерується записом такого значення в TWCR:

Таблиця 7.12 – Запис у регістр TWCR для формування STOP

TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
1	x	0	1	x	1	0	x
Скинути прапор TWINT			Сформувати умову STOP		Дозволити роботу TWI		

Умова REPEATED START генерується записом такого значення в TWCR:

Таблиця 7.13 – Запис у регістр TWCR для формування умови REPEATED START

TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
1	x	1	0	x	1	0	x
Скинути прапор TWINT		Сформувати умову START			Дозволити роботу TWI		

Після умови REPEATED START (стан 0x10) інтерфейс TWI може знову звернутися до того ж самого Slave або до нового Slave без передачі STOP умови. Умова REPEATED START дозволяє головному пристрою перемикатися між веденими пристроями, режимами головного передавача та головного приймача без втрати контролю над шиною.

Таблиця 7.14 – Коди статусу для режиму «Головний передавач»

Код статусу	**Стан шини і модуля TWI**	**Дії програми**					**Наступна дія, що виконується модулем TWI**
		в/із TWDR	**в регістр TWCR**				
			STA	**STO**	**TWINT**	**TWEA**	
1	2	3	4	5	6	7	8
$08	Умова START була передана.	Завантажити SLA + W	X	0	1	X	SLA+W буде передано; буде отримано ACK або NACK.
$10	Умова REPEATED START була передана.	Завантажити SLA + W	X	0	1	X	SLA+W буде передано; буде отримано ACK або NACK.
		Завантажити SLA + R	X	0	1	X	SLA+R буде передано; логіка переключиться на режим «Головний Приймач».
$18	SLA+W було передано; було отримано ACK.	Завантажити дані	0	0	1	X	Дані будуть передані, і буде отримано ACK або NACK.
		Немає дій	1	0	1	X	Буде передано REPEATED START.
		Немає дій	0	1	1	X	Буде передано умову STOP, і прапор TWSTO буде скинуто.
		Немає дій	1	1	1	X	Буде передано умову STOP, за якою слідує умова START, і прапор TWSTO буде скинуто.
$20	SLA+W було передано; було отримано NACK.	Аналогічно діям для коду статусу $18					

Продовження таблиці 7.14

1	2	3	4	5	6	7	8
$28	Байт даних був переданий; було отримано ACK.	Аналогічно діям для коду статусу $18					
$30	Байт даних був переданий; було отримано NACK.	Аналогічно діям для коду статусу $18					
$38	Втрата арбітражу під час передачі SLA+W або байтів даних.	Немає дій	0	0	1	X	Шина буде звільнена, і пристрій перейде в неадресований режим Slave.
		Немає дій	1	0	1	X	Умова START буде передана, коли шина стане вільною.

Режим «Головний Приймач»

У режимі «Головний Приймач» приймається кілька байтів даних від Веденого Передавача. Для входу в режим головного пристрою необхідно передати умову START. Формат наступного адресного пакету визначає, чи буде активований режим «Головний Передавач» чи «Головний Приймач». Якщо передається SLA+W, то активується режим «Головний Передавач», якщо SLA+R – режим «Головний Приймач». Всі статусні коди, згадані в цьому розділі, припускають, що біти переддільника рівні нулю або маскуються як нулі.

Умова START передається шляхом запису наступного значення в TWCR:

Таблиця 7.15 – Запису у регістр TWCR для формування умови START

TWINT	**TWEA**	**TWSTA**	**TWSTO**	**TWWC**	**TWEN**	**-**	**TWIE**
1	x	1	0	x	1	0	x
Скинути прапор TWINT		Сформувати умову START			Дозволити роботу TWI		

TWEN потрібно записати в одиницю для увімкнення інтерфейсу TWI, TWSTA потрібно записати в одиницю для передачі

умови START, а TWINT потрібно встановити в одиницю для очищення прапорця TWINT. Тоді TWI перевірить шину TWI і згенерує умову START, як тільки шина стане вільною. Після передачі умови START, прапорець TWINT буде встановлений апаратно, а код стану в TWSR буде 0x08 (див. табл. 7.19). Для того щоб увійти в режим «Головний Приймач», потрібно передати SLA+R. Це робиться шляхом запису SLA+R в TWDR. Після цього прапорець TWINT слід очистити (записавши в нього одиницю), щоб продовжити передачу. Це досягається записом наступного значення в TWCR:

Таблиця 7.16 – Занесення у регістр TWCR для перемикання модуля в режим «Головний Приймач»

TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
1	x	0	0	x	1	0	x
Скинути прапор TWINT					Дозволити роботу TWI		

Коли SLA+R було передано і отримано підтвердження, прапорець TWINT знову встановлюється, і в TWSR можуть бути можливі різні коди стану. Можливі коди стану в режимі Головного Приймача (MR) – 0x38, 0x40 або 0x48. Відповідні дії для кожного з цих кодів стану детально описані в таблиці 7.19. Отримані дані можна прочитати з реєстру TWDR, коли прапорець TWINT встановлений апаратно в одиницю. Ця схема повторюється, поки не буде отримано останній байт. Після отримання останнього байта Головний Приймач (MR) повинен сповістити Ведений Передавач (ST), надіславши NACK після останнього отриманого байта. Передачу завершує генерування умови STOP або повторної умови START. Умова STOP генерується шляхом запису наступного значення в TWCR:

Таблиця 7.17 – Запис у регістр TWCR для сформування на шині умови STOP

TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
1	x	0	1	x	1	0	x
Скинути прапор TWINT			Сформувати стан СТОП		Дозволити роботу I^2C		

Умова REPEATED START генерується шляхом запису наступного значення в TWCR:

Таблиця 7.18 – Запис у регістр TWCR для сформування на шині умови REPEATED START

TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
1	x	1	0	x	1	0	x
Скинути прапор TWINT		Сформувати стан СТАРТ			Дозволити роботу I^2C		

Після умови REPEATED START (стан 0x10) Головний пристрій може знову звернутися до того ж самого Веденого або до нового Веденого без передачі умови STOP. Повторний START дозволяє Головному змінювати Ведених, перемикатися між режимами Головного Передавача і Головного Приймача без втрати контролю над шиною.

Коди статусу для режиму «Головний Приймач» наведено в таблиці 7.19.

Таблиця 7.19 – Коди статусу для режиму «Головний Приймач»

Код статусу	Стан шини і модуля TWI	Дії програми					Наступна дія, що виконується модулем TWI
		в/із TWDR	в регістр TWCR				
			STA	STO	TWINT	TWEA	
1	2	3	4	5	6	7	8
$08	Умова START була передана.	Завантажити SLA + R	X	0	1	X	SLA+R буде передано; підтвердження (ACK) або непідтвердження (NACK) будуть отримані.
$10	Умова REPEATED START була передана.	Завантажити SLA + R	X	0	1	X	SLA+R буде передано; підтвердження (ACK) або непідтвердження (NACK) будуть отримані.
		Завантажити SLA + W	X	0	1	X	SLA+W буде передано; логіка перемикатиметься в режим «Головний Передавач».

Продовження таблиці 7.19

1	2	3	4	5	6	7	8
$38	Втрата арбітражу при передачі SLA+R або біт NACK.	Нема дій	0	0	1	X	Шина TWI буде звільнена, а пристрій перейде в режим неадресованого Веденого.
		Нема дій	1	0	1	X	Умова START буде передана, коли шина стане вільною.
$40	SLA+R було передано; отримано підтвердження (ACK).	Нема дій	0	0	1	0	Байт даних буде отримано, і повернеться NACK.
		Нема дій	0	0	1	1	Байт даних буде отримано, і повернеться ACK.
$48	SLA+R було передано; отримано непідтвердження (NACK).	Нема дій	1	0	1	X	Буде передана умова REPEATED START.
		Нема дій	0	1	1	X	Буде передана умова STOP, і прапор TWSTO буде скинутий.
		Нема дій	1	1	1	X	Буде передана умова STOP, за якою слідує умова START, і прапор TWSTO буде скинутий.
$50	Байт даних отримано; повернено підтвердження (ACK).	Прочитати дані	0	0	1	0	Буде отриманий байт даних, і буде повернено NACK.
		Прочитати дані	1	0	1	1	Буде отриманий байт даних, і буде повернено ACK.
$58	Байт даних отримано; повернено непідтвердження (NACK).	Прочитати дані	1	0	1	X	Буде передана умова REPEATED START.
		Прочитати дані	0	1	1	X	Буде передана умова STOP, і прапор TWSTO буде скинутий.
		Прочитати дані	1	1	1	X	Буде передана умова STOP, за якою слідує умова START, і прапор TWSTO буде скинутий.

Режим «Ведений Приймач»

У режимі Ведений Приймач кілька байтів даних отримуються від Головного Передавача. Усі статусні коди, згадані в цьому розділі, передбачають, що біти переддільника рівні нулю або маскуються як нулі.

Для ініціалізації режиму Веденого Приймача необхідно ініціалізувати TWAR та TWCR наступним чином:

Таблиця 7.20 – Запис у регістр TWAR

TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
Власна ведена адреса пристрою (Slave Address).							X

Старші 7 біт – це адреса, на яку інтерфейс двопровідної серійної передачі відповідатиме, коли буде адресований Головним. Якщо найменший значущий біт (LSB) встановлений, TWI відповідатиме на загальний виклик (адреса 0x00), в іншому випадку він ігноруватиме загальний виклик.

Таблиця 7.21 – Запис у регістр TWCR

TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
0	1	0	0	x	1	0	x
	Дозволити підтвердження				Дозволити роботу TWI		

TWEN має бути записаний в одиницю, щоб увімкнути TWI. Біт TWEA має бути записаний в одиницю для увімкнення підтвердження власної адреси пристрою або адреси загального виклику. Біти TWSTA і TWSTO мають бути записані в нуль.

Коли TWAR і TWCR ініціалізовані, TWI чекає, поки не буде адресовано його власною адресою пристрою (або адресою загального виклику, якщо це дозволено) разом із бітом напрямку передачі даних. Якщо біт напрямку «0» (запис), TWI працює в режимі «Ведений Приймач», в іншому випадку він переходить у режим «Ведений Передавач». Після отримання власної адреси пристрою та біта запису, флаг TWINT буде встановлений, і можна буде прочитати

дійсний код стану з TWSR. Код стану використовується для визначення відповідної дії програмного забезпечення. Деталі необхідних дій для кожного з цих кодів стану наведені в таблиці 7.22.

Таблиця 7.22 – Коди статусу для режиму «Ведений приймач»

Код статусу	Стан шини і модуля TWI	Дії програми					Наступна дія, що виконується модулем TWI
		в/із TWDR	в регістр TWCR				
			STA	STO	TWINT	TWEA	
1	2	3	4	5	6	7	8
$60	Отримано власну SLA+W; повернуто ACK	Немає дій	X	0	1	0	Байт даних буде отримано, і буде надіслано непідтвердження (NACK).
		Немає дій	X	0	1	1	Байт даних буде отримано, і буде надіслано підтвердження (ACK).
$68	Втрачено арбітраж у SLA+R/W в режимі Головного пристрою; отримано власну SLA+W; повернуто ACK	Немає дій	X	0	1	0	Байт даних буде отримано, і буде надіслано непідтвердження (NACK).
		Немає дій	X	0	1	1	Байт даних буде отримано, і буде надіслано підтвердження (ACK).
$70	Отримано адресу загального виклику; повернуто ACK	Немає дій	X	0	1	0	Байт даних буде отримано, і буде надіслано непідтвердження (NACK).
		Немає дій	X	0	1	1	Байт даних буде отримано, і буде надіслано підтвердження (ACK).
$78	Втрачено арбітраж в SLA+R/W в режимі Головного пристрою; отримано адресу загального виклику; повернуто ACK	Немає дій	X	0	1	0	Байт даних буде отримано, і буде надіслано непідтвердження (NACK).
		Немає дій	X	0	1	1	Байт даних буде отримано, і буде надіслано підтвердження (ACK).

Продовження таблиці 7.22

1	2	3	4	5	6	7	8
$80	Раніше адресовано власним SLA+W; байт даних отримано; повернуто ACK	Прочитати дані	X	0	1	0	Байт даних буде отримано, і буде надіслано непідтвердження (NACK).
		Прочитати дані	X	0	1	1	Байт даних буде отримано, і буде надіслано підтвердження (ACK).
$88	Раніше адресовано власним SLA+W; байт дані отримано; повернуто NACK	Прочитати дані	0	0	1	0	Перехід до режиму «неадресованого веденого пристрою»; немає розпізнавання власної SLA або адреси загального виклику.
$88	Раніше адресовано власним SLA+W; байт дані отримано; повернуто NACK	Прочитати дані	0	0	1	1	Перехід до режиму «неадресованого веденого пристрою»; власна SLA буде розпізнана; адресу загального виклику буде розпізнано, якщо TWGCE = «1».
		Прочитати дані	1	0	1	0	Перехід до режиму «неадресованого веденого пристрою»; немає розпізнавання власної SLA або адреси загального виклику; умова START буде передана, коли шина стане вільною.
		Прочитати дані	1	0	1	1	Перехід до режиму «неадресованого веденого пристрою»; власна SLA буде розпізнана; адресу загального виклику буде розпізнано, якщо TWGCE = «1»; умова START буде передана, коли шина стане вільною.

Продовження таблиці 7.22

$90	Раніше адресовано загальним викликом; дані отримано; повернуто ACK	Прочитати дані	X	0	1	0	Байт даних буде отримано, і буде надіслано непідтвердження (NACK).
		Прочитати дані	X	0	1	1	Байт даних буде отримано, і буде надіслано підтвердження (ACK).
$98	Раніше адресовано загальним викликом; дані отримано; повернуто ACK	Прочитати дані	0	0	1	0	Перехід до режиму «неадресованого веденого пристрою»; немає розпізнавання власної SLA або адреси загального виклику.
		Прочитати дані	0	0	1	1	Перехід до режиму «неадресованого веденого пристрою»; власна SLA буде розпізнана; адресу загального виклику буде розпізнано, якщо TWGCE=«1».
	Раніше адресовано загальним викликом; дані отримано; повернуто ACK	Прочитати дані	1	0	1	0	Перехід до режиму «неадресованого веденого пристрою»; немає розпізнавання власної SLA або адреси загального виклику; умова START буде передана, коли шина стане вільною.
		Прочитати дані	1	0	1	1	Перехід до режиму «неадресованого веденого пристрою»; власна SLA буде розпізнана; адресу загального виклику буде розпізнано, якщо TWGCE=«1»; умова START буде передана, коли шина стане вільною.

Продовження таблиці 7.22

$A0	Отримано умову STOP або REPEATED START, поки пристрій ще був адресований як Ведений	Прочитати дані	0	0	1	0	Перехід до режиму «неадресованого веденого пристрою»; немає розпізнавання власної SLA або адреси загального виклику.
		Прочитати дані	0	0	1	1	Перехід до режиму «неадресованого веденого пристрою»; власна SLA буде розпізнана; адресу загального виклику буде розпізнано, якщо TWGCE = «1».
		Прочитати дані	1	0	1	0	Перехід до режиму «неадресованого веденого пристрою»; немає розпізнавання власної SLA або адреси загального виклику; умова START буде передана, коли шина стане вільною.
		Прочитати дані	1	0	1	1	Перехід до режиму «неадресованого веденого пристрою»; власна SLA буде розпізнана; адресу загального виклику буде розпізнано, якщо TWGCE = «1»; умова START буде передана, коли шина стане вільною.

Режим «Ведений Приймач» також може бути ініційований, якщо відбулося втрата арбітражу під час роботи TWI в режимі головного пристрою (стани 0x68 і 0x78).

Якщо під час передачі біт TWEA скидається, TWI поверне «Непідтвердження» («1») на лінії SDA після отриманого наступного байту даних. Це може використовуватися для вказівки, що ведений пристрій більше не може приймати байти. Поки TWEA дорівнює

нулю, TWI не підтверджує свою власну адресу. Проте, інтерфейс шини TWI все ще моніториться, і розпізнавання адреси може поновитися в будь-який час після встановлення TWEA. Це означає, що біт TWEA може використовуватися для тимчасового ізолювання TWI від шини.

У всіх режимах сну, окрім режиму Idle, тактова система для TWI вимикається. Якщо біт TWEA встановлений, інтерфейс може все одно підтвердити свою власну адресу пристрою або адресу загального виклику, використовуючи тактовий сигнал шини TWI як джерело тактового сигналу. Пристрій тоді вийде зі сну, і TWI утримає лінію SCL в низькому стані під час пробудження і до того часу, поки флаг TWINT не буде очищений (записавши одиницю). Далі отримання даних буде здійснюватися як звичайно, з нормальним робочим станом тактових сигналів AVR. Зверніть увагу, що якщо AVR налаштований на довгий час запуску, лінія SCL може бути утримувана в низькому стані протягом тривалого часу, блокуючи інші передачі даних.

Зауважте, що реєстр даних інтерфейсу Two-wire Serial Interface – TWDR не відображає останній байт на шині при пробудженні з цих режимів сну.

Коди статусу для режиму «Ведений приймач» наведено в таблиці 7.23.

Режим «Ведений Передавач»

У режимі «Ведений Передавач» передається кілька байтів даних до Головного Приймача. Усі згадані статусні коди (табл. 7.24) припускають, що біти переддільника дорівнюють нулю або вони замасковані як нулі.

Щоб ініціалізувати режим Веденого Передавача, потрібно налаштувати TWAR і TWCR наступним чином:

Таблиця 7.23 – Запис у регістр TWAR

TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
Власна ведена адреса пристрою (Slave Address).							X

Сім старших біт – це адреса, на яку модуль TWI буде реагувати, коли буде адресований Головним пристроєм. Якщо наймолодший біт

увімкнений, TWI буде реагувати на адресу загального виклику (0x00), в іншому випадку він ігноруватиме адресу загального виклику.

Таблиця 7.24 – Запис у регістр TWCR для передачі даних головному

TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
0	1	0	0	x	1	0	x
	Дозволити підтвердження				Дозволити роботу TWI		

TWEN повинен бути записаний в одиницю, щоб увімкнути TWI. Біт TWEA повинен бути записаний в одиницю, щоб увімкнути підтвердження власної адреси пристрою або адреси загального виклику. Біти TWSTA і TWSTO повинні бути записані в нуль.

Після ініціалізації TWAR і TWCR, TWI чекає, поки не буде адресований своєю власною адресою (або адресою загального виклику, якщо це увімкнено), після чого передається біт напрямку даних. Якщо біт напрямку дорівнює «1» (читання), TWI працюватиме в режимі «Ведений Передавач», в іншому випадку буде увійдено в режим «Ведений Приймач». Після отримання власної адреси пристрою і біту запису, біт TWINT встановлюється, і можна прочитати код статусу з TWSR. Код статусу використовується для визначення відповідної дії програмного забезпечення. Відповідну дію для кожного коду статусу можна побачити в таблиці 7.26. Режим «Ведений Передавач» також може бути активований, якщо під час роботи в режимі Головного пристрою сталася втрата арбітражу (див. стан 0xB0).

Якщо біт TWEA буде записаний в нуль під час передачі, TWI передасть останній байт передачі. В залежності від того, чи передасть Головний Приймач NACK або ACK після останнього байта, буде введено стан 0xC0 або 0xC8. TWI буде переведено в режим неадресованого Slave, і він буде ігнорувати Головний пристрій, якщо той продовжить передачу. Таким чином, Головний Приймач отримає всі «1» як серійну передачу. Стан 0xC8 буде введено, якщо Головний пристрій вимагатиме додаткові байти даних (передаючи ACK), навіть якщо Ведений вже передав останній байт (TWEA нуль і очікується NACK від Головного пристрою).

Поки TWEA дорівнює нулю, TWI не реагує на свою власну адресу пристрою. Однак шина TWI все ще моніториться, і розпізнавання адреси може бути відновлене в будь-який момент, якщо TWEA буде знову увімкнено. Це означає, що біт TWEA може бути використаний для тимчасової ізоляції TWI від шини.

У всіх режимах сну, крім режиму Idle, система тактування для TWI вимкнена. Якщо біт TWEA встановлений, інтерфейс все ще може підтверджувати свою власну адресу пристрою або адресу загального виклику, використовуючи тактовий сигнал TWI як джерело такту. Пристрій прокидається з режиму сну, і TWI утримує лінію SCL на низькому рівні під час пробудження і до тих пір, поки біт TWINT не буде очищений (записом в один). Подальша передача даних відбуватиметься як зазвичай, з нормальним тактуванням AVR. Зверніть увагу, що якщо AVR налаштовано на довгий час запуску, лінія SCL може залишатися на низькому рівні довгий час, блокуючи інші передачі даних.

Зазначте, що при прокиданні з цих режимів сну, Регістр даних Two-wire Serial Interface – TWDR не відображає останній байт, присутній на шині.

Коди статусу для режиму «Ведений передавач» наведено в таблиці 7.25.

Таблиця 7.25 – Коди статусу для режиму «Ведений передавач»

Код статусу	**Стан шини і модуля TWI**	**Дії програми**					**Наступна дія, що виконується модулем TWI**
		в/із TWDR	**в регістр TWCR**				
			STA	**STO**	**TWINT**	**TWEA**	
1	2	3	4	5	6	7	8
$A8	Отримано власну SLA+R; повернуто ACK.	Завантажити дані	X	0	1	0	Останній байт даних буде передано, і має бути отримано NACK.
		Завантажити дані	X	0	1	1	Байт даних буде передано, і має бути отримано ACK.

Продовження таблиці 7.25

1	2	3	4	5	6	7	8
$B0	Втрата арбітражу в SLA+R/W в режимі Головного пристрою; отримано власну SLA+R; повернуто ACK.	Завантажити дані	X	0	1	0	Останній байт даних буде передано, і має бути отримано NACK.
		Завантажити дані	X	0	1	1	Байт даних буде передано, і має бути отримано ACK.
$B8	Байт даних у TWDR було передано; отримано ACK.	Завантажити дані	X	0	1	0	Останній байт даних буде передано, і має бути отримано NACK.
		Завантажити дані	X	0	1	1	Байт даних буде передано, і має бути отримано ACK.
$C0	Байт даних у TWDR було передано; отримано NACK.	Немає дій	0	0	1	0	Переключено в режим неадресованого веденого пристрою; не буде розпізнано ані свій SLA, ані адреса загального виклику.
		Немає дій	0	0	1	1	Переключено в режим неадресованого веденого пристрою; буде розпізнано свій SLA; адресу загального виклику буде розпізнано, якщо TWGCE=«1».
$C0	Був переданий байт даних і отримано непідтвердження (NACK).	Немає дій	1	0	1	0	Переключено в режим неадресованого веденого пристрою; не буде розпізнано ані свій SLA, ані адреса загального виклику; умова START буде передана, коли шина стане вільною.

Продовження таблиці 7.25

1	2	3	4	5	6	7	8
		Немає дій	1	0	1	1	Переключено в режим неадресованого веденого пристрою; буде розпізнано свій SLA; адресу загального виклику буде розпізнано, якщо TWGCE=«1»; умова START буде передана, коли шина стане вільною.
$C8	Останній байт даних у TWDR було передано (TWEA=«0»); отримано ACK.	Те саме, що й для коду $C0					

Різні стани

Є два кодів стану, які не відповідають визначеному стану TWI, див. табл. 7.26.

Статус 0xF8 вказує, що немає доступної релевантної інформації, оскільки прапор TWINT не встановлений. Це трапляється між іншими станами, коли TWI не бере участі в серійному передаванні. Статус 0x00 вказує на помилку шини, що сталася під час передачі по шині TWI. Помилка шини виникає, коли умова START або STOP відбувається в недозволеній позиції у форматі кадру. Прикладами таких недозволених позицій є під час серійної передачі байту адреси, байту даних або біту підтвердження. Коли відбувається помилка шини, біт TWINT встановлюється. Для відновлення після помилки шини необхідно встановити прапор TWSTO і очистити TWINT, записавши логічну одиницю в нього. Це призводить до того, що TWI переходить в режим неадресованого веденого пристрою і очищає прапор TWSTO (інші біти в TWCR не змінюються). Лінії SDA і SCL звільняються, і умова STOP не передається.

Таблиця 7.26 – Різні коди статусів

Код статусу	Стан шини і модуля TWI	Дії програми					Наступна дія, що виконується модулем TWI
		в/із TWDR	в регістр TWCR				
			STA	STO	TWINT	TWEA	
$F8	Немає доступної релевантної інформації про стан; TWINT = «0»	Немає дій	Немає дій				Очікувати або продовжити поточну передачу.
0x00	Помилка шини через недозволену умову START або STOP	Немає дій	0	1	1	X	Тільки внутрішнє обладнання зазнає впливу, жодна умова STOP не відправляється на шину. У всіх випадках шина звільняється, а TWSTO очищається.

Контрольні запитання до теми 7

1. Які основні компоненти складають інтерфейс SPI і як вони взаємодіють?
2. Яким чином визначається напрямок передачі даних для кожного з чотирьох сигнальних виводів SPI?
3. Що таке сигнал NSS (Slave Select) і як він використовується у протоколі SPI?
4. Які особливості передачі даних у режимі повнодуплексної передачі SPI?
5. Як головний і ведений режими SPI впливають на організацію обміну даними між пристроями?
6. Які основні функції виконує регістр управління SPI (SPCR) у мікроконтролері ATmega8?
7. Що визначає біт DORD у регістрі управління SPI (SPCR)? Які можливі значення цього біту і як вони впливають на порядок передачі даних?
8. Яким чином біти SPR1 і SPR0 у регістрі управління SPI (SPCR) впливають на швидкість передачі даних через інтерфейс SPI у режимі Master?
9. Що означає біт SPIF у регістрі статусу SPI (SPSR)? Як він використовується для управління передачею даних через SPI?
10. Як відрізнити режими передачі даних SPI за допомогою бітів CPOL і CPHA у мікроконтролері ATmega8?

11. Які основні компоненти входять до складу загальних рішень для систем, які використовують шину I2C?
12. Які переваги має шина I2C порівняно з іншими інтерфейсами зв'язку?
13. Яким чином здійснюється адресація пристроїв на шині I2C?
14. Як вирішується проблема колізій на шині I2C? Яка процедура використовується для цього?
15. Які особливості формату байту в передачі даних по шині I2C?
16. Які основні переваги використання двопровідного послідовного інтерфейсу TWI у мікроконтролерах AVR?
17. Які характеристики і особливості апаратного модуля TWI дозволяють забезпечити стабільну роботу шини в системах з великою кількістю підключених пристроїв?
18. Яким чином блок керування TWI виявляє події на шині і забезпечує обробку відповідних запитів від програми?
19. Які режими швидкості підтримує модуль TWI, і як вони регулюються в програмовому середовищі?
20. Які функціональні блоки складають модуль TWI, і яку роль виконує кожен з них в процесі передачі даних через шину?
21. Які основні характеристики двопровідного інтерфейсу TWI в мікроконтролерах AVR?
22. Які компоненти входять до складу модуля TWI мікроконтролера?
23. Як функціонує блок виявлення адреси в модулі TWI?
24. Як визначається частота синхронізації SCL у модулі TWI?
25. Які режими передачі підтримуються в модулі TWI?

Використана література

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. 57 с.
2. Atmel: ATMega8, ATmega8L : технічна документація на мікроконтролер. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2486-8-bit-AVR-microcontroller-ATmega8_L_datasheet.pdf
3. SPI Demystified: Understanding the Basics and Beyond. URL: https://www.circuitbread.com/tutorials/spi-demystified-understanding-the--basics-and-beyond
4. What is the I2C Communication Protocol? URL: https://www.circuitbread.com/tutorials/what-is-the-i2c-communication-protocol

8 АНАЛОГОВІ МОДУЛІ МІКРОКОНТРОЛЕРІВ СІМЕЙСТВА AVR

Метою вивчення теми є ознайомлення з аналоговими модулями мікроконтролерів сімейства AVR.

Завдання вивчення теми збігаються з переліком питань для розгляду, що наведений нижче.

Перелік питань до розділу:

8.1. Аналоговий компаратор.

8.2. Загальні відомості про АЦП.

8.3. Модуль АЦП мікроконтролерів AVR.

8.1 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-лічильника 1. 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 8.1.

Регістри аналогового компаратору мікроконтролера ATMega8.

Регістр спеціальних функцій вводу-виводу – SFIOR (рис. 8.2).

Біт 3 – ACME: увімкнення мультиплексора аналогового компаратора

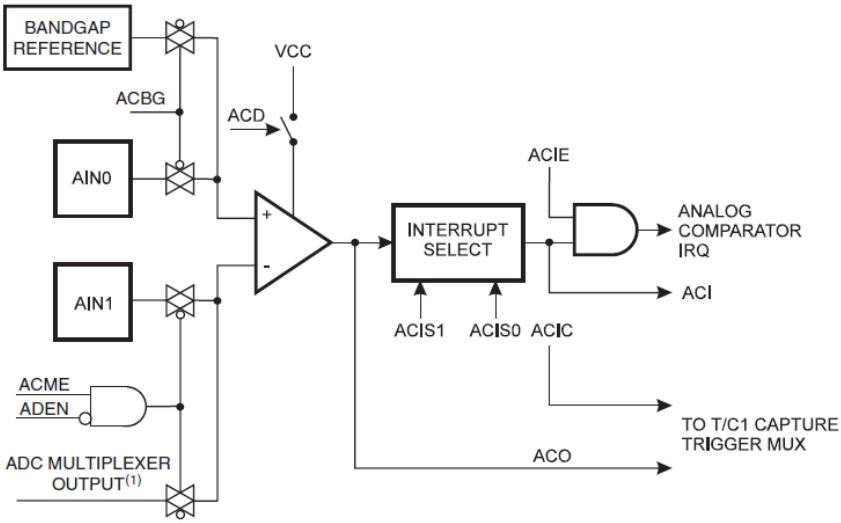


Рисунок 8.1 – Блок-схема компаратора та його оточуюча логіка

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 8.2 – Регістр спеціальних функцій вводу-виводу – SFIOR

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. 8.3).

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.

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 8.3 – Регістр управління та стану аналогового компаратора – ACSR

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 вектору 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 придушувач 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 8.1.

Таблиця 8.1 – 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 мультиплексора 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 8.2. Якщо ACME скинуто або встановлено ADEN, вхід AIN1 використовується як негативний вхід аналогового компаратора.

Таблиця 8.2 – 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

8.2 Загальні відомості про АЦП

АЦП (analog-to-digital converter, ADC або A/D) дає еквівалентне представлення аналогового сигналу у цифровому (двійковому) коді. Це дає можливість МК працювати з аналоговими пристроями, наприклад, знімати покази з різноманітних давачів, що мають аналоговий вихід, виконувати контроль за рівнем напруги живлення, контролювати робочий струм виконавчих пристроїв тощо.

Основними параметрами будь-якого АЦП є розрядність та швидкодія, тобто максимальна частота дискретизації (вибірок за сек.). Розрядність АЦП визначає кількість рівнів квантування, якими перетворювач може представити аналоговий сигнал. 8-ми розрядний АЦП забезпечує $2^8 = 256$ рівнів, тобто діапазон значень на виході перетворювача 0…255; 10-розрядний – $10^8 = 1024$ рівнів (значення від 0 до 1023). Якщо розрядність АЦП становить 10 біт, діапазон вхідної напруги від 0 до 5 В, тоді розрядність за напругою:(5–0) / 1024 = ~4,88 мВ.

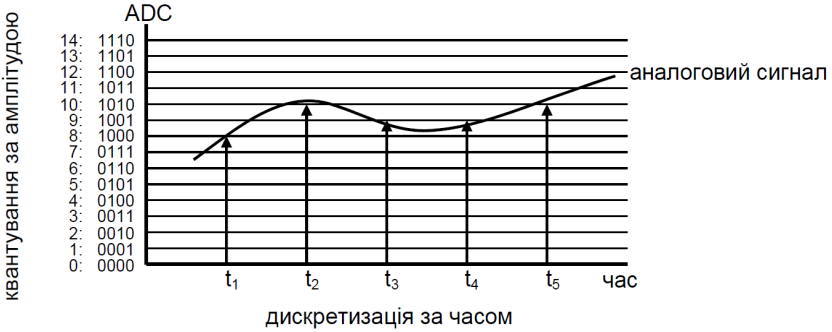


Рисунок 8.4 – Аналоговий сигнал за часом та амплітудою квантування

Частота дискретизації визначає, як часто ми можемо здійснювати вибірки цифрових значень з аналогового сигналу.

Швидкодія АЦП та його розрядність визначаються типом архітектури. Наприклад, паралельні АЦП можуть мати розрядність 8–12 біт та частоту дискретизації від 10 МГц до понад 1 ГГц; АЦП послідовного наближення – 10–16 біт та швидкодію від 100кГц до понад 1 МГц; сігма-дельта АЦП – 16–24 біт та швидкодію від 1 до 100 кГц; інтегруючі АЦП – 16–24 біт та швидкодію від 10 до 200 Гц. Розрядність та швидкодія взаємопов'язані параметри: більша розрядність – нижча швидкодія, і навпаки.

Деякі МК AVR мають інтегровані модулі АЦП послідовного наближення. Спрощена схема АЦП послідовних наближень

представлена на рисунку 8.5. Перетворювач складається з ЦАП (цифро-аналогового перетворювача), регістра послідовних наближень, компаратора та блоку синхронізації. У початковому стані у всіх розрядах (D7, … , D0) регістра послідовних наближень встановлені значення «0». Цикл вимірювання починається з того, що старший розряд D7 регістру встановлюється в «1». Після цього за допомогою компаратора порівнюється напруга на виході ЦАП та на вході АЦП. Якщо напруга на вході виявляється більшою, тоді значення логічного сигналу у розряді D7 зберігається рівним «1». У іншому випадку, старший розряд регістра обнулюється. Після цього розряд D6 встановлюється в «1», та знову проводиться порівняння напруг. Цей цикл операцій послідовно застосовується до всіх решта розрядів регістра. Робота схеми синхронізується за допомогою сигналів початку та закінчення перетворення.

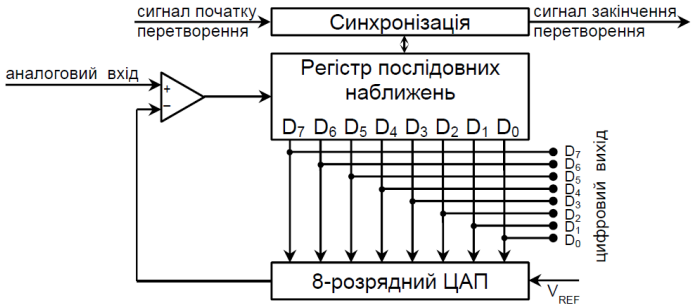


Рисунок 8.5 – Спрощена схема АЦП послідовних наближень

8.3 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 8.6.

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.

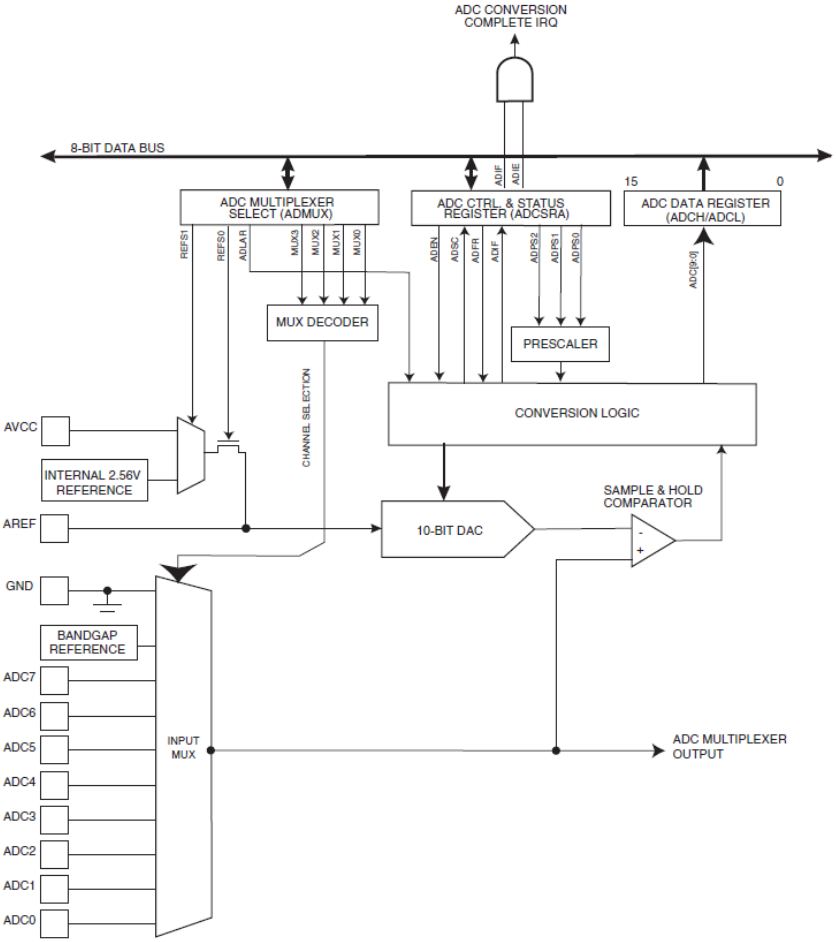


Рисунок 8.6 – Блок-схема АЦП

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. 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.

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.

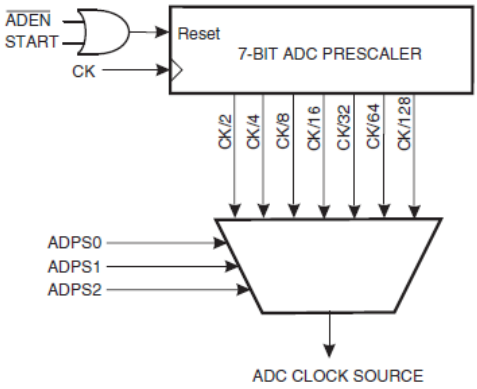


Рисунок 8.7 – Попереднє поділення АЦП

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 8.8. 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 конденсатор 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.

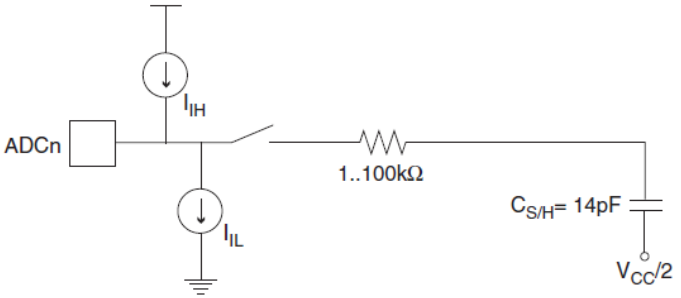


Рисунок 8.8 – Аналогова вхідна схема

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.

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.

Показники точності АЦП. n-розрядний АЦП лінійно перетворює напругу між GND і VREF на 2^n кроків (LSB). Найнижчий код читається як 0, а найвищий код читається як 2^{n-1}. Декілька параметрів описують відхилення від ідеальної поведінки.

Зміщення (рисунок 8.9)

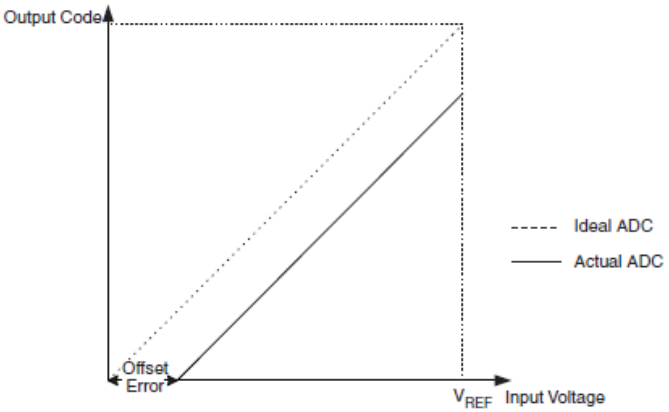


Рисунок 8.9 – Помилка зміщення

Це відхилення першого переходу (від 0x000 до 0x001) порівняно з ідеальним переходом (при 0,5 LSB). Ідеальне значення: 0 LSB.

Помилка посилення (рисунок 8.10)

Після коригування зсуву помилка посилення визначається як відхилення останнього переходу (0x3FE до 0x3FF) порівняно з ідеальним переходом (на 1,5 LSB нижче максимального). Ідеальне значення: 0 LSB.

Інтегральна нелінійність (INL) (рисунок 8.11)

Після коригування зсуву та похибки посилення INL є максимальним відхиленням фактичного переходу порівняно з ідеальним переходом для будь-якого коду. Ідеальне значення: 0 LSB.

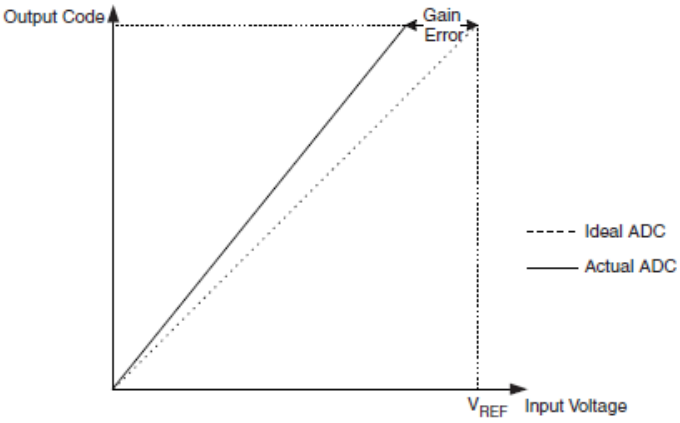


Рисунок 8.10 – Помилка посилення

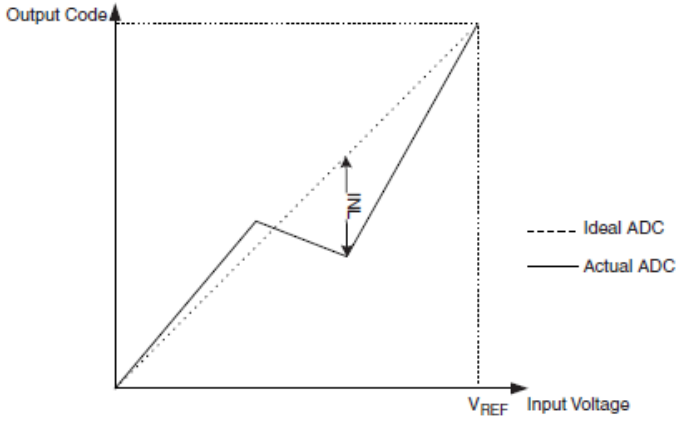


Рисунок 8.11 – Інтегральна нелінійність

Диференціальна нелінійність (DNL) (рисунок 8.12)

Максимальне відхилення фактичної ширини коду (інтервал між двома сусідніми переходами) від ідеальної ширини коду (1 LSB). Ідеальне значення: 0 LSB.

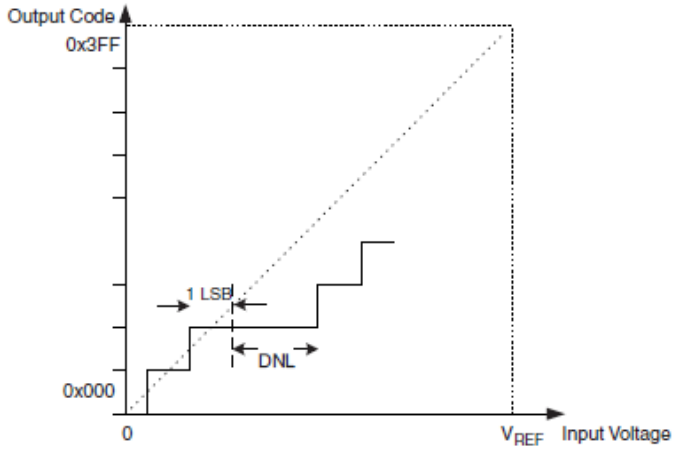


Рисунок 8.12 – Диференціальна нелінійність

Помилка квантування

Через квантування вхідної напруги в скінченну кількість кодів діапазон вхідних напруг (шириною 1 LSB) кодуватиме однакове значення. Завжди ± 0,5 LSB.

Абсолютна точність

Максимальне відхилення фактичного (нескоригованого) переходу порівняно з ідеальним переходом для будь-якого коду. Це сукупний ефект зміщення, помилки посилення, диференціальної помилки, нелінійності та помилки квантування. Ідеальне значення: ±0,5 LSB.

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

Результат визначається за наступною формулою:

$$ADC = \frac{V_{IN} \cdot 1023}{V_{REF}}, \tag{8.1}$$

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. 8.13):

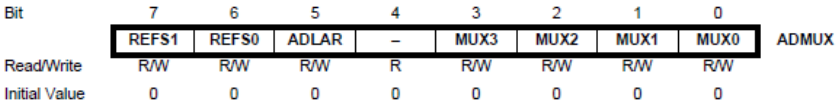


Рисунок 8.13 – Регістр мультиплексора АЦП – ADMUX

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 у 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 8.3 – 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 8.4). 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 8.4 – 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. 8.14):

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 8.14 – 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. 8.5).

Таблиця 8.5 – Біти вибору попереднього дільника АЦП

ADPS2	ADPS1	ADPS0	Коефіцієнт ділення
1	2	3	4
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8

Продовження таблиці 8.5

1	2	3	4
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Регістр даних АЦП – ADCL і ADCH (рис. 8.15).

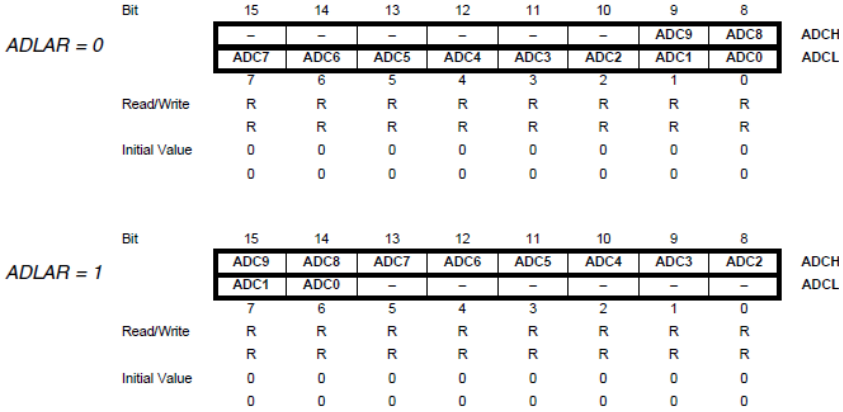


Рисунок 8.15 – Регістр даних АЦП – ADCL і ADCH

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.

ADC9:0: результат перетворення АЦП

Ці біти представляють результат перетворення.

Контрольні запитання до теми 8

1. Які основні функції виконує аналоговий компаратор?
2. Які вимоги до живлення аналогового компаратора?
3. Яким чином вибирається опорна напруга фіксованої забороненої зони?
4. Які режими спрацьовування переривань від аналогового компаратора доступні?
5. Як можна увімкнути або вимкнути аналоговий компаратор?
6. Як вибирається вхідний контакт для підключення до негативного входу аналогового компаратора?
7. Як компаратор ініціює функцію захоплення таймера-лічильника?
8. Яким чином налаштовується вихід аналогового компаратора?
9. Що таке АЦП і яке його призначення в мікроконтролерах?
10. Як розрядність впливає на точність вимірювань АЦП?
11. Які основні параметри АЦП потрібно враховувати при виборі для конкретного застосування?
12. Що таке швидкодія АЦП і чому вона важлива?
13. Які типи АЦП існують і які їхні відмінності?
14. Як працює АЦП послідовного наближення? Наведіть основні кроки.
15. Які можливості використання мультиплексора АЦП для підключення аналогових входів?
16. Які переваги та обмеження використання вбудованих АЦП в мікроконтролерах AVR?
17. Яка роздільна здатність АЦП мікроконтролера ATMega8?
18. Яка інтегральна нелінійність у АЦП ATMega8?
19. Яка абсолютна точність результату перетворення у мікроконтролера ATMega8?
20. Який час перетворення у режимі з максимальною роздільною здатністю?
21. Скільки мультиплексованих односторонніх вхідних каналів має АЦП ATMega8?
22. Як підключається внутрішня опорна напруга АЦП мікроконтролера ATMega8?
23. Як можна вибрати опорну напругу АЦП ATMega8 для перетворення?
24. Як відрізняється режим безперервного перетворення від режиму одноразового перетворення у АЦП ATMega8?
25. Які особливості вибору вхідного каналу та опорної напруги в режимі перетворення АЦП мікроконтролера ATMega8?

Використана література

1. Atmel: ATMega8, ATmega8L: технічна документація на мікроконтролер. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2486-8-bit-AVR-microcontroller-ATmega8_L_datasheet.pdf

9 ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ СІМЕЙСТВА AVR

Метою вивчення теми є ознайомлення з типами пам'яті мікроконтролерів AVR та їх програмуванні за допомогою послідовного інтерфейсу.

Завдання вивчення теми збігаються з переліком питань для розгляду, що наведений нижче.

Перелік питань до розділу:

9.1. Пам'ять мікроконтролерів AVR.
9.2. Самопрограмування пам'яті.
9.3. Біти фьюзів та блокування пам'яті.
9.4. Програмування пам'яті за допомогою послідовного інтерфейсу.

9.1 Пам'ять мікроконтролерів AVR

Вбудована перепрограмована флеш-пам'ять програм. ATmega8 містить вбудовану перепрограмовану флеш-пам'ять об'ємом 8 Кбайт, для зберігання програм. Оскільки всі інструкції AVR мають ширину 16 або 32 біти, флеш-пам'ять організовано як 4K × 16 бітів. Для безпеки програмного забезпечення простір флеш-пам'яті поділено на два розділи: розділ завантажувача і розділ програми.

Флеш-пам'ять має витривалість щонайменше 10 000 циклів запису/стирання. Лічильник програм ATmega8 (PC) має ширину 12 біт, таким чином маючи можливість адресувати пам'ять програм 4K. Функціонування розділу завантажувача та відповідних бітів блокування завантажувача для захисту програмного забезпечення детально розглянемо далі.

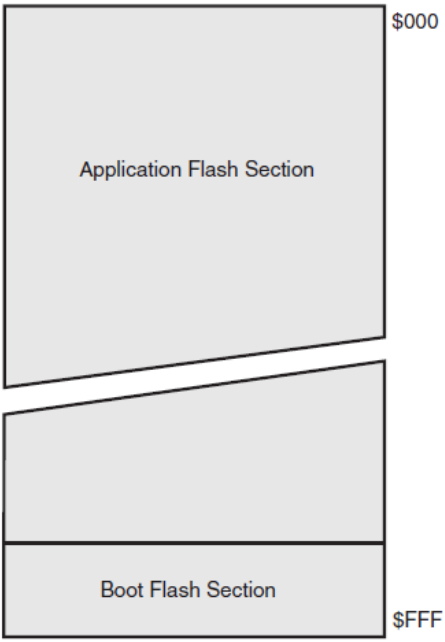


Рисунок 9.1 – Вбудована перепрограмована флеш-пам'ять програм

Таблиці констант можна розміщувати в межах адресного простору пам'яті програми за допомогою команди LPM (Load Program Memory – Читання з пам'яті програми).

Пам'ять даних SRAM. На рисунку 9.2 показано, як організована пам'ять SRAM.

Нижні 1120 комірок пам'яті даних адресують регістровий файл, пам'ять вводу-виводу та внутрішній SRAM даних. Перші 96 комірок адресують регістровий файл і пам'ять вводу-виводу, а наступні 1024 комірки адресують внутрішні дані SRAM.

П'ять різних режимів адресації для пам'яті даних охоплюють: прямий, непрямий із зміщенням, непрямий, непрямий із передекрементом і непрямий із постінкрементом. У регістровому файлі регістри від R26 до R31 містять регістри покажчиків непрямої адресації.

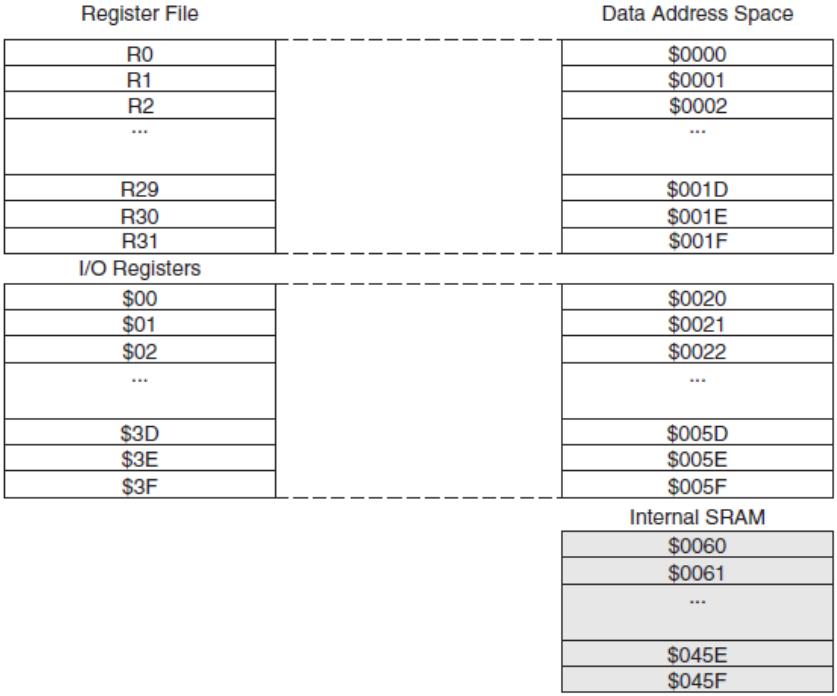


Рисунок 9.2 – Організація пам’яті SRAM

Пряма адресація охоплює весь простір даних.

Непрямий режим зі зміщенням досягає 63 адресних позицій від базової адреси, наданої Y-регістром або Z-регістром.

При використанні режимів непрямої адресації з автоматичним передекрементом і постінкрементом адресні регістри X, Y і Z декрементуються або інкрементуються.

32 робочі регістри загального призначення, 64 регістри вводу-виводу та 1024 байти внутрішньої пам’яті SRAM даних у ATmega8 доступні через усі ці режими адресації.

Пам’ять даних EEPROM. ATmega8 містить 512 байт пам’яті даних EEPROM. Вона організована як окремий простір даних, у якому можна читати та записувати окремі байти. EEPROM має

витривалість щонайменше 100 000 циклів запису/стирання. Доступ між EEPROM і центральним процесором описано нижче, використовуючи адресні регістри EEPROM, регістр даних EEPROM і регістр керування EEPROM.

Доступ для читання-запису EEPROM. Регістри доступу до EEPROM доступні в просторі вводу-виводу.

Час доступу до запису для EEPROM наведено в таблиці далі. Проте функція самосинхронізації дозволяє програмному забезпеченню користувача визначити, коли можна записати наступний байт. Якщо код користувача містить інструкції, які записують EEPROM, необхідно вжити деяких запобіжних заходів. У джерелах живлення з сильною фільтрацією VCC, ймовірно, буде повільно зростати або падати під час увімкнення/вимкнення живлення. Це змушує пристрій деякий час працювати з напругою, нижчою, ніж зазначено як мінімальне для використовуваної тактової частоти.

Щоб запобігти ненавмисному запису EEPROM, слід дотримуватися спеціальної процедури запису.

Коли EEPROM зчитується, ЦП зупиняється на чотири такти перед виконанням наступної інструкції. Коли EEPROM записується, ЦП зупиняється на два такти перед виконанням наступної інструкції.

Регістри пам'яті даних EEPROM:

Адресний регістр EEPROM – EEARH і EEARL (рис. 9.3):

Біти 15..9 – Res: зарезервовані біти

Ці біти є зарезервованими бітами в ATmega8 і завжди читатимуться як нуль.

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	–	EEAR8	EEARH
	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEARL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	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	X	
	X	X	X	X	X	X	X	X	

Рисунок 9.3 – Адресний регістр EEPROM – EEARH і EEARL

Біти 8..0 – EEAR8..0: адреса EEPROM

Адресні регістри EEPROM – EEARH і EEARL – визначають адресу EEPROM у просторі EEPROM розміром 512 байт. Байти даних EEPROM адресуються лінійно від 0 до 511. Початкове значення EEAR не визначено. Перед доступом до EEPROM необхідно записати правильне значення.

Регістр даних EEPROM – EEDR (рис. 9.4).

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	EEDR
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 9.4 – Регістр даних EEPROM – EEDR

Біти 7..0 – EEDR7..0: дані EEPROM

При операції запису EEPROM регістр EEDR містить дані для запису в EEPROM за адресою, наданою регістром EEAR. Для операції читання EEPROM EEDR містить дані, зчитані з EEPROM за адресою, наданою EEAR.

Регістр керування EEPROM – EECR (рис. 9.5):

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	EERIE	EEMWE	EEWE	EERE	EECR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	X	0	

Рисунок 9.5 – Регістр керування EEPROM – EECR

Біти 7..4 – Res: зарезервовані біти

Ці біти є зарезервованими бітами в ATmega8 і завжди читатимуться як нуль.

Біт 3 – EERIE: Дозвіл переривання при готовності EEPROM

Запис одиниці в EERIE увімкне переривання при готовності EEPROM, якщо встановлено біт I у регістрі SREG. Запис нуля в EERIE вимикає переривання. Переривання готовності EEPROM створює постійне переривання, поки біт EEWE очищено.

Біт 2 – EEMWE: дозвіл можливості запису в EEPROM

Біт EEMWE визначає, чи спричиняє встановлення біту EEWE в одиницю запис в EEPROM. Якщо встановлено EEMWE, встановлення біту EEWE протягом чотирьох тактів записуватиме дані в EEPROM за вибраною адресою. Якщо EEMWE дорівнює нулю, встановлення EEWE не матиме ефекту. Коли програмне забезпечення записує одиницю в EEMWE, апаратне забезпечення очищує цей біт через чотири такти.

Біт 1 – EEWE: дозвіл запису в EEPROM

Біт дозволу запису в EEPROM EEWE є стробом запису в EEPROM. Коли адреса та дані встановлені правильно, у біт EEWE потрібно записати одиницю, щоб записати значення в EEPROM. У біт EEMWE має бути записана одиниця перед тим, як одиниця буде записана в EEWE, інакше запис EEPROM не відбудеться. Під час запису EEPROM слід дотримуватися наступної процедури (порядок кроків 3 і 4 не є суттєвим):

1. Зачекайте, поки EEWE не стане рівним нулю.
2. Зачекайте, доки біт SPMEN у регістрі SPMCR не стане рівним нулю.
3. Запишіть нову адресу EEPROM до EEAR (необов'язково).
4. Запишіть нові дані EEPROM до EEDR (необов'язково).
5. Запишіть логічну одиницю в біт EEMWE, одночасно записуючи нуль у біт EEWE в регістрі EECR.
6. Протягом чотирьох тактів після встановлення EEMWE запишіть логічну одиницю в EEWE.

EEPROM не може бути записана під час запису ЦП у флеш-пам'ять. Програмне забезпечення повинно перевірити, чи завершено програмування флеш-пам'яті, перш ніж ініціювати новий запис EEPROM. Крок 2 має сенс, лише якщо програмне забезпечення містить завантажувач, який дозволяє ЦП програмувати флеш-пам'ять. Якщо вона ніколи не оновлюється ЦП, крок 2 можна пропустити.

Застереження: переривання між кроком 5 і кроком 6 призведе до збою циклу запису, оскільки тайм-аут основного дозволу запису EEPROM завершиться. Якщо процедура переривання доступу до EEPROM перериває інший доступ до EEPROM, регістр EEAR або EEDR буде змінено, що призведе до збою перерваного доступу

до EEPROM. Щоб уникнути цих проблем, рекомендовано зняти глобальний прапор переривання під час усіх кроків.

Коли час доступу до запису минув, біт EEWE очищається апаратно. Програмне забезпечення користувача може опитувати цей біт і чекати нуля перед записом наступного байту. Коли встановлено EEWE, ЦП зупиняється на два цикли перед виконанням наступної інструкції.

Біт 0 – EERE: дозвіл читання з EEPROM

Біт дозволу читання з EEPROM EERE є стробом читання EEPROM. Коли в регістрі EEAR встановлено правильну адресу, в біт EERE потрібно записати логічну одиницю, щоб запустити зчитування з EEPROM. Доступ для читання EEPROM виконується за одну інструкцію, і запитані дані доступні негайно. Коли EEPROM зчитується, ЦП зупиняється на чотири цикли перед виконанням наступної інструкції. Користувач повинен перевірити біт EEWE перед початком операції читання. Якщо виконується операція запису, неможливо ні прочитати EEPROM, ні змінити реєстр EEAR.

Робота з EEPROM. Калібрований осцилятор використовується для визначення часу доступу до EEPROM. У таблиці 9.1 наведено типовий час програмування для доступу до EEPROM з ЦП.

Таблиця 9.1 – Типовий час програмування для доступу до EEPROM

Операція	**Кількість циклів каліброваного RC осцилятора (Використовується тактова частоту 1 МГц незалежно від налаштувань бітів CKSEL)**	**Типовий час програмування**
Запис EEPROM (з ЦП)	8448	8.5 ms

У наведеному нижче прикладі коду показано функцію С для запису в EEPROM. Приклад передбачає, що переривання контролюються (наприклад, шляхом глобального вимкнення переривань), щоб під час виконання цієї функції не виникало жодних переривань. У прикладі також передбачається, що в програмному забезпеченні відсутній завантажувач Flash. Якщо такий код присутній,

функція запису EEPROM також повинна чекати завершення будь-якої поточної команди SPM.

```
void EEPROM_write(unsigned int uiAddress, unsigned
char ucData)
{
// Очікування на завершення попереднього запису
while(EECR & (1<<EEWE));
// Встановлення регістрів адреси і даних
EEAR = uiAddress;
EEDR = ucData;
// Запис логічної одиниці в EEMWE
EECR |= (1<<EEMWE);
// Початок запису в EEPROM, встановивши EEWE
EECR |= (1<<EEWE);
}
```

Наступний приклад коду показує функцію С для читання EEPROM. У прикладі припускається, що переривання керуються таким чином, що під час виконання цієї функцій не буде жодних переривань.

```
unsigned char EEPROM_read(unsigned int uiAddress)
{
// Очікування на завершення попереднього запису
while(EECR & (1<<EEWE));
// Встановлення регістру адреси
EEAR = uiAddress;
// Початок читання EERPM шляхом встановлення біту EERE
EECR |= (1<<EERE);
// Повертання зчитаних даних з реєстру даних
return EEDR;
}
```

Запис EEPROM під час режиму сну. Під час входу в сплячий режим вимкнення, коли операція запису EEPROM активна,

операція запису EEPROM продовжуватиметься та завершиться до того, як мине час доступу до запису. Однак після завершення операції запису осцилятор продовжує працювати, і, як наслідок, пристрій не вимикається повністю. Тому рекомендується переконатися, що операція запису EEPROM завершена перед входом у режим вимкнення.

Запобігання пошкодженню EEPROM. Під час зниженої напруги живлення дані EEPROM можуть бути пошкоджені, оскільки вона надто низька для належної роботи ЦП і EEPROM.

Пошкодження даних EEPROM може бути викликано двома ситуаціями, коли напруга надто низька. По-перше, звичайна послідовність запису в EEPROM вимагає мінімальної напруги для правильної роботи. По-друге, сам ЦП може неправильно виконувати інструкції, якщо напруга живлення занадто низька.

Пошкодження даних EEPROM можна легко уникнути, дотримуючись цієї рекомендації щодо проектування: утримуйте AVR RESET активним (низьким) протягом всього часу зниженої напруги живлення. Це можна зробити, увімкнувши внутрішній детектор зниження напруги живлення (BOD). Якщо рівень напруги спрацювання внутрішнього BOD не відповідає необхідному рівню виявлення, можна використати зовнішню схему захисту від зниження VCC. Якщо скидання відбувається під час операції запису, операція запису буде завершена за умови достатньої напруги живлення.

9.2 Самопрограмування пам'яті

Підтримка завантажувача забезпечує реальний механізм самопрограмування шляхом читання під час запису для завантаження та вивантаження програмного коду самим мікроконтролером. Ця функція дозволяє гнучко оновлювати прикладне програмне забезпечення під управлінням ЦП за допомогою програми завантажувача, що знаходиться у флеш-пам'яті. Програма завантажувача може використовувати будь-який доступний інтерфейс даних і пов'язаний протокол для читання коду та запису (програмування)

цього коду у флеш-пам'ять або читання коду з пам'яті програми. Програмний код у розділі завантажувача має можливість записувати всю флеш-пам'ять, включаючи пам'ять завантажувача. Таким чином, завантажувач може навіть змінювати себе, а також може стерти себе з коду, якщо функція більше не потрібна. Розмір пам'яті завантажувача налаштовується за допомогою фьюз-бітів. Крім того, завантажувач має два окремих набори бітів блокування завантаження, які можна встановити незалежно. Це дає користувачеві унікальну гнучкість у виборі різних рівнів захисту:

– самопрограмування шляхом читання під час запису;

– гнучкий розмір завантажувальної пам'яті;

– високий рівень безпеки (окремі біти блокування для гнучкого захисту);

– окремий фьюз для вибору вектору скидання;

– оптимізований розмір сторінки (Сторінка – це розділ у флеш-пам'яті, що складається з кількох байтів, який використовується під час програмування. Сторінкова організація не впливає на звичайну роботу);

– ефективний код завантажувача;

– ефективна підтримка читання – зміни – запису.

Розділи пам'яті програми користувача та завантажувача. Флеш-пам'ять складається з двох основних розділів: розділу програми та розділу завантажувача. Розмір різних секцій налаштовується фьюзами BOOTSZ, як буде показано далі. Ці дві секції можуть мати різний рівень захисту, оскільки вони мають різні набори бітів блокування.

Розділ програми – це частина флеш-пам'яті, яка використовується для зберігання коду програми. Рівень захисту для розділу програми можна вибрати за допомогою бітів блокування завантаження програми. Розділ програми ніколи не може зберігати будь-який код завантажувача, оскільки інструкція SPM (Store Program Memory – Запис в пам'ять програми) вимкнена під час виконання з розділу програми.

Хоча розділ програми використовується для зберігання коду програми, програмне забезпечення завантажувача має бути розташоване в області завантажувача, оскільки інструкція SPM може ініціювати програмування під час виконання лише з неї. Інструкція SPM може отримати доступ до всієї флеш-пам'яті, включаючи сам

розділ завантажувача. Рівень захисту для розділу завантажувача можна вибрати за допомогою бітів блокування завантажувача.

Флеш-секції «читання під час запису» і «без читання під час запису». Чи підтримує ЦП читання під час запису, чи він зупиняється під час оновлення програмного забезпечення завантажувача, залежить від адреси, яка програмується. На додаток до двох розділів, які можна налаштувати за допомогою фьюзів BOOTSZ, як описано вище, флеш-пам'ять також розділено на дві фіксовані секції: секцію «Читання під час запису» (RWW) і секцію «Без читання під час запису» (NRWW). Обмеження між розділами RWW і NRWW наведено в таблиці 9.2 і на рисунку 9.6.

Таблиця 9.2 – Обмеження між розділами RWW і NRWW

До якого розділу звертається вказівник Z під час програмування?	Який розділ можна прочитати під час програмування?	Процесор зупинено?	Читання під час запису підтримується?
Розділ RWW	Розділ NRWW	Ні	Так
Розділ NRWW	Немає	Так	Ні

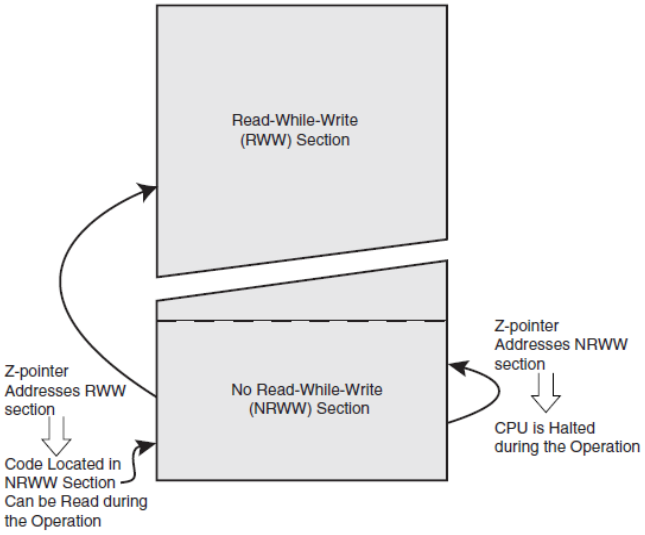


Рисунок 9.6 – Розділення флеш-пам'яті на дві фіксовані секції

Основна відмінність між цими двома розділами:

– під час стирання або запису сторінки, розташованої всередині розділу RWW, розділ NRWW може бути зчитаним під час операції;

– під час стирання або запису сторінки, розташованої всередині розділу NRWW, ЦП зупиняється протягом усієї операції.

Зверніть увагу, що програмне забезпечення користувача ніколи не може прочитати будь-який код, який знаходиться всередині розділу RWW під час роботи програмного забезпечення завантажувача. Фраза «розділ читання під час запису» вказує на те, який розділ програмується (стирається чи записується), а не на те, який розділ фактично читається під час оновлення програмного забезпечення завантажувача.

Біти блокування завантажувача. Якщо завантажувач не потрібен, вся пам'ять флеш доступна для програмного коду. Завантажувач має два окремих набори біт блокування завантаження, які можна встановити незалежно. Це дає користувачеві унікальну гнучкість у виборі різних рівнів захисту. Користувач може вибрати:

– для захисту всього Flash від оновлення програмного забезпечення процесором;

– для захисту лише розділу завантажувача від оновлення програмного забезпечення процесором;

– для захисту лише розділу Flash програми від оновлення програмного забезпечення процесором;

– дозволити оновлення програмного забезпечення в усьому розділі флеш.

Біти блокування завантажувача можуть бути встановлені в програмному забезпеченні та в режимі послідовного або паралельного програмування, але їх можна очистити лише командою стирання мікросхеми. Загальне блокування запису (режим блокування 2) не контролює програмування флеш-пам'яті за допомогою інструкції SPM. Подібним чином загальне блокування читання/запису (режим блокування 3) не контролює ні читання, ні запис за допомогою LPM/SPM.

Таблиця 9.3 – Біти блокування завантажувача режиму BLB0

Режим BLB0	BLB02	BLB01	Захист
1	1	1	Жодних обмежень для доступу SPM або LPM до розділу програми
2	1	0	SPM заборонено писати в розділ програми
3	0	0	SPM не дозволяється писати в розділ програми, а LPM, що виконується з розділу завантажувача, заборонено читати із розділу програми. Якщо вектори переривань розміщені в розділі завантажувача, переривання вимкнені під час виконання із розділу програми
4	0	1	LPM, що виконується із розділу завантажувача, заборонено читати з розділу програми. Якщо вектори переривань розміщені у розділі завантажувача, переривання вимкнені під час виконання із розділу програми

Таблиця 9.4 – Біти блокування завантажувача режиму BLB1

Режим BLB1	BLB12	BLB11	Захист
1	1	1	Жодних обмежень для доступу SPM або LPM до розділу завантажувача
2	1	0	SPM заборонено писати в розділ завантажувача
3	0	0	SPM не дозволяється писати в розділ завантажувача, а LPM, що виконується з розділу програми, заборонено читати із розділу завантажувача. Якщо вектори переривань розміщені в розділі програми, переривання вимкнені під час виконання із розділу завантажувача
4	0	1	LPM, що виконується із розділу програми, заборонено читати з розділу завантажувача. Якщо вектори переривань розміщені у розділі програми, переривання вимкнені під час виконання із розділу завантажувача

Вхід у програму завантажувача. Вхід до завантажувача відбувається шляхом переходу або виклику з програми користувача. Це може бути ініційовано тригером, таким як команда, отримана через інтерфейс USART або SPI. Крім того, фьюз завантажувача при скиді BOOTRST можна запрограмувати так, щоб вектор скидання вказував на початкову адресу завантажувача після скидання. У цьому випадку завантажувач запускається після скидання. Після завантаження коду програми мікроконтролер може розпочати її виконання. Зверніть увагу, що фьюзи не можуть бути змінені самим ЦП. Це означає, що коли фьюз завантажувача при скиді запрограмовано, вектор скидання завжди вказуватиме на скид завантажувача, а фьюз можна змінити лише через послідовний або паралельний інтерфейс програмування.

Таблиця 9.5 – Програмування фьюзу завантажувача при скиді

BOOTRST	Адреса при скиді
0	Скидання програми (адреса 0x0000)
1	Скидання завантажувача

Регістр керування пам'яттю програм – SPMCR (рис. 9.7):
Біт 7 – SPMIE: дозволено переривання SPM

Коли в біт SPMIE записано одиницю, а біт I в регістрі стану встановлений, буде включено переривання готовності SPM. Воно буде активним весь час, поки біт SPMEN у регістрі SPMCR очищено.

Bit	7	6	5	4	3	2	1	0	
	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	SPMCR
Read/Write	R/W	R	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 9.7 – Регістр керування пам'яттю програм – SPMCR

Біт 6 – RWWSB: розділ «читання під час запису» зайнятий

Коли починається операція самопрограмування (стирання сторінки або запис сторінки) у розділі RWW, RWWSB буде встановлено (один) апаратним забезпеченням. Коли встановлено біт RWWSB,

доступ до розділу RWW заборонено. Біт RWWSB буде очищено, якщо в біт RWWSRE буде записано одиницю після завершення операції самопрограмування. Крім того, біт RWWSB буде автоматично очищено, якщо розпочато операцію завантаження сторінки.

Біт 5 – Res: зарезервований біт

Цей біт є зарезервованим в ATmega8 і завжди читається як нуль.

Біт 4 – RWWSRE: увімкнути читання розділу «читання під час запису»

Під час програмування (стирання сторінки або запису сторінки) розділу RWW, він блокується для читання (біт RWWSB встановлюється апаратно). Щоб знову ввімкнути розділ RWW, програмне забезпечення користувача має дочекатися завершення програмування (поки SPMEN буде очищено). Потім, якщо біт в RWWSRE записати одиницю одночасно з SPMEN, наступна інструкція SPM протягом чотирьох тактових циклів повторно вмикає розділ RWW. Розділ RWW не можна повторно ввімкнути, поки флеш-пам'ять зайнята стиранням сторінки або записом сторінки (встановлено SPMEN). Якщо біт RWWSRE записати під час завантаження флеш-пам'яті, операцію завантаження флеш-пам'яті буде перервано, а завантажені дані буде втрачено (буфер сторінки буде очищено, коли розділ читання під час запису знову ввімкнено).

Біт 3 – BLBSET: встановити біт блокування завантаження

Якщо в цей біт записати одиницю одночасно з SPMEN, наступна інструкція SPM протягом чотирьох тактових циклів встановлює біти блокування завантаження відповідно до даних у R0. Дані в R1 і адреса у вказівнику Z ігноруються. Біт BLBSET буде автоматично очищено після завершення встановлення біта блокування або якщо жодна інструкція SPM не буде виконана протягом чотирьох тактів. Інструкція LPM протягом трьох циклів після встановлення BLBSET і SPMEN у регістрі SPMCR зчитує біти блокування або фьюз-біти (залежно від значення Z0 у вказівнику Z) у регістр призначення.

Біт 2 – PGWRT: запис сторінки

Якщо в цей біт записати одиницю одночасно з SPMEN, наступна інструкція SPM протягом чотирьох тактів виконує запис сторінки із збереженням даних у тимчасовому буфері. Адреса сторінки береться з верхньої частини вказівника Z. Дані в R1 і R0

ігноруються. Біт PGWRT автоматично очищається після завершення запису сторінки або якщо жодна інструкція SPM не виконується протягом чотирьох тактів. ЦП зупиняється протягом усієї операції запису сторінки, якщо адресовано розділ NRWW.

Біт 1 – PGERS: стирання сторінки

Якщо в цей біт записати одиницю одночасно з SPMEN, наступна інструкція SPM протягом чотирьох тактових циклів виконує стирання сторінки. Адреса сторінки береться з верхньої частини вказівника Z. Дані в R1 і R0 ігноруються. Біт PGERS автоматично очищається після завершення стирання сторінки або якщо жодна інструкція SPM не виконується протягом чотирьох тактів. ЦП зупиняється протягом усієї операції запису сторінки, якщо адресовано розділ NRWW.

Біт 0 – SPMEN: увімкнення пам'яті для збереження програм

Цей біт дозволяє інструкцію SPM протягом наступних чотирьох тактів. Якщо записати одиницю в цей біт разом із RWWSRE, BLBSET, PGWRT або PGERS, наступна інструкція SPM матиме особливе значення, див. опис вище. Якщо записати лише SPMEN, наступна інструкція SPM зберігатиме значення в R1:R0 у тимчасовому буфері сторінок, адресованому вказівником Z. LSB вказівника Z ігнорується. Біт SPMEN автоматично очищається після завершення інструкції SPM або якщо жодна інструкція SPM не виконується протягом чотирьох тактів. Під час стирання сторінки та запису сторінки біт SPMEN залишається високим до завершення операції.

Запис будь-якої іншої комбінації, крім «10001», «01001», «00101», «00011» або «00001», у молодші п'ять бітів цього регістру не матиме ефекту.

Адресація флеш під час самопрограмування. Вказівник Z використовується для адресації команд SPM.

Оскільки флеш організований у сторінки, програмний лічильник можна вважати двома різними розділами. Один розділ, що складається з молодших бітів, адресує слова на сторінці, тоді як старші біти адресують сторінки. Це показано на рисунку 9.9. Зверніть увагу, що операції стирання сторінки та запису сторінки адресуються незалежно. Тому дуже важливо, щоб програмне забезпечення завантажувача зверталося до однієї сторінки як під час

Bit	15	14	13	12	11	10	9	8
ZH (R31)	Z15	Z14	Z13	Z12	Z11	Z10	Z9	Z8
ZL (R30)	Z7	Z6	Z5	Z4	Z3	Z2	Z1	Z0
	7	6	5	4	3	2	1	0

Рисунок 9.8 – Адресація флеш під час самопрограмування

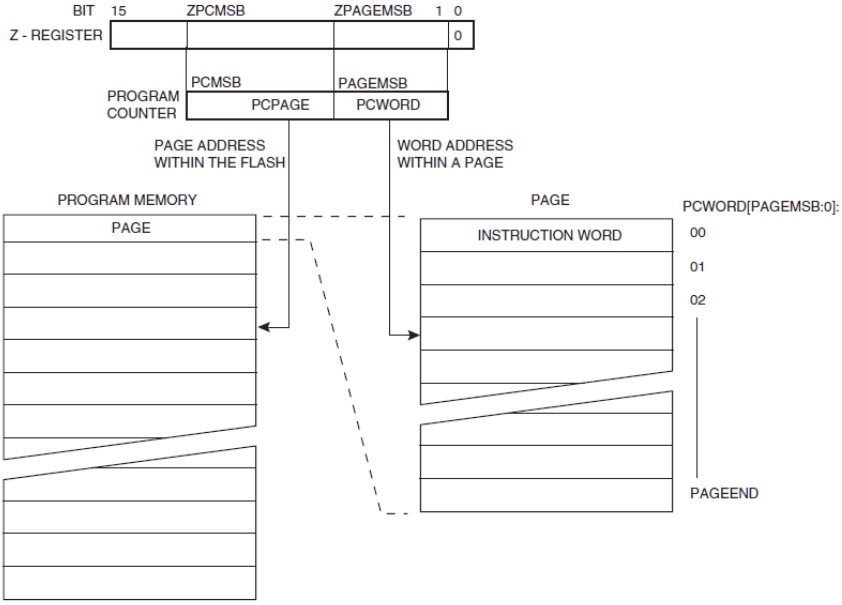


Рисунок 9.9 – Адресування флеш

операції стирання сторінки, так і під час її запису. Після початку операції програмування адреса фіксується, і вказівник Z можна використовувати для інших операцій. Єдиною операцією SPM, яка не використовує вказівник Z, є встановлення бітів блокування завантажувача. Вміст вказівника Z ігнорується та не матиме жодного впливу на роботу. Інструкція LPM також використовує вказівник Z для збереження адреси. Оскільки ця інструкція звертається до флеш-пам'яті побайтно, також використовується LSB (біт Z0) Z-вказівника.

Самопрограмування флеш-пам'яті. Пам'ять програми оновлюється посторінково. Перед програмуванням сторінки даними, що зберігаються у тимчасовому буфері сторінки, її необхідно стерти. Тимчасовий буфер сторінки заповнюється по одному слову за допомогою SPM, і він може бути заповнений перед командою стирання сторінки або між операцією стирання сторінки та запису сторінки:

Варіант 1, заповнити буфер перед стиранням сторінки:

1. Заповнити тимчасовий буфер сторінки.
2. Ввиконати стирання сторінки.
3. Виконати запис сторінки.

Варіант 2, заповнити буфер після стирання сторінки:

1. Виконати стирання сторінки.
2. Заповнити тимчасовий буфер сторінки.
3. Виконати запис сторінки.

Якщо потрібно змінити лише частину сторінки, решту сторінки потрібно зберегти (наприклад, у тимчасовому буфері сторінки) перед стиранням, а потім переписати. При використанні варіанта 1 завантажувач забезпечує ефективну функцію читання-зміни-запису, яка дозволяє програмному забезпеченню користувача спочатку прочитати сторінку, внести необхідні зміни, а потім записати змінені дані. Якщо використовується варіант 2, неможливо прочитати старі дані під час завантаження, оскільки сторінку вже стерто. Доступ до буфера тимчасової сторінки можна отримати у довільній послідовності. Важливо, щоб адреса сторінки, яка використовується як в операції стирання сторінки, так і в операції запису сторінки, була спрямована на ту саму сторінку.

Виконання стирання сторінки за допомогою SPM. Щоб виконати стирання сторінки, установіть адресу у вказівнику Z, запишіть "X0000011" у SPMCR і виконайте SPM протягом чотирьох тактів після запису SPMCR. Дані в R1 і R0 ігноруються. Адреса сторінки повинна бути записана в PCPAGE в Z-реєстрі. Під час цієї операції інші біти в Z-покажчику ігноруватимуться.

1. Стирання сторінки у розділі RWW: розділ NRWW можна прочитати під час стирання сторінки.

2. Стирання сторінки у розділі NRWW: ЦП зупиняється під час операції.

Примітка. Якщо в часовій послідовності виникає переривання, чотиритактний доступ не може бути гарантований.

Щоб забезпечити неподільну роботу, вимкніть переривання перед записом у SPMCSR.

Заповнення тимчасового буфера (завантаження сторінки). Щоб написати слово з інструкцією, встановіть адресу в Z-покажчику та дані в R1:R0, запишіть «00000001» у SPMCR і виконайте SPM протягом чотирьох тактів після запису SPMCR. Вміст PCWORD у Z-реєстрі використовується для адресації даних у тимчасовому буфері сторінки. Тимчасовий буфер автоматично стирається після операції запису сторінки або запису біта RWWSRE у SPMCR. Він також стирається після скидання системи. Зверніть увагу, що неможливо записати більше одного разу у кожну адресу без видалення тимчасового буфера.

Примітка. Якщо EEPROM записати в середині операції завантаження сторінки SPM, усі завантажені дані буде втрачено.

Виконання запису сторінки. Щоб виконати запис сторінки, установіть адресу у вказівнику Z, запишіть "X0000101" у SPMCR і виконайте SPM протягом чотирьох тактів після запису SPMCR. Дані в R1 і R0 ігноруються. Адреса сторінки повинна бути записана в PCPAGE. Інші біти в Z-покажчику повинні бути записані в нуль під час цієї операції.

1. Запис сторінки в розділ RWW: розділ NRWW можна прочитати під час запису сторінки.

2. Запис сторінки в розділ NRWW: ЦП зупиняється під час операції.

Використання переривання SPM. Якщо переривання SPM увімкнено, воно генеруватиме постійне переривання, коли біт SPMEN у SPMCR очищено. Це означає, що переривання можна використовувати замість опитування регістру SPMCR у програмному забезпеченні. При використанні переривання SPM вектори переривань слід перемістити до розділу завантажувача, щоб уникнути того, що переривання звертається до розділу RWW, коли його заблоковано для читання.

Замітки щодо оновлення завантажувача. Необхідно бути особливо обережним, якщо користувач дозволяє оновлювати розділ завантажувача, залишаючи незапрограмованим біт 11 блокування завантаження. Випадковий запис до самого завантажувача може пошкодити весь завантажувач, і подальше оновлення програмного забезпечення може бути неможливим. Якщо немає необхідності змінювати саме програмне забезпечення завантажувача, рекомендується запрограмувати біт 11 блокування завантажувача, щоб захистити програмне забезпечення завантажувача від будь-яких внутрішніх змін програмного забезпечення.

Запобігання читання розділу RWW під час самопрограмування. Під час самопрограмування (або стирання сторінки, або запису сторінки) розділ RWW завжди блокується для читання. Саме програмне забезпечення користувача повинно запобігати зверненню до цього розділу під час операції самопрограмування. RWWSB у SPMCR буде залишатися встановленим, доки розділ RWW зайнятий. Під час самопрограмування таблицю векторів переривань слід перемістити до розділу завантажувача, або переривання мають бути вимкнені. Перед зверненням до розділу RWW після завершення програмування програмне забезпечення користувача має очистити RWWSB, записавши біт RWWSRE.

Встановлення бітів блокування завантажувача за допомогою SPM. Щоб установити біти блокування завантажувача, запишіть потрібні дані в R0, запишіть “X0001001” у SPMCR і виконайте SPM протягом чотирьох тактів після запису SPMCR. Єдиними доступними бітами блокування є біти блокування завантажувача, які можуть запобігти будь-якому оновленню програмного забезпечення MCU для розділу програми та завантажувача.

Якщо біти 5..2 у R0 очищено, відповідний біт блокування завантаження буде запрограмований, якщо інструкція SPM виконується протягом чотирьох циклів після встановлення BLBSET і SPMEN у SPMCR. Вказівник Z не має значення під час цієї операції, але для майбутньої сумісності рекомендується завантажити 0x0001 у вказівник Z (те саме, що використовується для читання бітів блокування).

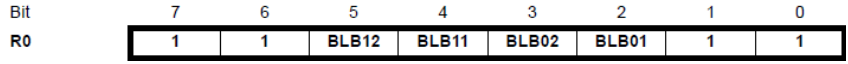


Рисунок 9.10 – Встановлення бітів блокування завантажувача за допомогою SPM

Для майбутньої сумісності також рекомендується встановити біти 7, 6, 1 і 0 у R0 у «1» під час запису бітів блокування. Під час програмування бітів блокування всю флеш-пам'ять можна зчитувати протягом всієї операції.

Запис EEPROM запобігає запису в SPMCR. Зверніть увагу, що операція запису EEPROM блокує все програмування флеш-пам'яті. Зчитування фьюзів і бітів блокування з програмного забезпечення також буде заблоковано під час операції запису EEPROM. Рекомендується, щоб користувач перевірив біт стану (EEWE) у регістрі EECR і переконався, що біт очищено перед записом у регістр SPMCR.

Зчитування фьюзів і бітів блокування з програмного забезпечення. З програмного забезпечення можна зчитувати як фьюзи, так і біти блокування. Щоб прочитати біти блокування, завантажте у вказівник Z 0x0001 і встановіть біти BLBSET і SPMEN у SPMCR. Коли інструкція LPM виконується протягом трьох тактів ЦП після встановлення бітів BLBSET і SPMEN у SPMCR, значення бітів блокування буде завантажено в регістр призначення. Біти BLBSET і SPMEN автоматично очищаються після завершення читання бітів блокування або якщо жодна інструкція LPM не виконується протягом трьох циклів ЦП або жодна інструкція SPM не виконується протягом чотирьох циклів ЦП.

Алгоритм зчитування бітів молодшого байту фьюза подібний до описаного вище для зчитування бітів блокування. Щоб прочитати біти молодшого байту фьюза, завантажте у вказівник Z 0x0000 і встановіть біти BLBSET і SPMEN у SPMCR. Коли інструкція

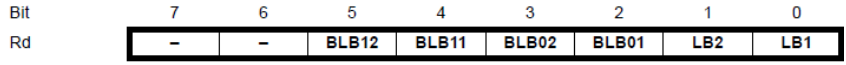


Рисунок 9.11 – Зчитування фьюзів і бітів блокування з програмного забезпечення

LPM виконується протягом трьох тактів після встановлення бітів BLBSET і SPMEN у SPMCR, значення бітів молодшого байта фьюза (FLB) буде завантажено в регістр призначення.

Bit	7	6	5	4	3	2	1	0
Rd	FLB7	FLB6	FLB5	FLB4	FLB3	FLB2	FLB1	FLB0

Рисунок 9.12 – Алгоритм зчитування бітів молодшого байта фьюза

Подібним чином, для зчитування бітів старшого байту фьюза, завантажте 0x0003 у вказівник Z. Коли інструкція LPM виконується протягом трьох тактів після встановлення бітів BLBSET і SPMEN у SPMCR, значення бітів старшого байту фьюза (FHB) буде завантажено в регістр призначення.

Bit	7	6	5	4	3	2	1	0
Rd	FHB7	FHB6	FHB5	FHB4	FHB3	FHB2	FHB1	FHB0

Рисунок 9.13 – Зчитування бітів старшого байта фьюза

Запрограмовані біти фьюзів та блокування будуть читатися як нуль. Незапрограмовані біти читатимуться як одиниці.

Запобігання пошкодженню флеш-пам'яті. Під час зниженої напруги живлення програма у флеш-пам'яті може бути пошкоджена, оскільки напруга живлення надто низька для належної роботи ЦП і флеш-пам'яті. Пошкодження програми у флеш-пам'яті може бути викликано двома ситуаціями, коли напруга надто низька. По-перше, звичайна послідовність запису у флеш вимагає мінімальної напруги для правильної роботи. По-друге, сам центральний процесор може виконувати інструкції некоректно, якщо напруга живлення для виконання інструкцій занадто низька. Пошкодження флеш-пам'яті можна легко уникнути, дотримуючись цих рекомендацій щодо дизайну (достатньо однієї):

1. Якщо в системі немає потреби в оновленні завантажувача, запрограмуйте біти блокування завантажувача, щоб запобігти будь-яким оновленням програмного забезпечення завантажувача.

2. Тримайте AVR RESET активним (низьким) протягом періодів недостатньої напруги живлення. Це можна зробити, увімкнувши внутрішній детектор зниження напруги живлення (BOD), якщо робоча напруга відповідає рівню виявлення. Якщо ні, можна використовувати зовнішню схему захисту. Якщо скидання відбувається під час операції запису, операція запису буде завершена за умови достатньої напруги живлення.

3. Утримуйте ядро AVR у режимі сну протягом періодів низького VCC. Це запобігатиме спробам центрального процесора декодувати та виконувати інструкції, ефективно захищаючи регістр SPMCR і, таким чином, флеш-пам'ять від ненавмисних записів.

Час програмування флеш-пам'яті при використанні SPM. Відкалібрований RC осцилятор використовується для визначення часу доступу до флеш. У таблиці 9.6 показано типовий час програмування для доступу до флеш-пам'яті із ЦП.

Таблиця 9.6 – Типовий час програмування для доступу до флеш-пам'яті із ЦП

Операція	Мін. час програмування	Макс. час програмування
Запис у флеш (стирання сторінки, запис сторінки, запис бітів блокування за допомогою SPM)	3,7 мс	4,5 мс

Параметри завантажувача ATmega8:

Конфігурація розміру завантажувача представлена у таблиці 9.7.

Таблиця 9.7 – Конфігурація розміру завантажувача

BOOTSZ1	BOOTSZ0	Розмір завантажувача	Сторінки	Розділ програми користувача	Розділ завантажувача	Кінець програми користувача	Адреса скидання завантажувача (початок розділу завантажувача)
1	1	128 слів	4	0x000-0xF7F	0xF80-0xFFF	0xF7F	0xBFF
1	0	256 слів	8	0x000-0xEFF	0xF00-0xFFF	0xEFF	0xF00
0	1	512 слів	16	0x000-0xDFF	0xE00-0xFFF	0xDFF	0xE00
0	0	1024 слова	32	0x000-0xBFF	0xC00-0xFFF	0xBFF	0xC00

Границі розділу «читання під час запису» розміщено у таблиці 9.8.

Таблиця 9.8 – Границі розділу «читання під час запису»

Розділ	Сторінки	Адреси
Розділ читання під час запису (RWW)	96	0x000-0xBFF
Розділ без читання під час запису (NRWW)	32	0xC00-0xFFF

Пояснення різних змінних, які використовуються на рисунку 9.9, і зіставлення з вказівником Z розміщено у таблиці 9.9.

Таблиця 9.9 – Пояснення змінних

Змінна	Біти	Відповідне значення Z	Опис
PCMSB	11		Старший біт у програмному лічильнику (Програмний лічильник є 12-бітним PC [11:0])
PAGEMSB	4		Старший біт, який використовується для адресації слів на одній сторінці (32 слова на сторінці, потрібні 5 біт PC [4:0])
ZPCMSB		Z12	Біт у Z-регістрі, який відображається на PCMSB. Оскільки Z0 не використовується, ZPCMSB дорівнює PCMSB + 1
ZPAGEMSB		Z5	Біт у Z-регістрі, який відображається на PAGEMSB. Оскільки Z0 не використовується, ZPAGEMSB дорівнює PAGEMSB + 1
PCPAGE	PC[11:5]	Z12:Z6	Адреса сторінки програмного лічильника: вибір сторінки для стирання та запису
PCWORD	PC[4:0]	Z5:Z1	Адреса слова програмного лічильника: вибір слова для заповнення тимчасового буфера (має дорівнювати нулю під час операції запису сторінки)

Біти Z15:Z13: завжди ігноруються

Біт Z0: має дорівнювати нулю для всіх команд SPM, вибирає байт для інструкції LPM.

9.3 Біти фьюзів та блокування пам'яті

Біти блокування пам'яті програм та даних. ATmega8 забезпечує шість бітів блокування, які можна залишити незапрограмованими («1») або запрограмувати («0») для отримання додаткових функцій, перелічених у таблиці. Біти блокування можна стерти до «1» лише за допомогою команди Chip Erase.

Таблиця 9.10 – Біти блокування пам'яті програм та даних

Біт блокування	Номер біта	Опис	Значення за замовчанням
	7	–	1 (незапрограмований)
	6	–	1 (незапрограмований)
BLB12	5	Біт блокування завантажувача	1 (незапрограмований)
BLB11	4	Біт блокування завантажувача	1 (незапрограмований)
BLB02	3	Біт блокування завантажувача	1 (незапрограмований)
BLB01	2	Біт блокування завантажувача	1 (незапрограмований)
LB2	1	Біт блокування	1 (незапрограмований)
LB1	0	Біт блокування	1 (незапрограмований)

Опис бітів LBx наведений у таблиці 9.11.

Таблиця 9.11 – Опис бітів LBx

Режим LB	LB2	LB1	Тип захисту
1	1	1	Функції блокування пам'яті вимкнено
2	1	0	Подальше програмування Flash та EEPROM вимкнено в режимі паралельного та послідовного програмування. Біти фьюзів заблоковані як у режимі послідовного, так і паралельного програмування
3	0	0	Подальше програмування та перевірка флеш-пам'яті та EEPROM вимкнені в режимі паралельного та послідовного програмування. Біти фьюзів заблоковані як у режимах послідовного, так і паралельного програмування

Фьюз-біти. ATmega8 має два байти фьюзів. Наступні дві таблиці (таблиця 9.12 та таблиця 9.13) коротко описують функціональні можливості всіх фьюзів і те, як вони розташовані в байтах фьюзів. Зауважте, що фьюзи зчитуються як логічний нуль, «0», якщо вони запрограмовані.

Таблиця 9.12 – Функціональні можливості фьюзів і їх розташування в старших байтах фьюзів

Старший байт фьюзів	Номер біта	Опис	Значення за замовчанням
RSTDISBL	7	Вибирає, чи є PC6 контактом вводу-виводу чи входом RESET	1 (незапрограмований, PC6 є входом RESET)
WDTON	6	Сторожовий таймер (WDT) завжди ввімкнений	1 (незапрограмований, WDT вмикається регістром WDTCR)
SPIEN	5	Увімкнене послідовне завантаження програм і даних	0 (запрограмований, програмування через SPI увімкнене)
CKOPT	4	Опції генератора тактового сигналу	1 (незапрограмований)
EESAVE	3	Пам'ять EEPROM зберігається при виконанні команди Chip Erase	1 (незапрограмований, EEPROM не зберігається)
BOOTSZ1	2	Вибір розміру завантажувача	0 (запрограмований)
BOOTSZ0	1	Вибір розміру завантажувача	0 (запрограмований)
BOOTRST	0	Вибір вектору скиду	1 (незапрограмований)

Таблиця 9.13 – Функціональні можливості фьюзів і їх розташування в молодших байтах фьюзів

Молодший байт фьюзів	Номер біта	Опис	Значення за замовчанням
1	2	3	4
BODLEVEL	7	Рівень спрацювання детектора зниженої напруги живлення	1 (незапрограмований)

Продовження таблиці 9.13

1	2	3	4
BODEN	6	Увімкнення детектора зниженої напруги живлення	1 (незапрограмований, BOD вимкнено)
SUT1	5	Вибір час запуску 1	1 (незапрограмований)
SUT0	4	Вибір час запуску 0	0 (запрограмований)
CKSEL3	3	Вибір джерела тактового сигналу 3	0 (запрограмований)
CKSEL2	2	Вибір джерела тактового сигналу 2	0 (запрограмований)
CKSEL1	1	Вибір джерела тактового сигналу 1	0 (запрограмований)
CKSEL0	0	Вибір джерела тактового сигналу 0	1 (незапрограмований)

Команда Chip Erase не впливає на стан бітів фьюзів. Зауважте, що ці біти заблоковані, якщо запрограмовано біт блокування 1 (LB1). Запрограмуйте фьюз-біти перед програмуванням бітів блокування.

Фіксування фьюзів. Значення фьюзів фіксуються, коли пристрій переходить у режим програмування, і зміни значень фьюзів не матимуть ефекту, доки мікроконтролер не вийде з режиму програмування. Це не стосується фьюза EESAVE, який починає діяти одразу після програмування. Фьюзи також фіксуються під час увімкнення живлення в нормальному режимі.

9.4 Програмування пам'яті за допомогою послідовного інтерфейсу

Як флеш-пам'ять, так і пам'ять EEPROM можна програмувати за допомогою послідовної шини SPI, коли вивід RESET підтягується до GND. Послідовний інтерфейс складається з контактів SCK, MOSI (вхід) і MISO (вихід). Після того, як на RESET встановлено низький рівень, спочатку потрібно виконати інструкцію Programming Enable, перш ніж можна буде виконувати операції програмування / стирання.

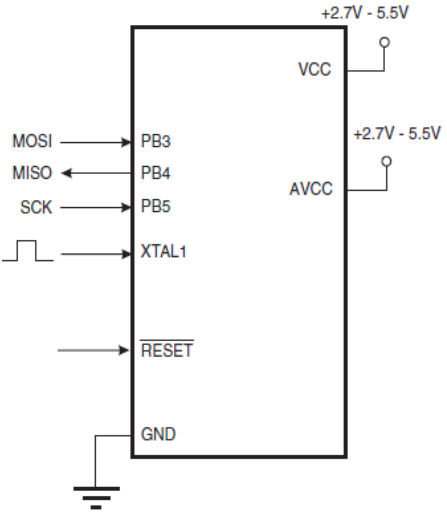


Рисунок 9.14 – Виводи для програмування через SPI

У таблиці 9.14 та на рисунку 9.14 наведено виводи для програмування через SPI.

Таблиця 9.14 – Виводи для програмування через SPI

Назва	Вивід	Вхід/вихід	Опис
MOSI	PB3	Вхід	Вхід послідовних даних
MISO	PB4	Вихід	Вихід послідовних даних
SCK	PB5	Вхід	Тактовий сигнал

Якщо пристрій тактується внутрішнім генератором, немає необхідності підключати джерело тактового сигналу до контакту XTAL1.

Під час програмування EEPROM цикл автоматичного стирання вбудовано в операцію автоматичного програмування (ТІЛЬКИ в послідовному режимі), і немає необхідності спочатку виконувати інструкцію Chip Erase. Операція Chip Erase перетворює вміст кожної комірки пам'яті як флеш, так і EEPROM, на 0xFF.

Залежно від значення фьюзів CKSEL, має бути присутнім правильний тактовий сигнал. Мінімальний низький і високий періоди для входу Serial Clock (SCK) визначаються таким чином:

Низький:> 2 такти ЦП для f_{ck} <12 МГц, 3 такти ЦП для f_{ck} >=12 МГц
Високий:> 2 такти ЦП для f_{ck} <12 МГц, 3 такти ЦП для f_{ck} >=12 МГц

Алгоритм послідовного програмування. Під час запису послідовних даних до ATmega8 дані синхронізуються з наростаючим фронтом SCK. Під час зчитування даних з ATmega8 дані синхронізуються зі спадаючим фронтом SCK (див. рис. 9.15).

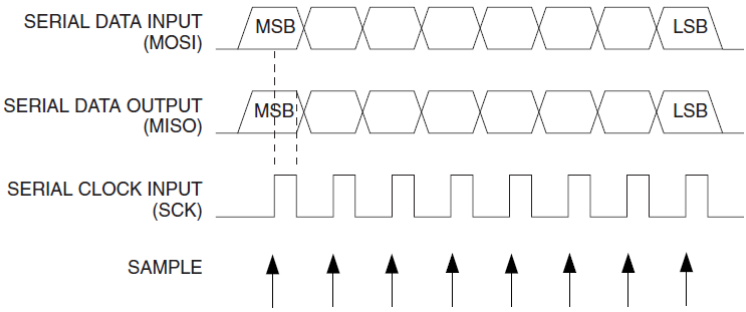


Рисунок 9.15 – Синхронізування даних зі спадаючим фронтом SCK

Для програмування та перевірки ATmega8 у режимі послідовного програмування рекомендується наступна послідовність:

1. Послідовність увімкнення: Подайте живлення між VCC і GND, поки на виводи RESET і SCK подається «0». У деяких системах програматор не може гарантувати, що SCK утримується на низькому рівні під час увімкнення. У цьому випадку на вивід RESET потрібно подати позитивний імпульс тривалістю принаймні два тактових цикли ЦП після того, як SCK було встановлено у «0».

2. Зачекайте принаймні 20 мс і ввімкніть послідовне програмування, надіславши інструкцію Programming Enable на вивід MOSI.

3. Інструкції послідовного програмування не працюватимуть, якщо зв'язок не синхронізований. Після синхронізації другий байт (0x53) відобразиться під час видачі третього байту інструкції "Programming Enable". Незалежно від того, правильний зворотній байт чи ні, всі чотири байти інструкції повинні бути передані. Якщо 0x53 не повертається назад, подайте на вивід RESET позитивний імпульс і видайте нову команду Programming Enable.

4. Флеш-пам'ять програмується по одній сторінці. Сторінка пам'яті завантажується по байтам за допомогою надання 5 молодших бітів адреси та даних разом із інструкцією Load Program memory Page. Щоб забезпечити правильне завантаження сторінки, молодший байт даних має бути завантажений перед тим, як старший байт даних буде передано до даної адреси. Сторінка пам'яті програми зберігається шляхом завантаження інструкції Write Program memory Page з 7 старшими бітами адреси. Якщо опитування пам'яті не використовується, користувач повинен зачекати принаймні 4.5 мс перед тим, як видати наступну сторінку.

Примітка. Якщо до завершення будь-якої операції запису (FLASH, EEPROM, біти блокування, фьюзи) застосовуються інші команди, крім опитування (читання), це може призвести до неправильного програмування.

5. Масив EEPROM програмується по одному байту шляхом надання адреси та даних разом із відповідною інструкцією Write. Комірка пам'яті EEPROM спочатку автоматично стирається перед записом нових даних. Якщо опитування пам'яті не використовується, користувач повинен зачекати принаймні 9 мс перед видачею наступного байту.

6. Будь-яку комірку пам'яті можна перевірити за допомогою інструкції Read, яка повертає вміст за вибраною адресою на послідовний вихід MISO.

7. Наприкінці сеансу програмування вивід RESET можна встановити у високий рівень, щоб розпочати нормальну роботу.

8. Послідовність вимкнення живлення (за потреби):

Встановіть RESET у «1».

Вимкніть живлення VCC.

Опитування флеш пам'яті. Коли сторінка програмується у флеш-пам'ять, читання будь-якої комірки з адреси на сторінці, що програмується, дасть значення 0xFF. Коли пристрій буде готовий до нової сторінки, запрограмоване значення читатиметься правильно. Це використовується, щоб визначити, коли можна записати наступну сторінку. Зауважте, що вся сторінка записується одночасно, і для опитування можна використовувати будь-яку адресу на сторінці. Опитування флеш-пам'яті не працюватиме для

значення 0xFF, тому під час програмування цього значення користувачеві доведеться чекати принаймні 4,5 мс перед програмуванням наступної сторінки. Оскільки пристрій зі стертими мікросхемами містить 0xFF у всіх комірках, програмування адрес, які мають містити 0xFF, можна пропустити.

Опитування пам'яті EEPROM. Коли новий байт був записаний і запрограмований в EEPROM, зчитування запрограмованої адреси дасть значення 0xFF. Коли пристрій буде готовий до нового байту, запрограмоване значення буде читатися правильно. Це використовується для визначення того, коли можна записати наступний байт. Це не працюватиме для значення 0xFF, але користувачеві слід пам'ятати наступне: оскільки пристрій із стертою пам'яттю містить 0xFF у всіх комірках, програмування адрес, які мають містити 0xFF, можна пропустити. Це не стосується випадку, коли EEPROM було перепрограмовано без стирання мікросхеми пристрою. У цьому випадку опитування даних не можна використовувати для значення 0xFF, і користувачеві доведеться чекати принаймні 9 мс перед програмуванням наступного байту.

Контрольні запитання до теми 9

1. Як організована пам'ять EEPROM у мікроконтролері ATmega8?
2. Яка є витривалість пам'яті EEPROM і чому це важливо для її ефективного використання?
3. Які регістри використовуються для доступу до пам'яті EEPROM, і яким чином вони конфігуруються?
4. Яким чином здійснюється запис в пам'ять EEPROM у мікроконтролері ATmega8? Покажіть приклад коду.
5. Як здійснюється читання з пам'яті EEPROM, і чому важливо перевіряти статус готовності перед операцією читання?
6. Які є основні процедури та умови для запобігання пошкодження даних у EEPROM в ATmega8?
7. Як впливає знижена напруга живлення на операції запису та читання в EEPROM, і які заходи слід прийняти для її захисту?
8. Які переваги надає функція самопрограмування пам'яті мікроконтролера?
9. Як завантажувач дозволяє гнучке оновлення прикладного програмного забезпечення?

10. Яким чином можна налаштувати розмір завантажувальної пам'яті за допомогою фьюз-бітів?
11. Які є рівні захисту завантажувача і як вони встановлюються?
12. Що таке розділ «читання під час запису» і «без читання під час запису» у флеш-пам'яті? Яка в них основна відмінність?
13. Як вибирається розділ, до якого звертається вказівник Z під час програмування?
14. Які можливості мають біти блокування завантажувача і як вони використовуються для захисту пам'яті?
15. Як відбувається вхід до програми завантажувача і які можливості надає фьюз BOOTRST?
16. Які функції виконують біти блокування пам'яті програм та даних у мікроконтролері ATmega8?
17. Які є можливості і обмеження щодо стирання та програмування бітів блокування пам'яті в ATmega8?
18. Як впливає біт LB1 на можливості програмування фьюзів у мікроконтролері ATmega8?
19. Які функції виконують старший і молодший байти фьюзів в ATmega8?
20. Які механізми фіксації фьюзів існують у мікроконтролері ATmega8 і як вони впливають на налаштування пристрою?
21. Які основні кроки для підготовки до програмування пам'яті через SPI у мікроконтролері ATmega8?
22. Як функціонує цикл автоматичного стирання EEPROM під час послідовного програмування в ATmega8?
23. Як впливає налаштування фьюзів CKSEL на процес програмування через SPI в ATmega8?
24. Яким чином синхронізуються дані під час запису та зчитування через послідовний інтерфейс у мікроконтролері ATmega8?
25. Які особливості програмування флеш-пам'яті та EEPROM у режимі послідовного програмування потрібно враховувати для забезпечення коректності операцій?

Використана література

1. Atmel: ATMega8, ATmega8L : технічна документація на мікроконтролер. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2486-8-bit-AVR-microcontroller-ATmega8_L_datasheet.pdf

10

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

Перелік питань до розділу:

10.1. Завдання.
10.2. Теоретичні дані.
10.2.1. Призначення програми SimulIDE.
10.2.2. Встановлення необхідних програм та засобів.
10.2.3. Подання на екрані та основні елементи керування.
10.2.4. Основні прийоми створення та коригування схеми.
10.2.5. Написання програмного коду.
10.2.6. Компіляція файлу прошивки та запуск симуляції.
10.3. Завдання до практичної роботи.

10.1 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

10.2 Теоретичні дані

10.2.1 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.

10.2.2 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. 10.1).

Після вибору версії (для прикладу була обрана версія для Windows 64), відкривається наступне вікно (рис. 10.2).

Рисунок 10.1 – Завантаження SimulIDE

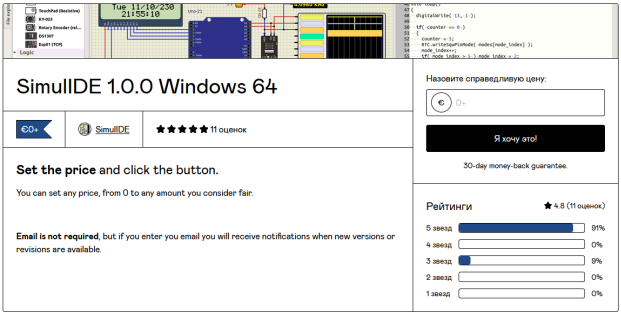


Рисунок 10.2 – Придбання SimulIDE

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

У наступному вікні (рис. 10.3) треба ввести вашу електронну адресу і натиснути на кнопку «Получить».

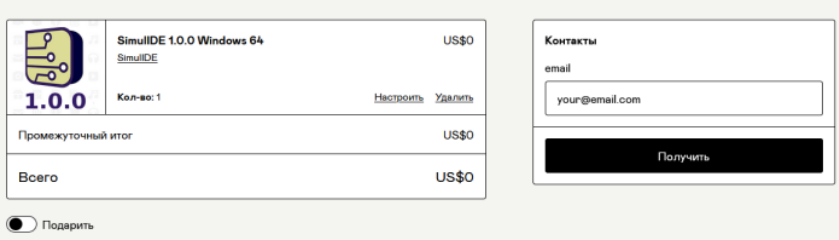


Рисунок 10.3 – Отримання SimulIDE

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. 10.4.

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. 10.5).

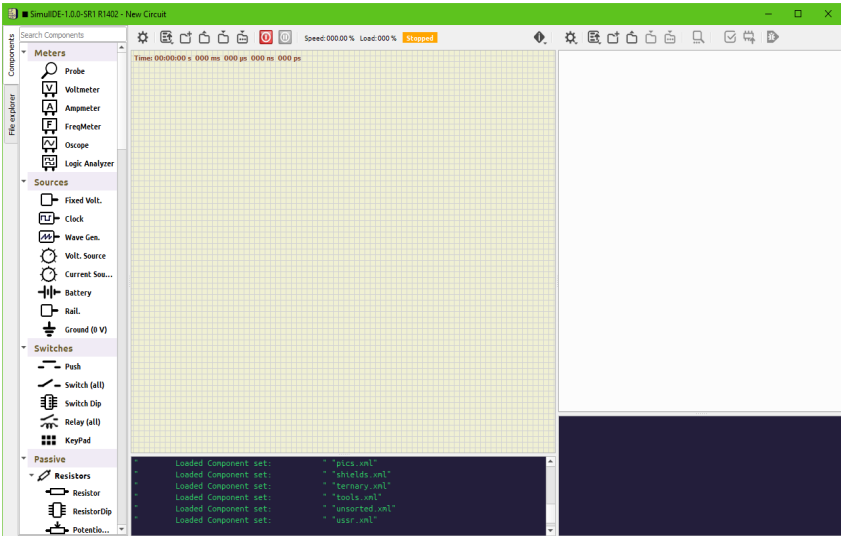


Рисунок 10.4 – Інтерфейс програми SimulIDE

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 10.5 – 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 на момент написання цього керівництва має для Windows 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.

10.2.3 Подання на екрані та основні елементи керування

Інтерфейс програми SimulIDE більш простий та інтуїтивно-зрозумілий порівняно з деякими іншими програмами (рис. 10.6). У ньому є такі основні елементи.

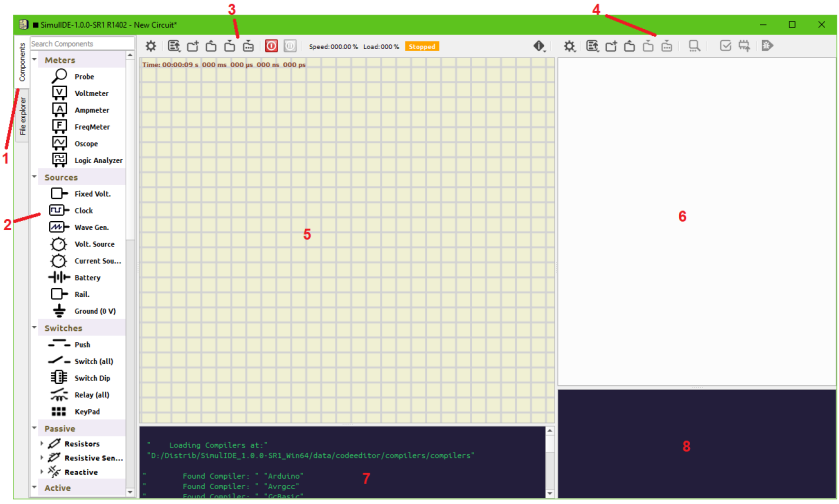


Рисунок 10.6 – Інтерфейс програми SimulIDE

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. 10.1).

4 – Панель керування програмним кодом, на якій знаходяться наступні кнопки (табл. 10.2).

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 10.1 – 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

Таблиця 10.2 – Кнопки панелі керування програмним кодом програми SimulIDE

Кнопка	Призначення
1	2
	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

Продовження таблиці 10.2

1	2
	Зберегти поточну програму з новим ім’ям
	Знайти або замінити текст у програмі
	Створити файл прошивки
	Завантажити файл прошивки у мікроконтролер
	Запустити налагодження програми

10.2.4 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. 10.7).

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. 10.8).

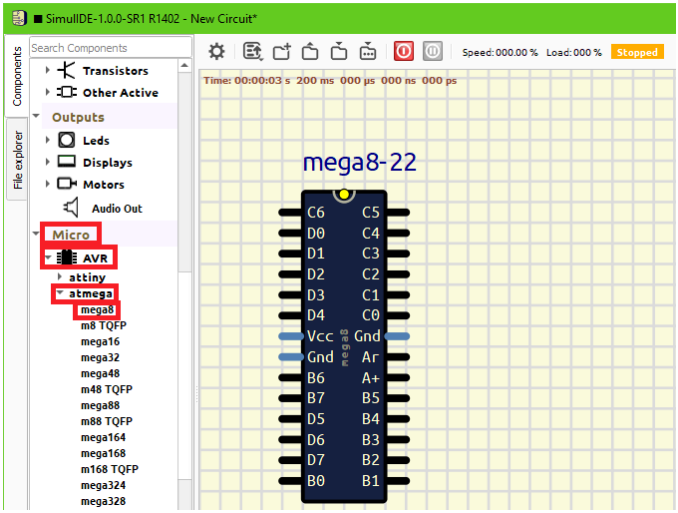


Рисунок 10.7 – Додавання мікроконтролера ATMega8 до схеми

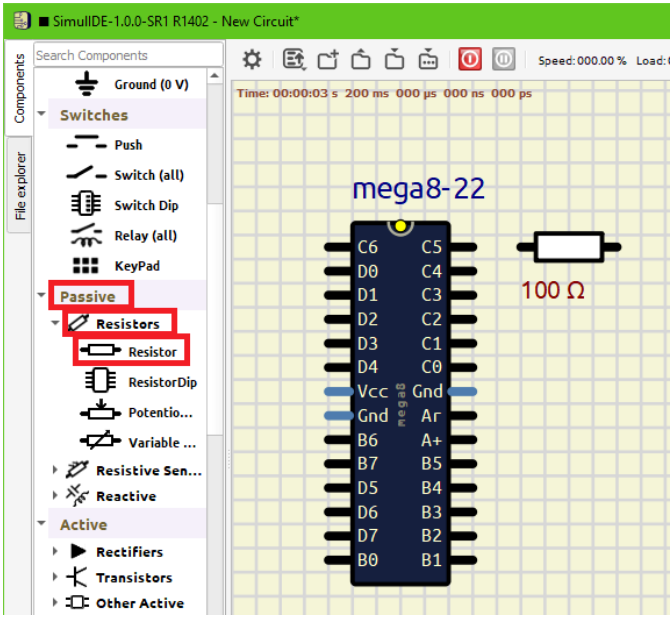


Рисунок 10.8 – Додавання резистора до схеми

Аналогічно, розкриємо підзаголовок “Outputs” – “LEDs”, та витягнемо до схеми компонент “LED” (рис. 10.9).

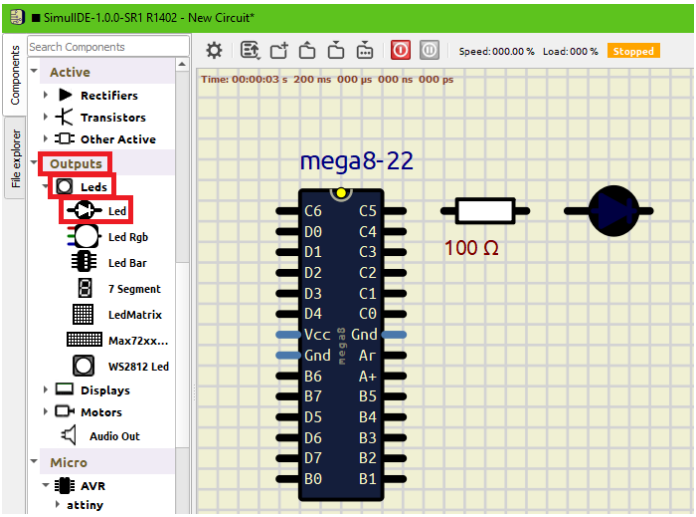


Рисунок 10.9 – Додавання світлодіоду до схеми

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. 10.10).

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. 10.11).

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

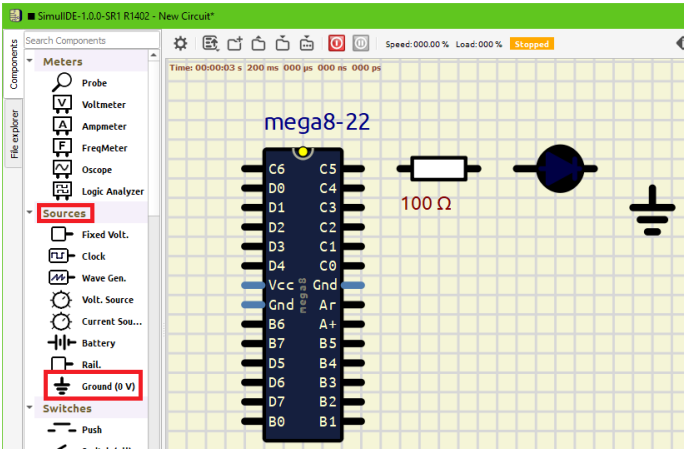


Рисунок 10.10 – Додавання землі до схеми

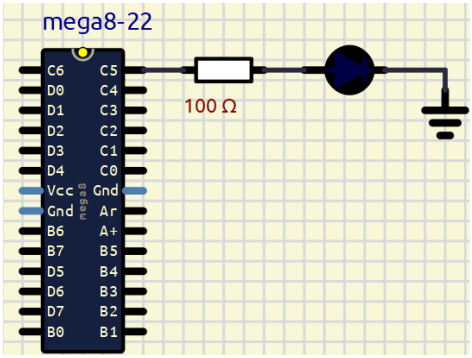


Рисунок 10.11 – З'єднана схема

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. 10.11.

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. 10.11) 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. 10.12).

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.

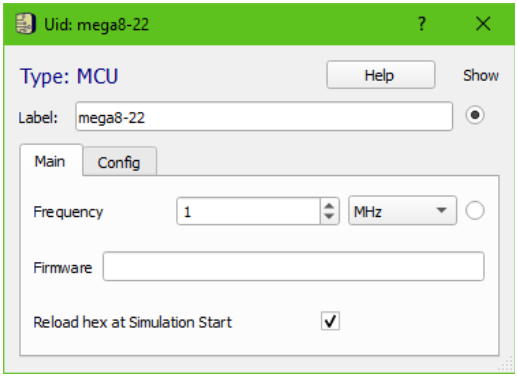


Рисунок 10.12 – Вікно налаштувань мікроконтролера

– 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. 10.13) 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.

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

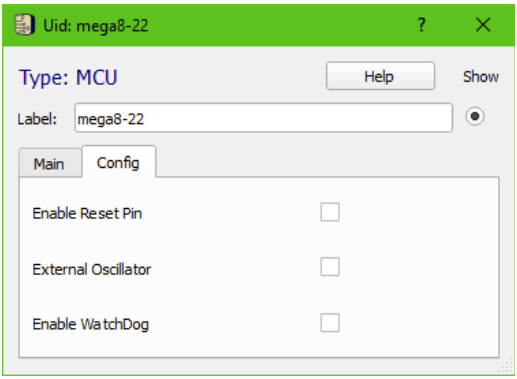


Рисунок 10.13 – Вікно налаштувань мікроконтролера

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 світлодіоду виглядає наступним чином (рис. 10.14).

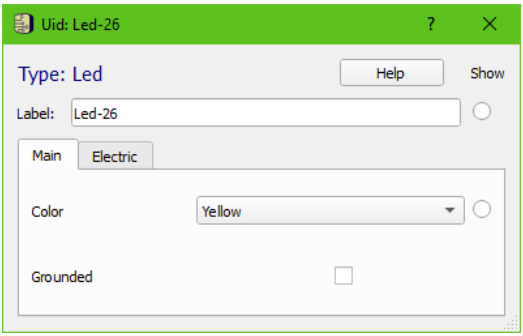


Рисунок 10.14 – Налаштування світлодіоду

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

На вкладці "Main" розташовані такі опції:

"Color" – колір світлодіоду. Можна обрати один з шести запропонованих варіантів. За замовчанням стоїть жовтий колір.

"Grounded" – якщо встановити цю опцію, то катод світлодіоду 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. 10.15):

– "Forward Voltage" – падіння напруги на світлодіоді у відкритому стані.

– "Max Current" – максимальний струм через світлодіод.

– "Resistance" – опір світлодіоду 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. 10.16.

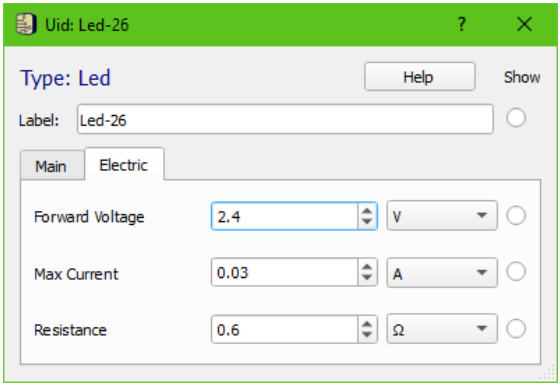


Рисунок 10.15 – Налаштування світлодіоду

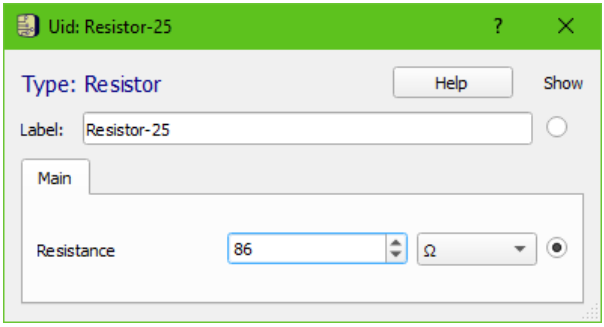


Рисунок 10.16 – Налаштування резистора

Як бачимо, у нього є лише один параметр “Resistance”, який задає опір резистора.

Давайте порахуємо цей опір базуючись на знаннях параметрів світлодіоду (рис. 10.15).

Мікроконтролер видає на своїх виводах близько 5 В, що відповідає логічній одиниці. Падіння напруги на світлодіоді складає 2.4 В (рис. 10.15). 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. 10.15), то опір резистора повинен бути 86.6 – 0.6 = 86 Ом, що ми й встановили на рис. 10.16.

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

10.2.5 Написання програмного коду

Щоб почати писати програму, треба у панелі задач (4 на рис. 10.6) натиснути на кнопку , а потім у полі 6 (рис. 10.6) написати наступний код (рис. 10.17).

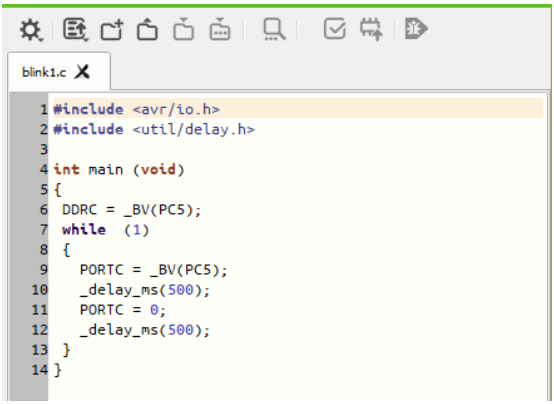


Рисунок 10.17 – Програмний код для блимання світлодіодом з частотою 1 Гц

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:

– 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. 10.17). 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: 001000002. 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 світлодіоду.

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 світлодіоду 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 світлодіоду.

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. 10.18).

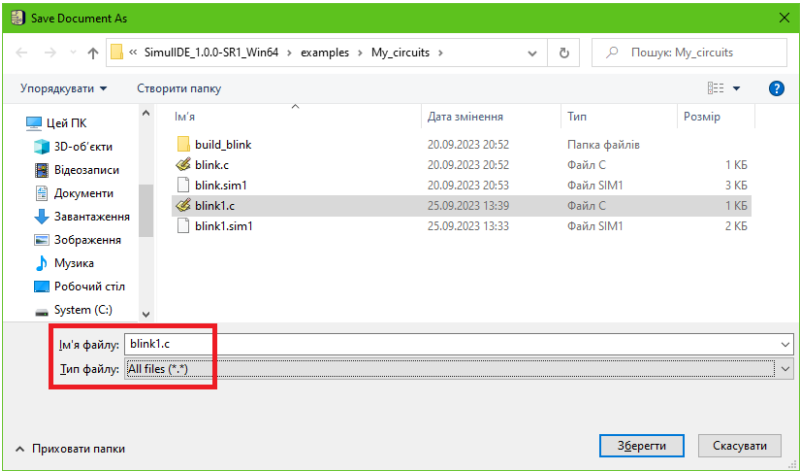


Рисунок 10.18 – Збереження файлу програми

10.2.6 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. 10.19).

У відкритому вікні треба обрати зі списку компілятор (“Compiler”) типу “Avrgcc” (рис. 10.20).

Тепер треба вказати шляхи, де встановлений власне компілятор “Tool Path” та де знаходяться заголовні файли “Include Path” (рис. 10.21).

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/».

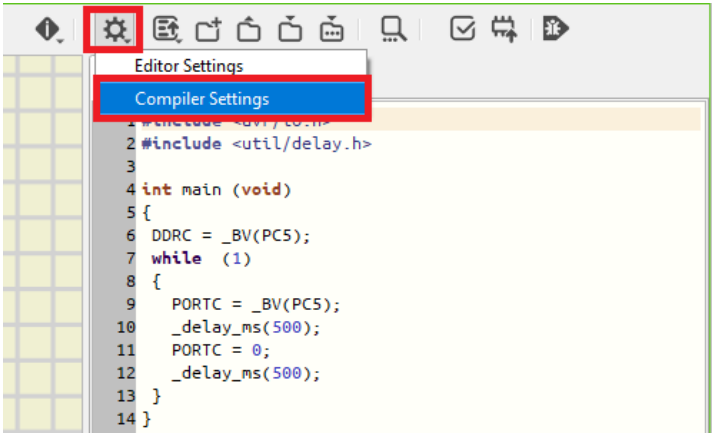


Рисунок 10.19 – Відкриття вікна налаштувань компілятора

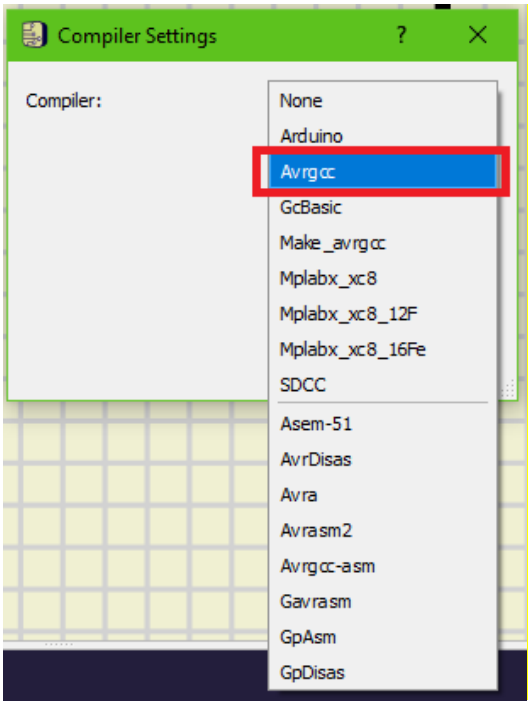


Рисунок 10.20 – Вибір компілятора

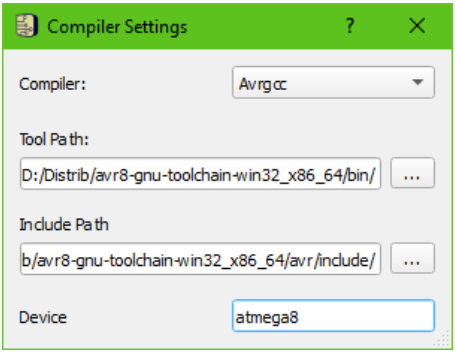


Рисунок 10.21 – Налаштування компілятора

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. 10.21).

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. 10.6) ви повинні побачити інформацію про успішне створення файлу «.hex” (рис. 10.22).

```
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 10.22 – Інформація про успішне створення файлу прошивки

Також на рис. 10.22 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. 10.12).

Тепер натискаємо на кнопку , щоб загрузити створений файл у мікроконтролер. У вікні 8 бачимо інформацію, що файл завантажено успішно (рис. 10.23).

```
FirmWare Uploaded to mega8
D:/Distrib/SimulIDE_1.0.0-SR1_Win64/examples/My_circuits/build_blink1/blink1.hex


Searching for variables... 0 variables found

Mapping Flash to Source... 1 lines mapped
```

Рисунок 10.23 – 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. 10.24).

На цьому завдання поточної практичної роботи можна вважати виконаним.

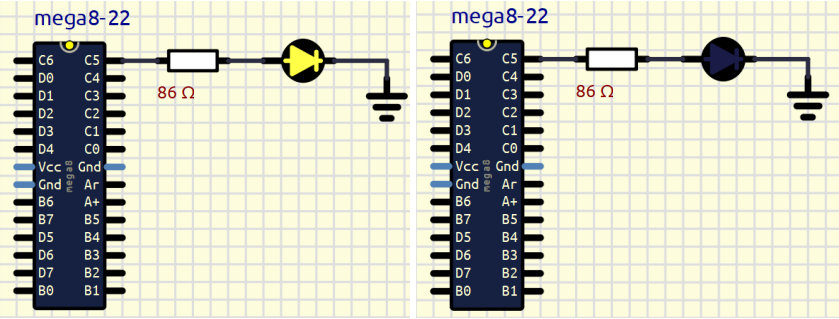


Рисунок 10.24 – Світлодіод у ввімкненому та вимкненому стані

10.3 Завдання до практичної роботи

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

Ознайомитись з поданням програми SimulIDE та її інтерфейсом.

Створити принципову електричну схему, схожу на рис. 10.11, яка включає мікроконтролер, обраний згідно з варіантом завдань (табл. 10.3), резистор та світлодіод. Останній підключити до виводу, також вказаному в таблиці 10.3. Змінити колір світлодіоду відповідно до таблиці 3. У другому стовпчику таблиці 10.3 у дужках вказано справжню назву мікроконтролеру, яку треба вводити у налаштуваннях компілятора (рис. 10.21).

Написати програму блимання світлодіодом з частотою, вказаною в таблиці 10.3.

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

Таблиця 10.3 – Варіанти завдань до практичної роботи № 1

Номер варіанта	Мікроконтролер У SimulIDE (у компіляторі)	Вивід для підключення світлодіоду	Частота блимання, Гц	Колір світлодіоду
1	2	3	4	5
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

Продовження таблиці 10.3

1	2	3	4	5
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

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.

Перелік рекомендованих джерел

1. SimulIDE Tutorial. URL: https://simulide.com/p/simulidekb/ (дата звернення: 02.12.2024).
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: 02.12.2024).
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: 02.12.2024).

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

Перелік питань до розділу:

11.1. Завдання.
11.2. Теоретичні дані.
11.2.1. Робота з портами вводу-виводу мікроконтролерів AVR.
11.2.2. Бітові операції на мові С.
11.2.3. Бітові операції.
11.2.4. Очищення та встановлення бітів.
11.2.5. Макрос керування значенням біту _BV().
11.2.6. Перевірка значення біту.
11.2.7. Приклад програми.
11.2.8. Принципова електрична схема.
11.2.9. Програмний код.
11.3. Завдання до практичної роботи.

11.1 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.

11.2 Теоретичні дані

11.2.1 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 рис. 11.1.

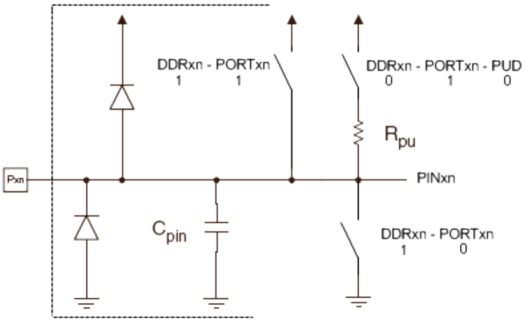


Рисунок 11.1 – Схема порту вводу-виводу

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 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.

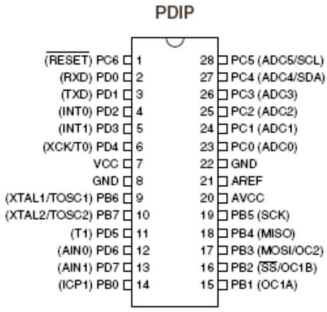


Рисунок 11.2 – Мікроконтролер Atmega8

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

З рис. 11.2 видно, що в МК Mega8 три порти вводу-виводу. Два повні порти (по 8 біт) – "PB" і "PD" і один неповний (7 біт) – "PC".

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 11.1.

Таблиця 11.1 – Варіанти установок бітів регістрів вводу-виводу

DDxn	PORTxn	PUD (у SFIOR)	Стан виводу	Підтягування	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.

11.2.2 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 – Least Significant Bit) 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.

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

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
uint8_t b=0x0F; // шістнадцяткове
uint8_t c=0b00001111; // двійкове
```

uint8_t – один з типів даних рівноширокого цілого типу, зі стандарту С99. Стандарт передбачає 8-бітний беззнаковий цілий тип. Типи даних стандарту С99 будуть і надалі використовуватись у цьому курсі.

11.2.3 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 С 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

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

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 С 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 С 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 операнду, а не двох, як інші. NOT просто перетворює кожен біт на протилежний. 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
```

11.2.4 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.

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

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

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
```

11.2.5 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

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

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.

11.2.6 Перевірка значення біту[6]

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

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.

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

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.

11.2.7 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.

11.2.8 Принципова електрична схема

Принципова електрична схема, що реалізує поставлену задачу, показана на рис. 11.3.

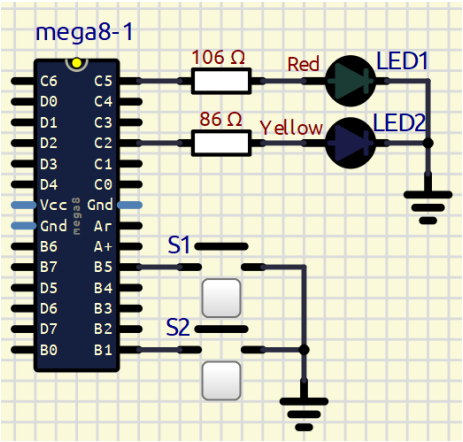


Рисунок 11.3 – Принципова електрична схема

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. 11.4).

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.

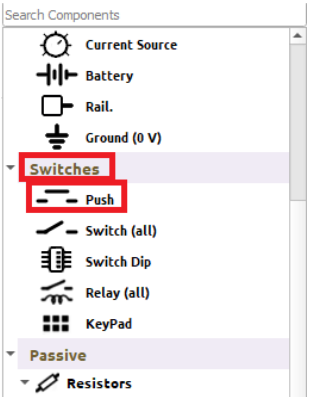


Рисунок 11.4 – Кнопка у бібліотеці компонентів

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. 11.1). 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. 11.3.

11.2.9 Програмний код

Розглянемо тепер програму, що реалізує поставлену задачу (рис. 11.5).

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

```c
#include <avr/io.h>
#include <util/delay.h>

int main (void)
{
  DDRC |= _BV(PC2) | _BV(PC5);
  DDRB &= ~(_BV(PB1) | _BV(PB5));
  PORTB |= _BV(PB1) | _BV(PB5);
  while  (1)
  {
    if (!(PINB & _BV(PB1)))
    {
      _delay_ms(30);
      if (!(PINB & _BV(PB1)))
      {
        while (!(PINB & _BV(PB1)));
        _delay_ms(30);
        if (PINB & _BV(PB1))
        {
          PORTC ^= _BV(PC2);
        }
      }
    }
    if (!(PINB & _BV(PB5)))
    {
      _delay_ms(30);
      if (!(PINB & _BV(PB5)))
      {
        while (!(PINB & _BV(PB5)))
        {
          PORTC ^= _BV(PC5);
          _delay_ms(250);
        }
        _delay_ms(30);
        if (PINB & _BV(PB1))
        {
          PORTC &= ~_BV(PC5);
        }
      }
    }
  }
}
```

Рисунок 11.5 – Програмний код

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. 11.2.4**.

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.

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. 11.2.2). 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.

11.3 Завдання до практичної роботи

1. Створити принципову електричну схему, згідно з варіантом завдань (таблиця 11.2).
2. Написати програму, що виконує поставлене завдання (табл. 11.2).
3. Створити файл прошивки, загрузити його у мікроконтролер та переконатися, що все працює, як треба.

Таблиця 11.2 – Варіанти завдань до практичної роботи № 2

Номер варіанта	Мікроконтролер У SimulIDE (у компіляторі)	Завдання
1	2	3
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. 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 6 світлодіод LED2 і знову почати блимати світлодіодом LED1. Процес повторювати циклічно.

Продовження таблиці 11.2

1	2	3
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 світлодіоду 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).

Продовження таблиці 11.2

1	2	3
10	m48 TQFP (atmega48)	Підключити кнопку S1 до виводу PD6, кнопку S2 до виводу PD3, а світлодіод до виводу PB7. При 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.

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?

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. 57 с.

2. Основи Програмування AVR C. DevZone. URL: https://devzone.org.ua/post/osnovy-prohramuvannia-avr-c (дата звернення: 02.12.2024).
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: 02.12.2024).
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: 02.1 2.2024).

12 ПРАКТИЧНА РОБОТА № 3 «ПРОГРАМУВАННЯ ТАЙМЕРІВ МІКРОКОНТРОЛЕРА AVR»

Перелік питань до розділу:

12.1. Завдання.
12.2. Теоретичні дані.
12.2.1. Переривання.
12.2.2. 8-бітний таймер-лічильник 0 мікроконтролера ATMega8.
12.3. Приклад програми.
12.3.1. Принципова електрична схема.
12.3.2. Програмний код.
12.4. Завдання до практичної роботи.

12.1 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.

12.2 Теоретичні дані

12.2.1 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 вектору 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 вектору 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 збережені, 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 вектору. 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 A таймера/лічильника 1	17	0x0010	ANA_COMP	Аналоговий компаратор
8	0x0007	TIMER1 COMPB	Збіг при порівнянні B 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 12.1 – 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 увімкнено 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 °С. 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. 12.2).

Bit	7	6	5	4	3	2	1	0	
	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
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 12.2 – 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 12.1.

Таблиця 12.1 – 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

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.

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. 12.3).**

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 12.3 – 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 увімкнено. 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 вектору 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 увімкнено. 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 вектору переривання INT0.

Загальний регістр прапорів переривань – GIFR (рис. 12.4).

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 12.4 – 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.

12.2.2 8-бітний таймер-лічильник 0 мікроконтролера ATMega8

Таймер-лічильник 0 – 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 12.5.

Регістри. Таймер-лічильник (TCNT0) є 8-розрядним регістром. Усі сигнали запиту на переривання (скорочено Int. Req. на рис. 12.5) відображаються в регістрі прапорів переривання таймера (TIFR). Всі переривання індивідуально маскуються за допомогою регістра маски переривання таймера (TIMSK). TIFR і TIMSK не показані на рис. 12.5, 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 12.2 також широко використовуються далі.

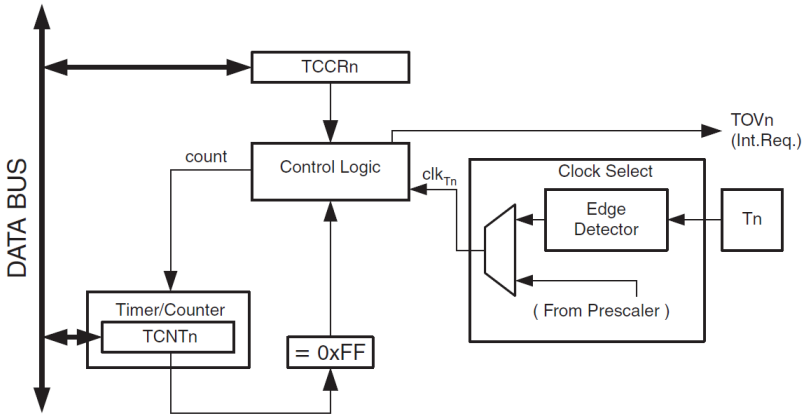


Рисунок 12.5 – Спрощена блок-схема 8-розрядного таймера-лічильника

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 12.2 – Визначення

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

Лічильник. Основною частиною 8-розрядного таймера-лічильника є блок програмованого лічильника. На рис. 12.6 показана 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.

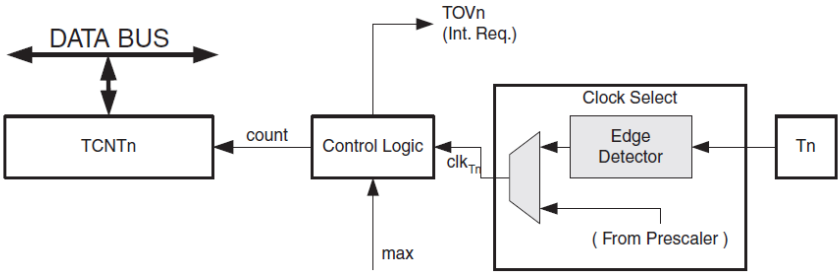


Рисунок 12.6 – Блок-схема лічильника та його оточення

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. Рис. 12.7 містить дані тактування при нормальній роботі таймера-лічильника. Показана послідовність рахунку, близька до максимального значення.

На рис. 12.8 показані ті самі дані про тактування, але з увімкненим попереднім дільником ($f_{clkI/O}/8$).

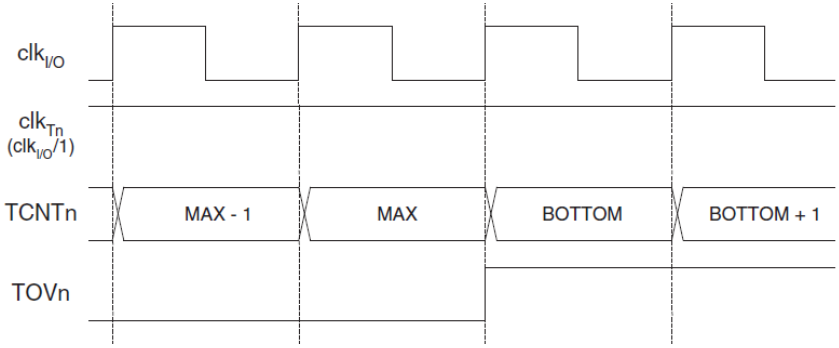


Рисунок 12.7 – Дані тактування при нормальній роботі таймера-лічильника

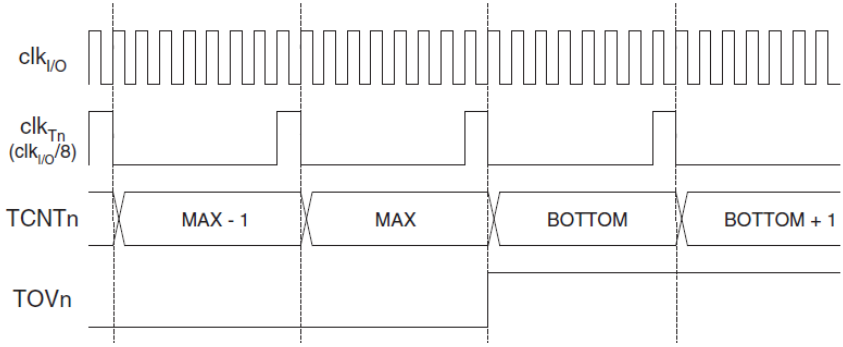


Рисунок 12.8 – Дані про тактування з увімкненим попереднім дільником

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

Регістр керування таймером-лічильником – TCCR0 зображено на рис. 12.9.

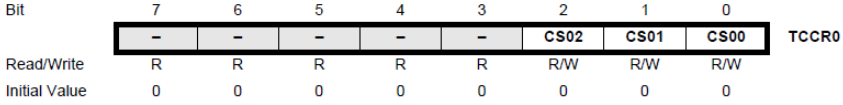


Рисунок 12.9 – Регістр керування таймером-лічильником – TCCR0

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 12.3.

Таблиця 12.3 – 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. 12.10.

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 12.11.

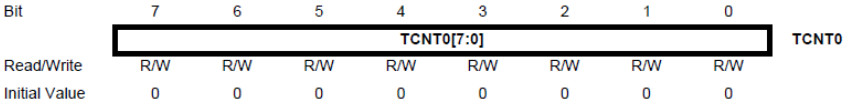


Рисунок 12.10 – Регістр таймера-лічильника – TCNT0

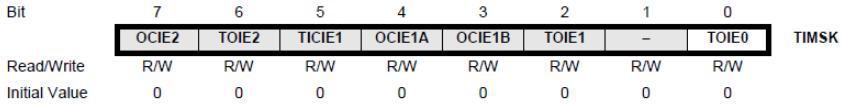


Рисунок 12.11 – Регістр маски переривання таймерів-лічильників – TIMSK

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 зображено на рис. 12.12.

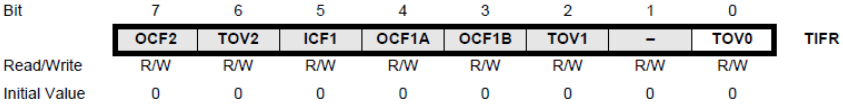


Рисунок 12.12 – Регістр прапорів переривання таймерів-лічильників – TIFR

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 вектору 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.

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 рис. 12.13.

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 12.13, оскільки ці регістри спільно використовуються іншими блоками таймерів.

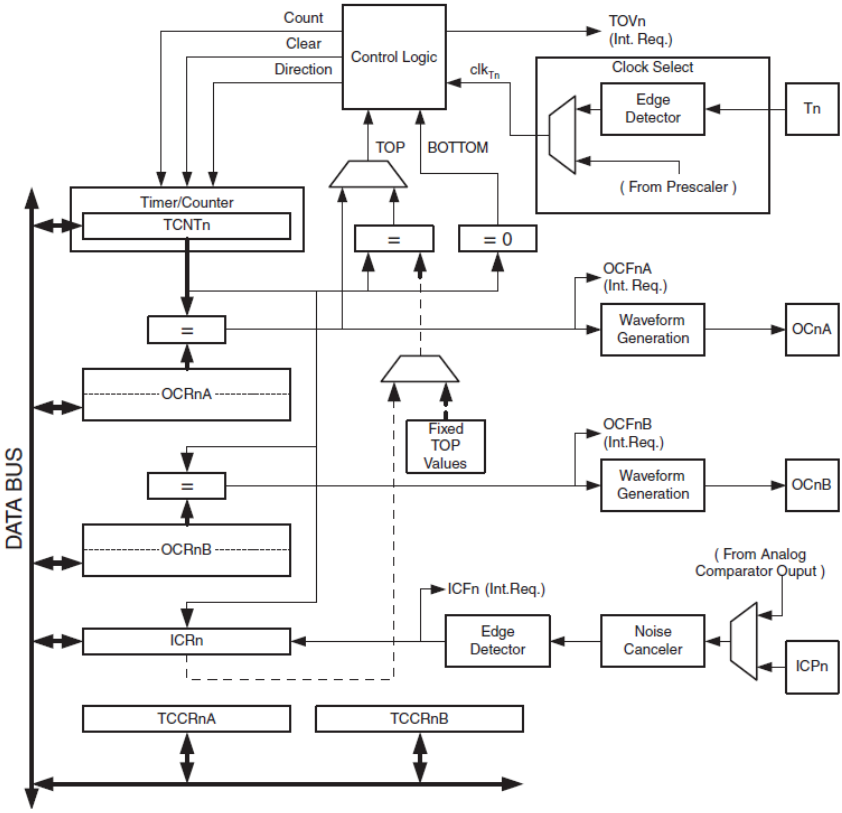


Рисунок 12.13 – Спрощена блок-схема 16-розрядного таймера-лічильника 1

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 12.4).

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 12.14 показана блок-схема лічильника та його оточення.

Таблиця 12.4 – Визначення

Визначення	Значення
Нижнє значення, BOTTOM	Лічильник досягає **нижнього значення**, коли стає 0x0000
Максимум, MAX	Лічильник досягає свого **максимуму**, коли він стає 0xFFFF (десяткове число 65535)
Верхнє значення, TOP	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

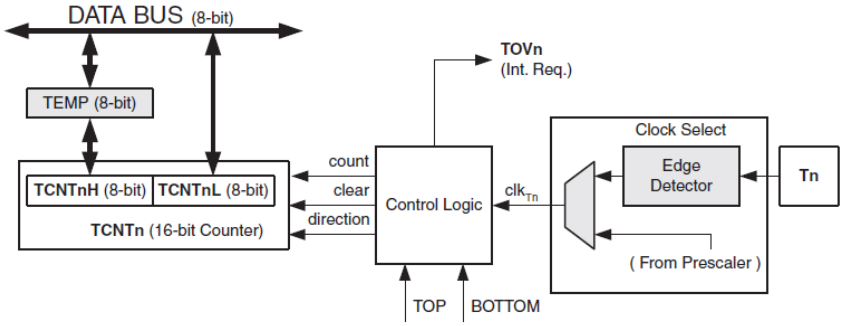


Рисунок 12.14 – Блок-схема лічильника та його оточення

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);

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 байту (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 увімкнено (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 А 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 рис. 12.15 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.

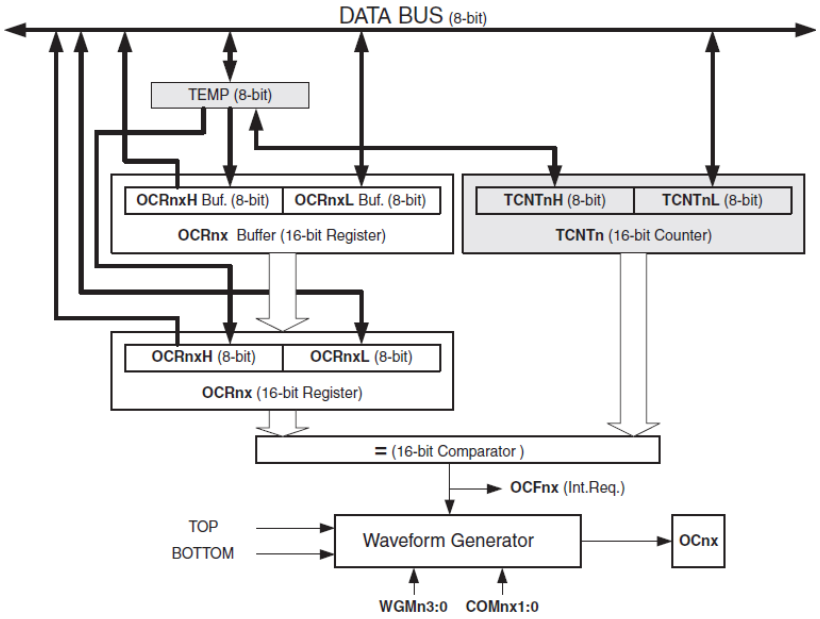


Рисунок 12.15 – Блок-схема блоку порівняння

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.

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 байту (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 байту 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 рис. 12.16 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».

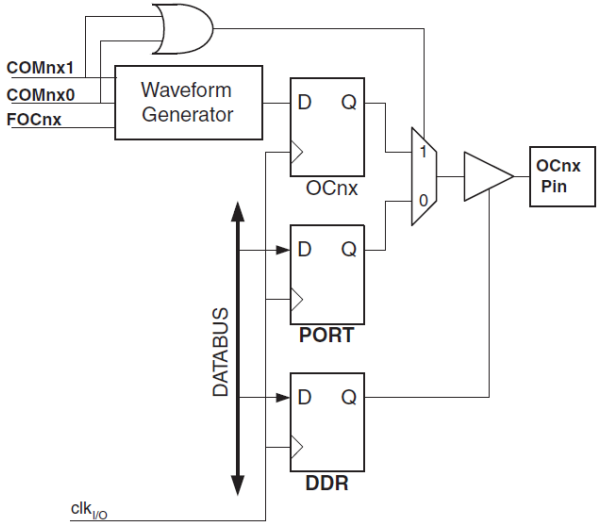


Рисунок 12.16 – Спрощена схема логіки блоку збігу

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 (ОТЗ). У режимі очищення таймеру при збігу (WGM13:0 = 4 або 12) регістри OCR1A або ICR1 використовуються для керування роздільною здатністю лічильника. У режимі ОТЗ 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 ОТЗ показана на рис. 12.17. Значення лічильника (TCNT1) збільшується, доки не відбудеться збіг зі значенням OCR1A або ICR1, а потім лічильник (TCNT1) очищується.

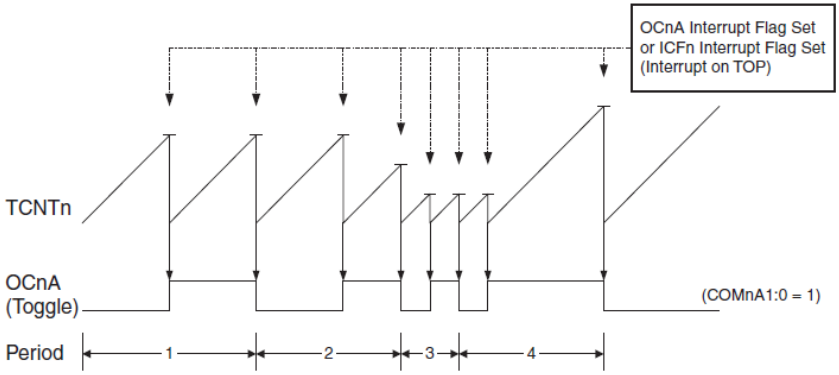


Рисунок 12.17 – Часова діаграма для режиму ОТЗ

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 ОТЗ 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 ОТЗ 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:

$$f_{OCnA} = \frac{f_{CLK\_I/O}}{2 \cdot N \cdot (1 + OCRnA)}. \qquad (12.1)$$

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:

$$R_{ШШІМ} = \frac{log\left(верхнє\ значення + 1\right)}{log\left(2\right)}. \qquad (12.2)$$

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 рис. 12.18. 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.

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.

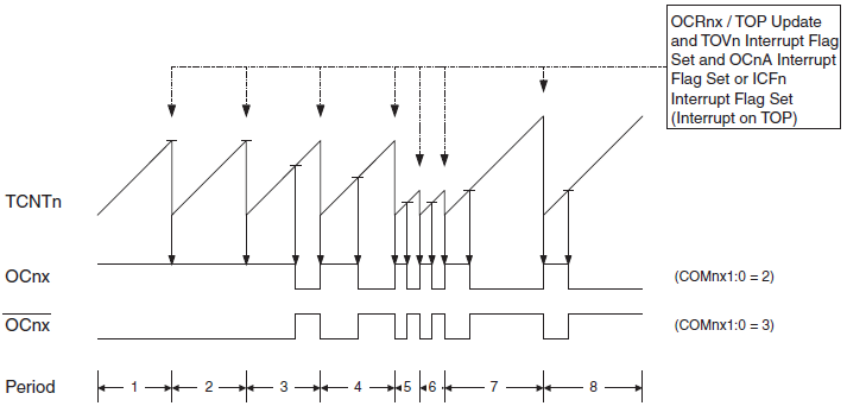


Рисунок 12.18 – Часова діаграма для режиму швидкої ШІМ

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:

$$f_{OCnxШІМ} = \frac{f_{CLK\_I/O}}{N \cdot (1 + верхнє\ значення)}. \qquad (12.3)$$

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 ОТЗ, 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 рис. 12.19.

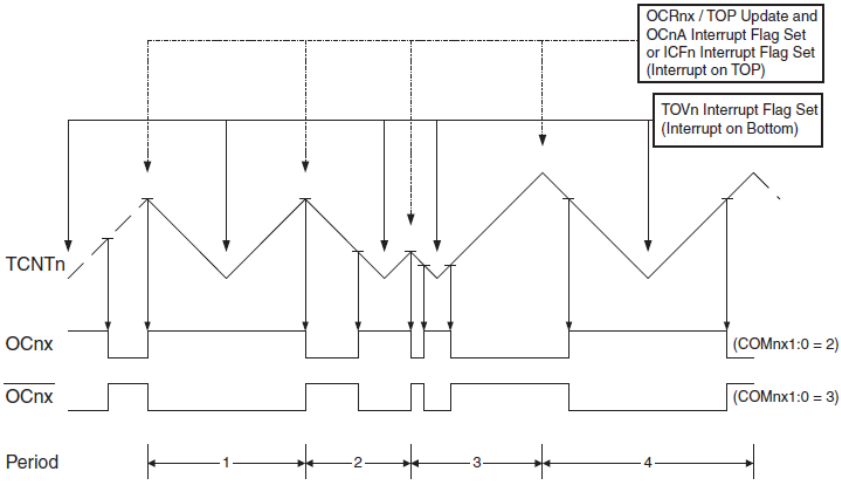


Рисунок 12.19 – Часова діаграма для фазо-коректного режиму ШІМ

На рис. 12.19 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.

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 (рис. 12.20).

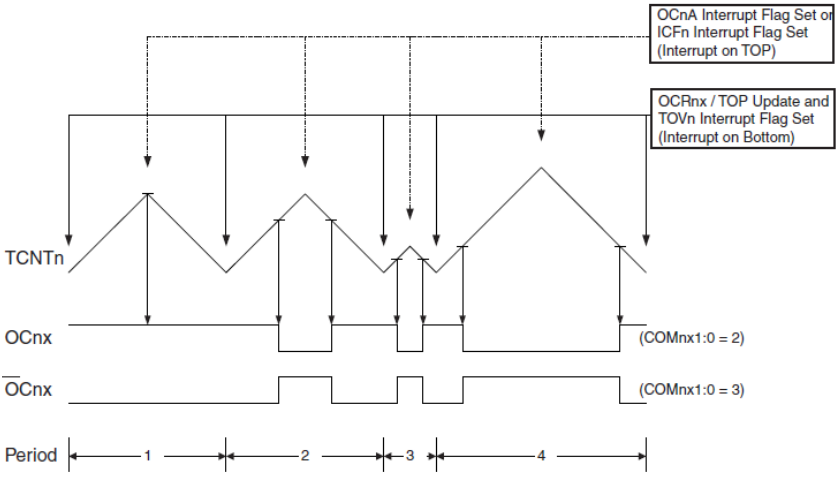


Рисунок 12.20 – Часова діаграма фазо- та частото-коректного ШІМ

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

Регістр керування А таймера-лічильника 1 – TCCR1A зображено на рис. 12.21

Біти 7:6 – COM1A1:0: режим порівняння для каналу А.

Біти 5:4 – COM1B1:0: режим порівняння для каналу В

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.

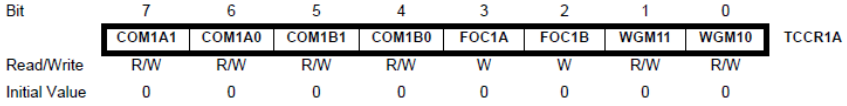


Рисунок 12.21 – Регістр керування А таймера-лічильника 1

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 12.5 показує функціональні можливості бітів COM1x1:0, коли біти WGM13:0 встановлено, як звичайний режим або режим ОТЗ (не ШІМ).

Таблиця 12.5 – Функціональні можливості бітів COM1x1:0, коли біти WGM13:0 встановлено, як звичайний режим або режим ОТЗ (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 12.6 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 12.7 показує функціональність бітів COM1x1:0, коли біти WGM13:0 встановлені, як фазо-коректний, або фазо- та частото-коректний ШІМ.

Таблиця 12.6 – 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
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 12.7 – 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 ОТЗ, використовуючи OCR1A як верхнє значення.

Біти FOC1A/FOC1B завжди читаються як нульові.

Біти 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 (ОТЗ) і три типи режимів широтно-імпульсної модуляції (ШІМ). Відповідність між значеннями бітів WGM13:0 та режимами роботи показана у таблиці 12.8.

Таблиця 12.8 – 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
1	2	3	4	5	6	7	8	9
0	0	0	0	0	Звичайний	0xFFFF	Одразу	Максимальному значенню
1	0	0	0	1	Фазо-коректний ШІМ (8-бит)	0x00FF	Верхнє значення	Нижньому значенню
2	0	0	1	0	Фазо-коректний ШІМ (9-бит)	0x01FF	Верхнє значення	Нижньому значенню
3	0	0	1	1	Фазо-коректний ШІМ (10-бит)	0x03FF	Верхнє значення	Нижньому значенню
4	0	1	0	0	ОТЗ	OCR1A	Одразу	Максимальному значенню
5	0	1	0	1	Швидкий ШІМ (8-бит)	0x00FF	Нижнє значення	Верхньому значенню
6	0	1	1	0	Швидкий ШІМ (9-бит)	0x01FF	Нижнє значення	Верхньому значенню

Продовження таблиці 12.8

1	2	3	4	5	6	7	8	9
7	0	1	1	1	Швидкий ШІМ (10-бит)	0x03FF	Нижнє значення	Верхньому значенню
8	1	0	0	0	Фазо- та часто-коректний ШІМ	ICR1	Нижнє значення	Нижньому значенню
9	1	0	0	1	Фазо- та часто-коректний ШІМ	OCR1A	Нижнє значення	Нижньому значенню
10	1	0	1	0	Фазо-коректний ШІМ	ICR1	Верхнє значення	Нижньому значенню
11	1	0	1	1	Фазо-коректний ШІМ	OCR1A	Верхнє значення	Нижньому значенню
12	1	1	0	0	ОТЗ	ICR1	Одразу	Максимальному значенню
13	1	1	0	1	Зарезервовано	-	-	-
14	1	1	1	0	Швидкий ШІМ	ICR1	Нижнє значення	Верхньому значенню
15	1	1	1	1	Швидкий ШІМ	OCR1A	Нижнє значення	Верхньому значенню

Регістр керування B таймера-лічильника 1 – TCCR1B зображено на рис. 12.22.

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 12.22 – 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 увімкнений.

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. 12.9). Ця функція дозволяє програмно контролювати рахунок.

Таблиця 12.9 – Налаштування таймера-лічильника в залежності від бітів

CS12	CS11	CS10	Опис
1	2	3	4
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 зображено на рис. 12.23.

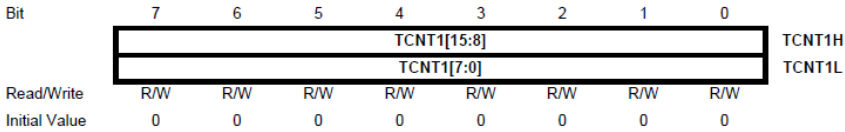


Рисунок 12.23 – Таймер-Лічильник 1 – TCNT1H і TCNT1L

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 байту (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 зображено на рис 12.24.

Регістр порівняння 1 B – OCR1BH і OCR1BL зображено на рис. 12.25.

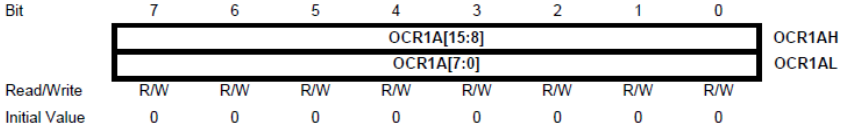


Рисунок 12.24 – Регістр порівняння 1 A – OCR1AH і OCR1AL

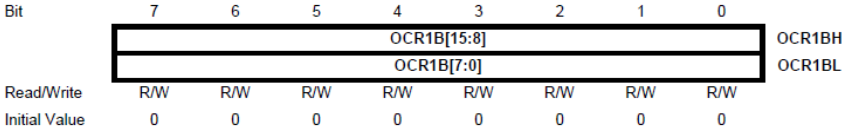


Рисунок 12.25 – Регістр порівняння 1 B – OCR1BH і OCR1BL

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 байту (TEMP). Цей тимчасовий регістр спільно використовується всіма іншими 16-розрядними регістрами.

Регістр захоплення 1 – ICR1H і ICR1L зображено на рис. 12.26.

Регістр захоплення оновлюється значенням лічильника (TCNT1) кожного разу, коли відбувається подія на виводі ICP1 (або за бажанням на виході аналогового компаратора для таймера-лічильника 1).

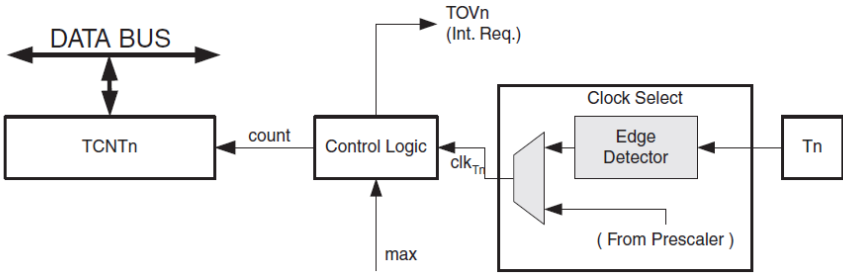


Рисунок 12.26 – Регістр захоплення 1 – ICR1H і ICR1L

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 байту (TEMP). Цей тимчасовий регістр спільно використовується всіма іншими 16-розрядними регістрами.

Регістр маски переривання таймерів-лічильників – TIMSK зображено на рис. 12.27.

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	–	TOIE0	TIMSK
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 12.27 – 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 увімкнено), переривання при захопленні для таймера-лічильника 1 увімкнуно. 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 зображено на рис. 12.28.

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 12.28 – 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 ОТЗ 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.

12.3 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 світлодіоду підвищувати на 1 при кожному переповненні таймера 1. При досяганні максимальної яскравості, встановити її рівною 0.

12.3.1 Принципова електрична схема

Принципова електрична схема, що реалізує поставлену задачу, показана на рис. 12.29.

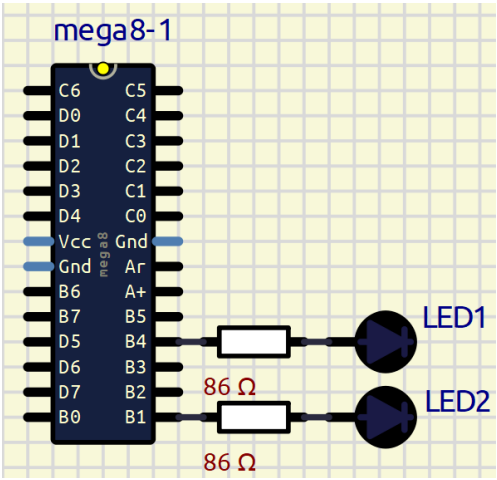


Рисунок 12.29 – Принципова електрична схема

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 можна підключати їх як завгодно.

12.3.2 Програмний код

Розглянемо тепер програму, що реалізує поставлену задачу (рис. 12.30).

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.

```
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 12.30 – Програмний код

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.

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

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

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 світлодіоду 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 1 000 000 Гц / 8 = 125 000 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 світлодіоду, 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 світлодіоду 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 світлодіоду. 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 світлодіоду 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. 12.1, 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 світлодіоду 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 світлодіоду 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. 12.31).

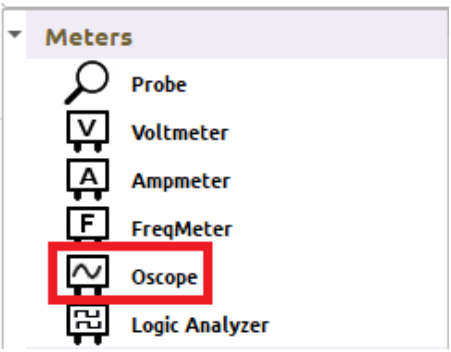


Рисунок 12.31 – Знаходження осцилографа у бібліотеці компонентів

Цей осцилограф є 4-канальним, тобто дозволяє одночасно дивитися до 4 сигналів. Схему з підключеним осцилографом показано на рис. 12.32.

Як бачите, його перший канал підключений до аноду світлодіоду 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. 12.33.

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".

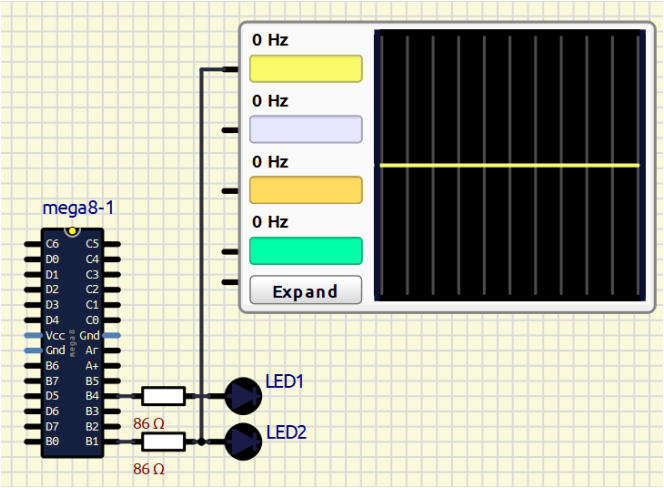


Рисунок 12.32 – Принципова електрична схема з осцилографом

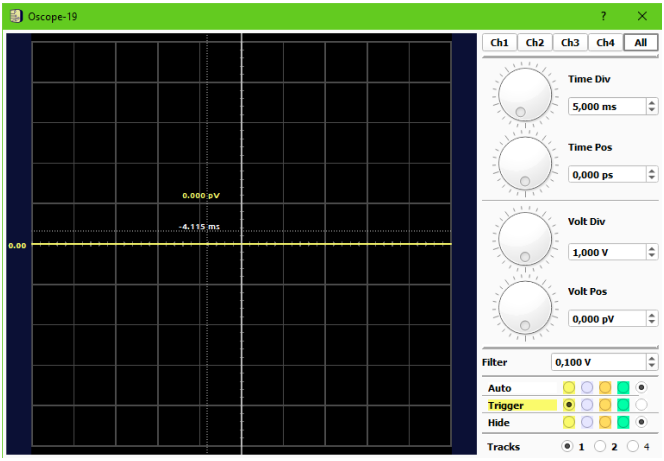


Рисунок 12.33 – Вікно осцилографа

Після запуску симуляції можна побачити, що при малій ширині імпульсів ШІМ яскравість світлодіоду LED2 невелика (рис. 12.34), а при її збільшенні вона також збільшується (рис. 12.35).

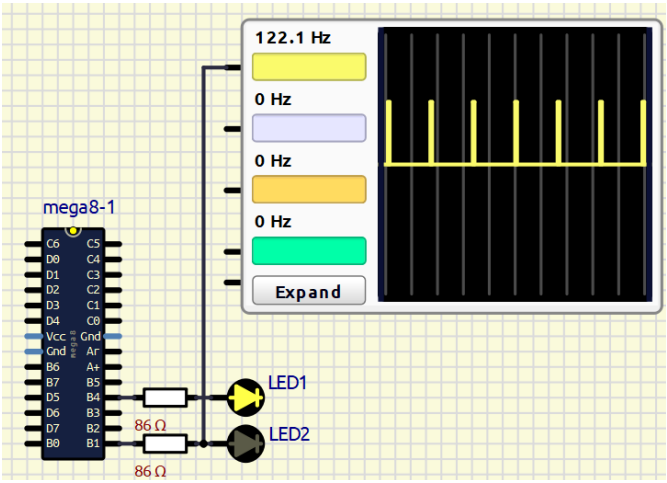


Рисунок 12.34 – Яскравість світлодіоду LED2 при малій ширині імпульсів

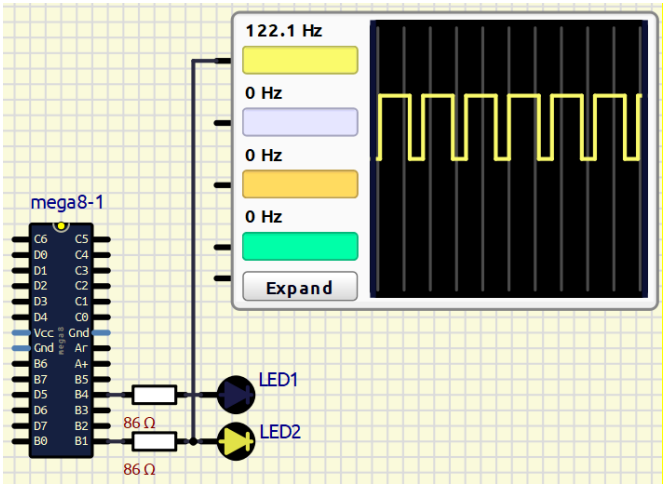


Рисунок 12.35 – Яскравість світлодіоду LED2 при великій ширині імпульсів

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

12.4 Завдання до практичної роботи

1. Створити принципову електричну схему, згідно з варіантом завдань (таблиця 12.10).

2. Написати програму, що виконує поставлене завдання.

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

Таблиця 12.10 – Варіанти завдань до практичної роботи № 3

Варіант	Завдання
1	2
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 світлодіоду 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 світлодіоду LED1, яскравість світлодіоду LED2 зменшувати. При досягненні світлодіодом LED1 максимальної яскравості, її стрибком зменшити до 0, а яскравість світлодіоду 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.

Продовження таблиці 12.10

1	2
	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 світлодіоду, 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. 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 світлодіоду.
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 світлодіоду.
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.

Продовження таблиці 12.10

1	2
	При кожному переповненні таймера 0 змінювати яскравість світлодіоду, 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 світлодіоду.
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 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

Продовження таблиці 12.10

1	2
	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 світлодіоду.

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?

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: 02.12.2024).
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: 02.12.2024).
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. 57 с.
4. Основи Програмування AVR C. DevZone. URL: https://devzone.org.ua/post/osnovy-prohramuvannia-avr-c (дата звернення: 02.12.2024).

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

Перелік питань до розділу:

13.1. Завдання.
13.2. Теоретичні дані.
13.2.1. Загальні відомості про інтерфейс UART.
13.2.2. Апаратна частина UART мікроконтролерів AVR.
13.2.3. Регістри вводу-виводу модуля USART.
13.2.4. Приклади налаштування швидкості передачі даних.
13.3. Приклад програми.
13.3.1. Принципова електрична схема.
13.3.2. Програмний код.
13.3.3. Симуляція роботи програми.
13.4. Завдання до практичної роботи.

13.1 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.

13.2 Теоретичні дані

13.2.1 Загальні відомості про інтерфейс UART[1]

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 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 13.1). 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»

[1] Даний підрозділ викладений на основі матеріалів [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 рис. 13.1 кадр даних складається з 12 бодів.

Формат кадру задається відповідними бітами регістрів керування UCSRB та UCSRC (див. рис. 13.2).

Вибір кількості стоп-бітів здійснюється за допомогою розряду USBS у регістрі керування UCSRC. Якщо цей розряд скинутий

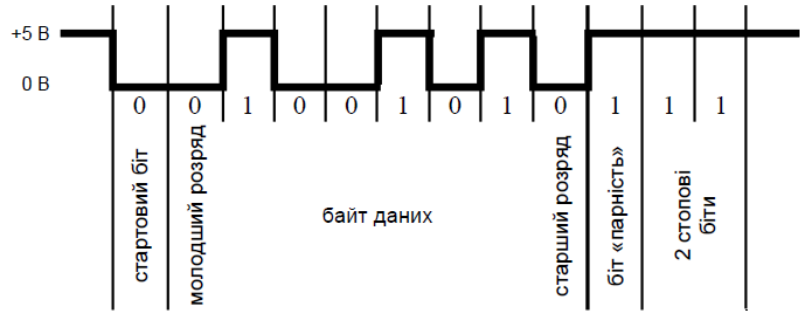


Рисунок 13.1 – Діаграма швидкості передачі

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 13.2 – Визначення розміру байту даних та керування контролем парності

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 13.3.

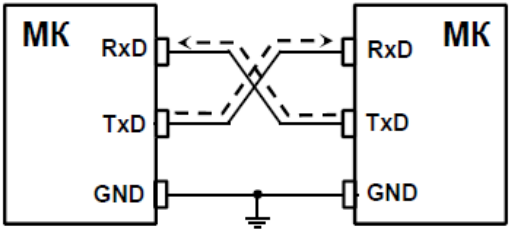


Рисунок 13.3 – З'єднання двох модулів UART для передачі даних

13.2.2 Апаратна частина UART мікроконтролерів AVR[2]

Модуль складається з 3-х основних частин (рисунок 13.4): 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.

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

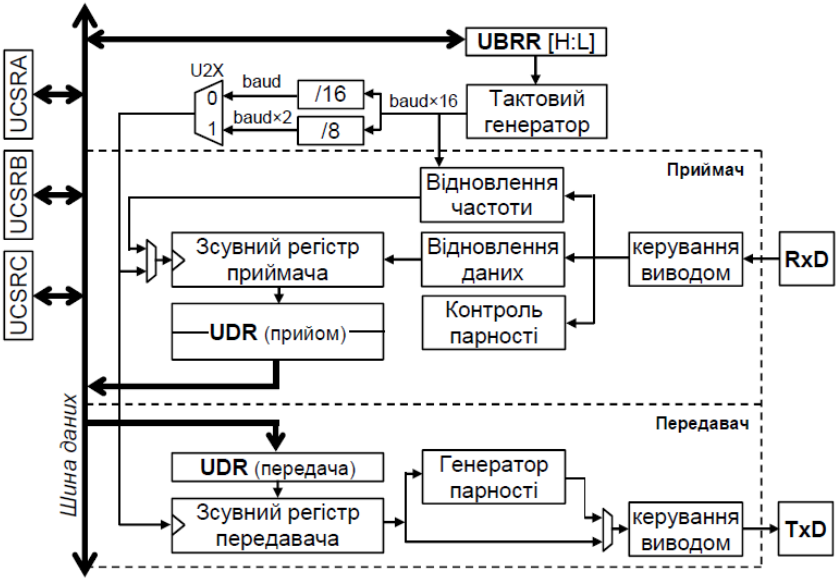


Рисунок 13.4 – Схема модуля UART

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

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:

$$\text{при } U2X = 0 : \boldsymbol{BAUD} = \frac{XTAL}{16\left(UBBR + 1\right)}; \ \boldsymbol{UBRR} = \frac{XTAL}{16 \cdot BAUD} - 1; \quad (13.1)$$

$$\text{при } U2X = 1 : \boldsymbol{BAUD} = \frac{XTAL}{8\left(UBBR + 1\right)}; \ \boldsymbol{UBRR} = \frac{XTAL}{8 \cdot BAUD} - 1. \quad (13.2)$$

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:

$$Error\left[\%\right] = \left(\frac{BAUD_{\text{розрах.}}}{BAUD} - 1 \right) \cdot 100\,\%. \quad (13.3)$$

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

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

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

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

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.

13.2.3 Регістри вводу-виводу модуля USART

Регістр даних USART – UDR (рис. 13.5).

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

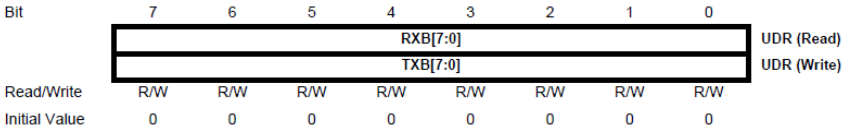


Рисунок 13.5 – Регістр даних USART – UDR

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. 13.6).

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

Рисунок 13.6 – 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. 13.7).

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 13.7 – 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. 13.8).

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 13.8 – 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 13.1 – 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. 13.2).

Таблиця 13.2 – Довжина символу 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

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).

Регістри швидкості передачі даних USART – UBRRL і UBRRH (рис. 13.9).

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 13.9 – 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 13.2.4. 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.

13.2.4 Приклади налаштування швидкості передачі даних

На рис. 13.10–13.13 представлені розраховані значення регістрів UBRRH та UBRRL для різних швидкостей передачі UART та для різних тактових частот мікропроцесора за формулами (13.1) та (13.2).

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. 13.10 – 13.13 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 13.10 – 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 13.11 – 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 13.12 – 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 13.13 – Налаштування регістрів UBRRH та UBRRL при тактових частотах процесора від 16 МГц до 20 МГц

13.3 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.

13.3.1 Принципова електрична схема

Принципова електрична схема, що реалізує поставлену задачу, показана на рис. 13.14.

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

Всі бачили такі індикатори у різних побутових приладах – пральних машинах, мультиварках, 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. 13.15.

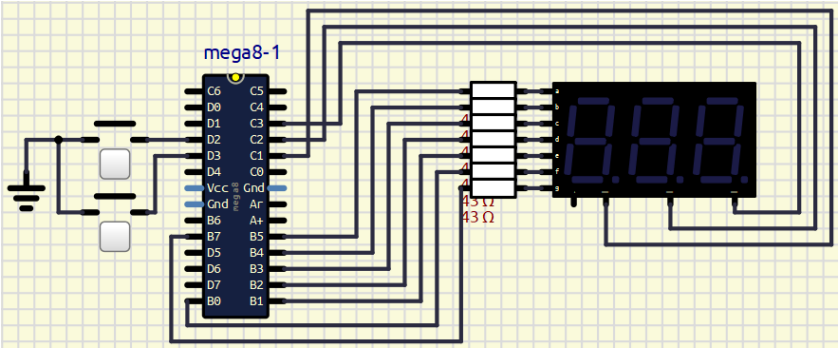


Рисунок 13.14 – Принципова електрична схема

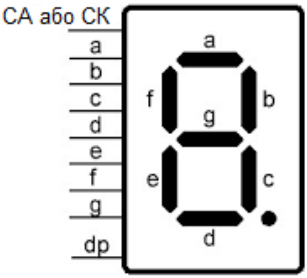


Рисунок 13.15 – Семисегментний індикатор

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. 13.16).

Якщо використовуються багаторазрядні індикатори (в нашому випадку – трирозрядний), то всі однойменні виводи сегментів об'єднані між собою (рис. 13.17). 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.

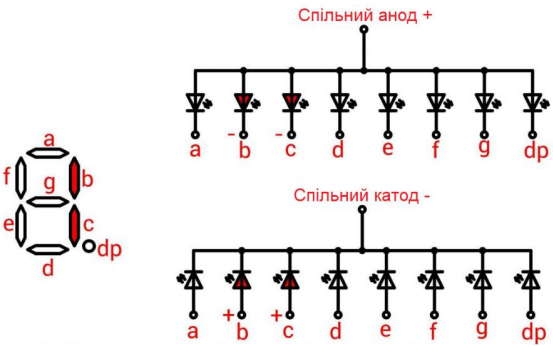


Рисунок 13.16 – Структура семисегментного індикатора

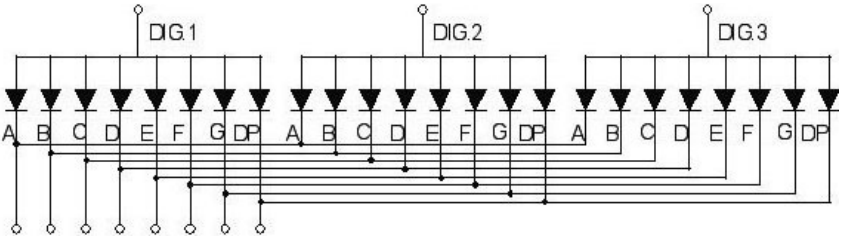


Рисунок 13.17 – Принципова електрична схема трирозрядного семисегментного індикатора зі спільним анодом

У програмі SimulIDE семисегментний індикатор знаходиться у підзаголовку “Outputs” – “Leds”, і називається «7 Segment” (рис. 13.18).

Він має досить багато налаштувань (рис. 13.19).

Більшість параметрів у нього такі самі, як і у світлодіоду (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.

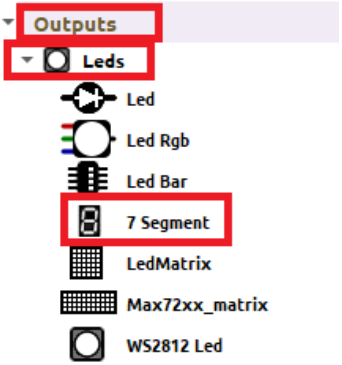


Рисунок 13.18 – Розташування семисегментного індикатора у бібліотеці компонентів

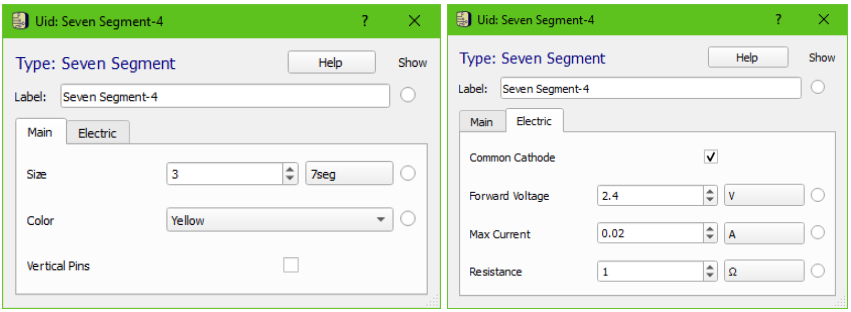


Рисунок 13.19 – Налаштування семисегментного індикатора

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.

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.

13.3.2 Програмний код

Розглянемо тепер програму, що реалізує поставлену задачу (рис. 13.20).

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

```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 13.20 – Програмний код

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. 13.14). 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 рядок, 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 коректний рядок прийнято.

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

У рядку 26 визначається змінна tx_len, яка містить довжину рядка, який потрібно передати.

У рядку 27 визначається масив tx_buf, що буде використовуватися для зберігання рядка, який 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 (рис. 13.2), 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 світлодіоду 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. 13.10). При скинутому біті U2X та значенні регістру UBRR = 6 похибка складає −7 %, що є неприпустимим. А при встановленому біті 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. 13.1). 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 13.1, 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 рядка з символами повернення каретки (’\r’) та переводу рядка (’\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 символьний рядок з аргументів різного типу. В нашому випадку ми формуємо рядок tx_buf, в який 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 рядка новий текстовий рядок починався з нового рядка 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 дані.

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.

13.3.3 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. 13.21).

Після цього відкриється вікно, показане на рис. 13.22. Розберемо його основні елементи більш детально:

1 – Поле для вводу рядка, який буде відправлено у мікроконтролер. Для того, щоб надіслати рядок, треба набрати його на клавіатурі та натиснути кнопку Enter. Після цього переданий рядок з’явиться у полі вхідних даних 5.

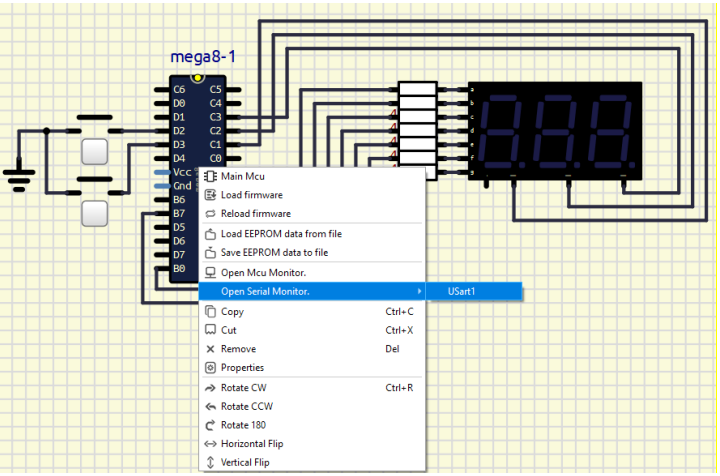


Рисунок 13.21 – Відкриття монітору послідовного порту

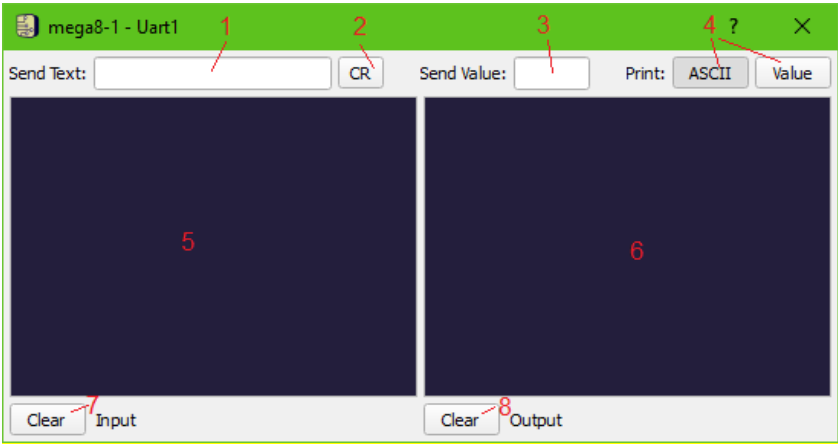


Рисунок 13.22 – Вікно монітору послідовного порту

2 – Кнопка, що дозволяє додати символ повернення каретки ’\r’ в кінці рядка. 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 звичайного рядка (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. 13.23).

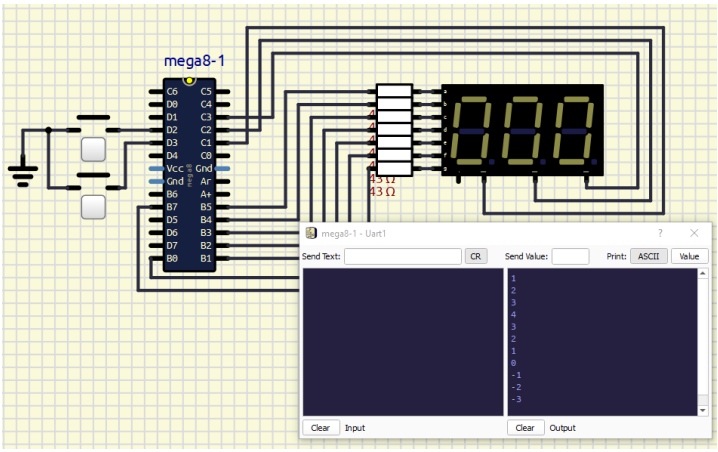


Рисунок 13.23 – Результат передачі даних мікроконтролером

Тепер перевіримо, як працює прийом даних. Для цього у полі вводу рядка 1 введемо будь-яке число, наприклад, 236. І натиснемо кнопку CR (2), щоб символ повернення каретки автоматично додавався до кінця рядка. 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. 13.24).

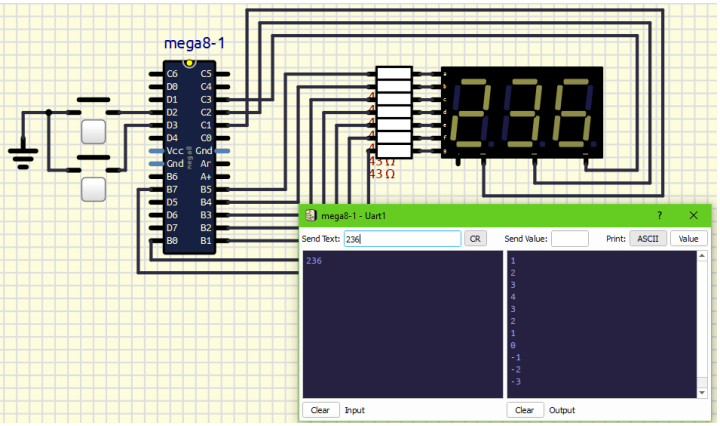


Рисунок 13.24 – Результат прийому даних мікроконтролером

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.

13.4 Завдання до практичної роботи

1. Створити принципову електричну схему, згідно з варіантом завдань (табл. 13.3).
2. Написати програму, що виконує поставлене завдання.
3. Створити файл прошивки, загрузити його у мікроконтролер та переконатися, що все працює, як треба.

Таблиця 13.3 – Варіанти завдань до практичної роботи № 4

Варіант	Завдання
1	**2**
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 рядка, перетворювати його на число зі знаком та відображати його на семисегментному індикаторі. Символом закінчення передачі рядка 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 вимкнути. У моніторі послідовного порту символ передавати за допомогою поля Send Text.

Продовження таблиці 13.3

1	2
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 рядка з символами повернення каретки (’\r’) та переводу рядка (’\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 діапазоні 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 рядка з символами повернення каретки (’\r’) та переводу рядка (’\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 рядка з символами повернення каретки (’\r’) та переводу рядка (’\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.

Продовження таблиці 13.3

1	2
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 восьмибітових чисел: старшого і молодшого байтів. Обробку натискання кнопки робити за допомогою переривання INT1.
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 рядка довжиною до 64 символів, що закінчується символом повернення каретки, рахувати кількість цифр у ньому 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 рядка довжиною до 100 символів, що закінчується символом повернення каретки, замінити всі малі літери на великі і передати новий рядка назад через UART.

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?

Перелік рекомендованих джерел

1. Конспект лекцій з навчальної дисципліни «Програмування мікропроцесорних засобів вимірювальної техніки» зі спеціальності 152 «Метрологія та інформаційно-вимірювальна техніка» за освітньо-професійною програмою «Комп'ютеризовані інформаційно-вимірювальні системи» / розробник Чепюк Л. А. Житомир : Державний університет «Житомирська політехніка», 2021. 144 с.

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: 02.12.2024).
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: 02.12.2024).
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. 57 с.
5. Основи Програмування AVR C. DevZone. URL: https://devzone.org.ua/post/osnovy-prohramuvannia-avr-c (дата звернення: 02.12.2024).

14
ПРАКТИЧНА РОБОТА № 5 «ПРОГРАМУВАННЯ АНАЛОГОВИХ МОДУЛІВ МІКРОКОНТРОЛЕРА AVR»

Перелік питань до розділу:

14.1. Завдання.
14.2. Теоретичні дані.
14.2.1. Аналоговий компаратор.
14.2.2. Модуль АЦП мікроконтролерів AVR.
14.3. Приклад програми з використанням аналогового компаратору та аналого-цифрового перетворювача.
14.4. Принципова електрична схема.
14.4.1. Програмний код.
14.4.2. Симуляція роботи програми.
14.5. Завдання до практичної роботи.

14.1 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.

14.2 Теоретичні дані

14.2.1 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-лічильника 1. 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 рис. 14.1.

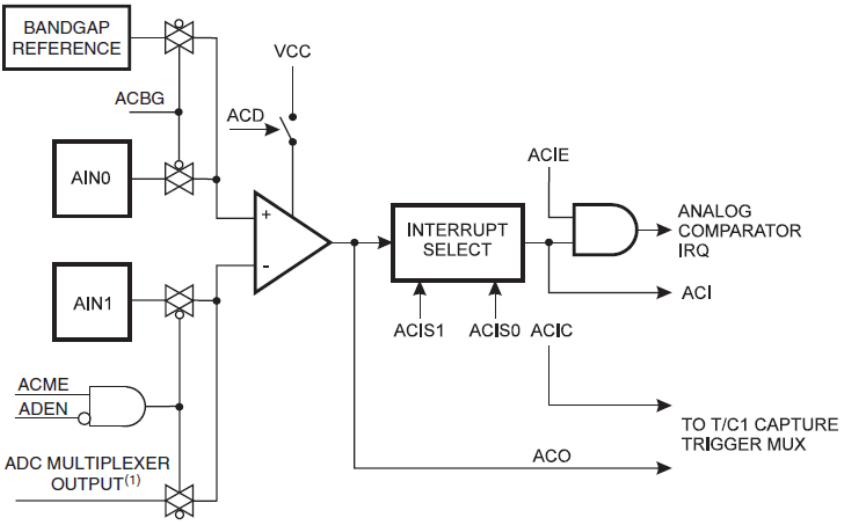


Рисунок 14.1 – Блок-схема компаратора та його оточуюча логіка

Регістри аналогового компаратору мікроконтролера ATMega8.

Регістр спеціальних функцій вводу-виводу – SFIOR (рис. 14.2).

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 14.2 – 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. 14.3).

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 14.3 – 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 вектору 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 придушувач 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 14.1.

Таблиця 14.1 – 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 14.2. Якщо ACME скинуто або встановлено ADEN, вхід AIN1 використовується як негативний вхід аналогового компаратора.

Таблиця 14.2 – Вибір вхідного контакту для підключення до негативного входу аналогового компаратора

ACME	ADEN	MUX2..0	Негативний вхід аналогового компаратора
1	2	3	4
0	x	xxx	AIN1
1	1	xxx	AIN1
1	0	000	ADC0

Продовження таблиці 14.2

1	2	3	4
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

14.2.2 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 рис. 14.4.

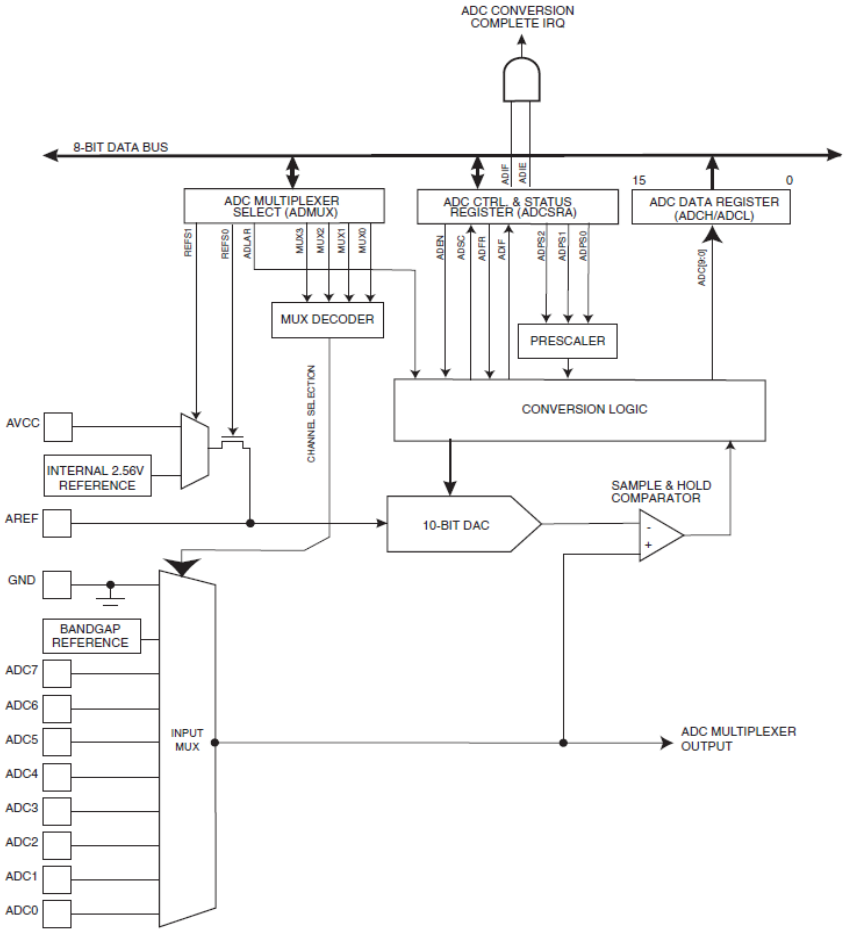


Рисунок 14.4 – Блок-схема АЦП

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. 14.5). 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.

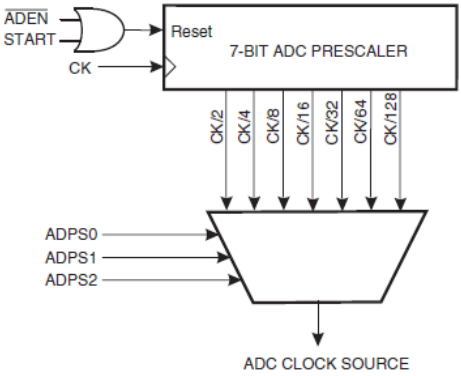


Рисунок 14.5 – Попереднє поділення АЦП

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

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 рис. 14.6. 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 конденсатор S/H через послідовний опір (комбінований опір у вхідному тракті).

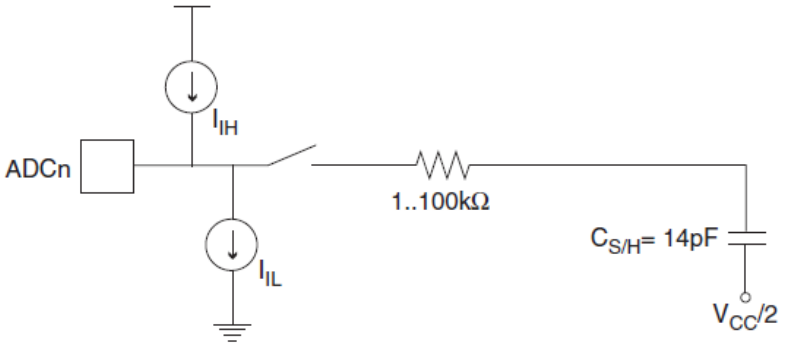


Рисунок 14.6 – Аналогова вхідна схема

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.

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:

$$ADC = \frac{V_{IN} \cdot 1023}{V_{REF}}, \qquad (14.1)$$

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. 14.7):

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 14.7 – 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 14.3 – 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 14.4). 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 14.4 – Біти вибору аналогового каналу

MUX3..0	Вхід
1	2
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	Зарезервовано

Продовження таблиці 14.4

1	2
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. 14.8):

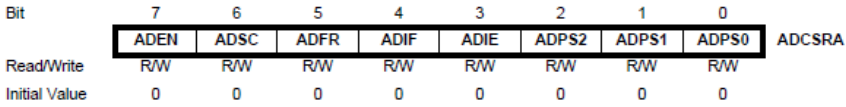


Рисунок 14.8 – Регістр управління та стану АЦП А – ADCSRA

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. 14.5).

Таблиця 14.5 – 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. 14.9).

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.

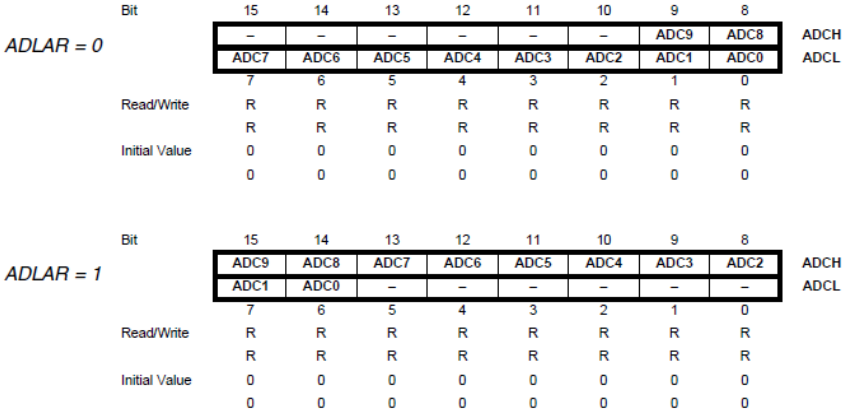


Рисунок 14.9 – Регістр даних АЦП – ADCL і ADCH

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

14.3 Приклад програми з використанням аналогового компаратора 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.

14.4 Принципова електрична схема

Принципова електрична схема, що реалізує поставлену задачу, показана на рис. 14.10.

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. 14.11).

Воно має лише один параметр – власне величину напруги (рис. 14.12), 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. 14.11). На схемі вони виглядають як прямокутники з круглим регулятором всередині (рис. 14.10). 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. 14.13).

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. 14.14)

Для того, щоб ввімкнути це джерело напруги, треба натиснути на надпис "--V" всередині нього, і тоді замість прочерку у надпису з'явиться величина поточної напруги, як показано на рис. 14.10.

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

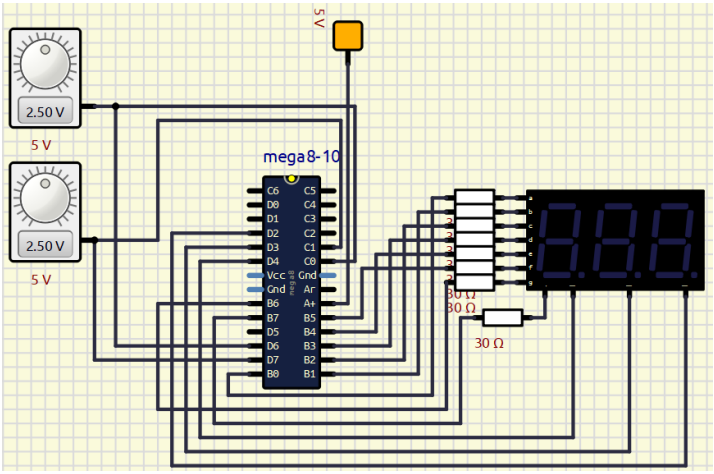


Рисунок 14.10 – Принципова електрична схема

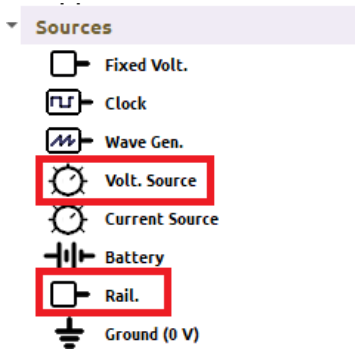


Рисунок 14.11 – Знаходження компонентів Rail та Volt. Source

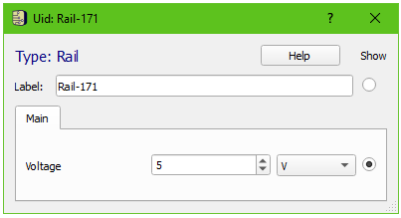


Рисунок 14.12 – Параметри компонента Rail

Як бачимо на рис. 14.10, регульовані джерела напруги одночасно підключені до входів компаратора та АЦП. За допомогою компаратора ми будемо визначати, яка з двох напруг більша, і перемикати вхідний мультиплексор АЦП, щоб виміряти цю напругу.

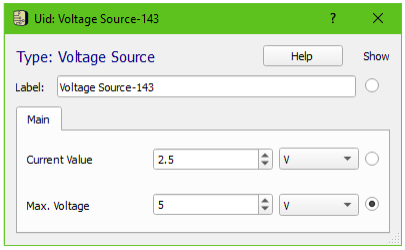


Рисунок 14.13 – Параметри компонента Volt. Source

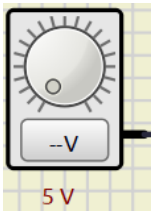


Рисунок 14.14 – Початковий вигляд регульованого джерела напруги

14.4.1 Програмний код

Розглянемо тепер програму, що реалізує поставлену задачу (рис. 14.15).

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 семи сегментним 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.

Спочатку треба налаштувати всі необхідні порти вводу-виводу. Виводи, до яких підключені і аноди, і катоди індикаторів, треба

```
#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 14.15 – Програмний код

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. 12.9), що в нашому випадку дорівнює 1 000 000/64 = 15 625 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. 14.1), для цього треба записати 1 у біт REFS0. Далі обираємо в якості початкового каналу ADC0, до якого підключене одне з джерел напруги (рис. 14.1). 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 (14.2), результат перетворення визначається за наступною формулою:

$$ADC = \frac{V_{IN} \cdot 1023}{V_{REF}}, \qquad (14.2)$$

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:

$$V_{IN} = \frac{V_{REF} \cdot ADC}{1023}. \qquad (14.3)$$

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 14.4 (рядок 69).

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

14.4.2 Симуляція роботи програми

Після компіляції програми та завантаження її у мікроконтролер на індикаторі з'явиться напруга джерела, що підключене до входу ADC0 (рис. 14.16).

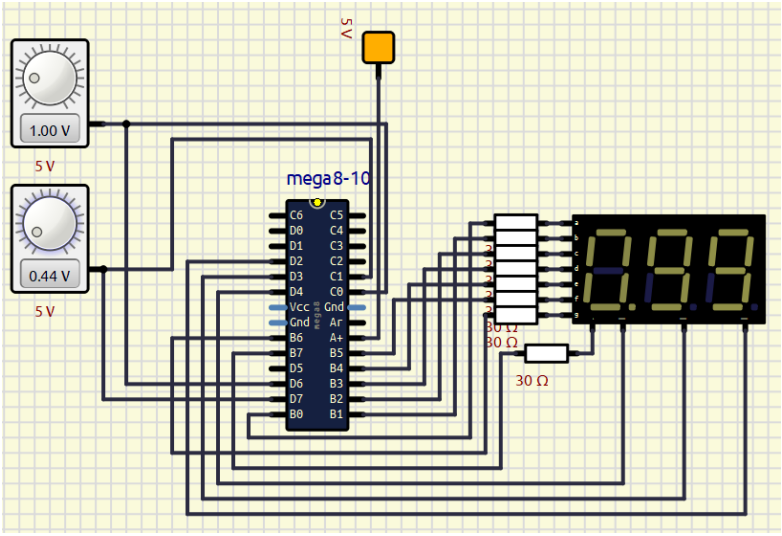


Рисунок 14.16 – Результат роботи програми

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. 14.17, 14.18), таким чином, наша програма працює вірно.

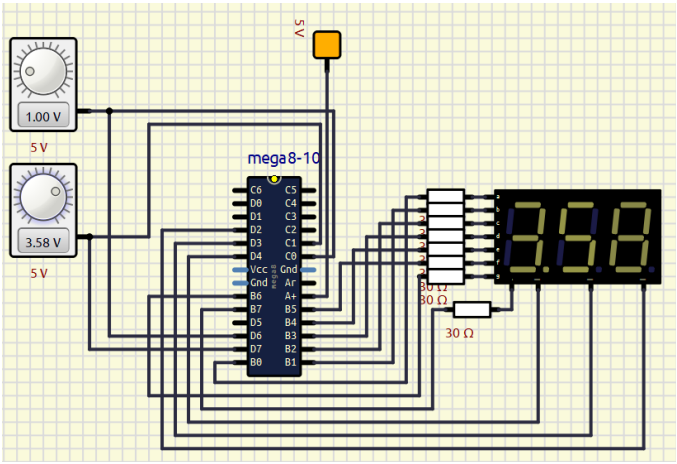


Рисунок 14.17 – Друга напруга більша за першу

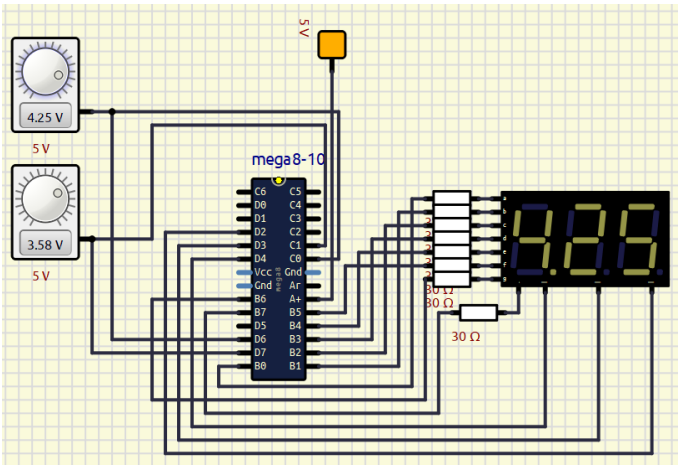


Рисунок 14.18 – Перша напруга більша за другу

14.5 Завдання до практичної роботи

1. Створити принципову електричну схему, згідно з варіантом завдань (табл. 14.6).

2. Написати програму, що виконує поставлене завдання.

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

Таблиця 14.6 – Варіанти завдань до практичної роботи № 5

Варіант	Завдання
1	2
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 світлодіоду від 0 до максимального значення. У протилежному випадку так само плавно знижувати яскравість від максимуму до 0. Керувати яскравістю за допомогою модуля порівняння 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 світлодіоду 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.

Продовження таблиці 14.6

1	2
4	У протилежному випадку частоту миготіння знизити до 1 Гц. Керувати частотою миготіння світлодіоду 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.
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.

Продовження таблиці 14.6

1	2
8	Встановлювати яскравість світлодіоду 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 світлодіоди. 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.

Продовження таблиці 14.6

1	2
	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.

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?

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: 02.12.2024).
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: 02.12.2024).
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. 57 с.
4. Основи Програмування AVR C. DevZone. URL: https://devzone.org.ua/post/osnovy-prohramuvannia-avr-c (дата звернення: 02.12.2024).

Наукове видання

СОКОЛ Сергій Петрович
КОЙФМАН Олексій Олександрович
ІСАЄВ Андрій Борисович
МІРОШНИЧЕНКО Вікторія Ігорівна

ЕЛЕКТРОНІКА ТА МІКРОПРОЦЕСОРНА ТЕХНІКА: ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ AVR

Навчальний посібник

Друкується за авторською редакцією

Дизайн обкладинки В. Савельєва
Технічний редактор О. Гринюк
Верстка Ю. Семенченко

Підписано до друку 23.04.2025 р.
Формат 60×84/16. Папір офсетний.
Цифровий друк. Гарнітура Times.
Ум. друк. арк. 24,88. Наклад 300.
Замовлення № 0225-004.

Видавництво та друк: Олді+
65101, м. Одеса, вул. Інглезі, 6/1
тел.: +38 (095) 559-45-45, e-mail: office@oldiplus.ua
Свідоцтво ДК № 7642 від 29.07.2022 р.

Замовлення книг:
тел.: +38 (050) 915-34-54, +38 (068) 517-50-33
e-mail: book@oldiplus.ua