Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут
імені Ігоря Сікорського»

А. О. Новацький

МІКРОПРОЦЕСОРНІ ТА МІКРОКОНТРОЛЕРНІ СИСТЕМИ

Частина 2

ПРОЕКТУВАННЯ МІКРОПРОЦЕСОРНИХ СИСТЕМ

Підручник

*Затверджено Вченою радою КПІ ім. Ігоря Сікорського
як підручник для студентів, які навчаються за освітньою програмою
«Інтегровані інформаційні системи» за спеціальністю
126 «Інформаційні системи та технології»*

Київ
КПІ ім. Ігоря Сікорського
2021

Рецензенти: Дорошенко Анатолій Юхимович, доктор фіз.-мат. наук, професор,
зав. відділом Інституту програмних систем НАН України
Чемерис Олександр Анатолійович, доктор техн. наук, ст.н.с.,
заступник директора з наукової роботи Інституту проблем моделювання
в енергетиці ім. Г.Є. Пухова НАН України
Відповідальний
редактор: Полторак Вадим Петрович, кандидат техн. наук, доцент,
КПІ ім. Ігоря Сікорського,
кафедра автоматики та управління в технічних системах

Гриф надано Вченою радою КПІ ім. Ігоря Сікорського (протокол № 3 від 15 березня 2021 р.)

Електронне мережне навчальне видання

Новацький Анатолій Олександрович, кандидат техн. наук, доцент

Мікропроцесорні та мікроконтролерні системи: Ч.2 «Проектування мікропроцесорних систем» [Електронний ресурс] : підручник для студ. освітньої програми «Інтегровані інформаційні системи» за спеціальністю 126 «Інформаційні системи та технології» / А.О. Новацький ; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 20,3 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2021. – 462 с.

Підручник охоплює теоретичний матеріал та практичні завдання, які необхідні для вивчення другої частини дисципліни «Мікропроцесорні та мікроконтролерні системи». Розглянуто структуру, програмну модель та характеристику команд типового 8-розрядного мікроконтролера. Наведено матеріал про проектування основних периферійних модулів мікропроцесорних систем з використанням мікроконтролерів сім`ї AVR: таймерів; паралельних портів; підсистеми переривань; послідовного синхронно-асинхронного інтерфейсу; пам’яті; аналого-цифрового та цифро-аналогового перетворювачів; послідовних інтерфейсів SPI, I^2C та CAN; широтно-імпульсного модулятора. Розглянуто моделювання у пакеті Proteus: універсального асинхронного інтерфейсу, послідовних синхронних інтерфейсів SPI та I^2C; широтно-імпульсного модулятора; годинника реального часу; аналого-цифрового перетворювача та цифрового вольтметра.

Робота може бути корисною студентам відповідних спеціальностей під час вивчення дисциплін, які пов’язано із проектуванням та використанням мікропроцесорних систем, а також при виконанні курсових проектів, бакалаврських робіт та магістерських дисертацій, в яких використовуються мікропроцесорні пристрої.

© А.О.Новацький, 2021
© КПІ ім.Ігоря Сікорського, 2021

ЗМІСТ

ВСТУП 10
СПИСОК СКОРОЧЕНЬ..... 12
1. ХАРАКТЕРИСТИКА AVR-МІКРОКОНТРОЛЕРІВ..... 16
1.1. Розвиток мікроконтролерів сімейства AVR 16
1.2. Характеристика AVR-мікроконтролерів 19
1.3. Основні параметри AVR-мікроконтролерів..... 24
1.4 Структура типового AVR-мікроконтролера 25
Контрольні запитання та завдання 30
2. ОРГАНІЗАЦІЯ ПАМ’ЯТІ 31
2.1. Загальна характеристика 31
2.2. Організація пам’яті програм 32
2.3. Організація пам'яті даних..... 33
2.3.1. Загальна характеристика 33
2.3.2. Статична пам'ять даних 33
2.3.3. Енергонезалежна пам'ять даних EEPROM 44
2.4. Програмування FLASH- та EEPROM-пам’яті 48
Контрольні запитання та завдання 53
3. ПРОГРАМУВАННЯ AVR-МІКРОКОНТРОЛЕРІВ 55
3.1. Послідовність розробки керуючої програми..... 55
3.2. Мови програмування 55
3.3. Програмна модель AVR-мікроконтролера 56
3.4. Функціонування конвеєра 62
3.5. Лічильник команд 63
3.6. Способи адресації операндів..... 64
3.7. Загальна характеристика команд AVR-мікроконтролерів..... 71
3.8. Формати команд 72
3.9. Формати даних 74
3.10. Довжина команд у байтах та їх розміщення у пам'яті програм 75
3.11. Вплив команд на прапорці 75
3.12. Час виконання команд 76
3.13. Базовий набір команд мікроконтролера 76
3.14 Нові команди AVR-мікроконтролерів 86
Контрольні запитання та завдання 99
4. САМОПРОГРАМУВАННЯ AVR-МІКРОКОНТРОЛЕРІВ 100
4.1. Загальні відомості 100
4.2. Області RWW та NRWW..... 101
4.3. Прапорець зайнятості секції RWW та його скидання 104
4.4. Керування процесом самопрограмування 104

4.5. Зміна вмісту пам’яті програм 110
4.6. Типові процедури оновлення Flash-пам’яті 114
4.7. Режими захисту Flash- та EEPROM-пам’яті 115
4.8. Читання конфігураційних комірок та комірок захисту 115
Контрольні запитання та завдання 116
5. ПРОЕКТУВАННЯ МОДУЛЯ ВВЕДЕННЯ/ВИВЕДЕННЯ 117
5.1. Загальна характеристика модуля 117
5.2. Способи обміну даними між ЗВПР та МК/МП у МПС 117
5.3.Порти введення/виведення AVR-мікроконтролерів 119
5.3.1. Загальна характеристика 119
5.3.2. Звернення до портів введення/виведення 121
5.3.3. Структура портів введення/виведення 121
5.3.4. Конфігурування виводів портів введення/виведення 125
5.3.5. Приклад конфігурування одного з портів мікроконтролера 126
Контрольні запитання та завдання 127
6. ПРОЕКТУВАННЯ МОДУЛЯ УАПП/УСАПП 128
6.1. Загальні відомості 128
6.2.Опис структури модулів УАПП/УСАПП 130
6.3. Опис керуючих регістрів модуля УСАПП 132
6.4. Швидкість прийому/передачі даних 132
6.5. Формат кадру 135
6.6. Передача даних 139
6.7. Прийом даних 142
6.8. Обмін даними через інтерфейс УСАПП у мікроконтролерній мережі .. 147
6.9. Робота модуля УСАПП в режимі MSPI 148
6.10. Розрахунок швидкості передачі інформації, тривалості одного біта та часу передачі одного байта 149
6.11. Моделювання модуля УСАПП у пакеті PROTEUS 150
6.11.1. Опис моделі 150
6.11.2. Порядок моделювання 151
6.11.3. Схема алгоритму роботи 157
6.11.4. Робоча програма 157
Контрольні запитання та завдання 161
7. ПРОЕКТУВАННЯ ПОСЛІДОВНОГО СИНХРОННОГО ІНТЕРФЕЙСУ TWI 162
7.1. Загальна характеристика інтерфейсу I^2C (TWI) 162
7.2. Архітектура інтерфейсу TWI на апаратному рівні 166
7.2.1. Загальна характеристика 166
7.2.2. Особливості обміну даними шиною TWI 167

7.2.3. Опис структури модуля TWI 173
7.2.4. Взаємодія прикладної програми з модулем TWI 181
7.2.5. Перелік мікроконтролерів AVR, які мають інтерфейс TWI 184
7.2.6. Перелік пристроїв, які підтримують інтерфейс TWI 185
7.3. Програмування інтерфейсу TWI 185
7.3.1. Режими роботи модуля TWI 185
7.3.2. Режим «Ведучий передавач» 185
7.3.3. Режим «Ведучий приймач» 190
7.3.4. Режим «Ведений приймач» 193
7.3.5. Режим «Ведений передавач» 199
7.3.6. Комбінування різних режимів 203
7.3.7. Арбітраж 204
7.4. Моделювання інтерфейсу TWI у пакеті PROTEUS 8.6 205
7.4.1. Опис моделі 205
7.4.2. Налаштування частоти тактових імпульсів лінії SCL 206
7.4.3. Запуск процесу моделювання 206
7.4.4. Схема алгоритму роботи моделі 209
7.4.5. Робоча програма 213
7.4.6. Створення проекту в AVR STUDIO 220
Контрольні запитання та завдання 221
8. ПРОЕКТУВАННЯ МОДУЛЯ SPI 223
8.1. Загальні відомості 223
8.2. Характеристика модуля SPI мікроконтролерів AVR 223
8.2.1. Опис структурної схеми модуля 223
8.2.2. Програмування модуля 225
8.2.3. Опис обміну даними між двома мікронтролерами 226
8.2.4. Структура SPI-мережі мікроконтролерів 229
8.2.5. Режими передачі даних інтерфейсом SPI 229
8.2.6. Програмування швидкості передачі даних 232
8.2.7. Використання виводу $\overline{SS}$ 233
8.2.8. Режим програмування пам’яті послідовним каналом 234
8.2.9. Периферійні пристрої з SPI-інтерфейсом 236
8.2.10. Універсальний послідовний інтерфейс USI 236
8.3. Моделювання модуля SPI 236
8.3.1. Опис моделі 236
8.3.2. Порядок моделювання 238
8.3.3. Схема алгоритму роботи 241
8.3.4. Робоча програма 244

Контрольні запитання та завдання 248
9. ПРОЕКТУВАННЯ МОДУЛЯ ТАЙМЕРІВ 249
9.1. Особливості архітектури таймерів AVR-мікроконтролерів 249
9.1.1. Загальні відомості 249
9.1.2. Призначення виводів таймерів/лічильників 250
9.1.3. Переривання від таймерів/лічильників 250
9.1.4. Попередні дільники таймерів/лічильників 253
9.1.5. Керування попередніми дільниками 254
9.1.6. Використання зовнішнього тактового сигналу 254
9.2. Восьмирозрядні таймери/лічильники 254
9.3. 16-розрядні таймери/лічильники 255
9.3.1. Загальна характеристика 255
9.3.2. Звернення до 16-розрядних регістрів 261
9.3.3. Керування тактовим сигналом 262
9.3.4. Режими роботи 262
9.4. Вартовий таймер 275
9.5. Моделювання модуля таймера для керування двигуном постійного струму 276
9.5.1. Опис моделі 276
9.5.2. Мікроконтролер 282
9.5.3. Модуль таймера 285
9.5.4. Драйвер двигуна 285
9.5.5. Захисні діоди 288
9.5.6. Програмування таймера мовою Асемблер 289
9.5.7. Схема алгоритму роботи 295
9.5.8. Робоча програма мовою С 296
9.6. Моделювання модуля таймера в якості годинника реального часу 297
9.6.1. Опис моделі 297
9.6.2. Схема алгоритму роботи 300
9.6.3. Робоча програма мовою С 307
9.6.4. Опис роботи програми 318
Контрольні запитання та завдання 321
10. ПРОЕКТУВАННЯ МОДУЛЯ АНАЛОГО-ЦИФРОВОГО ПЕРЕТВОРЮВАЧА 322
10.1. Загальні відомості 322
10.2. Функціонування модуля 322
10.3. Програмування модуля 324
10.4. Формування тактового сигналу 329
10.5. Часові діаграми роботи 330

10.6. Керування вхідним мультиплексором 333
10.7. Збереження результату перетворення 336
10.8. Рекомендації з перемикання вхідних каналів 337
10.9. Особливості підключення джерела опорної напруги 337
10.10. Результат перетворення 338
10.11. Підвищення точності перетворення 340
10.12. Відносна похибка 340
10.13. Моделювання модуля АЦП у пакеті PROTEUS 340
10.13.1. Опис моделі 340
10.13.2. Схема алгоритму роботи 343
10.13.3. Робоча програма мовою С 347
10.14. Моделювання у пакеті PROTEUS цифрового вольтметра 349
10.14.1. Особливості архітектури модуля АЦП у складі мікроконтролера ATmega32 349
10.14.2. Опис моделі цифрового вольтметра 350
10.14.3. Схема алгоритму роботи 353
10.14.4. Робоча програма мовою С 354
Контрольні запитання та завдання 357
11. ПРОЕКТУВАННЯ МОДУЛЯ ЦИФРО-АНАЛОГОВОГО ПЕРЕТВОРЮВАЧА 358
11. 1. Характеристика модуля ЦАП у складі мікроконтролерів XMega 359
11.2. Принцип роботи модуля 359
11.3. Джерела опорної напруги 360
11.4. Вихідні канали 360
11.5. Режими роботи 361
11.6. Тактування модуля 362
11.7. Обмеження часових характеристик 363
11.8. Режим енергозбереження 364
11.9. Система подій 364
11.10. Програмування модуля 365
11.10.1. Регістри даних 365
11.10.2. Регістри керування 366
11.10.3. Регістри калібрування 370
11.11. Характеристики ЦАП у складі мікроконтролерів AVR 371
11.12. Опис роботи та розрахунок цифро-аналогових перетворювачів на основі резисторної матриці R-2R з підсумовуванням напруг 373
11.12.1. Загальна характеристика 373
11.12.2. Принцип дії ЦАП з матрицею R-2R та підсумовуванням напруг . 373
11.12.3. Розрахунок ЦАП 374

Контрольні запитання та завдання 376
12. ПРОЕКТУВАННЯ МОДУЛЯ АНАЛОГОВОГО КОМПАРАТОРА 377
12.1. Загальні відомості 377
12.2. Аналогові компаратори на інтегральній мікросхемі операційного підсилювача 378
12.3. Аналоговий компаратор у складі AVR-мікроконтролерів 379
12.3.1. Загальні відомості 379
12.3.2.Функціонування та програмування компаратора 380
Контрольні запитання та завдання 384
13. СПЕЦІАЛЬНІ РЕЖИМИ РОБОТИ МІКРОКОНТРОЛЕРА 385
13.1. Тактування 385
13.1.1. Загальні відомості 385
13.1.2. Генератор із зовнішнім резонатором 387
13.1.3. Низькочастотний кварцовий генератор 387
13.1.4. Зовнішній сигнал синхронізації 388
13.1.5. Генератор з зовнішнім RC-ланцюгом 388
13.1.6. Внутрішній калібрований RC-генератор 389
13.1.7. Керування тактовою частотою 389
13.2. Режим зниженого енергоспоживання 390
13.2.1. Загальні відомості 390
13.2.2. Керування режимами зниженого енергоспоживання 390
13.3. Скидання 393
13.4. Підсистема переривань 393
13.4.1. Загальні відомості 393
13.4.2. Таблиця векторів переривань 395
13.4.3. Обробка переривань 396
13.4.4. Зовнішні переривання 398
13.4.5. Особливості використання модуля переривань у мікроконтролерах XMega 399
Контрольні запитання та завдання 399
14. ПРОЕКТУВАННЯ CAN-МОДУЛЯ 401
14.1. Загальні відомості 401
14.2. Основні характеристики CAN-протоколу 404
14.3. Структура повідомлень CAN-мережі 406
14.3.1. Загальні відомості 406
14.3.2. Кадр даних 407
14.3.3. Кадр віддаленого запиту даних 413
14.3.4. Кадр перевантаження 416
14.3.5. Міжкадровий простір 416

14.4. CAN-модуль мікроконтролера AT90CAN128 416
14.4.1. Загальні відомості 416
14.4.2. Структура CAN-модуля 417
14.4.3. Організація керуючих регістрів 418
14.4.4. Режими роботи 419
14.4.5. Структура блоку фільтрації 423
14.4.6. Структура переривань від CAN-модуля 424
14.4.7. Структура блоку CAN-таймера 427
14.4.8. Обробка помилок 429
14.4.9. Бітова синхронізація 430
14.5. Фізичний рівень CAN-протоколу 437
14.5.1. Загальні відомості 437
14.5.2. Трансивери для CAN-мережі 438
14.5.3. Роз'єми для CAN-мережі 440
14.6. Принципова схема вузла CAN-мережі 441
14.7. Тлумачний словник основних загальноприйнятих термінів CAN- технології 445
14.8. Схема алгоритму роботи CAN-модуля 445
14.9. Моделювання CAN-модуля 445
Контрольні запитання та завдання 455
ПРЕДМЕТНИЙ ПОКАЖЧИК 456
СПИСОК ЛІТЕРАТУРИ 459

ВСТУП

У цьому підручнику розглядається проектування мікропроцесорних систем з використанням сучасних AVR-мікроконтролерів.

Підручник орієнтовано на підготовку фахівців зі спеціальності «Інформаційні системи та технології», яка сьогодні використовується як інтеграційна основа під час організації та здійснення діяльності з розроблення інформаційних систем широкого спектру. Для успішного опановування майбутніми спеціалістами цією спеціальністю необхідно засвоїти принципи побудови та використання мікропроцесорних пристроїв, на основі яких розробляються та функціонують мікропроцесорні та обчислювальні системи. Підручник охоплює теоретичний матеріал та практичні завдання, які необхідні для вивчення другої частини дисципліни «Мікропроцесорні та мікроконтролерні системи» – «Проектування мікропроцесорних систем». Підручник складається із чотирнадцяти розділів, в яких розглядаються питання, пов’язані з проектуванням мікропроцесорних систем (МПС) на основі AVR-мікроконтролерів. Останні є одним з цікавих напрямів, що розвиваються корпораціями Atmel та Microchip. Об`єми продажів AVR у світі щорічно збільшується, неухильно росте число сторонніх фірм, що випускають програмні й апаратні засоби підтримки розробок для цих мікроконтролерів. Можна вважати, що AVR поступово став одним із індустріальних стандартів серед 8-розрядних мікроконтролерів загального призначення. На даний час у виробництві знаходяться три сімейства AVR – "tiny", "mega", "XMega", що розрізняються об‘ємами Flash-, EEPROM- і SRAM-пам'яті, набором периферійних пристроїв і побудовою схеми тактування.

У роботі розглянуто такі питання: характеристика, організація пам`яті, програмування та самопрограмування AVR-мікроконтролерів; проектування периферійних модулів МК: паралельних портів; таймерів; УАПП/УСАПП; I^2C; SPI; CAN; АЦП; ЦАП; аналогового компаратора та спеціальні режими роботи AVR-мікроконтролерів.

У кінці кожного розділу наведено контрольні запитання та завдання для самоконтролю.

Тематика підручника відповідає робочій програмі дисципліни «Проектування мікропроцесорних систем», яка є важливою дисципліною у навчальному плані підготовки бакалаврів і магістрів у Національному технічному університеті України «Київський політехнічний інститут імені Ігоря Сікорського» за освітньою програмою «Інтегровані інформаційні системи» зі спеціальності 126 «Інформаційні системи та технології».

Студент має знати: принципи побудови та функціонування мікропроцесорних та мікроконтролерних пристроїв та систем, принципи

вибору методів аналізу і розрахунку цих пристроїв та систем із заданими характеристиками, а також уміти: розраховувати, програмувати та моделювати мікропроцесорні та мікроконтролерні пристрої та системи; розробляти структурні, функціональні та принципові схеми та схеми алгоритмів роботи.

Підручник написано на основі досвіду викладання відповідних дисциплін згідно з програмами підготовки бакалаврів і магістрів спеціальності 126 «Інформаційні системи та технології» на кафедрі автоматики та управління в технічних системах у Національному технічному університеті України «Київський політехнічний інститут імені Ігоря Сікорського».

Курс забезпечується наступними дисциплінами: математика, фізика, програмування, основи системної інженерії, електротехніка, комп'ютерна електроніка та мікропроцесорні системи.

В свою чергу, курс забезпечує засвоєння студентами більшості наступних дисциплін спеціальності, оскільки в ньому розглядаються архітектурні особливості проектування сучасних мікроконтролерних пристроїв та систем, які є основою сучасних інтегрованих інформаційних систем.

Робота може бути корисною студентам відповідних спеціальностей під час вивчення дисциплін, які пов'язані із проектуванням та використанням мікропроцесорних систем, а також під час виконання курсових проектів, бакалаврських та магістерських робіт, в яких використовуються мікропроцесорні пристрої.

СПИСОК СКОРОЧЕНЬ

АК – аналоговий компаратор
АЛП – арифметико-логічний пристрій
АЦП – аналого-цифровий перетворювач
ДК – двійковий код
ДНЛ – диференціальна нелінійність
ДОН – джерело опорної напруги
ДРВВ – додаткові регістри введення/виведення
ЕСППЗП – електрично стираємий програмований постійний запам’ятовуючий пристрій
ЗВПР – зовнішній пристрій
ЗП – запам'ятовуючий пристрій
ЗПП – зовнішня пам'ять програм
ІМС ОП – інтегральна мікросхема операційного підсилювача
ІНЛ – інтегральна нелінійність
КЗ – канал зв’язку
КМОН – комплементарний метал-оксид-напівпровідник
КОП – код операції команди
МА – мова асемблера
МЗР – молодший значущий розряд
МК – мікроконтролер
МПЛ – мультиплексор
МПС – мікропроцесорна система
ОЗП – оперативний запам’ятовуючий пристрій
ОП – операційний підсилювач
ПВВ – порти введення/виведення
ПВЗ – пристрій вибірки-зберігання
ПДП – прямий доступ до пам'яті
ПЗП – постійний запам’ятовуючий пристрій
ПП – пам’ять програм
ППЗП – перепрограмовуваний постійний запам'ятовуючий пристрій
ППР – периферійний пристрій
РВВ – регістр введення/виведення
РЗП – регістр загального призначення
РПП – резидентна пам'ять програм
СЗР – старший значущий розряд
СОЗП – статичний оперативний запам’ятовуючий пристрій

СПД – статична пам’ять даних
ТТЛШ – транзисторно-транзисторна логіка з діодами Шоттки
УСАПП – універсальний синхронно/асинхронний приймач-передавач
ЦАП – цифро-аналоговий перетворювач
ЦПП – центральний процесорний пристрій
ША – шина адреси
ШД – шина даних
ШІМ – широтно-імпульсна модуляція
ACK – підтвердження (acknowledge)
ACME – підключення виходу АЦП до входу аналогово компаратора (Analog Comparator Multiplexer Enable)
BT – час передачі одного біта (bit time)
CAN – мережа контролерів (Controller Area Network)
CPHA – фаза тактового сигналу (Clock Phase)
CPOL – полярність тактового сигналу (Clock Polarity)
DMA – прямий доступ до пам’яті (direct memory access)
EEPROM-пам’ять – енергонезалежна пам'ять, що може бути електрично стерта та перепрограмована (Electrically Erasable Programmable Read-Only Memory)
ETIMSK – розширений регістр маски переривань від таймерів/лічильників (Extended Timer/Counter Interrupt MaSK Register)
FIFO – першим зайшов, першим вийшов (First In, First Out)
GTCCR – загальний регістр керування таймером/лічильником (General Timer/Counter Control Register)
IEEE – Інститут інженерів з електротехніки та електроніки (Institute of Electrical and Electronics Engineers)
I^2C – двонаправлена двопровідна шина для так званого міжмікросхемного застосування (InterIC, або IIC)
JTAG – інтерфейс, призначений для підключення складних цифрових мікросхем або пристроїв рівня друкованої плати до стандартної апаратури тестування і налагодження (Joint Test Action Group)
LSB – молодший значущий розряд (Least Significant Bit)
MIPS – мільйон інструкцій на секунду (Million Instructions Per Second)
MISO – сигнал входу для ведучого, виходу для веденого (Master In Slave Out)

MOb – об'єкт повідомлення (Message Objects)
MOSI – сигнал виходу для ведучого, входу для веденого (Master Out Slave In)
MSB – старший значущий розряд (Most Significant Bit)
MSPI – режим ведучого SPI (Master SPI Mode)
NACK – не підтвердження (no acknowledge)
NRZ – код без повернення до нуля (Non Return to Zero)
NRWW – немає читання під час запису (No Read-While-Write)
OCI – відкрита система взаємозв'язку (Open System Interconnection)
OCR – вихід регістра порівняння (Output Compare Register)
PC – лічильник команд (Program counter)
RWW – читання під час запису (Read-While-Write)
SCK – лінія тактового сигналу (Serial Clock)
SCL – послідовна лінія тактування (Serial CLock)
SDA – послідовна лінія даних (Serial Data)
SFIOR – регістр спеціальних функцій введення/виведення (Special Function Input Output Register)
SP – покажчик стека (Stack pointer)
SPI – послідовний периферійний інтерфейс (Serial Peripheral Interface)
SPM – зберігання пам'яті програм (Store Program Memory)
SPMCR – керуючий регістр зберігання пам'яті програм (Store Program Memory Control Register)
TAP – порт тестового доступу (Test Access Port)
TCCR – регістр керування таймером/лічильником (Timer/Counter Clock Control Register)
TIFR – регістр прапорців переривань від таймерів/лічильників (Timer/Counter Interrupt Flag Register)
TIMSK – регістр маски переривань від таймерів/лічильників (Timer/Counter Interrupt MaSK Register)
TTCAN – CAN-мережа, керована часом (Time-Triggered CAN)
TWI – двопровідний інтерфейс (Two Wire Interface)
UDR – буферні регістри приймача і передавача (Universal Data Register)
USART – універсальний синхронний/асинхронний приймач-передавач (Universal Synchronous and Asynchronous Receiver and Transmitter)

USI – універсальний послідовний інтерфейс (Universal Serial Interface

WDTCR – регістр керування вартовим таймером (Watch Dog Timer Control Register)

1. ХАРАКТЕРИСТИКА AVR-МІКРОКОНТРОЛЕРІВ

1.1. Розвиток мікроконтролерів сімейства AVR

«Традиційні» AVR-мікроконтролери починали свій розвиток з серії Classic (позначення AT90S). Зараз вони зняті з виробництва, але їх архітектутні властивості використано в сучасних трьох сімействах: Tiny (позначення ATtiny), Mega (позначення ATmega) і XMega (позначення ATXMega) (рис. 1.1).

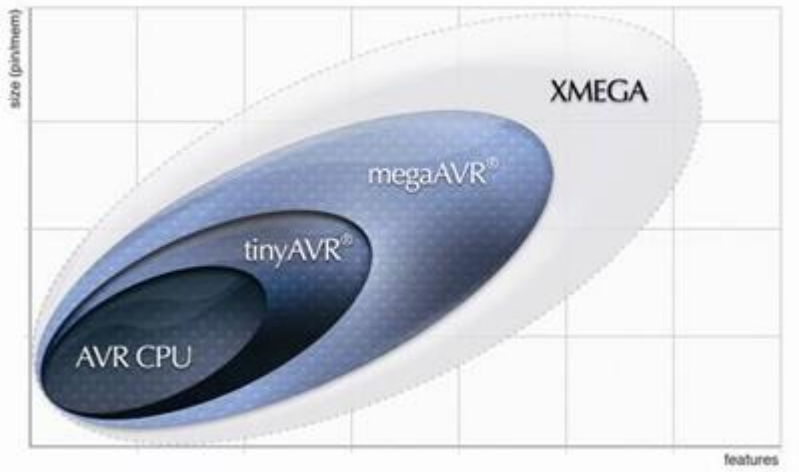


Рис. 1.1. Стандартні сімейства AVR

При припиненні серійного виробництва якого-небудь AVR-мікроконтролера компанія Atmel пропонувала його заміну, яка повністю сумісна повиводно і функціонально з попереднім представником, і є "гнучкішим", надійнішим і продуктивнішим екземпляром з додатковими функціональними можливостями. При переході на новий мікроконтролер розробники зазвичай не мають серйозних ускладнень. Як правило, потрібне лише незначне оновлення початкового тексту програми, освоєння додаткової функціональності, вивчення нових апаратних і програмних засобів підтримки.

Розробники AVR-мікроконтролерів поклопоталися про інженерів, що застосовують дану платформу, зробивши мікроконтролери різних серій повиводно сумісними (рис. 1.2). Таке рішення забезпечує свободу вибору кристала і можливість подальшого переходу між різними серіями.

Для AVR-мікроконтролерів також характерна програмна сумісність «знизу вверх», що забезпечується збереженням найменувань регістрів спеціального призначення в різних серіях і практично єдиним набором команд.

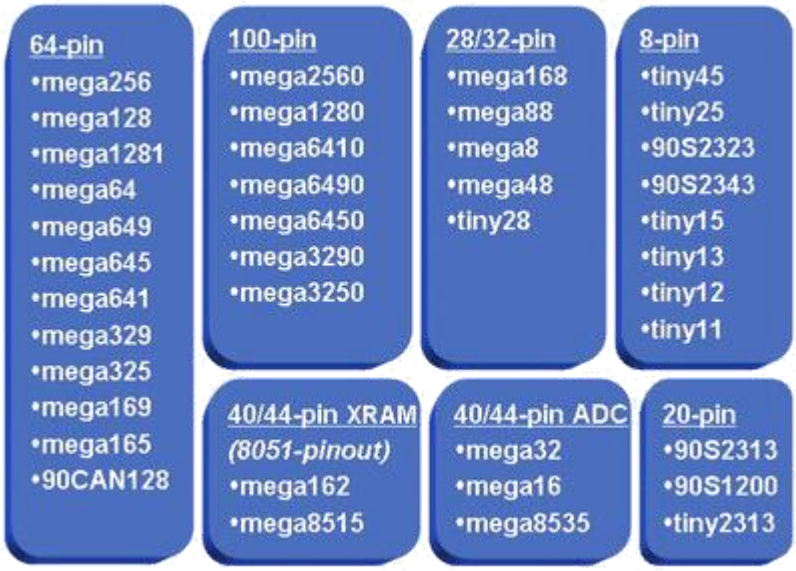


Рис. 1.2. Повиводна сумісність мікроконтролерів AVR

Можна підкреслити наступні тенденції розвитку AVR-мікроконтролерів: нарощування складності; збільшення кількості периферійних блоків; апаратну підтримку різних інтерфейсів та розширення кількості представників в сімействах. В деяких випадках відбувається виділення спеціалізованих мікроконтролерів в окремі сімейства. Як приклад можна привести сімейство AVR з апаратним драйвером (LCD AVR), яке представлено лінійкою 64-вивідних мікроконтролерів. В останні роки відбулося поповнення даного сімейства новою лінійкою мікроконтролерів в 100-вивідних корпусах.

Ще однією тенденцією розвитку AVR-мікроконтролерів можна назвати появу нових сімейств для спеціальних додатків. Даний напрям розвитку сформувало сімейство мікроконтролерів спеціального призначення (ASIC AVR), серед яких важлива поява мікроконтролерів серії AT90PWM для керування лампами денного світла і безколекторними двигунами постійного струму. Також до даного сімейства приєднаються мікроконтролери серії ATmega406, в які вперше в AVR включено два сигма-дельта АЦП з розрядністю 12 і 18 біт. ATmega406 призначено для виконання функцій моніторингу і заряду Li-Ion батарей живлення в різних портативних пристроях.

Слід зазначити постійне поліпшення характеристик енергоспоживання AVR-мікроконтролерів від покоління до покоління.

Споживаний струм у сучасних AVR-мікроконтролерах в активному режимі складає 350мкА на частоті 1МГц і напрузі живлення 1,8В. Мікроконтролери демонструють наднизькі показники за енергоспоживанням в режимах енергозбереження: 100нА в режимі Power-Down при напрузі живлення 1,8В. В нових версіях кристалів відбувається подальше зменшення енергоспоживання. Особливої уваги заслуговує той факт, що для зниження

енергоспоживання у всі AVR-мікроконтролери вводяться функції зміни частоти тактового генератора "на льоту". Для цієї мети розробники платформи AVR вводять програмований попередній дільник частоти тактового сигналу на: 1, 2, 4, 8, 16, 32, 64, 128 і 256. Відповідний регістр має 7 зарезервованих позицій, що припускає можливість появи додаткових значень попереднього дільника.

Платформа AVR, як і раніше, продовжує привертати увагу розробників збалансованим за функціональністю і вартістю набором мікроконтролерів у поєднанні з доступністю програмних і апаратних засобів підтримки розробок для них.

Фірма Atmel представила в 1993 році перший 8-розрядний FLASH-мікроконтролер і з тих пір безперервно відбувалось удосконалення технології. Прогрес даної технології спостерігався в зниженні питомого енергоспоживання (мА/МГц), розширення діапазону напруг живлення (до 1,8В) для продовження ресурсу батарейних систем, збільшення швидкодії до 16 млн. операцій в секунду, вбудовуванням емуляції в реальному масштабі часу, реалізації функції самопрограмування, вдосконалення і розширення кількості периферійних модулів, вбудовування спеціалізованих пристроїв: радіочастотний передавач, USB-контролер, драйвер ЖКІ, програмована логіка, контролер DVD, пристрої захисту даних і т. ін.

Успіх AVR-мікроконтролерів пояснюється можливістю простого виконання проекту з досягненням необхідного результату в найкоротші терміни, чому сприяє доступність великої кількості інструментальних засобів проектування, що поставляються, як безпосередньо корпорацією Atmel, так і сторонніми виробниками. Провідні сторонні виробники випускають повний спектр компіляторів, програматорів, асемблерів, відладчіків, роз'ємів і адаптерів. Відмінною рисою інструментальних засобів від Atmel є їх невисока вартість.

Іншою особливістю AVR-мікроконтролерів, що сприяло їх популяризації, це використання RISC-архітектури, яка характеризуються потужним набором інструкцій, більшість яких виконуються за один машинний цикл (1 такт). Це означає, що при рівній частоті тактового генератора вони забезпечують продуктивність в 12 разів більше продуктивності попередніх мікроконтролерів на основі CISC-архітектури (наприклад, MCS-51). AVR-мікроконтролери представляють більш широкі можливості щодо оптимізації продуктивності/енергоспоживання, що особливо важливо при розробці додатків з батарейним живленням. Мікроконтролери забезпечують продуктивність до 16 млн. операцій в секунду і підтримують FLASH-пам'ять програм досить великої ємності. AVR-архітектуру оптимізовано під мову високого рівня C, а більшість представників сімейств AVR Mega та XMega містять 8-канальний 10-розрядний АЦП, а також сумісний з IEEE 1149.1 інтерфейс JTAG або

debugWIRE для вбудованого відлагодження. Крім того, всі мікроконтролери AVR Mega та XMega з FLASH-пам'яттю можуть програмуватися через інтерфейс JTAG.

Мікроконтролери сім'ї AVR постійно розвиваються. Розроблено мікроконтролери для: USB-зв'язку; 32-розрядні мікроконтролери сімейства AVR32; мікроконтролери серії «automative»; мікроконтролери сімейства picoPower; мікроконтролери сімейства XMEGA і т. ін.

У підручнику в якості більшості прикладів розглядаються мікроконтролери сімейства MEGA, які мають базову архітектуру, яка відповідає іншим сімействам мікроконтролерів.

1.2. Характеристика AVR-мікроконтролерів

AVR-мікроконтролери є 8-бітними мікроконтролерами, які призначено для використання у вбудованих системах [1; 2; 7]. Вони виготовляються за малоспоживаючою КМОН-технологією, яка у поєднанні з удосконаленою RISC-архітектурою дозволяє досягти найкращого співвідношення вартість/швидкодія/енергоспоживання.

Мікроконтролер має швидке процесорне ядро, Flash-пам'ять програм – ROM, пам'ять даних – SRAM та EEPROM, порти введення/виведення і велику кількість програмованих периферійних модулів. Гарвардська архітектура AVR реалізує повний логічний і фізичний поділ не тільки адресних просторів, але й інформаційних шин для звернення до ROM і SRAM. Така побудова вже ближче до структури цифрових сигнальних процесорів і забезпечує істотне підвищення продуктивності. Використання однорівневого конвеєра також помітно скоротило цикл "вибірка – виконання" команди. В AVR-мікроконтролерах коротка команда виконується за один машинний цикл, який складає всього один період тактової частоти. Відмінною рисою архітектури AVR є регістровий файл швидкого доступу, що містить 32 байтових регістри загального призначення. Шість регістрів цього файлу можуть використовуватися як три 16-розрядних покажчики адреси при непрямій адресації даних (X, Y і Z Pointers), що істотно підвищує швидкість пересилання даних при роботі прикладної програми.

Flash-пам'ять програм може бути завантажена як за допомогою звичайного програматора, так і за допомогою SPI-інтерфейсу, у тому числі безпосередньо на робочій платі (функція ISP). Останні версії кристалів "Mega" та "XMega" мають можливість самопрограмування (функція SPM). Всі AVR мають також блок енергонезалежної пам'яті даних EEPROM, який доступний програмі мікроконтролера безпосередньо в ході її виконання. EEPROM звичайно використовується для збереження проміжних даних, констант, таблиць

перекодувань, каліброваних коефіцієнтів і т. ін. Ця пам’ять може бути завантажена ззовні як через SPI-інтерфейс, так і за допомогою звичайного програматора. Два програмованих біти дозволяють захистити ROM і енергонезалежну пам’ять даних EEPROM від несанкціонованого зчитування. У більшість мікроконтролерів входить внутрішня статична оперативна пам’ять SRAM. Для деяких мікроконтролерів можливе підключення зовнішньої пам'яті даних об’ємом до 64К.

Мікроконтролер має різноманітні способи синхронізації: вбудований RC-генератор з внутрішньою чи зовнішньою часозадаючою RC-ланкою, вбудований генератор із зовнішнім кварцовим або п’єзокерамічним резонатором. Внутрішній тактовий генератор може запускатися від зовнішнього генератора або кварцового резонатора, а також від внутрішнього або зовнішнього RC-ланцюга.

Усі AVR цілком статичні. Їх мінімальна робоча частота нічим не обмежена. Деякі мікроконтролери мають додатковий PLL-блок для апаратного збільшення основної тактової частоти в 16 разів. Ця частота може служити джерелом для одного з таймерів/лічильників мікроконтролера, що значно підвищує точність його роботи.

В AVR-мікроконтролери вбудовано детектор пониження напруги живлення (Brown-Out-Detector – BOD).

Мікроконтролери AVR мають декільна таймерів/лічильників загального призначення з розрядністю 8 або 16 біт. Загальні риси таймерів/лічильників наступні:

– наявність програмованого попереднього дільника вхідної частоти з різними коефіцієнтами ділення. Відміною рисою є можливість роботи таймерів/лічильників на основній тактовій частоті мікроконтролера без попереднього її зниження, що помітно підвищує точність генерації часових інтервалів системи;

– незалежне функціонування від режиму роботи процесорного ядра мікроконтролера (тобто вони можуть бути як зчитані, так і завантажені новим значенням у будь-який час);

– можливість роботи або від зовнішнього джерела опорної частоти, або як лічильник зовнішніх подій. Верхній частотний поріг визначений у цьому випадку як половина основної тактової частоти мікроконтролера. Вибір перепаду зовнішнього джерела (фронт або зріз) програмується користувачем;

– наявність різних векторів переривань для подій "переповнення вмісту", "захоплення", "порівняння";

– вартовий таймер в AVR має свій власний RC-генератор та містить програмований попередній дільник вхідної частоти, що дозволяє підлаштовувати часовий інтервал переповнення таймера і скидання мікроконтролера. Даний таймер можна програмно відключати під час роботи мікросхеми, як в активному режимі, так і в кожному з режимів зниженого енергоспоживання. В останньому випадку це приводить до значного зниження споживаного струму;

– AVR-мікроконтролери можуть використовуватись для реалізації системи реального часу (RTC). Таймер/лічильник RTC має окремий попередній дільник, що може бути програмним способом підключений або до джерела основної тактової частоти, або до додаткового асинхронного джерела опорної частоти 32,768 кГц (кварцовий резонатор або зовнішній синхросигнал). Для цієї мети зарезервовано два виводи мікросхеми.

Порти введення/виведення AVR мають значне число незалежних ліній "Вхід/вихід". Вихідні драйвери забезпечують струменеву навантажувальну здатність 20 мА на лінію порту (втікаючий струм) при максимальному значенні 40 мА, що дозволяє безпосередньо підключати до мікроконтролера світлодіоди і біполярні транзистори.

Підсистема введення/виведення AVR-мікроконтролерів має такі особливості:

– програмне конфігурування і вибір портів введення/виведення;
– виводи можуть бути запрограмовані як вхідні або як вихідні незалежно один від одного;
– вхідні буфери з тригером Шмітта на всіх виводах;
– можливість повного відключення цифрового порту введення/виведення від фізичного виводу мікросхеми;
– на всіх входах є внутрішні підтягуючі резистори з опором 20...50 кОм, що індивідуально відключаються.

AVR-мікроконтролери мають послідовні інтерфейси: USART/UART; SPI; I^2C (TWI) та CAN, що також розширює можливості їх застосування для обміну із сучасною периферією, іншими мікроконтролерами та персональними комп’ютерами.

До складу більшості МК входить аналоговий компаратор. Він має окремий вектор переривання у загальній системі переривань мікроконтролера. Тип перепаду, що викликає запит на переривання при спрацьовуванні компаратора, може бути запрограмований як фронт, зріз або перемикання. Важливою апаратною особливістю є те, що вихід компаратора може бути програмним чином підключено до входу одного з 16-розрядних таймерів/лічильників, що працює в режимі захоплення. Це дає можливість

вимірювати тривалості аналогових сигналів, а також реалізовувати АЦП двотактного інтегрування.

Мікроконтролери мають аналого-цифровий перетворювач, який побудовано за схемою послідовного наближення з пристроєм вибірки/збереження. Число незалежних каналів перетворення визначається типом мікроконтролера. Розрядність АЦП складає 10 біт. Час перетворення вибирається програмно за допомогою встановлення коефіцієнта дільника частоти, що входить до складу блоку АЦП. Важливою особливістю аналого-цифрового перетворювача є функція придушення шуму при перетворенні, коли на точність не впливають завади, що виникають при роботі процесорного ядра.

Мікроконтролери Xmega мають декілька каналів цифрово-аналогового перетворення, що значно розширює можливості їх застосування.

AVR-мікроконтролери можуть бути програмним шляхом переведені в один із шести режимів зниженого енергоспоживання. Для різних сімейств AVR і різних мікроконтролерів у межах кожного сімейства змінюються кількість і особливості цих режимів.

МК мають досить розвинуту систему команд, яка нараховує до 118 (а в останніх розробках і до 141) різних інструкцій. Майже всі команди мають фіксовану довжину в одне слово (16 біт), що дозволяє у більшості випадків поєднувати в одній команді код операції та операнд/операнди. В останніх версіях кристалів Mega та XMega реалізовано функцію множення. За різноманітністю і кількістю інструкцій AVR більше схожі на CISC-, чим на RISC-процесори. Наприклад, у PIC-контролерів система команд нараховує до 75 різних інструкцій, а в MCS-51 вона складає 111.

МК функціонують у широкому діапазоні напруг живлення: від 1,8 до 6,0 Вольт та мають комерційний і індустріальний температурні діапазони роботи.

Програмні й апаратні засоби підтримки для МК розробляються паралельно із самими кристалами і містять у собі компілятори, внутрішньосхемні емулятори, налагоджувачі, програматори і найпростіші налагоджувані плати – конструктори практично на будь-який смак. Активно йде процес співробітництва зі сторонніми фірмами, що випускають програмні засоби проектування і налагодження, операційні системи, різноманітні налагоджувані комплекси і внутрішньосхемні емулятори для AVR.

На базі AVR-мікроконтролерів в наш час широко випускаються плати Arduino, які можна використовувати для різноманітних цілей: від навчання до домашньої автоматики, від наукових цілей до комерційно доступних пристроїв,

а також багато інших застосувань, які люди вигадують та реалізовують для Arduino.

Все вище перераховане говорить про сучасність AVR-мікроконтролерів і можливість їх широкого застосування при проектуванні мікроконтролерних систем різного призначення.

На рис. 1.3 наведено структуру ядра AVR-мікроконтролерів.

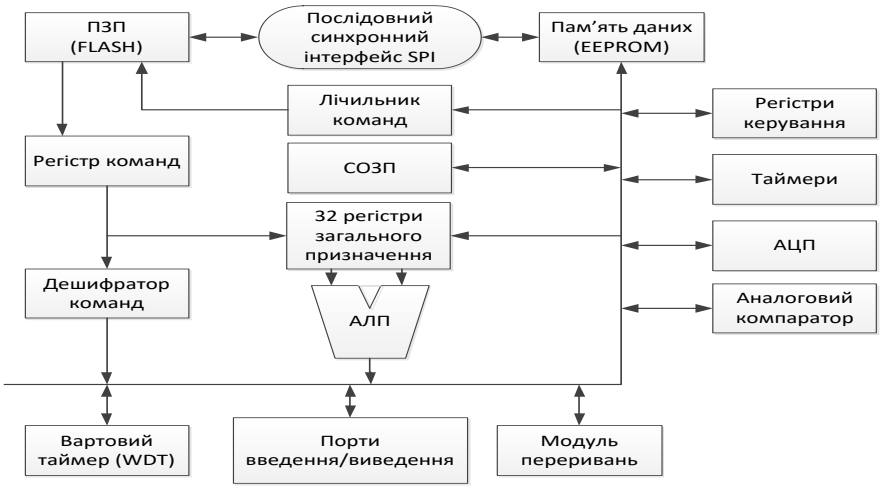


Рис. 1.3. Структура ядра AVR-мікроконтролерів

Арифметико-логічний пристрій (АЛП), що виконує усі обчислення, підключено безпосередньо до 32 робочих регістрів, які об'єднано в єдиний регістровий файл. Завдяки цьому, АЛП може виконувати одну операцію (читання вмісту регістрів, виконання операції і запис результату назад у регістровий файл) за один такт. Крім того, практично кожна з команд (за винятком команд, в яких одним з операндів є 16-бітна адреса) займає одну комірку пам'яті програм.

У мікроконтролерах реалізовано Гарвардську архітектуру, що характеризується розділенням пам'яті програм і даних, кожна з яких має власні шини доступу. Така організація дозволяє одночасно працювати як з пам'яттю програм, так і з пам'яттю даних. Поділ інформаційних шин дозволяє використовувати для кожного типу пам'яті шини різної розрядності, причому способи адресації і доступу до кожного типу пам'яті також різняться. У поєднанні з дворівневим конвеєром команд така архітектура дозволяє досягти продуктивності в 1 MIPS на кожен мегагерц тактової частоти.

1.3. Основні параметри AVR-мікроконтролерів

У сімейство AVR-мікроконтролерів на сьогоднішній день входять десятки моделей, які діляться на декілька груп в залежності від кількості виводів, максимального числа ліній введення/виведення, об`єму FLASH-пам’яті програм; EEPROM- та SRAM- пам’яті даних і т. ін. [1; 7].

Нижче, в якості прикладу, наведено параметри деяких сімей мікроконтролерів.

Tiny: 0,5 – 32 кБ FLASH-пам’яті програм; 32 – 2048 байти статичної пам'яті даних; 0 – 512 байтів енергонезалежної EEPROM-пам'яті даних; 6 – 32-вивідний корпус; обмежений набір внутрішніх периферійних пристроїв; напруга живлення 1,8 – 5,5 В.

Mega: 4 – 256 кБ FLASH-пам’яті програм; 0,25 – 32 кБ статичної пам’яті даних; 0,25 – 8 кБ нергонезалежної EEPROM-пам'яті даних; 28 – 100-вивідний корпус; розширений набір команд (команди множення, розширені методи адресації в командах LPM, SPM); розширений набір внутрішніх периферійних пристроїв; напруга живлення 1,8 – 5,5 В.

XMega: 16 – 384 кБ FLASH-пам'яті програм; 2 – 32 кБ статичної пам’яті даних; 1 – 4 кБ енергонезалежної EEPROM-пам'яті даних; 44/64/100-вивідний корпус; підвищена продуктивність за рахунок таких особливостей, як: ПДП; підтримка криптографії; «Система подій»; розширений набір внутрішніх периферійних пристроїв (наявність ЦАП); напруга живлення 1,6 – 3,6 В.

Нижче наведено позначення та властивості різних версій AVR-мікроконтролерів: AT (Mega/Tiny)xxx – базова версія; ATxxx**L** – версії контролерів, що працюють при пониженій (Low) напрузі живлення (2,7 В); ATxxx**V** – версії контролерів, що працюють при низькій напрузі живлення (1,8 В); ATxxx**P** – версії з малим енергоспоживанням (до 100 нА в режимі Power-down); застосовано технологію *picoPower*, повиводно та функціонально сумісні з попередніми версіями; ATxxx**A** – зменшений струм споживання, перекривається увесь діапазон тактових частот і напруг живлення двох попередніх версій (також, у деяких моделях, додані нові можливості та нові регістри, але збережена повна сумісність із попередніми версіями).

Номер моделі доповнюється індексом, що вказує варіант виготовлення. Цифри (8, 10, 16, 20) перед індексом означають максимальну частоту, на якій мікроконтролер може стабільно працювати при нормальній для нього напрузі живлення. Перша літера індексу означає варіант корпуса. Наступна літера індексу означає температурний діапазон і особливості виготовлення. Остання літера R означає упаковку у стрічки та котушки (Tape & Reel) для автоматизованих систем збірки.

Як приклад, основні параметри деяких мікроконтролерів сімейства, такі як об`єм пам'яті (програм і даних), число контактів введення/виведення, наявність відповідної периферії, тип корпусу, діапазон робочих частот і напруги живлення, наведено у табл. 1.1. Додатково слід зазначити, що одні моделі мікроконтролерів сімейства випускаються як в комерційному (діапазон робочих температур: 0...+70 °С), так і в промисловому (діапазон робочих температур: -40 ... +85 °С) виконаннях, а інші – тільки у промисловому.

У табл. 1.1 використано наступні позначення: I – вхід; O – вихід; I/O – вхід/вихід; P – вивід живлення; ША – шина адреси; ШД – шина даних.

Приклади розташування виводів деяких AVR-мікроконтролерів, а також опис їх виводів наведено у [7].

1.4 Структура типового AVR-мікроконтролера

Структури деяких мікроконтролерів сімейства Mega наведено у [7], а структуру одного із розвинених AVR-мікроконтролерів моделей ATmega1281x/2561x – в [1]. До цього мікроконтролера входять:

- 7 портів введення/виведення (порти A...F – 8-бітні, порт G – 6-бітний);
- два 8-бітних (T0, T2) та чотири 16-бітних (T1, T3, T4, T5) таймери/лічильники;
- 6 каналів ШІМ;
- 8-канальний 10-бітний АЦП;
- два інтерфейсні модулі USART, які можуть працювати в режимі SPI;
- по одному інтерфейсному модулю SPI та TWI;
- інтерфейс JTAG;
- і т. ін.

Таблиця 1.1. **Основні параметри AVR-мікроконтролерів**

Назва	MIPS	F МГц	Пам'ять			I/O	Таймери			ШІМ	RTC	Інтерфейси								Аналогові входи				Аналогові виходи		V В	I mA	T C
			Flash Kb	RAM Kb	EEPROM Kb		8- біт	16-біт	32-біт			UART	SPI	I^2C	USB	CAN	Ethernet	LIN	DMA	Розрядів АЦП	Кана-лів АЦП	Швидкодія АЦП	АК	Розряді в ЦАП	Каналів ЦАП			
ATtiny4	12	0…12	0.5	0.032	-	4	-	1	-	2	Hi												1			1.8-5.5		-40…85
ATtiny5	12	0…12	0.5	0.032	-	4	-	1	-	2	Hi												1			1.8-5.5		-40…85
ATtiny9	12	0…12	1	0.032	-	4	-	1	-	2	Hi												1			1.8-5.5		-40…85
ATtiny10	12	0…12	1	0,032	-	4	-	1	-	2	Hi									8	4		1			1,8…5,5	5	-40…85
ATtiny102	12	0…12	1	0.032	-	6	-	1	-	2	Hi	1								10	8		1			1,8…5,5		-40…85
ATtiny104	12	0…12	1	0.032	-	12	-	1	-	2	Hi	1								10	8		1			1,8…5,5		-40…85
ATtiny20	12	0…12	2	0.128	-	12	1	1	-	4	Hi		1	1						10	8		1			1,8…5,5		-40…85
ATtiny40	12	0…12	4	0.256	-	18	1	1	-	4	Hi		1	1						10	12		1			1,8…5,5		-40…85
ATtiny24A	20	0…20	2	0.128	0.128	12	1	1	-	2	Hi		1	1						10	8		1			1,8…5,5		-40…85
ATtiny44A	20	0…20	4	0.256	0.256	12	1	1	-	2	Hi		1	1						10	8		1			1,8…5,5		-40…85
ATtiny84A	20	0…20	8	0.512	0.512	12	1	1	-	2	Hi		1	1						10	8		1			1,8…5,5		-40…85
ATtiny25(V)	20	0…20	2	0.128	0.128	6	2	-	-	5	Hi		1	1						10	4		1			1,8…5,5		-40…85
ATtiny45(V)	20	0…20	4	0.256	0.256	6	2	-	-	5	Hi		1	1						10	4		1			1,8…5,5		-40…85
ATtiny85(V)	20	0..20	8	0.512	0.512	6	2	-	-	5	Hi		1	1						10	4		1			1,8…5,5		-40…85
ATtiny48	12	0…12	4	0.256	0.064	24	1	1	-	4	Hi		1	1						10	6.8		1			1,8…5,5		-40…85
ATtiny88	12	0…12	8	0.512	0.064	28	1	1	-	4	Hi		1	1						10	6.8		1			1,8…5,5		-40…85
ATtiny87	16	0…16	8	0.512	0.512	16	1	1	-	3	Hi	1	2	1						10	11		1			1,8…5,5		-40…85
ATtiny167	16	0…16	16	0.512	0.512	16	1	1	-	3	Hi	1	2	1						10	11		1			1,8…5,5		-40…85
ATtiny261A	20	0..20	2	0.128	0.128	16	1	1	-	6	Hi	1	1							10	11		1			1,8…5,5		-40…105
ATtiny461A	20	0..20	4	0.256	0.256	16	1	1	-	3	Hi	1	1							10	11		1			1,8…5,5		-40…105
ATtiny861A	20	0..20	8	0.512	0.512	16	1	1	-	3	Hi	1	1							10	11		1			1,8…5,5		-40…105
ATtiny202	20	0..20	2	0.128	0.064	6	-	2	-	+		1	1	1						10	12		1			1,8…5,5		-40…125
ATtiny204	20	0..20	2	0.128	0.064	12	-	2	-	+		1	1	1						10	12		1			1,8…5,5		-40…125
ATtiny402	20	0..20	4	0.256	0.128	6	-	2	-	+		1	1	1						10	12		1			1,8…5,5		-40…125
ATtiny404	20	0..20	4	0.256	0.128	12	-	2	-	+		1	1	1						10	12		1			1,8…5,5		-40…125
ATtiny406	20	0..20	4	0.256	0.128	12	-	2	-	+		1	1	1						10	12		1			1,8…5,5		-40…125
ATtiny804	20	0..20	8	0.512	0.128	12	-	2	-	+		1	1	1						10	12		1			1,8…5,5		-40…125
ATtiny806	20	0..20	8	0.512	0.128	18	-	2	-	+		1	1	1						10	12		1			1,8…5,5		-40…125
ATtiny807	20	0..20	8	0.512	0.128	22	-	2	-	+		1	1	1						10	12		1			1,8…5,5		-40…125

Продовження табл. 1.1.

Назва	MIPS	F МГц	Пам'ять			I/O	Таймери			ШІМ	RTC	Інтерфейси								Аналогові входи				Аналогові виходи		V В	I mA	T C
			Flash Kb	RAM Kb	EEPROM Kb		8- біт	16-біт	32-біт			UART	SPI	I²C	USB	CAN	Ethernet	LIN	DMA	Розрядів АЦП	Кана-лів АЦП	Швидкодія АЦП	АК	Розряді в ЦАП	Каналів ЦАП			
ATtiny1604	20	0..20	16	1	0.256	12	-	2	-	+		1	1	1						10	12		1			1,8…5,5		-40…125
ATtiny1606	20	0..20	16	1	0.256	18	-	2	-	+		1	1	1						10	12		1			1,8…5,5		-40…125
ATtiny1607	20	0..20	16	1	0.256	22	-	2	1	+		1	1	1						10	12		1			1,8…5,5		-40…125
ATtiny212	20	0..20	2	0.128	0.064	6	1	2	-	+		1	1	1						10	12		1			1,8…5,5		-40…125
ATtiny214	20	0..20	2	0.128	0.064	12	1	2	-	+		1	1	1						10	12		1			1,8…5,5		-40…125
ATtiny412	20	0..20	4	0.256	0.128	6	-	2	-	+		1	1	1						10	12		1			1,8…5,5		-40…125
ATtiny414	20	0..20	4	0.256	0.128	12	1	2	-	+		1	1	1						10	12		1			1,8…5,5		-40…125
ATtiny416	20	0..20	4	0.256	0.128	18	1	2	-	+		1	1	1						10	12		1			1,8…5,5		-40…125
ATtiny417	20	0..20	4	0.256	0.128	22	-	2	-	6		1	1	1						10	12		1			1,8…5,5		-40…125
ATtiny814	20	0..20	8	0.256	0.128	12	1	2	-	6		1	1	1						10	12		1	8		1,8…5,5		-40…125
ATtiny816	20	0..20	8	0.512	0.128	18	1	2	-	6		1	1	1						10	12		1	8		1,8…5,5		-40…125
ATtiny817	20	0..20	8	0.512	0.128	22	1	2	-	6		1	1	1						10	12		1	8		1,8…5,5		-40…125
ATtiny1614	20	0..20	16	2	0.256	12	1	3	-	6		1	1	1						10	12		1	8		1,8…5,5		-40…125
ATtiny1616	20	0..20	16	2	0.128	18	1	3	-	6		1	1	1						10	16		1	8		1,8…5,5		-40…125
ATtiny1617	20	0..20	16	2	0.256	22	1	3	-	6		1	1	1						10	20		3	8		1,8…5,5		-40…125
ATtiny3216	20	0..20	32	2	0.256	18	1	3	-	6		1	1	1						10	16		3	8		1,8…5,5		-40…125
ATtiny3217	20	0..20	32	2	0.256	22	1	3	-	6		1	1	1						10	20		3	8		1,8…5,5		-40…125
ATtiny441	16	0…16	4	0.256	0.256		1	2		6		2	1	1						10	12		2			1,7…5,5		-40…85
ATtiny841	16	0…16	8	0.512	0.512		1	2		6		2	1	1						10	12		2			1,7…5,5		-40…85
ATtiny1634	16	0…16	16	1	0.256		1	1		2		2		1						10	12		1			1,8…5,5		-40…85
ATtiny2313A	20	0..20	2	0.128	0.128		1	1		4		1	2	1						10	12		1			1,8…5,5		-40…85
ATmega8A	16	0…16	8	1	0.512	30	2	1	-	3		1	1	1						10	6-8		1			2.7…5.5		-40…85
ATmega16A	16	0…16	16	1	0.512	32	2	1	-	4		1	1	1						10	8		1			2.7…5.5		-40…85
ATmega32A	16	0…16	32	2	1	32	2	1	-	4		1	1	1						10	8		1			2.7…5.5		-40…85
ATmega8U2	16	0…16	8	0.512	0.512	22	1	1	-	5		1	2										1			2.7…5.5		-40…85
ATmega16U2	16	0…16	16	0.512	0.512	22	1	1	-	5		1	2										1			2.7…5.5		-40…85
ATmega32U2	16	0…16	32	1	1	22	1	1	-	5		1	2										1			2.7…5.5		-40…85
ATmega16U4	16	0…16	16	1.25	0.512	26	2	2	-	12		1	2	1						10	12		1			2.7…5.5		-40…85
ATmega32U4	16	0…16	32	2.5	0.512	26	2	2	-	12		1	2	1						10	12		1			2.7…5.5		-40…85

Продовження табл. 1.1

Назва	MIPS	F МГц	Пам'ять			I/O	Таймери			ШІМ	RTC	Інтерфейси								Аналогові входи				Аналогові виходи		V В	I mA	T C
			Flash Kb	RAM Kb	EEPROM Kb		8- біт	16-біт	32-біт			UART	SPI	I²C	USB	CAN	Ethernet	LIN	DMA	Розрядів АЦП	Кана-лів АЦП	Швидкодія АЦП	АК	Розряді в ЦАП	Каналів ЦАП			
ATmega48PB	20	0…20	4	0.512	0.256	27	2	-	-	6		1	2	1						10	8		1			1.8…5.5		-40…85
ATmega88PB	20	0…20	8	1	0.512	27	2	-	-	6		1	2	1						10	8		1			1.8…5.5		-40…85
ATmega168PB	20	0…20	16	1	0.512	27	2	-	-	6		1	2	1						10	8		1			1.8…5.5		-40…85
ATmega328PB	20	0…20	32	2	1	27	2	3	-	10		2	2	2						10	8		1			1.8…5.5		-40…85
ATmega3208	20	0…20	32	4	0.256	27	-	4	-	9		3	1	1						10	12		1			1.8…5.5		-40…85
ATmega3209	20	0…20	32	4	0.256	41	-	5	-	11		4	1	1						10	16		1			1.8…5.5		-40…85
ATmega4808	20	0…20	48	6	0.256	27	-	4	-	9		3	1	1						10	12		1			1.8…5.5		-40…85
ATmega4809	20	0…20	48	6	0.256	41	-	5	-	11		4	1	1						10	16		1			1.8…5.5		-40…85
ATmega64A	16	0…16	64	4	2	53	2	2	-	8		2	1	1						10	8		1			1.8…5.5		-40…85
ATmega128A	16	0…16	128	4	4	53	2	2	-	8		2	1							10	8		1			1.8…5.5		-40…85
ATmega164PA	20	0…20	16	1	0.512	32	2	1	-	6		2	3	1						10	8		1			1.8…5.5		-40…85
ATmega324PA	20	0…20	32	2	1	32	2	1	-	6		2	3	1						10	8		1			1.8…5.5		-40…85
ATmega644PA	20	0…20	64	4	2	32	2	1	-	6		2	3	1						10	8		1			1.8…5.5		-40…85
ATmega1284P	20	0…20	128	16	4	32	2	1	-	6		2	3	1						10	8		1			1.8…5.5		-40…85
ATmega165PA	16	0…16	16	1	0.512	54	2	1	-	4		1	2	1						10	8		1			1.8…5.5		-40…85
ATmega325PA	20	0…20	32	2	1	54	2	1	-	4		1	2	1						10	8		1			1.8…5.5		-40…85
ATmega645P	16	0…16	64	4	2	54	2	1	-	4		1	2	1						10	8		1			1.8…5.5		-40…85
ATmega169PA	16	0…16	16	1	0.512	54	2	1	-	4		1	2	1						10	8		1			1.8…5.5		-40…85
ATmega329PA	20	0…20	32	2	1	54	2	1	-	4		1	2	1						10	8		1			1.8…5.5		-40…85
ATmega649P	16	0…16	64	4	2	54	2	1	-	4		1	2	1						10	8		1			1.8…5.5		-40…85
ATmega324PB	20	0…20	32	2	1	27	2	3	-	10		3	2	2						10	8		1			1.8…5.5		-40…85
ATmega640	16	0…16	64	8	4	86	2	4	-	16		4	5	1						10	16		1			1.8…5.5		-40…85
ATmega1280	16	0…16	64	8	4	86	2	4	-	16		4	5	1						10	16		1			1.8…5.5		-40…85
ATmega2560	16	0…16	256	8	4	86	2	4	-	16		4	5	1						10	16		1			1.8…5.5		-40…85
ATmega1281	16	0…16	128	8	4	54	2	4	-	16		2	3	1						10	16		1			1.8…5.5		-40…85
ATmega2561	16	0…16	256	16	8	54	2	4	-	16		2	3	1						10	16		1			1.8…5.5		-40…85
ATmega3290PA	20	0…20	32	2	1	69	2	1	-	4		2	3	1						10	8		1			1.8…5.5		-40…85
ATmega6490P	20	0…20	64	4	2	69	2	1	-	4		1	2	1						10	8		1			1.8…5.5		-40…85
ATmega3250PA	20	0…20	32	2	1	69	2	1	-	4		1	2	1						10	8		1			1.8…5.5		-40…85

Закінчення табл. 1.1

Назва	MIPS	F МГц	Пам'ять			I/O	Таймери			ШІМ	RTC	Інтерфейси								Аналогові входи				Аналогові виходи		V В	I mA	T C
			Flash Kb	RAM Kb	EEPROM Kb		8- біт	16-біт	32-біт			UART	SPI	I²C	USB	CAN	Ethernet	LIN	DMA	Розрядів АЦП	Кана-лів АЦП	Швидкодія АЦП	АК	Розрядів ЦАП	Каналів ЦАП			
ATXMega1281U	32	0…32	128	8	2	78		8		24		8	12	4	1					12	16	2	4	12	2	1,6…3,6		-40…85
ATXMega64A1U	32	0…32	64	4	2	78		8		24		8	12	4	1					12	16	2	4	12	2	1,6…3,6		-40…85
ATXMega256A3U	32	0…32	256	8	4	50		7		22		7	10	2	1					12	16	2	4	12	2	1,6…3,6		-40…85
ATXMega192A3U	32	0…32	192	8	2	50		7		22		7	10	2	1					12	16	2	4	12	2	1,6…3,6		-40…85
ATXMega128A3U	32	0…32	128	8	2	50		7		22		7	10	2	1					12	16	2	4	12	2	1,6…3,6		-40…85
ATXMega64A3U	32	0…32	64	4	2	50		7		22		7	10	2	1					12	16	2	4	12	2	1,6…3,6		-40…85
ATXMega128A4U	32	0…32	128	8	2	34		5		16		5	7	2	1					12	12	2	2	12	2	1,6…3,6		-40…85
ATXMega64A4U	32	0…32	64	4	2	34		5		16		5	7	2	1					12	12	2	2	12	2	1,6…3,6		-40…85
ATXMega32A4U	32	0…32	32	4	1	34		5		16		5	7	2	1					12	12	2	2	12	2	1,6…3,6		-40…85
ATXMega16A4U	32	0…32	16	2	1	34		5		16		5	7	2	1					12	12	2	2	12	2	1,6…3,6		-40…85
ATXMega B3	32	0…32	128	4	2	36		2		6		1	2	1	1					12	8		2			1,6…3,6		-40…85
ATXMega32D3	32	0…32	32	4	1	50		5		18		3	5	2						12	16		2			1,6…3,6		-40…85
ATXMega64D3	32	0…32	64	4	2	50		5		18		3	5	2						12	16		2			1,6…3,6		-40…85
ATXMega128D3	32	0…32	128	8	2	50		5		18		3	5	2						12	16		2			1,6…3,6		-40…85
ATXMega192D3	32	0…32	192	16	2	50		5		18		3	5	2						12	16		2			1,6…3,6		-40…85
ATXMega256D3	32	0…32	256	16	4	50		5		18		3	5	2						12	16		2			1,6…3,6		-40…85
ATXMega384D3	32	0…32	384	32	4	50		5		18		3	5	2						12	16		2			1,6…3,6		-40…85
ATXMega16C4	32	0…32	16	2	1	34		4		14		2	4	2	1					12	16		2			1,6…3,6		-40…85
ATXMega32C4	32	0…32	32	4	1	34		4		14		2	4	2	1					12	16		2			1,6…3,6		-40…85
ATXMega128D4	32	0…32	128	8	2	34					1	2	4	2						12	12		2			1,6…3,6		-40…85
ATXMega64D4	32	0…32	64	4	2	34					1	2	4	2						12	12		2			1,6…3,6		-40…85
ATXMega32D4	32	0…32	32	4	1	34					1	2	4	2						12	12		2			1,6…3,6		-40…85
ATXMega16D4	32	0…32	16	2	1	34					1	2	4	2						12	12		2			1,6…3,6		-40…85
ATXMega32E5	32	0…32	32	1	0.5	26	2	1		8		2	1	1						12	16		2	12	2			
ATXMega16E5	32	0…32	16	2	0.5	26	2	1		8		2	1	1						12	16		2	12	2			
ATXMega8E5	32	0…32	8	4	1	26	2	1		8		2	1	1						12	16		2	12	2			

Контрольні запитання та завдання

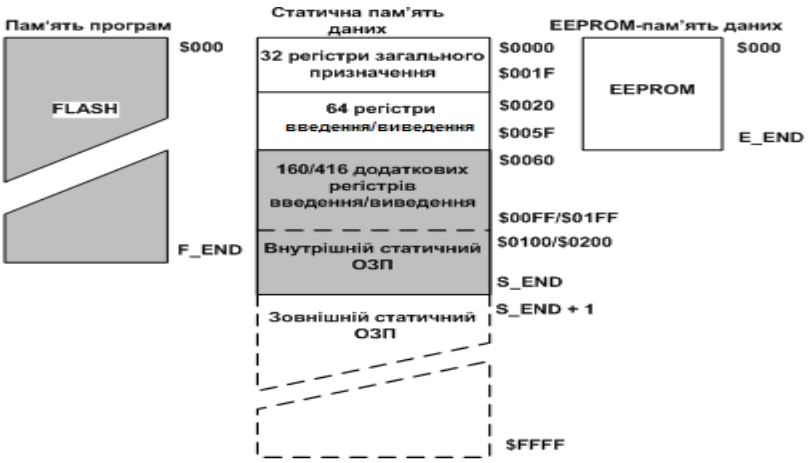
1. З якої серії почався випуск «Традиційних» AVR-мікроконтролерів?
2. На які три сімейства сьогодні розділився випуск «Традиційних» AVR-мікроконтролерів?
3. Опишіть рівень сумісності AVR-мікроконтролерів різних серій.
4. Як забезпечується програмна сумісності AVR-мікроконтролерів?
5. Опишіть тенденції розвитку AVR-мікроконтролерів.
6. Яку розрядність мають операнди більшості AVR-мікроконтролерів?
7. Опишіть особливості «Гардварської архітектури» мікроконтролера.
8. За скільки тактів виконуються більшість команд МК?
9. Яку розрядність має одна комірка пам`яті програм?
10. Опишіть особливості підсистеми введення/виведення МК.
11. Опишіть структуру ядра AVR-мікроконтролерів.
12. Що являє собою регістровий файл МК?
13. Опишіть структуру AVR-мікроконтролерів моделей ATmega1281x/2561x.
14. Яку кількість таймерів може мати AVR-мікроконтролер?
15. В якому діапазоні напруг живлення функціонують AVR-мікроконтролери?
16. Яку розрядність має АЦП AVR-мікроконтролерів?
17. Назвіть кількість команд, яку можуть виконувати AVR-мікроконтролери.
18. Назвіть максимальний об`єм FLASH-пам'яті програм AVR-мікроконтролерів сімейства Mega.
19. Назвіть максимальний об`єм EEPROM-пам'яті даних AVR-мікроконтролерів сімейства Mega.
20. Які послідовні інтерфейси мають AVR-мікроконтролери?
21. Яку струменеву навантажувальну здатність забезпечують вихідні драйвери портів AVR-мікроконтролерів?
22. Опишіть особливості позначень різних версій AVR-мікроконтролерів.
23. Опишіть особливості таймера/лічильника RTC.
24. Опишіть загальні риси таймерів/лічильників AVR-мікроконтролерів.
25. Які способи синхронізації мають AVR-мікроконтролери?
26. Для чого можуть використовуватись плати Arduino? На базі яких МК вони виконуються?
27. Яку розряднісь мають паралельні порти AVR-мікроконтролерів?
28. МК якого сімейства AVR-мікроконтролерів мають модулі ЦАП?

2. ОРГАНІЗАЦІЯ ПАМ’ЯТІ

2.1. Загальна характеристика

В AVR-мікроконтролерах реалізовано Гарвардську архітектуру, у відповідності з якою розділено не лише адресні простори пам’яті програм та пам’яті даних, але також і шини доступу до них [1; 7]. Способи адресації та доступу до цих областей пам’яті також є різними. Така структура дає змогу центральному процесору одночасно працювати як з пам’яттю програм так і з пам’яттю даних, що суттєво збільшує продуктивність. Кожна з областей пам’яті даних – статична пам`ять даних (СПД) та EEPROM-пам`ять також розташовані у своєму адресному просторі.

На рис. 2.1 наведено карту пам’яті деяких AVR-мікроконтролерів сімейства Mega.



Модель	Пам'ять програм (FLASH)		Статична пам'ять даних (СПД)				Пам'ять даних (EEPROM)	
	Верхня границя [F_END]	Об'єм [слів]	Верхня Границя [S_END]	Об'єм СОЗП [байт]	Кількість дод. регістрів введ./вивед.	Зовнішній СОЗП	Верхня границя [F_END]	Об'єм [байт]
ATmega48x	$007FF	2 K	$02FF	512	160		$0FF	256
ATmega8515x	$00FFF	4 K	$025F	512	0	•	$1FF	512
ATmega8535x	$00FFF	4 K	$025F	512	0		$1FF	512
ATmega8x	$00FFF	4 K	$045F	1 K	0		$1FF	512
ATmega88x	$00FFF	4 K	$04FF	1 K	160		$1FF	512
ATmega16x	$01FFF	8 K	$045F	1 K	0		$1FF	512
ATmega162x	$01FFF	8 K	$04FF/$045F	1 K	160/0	•	$1FF	512
ATmega164x	$01FFF	8 K	$04FF	1 K	160		$1FF	512
ATmega165x	$01FFF	8 K	$04FF	1 K	160		$1FF	512
ATmega168x	$01FFF	8 K	$04FF	1 K	160		$1FF	512
ATmega32x	$03FFF	16 K	$085F	2 K	0		$3FF	1 K
ATmega324x	$03FFF	16 K	$08FF	2 K	160		$3FF	1 K
ATmega325x	$03FFF	16 K	$08FF	2 K	160		$3FF	1 K
ATmega3250x	$03FFF	16 K	$08FF	2 K	160		$3FF	1 K
ATmega64x	$07FFF	32 K	$10FF	4 K	160	•	$7FF	2 K
ATmega640x	$07FFF	32 K	$21FF	8 K	416	•	$FFF	4 K
ATmega644x	$07FFF	32 K	$10FF	4 K	160		$7FF	2 K
ATmega645x	$07FFF	32 K	$10FF	4 K	160		$7FF	2 K
ATmega6450x	$07FFF	32 K	$10FF	4 K	160		$7FF	2 K
ATmega128x	$0FFFF	64 K	$10FF	4 K	160	•	$FFF	4 K
ATmega1280x	$0FFFF	64 K	$21FF	8 K	416	•	$FFF	4 K
ATmega1281x	$0FFFF	64 K	$21FF	8 K	416	•	$FFF	4 K
ATmega2560x	$1FFFF	128 K	$21FF	8 K	416	•	$FFF	4 K
ATmega2561x	$1FFFF	128 K	$21FF	8 K	416	•	$FFF	4 K

Рис. 2.1. Карта пам’яті деяких мікроконтролерів сімейства Mega

Примітка: позначкою «•» відмічені моделі, до яких можна підключити зовнішній СОЗП.

Оскільки AVR-мікроконтролери мають 16-бітну систему команд, об’єм пам’яті програм на рисунку вказано не у байтах, а в 16-бітових словах. Символ «$» є ознакою шістнадцяткової системи числення.

2.2. Організація пам’яті програм

В мікропроцесорних системах в пам’яті програм зберігаються команди, що керують роботою мікроконтролера, та константи, що не змінюються під час роботи програми.

Пам’ять програм має шістнадцятибітну організацію, для якої довжина кожної команди кратна одному слову – 16 біт. Наприклад, об’єм пам’яті програм мікроконтролерів сімейства Mega складає від 2К (2*1024) до більш ніж 128К (128*1024) шістнадцятибітних слів. У частини моделей мікроконтролерів сімейства Mega пам’ять програм логічно поділено на 2 рівні частини: область прикладної програми і область завантажувача [1; 7].

В останній може розташовуватися спеціальна програма – завантажувач, що дозволяє мікроконтролеру самостійно керувати завантаженням та вивантаженням прикладних програм. Використання цієї області та реалізація програми-завантажувача будуть розглянуті нижче окремо. Якщо самопрограмування мікроконтролера не використовується, прикладна програма може розташовуватися і в області завантажувача.

Для адресації пам'яті програм використовується лічильник команд (Program Counter – PC). Розмір лічильника команд залежить від об'єму пам'яті, що адресується.

За адресою $0000 пам’яті програм знаходиться вектор скидання. Після ініціалізації (скидання) мікроконтролера виконання програми починається з цієї адреси. За цією адресою повинна розміщуватися команда переходу до основної частини ініціалізації програми. Починаючи з адреси $0001 пам’яті програм (моделі з пам’яттю програм 8 Кбайт і менше) або $0002 (останні моделі) розташовується таблиця векторів переривань. Розмір цієї області залежить від моделі мікроконтролера (розподіл області векторів переривання буде розглянуто окремо).

При виникненні переривання після збереження у стеку поточного значення лічильника команд відбувається виконання команди, розташованої за адресою відповідного вектора. За цими адресами розташовуються команди переходу до підпрограм обробки переривань. В моделях з пам’яттю програм невеликого об'єму (8 Кбайт і менше) в таблицях векторів переривань використовуються команди відносного переходу – RJMP, а в останніх моделях – команди абсолютного переходу – JMP.

У більшості мікроконтролерів сімейства Mega положення вектора скидання і таблиці векторів переривань може бути змінено. Вони можуть розташовуватися не лише на початку пам’яті програм, як описано вище, але і на початку області завантажувача.

В якості пам’яті програм AVR-мікроконтролерів використовується FLASH-ПЗП, який розрахований, як мінімум, на 10000 циклів очищення/запису.

2.3. Організація пам’яті даних

2.3.1. Загальна характеристика

Пам’ять даних AVR-мікроконтролерів розділено на три частини: регістрова пам’ять, статичний оперативний запам`ятовуючий пристрій (СОЗП) і енергонезалежна EEPROM-пам’ять.

Регістрова пам’ять включає 32 регістри загального призначення (РЗП), об'єднаних у файл, і 64 службові регістри введення/виведення (РВВ). В моделях з розвиненою периферією є також область додаткових (extended) регістрів введення/виведення (ДРВВ). Під РВВ в пам’яті мікроконтролера відводиться 64 байти, а під ДРВВ – 160 або 416 байт (залежно від моделі).

В обох областях регістрів введення/виведення розташовуються різні службові регістри: регістри керування мікроконтролером, регістри стану і т. ін., а також регістри керування периферійними пристроями, що входять до складу мікроконтролера. Загальна кількість РВВ і ДРВВ залежить від конкретної моделі мікроконтролера.

Для зберігання змінних окрім регістрів загального призначення використовується також СОЗП. Ряд мікроконтролерів сімейства, крім того, мають можливість підключення зовнішнього статичного ОЗП об’ємом до 64 Кбайт.

Регістрову пам`ять разом із СОЗП називають статичною пам`ятю даних (СПД). Фізично таку пам’ять виконано на основі тригерів, тому вона є енергозалежною – зберігає інформацію, доки є живлення.

Для довготривалого зберігання різної інформації, яка може змінюватися в процесі функціонування готової системи (калібрувальні константи, серійні номери, ключі і т. ін.), в мікроконтролерах сімейства може використовуватися вбудована енергонезалежна EEPROM-пам’ять. Її об’єм складає для різних моделей від 256 до більш ніж 4 Кбайт. EEPROM-пам’ять розташовано в окремому адресному просторі, а доступ до неї здійснюється за допомогою відповідних РВВ.

2.3.2. Статична пам’ять даних

В AVR-мікроконтролерах використовується лінійна організація статичної пам’яті даних (рис. 2.2).

До складу статичної пам’яті даних входять регістри загального призначення (РЗП), регістри введення/виведення (РВВ), а також статичний ОЗП (СОЗП). Максимальний об'єм СПД складає 65536 байт та залежить від конкретної моделі сімейства.

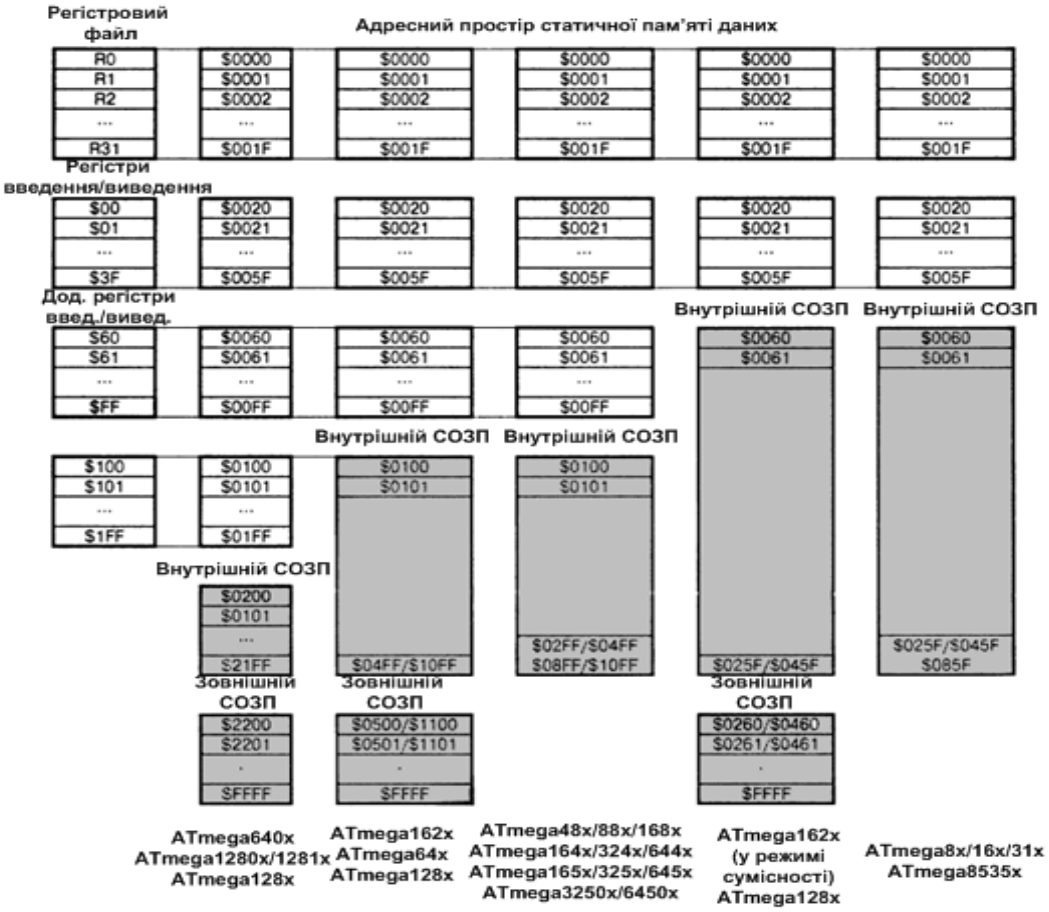


Рис. 2.2. Організація статичної пам’яті даних

Регістри загального призначення

32 регістри загального призначення об’єднано в регістровий файл швидкого доступу, структуру якого показано на рис. 2.3.

В AVR-мікроконтролерах усі РЗП безпосередньо доступні арифметико-логічному пристрою (АЛП), на відміну від 8-бітових мікроконтролерів деяких інших фірм, в яких є лише один такий регістр – робочий регістр A/W (акумулятор). Завдяки цьому будь-який РЗП може використовуватися практично у всіх командах і як операнд-джерело, і як операнд-приймач. Таке рішення (у поєднанні з конвеєрною обробкою) дозволяє АЛП виконувати за один такт одну операцію: читання операндів з регістрового файлу, виконання команди і запис результату назад у регістровий файл.

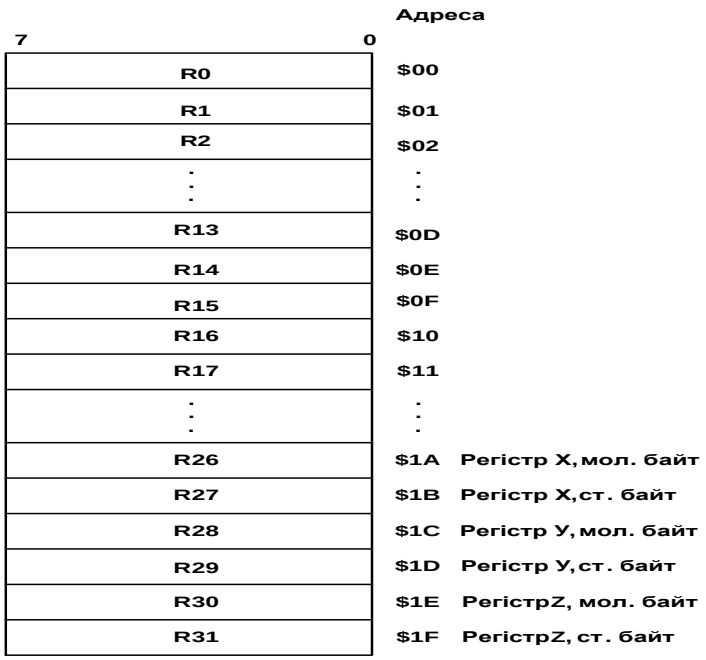


Рис. 2.3. Структура регістрів загального призначення

Останні 6 регістрів файлу (R26...R31) можуть об'єднуватися у три 16-бітних регістри: X, Y та Z (рис. 2.4), що використовуються як покажчики при непрямій адресації статичної пам'яті даних.

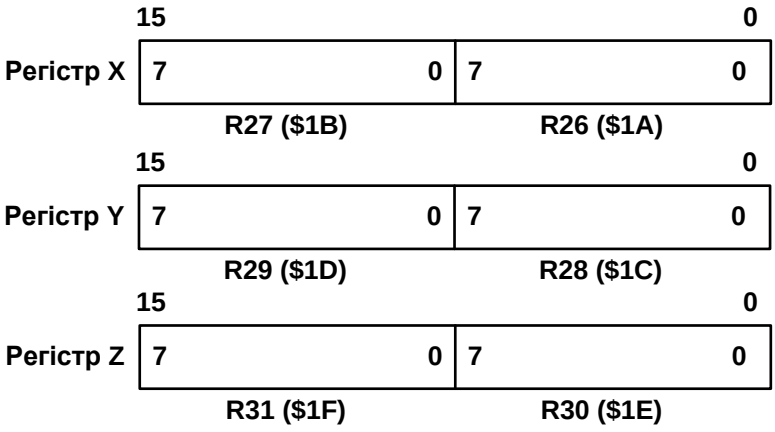


Рис. 2.4. Регістри-покажчики X,Y та Z

Як показано на рис. 2.4, кожен регістр файлу має свою власну адресу в просторі статичної пам'яті даних. Тому до них можна звертатися двома способами – як до регістрів та як до статичної пам'яті, не дивлячись на те, що фізично ці регістри не є комірками СОЗП. Таке рішення є ще однією відмінною особливістю архітектури AVR, що підвищує ефективність роботи мікроконтролера і його продуктивність.

Стек

В мікропроцесорних системах існують два різновиди стека: перший, який умовно можна назвати «апаратним» та другий, який будемо називати – «програмним». В «апаратному» стеку зберігаються адреси повернення із підпрограм, а «програмний» стек також призначено для тимчасового зберігігання частин СПД, які використовуються підпрограмами. Для роботи з «програмним» стеком використовуються команди: PUSH та POP. AVR-мікроконтролери мають обидва різновиди стека (в залежності від моделі). У цьому разі стек розміщується у статичному ОЗП, і його глибина визначається тільки розміром вільної області пам'яті.

В залежності від ємності СОЗП, як покажчик стека використовується або один регістр введення/виведення SPL (SP), розташований за адресою $3D ($5D), або пара регістрів SPH:SPL, розташованих за адресами $3E ($5E) і $3D ($5D) відповідно.

Регістри-покажчики стека є звичайними регістрами введення/виведення і, відповідно, програмно доступні. В наборі команд AVR-мікроконтролерів є команди занесення в стек – PUSH і зчитування зі стека – POP, що дозволяє програмі використовувати стек для своїх потреб. Оскільки після подачі напруги живлення або після скидання, покажчик стека дорівнює нулю, на початку програми його необхідно проініціалізувати, записавши в нього значення верхньої вільної адреси стекової пам'яті даних (зазвичай, максимальна адреса СПД). Стек заповнюється у напряму зменшення значення регістра SP. Початкове значення SP адресує вершину стека – першу вільну комірку стекової пам'яті.

При виклику підпрограм адреса команди, розташованої за командою виклику, зберігається у стеку. Значення покажчика стека при цьому зменшується на 2, тому що для збереження лічильника команд потрібно 2 байти. При поверненні з підпрограми ця адреса відновлюється зі стека і завантажується в лічильник команд. Значення покажчика стека відповідно збільшується на 2. Те ж відбувається і під час переривання. При виникненні переривання адреса наступної команди також зберігається у стеку, а при поверненні з підпрограми обробки переривання, вона відновлюється зі стека.

Регістри введення/виведення

Всі регістри введення/виведення умовно можна розділити на дві групи: службові регістри мікроконтролера і регістри, що відносяться до конкретних периферійних пристроїв (у тому числі регістри портів введення/виведення).

В AVR-мікроконтролерах частина регістрів введення/виведення розташовуються в так званому основному просторі введення/виведення, розміром

64 байти. У більшості моделей сімейства є також простір додаткових регістрів введення/виведення розміром 160 або 416 байт. Введення додаткових РВВ пов'язане з тим, що для підтримки всіх периферійних пристроїв, наявних в цих моделях, адрес перших 64-х РВВ недостатньо.

Кількість та розподіл адрес простору введення/виведення (як основного, так і додаткового) залежить від конкретної моделі мікроконтролера або, якщо точніше, від складу і можливостей периферійних пристроїв даної моделі [1; 7].

При вказівці адрес деяких РВВ, в дужках вказуються відповідні їм адреси додаткових РВВ – комірок СПД. Відповідно, якщо адреса регістра вказується лише в дужках, цей регістр розташовано в просторі додаткових РВВ. Якщо адреса в таблиці РВВ не вказана, це значить, що для даної моделі він зарезервований, і запис за цією адресою заборонено (для сумісності з майбутніми моделями). Якщо адреса, наприклад регістра SREG вказана як $3F ($5F), то це означає що за адресою $3F можна відповідними командами звернутися до РВВ основного простору регістрів введення/виведення, розміром 64 байти, а за адресою $5F можна звернутися до РВВ, як частині простору СПД. Різниця між цими адресами складає $20, тому що у просторі СПД перед РВВ знаходяться 32 РЗП.

Приклади регістрів введення/виведення деяких моделей Mega-AVR-мікроконтролерів наведено у [1; 7].

До регістрів введення/виведення, розташованих в основному просторі введення/виведення, можна безпосередньо звернутися за допомогою команд IN і OUT, що виконують пересилання даних між одним з 32-х РЗП і регістром основного простору введення/виведення. В системі команд є також чотири команди побітового доступу, що використовують як операнди регістри введення/виведення: команди встановлення/скидання окремого біта – SBI і CBI) і команди перевірки стану окремого біта – SBIS і SBIC. Ці команди можуть звертатися лише до першої половини основного простору введення/виведення – адреси $00...$1F.

Окрім адресації до регістрів основного простору введення/виведення за допомогою команд IN і OUT, до РВВ можна звертатися як до комірок СПД за допомогою команд ST/SD/SDD і LD/LDS/LDD (для додаткових РВВ тільки цей спосіб є можливим).

Серед РВВ є один регістр, що часто використовується в процесі виконання програм – регістр стану SREG. Він входить до складу регістрів основного простору введення/виведення та розташовується за адресою $3F ($5F) і містить набір прапорців, що показують поточний стан мікроконтролера. Більшість прапорців при настанні певних подій відповідно до результату виконання команд автоматично встановлюються в одиницю або скидаються в нуль. Всі біти цього

регістра доступні як для читання, так і для запису. Після скидання мікроконтролера вони скидаються в нуль. Формат регістра SREG показано на рис. 2.5, а опис його його окремих розрядів – в табл. 2.1.

	7	6	5	4	3	2	1	0
	I	T	H	S	V	N	Z	C
Читання(R)\Запис(W)	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Початкове значення	0	0	0	0	0	0	0	0

Рис. 2.5. Формат регістра стану SREG

В останніх моделях мікроконтролерів сімейства, з'явилися 3 регістри введення/виведення загального призначення – GPIOR0, GPIOR1 і GPIOR2. У цих регістрах можна зберігати будь-яку інформацію, проте основне їх призначення – збереження глобальних змінних. Регістри GPIOR0, GPIOR1 і GPIOR2 розташовані в молодшій половині основного простору введення/виведення і відповідно можуть використовуватися в командах побітового доступу SBI/CBI і SBIS/SBIC. Інші РВВ буде описано у відповідних розділах підручника.

Використання зовнішнього ОЗП

Частина AVR-мікроконтролерів, наприклад, ATmega8515x, ATmegal62x, ATmega64x/128x і ATmega640x/1280x/1281x/2560x/2561x мають можливість підключення зовнішнього статичного ОЗП об'ємом до 64 Кбайт.

Виводи деяких мікроконтролерів, які використовуються для підключення зовнішнього ОЗП, наведено у табл. 2.2.

У всіх моделях ці виводи є лініями портів введення/виведення загального призначення. При дозволі роботи із зовнішнім ОЗП режим роботи цих виводів визначається не вмістом регістра напряму передачі даних, а самим мікроконтролером.

Якщо роботу із зовнішнім ОЗП дозволено, то при зверненні за адресою, що перебуває поза границею внутрішнього ОЗП, автоматично відбувається звернення до зовнішнього ОЗП. Після формування на виводах порту А молодшого байта адреси, вивід ALE міняє свій стан з логічної одиниці на логічний нуль і залишається в цьому стані протягом усього циклу зчитування/запису. При дозволеній роботі із зовнішнім ОЗП звернення до внутрішнього ОЗП не впливає на роботу схеми, оскільки сигнали стробів зчитування – $\overline{RD}$ і запису – $\overline{WR}$ при цьому перебувають у неактивному одиничному стані.

Таблиця 2.1. **Регістр стану SREG**

Розряд	Назва	Опис
7	I	**Загальний дозвіл переривань.** Щоб дозволити переривання цей прапорець необхідно встановити в одиницю. Дозволити/заборонити окремі переривання можна встановленням або скиданням відповідних розрядів регістрів масок переривань. Якщо прапорець скинуто в нуль, то переривання заборонено незалежно від стану цих розрядів. Прапорець скидається апаратно після входу у підпрограму обробки переривання і відновлюється командою RETI, дозволяючи обробку наступних переривань
6	T	**Зберігання копійованого біта.** Цей розряд регістра використовується як джерело або приймач командами копіювання біта BLD (Bit Load) та BST (Bit STore). Вказаний розряд будь-якого РЗП можна скопіювати у цей розряд командою BST або змінити відповідно до вмісту даного розряду командою BLD
5	H	**Прапорець половинного перенесення**. Цей прапорець встановлюється в одиницю при виконанні деяких арифметичних операцій, якщо було або перенесення з молодшої тетради байта (з 3-го розряду в 4-й), або позика зі старшої тетради
4	S	**Прапорець знака.** Цей прапорець дорівнює результату операції «виключне АБО» – XOR між прапорцями N (від'ємний результат) і V (переповнення числа у додатковому коді). Відповідно цей прапорець встановлюється в одиницю, якщо результат виконання арифметичної операції менший від нуля. Прапорець дозволяє збільшити розрядність результату арифметичних операцій у додатковому коді з семи значущих біт до восьми із прапорцем S у якості дев'ятого знакового біта
3	V	**Прапорець переповнення додаткового коду.** Цей прапорець встановлюється в одиницю при переповненні розрядної сітки знакового результату. Використовується при роботі зі знаковими числами, які подано у додатковому коді
2	N	**Прапорець від'ємного значення**. Цей прапорець встановлюється в одиницю, якщо старший – 7-й розряд результату операції дорівнює одиниці. Інакше прапорець дорівнює нулю
1	Z	**Прапорець нуля.** Цей прапорець встановлюється в одиницю, якщо результат виконання операції дорівнює нулю
0	C	**Прапорець перенесення.** Цей прапорець встановлюється в одиницю, якщо в результаті виконання операції відбувся вихід за межі байта

При відсутності звернення до зовнішньої пам'яті виводи порту A переводяться мікроконтролером у третій стан. Цього можна уникнути, якщо підключити до виходів порту внутрішні резистори, що підтягують, або встановити в логічну одиницю розряд XMBK регістра SFIOR (моделі ATmega8515x та ATmega162x) або регістра XMCRB (інші моделі). При встановленому розряді на виводах порту A завжди зберігається останнє виведене значення.

Таблиця 2.2. **Виводи, що використовуються для підключення зовнішнього ОЗП**

Назва	**ATmega8515x**	**ATmega161x**	**ATmega162x**	**ATmega64x**	**ATmega128x**	**Опис**
AD0…AD7	PA0…PA7 (PORTA)					Мультиплексована ША/ШД
A8…A15	PC0…PC7 (PORTC)					Старший байт ША
ALE	PE1			PG2		Строб адреси
$\overline{RD}$	PD7			PG1		Строб зчитування
$\overline{WR}$	PD6			PG0		Строб запису

Підключення зовнішнього ОЗП до мікроконтролера показано на рис. 2.6.

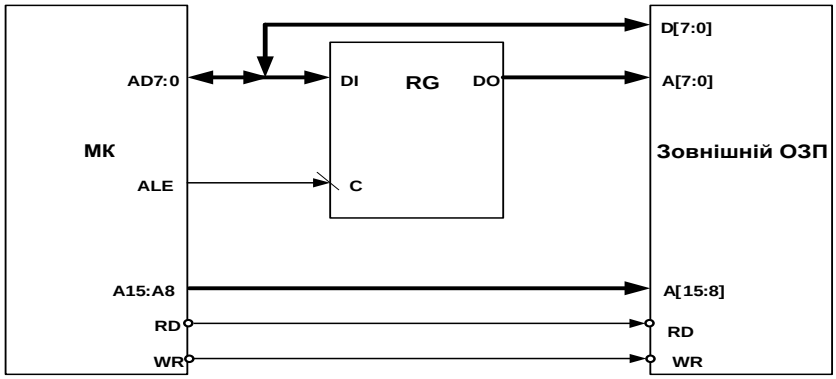


Рис. 2.6. Підключення зовнішнього ОЗП до мікроконтролера

Як видно з рисунка, для цього додатково буде потрібен регістр-защіпка – RG.

В якості такої защіпки, як правило, використовують мікросхему типу 74x573, наприклад, SN74AHC573 або аналогічну, запис даних у яку відбувається за низьким рівнем керуючого сигналу.

Мікроконтролери сімейства, які допускають підключення зовнішнього ОЗП, мають наступні можливості по роботі із нею:

– керування тривалістю циклу звернення до пам’яті;

– поділ пам’яті на два сектори з можливістю задання різної тривалості циклу звернення для кожного сектора;

– керування розрядністю шини адреси;

– утримання значень на шині даних для зменшення споживання

струму.

Для керування зовнішнім ОЗП, використовуються керуючі регістри, опис яких наведено у [1; 7].

Конфігурацію зовнішнього ОЗП, що підключений до мікроконтролера, показано на рис. 2.7.

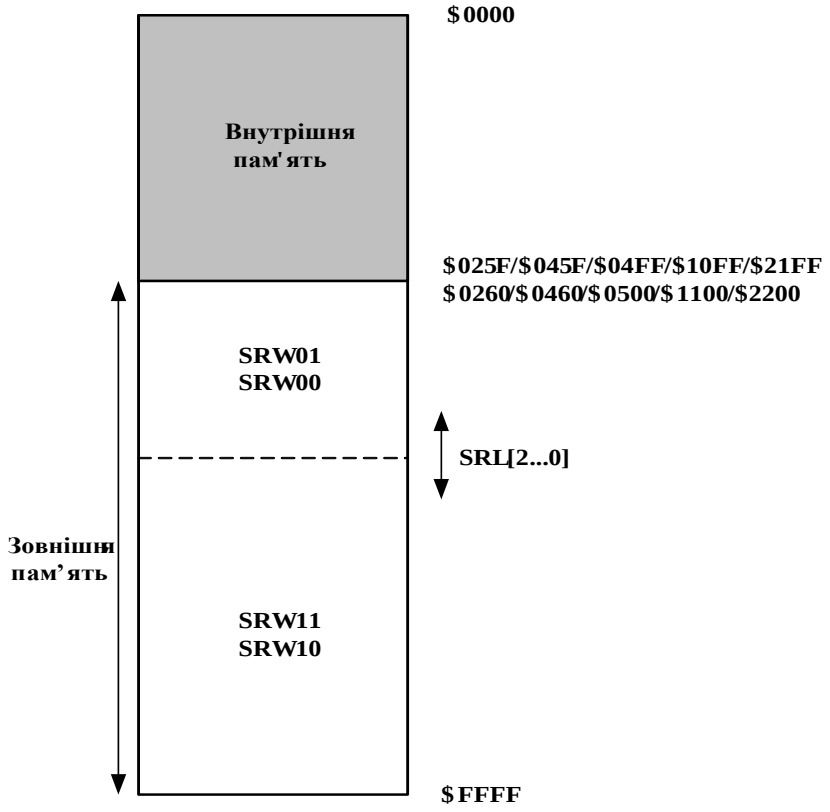


Рис. 2.7. Конфігурація зовнішнього ОЗП

Мікроконтролери сімейства дозволяють використовувати мікросхеми зовнішнього ОЗП з різним часом доступу. Підстроювання під різні мікросхеми здійснюється зміною тривалості циклу звернення до зовнішньої пам'яті. Крім того, є можливість розбити весь адресний простір зовнішнього ОЗП на два сектори, для кожного з яких може бути задана своя тривалість циклу звернення.

Положення границі між секторами визначається вмістом розрядів SRL2...SRL0 відповідно до таблиці, яку наведено у [1; 7]. У початковому стані ці розряди скинуто в нуль, тобто вся область пам'яті складається з одного сектора.

Зміна тривалості циклу звернення до зовнішньої пам'яті здійснюється заданням додаткових тактів очікування за допомогою розрядів SRW01:SRW00 для першого сектора та SRW11:SRW10 – для другого [1; 7].

Часові діаграми звернення до зовнішнього ОЗП при різних значеннях розрядів SRWn1: SRWn0, де n = 0 (нижній) або 1 (верхній) сектор, наведено у [1; 7].

При роботі із зовнішнім ОЗП використовується 16-розрядна шина адреси, що дозволяє адресувати 64К адрес. Для багатьох програм не потрібно такого великого об'єму зовнішньої пам'яті. Для таких випадків майже всі мікроконтролери дозволяють зменшити розрядність шини адреси.

Число виводів порту С, які задіяно під шину адреси, визначається вмістом розрядів XMM2...XMM0 відповідно до табл. 2.3. Незадіяні виводи можуть використовуватися як лінії введення/виведення загального призначення.

Таблиця 2.3. **Задання розрядності старшого байта шини адреси**

XMM2	XMM1	XMM0	Кількість розрядів, задіяних під шину адреси	Вивільнені контакти введення/виведення
0	0	0	8	Немає
0	0	1	7	PC7
0	1	0	6	PC7…PC6
0	1	1	5	PC7...PC5
1	0	0	4	PC7...PC4
1	0	1	3	PC7...PC3
1	1	0	2	PC7...PC2
1	1	1	Старший байт адреси не використовується	Порт С

Молодші адреси зовнішньої пам'яті (рис. 2.8), максимальний об'єм якої може бути 64Кбайт, відображуються на адресний простір внутрішньої пам'яті, внаслідок чого частина можливої ємності зовнішнього ОЗП залишається незадіяною.

Є можливість повного використання ємності мікросхем зовнішнього ОЗП. Якщо об'єм зовнішнього ОЗП менше 64 Кбайт, то для доступу до молодших адрес мікросхем зовнішнього ОЗП досить просто використовувати адреси, які лежать за межами області фізичних адрес зовнішнього ОЗП. Оскільки старші біти шини адреси не використовуються, то при зверненні за цими адресами адресуватимуться комірки зовнішнього ОЗП з молодшими адресами, починаючи з 0x0000. Візьмемо, наприклад, мікроконтролер ATmega128x із зовнішнім ОЗП об'ємом 32 Кбайт (рис. 2.8).

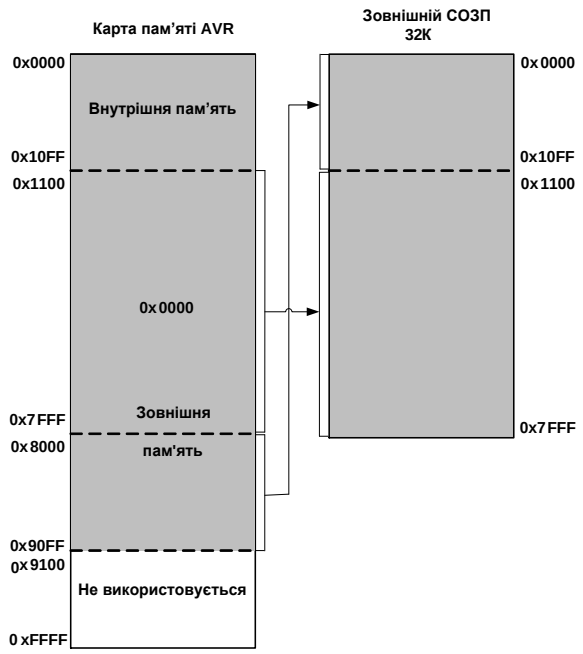


Рис. 2.8. Повне використання ємності мікросхеми ОЗП (32 Кбайт) на прикладі мікроконтролера ATmega128x

В цьому випадку при зверненні за адресами 0x8000...0x90FF ми насправді будемо звертатися до комірок зовнішнього ОЗП з адресами 0x0000...0x10FF. Таким чином, у розпорядженні прикладної програми виявиться лінійна область пам'яті розміром 32 Кбайт з адресами 0x1100...0x90FF.

Повністю використовувати місткість зовнішнього ОЗП можна також керуючи програмно старшими розрядами адреси. Для цього, порт С повинен бути налаштований на виведення, а в защіпці порту повинне бути записане «$00». При маскуванні старших розрядів адреси на відповідні виводи порту С будуть виставлені нулі і відбудеться звернення до молодших адрес зовнішнього ОЗП.

Нижче наведено два приклади програмної реалізації описаних дій для мікроконтролера ATmega128.

Приклад на асемблері

```
; Константу OFFSET визначено як $2000 для гарантованого звернення
; до зовнішньої пам'яті
ldi     rl6, $FF
out     DDRC, rl6                ;Налаштування лінії порту С на виведення
ldi     r16, $00
out     PORTC, rl6               ;Запис в защіпку порту С $00
ldi     rl6, (1<<XMM1)|(1<<XMM0)
out     XMCRB, rl6               ;Звільнення розрядів PC7...PC5
ldi     rl6, $AA
sts     $0001+OFFSET, r16        ;Запис $AA за адресою (OFFSET+1) зовн. пам'яті
```

```
ldi     rl6, (0<<XMM1)|(0<<XMM0)
out     XMCRB, r16              ;Перепризначення розрядів PC7...PC5
ldi     rl6, $55
sts     $0001+OFFSET, rl6       ;Запис $55 за адресою (OFFSET+1)
                                ;зовнішньої пам’яті
```

Приклад на С

```
#define OFFSET 0x2000
void XRAM_example(void)
{       unsigned char *p=(unsigned char *)(OFFSET+1);
DDRC = 0xFF; PORTC = 0x00;
XMCRB = (1<<XMM1)|(1<<XMM0);
*p = 0xAA;
XMCRB = 0x00;
*p = 0x55;
}
```

У порівнянні зі зверненням до внутрішнього ОЗП звернення до зовнішнього ОЗП вимагає на 1, 2, 3 або 4 машинних циклів (залежить від режиму звернення) більше для кожного байта, який обробляється командою. Таким чином, час виконання команд – LD, ST, LDS, STS, PUSH та POP збільшується на 1/2/3/4 цикли.

Якщо у зовнішньому ОЗП розташовано стек, то час переходу до обробки переривань, виклику та повернення з підпрограм збільшується на 2/4/6/8, або – на 3/5/7/9 машинних циклів (залежить від моделі МК). Це пов'язано з тим, що під час виконання зазначених операцій відбувається збереження або відновлення лічильника команд різної розрядності і, крім того, при зверненні до зовнішньої пам’яті не використовується конвеєризація.

Часові діаграми звернення до зовнішнього ОЗП з зазначенням усіх параметрів сигналів наведено у [1; 7].

2.3.3. Енергонезалежна пам’ять даних EEPROM

Частина мікроконтролерів сімейства Tiny та всі мікроконтролери сімейств Mega та XMega мають у своєму складі енергонезалежну пам’ять даних (EEPROM-пам’ять). Цю пам’ять розташовано у власному адресному просторі, а її об'єм становить від шестидесяти байт (для сімейства Tiny) та від 256 байт до більш ніж 4 Кбайт для сімейств Mega та XMega.

Для запису та читання EEPROM-пам’яті в готовому пристрої використовуються три регістри введення/виведення: регістр адреси – EEAR, регістр даних – EEDR та регістр керування – EECR. Регістр адреси EEAR фізично

розміщується у двох РВВ: EEARH:EEARL. Адреси, формати та опис окремих розрядів регістрів керування EEPROM-пам'яті наведено у [1; 7].

Процедура запису одного байта в EEPROM-пам'ять складається з наступних етапів:

1. Для визначення готовності пам'яті до запису даних дочекатися коли скинеться прапорець EEPE (EEWE) регістра EECR.
2. Для визначення завершення запису у FLASH-пам'ять програм дочекатися коли скинеться прапорець SPMEN регістра SPMCR.
3. Завантажити байт даних у регістр EEDR, а необхідну адресу – у регістр EEAR.
4. Встановити в одиницю прапорець EEMPE (EEMWE) регістра EECR.
5. Протягом 4-х машинних циклів записати в розряд EEPE (EEWE) регістра EECR одиницю. Після встановлення цього розряду процесор пропускає 2 такти перед виконанням наступної інструкції.

Другий пункт введено через те, що запис в EEPROM-пам'ять не може виконуватися одночасно із записом у FLASH-пам'ять. Тому перед виконанням запису в EEPROM-пам'ять варто переконатися, що програмування FLASH-пам'яті завершено. Якщо в програмі відсутній завантажувач, тобто мікроконтролер ніколи не змінює вміст пам'яті програм, другий крок може бути пропущений.

На процес звернення до EEPROM-пам'яті впливає внутрішній RC-генератор, що калібрується. Відповідно, тривалість циклу запису залежить від частоти цього генератора, напруги живлення та температури [1; 7]. За закінченням циклу запису розряд EEPE (EEWE) апаратно скидається, після чого програма може почати запис наступного байта.

При запису в EEPROM можуть виникнути деякі проблеми, викликані перериваннями:

1. При виникненні переривання між 4-м та 5-м етапами описаної вище послідовності, запис в EEPROM буде перервано, тому що за час обробки переривання прапорець EEMPE (EEMWE) скинеться в нуль.
2. Якщо в підпрограмі обробки переривання, що виникло під час запису в EEPROM-пам'ять, також відбувається звернення до неї, то буде змінено вміст регістрів адреси та даних EEPROM. В результаті перший перерваний запис буде зірвано.

Для запобігання описаних проблем рекомендується забороняти всі переривання (скидати біт I регістра SREG) на час виконання пунктів 2...5 описаної вище послідовності.

Нижче наведено два приклади реалізації функції запису в EEPROM-пам'ять (керування перериваннями не виконується).

Приклад на асемблері

```
EEPROM_write:
sbic EECR, EEWE; очікування, коли скинеться прапорець EEWE
rjmp  EEPROM_write; відносний безумовний перехід на мітку
; EEPROM_write
out    EEARH, r18; запис старшого байта адреси з регістра r18 в EEARH
out    EEARL, r17; запис молодшого байта адреси з регістра r17 в EEARL
out    EEDR, r16; запис даних з регістра r16 у регістр даних EEDR
sbi    EECR, EEMWE; встановлення прапорця EEMWE регістра EECR
sbi    EECR, EEWE; встановлення прапорця EEWE регістра EECR
ret; вихід з підпрограми
```

Приклад на С

```
void EEPROM_write (unsigned int uiAddress, unsigned char ucData)
{
While (EECR & (1<<EEWE)); /* Очікування завершення попереднього
                             запису*/
        EEAR = uiAddress;            /*Ініціалізація регістрів*/
        EEDR = ucData;
        EECR | = (1<<EEMWE);         /*Встановлення прапорця EEMWE*/
        EECR | = (1<<EEWE);   /*Початок запису в EEPROM*/
}
```

Перед виконанням операції зчитування також необхідно проконтролювати стан прапорця EEPE (EEWE), тому що поки виконується операція запису в EEPROM-пам’ять (прапорець EEWE встановлено), не можна виконувати ні зчитування EEPROM-пам’яті, ні зміни регістра адреси.

Після завантаження необхідної адреси в регістр EEAR необхідно встановити в одиницю розряд EERE регістра EECR. Коли зчитані дані буде поміщено в регістр даних EEDR, відбудеться апаратне скидання цього розряду. Однак стежити за станом розряду EERE для визначення моменту завершення операції зчитування не потрібно, тому що операція зчитування з EEPROM завжди виконується за один такт. Крім того, після встановлення розряду EERE в одиницю процесор пропускає 4 такти перед початком виконання наступної інструкції.

Нижче наведено два приклади реалізації функції зчитування з EEPROM-пам’яті (керування перериваннями не виконується).

Приклад на асемблері

```
EEPROM_read:
sbic   EECR, EEWE; очікування, коли скинеться прапорець EEWE
rjmp   EEPROM_read; відносний безумовний перехід на мітку
       ;EEPROM_read
out    EEARH, r18; запис старшого байта адреси з регістра r18 в EEARH
out    EEARL, r17; запис молодшого байта адреси з регістра r17 в EEARL
sbi    EECR, EERE; встановлення прапорця EERE регістра EECR
in     r16, EEDR; пересилання даних з регістра EEDR в регістр r16
ret; вихід з підпрограми
```

Приклад на С

```
void EEPROM_write (unsigned int uiAddress)
{
while(EECR & (1<<EEWE));  /* Очікування завершення попереднього*/
                          /*запису */
      EEAR = uiAddress;          /* Ініціалізація регістрів адреси */
      EECR | = (1<<EERE);   /*Виконання зчитування */
Return EEDR;
}
```

При використанні EEPROM-пам’яті необхідно дотримуватися деяких запобіжних заходів, щоб уникнути ушкодження даних, що в ній зберігаються. При зниженні напруги живлення нижче деякої величини дані, що зберігаються в пам’яті, можуть бути ушкоджені. Якщо напруга живлення буде нижче норми, то сам мікроконтролер може виконувати команди некоректно,

Щоб уникнути ушкодження даних, що зберігаються в EEPROM, треба скористатися однієї із трьох наступних рекомендацій:

1. Утримувати мікроконтролер у стані скидання увесь час, поки напруга живлення перебуває нижче норми. Для цього варто використовувати вбудований детектор зниженої напруги живлення – Brown-out Detector (BOD).

2. Утримувати мікроконтролер у «сплячому» режимі – Power Down поки напруга живлення перебуває нижче норми. Оскільки в цьому режимі мікроконтролер не може виконувати ніяких команд, таке рішення ефективно захищає службові регістри EEPROM від ненавмисного запису.

3. Зберігати константи у FLASH-пам’яті програм, якщо вони не повинні мінятися під час роботи програми. Мікроконтролер не може

самостійно робити запис у FLASH-пам’ять, тому при зниженні напруги живлення її вміст не буде ушкоджено.

2.4. Програмування FLASH- та EEPROM-пам’яті

Загальні відомості

Одним з питань, яке треба вирішувати при розробці мікроконтролерної системи, є програмування FLASH- та EEPROM-пам’яті. AVR-мікроконтролери підтримують наступні режими програмування (табл. 2.4) [1; 7]:

– режим послідовного програмування (за SPI-інтерфейсом);
– режим паралельного програмування при високій напрузі;
– режим програмування за JTAG-інтерфейсом.

Таблиця 2.4. **Режими програмування AVR-мікроконтролерів**

Режим програмування	**ATmega8515x/8535x**	**ATmega8x**	**ATmega64x/32x**	**ATmega64x/128x**	**ATmega48x/88x/168x**	**ATmega162x**	**ATmega164x/324x/644x**	**ATmega165x**	**ATmega325x/3250x, ATmega645x/6450x**	**ATmega640x, ATmega1280x/1281x, ATmega2560x/2561x**
1. Послідовне, за SPI-інтерфейсом	•	•	•	•	•	•	•	•	•	•
2. Паралельне при високій напрузі	•	•	•	•	•	•	•	•	•	•
3. За JTAG-інтерфейсом			•	•		•	•	•	•	•

Під «високою» напругою розуміється керуюча напруга – 12 В, яка подається на вивід RESET мікроконтролера для переводу останнього в режим програмування.

Крім того, у мікроконтролерах сімейств Mega та XMega є можливість самопрограмування – зміна під керуванням мікроконтролера вмісту пам’яті програм в готовому пристрої. Цей режим буде розглянуто у розділі 4.

В процесі програмування можуть виконуватися наступні операції:

– очищення кристалу (chip erase);
– читання/запис FLASH-пам’яті програм;
– читання/запис EEPROM-пам’яті даних;
– читання/запис конфігураційних комірок;
– читання/запис комірок захисту;

– читання комірок ідентифікатора;
– читання калібрувального байта.

Всі моделі мікроконтролера постачаються зі стертою пам’яттю програм та пам’яттю даних (у всіх комірках знаходиться числа $FF) і придатні до відповідного програмування.

Захист коду і даних

Вміст FLASH-пам’яті програм, та EEPROM-пам’яті даних може бути захищено від запису і/або читання за допомогою програмування комірок захисту (Lock Bits) LB1 і LB2. Можливі режими захисту, які відповідають різним станам цих комірок наведено у [1; 7].

У вихідному (незапрограмованому) стані у всіх комірках захисту міститься одиниця, після програмування – нуль. Очищення комірок (запис в них логічної одиниці) може бути тільки при виконанні команди «Очищення кристалу», яка також знищує вміст FLASH- і EEPROM-пам’яті.

Конфігураційні комірки

При програмуванні мікроконтролера можна виконувати налаштування його конфігурації. Для цього призначено конфігураційні комірки (Fuse Bits), які розташовано в окремому адресному просторі, який доступний тільки при програмуванні. Всі конфігураційні комірки згруповано в кілька байтів, а їх склад залежить від конкретного мікроконтролера.

Для зміни вмісту конфігураційних комірок використовуються спеціальні команді програмування [1; 7]. Конфігурацію мікроконтролера необхідно задавати до програмування комірок захисту.

Ідентифікатор

Всі AVR-мікроконтролери мають три 8-бітні комірки, вміст яких дозволяє ідентифікувати пристрій. В першій комірці міститься код виробника, у другій – код об’єму FLASH-пам’яті, а в третій – код пристрою. Комірки ідентифікатора також розташовано в окремому адресному просторі, доступ до якого можливий тільки в режимі програмування. На відміну від конфігураційних комірок, комірки ідентифікатора доступні тільки для читання. Вміст комірок ідентифікатора для окремих мікроконтролерів сімейства наведено в [1; 7].

Калібрувальні комірки

При виготовлені мікроконтролера в спеціальні калібрувальні комірки заносяться константи, які призначено для налаштування на номінальну частоту внутрішнього RC-генератора [1; 7].

Сторінкова організація пам’яті програм і даних

В AVR-мікроконтролерах використовується сторінкова організація FLASH-пам’яті пам’яті програм та EEPROM-пам’яті даних [1; 7].

При програмуванні FLASH-пам’яті весь її об’єм розбивається на сторінки, розмір і кількість яких залежить від об’єму пам’яті програм мікроконтролера (табл. 2.5). Дані спочатку завантажуються в буфер сторінки і тільки потім заносяться безпосередньо в пам’ять програм. Прошивка всіх комірок сторінки при цьому відбувається одночасно.

Таблиця 2.5. **Параметри сторінкової організації пам’яті програм**

Параметр	**Об’єм пам’яті програм (байт)**						
	4К	8К	16К	32К	64К	128К	256К
Розмір сторінки, слів	32	64	64	128	128	128	256
Кількість сторінок	64	128	256	256	512	1024	1024

Аналогічно організовано EEPROM-пам’ять. Розмір сторінок, а також їх кількість для наведено в табл. 2.6.

Таблиця 2.6. **Параметри сторінкової організації EEPROM-пам’яті**

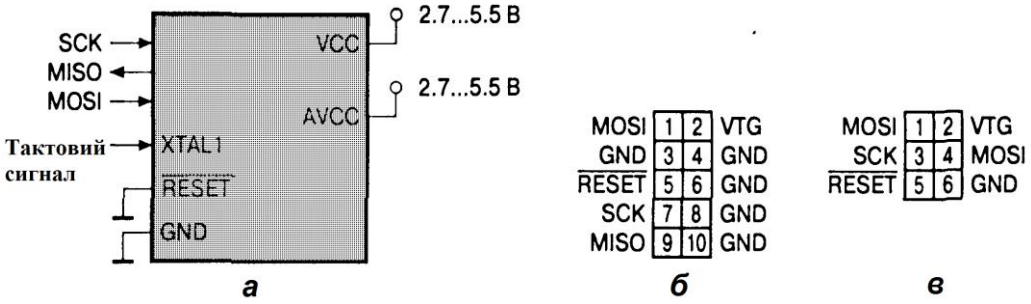
Параметр	**Об’єм EEPROM-пам’яті (байт)**				
	256	512	1К	2К	4К
Розмір сторінки, байт	4	4	4	8	8
Кількість сторінок	64	128	256	256	512

В багатьох моделях сторінкова організація EEPROM-пам’яті використовується тільки при програмуванні в паралельному режимі, а програмування за послідовним каналом відбувається побайтно.

Програмування пам’яті за послідовним каналом

Для програмування або перепрограмування пам’яті програм і даних безпосередньо у пристрої використовують послідовний SPI-інтерфейс (рис. 2.9).

На рис. 2.9*а* наведено схему включення мікроконторолера в цьому режимі. На рис. 2.9*б, в* показано два варіанти розводки колодки для підключення програматора, які рекомендовано компанією Atmel. Зокрема, перший варіант розводки (рис. 2.9*б*) використовується в програматорі AVRISP і налагоджувальних платах STK200/300 компанії Atmel.



Примітка. Якщо в якості тактового використовується внутрішній RC-генератор, вивід XTAL1 залишають непідключеним

Рис. 2.9. Включення мікроконтролера в режимі програмування за послідовним каналом

Другий варіант розводки (рис. 2.9*в*) використовується в більш нових пристроях компанії – програматорі AVRISP mkll і платі STK500.

Як видно з рис. 2.9*а*, для обміну даними між програматором і пристроєм використовується три лінії SPI-інтерфейсу: SCK – тактовий сигнал, MOSI – вхід даних і MISO – вихід даних. Відповідність між ціма лініями інтерфейсу і контактами портів введення/виведення деяких мікроконтролерів наведено в табл. 2.7.

Таблиця 2.7. **Виводи, які використовуються при програмуванні за послідовним каналом**

Назва лінії інтерфейсу	**Atmega8515x/8535x**	**Atmega8x**	**Atmega16x/32x**	**Atmega64x/128x**	**Atmega48x/88x/168x**	**Atmega162x**	**Atmega164x/324x/644x**	**Atmega165x, Atmega325x/3250x, Atmega645x/6450x**	**Atmega640x/1280x/2560x**	**Atmega1281x/2561x**	**Призначення виводів**
SCK	PB7	PB5	PB7	PB1	PB5	PB7	PB7	PB1	PB1	PB1	Вхід тактового сигналу
MISO	PB6	PB4	PB6	PE1	PB4	PB6	PB6	PB3	PE1	PB3	Вихід даних
MOSI	PB5	PB3	PB5	PE0	PB3	PB5	PB5	PB2	PE0	PB2	Вхід даних

В деяких моделях мікроконтролерів виводи, що використовуються для програмування, не співпадають з виводами модуля SPI.

Часові діаграми, параметри сигналів, команди та послідовність програмування наведено на в [1; 7].

Паралельне програмування

В режимі паралельного програмування використовується окремий пристрій – спеціалізований програматор, від якого до мікроконтролера передаються одночасно всі біти коду команди або байта даних. Цей режим залучає значну кількість виводів мікроконтролера та потребує використання додаткового джерела підвищеної напруги – 12В. Основне застосування цього режиму – «прошивка» мікроконтролерів в умовах масового виробництва перед їх встановленням на плату [1; 7].

Програмування за JTAG-інтерфейсом

JTAG-інтерфейс було розроблено групою провідних спеціалістів з проблем тестування електронних компонентів (Join Test Action Group) [1; 7]. В подальшому він був зареєстрований Інститутом інженерів з електротехніки та електроніки (IEEE) в якості промислового стандарту IEEE Std 1149.1-1990 (IEEE Standard Test Access Port and Boundary-Scan Architecture).

JTAG-інтерфейс, якій вбудовано у більшість AVR-мікроконтролерів, може бути використано для наступних цілей:

- тестування друкованих плат;
- конфігурування (програмування) кристала;
- внутрішньосхемного налагоджування.

Доступ до модуля JTAG відбувається через чотири виводи мікроконтролера, які складають так званий «порт тестового доступу» – Test Access Port (TAP): TMS, TCK, TDI і TDO. Відповідність цих виводів контактам портів введення/виведення мікроконтролера, опис їх функцій, стандартну розводку 10-контактного роз’єму для підключення JTAG-пристрою та рекомендації по використанню JTAG-інтерфейсу при програмуванні наведено у [1; 7].

Контрольні запитання та завдання

1. Назвіть розрядність однієї комірки пам'яті програм мікроконтролера AVR.
2. Опишіть організацію та призначення пам'яті програм.
3. На які три частини розділено пам'ять даних мікроконтролерів сімейства? Опишіть призначення та структуру кожного з них.
4. Які види стека ви знаєте? Який вид стека використовується у мікроконтролерах і де він розміщується?
5. За якою архітектурою виконано організацію пам'яті AVR-мікроконтролерів сімейства Mega?
6. Наведіть структурну схему карти пам'яті AVR-мікроконтролерів сімейства Mega. Опишіть структуру і призначення кожного виду пам'яті.
7. Опишіть структуру регістрового файлу.
8. Де в AVR-мікроконтролерах розташовуються регістри введення/виведення? Який з них використовується найчастіше?
9. Наведіть схему підключення зовнішнього ОЗП до мікроконтролера. Який можливий максимальний розмір ОЗП?
10. Для чого в схемі підключення зовнішнього ОЗП до мікроконтролера використовується регістр RG?
11. Які виводи мікроконтролера використовуються для підключення зовнішнього ОЗП?
12. Які можливості для роботи з зовнішньою пам'яттю даних є у мікроконтролерів сімейства? Перерахуйте основні регістри, що використовуються для реалізації цих можливостей.
13. Яким чином здійснюється зміна тривалості циклу звернення до зовнішньої пам'яті даних?
14. Що таке енергонезалежна пам'ять даних (EEPROM)? Які регістри використовуються для керування EEPROM?
15. Опишіть процедуру запису одного байта в EEPROM-пам'ять даних.
16. Опишіть процедуру зчитування інформації з EEPROM-пам'яті даних.
17. Які режими програмування підтримують мікроконтролери сімейства Mega?
18. Які операції можуть виконуватися в процесі програмування?
19. Для чого існують комірки захисту?
20. Опишіть призначення комірок захисту та комірок ідентифікатора.
21. Що міститься у калібрувальних комірках?
22. Які лінії використовуються для обміну даними між програматором і пристроєм при програмуванні за послідовним каналом?

23. Які дії необхідно виконати для переведення мікроконтролера в режим програмування за послідовним каналом?
24. Якими способами відбувається програмування Flash-пам’яті та EEPROM-пам’яті мікроконтролерів сімейства Mega?
25. Які дії необхідно виконати для переведення мікроконтролера в режим програмування?
26. Коли повинна виконуватися команда «Очищення кристала»?
27. Опишіть послідовність дій при записі/читанні FLASH/EEPROM-пам’яті.
28. Яким чином відбувається запис/читання конфігураційних комірок та комірок захисту?
29. В якій послідовності відбувається читання комірок ідентифікатора та калібрувальних комірок?
30. В яких цілях може бути використано інтерфейс JTAG?
31. Який цифровий електронний елемент може використовуватись для зберігання одного біта інформації?
32. Яку функцію виконують виводи AD0…AD7 мікроконтролера?
33. Поясніть особливості сторінкової організації пам’яті програм.
34. Від чого залежить розмір і кількість сторінок при сторінкової організації пам’яті?
35. Назвіть команди, які можна використовувати для звернення до регістрів введення/виведення СПД.
36. Назвіть команди, які можна використовувати для звернення до тридцятидвох регістрів загального призначення СПД.
37. Опишіть формат та призначення окремих розрядів рогістра SREG.
38. Що означає символ «$» при наведенні адрес пам’яті?
39. Для чого використовується лічильник команд (Program Counter – PC) та від чого залежить його розмір?
40. Що знаходиться за адресою $0000 в пам’яті програм?
41. Які регістри розташовуюься в областях регістрів введення/виведення?
42. Скільки байт виділяється для додаткових регістрів введення-виведення в просторі статичної пам’яті даних?
43. Поясніть призначення 16-бітних регістрів: X, Y та Z.
44. Від чого залежить розмір регістра-покажчик стека?
45. Чому при вказівці адрес PBB деякі з них вказуються в дужках?
46. Назвіть приблизну кількість циклів перепрограмування FLASH/EEPROM- пам’яті.

3. ПРОГРАМУВАННЯ AVR-МІКРОКОНТРОЛЕРІВ

3.1. Послідовність розробки керуючої програми

Якщо мікропроцесорну систему (МПС) виконано на основі мікроконтролера (МК), то керуюча програма знаходиться у пам’яті програм (ПП). Без керуючої програми жодна МПС працювати не буде.

Нижче наведено основні етапи створення керуючої програми:

– розробка схеми алгоритму, яку повинна реалізувати конкретна керуюча програма;
– якщо задача складна, то програма може бути реалізована і відлагоджена або мовою високого рівня, або мовою асемблера;
– компіляція – переведення програми з мови високого рівня або асемблера у машинні коди конкретного МК (створення об'єктного модуля);
– введення керуючої програми в ПП за допомогою програматора.

В процесі початкової ініціалізації системи у програмний лічильник завантажується початкова адреса програми.

Керуюча програма може включати підпрограми – частину основної програми, яку можна викликати з основної програми або за перериванням.

У МПС при створенні робочих програм використовують такі системні керуючі програми:

– транслятори – перетворюють програму, яку написано однією мовою програмування, в програму, яку написано іншою мовою;
– компілятори – є різновидом трансляторів та переводять програму з мови високого рівня в машинні коди;
– асемблери – це окремий випадок компіляторів, які перетворюють програму з мови асемблера у двійкові (машинні) коди;
– відлагоджувачі – керуючі програми для відлагодження програми у відповідній системі.

3.2. Мови програмування

У МПС використовують мови програмування високого рівня, наприклад Сі, і низького рівня – асемблер. Останній називають машинно-орієнтованим, тому що при написанні програми використовують команди конкретного МК, які не можна змінити, а можна вивчити і використовувати під час програмування. Інші МК не зможуть виконувати таку програму, тому що, мають свою систему команд.

Окрім команд, які при компіляції перетворюються у машинний код, програма на мові асемблера (МА) включає псевдокоманди та директиви, які вказують програмі компілятора, яка також зветься асемблером, як виконати відповідні дії по створенню виконуваного файла для МК.

Деякі особливості мови асемблера AVR-мікроконтролерів наведено у [1].

3.3. Програмна модель AVR-мікроконтролера

Загальні відомості

Однією з основних характеристик архітектури будь-якого мікроконтролера є програмна модель (модель програміста), яка включає частину структури мікроконтролера, що доступна програмісту за допомогою системи команд. На рис. 3.1 в якості приклада наведено програмну модель одного з AVR-мікроконтролерів. Ця модель має регістри загального призначення (РЗП), регістри введення/виведення (РВВ), статичний оперативний запам'ятовуючий пристрій (СОЗП), який в технічній літературі англійською мовою називають SRAM, ємністю 128 байт, EEPROM-пам'ять даних, ємністю 512 Кбайт та постійний запам'ятовуючий пристрій FLASH-типу, ємністю 2К слів (4 Кбайт). РЗП, РВВ та СОЗП називають статичною пам'яттю даних, тому що вони є енергозалежними. Відповідно EEPROM-пам'ять даних є енергозалежною.

Регістри загального призначення

32 регістри загального призначення об'єднано у файл, структуру якого показано на рис. 3.2. В AVR-мікроконтролерах усі РЗП безпосередньо доступні АЛП на відміну від мікроконтролерів фірм, у яких є тільки один такий регістр, наприклад, робочий регістр A/W – акумулятор. Завдяки цьому будь-який РЗП може використовуватись командами як операнд-джерело, або як операнд-приймач. Винятком є лише п'ять арифметичних і логічних команд, що виконують дії між константою і регістром – SBCI, SUBI, CPI, ANDI, ORI, а також команда завантаження константи в регістр – LDI. Ці команди можуть звертатися тільки до другої половини РЗП – R16...R31. Декілька регістрів загального призначення використовуються як покажчики при непрямій адресації статичної пам'яті даних.

Як показано на рис. 3.2, кожен регістр файлу, має свою власну адресу у просторі статичної пам'яті даних (СПД). Тому до них можна звертатися також як до комірок СПД.

На рис. 3.3 наведено регістри-покажчики X,Y і Z, які використовуються при непрямій адресації операндів.

Регістри введення/виведення

Регістри введення/виведення (РВВ) розташовуються в так званому основному просторі введення/виведення розміром 64 байти. Усі РВВ можна розділити на дві групи: службові регістри мікроконтролера і регістри, що відносяться до периферійних пристроїв, у тому числі порти введення/виведення. Розмір кожного регістра – 8 розрядів.

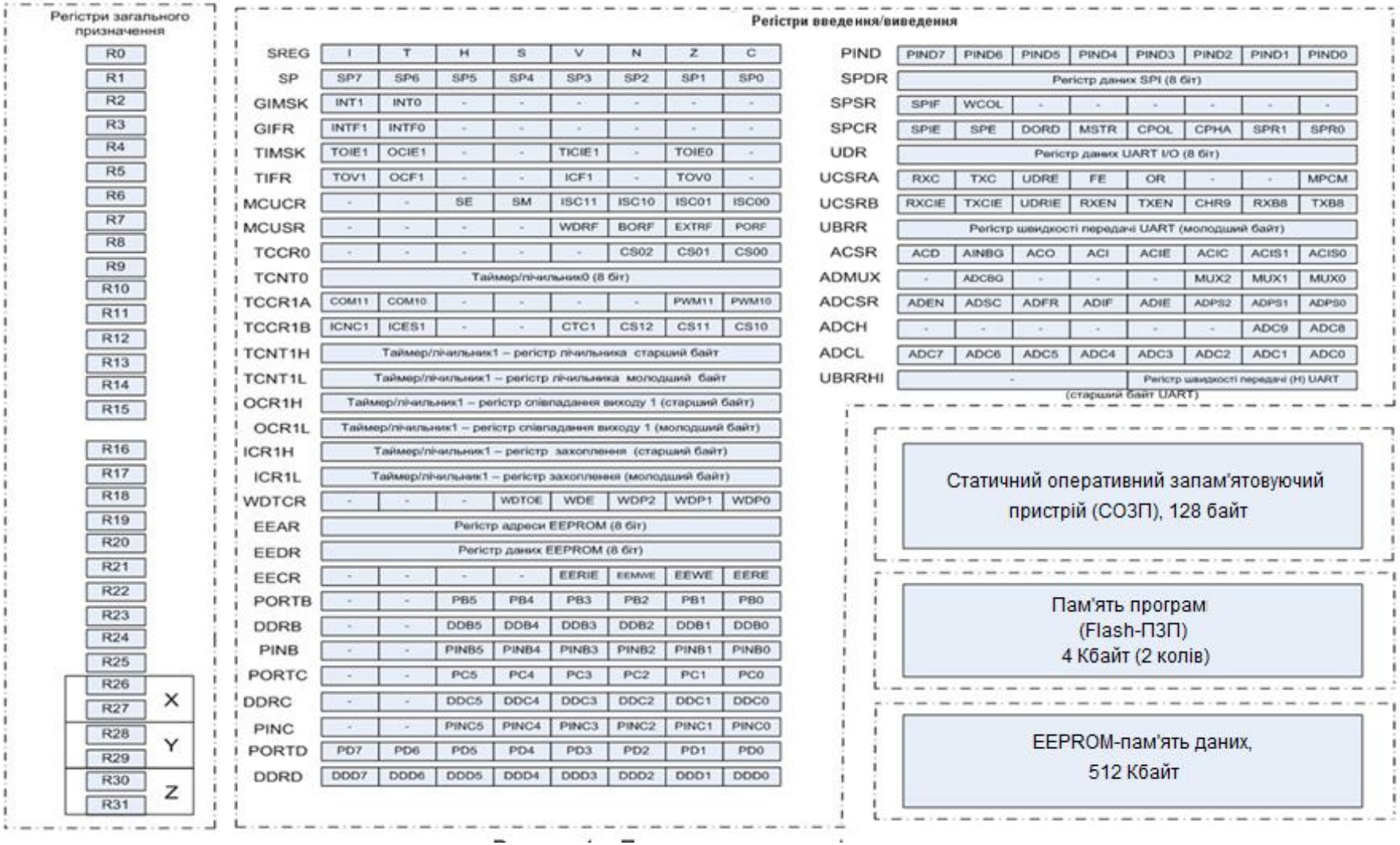


Рис. 3.1. Програмна модель мікроконтролера

7р 0р	7р 0р	Адреса
R0	R0	$00
R1	R1	$01
R2	R2	$02
...	...	
R13	R13	$0D
R14	R14	$0E
R15	R15	$0F
R16	R16	$10
R17	R17	$11
...	...	
R26	R26	$1A Регістр X, мол. байт
R27	R27	$1B Регістр X, ст. байт
R28	R28	$1C Регістр Y, мол. байт
R29	R29	$1D Регістр Y, ст. байт
R30 (Регістр Z)	R30	$1E Регістр Z, мол. байт
R31	R31	$1F Регістр Z, ст. байт
AT90S1200	Інші моделі	

Рис. 3.2. Структура файлу регістрів загального призначення

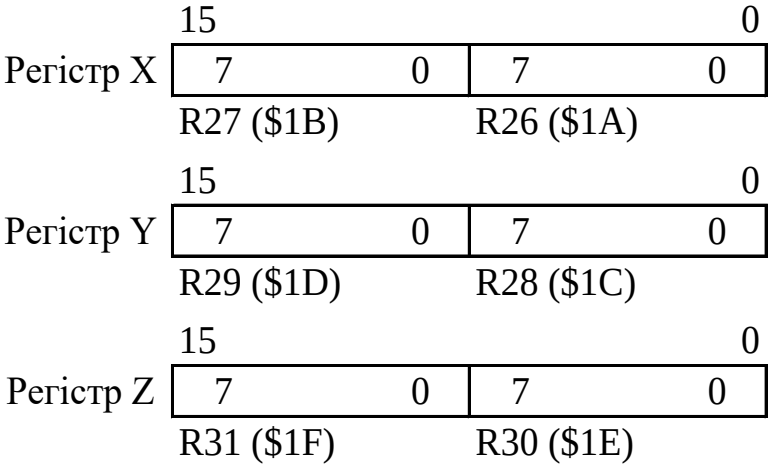


Рис. 3.3. Регістри-покажчики X,Y і Z

Розподіл адрес основного простору введення/виведення залежить від конкретної моделі мікроконтролера, тому що різні моделі мають різний склад периферійних пристроїв і, відповідно, різну кількість регістрів [1; 7]. У табл. 3.1, як приклад, наведено розміщення в адресному просторі введення/виведення деяких РВВ, спільних для більшості мікроконтролерів сімейства. Інші регістри введення/виведення будуть розглянуто в наступних розділах.

Якщо адресу в таблиці не зазначено, це означає, що її зарезервовано, і запис за цією адресою не рекомендується.

Таблиця 3.1. **Приклад регістрів введення/виведення**

Назва	**Функція**	**Адреса**
ACSR	Регістр керування і стану аналогового компаратора	$08 ($28)*
ADCH	Регістр даних АЦП (старший байт)	$05 ($25)
ADCL	Регістр даних АЦП (молодший байт)	$04 ($24)
ADCSR	Регістр керування і стану АЦП	$06 ($26)
ADMUX	Регістр керування мультиплексором АЦП	$07 ($27)
DDRB	Регістр напряму даних порту B	$17 ($37)
DDRC	Регістр напряму даних порту C	$14 ($34)
DDRD	Регістр напряму даних порту D	$11 ($31)
EEAR	Регістр адреси EEPROM	$1E($3E)
EECR	Регістр керування EEPROM	$1C($3C)
EEDR	Регістр даних EEPROM	$10 ($30)
GIFR	Загальний регістр прапорців переривань	$3A ($5A)
GIMSK	Загальний регістр маски переривань	$3B ($5B)
ICR1H	Регістр захоплення таймера/лічильника 1 (старший байт)	$27 ($47)
ICR1L	Регістр захоплення таймера/лічильника 1 (молодший байт)	$26 ($46)
MCUCR	Загальний регістр керування мікроконтролером	$35 ($55)
OCR1H	Регістр збігу таймера/лічильника 1 (старший байт)	$2B ($4B)
OCR1L	Регістр збігу таймера/лічильника 1 (молодший байт)	$2A ($4A)

Закінчення табл. 3.1

Назва	**Функція**	**Адреса**
PINB	Виводи порту B	$16 ($36)
PINC	Виводи порту C	$13 ($33)
FIND	Виводи порту D	$10 ($30)
PORTC	Регістр даних порту C	$15 ($35)
PORTD	Регістр даних порту D	$12 ($32)
SP	Покажчик стека	$3D ($5D)
SPCR	Регістр керування SPI	$00 ($20)
SPDR	Регістр даних SPI	$0F ($2F)
SPSR	Регістр стану SPI	$0E ($2E)
SREG	Регістр стану	$3F ($5F)
TCCR0	Регістр керування таймером/лічильником 0	$33 ($53)
TCCR1A	Регістр керування A таймером/лічильником 1	$2F ($4F)
TCCR1B	Регістр керування B таймером/лічильником 1	$2E($4E)
TCNT0	Лічильний регістр таймера/лічильника 0 (8-розрядний)	$32($52)
TCNT1H	Лічильний регістр таймера/лічильника 1 (старший байт)	$2D ($4D)
TCNT1L	Лічильний регістр таймера/лічильника 1 (молодший байт)	$2C ($4C)
TIFR	Регістр прапорців переривань від таймерів/лічильників	$38 ($58)
TIMSK	Регістр маски переривань від таймерів/лічильників	$39($59)
UBRR	Регістр швидкості передачі UART (молодший байт)	$09 ($29)
UBRRHI	Регістр швидкості передачі UART (старший байт)	$03 ($23)
UCR	Регістр керування UART	$0A ($2A)
UDR	Регістр даних UART	$0C ($2C)
USR	Регістр стану UART	$0B ($2B)
WDTCR	Регістр керування вартовим таймером	$21 ($41)

* У дужках указано адреси РВВ, якщо вони адресуються як комірки статичної пам’яті даних.

До будь-якого регістра введення/виведення можна звернутися за допомогою команд IN і OUT, що виконують пересилання даних між одним з тридцятидвох РЗП та РВВ. В чотирьох командах порозрядного доступу,

операндами є регістри введення/виведення: команди встановлення/скидання окремого розряду РВВ – SBI та CBI і команди перевірки стану окремого розряду – SBIS і SBIC. Останні чотири команди можуть звертатися тільки до першої половини регістрів введення/виведення – адреси $00...$1F.

До регістрів введення/виведення можна звертатися двома способами:

- командами IN і OUT;
- командами роботи з СПД.

У першому випадку використовуються адреси РВВ, що належать основному простору введення/виведення – $00...$3F. У другому випадку адресу простору введення/виведення РВВ необхідно збільшити на $20. Далі при наведенні адрес РВВ після адреси простору введення/виведення в дужках вказуються відповідні їм адреси комірок СПД, яка включає: РЗП, РВВ та СОЗП.

Нижче розглянуто деякі службові регістри мікроконтролера, адреси яких, як правило, не змінюються від моделі до моделі. Наприклад, регістр SREG завжди розташовано за адресою $3F ($5F), регістр GIMSK – за адресою $3B ($5B) і т. ін.

Регістр стану SREG

Регістр стану SREG являє собою набір прапорців, які показують поточний стан програми, яку виконує мікроконтролер. Ці прапорці автоматично встановлюються в одиницю, або скидаються в нуль відповідно до результату виконання команд, які впливають на стан прапорців. Усі розряди цього регістра доступні як для читання, так і для запису в будь-який момент часу. Після скидання мікроконтролера всі розряди регістра скидаються в нуль. Вміст цього регістра та його опис наведено вище у 2.3.

Регістр-покажчик стека SP

У мікроконтролерах, які мають ємність СОЗП до 256 байт, покажчик стека реалізовано на одному регістрі SPL (SP), який розташовано за адресою $3D ($5D). В іншому випадку покажчик стека виконано на парі регістрів SPH:SPL, які відповідно мають адреси: $3E ($5E) і $3D ($5D). Усі розряди цих регістрів доступні як для читання, так і для запису в будь-який момент часу. Після скидання мікроконтролера вміст регістрів дорівнює нулю. Тому на початку програми покажчик стека треба завантажити відповідним значенням – як правило, це найбільша адреса СПД для конкретного мікроконтролера.

Регістр стану мікроконтролера MCUSR

Регістр стану мікроконтролера містить прапорці, стан яких дозволяє визначити причину, через яку відбулося скидання мікроконтролера [1; 7]. Регістр розташовано за адресою $34 ($54).

3.4. Функціонування конвеєра

Для підвищення швидкодії AVR-мікроконтролерів при виконанні програми використовується дворівневий конвеєр (рис. 3.4).

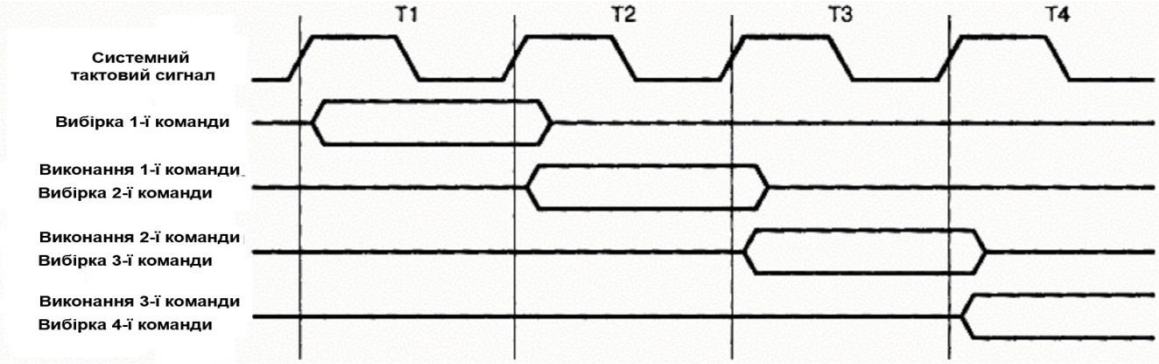


Рис. 3.4. Послідовність виконання команд у конвеєрі

Під час першого машинного циклу відбувається вибірка команди з пам'яті програм та її декодування.

У наступному циклі ця команда виконується і паралельно відбувається вибірка і декодування другої команди і так далі. Фактичний час виконання більшості команд дорівнює одному машинному циклу, що дозволяє досягати продуктивності до 1 MIPS (Million Instructions Per Second) на один мегагерц тактової частоти.

Завдяки підключенню АЛП безпосередньо до регістрового файлу (РЗП) він виконує одну команду – читання вмісту двох регістрів, виконання операції і запис результату в регістр-приймач за один такт (рис. 3.5).

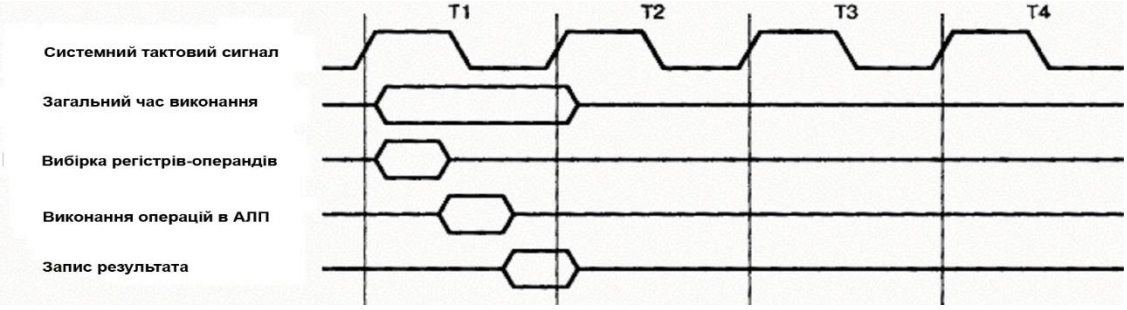


Рис. 3.5. Функціонування АЛП

Звернення до внутрішнього СОЗП виконується за два такти [1; 7].

Вище було описано послідовність виконання команд програми в ідеальному випадку. Однак в дійсності дуже часто відбувається порушення нормального порядку функціонування конвеєра. Прикладом команд, що викликають таке порушення, є команди умовного переходу, а також команди типу «перевірка/пропуск» – пропускають наступну команду, якщо результат перевірки

позитивний. Якщо умова, що перевіряється командою умовного переходу, істинна, виконання програми повинно продовжуватись з деякої іншої адреси. Оскільки в конвеєрі уже відбулася вибірка команди, розташованої після команди переходу, час виконання команди переходу збільшується на один машинний цикл, під час якого відбувається вибірка команди, розташованої за адресою переходу.

При виконанні команд типу «перевірка/пропуск», наступна команда не виконується у разі істинності умови, яка перевіряється. Однак вибірка команди, що пропускається, уже відбулася. Внаслідок того, що команда не виконується, у конвеєрі утвориться «дірка», що полягає в пропуску одного або двох (в залежності від команди, що пропускається) машинних циклів. Відповідно, команди типу «перевірка/пропуск» виконуються за один машинний цикл, якщо результат перевірки умови негативний, і за два або три цикли, якщо він позитивний.

Команди безумовного переходу – JMP, RJMP і IJMP, команди виклику підпрограм – CALL, RCALL і ICALL і команди повернення з підпрограм – RET і RETI також змінюють вміст лічильника команд – PC, що викликає перехід у пам'яті програм. У результаті виконання цих команд відбувається «розрив» у роботі конвеєра, у наслідок чого виникає затримка виконання програми на декілька машинних циклів. В залежності від команди тривалість затримки становить від двох до чотирьох машинних циклів.

З тієї ж причини порушення нормального функціонування конвеєра відбувається і при виникненні переривання. Максимальна затримка при цьому становить 4 машинних цикли.

3.5. Лічильник команд

Одним з важливих регістрів мікроконтролера є лічильник команд (Program Counter) – PC. Цей регістр при виконанні програми використовується для адресації комірок пам'яті програм.

Після увімкнення живлення, а також після скидання мікроконтролера в лічильник програм автоматично завантажується значення $000. За цією адресою розташовується команда відносного переходу до частини програми, яка виконує ініціалізацію, – RJMP.

Розмір лічильника команд залежить від ємності адресованої пам'яті. При нормальному виконанні програми вміст лічильника команд автоматично збільшується на 1, або на 2 в залежності від довжини виконуваної команди. Цей порядок порушується при виконанні команд переходу, виклику і повернення з підпрограм, а також при виникненні переривань. При виникненні переривання в лічильник команд завантажується адреса відповідного вектора переривання – $001...$010. Якщо переривання використовуються в програмі, за цими адресами

повинні розміщатися команди відносного переходу до підпрограм обробки переривань.

Доступ із програми безпосередньо до лічильника команд не можливий.

3.6. Способи адресації операндів

Загальні відомості

AVR-мікроконтролери підтримують наведені нижче способи адресації операндів – даних, що беруть участь в операціях:

- неявна;
- безпосередня;
- пряма;
- непряма адресації.

Деякі способи адресації мають кілька різновидів в залежності від того, до якої області пам'яті виконується звернення – при прямій адресації, або які додаткові дії виконуються над індексним регістром – при непрямій адресації.

Неявна адресація

При неявній адресації в команді відсутня адреса операнда в явному вигляді. Інформацію про потрібну адресу операнда контролер отримує з коду операції команди, тобто кажуть, що операнд адресується неявно. Так, наприклад, можуть адресуватися окремі прапорці регістра стану в командах типу BRTS k; BRVS k; BRPL k і т. ін.

Безпосередня адресація

При безпосередній адресації операнд входить в саму команду та читається із пам'яті програм у складі машинного коду команди. Це, наприклад, команди ADIW Rd, K6 (K6 = 6 біт); SBCI Rd*, K (K = 8біт) і т. ін.

Пряма адресація

При прямій адресації адреси операндів містяться у машинному коді команди. Ця адресація використовується для звернення до комірок СПД. У відповідності до її структури існують наступні різновиди прямої адресації: пряма адресація одного РЗП, пряма адресація двох РЗП, пряма адресація РВВ, пряма адресація всієї СПД – РЗП, РВВ та СОЗП.

Пряма адресація одного регістра загального призначення

Цей спосіб адресації використовується в командах, що оперують з одним з регістрів загального призначення. При цьому адреса регістра-операнда (його номер) міститься в розрядах 8...4 (5 біт) машинного коду команди (рис. 3.7).

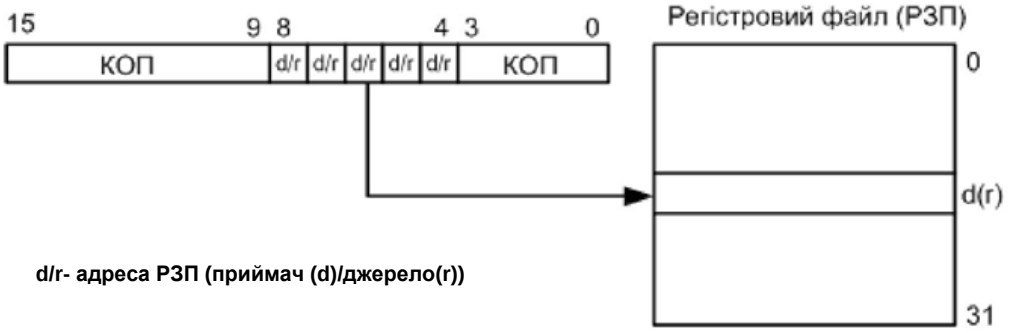


Рис. 3.7. Пряма адресація одного регістра загального призначення

Прикладом команд, що використовують цей спосіб адресації, є команди роботи зі стеком – PUSH, POP, команди інкременту – INC, декременту – DEC, а також деякі команди арифметичних операцій.

Пряма адресація двох регістрів загального призначення

Цей спосіб адресації використовується в командах, що оперують одночасно двома регістрами загального призначення. При цьому адреса регістра-джерела міститься в розрядах 9, 3...0 (5 біт), а адреса регістра-приймача в розрядах 8...4 (5 біт) машинного коду команди (рис. 3.8).

До команд, що використовують цей спосіб адресації, належить команда пересилання даних з одного РЗП у другий – MOV, а також більшість команд арифметичних операцій.

Деякі команди мають тільки один регістр-операнд, але використовують цей спосіб адресації. У цьому випадку той самий регістр є джерелом і приймачем. Як приклад можна навести команду очищення регістра – CLR Rd, яка виконує операцію «виключне АБО» регістра із самим собою – EOR Rd, Rd.

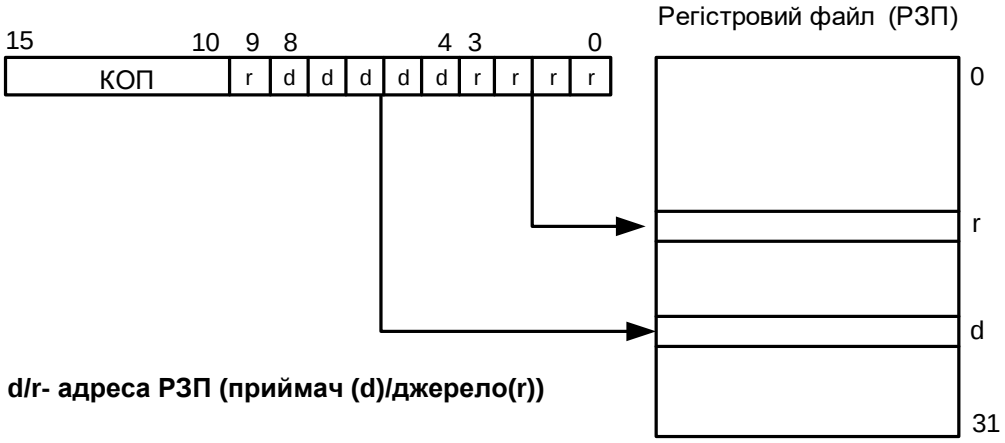


Рис. 3.8. Пряма адресація двох регістрів загального призначення

Пряма адресація регістра введення/виведення

Цей спосіб адресації використовується командами пересилання даних між регістром введення/виведення і РЗП (регістровим файлом) – IN і OUT. У цьому разі адреса регістра введення/виведення міститься в розрядах 10, 9, 3...0 – 6 біт, а адреса РЗП – у розрядах 8...4 – 5 біт машинного коду команди (рис. 3.9).

Рис. 3.9. Пряма адресація регістрів введення/виведення

Пряма адресація статичної пам’яті даних

Цей спосіб використовується при зверненні до всього адресного простору статичної пам’яті даних, яка включає: РЗП, РВВ та СОЗП.

Є тільки дві команди, що використовують цей спосіб адресації. Це команди пересилання байта між одним з РЗП і коміркою СПД – LDS і STS. Кожна з цих команд займає в пам’яті програм два слова (32 біти). У першому слові міститься код операції та адреса регістра загального призначення (у розрядах з 8-го по 4-й). У другому слові міститься адреса комірки пам’яті, до якої відбувається звернення (рис. 3.10).

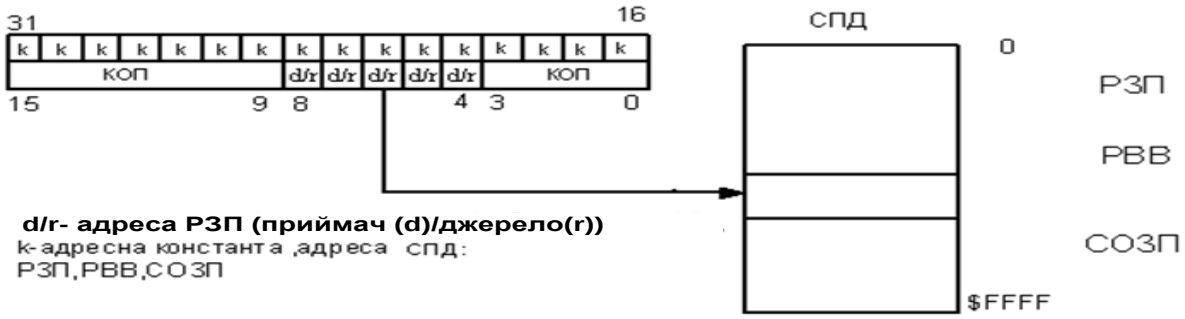


Рис. 3.10. Пряма адресація статичної пам`яті даних

У СПД за адресами $00...$1F розташовано регістри загального призначення, за адресами $20...$5F – регістри введення/виведення, а за адресами $60...$ FF – СОЗП.

Непряма адресація

При непрямій адресації адреса комірки статичної пам’яті даних міститься в одному з індексних регістрів X, Y і Z.

Є наступні різновиди непрямої адресації:

- проста непряма адресація;
- відносна непряма адресація;
- непряма адресація з переддекрементом;
- непряма адресація з постінкрементом.

Проста непряма адресація

При використанні простої непрямої адресації звернення виконується до СПД за адресою, що міститься в одному з індексних регістрів: X, Y або Z (рис. 3.11). Жодних дій із вмістом індексного регістра при цьому не виконується.

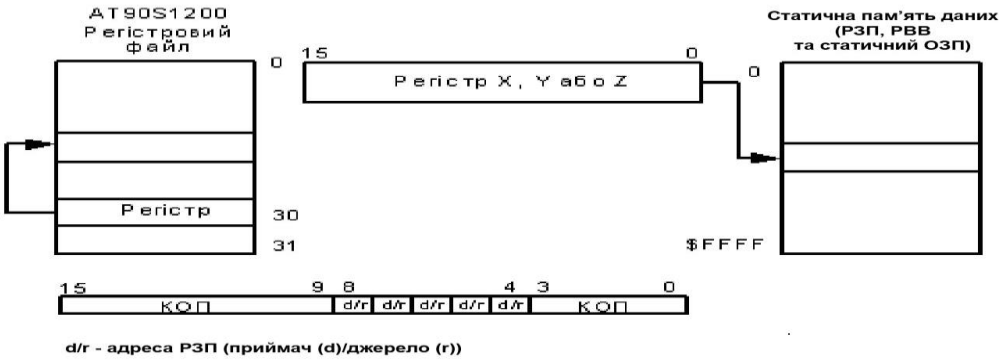


Рис. 3.11. Проста непряма адресація

Мікроконтролери підтримують 6 команд (по 2 для кожного індексного регістра) простої непрямої адресації: LD Rd, X/Y/Z (пересилання байта з СПД в РЗП) і ST X/Y/Z, Rr (пересилання байта з РЗП в СПД). Адреса регістра загального призначення міститься в розрядах 8...4 машинного коду команди.

Відносна непряма адресація

У командах відносної непрямої адресації адреса комірки статичної пам’яті даних, до якої виконується звернення, обчислюється додаванням вмісту індексного регістра – Y або Z і константи, що міститься в машинному коді команді (рис. 3.12).

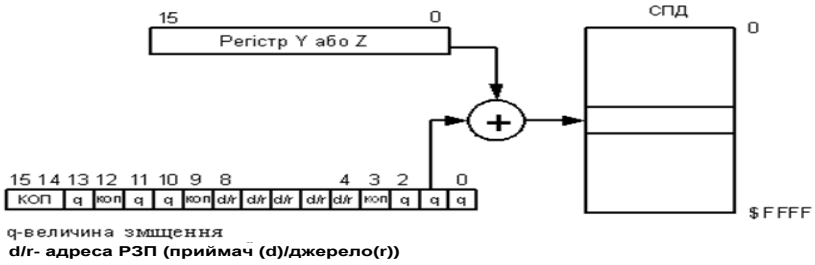


Рис. 3.12. Відносна непряма адресація

Мікроконтролери підтримують 4 команди відносної непрямої адресації (дві для регістра Y і дві для регістра Z): LDD Rd, Y+q/Z+q (пересилання байта із СПД у РЗП) і STD Y+q/Z+q, Rr (пересилання байта із РЗП у СПД). Адреса регістра загального призначення міститься в розрядах 8...4 машинного коду команди, а величина зміщення (q) – у розрядах 13, 11, 10, 2...0. Оскільки під значення зміщення виділяється тільки 6 біт, воно не може перевищувати 63 ($0 \le q \le 63$).

Непряма адресація з попереднім декрементом (переддекрементом)

При виконанні команд непрямої адресації з переддекрементом вміст індексного регістра спочатку зменшується на одиницю, а потім виконується звернення за отриманою адресою (рис. 3.13).

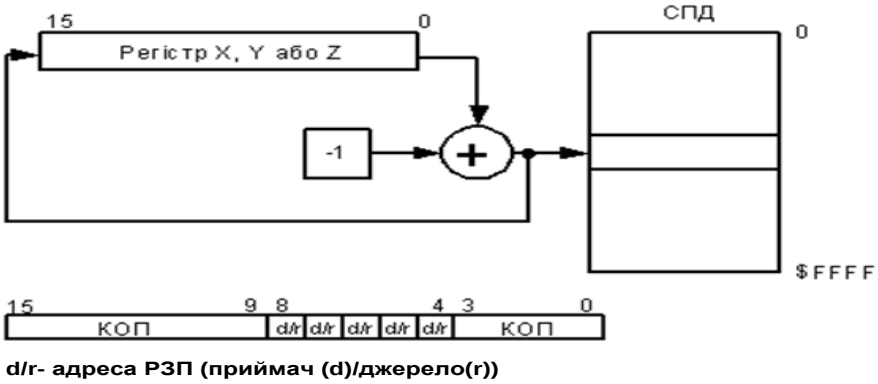


Рис. 3.13. Непряма адресація з переддекрементом

Мікроконтролери сімейства підтримують 6 команд (по 2 для кожного індексного регістра) непрямої адресації з переддекрементом: LD Rd, -X/-Y/-Z (пересилання байта з СПД в РЗП) і ST -X/-Y/-Z, Rr (пересилання байта з РЗП в СПД). Адреса регістра загального призначення міститься в розрядах 8...4 машинного коду команди, а регістри X/Y/Z адресуються неявно.

Непряма адресація з постінкрементом

При виконанні команд непрямої адресації з постінкрементом (наступним інкрементом) після звернення за адресою, що міститься в індексному регістрі, вміст індексного регістра збільшується на одиницю (рис. 3.14).

Мікроконтролери сімейства підтримують 6 команд (по 2 для кожного індексного регістра) непрямої адресації з постінкрементом: LD Rd, X+/Y+/Z+ (пересилання байта з СПД в РЗП) і ST X+/Y+/Z+, Rr (пересилання байта з РЗП в СПД). Адреса регістра загального призначення міститься в розрядах 8...4 машинного коду команди, а регістри X/Y/Z адресуються неявно.

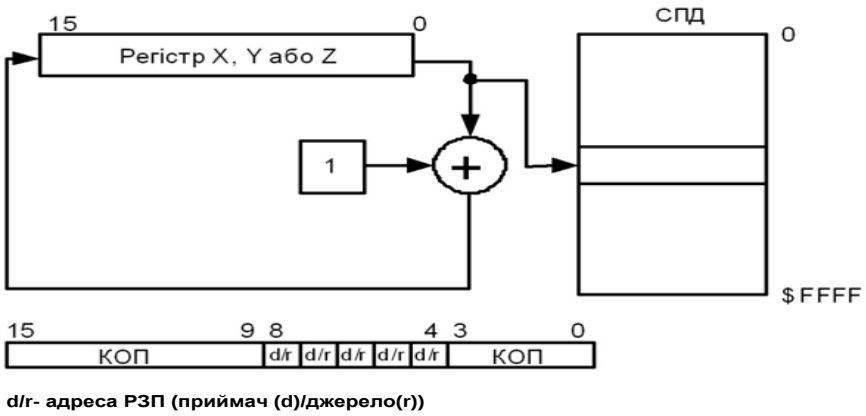


Рис. 3.14. Непряма адресація з постінкрементом

Непряма адресація пам’яті програм

Такий спосіб адресації використовується в командах IJMP, ICALL (рис. 3.15).

В результаті виконання такої команди програма продовжує виконуватися з адреси, що міститься в індексному регістрі Z. Таким чином, дія команди зводиться до завантаження вмісту індексного регістра в лічильник команд.

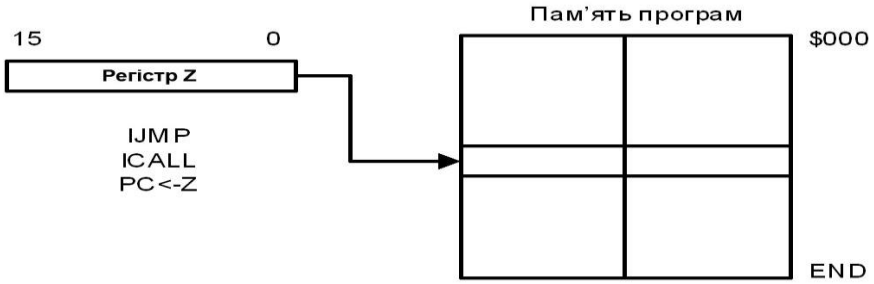


Рис. 3.15. Непряма адресація пам’яті програм

На відміну від команд відносного переходу ця команда не має обмежень за областю дії. Оскільки індексний регістр – 16-розрядний, то максимально можлива величина переходу становить 64 Кслів (128 Кбайт).

Як і команда відносного переходу, команда непрямого переходу виконується за 2 машинних цикли.

Непряма адресація констант в пам’яті програм

Цей спосіб адресації використовується в команді LPM, яка завантажує один байт із пам’яті програм у регістр загального призначення R0 (рис. 3.16).

Адреса комірки пам’яті, до якої відбувається звернення, міститься в індексному регістрі Z. При цьому старші п’ятнадцять розрядів регістра – Z1…Z15 адресують слово в пам’яті програм, а молодший біт – Z0 адресує байт в обраному слові. Якщо Z0 = 0, то адресується молодший байт (МБ) слова, а якщо Z0 = 1 – старший байт (СБ).

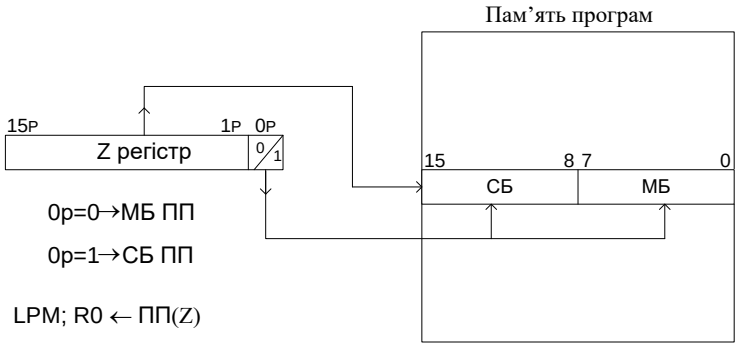


Рис. 3.16. Непряма адресація констант в пам’яті програм

Доступний об’єм пам’яті програм для цієї команди не може перевищувати 2^{15} = 32768 слів. Для звернення до розширеної пам’яті програм може використовуватися команда ELPM.

Відносна адресація пам’яті програм

Цей спосіб адресації використовується в командах RJMP, RCALL (рис. 3.17).

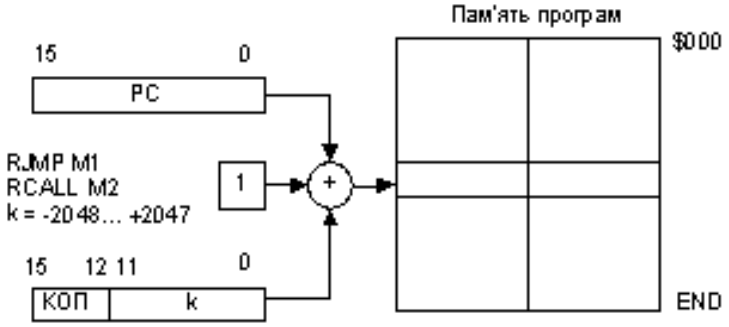


Рис. 3.17. Відносна адресація пам’яті програм

При виконанні команди до поточного вмісту лічильника команд (адреси наступної команди) додається дванадцятирозрядне число зі знаком – k, яке представлено у додатковому двійковому коді.

Ця команда має обмеження за областю дії. Через те, що операнд являє собою 12-розрядне число зі знаком, максимальна величина переходу відносно адреси наступної команди становить від -2048 до +2047 слів (приблизно, ±4 Кбайт).

У програмах в якості операндів цієї команди замість адрес використовуються мітки. Компілятор обчислює величину зміщення відносно адреси наступної команди – k. Для цього він віднімає від адреси мітки адресу наступної команди і підставляє це значення у відповідні розряди машинного коду команди.

3.7. Загальна характеристика команд AVR-мікроконтролерів

Мнемоніка команди та мнемокод

Для написання програм мовою «Асемблер» використовують мнемоніки команд та мнемокоди. Мнемоніки введені для полегшення написання програм, і складають основу мови *асемблера.*

Мнемоніка команди – це представлення коду операції у вигляді сполучення латинських літер, що мають визначений зміст (використовуються англійські слова або скорочення, наприклад, MOV, PUSH, POP, JMP, CLR, NOP і т. ін.*)*.

Мнемокод включає в себе мнемоніку команди та символічний опис операндів, які беруть участь в операції.

Код операції команди

Код операції команди (КОП) – це комбінація бітів (не більше восьми для 8-розрядних МК), що знаходяться на початку машинного коду команди і визначають тип операції, що підлягає виконанню у конкретний момент часу. КОП читається з пам’яті і розміщується в програмно недоступний регістр команд мікроконтролера. Потім він декодується і визначається тип команди, яка повинна бути виконана. Крім КОП в машинний код команди можуть входити адреси та операнди.

Машинний код команди

Машинний код команди представляє собою двійковий код команди, який формується компілятором та складається з одного або декількох байтів в залежності від типу команд конкретного МК.

Операнди

Операндами у мікропроцесорній техніці називають дані, які приймають участь у виконанні тієї чи іншої команди (операції). В залежності від способу адресації операндів, дані можуть знаходитися у регістрах, пам'яті, у машинному коді команди і т. ін.

3.8. Формати команд

Більшість мікроконтролерів сімейства AVR мають 14 форматів (типів) базового набору команд, наведених на рис. 3.18. На цьому рисунку використано наступні скорочення: КОП – код операції; K – константа даних; k – адресна константа; b –номер біта в регістрі введення/виведення або регістрового файлу (РЗП) – 3 біти; s – номер біта в регістрі стану – 3 біти; A – адреса регістра в просторі введення/виведення; q – зміщення для непрямої адресації – 6 біт; d(r) – регістр-приймач (джерело) з області регістрового файлу.

До першого типу належать команди, код операції яких займає всю довжину команди – 16 біт. До цієї групи належить, наприклад, команда LPM – завантаження регістра загального призначення R0 з пам'яті програм за адресою, яка міститься у регістровій парі Z = R31:R30. Такий формат мають також команди IJMP, ICALL, RET, RETI, NOP, SLEEP та WDT.

До другого типу належать команди, які окрім коду операції містять п'ятирозрядну адресу одного з регістрів загального призначення. Наприклад, це команди DEC Rd; LD Rd, -X; ST Y, Rr і т. ін.

Третій тип мають команди, в яких адресуються два операнди – регістри загального призначення: Rd – приймач, Rr – джерело. Це, наприклад, команди ADD Rd, Rr; CPSE Rd, Rr; AND Rd, Rr і т. ін

До четвертого типу належать двооперандні команди, в яких один операнд: K6 = 6 біт (K = 0...63) входить в саму команду – безпосередня адресація, а другим є пара регістрів: R24, R25; R26, R27; R28, R29; R30, R31, які кодуються двома бітами команди – dd: 00 – R24, R25; 01 – R26, R27; 10 – R28, R29; 11 – R30, R31. Наприклад, команди ADIW Rdl, K6; SBIW Rdl, K6, де Rdl = R24/R26/R28/R30.

П'ятий тип мають двооперандні команди, наприклад, SBCI Rd*, K; ORI Rd*, K, в яких Rd* – один із шістнадцяти РЗП (R16...R31), а другий K – восьмибітна константа (безпосередній операнд): $0 \le K \le 255$.

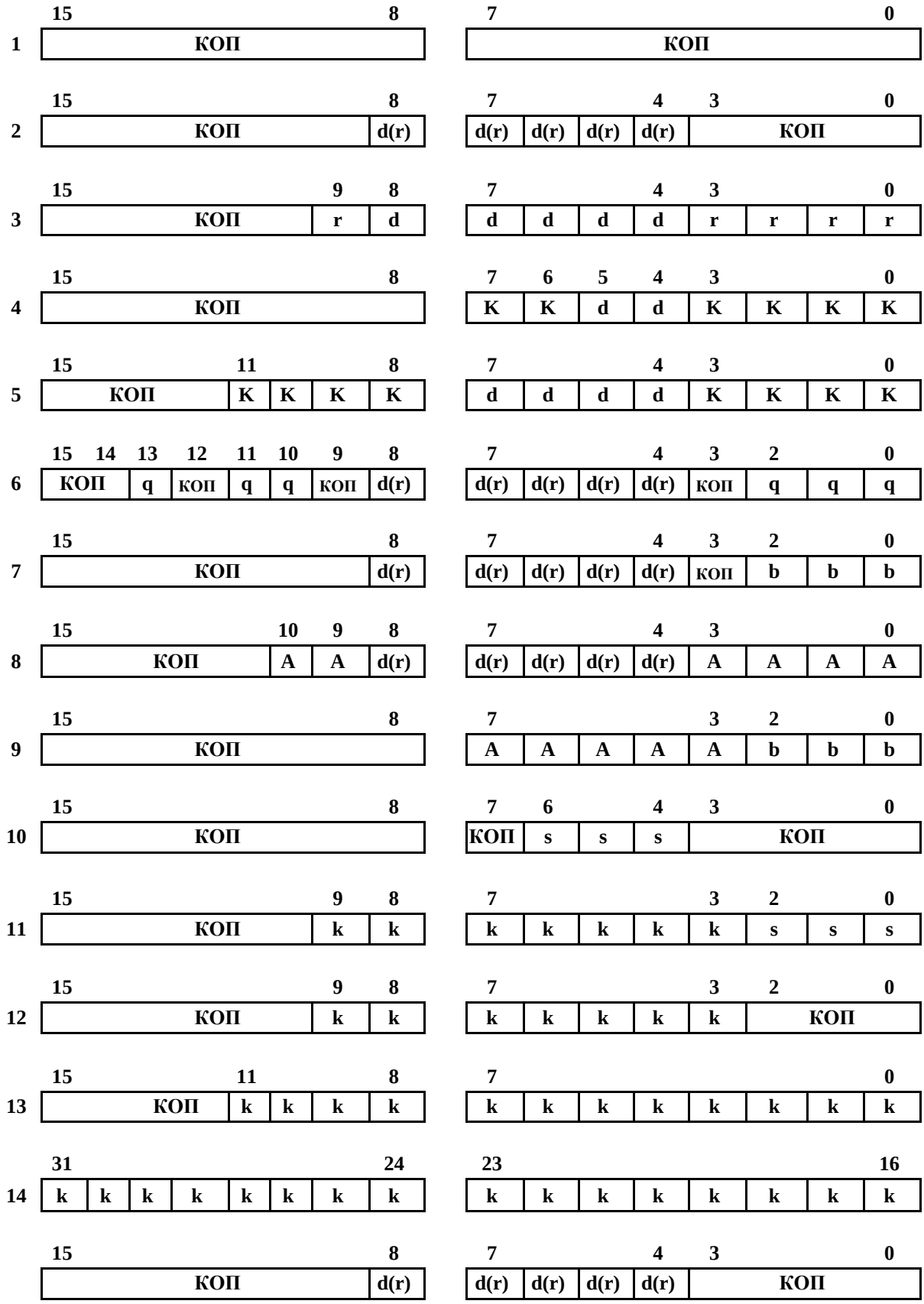


Рис. 3.18. Формати базового набору команд

До шостого типу належать команди, в яких один з двох операндів може бути РЗП (Rd – приймач, Rr – джерело), а другий міститься у СПД і адресується за допомогою непрямої відносної адресації, коли вміст індексного регістра Y або Z додається до зміщення q ($0 \le q \le 63$). Це, наприклад, команди LDD Rd, z+q; STD Y+q, Rr і т. ін.

Сьомий тип мають команди, в яких адресується один із восьми бітів РЗП. Наприклад, команди BLD Rd, b; BST Rr, b; SBRC Rr, b і т. ін., де Rd – приймач, Rr – джерело (один з 32-х РЗП), а b = 0...7 – номер біта РЗП.

До восьмого типу належать команди, в яких одним операндом є РЗП (Rd – приймач/Rr – джерело), а другий міститься в одному з 64-х РВВ. Наприклад, команди IN Rd, P; OUT P, Rr, де P – адреса РВВ (P = 0…63).

Дев'ятий тип мають команди, в яких адресуються окремі біти молодшої половини РВВ (P* = 0...31). Наприклад, команди SBI P*, b, де b = 0...7 – номер біта РВВ.

До десятого типу належать команди, в яких адресується один із 8-ми бітів регістра прапорців SREG. Наприклад, команди BSET s; BCLR s, де s = 0...7 – номер біта у регістрі стану.

Одинадцятий тип мають команди умовного переходу в залежності від значення вказаних бітів регістра стану. Наприклад, команди BRBS s, k; BRBC s, k, де s = 0...7 – номер біта регістра стану SREG, а k = 7 біт ($-64 \le k \le 63$) при переході додається до адреси наступної команди.

Дванадцятий тип мають команди умовного відносного переходу у залежності від значень окремих прапорців регістра стану (прапорці адресуються неявно). Наприклад, команди BREQ k; BRCC k і т. ін., де k = 7 біт ($-64 \le k \le 63$) при переході додається до адреси наступної команди.

До тринадцятого типу належать команди відносного безумовного переходу: RJMP k та відносного безумовного виклику підпрограм: RCALL k, де k = 12 біт. Діапазон переходів та виклику підпрограм: -2048 + 2047).

Чотирнадцятий тип мають команди: LDS Rd, k – пряме завантаження та STS k, Rr – пряме збереження, в яких одним операндом є РЗП (Rd – приймач/Rr – джерело), а другий міститься у СПД (РЗП, РВВ та СОЗП). Оскільки k = 16 біт, то кількість комірок СПД = 2^{16} = 65536.

3.9. Формати даних

При роботі з мікроконтролером необхідно знати не тільки формати (типи) команд (рис. 3.18), які керують роботою мікроконтролера, але й формати (типи) даних (операндів), що беруть участь у виконанні тієї або іншої команди (операції) (рис. 3.19).

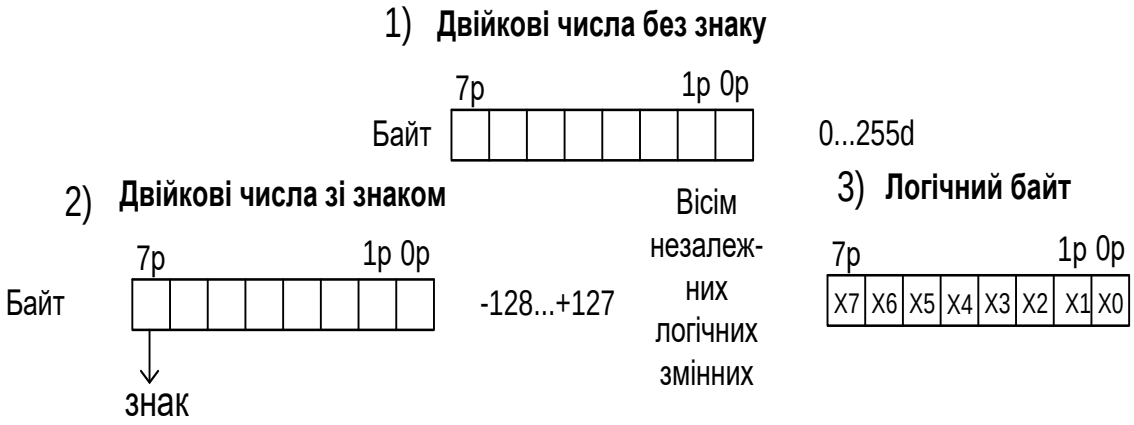


Рис. 3.19. Типи (формати) даних мікроконтролера

3.10. Довжина команд у байтах та їх розміщення у пам'яті програм

Відповідно до рис. 3.18 команди мікроконтролерів сімейства AVR мають довжину – одне слово (2 байти), за виключенням команди чотирнадцятого типу, яка має довжину 2 слова (4 байти).

Програма, яка керує роботою МК, послідовно, команда за командою, розміщується в сусідніх комірках пам'яті в порядку зростання їх адрес. Адреса команди, яка повинна виконуватись, знаходиться у програмному лічильнику PC. Пристрій керування мікроконтролера на основі прочитаного коду операції, що міститься в першому байті команди, визначає, скільки ще байтів міститься в команді, та керує їх читанням з пам'яті шляхом збільшення на одиницю адреси, яка видається при кожному зверненні до ЗП. Після читання з пам'яті чергової команди МК формує адресу КОП наступної команди.

3.11. Вплив команд на прапорці

Як відзначалося раніше, одним з основних регістрів МК є регістр прапорців (ознак), який було описано у підрозділі. 2.3. Прапорці регістра ознак встановлюються під час виконання ряду команд МК. Під час опису системи команд, яка, як правило, оформлюється у вигляді таблиці, в одній з колонок показується вплив окремих команд на ті або інші прапорці (табл. 3.2, 3.4).

Як правило, команди пересилання не змінюють прапорці, окрім команд, які пересилають дані у регістр прапорців. Частіше прапорці змінюють арифметичні, логічні команди і команди порівняння.

Окремі прапорці (ознаки) аналізуються під час виконання команд умовних переходів, виклику і повернення з підпрограм, що дозволяє передавати керування в програмі в залежності від поточного значення прапорців.

3.12. Час виконання команд

В AVR-мікроконтролерах окремі команди виконуються за 1, 2, 3 або 4 такти. Тривалість одного такту дорівнює одному періоду тактової частоти f_{BQ}.

3.13. Базовий набір команд мікроконтролера

Загальні відомості

AVR-мікроконтролери мають RISC-архітектуру, основною перевагою якої є збільшення швидкодії за рахунок скорочення кількості операцій обміну з пам'яттю програм. Практично всі команди займають одну комірку пам'яті. Виняток становлять команди, в яких одним з операндів є 16-розрядна адреса СПД (РЗП, РВВ та СОЗП). Підвищення швидкодії досягнуто не за рахунок скорочення кількості команд процесора, а за рахунок збільшення розрядності комірки пам'яті програм до 16. Більшість команд виконується за один машинний цикл (1 такт).

Всі команди AVR-мікроконтролерів можна розділити на декілька груп:

- логічні операції;
- арифметичні операцій та команди зсуву;
- операції з бітами;
- команди пересилання даних;
- команди передачі керування;
- команди керування системою.

Нижче скорочено описано ці команди. Детальнішу інформацію наведено в [1; 7].

У табл. 3.2 наведено базовий набір команд, якими можна користуватись в типовому AVR-мікроконтролері, а також основні відомості про команди, такі як мнемонічне позначення команди, її опис, тип, кількість тактів, необхідних для її виконання, а також прапорці регістра SREG, на які впливає ця команда.

В цій таблиці використано наступні позначення:

Rd – регістр-приймач результату, $0 \le d \le 31$;

Rd* – регістр-приймач результату, $16 \le d \le 31$;

Rdl – R24, R26, R28, R30 для команд ADIW і SBIW ;

Rr – регістр-джерело;

P – адреса регістра введення/виведення;

P* – адреса регістра введення/виведення, який адресується побітово (адреси $00…$1F);

K – символьна або числова константа (8 біт);

K6 – символьна або числова константа (6 біт);

k – адресна константа ;

b – номер біта в регістрі (3 біти);

q – константа (6 біт), може бути константний вираз;

s – номер біта в регістрі стану (3 біти);

X, Y, Z – індексні регістри непрямої адресації (X = R27:R26, Y = R29:R28, Z = R31:R30).

Нижче наведено короткий опис окремих команд базового набору.

Арифметичні операції і команди зсуву

До даної групи належать команди, що виконують такі операції, як додавання, віднімання, зсув – вправо/вліво та інкремент/декремент. Усі операції виконуються тільки над регістрами загального призначення. Операнди можуть бути знаковими/беззнаковими числами, а також подані у додатковому коді.

Усі команди цієї групи виконуються за один машинний цикл, за винятком команд, що оперують двобайтовими значеннями, що виконуються за два цикли.

Логічні операції

Ці команди дозволяють виконувати стандартні логічні операції над байтами: «логічне множення» – І, «логічне додавання» – АБО, операцію «виключне АБО», а також обчислення зворотного і додаткового кодів числа. До цієї групи можна віднести також команди очищення/встановлення регістрів і команду перестановки тетрад. Всі логічні операції виконуються за один машинний цикл над регістрами загального призначення, а результат зберігається в одному з РЗП.

Команди операцій з бітами

До даної групи належать команди, що виконують встановлення або скидання заданого розряду РЗП або РВВ. Є також команди для зміни розрядів регістра стану SREG, тому що перевірка стану розрядів саме цього регістра виконується найчастіше. Умовно до цієї групи можна віднести також дві команди передачі керування типу «перевірка/пропуск», що пропускають наступну команду в залежності від стану розряду РЗП або РВВ.

Усі задіяні розряди РВВ мають свої символічні імена. Визначення цих імен описано у тому ж файлі, що й визначення символічних імен для адрес регістрів [1; 7]. Таким чином, після включення у програму зазначеного файлу в командах замість числових значень номерів розрядів можна буде вказувати їхні символічні імена.

Таблиця 3.2. **Базовий набір команд AVR-мікроконтролера**

№	Мнемонічне позначення	Операнди	Опис	Операція	Прапорці	Тип	Кільк. тактів
АРИФМЕТИЧНІ, ЛОГІЧНІ КОМАНДИ ТА КОМАНДИ ЗСУВУ							
1	ADD	Rd, Rr	Додавання без перенесення	Rd ← Rd + Rr	Z,C,N, V,H,S	3	1
2	ADC	Rd, Rr	Додавання з перенесенням	Rd ← Rd + Rr + C	Z,C,N, V,H,S	3	1
3	ADIW	Rdl, K6	Додавання двох байт із константою	Rdh:Rdl ← Rdh:Rdl+K6	Z,C,N,V,S	4	2
4	SUB	Rd, Rr	Віднімання без перенесення	Rd ← Rd – Rr	Z,C,N, V,H,S	3	1
5	SUBI	Rd*, K	Віднімання константи	Rd* ← Rd* – K	Z,C,N, V,H,S	5	1
6	SBC	Rd, Rr	Віднімання з перенесенням	Rd ← Rd – Rr – C	Z,C,N, V,H,S	3	1
7	SBCI	Rd*, K	Віднімання константи з перенесенням	Rd* ← Rd* – K – C	Z,C,N, V,H,S	5	1
8	SBIW	Rdl, K6	Віднімання з двох байт константи	Rdh:Rdl ← Rdh:Rdl – K6	Z,C,N,V,S	4	2
9	AND	Rd, Rr	Логічне І	Rd ← Rd • Rr	Z,N,V,S	3	1
10	ANDI	Rd*, K	Логічне І з константою	Rd* ← Rd* • K	Z,N,V,S	5	1
11	OR	Rd, Rr	Логічне АБО	Rd ← Rd v Rr	Z,N,V,S	3	1
12	ORI	Rd*, K	Логічне АБО з константою	Rd* ← Rd* v K	Z,N,V,S	5	1
13	EOR	Rd, Rr	Логічне виключне АБО	Rd ← Rd ⊕ Rr	Z,N,V,S	3	1
14	COM	Rd	Побітова інверсія	Rd ← $FF – Rd	Z,C,N,V,S	2	1
15	NEG	Rd	Зміна знака (додатковий код)	Rd ← $00 – Rd	Z,C,N, V,H,S	2	1
16	INC	Rd	Інкремент значення регістра	Rd ← Rd + 1	Z,N,V,S	2	1
17	DEC	Rd	Декремент значення регістра	Rd ← Rd – 1	Z,N,V,S	2	1
18	ASR	Rd	Арифметичний зсув вправо	Rd(n) ← Rd(n+1), n=0..6, C←Rd(0), Rd(7)←Rd(7)	Z,C,N,V	2	1
19	LSL	Rd	Логічний/арифметичний зсув вліво	Rd(n+1) ← Rd(n), Rd(0) ← 0, C←Rd(7)	Z,C,N,V	2	1
20	LSR	Rd	Логічний зсув вправо	Rd(n) ← Rd(n+1), Rd(7) ← 0, C←Rd(0)	Z,C,N,V,S	2	1
21	ROL	Rd	Циклічний зсув вліво через С	Rd(0)←C,Rd(n+1)← Rd(n), C←Rd(7)	Z,C,N, V,H,S	2	1
22	ROR	Rd	Циклічний зсув вправо через С	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V,S	2	1
23	TST	Rd	Перевірка на нуль чи від'ємне значення	Rd ← Rd • Rd	Z,N,V,S	2	1
24	CLR	Rd	Очищення регістра	Rd ← Rd ⊕ Rd	Z,N,V,S	2	1
25	SWAP	Rd	Обмін тетрадами	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)	-	2	1
26	SER	Rd	Встановлення регістра	Rd ← $FF	-	2	1
КОМАНДИ ПЕРЕДАЧІ КЕРУВАННЯ							
1	RJMP	k	Відносний безумовний перехід	PC ← PC + k + 1	-	13	2
2	IJMP		Непрямий безумовний перехід	PC ← Z	-	1	2
3	RCALL	k	Відносний безумовний виклик підпрограми	STACK ←PC+1, PC ← PC + k + 1, SP←SP-2	-	13	3
4	ICALL		Непрямий безумовний виклик підпрограми	STACK ←PC+1, PC ← Z, SP←SP-2	-	1	3

Продовження табл. 3.2.

№	Мнемо-нічне позначе ння	Операнди	Опис	Операція	Прапорці	Тип	Кільк. тактів
5	RET		Повернення з підпрограми	PC ← STACK, SP←SP+2	-	1	4
6	RETI		Повернення з підпрограми обробки переривання	PC ← STACK, SP←SP+2, I←1	I	1	4
7	CPSE	Rd, Rr	Порівняти, пропустити, якщо рівні	if (Rd = Rr). то PC ← PC + 2/3	-	3	1/2/3
8	CP	Rd, Rr	Порівняти	Rd - Rr	Z,N,V, C,H,S	3	1
9	CPC	Rd, Rr	Порівняти з перенесенням	Rd - Rr - C	Z,N,V, C,H,S	3	1
10	CPI	Rd*, K	Порівняти з константою	Rd* - K	Z,N,V, C,H,S	5	1
11	SBRC	Rr, b	Пропустити, якщо біт у регістрі скинутий	if (Rr(b)=0), то PC ← PC + 2/3	-	7	1/2/3
12	SBRS	Rr, b	Пропустити, якщо біт у регістрі встановлений	if (Rr(b)=1), то PC ← PC + 2/3	-	7	1/2/3
13	SBIC	P*, b	Пропустити, якщо біт у порту скинутий	if (P*(b)=0), то PC ← PC + 2/3	-	9	1/2/3
14	SBIS	P*, b	Пропустити, якщо біт у порту встановлений	if (P*(b)=1), то PC ← PC + 2/3	-	9	1/2/3
15	BRBS	s, k	Перейти, якщо прапорець у SREG встановлений	if (SREG(s) = 1) then C←PC+k+1	-	11	1/2
16	BRBC	s, k	Перейти, якщо прапорець у SREG скинутий	if(SREG(s) = 0) then PC←PC+k+1	-	11	1/2
17	BREQ	k	Перейти, якщо дорівнює	if (Z = 1) then PC ← PC + k + 1	-	12	1/2
18	BRCS	k	Перейти, якщо прапорець перенесення встановлений	if (C = 1) then PC ← PC + k + 1	-	12	1/2
19	BRNE	k	Перейти, якщо не дорівнює	if (Z = 0) then PC ← PC + k + 1	-	12	1/2
20	BRCC	k	Перейти, якщо прапорець перенесення скинутий	if (C = 0) then PC ← PC + k + 1	-	12	1/2
21	BRSH	k	Перейти, якщо дорівнює або більше	if (C = 0) then PC ← PC + k + 1	-	12	1/2
22	BRLO	k	Перейти, якщо менше	if (C = 1) then PC ← PC + k + 1	-	12	1/2
23	BRMI	k	Перейти, якщо мінус	if (N = 1) then PC ← PC + k + 1	-	12	1/2
24	BRPL	k	Перейти, якщо плюс	if (N = 0) then PC ← PC + k + 1	-	12	1/2
25	BRGE	k	Перейти, якщо більше або дорівнює (зі знаком)	if (N ⊕ V= 0) then PC ← PC +k+1	-	12	1/2
26	BRLT	k	Перейти, якщо менше (зі знаком)	if (N ⊕ V= 1) then PC← PC+k+1	-	12	1/2
27	BRHS	k	Перейти, якщо прапорець половинного перенесення встановлений	if (H = 1) then PC ← PC + k + 1	-	12	1/2
28	BRHC	k	Перейти, якщо прапорець половинного перенесення скинутий	if (H = 0) then PC ← PC + k + 1	-	12	1/2
29	BRTS	k	Перейти, якщо прапорець T встановлений	if (T = 1) then PC ← PC + k + 1	-	12	1/2
30	BRTC	k	Перейти, якщо прапорець T скинутий	if (T = 0) then PC ← PC + k + 1	-	12	1/2
31	BRVS	k	Перейти, якщо прапорець переповнення встановлений	if (V = 1) then PC ← PC + k + 1	-	12	1/2
32	BRVC	k	Перейти, якщо прапорець переповнення скинутий	if (V = 0) then PC ← PC + k + 1	-	12	1/2
33	BRIE	k	Перейти, якщо переривання дозволені	if (I = 1) then PC ← PC + k + 1	-	12	1/2
34	BRID	k	Перейти, якщо переривання заборонені	if (I = 0) then PC ← PC + k + 1	-	12	1/2

Продовження табл. 3.2.

№	Мнемо-нічне позначення	Операнди	Опис	Операція	Прапорці	Тип	Кіл. тактів
КОМАНДИ ПЕРЕСИЛАННЯ ДАНИХ							
1	MOV	Rd, Rr	Копіювання регістра	Rd ← Rr	-	3	1
2	LDI	Rd*, K	Завантаження константи	Rd* ← K	-	5	1
3	LD	Rd, X	Непряме завантаження	Rd ← (X)	-	2	2
4	LD	Rd, X+	Непряме завантаження з постінкрементом	Rd ← (X), X ← X + 1	-	2	2
5	LD	Rd, -X	Непряме завантаження з переддекрементом	X ← X - 1, Rd ← (X)	-	2	2
6	LD	Rd, Y	Непряме завантаження	Rd ← (Y)	-	2	2
7	LD	Rd, Y+	Непряме завантаження з постінкрементом	Rd ← (Y), Y ← Y + 1	-	2	2
8	LD	Rd, -Y	Непряме завантаження з переддекрементом	Y ← Y - 1, Rd ← (Y)	-	2	2
9	LDD	Rd, Y+q	Непряме завантаження зі зміщенням	Rd ← (Y + q)	-	6	2
10	LD	Rd, Z	Непряме завантаження	Rd ← (Z)	-	2	2
11	LD	Rd, Z+	Непряме завантаження з постінкрементом	Rd ← (Z), Z ← Z+1	-	2	2
12	LD	Rd, -Z	Непряме завантаження з переддекрементом	Z ← Z - 1, Rd ← (Z)	-	2	2
13	LDD	Rd, Z+q	Непряме завантаження зі зміщенням	Rd ← (Z + q)	-	6	2
14	LDS	Rd, k	Пряме завантаження	Rd ← (k)	-	14	2
15	ST	X, Rr	Непряме збереження	(X) ← Rr	-	2	2
16	ST	X+, Rr	Непряме збереження з постінкрементом	(X) ← Rr, X ← X + 1	-	2	2
17	ST	- X, Rr	Непряме збереження з переддекрементом	X ← X - 1, (X) ← Rr	-	2	2
18	ST	Y, Rr	Непряме збереження	(Y) ← Rr	-	2	2
19	ST	Y+, Rr	Непряме збереження з постінкрементом	(Y) ← Rr, Y ← Y + 1	-	2	2
20	ST	- Y, Rr	Непряме збереження з переддекрементом	Y ← Y - 1, (Y) ← Rr	-	2	2
21	STD	Y+q, Rr	Непряме збереження зі зміщенням	(Y + q) ← Rr	-	6	2
22	ST	Z, Rr	Непряме збереження	(Z) ← Rr	-	2	2
23	ST	Z+, Rr	Непряме збереження з пост/інкрементом	(Z) ← Rr, Z ← Z + 1	-	2	2
24	ST	-Z, Rr	Непряме збереження з перед/декрементом	Z ← Z - 1, (Z) ← Rr	-	2	2
25	STD	Z+q, Rr	Непряме збереження зі зміщенням	(Z + q) ← Rr	-	6	2
26	STS	k, Rr	Пряме збереження	(k) ← Rr	-	14	2
27	LPM*		Завантаження з програмної пам’яті	R0 ← (Z)	-	1	3
28	IN	Rd, P	Читання порту	Rd ← P	-	8	1
29	OUT	P, Rr	Запис у порт	P ← Rr	-	8	1
30	PUSH	Rr	Занесення регістра в стек	STACK ← Rr; SP← SP - 1	-	2	2
31	POP	Rd	Витягнення регістра зі стека	SP← SP + 1, Rd ← STACK	-	2	2
КОМАНДИ РОБОТИ З БІТАМИ							
1	SBR	Rd*, K	Встановити біт (біти) у регістрі	Rd* ← Rd* v K	Z,N,V,S	5	1
2	CBR	Rd* ,K	Скинути біт (біти) у регістрі	Rd* ← Rd* • ($FF - K)	Z,N,V,S	5	1
3	SBI	P*,b	Встановити біт у РВВ	I/O(P*,b) ← 1	-	9	2
4	CBI	P*,b	Скинути біт у РВВ	I/O(P*,b) ← 0	-	9	2
5	BSET	S	Встановити вказаний розряд регістра SREG	SREG(s) ← 1	SREG(s)	10	1
6	BCLR	S	Скинути заданий розряд регістра SREG	SREG(s) ← 0	SREG(s)	10	1
7	BLD	Rd, b	Завантажити біт з T у регістр	Rd(b) ← T	-	7	1
8	BST	Rr, b	Зберегти біт з регістра в T	T ← Rr(b)	T	7	1
9	SEC		Встановити прапорець перенесення	C ← 1	C	1	1
10	CLC		Скинути прапорець перенесення	C ← 0	C	1	1
11	SEN		Встановити прапорець від’ємного числа	N ← 1	N	1	1
12	CLN		Скинути прапорець від‘ємного числа	N ← 0	N	1	1

Закінчення табл. 3.2

№	Мнемонічне позначення	Операнди	Опис	Операція	Прапорці	Тип	Кільк. тактів
13	SEZ		Встановити прапорець нуля	Z←1	Z	1	1
14	CLZ		Скинути прапорець нуля	Z←0	Z	1	1
15	SEI		Встановити прапорець переривань	I←1	I	1	1
16	CLI		Скинути прапорець переривань	I←0	I	1	1
17	SES		Встановити прапорець знака	S←1	S	1	1
18	CLS		Скинути прапорець знака	S←0	S	1	1
19	SEV		Встановити прапорець переповнення	V←1	V	1	1
20	CLV		Скинути прапорець переповнення	V←0	V	1	1
21	SET		Встановити прапорець T	T←1	T	1	1
22	CLT		Скинути прапорець T	T←0	T	1	1
23	SEH		Встановити прапорець половинного перенесення	H←1	H	1	1
24	CLH		Очистити прапорець половинного перенесення	H←0	H	1	1
КОМАНДИ КЕРУВАННЯ МІКРОКОНТРОЛЕРОМ							
1	NOP		Пуста операція	__	-	1	1
2	SLEEP		Переведення у режим сну	__	-	1	3
3	WDR		Скидання вартового таймера	__	-	1	1

Усі команди цієї групи виконуються за один машинний цикл, за винятком випадків, коли в результаті перевірки відбувається пропуск наступної команди. У цьому разі команда виконується за два або три машинних цикли залежно від довжини команди, що пропускається.

Команди пересилання даних

Команди цієї групи призначено для пересилання вмісту комірок, що лежать в просторі адрес статичної пам’яті даних – РЗП, РВВ та СОЗП. Поділ простору адрес на три частини обумовлює різноманітність команд даної групи. Пересилання даних може виконуватись в наступних напрямках:

– РЗП <=> РЗП;
– РЗП <=> РВВ;
– РЗП <=> статична пам’ять даних.

До даної групи також віднесяться команди PUSH і POP, які дозволяють зберігати у стеку і відновлювати зі стека вміст РЗП.

На виконання команд пересилання потрібно від одного до трьох машинних циклів в залежності від типу команди.

Команди передачі керування

У цю групу входять команди переходу, виклику підпрограм, повернення з них і команди типу «перевірка/пропуск», що пропускають наступну за ними команду при виконанні деякої умови. Також до цієї групи належать команди порівняння, що формують прапорці регістра SREG, які призначено переважно для роботи разом з командами умовного переходу. У командах цього типу виконується перевірка умови, результат якої впливає на виконання наступної команди. Якщо умова істинна, наступна команда ігнорується. Наприклад, команда SBRS Rd, b перевіряє розряд b регістра Rd і ігнорує (пропускає) наступну команду, якщо цей розряд дорівнює одиниці. Перехід до наступної інструкції виконується збільшенням лічильника команд на одиницю, а пропуск команди викликає завантаження нового значення в лічильник команд. Отже, коли умова, що перевіряється, істинна, у конвеєрі виникає затримка. Тривалість затримки залежить від довжини команди, що пропускається, і становить від одного до двох машинних циклів.

Команди передачі керування порушують нормальне (лінійне) виконання основної програми. Щоразу, коли виконується команда з цієї групи (окрім команд порівняння), нормальне функціонування конвеєра порушується. Перед завантаженням у конвеєр нової адреси він зупиняється й очищується послідовність виконуваних команд. Реініціалізація конвеєра призводить до того, що такі команди виконуються протягом декількох машинних циклів [1; 7].

У системі команд мікроконтролерів сімейства, є команди як безумовного, так і умовного переходів. Команди непрямого – JMP і відносного – RJMP безумовного переходу є найпростішими в цій групі. Їх функція полягає тільки у записі нової адреси в лічильник команд.

При виконанні команди RJMP змінюєтьсь вміст лічильника команд шляхом додавання до нього або віднімання з нього деякого значення, що є операндом команди. Ця команда має обмеження за областю дії. Через те, що операнд є 12-розрядним числом зі знаком, максимальна величина переходу відносно адреси наступної команди становить від -2048 до +2047 слів (приблизно ±4 Кбайт).

У програмах в якості операндів цієї команди замість констант використовуються мітки. Компілятор (асемблер) віднімає від адреси мітки адресу наступної команди і підставляє отримане значення у відповідне місце машинного коду команди. Нижче наведено приклад, який ілюструє сказане:

cpi r16, $42*;* порівння регістра r16 з числом $42;
brne error; перехід на мітку error, якщо r16 ≠ $42;

```
        rjmp ok ; безумовний перехід на мітку ok, якщо r16 = $42;
error:
        . . .
ok:     nop; місце переходу за командою RJMP.
```

Оскільки команда відносного переходу змінює вміст лічильника команд, вона виконується за 2 машинних цикли.

У результаті виконання команди непрямого переходу IJMP програма продовжує виконуватися з адреси, що міститься в індексному регістрі Z. Таким чином, дія команди зводиться до завантаження вмісту індексного регістра в лічильник команд.

На відміну від команди відносного переходу ця команда має менше обмежень за областю дії. Насправді, оскільки індексний регістр Z – 16-розрядний, максимально можлива величина переходу становить: 64 Кслів (128 Кбайт). Як і команда відносного переходу, команда непрямого переходу виконується за 2 машинних цикли.

Команди умовного переходу

У цих командах виконується перевірка умови, результат якої впливає на стан лічильника команд. Якщо умова істинна, відбувається перехід за вказаною адресою, якщо ж умова хибна, виконується наступна команда.

Команди умовного переходу мають обмеження за областю дії. Нове значення лічильника команд обчислюється додаванням до нього або відніманням з нього деякого зміщення. Оскільки під значення зміщення в слові команди виділяється лише 7 біт, максимальна величина переходу відносно адреси наступної команди становить від -64 до +63 слів.

Оскільки перехід за вказаною адресою здійснюється завантаженням нового значення в лічильник команд, то у випадку істинності умови, що перевіряється, у конвеєрі виникає затримка тривалістю в один машинний цикл.

Усі команди умовного переходу можна розділити на дві підгрупи. Перша підгрупа – команди умовного переходу загального призначення. У цю підгрупу входять дві команди BRBS s, k і BRBC s, k, в яких явно вказується номер прапорця регістра SREG, який перевіряється. Відповідно, перехід здійснюється при SREG.s = 0 (BRBC) або SREG.s = 1 (BRBS). Іншу підгрупу складають 18 команд, кожна з яких виконує перехід за якою-небудь конкретною умовою: «дорівнює», «більше або дорівнює», «було перенесення» і т. ін. Одні з цих команд використовуються після порівняння беззнакових чисел, інші – після порівняння чисел зі знаком. Можливі умови, що перевіряються, а також відповідні їм команди умовного переходу наведено в табл. 3.3.

Таблиця 3.3. **Зведена таблиця команд умовного переходу**

Перевірка	Логіч. умова	Команда	Зворотна перевірка	Логіч. умова	Команда	Тип даних
Rd>Rr	(N⊕V)=0	BRLT*	Rd≤Rr	(N⊕V)=1	BRGE*	Зі знаком
Rd≥Rr	(N⊕V)=0	BRGE	Rd<Rr	(N⊕V)=1	BRLT	Зі знаком
Rd=Rr	Z=1	BREQ	Rd≠Rr	Z=0	BRNE	Зі знаком
Rd≤Rr	(N⊕V)=1	BRGE*	Rd>Rr	(N⊕V)=0	BRLT*	Зі знаком
Rd<Rr	(N⊕V)=1	BRLT	Rd≥Rr	(N⊕V)=0	BRGE	Зі знаком
Rd>Rr	C=0	BRLO*	Rd≤Rr	C=1	BRSH*	Без знака
Rd≥Rr	C=0	BRSH/BRCC	Rd<Rr	C=1	BRLO/BRCS	Без знака
Rd=Rr	Z=1	BREQ	Rd≠Rr	Z=0	BRNE	Без знака
Rd≤Rr	C=1	BRSH*	Rd>Rr	C=0	BRLO*	Без знака
Rd<Rr	C=1	BRLO/BRCS	Rd≥Rr	C=0	BRSH/BRCC	Без знака
«Перенесення»	C=1	BRCS	«Немає перенесення»	C=0	BRCC	—
«Менше за нуль»	N=1	BRMI	«Більше за нуль»	N=0	BRPL	—
«Переповнення»	V=1	BRVS	«Немає переповнення»	V=0	BRVC	—
«Нуль»	Z=1	BREQ	«Не нуль»	Z=0	BRNE	—
«Половинне перенесення»	H=1	BRHS	«Немає половинного перенесення»	H=0	BRHC	—

* Оскільки у мікроконтролері відсутні команди переходів за умовами: більше та менше, або дорівнює, то для переходу за цими умовами операнди попередньої команди порівняння повинні бути записані в зворотному порядку, тобто замість CP Rd, Rr –> CP Rr, Rd.

Команди, які наведено в табл. 3.3, є тільки еквівалентними мнемонічними позначеннями команд BRBS s, k та BRBC s, k з визначеними значеннями операнда – s. Наприклад, команда BREQ k має, такий же код операції, що і команда BRBS 1, k, а команда BRGE k – такий же, що і BRBC 4 , k. За останньою командою відбувається перехід за значенням k, якщо прапорець S = 1. Значення S = (N+V), де N – значення прапорця від'ємного значення, а V – прапорця переповнення додаткового коду.

Команди виклику підпрограм

Для виклику підпрограм є дві команди: команда відносного виклику – RCALL і команда непрямого виклику – ICALL.

Враховуючі деякі відмінності, які описано нижче, команда RCALL працює так само, як і команда відносного безумовного переходу RJMP.

Команда RCALL зберігає у стеку значення лічильника команд (адресу наступної команди). Потім вміст лічильника команд збільшується або зменшується на деяке значення, що є операндом команди. Оскільки останній є

12-розрядним числом зі знаком, максимальна величина переходу відносно адреси наступної команди становить від -2048 до +2047 слів (приблизно ±4 Кбайт).

У програмах в якості операндів команди RCALL, як і у випадку команди RJMP, використовуються мітки. Асемблер сам обчислює величину зміщення відносно адреси наступної команди шляхом віднімання від адреси мітки адреси наступної команди і підставляє це значення в машинний код команди.

Команда відносного виклику підпрограм виконується за 3 машинних цикли, два з яких витрачаються на збереження у стеку двох байт лічильника команд (адреси наступної команди).

Враховуючі деякі відмінності, які описано нижче, команда ICALL працює так само, як і команда непрямого безумовного переходу IJMP.

Команда ICALL зберігає у стеку значення лічильника команд (адресу наступної команди). Потім у лічильник команд завантажується вміст індексного регістра. Оскільки індексний регістр – 16-розрядний, максимально можлива величина переходу становить 64 Кслів (128 Кбайт). Як і команда RCALL, команда непрямого виклику підпрограм виконується за 3 машинних цикли.

Підпрограма повинна закінчуватися командою повернення RET, як показано в наступному прикладі:

```
rcall   sp_test    ;  виклик підпрограми  sp_test
...                ;  текст основної програми
sp_test            ;  мітка підпрограми
push    r2         ;  збереження r2 у стеку
...                ;  виконання підпрограми
pop     r2         ;  відновити r2 зі стека
ret                ;  повернення з підпрограми
```

У наведеному вище прикладі команда ret замінює адресу, що міститься в лічильнику команд, адресою команди, наступної за командою rcall.

Команди повернення з підпрограм

В кінці кожної підпрограми обов’язково повинна міститися команда повернення з неї. У системі команд AVR-мікроконтролерів є дві таких команди. Для повернення з підпрограми, що викликається командами RCALL і ICALL, використовується команда RET. Для повернення з підпрограми обробки переривання використовується команда RETI.

Обидві команди відновлюють зі стека вміст лічильника команд, який зберігається там перед переходом до підпрограми. Команда повернення з підпрограми RETI додатково встановлює в одиницю прапорець глобального

дозволу переривань I регістра SREG, що скидається апаратно при виникненні переривання.

На виконання кожної з команд повернення з підпрограми потрібно 4 машинних цикли.

Команди керування мікроконтролером

У цю групу входять 3 команди:

– NOP – пуста операція;

– SLEEP – переведення мікроконтролера в режим зниженого енергоспоживання;

– WDR – скидання вартового таймера.

Команди NOP і WDR виконуються за один машинний цикл, а команда SLEEP – за три машинних цикли.

Вище скорочено було розглянуто 118 базових команд. Реальна кількість команд більша, тому що за деякими номерами наведено опис команд, які виконуються за схожими алгоритмами, але відрізняються способами адресації операндів. Детальний опис базових команд наведено у [1; 7].

3.14 Нові команди AVR-мікроконтролерів

Архітектура AVR-мікроконтролерів постійно оновлюється. Це оновлення стосується як структури мікроконтролера, так і його системи команд. Кожне сімейство, послідовно успадковує набір команд попереднього сімейства. Але є деякі виключення. Нижче розглядаються нові команди, які на даний час з'явилися у нових AVR-мікроконтролерах: Mega та XMega (табл. 3.4) [1; 18].

JMP

Код операції:

15														1	0
1	0	0	1	0	1	0	k	k	k	k	k	1	1	0	k
31														17	16
k	k	k	k	k	k	k	k	k	k	k	k	k	k	k	k

Алгоритм виконання:

$$PC \leftarrow k.$$

Опис: для контролерів з 22-розрядним програмним лічильником (PC), максимальна ємність пам'яті програм дорівнює 2^{22} = 4 Мслів (8 Мбайт), наприклад у контролері AT90SC6464C-USB, а для контролерів з 16-розрядним PC, відповідно 2^{16} = 64 Кслів (128 Кбайт).

Довжина команди: 2 слова (4 байти).

Таблиця 3.4. **Нові команди мікроконтролерів AVR**

Команда	**Операція**	**Опис**	**Дія**	**Прапорці**	**Такти**
Classic Core AVR (до 128К програмного простору)					
JMP	k	Безумовний прямий перехід у пам'яті програм ємністю 64 К/4 Мслів	PC ← k (16/22 біти)	-	3/4
CALL	k	Безумовний прямий виклик підпрограми із пам'яті програм ємністю 64 К/4 Мслів	STACK ← PC + 2 SP ← SP - 2/SP - 3 PC ← k (16/22 біти)	-	4/5
Enhanced Core AVR (до 8К програмного простору)					
MUL	Rd, Rr	Множення беззнакових чисел	R1:R0 ← RdxRr	Z,C	2
MULS	Rd, Rr	Множення знакових чисел	R1:R0 ← RdxRr	Z,C	2
MULSU	Rd, Rr	Множення знакового числа на беззнакове	R1:R0 ← RdxRr	Z,C	2
FMUL	Rd, Rr	Множення дробових беззнакових чисел	R1:R0←RdxRr	Z,C	2
FMULS	Rd, Rr	Множення дробових знакових чисел	R1:R0 ← RdxRr	Z,C	2
FMULSU	Rd, Rr	Множення дробового знакового числа на беззнакове	R1:R0 ← RdxRr	Z,C	2
MOVW	Rd, Rr	Копіювання слова	Rd+1:Rd ← Rr+1:Rr	-	1
LPM Rd, Z	Rd	Завантаження регістра Rd із пам'яті програм з адресою у індексному регістрі Z	Rd ←(Z)	-	3

Продовження табл. 3.4

Команда	Операція	Опис	Дія	Прапорці	Такти
LPM Rd, Z+	Rd	Завантаження регістра Rd із пам'яті програм з адресою у індексному регістрі Z та наступним інкрементом вмісту Z	Rd ← (Z) Z ← Z + 1	-	3
SPM		Використовується для самопрограмування мікроконтролера, який має відповідний блок	Дивись опис команди, наведений нижче	-	Виконується із пам'яті завантажувача
Enhanced Core AVR (до 128К програмного простору)					
BREAK		Зупинка процесора після виконання команди	PC ← PC + 1 зупинка виконання програми	-	1
Enhanced Core AVR (до 4М програмного простору)					
EIJMP		Безумовний непрямий перехід у пам'яті програм ємністю 4 Мслів	PC← (EIND:Z)	-	2
EICALL		Безумовний непрямий виклик підпрограми із пам'яті програм ємністю 4 Мслів	STACK← PC + 1 SP ← SP - 3 PC ← (EIND:Z)	-	4
ELPM		Завантаження регістра R0 із розширеної пам'яті програм, з адресою у регістрах RAMPZ та Z	R0 ← (RAMPZ: Z)	-	3
ELPM Rd, Z	Rd	Завантаження регістра Rd із розширеної пам'яті програм, з адресою у регістрах RAMPZ та Z	Rd ← (RAMPZ:Z)	-	3

Закінчення табл. 3.4.

Команда	Операція	Опис	Дія	Прапорці	Такти
ELPM Rd, Z-	Rd	Завантаження регістра Rd із розширеної пам'яті програм, з адресою у регістрах RAMPZ та Z, та наступним інкрементом вмісту RAMPZ:Z	Rd ← (RAMPZ:Z) RAMPZ:Z ← RAMPZ:Z+1	-	3
XMEGA CoreAVR					
DES	K	Шифрування даних	якщо H=0: R15:R0←ENCRYPT (R15:R0, K) якщо H=1: R15:R0←DECRYPT (R15:R0, K)	-	1/2
LAC Z, Rd	Rd	Запис Rd у Temp, запис (Z) у Rd, очищення бітів (Z) за маскою, яку вказано в Temp	Temp ← Rd; Rd ← (Z); (Z) ← ($FF – Temp)*(Z); PC ← PC + 1	-	1
LAS Z, Rd	Rd	Запис Rd у Temp, запис (Z) у Rd, встановлення бітів (Z) за маскою, яку вказано в Temp	Temp ← Rd; Rd ← (Z); (Z) ← Temp v (Z); PC ← PC + 1	-	1
LAT Z, Rd	Rd	Запис Rd у Temp, запис (Z) у Rd, підсумовування за модулем 2 бітів (Z) з маскою, яку вказано в Temp	Temp ← Rd; Rd ← (Z); (Z) ← Temp xor (Z); PC ← PC + 1	-	1
XCH Z, Rd	Rd	Обмін даними між Rd та (Z)	Temp ← Rd; Rd ← (Z); (Z) ← Temp; PC ←PC+1	-	1

CALL

Код операції:

15														1	0
1	0	0	1	0	1	0	k	k	k	k	k	1	1	1	k
31															16
k	k	k	k	k	k	k	k	k	k	k	k	k	k	k	k

Алгоритм виконання:

STACK ← PC + 2
SP ← SP - 2
PC ← k, якщо k = 16 біт

Безумовний прямий виклик підпрограми із пам'яті програм ємністю 2^{16} = 64К слів (128 Кбайт)

STACK ← PC + 2
SP ← SP - 3
PC ← k, якщо k = 22 біт

Безумовний прямий виклик підпрограми із пам'яті програм ємністю 2^{22} = 4М слів (8 Мбайт)

Довжина команди: 2 слова (4 байти).

MUL

Код операції:

15							8	7							0
1	0	0	1	1	1	r	d	d	d	d	d	r	r	r	r

Алгоритм виконання:

R1:R0 ← RdxRr
PC ← PC + 1.

Операнди: 0≤d≤31; 0≤r≤31 – беззнакові величини.

Довжина команди: 1 слово (2 байти).

MULS

Код операції:

15							8	7							0
0	0	0	0	0	0	1	0	d	d	d	d	r	r	r	r

Алгоритм виконання:

R1:R0 ← RdxRr
PC ← PC + 1.

Операнди: 16≤d≤31; 16≤r≤31 – знакові величини.

Довжина команди: 1 слово (2 байти).

MULSU

Код операції:

15							8	7							0
1	0	0	1	0	0	1	1	0	d	d	d	0	r	r	r

Алгоритм виконання:

R1:R0 ← RdxRr

PC ← PC + 1.

Операнди: 16≤d≤23; 16≤r≤23 (у Rr – беззнакове число, Rd – знакове число)

Довжина команди: 1 слово (2 байти).

FMUL

Код операції:

15							8	7							0
0	0	0	0	0	0	1	1	0	d	d	d	0	r	r	r

Алгоритм виконання:

R1:R0 ← RdxRr

PC ← PC + 1.

Опис: числа подаються у вигляді N.Q, де N – значення числа до коми у двійковому вигляді, Q – значення числа після коми у двійковому вигляді. Таким чином операнди подаються в вигляді (N1.Q1) і (N2.Q2). Беззнакові 8-розрядні дробові числа використовують формат, в якому числа можуть лежати у діапазоні: [0, 2]. Біти 6…0 являють собою дробову частину, а 7-й біт – цілу частину (0 або 1), тобто 1.7 формат. Результат множення (формат результату): 2.14 зсувається вліво на один розряд для приведення до формату: 1.15 та записується у пару регістрів R1:R0.

Операнди: 16≤d≤23; 16≤r≤23 – беззнакові дробові величини.

Довжина команди: 1 слово (2 байти).

FMULS

Код операції:

15							8	7							0
0	0	0	0	0	0	1	1	1	d	d	d	0	r	r	r

Алгоритм виконання:

R1:R0 ← RdxRr

PC ← PC + 1.

Опис: числа подаються в вигляді N.Q, де N – значення числа до коми у двійковому вигляді, Q – значення числа після коми у двійковому вигляді. Таким чином операнди подаються в вигляді (N1.Q1) і (N2.Q2). Дробові числа зі знаком подібно до цілих чисел зі знаком використовують формат двійкового доповнення, що дозволяє подавати перші у діапазоні: [-1,1]. Якщо, наприклад, ціле число без знака має двійковий код: 10110010, то його десятковий еквівалент дорівнює: 128+32+16+2 = 178. Відповідно, десятковий еквівалент цього числа як знакового цілого дорівнює: 178-256 = -78. Якщо наведене вище двійкове число подати у форматі беззнакового дробового, то отримаємо:

$$1 + 0{,}25 + 0{,}125 + 0{,}015625 = 1{,}390625.$$

Аналогічно наведеному вище правилу обчислення цілих чисел зі знаком можна отримати десятковий еквівалент двійкового числа 10110010, якщо вважати що воно відповідає дробовому числу зі знаком: 1,390625 - 2 = -0,609375.

Результат множення (формат результату): 2.14 зсувається вліво на один розряд для приведення до формату: 1.15 та записується у пару регістрів R1:R0.

Операнди: $16 \le d \le 23$; $16 \le r \le 23$ – дробові величини зі знаком.

Довжина команди: 1 слово (2 байти).

FMULSU

Код операції:

15							8	7							0
0	0	0	0	0	0	1	1	1	d	d	d	1	r	r	r

Алгоритм виконання:

$$R1{:}R0 \leftarrow RdxRr$$
$$PC \leftarrow PC + 1.$$

Опис: числа подаються в вигляді N.Q, де N – значення числа до коми у двійковому вигляді, Q – значення числа після коми у двійковому вигляді. Таким чином операнди подаються в вигляді: N1.Q1 і N2.Q2 (дивись опис команди FMULS).

Операнди: $16 \le d \le 23$; $16 \le r \le 23$ (обидві величини дробові, одна з яких, у Rr – беззнакова, друга величина, у Rd – знакова).

Довжина команди: 1 слово (2 байти).

MOVW

Код операції:

15							8	7							0
0	0	0	0	0	0	0	1	d	d	d	d	r	r	r	r

Алгоритм виконання:

$$Rd+1:Rd \leftarrow Rr+1:Rr$$
$$PC \leftarrow PC + 1.$$

Опис: команда копіює вміст однієї пари регістрів загального призначення в іншу пару регістрів загального призначення. Пара регістрів джерела (Rr+1:Rr) залишається незмінною. Пара регістрів приймача (Rd+1:Rd) завантажується копією регістрів Rr+1:Rr. Номери регістрів Rr, Rd повинні бути парними.

Операнди: $0 \le d \le 30$; $0 \le r \le 30$.

Довжина команди: 1 слово (2 байти).

LPM Rd, Z

Код операції:

15							8	7							0
1	0	0	1	0	0	0	d	d	d	d	d	0	1	0	0

Алгоритм виконання:

$$Rd \leftarrow ПП\ (Z)$$
$$PC \leftarrow PC + 1.$$

Операнди: $0 \le d \le 31$, ПП (Z) – комірка пам’яті програм, адреса якої міститься в індексному регістрі Z.

Довжина команди: 1 слово (2 байти).

LPM Rd, Z+

Код операції:

15							8	7							0
1	0	0	1	0	0	0	d	d	d	d	d	0	1	0	1

Алгоритм виконання:

$$Rd \leftarrow ПП\ (Z),\ Z \leftarrow Z + 1$$
$$PC \leftarrow PC + 1.$$

Операнди: $0 \le d \le 31$, ПП (Z) – комірка пам’яті програм, адреса якої міститься в індексному регістрі Z.

Довжина команди: 1 слово (2 байти).

SPM

Код операції:

15							8	7							0
1	0	0	1	0	1	0	1	1	1	1	0	1	0	0	0

Опис: ця команда використовується для самопрограмування мікроконтролера, який має відповідний блок. Команда зберігається, і відповідно виконується, у спеціальній області пам'яті, яка називається завантажувачем. Існує п'ять режимів використання команди, які програмуються за допомогою додаткових регістрів керування:

(RAMPZ : Z) ← $FFFF, очищення сторінки пам'яті програм;

(RAMPZ : Z)← R1:R0, запис слова у тимчасовий буфер пам'яті програм;

(RAMPZ : Z) ← R1:R0, запис сторінки у тимчасовий буфер пам'яті програм;

(RAMPZ : Z)← TEMP, запис тимчасового буфера у пам'ять програм;

BLBITS ← R1:R0, встановлення бітів блокування завантаження програм.

Більш детальний опис цієї команди наведено у [1].

BREAK

Код операції:

15							8	7							0
1	0	0	1	0	1	0	1	1	0	0	1	1	0	0	0

Алгоритм виконання:

PC ← PC+1.

Опис: ця інструкція використовується налагоджувачем, який вбудовано у мікроконтролер, та зазвичай в робочих програмах не використовується. Після виконання команди BREAK процесор зупиняється, що дає можливість налагоджувачеві працювати із внутрішніми ресурсами.

Операнди: немає.

Довжина команди: 1 слово (2 байти).

EIJMP

Код операції:

15							8	7							0
1	0	0	1	0	1	0	0	0	0	0	1	1	0	0	1

Алгоритм виконання:

PC(15:0) ← Z(15:0)

PC(21:16) ← EIND (6 біт),

де EIND – ім'я додаткового регістра керування.

Довжина команди: 1 слово (2 байти).

EICALL

Код операції:

15							8	7							0
1	0	0	1	0	1	0	1	0	0	0	1	1	0	0	1

Алгоритм виконання:

STACK ← PC + 1

SP ← SP - 3

PC(15:0) ← Z(15:0)

PC(21:16) ← EIND (6 біт),

де EIND – ім’я додаткового регістра керування.

Довжина команди: 1 слово (2 байти).

ELPM

Код операції:

15							8	7							0
1	0	0	1	0	1	0	1	1	1	0	1	1	0	0	0

Алгоритм виконання:

R0←ПП (RAMPZ:Z),

де RAMPZ – ім’я додаткового регістра керування;

PC ← PC + 1.

Операнди: R0 – регістр загального призначення; (RAMPZ:Z) – комірка пам’яті програм, адреса якої міститься у двох регістрах: RAMPZ (6 біт) та Z – 16 біт. Довжина команди: 1 слово (2 байти).

ELPM Rd, Z

Код операції:

15							8	7							0
1	0	0	1	0	0	0	d	d	d	d	d	0	1	1	0

Алгоритм виконання:

Rd ← ПП (RAMPZ:Z),

де RAMPZ – ім’я додаткового регістра керування;

PC ← PC + 1.

Операнди: Rd ($0 \le d \le 31$) – регістр загального призначення;

ПП (RAMPZ:Z) – комірка пам’яті програм, адреса якої міститься у двох регістрах: RAMPZ (6 біт) та Z – 16 біт.

Довжина команди: 1 слово (2 байти).

ELPM Rd, Z+

Код операції:

15							8	7							0
1	0	0	1	0	0	0	d	d	d	d	d	0	1	1	1

Алгоритм виконання:

Rd ← ПП (RAMPZ:Z), RAMPZ:Z ← RAMPZ:Z+1,

де RAMPZ – ім’я додаткового регістра керування;

PC ← PC + 1.

Операнди: Rd (0≤d≤31) – регістр загального призначення;

ПП (RAMPZ:Z) – комірка пам’яті програм, адреса якої міститься у двох регістрах: RAMPZ (6 біт) та Z – 16 біт.

Довжина команди: 1 слово (2 байти).

DES

Алгоритм виконання:

R15:R0 ← ENCRYPT(R15:R0, K), якщо прапорець H=0,

R15:R0 ← DECRYPT(R15:R0, K), якщо прапорець H=1.

Опис: команда DES використовується в сучасних мікроконтролерах сімейства XMega для шифрування інформації. Більш детальну інформацію про команду та метод кодування, наведено у додатковій літературі [1; 18].

LAC

Код операції:

15							8	7							0
1	0	0	1	0	0	1	d	d	d	d	d	0	1	1	0

Алгоритм виконання:

Temp ← Rd;

Rd ← (Z);

(Z) ← ($FF – Temp)*(Z);

PC ← PC + 1.

Опис: команда LAC записує Rd у тимчасовий регістр Temp, пересилає (Z) в Rd, інвертоване значення Temp множить на (Z) та результат записує в (Z). Таким чином до виконання команди в Rd знаходиться маска, за якою необхідно скинути у нуль відповідні біти (Z), а після виконання команди результат записується у (Z) з обнуленими бітами на тих місцях, де у масці знаходилась одиниця. При виконанні команди початкове значення (Z) записується у Rd.

Операнди: 0≤d≤ 31.

Довжина команди: 1 слово (2 байти).

LAS

Код операції:

15							8	7							0
1	0	0	1	0	0	1	d	d	d	d	d	0	1	0	1

Алгоритм виконання:

Temp ← Rd;
Rd ← (Z);
(Z) ← Temp v (Z);
PC ← PC + 1.

Опис: команда LAS записує Rd у тимчасовий регістр Temp, пересилає (Z) в Rd, логічно підсумовує Temp і (Z) та результат записує в (Z). Таким чином до виконання команди в Rd знаходиться маска, за якою необхідно встановити в одиницю відповідні біти (Z), а після виконання команди результат записується у (Z) з встановленими бітами на тих місцях, де у масці знаходилась одиниця. При виконання команди початкове значення (Z) записується у Rd.

Операнди: $0 \le d \le 31$.

Довжина команди: 1 слово (2 байти).

LAT

Код операції:

15							8	7							0
1	0	0	1	0	0	1	d	d	d	d	d	0	1	1	1

Алгоритм виконання:

Temp ← Rd;
Rd ← (Z);
(Z) ← Temp xor (Z);
PC ← PC + 1.

Опис: команда LAT записує Rd у тимчасовий регістр Temp, пересилає (Z) в Rd, виконує логічне «Виключне АБО» (XOR) між Temp і (Z) та результат записує в (Z). Таким чином до виконання команди в Rd знаходиться маска, за якою необхідно проінвертувати відповідні біти (Z), а після виконання команди результат записується у (Z) з проінвертованими бітами на тих місцях, де у масці знаходилась одиниця. При виконання команди початкове значення (Z) записується у Rd.

Операнди: $0 \le d \le 31$.

Довжина команди: 1 слово (2 байти).

XCH

Код операції:

15							8	7							0
1	0	0	1	0	0	0	d	d	d	d	d	0	1	0	0

Алгоритм виконання:

Temp ← Rd;
Rd ← (Z);
(Z) ← Temp;
PC ←PC+1.

Опис: команда XCH міняє місцями значення, що знаходиться за адресою, вказаною в Z, зі значенням Rd.

Операнди: $0 \leq d \leq 31$.

Довжина команди: 1 слово (2 байти).

Контрольні запитання та завдання

1. Назвіть основні етапи створення керуючої програми для МК.
2. Назвіть складові програмної моделі типового AVR-мікроконтролера.
3. Скільки РЗП має типовий AVR-мікроконтролер?
4. Як реалізовано покажчик стека?
5. Назвіть призначення лічильника команд.
6. Надайте характеристику командам типу «перевірка/пропуск».
7. Назвіть та надайте характеристику командам виклику та повернення з підпрограм.
8. Назвіть способи адресації операндів.
9. Які РЗП використовуються для непрямої адресації?
10. Дайте визначення мнемоніки команди та мнемокоду.
11. Дайте визначення коду операції команди та машинного коду команди.
12. Скільки типів команд має більшість мікроконтролерів сімейства AVR? Наведіть коротку характеристику цих типів.
13. Наведіть типи (формати) даних мікроконтролерів AVR.
14. Назвіть довжину команд у байтах та опишіть їх розміщення у пам’яті програм.
15. Опишіть особливості RISC-архітектури мікроконтролерів AVR.
16. Назвіть основні групи команд з базового набору мікроконтролерів AVR. Дайте коротку характеристику кожній з груп.
17. Назвіть та дайте характеристику новим командам.
18. Що спільного та які вімінності при виконанні команд JMP та CALL?
19. Опишіть призначення та відмінності команд RET та RETI.
20. Поясніть особливості виконання команди RJMP.
21. Як компілятор використовує мітки в командах умовних переходів при формування машинних кодів команд?
22. Назвіть команди, які можуть використовуватись для виконання «булевих» операцій. Наведіть приклади їх використання.
23. Чим відрізняється алгоритми виконання команд арифметичних та логічних зсувів? Наведіть приклади їх використання.
24. Який діапазон десяткових чисел відображає восьмирозрядне число зі знаком?
25. Чим відрізняються мнемоніка та мнемокод команд?
26. Для чого використовується регістр PC?
27. Опишіть роботу конвеєра при виконанні програми.
28. Для чого використовуються регістри-покажчики X,Y і Z?

4. САМОПРОГРАМУВАННЯ AVR-МІКРОКОНТРОЛЕРІВ

4.1. Загальні відомості

Сучасні AVR-мікроконтролери, наприклад, всі мікроконтролери сімейства Mega та XMega мають можливість самопрограмування, тобто самостійної зміни вмісту своєї пам'яті програм під час виконання програми. Ця особливість дозволяє створювати на їхній основі дуже гнучкі системи, алгоритм роботи яких буде змінюватися самим мікроконтролером залежно від відповідних внутрішніх умов або зовнішніх подій.

Для підтримки самопрограмування всю область пам'яті програм логічно розділено на дві секції – секцію прикладної програми (Application Section) і секцію завантажувача (Boot Loader Section) (рис. 4.1). Програмування пам'яті програм здійснюється програмою-завантажувачем, яку розташовано в однойменній секції.

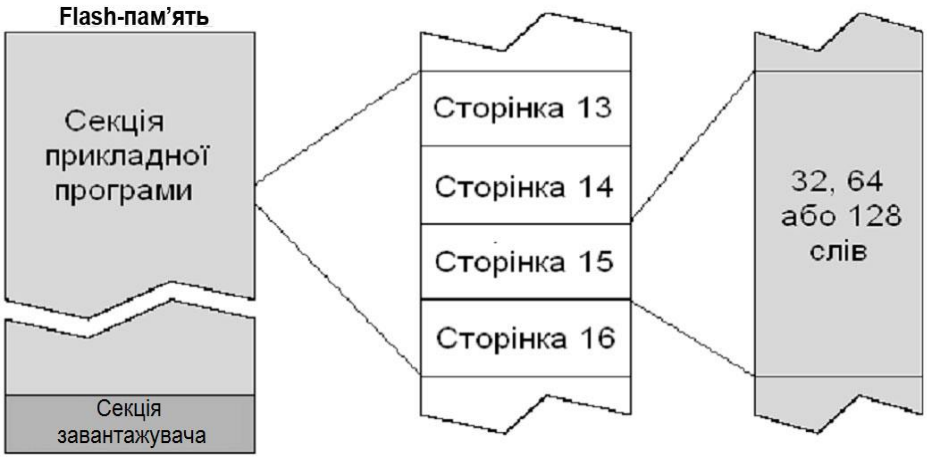


Рис. 4.1. Розділення пам'яті програм на дві секції

Розмір секції завантажувача, її розташування в пам`яті програм та розмір секції прикладної програми у більшості мікроконтролерів визначається двома конфігураційними комірками BOOTSZ1:BOOTSZ0 [1; 7].

Для завантаження нового вмісту пам'яті програм, а також для вивантаження старого вмісту програма-завантажувач може використовувати будь-який інтерфейс передачі даних – USART/UART, SPI, TWI, наявний у складі конкретного мікроконтролера. Завантажувач може змінювати вміст обох секцій, що дозволяє модифікувати власний код та видаляти себе з пам'яті, якщо потреба в ньому відпаде. Рівень доступу (читання/запис) до кожної із секцій визначається користувачем в комірках захисту (Lock Bits) – BLB02:BLB01 та BLB12:BLB11. Комірки BLB02:BLB01 визначають рівень доступу із секції завантажувача до коду, який розташовано в секції прикладної програми, а комірки BLB12:BLB11 навпаки, визначають рівень доступу із секції прикладної програми до коду,

розташованого в секції завантажувача. Всі перераховані комірки захисту згруповано в одному байті.

Можливі режими захисту, що відповідають різним станам цих комірок, та їх розташування наведено у [1; 7].

У незапрограмованому стані у всіх комірках захисту записано одиницю, після програмування – нуль. Очищення комірок виконується командою «очищення кристала». При цьому знищується також вміст FLASH- та EEPROM-пам’яті.

Програму-завантажувача можна викликати з основної програми командами CALL/JMP, або перемістити вектор скидання на початок секції завантажувача.

У другому випадку запуск програми-завантажувача буде здійснюватися автоматично після кожного скидання мікроконтролера. Положення вектора скидання визначається станом конфігураційної комірки BOOTRST. Якщо в ній зберігається одиниця, вектор скидання розташовується на початку пам’яті програм за адресою $0000. При запрограмованій комірці, коли в ній зберігається нуль, вектор скидання розташовується на початку секції завантажувача.

4.2. Області RWW та NRWW

Пам’ять програм ділять на дві області фіксованого розміру, які називають «читання під час запису» – Read-While-Write, (RWW) та «немає читання під час запису» – No Read-While-Write (NRWW). У табл. 4.1 наведено розміри цих областей для деяких AVR-мікроконтролерів сімейства Mega.

Таблиця 4.1. **Розміри областей RWW та NRWW**

Модель	**Область RWW**		**Область NRWW**	
	Кількість сторінок	**Адреса**	**Кількість сторінок**	**Адреса**
ATmega8515x/8535x, ATmega8x, ATmega88x	96	$000... $BFF	32	$C00... $FFF

Закінчення табл. 4.1.

Модель	Область RWW		Область NRWW	
	Кількість сторінок	**Адреса**	**Кількість сторінок**	**Адреса**
ATmega16x, 162x, ATmega164x, ATmega165x, ATmega168x	112	$0000... $1BFF	16	$1C00... $1FFF
ATmega32x, ATmega324x, ATmega325x/3250x	224	$0000... $37FF	32	$3800... $3FFF
ATmega64x, ATmega640x, ATmega645x/6450x	224	$0000... $6FFF	32	$7000... $7FFF
ATmega128x, ATmega1280x/1281x	480	$0000... $EFFF	32	$F000... $FFFF
ATmega2560x/2561x	992	$00000... $1EFFF	32	$1F000... $1FFFF

Відмінністю між цими областями є різне поводження центрального процесора при зміні розташованих у них даних:

- під час виконання операції очищення або запису сторінки пам’яті програм, розташованої в області RWW, процесор може здійснювати читання тільки з області NRWW. Читання області RWW заборонено;
- під час виконання операції очищення або запису сторінки пам’яті програм, розташованої в області NRWW, процесор зупиняється до закінчення цієї операції (рис. 4.2).

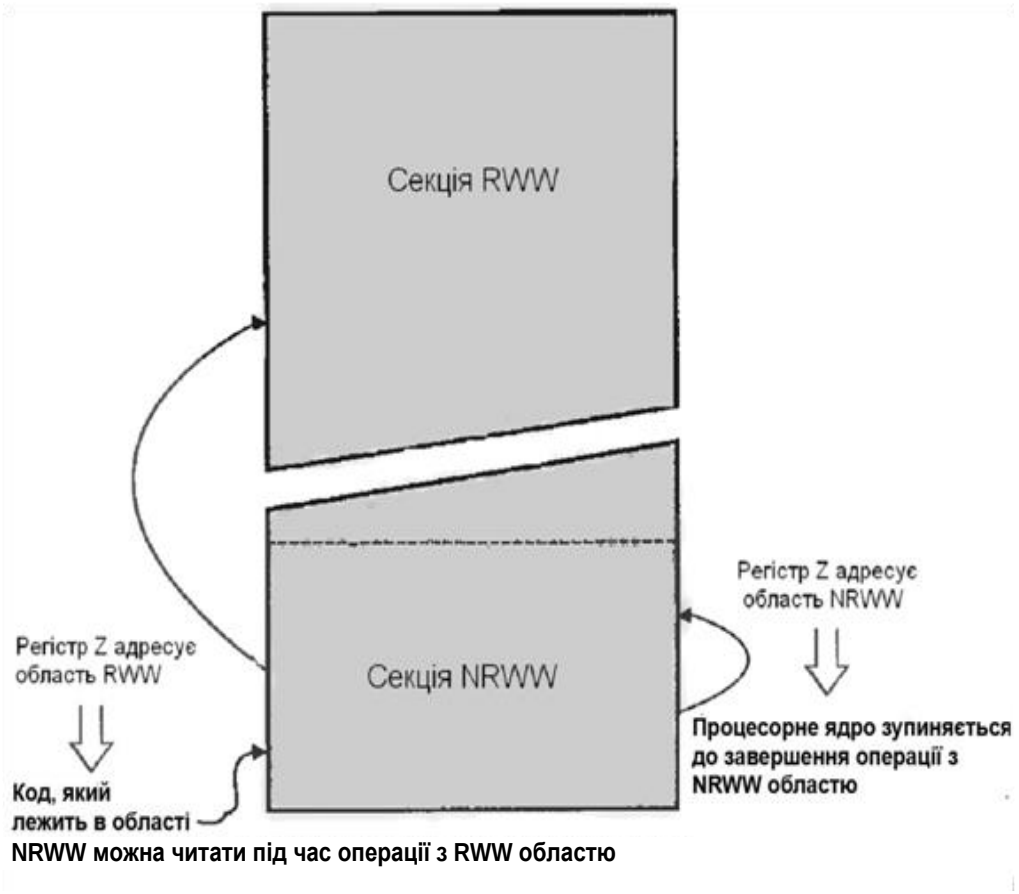


Рис. 4.2. Відмінність між областями RWW й NRWW

Звернення при виконання команд CALL/JMP/LPM або за перериванням до коду, що міститься в області RWW, може призвести до непередбачених наслідків. Для уникнення цього, треба або заборонити переривання, або перенести таблицю векторів переривань у секцію завантажувача, що завжди міститься в області NRWW.

Для визначення дозволу читання з області RWW призначено прапорець RWWSB регістра SPMCR. Якщо прапорець встановлено в одиницю, то область RWW заблоковано для читання. Після закінчення операції програмування, прапорець RWWSB необхідно програмно скинути.

Під час зміни сторінки пам’яті програм, яка міститься в області RWW, можна зчитувати код, який розташовано в області NRWW. А при зміні вмісту області NRWW процесор зупиняється до завершення операції.

У деяких моделях (наприклад, ATmega161x, ATmega163x й ATmega323x) поділу на області RWW та NRWW немає. Однак умовно можна вважати, що секція прикладної програми є областю RWW, секція завантажувача – областю NRWW.

4.3. Прапорець зайнятості секції RWW та його скидання

Під час запису та очищення сторінок RWW-секції апаратно встановлюється прапорець RWWSB, вказуючи на те, що секція недоступна. Після закінчення операцій з RWW-секцією прапорець RWWSB необхідно програмно скинути. Це виконується встановленням бітів RWWSRE та SPMEN у регістрі SPMCR з наступним виконанням команди SPM. Вміст регістра Z та регістрів R1:R0 при цьому ігнорується. Прапорець RWWSB також можна скинути шляхом запуску завантаження буфера сторінок. Окремі частини прикладної програми можуть використовувати прапорець RWWSB для контролю доступності RWW-секції.

Якщо після виконання операції запису або зчитування секція RWW стала доступна без переактивації, то всі адреси в ній зчитуються як 0xFFFF. Це відбудеться після зчитування FLASH-пам’яті за допомогою LPM і виконання виклику або переходу в секцію RWW. Якщо переходити до секції RWW без її переактивації, то виконується команда, яка лежить за адресою 0xFFFF, а всі попередні команди, які мали виконатися, буде пропущено.

4.4. Керування процесом самопрограмування

Керування процесом програмування здійснюється командою SPM при використанні регістра введення/виведення SPMCR (Store Program Memory Control Register) або SPMCSR. У всіх моделях, за винятком ATmega64x/128x, цей регістр розташовано за адресою $37 ($57). У мікроконтролерах ATmega64x/128x цей регістр розташовано у просторі додаткових регістрів введення/виведення за адресою $68. Формат цього регістра для різних моделей сімейства показано на рис. 4.3, а опис його розрядів наведено в табл. 4.2.

Запис значень, відмінних від «10001», «01001», «00101», «00011» й «00001», у молодші п’ять розрядів регістра ігнорується.

Для зміни вмісту пам’яті програм та комірок захисту завантажувача використовується команда SPM (Store Program Memory). Адреса області пам’яті попередньо завантажується в індексний регістр RAMPZ:Z, а дані, при необхідності, – у регістрову пару R1:R0.

Звернемо увагу на те, що під час запису в EEPROM-пам’ять зміна вмісту регістра SPMCR неможлива. Тому, перед тим як записати будь-яке значення в регістр SPMCR, рекомендується дочекатися скидання прапорця EEWE регістра EECR.

SPMCR	7	6	5	4	3	2	1	0	
Читання(R)/ запис(W)	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	ATmega8515x/ 8535x/ 8x/ 16x/ 32x/ 64x/ 128x/ 162x
Початкове значення	R/W 0	R 0	R 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	
SPMCSR	7	6	5	4	3	2	1	0	ATmega164x/ 324x/ 644x/ 165x/ 325x/ 3250x/ 645x/ 6450x/ 1280x/ 1261x/ 2560x/ 2561x
Читання(R)/ запис(W)	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	
Початкове значення	R/W 0	R 0	R 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	
SPMCSR	7	6	5	4	3	2	1	0	ATmega48x/ 88x/ 168x
Читання(R)/ запис(W)	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SELFPRGEN	
Початкове значення	R/W X	R 0	R 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	

Рис. 4.3. Формат регістра SPMCR/SPMCSR

Для адресації пам’яті програм при використанні команди SPM використовується індексний регістр Z, який є об’єднанням двох старших регістрів загального призначення R30 (молодший байт) і R31 (старший байт), а в моделях з об’ємом пам’яті вище 64 Кбайт – ще регістр введення/виведення RAMPZ. Оскільки пам’ять програм у мікроконтролерах сімейства Mega має сторінкову організацію (табл. 4.3), лічильник команд можна умовно розділити на дві частини.

Перша частина (молодші розряди) адресують комірки на сторінці, а друга частина визначає сторінку (рис. 4.4, табл. 4.4).

Після запуску операції програмування вміст регістра Z фіксується і його можна використати для інших цілей. В командах LPM регістр Z може використовуватись для адресації окремих байтів пам’яті програм (ПП). При цьому старші розряди регістра Z, крім нульового, містять адресу потрібного слова ПП, а молодший біт – Z0 адресує молодший (Z0 = 0), або старший (Z0 = 1) байт обраного слова. При самопрограмуванні регістр Z буде використовуватись для адресації окремих слів в ПП, тому молодший біт регістра Z: Z0 = 0, а інші біти адресують слова.

Таблиця 4.2. **Розряди регістра SPMCR**

Розряд	Назва	Опис
7	SPMIE	**Дозвіл переривання SPM**. Якщо в цей розряд записано одиницю і також встановлено в одиницю прапорець I регістра SREG, то дозволяється переривання SPM. Переривання генерується весь час, поки розряд SPMEN регістра скинуто в нуль
6	RWWSB	**Заборона доступу до області RWW**. Цей прапорець показує можливість звернення за адресами, розташованими в області RWW. Якщо прапорець встановлено в одиницю, то доступ до області RWW заборонено. Якщо прапорець скинуто в нуль, то доступ дозволено. Встановлення цього прапорця здійснюється апаратно при виконанні операцій запису або очищення сторінки пам'яті. Прапорець можна скинути або програмно – записом одиниці у розряд RWWSRE по закінченні операції, або апаратно – при запуску операції завантаження буфера сторінки
5	–	Зарезервовано, читається як нуль
4	RWWSRE	**Дозвіл читання області RWW**. Одночасне встановлення в одиницю цього розряду та розряду SPMEN дозволяє доступ до області RWW. Доступ до цієї області здійснюється командою SPM, запущеною протягом чотирьох машинних циклів після встановлення зазначених розрядів. Дозволити доступ до області RWW можна тільки після завершення операції програмування – після скидання прапорця SPMEN
3	BLBSET	**Зміна комірок захисту завантажувача та прикладної програми**. При одночасному встановленні цього розряду та розряду SPMEN команда SPM, запущена протягом чотирьох машинних циклів, здійснить встановлення захисних комірок завантажувача та прикладної програми відповідно до вмісту регістра R0. Скидання розряду BLBSET здійснюється апаратно після встановлення комірок захисту або після закінчення зазначеного часу. За командою LPM, запущеною протягом трьох машинних циклів після встановлення зазначених розрядів, буде здійснено читання або комірок конфігурації, або комірок захисту (залежить від значення розряду Z0 регістра Z)
2	PGWRT	**Запис сторінки.** При одночасному встановленні цього розряду та розряду SPMEN команда SPM, запущена протягом чотирьох машинних циклів, запише сторінку пам'яті програм. Адресу сторінки необхідно завантажити у старші біти регістра-покажчика Z. Скидання розряду PGWRT здійснюється апаратно по закінченню запису сторінки або після закінчення відповідного часу. Під час запису в секцію NRWW центральний процесор зупиняється на час виконання операції*.

Закінчення табл. 4.2.

Розряд	Назва	Опис
1	PGERS	**Очищення сторінки**. При одночасному встановленні цього розряду та розряду SPMEN команда SPM, запущена протягом 4-х машинних циклів, здійснить очищення сторінки пам'яті програм. Адресу сторінки попередньо необхідно завантажити у старші біти регістра-покажчика Z. Розряд PGERS скидається апаратно по закінченню очищення сторінки або після закінчення зазначеного часу. Під час запису в секцію NRWW центральний процесор зупиняється на час виконання операції*
0	SPMEN (SELFPRGEN)**	**Дозвіл виконання команди SPM**. Встановлення цього розряду дозволяє протягом 4-х машинних циклів запуск команди SPM. Якщо розряд SPMEN встановлюється одночасно з одним з розрядів RWWSRE, BLBSET, PGWRT або PGERS, виконується операція, визначена цим розрядом (див. опис розрядів). Якщо встановлюється тільки розряд SPMEN, то вміст регістрів R1:R0 зберігається у тимчасовому буфері за адресою, що міститься в регістрі Z (МЗР регістра ігнорується). Розряд SPMEN скидається апаратно після завершення операції або після закінчення зазначеного часу
* У моделях ATmega48x, процесор зупиняється при зверненні до будь-якої комірки пам'яті програм		
** У моделях ATmega48x/88x/168x		

Таблиця 4.3. **Сторінкова організація пам'яті програм**

Характеристики пам'яті / Тип МК	Загальна ємність	Кількість сторінок	Кількість слів на сторінці
ATmega48x	$2^6 \cdot 2^5 = 2K$	$2^6 = 64$	$2^5 = 32$
ATmega8x ATmega8515x ATmega8535x ATmega88x	$2^7 \cdot 2^5 = 4K$	$2^7 = 128$	$2^5 = 32$
ATmega16x ATmega162x ATmega164x ATmega165x ATmega168x	$2^7 \cdot 2^6 = 8K$	$2^7 = 128$	$2^6 = 64$

Закінчення табл. 4.3.

Характеристики пам'яті / Тип МК	Загальна ємність	Кількість сторінок	Кількість слів на сторінці
ATmega32x ATmega324x ATmega325x ATmega3250x	$2^8 \cdot 2^6 = 16K$	$2^8 = 256$	$2^6 = 64$
ATmega64x ATmega640 ATmega645x ATmega6450x	$2^8 \cdot 2^7 = 32K$	$2^8 = 256$	$2^7 = 128$
ATmega128x ATmega1280x ATmega1281x	$2^9 \cdot 2^7 = 64K$	$2^9 = 512$	$2^7 = 128$
ATmega2560x ATmega2561x	$2^{10} \cdot 2^7 = 128K$	$2^{10} = 1024$	$2^7 = 128$

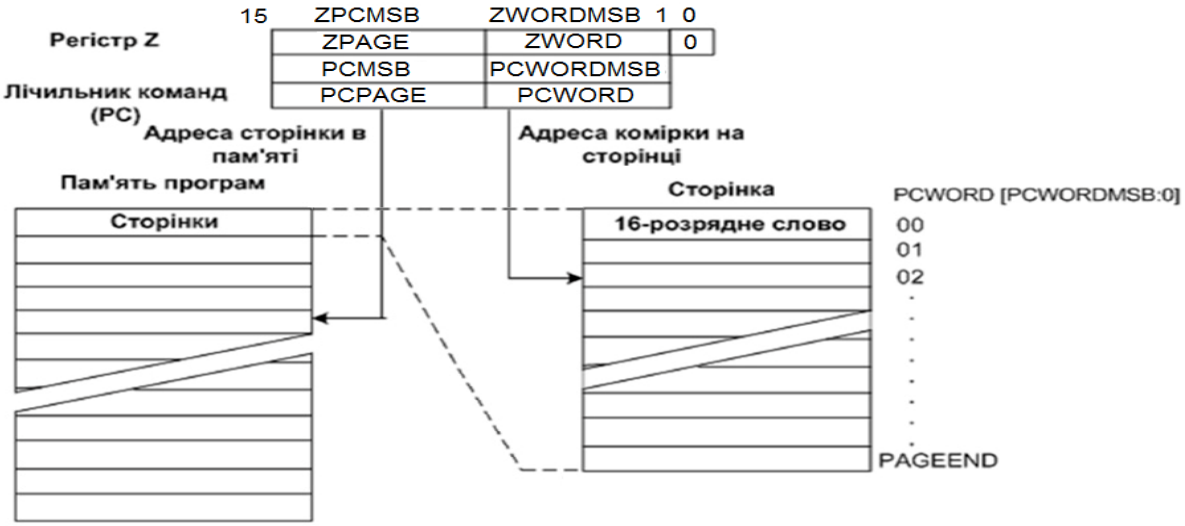


Рис. 4.4. Адресація пам'яті програм при використанні команди SPM

Як приклад, в табл. 4.5 для мікроконтролера Mega8515, який має 32 слова на сторінці, наведено у двійковому та десятковому кодах адреси окремих слів, та адреси молодших байтів.

Як видно із цієї таблиці адреси молодших байтів слів – парні.

Таблиця 4.4. **Адресація пам'яті програм при використанні команди SPM**

Параметр	**ATmega48x**	**ATmega8515x/ 8535x/8x/88x**	**ATmega16x/ 162x/ 164x/165x/ 168x**	**ATmega32x/ 324x/ 325x/ 3250x**	**ATmega64x/ 640x/ 644x/ 645x/ 6450x**	**ATmega128x/ 1280x/ 1281x**	**ATmega2560x/ 2561x**
Розмір пам'яті програм, слів	2K	4K	8K	16K	32K	64K	128K
PCMSB	10	11	12	13	14	15	16
PCWORDMSB	4	4	5	5	6	6	6
ZPCMSB	11	12	13	14	15	16	17
ZWORDMSB	5	5	6	6	7	7	7
PCWORD	PC[4:0]	PC[4:0]	PC[5:0]	PC[5:0]	PC[6:0]	PC[6:0]	PC[6:0]
	Z5-Z1	Z5-Z1	Z6:Z1	Z6:Z1	Z7:Z1	Z7:Z1	Z7:Z1
PCPAGE	PC[10:5]	PC[11:5]	PC[12:6]	PC[13:6]	PC[14:7]	PC[15:7]	PC[16:7]
	Z11:Z6	Z12:Z6	Z13:Z7	Z14:Z7	Z15:Z8	Z16:Z8	Z17:Z8

Примітка. Біти Z17 та Z16 розташовуються в регістрі RAMPZ, MSB – старший значущий розряд (СЗР)

Таблиця 4.5. **Адреси слів на сторінці та адреси молодших байтів**

Адреса слова на сторінці		**Адреса молодшого байта слова**	
Двійковий код	**Десятковий код**	**Двійковий код**	**Десятковий код**
00000	0	00000 0	0
00001	1	00001 0	2
00010	2	00010 0	4
00011	3	00011 0	6
00100	4	00100 0	8
00101	5	00101 0	10
00110	6	00110 0	12
00111	7	00111 0	14
01000	8	01000 0	16
01001	9	01001 0	18
01010	10	01010 0	20
01011	11	01011 0	22

Закінчення табл. 4.5.

Адреса слова на сторінці		Адреса молодшого байта слова	
Двійковий код	**Десятковий код**	**Двійковий код**	**Десятковий код**
01100	12	01100 0	24
01101	13	01101 0	26
01110	14	01110 0	28
01111	15	01111 0	30
10000	16	10000 0	32
10001	17	10001 0	34
10010	18	10010 0	36
10011	19	10011 0	38
10100	20	10100 0	40
10101	21	10101 0	42
10110	22	10110 0	44
10111	23	10111 0	46
11000	24	11000 0	48
11001	25	11001 0	50
11010	26	11010 0	52
11011	27	11011 0	54
11100	28	11100 0	56
11101	29	11101 0	58
11110	30	11110 0	60
11111	31	11111 0	62

4.5. Зміна вмісту пам’яті програм

Загальні відомості

Зміна вмісту пам’яті програм здійснюється в наступній послідовності:

1. Заповнення тимчасового буфера сторінки новим вмістом.
2. Очищення попередньої сторінки.
3. Перенесення вмісту буфера у пам’ять програм.

Слід зазначити, що очищення сторінки може виконуватися як після заповнення буфера, так і перед його заповненням. Однак, якщо необхідно змінити тільки частину сторінки, наведений порядок дій є єдино можливим. У цьому випадку вміст комірок, які не потребують зміни, зберігається в буфері перед очищенням сторінки.

Для визначення моменту закінчення виконання SPM-операцій можна або опитувати стан прапорця SPMEN регістра SPMCR, чекаючи на його скидання, або скористатися перериванням «Готовність SPM».

Це переривання генерується увесь час, поки прапорець SPMEN скинуто. У цьому разі таблиця векторів переривань повинна міститися в секції завантажувача, а переривання необхідно дозволити встановленням прапорця SPMIE регістра SPMCR.

Для очищення сторінки пам'яті програм необхідно занести адресу сторінки в регістр Z (секція ZPAGE) (рис. 4.4), записати значення «x0000011» у регістр SPMCR і протягом чотирьох машинних циклів виконати команду SPM. Вміст регістрів R1 та R0 при цьому ігнорується. У мікроконтролерах з числом сторінок 512 або 1024 для адресації використовуються додаткові молодші біти регістра RAMPZ: Z16 = RAMPZ0, Z17 = RAMPZ1 [1; 7].

Для занесення слова команди в буфер слід завантажити адресу комірки в регістр Z (секція ZWORD) (рис. 4.4), а слово команди – у регістри R1:R0. Після цього необхідно записати значення «x0000001» у регістр SPMCR і протягом чотирьох машинних циклів, виконати команду SPM.

Очищення буфера здійснюється автоматично за закінченням запису сторінки в пам'ять або вручну, записом одиниці у розряд RWWSRE регістра SPMCR. Відзначимо, що запис за тією самою адресою в буфері неможливий без його очищення.

Запис вмісту буфера до пам'яті програм здійснюється аналогічно. У регістр Z (секція ZPAGE) заноситься адреса сторінки, у регістр SPMCR записується значення «x0000101» і протягом чотирьох машинних циклів виконується команда SPM. Вміст регістрів R1 та R0 при цьому ігнорується.

Нижче наведено більш детальний опис наведених вище дій.

Заповнення тимчасового буфера сторінки новим вмістом

Перед записом нових даних у сторінку необхідно спочатку завантажити тимчасовий буфер сторінок. Тимчасовий буфер сторінок – це доступний тільки для запису окремий буфер, що не входить до складу ОЗП, і який містить одну тимчасову сторінку. Слова в нього повинні заноситися послідовно. Перезапис буфера сторінок у Flash-пам'ять відбувається за одну операцію.

Для вибору адреси слова, що буде записано в буфер, використовується регістр Z. Молодший значущий розряд регістра Z ігнорується (містить нуль), тому що запис двох байтів слова відбувається за одну операцію. При заповненні буфера сторінок ігноруються старші біти регістра Z, які не використовуються. Як приклад, структуру регістра Z для мікроконтролера з 32-слівними (64-байтовими) сторінками показано на рис. 4.5.

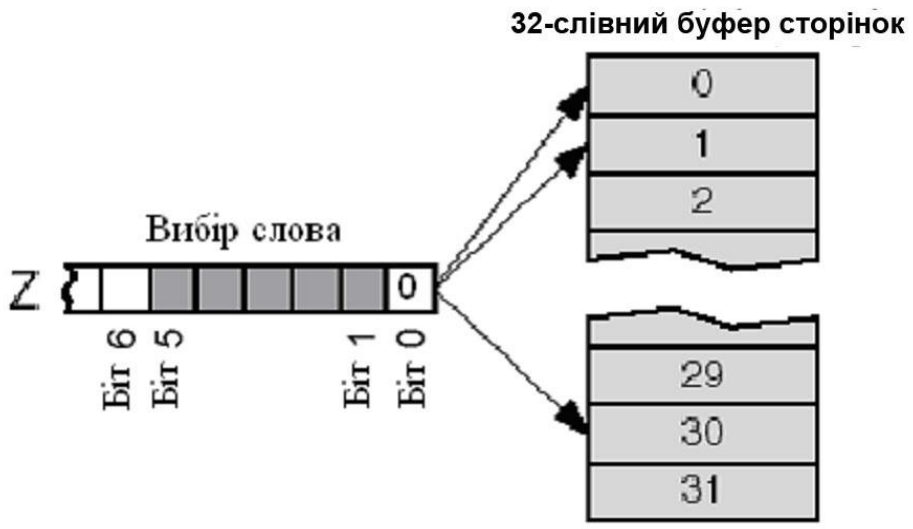


Рис. 4.5. Запис слова у буфер сторінок

Мікроконтролери, що мають більші розміри сторінки для вибору слів використовують більше біт. Наприклад, ATmega128 для адресації слів використовує молодші біти Z1...Z7, тому що одна сторінка містить $2^7 = 128$ слів ($2^8 = 256$ байт).

Щоб записати слово в буфер сторінок необхідно завантажити його в регістри R1:R0 та встановити біт SPMEN у регістрі SPMCR. Протягом чотирьох машинних циклів після цього необхідно виконати команду SPM.

Оновлення Flash-пам'яті

Flash-пам'ять оновлюється сторінками. Перед записом нових даних сторінку необхідно стерти.

Для вибору сторінки, що підлягає стиранню, використовується регістр Z (секція ZPAGE) (рис. 4.5). Він призначений для вказання номера сторінки, яка стирається. Молодші біти, що вибирають слово на сторінці, ігноруються. Наприклад, у мікроконтролері, що має розмір сторінки 32 слова (64 байти), ігноруються шість молодших біт регістра Z.

Для того щоб стерти сторінку необхідно встановити біти PGERS та SPMEN у регістрі SPMCR і протягом чотирьох машинних циклів виконати команду SPM.

Після завантаження даних у тимчасовий буфер сторінок його необхідно записати у Flash-пам'ять. Для цього необхідно записати номер сторінки, яка переноситься, в регістр Z. Потім встановлюються біти PGWRT та SPMEN у регістрі SPMCR, і протягом чотирьох машинних циклів необхідно виконати команду SPM. Вміст регістрів R1:R0 при цьому ігнорується. Як приклад, використання регістра Z для мікроконтролера, сторінки якого містять 32 слова (64 байти) показано на рис. 4.6.

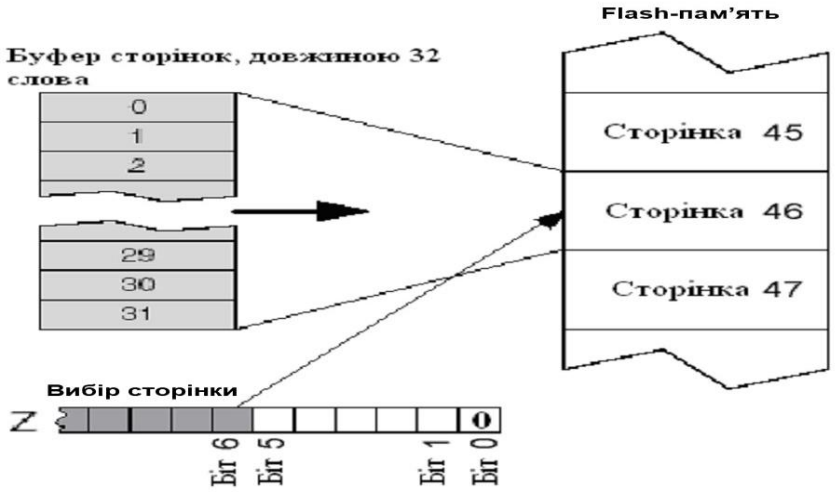


Рис. 4.6. Запис сторінки у Flash-пам'ять

Для визначення готовності мікроконтролера до наступного оновлення необхідно опитувати біт SPMEN.

Зміна комірок захисту завантажувача та прикладної програми

Існують комірки захисту завантажувача BLB12:BLB11 і прикладної програми BLB02:BLB01 (рис. 4.7), програмуванням яких можна заборонити доступ із прикладної програми до програми завантажувача та забезпечити завантажувачу можливість доступу до секції прикладної програми.

	7	6	5	4	3	2	1	0
R0	1	1	BLB12	BLB11	BLB02	BLB01	1	1

Рис. 4.7. Комірки захисту завантажувача та прикладної програми

Комірки захисту знаходяться у регістрі R0 [1; 7]. Для програмування комірки треба скинути відповідний розряд регістра. Для цього необхідне значення завантажуєься в регістр R0, у регістр SPMCR записується значення «x0001001» і протягом чотирьох машинних циклів виконується команда SPM. Вміст регістра Z при цьому ігнорується, однак для сумісності з майбутніми пристроями рекомендується записувати в нього значення $0001. Під час програмування комірок захисту можна звертатися до будь-якої області пам'яті програм.

SPM-переривання

У всіх мікроконтролерів, що підтримують режим самопрограмування, за допомогою переривань можна контролювати процес оновлення Flash-пам'яті. Встановлення біта SPMIE у регістрі SPMCR дозволить формувати SPM-переривання, яке може використовуватися для відстеження закінчення режиму SPM.

Конфлікти EEPROM-пам’яті

Перед виконанням SPM-команди необхідно закінчити всі операції з EEPROM-пам’яттю. Запис/очищення Flash- і EEPROM-пам’яті не може відбуватися одночасно.

4.6. Типові процедури оновлення Flash-пам’яті

На рис. 4.8 показано дві стандартні схеми алгоритмів оновлення Flash-пам’яті.

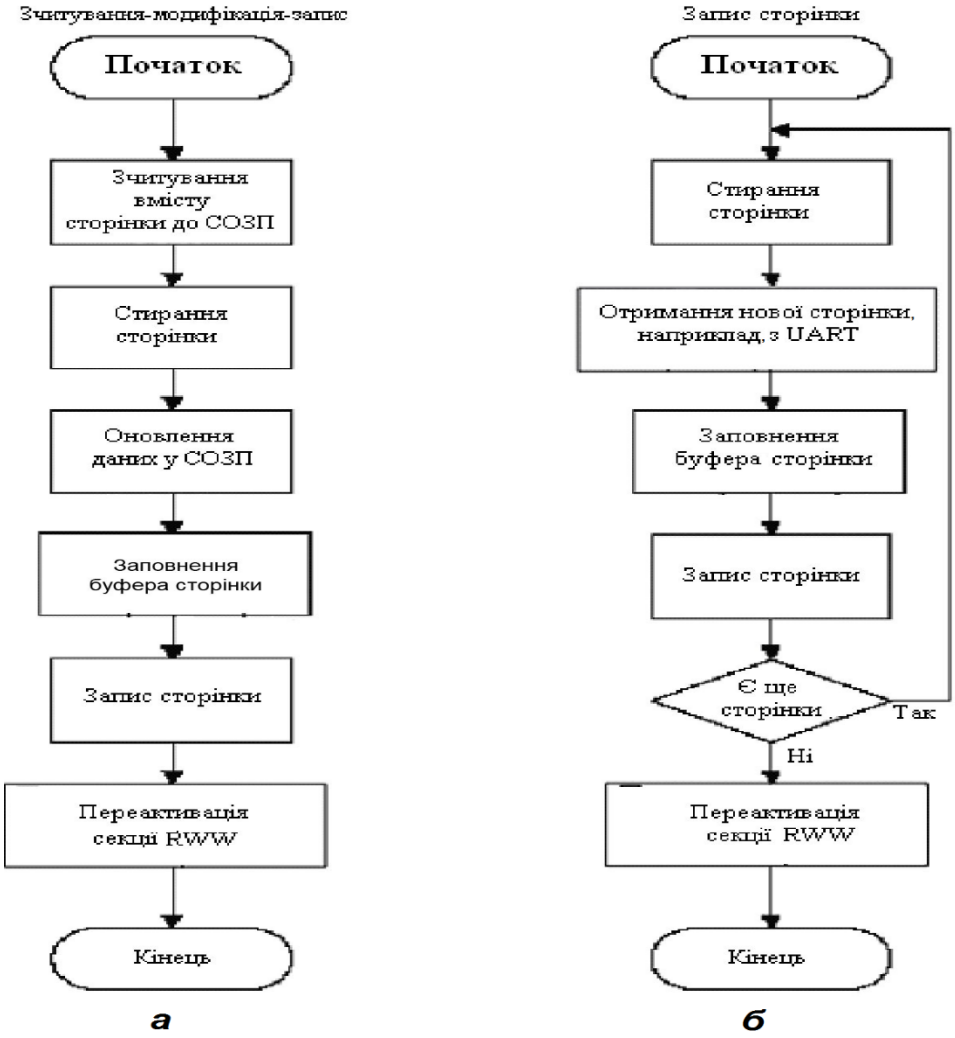


Рис. 4.8. Схеми алгоритмів оновлення Flash-пам’яті

Примітка: Необхідні дії для переактивації секції RWW описано вище у підрозділі 4.3.

Схема ліворуч (рис. 4.8*а*) описує алгоритм «зчитування-модифікації-запису» невеликих частин Flash-пам’яті, наприклад, констант. Схема праворуч (рис. 4.8*б*) описує алгоритм запису сторінки без попереднього зчитування її вмісту, наприклад, виконується запис даних, що надійшли від UART.

4.7. Режими захисту Flash- та EEPROM-пам’яті

Вміст Flash- та EEPROM-пам’яті можна захистити від запису та читання за програмуванням комірок захисту (Lock Bits) LB1 та LB2 [1; 7]. Можливі режими захисту, що відповідають різним станам цих комірок, наведено в табл. 4.6.

Таблиця 4.6. **Режими захисту пам’яті**

№ режиму	Комірки захисту		Опис
	LB1	LB2	
1	1	1	Захист коду та даних відключено
2	0	1	Заборонено запис FLASH- та EEPROM-пам’яті
3	0	0	Заборонено запис та читання FLASH- та EEPROM-пам’яті

Ці біти можна змінити тільки за допомогою послідовного або паралельного програмування [1; 7].

4.8. Читання конфігураційних комірок та комірок захисту

Крім програмування мікроконтролера, завантажувач може також зчитувати вміст конфігураційних комірок та комірок захисту. Для читання байта захисту слід завантажити в регістр Z число $0001, записати в регістр SPMCR значення «x0001001 » і протягом трьох машинних циклів виконати команду LPM. У результаті вміст байта захисту буде занесено у регістр загального призначення R0 (рис. 4.7).

Читання конфігураційних байтів здійснюється аналогічно. У регістр Z завантажується адреса байта ($0000 – молодший байт, $0003 – старший байт, $0002 – додатковий байт), після чого необхідно записати у регістр SPMCR значення «x0001001» і протягом трьох машинних циклів виконати команду LPM. В результаті виконання команди вміст обраного байта конфігурації буде занесено в регістр загального призначення R0 [1; 7].

В [1] також наведено приклад реалізації програми-завантажувача, яка виконує копіювання сторінки пам’яті програми із ОЗП у Flash- пам’ять.

Контрольні запитання та завдання

1. Поясніть, що таке самопрограмування МК та для чого воно використовується.
2. Як логічно розділено всю область пам’яті програм для підтримки самопрограмування?
3. Дайте характеристику різних режимів захисту секції прикладної програми.
4. Поясніть, у чому полягає відмінність між RWW- та NRWW-областями пам’яті програм.
5. Для чого використовується прапорець RWWSB? Як можна виконати його скидання?
6. Яким чином відбувається керування процесом самопрограмування мікроконтролера?
7. Наведіть алгоритм зміни вмісту пам’яті програм.
8. Опишіть процес очищення сторінки пам’яті.
9. Поясніть, у чому полягає особливість використання переривань при самопрограмуванні.
10. Поясніть, як виконати переактивацію RWW-секції.
11. Як може виконуватись завантаження нового вмісту пам’яті програм у мікроконтролер?
12. Яким чином може здійснюватися перехід до програми-завантажувача?
13. Чим визначається розмір секції завантажувача та розмір секції прикладної програми?
14. За допомогою якої команди може здійснюватися зміна вмісту пам’яті програм та комірок захисту завантажувача?
15. Опишіть призначення окремих розрядів регістра SPMCR.
16. Поясніть адресацію пам’яті програм при використанні команди SPM.
17. Як відбувається заповнення тимчасового буфера сторінки новим вмістом?
18. Як відбувається перенесення вмісту буфера у пам’ять програм?
19. В якому регістрі розташовуються біти Z17 та Z16?
20. Чи можна використовувати тимчасовий буфер сторінок для читання?
21. За скільки операцій відбувається перезапис буфера сторінок у Flash-пам’ять?
22. Яким чином новий вміст пам’яті програм під час самопрограмування попадає в МК?
23. Поясніть призначення та використання SPM-переривання.
24. Чи можна при виконанні SPM-команди виконувати операції з EEPROM-пам’яттю?

5. ПРОЕКТУВАННЯ МОДУЛЯ ВВЕДЕННЯ/ВИВЕДЕННЯ

5.1. Загальна характеристика модуля

При розгляді зв’язків між окремими частинами мікропроцесорної системи використовують поняття інтерфейс, що є границею між кількома пристроями, наприклад, між мікропроцесором/мікроконтролером (МП/МК) та зовнішнім пристроєм (ЗВПР) .

Під інтерфейсом розуміють сукупність уніфікованих технічних і програмних засобів, які необхідні для підключення пристроїв до системи чи однієї системи до іншої. Серед властивостей інтерфейсу можна відзначити розв’язування задач синхронізації, вибору напряму передачі даних, а інколи приведення у відповідність рівнів чи форм сигналів.

У даному розділі розглядаються інтерфейси AVR-мікроконтролерів.

Основною частиною інтерфейсу є технічні (апаратні) засоби, що забезпечують зв'язок МК та ЗВПР. Ці засоби у мікропроцесорній техніці часто називають «модулем введення/виведення». Апаратну основу цього модуля складають порти введення/виведення інформації, які виконано на основі регістрів. Крім наявності необхідних апаратних засобів модуль введення/виведення повинен бути програмованим, тобто мати відповідні регістри для запису керуючих слів і регістри, які відображають стан інтерфейсу.

Сучасні МП та МК-ри мають команди обміну даними з необхідною периферією. Введення (читання, прийом) відповідає потоку даних від ЗВПР в МП (МК), а виведення (запис, передача) – потоку даних із МП (МК) до ЗВПР.

5.2. Способи обміну даними між ЗВПР та МК/МП у МПС

У МПС використовуються два способи обміну даними між ЗВПР та МК/МП:

- паралельний, коли одночасно передаються всі біти або декілька біт слова даних;
- послідовний, коли біти слова даних пересилаються по черзі, починаючи, наприклад, з його молодшого розряду.

ЗВПР зв'язується з МК/МП лініями зв'язку, довжина яких при паралельному обміні обмежена і складає кілька метрів.

При послідовному обміні даними довжина ліній зв'язку при використанні, наприклад, модемів, може значно збільшуватися. Ця обставина, а також бажання використовувати для дистанційного обміну інформацією між ЗВПР і МК/МП існуючі канали зв'язку (КЗ), обумовили широке поширення послідовного обміну даними в МПС, або між декількома МПС.

Обмін інформацією в середені МП або МК здійснюється в паралельній формі. Тому при послідовному обміні даними необхідно: при передачі даних від МК/МП до ЗВПР перетворити дані з паралельної форми в послідовну, а при прийомі інформації від ЗВПР та введенні її у МК/МП – перетворити з послідовної форми в паралельну.

Для перетворення даних з паралельної форми в послідовну, інформація завантажується у регістр зсуву (рис. 5.1).

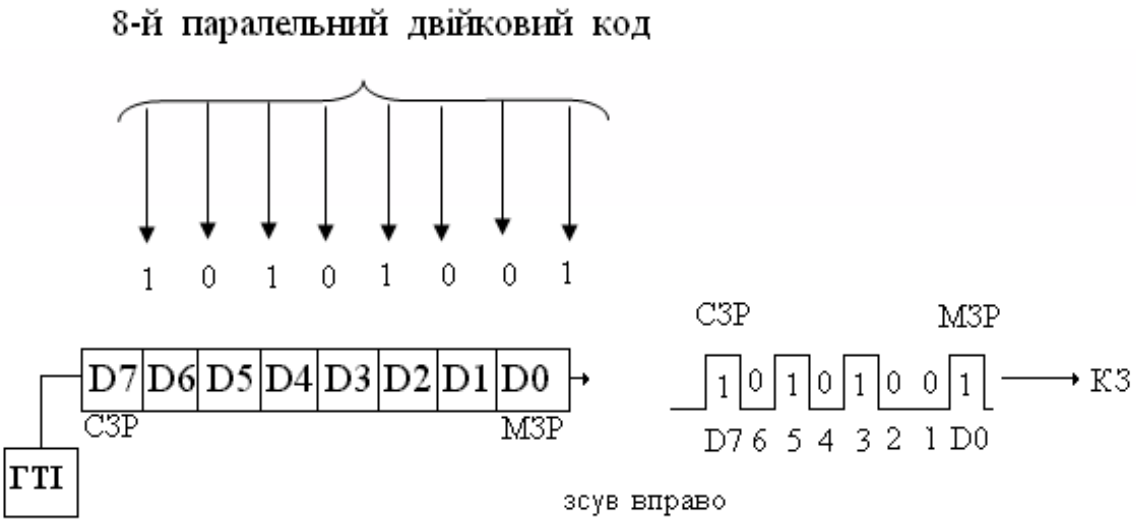


Рис. 5.1. Перетворення даних з паралельної форми в послідовну

При надходженні кожного тактового імпульсу від генератора тактових імпульсів (ГТІ) вміст регістра зсуву послідовно зсувається на один розряд праворуч.

Дані на виході такого регістра мають послідовну форму. Часто при послідовній передачі в канал зв’язку першим передається молодший біт слова даних (МЗР), останнім – старший біт (СЗР).

Для зворотного перетворення даних з послідовної форми в паралельну необхідно виконати дії, протилежні відносно описаного. Дані, що надходять з каналу зв'язку в послідовній формі, вводяться біт за бітом у регістр зсуву. Після заповнення регістра зсуву інформація з нього в паралельній формі передається у МК/МК.

Подібні перетворення виконуються в МК при обміні даними через його послідовний порт (рис. 5.2) [3].

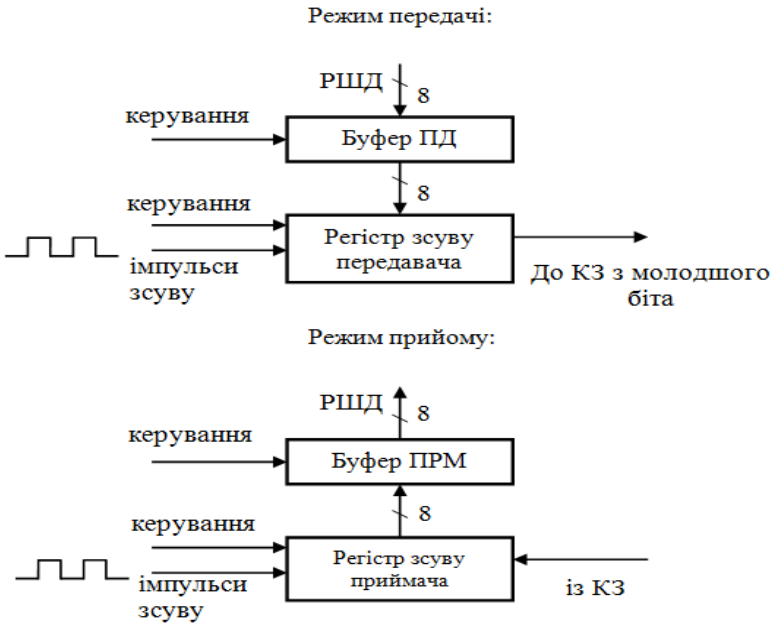


Рис. 5.2. Спрощена структура послідовного порту мікроконтролера

5.3. Порти введення/виведення AVR-мікроконтролерів

5.3.1. Загальна характеристика

Мікроконтролери сімейства AVR мають модулі введення/виведення, апаратну частину яких виконано на базі портів введення/виведення (регістрів). Кожен порт має певне число виводів, через які мікроконтролер може приймати або передавати цифрові сигнали [1; 7].

Задання напряму передачі даних через будь-який контакт введення/виведення виконується програмно. На входах портів знаходяться тригери Шмітта. Якщо лінії сконфігуровані як вхідні, то є можливість підключення внутрішнього підтягуючого резистора з опором 35...120 кОм між входом і шиною живлення V_{CC}. Цей вхід може служити джерелом струму, якщо між входом із задіяним внутрішнім підтягуючим резистором і спільною шиною підключити навантаження,

У портах введення/виведення МК сімейства реалізовано функція «читання/модифікація/запис». Завдяки цьому, використовуючи команди SBI та CBI, можна виконувати операції над будь-яким виводом порту, не впливаючи на інші виводи. Це стосується зміни режиму роботи контакту введення/виведення, зміни вихідного значення і зміни стану внутрішнього підтягуючого резистора (для входів).

Мікроконтролери з кожної групи моделей сімейства мають різну кількість портів і, відповідно, контактів введення/виведення (табл. 5.1).

Таблиця 5.1. **Порти введення/виведення мікроконтролерів сімейства Mega**

Порт введення/виведення	**ATmega8515x**	**ATmega8535x**	**ATmega8x**	**ATmega16x/32x**	**ATmega64x/128x**	**ATmega48x/88x/168x**	**ATmega162x**	**ATmega164x/324x/644x**	**ATmega165x**	**ATmega325x/645x**	**ATmega3250x/6450x**	**ATmega1281x/2561x**	**ATmega640x, ATmega1280x/2560x**
PORTA	●	●	-	●	●	-	●	●	●	●	●	●	●
PORTB	●	●	●	●	●	●	●	●	●	●	●	●	●
PORTC	●	●	7	●	●	7	●	●	●	●	●	●	●
PORTD	●	●	●	●	●	●	●	●	●	●	●	●	●
PORTE	3	-	-	-	●	-	3	-	●	●	●	●	●
PORTF	-	-	-	-	●	-	-	-	●	●	●	●	●
PORTG	-	-	-	-	5	-	-	-	5	6[1)]	6[1)]	6	6
PORTH	-	-	-	-	-	-	-	-	-	-	●	-	●
PORTJ	-	-	-	-	-	-	-	-	-	-	7	-	●
PORTK	-	-	-	-	-	-	-	-	-	-	-	-	●
PORTL	-	-	-	-	-	-	-	-	-	-	-	-	●
Число контактів введення/ виведення	35	32	23	32	53	23	35	32	53	54	69	54	86

[1)] Вивід PG5 – тільки вхід з постійно включеною підтяжкою (суміщений з входом RESET).

Примітка. Цифра позначає розрядність порту (відмінну від восьми).

В усіх МК сімейства переважна більшість контактів введення/виведення мають додаткові (альтернативні функції) та використовуються відповідними периферійними пристроями МК.

5.3.2. Звернення до портів введення/виведення

Звернення до портів виконується через регістри введення/виведення. За деякими винятками, під кожен порт в адресному просторі введення/виведення зарезервовано по 3 адреси [1; 7]. За цими адресами розміщуються три регістри: регістр даних порту PORTx, регістр напряму даних DDRx і регістр виводів порту PINx. Дійсні назви регістрів отримуються підстановкою назви порту замість символу x. Відповідно, регістри порту A називаються PORTA, DDRA, PINA, порту B – PORTB, DDRB, PINB і т. ін.

Регістри PINx, де x = A/B/D/ і т. ін., фізично не є регістрами. За цими адресами здійснюється доступ до значень сигналів на виводах порту.

При виведенні у порт відбувається запис у відповідний регістр даних порту PORTx. При введенні виконується читання регістра виводів порту PINx (сигналів, які присутні на виводах порту). При читанні регістра PORTx у МК вводяться дані з регістра-защіпки порту. При знаходженні мікроконтролера у стані скидання виводи всіх портів перебувають у третьому (високоімпедансному) стані – Hi-Z [9].

Адреси регістрів, що відносяться до портів введення/виведення, наведено в табл. 5.2.

5.3.3. Структура портів введення/виведення

Спрощену структурну схему одного з каналів порту введення/виведення Pxn при роботі його в якості цифрового входу/виходу загального призначення наведено на рис. 5.3 [1].

Кожному каналу порту відповідають розряди трьох регістрів введення/виведення: PORTxn (регістр PORTx), DDRxn (регістр DDRx) і PINxn (регістр PINx). Дійсні назви розрядів регістрів отримуються підстановкою назви порту замість символу «x» і номера розряду замість символу «n». Порядковий номер виводу порту відповідає порядковому номеру розряду регістрів цього порту. Якщо розрядність порту менше восьми, у регістрах порту використовується відповідне число молодших розрядів. Незадіяні старші розряди регістрів доступні тільки для читання і завжди містять нуль.

У наведену структуру контакту окрім цифрових тригерів та логічних елементів входять буфери – B, вихідні сигнали яких мають три стани: логічна одиниця; логічний нуль та третій (висоімпедансний) стан. Використання останніх пояснюється тим, що виходи цих буферів підключено до спільних шин. Щоб уникнути спотворення сигналів на шині, тільки один з буферів в конкретний момент часу під впливом керуючого сигналу – «E» буде активним. Виходи інших (не обраних) буферів на час передачі переводяться в третій стан.

Таблиця 5.2. **Адреси регістрів портів введення/виведення**

<table>
<tr><th>Порт</th><th>Регістр</th><th>ATmega8515x</th><th>ATmega8535x</th><th>ATmega8x</th><th>ATmega16x</th><th>ATmega162x</th><th>ATmega64x, ATmega128x</th><th>ATmega48x/88x/168x</th><th>ATmega164x/324x/644x</th><th>ATmega165x, ATmega325x/645x</th><th>ATmega3250x/6450x</th><th>ATmega1281x/2561x</th><th>ATmega640x, ATmega1280x/2560x</th></tr>
<tr><td rowspan="3">A</td><td>PORTA</td><td colspan="2">$1B ($3B)</td><td>-</td><td colspan="3">$1B ($3B)</td><td>-</td><td colspan="5">$02 ($22)</td></tr>
<tr><td>DDRA</td><td colspan="2">$1A ($3A)</td><td>-</td><td colspan="3">$1A ($3A)</td><td>-</td><td colspan="5">$01 ($21)</td></tr>
<tr><td>PINA</td><td colspan="2">$19 ($39)</td><td>-</td><td colspan="3">$19 ($39)</td><td>-</td><td colspan="5">$00 ($20)</td></tr>
<tr><td rowspan="3">B</td><td>PORTB</td><td colspan="6">$18 ($38)</td><td colspan="6">$05 ($25)</td></tr>
<tr><td>DDRB</td><td colspan="6">$17 ($37)</td><td colspan="6">$04 ($24)</td></tr>
<tr><td>PINB</td><td colspan="6">$16 ($36)</td><td colspan="6">$03 ($23)</td></tr>
<tr><td rowspan="3">C</td><td>PORTC</td><td colspan="6">$15 ($35)</td><td colspan="6">$08 ($28)</td></tr>
<tr><td>DDRC</td><td colspan="6">$14 ($34)</td><td colspan="6">$07 ($27)</td></tr>
<tr><td>PINC</td><td colspan="6">$13 ($33)</td><td colspan="6">$06 ($26)</td></tr>
<tr><td rowspan="3">D</td><td>PORTD</td><td colspan="6">$12 ($32)</td><td colspan="6">$0B ($2B)</td></tr>
<tr><td>DDRD</td><td colspan="6">$11 ($31)</td><td colspan="6">$0A ($2A)</td></tr>
<tr><td>PIND</td><td colspan="6">$10 ($30)</td><td colspan="6">$09 ($29)</td></tr>
<tr><td rowspan="3">E</td><td>PORTE</td><td colspan="2">$07 ($27)</td><td>-</td><td>-</td><td>$07 ($27)</td><td>$03 ($23)</td><td>-</td><td>-</td><td colspan="4">$0E ($2E)</td></tr>
<tr><td>DDRE</td><td colspan="2">$06 ($26)</td><td>-</td><td>-</td><td>$06 ($26)</td><td>$02 ($22)</td><td>-</td><td>-</td><td colspan="4">$0D ($2D)</td></tr>
<tr><td>PINE</td><td colspan="2">$05 ($25)</td><td>-</td><td>-</td><td>$05 ($25)</td><td>$01 ($21)</td><td>-</td><td>-</td><td colspan="4">$0C ($2C)</td></tr>
<tr><td rowspan="3">F</td><td>PORTF</td><td>$03 ($23)</td><td>-</td><td>-</td><td>-</td><td>-</td><td>($62)</td><td>-</td><td>-</td><td colspan="4">$11 ($31)</td></tr>
<tr><td>DDRF</td><td>$02 ($22)</td><td>-</td><td>-</td><td>-</td><td>-</td><td>($61)</td><td>-</td><td>-</td><td colspan="4">$10 ($30)</td></tr>
<tr><td>PINF</td><td>$01 ($21)</td><td>-</td><td>-</td><td>-</td><td>-</td><td>$00 ($20)</td><td>-</td><td>-</td><td colspan="4">$0F ($2F)</td></tr>
</table>

Закінчення табл. 5.2.

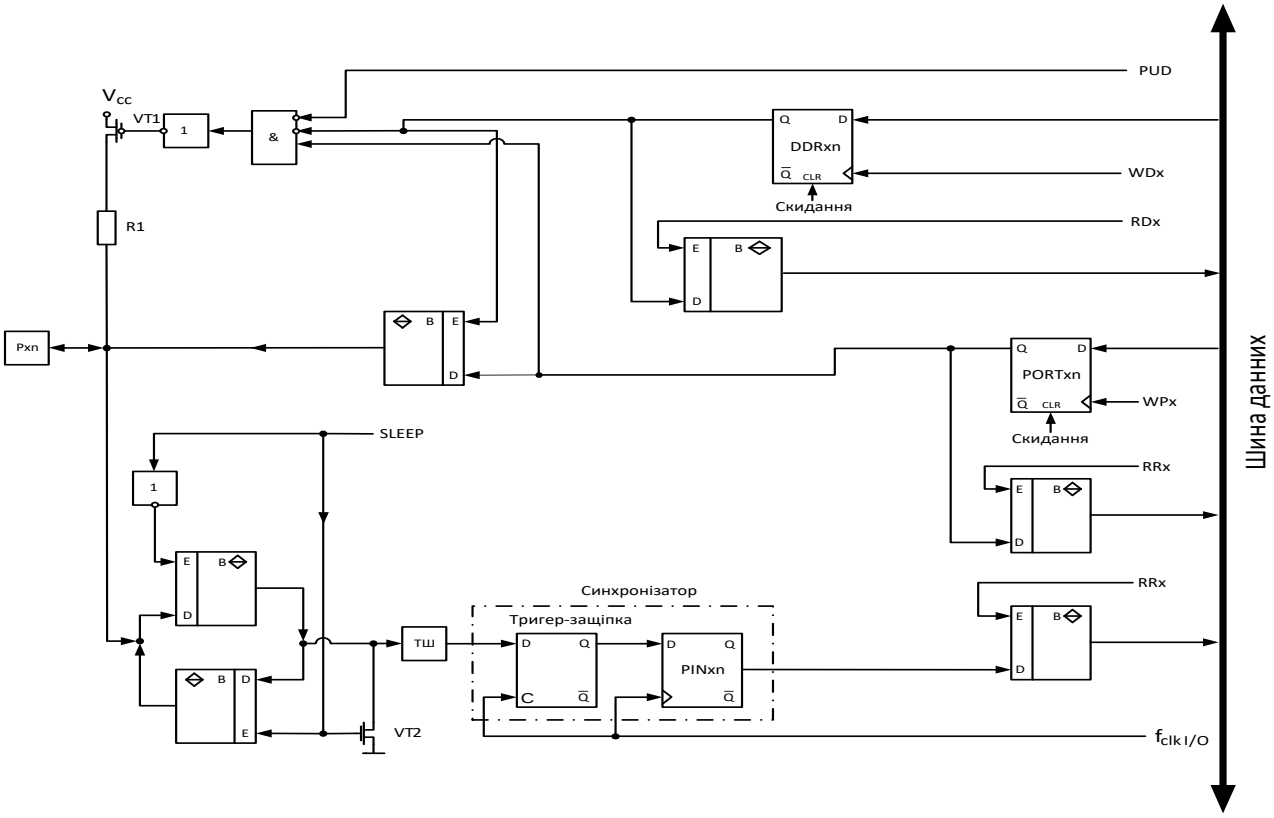
Порт	**Регістр**	**ATmega8515x**	**ATmega8535x**	**ATmega8x**	**ATmega16x**	**ATmega162x**	**ATmega64x, ATmega128x**	**ATmega48x/88x/168x**	**ATmega164x/324x/644x**	**ATmega165x, ATmega325x/645x**	**ATmega3250x/6450x**	**ATmega1281x/2561x**	**ATmega640x, ATmega1280x/2560x**
G	PORTG	-	-	-	-	-	($65)	-	-	$14 ($34)			
	DDRG	-	-	-	-	-	($64)	-	-	$13 ($33)			
	PING	-	-	-	-	-	($63)	-	-	$12 ($32)			
H	PORTH	-	-	-	-	-	-	-	-	-	($DA)	-	($102)
	DDRH	-	-	-	-	-	-	-	-	-	($D9)	-	($101)
	PINH	-	-	-	-	-	-	-	-	-	($D8)	-	($100)
J	PORTJ	-	-	-	-	-	-	-	-	-	($DD)	-	($105)
	DDRJ	-	-	-	-	-	-	-	-	-	($DC)	-	($104)
	PINJ	-	-	-	-	-	-	-	-	-	($DB)	-	($103)
K	PORTK	-	-	-	-	-	-	-	-	-	-	-	($108)
	DDRK	-	-	-	-	-	-	-	-	-	-	-	($107)
	PINK	-	-	-	-	-	-	-	-	-	-	-	($106)
L	PORTL	-	-	-	-	-	-	-	-	-	-	-	($10B)
	DDRL	-	-	-	-	-	-	-	-	-	-	-	($10A)
	PINL	-	-	-	-	-	-	-	-	-	-	-	($109)

Розряд DDRxn регістра DDRx визначає напрям передачі даних через контакт введення/виведення. Якщо цей розряд встановлено в одиницю, то n-й вивід порту є виходом, якщо ж скинуто в нуль – входом.

Розряд PORTxn регістра PORTx виконує подвійну функцію. Якщо вивід функціонує як вихід (DDRxn = 1), цей розряд визначає стан виводу порту. Якщо розряд встановлено в одиницю, на виводі встановлюється напруга ВИСОКОГО рівня. Якщо розряд скинуто в нуль, на виводі встановлюється напруга НИЗЬКОГО рівня.

Якщо ж вивід функціонує як вхід (DDRxn = 0), розряд PORTxn визначає стан внутрішнього підтягуючого резистора для даного виводу. При встановленні розряду PORTxn в одиницю підтягуючий резистор підключається між виводом

мікроконтролера і живленням, за умови, що біт PUD = 0 (рис. 5.4).



PUD – відключення резистора, який підтягує; SLEEP – керування сплячим режимом; $f_{clk\ I/O}$ – тактовий сигнал підсистеми введення/виведення; WDx – запис регістра DDRx; RDx – читання регістра DDRx; WPx – запис регістра PORTx; RRx – читання регістра PORTx; RPx – читання виводів порту Px.

Примітка: сигнали WPx, WDx, RRx, RPx, RDx – спільні для всіх виводів одного порту; сигнали $f_{clk\ I/O}$, SLEEP та PUD – спільні для всіх портів мікроконтролера.

Рис. 5.3. Структурна схема контакту введення/виведення

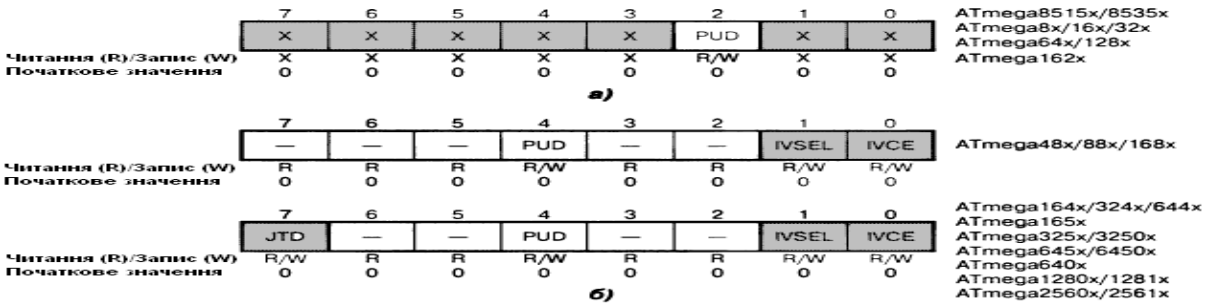


Рис. 5.4. Регістри керування підтяжкою SFIOR (а) та MCUCR (б)

Керування підтягуючими резисторами здійснюється на двох рівнях.

Загальне керування для всіх виводів портів здійснюється розрядом PUD регістра спеціальних функцій SFIOR або регістра керування мікроконтролера MCUCR [1; 7]. Формати цих регістрів наведено вище на рис. 5.4.

Якщо розряд PUD у початковому стані скинуто в нуль, стан підтягуючих резисторів для кожного входу порту буде визначатися станом розрядів PORTxn, за умови, що відповідний розряд DDRxn скинуто в нуль. Якщо ж розряд PUD встановлено в одиницю, підтягуючі резистори відключаються від усіх виводів портів мікроконтролера.

5.3.4. Конфігурування виводів портів введення/виведення

Різні можливості конфігурування виводів портів введення/виведення наведено у табл. 5.3.

Таблиця 5.3. **Конфігурації виводів портів**

DDRxn	PORTxn	PUD*	Функція виводу	Резистор	Примітки
0	0	X	Вхід	Відключений	Третій стан (Hi-Z)**
0	1	0	Вхід	Підключений	При підключенні навантаження між виводом і спільним проводом вивід є джерелом струму
0	1	1	Вхід	Відключений	Третій стан (Hi-Z) для виходу
1	0	X	Вихід	Відключений	Вихід скинуто в нуль
1	1	X	Вихід	Відключений	Вихід встановлено в одиницю
* Відсутній у моделях ATmega161x. ** Стан виводів порту при скиданні.					

Стан виводу мікроконтролера (незалежно від розряду DDRxn) може бути отримано шляхом читання розряду PINxn регістра PINx. При цьому варто пам’ятати, що між дійсною зміною сигналу на виводі та зміною розряду PINxn існує затримка. Ця затримка вноситься вузлом синхронізації, що складається, як показано на рис. 5.5, з розряду тригера PINxn і додаткового тригера-защіпки. Значення сигналу на виводі мікроконтролера фіксується тригером-защіпкою за високим рівнем тактового сигналу і переписується потім у розряд PINxn за наростаючим фронтом тактового сигналу ($f_{CLKI/O}$). Відповідно, величина затримки може становити від 0,5 до 1,5 періоду системного тактового сигналу, як показано на рис. 5.5*а*.

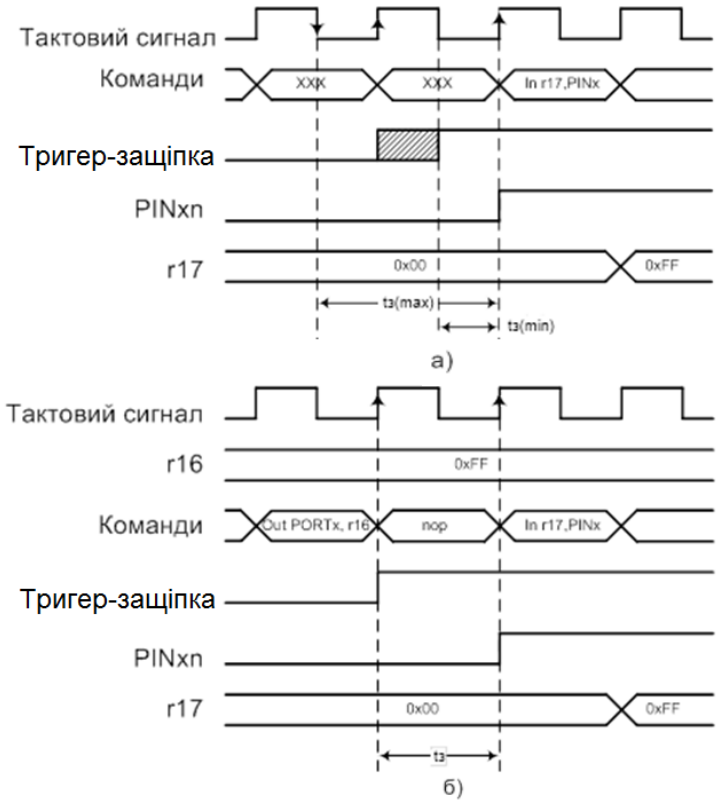


Рис. 5.5. Синхронізація при читанні стану виводу:
а – при зчитуванні стану розряду PINxn; б – при повторному зчитуванні

З цієї ж причини між операціями зміни, виведення і повторного зчитування станів виводу необхідно вставляти команду NOP. Оскільки команда OUT записує інформацію в тригер PORTxn за додатним фронтом тактового сигналу, затримка в цьому випадку дорівнює одному періоду тактового сигналу (рис. 5.5*б*).

5.3.5. Приклад конфігурування одного з портів мікроконтролера

У наведеному нижче прикладі виходи 0 і 1 порту B встановлюються в одиницю, виходи 2 і 3 – скидаються в нуль. Виводи 4...7 порту конфігуруються як входи, при цьому до виводів 6 і 7 підключаються підтягуючі резистори.

Приклад на асемблері

```
…
ldi r16, (1<<PB7) | (1<<PB6) | (1<<PB1) | (1<<PB0)
ldi rl7, (1<<DDRB3) | (1<<DDRB2) | (1<<DDRB1) | (1<<DDRB0)
out P0RTB, rl6              ; Задання стану виходів і
                            ; підтягуючих резисторів
out DDRB, rl7;              ; Задання режимів роботи виводів
nop                         ; Порожня операція (затримка на один такт)
in r16, PINB                ; Читання стану виводів порту
```

Приклад на С

```
unsigned char i;
…
/* Задання стану виходів і підтягуючих резисторів */
/* Задання режимів роботи виводів */
PORTB = (1<<PB7) | (1«PB6) | (1«PB1) | (1<<PB0);
DDRB = (1<< DDRB3) | (1<< DDRB2) | (1<< DDRB1) | (1<<DDRB0);
  _NOP(); /* Функція (порожня операція – затримка на  один такт) */
rl6 = PINB;                          /* Читання стану виводів порту */
…
```

Переважна більшість контактів введення/виведення мікроконтролерів сімейства мають додаткові функції і можуть використовуватися різними периферійними пристроями мікроконтролерів. В цьому випадку користувач повинен або самостійно задавати конфігурацію виводу, або – вивід конфігурується автоматично при включенні відповідного периферійного пристрою [1; 7].

Контрольні запитання та завдання

1. Яким чином виконується звернення до портів введення/виведення?
2. Які три регістри пов’язані з кожним портом введення/виведення?
3. Що означають x та n у позначеннях PORTxn, DDRxn і PINxn?
4. Як здійснюється загальне керування резисторами, що підтягують?
5. Яким чином може бути отримано стан виводу мікроконтролера?
6. Чим пояснюється наявність затримки між дійсною зміною сигналу на виводі порту та зміною розряду PINxn?
7. Дайте визначення поняттю «інтерфейс».
8. Що складає основу «модуля введення/виведення»?
9. Назвіть та поясніть способи обміну даними між ЗВПР та МК/МП.
10. Якою може бути довжина лінії зв`язку при паралельному обміні?
11. Опишіть особливість послідовного обміну.
12. Який цифровий електронний пристрій використовується при перетворенні даних з паралельної форми в послідовну?
13. Поясніть, що означають скорочення МЗР та СЗР?
14. Опишіть спрощену структуру послідовного порту мікроконтролера.
15. Що означає функція «читання/модифікація/запис?
16. Що означає третій стан сигналу для виходу?

6. ПРОЕКТУВАННЯ МОДУЛЯ УАПП/УСАПП

6.1. Загальні відомості

Більшість мікроконтролерів сімейства AVR мають у своєму складі модуль універсального асинхронного приймача/передавача (УАПП), або універсального синхронно/асинхронного приймача/передавача (УСАПП). У деяких моделях міститься по декілька таких модулів (табл. 6.1) [1; 7].

Таблиця 6.1. **Модулі УСАПП в мікроконтролерах сімейства Mega**

Модуль приймача-передавача	**ATmega8515x/ 8535x**	**ATmega8x/16x/32x**	**ATmega64x/128x**	**ATmega48/x88x/ 168x**	**ATmega162x**	**ATmega164x/324x/ 644x**	**ATmega165x, ATmega325x/350x, ATmega645x/6450x**	**ATmega640x/ 1280x/1281x**	**ATmega2560x/ 2561x**
USART0	•	•	•	• 1)	•	• 1)	•	• 1)	• 1)
USART1			•		•	• 1)		• 1)	• 1)
USART2								• 1)	
USART3								• 1)	
1) Може працювати в режимі SPI									

В іноземній літературі модуль УАПП позначається, як UART (Universal Asynchronous Receiver and Transmitter) – універсальний асинхронний приймач-передавач, а модуль УСАПП – як USART (Universal Synchronous and Asynchronous Receiver and Transmitter) – універсальний синхронний/асинхронний приймач-передавач.

Модулі УСАПП, які мають всі мікроконтролери сімейства Mega, при роботі в асинхронному режимі сумісні з модулями УАПП як за розміщенням розрядів керуючих регістрів, так і за функціонуванням. В деяких моделях модулі УСАПП можуть використовуватись в якості ведучого шини SPI [7].

Всі модулі УАПП/УСАПП підтримують дуплексний обмін за послідовним каналом. При цьому швидкість передачі даних може змінюватись у доволі широких межах. У модулях УАПП пакет може бути 8-ми чи 9-ти розрядним, а в модулях УСАПП його довжина може складати від 5-ти до 9-ти розрядів. Ще однією особливістю модулів УСАПП у порівнянні з УАПП є наявність схем формування та контролю парності.

Модулі УАПП/УСАПП можуть виявляти наступні не типові ситуації:

– переповнення;
– помилку кадрування;
– некоректний старт-біт.

В модулях є також корисна функція – фільтрація завад.

Для програмування модулів передбачено 3 переривання, запит на генерацію яких формується при виникненні наступних подій:

– передачу завершено;
– регістр даних передавача пустий;
– прийом завершено.

Виводи мікроконтролера, що використовуються модулями УАПП/УСАПП, є лініями портів введення/виведення загального призначення. Як приклад, виводи деяких мікроконтролерів, що використовуються модулями УСАПП сімейства Mega, та їх функції наведено у табл. 6.2.

Таблиця 6.2. **Виводи, що використовуються модулями УСАПП**

Назва	**ATmega8515x**	**ATmega8535x**	**ATmega8x**	**ATmega16/32x**	**ATmega64x/128x**	**ATmega48x/88x/168x**	**ATmega162x**	**ATmega164x/324x/644x**	**ATmega165x, ATmega325x/350x, ATmega645x/6450x**	**ATmega640x/1280x/2560x**	**ATmega1281x/2561x**	**Опис**
RXD	PD0	PD0	PD0	PD0	-	PD0	-	-	PE0	-	-	Вхід USART/ USART0
RXD0	-	-	-	-	PE0	-	PD0	PD0	-	PE0	PE0	
TXD	PD1	PD1	PD1	PD1	-	PD1	-	-	PE1	-	-	Вихід USART/ USART0
TXD0	-	-	-	-	PE1	-	PD1	PD1	-	PE1	PE1	
XCK	PD4	PB0	PD4	PB0	-	PD4	PD4	-	PE2	-	-	Вхід/вихід зовнішнього тактового сигналу USART/ USART0
XCK0	-	-	-	-	PE2	-	-	PB0	-	PE2	PE2	
RXD1	-	-	-	-	PD2	-	PB2	PD2	-	PD2	PD2	Вхід USART1
TXD1	-	-	-	-	PD3	-	PB3	PD3	-	PD3	PD3	Вихід USART1

Закінчення табл. 6.2.

Назва	**ATmega8515x**	**ATmega8535x**	**ATmega8x**	**ATmega16/32x**	**ATmega64x/128x**	**ATmega48x/88x/168x**	**ATmega162x**	**ATmega164x/324x/644x**	**ATmega165x, ATmega325x/350x, ATmega645x/6450x**	**ATmega640x/1280x/2560x**	**ATmega1281x/2561x**	**Опис**
XCK1	-	-	-	-	PD5	-	PD2	PD4	-	PD5	PD5	Вхід/вихід зовнішнього тактового сигналу USART1
RXD2	-	-	-	-	-	-	-	-	-	PH0	-	Вхід USART2
TXD2	-	-	-	-	-	-	-	-	-	PH1	-	Вихід USART2
XCK2	-	-	-	-	-	-	-	-	-	PH2	-	Вхід/вихід зовнішнього тактового сигналу USART2
RXD3	-	-	-	-	-	-	-	-	-	PJ0	-	Вхід USART3
TXD3	-	-	-	-	-	-	-	-	-	PJ1	-	Вихід USART3
XCK3	-	-	-	-	-	-	-	-	-	PJ2	-	Вхід/вихід зовнішнього тактового сигналу USART3

6.2. Опис структури модулів УАПП/УСАПП

Спрощену структурну схему модуля УАПП/УСАПП наведено на рис. 6.1. Елементи, які виділені на рисунку сірим кольором, є тільки у складі модулів УСАПП [1; 7].

Модуль має три частини: блок тактування, блок передавача та блок приймача.

Блок тактування включає схему синхронізації, яка використовується при роботі у синхронному режимі, і контролер швидкості передачі. У модулях УАПП блок тактування складається тільки з контролера швидкості передачі.

Блок передавача має однорівневий буфер UDR, регістр зсуву, схему формування біта парності (тільки УСАПП) і схему керування.

Блок приймача, включає схеми відновлення тактового сигналу і даних; схему контролю парності (тількиУСАПП); буфер UDR: дворівневий в УСАПП та однорівневий в УАПП; регістр зсуву, а також схему керування.

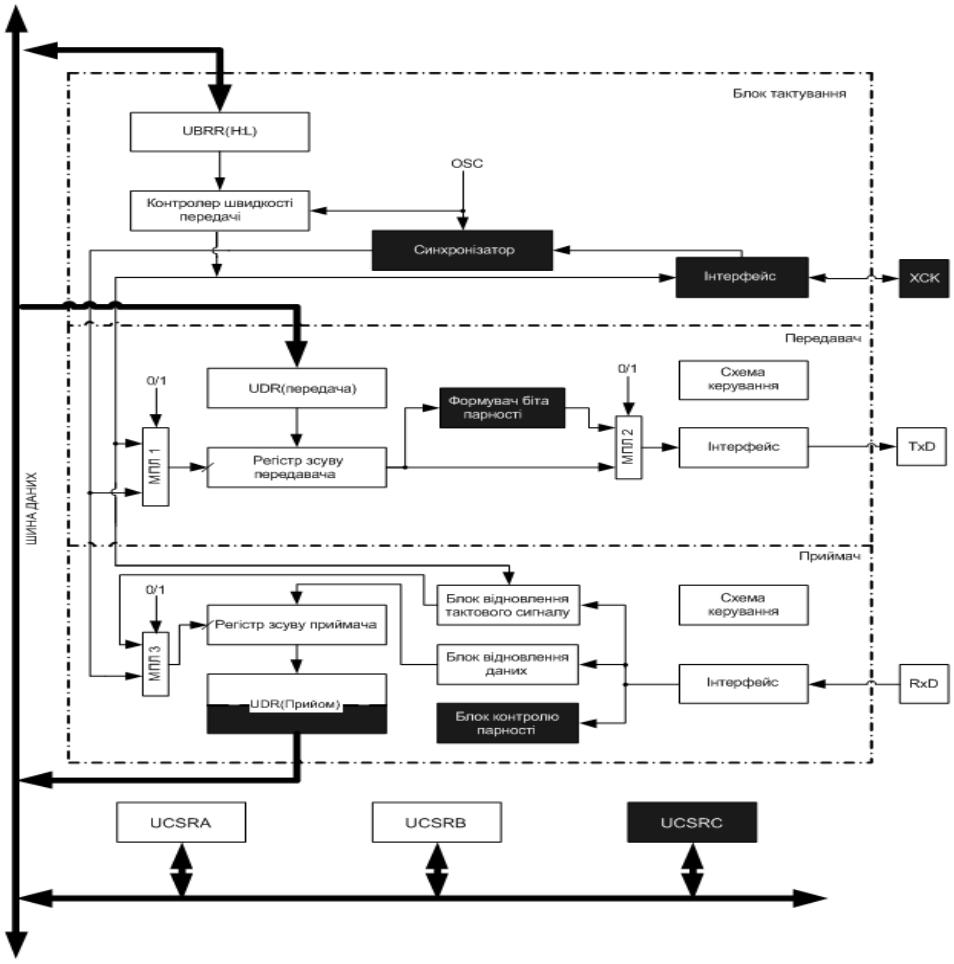


Рис. 6.1. Спрощена структурна схема модуля УАПП/УСАПП

Буферні регістри приймача і передавача розміщуються за однаковою адресою простору регістрів введення/виведення і позначаються як регістр даних UDR (Universal Data Register – UDRn, де n = 0/1). У цих регістрах зберігаються молодші 8 розрядів даних, які приймаються чи передаються. Під час читання UDR виконується звернення до буферного регістра приймача, а у разі запису – до буферного регістра передавача. Розміщення регістрів даних для деяких моделей мікроконтролерів наведено у [1; 7].

Дворівневий буфер приймача у модулях УСАПП є FIFO-буфером (First In First Out, першим зайшов, першим вийшов). Зміна стану буфера відбувається при будь-якому зверненні до регістра UDR. Тому, не варто використовувати регістр UDR в якості операндів команд типу «читання/модифікація/запис» (SBI та CBI) та команди перевірки SBIC та SBIS, які також змінюють стан буфера приймача.

6.3 . Опис керуючих регістрів модуля УСАПП

Для керування модулями УСАПП використовуються три регістри: UCSRA (UCSRnA), UCSRB (UCSRnB) та UCSRC (UCSRnC), де n = 0, 1, 2 або 3. Адреси цих регістрів, їх формати та опис окремих розрядів наведено у [1; 7].

6.4 . Швидкість прийому/передачі даних

В асинхронному режимі, а також у синхронному режимі при роботі у якості ведучого, швидкість прийому та передачі даних задається контролером швидкості передачі, що функціонує як дільник системного тактового сигналу з програмованим коефіцієнтом ділення. Коефіцієнт визначається вмістом регістра контролера UBRR (UBRRn). Після дільника у блок приймача сигнал надходить одразу, а у блок передавача – через додатковий дільник, коефіцієнт ділення якого: 2, 8 чи 16 залежить від режиму роботи модуля УАПП/УСАПП.

На рис. 6.2 у якості прикладу наведено структурну схему блока синхронізації модуля УСАПП для мікроконтролера Mega 128 [1].

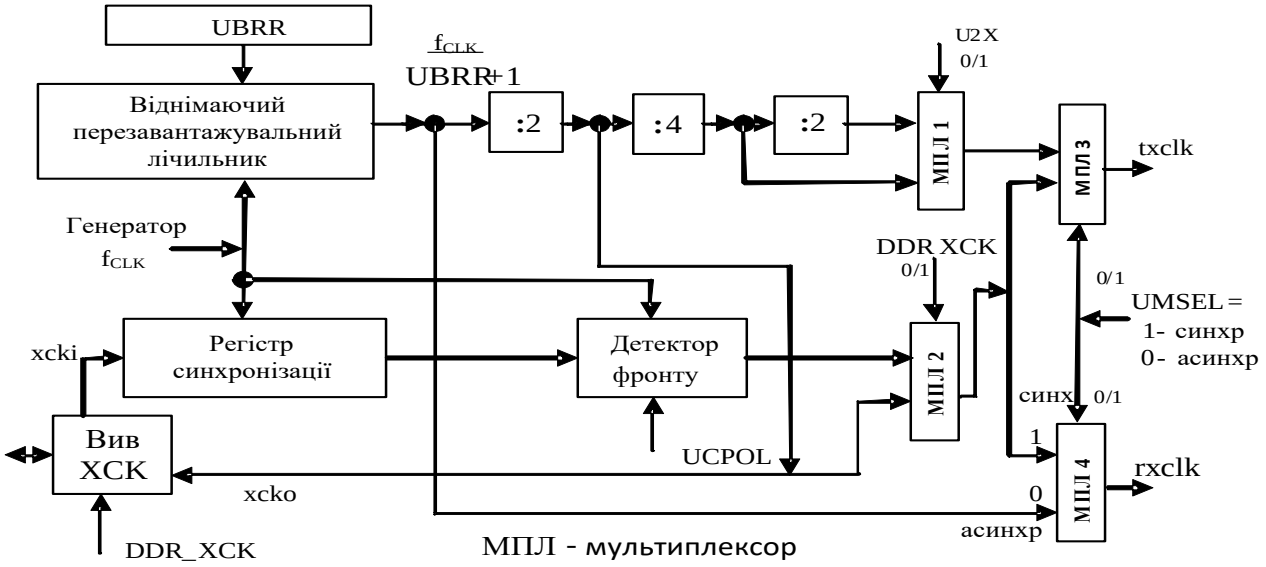


Рис. 6.2. Структурна схема блоку синхронізації модуля УСАПП

На схемі використовуються наступні сигнали:

- txclk – синхронізація передавача (внутрішній сигнал);
- rxclk – синхронізація приймача (внутрішній сигнал);
- xcki – вхідний сигнал від виводу XCK (внутрішній сигнал), який використовується для синхронної підлеглої (веденої) роботи;
- xcko – вихідний сигнал синхронізації до виводу XCK (внутрішній сигнал), який використовується у ведучому синхронному режимі;
- f_{CLK} – частота системного тактового генератора.

УСАПП підтримує чотири режими роботи синхронізації:

- нормальна асинхронна;
- асинхронна з подвоєнням швидкості;
- ведуча синхронна;
- та підлегла (ведена) синхронна.

Для програмування асинхронного чи синхронного режиму роботи призначено біт UMSEL у регістрі керування та статусу UCSRC. Якщо UMSEL = 0, то модуль буде працювати в асинхронному режимі, якщо UMSEL = 1, то у синхронному.

В асинхронному режимі можливе подвоєння швидкості передачі. Для цього призначено біт U2X регістра UCSRA [1].

При використанні синхронного режиму біт DDR_XCK у регістрі напряму даних для виводу XCK задає, чи буде синхронізація внутрішньою: DDR_XCK = 1 (ведучий режим) чи зовнішньою: DDR_XCK = 0 (ведений режим). Вивід XCK активний тільки при використанні синхронного режиму.

Внутрішня синхронізація використовується для асинхронного та ведучого синхронного режимів роботи. Зовнішня синхронізація використовується у синхронному підлеглому (веденому) режимі роботи.

Регістр UBRR є 12-розрядним та фізично розміщений у двох регістрах введення/виведення: UBRRH (UBRRnH) та UBRRL (UBRRnL). Адреси і назви цих регістрів для деяких моделей мікроконтролерів наведено у [1; 7].

Регістр UBRRH (UBRRnH у ряді моделей, наприклад, ATmega8x, ATmega8515x, ATmega16x, ATmega162x, ATmega32x та ATmega323x) має однакову адресу з регістром керування UCSRC (UCSRnC). Тому, при зверненні за цими адресами, необхідно виконати ряд додаткових дій для вибору одного з цих регістрів.

При запису один з двох цих регістрів визначається станом старшого розряду числа, яке записано у регістр UCSRC (UCSRnC) – URSEL (URSELn). Якщо цей розряд скинуто у нуль, змінюється вміст регістра UBRRH (UBRRnH). Якщо старший розряд числа встановлено в одиницю, змінюється вміст регістра керування UCSRC (UCSRnC). Нижче наведено фрагменти програм, які ілюструють сказане.

Приклад на асемблері

```
. . .
; Запис 2 у UBRRH
ldi rl6, 0x02
out UBRRH, rl6
; Встановлення розрядів URSEL, USBS та UCSZ1 регістра UCSRC
```

```
ldi rl6, (l<<URSEL) | (1<<USBS) | (1<<UCSZ1)
out UCSRC, rl6
...
```

Приклад на С

```
...
/*  Запис 2 у UBRRH */
UBRRH = 0x02;
/* Встановлення розрядів URSEL, USBS та UCSZ1 регістра UCSRC */
UCSRC = (1<<URSEL) | (1<<USBS) | (1<<UCSZ1);
...
```

Для вибору одного з двох розглянутих вище регістрів при читанні використовується відповідна часова послідовність. При першому зверненні за вказаною адресою повертається значення регістра UBRRH (UBRRHn). При повторному зверненні за цією адресою у наступному машинному циклі повертається значення регістра UCSRC (UCSRnC).

Нижче наведено фрагменти програм, які ілюструють сказане. Переривання при виконанні цієї послідовності команд мають бути заборонені.

Приклад на асемблері

```
USART_ReadUCSRC:      ; Читання регістра UCSRC
in rl6, UBRRH         ; rl6 = UBRRH
in rl6, UCSRC         ; Повернення значення регістра UCSRC у регістр r16
ret ; Повернення з підпрограми.
```

Приклад на С

```
unsigned char USART_ReadUCSRC(void)
{
        unsigned char ucsrc;
        ucsrc = UBRRH;
        ucsrc = UCSRC;
        return ucsrc;
}
```

При роботі в асинхронному режимі швидкість обміну визначається не тільки вмістом регістра UBRR, але й станом розряду U2X (U2Xn) регістра UCSRA (UCSRnA) [1]. Якщо цей розряд встановлено в одиницю, коефіцієнт ділення

попереднього дільника зменшується у два рази, а швидкість обміну відповідно подвоюється. При роботі у синхронному режимі цей розряд має бути скинуто.

Швидкість обміну в модулях УАПП/УСАПП визначається наступними формулами:

– звичайний асинхронний режим (U2Xn = 0):

$$BAUD = f_{clк} / (16(UBRR+1));$$

– пришвидшений асинхронний режим (U2Xn = 1):

$$BAUD = f_{clк} / (8(UBRR+1));$$

– синхронний ведучий режим:

$$BAUD = f_{clк} / (2(UBRR+1)),$$

де BAUD – швидкість передачі у бодах, $f_{clк}$ – тактова частота мікроконтролера, UBRR – вміст регістра контролера швидкості передачі: 0...4095.

В якості прикладу в табл. 6.3 наведено значення регістра UBRR, що дозволяють отримати стандартні для асинхронного режиму швидкості передачі при використанні деяких резонаторів, а також значення похибок, що отримуються, відносно стандартних швидкостей [7].

При максимальному значенні швидкості UBRR = 0, похибка = 0,0%. Рекомендується використовувати значення регістра UBRR, при яких отримувана швидкість передачі відрізняється від необхідного значення менше ніж на 0,5% (у табл. 6.3 виділено жирним шрифтом).

Швидкість прийому та передачі при роботі в синхронному режимі в якості веденого визначається частотою сигналу, що поступає на вивід XCK (XCKn). Частота цього сигналу має задовольняти виразу $f_{XCK} < f_{CLK}/4$ [7]. Це обмеження пов'язано з тим, що сигнал з виводу XCK (XCKn) спочатку синхронізується з тактовою частотою мікроконтролера, а потім проходить через детектор фронтів.

6.5. Формат кадру

Сукупність одного слова (байта) даних і додаткової інформації під час передачі називається кадром (рис. 6.3).

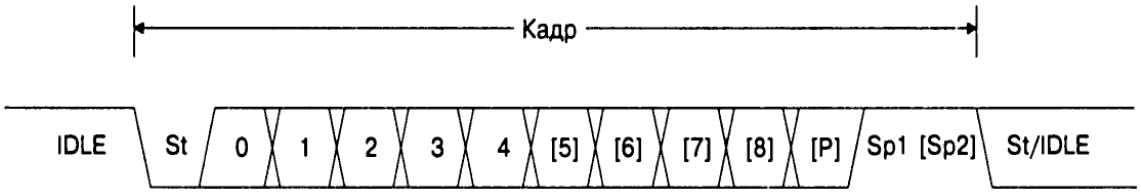


Рис. 6.3. Формат кадру

Таблиця 6.3. **Приклад значень регістра UBRR**

Швидкість [бод]	1.0000 МГц				1.8432 МГц				2.0000 МГц				3.6864 МГц				4.0000 МГц				7.3728 МГц			
	U2X=0		U2X=1		U2X=0		U2X=1		U2X=0		U2X=1		U2X=0		U2X=1		U2X=0		U2X=1		U2X=0		U2X=1	
	UBRR	Похибка [%]	UBRR	Похибка [%]	UBRR	Похибка [%]	UBRR	Похибка [%]	UBRR	Похибка [%]	UBRR	Похибка [%]	UBRR	Похибка [%]	UBRR	Похибка [%]	UBRR	Похибка [%]	UBRR	Похибка [%]	UBRR	Похибка [%]	UBRR	Похибка [%]
2400	**25**	0.2	**51**	0.2	**47**	0.0	**95**	0.0	**51**	0.2	**103**	0.2	**95**	0.0	**191**	0.0	**103**	0.2	**207**	0.2	**191**	0.0	**383**	0.0
4800	**12**	0.2	**25**	0.2	**23**	0.0	**47**	0.0	**25**	0.2	**51**	0.2	**47**	0.0	**95**	0.0	**51**	0.2	**103**	0.2	**95**	0.0	**191**	0.0
9600	6	7.5	**12**	0.2	**11**	0.0	**23**	0.0	**12**	0.2	**25**	0.2	**23**	0.0	**47**	0.0	**25**	0.2	**51**	0.2	**47**	0.0	**95**	0.0
14400	3	7.8	8	-3.5	**7**	0.0	**15**	0.0	8	-3.5	16	2.1	**15**	0.0	**31**	0.0	16	2.1	34	-0.8	**31**	0.0	**63**	0.0
19200	2	7.8	6	-7.0	**5**	0.0	**11**	0.0	6	-7.0	**12**	0.2	**11**	0.0	**23**	0.0	**12**	2.2	**25**	0.2	**23**	0.0	**47**	0.0
28800	1	7.8	3	8.5	**3**	0.0	**7**	0.0	3	8.5	8	-3.5	**7**	0.0	**15**	0.0	8	-3.5	16	3.1	**15**	0.0	**31**	0.0
38400	1	22.9	2	8.5	**2**	0.0	**5**	0.0	2	8.5	6	-7.0	**5**	0.0	**11**	0.0	6	-7.0	**12**	0.2	**11**	0.0	**23**	0.0
57600	0	7.8	1	8.5	**1**	0.0	**3**	0.0	1	8.5	3	8.5	**3**	0.0	**7**	0.0	3	8.5	8	-3.5	**7**	0.0	**15**	0.0
76800	—	—	1	-18.6	**1**	-25.0	**2**	0.0	1	-18.6	2	8.5	**2**	0.0	**5**	0.0	2	8.5	6	-7.0	**5**	0.0	**11**	0.0
115200	—	—	0	8.5	**0**	0.0	**1**	0.0	0	8.5	1	8.5	**1**	0.0	**3**	0.0	1	8.5	3	8.5	**3**	0.0	**7**	0.0
230400	—	—	—	—	—	—	**0**	0.0	—	—	—	—	**0**	0.0	**1**	0.0	0	8.5	1	8.5	**1**	0.0	**3**	0.0
250000	—	—	—	—	—	—	—	—	—	—	0	0.0	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
1.0M	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0	-7.8
Max [1]	62.5K		125K		115.2K		230.4K		125K		250K		230.4K		460.8K		250K		0.5M		460.8K		921.6K	

[1] При UBRR = 0 і похибці 0.0%

Закінчення табл. 6.3

Швидкість [бод]	8.0000 МГц				11.0592 МГц				14.7456 МГц				16.0000 МГц				18.4320 МГц				20.0000 МГц			
	U2X=0		U2X=1		U2X=0		U2X=1		U2X=0		U2X=1		U2X=0		U2X=1		U2X=0		U2X=1		U2X=0		U2X=1	
	UBRR	Похибка [%]	UBRR	Похибка [%]	UBRR	Похибка [%]	UBRR	Похибка [%]	UBRR	Похибка [%]	UBRR	Похибка [%]	UBRR	Похибка [%]	UBRR	Похибка [%]	UBRR	Похибка [%]	UBRR	Похибка [%]	UBRR	Похибка [%]	UBRR	Похибка [%]
2400	**207**	0.2	**416**	-0.1	**287**	0.0	**575**	0.0	**383**	0.0	**767**	0.0	**416**	-0.1	**832**	0.0	**479**	0.0	**959**	0.0	**520**	0.0	**1041**	0.0
4800	**103**	0.2	**207**	0.2	**143**	0.0	**287**	0.0	**191**	0.0	**383**	0.0	**207**	0.2	**416**	-0.1	**239**	0.0	**479**	0.0	**259**	0.2	**520**	0.0
9600	**51**	0.2	**103**	0.2	**71**	0.0	**143**	0.0	**95**	0.0	**191**	0.0	**103**	0.2	**207**	0.2	**119**	0.0	**239**	0.0	**129**	0.2	**259**	0.2
14400	34	-0.8	68	0.6	**47**	0.0	**95**	0.0	**63**	0.0	**127**	0.0	68	0.6	138	-0.1	**79**	0.0	**159**	0.0	**86**	-0.2	**173**	-0.2
19200	**25**	0.2	**51**	0.2	**35**	0.0	**71**	0.0	**47**	0.0	**95**	0.0	**51**	0.2	**103**	0.2	**59**	0.0	**119**	0.0	**64**	0.2	**129**	0.2
28800	16	2.1	34	-0.8	**23**	0.0	**47**	0.0	**31**	0.0	**63**	0.0	34	-0.8	68	0.6	**39**	0.0	**79**	0.0	42	0.9	**86**	-0.2
38400	12	0.2	**25**	0.2	**17**	0.0	**35**	0.0	**23**	0.0	**47**	0.0	**25**	0.2	**51**	0.2	**29**	0.0	**59**	0.0	32	-1.4	**64**	0.2
57600	8	-3.5	16	2.1	**11**	0.0	**23**	0.0	**15**	0.0	**31**	0.0	16	2.1	34	0.8	**19**	0.0	**39**	0.0	21	-1.4	42	0.9
76800	6	-7.0	**12**	0.2	**8**	0.0	**17**	0.0	**11**	0.0	**23**	0.0	**12**	0.2	**25**	0.2	**14**	0.0	**29**	0.0	15	1.7	32	-1.4
115200	3	8.5	8	-3.5	**5**	0.0	**11**	0.0	**7**	0.0	**15**	0.0	8	-3.5	16	2.1	**9**	0.0	**19**	0.0	10	-1.4	21	-1.4
230400	1	8.5	3	8.5	**2**	0.0	**5**	0.0	**3**	0.0	**7**	0.0	3	8.5	8	-3.5	**4**	0.0	**9**	0.0	4	8.5	10	-1.4
250000	1	0.0	**3**	0.0	2	-7.8	5	-7.8	3	-7.8	6	5.3	**3**	0.0	**7**	0.0	4	-7.8	8	2.4	**4**	0.0	**9**	0.0
0.5M	0	0.0	**1**	0.0	—	—	2	-7.8	1	-7.8	3	-7.8	**1**	0.0	**3**	0.0	—	—	4	-7.8	—	—	**4**	0.0
1.0M	—	—	**0**	0.0	—	—	—	—	0	-7.8	1	-7.8	**0**	0.0	**1**	0.0	—	—	—	—	—	—	—	—
Max [1]	0.5M		1.0M		691.2K		1.3824M		921.6K		1.8432M		1.0M		2.0M		1.152M		2.304M		1.25M		2.5M	

[1] При UBRR = 0 і похибці 0.0%

На рис. 6.3 використано наступні скорочення:

- St – старт-біт, завжди має низький рівень;
- 0…8 – номер біта даних;
- P – біт паритету: парність або непарність;
- Sp1, Sp2 – стоп-біти, завжди мають високий рівень;
- IDLE – стан очікування, в якому призупинено обмін лінією RxD чи TxD.

У стані очікування на лінії має бути високий рівень.

До початку передачі в лінію передається високий рівень цифрового сигналу, що говорить приймачу про справність лінії зв`язку. Кадр починається із нульового старт-біта, який використовується схемою синхронізації приймача для підлаштування фази синхрочастоти приймача під отримувані посилки (біти). Далі передаються біти слова даних, починаючи з молодшого розряда. Після останнього старшого розряда слова даних передаються один або два стоп-біти. Якщо

запрограмовано формування біта парності, то він розміщується між старшим розрядом слова даних і першим стоп-бітом.

Формат кадру визначається кількома розрядами регістрів UCSRB (UCSRnB) та UCSRC (UCSRnC) [1; 7]. Розмір слова даних в USART визначається відповідно до табл. 6.4.

Таблиця 6.4. **Визначення розміру слова даних у модулях УСАПП**

UCSZ2 (UCSZn2)	UCSZ1 (UCSZnl)	UCZ0 (UCSZn0)	Розмір слова даних
0	0	0	5 розрядів
0	0	1	6 розрядів
0	1	0	7 розрядів
0	1	1	8 розрядів
1	0	0	Зарезервовано
1	0	1	Зарезервовано
1	1	0	Зарезервовано
1	1	1	9 розрядів

Примітка: n = 0, 1, 2 чи 3.

У модулях УАПП слово даних може бути тільки 8- або 9-розрядним, що визначається станом прапорця CHR9 (CHR9n) регістра UCSRB (UCSRnB). Якщо цей прапорець скинуто в нуль, розмір слова дорівнює 8 розрядам, якщо встановлено в одиницю – 9 розрядам.

Вибір кількості стоп-бітів в модулях УСАПП здійснюється за допомогою розряду USBS (USBSn) регістра UCSRC (UCSRnC). Якщо цей розряд скинуто в нуль, блок передавача у кінці посилки формує один стоп-біт. Якщо розряд встановлено в одиницю, блок передавача формує два стоп-біти. Слід зазначити, що приймачем другий стоп-біт ігнорується, і, відповідно, помилки кадрування виявляються тільки для першого стоп-біта.

Функціонування схеми контролю парності модулів УСАПП визначають розряди UPM1:UPM0 (UPMn1:UPMn0) регістра UCSRC (UCSRnC) відповідно до табл. 6.5.

Значення біта парності отримується шляхом виконання операції «виключне АБО» над усіма розрядами слова даних, яке передається. Якщо використовується перевірка на непарність (odd parity), то отриманий результат інвертується:

$$P_{EVEN} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0,$$
$$P_{ODD} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1.$$

Таблиця 6.5. **Керування контролем парності**

UPM1 (UPMnl)	**UPM0 (UPMn0)**	**Режим роботи**
0	0	Контроль відключено
0	1	Зарезервовано
1	0	Контроль включено, перевірка на непарність (odd parity)
1	1	Контроль включено, перевірка на парність (even parity)

Примітка: n = 0, 1, 2 чи 3.

Якщо контроль парності включено, біт парності вставляється передавачем між старшим розрядом даних, які передаються, і першим стоп-бітом.

6.6. Передача даних

Для дозволу роботи передавача необхідно встановити в одиницю розряд TXEN (TXENn) регістра UCSRB (UCSRnB). Після цього вивід TXD (TXDn) підключається до передавача модуля УАПП/УСАПП і починає функціонувати, як вихід, незалежно від значень у регістрах керування напрямом обміну портом. Якщо використовується синхронний режим роботи, треба програмно перевизначити функціонування виводу ХСК (ХСКn) на введення або виведення.

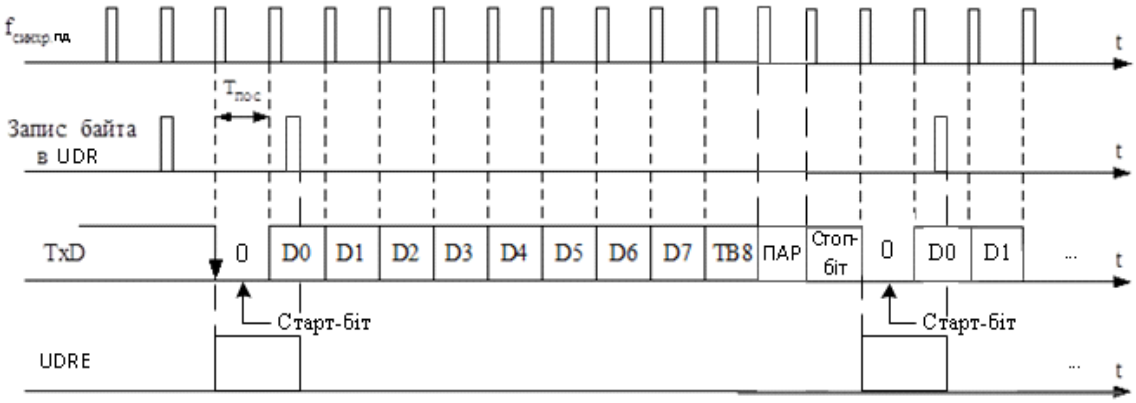
В асинхронному режимі передача ініціюється записом даних, які передаються, у буферний регістр передавача – UDRn (рис. 6.4*а*).

З регістра UDRn дані пересилаються у регістр зсуву передавача. Якщо використовуються 9-розрядний формат даних, значення розряду TXB8 (TXB8n) регістра UCSRB (UCSRnB) копіюється у 9-й розряд регістра зсуву. 9-й розряд даних повинен бути завантажений у розряд TXB8 (TXB8n) до запису байта даних у регістр даних.

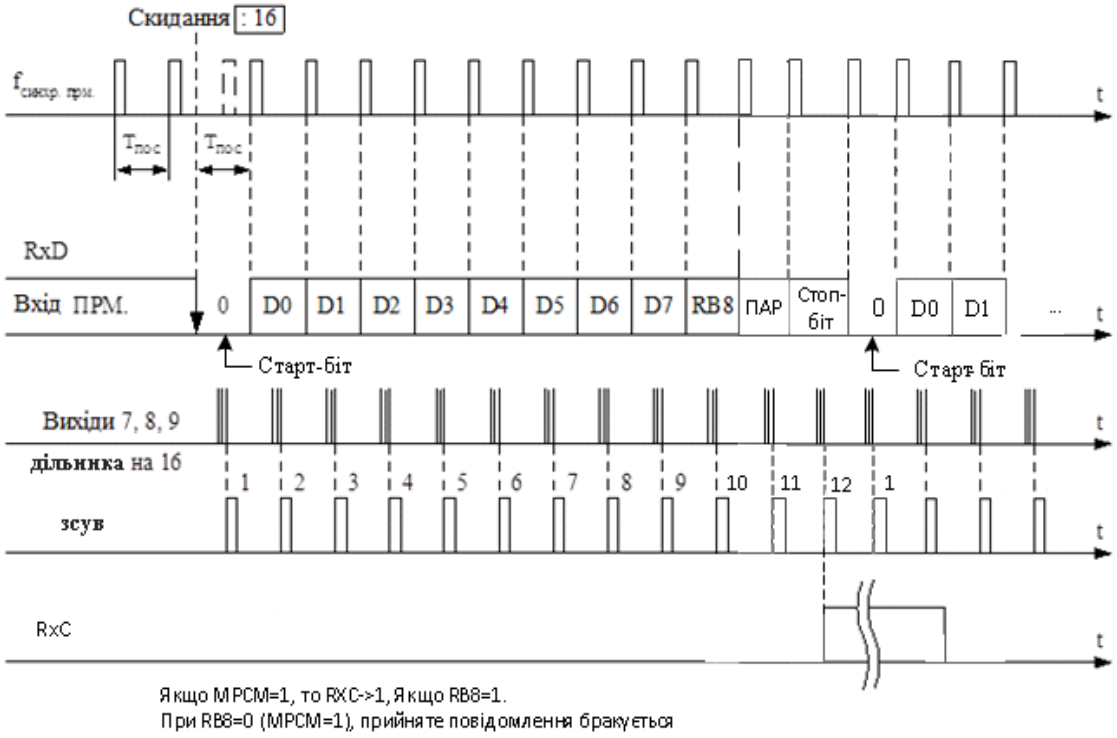
Можливі два варіанти реакції на запис даних у регістр UDR:

– якщо запис у регістр здійснюється в той момент, коли передавач знаходиться у стані очікування (попередні дані вже передано), то у цьому випадку дані пересилаються у регістр зсуву одразу ж після запису в регістр UDR;

– якщо запис у регістр UDR здійснюється під час передачі, то в цьому випадку дані пересилаються у регістр зсуву після передачі останнього стоп-біта поточного кадру.



а



б

Рис. 6.4. Часові діаграми роботи інтерфейсу в асинхронному режимі: а – передача; б – прийом даних

Після пересилання слова даних у регістр зсуву, прапорець UDRE (UDREn) регістра UCSRA (UCSRnA) встановлюється в одиницю, що означає готовність передавача до отримання нового байта даних. У цьому стані прапорець залишається до наступного запису в буфер. Одночасно з пересиланням у регістрі зсуву формується службова інформація – старт-біт, можливий біт парності (тільки в USART), а також один або два стоп-біти.

Після завантаження регістра зсуву його вміст починає зсуватися вправо і поступати на вивід TXD (TXDn) у порядку, розглянутому в підрозділі 6.5. Швидкість зсуву визначається налаштуванням контролера тактових сигналів. При роботі в синхронному режимі зміна стану виводу TXD (TXDn) відбувається за одним з фронтів сигналу XCK (XCKn). Якщо розряд UCPOL (UCPOLn) регістра UCSRC (UCSRnC) скинуто в нуль, зміна стану виводу відбувається за наростаючим фронтом сигналу XCK (XCKn), якщо ж встановлений в одиницю – за спадаючим фронтом сигналу, як показано на рис. 6.5.

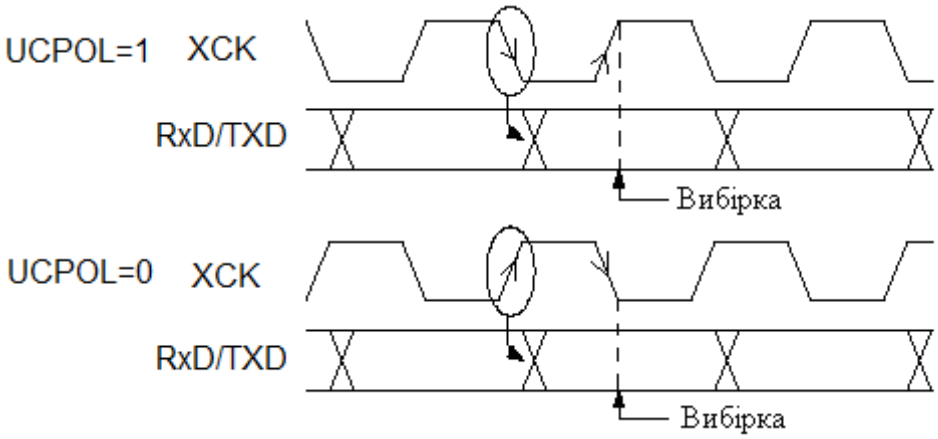


Рис. 6.5. Часові діаграми для синхронного режиму роботи модуля УСАПП

Якщо під час передачі попереднього слова у регістр UDR було записано нове слово даних, то після передачі останнього стоп-біта попереднього слова в регістр зсуву пересилається нове слово. Якщо ж до моменту закінчення передачі кадру нового запису виконано не було, то встановлюється прапорець переривання TXC (TXCn) регістра UCSRA (UCSRnA) – «Передачу завершено». Скидання прапорця здійснюється апаратно при вході у підпрограму обробки переривання або програмно, записом в цей розряд одиниці.

Відключення передавача здійснюється скиданням розряду TXEN (TXENn) регістра UCSRB (UCSRnB). Якщо у момент виконання цієї команди здійснювалася передача, скидання розряду відбудеться тільки після завершення поточної та відкладеної передач, тобто після очищення буферного регістра та регістра зсуву передавача. При вимкненому передавачеві вивід TXD (TXDn) може використовуватися, як лінія введення/виведення загального призначення.

Нижче наведено приклади підпрограм передачі інтерфейсом УАПП/УСАПП. Ці підпрограми чекають на очищення буфера передавача, а потім завантажують в нього нове значення.

Приклад на асемблері

```
USART_Transmit:
;Чекання очищення буфера передавача
sbis UCSRA,UDRE; Пропуск наступної команди, якщо біт UDRE
;встановлено
rjmp USART_Transmit; Перехід на мітку USART_Transmit
;Копіювання 9-го розряду даних з R17.0 в TXB8
cbi UCSRB,TXB8; Скидання біта TXB8 у UCSRB
sbrc Rl7,0; Пропуск наступної команди, якщо нульовий біт у
;регістрі Rl7 скинуто
sbi UCSRB,TXB8;Встановлення біта TXB8 у UCSRB
;Завантаження байта даних у буфер, початок передачи
out UDR,Rl6;Запис Rl6 (байт для передачі) у регістр UDR
ret
```

Приклад на С

```
void USART_Transmit(unsigned int data)
{
     /* чекання очищення буфера передавача */
     while ( !( UCSRA & (1«UDRE)))
     /* копіювання 9-го розряду даних з R17.0 в TXB8 */
     UCSRB &= ~(1<<TXB8);
     if (data & 0x01)
     UCSRB |= (1<<TXB8) ;
     /* завантаження байта даних у буфер, початок передачи */
     UDR = data;
}
```

6.7. Прийом даних

Для дозволу робота приймача треба встановити розряд RXEN (RXENn) регістра UCSRB (UCSRnB) [1; 7]. Після цього вивід RXD (RXDn) підключається до приймача УАПП/УСАПП і починає функціонувати як вхід, незалежно від значень регістра керування портом. Якщо використовується синхронний режим роботи, треба програмно перевизначити функціонування виводу ХСК (ХСКn).

Прийом даних починається після виявлення приймачем коректного старт-біта (див. вище рис. 6.4*б*). Потім кожен розряд кадру зчитується з частотою, яка визначається контролером швидкості передачі або тактовим сигналом ХСК (ХСКn). Отримані розряди даних послідовно розміщуються у регістр зсуву приймача до виявлення першого стоп-біта кадру. Після цього вміст регістра зсуву пересилається у буфер приймача, з якого прийняте значення може бути зчитано. При використанні 9-розрядних слів даних значення старшого восьмого розряду,

оскільки нумерація розрядів ведеться від нуля, може бути визначене за станом прапорця RXB8 (RXB8n) регістра UCSRB (UCSRnB). У модулях USART вміст цього розряду має бути зчитаний до звернення до регістра даних. Це пов'язано з тим, що прапорець RXB8 (RXB8n) відображає значення старшого розряду слова даних кадру, який знаходиться на верхньому рівні буфера приймача, стан якого при читанні регістра даних зміниться.

Якщо було запрограмовано контроль парності (тільки УСАПП), то схема контролю парності обчислює біт парності для всіх розрядів прийнятого слова даних і порівнює його з прийнятим бітом парності. Результат перевірки запам'ятовується у буфері приймача разом з прийнятим словом даних і стоп-бітами. Наявність або відсутність помилки контролю парності може бути потім визначено за станом прапорця UPE (UPEn) регістра UCSRA (UCSRnA). Цей прапорець встановлюється в одиницю якщо наступне слово, яке може бути прочитане з буфера, має помилку контролю парності. При вимкненому контролі парності прапорець UPE (UPEn) завжди читається як нуль.

У регістрі UCSRA(UCSRnA) блоку приймача модулів УАПП/УСАПП є ще два прапорці, які показують стан обміну: прапорець помилки кадрування FE (FEn) і прапорець переповнення DOR (DORn)/OR (Orn) [1; 7]. Прапорець FE (Fen) встановлюється в одиницю, якщо значення першого стоп-біта прийнятого кадру не відповідає потрібному, тобто дорівнює нулю.

Прапорці DOR (DORn) в УСАПП та OR (Orn) в УАПП відображають втрату даних через переповнення буфера приймача. В УАПП прапорець встановлюється в одиницю, якщо до моменту закінчення прийому кадру (заповнення регістра зсуву приймача) дані попереднього кадру не були зчитані з регістра даних. В УСАПП прапорець встановлюється в одиницю у разі прийому старт-біта нового кадру при заповнених буфері та регістрі зсуву приймача. Встановлений прапорець DOR (DORn)/OR (Orn) означає, що між минулим байтом, який зчитано з регістра UDR, та байтом, зчитаним у даний момент, відбулася втрата одного або декількох кадрів.

Обробка прапорців в модулях УАПП/УСАПП дещо відрізняється.

У модулях УАПП прапорець помилки кадрування FE (Fen) має бути прочитаний перед зверненням до регістра даних, а прапорець переповнення OR (Orn) – після звернення до цього регістра.

У модулях УСАПП всі прапорці помилок буферизуються разом із словом даних, тобто відповідні розряди регістра UCSRA (UCSRnA) відносяться до кадру, слово даних якого буде прочитано при наступному зверненні до регістра даних UDR (UDRn). Тому стан цих прапорців треба зчитати перед зверненням до регістра даних.

Для індикації стану приймача в модулях УАПП/УСАПП використовується прапорець переривання «Прийом завершено» RXC (RXCn) регістра UCSRA (UCSRnA). Цей прапорець встановлюється в одиницю за наявності в буфері приймача не прочитаних даних. У модулях УАПП цей прапорець скидається після читання регістра даних, а у модулях УСАПП – при звільненні буфера (після зчитування всіх даних, які знаходяться у ньому).

Вимкнення приймача здійснюється скиданням розряду RXEN (RXENn) регістра UCSRB (UCSRnB). На відміну від передавача, приймач вимикається відразу ж після скидання розряду, тобто кадр, який приймається у цей момент, втрачається. При вимкненні приймача очищується його буфер, тобто втрачаються також всі непрочитані дані. При вимкненому приймачі вивід RXD (RXDn) може використовуватися як лінія введення/виведення загального призначення.

Нижче наведено приклад підпрограми прийому інтерфейсом УСАПП кадру з дев'ятьма бітами даних. Як і в попередньому прикладі, тут використовується опитування прапорця переривання.

Приклад на асемблері

```
USART_Receive:
;Чекання завантаження даних у буфер приймача
sbis UCSRA,RXC; Пропуск наступної команди, якщо біт RXC в
регістрі UCSRA ;встановлено
rjmp USART_Receive; Перехід на мітку USART_Receive
;Читання прапорців стану та 9-го розряду даних
in rl8,UCSRA;Передача регістра UCSRA в rl8
in rl7,UCSRB; Читання регістра UCSRB в rl7
;Читання молодшого байта даних
in rl6,UDR; Читання UDR в rl6
;У випадку помилки повернути -1
andi rl8, (1«FE) | (1«DOR) | (1«UPE)
breq no_error;Перехід на мітку: no_error, якщо прапорець Z = 1
ldi rl7,HIGH(-l)
ldi rl8,L0W(-l)
no_error:
;Виділення 9-го розряду даних
lsr rl7;Логічний зсув rl7 вправо
andi rl7,0x01;Логічне I rl7 з константою 0x01
ret
```

Приклад на С

```
unsigned int USART_Receive(void)
{
```

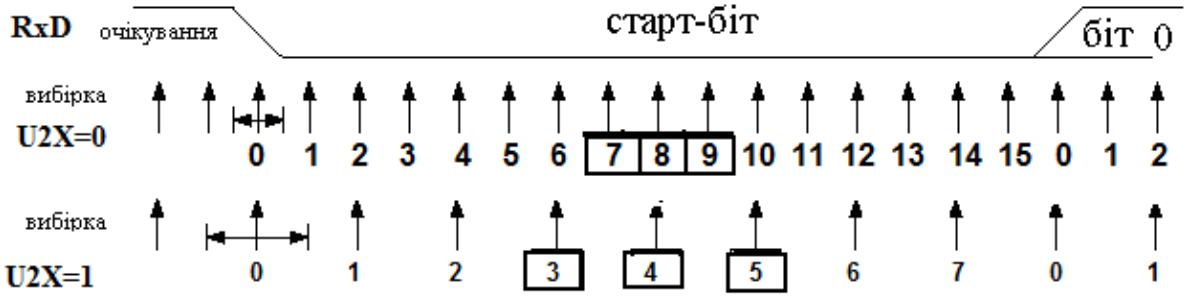
```
        unsigned char status, resh, resl
        /* Чекання заповнення буфера приймача */
        while (! (UCSRA & (1«RXC) ) )
        /* Читання 9-го розряду даних та прапорців стану */
        status = UCSRA;
        resh = UCSRB;
        /* Читання молодшого байта даних */
        resl = UDR;
        /* У випадку помилки повернути -1*/
        if (status & (1«FE) | (1«DOR) | (1«UPE) )
        return -1;
        /* Виділення 9-го розряду даних */
        resh = (resh»l) & 0x01;
        return ( (resh<<8) | resl);
}
```

Прийом всіх розрядів кадру здійснюється по-різному, залежно від режиму роботи модуля.

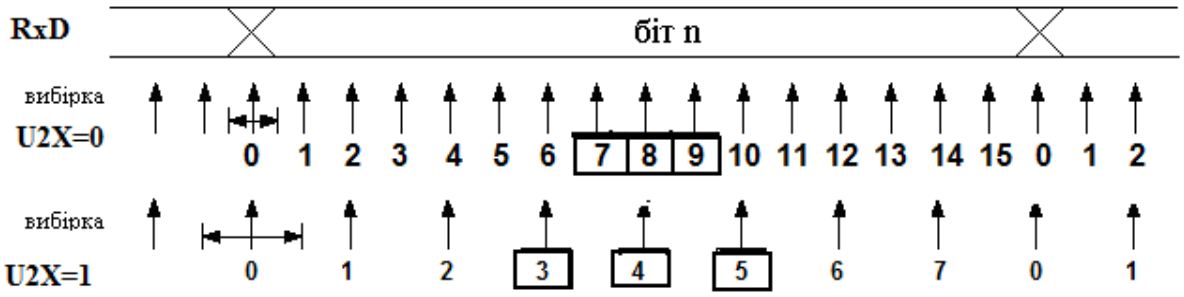
При роботі модуля УСАПП у синхронному режимі стан виводу RXD (RXDn) читається за одним з фронтів сигналу ХСК (ХСКn). Якщо розряд UCPOL (UCPOLn) регістра UCSRC (UCSRnC) скинуто в нуль, зчитування стану виводу RXD відбувається за спадаючим фронтом сигналу ХСК (ХСКn), якщо ж встановлено в одиницю – за наростаючим фронтом сигналу. Іншими словами, зчитування даних з виводу RXD (RXDn) і їх видача на вивід TXD (TXDn) відбуваються за протилежними фронтами (див. вище рис. 6.5).

При прийомі в асинхронному режимі роботи використовуються схеми відновлення тактового сигналу і даних. Схема відновлення тактового сигналу призначена для синхронізації внутрішнього тактового сигналу, який формується контролером швидкості передачі, і кадрів, які поступають на вивід RXD (RXDn) мікроконтролера. Схема відновлення даних здійснює читання та фільтрацію кожного розряду кадру, що приймається.

Схема відновлення тактового сигналу опитує вхід приймача з метою визначення старт-біта кадру (рис. 6.6).



а



б

Рис. 6.6. Розпізнавання розрядів кадру: а – старт-біт; б – інші розряди

Частота опитування залежить від стану розряду U2X (U2Xn) регістра UCSRA (UCSRnA). У звичайному режимі (при U2Xn = 0) частота опитування у 16 разів перевищує швидкість передачі даних, а у прискореному режимі (при U2X = 1) – у 8 разів. Горизонтальні стрілки на рис. 6.6 ілюструють можливий відхід синхронізації у процесі опитування. Більш високу розсинхронізацію у часі має прискорений обмін (біт U2X = 1).

Виявлення в процесі очікування зміни сигналу на виводі RXD (RXDn) з одиниці на нуль інтерпретується як можлива поява переднього фронту старт-біта. Коли виявлено цю зміну, скидається лічильник-дільник на 16 у ланцюзі формування сигналу «Синхр. RXD». В результаті цього відбувається суміщення моментів переповнення цього лічильника-дільника з межами зміни на вході RXD бітів кадру, що приймається. Шістнадцять станів лічильника-дільника ділять час, протягом якого кожен біт посилки, що приймається, присутній на вході RXD, на 16 фаз (вибірок), з 0-ї по 15-у для кожного біта. У нормальному режимі перевіряється значення 7-ої, 8-ої і 9-ої вибірок вхідного сигналу, а в прискореному режимі – 3-ої, 4-ої і 5-ої вибірок (рис. 6.6*а*). Якщо значення хоча б двох вибірок із вказаних дорівнює одиниці, старт-біт вважається помилковим (завада), а приймач переходить до очікування наступної зміни вхідного сигналу з одиниці на нуль.

Інакше вважається, що виявлено старт-біт нової послідовності даних, з яким синхронізується внутрішній тактовий сигнал приймача. Після цього починає працювати схема відновлення даних.

Рішення про значення кожного прийнятого розряду також ухвалюється за результатами 7-ої, 8-ої і 9-ої (3-ої, 4-ої і 5-ої) вибірок вхідного сигналу (рис. 6.6*б*). Станом розряду вважається логічне значення, яке було отримане щонайменше у двох з трьох вибірок. Процес розпізнавання повторюється для всіх розрядів кадру, що приймається, включаючи перший стоп-біт.

Таким чином, старт-біт нового кадру може передаватись відразу ж після останньої вибірки, яка використовується для визначення значення першого стоп-біта. У звичайному режимі роботи формування старт-біта може початися в момент А, а в прискореному режимі – в момент В (рис. 6.7). Момент С на рисунку визначає максимальну тривалість стоп-біта. Використання більш раннього виявлення старт-біта впливає на робочій діапазон синхрочастот приймача (допустиме розходження синхрочастоти передавача та приймача при фіксованій довжині посилки) [1; 7].

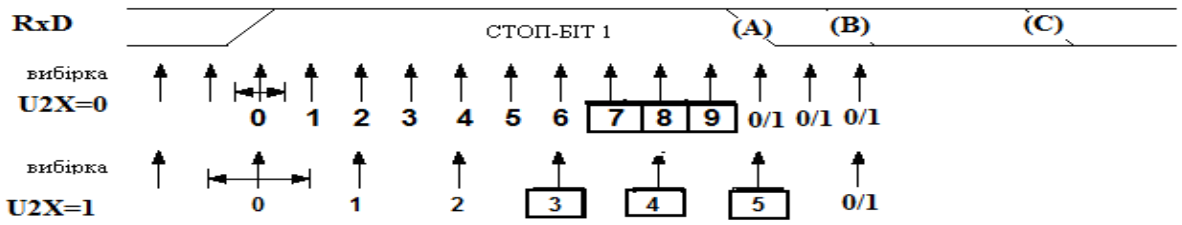


Рис. 6.7. Розпізнавання стоп-біта та наступного старт-біта

6.8. Обмін даними через інтерфейс УСАПП у мікроконтролерній мережі

У мікроконтролерних мережах відбувається обмін інформацією між одним ведучим та одним з декількох ведених мікроконтролерів. Кожен ведений мікроконтролер мережі має свою унікальну адресу. Якщо ведучий мікроконтролер хоче що-небудь передати, то він посилає адресний байт, який визначає, до якого з ведених мікроконтролерів він збирається звернутися. Коли один з ведених мікроконтролерів розпізнав свою адресу, він переходить у режим прийому даних і відповідно приймає подальші байти як дані. Решта ведених мікроконтролерів ігнорують байти даних, що приймаються, до посилки ведучим нового адресного байта. Програмування режиму фільтрації кадрів даних, що приймаються і не містять адреси, здійснюється встановленням в одиницю розряду MPCM (MPCMn) регістра UCSRA (UCSRnA) [1; 7].

У мікроконтролері, який виконує роль ведучого, треба встановити режим передачі 9-розрядних даних. При передачі адресного байта старший – 8-й розряд, за умови, що перший (молодший) розряд при двійковому кодуванні має номер – нуль, повинен встановлюватися в одиницю, а при передачі байтів даних він повинен скидатися в нуль.

У ведених мікроконтролерів механізм прийому залежить від режиму роботи приймача. При прийомі 9-розрядних даних ідентифікація вмісту кадру здійснюється за старшим дев`ятим розрядом слова даних. Якщо вказаний розряд встановлено в одиницю, то кадр містить адресу, якщо скинуто в нуль – кадр містить дані.

Для програмування обміну даними у мікроконтролерній мережі виконуються наступні дії:

1) У всіх ведених мікроконтролерів встановлюється в одиницю розряд MPCM (MPCMn) регістра UCSRA (UCSRnA).

2) Всі мікроконтролери програмуються в режим передачі дев`ятирозрядних кадрів.

3) Ведучий мікроконтролер посилає адресний кадр з TB8 = 1, а всі ведені мікроконтролери його приймають. У кожному з ведених мікроконтролерів встановлюється прапорець RXC (RXCn) регістра UCSRA (UCSRnA).

4) Кожен ведений мікроконтролер читає вміст регістра даних (адресний кадр) та порівнює його із своєю мережевою адресою. Мікроконтролер, адреса якого співпала з адресою, посланою ведучим, скидає в нуль розряд MPCM (MPCMn).

5) Адресований мікроконтролер починає приймати кадри, що містять дані, оскільки їх дев`ятий розряд дорівнює нулю. У решті не адресованих ведених мікроконтролерів розряд MPCM (MPCMn) залишається встановленим в одиницю, через це кадри даних з бітом TB8 = 0 будуть ними ігноруватися.

6) Після прийому останнього байта даних адресований мікроконтролер встановлює в одиницю розряд MPCM (MPCMn) і знову чекає прихід кадру з адресою. Процес повторюється з пункту 3.

6.9. Робота модуля УСАПП в режимі MSPI

В ряді мікроконтролерів, наприклад, в ATmega48x/88x/168x, ATmega164x/324x/644x і ATmega640x/1280x/1231x/ 2560x/2561x, модулі USART можуть працювати в якості ведучого шини SPI (див. розділ 8). В цьому режимі, який називається MSPI (Master SPI Mode), модулі УСАПП підтримують усі чотири режими SPI (0, 1, 2 і 3) [7]. При цьому доступні як подвійна буферизація

даних при прийомі і передачі, так і більшість переривань модуля. Функції виводів модуля УСАПП в режимі MSPI вказано у табл. 6.6.

Таблиця 6.6. **Виводи УСАПП в режимі MSPI**

УСАПП	SPI	Опис
TXDn	MOSI	Тільки вихід ведучого
RXDn	MISO	Тільки вхід ведучого
XCKn	SCK	Вихід тактового сигналу
–	$\overline{SS}$	Не підтримується

Напрям передачі даних для виводу XCKn необхідно задавати програмно (відповідний розряд регістра DDRx повинен бути встановлений в одиницю). Налаштування цього виводу рекомендується робити до дозволу роботи приймача і передавача USART.

6.10. Розрахунок швидкості передачі інформації, тривалості одного біта та часу передачі одного байта

Нижче наведено приклад розрахунку швидкості передачі інформації та тривалості одного біта для асинхронного режиму роботи (біт U2Xn = 0). У цьому випадку швидкість передачі даних (обміну):

$$V_{ПД} = \frac{f_{CLK}}{16 \cdot (UBRR + 1)},$$

де f_{CLK} – частота системного тактового генератора,

UBRR – вміст регістра контролера швидкості передачі, що змінюється програмно (UBRR = 0...4095).

Наприклад, якщо f_{CLK} = 16 МГц, а UBRR = 24, тоді:

$$V_{ПД} = \frac{16000000}{16 \cdot 25} = 40000 \ \ посилок/секунду = 40000 \ \ біт/секунду = 40 Кбіт/с.$$

Відповідно тривалість одного біта (однієї посилки):

$$t_{ПОС} = \frac{1}{V_{ПД}} = \frac{1000000}{40000} = 25 \ \ мксек.$$

Часові діаграми роботи цього модуля для асинхронного режиму наведено вище на рис. 6.4.

У лінію зв'язку передаються наступні біти: старт-біт, 8 інформаційних біт, 9-й службовий біт TB8, додатковий перевірочний біт на парність і один стоп-біт.

Тривалість цих 12 посилок однакова і при $V_{ПД} = 40$ Кбіт/с дорівнює $t_{ПОС} = 25$ мксек.

Оскільки передавач містить буферний регістр даних UDR, то в нього програмно завантажується черговий байт для передачі поки з регістра зсуву видаються посилки попереднього байта. Після передачі стоп-біта новий байт з регістра UDR пересилається в регістр зсуву та ініціюється новий цикл передачі.

Таким чином, час передачі одного байта дорівнює $12*t_{посилок}$, що при тривалості $t_{ПОС} = 25$ мксек становить $t_Б = 25*12 = 300$ мксек.

6.11. Моделювання модуля УСАПП у пакеті PROTEUS

6.11.1. Опис моделі

Робочу модель дослідження універсального синхронно/асинхронного приймача-передавача (УСАПП) у пакеті PROTEUS 8.6 наведено на рис. 6.8.

Зліва в моделі наведено вісім кнопок, за допомогою яких можна задавати байт (8 біт) даних, що буде передаватися через модуль УСАПП.

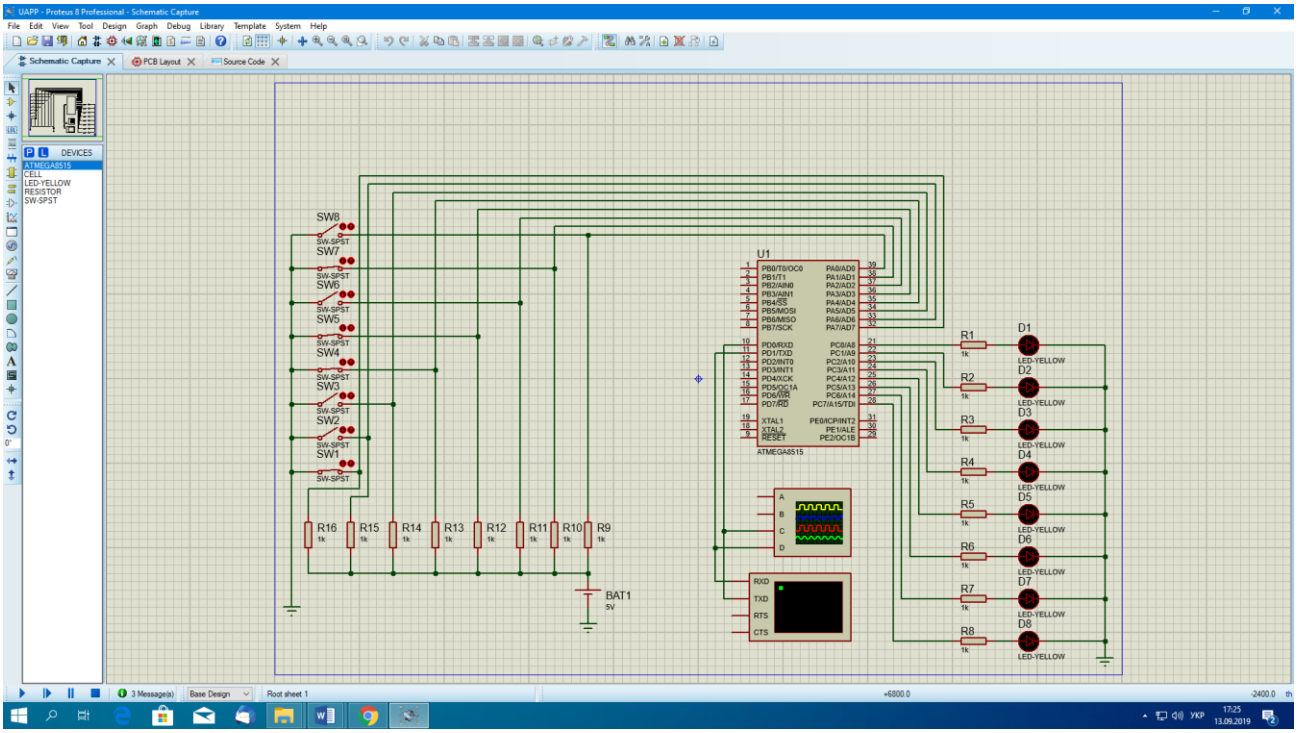


Рис. 6.8. Схема робочої моделі УСАПП

Кнопки підключено до восьми ліній порту А мікроконтролера ATmega8515: PA.0, PA.1, … , PA.7. Якщо якась із кнопок відпущена, то через резистори R9...R16 на відповідну лінію порту PA подається одиниця. Якщо кнопка нажата, то вводиться нуль. Праворуч зображено вісім світлодіодів D1…D8, які підключено до ліній порту С: PC.0, PC.1, … , PC.7. Катоди світлодіодів підключено до спільного дроту (землі), а аноди через резистори R1…R8, які обмежують струм, підключено до виходів порту С. Коли треба засвітити світлодіод на відповідний вихід порту С виводиться логічна одиниця.

До ліній TxD – вихід передавача УАПП та RxD – вхід приймача підключено осцилограф, та віртуальний термінал (Virtual Terminal), які вибрано з віртуальних інструментів програми Proteus 8.6. На них будуть відображатися повідомлення, які вводяться та виводяться через модуль УСАПП в/з мікроконтролера.

До виводів XTAL1 та XTAL2 в реальних схемах підключають кварцовий резонатор з частотою, яка визначає тактову частоту генератора мікроконтролера. В нашому прикладі вона дорівнює f_{BQ} = 8МГц. Для підвищення стабільності роботи системного генератора також підключають конденсатори С1 та С2, ємністю 30пФ (див. нижче рис. 13.2). Резонатор та конденсатори потрібні в практичній схемі. В модель Proteus їх можна не вводити.

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

При моделюванні треба двічі лівою кнопкою миші клацнути на мікроконтролері та в опції Clock Frequency вказати значення частоти генератора.

6.11.2. Порядок моделювання

1. Перш за все, потрібно налаштувати Terminal. Для цього треба двічі клацнути на нього лівою кнопкою миші, та виставити відповідні налаштування. Якщо треба використовувати перевірку на парність або непарність, то це треба вказати в опції Parity. Нижче наведено приклад для випадку, коли потрібно передавати зі швидкістю 9600 біт/сек: 8 біт даних та один стоп-біт з перевіркою на непарність (рис. 6.9).

2. Запускаємо процес моделювання, натискаючи на кнопку Start VSM Debugging (рис. 6.10).

3. Натискаємо на кнопку Pause VSM Debugging, переходимо на вкладку AVR Source Code та ставимо дві точки зупинки. Для цього потрібно лівою кнопкою миші двічі клацнути з лівої сторони двох команд, як показано на рис. 6.11.

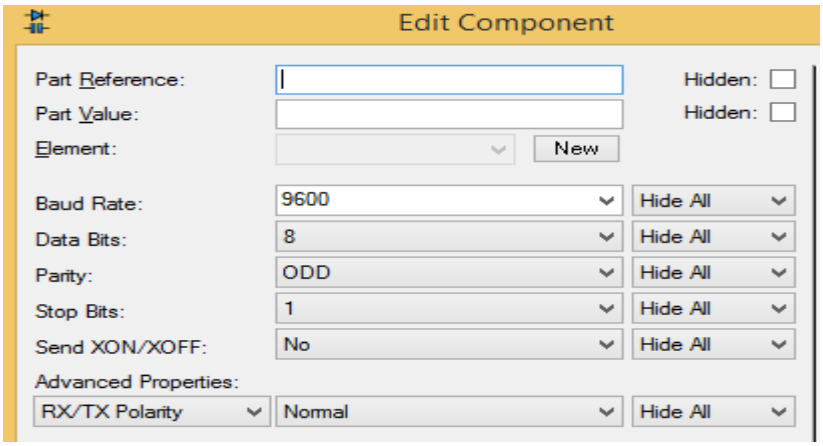


Рис. 6.9. Налаштування термінала

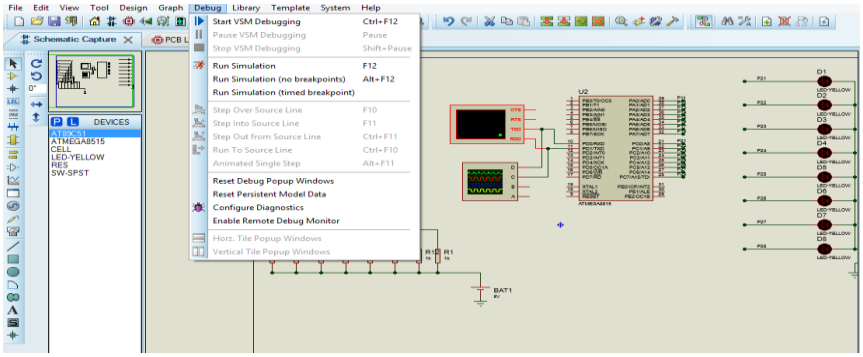


Рис. 6.10. Запуск процесу моделювання

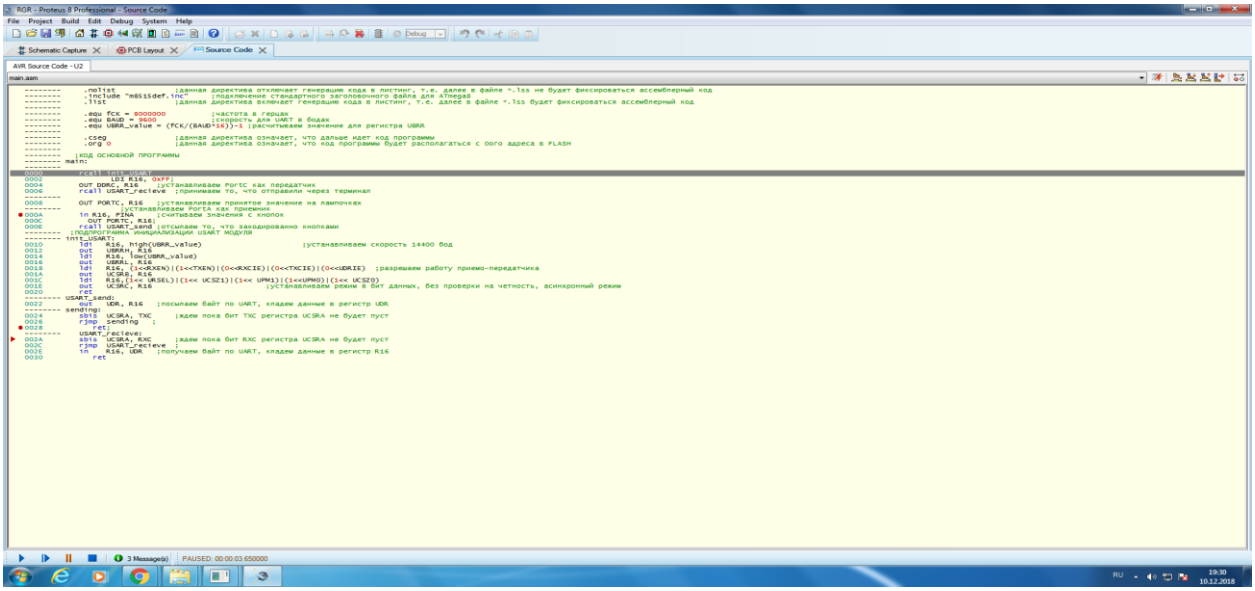
4. 

Рис. 6.11. Налаштування двох точок зупинки

4. Тиснемо на кнопку Run Simulation (F12).

5. Переходимо до вікна Virtual Terminal, тиснемо правою кнопкою миші на вікні, та обираємо Hex Display Mode та Echo Typed Characters (рис. 6.12).

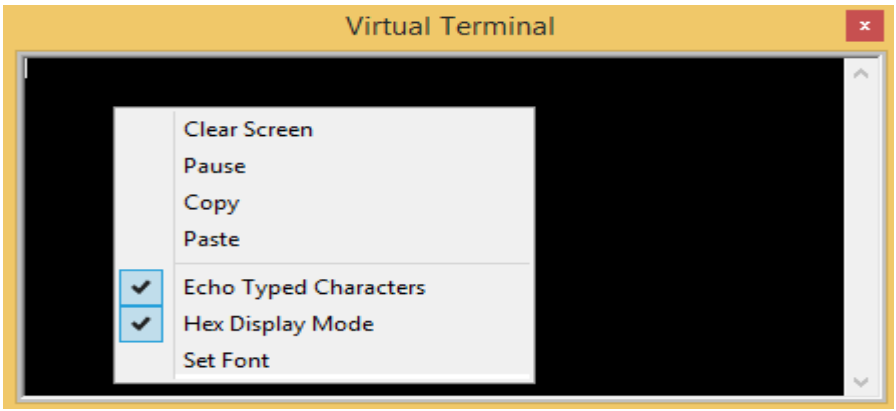


Рис. 6.12. Налаштування Virtual Terminal

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

6. Вводимо символ, який хочемо надіслати від віртуального термінала до мікроконтролера через УСАПП. При введенні символу у вікні терміналу повинен стояти курсор. Для того, щоб побачити сигнал на осцилографі, потрібно натиснути кнопку на опції One-Shot (один кадр), як показано на рис. 6.13.

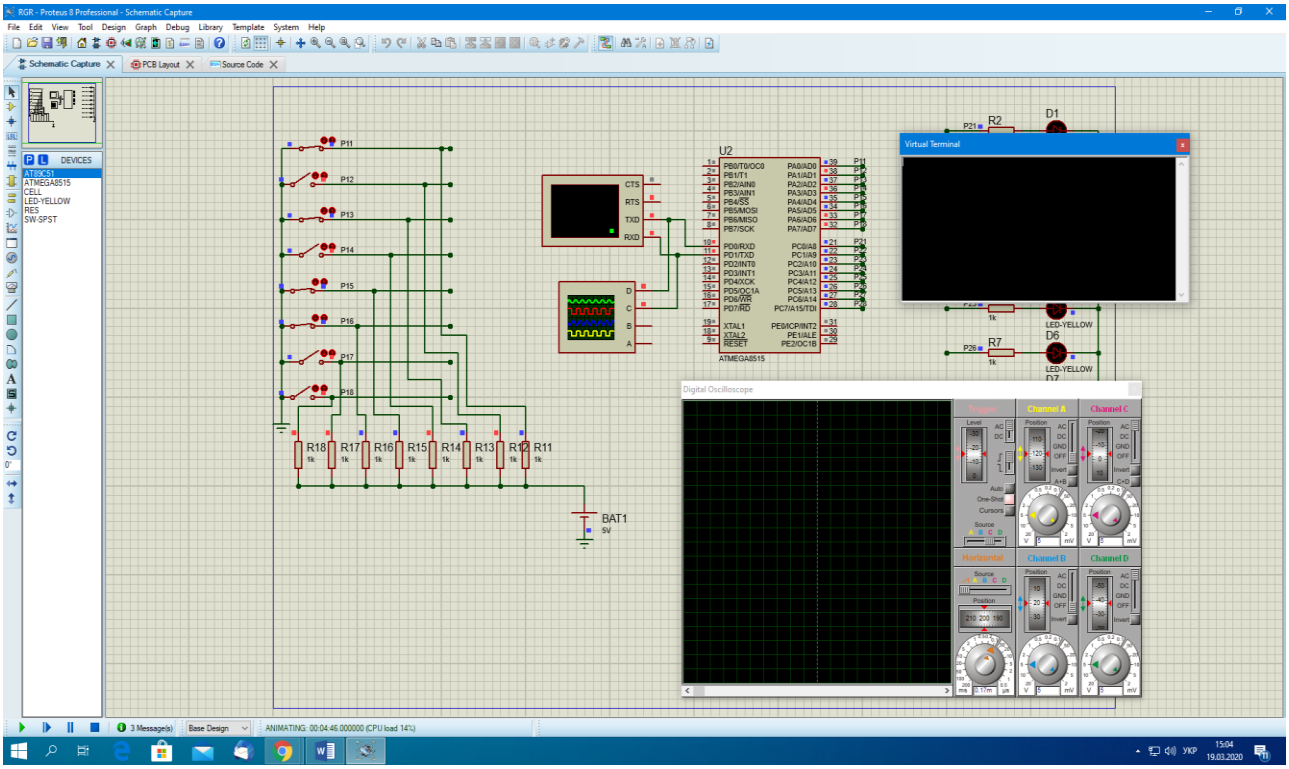


Рис. 6.13. Налаштування віртуального термінала та осцилографа

Символ, який ми вводимо з клавіатури, віртуальний термінал перетворює в ASCII-код згідно таблиці, яку наведено у [3].

Вводимо, наприклад, цифру 7, ASCII-код якої 37 (hex) = 00110111 (bin). Програма перейде на рядок, де ми поставили першу точку зупинки. Переходимо на вкладку зі схемою Schematic Capture. Звертаємо увагу на світлодіоди, які стоять на виході порту С (рис. 6.14).

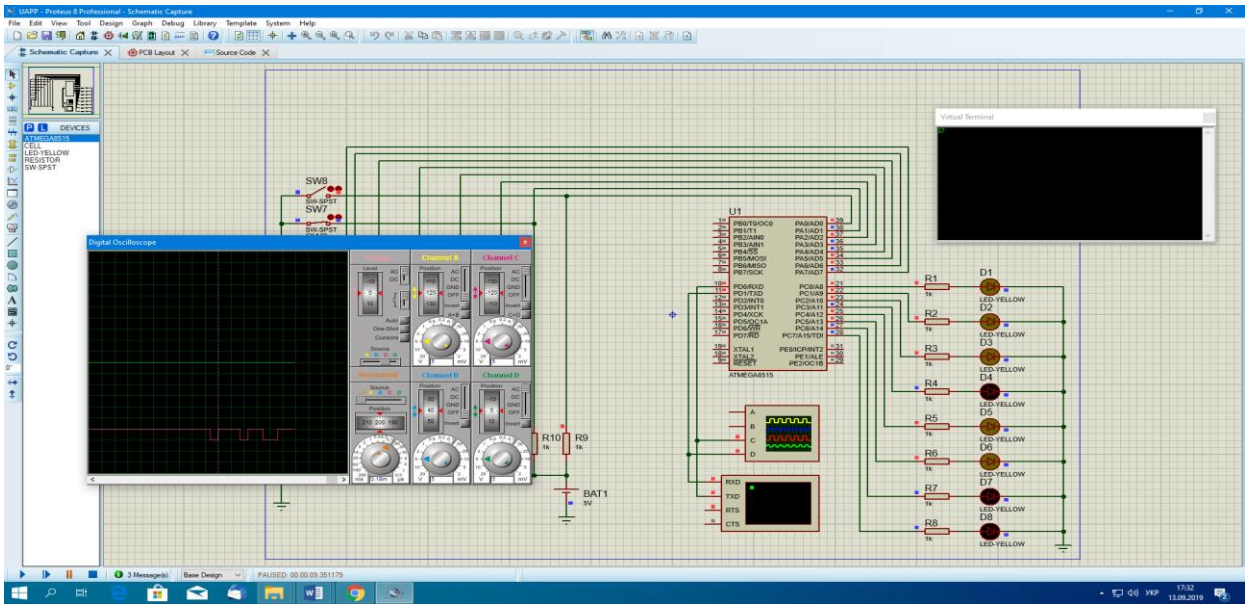


Рис. 6.14. Відображення на світлодіодах та осцилографі символу, який прийнято від термінала

Світлодіоди точно відображають бінарний код символу, який ми прийняли від термінала (якщо дивитися знизу уверх, а бінарне число читати зліва на право), нуль – світлодіод вимкнено, одиниця – світлодіод увімкнено (прийнято символ 00110111в = 37h). Нижній (червоний) луч осцилографа також вірно відображає прийнятий символ, який було передано від термінала в асинхронному старт-стопному режимі.

7. Перевіримо, як мікроконтролер через УСАПП відправить символ, який закодували кнопками: замкнута кнопка – нуль, а розімкнута – одиниця (рис. 6.15).

Якщо дивитися на кнопки знизу уверх, то на рис. 6.15 набрано число 01100101 (bin) = 65 (hex).

Для того, щоб побачити переданий сигнал на осцилографі, потрібно натиснути кнопку на опції One-Shot (один кадр), як показано вище на рис. 6.13.

8. Натиснемо Run Simulation для продовження виконання програми. Програма дійде до другої точки зупину.

9. Розберемо більш детально сигнали, що відображаються на осцилографі (рис. 6.15).

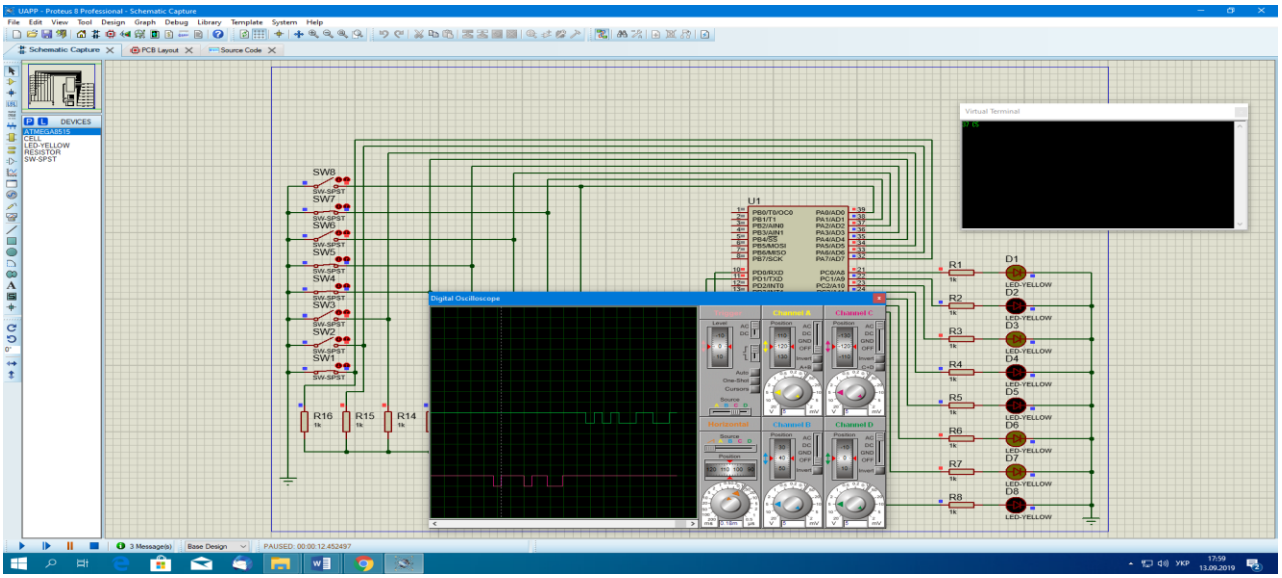


Рис. 6.15. Відображення сигналів на осцилографі: від терміналу – зелений; від МК (кнопок) – червоний

Осцилограф відображає сигнали за двома променями: верхній (зелений) – сигнал, який передається від МК (кнопок), нижній (червоний) – сигнал, який приймається від терміналу: 37(hex).

Розглянемо сигнал, який передається від МК (кнопок).

Перший перепад із високого рівня у низький – старт-біт.

Потім ідуть вісім інформаційних бітів, починаючи з молодшого розряду, які будуть відображені на віртуальному терміналі та осцилографі, як 01100101(bin) = 65 (hex). Далі іде 9-й біт, який дорівнює нулю, тому що число одиниць в байті, який було передано – парне (розряд P в регістрі прапорців мікроконторолера дорівнює нулю). Останній 10-й біт, який дорівнює одиниці, це стоп-біт.

10. Визначимо отриману швидкість передачі, яку було обрано 9600 біт/сек = 9600 пос/сек = 9600 бод. Для цього підберемо таку розгортку сигналу на осцилографі, щоб сумістити бічні сторони імпульсу із вертикальними лініями сітки осцилографа. Для швидкості передачі 9600 біт/сек отримаємо наступну картину (рис. 6.16).

Як ми бачимо, тривалість одного імпульсу дорівнює 0,104 мс. Якщо розділити один біт на цей час, та помножити на 1000, отримаємо задану швидкість:

$$V = \frac{1}{0.104} * 1000 = 9600\ \frac{\text{біт}}{\text{сек}}.$$

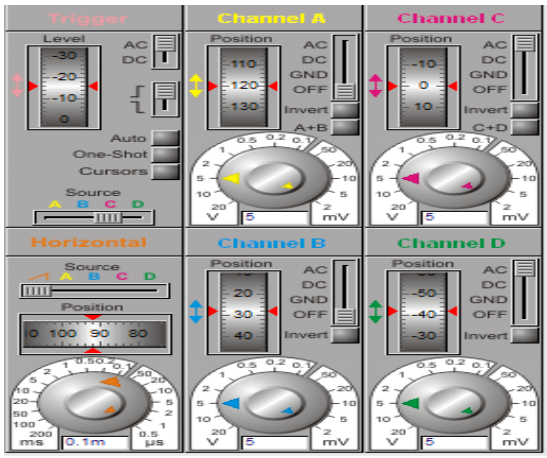


Рис. 6.16. Визначення отриманої швидкості передачі на осцилографі

Нижче розглянуто ще один приклад (рис. 6.17, 6.18), коли на кнопках було набрано число 67(hex) = 01100111(bin). На верхньому зеленому промені осцилографа побачимо: перший нуль – старт-біт, потім 3 одиниці та один 0 – це число 7 на кнопках (молодша тетрада байта), далі йде нуль, дві одиниці та нуль – це число 6 на кнопках (старша тетрада), а потім на осцилографі бачимо 1 – це біт доповнення до парності, та наступна одиниця – стоп-біт.

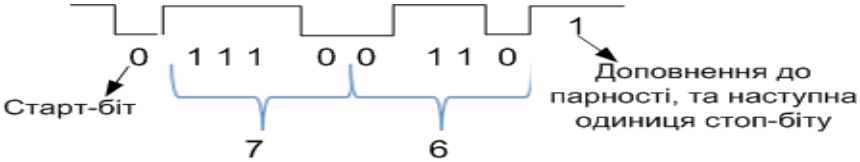


Рис. 6.17. Передача числа: 67(hex) = 01100111(bin)

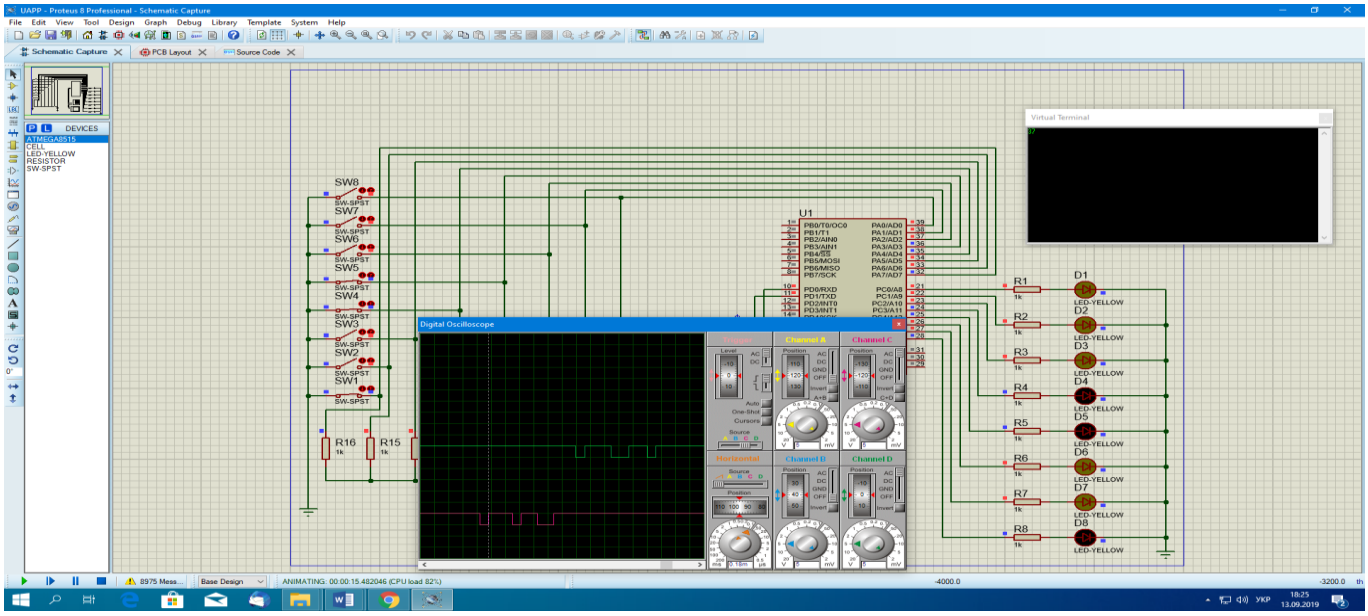


Рис. 6.18. Відображення сигналів на осцилографі: від терміналу – червоний, від МК (кнопок) – зелений

6.11.3. Схема алгоритму роботи

Схему алгоритму роботи моделі наведено на рис. 6.19.

6.11.4. Робоча програма

Нижче представлено лістинг робочої програми мовою Асемблер, яку розроблено згідно з алгоритмом, який наведено на рис. 6.19. В дужках праворуч біля кожної команди цифрою відмічено номер блоку схеми алгоритму, який реалізує ця команда.

Рис. 6.19. Схема алгоритму роботи моделі

Робоча програма мовою Асемблер

```
.nolist                         ; директива відключає генерацію коду у лістинг,
        ; тобто далі у файлі *.lss не буде фіксуватися асемблерний код
.include "m8515def.inc"         ; підключення стандартного заголовного
                                ; файлу для ATmega8515
.list                           ; директива включає генерацію коду у лістинг,
        ; тобто далі у файлі *.lss буде фіксуватися асемблерний код
.equ fCK = 8000000                      ; частота в герцах
.equ BAUD = 9600                        ; швидкість для УСАПП у бодах
.equ UBRR_value = (fCK/(BAUD*16))-1 ; розрахунок значення для регістра
                                        ; UBRR
.cseg                   ; директива означає, що далі іде код програми
.org 0                  ; директива означає, що  код програми у FLASH буде
                        ; розміщено з нульової адреси
; КОД ОСНОВНОЇ ПРОГРАМИ

main:
; ініціалізації з мітки init_USART
rcall init_USART        ; (блок 2) відносний виклик підпрограми
; налаштування порту С на передачу
LDI R16, 0xFF           ; (блок 2) R16← 0xFF
OUT DDRC, R16           ; (блок 2) DDRC← R16
; (блок 3) отримуємо те, що відправив віртуальний термінал
rcall USART_recieve     ; (блок 3) відносний виклик підпрограми з мітки
                                ; USART_recieve
; (блок 4) виводимо отримане від терміналу на світлодіоди
OUT PORTC, R16          ; (блок 4) PORTC←R16
; (блок 5) налаштовуємо порт А як передавач
LDI R16, 0x00           ; (блок 5) R16←0x00
OUT DDRA, R16           ; (блок 5) DDRA← R16
                        ; (блок 6) читаємо стан кнопок
in R16, PINA            ; (блок 6) R16← PINA
; (блок 7) виводимо отримане від кнопок на світлодіоди
OUT PORTC, R16          ; (блок 7) PORTC←R16
; (блок 8)  підпрограма з міткою USART_send відправляє стан кнопок на
осцилограф та термінал
```

```
   rcall USART_send          ; (блок 8) відносний виклик підпрограми з мітки
                                     ; USART_ send

   ; (блок 1) ПІДПРОГРАМА ІНІЦІАЛІЗАЦІЇ УСАПП-МОДУЛЯ
   init_USART:
   ; програмуємо швидкість обміну 9600 бод
ldi   R16, high(UBRR_value)       ; (блок 1) R16←старший байт UBRR_value
out   UBRRH, R16                  ; (блок 1) UBRRH←R16
ldi   R16, low(UBRR_value)        ; (блок 1) R16←молодший байт UBRR_value
out   UBRRL, R16                  ; (блок 1) UBRRL←R16
   ; дозволяємо роботу приймача-передавача модуля USART
ldi   R16, (1<<RXEN)|(1<<TXEN)
out   UCSRB, R16                          ; (блок 1) UCSRB←R16
   ; програмуємо передачу 8 біт з перевіркою на парність та одним стоп-бітом
   ; в асинхроному режимі
ldi   R16, (1<< URSEL)|(1<<UPM1)|(1<<UPM0)|(1<< UCSZ1)|(1<< UCSZ0)
   out   UCSRC, R16                  ; (блок 1) UCSRC← R16
   ret                               ; (блок 1) повернення з підпрограми ініціалізації

   ; передача через модуль УСАПП
   USART_send:
   out   UDR, R16                            ; (блок 8) регістр даних UDR← R16
   sending:
   ; чекаємо завершення передачі
   sbis  UCSRA, TXC        ; (блок 8) якщо біт TXC = 1, то наступна
                           ; команда пропускається, інакше  – наступна команда
   rjmp  sending           ; блок 8) безумовний перехід на мітку sending
   ret                     ; (блок 8) повернення з підпрограми

   USART_recieve:
   sbis  UCSRA, RXC          ; (блок 3), якщо біт RXC = 1 (в буфері ПРМ є
   ; отриманий байт) , то пропустити наступну команду,
   ; інакше наступна команда
   rjmp  USART_recieve     ; (блок 3) безумовний перехід на мітку
                           ; USART_recieve
   in  R16, UDR            ; (блок 3) R16←UDR (завантажуємо в
                           ; регістр  R16 отриманий байт від термінала
   ret                     ; (блок 3) повернення з підпрограми
```

Контрольні запитання та завдання

1. Що таке УСАПП (USART) та УАПП (UART)?
2. З яких трьох частин складається структура модуля УСАПП/УАПП? Які елементи містить кожна з них?
3. Які регістри використовуються для керування модулями УАПП та УСАПП?
4. Чим в асинхронному режимі задається швидкість прийому та передачі даних?
5. Поясніть структурну схему блока синхронізації модуля УСАПП для мікроконтролера Mega 128.
6. Які режими синхронізації підтримує УСАПП?
7. Як визначається швидкість обміну в модулі УСАПП?
8. Як визначається розмір слова даних у модулях УСАПП?
9. Як працює схема контролю парності?
10. Як проходить передача даних в модулях УСАПП?
11. Як проходить прийом даних в модулях УСАПП?
12. Що таке режим мультиконтролерного обміну?
13. Опишіть послідовність дій для здійснення обміну даними у мікроконтролерній мережі.
14. Як визначити швидкість передачі даних для асинхронного звичайного режиму?
15. Як визначити тривалість передачі кадру для асинхронного прискоренного режиму?
16. Як в моделі підключено кнопки, які задають байт для передачі?
17. Як в моделі підключено світлодіоди, які відображають передану в моделі інформацію?
18. Як в моделі визначається тривалість переданого біта?
19. Опишіть формат та призначення розрядів регістра UCSRC.
20. Як в програмі відбувається налаштування порту А на передачу?
21. Як в програмі визначається час завершення передачі та прийому чергового байта?
22. Дайти характеристику ASCII-коду.
23. Поясніть як в моделі підключено світлодіоди та в якому випадку вони будуть світитися.
24. Як працюють схеми відновлення тактового сигналу і даних при асинхронному прийомі?

7. ПРОЕКТУВАННЯ ПОСЛІДОВНОГО СИНХРОННОГО ІНТЕРФЕЙСУ TWI

7.1. Загальна характеристика інтерфейсу I^2C (TWI)

Інтерфейс I^2C (InterIC, або IIC) є двопровідним послідовним синхронним інтерфейсом, який розроблено фірмою “Philips Corporation”, і призначено для зв'язку між інтегральними мікросхемами або модулями. Існує ціла група I^2C-сумісних пристроїв для різних додатків: цифро-аналогові та аналого-цифрові перетворювачі, мікросхеми пам’яті, давачі, мікроконтролери, що містять модуль I^2C і т. ін [1…7]. Інтерфейс I^2C в AVR-мікроконтролерах має назву TWI.

Шина інтерфейсу I^2C складається із двох ліній:

– двонаправленої лінії даних (SDA);

– лінії тактових (синхро) імпульсів (SCL).

На рис. 7.1 наведено структурну схему типової мікроконтролерної мережі, що використовує для обміну даними інтерфейс (шину) I^2C.

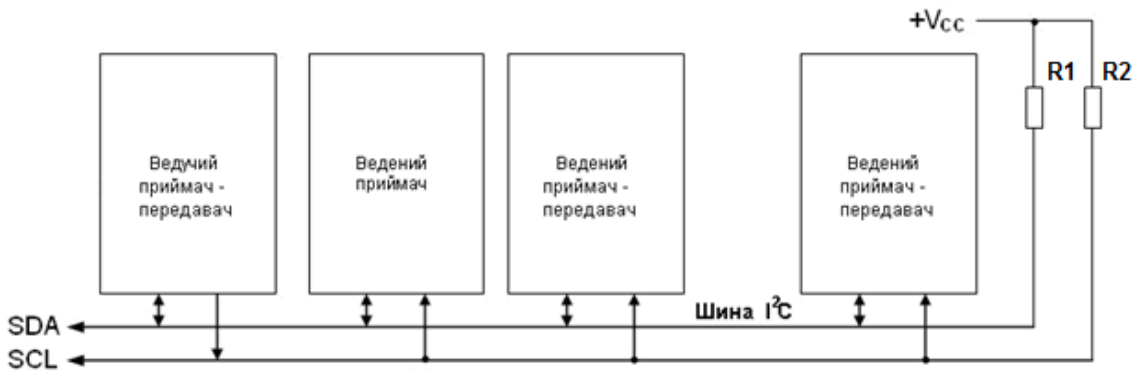


Рис. 7.1. Структурна схема мережі з інтерфейсом I^2C

Лінії SDA і SCL шини з'єднані з додатним полюсом напруги живлення (+*V*cc) через підтягуючі резистори: *R*1 та *R*2.

Передавач генерує і передає повідомлення, а приймач його приймає.

Один із двох пристроїв, які беруть участь у обміні, є ведучим (master), а інший – веденим (slave). Ведучий пристрій керує роботою шини та формує тактові сигнали (синхросигнали) SCL.

Кожний пристрій, який використовує для обміну інтерфейс I^2C, має свою адресу. Коли ведучий пристрій бажає ініціювати обмін даними, він передає на лінію SDA адресу пристрою, з яким буде виконуватися обмін (передача/прийом). Всі ведені пристрої слідкують за адресою, яка виставляється на шину, і порівнюють її із власною мережевою адресою. Після адреси ведучий передає біт

напряму $R/\overline{W}$, який визначає, чи буде ведучий читати дані від веденого ($R/\overline{W}=1$), чи буде передавати дані веденому ($R/\overline{W}=0$). Ведений приймач після одержання адреси або даних видає на шину SDA біт підтвердження (логічний нуль). Інтерфейс I^2C може використовувати два формати адреси:

– 7-ми бітну адресу;
– 10-ти бітну адресу.

В AVR-мікроконтролерах сімейства Mega використовуєть сьомибітна адреса.

На рис. 7.2 наведено формат адреси, на якому використано наступні позначення: S – умова «СТАРТ», $R/\overline{W}$ – біт «*читання* / $\overline{запис}$», $\overline{ACK}$ – біт підтвердження.

S	A6	A5	A4	A3	A2	A1	A0	$R/\overline{W}$	$\overline{ACK}$

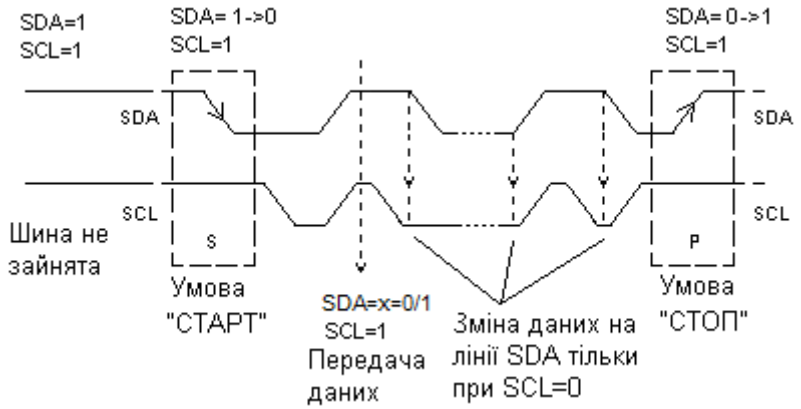
Рис. 7.2. Формат 7-бітної адреси

На рис. 7.3 наведено часові діаграми, які відображають стани на шині (рис. 7.3*а*), а також пояснюють формування сигналу підтвердження (рис. 7.3*б*).

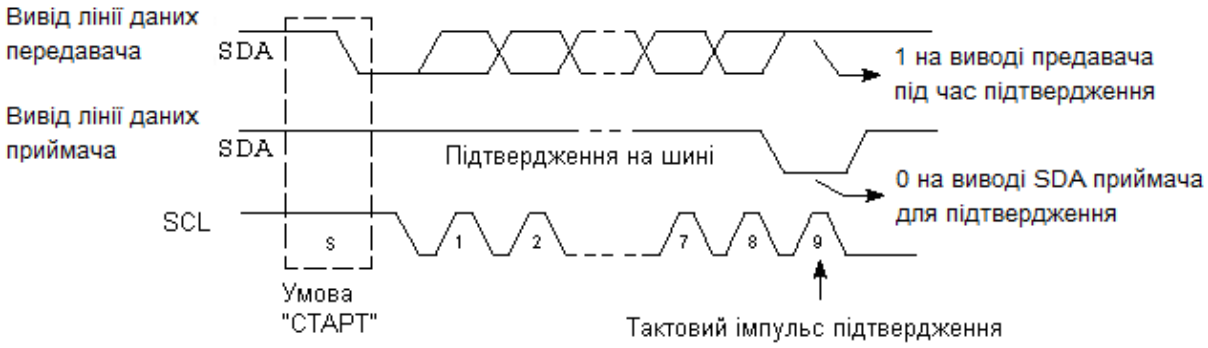
Наведені діаграми відображають наступні коректні стани сигналів на шині під час обміну даними:

– шина не зайнята: на обох лініях одиниці (SDA = SCL = 1);
– початок обміну даними (умова «СТАРТ» – S): зміна сигналу на лінії даних SDA з одиниці в нуль при одиничному значенні сигналу на лінії SCL;
– припинення передачі (умова «СТОП» – P): зміна сигналу на лінії даних з нуля в одиницю при одиничному значенні сигналу на лінії SCL;
– коректність даних: щоб не сформувати невірну умову «СТАРТ» або «СТОП» стан лінії даних не повинен змінюватися при одиничному значенні сигналу на лінії SCL. Дані можна змінювати, якщо на лінії SCL присутній низький рівень сигналу (логічний нуль). На один біт інформації на лінії SDA міститься один тактовий імпульс на лінії SCL. Кожний цикл обміну даними починається умовою «СТАРТ» і закінчується умовою «СТОП». Кількість інформаційних бітів даних, переданих між цими станами, необмежена. Дані передаються побайтово. Приймач підтверджує одержання чергового байта, посилаючи біт підтвердження (логічний нуль) після прийому кожного байта;

– біт підтвердження: передається після прийому кожного байта даних або адреси. Активний передавач (ведений або ведучий) після передачі чергового байта формує на лінії SDA сигнал високого рівня. Ведучий пристрій (приймач або передавач) формує на лінії SCL тактовий імпульс, а приймач (ведучий або ведений) видає на лінію даних SDA сигнал підтвердження низького рівня. Приймач, що генерує біт підтвердження, підключає лінію SDA до низького рівня і утримує її в цьому стані доти, доки тактовий імпульс лінії SCL не переключиться у стан низького рівня. Для припинення обміну приймач повинен залишити останній прийнятий байт без підтвердження, що автоматично викликає формування активним передавачем умови «СТОП».



а



б

Рис.7.3. Часові діаграми: *а* – передача даних; *б* – формування сигналу підтвердження

Для інтерфейсу I^2C можливі чотири режими (типи) обміну даними:

1) *Ведучий передавач*: на вихід SDA передавача виводяться дані, а на лінію SCL – синхроімпульси. Перший переданий байт містить адресу веденого приймача (7 біт) і біт напряму обміну даними $R/\overline{W}=0$, що говорить про те, що буде проводитися запис (передача). Дані передаються послідовно по 8 біт. Після передачі чергового байта (адреси або даних), ведучий передавач очікує від веденого приймача біт підтвердження $\overline{ACK}$. Для задання початку і кінця сеансу обміну даними ведучий передавач формує умови «СТАРТ» і «СТОП».

2) *Ведучий приймач*: на початку сеансу обміну ведучий приймач передає на лінію SDA адресу веденого передавача (7 біт) і біт напряму обміну $R/\overline{W}=1$, що говорить про те, що ведучий буде здійснювати прийом. Ведучий приймач формує імпульси синхронізації, які передаються лінією SCL. Після прийому адреси ведений передавач виставляє на лінію SDA сигнал підтвердження $\overline{ACK}$, а потім передає дані. Дані від веденого передавача передаються послідовно по 8 біт лінією SDA. Після прийому чергового байта ведучий приймач виставляє на лінію SDA сигнал підтвердження $\overline{ACK}$. Умови «СТАРТ» і «СТОП» формуються ведучим пристроєм для вказання початку і кінця сеансу обміну послідовними даними.

3) *Ведений приймач*: ведучий передавач видає на лінію SDA адресу веденого приймача й біт напряму $R/\overline{W}=0$, що говорить про те, що буде виконуватися запис (передача). На лінії SCL ведучий передавач видає синхроімпульси. Після одержання адреси ведений приймач передає сигнал підтвердження $\overline{ACK}$, після чого ведучий передавач послідовно передає на лінію SDA дані. Ведений приймач після прийому чергового байта даних передає сигнал підтвердження $\overline{ACK}$, що надходить до ведучого передавача лінією SDA. Умови «СТАРТ» і «СТОП» формуються ведучим передавачем.

4) *Ведений передавач*: перший байт (адреса) на шині SDA приймається і обробляється веденим передавачем так само, як і в режимі веденого приймача. При цьому біт напряму $R/\overline{W}=1$, що говорить про те, що ведучий буде здійснювати прийом. Дані послідовно передаються лінією SDA від веденого передавача у той час, як синхроімпульси передаються лінією SCL від ведучого приймача. Після передачі кожного байта ведений передавач аналізує наявність на лінії SDA біта підтвердження $\overline{ACK}$, який передає ведучий приймач. Умови «СТАРТ» і «СТОП» формує ведучий приймач.

У підпорядкованому (веденому) режимі апаратні засоби інтерфейсу I^2C здійснюють пошук своєї власної підпорядкованої адреси або адреси загального виклику. Якщо детектується одна із цих адрес, відбувається запит на переривання

і виконуються відповідні дії. Якщо модуль I^2C хоче захопити шину й стати ведучим, то він чекає, поки шина звільниться (SDA = SCL = 1). Можливе функціонування в якості веденого при цьому не переривається.

Два і більше пристрої можуть спробувати стати ведучими і одночасно згенерувати умову «СТАРТ». У цьому випадку здійснюється арбітраж шини в моменти, коли шина SCL перебуває у високому стані. Якщо один ведучий передає на лінію даних низький рівень, а інший – високий, то останній відключається від лінії, тому що стан шини SDA (низький) не відповідає високому стану внутрішньої шини даних пристрою, який бажає стати ведучим.

Якщо арбітраж шини загублено у головному (ведучому) режимі, то відповідний пристрій I^2C переключається у підпорядкований режим і може розпізнавати свою власну підпорядковану адресу.

На рис. 7.4 наведені часові діаграми, які відображають обмін даними шиною I^2C.

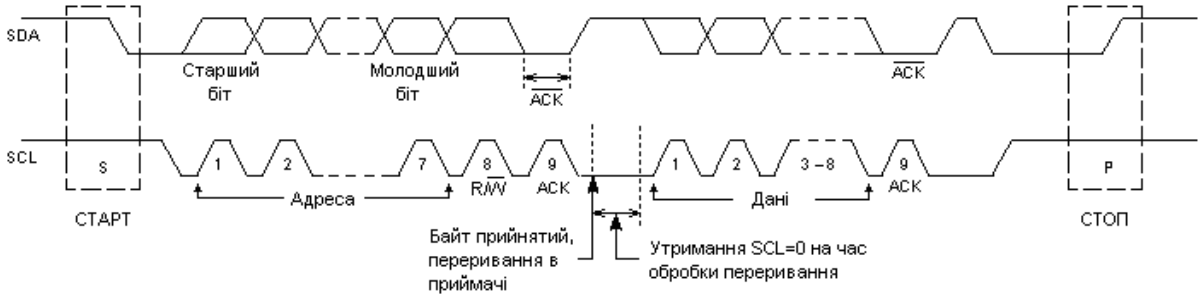


Рис. 7.4. Приклад обміну даними шиною I^2C

7.2. Архітектура інтерфейсу TWI на апаратному рівні

7.2.1. Загальна характеристика

До складу практично всіх мікроконтролерів AVR сімейства Mega входить модуль двопровідного синхронного послідовного інтерфейсу (Two Wire (Serial) Interface). В деяких моделях функцію TWI може виконувати модуль USI [1; 7].

Інтерфейс TWI являється повним аналогом базової версії інтерфейсу I^2C фірми «Philips». Інтерфейс TWI дозволяє об'єднати разом до 128 різних пристроїв за допомогою двонаправленої шини, яка складається із двох ліній: лінії тактового сигналу (SCL) і лінії даних (SDA). Додатково для реалізації шини використовуються два підтягуючі резистори: $R1$, $R2$, по одному на кожну лінію (рис. 7.5).

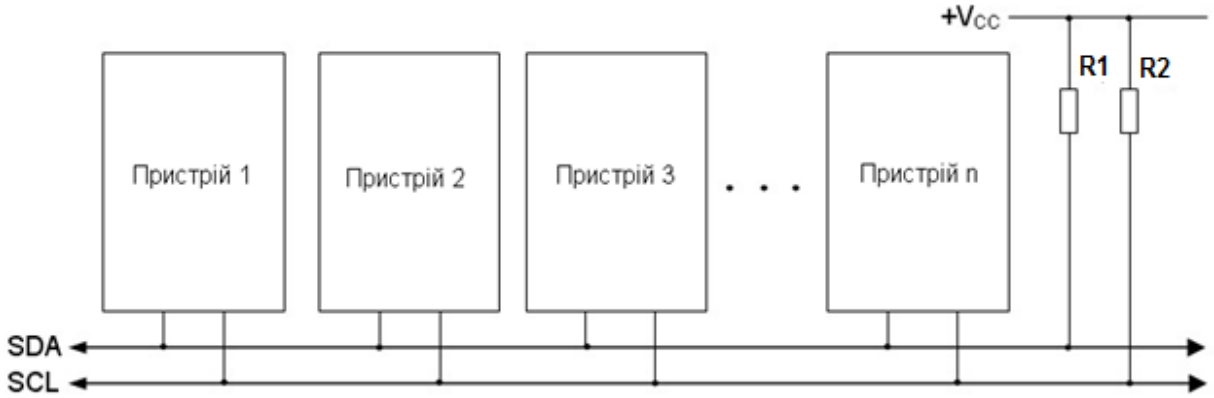


Рисунок 7.5. Використання підтягуючих резисторів в мережі TWI

Шинні формувачі на вихідних лініях всіх TWI-сумісних пристроїв виконуються за схемою з відкритим колектором (стоком), що дозволяє реалізувати функцію «монтажне АБО/І». Відповідно, НИЗЬКИЙ рівень на лінії встановлюється тоді, коли один або більше пристроїв виставляють на лінію сигнал логічного нуля (функція АБО для нулів), а ВИСОКИЙ рівень на лінії встановлюється тоді, коли всі пристрої, які підключені до неї, встановлюють свої виходи в одиничний або третій стан (функція І для одиниць).

Шина TWI повністю статична і швидкість обміну може бути досить низькою. Максимально допустима кількість мікросхем, які можна під'єднати до однієї шини, обмежується максимальною ємністю шини: 400 пФ.

Швидкість передачі даних для основного режиму роботи дорівнює: 100 кбіт/с., а для режиму із заниженою швидкістю – 10 кбіт/с.

Терміни, що нижче використовуються при описі модуля TWI, наведено у табл. 7.1.

7.2.2. Особливості обміну даними шиною TWI

Інформація порозрядно передається лінією SDA. Кожей розряд супроводжується тактовим імпульсом на лінії SCL. Причому сигнал на лінії SDA повинен бути стабільним увесь час, поки на шині SCL присутній одиничний сигнал (рис. 7.6). Єдиним винятком із цього правила є два особливих стани шини TWI: «СТАРТ» і «СТОП», які формуються ведучим на початку і наприкінці передачі даних відповідно. Між цими станами шина вважається зайнятою, та інші ведучі не повинні намагатися керувати нею.

Якщо ведучий хоче почати передачу нового блоку даних без втрати контролю над шиною, він може сформувати стан «повторний СТАРТ» («ПОВСТАРТ») до формування стану«СТОП». Шина буде вважатися зайнятою до наступного стану «СТОП».

Таблиця 7.1. **Терміни, що використовуються при описі модуля TWI**

Термін	**Опис**
Ведучий (Master)	Пристрій, який ініціює і завершує передачу даних шиною, а також генерує тактовий сигнал SCL
Ведений (Slave)	Пристрій, який адресується ведучим
Передавач (Transmitter)	Пристрій, який видає дані на шину
Приймач (Reciever)	Пристрій, який зчитує дані з шини

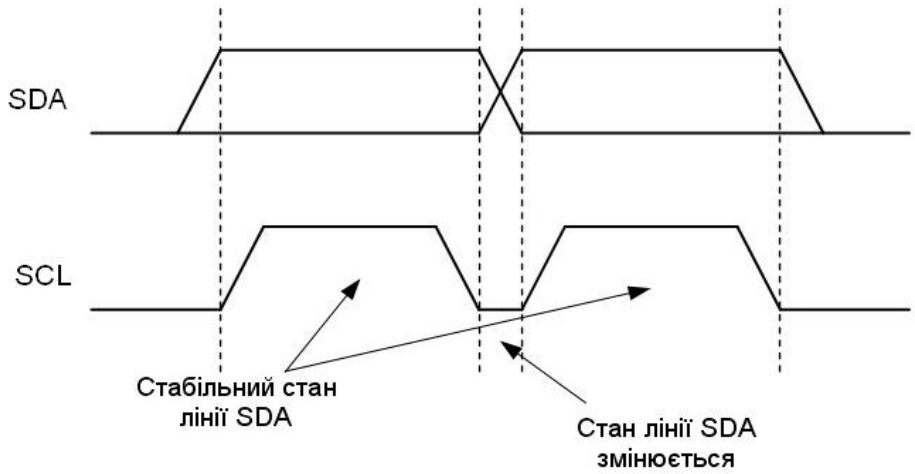


Рис. 7.6. Коректна видача даних на шину SDA

Стани «СТАРТ» і «СТОП» формуються шляхом зміни рівня сигналу на лінії SDA при ВИСОКОМУ рівні на лінії SCL. Стану «СТАРТ» відповідає зміна рівня з одиниці в нуль, а стану «СТОП» – навпаки, з нуля в одиницю (рис. 7.7).

Протокол інтерфейсу TWI дозволяє підключати до шини кілька ведучих пристроїв (режим Multi-Master). При цьому можуть виникати різні проблеми, однією з яких є розбіжність частот тактових сигналів, які генеруються різними ведучими.

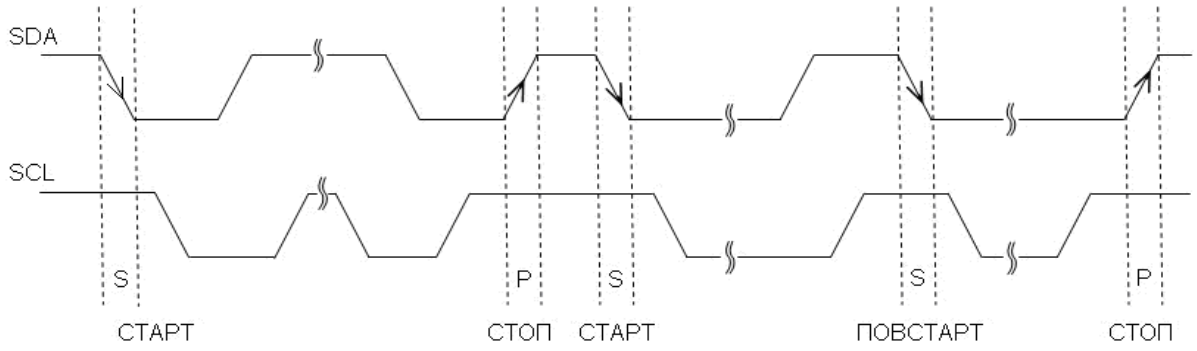


Рис. 7.7. Стани «СТАРТ», «СТОП» і «ПОВСТАРТ»

Задача синхронізації вирішується завдяки приєднанню всіх пристроїв до лінії SCL за схемою «монтажне АБО/І». Всі ведучі контролюють рівень, що присутній на лінії SCL, і визначають момент початку відліку імпульсу або паузи тактового сигналу за відповідною зміною сигналу (рис. 7.8).

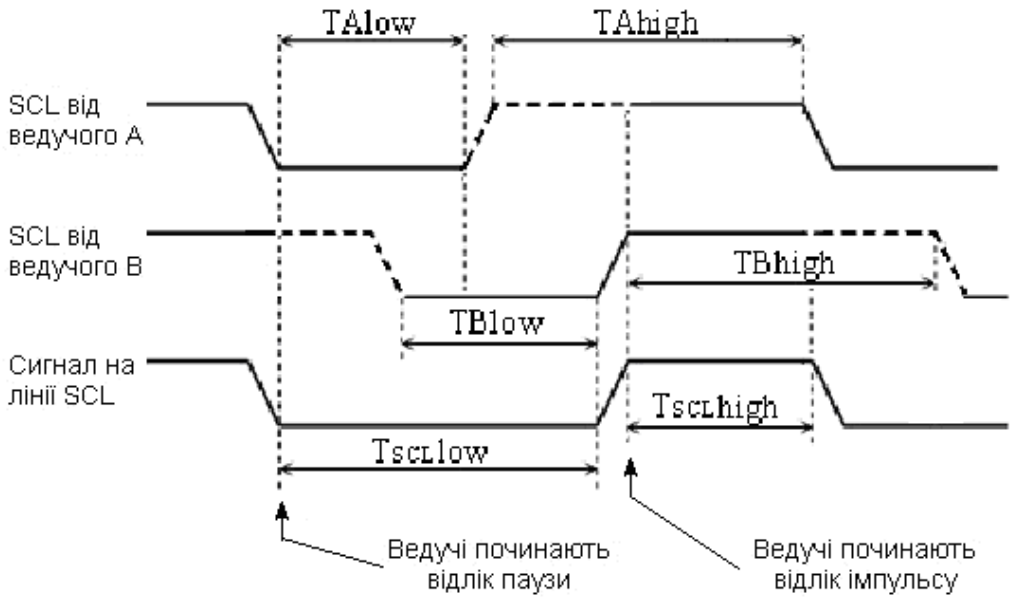


Рис. 7.8. Синхронізація сигналу SCL від декількох ведучих

Іншою задачею, яку доводиться вирішувати при підключенні до шини декількох ведучих пристроїв, є задача розподілу пріоритетів, у випадку, якщо два і більше ведучих одночасно намагаються почати передачу. При виникненні такої ситуації передачу може здійснити тільки один ведучий, інші ж повинні переключитися в режим веденого. Причому передані дані під час розподілу пріоритетів не повинні спотворюватися.

Для розв’язання описаної задачі всі ведучі пристрої після видачі даних на лінію SDA контролюють її стан. Якщо стан лінії відрізняється від того, в який її переводив ведучий, він втрачає пріоритет (рис. 7.9).

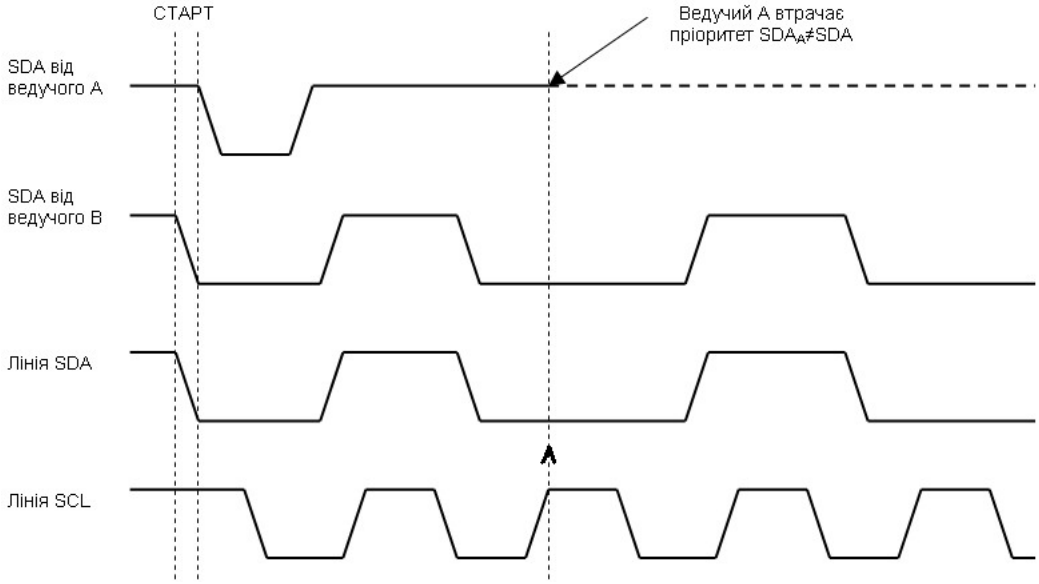


Рис. 7.9. Розподіл пріоритетів між двома ведучими

Це відбувається у випадку видачі одним з ведучих сигналу ВИСОКОГО рівня під час видачі іншими ведучими (які отримали пріоритет) сигналу НИЗЬКОГО рівня. Ведучий, який втратив пріоритет, переключається в режим веденого і перевіряє, чи не був він адресований яким-небудь із ведучих, що отримали пріоритет. При цьому він повинен продовжувати утримувати на лінії SDA сигнал ВИСОКОГО рівня та може продовжувати генерувати тактовий сигнал до закінчення передачі поточного пакета.

Процес розподілу пріоритетів триває доти, доки на шині не залишиться тільки один ведучий. У випадку, якщо декілька ведучих намагаються адресувати одного й того ж веденого, процес розподілу пріоритетів триває і під час передачі пакета даних. Таким чином, всі цикли обміну повинні містити однакову кількість пакетів даних. У іншому випадку результат процесу розподілу пріоритетів буде невизначеним.

При передачі шиною TWI даних, разом з ними передається певна службова інформація. Сукупність даних і відповідної службової інформації називається пакетом. Розрізняють адресні пакети і пакети даних.

Формат адресного пакета

Всі адресні пакети, які передаються шиною TWI, мають довжину 9 біт. Пакет включає 7-розрядну адресу (першим передається старший розряд), керуючий біт $R/\overline{W}$ і біт підтвердження $\overline{ACK}$ (рис. 7.10).

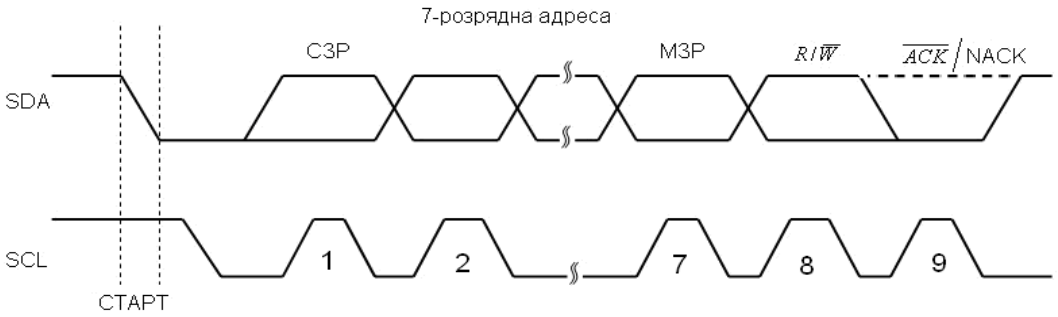


Рис. 7.10. Формат адресного пакета

Значення керуючого біта $R/\overline{W}$ визначає напрям передачі даних шиною. Скинутий в нуль біт означає передачу даних, а встановлений в одиницю – запит даних (читання). Пакети зі скинутим і встановленим керуючим бітом позначаються відповідно: $SLA+\overline{W}$ і $SLA+R$.

При розпізнаванні веденим своєї адреси він повинен під час 9-го тактового імпульсу сформувати підтвердження шляхом видачі на лінію SDA сигналу НИЗЬКОГО рівня ($\overline{ACK}$). Якщо ведений з якихось причин не може обслужити запит ведучого, він повинен під час 9-го тактового імпульсу утримувати на лінії SDA сигнал ВИСОКОГО рівня (NACK).

Веденим пристроям можуть бути призначені будь-які адреси, за винятком нульової адреси і адреси із діапазону «1111000...1111111» [7]. Доступних адрес, які можуть бути призначені веденим пристроям, всього 119. Нульова адреса зарезервована для реалізації так званих загальних викликів, а адреси формату «1111xxx», де x = 0/1 зарезервовані для використання в подальшому.

Загальний виклик використовується тоді, коли ведучий хоче передати повідомлення всім веденим, які підключено до шини. При прийомі адресного пакета з нульовою адресою і скинутим керуючим бітом (запит на передачу) всі ведені пристрої, які підключено до шини, повинні сформувати підтвердження. Винятки становлять лише ті пристрої, у яких розпізнавання загального виклику з якихось причин заборонено. Відповідно, наступні пакети даних, що посилаються ведучим, будуть отримані всіма веденими пристроями, які розпізнали адресу загального виклику.

Загальний виклик із встановленим керуючим розрядом (запит на читання) не має сенсу, оскільки різні пристрої не можуть одночасно здійснювати передачу даних шиною.

Формат пакета даних

Всі пакети даних, які передаються шиною TWI, теж мають довжину 9 біт (рис. 7.11). Пакет складається з байта даних (першим передається старший розряд) і біта підтвердження $\overline{ACK}$.

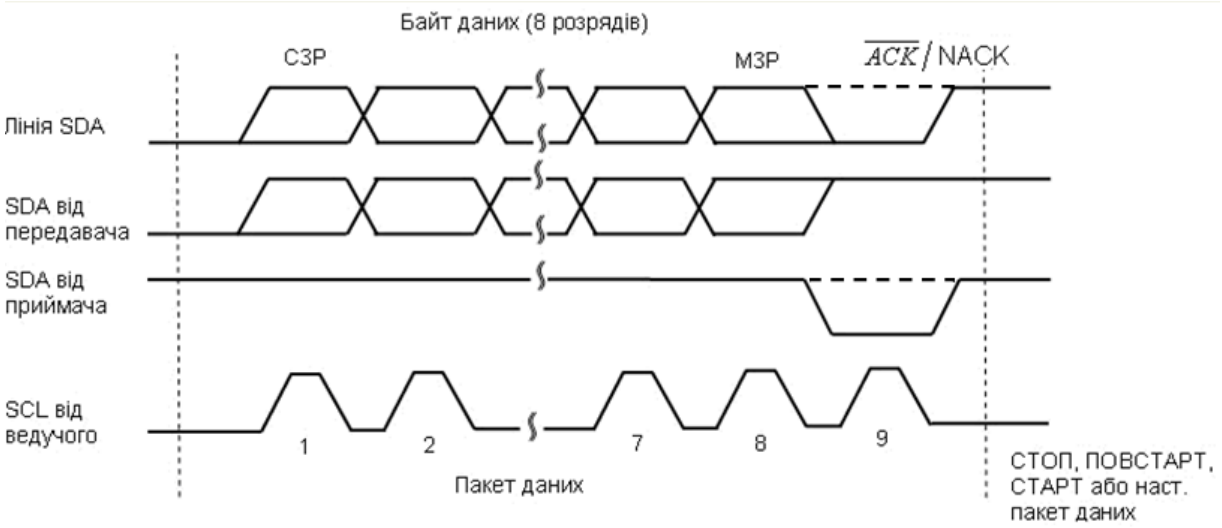


Рис. 7.11. Формат пакета даних

Генерація тактового сигналу і формування станів «СТАРТ» і «СТОП» здійснюється ведучим. Приймач, у свою чергу, повинен після прийому кожного байта формувати підтвердження ($\overline{ACK}$) шляхом видачі на лінію SDA сигналу НИЗЬКОГО рівня під час 9-го тактового імпульсу. Якщо приймач одержав останній байт або з яких-небудь інших причин не може продовжувати прийом даних, він повинен під час 9-го тактового імпульсу утримувати на лінії сигнал ВИСОКОГО рівня (NACK).

Не одержавши підтвердження від приймача, ведучий припиняє передачу даних, сформувавши стан «СТОП».

На практиці кожен цикл обміну шиною TWI складається з наступних етапів (рис. 7.12):

– формування стану «СТАРТ»;
– передача адресного пакета $SLA + R/\overline{W}$;
– передача одного або декількох пакетів даних;
– формування стану «СТОП».

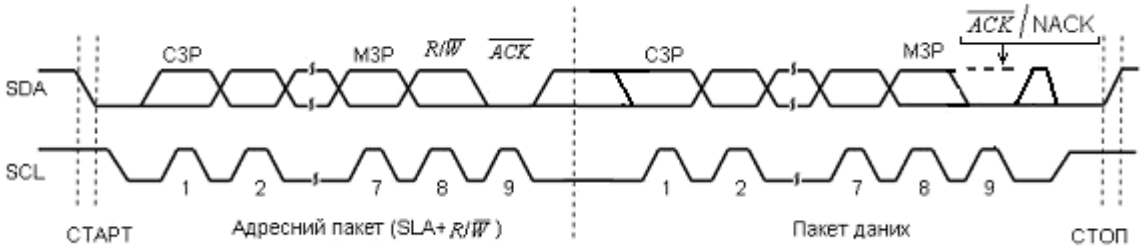


Рис. 7.12. Типовий цикл обміну

Швидкість обміну шиною TWI задається ведучим, тому що саме він генерує тактові імпульси. Однак, якщо ведений не може приймати дані з такою швидкістю або йому просто потрібен час на обробку даних між прийомом пакетів, він може збільшити паузу між тактовими імпульсами, утримуючи на лінії SCL сигнал НИЗЬКОГО рівня. На тривалість тактових імпульсів це не впливає.

7.2.3. Опис структури модуля TWI

Структурну схему модуля TWI наведено на рис. 7.13.

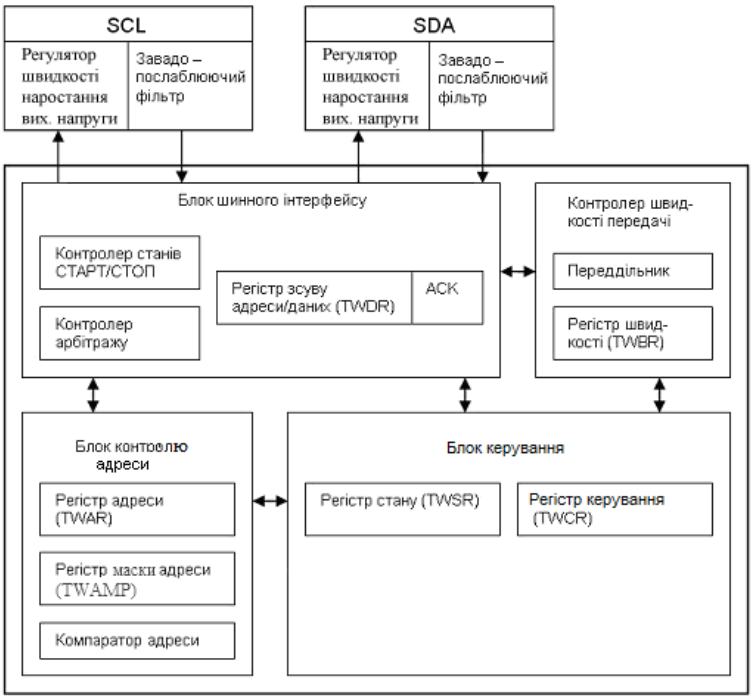


Рис. 7.13. Структурна схема модуля TWI

Для керування модулем використовуються п’ять (у нових моделях – шість) регістрів введення/виведення [7]. Назви і призначення цих регістрів, а також їх положення в адресному просторі регістрів введення/виведення деяких моделей сімейства Mega наведено в табл. 7.2.

Таблиця 7.2. **Регістри керування модулем TWI**

Регістр	**ATmega8535x**	**ATmega8x/16x/32x**	**ATmega64x/128x**	**ATmega48x/88x/168x**	**ATmega164x/324x/644x**	**ATmega640x ATmega1280x/1281x ATmega2560x/2561x**	**Призначення**
TWBR	$00 ($20)	$00 ($20)	($70)	($B8)	($B8)	($B8)	Регістр швидкості передачі
TWSR	$01 ($21)	$01 ($21)	($71)	($B9)	($B9)	($B9)	Регістр стану
TWAR	$02 ($22)	$02 ($22)	($72)	($BA)	($BA)	($BA)	Регістр адреси
TWDR	$03 ($23)	$03 ($23)	($73)	($BB)	($BB)	($BB)	Регістр даних
TWCR	$36 ($56)	$36 ($56)	($74)	($BC)	($BC)	($BC)	Регістр керування
TWAMR	–	–	–	($BD)	($BD)	($BD)	Регістр маски адреси

Для підключення модуля до однойменних ліній шини використовуються виводи: SCL і SDA. Відповідно до специфікації інтерфейсу TWI, вихідні каскади, які підключено до виводів, містять обмежувачі швидкості наростання сигналу, а вхідні каскади – фільтр, який придушує паразитні імпульси тривалістю менше 50 нс.

Обидва виводи є лініями портів введення/виведення мікроконтролерів (табл. 7.3). При використанні зазначених виводів модулем TWI, до них, як і при звичайному їхньому використанні, можна підключити внутрішні підтягуючі резистори. Це дозволить у ряді випадків обійтися без зовнішніх підтягуючих резисторів, що необхідно згідно специфікації інтерфейсу TWI.

Контролер швидкості передачі

Синхроімпульси, які видаються ведучим мікроконтролером на лінію SCL, формуються контролером швидкості передачі (Bit Rate Generator).

Таблиця 7.3. **Виводи деяких мікроконтролерів, що використовуються модулем TWI**

Назва	**ATmega8x**	**ATmega16x/32x**	**ATmega163x**	**ATmega48x/88x/168x**	**ATmega323x/324x ATmega464x/644x**	**ATmega64x/128x/640x ATmega1280x/1281x ATmega2560x/2561x**	**Призначення**
SDA	PC4	PC1	PC1	PC4	PC1	PD1	Лінія даних
SCL	PC5	PC0	PC0	PC5	PC0	PD0	Лінія тактового сигналу

У деяких моделях керування швидкістю здійснюється за допомогою регістра TWBR. Частота сформованого сигналу SCL (при роботі пристрою в режимі ведучого) для цих моделей визначається виразом (7.1) [7].

$$f_{sck} = f_{clk} /(16 + 2TWBR) . \qquad (7.1)$$

В нових моделях, наприклад сімейства Mega, для керування швидкістю разом з регістром TWBR використовуються два молодших розряди (TWPS1:TWPS0) регістра TWSR. Для цих моделей частота сформованого сигналу SCL визначається виразом (7.2) [7]:

$$f_{scl} = f_{clk}/(16 + 2TWBR \cdot 4^{TWPS}). \qquad (7.2)$$

У наведених формулах (7.1) і (7.2) використано наступні позначення:

f_{scl} – частота сигналу SCL (визначає швидкість передачі); f_{clk} – тактова частота мікроконтролера; TWBR – значення, яке записано в регістрі TWBR; TWPS – десятковий еквівалент розрядів TWPS1:TWPS0 регістра TWSR. Значення 4^{TWPS} обчислюється згідно табл. 7.4.

Регістр TWBR і розряди TWPS1:TWPSO регістра TWSR доступні як для читання, так і для запису в будь-який момент часу. При роботі в режимі веденого вміст зазначених розрядів не має значення, однак тактова частота мікроконтролера в цьому випадку повинна бути, як мінімум, в 16 разів вище частоти сигналу SCL [7].

Таблиця 7.4 . **Обчислення значення 4^{TWPS}**

TWPS1	TWPS0	Значення 4^{TWPS}
0	0	1
0	1	4
1	0	16
1	1	64

Оскільки будь-який пристрій, який підключено до шини TWI, може збільшувати тривалість паузи між тактовими імпульсами, швидкість передачі даних шиною може знижуватись.

Блок шинного інтерфейсу

До складу блоку шинного інтерфейсу (Bus Interface Unit) входять регістр зсуву адреси/даних TWDR, який містить адресу і дані переданого або прийнятого пакета. Одночасно з передачею вмісту регістра на шину дані з неї записуються у цей регістр. Таким чином, майже завжди, за винятком моменту виходу мікроконтролера зі «сплячого» режиму, в регістрі TWDR утримується останній байт, що був на шині. При включенні живлення всі розряди цього регістра встановлюються в одиницю. При цьому ініціалізація регістра користувачем може бути здійснена тільки після генерації першого переривання [7]. Крім регістра TWDR у складі блоку є спеціальний регістр, один з розрядів якого містить значення переданого або прийнятого біта підтвердження. При прийомі стан цього біта, що передається передавачу, задається одним з розрядів регістра керування TWCR, а при передачі стан отриманого біта підтвердження може бути визначений кодом, що перебуває в регістрі стану TWSR.

Окрім регістра зсуву блок шинного інтерфейсу містить два додаткових вузли:

– контролер станів «СТАРТ/СТОП», який формує і виявляє стани «СТАРТ», «ПОВСТАРТ» і «СТОП». Моніторинг стану шини ведеться навіть при знаходженні мікроконтролера у «сплячому» режимі. Завдяки цьому забезпечується вихід мікроконтролера зі «сплячого» режиму при адресації його яким-небудь ведучим;
– контролер арбітражу, який визначає наявність конфліктів на шині при роботі мікроконтролера в режимі ведучого. У випадку втрати пристроєм пріоритету контролер інформує про це блок керування, який робить необхідні дії та формує відповідні коди стану.

Блок контролю адреси

Блок контролю адреси (Address Match Unit) перевіряє прийняту адресу на відповідність значенню, яке перебуває у старших семи розрядах регістра адреси TWAR. Він також перевіряє наявність загальних викликів, якщо дозволено їхнє розпізнавання. При виявленні коректної адреси інформація про це передається блоку керування. Контроль адресних пакетів здійснюється блоком навіть при знаходженні мікроконтролера у «сплячому» режимі, що дозволяє перевести мікроконтролер у робочий режим у випадку адресації його ведучим пристроєм. Формат регістра TWAR показано на рис. 7.14, а опис його розрядів наведено в табл. 7.5.

	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
Читання(R)/Запис($\overline{W}$)	$R/\overline{W}$	$R/\overline{W}$	$R/\overline{W}$	$R/\overline{W}$	$R/\overline{W}$	$R/\overline{W}$	$R/\overline{W}$	$R/\overline{W}$
Початкове значення	1	1	1	1	1	1	1	0

Рис. 7.14. Формат регістра TWAR

Таблиця 7.5. **Опис розрядів регістра адреси TWAR**

Розряд	Назва	Опис
7...1	TWA6...TWA0	**Адреса пристрою** У цих розрядах утримується адреса, на яку пристрій буде відзиватися при роботі в режимі веденого. При роботі пристрою в режимі ведучого вміст цих розрядів не має значення
0	TWGCE	**Дозвіл розпізнавання загальних викликів** Якщо цей розряд встановлено в одиницю, пристрій буде озиватися на загальні виклики (пакети з адресою $00) так само, як і на виклики з адресою, що перебуває в розрядах TWA6...0 регістра. Якщо розряд скинуто в нуль, розпізнавання загальних викликів заборонено

У деяких моделях, крім регістра адреси TWAR, є додатковий регістр маски адреси (TWAMR) [7]. При встановленні будь-якого біта цього регістра в одиницю він буде маскувати (анулювати) значення відповідного біта регістра адреси TWAR. Інакше кажучи, якщо біт регістра маски адреси встановлено в одиницю, то порівняння між відповідними бітами вмісту регістра TWAR і прийнятої адреси не відбувається.

Формат регістра TWAMR наведено на рис. 7.15, а спрощену структурну схему блоку контролю адреси наведено на рис. 7.16.

	7	6	5	4	3	2	1	0
	TWAM6	TWAM5	TWAM4	TWAM3	TWAM2	TWAM1	TWAM0	–
Читання(R)/Запис(W)	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R
Початкове значення	0	0	0	0	0	0	0	0

Рис. 7.15. Формат регістра TWAMR

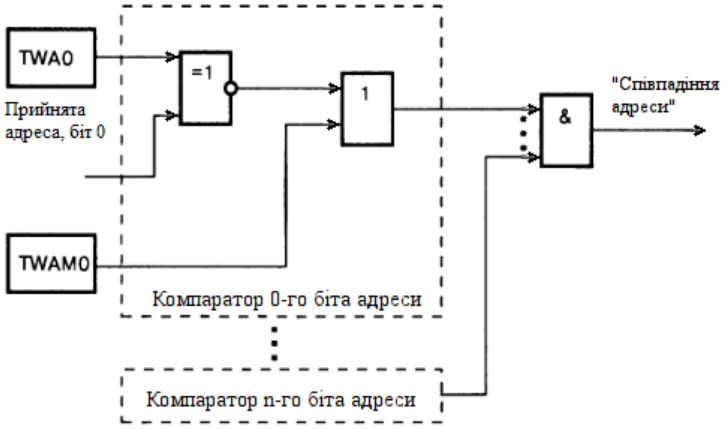


Рис. 7.16. Структурна схема блоку контролю адреси

Блок керування

Керування модулем TWI здійснює блок керування (Control Unit) відповідно до значень регістра керування TWCR та інформації, яка надходить до нього від інших блоків модуля. При настанні певних подій, зазначених нижче, блок керування формує в регістрі стану TWSR код статусу, що відповідає події, і встановлює в регістрі TWCR прапорець запиту на переривання TWINT. До моменту скидання цього прапорця на лінії SCL утримується НИЗЬКИЙ рівень, припиняючи тим самим передачу даних шиною.

Формування запиту на переривання здійснюється при виникненні наступних подій:

– закінчення формування стану «СТАРТ/ПОВСТАРТ»;
– закінчення передачі адресного пакета $(SLA + R/\overline{W})$;
– закінчення передачі байта даних;
– втрата пристроєм пріоритету;
– адресація пристрою або наявність загального виклику;
– закінчення прийому байта даних;
– виникнення помилок на шині, обумовлених неприпустимими умовами формування станів «СТАРТ/СТОП».

Формат регістра TWCR показано на рис. 7.17, а опис його розрядів наведено в табл. 7.6.

	7	6	5	4	3	2	1	0
	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Читання(R)/Запис($\overline{W}$)	$R/\overline{W}$	$R/\overline{W}$	$R/\overline{W}$	$R/\overline{W}$	R	$R/\overline{W}$	R	$R/\overline{W}$
Початкове значення	1	1	1	1	1	1	0	0

Рис. 7.17. Формат регістра TWCR

Таблиця 7.6. **Опис розрядів регістра керування TWCR**

Розряд	Назва	Опис
7	TWINT	**Прапорець переривання від модуля TWI** Цей прапорець встановлюється апаратно після виконання чергової операції, коли модуль очікує відгук з боку програми. Якщо встановлено прапорці: I регістра SREG і TWIE регістра TWCR, генерується переривання і здійснюється виклик відповідного оброблювача. Доки прапорець TWINT встановлено, на лінії SCL утримується сигнал НИЗЬКОГО рівня. Скидання прапорця може бути здійснено тільки записом у нього логічної одиниці
6	TWEA	**Дозвіл біта підтвердження** Розряд TWEA керує формуванням біта підтвердження. Якщо цей розряд встановлено в одиницю, то пристрій формує сигнал підтвердження, коли це необхідно. При скиданні розряду в нуль пристрій віртуально відключається від шини TWI, тобто біт підтвердження не формується
5	TWSTA	**Прапорець стану «СТАРТ»** При записуванні в розряд TWSTA логічної одиниці модуль перевіряє стан шини TWI та, якщо шина вільна, формує стан «СТАРТ». Якщо шина зайнята, модуль TWI очікує появи стану «СТОП», і тільки після цього формує стан «СТАРТ». Прапорець скидається апаратно по закінченні формування стану «СТАРТ»
4	TWSTO	**Прапорець стану «СТОП»** Встановлення прапорця TWSTO в одиницю в режимі ведучого приводить до формування на шині стану «СТОП». Прапорець скидається апаратно по закінченні формування стану «СТОП». Встановлення прапорця TWSTO в режимі веденого може використовуватися для виходу з помилкових ситуацій. Після запису в цей розряд одиниці модуль TWI повертається в режим не адресованого веденого, а виводи SCL і SDA для передачі переводяться у третій стан. Стан «СТОП» не формується
3	TWWC	**Прапорець конфлікту запису** Прапорець встановлюється в одиницю при спробі запису в регістр TWDR, коли прапорець переривання TWINT скинуто. Прапорець скидається при записуванні в регістр TWDR, коли прапорець переривання TWINT встановлено

Закінчення табл. 7.6.

Розряд	Назва	Опис
2	TWEN	**Дозвіл роботи модуля TWI** Розряд TWEN керує функціонуванням модуля TWI. При записуванні в цей розряд одиниці модуль TWI включається і бере на себе керування контактами введення/виведення мікроконтролера, які відповідають виводам SCL і SDA. При скиданні розряду TWEN в нуль модуль TWI вимикається
1	–	Зарезервовано, читається як нуль
0	TWIE	**Дозвіл переривання від модуля TWI** Якщо в цей розряд записано одиницю і прапорець I регістра SREG також встановлено в одиницю, то переривання від модуля TWI дозволено

Формат регістра TWSR показано на рис. 7.18, а опис його розрядів наведено в табл. 7.7.

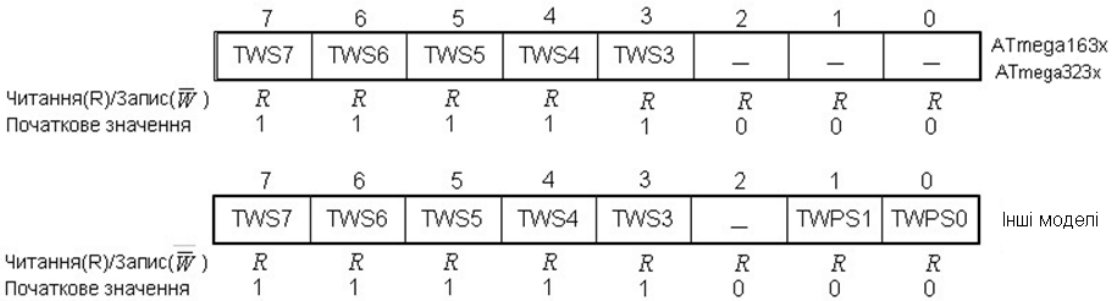


Рис. 7.18. Формат регістра TWSR

Таблиця 7.7. **Опис розрядів регістра стану TWSR**

Розряд	Назва	Опис
7...3	TWS7...TWS3	**Стан модуля TWI** Значення, яке утримується в цих розрядах, відображає стан вузлів модуля і шини TWI. Можливі коди статусу будуть описано при подальшому розгляді модуля TWI. Розряди TWS7...3 доступні тільки для читання
2	–	Зарезервовано, читається як «0»
1, 0	TWPS1, TWPS0	**Коефіцієнт ділення попереднього дільника контролера швидкості передачі** Стан цих розрядів визначає коефіцієнт ділення попереднього дільника контролера швидкості передачі, який керує частотою сигналу SCL, що генерується (див. вище опис контролера швидкості передачі)

7.2.4. Взаємодія прикладної програми з модулем TWI

Взаємодія прикладної програми з модулем TWI базується на використанні переривання від модуля, яке виникає після кожної події, що відбулася на шині (прийом байта, формування станів «СТАРТ/СТОП» і т. ін.). Відповідно, під час передачі даних шиною програма може виконувати інші задачі. Якщо переривання від модуля TWI з якихось причин використовувати не можна, його можна заборонити. У цьому випадку програма повинна буде постійно слідкувати за станом прапорця TWINT для реагування на події, які відбуваються на шині.

Встановлення прапорця TWINT регістра TWCR означає, що модуль TWI закінчив виконання чергової операції і очікує реакції програми. При цьому у старших п'яти розрядах регістра TWSR формується відповідне значення, яке характеризує поточний стан шини TWI. Відповідно, програма повинна проаналізувати це значення і задати подальше поводження модуля TWI, змінюючи вміст регістрів TWCR та TWDR.

На рис. 7.19 показано взаємодію прикладної програми з модулем TWI на прикладі передачі одного байта даних від ведучого до веденого.

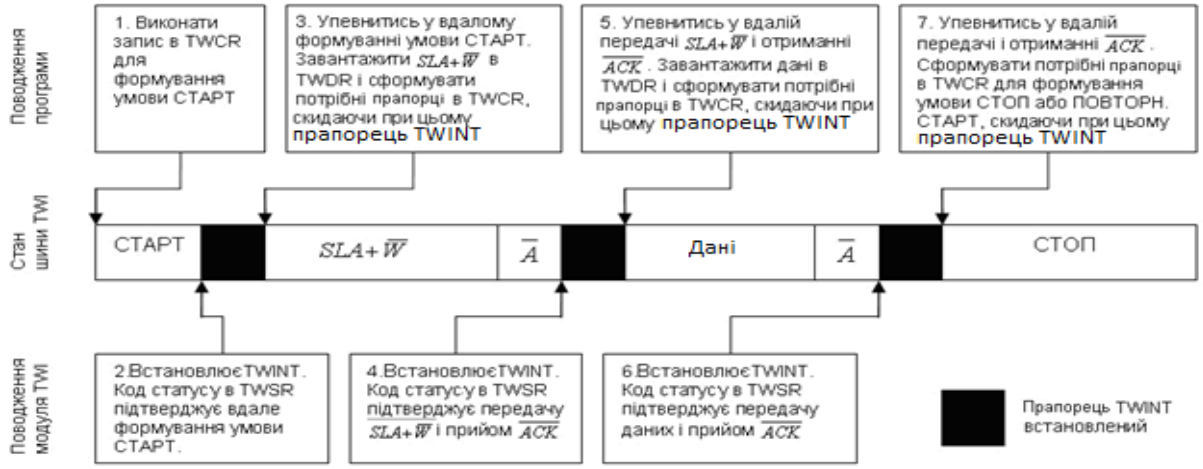


Рис. 7.19. Приклад взаємодії програми з модулем TWI

Передача даних відбувається у наступній послідовності.

1) Першою операцією при передачі даних шиною TWI є формування стану «СТАРТ». Для цього треба записати відповідне значення в регістр TWCR. Формування стану «СТАРТ» почнеться відразу ж після скидання прапорця TWINT. Для скидання прапорця TWINT розряд TWCR, який відповідає цьому прапорцю, треба встановити в одиницю. Після успішного формування стану

«СТАРТ» встановлюється прапорець TWINT. Число, яке перебуває у регістрі стану TWSR, відображає результат успішного виконання цієї операції.

2) Впевнитися в успішному формуванні стану «СТАРТ», перевіривши вміст регістра TWSR. Якщо код статусу відповідає очікуваному, завантажити в регістр TWDR вміст пакета $SLA+\overline{W}$ і сформувати в регістрі TWCR команду на передачу адресного пакета (не забуваючи скинути при цьому прапорець TWINT). Передача пакета почнеться відразу ж після скидання прапорця.

3) По закінченні передачі адресного пакета встановлюється прапорець TWINT. Код статусу, що перебуває в регістрі TWSR, говорить про успішну передачу пакета, а також про одержання підтвердження від веденого пристрою.

4) Впевнитися в успішній передачі адресного пакета та одержанні підтвердження, перевіривши вміст регістра TWSR. Якщо код статусу відповідає очікуваному, завантажити дані в регістр TWDR, а потім сформувати в регістрі TWCR команду на передачу пакета даних (не забуваючи скинути при цьому прапорець TWINT). Передача пакета даних почнеться відразу ж після скидання прапорця.

5) По закінченні передачі пакета даних встановлюється прапорець TWINT. Код статусу, що перебуває в регістрі TWSR, говорить про успішну передачу пакета, а також про одержання підтвердження від веденого пристрою.

6) Впевнитися в успішній передачі пакета даних та одержанні підтвердження, перевіривши вміст регістра TWSR. Якщо код статусу відповідає очікуваному, слід записати в регістр TWCR значення, відповідно до якого модуль TWI сформує на шині стан «СТОП» або «ПОВСТАРТ» (не забуваючи при цьому скинути прапорець TWINT). Формування стану «СТОП» або «ПОВСТАРТ» почнеться відразу ж після скидання прапорця TWINT.

Таким чином, взаємодії прикладної програми з модулем TWI складається із наступних трьох частин.

1) Після завершення модулем виконання чергової операції, він встановлює у регістрі TWCR прапорець TWINT, записує у регістр TWSR код статусу і очікує реакції програми. Доки прапорець встановлено в одиницю, на лінії SCL утримується НИЗЬКИЙ рівень.

2) Після встановлення прапорця TWINT, програма повинна занести в регістр TWDR значення, яке відповідає наступному етапу обміну.

3) Після оновлення вмісту регістра TWDR, програма повинна сформувати в регістрі TWCR команду для виконання наступного етапу обміну. При завантаженні в регістр нового значення необхідно скинути прапорець TWINT записом у нього одиниці. Після скидання прапорця модуль почне виконання операції, обумовленої вмістом регістра TWCR.

Нижче буде наведено опис конкретних режимів роботи модуля TWI з наведенням можливих значень кодів статусу.

При описанні дій програми у відповідь на отримання відповідного коду статусу, розглядаються тільки чотири прапорці регістра TWCR із семи: TWSTA (STA), TWSTO (STO), TWINT і TWEA. Інші три прапорці мають такі значення: TWWC = x (0/1), що залежить від відсутності/наявності помилки запису при передачі; TWEN = 1 (модуль TWI – включений); TWIE = x (0/1) – забороняє/дозволяє переривання при встановленні прапорця TWINT.

Є два коди стану загального призначення ($F8 і $00), які не пов'язано з певним режимом роботи (табл. 7.8).

Таблиця 7.8. **Коди статусу загального призначення**

Код статусу	**Стан шини і модуля TWI**	**Дії програми**					**Наступна дія, яку виконує модуль TWI**
		в/із TWDR	**Запис у регістр TWCR**				
			TWSTA	**TWSTO**	**TWINT**	**TWEA**	
$F8	Немає інформації; TWINT = 0	Немає дій	Немає дій				Чекає встановлення прапорця TWINT
$00	Помилка на шині в результаті некоректного формування стану «СТАРТ» або «СТОП»	Немає дій	0	1	1	x	Всі дії виконуються апаратно. Шина звільняється, а прапорець TWSTO скидається в нуль

Стан з кодом $F8 є проміжним. Наявність цього коду означає відсутність інформації у зв'язку з тим, що прапорець TWINT не встановлено.

Код статусу $00 сигналізує про помилку на шині. Така помилка виникає при формуванні ведучим станів «СТАРТ» або «СТОП» у неналежному місці, наприклад, під час передачі байта адреси, байта даних або біта підтвердження. Для виходу з таких ситуацій необхідно записати одиницю у розряди TWSTO і TWINT. В результаті модуль перейде в режим не адресованого веденого, звільнить лінії SDA і SCL і скине прапорець TWSTO. Стан «СТОП» у цій ситуації сформовано не буде.

7.2.5. Перелік мікроконтролерів AVR, які мають інтерфейс TWI

Нижче наведено перелік деяких мікроконтролерів AVR, які мають модуль TWI (табл. 7.9).

Таблиця 7.9. **Перелік мікроконтролерів AVR, які мають модуль TWI**

Сімейство	**Модель мікросхеми**
XMega	всі моделі
Mega	ATmega48 ATmega8 ATmega88 ATmega8535 ATmega16 ATmega168 ATmega32 ATmega644 ATmega640 ATmega128 ATmega1281 ATmega1280 ATmega2561 ATmega2560
Tiny	ATtiny48 ATtiny88
picoPower	ATmega48P ATmega88P ATmega168P ATmega328P* ATmega324P ATmega644P ATmega1284P**
SmartBattery AVR	ATmega406
Automotive	ATmega164P Automotive ATmega324P Automotive ATmega644P Automotive ATmega48 Automotive ATmega88 Automotive ATmega168 Automotive AT90CAN32 Automotive AT90CAN64 Automotive AT90CAN128 Automotive

7.2.6. Перелік пристроїв, які підтримують інтерфейс TWI

Нижче наведено перелік деяких пристроїв, які підтримують інтерфейс TWI: EEPROM-пам`ять даних; Flash-пам`ять програм; давачі і т. ін. [7].

7.3. Програмування інтерфейсу TWI

7.3.1. Режими роботи модуля TWI

Модуль TWI може працювати в наступних режимах:

– ведучий передавач (MasterTransmitter);
– ведучий приймач (MasterReceiver);
– ведений передавач (SlaveTransmitter);
– ведений приймач (SlaveReceiver).

Вибір конкретного режиму залежить від роботи програми і, відповідно, дій, які вона виконує.

7.3.2. Режим «Ведучий передавач»

В режимі «Ведучий передавач» (MasterTransmitter) здійснюється передача даних від ведучого пристрою до веденого. Для перемикання пристрою в режим «Ведучий» необхідно записати у регістр TWCR керуюче слово, яке переводить модуль TWI у стан «СТАРТ». Формат адресного пакета, який потім передається, визначає, в якому з режимів буде працювати ведучий. При передачі пакета $SLA+\overline{W}$ модуль переходить у режим «Ведучий передавач», а при передачі пакета $SLA+R$ – у режим «Ведучий приймач».

Для формування стану «СТАРТ» треба записати в регістр TWCR наступне значення:

TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
1	0/1	1	0	0/1	1	0	0/1

Дозволити роботу модуля TWI
Сформувати стан СТАРТ
Скинути прапорець TWINT

Після цього модуль TWI почне контролювати стан шини і сформує стан «СТАРТ» відразу ж, як тільки вона стане вільною.

По закінченні формування стану «СТАРТ» встановлюється прапорець TWINT, а код статусу має значення $08.

Для перемикання модуля в режим «Ведучий передавач» необхідно передати шиною пакет $SLA+\overline{W}$. Для цього вміст пакета завантажується в регістр TWDR, а в регістр TWCR заноситься наступне значення:

TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
1	0/1	0	0	0/1	1	0	0/1

Дозволити роботу модуля TWI

Скинути прапорець TWINT

Після передачі адресного пакета і прийому біта підтвердження прапорець TWINT знову встановлюється в одиницю. Код статусу може мати одне з наступних значень: $18, $20 або $38. Дії, які необхідно виконати при виявленні того або іншого коду, описано нижче в табл. 7.10.

Після передачі адресного пакета передаються пакети даних. Значення байта даних завантажується в регістр TWDR. Передача пакета даних починається після запису в регістр TWCR наступного значення:

TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
1	0/1	0	0	0/1	1	0	0/1

Дозволити роботу модуля TWI

Скинути прапорець TWINT

Описана процедура використовується для передачі всіх пакетів даних. Після передачі останнього байта даних, ведучий повинен сформувати на шині стан «СТОП» або «ПОВСТАРТ». Формування стану «СТОП» почнеться після запису в регістр TWCR наступного значення:

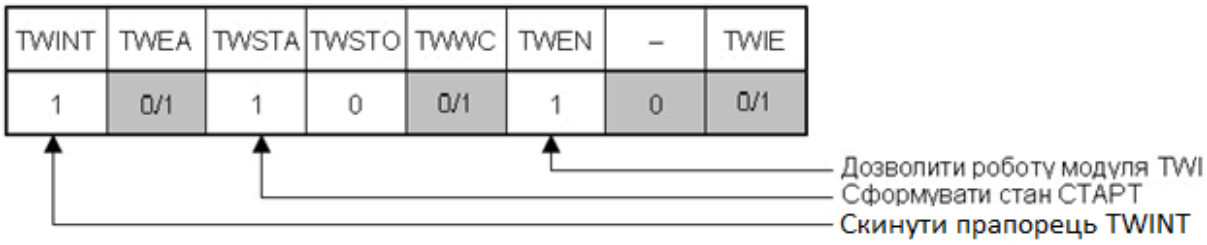
TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
1	0/1	0	1	0/1	1	0	0/1

Дозволити роботу модуля TWI

Сформувати стан СТОП

Скинути прапорець TWINT

Для формування стану «ПОВСТАРТ» у регістр TWCR необхідно занести наступне значення:



Після формування на шині стану «ПОВСТАРТ» (код статусу $10) ведучий може адресувати той же або інший ведений, не формуючи стан «СТОП». Інакше кажучи, використання стану «ПОВСТАРТ» дозволяє здійснювати зміну ведених пристроїв, а також переключатися між режимами «Ведучий передавач» і «Ведучий приймач» без втрати контролю над шиною.

Таблиця 7.10. **Коди статусу для режиму «Ведучий передавач»**

Код статусу	**Стан шини та модуля TWI**	**Дії програми**						**Наступна дія, що виконується модулем TWI**
		в TWDR	**в регістр TWCR**					
			TWSTA	**TWSTO**	**TWINT**	**TWEA**	**TWEN**	
$08	Було сформовано стан СТАРТ	Завантажити $SLA+\overline{W}$	0	0	1	X	1	Буде передано $SLA+\overline{W}$ і отримано $\overline{ACK}$ або $NACK$
$10	Було сформовано стан ПОВСТАРТ	Завантажити $SLA+\overline{W}$	0	0	1	X	1	Буде передано $SLA+\overline{W}$ і отримано $\overline{ACK}$ або $NACK$
		Завантажити $SLA+R$	0	0	1	X	1	Буде передано $SLA+R$. Модуль переключиться в режим «Ведучий приймач»
$18	Було передано пакет $SLA+\overline{W}$ і прийнято підтвердження ($\overline{ACK}$)	Завантажити дані	0	0	1	X	1	Буде передано байт даних і отримано $\overline{ACK}$ або $NACK$
		Немає дій	1	0	1	X	1	Буде сформовано стан ПОВСТАРТ
		Немає дій	0	1	1	X	1	Буде сформовано стан СТОП (прапорець TWSTO буде скинуто)
		Немає дій	1	1	1	X	1	Буде сформовано стан СТОП, а потім стан СТАРТ (прапорець TWSTO буде скинуто)

Продовження табл. 7.10.

Код статусу	Стан шини та модуля TWI	Дії програми						Наступна дія, що виконується модулем TWI
		в TWDR	в регістр TWCR					
			TWSTA	TWSTO	TWINT	TWEA	TWEN	
$20	Було передано пакет $SLA+\overline{W}$ і прийнято непідтвердження (*NACK*)	Завантажити дані	0	0	1	X	1	Буде передано байт даних і отримано $\overline{ACK}$ або *NACK*
		Немає дій	1	0	1	X	1	Буде сформовано стан ПОВСТАРТ
		Немає дій	0	1	1	X	1	Буде сформовано стан СТОП (прапорець TWSTO буде скинуто)
		Немає дій	1	1	1	X	1	Буде сформовано стан СТОП, а потім стан СТАРТ (прапорець TWSTO буде скинуто)
$28	Було передано пакет даних і прийнято підтвердження ($\overline{ACK}$)	Завантажити дані	0	0	1	X	1	Буде передано байт даних і отримано $\overline{ACK}$ або *NACK*
		Немає дій	1	0	1	X	1	Буде сформовано стан ПОВСТАРТ
		Немає дій	0	1	1	X	1	Буде сформовано стан СТОП (прапорець TWSTO буде скинуто)
		Немає дій	1	1	1	X	1	Буде сформовано стан СТОП, а потім стан СТАРТ (прапорець TWSTO буде скинуто)
$30	Було передано пакет даних і прийнято непідтвердження (*NACK*)	Завантажити дані	0	0	1	X	1	Буде передано байт даних і отримано $\overline{ACK}$ або *NACK*
		Немає дій	1	0	1	X	1	Буде сформовано стан ПОВСТАРТ
		Немає дій	0	1	1	X	1	Буде сформовано стан СТОП (прапорець TWSTO буде скинуто)
		Немає дій	1	1	1	X	1	Буде сформовано стан СТОП, а потім стан СТАРТ (прапорець TWSTO буде скинуто)

Закінчення табл. 7.10.

Код статусу	Стан шини та модуля TWI	Дії програми						Наступна дія, що виконується модулем TWI
		в TWDR	в регістр TWCR					
			TWSTA	TWSTO	TWINT	TWEA	TWEN	
$38	Втрата пріоритету при передачі пакета адреси або даних	Немає дій	0	0	1	X	1	Пристрій звільнить шину і перейде в режим не адресованого веденого
		Немає дій	1	0	1	X	1	Після звільнення шини буде сформовано стан «СТАРТ»

У [1] наведено схему алгоритму роботи модуля в даному режимі та його програмну реалізацію.

На рис. 7.20 наведено часові діаграми роботи та різні стани модуля TWI у режимі «Ведучий передавач».

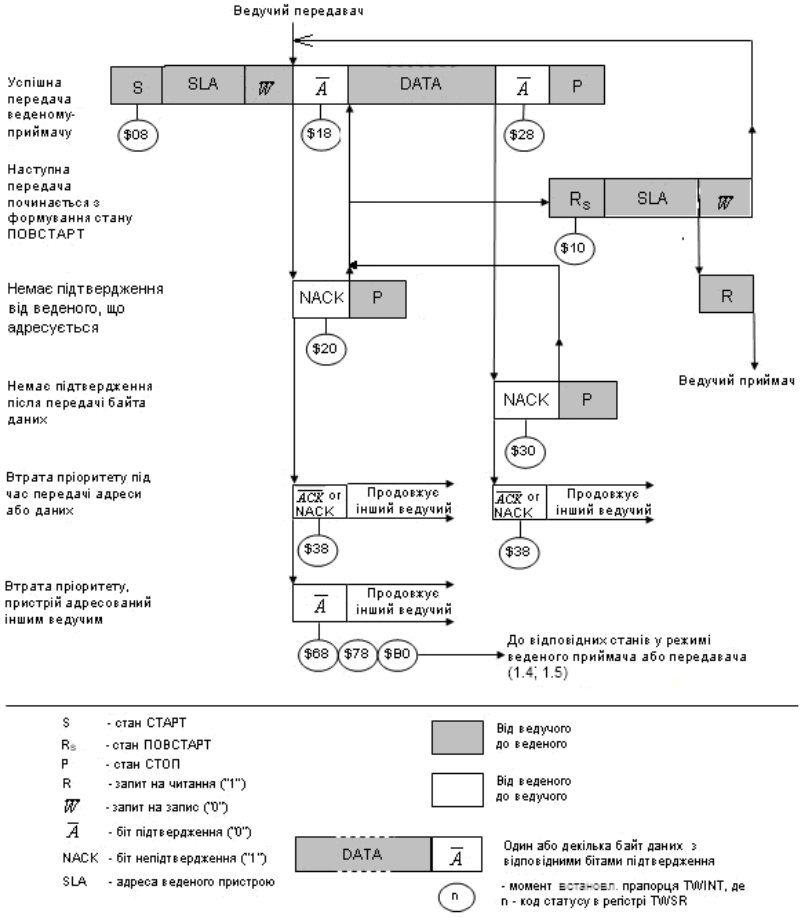


Рис. 7.20. Часові діаграми роботи та стани модуля TWI в режимі “Ведучий передавач”

7.3.3. Режим «Ведучий приймач»

В режимі «Ведучий приймач» ведучий здійснює прийом даних від веденого пристрою. Для переключення пристрою в режим «Ведучий» необхідно записати в регістр команд команду для формування сигналу «СТАРТ». Формат адресного пакета, який передається потім, визначає, в якому з режимів буде працювати ведучий. При передачі пакета $SLA+\overline{W}$ модуль переходить у режим «Ведучий передавач», а при передачі пакета $SLA+R$ переходить у режим «Ведучий приймач».

Формування стану «СТАРТ» починається після запису в регістр TWCR наступного значення:

TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
1	0/1	1	0	0/1	1	0	0/1

Дозволити роботу модуля TWI
Сформувати стан СТАРТ
Скинути прапорець TWINT

В результаті запису зазначеного значення модуль TWI почне контролювати стан шини і сформує стан «СТАРТ» одразу ж, як тільки вона стане вільною.

По закінченні формування стану «СТАРТ» встановлюється прапорець TWINT. Код статусу повинен при цьому мати значення, що дорівнює $08. Для переключення модуля в режим «Ведучий приймач» необхідно передати шиною пакет $SLA+R$. Для цього вміст пакета завантажується в регістр TWDR, а в регістр TWCR заноситься наступне значення:

TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
1	0/1	0	0	0/1	1	0	0/1

Дозволити роботу модуля TWI
Скинути прапорець TWINT

Після передачі адресного пакета і прийому біта підтвердження прапорець TWINT знову встановлюється в одиницю. Код статусу на цьому етапі може мати одне з наступних значень: $38, $40 або $48. Дії, які необхідно почати при виявленні того або іншого коду, розглянуто нижче в табл. 7.11.

Для скидання біта TWINT і дозволу формування біта підтвердження $\overline{ACK}$ у відповідь на прийнятий байт від веденого пристрою необхідно занести в регістр TWCR наступне значення:

TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
1	1	0	0	0/1	1	0	0/1

Дозволити роботу модуля TWI
Дозволити підтвердження
Скинути прапорець TWINT

Після прийому байта код статусу має значення: $50 або $58. Дії модуля TWI у відповідь на це наведено нижче в табл. 7.11.

Описана процедура використовується для прийому всіх пакетів даних. Після прийому останнього байта даних ведучий повинен проінформувати про це веденого передавача, пославши сигнал не підтвердження (NACK). Потім ведучий повинен сформувати на шині стан «СТОП» або «ПОВСТАРТ». Формування стану «СТОП» почнеться після запису в регістр TWCR наступного значення:

TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
1	0/1	0	1	0/1	1	0	0/1

Дозволити роботу модуля TWI
Сформувати стан СТОП
Скинути прапорець TWINT після прийому байта даних

Для формування стану «ПОВСТАРТ» у регістр TWCR необхідно занести наступне значення:

TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
1	0/1	1	0	0/1	1	0	0/1

Дозволити роботу модуля TWI
Сформувати стан СТАРТ
Скинути прапорець TWINT

Після формування на шині стану «ПОВСТАРТ» (код статусу $10) ведучий може адресувати той же або інший ведений, не формуючи стану «СТОП». Використання стану «ПОВСТАРТ» дозволяє здійснювати зміну ведених пристроїв, а також переключатися між режимами «Ведучий передавач» і «Ведучий приймач» без втрати контролю над шиною.

Таблиця 7.11. **Коди статусу для режиму «Ведучий приймач»**

Код статусу	**Стан шини модуля TWI**	**Дії програми**						**Наступна дія, що виконується модулем TWI**
		в/із TWDR	**в регістр TWCR**					
			TWSTA	**TWSTO**	**TWINT**	**TWEA**	**TWEN**	
$08	Було сформовано стан «СТАРТ»	Завантажити $SLA+R$	0	0	1	X	1	Буде передано $SLA+R$. Буде отримано $\overline{ACK}$ або $NACK$
$10	Було сформовано стан «ПОВСТАРТ»	Завантажити $SLA+R$	0	0	1	X	1	Буде передано $SLA+R$. Буде отримано $\overline{ACK}$ або $NACK$
		Завантажити $SLA+\overline{W}$	0	0	1	X	1	Буде передано $SLA+\overline{W}$. Модуль переключиться в режим «Ведучий передавач»
$38	Втрата пріоритету при передачі пакета адреси або даних	Немає дій	0	0	1	X	1	Пристрій звільнить шину й перейде в режим не адресованого веденого
$40	Було передано пакет $SLA+R$ і прийнято підтвердження $\overline{ACK}$	Немає дій	0	0	1	0	1	Буде прийнято байт даних і передано не підтвердження (*NACK)*
		Немає дій	0	0	1	1	1	Буде прийнято байт даних і передано підтвердження ($\overline{ACK}$)
$48	Було передано пакет $SLA+R$ і прийнято не підтвердження *NACK*	Немає дій	1	0	1	X	1	Буде сформовано стан «ПОВСТАРТ»
		Немає дій	0	1	1	X	1	Буде сформовано стан «СТОП»(прапорець TWSTO буде скинуто)
		Немає дій	1	1	1	X	1	Буде сформовано стан «СТОП», а потім стан «СТАРТ» (прапорець TWSTO буде скинуто)

Закінчення табл. 7.11.

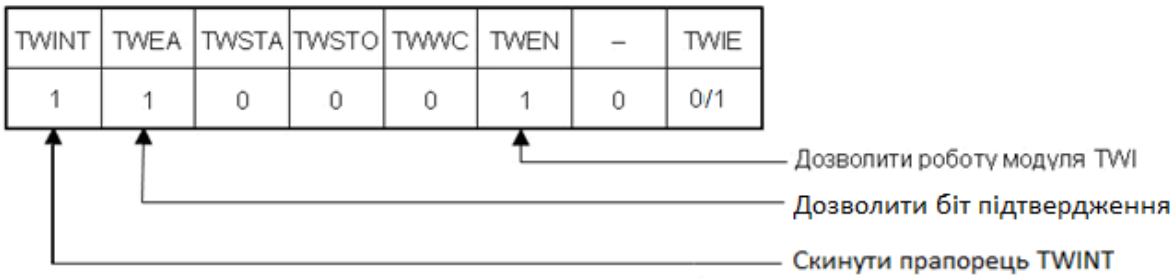
Код статусу	Стан шини модуля TWI	Дії програми						Наступна дія, що виконується модулем TWI
		в/із TWDR	в регістр TWCR					
			TWSTA	TWSTO	TWINT	TWEA	TWEN	
$50	Було прийнято байт даних і передано підтвердження $\overline{ACK}$	Зчитати дані	0	0	1	0	1	Буде прийнято байт даних і передано *NACK*
			0	0	1	1	1	Буде прийнято байт даних і передано підтвердження $\overline{ACK}$
$58	Було прийнято байт даних і передано не підтвердження (*NACK*)	Зчитати дані	1	0	1	X	1	Буде сформовано стан «ПОВСТАРТ»
		Зчитати дані	0	1	1	X	1	Буде сформовано стан «СТОП» (прапорець TWSTO буде скинуто)
		Зчитати дані	1	1	1	X	1	Буде сформовано стан «СТОП», а потім стан «СТАРТ» (прапорець TWSTO буде скинуто)

У [1] наведено схему алгоритму роботи модуля в даному режимі та його програмну реалізацію.

На рис. 7.21 наведено часові діаграми роботи та різні стани модуля TWI в режимі «Ведучий приймач».

7.3.4. Режим «Ведений приймач»

В режимі «Ведений приймач» ведений пристрій здійснює прийом даних від ведучого. Перед тим як переключити модуль у режим «Ведений приймач», слід занести у старші розряди регістра TWAR адресу пристрою і відповідно до логіки роботи програми встановити або скинути молодший розряд цього регістра (TWGCE). Потім необхідно записати в регістр TWCR наступне значення:



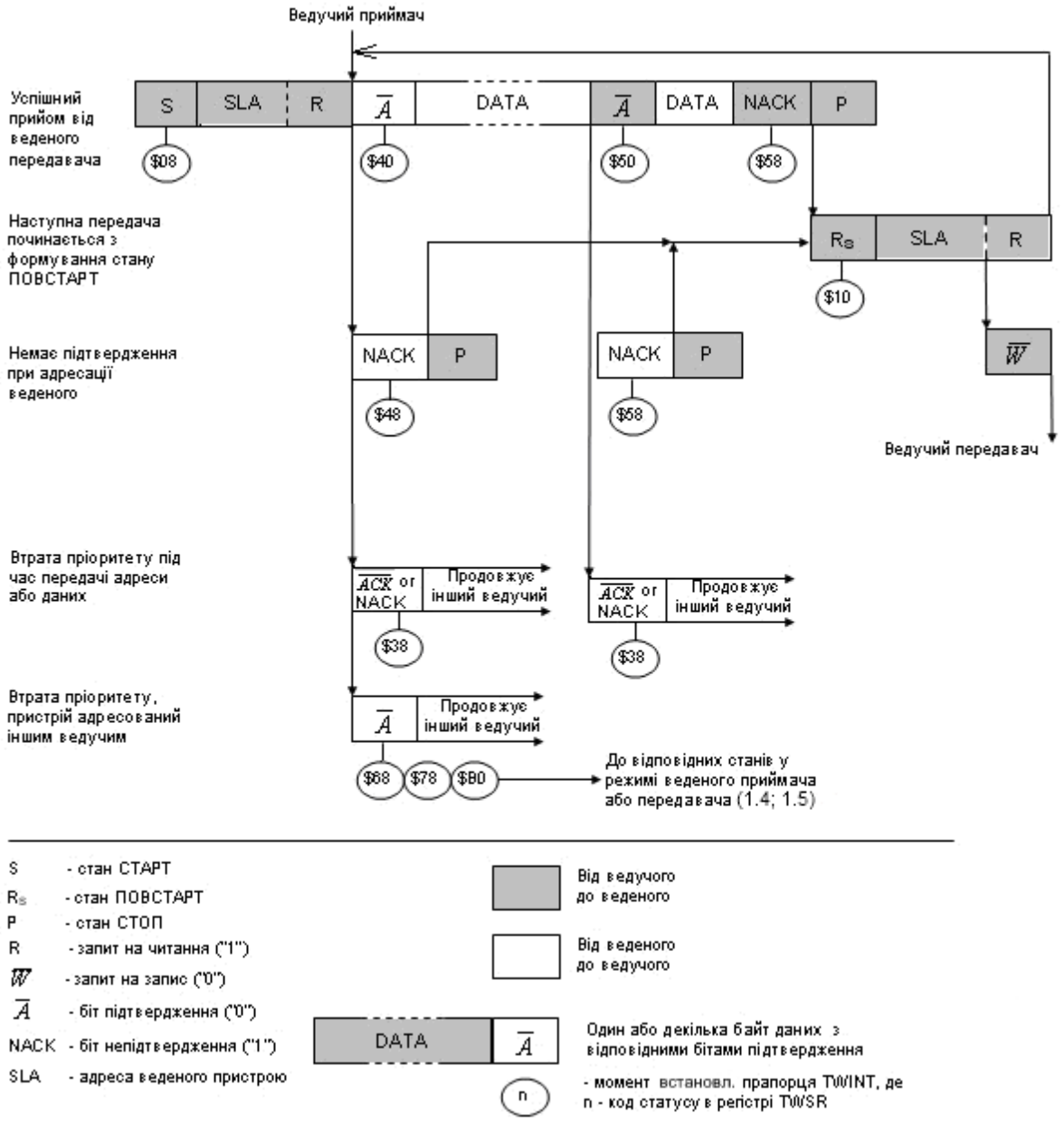


Рис. 7.21. Часові діаграми роботи та стани модуля TWI у режимі «Ведучий приймач»

Після ініціалізації регістрів TWAR і TWCR модуль буде чекати на адресний пакет з адресою, зазначеною в регістрі TWAR, або загальний виклик (якщо його розпізнавання дозволено). Наступне за адресою значення біта напряму визначить режим, в який переключиться модуль. Якщо цей біт буде скинуто в нуль (запис), модуль TWI переключиться в режим «Ведений приймач». У іншому випадку модуль переключиться в режим «Ведений передавач». Пристрій також може самостійно переключитися в режим «Ведений приймач» з режиму ведучого при втраті пріоритету (див. нижче опис кодів статусу $68 і $78 в табл. 7.12).

Після прийому пакета $SLA+\overline{W}$ встановиться прапорець TWINT і стан обміну шиною можна буде, як зазвичай, визначити за кодом статусу. Можливі дії

з боку програми, що відповідають тому або іншому коду статусу, зазначено в табл. 7.12.

Таблиця 7.12. **Коди статусу для режиму «Ведений приймач»**

Код статусу	Стан шини модуля TWI	Дії програми						Наступна дія, що виконується модулем TWI
		в/із TWDR	в регістр TWCR					
			TWSTA	TWSTO	TWINT	TWEA	TWEN	
$60	Було прийнято $SLA+\overline{W}$ з власною адресою і послано підтвердження ($\overline{ACK}$)	Немає дій	0	0	1	0	1	Буде прийнято байт даних і передано не підтвердження (*NACK*)
		Немає дій	0	0	1	1	1	Буде прийнято байт даних і передано підтвердження ($\overline{ACK}$)
$68	Втрата пріоритету в режимі ведучого під час передачі $SLA+R/\overline{W}$; було прийнято $SLA+\overline{W}$ з власною адресою і послано ($\overline{ACK}$)	Немає дій	0	0	1	0	1	Буде прийнято байт даних і передано не підтвердження (*NACK*)
		Немає дій	0	0	1	1	1	Буде прийнято байт даних і передано підтвердження ($\overline{ACK}$)
$70	Було прийнято загальний виклик і послано підтвердження ($\overline{ACK}$)	Немає дій	0	0	1	0	1	Буде прийнято байт даних і передано не підтвердження (*NACK*)
		Немає дій	0	0	1	1	1	Буде прийнято байт даних і передано підтвердження ($\overline{ACK}$)
$78	Втрата пріоритету в режимі ведучого під час передачі $SLA+R/\overline{W}$; було прийнято загальний виклик і послано підтвердження ($\overline{ACK}$)	Немає дій	0	0	1	0	1	Буде прийнято байт даних і передано не підтвердження (*NACK*)
		Немає дій	0	0	1	1	1	Буде прийнято байт даних і передано підтвердження ($\overline{ACK}$)
$80	Пристрій уже адресовано: було прийнято байт даних і послано підтвердження ($\overline{ACK}$)	Зчитати дані	0	0	1	0	1	Буде прийнято байт даних і передано *NACK*
		Зчитати дані	0	0	1	1	1	Буде прийнято байт даних і передано ($\overline{ACK}$)

Продовження табл. 7.12.

Код статусу	Стан шини модуля TWI	Дії програми						Наступна дія, що виконується модулем TWI
		в/із TWDR	в регістр TWCR					
			TWSTA	TWSTO	TWINT	TWEA	TWEN	
$88	Пристрій уже адресовано: було прийнято байт даних і послано не підтвердження (*NACK*)	Зчитати дані	0	0	1	0	1	Переключиться в режим не адресованого веденого; розпізнавання викликів відключено
		Зчитати дані	0	0	1	1	1	Переключиться в режим не адресованого веденого; дозволено розпізнавання SLA із власною адресою; дозволено розпізнавання загальних викликів, якщо TWGCE = «1»
		Зчитати дані	1	0	1	0	1	Переключиться в режим не адресованого веденого; розпізнавання будь-яких викликів заборонено; після звільнення шини буде сформовано стан «СТАРТ»
		Зчитати дані	1	0	1	1	1	Переключиться в режим не адресованого веденого; дозволено розпізнавання SLA із власною адресою; дозволено розпізнавання загальних викликів, якщо TWGCE = «1»; після звільнення шини буде сформ. стан «СТАРТ»
$90	Пристрій уже адресовано (загальний виклик): було прийнято байт даних і послано підтвердження ($\overline{ACK}$)	Зчитати дані	0	0	1	0	1	Буде прийнято байт даних і передано *NACK*
		Зчитати дані	0	0	1	1	1	Буде прийнято байт даних і передано $\overline{ACK}$

Продовження табл. 7.12.

Код статусу	Стан шини модуля TWI	Дії програми						Наступна дія, що виконується модулем TWI
		в/із TWDR	в регістр TWCR					
			TWSTA	TWSTO	TWINT	TWEA	TWEN	
$98	Пристрій уже адресовано (загальний виклик): було прийнято байт даних і послано не підтвердження (*NACK*)	Зчитати дані	0	0	1	0	1	Переключиться в режим не адресованого веденого; розпізнавання будь-яких викликів відключено
		Зчитати дані	0	0	1	1	1	Переключиться в режим не адресованого веденого; дозволено розпізнавання SLA із власною адресою; дозволено розпізнавання загальних викликів, якщо TWGCE = «1»
		Зчитати дані	1	0	1	0	1	Переключиться в режим не адресованого веденого; розпізнавання будь-яких викликів заборонено; після звільнення шини буде сформовано стан «СТАРТ»
		Зчитати дані	1	0	1	1	1	Переключиться в режим не адресованого веденого; дозволено розпізнавання SLA із власною адресою; дозволено розпізнавання загальних викликів, якщо TWGCE = «1»; після звільнення шини буде сформовано стан «СТАРТ»

Закінчення табл. 7.12.

Код статусу	Стан шини модуля TWI	Дії програми						Наступна дія, що виконується модулем TWI
		в/із TWDR	в регістр TWCR					
			TWSTA	TWSTO	TWINT	TWEA	TWEN	
$A0	Було виявлено стан «СТОП» або «ПОВСТАРТ» у той час, коли пристрій було адресовано в якості веденого	Немає дій	0	0	1	0	1	Переключиться в режим не адресованого веденого; розпізнавання будь-яких викликів заборонено
		Немає дій	0	0	1	1	1	Переключиться в режим не адресованого веденого; дозволено розпізнавання SLA із власною адресою; дозволено розпізнавання загальних викликів, якщо TWGCE = «1»
		Немає дій	1	0	1	0	1	Переключиться в режим не адресованого веденого; розпізнавання будь-яких викликів заборонено; після звільнення шини буде сформовано стан «СТАРТ»
		Немає дій	1	0	1	1	1	Переключиться в режим не адресованого веденого; дозволено розпізнавання SLA із власною адресою; дозволено розпізнавання загальних викликів, якщо TWGCE = «1»; після звільнення шини буде сформовано стан «СТАРТ»

Щоб перервати потік даних, варто скинути в нуль розряд TWEA регістра TWCR (це можна зробити і під час обміну, тобто при скинутому прапорці TWINT). В результаті після прийому наступного байта на лінію SDA буде видано сигнал не підтвердження, що сигналізує ведучому про те, що ведений не може більше здійснювати прийом даних. Обробка адресних пакетів при скинутому розряді TWEA припиняється, але може бути відновлена в будь-який момент часу повторним встановленням цього розряду.

Якщо адресація пристрою відбудеться при перебуванні мікроконтролера у «сплячому» режимі і розряд TWEN буде при цьому встановлено, модуль TWI переведе мікроконтролер у робочий режим.

У [1] наведено схему алгоритму роботи модуля в даному режимі та його програмну реалізацію.

На рис. 7.22 наведено часові діаграми роботи та різні стани модуля TWI у режимі “Ведений приймач”.

7.3.5. Режим «Ведений передавач»

В режимі «Ведений передавач» ведений пристрій здійснює передачу даних ведучому, який у цьому випадку є приймачем. Перед тим як переключити модуль у цей режим, слід занести у старші розряди регістра TWAR адресу пристрою і відповідно до логіки роботи програми встановити або скинути молодший розряд регістра (TWGCE). Потім необхідно записати в регістр TWCR наступне значення:

TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
1	1	0	0	0	1	0	

Дозволити роботу модуля TWI

Дозволити біт підтвердження

Скинути прапорець TWINT

Після ініціалізації регістрів TWAR і TWCR модуль буде чекати на адресний пакет з адресою, зазначеною в регістрі TWAR. Наступне за адресою значення біта напряму визначить режим, у який переключиться модуль. Якщо цей біт буде встановлено в одиницю (читання), модуль TWI переключиться в режим «Ведений передавач». У іншому випадку модуль переключиться в режим «Ведений приймач». Варто пам’ятати, що пристрій також може автоматично переключитися в режим «Ведений передавач» з режиму ведучого при втраті пріоритету (див. нижче опис коду статусу $B0 в табл. 7.13).

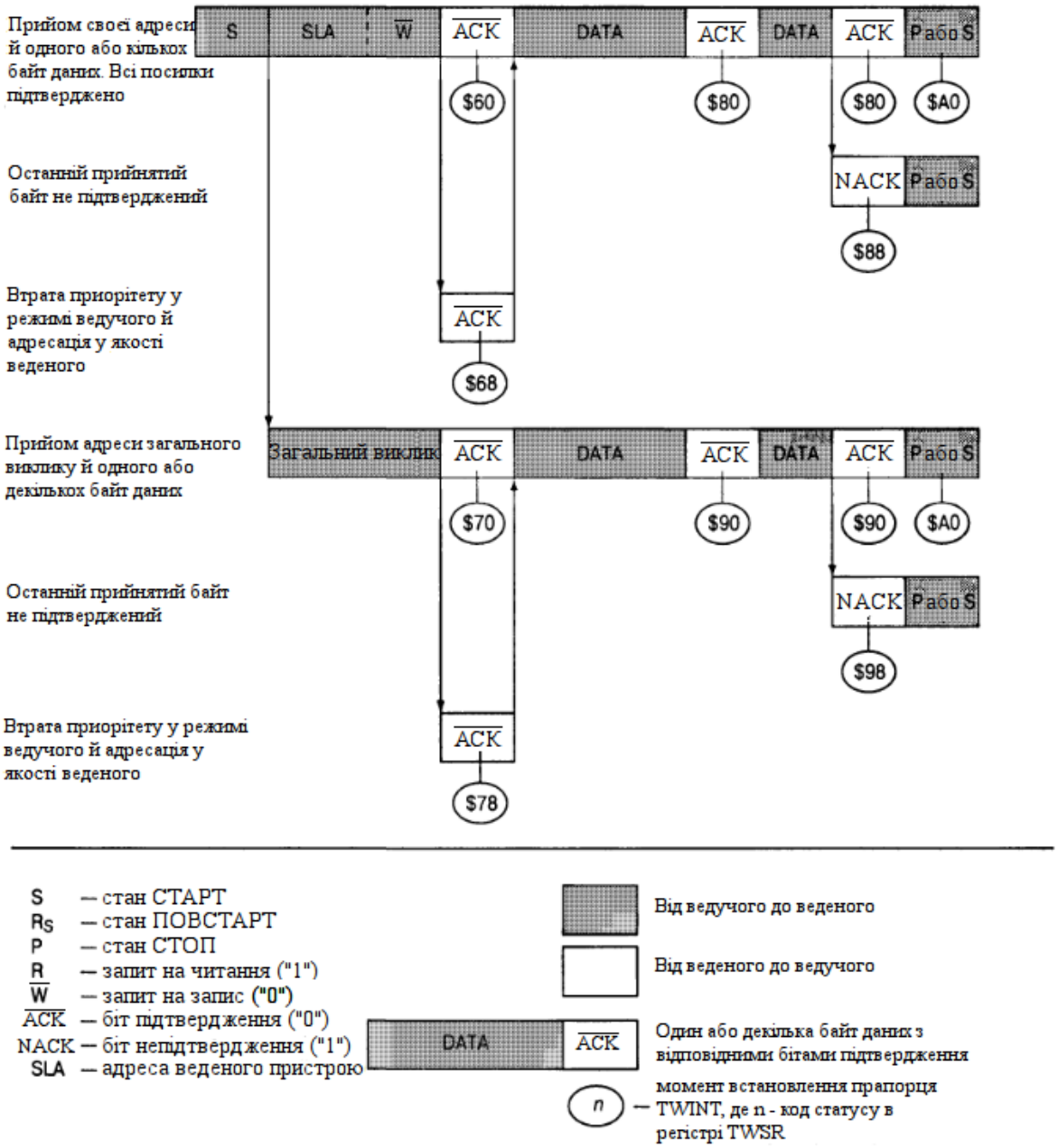


Рис. 7.22. Часові діаграми роботи та стани модуля TWI в режимі “Ведений приймач”

Після прийому пакета $SLA+R$ встановиться прапорець TWINT, і стан обміну можна буде, як зазвичай, визначити за кодом статусу. Можливі дії з боку програми, які відповідають тому або іншому коду статусу, зазначено в табл. 7.13.

При передачі останнього байта даних необхідно скинути в нуль розряд TWEA. Після цього модуль перейде в стан з кодом статусу $C0 або $C8 залежно від того, який сигнал ($\overline{ACK}$ або *NACK*) передасть ведучий у відповідь. У стан з кодом статусу $C8 модуль TWI перейде у випадку, якщо ведучий забажав додаткові дані, передавши підтвердження ($\overline{ACK}$), не зважаючи на те, що ведений

передавач передав останній байт і очікував сигналу *NACK*. Але реакція передавача на стан $C8 подібна реакції на $C0.

Після переходу в кожний із двох останніх станів модуль TWI буде ігнорувати звернення до нього ведучого. Відповідно, якщо ведучий буде продовжувати обмін шиною, він буде постійно приймати значення одиниця. Обробка адресних пакетів при скинутому розряді TWEA також припиняється, але може бути відновлена в будь-який момент часу повторним встановленням цього розряду.

Таблиця 7.13. **Коди статусу для режиму «Ведений передавач»**

Код статусу	**Стан шини модуля TWI**	**Дії програми**						**Наступна дія, що виконується модулем TWI**
		в TWDR	**в регістр TWCR**					
			TWSTA	**TWSTO**	**TWINT**	**TWEA**	**TWEN**	
$A8	Було прийнято $SLA+R$ з власною адресою і послано підтвердження ($\overline{ACK}$)	Завантажити дані	0	0	1	0	1	Було передано останній байт даних; повинно бути отримано не підтвердження (NACK)
		Завантажити дані	0	0	1	1	1	Буде передано черговий байт даних; повинно бути отримано підтвердження ($\overline{ACK}$)
$B0	Втрата пріоритету в режимі ведучого під час передачі $SLA+R/\overline{W}$; був прийнято $SLA+R$ із власною адресою і послано підтвердження ($\overline{ACK}$)	Завантажити дані	0	0	1	0	1	Буде передано останній байт даних; повинно бути отримано не підтвердження (*NACK*)
		Завантажити дані	0	0	1	1	1	Буде передано черговий байт даних; повинно бути отримано підтвердження ($\overline{ACK}$)
$B8	Було передано байт даних і отримано підтвердження ($\overline{ACK}$)	Завантажити дані	0	0	1	0	1	Буде передано останній байт даних; повинно бути отримано не підтвердження (*NACK*)
		Завантажити дані	0	0	1	1	1	Буде передано черговий байт даних; повинно бути отримано підтвердження ($\overline{ACK}$)

Продовження табл. 7.13.

Код статусу	Стан шини модуля TWI	Дії програми						Наступна дія, що виконується модулем TWI
		в TWDR	в регістр TWCR					
			TWSTA	TWSTO	TWINT	TWEA	TWEN	
$C0	Було передано останній байт даних і отримано непідтвердження (*NACK*)	Немає дій	0	0	1	0	1	Переключиться в режим не адресованого веденого; розпізнавання будь-яких викликів заборонено
		Немає дій	0	0	1	1	1	Переключиться в режим не адресованого веденого; дозволено розпізнавання SLA із власною адресою; дозволено розпізнавання загальних викликів, якщо TWGCE = «1»
		Немає дій	1	0	1	0	1	Переключиться в режим не адресованого веденого; розпізнавання будь-яких викликів заборонено; після звільнення шини буде сформовано стан СТАРТ
		Немає дій	1	0	1	1	1	Переключиться в режим не адресованого веденого; дозволено розпізнавання SLA із власною адресою; дозволено розпізнавання загальних викликів, якщо TWGCE = «1»; після звільнення шини буде сформовано стан «СТАРТ»

Закінчення табл. 7.13.

Код статусу	Стан шини модуля TWI	Дії програми						Наступна дія, що виконується модулем TWI
		в TWDR	в регістр TWCR					
			TWSTA	TWSTO	TWINT	TWEA	TWEN	
$C8	Було передано останній байт даних і отримано підтвердження ($\overline{ACK}$)	Немає дій	0	0	1	0	1	Переключиться в режим не адресованого веденого; розпізнавання будь-яких викликів заборонено
		Немає дій	0	0	1	1	1	Переключиться в режим не адресованого веденого; дозволено розпізнавання SLA із власною адресою; дозволено розпізнавання загальних викликів, якщо TWGCE = «1»
		Немає дій	1	0	1	0	1	Переключиться в режим не адресованого веденого; розпізнавання будь-яких викликів заборонено; після звільнення шини буде сформовано стан СТАРТ
		Немає дій	1	0	1	1	1	Переключиться в режим не адресованого веденого; дозволено розпізнавання SLA із власною адресою; дозволено розпізнавання загальних викликів, якщо TWGCE = «1»; після звільнення шини буде сформовано стан «СТАРТ»

У [1] наведено схему алгоритму роботи модуля в даному режимі та його програмну реалізацію.

На рис. 7.23 наведено часові діаграми роботи та різні стани модуля TWI в режимі «Ведений передавач».

7.3.6. Комбінування різних режимів

На практиці при виконанні чергової операції шиною TWI пристрою доводиться використовувати кілька режимів, переключаючись між ними при необхідності. Як приклад нижче розглянуто операцію читання даних із зовнішньої пам’яті даних типу EEPROM.

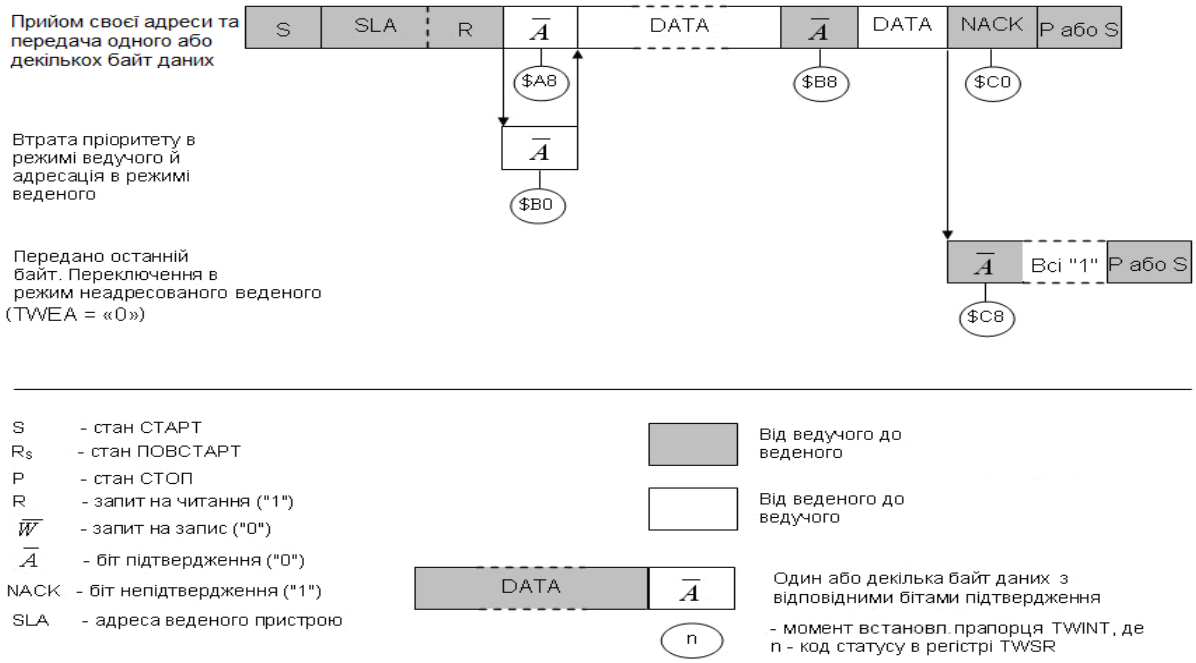


Рис. 7.23. Часові діаграми роботи та стани модуля TWI в режимі «Ведений передавач»

При виконанні операції відбувається передача інформації як від ведучого до веденого, так і навпаки. Після ініціювання обміну шиною ведучий повинен перебувати в режимі «Ведучий передавач», щоб повідомити веденому адресу комірки пам’яті, за якою він має намір здійснити читання. Для виконання наступного етапу операції (читання даних) ведучий повинен переключитися в режим «Ведучий приймач». При цьому він повинен зберігати контроль над шиною під час виконання всіх етапів операції. Для цього використовується стан «ПОВСТАРТ». Ведучий формує цей стан між передачею адреси і прийомом даних, як показано на рис. 7.24.

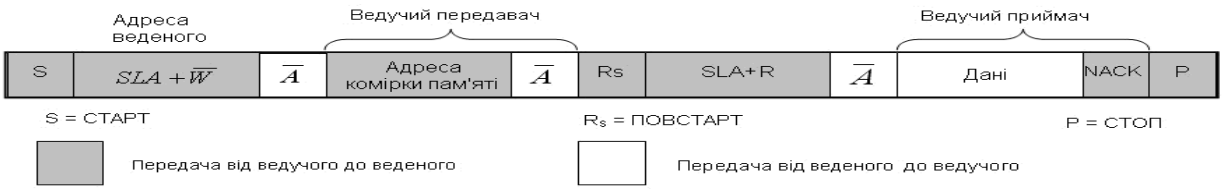


Рис. 7.24. Приклад звернення до зовнішньої пам’яті даних типу EEPROM

7.3.7. Арбітраж

У випадку присутності на шині декількох ведучих можлива ситуація, при якій декілька ведучих одночасно почнуть процес обміну. Для таких випадків специфікацією TWI передбачено процес розподілу пріоритетів (арбітраж), в результаті виконання якого на шині залишається тільки один ведучий пристрій.

Принципи цього процесу було розглянуто в 7.1, однак його виконання може розвиватися за різними сценаріями [1]:

1) Два або більше ведучих здійснюють однотипний обмін з одним і тим самим веденим. У цьому випадку ніхто з них не зможе розпізнати конфлікт на шині.

2) Два або більше ведучих звертаються до одного й того ж самого веденого з різними даними або різними типами обміну (читання/запис). У цьому випадку розподіл пріоритетів відбудеться під час передачі бітів даних або біта напряму. Ведучий, що втратив пріоритет, може або переключитися в режим не адресованого веденого, або дочекатися звільнення шини і сформувати стан «СТАРТ» для її захоплення.

3) Два або більше ведучих звертаються до різних ведених. У цьому випадку розподіл пріоритетів почнеться під час передачі бітів адресного пакета. Ведучий, що втратив пріоритет, переключиться в режим веденого для перевірки, чи не був він адресований іншим ведучим. Якщо колишній ведучий був адресований, він переключиться в режим «Ведений передавач» або «Ведений приймач» залежно від значення біта напряму. Якщо ж він не був адресований, він переключиться в режим не адресованого веденого, або дочекається звільнення шини і сформує новий стан «СТАРТ».

7.4. Моделювання інтерфейсу TWI у пакеті PROTEUS 8.6

7.4.1. Опис моделі

Схему моделювання інтерфейсу TWI у пакеті Proteus 8.6 наведено на рис. 7.25.

В моделі використано мікроконтролер ATmega16. Резистори R1 та R2 підключають (підтягують) лінії SDA та SCL шини I^2C до джерела живлення. Осцилограф, I^2C-відлагоджувач та EEPROM-пам’ять даних із вбудованим інтерфейсом I^2C – 24LC643 підключено до моделі зі вкладки «віртуальні інструменти». Мікросхема EEPROM-пам’ять (U2) має 8 виводів, які, крім живлення і землі, відображено на моделі.

В налаштуваннях мікроконтролера вказано частоту тактування 16 МГц і виставлено CKSEL-фьюзі-біти для кристала [24]. В налаштуваннях I^2C-відлагодувача вказано, що тактова частота імпульсів шини дорівнює 400 КГц.

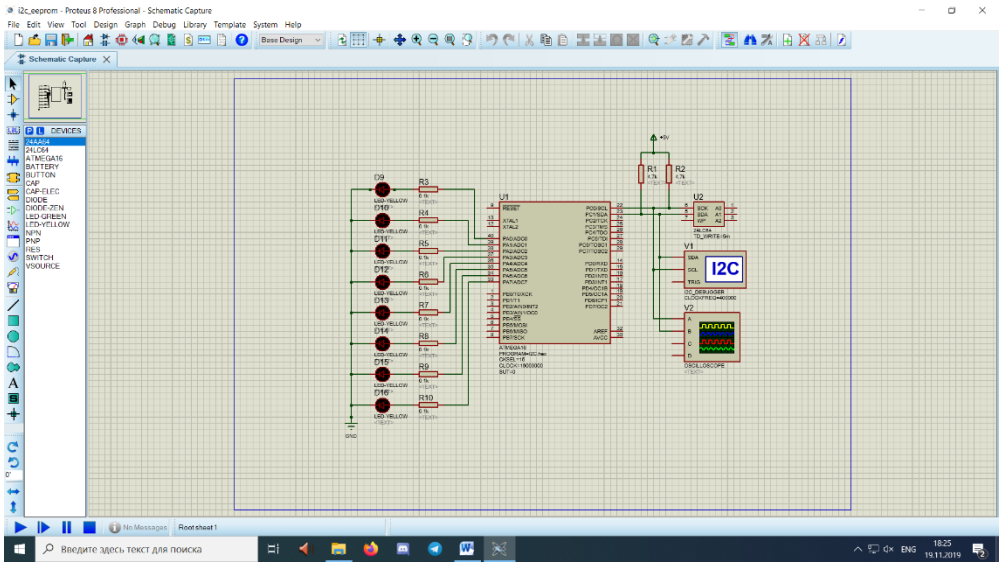


Рис. 7.25. Схема моделювання інтерфейсу I^{2}C у пакеті Proteus 8.6

7.4.2. Налаштування частоти тактових імпульсів лінії SCL

Як було зазначено вище, для мікроконтролера ATmega16 частота сформованого сигналу SCL визначається виразом (7.2). Підставляючи у цю формулу частоти f_{CLK}; f_{SCL} та значення 4^{TWPS}, ми зможемо знайти значення для регістра TWBR. Для обраної в моделі мікросхеми EEPROM-пам'яті тактова частота імпульсів шини f_{SCL} дорівнює 400 КГц. Тактова частота мікроконтролера f_{CLK} = 16 МГц. При заданні $TWPS_{10}$ значення нуль (табл. 7.14) отримаємо $4^{TWPS} = 1$.

Для вибору значення регістра TWBR можна використати таблицю 7.15.

Таблиця 7.14. **Розрахунок значення переддільника швидкості обміну шиною**

TWPS1	TWPS0	$TWPS_{10}$	Значення переддільника 4^{TWPS}
0	0	0	1
0	1	1	4
1	0	2	16
1	1	3	64

7.4.3. Запуск процесу моделювання

На рис. 7.26 наведено стан моделі після запуску процесу моделювання (після натискання клавіші «START»).

Таблиця 7.15. **Вибір значення регістра TWBR**

Частота МК, МГц	TWBR	TWPS	Частота SCL, КГц
16.0	12	0	400
16.0	72	0	100
14.4	10	0	400
14.4	64	0	100
12.0	52	0	100
8.0	32	0	100
4.0	12	0	100
3.6	10	0	100

Адреса веденого пристрою (EEPROM-пам`яті) в нашому прикладі дорівнює: 0x19, а байт даних, який спочатку записується, а потім зчитується за цією адресою, дорівнює: 0x0A.

При налаштуванні схеми необхідно перевести світлодіоди у цифровий режим. Для цього треба двічі клікнути на них і обрати необхідний режим (рис. 7.27).

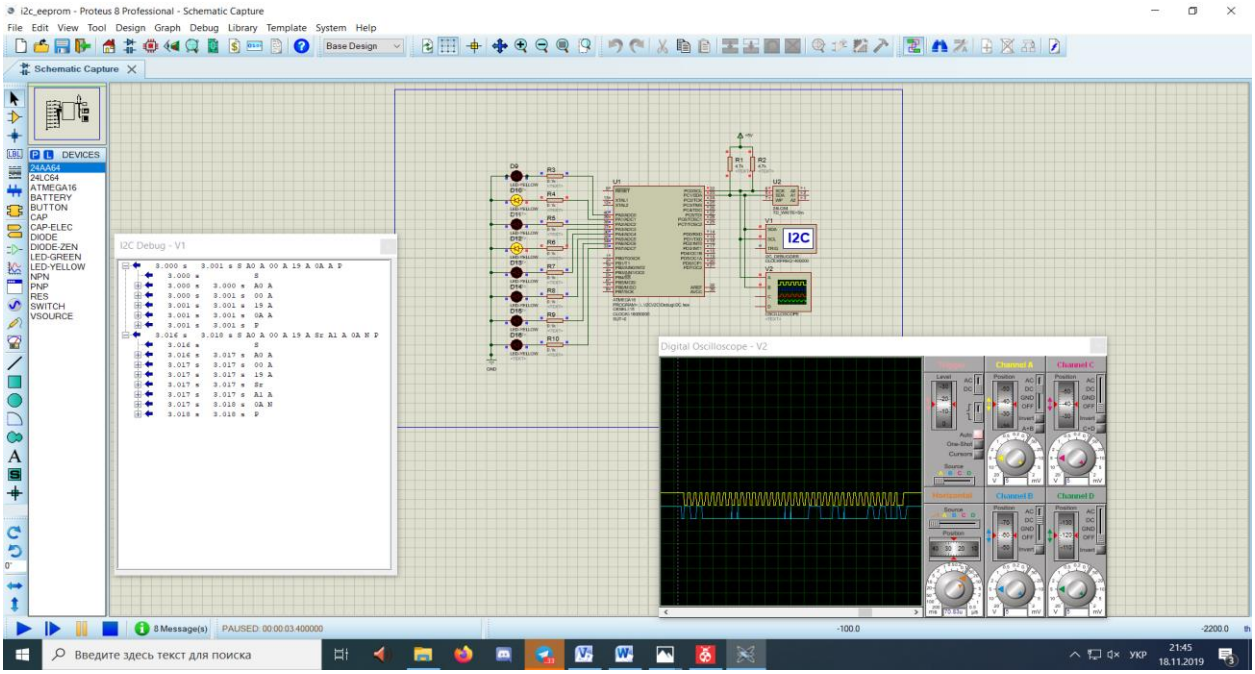


Рис. 7.26. Стан моделі після запуску процесу моделювання

Як бачимо на рис. 7.26, що в PORTA, до якого підключено 8 світлодіодів, записано число 0b0000'1010 = 0x0A = 10. Тобто після читання пам’яті 24LC64 за адресою 0x19 змінна uint8_t byte (див. програму, яку наведено нижче), містить вірне значення.

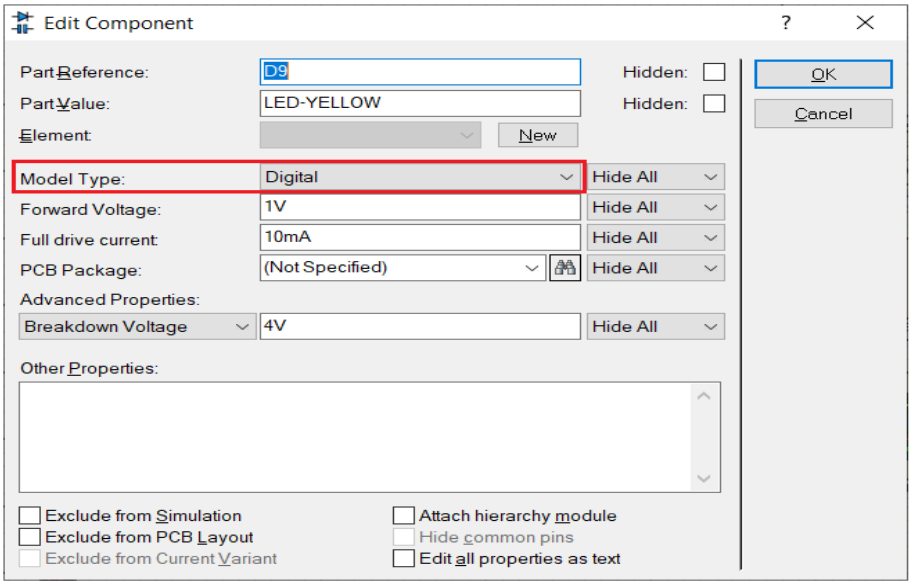
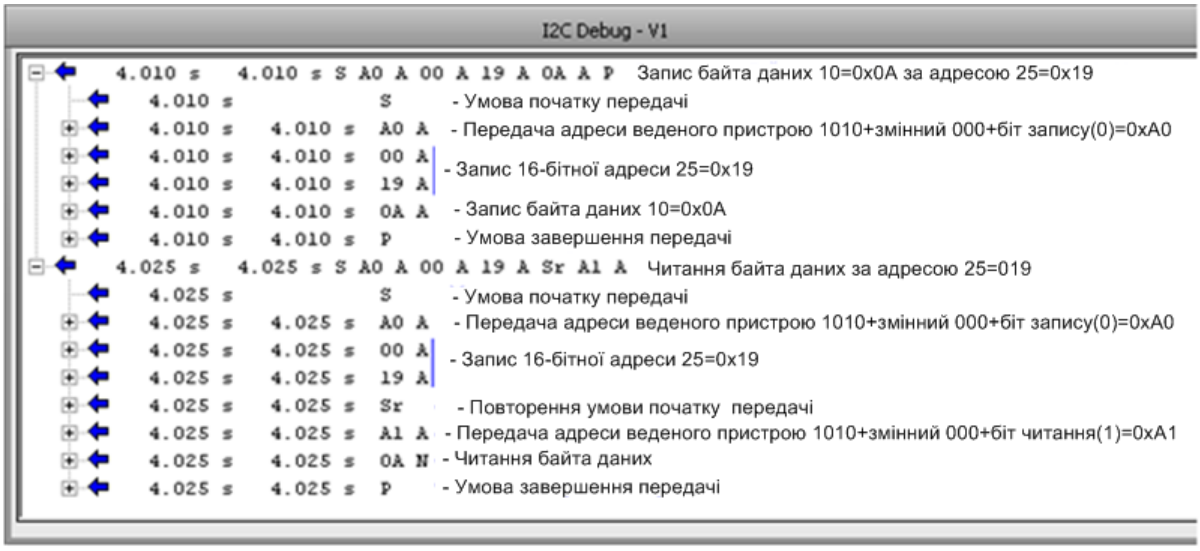


Рис. 7.27. Налаштування світлодіодів

У вікні I²C Debug наведено пояснення окремих етапів обміну шиною TWI (I²C) після запуску моделювання (рис. 7.28).

Часові діаграми обміну між мікроконтролером та EEPROM-пам`яттю, які отримано на екрані осцилографа наведено на рис. 7.29.

Як видно з цих рисунків, результати моделювання підтверджують теоретичні відомості, які наведено вище на рис. 7.24.



A – підтвердження від ведучого або веденого, N – відсутність підтвердження

Рис. 7.28. Пояснення окремих етапів обміну шиною TWI (I²C)

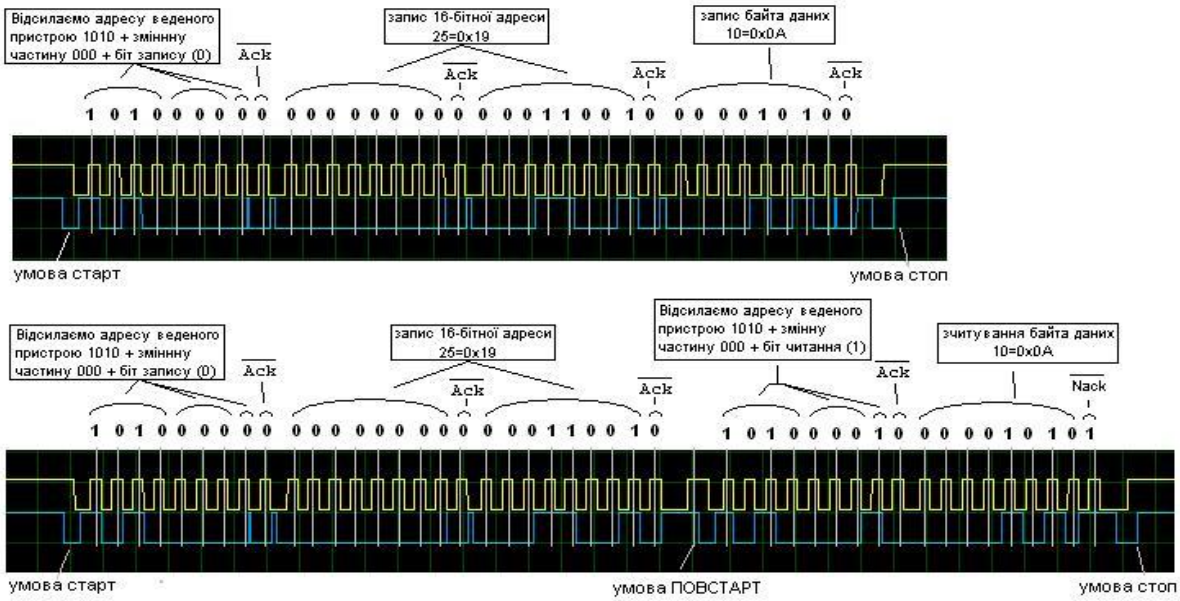


Рис. 7.29. Часові діаграми обміну між мікроконтролером та EEPROM-пам`яттю, отриманих на екрані осцилографа

7.4.4. Схема алгоритму роботи моделі

Схему алгоритму роботи моделі наведено на рис. 7.30.

Згідно з описом роботи моделі, який наведено вище, спочатку МК передає веденому пристрою (EEPROM-пам`яті) байт даних 0x10 за адресою 0x19, а потім читає вміст комірки EEPROM-пам`яті з тією самою адресою. Отримання в цьому випадку значення 0x10 говорить про те, що модель працює правильно. Через те, що МК моделі спочатку працює як ведучий передавач, а потім, як ведучий приймач, а EEPROM-пам`ять спочатку виконує функцію веденого приймача, а потім веденого передавача, нижче на рисунках 7.31 та 7.32 наведено відповідно схеми алгоритів запису у EEPROM-пам’ять та читання EEPROM-пам`яті.

Рис. 7.30. Схема алгоритму роботи моделі

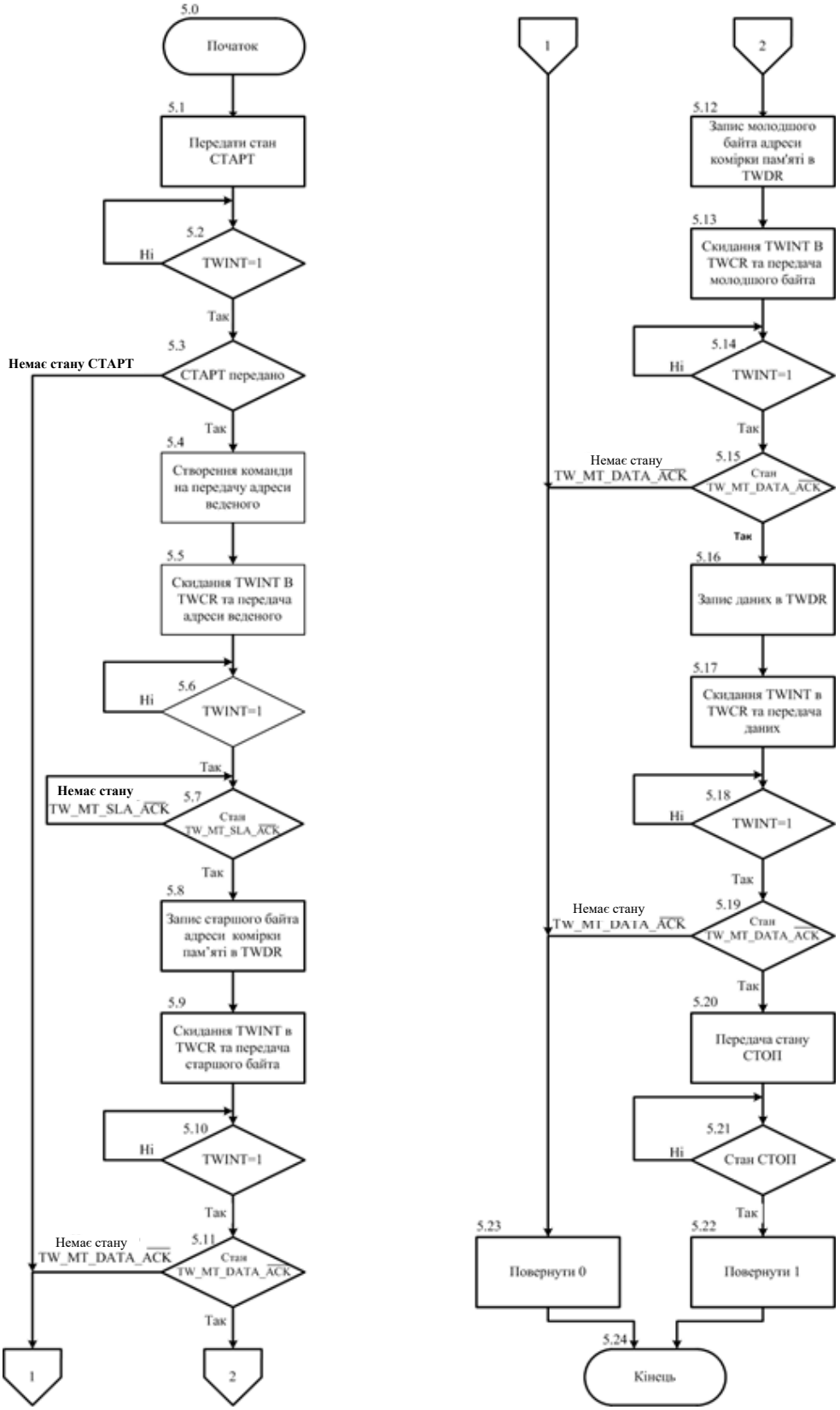


Рис. 7.31. Схема алгоритму запису в EEPROM-пам’ять

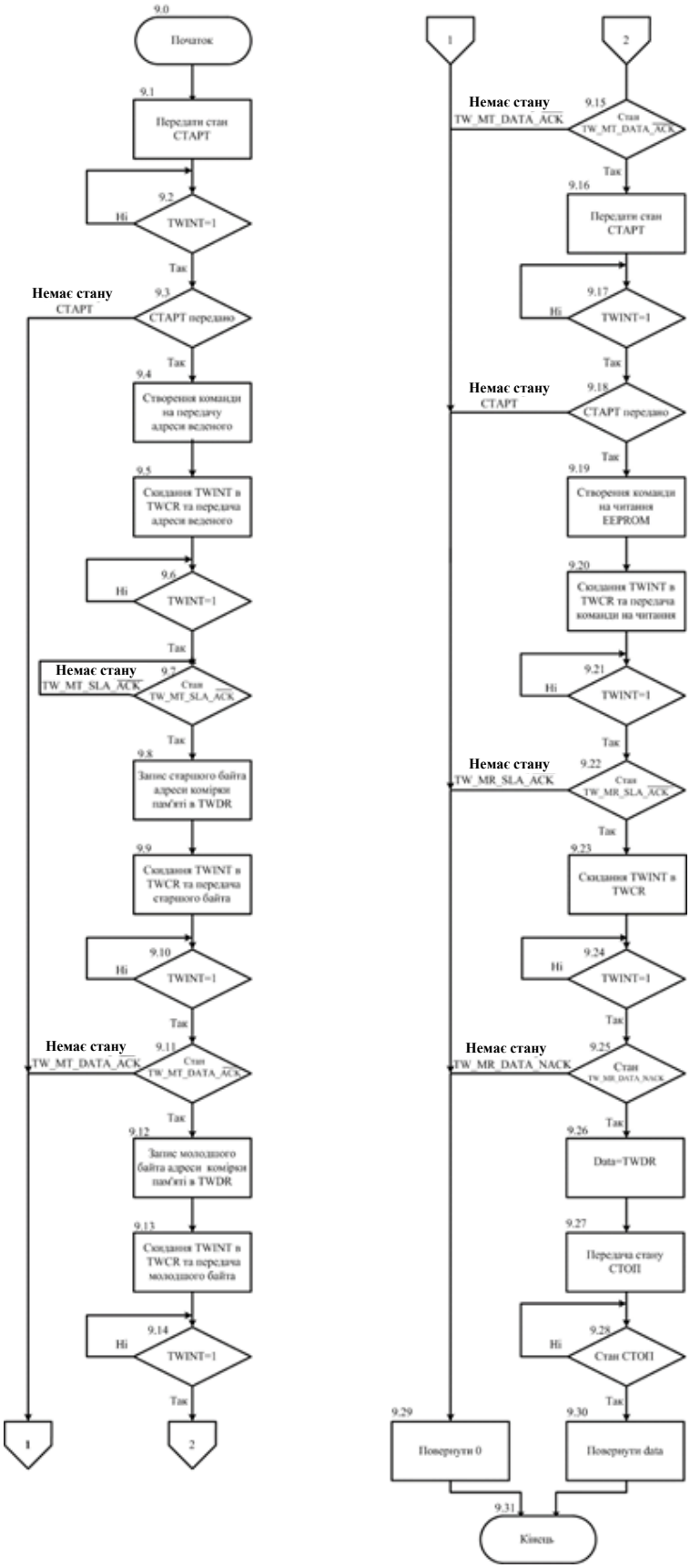


Рис. 7.32. Схема алгоритму читання EEPROM-пам’яті

7.4.5. Робоча програма

Всі функції (ініціалізація TWI, запис зовнішньої пам’яті та читання пам’яті) винесено в окремі файли: i2c_eeprom.h та i2c_eeprom.c.

Зміст i2c_eeprom.h

```
#define false 0
#define true 1

#define slaveF_SCL 400000 //400 Khz

#define slaveAddressConst 0b1010
#define slaveAddressVar 0b000

#define READFLAG 1 // read
#define WRITEFLAG 0 // write

void eeInit(void);
uint8_t eeWriteByte(uint16_t address,uint8_t data);
uint8_t eeReadByte(uint16_t address);

                                    // Master
#define TW_START            0x08 // Було сформовано стан СТАРТ
#define TW_REP_START        0x10 // Було сформовано стан ПОВСТАРТ

// Master Transmitter

#define TW_MT_SLA_ACK           0x18 // Було передано стан SLA+W і
                                     // прийнято підтвердження ACK
#define TW_MT_SLA_NACK          0x20 // Було передано пакет SLA+W і
                                     // прийнято непідтвердження NACK
#define TW_MT_DATA_ACK          0x28 // Було передано пакет даних і
                                     // прийнято підтвердження ACK
#define TW_MT_DATA_NACK         0x30 // Було передано пакет даних і
                                     // прийнято непідтвердження NACK
#define TW_MT_ARB_LOST          0x38 // Втрата пріоритету при передачі
                                     // пакета адреси або даних
// Master Receiver

#define TW_MR_ARB_LOST          0x38 // Втрата пріоритету при передачі
                                     // пакета адреси або даних
#define TW_MR_SLA_ACK           0x40 // Було передано пакет SLA+R і
                                     // прийнято підтвердження ACK
```

```
#define TW_MR_SLA_NACK        0x48 // Було передано пакет SLA+R і
                                   // прийнято непідтвердження NACK
#define TW_MR_DATA_ACK        0x50 // Було прийнято байт даних і
                                   // передано підтвердження ACK
#define TW_MR_DATA_NACK       0x58 // Було прийнято байт даних і
                                   // передано не підтвердження NACK
```

// Slave Transmitter

```
#define TW_ST_SLA_ACK         0xA8 // Було прийнято SLA+R з власною
                                   // адресою і послано підтвердження ACK
#define TW_ST_ARB_LOST_SLA_ACK  0xB0  // Втрата пріоритету в
                                   // режимі ведучого
                                   // під час передачі SLA+R/W
#define TW_ST_DATA_ACK        0xB8 // Було передано байт даних і
                                   // передано підтвердження ACK
#define TW_ST_DATA_NACK       0xC0  // Було передано останній байт
                                   //даних і отримано
                                   //не підтвердження NACK
#define TW_ST_LAST_DATA       0xC8 // Було передано останній байт даних
                                   // і отримано підтвердження ACK
```

// Slave Receiver

```
#define TW_SR_SLA_ACK         0x60 // Було прийнято SLA+W з власною
                                   // адресою і послано підтвердження ACK
#define TW_SR_ARB_LOST_SLA_ACK  0x68 // Втрата пріоритету в
                                   // режимі ведучого
                                   // під час передачі SLA+R/W
#define TW_SR_GCALL_ACK       0x70 // Було прийнято загальний виклик і
                                   // послано підтвердження ACK
#define TW_SR_ARB_LOST_GCALL_ACK  0x78 // Втрата пріоритету в
                                   // режимі ведучого
                                   // під час передачі SLA+R/W
#define TW_SR_DATA_ACK        0x80 // Пристрій уже адресовано: було
                                   // прийнято байт даних і
                                   / /послано підтвердження ACK
#define TW_SR_DATA_NACK       0x88 // Пристрій уже адресовано:
                                   // було прийнято байт даних і
                                   // послано непідтвердження NACK
#define TW_SR_GCALL_DATA_ACK  0x90 // Пристрій уже адресовано
                                   // (загальний виклик):
                                   // було прийнято байт даних і
                                   / /послано підтвердження ACK
#define TW_SR_GCALL_DATA_NACK 0x98 // Пристрій уже адресовано
                                   // (загальний виклик): було прийнято
```

```
                                                    // байт даних та послано
                                                    // непідтвердження NACK
#define TW_SR_STOP      0xA0 // Було виявлено стан СТОП або ПОВСТАРТ
                             // у той час, коли пристрій було
                             // адресовано в якості веденого
```

// Misc

```
#define TW_NO_INFO              0xF8 // Немає інформації TWINT = 0
#define TW_BUS_ERROR            0x00 // Помилка на шині в результаті
                                     //не коректного формування стану
                                     //СТАРТ або СТОП
```

При написанні програми мовою С у змінні: TW_MT_SLA_ACK; TW_MT_DATA_ACK та TW_MR_SLA_ACK входить скорочення ACK (acknowledge) – підтвердження. Це скорочення відображає отримання сигналу низького рівня (підтвердження) після передачі адреси або даних (рис. 7.10, 7.11).

Щоб підкреслити активний низький рівень цього сигналу на рисунках 7.30...7.32 він помічений інверсією. Але при написанні програм мовою С використання інверсії заборонено. Тому названі вище змінні мають скорочення ACK без інверсії.

Лістинг файлу i2c_eeprom.c

```
#include <avr/io.h>
#include <util/delay.h>

#include "i2c_eeprom.h"

void eeInit(void)
{
   /* Налаштування генератора швидкості зв'язку */
   TWBR = (F_CPU/slaveF_SCL - 16)/(2 * /* TWI_Prescaler= 4^TWPS */1);

/*
Якщо TWI працює у ведучому режимі, то значення TWBR повинно бути не
менше 10. Якщо значення TWBR менше 10, то ведучий пристрій шини може
генерувати не коректні сигнали на лініях SDA і SCL під час передачі байта.*/

   if(TWBR < 10)
      TWBR = 10;

   /*
```

Налаштування переддільника у регістрі статусу блоку керування.
Очищаються біти TWPS0 і TWPS1 регістра статусу, встановлюючи тим самим, значення переддільника = 1.

```
  */
   TWSR &= (~((1<<TWPS1)|(1<<TWPS0)));
}

uint8_t eeWriteByte(uint16_t address,uint8_t data)
{
   do
   {
        TWCR=(1<<TWINT)|(1<<TWSTA)|(1<<TWEN);  // 5.1 Передача стану
                                                // СТАРТ

        while(!(TWCR & (1<<TWINT))); // 5.2 Очікування встановлення TWINT

        if((TWSR & 0xF8) != TW_START)         // 5.3 Перевірка регістру стану
                                              // 5.3 return false;

TWDR = (slaveAddressConst<<4) +  // 5.4 Створення команди на
(slaveAddressVar<<1) + (WRITEFLAG);  // 5.4 передачу

TWCR=(1<<TWINT)|(1<<TWEN);  // 5.5 Скидання TWINT в TWCR та
                            // передача адреси веденого


while(!(TWCR & (1<<TWINT))); // 5.6 Очікування встановлення TWINT

   }while((TWSR & 0x18) != TW_MT_SLA_ACK);   // 5.7 Перевірка стану
                                             // 5.7 TW_MT_SLA_ACK
 TWDR=(address>>8);            // 5.8 Завантаження старшого
                               // байта адреси комірки пам’яті у TWDR

TWCR=(1<<TWINT)|(1<<TWEN);            // 5.9 Скидання TWINT та
                                      / /передача старшого байта адреси
                                      // комірки в EEPROM

 while(!(TWCR & (1<<TWINT)));   // 5.10 Очікування встановлення TWINT


if((TWSR & 0x28) != TW_MT_DATA_ACK)       // 5.11 Перевірка стану
                                          //  TW_MT_DATA_ACK

return false;                             //5.23
```

```
  TWDR=(address);                          // 5.12 Завантаження молодшого
                                           // байта адреси комірки пам’яті у TWDR

  TWCR=(1<<TWINT)|(1<<TWEN);               // 5.13 Скидання TWINT та
                                           // передача молодшого байта
                                           // адреси комірки в EEPROM
  while(!(TWCR & (1<<TWINT)));    // 5.14 Очікування встановлення TWINT
   if((TWSR & 0x28) != TW_MT_DATA_ACK)       // 5.15 Перевірка стану
                                             // TW_MT_DATA_ACK
       return false;                         // 5.23

              /***** Передача даних в EEPROM ********/


   TWDR=(data);                          // 5.16 Завантаження даних у TWDR

   TWCR=(1<<TWINT)|(1<<TWEN);                // 5.17 Скидання TWINT та
                                             // передача даних в EEPROM
   while(!(TWCR & (1<<TWINT)));  // 5.18 Очікування встановлення TWINT

   if((TWSR & 0x28) != TW_MT_DATA_ACK)       // 5.19 Перевірка стану
                                             // TW_MT_DATA_ACK
return false;                                // 5.23


TWCR=(1<<TWINT)|(1<<TWEN)|(1<<TWSTO); // 5.20 Передача умови СТОП
   while(TWCR & (1<<TWSTO));      // 5.21 Очікування передачі умови СТОП

   return true;                          //5.22
}

uint8_t eeReadByte(uint16_t address)
{
   uint8_t data; // Змінна, до якої записується прочитаний байт

              /***** Встановлення зв'язку із веденим *****/
   do
   {
      TWCR=(1<<TWINT)|(1<<TWSTA)|(1<<TWEN);  // 9.1 Передача стану
СТАРТ

      while(!(TWCR & (1<<TWINT)));   // 9.2 Очікування встановлення TWINT
```

```
    if((TWSR & 0xF8) != TW_START)         // 9.3 Перевірка регістру стану
      return false;                       // 9.29

     // 9.4 Створення команди на передачу
    TWDR = (slaveAddressConst<<4) +(slaveAddressVar<<1) + WRITEFLAG;

    TWCR=(1<<TWINT)|(1<<TWEN);    // 9.5 Скидання TWINT в TWCR та
                                  // передача адреси веденого
    while(!(TWCR & (1<<TWINT)));  // 9.6 Очікування встановлення TWINT

  }while((TWSR & 0x18) != TW_MT_SLA_ACK);    // 9.7 Перевірка стану
                                             // TW_MT_SLA_ACK

            /*****Передача адреси читання********/

  TWDR=(address>>8);                  // 9.8 Завантаження старшого
                                      // байта адреси комірки EEPROM у TWDR
TWCR=(1<<TWINT)|(1<<TWEN);            // 9.9 Скидання TWINT та
                                      // передача старшого байта адреси комірки EEPROM
  while(!(TWCR & (1<<TWINT)));    // 9.10 Очікування встановлення TWINT

  if((TWSR & 0x28) != TW_MT_DATA_ACK)        // 9.11 Перевірка стану
                                             // TW_MT_DATA_ACK
return false;                                // 9.29
  TWDR=(address);                     // 9.12 Завантаження молодшого байта
                                      // адреси комірки пам'яті у TWDR
  TWCR=(1<<TWINT)|(1<<TWEN);  // 9.13 Скидання TWINT та передача
                              // молодшого байта адреси комірки
                              // в EEPROM
  while(!(TWCR & (1<<TWINT)));   // 9.14 Очікування встановлення TWINT

  if((TWSR & 0xF8) != TW_MT_DATA_ACK)   // 9.15 Перевірка стану
                                        // TW_MT_DATA_ACK
    return false;                       // 9.29

            /*****Перехід у режим читання********/

/* Знову зв'язок з веденим, тому що раніше було передано адресний пакет
(slaveAddressConst <<4) + (slaveAddressVar <<1) + WRITEFLAG, щоб записати
адресу читання байта даних. Тепер потрібно перейти у режим читання, щоб
прочитати байт даних), для цього передаємо новий пакет
(slaveAddressConst <<4) + (slaveAddressVar <<1) + READFLAG .*/

  // Повторення умови початку передачі
```

```
TWCR=(1<<TWINT)|(1<<TWSTA)|(1<<TWEN);     // 9.16 Передача стану
                                          //Повторний старт
   while(!(TWCR & (1<<TWINT)));                // 9.17 Очікування
TWINT=1

   if((TWSR & 0x10) != TW_REP_START)          // 9.18 Перевірка регістру
стану
      return false;                           // 9.29

  // 9.29 Створення команди на читання
   TWDR = (slaveAddressConst<<4) + (slaveAddressVar<<1) + READFLAG;

        //Передача
TWCR=(1<<TWINT)|(1<<TWEN);          // 9. 20 Скидання TWINT та передача
                                    // адреси веденого
while(!(TWCR & (1<<TWINT)));     // 9. 21 Очікування встановлення TWINT

/* Перевірка, що знайшовся ведений з адресою 1010'000 і він готовий до читання
*/
   if((TWSR & 0x40) != TW_MR_SLA_ACK)       // 9. 22 Перевірка стану
                                            // TW_MR_SLA_ACK
return false;                               // 9. 29


              /*****Прийом  байта даних********/

/* Початок прийому даних шляхом очищення прапорця переривання TWINT.
Байт, який буде прийнятий, записується у регістр TWDR.*/
   TWCR=(1<<TWINT)|(1<<TWEN);               // 9.23 Скидання TWINT та
                                            // встановлення TWEN=1

   //Чекання закінчення прийому.
   while(!(TWCR & (1<<TWINT)));             // 9.24 Чекання закінчення
прийому

/* Перевірка регістра статусу. За протоколом, прийом даних повинен
закінчуватися без підтвердження з боку ведучого (TW_MR_DATA_NACK =
0x58)*/
   if((TWSR & 0x58) != TW_MR_DATA_NACK)  // 9.25 Перевірка регістра
статусу
      return false;                         // 9.29

   data=TWDR;                          // 9.26 Надання змінній data значення TWDR
```

```
TWCR=(1<<TWINT)|(1<<TWEN)|(1<<TWSTO);  // 9.27 Встановлення умови СТОП

   while(TWCR & (1<<TWSTO));    // 9.28 Чекання встановлення умови СТОП

   return data;                 // 9.30 Повернення прочитаного байта
}                               // 9.31
```

7.4.6. Створення проекту в AVR STUDIO

У налаштуваннях проекту вказують ATmega16 та підключають файли i2c_eeprom.c та i2c_eeprom.h (рис. 7.33).

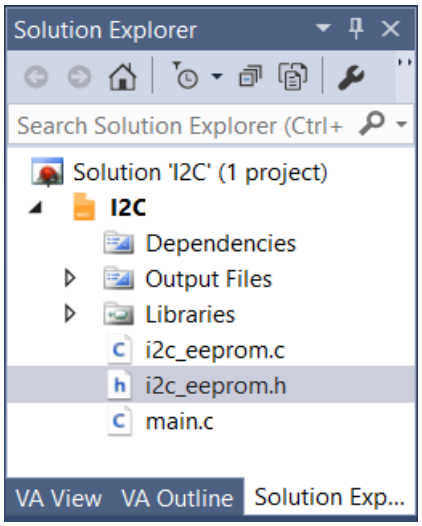


Рис. 7.33. Структура проекту

Зміст файлу main.c

```
#define F_CPU 16000000UL

#include <avr/io.h>                     // 1
#include "util/delay.h"
#include "i2c_eeprom.h"

int main(void)
{
        uint8_t byte = 10;              // 2
        uint16_t address = 25;          // 2

/*Треба почекати, тому що віртуальний осцилограф у Proteus включається із затримкою */
        _delay_ms(300);                 // 3
```

```
	// Налаштовання TWI
	eeInit();		// 4

	// Запис байта даних 10 = 0xA0 за адресою 25 = 0x19
	// Якщо запис успішний, то повернеться true
	if(eeWriteByte(address, byte))		// 5; 6
	{
		// Очищення змінної
		byte = 0;		// 7

		// Чекаєння 15 с, щоб візуально розрізнити осцилограми
		_delay_ms(15);		// 8

		// Читання байта, розташованого за адресою 25 = 0x19
		byte = eeReadByte(address);		// 9

		// Перевірка виведення цього байта у порт виведення (PORTA)

		PORTA = byte;		// 10
	}
}		// 11
```

Контрольні запитання та завдання

1. Якою фирмою було розроблено інтерфейс I^2C та для чого він призначений?
2. Чим відрізняється інтерфейс I^2C від інтерфейсу TWI?
3. З яких двох ліній складається шина інтерфейсу I^2C (TWI) та як вони повінні бути піключені в мережі?
4. Поясніть призначення функції «монтажне АБО/І» та як вона використовується в I^2C (TWI)-мережі.
5. Наведіть та поясніть структурну схему типової мережі, яка використовує для обміну даними шину I^2C (TWI).
6. Опишіть роботу пристроїв, які використовують для обміну інтерфейс I^2C (TWI).
7. Опишіть чотири можливі режими обміну даними інтерфейсом I^2C (TWI).
8. Поясніть призначення станів «СТАРТ» і «СТОП» шини TWI. Яким чином вони формуються.
9. У чому полягає задача розподілу пріоритетів при підключенні до шини TWI декількох ведучих пристроїв? Як вона вирішується?

10. Опишіть формат адресного пакету та пакету даних.
11. Наведіть та поясніть структурну схему типового модуля TWI.
12. Що задає контролер швидкості передачі?
13. Які два вузли входять до складу блоку шинного інтерфейсу? Які їх функції?
14. Що перевіряє блок контролю адреси?
15. При виникненні яких подій блок керування здійснює формування запиту на переривання?
16. Назвіть та опишіть використання регістрів керування TWI-модулем.
17. На чому базується взаємодія прикладної програми з модулем TWI?
18. З яких трьох частин складається будь-який етап взаємодії прикладної програми з модулем TWI? Поясніть цю взаємодію.
19. Назвіть деякі пристрої, які підтримують інтерфейс TWI.
20. У яких режимах може працювати модуль TWI, реалізований у мікроконтролерах сімейства Mega?
21. Опишіть роботу модуля TWI у режимі «Ведучий передавач».
22. Опишіть роботу модуля TWI у режимі «Ведучий приймач».
23. Опишіть роботу модуля TWI у режимі «Ведений приймач».
24. Опишіть роботу модуля TWI у режимі «Ведений передавач».
25. Які можливості надає використання стану «ПОВСТАРТ»?
26. Що трапиться, якщо адресація пристрою відбудеться при перебуванні мікроконтролера у «сплячому» режимі?
27. За якими сценаріями може розвиватися процес арбітражу?
28. З якою швидкістю може виконуватися обмін інформацією в I^2C (TWI)-мережі?
29. На яку відстань може передаватися інформація в I^2C (TWI)-мережі?
30. Скільки ведучих пристроїв може одночасно бути I^2C (TWI)-мережі?
31. Опишіть схему моделювання інтерфейсу TWI у пакеті Proteus 8.6.
32. Опишіть схему алгоритму роботи моделі.
33. Опишіть схему алгоритму запису в EEPROM-пам`ять.
34. Опишіть схему алгоритму читання EEPROM-пам`яті.
35. Поясніть окремі етапи обміну шиною TWI (I2C), які отримано у вікні I2C Debug після запуску моделювання.
36. Поясніть часові діаграми обміну між мікроконтролером та EEPROM-пам`яттю, отриманих на екрані осцилографа після запуску моделювання.

8. ПРОЕКТУВАННЯ МОДУЛЯ SPI

8.1. Загальні відомості

В мікропроцесорних системах крім задачі передачі неперервного потоку інформації, достатньо часто необхідно передавати окремі цифрові пакети чи команди керування. Саме для передачі такого виду інформації призначено синхронний послідовний периферійний інтерфейс – SPI (Serial Peripheral Interface) [1; 7].

Шину інтерфейсу SPI було розроблено компанією Motorola наприкінці 1980-х для мікроконтролерів серії 68000. Завдяки простоті та популярності цієї шини, багато інших виробників вже багато років використовують цей стандарт.

Всі мікроконтролери сімейства Mega та XMega мають інтерфейс SPI, який має подвійне призначення. По-перше, з його допомогою може здійснюватися обмін даними на відстані до 3-ох метрів зі швидкістю до 2 Мбіт/с між мікроконтролером і різними периферійними пристроями, такими, як цифрові потенціометри, ЦАП/АЦП, FLASH-ПЗП і т. ін. За допомогою цього інтерфейсу також може відбуватися обмін між декількома AVR-мікроконтролерами.

В даному розділі в першу чергу розглядається використання інтерфейсу SPI в якості досить високошвидкісного каналу зв'язку.

Крім того, через інтерфейс SPI може бути здійснено програмування мікроконтролера (режим послідовного програмування).

При обміні даними інтерфейсом SPI AVR-мікроконтролер може працювати як ведучий (режим «Master»), або як ведений (режим «Slave»). При цьому користувач може задавати швидкість передачі (сім програмованих значень) і формат передачі (від молодшого розряду до старшого або навпаки).

Додатково при надходженні даних за допомогою інтерфейсу SPI можна виводити мікроконтролер з режиму зниженого енергоспоживання Idle.

8.2. Характеристика модуля SPI мікроконтролерів AVR

8.2.1. Опис структурної схеми модуля

Структурну схему модуля SPI наведено на рис. 8.1. Модуль використовує чотири виводи мікроконтролера, які є лініями портів введення/виведення загального призначення. Як приклад, в табл. 8.1 наведено виводи, які використовуються модулем SPI деяких МК сімейства Mega.

Режим роботи зазначених виводів (напрям передачі даних) при включеному модулі SPI перевизначається згідно з табл. 8.2.

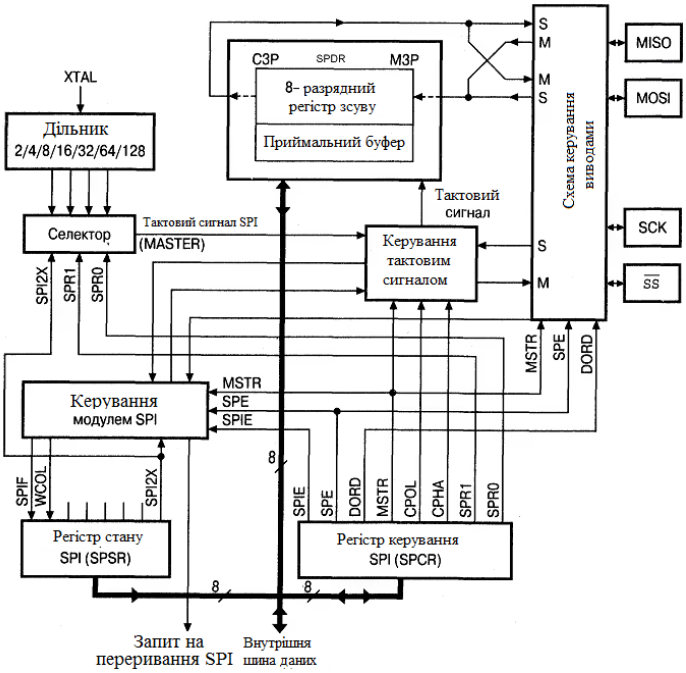


Рис. 8.1. Структурна схема модуля SPI

Таблиця 8.1. **Виводи, що використовуються модулем SPI**

Вивід	**ATmega8x /48x/88x/168x**	**ATmega8515x/8535x**	**ATmega16x/32x**	**ATmega161x/323x**	**ATmega162x/163x**	**ATmega164x/324x/644x**	**Atmega165x/64x/128x**	**ATmega325x/3250x/645x/6450x**	**ATmega640x/1280x/1281x/2560x/2561x**	**Призначення**
SCK	PB5	PB7	PB7	PB7	PB7	PB7	PB1	PB1	PB1	Вихід (master) /вхід (slave) тактовий сигнал
MISO	PB4	PB6	PB6	PB6	PB6	PB6	PB3	PB3	PB3	Вхід (master)/вихід (slave) даних
MOSI	PB3	PB5	PB5	PB5	PB5	PB5	PB2	PB2	PB2	Вихід (master)/вхід (slave) даних
$\overline{SS}$	PB2	PB4	PB4	PB4	PB4	PB4	PB0	PB0	PB0	Вибір веденого пристрою

Таблиця 8.2. **Перепризначення режиму роботи виводів модуля SPI**

Вивід	**Режим «Master»**	**Режим «Slave»**
MOSI	Визначається користувачем як вихід	Вхід
MISO	Вхід	Визначається користувачем як вихід
SCK	Визначається користувачем як вихід	Вхід
$\overline{SS}$	Визначається користувачем як вхід або вихід	Вхід

Напрям обміну даними: передача (вихід) або прийом (вхід) визначається або програмуванням відповідних розрядів регістра DDRB, або задається схемою керування модулем в залежності від режиму роботи «Master»/«Slave». При цьому зберігається можливість керування внутрішніми підтягуючими резисторами виводів, що працюють як входи.

Вивід SCK мікроконтролера в режимі Master працює як вихід, тому відповідна лінія порту введення/виведення загального призначення повинна бути запрограмована на виведення.

8.2.2. Програмування модуля

Для програмування модуля SPI призначено регістр керування SPCR, який в моделях ATmega8515x/8535x/162x/8x/16x/32x/64x/128x розташовано за адресою $0D ($2D), а в інших моделях – за адресою $2C ($4C). Формат цього регістра наведено на рис. 8.2, а опис його розрядів – в табл. 8.3.

	7	6	5	4	3	2	1	0
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
Читання(R)/ Запис ($\overline{W}$)	R/$\overline{W}$	R/$\overline{W}$	R/$\overline{W}$	R/$\overline{W}$	R/$\overline{W}$	R/$\overline{W}$	R/$\overline{W}$	R/$\overline{W}$
Початкове значення	0	0	0	0	0	0	0	0

Рис. 8.2. Формат регістра SPCR

Контроль стану модуля, а також додаткове керування швидкістю обміну здійснюється за допомогою регістра стану SPSR, який розташовано за адресою $0E ($2E) в моделях ATmega8515x/8535x/162x/8x/16x/32x/64x/128x, та за адресою $2D ($4D) – в інших моделях. Розряди з 7-го по 1-й цього регістра доступні тільки для читання, а 0-й розряд – як для читання, так і для запису. Формат цього регістра наведено на рис. 8.3, а опис його розрядів – в табл. 8.4

Таблиця 8.3. **Призначення розрядів регістра SPCR**

Розряд	Назва	Опис
7	SPIE	Дозвіл переривання від SPI, якщо SPIE = 1
6	SPE	Ввімкнення/вимкнення модуля SPI: SPE = 1 – ввімкнений; SPE = 0 – вимкнений
5	DORD	Порядок передачі даних: DORD = 1 – першим передається МЗР, DORD = 0 – першим передається СЗР
4	MSTR	Вибір режиму роботи («Master»/«Slave»): MSTR = 1 – «Master», MSTR = 0 – «Slave»
3	CPOL	Визначає полярність тактового сигналу, див. табл. 8.5
2	CPHA	Фаза тактового сигналу – визначає момент зчитування сигналу, див. табл. 8.5
1, 0	SPR1:SPR0	Швидкість передачі, див. табл. 8.6

	7	6	5	4	3	2	1	0
	SPIF	WCOL	–	–	–	–	–	SPI2X
Читання(R)/ Запис ($\overline{W}$)	R	R	R	R	R	R	R	R/$\overline{W}$
Початкове значення	0	0	0	0	0	0	0	0

Рис. 8.3. Формат регістра SPSR

Дані, що передаються, записуються в регістр даних SPDR, а дані, які приймаються, також зчитуються з цього регістра, який розташовано за адресою $0F ($2F) в моделях ATmega8515x/8535x/162x/8x/16x/32x/64x/128x, а в інших моделях – за адресою $2E ($4E). Запис у цей регістр ініціює початок передачі, а при його читанні зчитується вміст приймаючого буфера регістра зсуву. Тобто регістр даних при читанні служить буфером між регістровим файлом мікроконтролера і регістром зсуву модуля SPI. При записі додаткового буфера нема, а роль регістра даних виконує регістр зсуву.

8.2.3. Опис обміну даними між двома мікронтролерами

З'єднання двох мікроконтролерів (ведучий-ведений) інтерфейсом SPI показано на рис. 8.4. Вивід SCK ведучого мікроконтролера є виходом тактового сигналу, а веденого мікроконтролера – входом.

Таблиця 8.4. **Опис розрядів регістра SPSR**

Розряд	Назва	Опис
7	SPIF	**Прапорець «Кінець передачі»** Даний прапорець встановлюється в одиницю по закінченні передачі/прийому чергового байта. Якщо прапорець SPIE регістра SPCR також встановлено в одиницю, то генерується переривання від модуля SPI. Прапорець SPIF також встановлюється в одиницю при переведенні мікроконтролера з режиму «Master» у режим «Slave» за допомогою виводу $\overline{SS}$, див. 8.2.5. Прапорець скидається апаратно, або при старті підпрограми обробки переривання, або після читання регістра стану SPI з наступним зверненням до регістра даних SPDR
6	WCOL	**Прапорець конфлікту запису** Даний прапорець встановлюється в одиницю при спробі запису у регістр даних SPDR під час передачі чергового байта. Прапорець скидається апаратно після читання регістра стану SPI з наступним зверненням до регістра даних SPDR
5...1	-	Зарезервовано, читаються як «0»
0	SPI2X	**Подвоєння швидкості обміну** При встановленні цього розряду в одиницю та роботі мікроконтролера в режимі «Master» частота сигналу SCK подвоюється

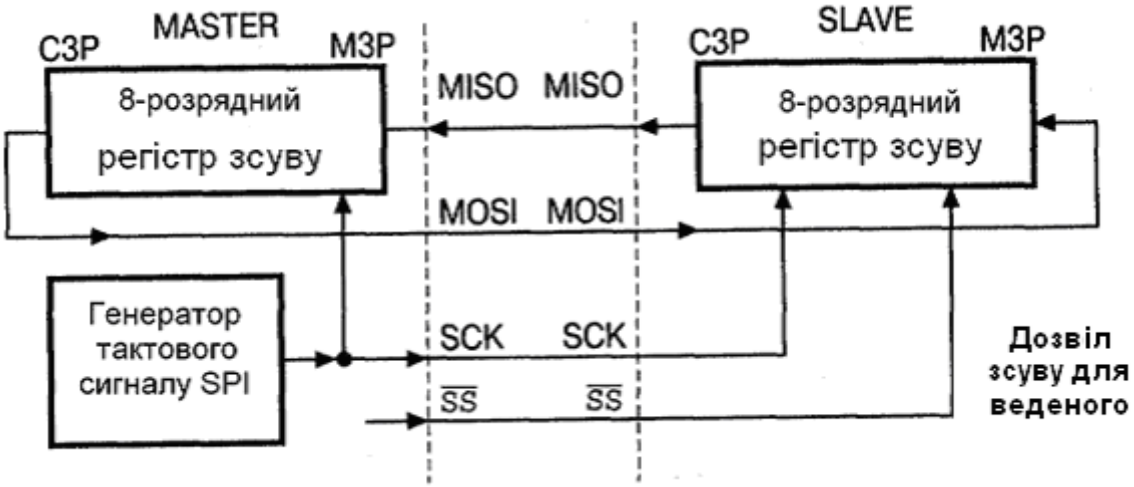


Рис. 8.4. З’єднання мікроконтролерів інтерфейсом SPI

Перед виконанням обміну необхідно дозволити роботу модуля SPI. Для цього потрібно встановити в одиницю розряд SPE регістра SPCR. Режим роботи визначається станом розряду MSTR цього регістра: якщо розряд встановлено в одиницю, мікроконтролер працює в режимі «Master», якщо скинуто внуль – у режимі «Slave».

Передача даних здійснюється наступним чином. У разі запису в регістр даних SPI ведучого мікроконтролера запускається генератор тактового сигналу модуля SPI. Після цього дані починають порозрядно видаватися на вивід MOSI і, відповідно, надходити на вивід MOSI веденого мікроконтролера. Порядок передачі розрядів даних визначається станом розряду DORD регістра SPCR. Якщо розряд встановлено в одиницю, першим передається молодший розряд байта, якщо скинуто в нуль – старший розряд. Після видачі останнього розряду поточного байта генератор тактового сигналу зупиняється з одночасним встановленням в одиницю прапорця «Кінець передачі» SPIF). Якщо переривання від модуля SPI дозволено (прапорець SPIE регістра SPCR встановлено в одиницю), генерується запит на переривання.

Після цього в режимах 2, 3 ведучий мікроконтролер записом наступного байта в регістр SPDR може почати його передачу.

В режимах 0, 1 подавши на вхід $\overline{SS}$ веденого мікроконтролера напругу високого рівня, треба перевести останній у стан очікування. Після цього, подавши на вхід $\overline{SS}$ веденого напругу низького рівня, почати передачу наступного байта.

Одночасно з передачею даних від ведучого до веденого відбувається передача у зворотному напрямку за умови, що на вході $\overline{SS}$ веденого присутня напруга низького рівня. Таким чином, у кожному циклі зсуву відбувається обмін даними між двома пристроями. Наприкінці кожного циклу прапорець SPIF встановлюється в одиницю як у ведучому мікроконтролері, так і у веденому. Прийняті байти зберігаються в приймальних буферах для подальшого використання.

У модулі реалізовано одинарну буферизацію при передачі і подвійну при прийомі. Це означає, що готовий для передачі байт даних не може бути записано у регістр даних SPI до закінчення попереднього циклу обміну. При спробі змінити вміст регістра даних під час передачі встановлюється в одиницю прапорець WCOL регістра SPSR. Цей прапорець скидається після читання регістра SPSR з наступним зверненням до регістра даних SPDR.

При прийомі даних прийнятий байт треба прочитати з регістра даних до того, як у регістр зсуву надійде останній розряд наступного байта. В іншому випадку попередній байт буде загублено.

8.2.4. Структура SPI-мережі мікроконтролерів

Структуру SPI-мережі мікроконтролерів наведено на рис. 8.5.

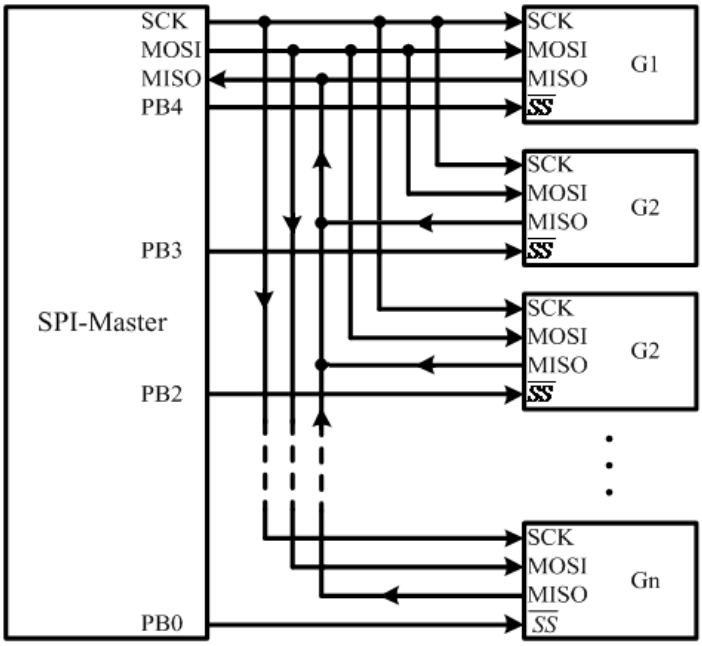


Рис. 8.5. Структура SPI-мережі мікроконтролерів

До інтерфейсу ведучого SPI можна одночасно підключити декілька ведених периферійних пристроїв, але активним буде тільки той, у якого на вхід $\overline{SS}$ через відповідну лінію порту ведучого пристрою буде подано рівень логічного нуля. Виводи MISO не задіяних ведених блоків знаходяться у високоомному стані і не впливають на процес передачі даних.

На рис. 8.5 блокам G1, G2 … Gn відповідають ведені інтерфейси SPI, в яких дані передаються в обох напрямах. Один з ведених блоків з точки зору ведучого блоку, може бути, наприклад, тільки блоком видачі. В якості такого блоку можна привести, наприклад, аналого-цифровий перетворювач з інтерфейсом SPI. Іншим веденим може бути, наприклад, цифро-аналоговий перетворювач, який буде використовуватись як блок прийому.

8.2.5. Режими передачі даних інтерфейсом SPI

Існують чотири режими передачі даних інтерфейсом SPI: 0…3 (рис. 8.6; 8.7).

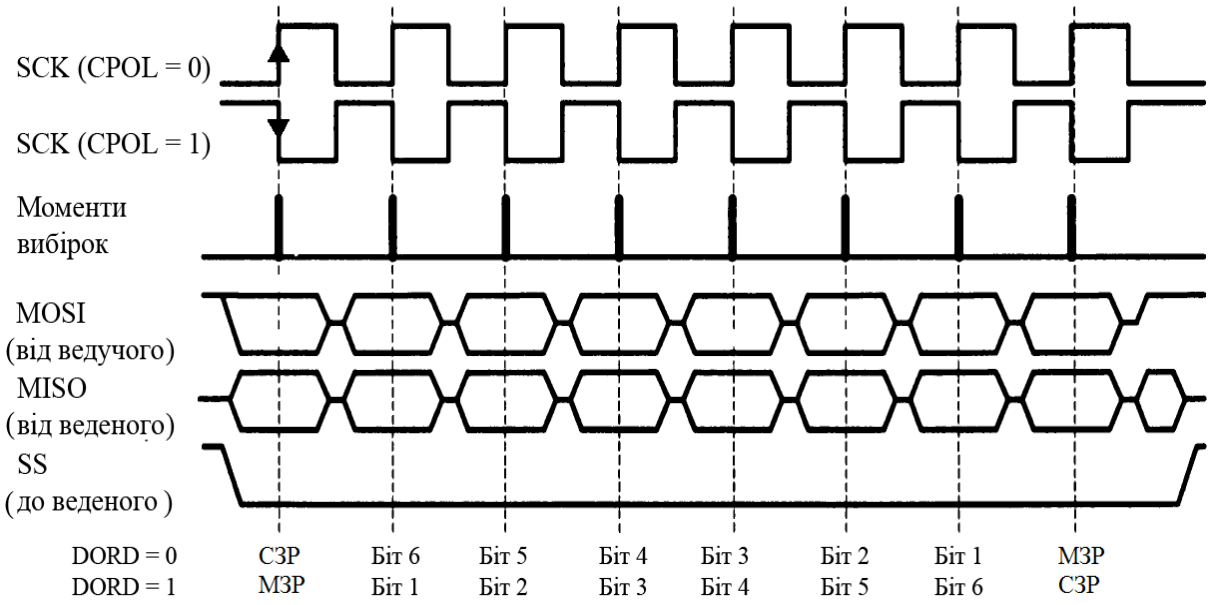


Рис. 8.6. Передача даних при CPHA = «0» (режими 0, 1)

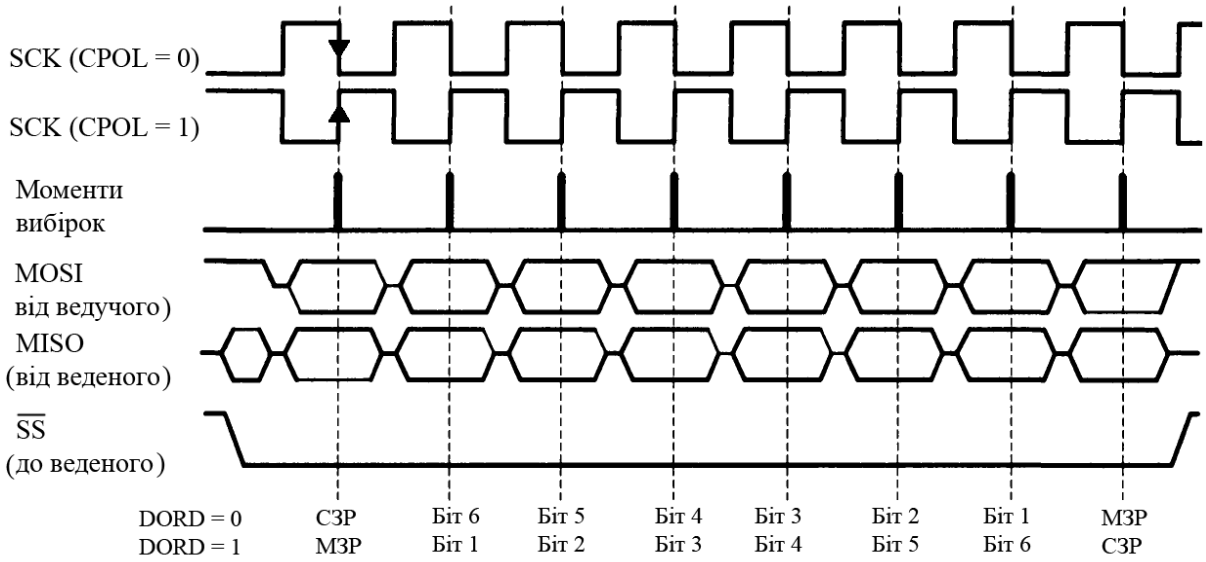


Рис. 8.7. Передача даних при CPHA = «1» (режими 2, 3)

Ці режими розрізняються відповідністю між фазою тактового сигналу SCK (визначає момент зчитування сигналу), його полярністю та даними, які передаються. Існують чотири комбінації розрядів CPHA і CPOL регістра SPCR (табл. 8.5), які визначають відповідний режим роботи модуля.

На рис. 8.6, 8.7 передача даних ведеться від старшого розряду до молодшого.

Таблиця 8.5. **Задання режиму передачі даних**

Розряд	Опис
CPOL	**Полярність тактового сигналу** «0» – генеруються імпульси додатної полярності, при відсутності імпульсів на виводі присутній низький рівень; «1» – генеруються імпульси від'ємної полярності, при відсутності імпульсів на виводі присутній високий рівень (рисунки 8.6, 8.7)
CPHA	**Фаза тактового сигналу** «0» – обробка даних виконується за переднім фронтом імпульсів сигналу SCK (для CPOL = «0» – за наростаючим фронтом, а для CPOL = «1» – за спадаючим фронтом) (рис. 8.6); «1» – обробка даних виконується за заднім фронтом імпульсів сигналу SCK (для CPOL = «0» – за спадаючим фронтом, а для CPOL = «1» – за наростаючим фронтом) (рис. 8.7)

В режимах 0, 1 (CPHA = 0) початок обміну визначається встановленням сигналу на виводі $\overline{SS}$ веденого в активний стан – логічний нуль. При цьому ведений виставляє на лінію MISO старший розряд даних, якщо першим передається СЗР, або молодший – якщо першим передається МЗР (рис. 8.6). Ведучий видає на лінію MOSI СЗР (МЗР) на $0{,}5*T_{SCK}$ раніше, ніж на лінії SCK з'являється перший імпульс T_{SCK} – період синхроімпульсів.

Перед входами восьмирозрядних регістрів зсуву ведучого та веденого мікроконтролерів (рис. 8.4, див. вище) знаходяться два синхронних тригери (буфери) (на рисунку не показано), в які першим перепадом сигналу на лінії SCK записується значення сигналу, яке присутнє на їх інформаційному вході. Цей момент на рис. 8.6 відмічено на лінії «Моменти вибірок».

Другим перепадом імпульсів на лінії SCK відбувається зсув інформації вліво відповідно до рис. 8.4. При цьому стан буферів переписується в молодші розряди регістрів зсуву, а черговий вихідний біт з регістрів зсуву виставляється на лінії MOSI/MISO та з затримкою надходить на вхід буферів.

Вказаний зсув у часі між моментом видачі чергового розряду в лінію зв'язку між мікроконтролерами і моментом фіксації цього біта в буфері дозволяє компенсувати часові затримки при передачі сигналів між мікроконтролерами.

Далі аналогічно здійснюється обмін між ведучим та веденим всіма наступними бітами. З кожним непарним перепадом сигналу SCK відбувається фіксація чергового біта в буфері, а з кожним парним – зсув інформації вліво.

В режимах 0, 1 сигнал на лінії вибору веденого треба повернути у не активний стан (логічна одиниця) після передачі кожного байта. Тоді передача кожного нового

байта буде супроводжуватися попереднім встановленням $\overline{SS}$ в нуль (рис. 8.8*а*), що буде визначати початок обміну наступним байтом.

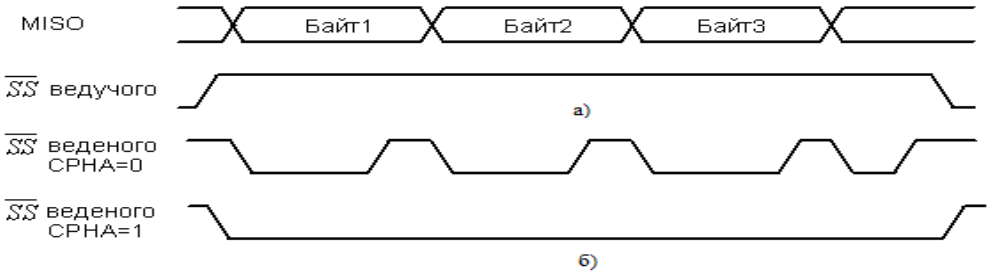


Рис. 8.8. Вплив сигналу $\overline{SS}$ на початок обміну даними: а – режими 0, 1; б – режими 2, 3

В режимах 2, 3 (CPHA = 1) початок обміну визначає перша зміна сигналу на лінії SCK після встановлення сигналу вибору веденого $\overline{SS}$ в активний стан логічний нуль (рис. 8.7*б*, див. вище).

Моменти фіксації біта в буферах перед входами регістрів зсуву передавача та приймача на рис. 8.7 відмічено на лінії «Моменти вибірок». При передачі даних від ведучого до веденого та у зворотному напряму всі непарні перепади SCK викликають видачу чергового біта посилки з регістра зсуву передавача на лінію. Кожний парний перепад використовується для запису цього біта в буфер перед регістром зсуву. Тому сигнал вибору веденого може залишатися в активному стані – логічний нуль протягом передачі декількох байт інформації (рис. 8.8*б*, див. вище). Це трохи спрощує логіку програмування модуля SPI.

Таким чином, при передачі біти даних висуваються з виходів регістрів зсуву від`ємними або додатними фронтами сигналу SCK, а фіксуються в буферах на входах регістрів додатними або від`ємними фронтами синхросигналу SCK, які зсунуто відносно моментів зміни бітів на виході регістрів зсуву на половину періоду (рис. 8.6; 8.7, див. вище). Це гарантує достатній час на встановлення даних на входах відповідних регістрів після висування.

8.2.6. Програмування швидкості передачі даних

Частота тактового сигналу SCK і, відповідно, швидкість передачі даних інтерфейсом визначаються станом розрядів SPR1:SPR0 регістра SPCR і розряду SPI2X регістра SPSR мікроконтролера, який працює в режимі «Master», тому що саме він є джерелом тактового сигналу (табл. 8.6).

Таблиця 8.6. **Задання частоти тактового сигналу SCK**

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

Примітка: f_{CLK} – тактова частота мікроконтролера.

Для пристрою, який перебуває в режимі «Slave», стан цих розрядів байдужий. Нормальне функціонування мікроконтролера в режимі «Slave» гарантується тільки на частотах, менших або рівних $f_{CLK}/4$ [1; 7].

8.2.7. Використання виводу $\overline{SS}$

Цей вивід призначено для вибору активного веденого пристрою та у режимі «Slave» завжди є входом (рис. 8.5, див. вище). При подачі на нього напруги низького рівня модуль SPI активується і вивід MISO перемикається в режим виведення даних (якщо це задано користувачем, табл. 8.2, див.вище). Інші виводи модуля SPI в цьому режимі є входами. При подачі на вивід $\overline{SS}$ напруги високого рівня всі виводи модуля SPI перемикається в режим введення даних. При цьому модуль переходить у не активний стан і прийом даних не відбувається. Як правило, у цьому стані програма змінює вміст регістра даних.

Коли на вивід $\overline{SS}$ веденого подається напруга високого рівня, відбувається скидання модуля SPI. Відповідно, якщо зміна стану цього виводу відбудеться під час обміну даними, прийом і передача негайно припиняться, а переданий і прийнятий байти будуть загублено.

Якщо мікроконтролер перебуває в режимі «Master» (розряд MSTR регістра SPCR встановлено в одиницю), напрям передачі даних через вивід $\overline{SS}$ визначається користувачем. Якщо вивід сконфігуровано як вихід, він використовується для керування виводом $\overline{SS}$ мікроконтролера, що працює в режимі «Slave».

Якщо в режимі «Master» вивід сконфігуровано як вхід, то для забезпечення нормальної роботи модуля SPI на нього треба подати напругу високого рівня.

Подача на цей вхід напруги низького рівня від якої-небудь зовнішньої схеми буде сприйнято модулем SPI як вибір даного мікроконтролера в якості веденого, і відповідно, початок передачі йому даних.

Щоб уникнути конфлікту на шині модуль SPI у таких випадках виконує наступні дії:

1) Прапорець MSTR регістра SPCR скидається, і мікроконтролер перемикається в режим «Slave». Як наслідок, виводи MOSI і SCK починають функціонувати як входи (табл. 8.2).

2) Встановлюється прапорець SPIF регістра SPSR, генеруючи запит на переривання від SPI. Якщо переривання від SPI дозволено і прапорець I регістра SREG встановлено в одиницю, відбувається запуск підпрограми обробки переривання.

Таким чином, якщо ведучий мікроконтролер використовує передачу даних, керовану перериванням, та існує ймовірність подачі на вхід $\overline{SS}$ напруги низького рівня, у підпрограмі обробки переривання від SPI обов'язково повинна здійснюватися перевірка стану прапорця MSTR. При виявленні скидання цього прапорця його треба знову програмно встановити в одиницю для зворотного переведення мікроконтролера в режим «Master».

8.2.8. Режим програмування пам’яті послідовним каналом

В мікропроцесорних системах з використанням мікроконтролерів сімейств Mega, XMega, а також деяких мікроконтролерів сімейства Tiny для програмування пам’яті може використовуватись послідовний інтерфейс SPI. Як правило, цей режим використовується для програмування (перепрограмування) мікроконтролера безпосередньо у пристрої (рис. 8.9) [1; 7].

Рис. 8.9. Включення мікроконтролерів у режимі програмування послідовним каналом

Як видно з рис. 8.9, для підключення програматора до пристрою використовуються три лінії інтерфейсу: SCK (тактовий сигнал), MOSI (вхід

даних) і MISO (вихід даних). Відповідність між лініями інтерфейсу і контактами портів виведення/виведення деяких AVR-мікроконтролерів наведено в табл. 8.7.

Таблиця 8.7. **Виводи, що використовуються при програмуванні послідовним каналом**

Назва лінії інтерфейсу	**ATmega8515x/8535x**	**ATmega8x**	**ATmega16x/32x**	**ATmega64x/128x**	**ATmega48x/88x/168x**	**ATmega162x**	**ATmega164x/324x/644x**	**ATmega165x ATmega325x/3250x ATmega645x/6450x**	**ATmega640x/1280x/2560x**	**ATmega1281xATmega2561x**	**Призначення виводів**
SCK	PB7	PB5	PB7	PB1	PB5	PB7	PB7	PB1	PB1	PB1	Вхід тактового сигналу
MISO (PDO)	PB6	PB4	PB6	PE1	PB4	PB6	PB6	PB3	PE1	PB3	Вихід даних
MOSI (PDI)	PB5	PB3	PB5	PE0	PB3	PB5	PB5	PB2	PE0	PB2	Вхід даних

В окремих мікроконтролерах, наприклад, ATmega64x та ATmega128x виводи, які використовуються для програмування (табл. 8.7), не збігаються з виводами, призначеними для штатної роботи інтерфейсу SPI.

Часові діаграми сигналів при програмуванні мікроконтролерів у розглянутому режимі представлено на рис. 8.10, а значення параметрів сигналів наведено в [1; 7].

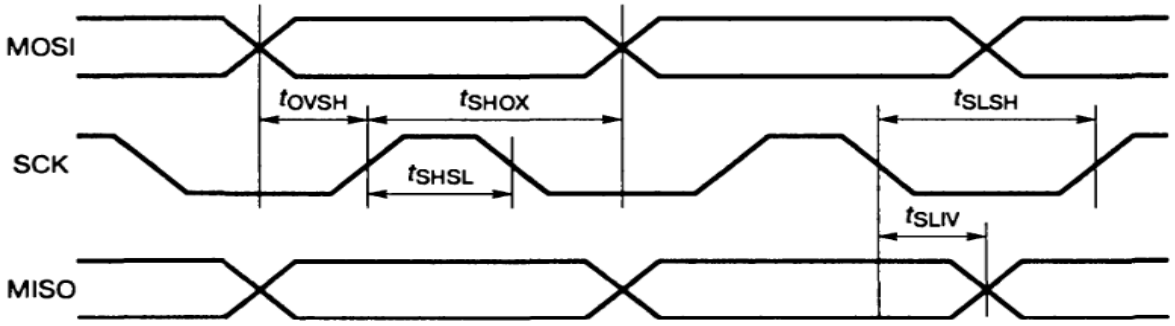


Рис. 8.10. Часові діаграми сигналів при програмуванні послідовним каналом

Як і в робочому режимі, при програмуванні послідовним каналом мікроконтролеру потрібно джерело тактового сигналу. В якості такого може використовуватися кожне із можливих для мікроконтролера джерел. При цьому

повинна виконуватися наступна умова: тривалість імпульсів як низького рівня, так і високого рівня сигналу SCK повинна бути більше 2-х (при f_{CLK} < 12 МГц) або 3-х (при $f_{CLK} \geq 12$ МГц) періодів тактового сигналу мікроконтролера.

Програмування здійснюється шляхом посилки 4-байтних команд на вивід MOSI мікроконтролера. Результат виконання команд читання знімається з виводу MISO мікроконтролера. Передача команд і видача результатів їхнього виконання здійснюється від старшого розряду до молодшого. При цьому запис вхідних даних виконується за наростаючим фронтом сигналу SCK, а зсув вихідних даних – за спадаючим (рис. 8.11).

Більш детально це питання розглянуто у [1; 7].

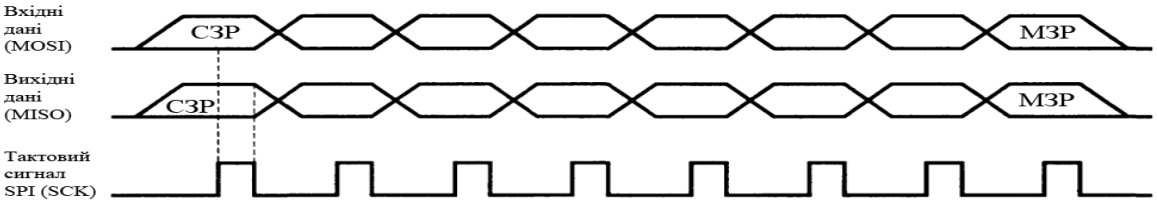


Рис. 8.11. Передача даних при програмуванні послідовним каналом

8.2.9. Периферійні пристрої з SPI-інтерфейсом

Для спрощення вибору елементної бази при проектуванні електронних схем у [1] наведено деякі мікросхеми: АЦП; ЦАП; давачі і т. ін, що містять інтерфейс SPI.

8.2.10. Універсальний послідовний інтерфейс USI

Деякі моделі, наприклад ATmega165x, ATmega325x/3250/645x/6450x мають модуль універсального послідовного інтерфейсу (Universal Serial Interface – USI) [1; 7]. Цей модуль є свого роду “напівфабрикатом”, який надає базові апаратні ресурси, що необхідні для здійснення обміну послідовним каналом. Використовуючи даний модуль, можна досягти суттєво більшої швидкості передачі та отримати більш компактний код, ніж при чисто програмній реалізації різноманітних протоколів обміну.

8.3. Моделювання модуля SPI

8.3.1. Опис моделі

На рис. 8.12 наведено схему моделі дослідження послідовного периферійного інтерфейсу SPI.

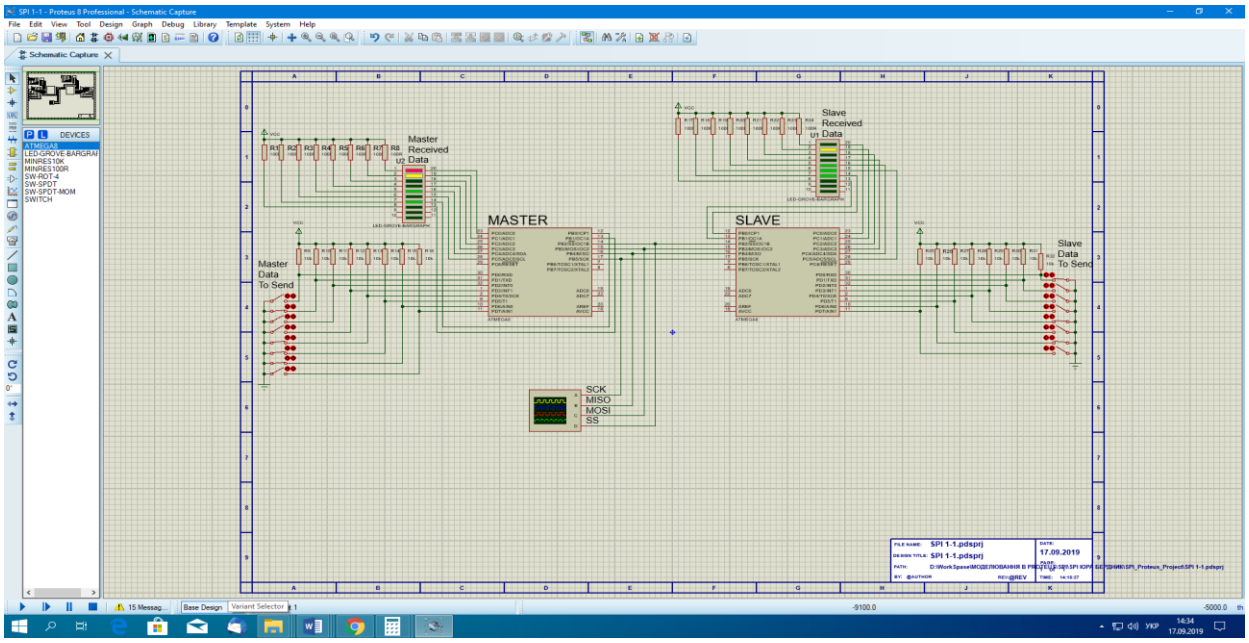


Рис. 8.12. Схема робочої моделі

Ліворуч на схемі розташовано мікроконтролер, що працює в режимі «Master», а праворуч – мікроконтролер, який працює в режимі «Slave». В якості веденого та ведучого мікроконтролерів використано мікросхеми ATmega8.

Блок кнопок «Master Data To Send» призначено для задання байта даних для передачі від ведучого до веденого. Всі вісім кнопок цього блоку підключено до ліній порту D. Нижнє положення кнопки відповідає подачі на відповідну лінію порту D мікроконтролера напруги низького рівня, тобто логічного нуля, а верхнє положення кнопки – подачі на відповідну лінію порту напруги високого рівня, тобто логічної одиниці. Резистори R9…R16, що підтягують, призначено для формування напруги високого рівня на відповідній лінії порту мікроконтролера, коли контакти відповідної кнопки розімкнено.

Світлодіодну лінійку U2 «Master Received Data» призначено для відображення байта даних, прийнятого ведучим мікроконтролером від веденого. Верхній за схемою сегмент лінійки відображає молодший біт прийнятого байта даних, а нижній – старший. Світіння сегмента відповідає тому, що було прийнято логічний нуль, тобто натиснуто відповідну кнопку з боку передавача. Катоди сегментів світлодіодної лінійки, що відображають 6 молодших біт прийнятого байта підключено до ліній PC0…PC5 порту C мікроконтролера, а катоди сегментів, що відображають 2 старші біти – до ліній PB0, PB1 порту B. Аноди всіх сегментів світлодіодної лінійки через обмежуючі резистори R1…R8 підключено до лінії живлення +5В.

До ліній SCK, $\overline{SS}$, MOSI та MISO ведучого мікроконтролера підключено відповідні лінії веденого мікроконтролера та входи відповідних каналів віртуального осцилографа.

Підключення веденого мікроконтролера виконано аналогічно. Кнопки «Slave Data To Send» призначено для введення байта даних для передачі від веденого до ведучого. Світлодіодна лінійка U1 «Slave Received Data» відображає байт даних, прийнятий веденим від ведучого. Резистори R17…R24 обмежують струм через світлодіоди лінійки U1, а резистори R25…R31 призначено для формування напруги високого рівня на відповідних лініях порту D веденого мікроконтролера, коли контакти відповідної кнопки розімкнено.

8.3.2. Порядок моделювання

1. Для запуску моделювання натискаємо кнопку «Run simulation» «». Після чого кнопками «Master Data To Send» задаємо байт для передачі ведучим МК веденому та одночасно спостерігаємо за відображенням його прийому на лінійці світлодіодів U1 «Slave Received Data». Також спостерігаємо за зміною сигналів, що відображаються у вікні віртуального осцилографа (рис. 8.13).

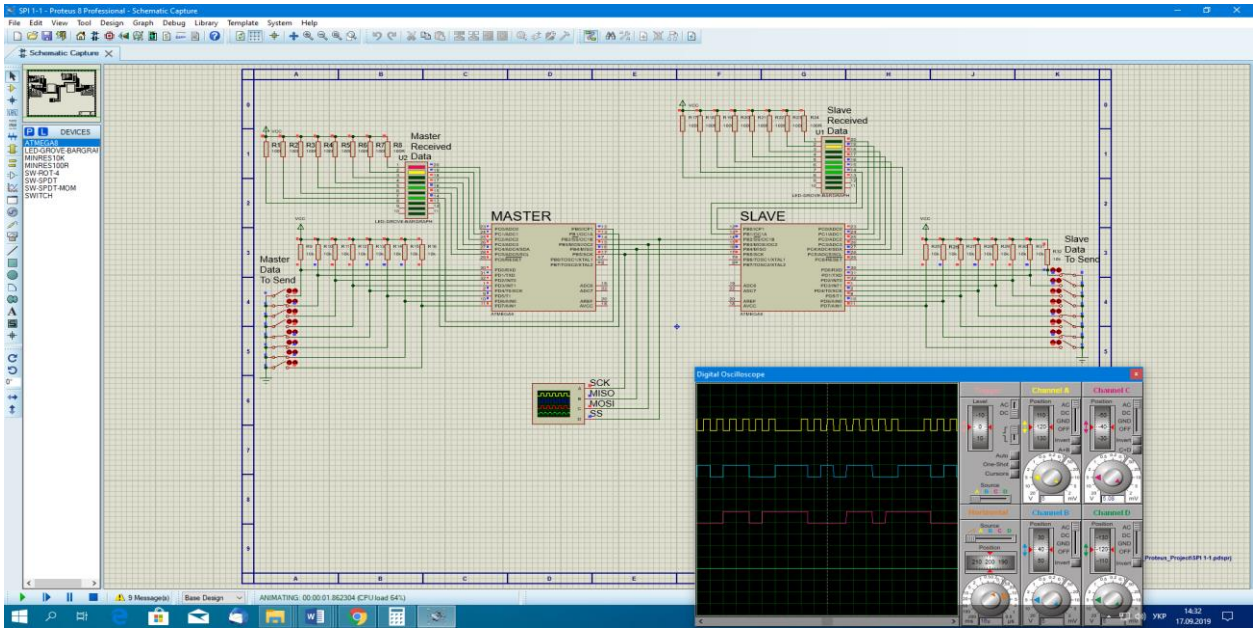


Рис. 8.13. Запуск моделювання

2. Кнопками «Slave Data To Send» задаємо байт для передачі веденим МК ведучому та одночасно спостерігаємо за відображенням його прийому на лінійці світлодіодів U2 «Master Received Data» та за зміною сигналів, що відображаються у вікні віртуального осцилографа.

Кнопками «Master Data To Send» задаємо байт «10001101» для передачі від ведучого мікроконтролера («Master») до веденого («Slave»). Після цього спостерігаємо прийом даного байта веденим мікроконтролером та відображення його прийому на лінійці світлодіодів U1 «Slave Received Data». Відповідні фрагменти схеми моделі наведено на рис. 8.14. Заданий байт «10001101» від ведучого веденому починаючи із СЗР послідовно передається по лінії MOSI, що видно на відповідному каналі (червоний колір) віртуального осцилографа (рис. 8.15).

Кнопками «Slave Data To Send» задамо байт «10101110» для передачі від веденого мікроконтролера до ведучого. Після чого спостерігаємо прийом даного байта ведучим мікроконтролером та відображення його прийому на лінійці світлодіодів U2 «Master Received Data». Відповідні фрагменти схеми моделі наведено на рис. 8.16. Заданий байт «10101110» від веденого ведучому передається послідовно лінією MISO, що видно на відповідному каналі (синій колір) віртуального осцилографа.

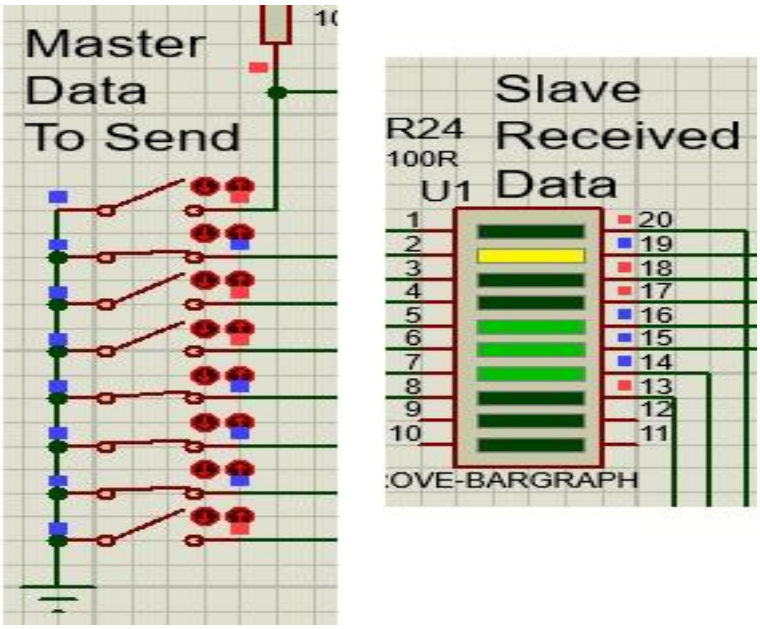


Рис. 8.14. Фрагменти схеми моделі для відображення передачі від ведучого до веденого

3. Визначаємо отриману швидкість передачі. Для цього підберемо таку розгортку сигналу на осцилографі, щоб сумістити із вертикальними лініями сітки осцилографа лінії, які відповідають періоду імпульсів сигналу SCK.

В нашому випадку період імпульсів $t_{ім.SCK} = 16МКС$. Тоді тактова частота SPI-інтерфейса

$$f_{SCK}=1*1000000/16=62500Гц.$$

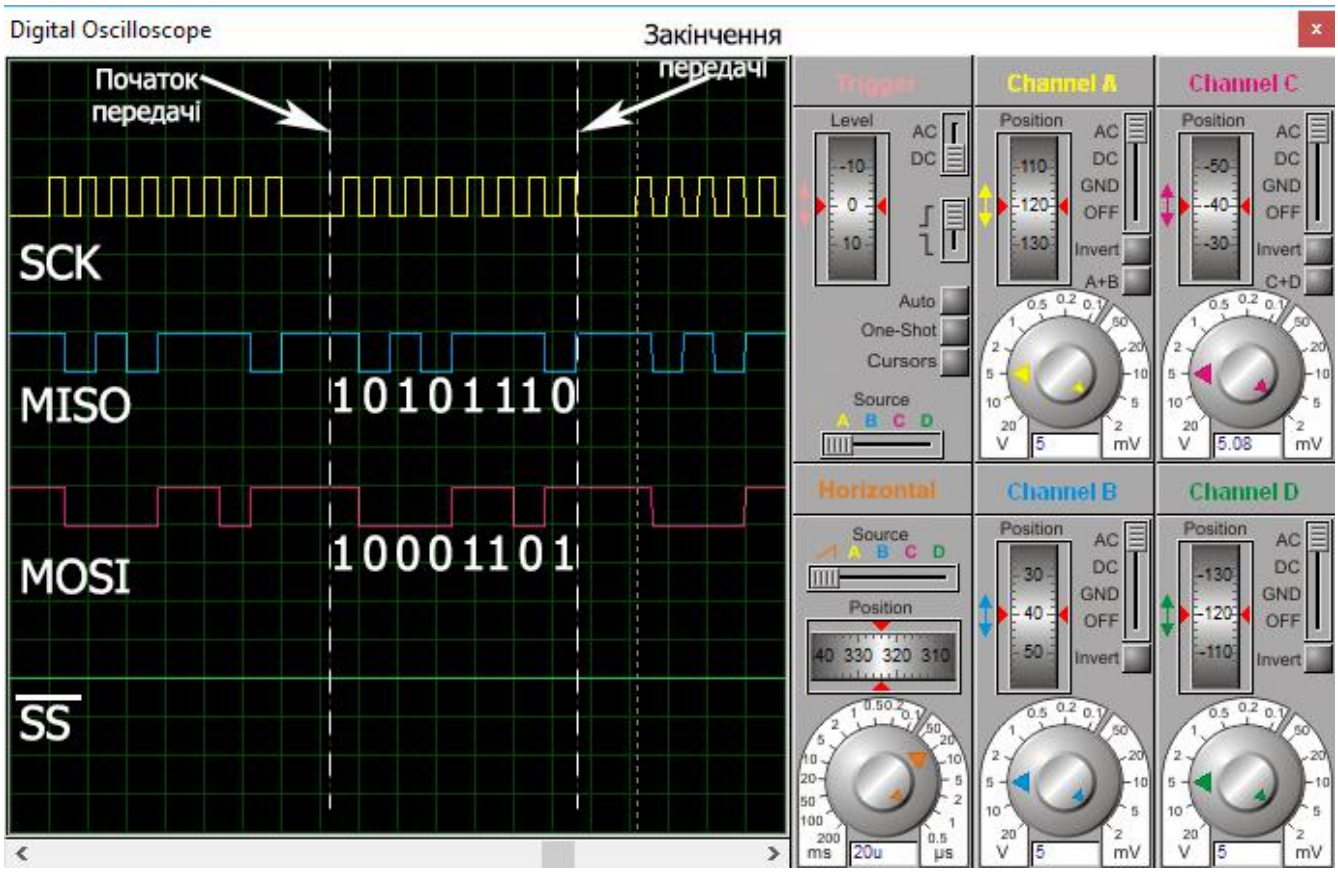


Рис. 8.15. Вікно віртуального осцилографа

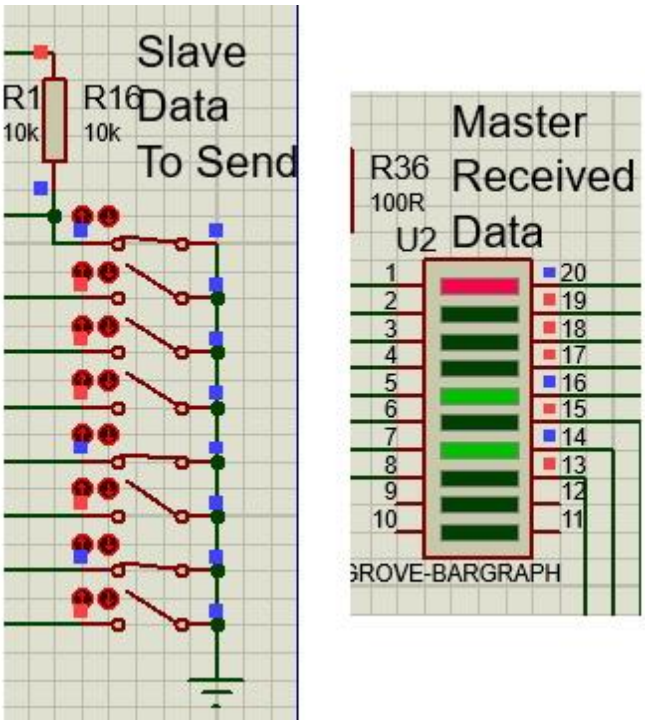


Рис. 8.16 Фрагменти схеми моделі для відображення передачі від веденого до ведучого

4. Частота тактового сигналу SCK і, відповідно, швидкість передачі даних інтерфейсом визначаються станом розрядів SPR1:SPR0 регістра SPCR і розряду

SPI2X регістра SPSR (табл. 8.6, див. вище) мікроконтролера, який працює в режимі «Master», тому що саме він є джерелом тактового сигналу.

Відповідно до табл. 8.6, тактова частота ведучого мікроконтролера при SPI2X = SPR1 = 0, SPR0 = 1

$$F_{clk} = f_{SCK} * 16 = 62500*16=1000000Гц = 1\ МГц,$$

що відповідає його початковому налаштуванні (див. лістинг керуючої програми).

8.3.3. Схема алгоритму роботи

На рис. 8.17 та 8.18 наведено відповідно схеми алгоритмів роботи ведучого та веденого мікроконтролерів.

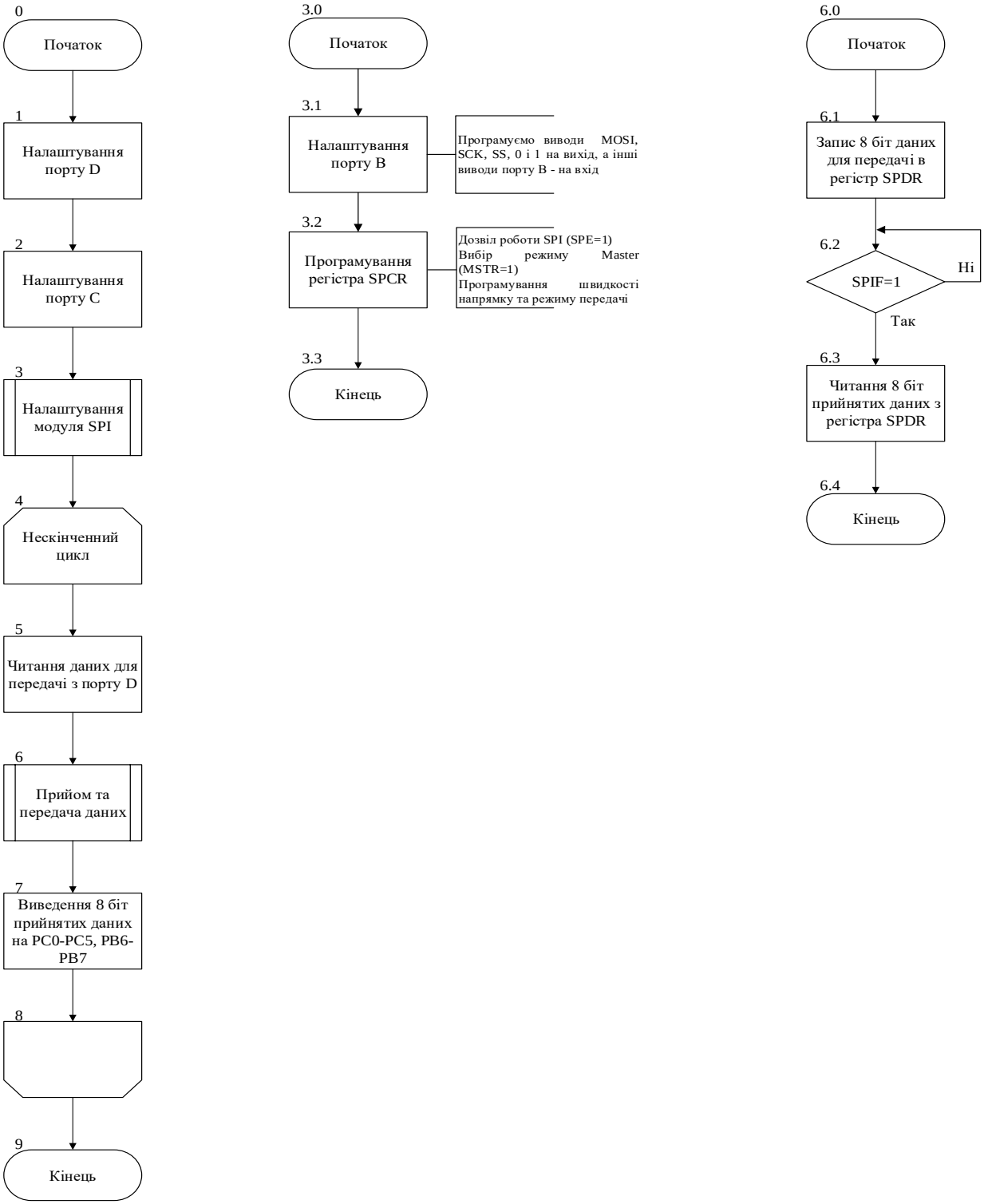


Рис. 8.17. Схема алгоритму роботи ведучого МК

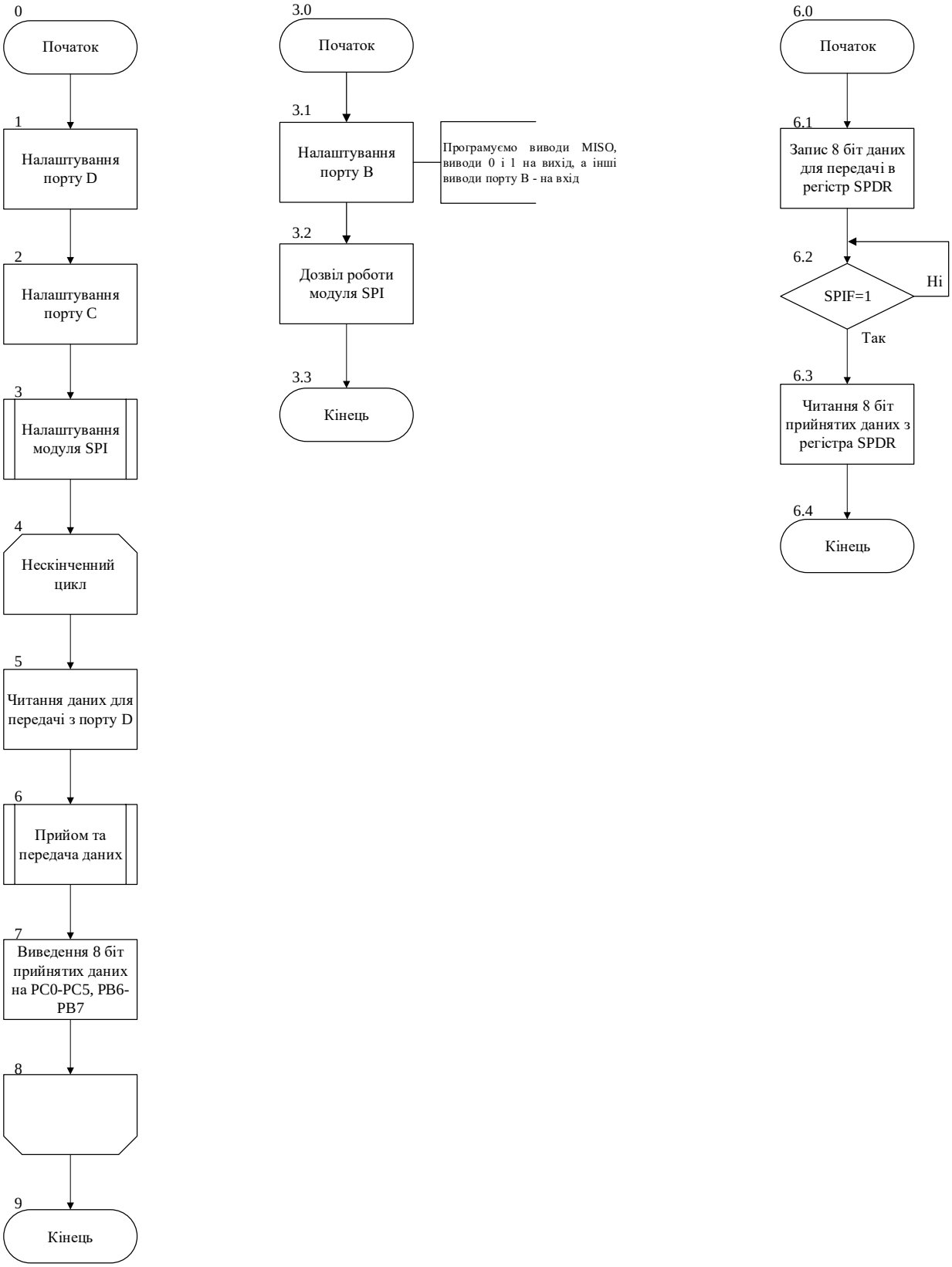


Рис. 8.18. Схема алгоритму роботи веденого МК

8.3.4. Робоча програма

Відповідно до схем алгоритмів, наведених на рисунках 8.17 та 8.18, розроблено керуючі програми для ведучого та веденого мікроконтролерів. Лістинги розроблених програм наведено нижче.

SPI Master:

```
#define F_CPU 1000000
#include <avr/io.h>
#include <util/delay.h>

//3.0 налаштування модуля SPI
void SPI_MasterInit(void)
{
// 3.1 програмуання виводів MOSI та SCK, SS, виводи 0 і 1 на вихід, а інші на
вхід
DDRB = 0b00101111;
// 3.2 дозвіл роботи SPI, режим Master, встановлення швидкості передачі (fck/16)
SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0);
} // 3.3

//6 прийом та передача даних
char SPI_MasterTransmit_Receive(char cData)
{ //6.0
	char received_Data;
	/* Початок передачі */
// 6.1 запис 8 біт даних для передачі в SPDR
	SPDR = cData;
// 6.2 чекання завершення передачі
	while(!(SPSR & (1<<SPIF)))
	;
// 6.3 читання прийнятих даних з регістра SPDR
	received_Data=SPDR;
//6.4 вихід з підпрограми
	return received_Data;
}
```

```
// 0 основна програма
int main(void)
{
char data; // 1 Об`явлення змінної data, яка використовується при читанні порту D
// 1 Програмування порту D на вхід
        DDRD=0x00;
// 1 Відключення внутрішніх підтягуючіх резисторів, оскільки вони є на схемі.

        PORTD=0x00;
// 2 Програмування виводів порту C на вихід
        DDRC=0xff;
// 2 Встановлення виводів порту C в одиницю
        PORTC=0xFF;
// 3 Налаштування модуля SPI
        SPI_MasterInit();
        while (1)
        {                                   // 4 початок нескінченного цикла

// 5 Читання даних для передачі з порту D
        data = PIND;
// 6 Прийом/передача даних
        char r_Data = SPI_MasterTransmit_Receive(data);
// 7 Виведення 6 молодших біт прийнятих даних на   PC0…PC5
        PORTC = r_Data;
// Виведення 2 старших біт прийнятих даних на PB0-PB1
                char temp = PORTB;
                temp &= 0b11111100;
                temp |= ((r_Data>>6) & 0b00000011);
                PORTB = temp;
        }                                   // 8 кінець нескінченного цикла
}                                           // 9 кінець основної програми SPI Master
```

SPI Slave:

```
#define F_CPU 1000000
#include <avr/io.h>
#include <util/delay.h>
// 3.0 підпрограма налаштування модуля SPI
```

```
void SPI_SlaveInit(void)
{
// 3.1 Програмування виводу MISO, виводи 0 і 1 на вихід, а інші – на вхід
        DDRB = 0b00010011;
// 3.2 Дозвіл роботи модуля SPI
        SPCR = (1<<SPE);
}       //3.3 кінець підпрограми SPI_SlaveInit

//6.0 підпрограма прийому та передачі даних
char SPI_SlaveTransmit_Receive(char cData)
{
        char received_Data;
// Початок передачі
// 6.1 Запис 8 біт даних для передачі в SPDR
        SPDR = cData;
// 6.2 Чекання завершення передачі
        while(!(SPSR & (1<<SPIF)))
        ;
// 6.3 Читання прийнятих даних з регістра SPDR
        received_Data=SPDR;
// 6.4 Кінець підпрограми SPI_SlaveTransmit_Receive
        return received_Data;
}

//  0 основна програма
        int main(void)
        {
// 1 Програмування порту D на вхід
        DDRD=0x00;
// 1 Відключення внутрішніх підтягуючих резисторів, оскільки вони є на схемі
                PORTD=0x00;
// 2 Програмування виводів порту C на вихід
        DDRC=0xff;
// 2 Встановлення виводів порту C в одиницю
        PORTC=0xFF;
// 3 Налаштування модуля SPI, виклик підпрограми SPI_SlaveInit
        SPI_SlaveInit();
        char data;
```

```
        while (1)                                   // 4 початок нескінченного цикла
        {
// 5 Читання даних для передачі з порту D
        data = PIND;
// 6 Прийом та передача даних, виклик підпрограмми  SPI_SlaveTransmit_Receive
        char r_Data = SPI_SlaveTransmit_Receive (data);
// 7 Виведення 6 молодших біт прийнятих даних на  PC0…PC5
        PORTC = r_Data;
// Виведення 2 старших біт прийнятих даних на PB0…PB1:
        char temp = PORTB;                          // 7
        temp &= 0b11111100;                         // 7
        temp |= ((r_Data>>6) & 0b00000011); // 7
        PORTB = temp;                               // 7
                }                                           // 8 кінець нескінченного цикла
        }                                   // 9 кінець основної програми SPI Slave
```

Контрольні запитання та завдання

1. Який режим обміну використовується в модулі SPI: асинхронний чи синхронний? Відповідь пояснити.
2. На яку відстань і з якою швидкістю можна здійснювати обмін даними через інтерфейс SPI?
3. Поясніть структурну схему модуля SPI.
4. В яких режимах може працювати мікроконтролер при обміні даними інтерфейсом SPI?
5. Скільки та які виводи мікроконтролера використовує модуль SPI?
6. Як виконується перепризначення режиму роботи виводів модуля SPI?
7. Поясніть схему з'єднання двох мікроконтролерів інтерфейсом SPI.
8. Які пристрої знаходяться перед входами восьмирозрядних регістрів зсуву ведучого та веденого мікроконтролерів? Яку функцію вони виконують?
9. Який регістр призначено для керування модулем SPI?
10. Який регістр використовується для контролю стану модуля, а також для додаткового керування швидкістю обміну?
11. Як задати режим роботи мікроконтролера в режимі «MASTER»?
12. В який спосіб здійснюється передача даних в модулі SPI?
13. Як визначити кінець передачі байта?
14. За якої умови при передачі даних від ведучого до веденого можлива передача у зворотному напрямі?
15. Що означає те, що у модулі реалізовано одинарну буферизацію при передачі та подвійну при прийомі?
16. Поясніть структура SPI-мережі мікроконтролерів.
17. Як обирається потрібний ведений мікроконтролер в SPI-мережі?
18. Поясніть вплив сигналу $\overline{SS}$ на початок обміну даними в різних режимах роботи модуля.
19. Чи можна до інтерфейсу підключати декілька периферійних пристроїв?
20. Назвіть кількість режимів передачі даних та принцип їх роботи.
21. Що є джерелом тактового сигналу при роботі модуля? Як формується цей сигнал?
22. При яких частотах гарантується функціонування мікроконтролера в режимі «Slave»?
23. З яких етапів складається обмін в режимі SPI?
24. Поясніть схему включення мікроконтролерів в режимі програмування послідовним каналом при використанні інтерфейсу SPI.

9. ПРОЕКТУВАННЯ МОДУЛЯ ТАЙМЕРІВ

9.1. Особливості архітектури таймерів AVR-мікроконтролерів

9.1.1. Загальні відомості

Всі сучасні мікроконтролери мають у своєму складі модуль таймерів/лічильників (T/CNT). Нижче розглядаються особливості архітектури модуля таймерів AVR-мікроконтролерів на прикладі сімейства Mega. Ці мікроконтролери, залежно від моделі, мають у своєму складі до шести таймерів/лічильників загального призначення (табл. 9.1) [1; 7].

Таблиця 9.1. **Таймери/лічильники загального призначення**

Таймер/лічильник	**ATmega8515x**	**ATmega8535x**	**ATmega8x/16x/32x**	**ATmega64x/128x**	**ATmega48x/88x/168x**	**ATmega162x**	**ATmega164x/324x/644x**	**ATmega165x**	**ATmega325x/3250x/645x/6450**	**ATmega640x, ATmega1280x/1281x, ATmega2560x/2561x**
Таймер/лічильник T0 (8-розрядний)	•	•	•	•[1)]	•	•	•	•	•	•
Таймер/лічильник T1 (16-розрядний)	•	•	•	•	•	•	•	•	•	•
Таймер/лічильник T2 (8-розрядний)		•[1)]	•[1)]	•	•[1)]	•[1)]	•[1)]	•[1)]	•[1)]	•[1)]
Таймер/лічильник T3 (16-розрядний)				•		•				•
Таймер/лічильник T4 (16-розрядний)										•
Таймер/лічильник T5 (16-розрядний)										•

[1)] Асинхронний таймер/лічильник.

У всіх моделях мікроконтролерів сімейства Mega присутні, як мінімум, два таймери/лічильники – T0 і T1. Таймер/лічильник T0 має мінімальний набір функцій, який залежить від моделі мікроконтролера. У деяких моделях він може використовуватися тільки для відліку та вимірювання часових інтервалів, або як

лічильник зовнішніх подій. В інших моделях до цих функцій додається можливість генерації сигналів із широтно-імпульсною модуляцією (ШІМ) фіксованої розрядності, а також можливість працювати в асинхронному режимі в якості годинника реального часу. Таймер/лічильник T1 також може використовуватися для відліку часових інтервалів і як лічильник зовнішніх подій. Крім того, він може виконувати запам'ятовування свого стану за зовнішнім сигналом. Він може працювати також як широтно-імпульсний модулятор, але вже змінної розрядності та багатоканальний (кількість каналів залежить від моделі). Таймер/лічильник T2 повністю аналогічний таймеру/лічильнику T0. При наявності у мікроконтролері обох таймерів/лічильників, один з них може працювати в асинхронному режимі, а інший – як лічильник зовнішніх подій. Таймери/лічильники T3…T5 за функціональними можливостями ідентичні таймеру/лічильнику T1. У складі всіх AVR-мікроконтролерів є також вартовий таймер, що є неодмінним атрибутом всіх сучасних мікроконтролерів. Цей таймер дозволяє уникнути несанкціонованого зациклення програми, яке може виникати з тих чи інших причин.

9.1.2. Призначення виводів таймерів/лічильників

Кожен таймер/лічильник використовує один або більше виводів мікроконтролера. Як правило, ці виводи є лініями портів введення/виведення загального призначення, а функції, які ними реалізуються при роботі разом з таймерами/лічильниками, є їхніми альтернативними функціями. Всі виводи AVR-мікроконтролерів сімейства Mega, що використовуються таймерами/лічильниками загального призначення, зведено у табл. 9.2.

У цій таблиці також описано функції цих виводів. При використанні альтернативних функцій ліній портів введення/виведення необхідно, як правило, самостійно сконфігурувати ці виводи відповідно до їхнього функціонального призначення.

9.1.3. Переривання від таймерів/лічильників

Для дозволу/заборони переривань від таймерів/лічильників T0, T1 і T2 у частині моделей призначено регістр TIMSK (Timer/Counter Interrupt MaSK Register – регістр маски переривань від таймерів/лічильників).

У нових моделях кількість таких регістрів (3…6) дорівнює числу таймерів/лічильників у конкретній моделі. Так само відбувається з регістрами, що містять прапорці переривань. Назва та формати цих регістрів наведено у [1; 7].

Таблиця 9.2. **Виводи, що використовуються таймерами/лічильниками загального призначення**

Назва	**ATmega8515x**	**ATmega8535x**	**ATmega8x**	**ATmega16x/32x**	**ATmega64x/128x**	**ATmega48x/88x/168x**	**ATmega162x**	**ATmega164x/324x/644x**	**ATmega165x/325x/645x**	**ATmega3250x/6450x**	**ATmega1281x/2561x**	**ATmega640x/1280x/2560x**	**Опис**
T0	PB0	PB0	PD4	PB0	–	PD4	PB0	PB0	PG4	PG4	PD7	PD7	Вхід зовнішнього сигналу таймера T0
OC0	PB0	PB3	–	PB3	PB4	–	PB0	–	–	–	–	–	Вихід схеми порівняння таймера T0
OC0A	–	–	–	–	–	PD6	–	PB3	PB4	PB4	PB7	PB7	
OC0B	–	–	–	–	–	PD5	-	PB4	–	–	PG5	PG5	
T1	PB1	PB1	PD5	PB1	PD6	PD5	PB1	PB1	PG3	PG3	PD6	PD6	Вхід зовнішнього сигналу таймера T1
ICP	PE0	–	–	–	–	–	–	–	–	–	–	–	Вхід захоплення таймера T1
ICP1	-	PD6	PB0	PD6	PD4	PB0	PE0	PD6	PD0	PD0	PD4	PD4	
OC1A	PD5	PD5	PB1	PD5	PB5	PB1	PD5	PD5	PB5	PB5	PB5	PB5	Вихід схеми порівняння таймера T1
OC1B	PE2	PD4	PB2	PD4	PB6	PB2	PE2	PD4	PB6	PB6	PB6	PB6	
OC1C	–	–	–	–	PB7	–	–	–	–	–	PB7	PB7	
T2	–	–	–	–	PD7	–	–	–	–	–	–	–	Вхід зовнішнього сигналу таймера T2
OC2	–	–	PB0	PD7	PB7	–	PB1	–	–	–	–	–	Вихід схеми порівняння таймера T2
OC2A	–	–	–	–	–	PB3	–	PD7	PB7	PB7	PB4	PB4	
OC2B	–	–	–	–	–	PD3	–	PD6	–	–	–	PH6	

Закінчення табл. 9.2.

Назва	**ATmega8515x**	**ATmega8535x**	**ATmega8x**	**ATmega16x/32x**	**ATmega64x/128x**	**ATmega48x/88x/168x**	**ATmega162x**	**ATmega164x/324x/644x**	**ATmega165x/325x/645x**	**ATmega3250x/6450x**	**ATmega1281x/2561x**	**ATmega640x/1280x/2560x**	**Опис**
T3	–	–	–	–	PE6	–	–	–	–	–	PE6	PE6	Вхід зовнішнього сигналу таймера T3
ICP3	–	–	–	–	PE7	–	PD3	–	–	–	PE7	PE7	Вхід захоплення таймера T3
OC3A	–	–	–	–	PE3	–	PD4	–	–	–	PE3	PE3	Вихід схеми порівняння таймера T3
OC3B	–	–	–	–	PE4	–	PB4	–	–	–	PE4	PE4	
OC3C	–	–	–	–	PE5	–	–	–	–	–	PE5	PE5	
T4	–	–	–	–	–	–	–	–	–	–	–	PH7	Вхід зовнішнього сигналу таймера T4
ICP4	–	–	–	–	–	–	–	–	–	–	–	PL0	Вхід захоплення таймера T4
OC4A	–	–	–	–	–	–	–	–	–	–	–	PH3	Вихід схеми порівняння таймера T4
OC4B	–	–	–	–	–	–	–	–	–	–	–	PH4	
OC4C	–	–	–	–	–	–	–	–	–	–	–	PH5	
T5	–	–	–	–	–	–	–	–	–	–	–	PL2	Вхід зовнішнього сигналу таймера T5
ICP5	–	–	–	–	–	–	–	–	–	–	–	PL1	Вхід захоплення таймера T5
OC5A	–	–	–	–	–	–	–	–	–	–	–	PL3	Вихід схеми порівняння таймера T5
OC5B	–	–	–	–	–	–	–	–	–	–	–	PL4	
OC5C	–	–	–	–	–	–	–	–	–	–	–	PL5	
TOSC1	–	PC6	PB6	PC6	PG4	PB6	PD4	PC6	– [1)]	– [1)]	PG4	PG4	Вхід для підключення резонатора
TOSC2	–	PC7	PB7	PC7	PG3	PB7	PD5	PC7	– [1)]	– [1)]	PG3	PG3	Вихід для підключення резонатора

[1)] Окремі контакти введення/виведення, суміщені із XTAL1/XTAL2.

9.1.4. Попередні дільники таймерів/лічильників

Таймери/лічильники містять блок попередніх дільників, які призначено для формування тактових сигналів для таймерів/лічильників: f_{clkTn}, де n – номер таймера. Спрощену структурну схему блоку попереднього дільника таймерів, що не мають асинхронного режиму роботи, наведено на рис. 9.1.

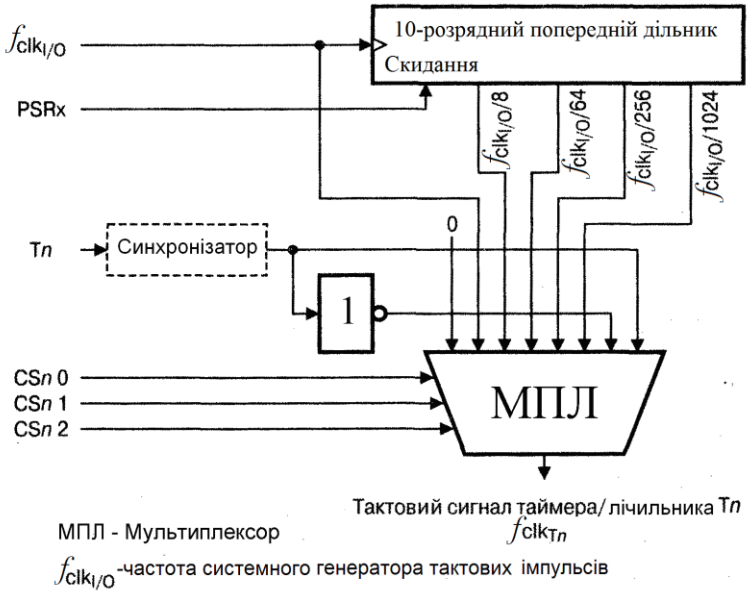


Рис. 9.1. Блок попереднього дільника таймера/лічильника без асинхронного режиму

На рис. 9.2 наведено структурну схему блоку попереднього дільника таймерів/лічильників, що мають можливість роботи в асинхронному режимі.

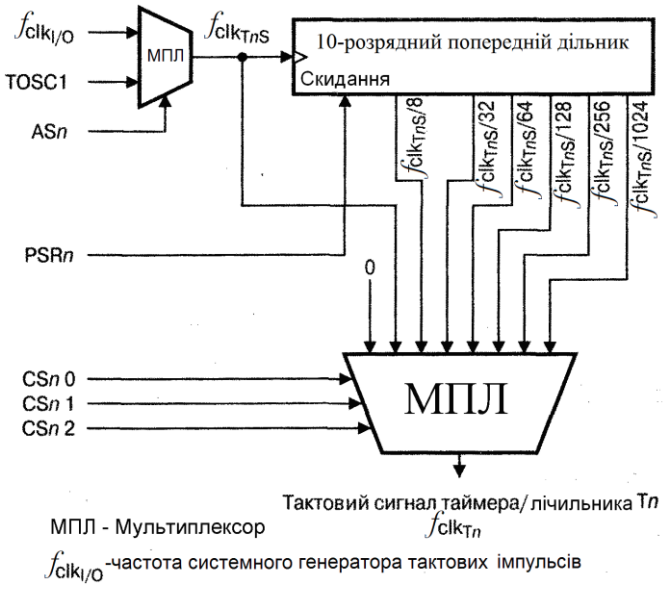


Рис. 9.2. Блок попереднього дільника таймера/лічильника з асинхронним режимом

До складу кожного блоку входять власне 10-розрядний попередній дільник і вихідний мультиплексор МПЛ – селектор тактового сигналу, а для таймерів, що мають можливість роботи в асинхронному режимі, – ще вхідний мультиплексор тактового сигналу. Попередній дільник таймера/лічильника Т0 моделей ATmega64x/128x і таймера/лічильника Т2 інших моделей виконано за схемою, яку наведено на рис. 9.2.

Всі таймери/лічильники кожної моделі сімейства, що не мають асинхронного режиму роботи, використовують той самий 10-розрядний попередній дільник. При цьому керування тактовим сигналом кожного таймера/лічильника здійснюється індивідуально і буде описано при їхньому огляді. Варто розуміти, що попередні дільники працюють незалежно від таймерів/лічильників. Наслідком цього є, зокрема, невизначений проміжок часу (1…N+1 тактів вхідного сигналу, де *N* – коефіцієнт ділення попереднього дільника) між запуском таймера/лічильника та першим його відліком при роботі разом із попереднім дільником. Щоб уникнути цієї невизначеності, можна скористатися засобами, які описано у наступному підрозділі.

9.1.5. Керування попередніми дільниками

Крім керування тактовим сигналом таймера/лічильника, всі мікроконтролери сімейства дозволяють здійснювати скидання попередніх дільників, а деякі моделі дозволяють також здійснювати їхню зупинку. Для цього використовується регістр спеціальних функцій SFIOR а для останніх моделей – регістр GTCCR. Формати цих регістрів для деяких моделей мікроконтролерів наведено у [1; 7] .

9.1.6. Використання зовнішнього тактового сигналу

В якості тактового сигналу деяких таймерів/лічильників, що не мають асинхронного режиму роботи, може використатися зовнішній сигнал (табл. 9.2, див. вище).

Зовнішній сигнал надходить на вхід Т*n (n* = 0...5) мікроконтролера та перш ніж надійти на вхід селектора тактового сигналу, проходить через спеціальний вузол, що має схему синхронізації та детектор фронтів [1; 7].

9.2. Восьмирозрядні таймери/лічильники

Більшість мікроконтролерів сімейства Mega мають два восьмирозрядних таймери/лічильники: Т0 та Т2, які мають три варіанти виконання. Приклади структурних схем всіх трьох варіантів, функції, які вони можуть виконувати, регістри керування та режими роботи описано у [1; 7].

Опис основних режимів роботи восьмирозрядних таймерів/лічильників буде наведено нижче при розгляді шістнадцятирозрядних таймерів/лічильників.

Таймери деяких мікроконтролерів можуть працювати у асинхронному режиму роботи (мають блок RTC), який при відповідному налаштуванні викликає переривання з періодом, який дорівнює 1 секунді. На відміну від основного кварцу мікроконтролера, блок RTC використовує додатковий кварц, який має частоту 32768 Гц. Це дозволяє при діленні 32768/128/256 отримати рівно одну секунду. В моделях ATmega64x і ATmega128x, наприклад, восьмирозрядний таймер/лічильник T0 має блок RTC та може працювати в асинхронному режимі. В цьому режимі на вхід попереднього дільника надходить сигнал від кварцового генератора, що дозволяє використовувати таймер/лічильник як годинник реального часу. Задавачем частоти сигналу може бути як кварцовий резонатор, що підключається до виводів TOSC1 та TOSC2 мікроконтролера, так і сигнал від зовнішньої схеми, що подається на вивід TOSC1. Незважаючи на те що тактовий генератор таймера/лічильника налаштовано на частоту 32768 Гц, частота кварцового резонатора або сигналу від зовнішньої схеми може лежати в межах 0...256 кГц. При цьому вона повинна бути в чотири рази менше частоти тактового сигналу мікроконтролера.

9.3. 16-розрядні таймери/лічильники

9.3.1. Загальна характеристика

У всіх моделях мікроконтролерів сімейства Mega є шістнадцятирозрядний таймер/лічильник T1, а в останніх моделях мікроконтролерів Mega є ще два шістнадцятирозрялних таймери: T4 та T5. Як і восьмирозрядні таймери/лічильники T0 та T2, вони можуть використовуватися для формування часових інтервалів, для підрахунку кількості зовнішніх подій та генерації ШІМ-сигналів змінної розрядності. Шістнадцятирозрядні таймери/лічильники можуть за зовнішнім сигналом зберігати свій поточний стан в окремому регістрі введення/виведення (режим «захоплення»). Таймери/лічильники різних моделей відрізняються тільки кількістю блоків порівняння і відповідно кількістю каналів генерації ШІМ-сигналів [1; 7]. Спрощену структурну схему одного з таймерів/лічильників, наприклад, моделі ATmega64x/ATmega128x, наведено на рис. 9.3.

До складу кожного таймера/лічильника входять наступні регістри введення/виведення (де n = 1, 3, 4, 5):

- 16-розрядний лічильний регістр TCNTn;
- 16-розрядний регістр захоплення ICRn;
- два або три 16-розрядних регістри порівняння OCRnA, OCRnB,

OCRnC;

– два або три 8-розрядних регістри керування TCCRnA, TCCRnB, TCCRnC.

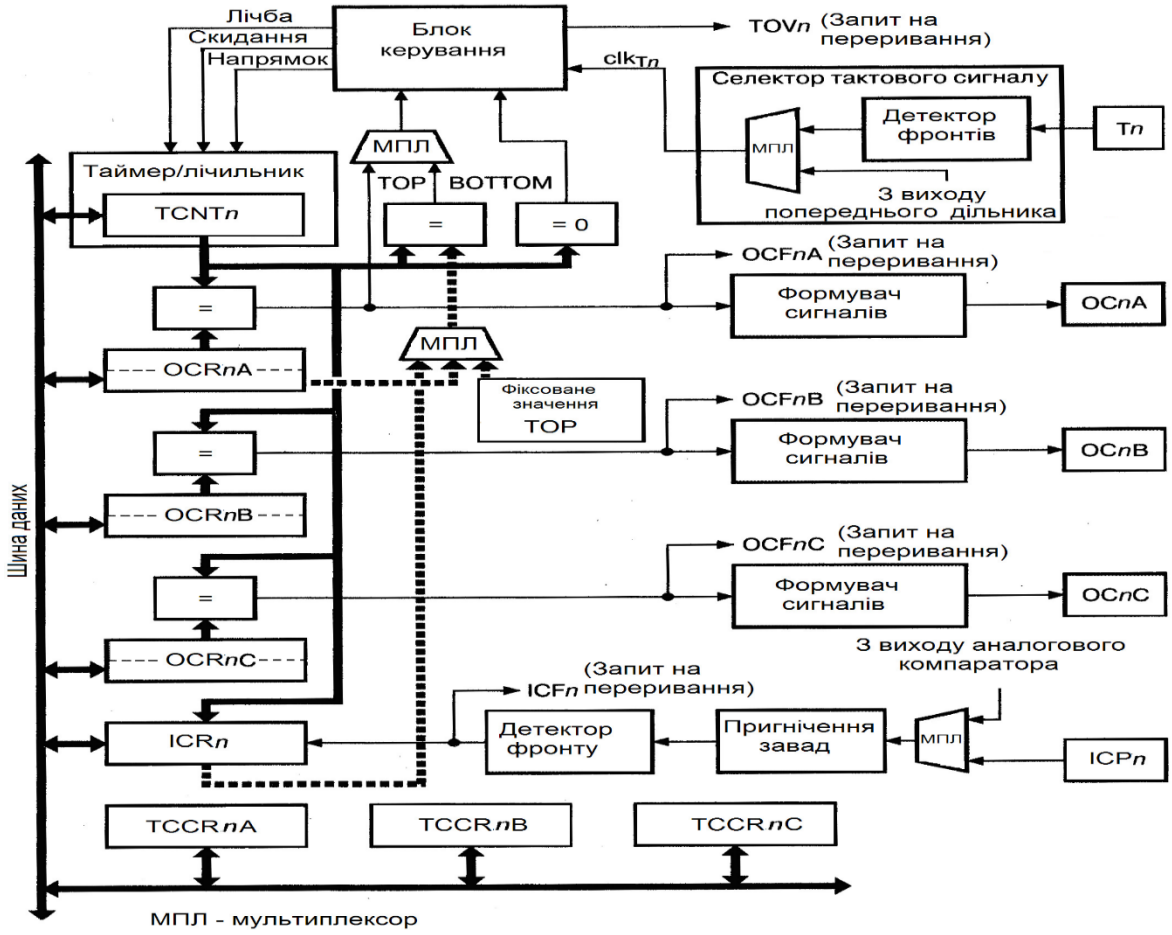


Рис. 9.3. Структурна схема 16-розрядних таймерів/лічильників T1, T3…T5

Кожен із 16-розрядних регістрів фізично розміщено в двох регістрах введення/виведення, назви яких утворюються додаванням до назви регістра літери "H" (старший байт) і "L" (молодший байт). Лічильний регістр таймера лічильника TCNT1, наприклад, розміщується в регістрах TCNT1H:TCNT1L. Адреси регістрів 16-розрядних таймерів/лічильників наведено у табл. 9.3.

Таблиця 9.3. **Регістри 16-розрядних таймерів/лічильників**

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

Закінчення табл. 9.3.

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

Таймери/лічильники T1, T3, T4, T5 можуть генерувати переривання при виникненні наступних подій:

– при переповненні лічильного регістра;
– при рівності лічильного регістра і регістра порівняння (по одному перериванню на кожен блок порівняння);
– при збереженні вмісту лічильного регістра у регістрі захоплення.

Прапорці всіх переривань 16-розрядних таймерів/лічильників знаходяться в регістрах TIFR/ETIFR/TIFRn, а дозвіл/заборона цих переривань здійснюється встановленням/скиданням відповідних прапорців регістрів TIMSK/ETIMSK/TIMSKn [1;7].

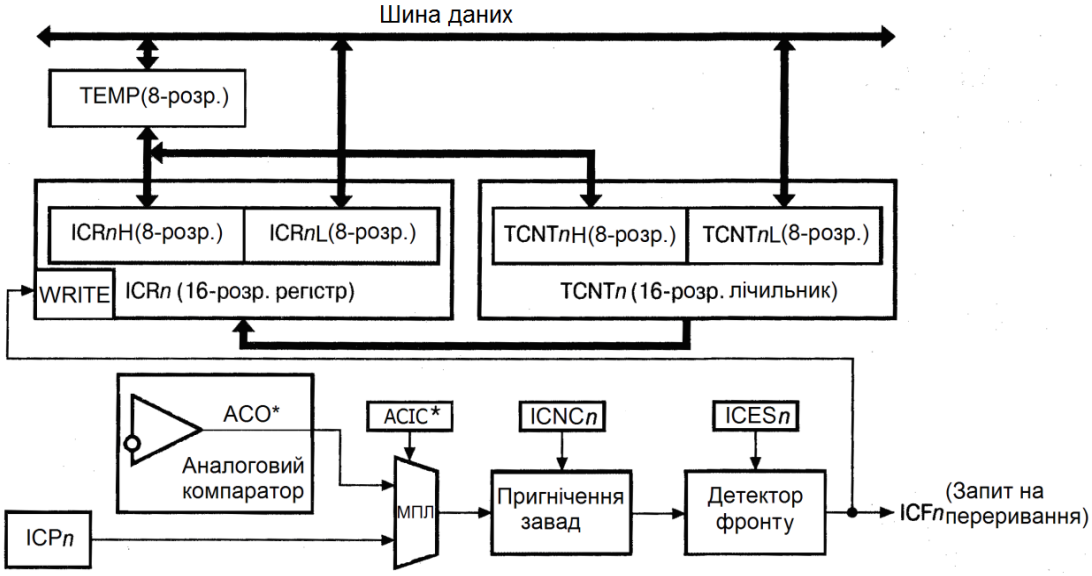
Лічильний регістр таймера/лічильника TCNTn входить до складу основного блоку модуля – блоку реверсивного лічильника. В залежності від режиму роботи модуля вміст лічильного регістра скидається, інкрементується, або декрементується з кожним імпульсом тактового сигналу таймера/лічильника f_{clkn}. Незалежно від того, присутній тактовий сигнал чи ні, регістр доступний у будь-який момент часу, як для зчитування, так і для запису. При цьому будь-яка операція запису в лічильний регістр блокує роботу всіх блоків порівняння на час одного періоду тактового сигналу таймера/лічильника. Після подачі напруги живлення в регістрі TCNTn знаходиться нульове значення. При деяких змінах стану таймера/лічильника, обумовлених режимом його роботи, встановлюється прапорець TOVn відповідного регістра. Дозвіл переривання здійснюється встановленням в одиницю розряду TOIEn відповідного регістра маски.

Регістри OCRnA/OCRnB/OCRnC входять до складу блоків порівняння. Під час роботи таймера/лічильника у кожному машинному циклі виконується порівняння цих регістрів з регістром TCNTn. У випадку рівності вмісту регістра порівняння і лічильного регістра в наступному машинному циклі встановлюється відповідний прапорець OCFnA/OCFnB/OCFnC відповідного регістра прапорців і генерується переривання, якщо воно дозволено.

При виникненні цієї події може змінюватися стан виводу OCnA/OCnB/OCnC мікроконтролера. Щоб таймер/лічильник міг керувати станом якого-небудь з цих виводів, той повинен бути сконфігурований як вихідний (відповідний розряд регістра DDRx повинен бути встановлений у одиницю).

Особливістю роботи блоку порівняння в режимах, які призначено для формування ШІМ-сигналів, є подвійна буферизація запису в регістри порівняння. Вона полягає в тому, що число, яке записується, насправді тимчасово зберігається в спеціальному буферному регістрі. Зміна вмісту регістра порівняння відбувається тільки при досягненні лічильником максимального або мінімального значення.

На рис. 9.4 наведено спрощену структурну схему блоку захоплення.



** Вихід аналогового компаратора може впливати на таймер/лічильник T1 (ACIC=1, див. тему «Аналоговий компаратор»).*

Рис. 9.4. Структурна схема блоку захоплення

До складу блоку захоплення входить регістр захоплення ICRn, який призначено для збереження у визначений момент часу стану таймера/лічильника у регістрі захоплення ICRn. Ця дія може виконуватися або за активним фронтом сигналу на виводі ICPn мікроконтролера, або, наприклад, для таймера/лічильника T1, за сигналом від аналогового компаратора. Одночасно із записом у регістр захоплення встановлюється прапорець ICFn відповідного регістра прапорців і генерується запит на переривання. Дозвіл цього переривання здійснюється встановленням в одиницю розряду TICIEn відповідного регістра маски. У режимах, в яких регістр захоплення визначає модуль лічби таймера/лічильника (табл. 9.8), можливо робити запис у регістр ICRn програмно. Вивід ICPn у цих режимах відключено і функцію захоплення, відповідно, вимкнено.

Схема має 8-розрядний програмно недоступний регістр тимчасового зберігання старшого байта регістрів TCNTn і ICRn – TEMP (див. 9.3.2).

Для керування схемою захоплення використовуються два розряди регістра TCCRnB – ICNCn і ICESn. Розряд ICNCn керує схемою пригнічення завад. Якщо цей розряд скинуто у нуль, схему пригнічення завад вимкнено і захоплення виконується за першим активним фронтом на виході мультиплексора (рис. 9.8). Якщо ж цей розряд встановлено в одиницю, то з появою активного фронту виконується 4 вибірки з частотою, рівною тактовій частоті мікроконтролера. Захоплення буде виконано тільки в тому випадку, якщо усі вибірки мають рівень, що відповідає активному фронту сигналу (логічна одиниця для наростаючого та логічний нуль для спадаючого).

Активний фронт сигналу, тобто фронт, за яким буде виконано збереження вмісту лічильного регістра у регістрі захоплення, визначається станом розряду ICESn. Якщо цей розряд скинуто у нуль, то активним є спадаючий фронт. Якщо розряд встановлено в одиницю, то активним є наростаючий фронт. Для захоплення за сигналом з виводу ICPn, цей вивід повинен бути сконфігурований як вхідний (розряд регістра DDRx, що відповідає виводу, повинен бути скинутий у нуль). Якщо ж він буде сконфігурований як вихідний, захоплення можна буде здійснювати програмно, керуючи відповідним розрядом порту.

Між зміною стану входу блоку захоплення і копіюванням лічильного регістра в регістр захоплення таймера/лічильника проходить деякий час. Цю затримку вносить каскад, що складається із синхронізатора (на рис. 9.4 не показано) і детектора фронту. Величина затримки складає 2,5...3,5 тактів. При ввімкненні схеми пригнічення завад затримка збільшується ще на 4 такти. Для керування таймером/лічильником використовуються три регістри керування: TCCRnA, TCCRnB, TCCRnC. Формат та опис цих регістрів наведено у [1; 7].

9.3.2. Звернення до 16-розрядних регістрів

Кожен 16-розрядний регістр таймерів/лічильників фізично розміщується в двох 8-розрядних регістрах. Для звернення до них потрібно виконати по дві операції зчитування або запису. Для того щоб запис або зчитування обох байтів вмісту 16-розрядного регістра відбувалися одночасно, у складі кожного таймера/лічильника є спеціальний 8-розрядний регістр TEMP (рис. 9.4), який призначено для зберігання значення старшого байта. Цей регістр використовується тільки процесором і програмно недоступний. Для виконання циклу запису 16-розрядного регістра першим повинен бути завантажений старший байт, що записується у регістр TEMP. При наступному записі молодшого байта він з’єднується з вмістом регістра TEMP, і обидва байти у тому самому машинному циклі записуються у 16-розрядний регістр. Якщо потрібно змінити кілька 16-розрядних регістрів таймера/лічильника, а старші байти всіх значень, що записуються, однакові, завантаження старшого байта достатньо виконати тільки один раз. Для виконання циклу зчитування 16-розрядного регістра першим повинен бути зчитаний молодший байт. При його зчитуванні вміст старшого байта записується у регістр TEMP. При наступному зчитуванні старшого байта повертається значення, яке збережено у регістрі TEMP. Виняток становлять тільки регістри порівняння OCRn A/B/C, при зчитуванні яких регістр TEMP не використовується. При виконанні циклу звернення до 16-розрядного регістра таймера/лічильника переривання повинні бути заборонені. В іншому випадку, якщо переривання відбудеться між двома командами звернення до 16-розрядного

регістра, а в підпрограмі обробки цього переривання теж буде зроблено звернення до одного із 16-розрядних регістрів того ж таймера/лічильника, вміст регістра TEMP буде змінено. Як наслідок, результат звернення до 16-розрядного регістра в основній програмі буде неправильним.

9.3.3. Керування тактовим сигналом

Формування тактового сигналу для таймерів/лічильників f_{clkTn} (n = 1, 3, 4, 5) здійснюється блоком попереднього дільника, що був розглянутий у 9.1.4. В якості тактового сигналу f_{clkTn} таймерів/лічильників T1, T3, T4 і T5 може використовуватися:

- системний тактовий сигнал ($f_{clkTn} = f_{clk\ I/0}$);
- масштабований системний тактовий сигнал ($f_{clkTn} = f_{clkI/0}/N$, де N – коефіцієнт ділення попереднього дільника);
- зовнішній сигнал, що надходить на вхід Tn мікроконтролера ($f_{clkTn} = f_{clk\ EXT}$).

Виняток становить лише таймер/лічильник T3 моделей ATmega162x, що не може працювати від зовнішнього тактового сигналу.

Вибір джерела тактового сигналу, а також запуск і зупинка таймерів/лічильників здійснюються за допомогою розрядів CSn2...CSn0 регістра керування таймером TCCRnB відповідно до табл. 9.4.

9.3.4. Режими роботи

Режим роботи таймерів/лічильників T1, T3, T4, T5 визначається станом відповідних розрядів регістра TCCRnA (табл. 9.5). У моделях ATmega161x/163x/323x режими 8...15 відсутні.

Режим Normal

Режим Normal є найпростішим режимом роботи таймерів/лічильників. У ньому лічильний регістр функціонує як звичайний лічильник, що підсумовує. З кожним імпульсом тактового сигналу f_{clkTn} (f_{clkTn}) здійснюється інкремент лічильного регістра.

При переході через значення $FFFF виникає переповнення, і лічба продовжується зі значення $0000. У тому ж такті сигналу f_{clkTn}, в якому скидається в нуль регістр TCNTn, встановлюється в одиницю прапорець переривання за переповненням TOVn. Блоки порівняння обох таймерів можуть використовуватися як для генерації переривань, так і для формування сигналів на виходах OCnA/OCnB/OCnC.

Таблиця 9.4. **Вибір джерела тактового сигналу таймерів/лічильників Tn**

CSn2	CSn1	CSn0	Джерело тактового сигналу	
			T3 в моделях ATmega162x	**Інші**
0	0	0	Таймер/лічильник зупинено	Таймер/лічильник зупинено
0	0	1	$f_{clkI/0}$	$f_{clkI/0}$
0	1	0	$f_{clkI/0}/8$	$f_{clkI/0}/8$
0	1	1	$f_{clkI/0}/64$	$f_{clkI/0}/64$
1	0	0	$f_{clkI/0}/256$	$f_{clkI/0}/256$
1	0	1	$f_{clkI/0}/1024$	$f_{clkI/0}/1024$
1	1	0	$f_{clkI/0}/16$	Вивід Tn, лічба виконується за спадаючим фронтом
1	1	1	$f_{clkI/0}/32$	Вивід Tn, лічба виконується за наростаючим фронтом

Примітка. n = 1, 3, 4 або 5.

Поведінка виходів OCnA/OCnB/OCnC кожного з блоків порівняння таймерів/лічильників Tn визначається станом розрядів COMnx1:COMnx0 регістрів TCCRnA, як показано в табл. 9.6.

Стан виходу будь-якого блоку порівняння також може бути змінено примусово, записом одиниці у розряд FOCnA/FOCnB/FOCnC регістра TCCRnC або регістра TCCRnA. Переривання при цьому не генерується.

Режим «Скидання за збігом» (Chop on Timer Coincidence)

У цьому режимі лічильний регістр теж функціонує як звичайний лічильник, що підсумовує, інкремент якого здійснюється кожним імпульсом тактового сигналу f_{clkTn}. Однак максимально можливе значення лічильного регістра (модуль лічби – TOP) та роздільна здатність лічильника визначається або регістром порівняння OCRnA, або регістром захоплення ICRn (табл. 9.5). А після досягнення максимального значення лічба продовжується зі значення $0000. У тому ж такті сигналу f_{clkTn}, в якому скидається в нуль лічильний регістр, встановлюється прапорець переривання TOVn. Часові діаграми для цього режиму роботи таймера/лічильника приведено на рис. 9.5.

Таблиця 9.5. **Режими роботи 16-розрядних таймерів/лічильників**

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

Примітка. n = 1, 3, 4 або 5.

При досягненні лічильником максимального значення встановлюється прапорець:

– OCFnA, якщо модуль лічби визначається регістром порівняння OCRnA, режим 4;
– ICFnA, якщо модуль лічби визначається регістром захоплення ICRnA, режим 12.

Таблиця 9.6. **Керування виводом OCnA/OCnB/OCnC в режимі Normal**

COMnx1	COMnx0	Опис
0	0	Таймер/лічильник Tn відключено від виводу OCnx
0	1	Стан виводу змінюється на протилежний
1	0	Вивід скидається в нуль
1	1	Вивід встановлюється в одиницю

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

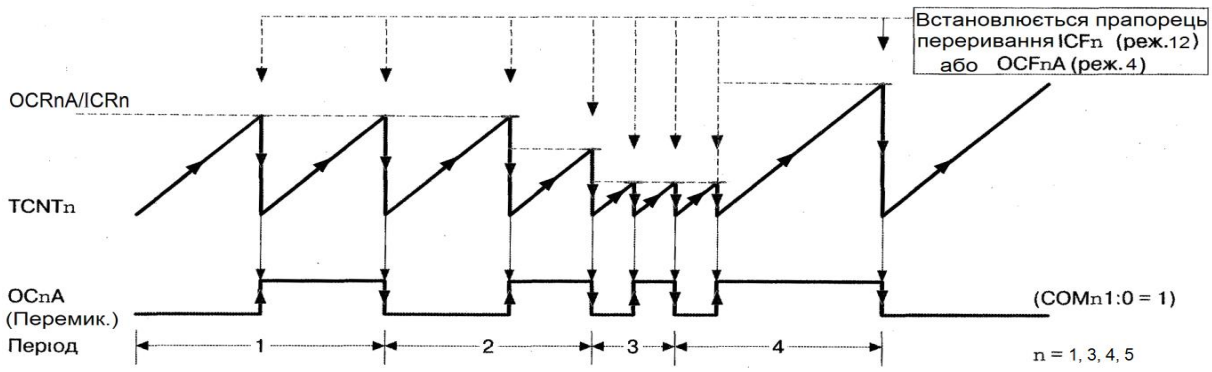


Рис. 9.5. Часові діаграми для режиму CTC

Одночасно із встановленням прапорця може змінюватися стан виводу OCnx мікроконтролера. Поведінка виводу визначається станом розрядів COMnx1:COMnx0 регістрів TCCRnA (табл. 9.7).

Для генерації сигналу заданої частоти необхідно записати в розряди COMnx1:COMnx0 значення "01" (перемикання стану виводу).

Частота сигналу, тривалість імпульсу та період сигналу, який генерується, визначаються виразами:

$$f_{OCnx} = f_{clkI/0}/2*N(1+Y),$$

$$t_{імп}= N*(1+Y)*T_{clkI/0},$$

$$T_{OCnx}=2\ t_{імп}=2*N*(1 + Y)*T_{clkI/0},$$

де N – коефіцієнт ділення попереднього дільника; Y – модуль лічби, обумовлений значенням, що знаходиться в регістрі OCRnA або ICRnA; $f_{clkI/0}$,$T_{clkI/0}$ – відповідно частота та період генератора тактових імпульсів підсистеми введення/виведення.

Таблиця 9.7. **Керування виводом OCnA в режимі CTC**

COMnx1	COMnx0	Опис
0	0	Таймер/лічильник *Tn* відключено від виводу OCnx
0	1	Стан виводу змінюється на протилежний
1	0	Вивід скидається у нуль
1	1	Вивід встановлюється в одиницю

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

Як і в режимі Normal, стан виводу OCnx при необхідності може бути змінено примусово записом одиниці у розряд FOCnx регістра TCCRnC або регістра TCCRnA. Переривання при цьому не генерується і скидання лічильного регістра не виконується.

Режим Fast PWM («Швидкодіючий ШІМ»)

Режим Fast PWM ("Швидкодіючий ШІМ") дозволяє генерувати високочастотний сигнал із широтно-імпульсною модуляцією. Цей режим практично повністю ідентичний однойменному режиму 8-розрядних таймерів/лічильників [1; 7]. Відмінність полягає тільки в тому, що максимальне значення, до якого буде лічити лічильник, може програмно змінюватися (табл. 9.5).

Лічильний регістр у цьому режимі функціонує як лічильник, що підсумовує, інкремент якого здійснюється кожним імпульсом тактового сигналу f_{clkTn}. Стан лічильника змінюється від $0000 до максимального значення, після чого лічильний регістр скидається і цикл повторюється. Максимальне значення лічильника (роздільна здатність ШІМ-сигналу) є або фіксованим значенням, або визначається вмістом окремих розрядів керуючих регістрів таймерів/лічильників (табл. 9.5, див. вище).

Роздільна здатність R визначається виразом:

$$R = \log(TOP + 1)/\log 2 , \qquad (9.1)$$

де TOP – модуль лічби.

Роздільну здатність модулятора у режимі Fast PWM приведено у табл. 9.8.

Таблиця 9.8. **Роздільна здатність модулятора у режимі Fast PWM**

Номер режиму	WGMn3	WGMn2 (CTC1)*	WGMn1 (PWM11)*	WGMn0 (PWM10)*	Роздільна здатність	Модуль лічби (TOP)
5	0	1	0	1	8 розрядів	$00FF
6	0	1	1	0	9 розрядів	$01FF
7	0	1	1	1	10 розрядів	$03FF
14**	1	1	1	0	змінна (2...16)	ICRn ($0003...$FFFF)
15**	1	1	1	1	змінна (2...16)	OCRnA ($0003...$FFFF)
* В моделях ATmega161x/163x/323x. ** В моделях ATmega161x/163x/323x відсутні.						

Примітка. n = 1, 3, 4 або 5.

При досягненні лічильником максимального значення встановлюється прапорець переривання TOVn. Одночасно з ним встановлюється прапорець ICFn (режим 14) або OCFn (режим 15). При рівності вмісту лічильного регістра та якого-небудь регістра порівняння встановлюється відповідний прапорець переривання OCFnA/OCFnB/OCFnC. Одночасно змінюється стан виходу блоку порівняння OCnA/OCnB/OCnC. Поведінка цих виходів визначається вмістом розрядів COMnx1:COMxn0 регістрів TCCRnA (табл. 9.9). Часові діаграми для випадку, коли модуль лічби визначається вмістом регістра OCRnA, показано на рис. 9.6.

При роботі з фіксованим значенням модуля лічильника для задання модуля лічби рекомендується використовувати регістр захоплення ICRn. При цьому регістр OCRnA може бути використано для формування ШІМ-сигналу. Якщо в процесі формування ШІМ-сигналу його частота змінюється дуже часто, для задання модуля лічби лічильника рекомендується використовувати регістр порівняння OCRnA. В цьому випадку за рахунок буферизації запису в регістр порівняння виключається можливість появи несиметричних імпульсів сигналу на виході модулятора [1; 7].

Якщо вміст регістра порівняння дорівнює модулю лічби, то вихід відповідного блоку порівняння переключиться в стійкий стан, обумовлений станом розрядів COMnx1:COMnx0 (табл. 9.9).

Таблиця 9.9. **Поведінка виводу OCnA/OCnB/OCnC в режимі Fast PWM**

COMnx1	COMnx0	Опис
0	0	Таймер/лічильник Tn відключено від виводу OCnx
0	1	WGMn3 = 0: таймер/лічильник Tn відключено від виводу OCnA/OCnB/OCnC*; WGMn3 = 1: стан виводу OCnA змінюється на протилежний
1	0	Скидається в нуль при рівності лічильного регістра і відповідного регістра порівняння. Встановлюється в одиницю при досягненні лічильником максимального значення (неінвертований ШІМ-сигнал)
1	1	Встановлюється в одиницю при рівності лічильного регістра і відповідного регістра порівняння. Скидається в нуль при досягненні лічильником максимального значення (інвертований ШІМ-сигнал)
* Також для моделей ATmega161x/163x/323x.		

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

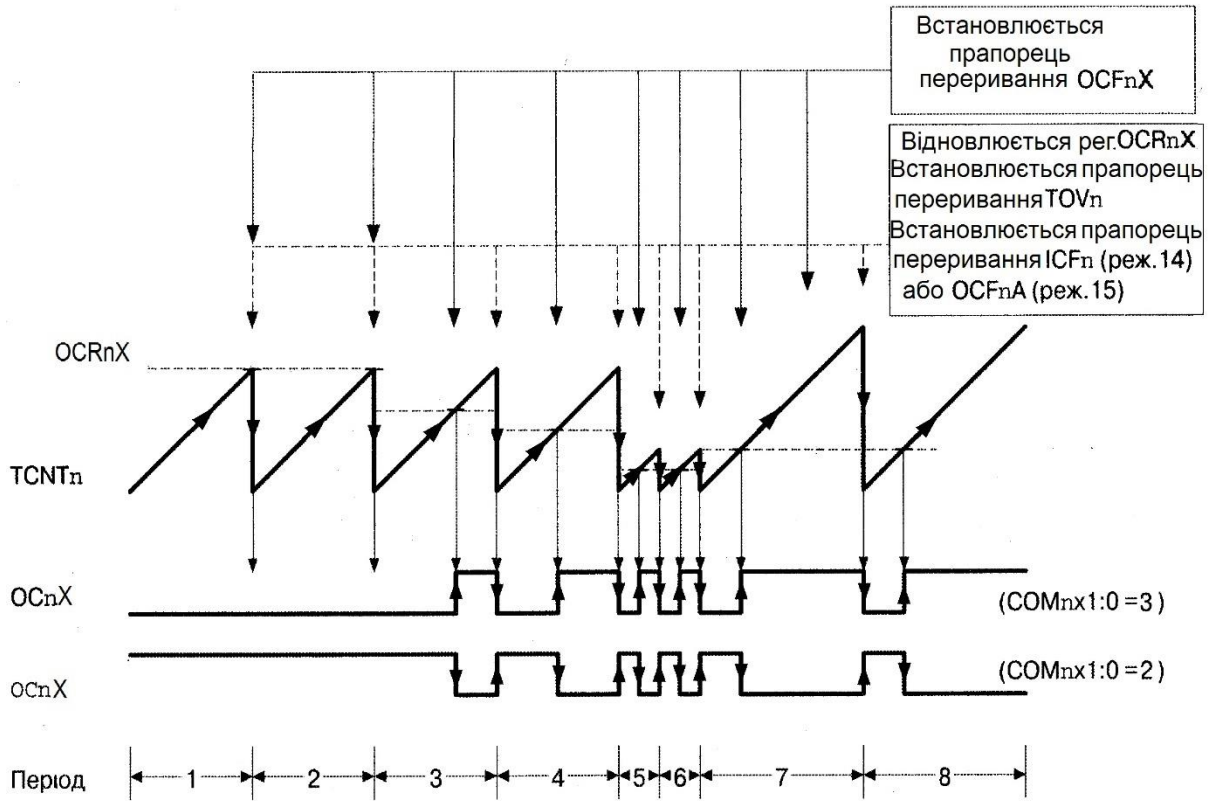


Рис. 9.6. Формування ШІМ-сигналу в режимі Fast PWM

Частота сигналу, тривалість імпульсу та період сигналу, які генеруються, визначаються виразами:

$$f_{OCnx} = f_{clkI/0}/N*(TOP+1),$$

$$t_{імп}= T_{clkI/0} *N*(1 + OCRnx),$$

$$T_{OCnx}=T_{clkI/0}*N*(TOP+1),$$

де N – коефіцієнт ділення попереднього дільника; TOP – модуль лічби; OCRnx – вміст регістра OCRnx; $f_{clkI/0}$, $T_{clkI/0}$ – відповідно частота та період генератора тактових імпульсів підсистеми введення/виведення; n = 1, 3, 4 або 5; x = A, B або C.

При необхідності блок порівняння "A" у цьому режимі може також використовуватися для генерації сигналу «Меандр». Для цього необхідно записати до розрядів COMnA1:COMnA0 значення "01" та встановити WGMn3 = 1, що задає перемикання стану виходу OCnA при виникненні події "Збіг".

Режим Phase Correct PWM («ШІМ з корекцією фази»)

Режим Phase Corect PWM («ШІМ з корекцією фази»), як і режим Fast PWM призначено для генерації сигналів із широтно-імпульсною модуляцією. Однак у цьому режимі лічильний регістр функціонує як реверсивний лічильник, стан якого спочатку змінюється від $0000 до максимального значення, а потім назад до $0000. Відповідно максимальна частота сигналу у цьому режимі в два рази менше максимальної частоти сигналу у режимі Fast PWM.

Максимальне значення лічильника (роздільна здатність модулятора ШІМ-сигналу) є або фіксованим значенням, або визначається вмістом визначених розрядів керуючих регістрів таймера/лічильника (табл. 9.10).

Роздільна здатність R визначається виразом R = log(TOP + 1)/log2, де TOP – модуль лічби.

Як і в режимі Fast PWM, при роботі з якими-небудь фіксованими значеннями модуля лічби, що змінюються інколи, для його задання рекомендується використовувати регістр захоплення. При цьому регістр OCRnA може використовуватися для формування ШІМ-сигналу. Якщо ж у процесі формування ШІМ-сигналу його частота змінюється дуже часто, для задання модуля лічби рекомендується використовувати регістр порівняння OCRnA [1; 7].

Таблиця 9.10. **Роздільна здатність модулятора в режимі Phase Correct PWM**

Номер режиму	WGMn3	WGMn2 (CTC1)*	WGMn1 (PWM11)*	WGMn0 (PWM10)*	Роздільна здатність	Модуль лічби (TOP)
1	0	0	0	1	8 розрядів	$00FF
2	0	0	1	0	9 розрядів	$01FF
3	0	0	1	1	10 розрядів	$03FF
10**	1	0	1	0	Змінна (2…16)	ICRn ($0003...$FFFF)
11**	1	0	1	1	Змінна (2...16)	OCRnA ($0003...$FFFF)
* В моделях ATmega161x/163x/323x. ** В моделях ATmega161x/163x/323x ці режими відсутні.						

Примітка. n = 1, 3, 4 або 5.

При досягненні лічильником максимального значення відбувається зміна напряму лічби, але лічильник залишається в цьому стані протягом одного періоду сигналу f_{clkTn}. У цьому ж такті відбувається відновлення вмісту регістра порівняння.

Якщо модуль лічби визначається регістром ICRn (режим 10) або OCRnA (режим 11), одночасно з відновленням регістра порівняння встановлюється прапорець ICFn або OCFn відповідно. При досягненні лічильником мінімального значення ($0000) також відбувається зміна напряму лічби та одночасно встановлюється прапорець переривання TOVn. При рівності вмісту лічильного регістра і якого-небудь регістра порівняння встановлюється відповідний прапорець OCFnA/OCFnB/OCFnC регістра TIFR. Одночасно змінюється стан виходу блоку порівняння OCnA/OCnB/OCnC. Поведінка виходу визначається вмістом розрядів COMnx1:COMnx0 регістрів TCCRnA (табл. 9.11).

Часові діаграми для випадку, коли модуль лічби визначається вмістом регістра ICRn або OCRnA, показано на рис. 9.7.

При зміні модуля лічби під час роботи таймера/лічильника на виході блоків порівняння можуть з'явитися імпульси, несиметричні відносно середини періоду модуляції. Оскільки відновлення вмісту регістра порівняння відбувається в момент досягнення лічильником максимального значення, період ШІМ-сигналу дорівнює часу між цими моментами. При цьому час прямої лічби визначається

попереднім значенням модуля лічби, а час зворотної лічби – новим значенням. Якщо ці значення різні, то час прямої і зворотної лічби також відрізняються. Результатом цього і є несиметричні імпульси на виході блоків порівняння, як показано на рис. 9.7 (3-й період сигналу).

Таблиця 9.11. **Поведінка виводу OCnA/OCnB/OCnC в режимі Phase Correct PWM**

COMnx1	COMnx0	Опис
0	0	Таймер/лічильник Tn відключено від виводу OCnA/OCnB/OCnC
0	1	WGMn3 = 0: таймер/лічильник Tn відключено від виводу OCnA/OCnB/OCnC*; WGMn3 = 1: стан виводу OCnA змінюється на протилежний
1	0	Скидається в "0" при прямій лічбі і встановлюється в "1" при зворотній лічбі не інвертований ШІМ-сигнал)
1	1	Встановлюється в "1" при прямій лічбі і скидається в "0" при зворотній лічбі (інвертований ШІМ-сигнал)
* Також для моделей ATmega161x/163x/323x.		

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

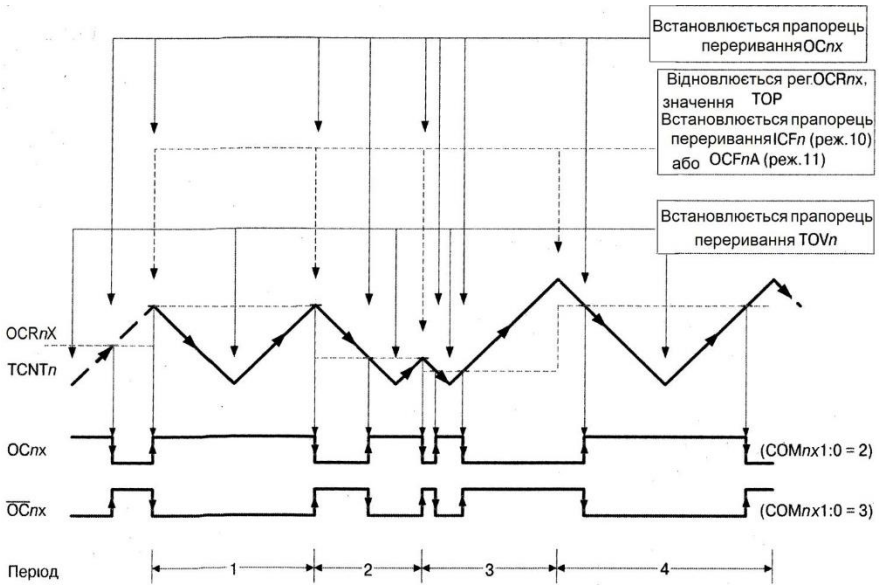


Рис. 9.7. Формування ШІМ-сигналу в режимі Phase Correct PWM

При частій зміні модуля лічби під час роботи таймера/лічильника рекомендується використовувати режим Phase and Frequency Correct PWM, який розглянуто нижче. Якщо ж використовується постійне значення модуля лічби, між цими двома режимами немає ніякої різниці.

Якщо значення, що знаходиться у регістрі порівняння, дорівнює $0000 або модулю лічби (TOP), то при наступному збігу стану лічильника і вмісту регістра порівняння вихід схеми порівняння переключиться в стійкий стан відповідно до табл. 9.12.

Таблиця 9.12. **Стійкі стани виходу схеми порівняння**

COMnx1	COMnx0	Регістр OCRnx	Стан виводу OCnx
1	0	$0000	0
1	0	TOP	1
1	1	$0000	1
1	1	TOP	0

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

Частота, тривалість імпульсу та період сигналу, що генерується, визначаються виразами:

$$f_{OCnx} = f_{clkI/0}/2*N*(TOP+1),$$

$$t_{імп}= T_{clkI/0} *2*N*(1 + OCRnx),$$

$$T_{OCnx}=T_{clkI/0}*2*N*(TOP+1),$$

де *N* – коефіцієнт ділення попереднього дільника; TOP – модуль лічби; OCRnx – вміст регістра OCRnx; $f_{clkI/0}, T_{clkI/0}$ – відповідно частота та період генератора тактових імпульсів підсистеми введення/виведення; n = 1, 3, 4 або 5; x = A, B або C.

Режим «ШІМ з корекцією фази та частоти» (Phase and Frequency Correct PWM)

Режим «ШІМ з корекцією фази та частоти» (Phase and Frequency Correct PWM) дуже схожий на режим Phase Corect PWM. Єдина принципова різниця між ними – момент відновлення вмісту регістра порівняння (рис. 9.8). Відновлення вмісту регістра порівняння відбувається в момент досягнення лічильником мінімального значення, тому кожен період вихідного сигналу є повністю симетричним. Час прямої лічби завжди дорівнює часу зворотної лічби. Вихідні імпульси симетричні відносно періоду і, відповідно, частота сигналу, що

генерується, залишається сталою.

Максимальне значення лічильника (роздільна здатність ШІМ-сигналу) у цьому режимі може визначатися тільки регістрами ICRn або OCRnA таймера/лічильника Tn, як показано у табл. 9.5, див. вище.

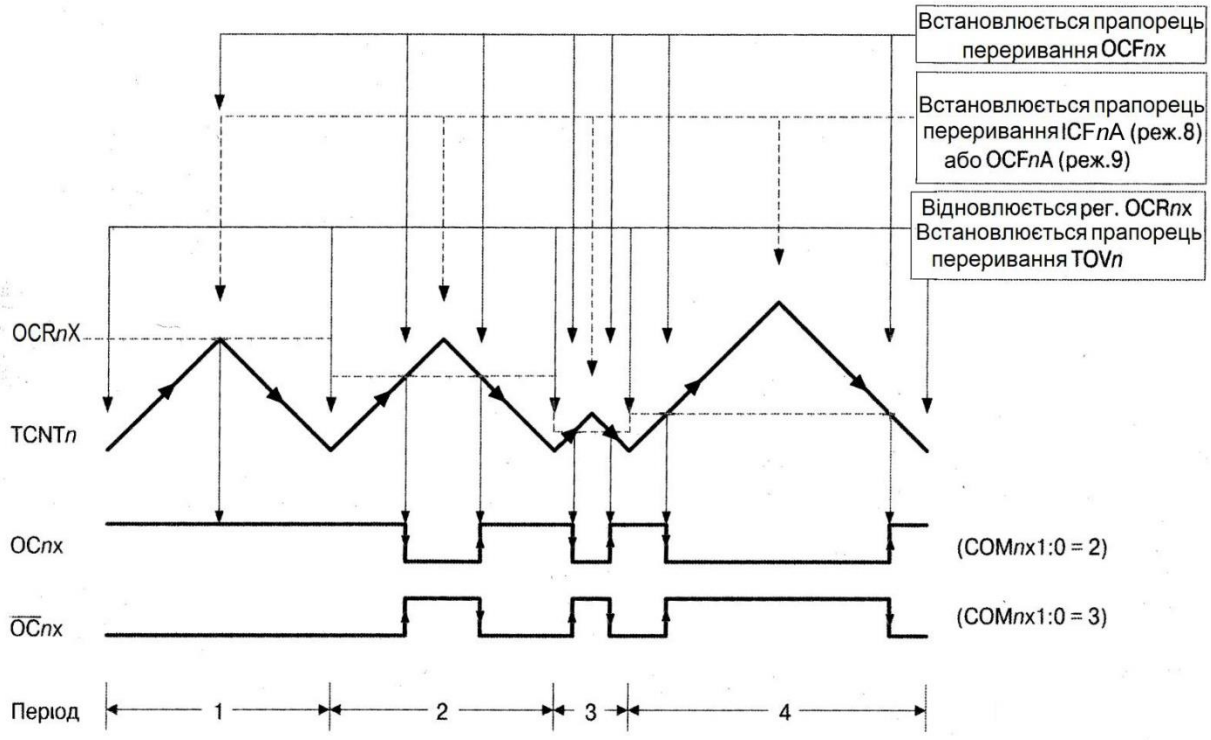


Рис. 9.8. Формування ШІМ-сигналу в режимі Phase and Frequency Correct PWM

Роздільна здатність визначається виразом R = log(TOP+1)/log2, де TOP – модуль лічби (табл. 9.13).

Таблиця 9.13. **Роздільна здатність модулятора в режимі Phase and Frequency Correct PWM**

Номер режиму	**WGMn3**	**WGMn2**	**WGMn1**	**WGMn0**	**Роздільна здатність**	**Модуль лічби (TOP)**
8	1	0	1	0	Змінна (2...16)	ICRnA ($0003...$FFFF)
9	1	0	1	1	Змінна (2... 16)	OCRnA ($0003...$FFFF)

Примітка. n = 1, 3, 4 або 5.

Як і в інших режимах, при роботі з фіксованими значеннями модуля лічби, що змінюються рідко, для його задання рекомендується використовувати регістр захоплення ICRn. При цьому регістр OCRnA може використовуватися для формування ШІМ-сигналу. Якщо ж у процесі формування ШІМ-сигналу його частота змінюється дуже часто, для модуля лічби рекомендується використовувати регістр порівняння OCRnA.

При досягненні лічильником максимального значення відбувається зміна напряму лічби, але лічильник залишається в цьому стані протягом одного періоду сигналу f_{clkTn}. У цьому ж такті встановлюється прапорець ICFn або OCFnA (залежить від того, який з регістрів використовується для задання модуля лічби). При досягненні лічильником мінімального значення ($0000) напрям лічби знову змінюється. При цьому встановлюється прапорець переривання TOVn відповідного регістра і відбувається відновлення вмісту регістра порівняння.

При рівності вмісту лічильного регістра і якого-небудь регістра порівняння встановлюється відповідний прапорець OCFnA/OCFnB/OCFnC. Одночасно змінюється стан виходу блоку порівняння OCnA/OCnB/OCnC. Поведінка виходу визначається вмістом розрядів COMnx1:COMnx0 регістрів TCCRnA (табл. 9.14).

Таблиця 9.14. **Поведінка виводу OCnA/OCnB/OCnC в режимі Phase and Frequency Correct PWM**

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

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

Частота, тривалість імпульсу та період сигналу, що генерується, визначаються виразами:

$$f_{OCnx} = f_{clkI/0}/2*N*(TOP+1),$$

$$t_{імп} = T_{clkI/0} *2*N*(1 + OCRn),$$

$$T_{OCnx} = T_{clkI/0}*2*N*(TOP+1),$$

де *N* – коефіцієнт ділення попереднього дільника; TOP – модуль лічби; OCRnx – вміст регістра OCRnx; $f_{clkI/0}, T_{clkI/0}$ – відповідно частота та період генератора тактових імпульсів підсистеми введення/виведення; n = 1, 3, 4 або 5; x = A, B або C.

Якщо значення, що знаходиться в регістрі порівняння, дорівнює $0000 або модулю лічби (TOP), то при наступному збігу стану лічильника і вмісту регістра порівняння вихід схеми порівняння переключиться в стійкий стан відповідно до табл. 9.12, див. вище.

9.4. Вартовий таймер

Всі AVR-мікроконтролери мають вартовий таймер, основна функція якого – захист пристрою від збоїв. Завдяки вартовому таймеру можна перервати виконання програми, що зациклилася або вийти з інших непередбачених ситуацій, що перешкоджають нормальному виконанню програми. В мікроконтролерах сімейства можна зустріти вартовий таймер у двох виконаннях: стандартному та розширеному [1; 7]. Структурну схему вартового таймера у стандартному режимі приведено на рис. 9.9.

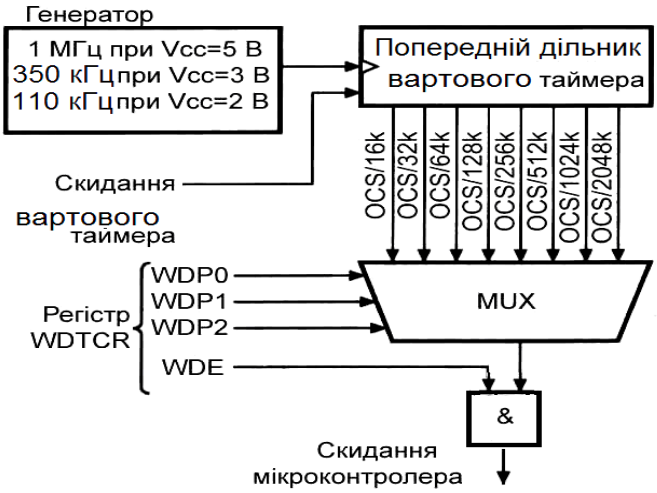


Рис. 9.9. Структурна схема вартового таймера

Вартовий таймер має незалежний тактовий генератор і працює навіть у сплячому режимі Power Down. Частота цього генератора залежить від напруги живлення пристрою, температури та технологічного розкиду [1; 7].

Якщо вартовий таймер включено, то через визначені проміжки часу (при настанні тайм-ауту) виконується скидання мікроконтролера. При нормальному виконанні програми вартовий таймер повинен періодично (через проміжки часу, менші його періоду) скидатися командою WDR. Для керування вартовим таймером у більшості моделей призначено регістр WDTCR (Watch Dog Timer Control Register), який розташовано за адресою $21 ($41). Формат цього регістра та опис його розрядів наведено у [1; 7].

9.5. Моделювання модуля таймера для керування двигуном постійного струму

9.5.1. Опис моделі

Для керування двигуном постійного струму може використовуватись таймер мікроконтролера, який формує ШІМ-сигнал.

На рис. 9.10 наведено робочу модель пристрою керування двигуном постійного струму у пакеті PROTEUS 8.6.

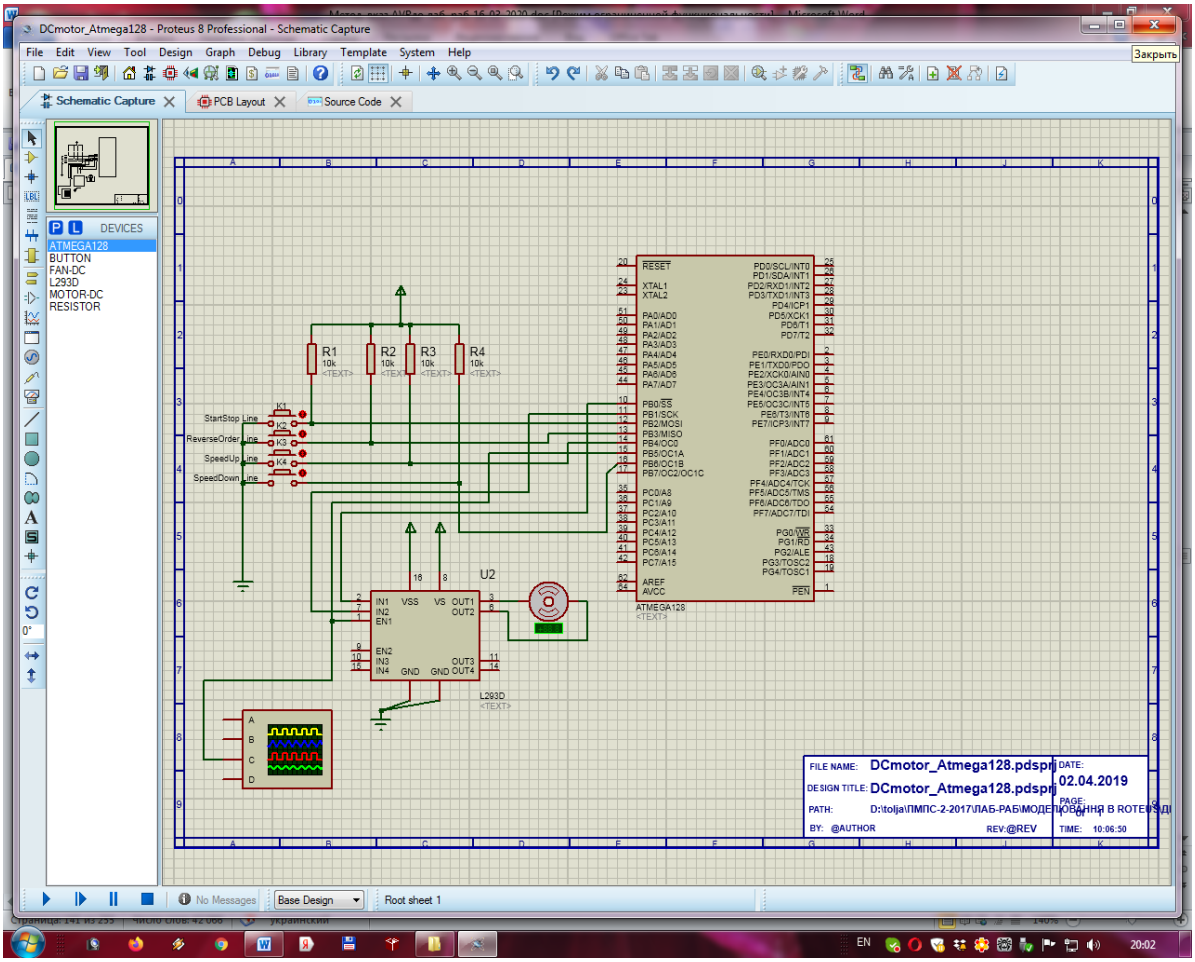


Рис. 9.10. Схема моделі пристрою керування двигуном постійного струму

З лівого боку моделі наведено кнопки керування: K1, K2, K3, K4, на які через резистори R1, R2, R3, R4 відповідно, подається напруга живлення. У правому нижньому куті зображено двигун постійного струму (DC Motor), а трохи лівіше – мікросхему L293D, через яку мікроконтролер керує двигуном за допомогою ШІМ-сигналу. В якості мікроконтролера обрано мікросхему ATmega128, яка знаходиться у правому куті.

Зовнішні резистори використовуються через те, що на вхідних лініях мікроконтролера не підключено внутрішні підтягуючи резистори. На рис. 9.11 наведено зовнішній вигляд та позначення виводів мікроконтролера.

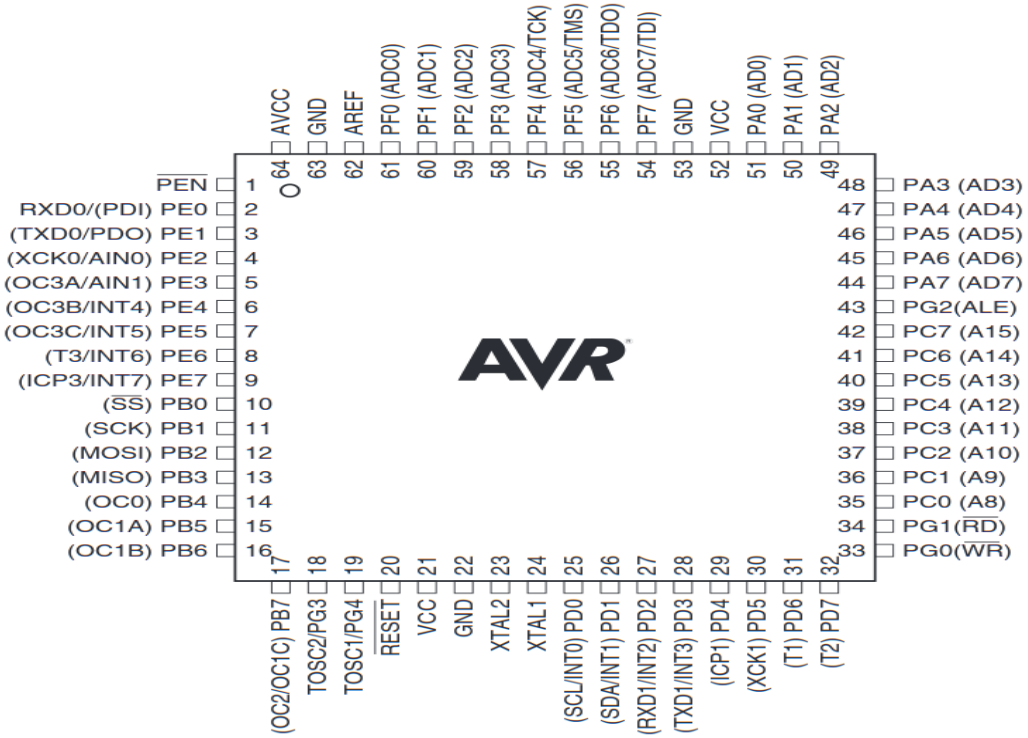


Рис. 9.11. Зовнішній вигляд та позначення виводів мікроконтролера ATmega128

На рис. 9.12 наведено збільшений вигляд кнопок керування.

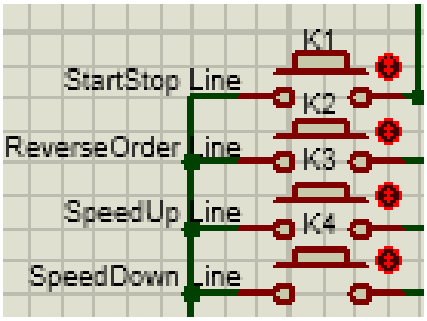


Рис. 9.12. Порядок розташування елементів керування

У вихідному стані кнопки є нормально розімкненими. В цьому випадку на відповідні лінії порту B мікроконтролера (PB) подаються високі рівні напруги,

тобто логічні одиниці. Коли кнопки замикаються, на входи мікроконтролера подаються низьки рівні напруги, тобто логічні нулі. Перемикання сигналу з логічного нуля у логічну одиницю від обраної кнопки сприймається мікроконтролером як відповідний сигнал керування.

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

Кнопка K2 (ReverseOrder Line) відповідає за зміну напрямку обертання двигуна. Одне натискання – одна зміна напрямку. Потрібно мати на увазі, що двигун пристрій інерційний, і він не може зупинитися миттєво і одразу почати обертатися в інший бік. Після зміни напрямку програмно, у нього витрачається певний період часу, щоб фізично зупинитись, та почати обертатись в іншу сторону. Кнопку K2 підключено до 13-го виводу мікроконтролера – PB3, який повинен бути запрограмований як вхід.

Кнопка K3 (SpeedUp Line) відповідає за збільшення швидкості обертання двигуна шляхом поступового зменшення шпаруватості та збільшення постійної еквівалентної напруги імпульсного ШІМ-сигнала до максимуму, який дорівнює амплітуді імпульса – значенню напруги живлення на виводі VS мікросхеми L293D (рис. 9.13).

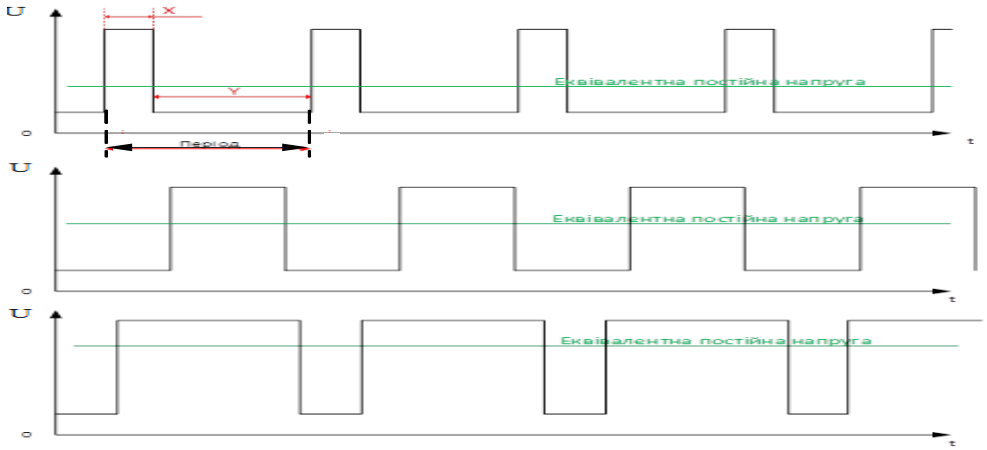


Рис. 9.13. Еквівалентна постійна напруга ШІМ-сигналу

Оскільки двигун інерційний, то швидкість обертання збільшується не стрибкоподібно, а поступово, і потребує певного проміжку часу, щоб вийти на новий програмно встановлений рівень.

Кнопку K3 підключено до 14-го виводу мікроконтролера – PB4, який повинен бути запрограмований як вхід.

Кнопка K4 (SpeedDown Line) відповідає за зменшення швидкості обертання двигуна, для чого вона поступово збільшує шпаруватість та зменшує значення постійної еквівалентної напруги імпульсного ШІМ-сигнала.

Через інерційність двигуна швидкість обертання зменшується не стрибками, а поступово і вимагає для свого зменшення певного проміжку часу. Кнопку K4 підключено до 16-го виводу мікроконтролера – PB6, який повинен бути запрограмований як вхід.

На рис. 9.14 наведено підключення до мікроконтролера мікросхеми L293D.

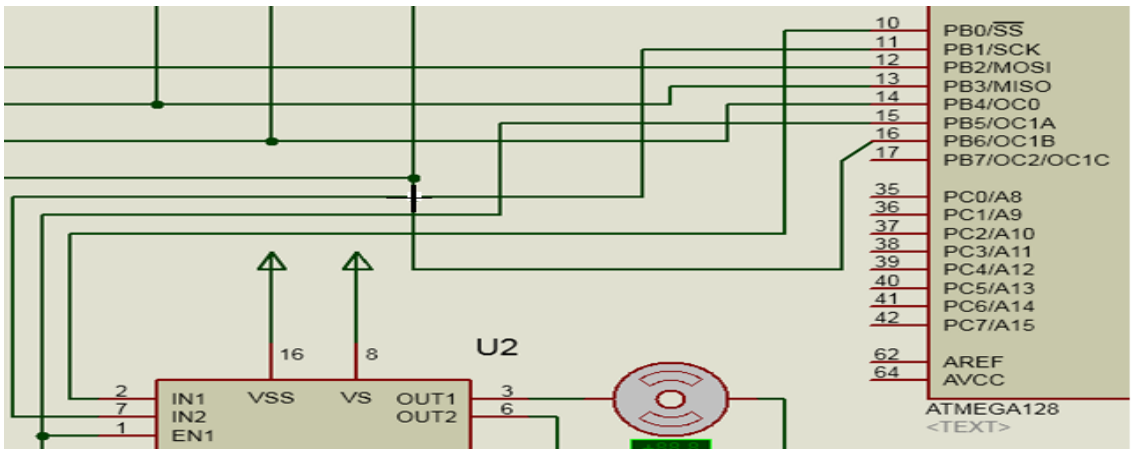


Рис. 9.14. Підключення мікросхеми L293D до мікроконтролера

Виводи 10 та 11 мікроконтролера (PB0 та PB1) запрограмовані як виходи і підключені, відповідно, до входів IN1 та IN2 мікросхеми L293D (U2). В залежності від комбінації сигналів керування на цих входах – нуль або одиниця, двигун буде обертатись у відповідному напрямку.

15-й вивід мікроконтролера – PB5 запрограмовано як вихід і підключено до входу EN1 мікросхеми L293D. З цього виходу мікроконтролера знімається ШІМ-сигнал. У свою чергу вхід EN1 відповідає за роботу першої мостової схеми у складі мікросхеми (див. підрозділ 9.5.4). Коли на ньому високий рівень, на перший міст подається живлення, яке підведено до виводу VS мікросхеми (рис. 9.24). Коли на вході EN1 низький рівень, на перший міст напруга живлення не подається. Завдяки сигналу на цьому вході можна змінювати швидкість обертання двигуна.

Два входи мікросхеми L293D – GND заземлено. На вхід VSS (рис. 9.24) подається напруга живлення самої мікросхеми, а на вхід VS – напруга живлення двигуна.

Підключенням двигуна до мікросхеми L293D зображено на рис. 9.15.

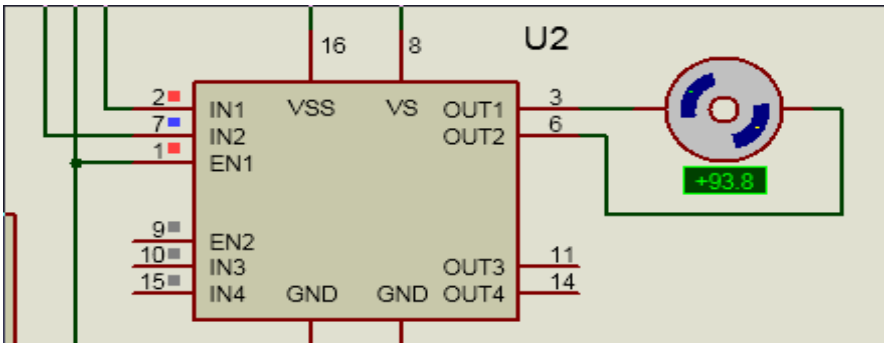


Рис. 9.15. Підключення двигуна постійного струму до мікросхеми L293D

Двигун підключено до 3 та 6 виводів мікросхеми L293D, а саме OUT1 та OUT2. На ці виходи, в залежності від того, чи ввімкнене живлення першої мостової схеми, що керується входом EN1, та в залежності від стану ключів у першій мостовій схемі, які керуються входами IN1 та IN2, або подається напруга живлення, яку ми подали на вхід VS, або напруга не подається.

У випадку, коли на один з виводів – OUT1 чи OUT2 подається напруга живлення, а на інший не подається, виникне різниця потенціалів. Це викличе протікання струму обмоткою збудження двигуна, внаслідок чого він почне обертатися.

На рисунках 9.16…9.20 наведено деякі фрагменти, які демонструють роботу моделі.

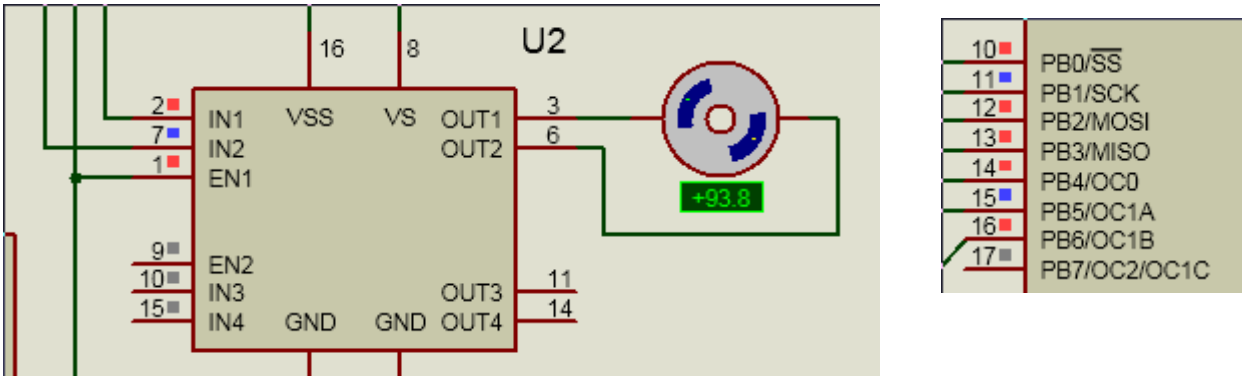


Рис. 9.16. Стан моделі після натискання кнопки «старт/стоп»

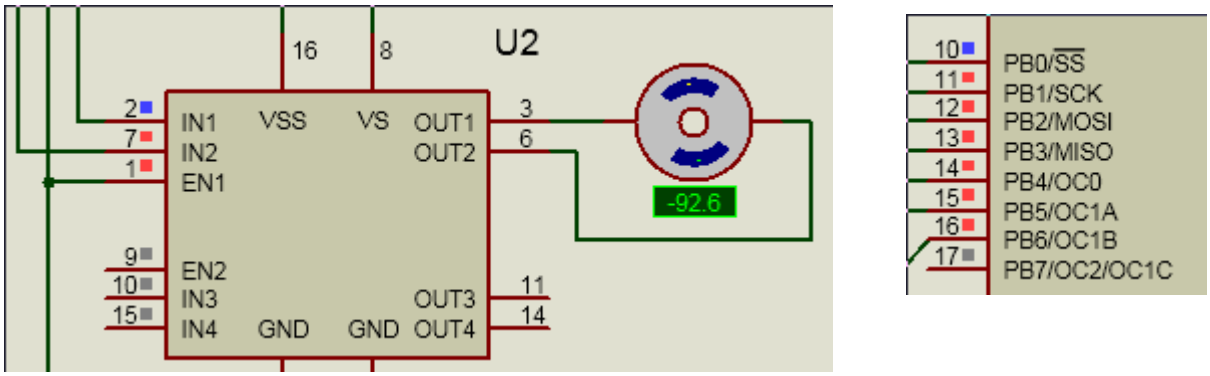


Рис. 9.17. Стан моделі після активації режиму реверсу

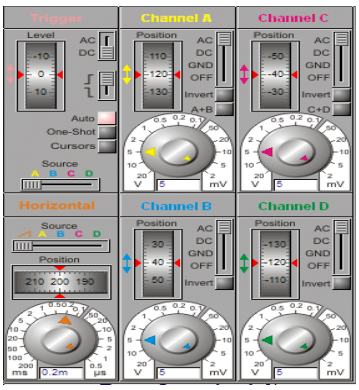


Рис. 9.18. Налаштування осцилографа

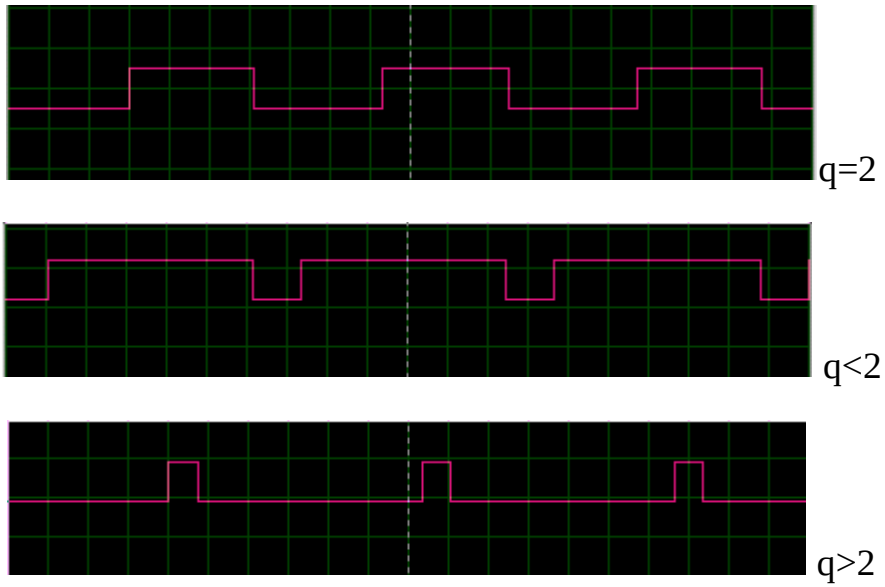


Рис. 9.19. ШІМ-сигнал, що подається на двигун (шпаруватість q = 2, q < 2, q > 2)

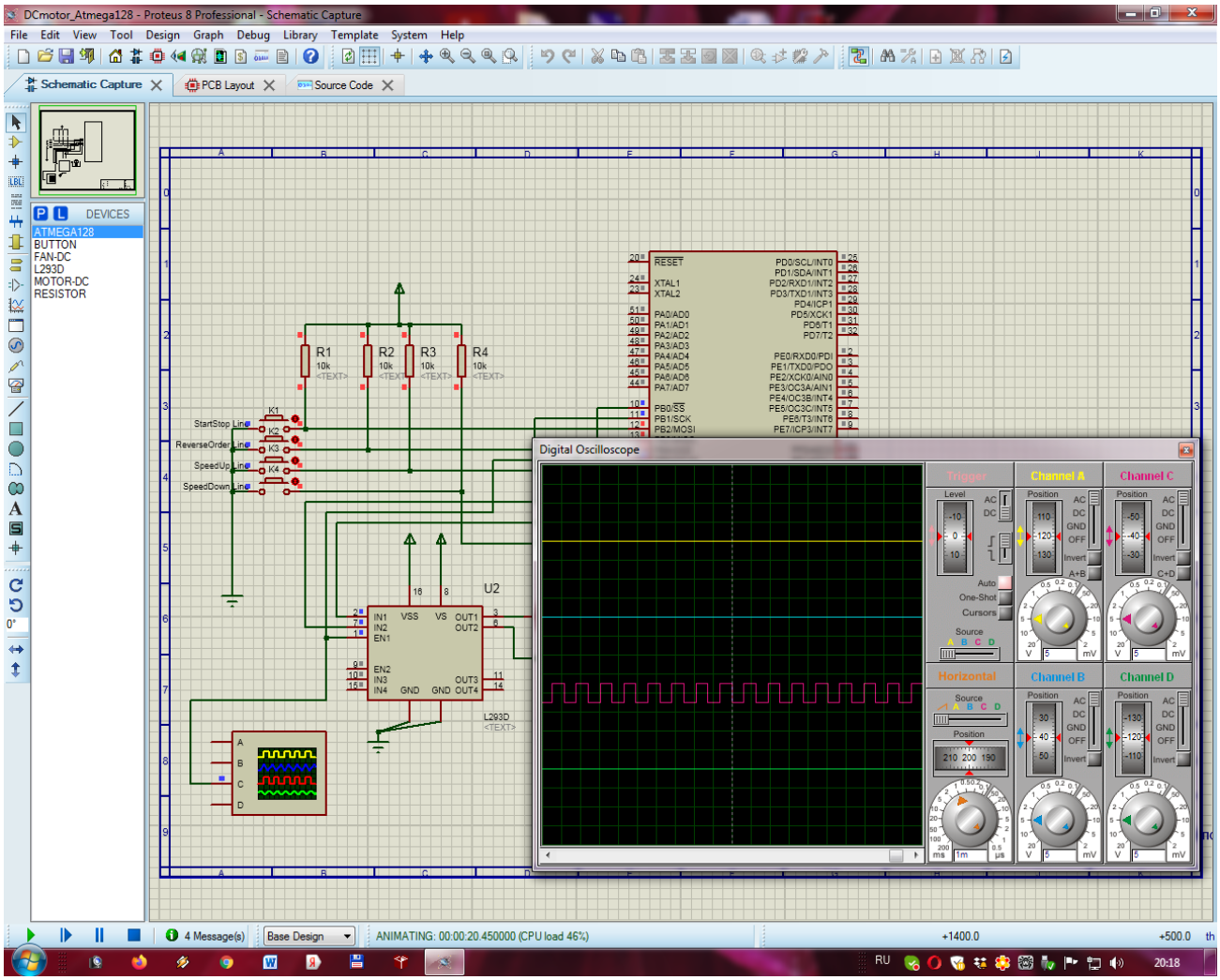


Рис. 9.20. Стан моделі після запуску сценарію

9.5.2. Мікроконтролер

Для контролю та виконання всіх функцій, які покладено на модель, було обрано МК сім`ї AVR – ATmega 128. Він є малопотужним 8-розрядним КМОН-мікроконтролером, який має розширену RISC-архітектуру [1; 7]. Швидкодія МК досягає 1 мільйона операцій в секунду, оскільки більшість команд виконується за один машинний такт.

ATmega 128 має 64 виводи типу вхід – вихід. У мікроконтролер вбудовано Flash-пам’ять програм, яку можна програмувати та перепрограмувати.

Основні характеристики мікроконтролера:

1) Високопродуктивний.
2) Споживає мало енергії.
3) Має розвинену RISC-архітектуру:

а) 131 основна команда, більшість з яких виконується за один машинний такт;

б) 32 робочих регістри загального призначення;

в) Повністю статичний режим роботи.

4) Має енергонезалежну пам’ять програм та даних.
5) Має 128 Кбайт внутрішньої Flash-пам’яті програм з кількістю циклів перепрограмування до 10 000.
6) Має 4096 байт EEPROM-пам’яті даних із кількістю циклів перепрограмування до 100 000.
7) Має 64 виводи типу вхід – вихід.
8) Має JTAG (IEEE1149.1 сумісний) інтерфейс.
9) Можливе сканування пам’яті відповідно до JTAG-стандарту.
10) Периферійні функції:

а) два 8-бітових таймери/лічильники із програмованим режимом порівняння;

б) два 16-бітових таймери/лічильники із програмованим режимом порівняння та захоплення;

в) лічильник реального часу із програмованим генератором;

г) чотири ШІМ-генератори;

д) 8 вхідних каналів 10-бітового АЦП.

11) Синхронні послідовні SPI- та I^2C-інтерфейси.
12) Вбудований аналоговий компаратор.
32) Спеціальні функції:

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

б) внутрішній калібрований RC-генератор;

в) зовнішні та внутрішні джерела переривання;

13) 64 вивідний корпус TQFP та 64-контактний MLF.
14) Напруга живлення: від 4.5 В до 5.5 В.
15) Тактова частота: від 0 до 16 МГЦ.

Опис виводів мікроконтролера:

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

– порт B (PB7...PB0) – 8-розрядний порт двонаправленого введення/виведення із внутрішніми підтягуючими резисторами, які вибираються окремо для кожного розряду;
– порт C (PC7...PC0) – 8-розрядний порт двонаправленого введення/виведення із внутрішніми підтягуючими резисторами, які вибираються окремо для кожного розряду;
– порт D (PD7...PD0) – 8-розрядний порт двонаправленого введення/виведення із внутрішніми підтягуючими резисторами, які вибираються окремо для кожного розряду;
– порт E (PE7...PE0) – 8-розрядний порт двонаправленого введення/виведення із внутрішніми підтягуючими резисторами, які вибираються окремо для кожного розряду;
– порт F (PF7...PF0) – 8-розрядний порт двонаправленого введення/виведення із внутрішніми підтягуючими резисторами, які вибираються окремо для кожного розряду. Якщо активовано інтерфейс JTAG, то резистори на лініях PF7 (TDI), PF5 (TMS) і PF4 (TCK) будуть підключені, навіть якщо виконується скидання. Порт F виконує також функції інтерфейсу JTAG;
– порт G (PG4...PG0) – 5-розрядний порт двонаправленого введення/виведення із внутрішніми підтягуючими резисторами, які вибираються окремо для кожного розряду. Порт G також виконує деякі спеціальні функції;
– RESET – вхід скидання. Якщо на цей вхід подати низький рівень напруги тривалістю більше мінімально необхідної, то буде згенеровано скидання незалежно від роботи схеми синхронізації. Дія імпульсу меншої тривалості не гарантує генерацію скидання;
– XTAL1 – вхід інвертуючого підсилювача внутрішнього генератора та вхід зовнішньої синхронізації;
– XTAL2 – вихід інвертуючого підсилювача внутрішнього генератора;
– AVCC – вивід для подачі живлення порту F і аналого-цифрового перетворювача. Він повинен бути зовні пов’язаний з VCC-входом, навіть якщо АЦП не використовується. При використанні АЦП цей вивід пов’язано з VCC через фільтр низьких частот;
– AREF – вхід підключення джерела опорної напруги АЦП;
– PEN – вхід дозволу програмування для режиму послідовного програмування через інтерфейс SPI. Якщо під час дії скидання при подачі живлення на цей вхід подати низький рівень, то

мікроконтролер переходить у режим послідовного програмування через SPI.

9.5.3. Модуль таймера

Для керування двигуном постійного струму в моделі використовується модуль таймера мікроконтролера, який працює в режимі широтно-імпульсної модуляції. Регулювання швидкості обертання двигуна забезпечується зміною шпаруватості керуючих імпульсів, що у свою чергу викликає зміну постійної складової імпульсного сигналу. Чим менше шпаруватість, а відповідно більше величина відношення «тривалість імпульсу/період», тим більша напруга надходить на двигун, який буде обертатись із більшою швидкістю.

Рекомендована частота сигналу з широтно-імпульсною модуляцією повинна перевищувати 20 кГц, що дозволяє виключити звукові ефекти у двигуні при зміні магнітного поля, яке в ньому утворюється [1].

9.5.4. Драйвер двигуна

Для керування двигуном постійного струму між ним та мікроконтролером використовують відповідну мостову схему, яку зображено на рис. 9.21.

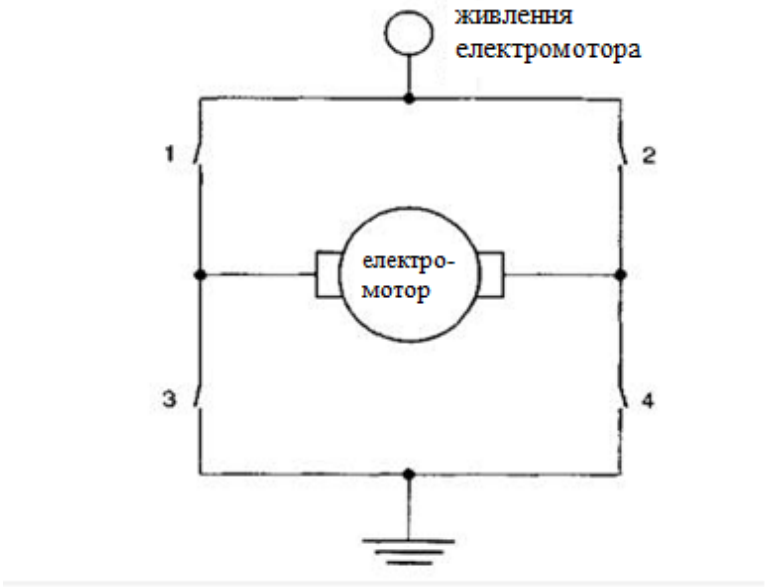


Рис. 9.21. Мостова схема для підключення двигуна до мікроконтролера

Подібну схему мають драйвери двигунів, які використовуються для перетворення керуючих сигналів малої потужності від мікроконтролера у сигнали, потужність яких достатня для керування двигунами..

Існує декілька схем драйверів, які відрізняються як потужністю, так і елементною базою, на якій вони виконані. В моделі застосовано драйвер, який

виконано у вигляді готової до роботи мікросхеми – L293D. Її зовнішній вигляд показано на рис. 9.22.

Рис. 9.22. Зовнішній вигляд мікросхеми L293D

Мікросхема має два драйвери для керування двома електродвигунами досить великої потужності. В мікросхемі є дві пари входів для сигналів керування і дві пари виходів для підключення електродвигунів (рис. 9.23). Крім того, у L293D є два входи для вмикання кожного з драйверів. Ці входи використовуються для керування швидкістю обертання електродвигунів за допомогою широтно-імпульсної модуляції сигналів. Окрім цього, L293D забезпечує розділене живлення для мікросхеми і для керованих нею двигунів, що дозволяє підключати електродвигуни з більшою напругою живлення, ніж у мікросхеми. Останнє корисне також для зменшення завад, які можуть викликатися стрибками напруги під час роботи двигунів.

Обидва драйвери, що входять у склад мікросхеми, мають ідентичні принципи роботи, тому розглянемо лише один з них. Спрощений вид драйвера наведено на рис. 9.23.

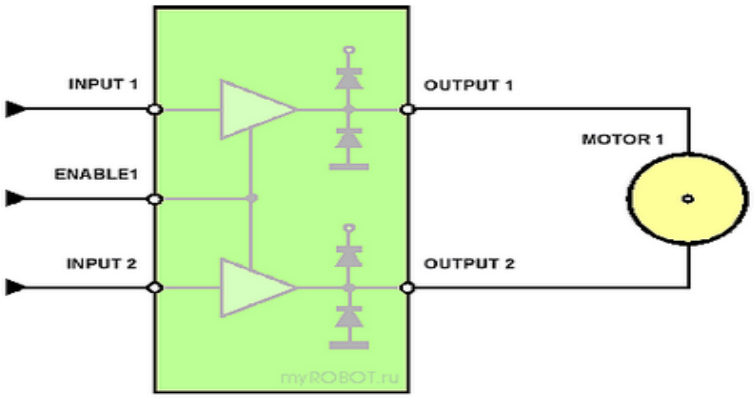


Рис. 9.23. Спрощений вид драйвера мікросхеми L293D

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

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

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

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

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

На рис. 9.24 показано нумерацію та позначення виводів мікросхеми.

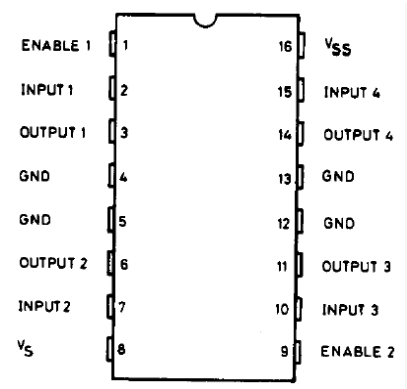


Рис. 9.24. Нумерація та позначення виводів мікросхеми L293D

Призначення виводів:

- входи ENABLE1 та ENABLE2 відповідають за вмикання відповідного драйвера, що входить у склад мікросхеми. На ці входи подається ШІМ-сигнал;
- входи INPUT1 та INPUT2 керують двигуном, який підключено до виходів OUTPUT1 та OUTPUT2;
- входи INPUT3 та INPUT4 керують другим двигуном, який підключено до виходів OUTPUT3 та OUTPUT4;
- контакт Vs з'єднують з додатним полюсом джерела живлення двигунів або просто з додатним полюсом джерела живлення, якщо схема і двигуни живляться від одного джерела. Цей контакт відповідає за живлення електродвигунів;

– контакт Vss з’єднують з додатним полюсом джерела живлення. Цей контакт забезпечує живлення самої мікросхеми;
– чотири контакти GND з’єднують з “землею”(спільним дротом – від`ємним полюсом джерела живлення). Крім того, за допомогою цих контактів зазвичай забезпечують тепловідвід від мікросхеми, тому їх краще всього паяти на достатньо широку контактну поверхню.

Дана мікросхема має наступні технічні характеристики:
– напруга живлення двигунів Vs: 4,5...36 В;
– напруга живлення мікросхеми Vss – 5 В;
– допустимий струм навантаження на кожен канал – 600 мА;
– максимальний струм на виході на кожен канал – 1,2 А;
– логічний нуль вхідної напруги – до 1,5 В;
– логічна одиниця вхідної напруги – 2,3...7 В;
– швидкість перемикань до 5 кГц;
– має захист від перегріву.

Інший спосіб керування двигунами постійного струму засновано на використанні мостових схем типу L298N [1]. Це двоканальний пристрій, який працює від ТТЛШ-рівнів з напругою до 46 В та струмом до 2 А на кожен канал.

9.5.5. Захисні діоди

Як видно з рисунка 9.23 (див вище) схема кожного драйвера містить 4 захисних діоди. Ці діоди призначено для захисту транзисторних ключів у складі драйвера від додатних і від`ємних паразитних імпульсів досить високої амплітуди, які з’являються на обмотках двигуна при їх комутації. Додатні імпульси виникають при запиранні (вимиканні) ключів, а від`ємні – при включенні. Механізм виникнення цих імпульсів описано нижче (рис. 9.25).

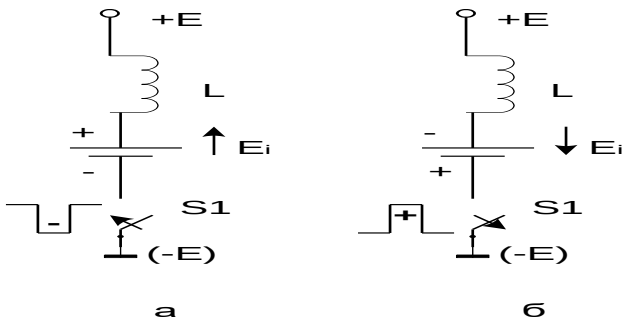


Рис. 9.25. Виникнення паразитних імпульсів при комутації ключів:
а – від’ємний імпульс при включенні ключа; б – додатний імпульс при відключенні ключа

Коли сила струму в обмотці (котушці) змінюється, змінюється і магнітний потік в середині котушки, який збуджує у ній електрорушійну силу (ЕРС) індукції. Ця ЕРС протидіє зміні магнітного потоку (правило Ленца). Якщо, наприклад, струм у котушці різко збільшується (включення ключа), то наростаючий магнітний потік індукує в котушці ЕРС, під дією якої виникає струм, протилежний первісному струму і прагне загальмувати його. При цьому на ключі виникає від'ємний імпульс (рис. 9.25*а*) достатньо великої амплітуди, який може вивести ключ з ладу.

Якщо ж струм в котушці різко зменшується (при виключенні ключа), то магнітний потік, що зменшується, збуджує ЕРС індукції, яка створює струм, спрямований аналогічно вихідного струму, що підтримує в котушці початковий струм. При цьому на ключі виникає додатний імпульс (рис. 9.25*б*) достатньо великої амплітуди, який також може вивести ключ з ладу.

Паразитні імпульси через відкриті діоди передаються на спільний провід: від’ємні – напряму, а додатні через невеликий власний опір джерела живлення.

9.5.6. Програмування таймера мовою Асемблер

Нижче наведено приклад розрахунку та програмування таймера мікроконтролера мовою Асемблер.

Вхідні дані

Треба виконати розрахунок та програмування таймера на основі таких вхідних даних:

- тип МК: AT mega 128;
- номер таймера: Т/С 1;
- на виході PB5 МК треба сформувати інвертований ШІМ-сигнал;
- частота ШІМ-сигналу: f_{PB5} = 25 кГц;
- частота тактового сигналу підсистеми введення/виведення МК: $f_{CLKI/0}$ = 16 МГц ;
- режим роботи таймера: Phase and Frequency Correct PWM (PFCPWM);
- шпаруватість ШІМ-сигналу: Q_{PB5} = 2.

Завдання

Розрахувати:

- коефіцієнт ділення переддільника: Кділ = N;
- модуль лічби: TOP;
- період ШІМ-сигналу: T_{PB5};

– тривалість імпульсу ШІМ-сигналу: $t_{імпPB5}$.

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

Рішення завдання

На рисунку 9.26 наведено формування таймером ШІМ-сигналу у режимі Phase and Frequency Correct PWM.

Величина частоти ШІМ-сигналу f_{PB5} визначається виразом:

$$f_{PB5} = \frac{f_{CLK\ I/O}}{2N(TOP+1)}, \qquad (9.1)$$

де N – коефіцієнт ділення попереднього дільника; TOP – модуль лічби; $f_{clkI/0}$ –частота генератора тактових імпульсів підсистеми введення/виведення.

Згідно табл. 9.5 (див. вище) обираємо режим роботи таймера з номером 8.

Тоді значення модуля лічби TOP визначається вмістом регістра ICR1. При f_{PB5} 25кГц та Кділ = N = 8 шукаємо значення ICR1 = TOP згідно формули:

$$TOP = ICR1 = \frac{f_{CLK\ I/O}}{2*N*f_{PB5}} = \frac{16*10^6}{2*8*25*10^3} = 40 = 00101000B = \$0028. \qquad (9.2)$$

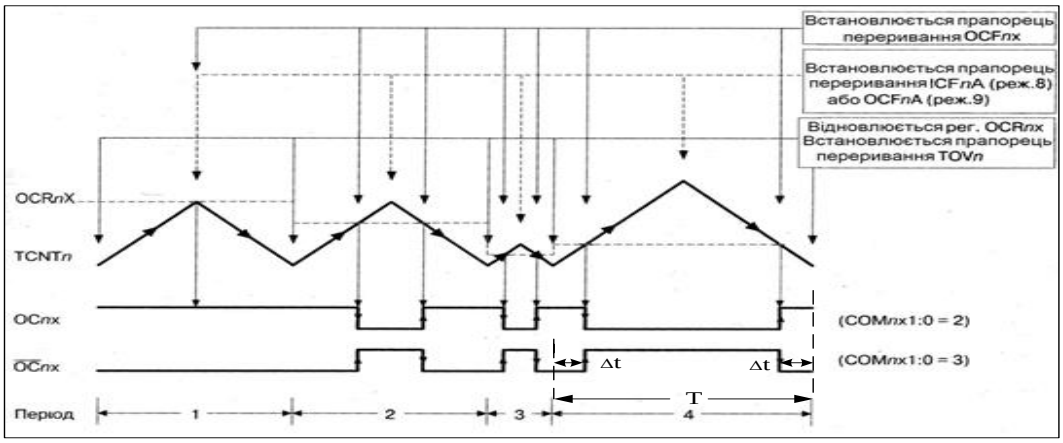


Рис. 9.26. Формування таймером ШІМ-сигналу у режимі Phase and Frequency Correct PWM

Період ШІМ-сигналу:

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

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

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

Згідно з рисунком 9.26

$$t_{імпPB5} = T_{PB5} - 2\,\Delta\,t. \qquad (9.3)$$

Із виразу (9.3) $\Delta\,t = \frac{T_{PB5} - t_{імпPB5}}{2} = \frac{40-20}{2} = 10$ мкс.

У свою чергу, згідно з роботою таймера у режимі Phase and Frequency Correct PWM

$$\Delta t = T_{CLKI/O} * N * OCR1A, \tag{9.4}$$

де OCR1A – регістр порівняння каналу А таймера 1.

Із виразу (9.4)

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

Тоді шпаруватість:

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

$$= \frac{1}{1-2*\frac{1}{f_{CLKI/O}}*N*OCR1A*f_{PB5}} \tag{9.5}$$

Згідно з (9.5) при OCR1A = 20 шпаруватість Q повинна дорівнювати 2:

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

Обчислимо значення Q при збільшенні OCR1A на 5 та при зменшенні на 5:

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

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

Розробка окремих фрагментів програми мовою Асемблер

Зупинка таймера

Для зупинки таймера треба записати в регістр керування TCCR1B (рис. 9.27), адреса якого згідно із табл. 9.3 (див. вище) дорівнює $004E, наведене нижче керуюче слово КС1 = 00000000B = $00:

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

	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	—	WGM13	WGM12	CS12	CS11	CS10	Все модели
Читання (R)/Запис (W)	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Початкове значення	0	0	0	0	0	0	0	0	

а)

	7	6	5	4	3	2	1	0	
	ICNC3	ICES3	—	WGM33	WGM32	CS32	CS31	CS30	ATmega64x/128x ATmega162x ATmega640x/1280x/1281x ATmega2560x/2561x
Читання (R)/Запис (W)	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Початкове значення	0	0	0	0	0	0	0	0	

б)

	7	6	5	4	3	2	1	0	
	ICNC4	ICES4	—	WGM43	WGM42	CS42	CS41	CS40	ATmega640x/1280x/1281x ATmega2560x/2561x
Читання (R)/Запис (W)	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Початкове значення	0	0	0	0	0	0	0	0	

в)

	7	6	5	4	3	2	1	0	
	ICNC5	ICES5	—	WGM53	WGM52	CS52	CS51	CS50	ATmega640x/1280x/1281x ATmega2560x/2561x
Читання (R)/Запис (W)	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Початкове значення	0	0	0	0	0	0	0	0	

г)

Рис. 9.27. Формат регістрів TCCR1B (а), TCCR3B (б), TCCR4B (в), TCCR5B (г)

Тоді програма має вигляд:

LDI R18, $00; R18← КС1 = $00;

$$\left.\begin{array}{l}\text{LDI R27, \$00; R27} \leftarrow \$00\\ \text{LDI R26, \$4E; R26} \leftarrow \$4E\end{array}\right\}\text{X(R27, R26)} \leftarrow \$004\text{E};$$

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

Завантаження регістра TCCR1A

Нижче на рис. 9.28 наведено формат регістра TCCR1A.

Для нашої задачі для формування інвертованого ШІМ-сигналу згідно табл. 9.14 (див. вище) необхідно встановити біти COM1A1 = COM1A0 = 1.

	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	ATmega8515x/8535x ATmega8x/16x/32x ATmega162x
Читання (R)/Запис (W)	R/W	R/W	R/W	R/W	W	W	R/W	R/W	
Початкове значення	0	0	0	0	0	0	0	0	

	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10	ATmega64x/128x ATmega640x/1280x/1281x ATmega2560x/2561x
Читання (R)/Запис (W)	R/W	R/W	R/W	R/W	W	W	R/W	R/W	
Початкове значення	0	0	0	0	0	0	0	0	

	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	—	—	WGM11	WGM10	ATmega48x/88x/168x ATmega164x/324x/644x ATmega165x ATmega325x/3250x/645x/6450x
Читання (R)/Запис (W)	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Початкове значення	0	0	0	0	0	0	0	0	

а)

	7	6	5	4	3	2	1	0	
	COM3A1	COM3A0	COM3B1	COM3B0	FOC3A	FOC3B	WGM31	WGM30	ATmega162x
Читання (R)/Запис (W)	R/W	R/W	R/W	R/W	W	W	R/W	R/W	
Початкове значення	0	0	0	0	0	0	0	0	

	7	6	5	4	3	2	1	0	
	COM3A1	COM3A0	COM3B1	COM3B0	COM3C1	COM3C0	WGM31	WGM30	ATmega64x/128x ATmega640x/1280x/1281x ATmega2560x/2561x
Читання (R)/Запис (W)	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Початкове значення	0	0	0	0	0	0	0	0	

б)

	7	6	5	4	3	2	1	0	
	COM4A1	COM4A0	COM4B1	COM4B0	COM4C1	COM4C0	WGM41	WGM40	ATmega640x/1280x/1281x ATmega2560x/2561x
Читання (R)/Запис (W)	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Початкове значення	0	0	0	0	0	0	0	0	

в)

	7	6	5	4	3	2	1	0	
	COM5A1	COM5A0	COM5B1	COM5B0	COM5C1	COM5C0	WGM51	WGM50	ATmega640x/1280x/1281x ATmega2560x/2561x
Читання (R)/Запис (W)	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Початкове значення	0	0	0	0	0	0	0	0	

г)

Рис. 9.28. Формат регістрів TCCR1A (а), TCCR3A (б), TCCR4A (в), TCCR5A (г)

Для програмування режиму роботи 8 згідно табл. 9.5 (див. вище) необхідно запрограмувати біти: WGM11 = 0; WGM10 = 0.

Інші біти регістра TCCR1A у нашому прикладі не використовуються, тобто запишемо у них нулі. Тоді керуюче слово КС2 = 11000000B = $C0:

7 р.	6р.	5р.	4р.	3р.	2р.	1р.	0р.	КС2
COM1A1	COM1A0	COB1B1	COM1BO	COM1C1	COM1C0	WGM11	WGM10	
1	1	0	0	0	0	0	0 ($C0 =КС2)	

Згідно табл. 9.3 (див. вище) адреса регістра TCCR1A=$004F.

Тоді програма має вигляд:

```
LDI R17, $C0; R17← КС2 = $C0;
LDI R29,  $00;  R27 ← $00
LDI R28,  $4F;  $26 ← $4F } Y(R29, R28)  ← $004F;
ST Y, R17; TCCR5A←R17=$C0.
```

Програмування лінії PB5 на виведення

Для програмування лінії PB5 на виведення необхідно встановити п`ятий розряд регістра DDRB в одиницю. Згідно табл. 5.2 (див. вище) адреса регістра: $0037.

Тоді програма має вигляд:

```
LDI R31, $00;                 } Z(R31, R30)  ← $0037
LDI R30,  $37;  R30 ← $37 ;
LD R19,  Z;  R19← DDRB
ORI R19, $20; R19←00100000B + R19; R19.5←1, інші біти без змін;
ST Z, R19; DDRB←R19; DDRB. 5 ←1.
```

Завантаження регістра ICR1

Згідно табл. 9.3 16-розрядний регістр ICR1 має адресу: $0047 (СБ):$0046 (МБ). В ICR1 треба завантажити: 40 = 00101000B = $0028 (див. вище).

Тоді програма має вигляд:

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

Завантаження регістра OCR1A

Згідно табл. 9.3, 16-розрядний регістр OCR1A має адресу: $004B (СБ): $004A (МБ). В OCR1A треба завантажити: 20 = 00010100B = $0014 (див. вище).

Тоді програма має вигляд:

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

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

Вище при завантаженні регістра TCCR1A було записано WGM11 = 0 та WGN10 = 0, що разом з двома бітами WGM12 = 0 та WGM13 = 1 регістра TCCRB програмують режим роботи №8 (табл. 9.5, див. вище).

Для програмування Кділ = 8 (табл. 9.4. див. вище) та запуску таймера також використовують регістр TCCR1B (рис. 9.27, див вище), в якій треба завантажити керуюче слово: КС3 = $12.

7 р.	6р.	5р.	4р.	3р.	2р.	1р.	0р.
ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10
0	0	0	1	0	0	1	0 (КС3 =12)

(режим 8) (Кділ = 8)

Адреса регістра TCCR1B згідно із табл. 9.3 (див. вище) дорівнює $004E.

Тоді програма має вигляд:

LDI R17, $12; R17← $12;
LDI R29, $00; R29← $00;
LDI R28, $4E; R28← $4E;
ST Y, R17; TCCR1B ← R17 = $12.

9.5.7. Схема алгоритму роботи

Для моделювання пристрою керування двигуном постійного струму у пакеті PROTEUS 8.6 (рис. 9.10, див. вище) розроблено схему алгоритму роботи (рис. 9.29) та робочу програму мовою С, які наведено нижче.

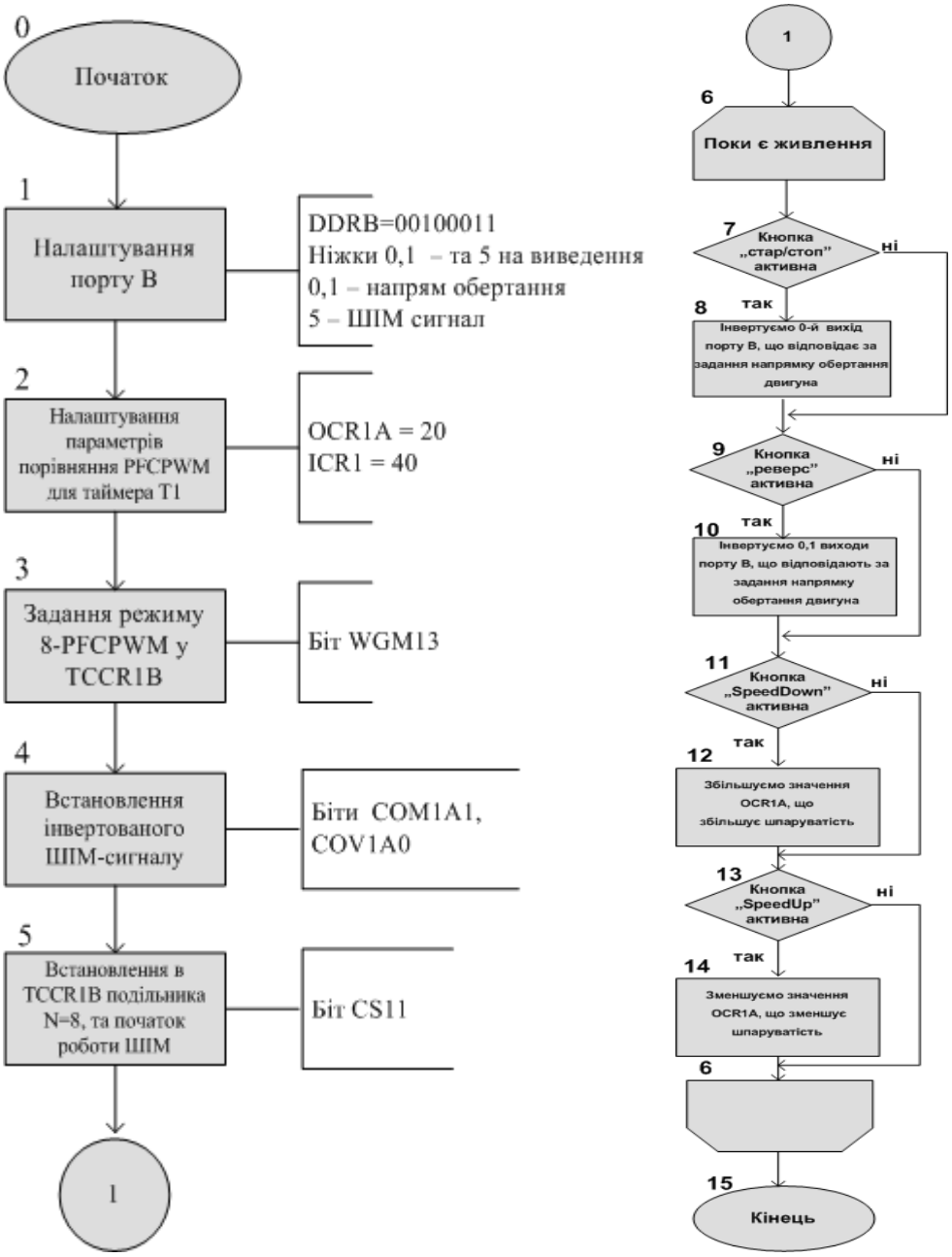


Рис. 9.29. Схема алгоритму роботи

9.5.8. Робоча програма мовою С

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

23 while(1)  // 6
24 {
25 // 7: Start– Stop button action
26 if(!(PINB&4))
27 { flag_start_stop=1; _delay_ms(10); }
28 if(( flag_start_stop==1 )&&(PINB&4))
29 {PORTB^=1; flag_start_stop=0; }      // 8
30 // 9: Reverse button action
31 if(!(PINB&8))
32 { flag_reverse=1; _delay_ms(10); }
33 if(( flag_reverse==1 )&&(PINB&8))
34 {PORTB^=3; flag_reverse=0; }         // 10
35 // 11: Speed –  button action
```

```
36 if(!(PINB&16))
37 { flag_speeddown=1; _delay_ms(10); }
38 if(( flag_speeddown==1 )&&(PINB&16))
39 { if (OCR1A!=40) OCR1A+=delay; flag_speeddown=0; } // 12
40 // 13: Speed + button action
41 if(!(PINB&64))
42 { flag_speedup=1; _delay_ms(10); }
43 if(( flag_speedup==1 )&&(PINB&64))
44 {
45 if (OCR1A!=0)OCR1A– =delay; // 14
46 flag_speedup=0;
47 }
48 } // 6: End of loop
49 } // 15: End of program
```

9.6. Моделювання модуля таймера в якості годинника реального часу

9.6.1. Опис моделі

Робочу модель годинника реального часу у пакеті PROTEUS 8.6 показано на рис. 9.30.

Основним компонентом моделі є мікроконтролер ATmega128 (U2). Мікроконтролер виконує всі обчислення і керування зовнішніми пристроями. Як задаючий елемент для відліку часу використовується годинниковий кварцовий резонатор із частотою 32768 Гц (X1). Кварц підключено до виводів TOSC1, TOSC2 мікроконтролера. У мікроконтролері сигнал з цих виводів проходить блок генератора, передподільника частоти та надходить у 8-розрядний таймер/лічильник T0, де використовується в якості опорної частоти. Крім годинникового кварцу в схемі є основний кварцовий резонатор (X2), з частотою 4МГц. Сигнал з цього кварцу використовується для тактування ядра мікроконтролера та його периферії.

Для індикації значення годин, хвилин і секунд застосовано семисегментний світлодіодний індикатор на 6 розрядів, що має спільний катод. Індикатор відображає час по 2 розряди: 1 і 2 – години, 3 і 4 – хвилини, 5 і 6 – секунди, якщо рахувати зліва, між якими ставиться крапка, наприклад: "12.45.17".

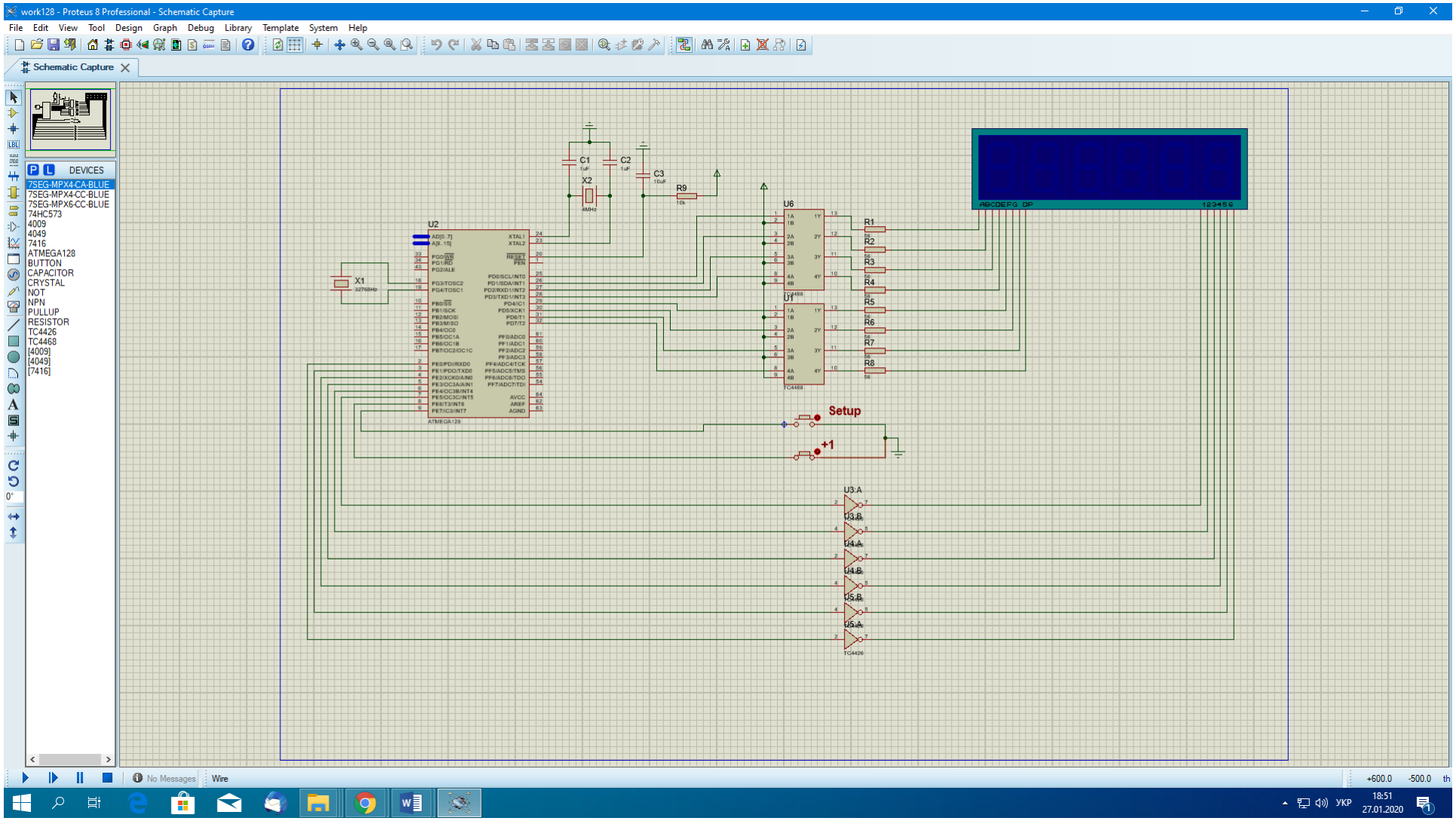


Рис. 9.30. Робоча модель годинника реального часу

Для управління індикатором потрібно подавати керуючі сигнали на лінії включення сегментів (A…G, DP) і обирати активний розряду за допомогою ліній 1…6. Оскільки виходи мікроконтролера мають не достатню здатність навантаження за струмом, перед індикатором використовуються дві додаткові мікросхеми: TC4468 (U2, U3), які виконують функцію буфера-защіпки з трьома станами. Ці мікросхеми мають 4 входи і 4 виходи, логічні стани яких ідентичні входам, але здатні віддавати у навантаження струм до 35 мА. Крім цих мікросхем для керування індикатором використовується також мікросхеми TC4426 (U4…U6), які виконують функцію буфера з відкритим стоком, який інвертує. Цей буфер здатний віддавати струм у навантаження до 1,5А, коли його вихід знаходиться в стані логічного нуля. Наявність інвертора дозволяє вибирати потрібний розряд на індикаторі за допомогою логічної одиниці на відповідному виході мікроконтролера. Для обмеження струму через світлодіодний індикатор використовуються резистори R1…R8.

Для виведення цифр в програмі застосовано метод динамічної індикації. Кожна цифра виводиться окремо одна за одною. В певний момент часу на індикаторі горить лише одна цифра, але за рахунок високої швидкості виведення для ока цей процес не помітний, і все виглядає так, ніби все цифри горять одночасно.

Для виведення бітового образу числа на відповідний розряд індикатора використовується порт D, а для вибору активного розряду індикатора – шість молодших розрядів порту E мікроконтролера.

Старші два біти порту E використовуються як входи зовнішніх переривань INT6 і INT7. До цих входів підключено кнопки: «+1» і «Setup». Кнопка «Setup» виконує циклічне переключення режимів (робочий режим -> налаштування годин -> налаштування хвилин -> налаштування секунд -> робочий режим). Кнопка «+1» працює тільки в режимах налаштування і виконує інкремент значення часу (в залежності від режиму налаштування – годин, хвилин або секунд). При натисканні та утриманні цієї кнопки відбувається автоматичний інкремент відповідного значення.

Для живлення схеми в реальному пристрої може використовуватися стабілізатор напруги LM7805, з вихідним напругою 5В і максимальним струмом в 1А. Для додаткової стабілізації напруги застосовуться конденсатори на вході і виході стабілізатора. Для фільтрації високочастотних завад використовується конденсатор C5 ємністю 100нФ, розміщений в безпосередній близькості від мікроконтролера.

Для автоматичного формування сигналу RESET на однойменному вході мікроконтролера при включенні живлення застосовано RC-ланцюжок (C3, R9).

Роботу годинника засновано на використанні RTC-блоку 8-розрядного таймера T0 мікроконтролера. Цей блок при відповідному налаштуванні викликає переривання з періодом, який дорівнює 1 секунді. На відміну від основного кварцу мікроконтролера, блок RTC використовує додатковий кварц, який має частоту 32768 Гц. Це дозволяє при діленні 32768/128/256 отримати рівно 1с. Для керування роботою годинника використовуються дві кнопки, контроль стану яких виконується за допомогою зовнішніх переривань INT6 та INT7.

Мікросхема TC4468

Ця мікросхема є чотирьохканальним буфером-защіпкою з третім станом. Завдяки наявності цього стану і високої здатності навантаження виходів, дана мікросхема підходить для використання в якості шинного формувача.

Регістр-защіпка керується за допомогою входу 1B/2B/3B/4B (активація засувки). Коли на цьому вході присутня логічна одиниця, значення на виходах Y будуть повторювати значення з входів A. Якщо на вході 1B/2B/3B/4B логічний нуль, то виходи будуть зберігати своє значення до тих пір поки на цьому вході знову не з`явиться логічна одиниця.

Мікросхема TC4426

Дана мікросхема містить два інвертуючих буфери з виходами типу "відкритий сток". У пристрої є три мікросхеми, які виконують функцію шести незалежних інверторів. Кожен буфер здатний віддавати струм у навантаження до 1,5A, коли його вихід знаходиться в стані логічного нуля. Наявність інвертора дозволяє вибирати потрібний розряд на індикаторі за допомогою логічної одиниці на відповідному виході мікроконтролера.

9.6.2. Схема алгоритму роботи

Схему алгоритму роботи наведено на рис. 9.31.

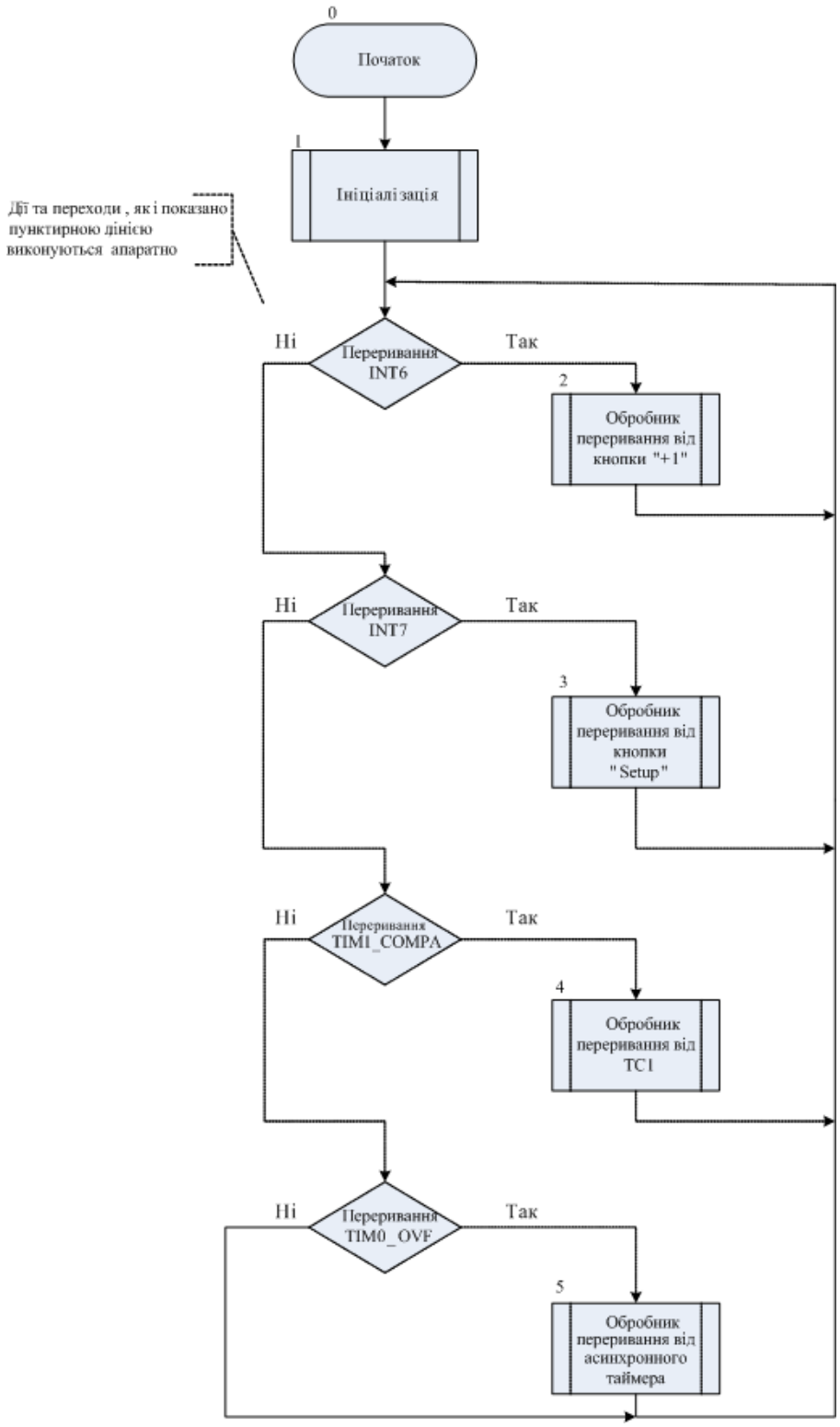


Рис. 9.31. Схема алгоритму роботи

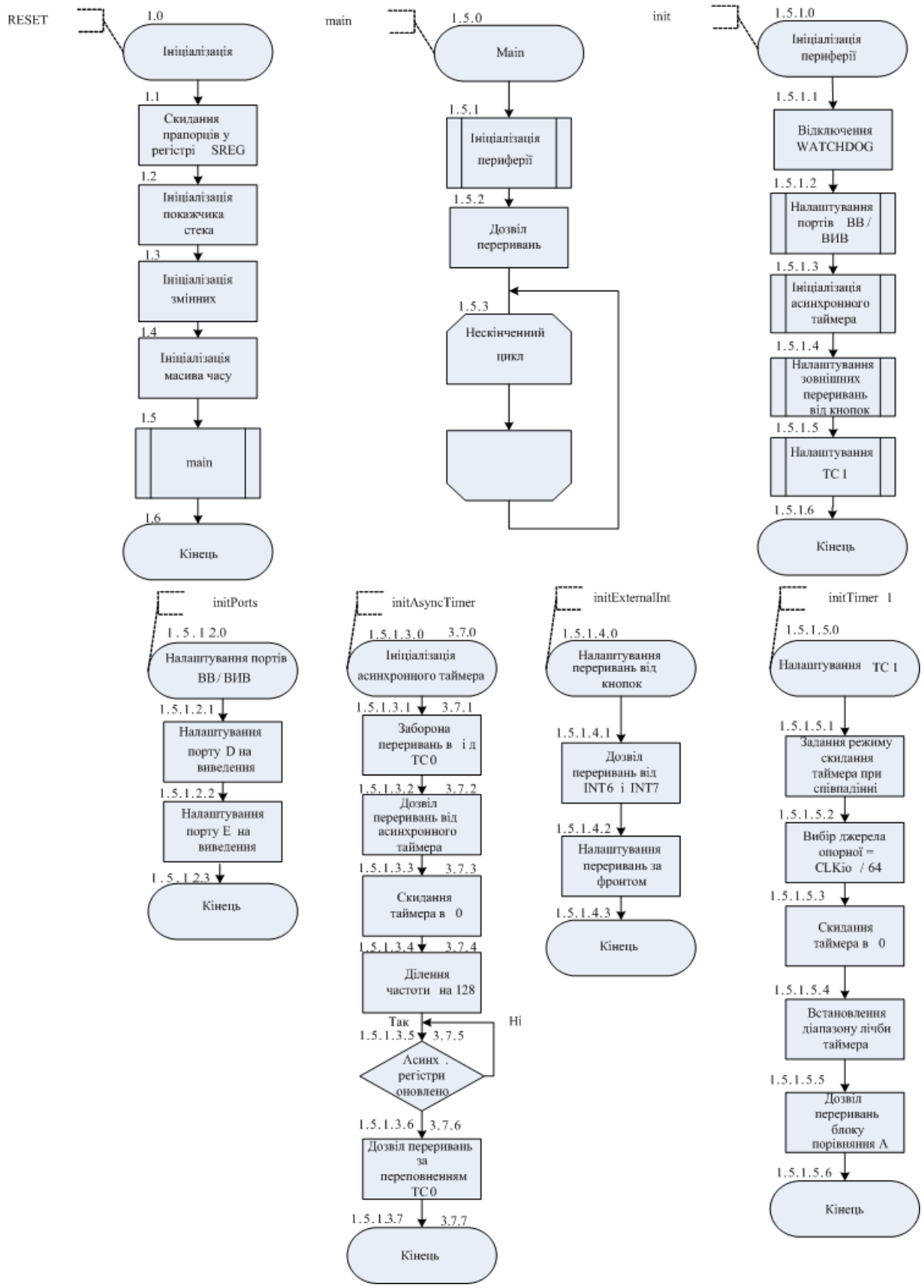


Рис. 9.31. (Продовження). Див. також с. 301

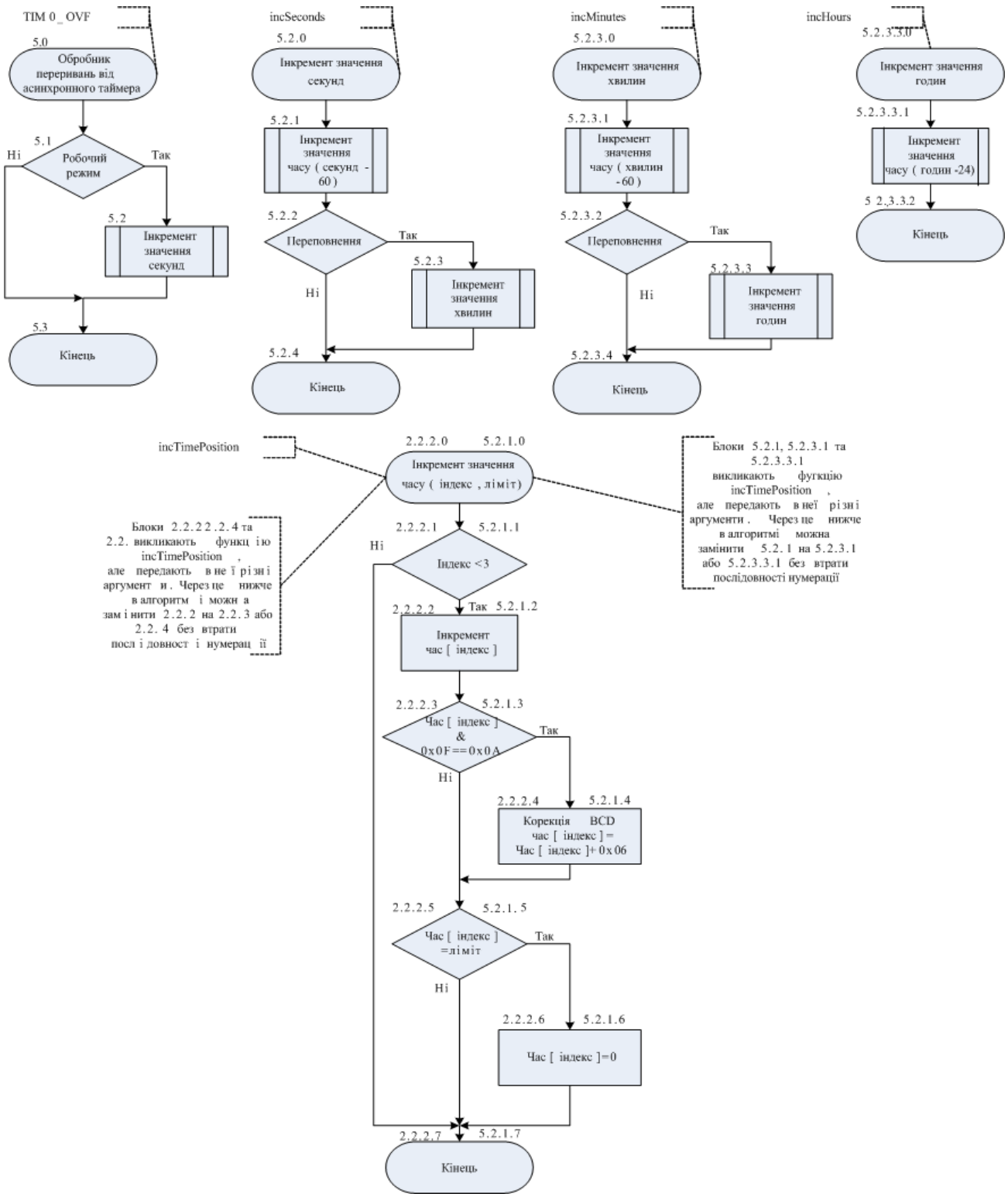


Рис. 9.31. (Продовження). Див. також с. 301, 302

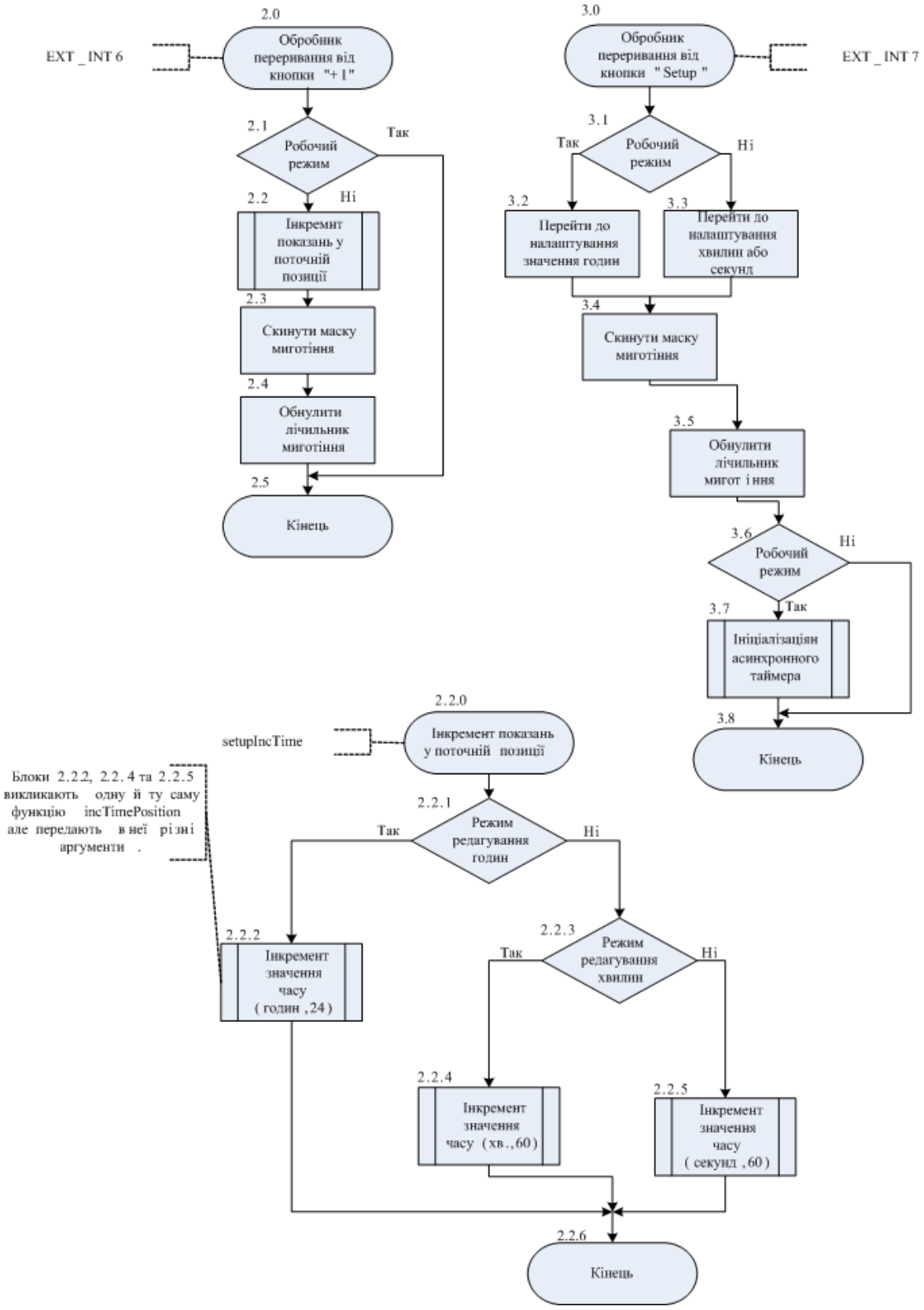


Рис. 9.31. (Продовження). Див. також с. 301…303

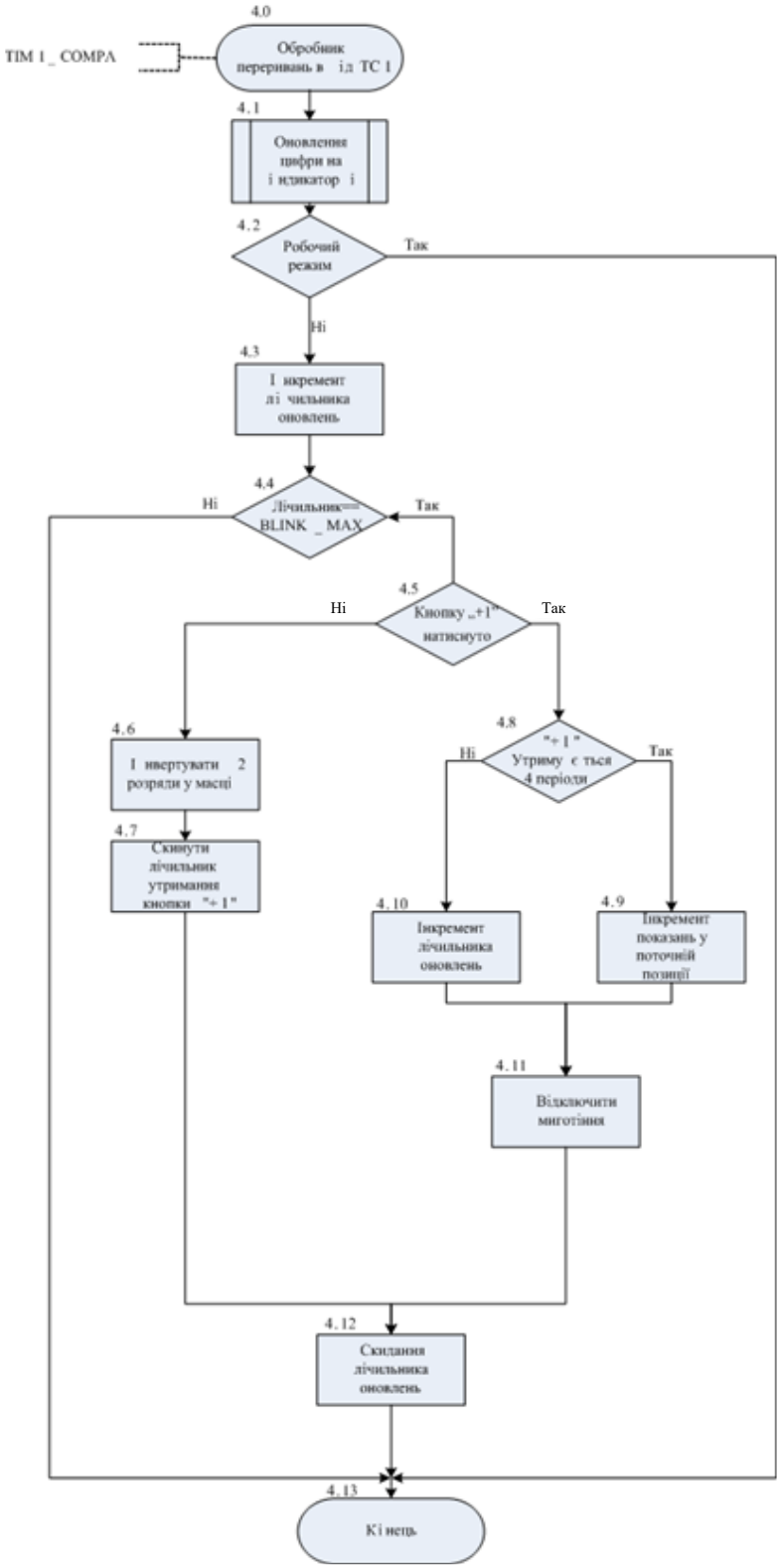


Рис. 9.31. (Продовження). Див. також с. 301…304

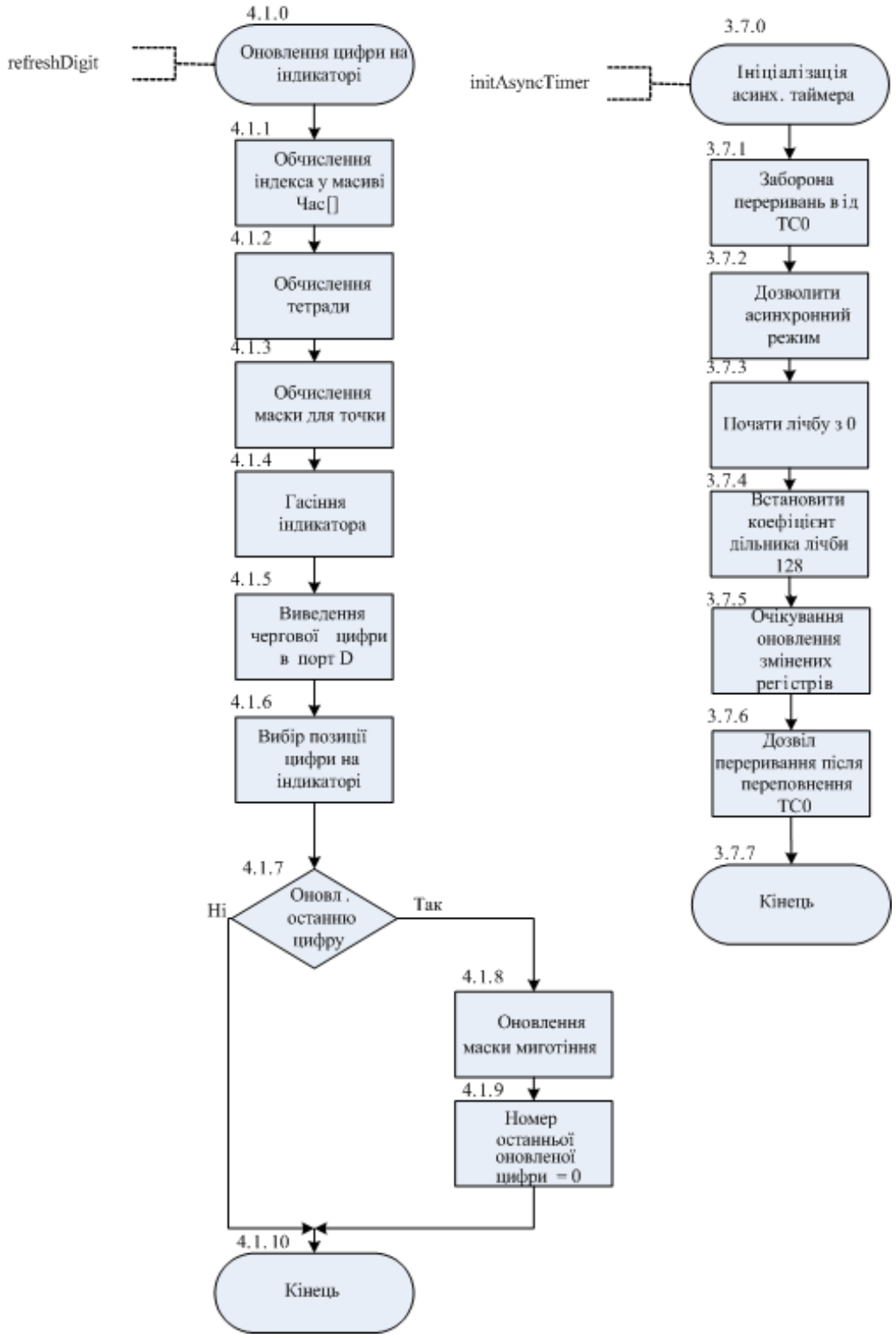


Рис. 9.31. (Закінчення). Див. також с. 301…305

9.6.3. Робоча програма мовою С

Нижче наведено робочу програму мовою С.

```
#include <avr/io.h>                                    //1.0

                                                       //1.1
#include <avr/interrupt.h>


#define TRUE       1
                                                       //1.2
#define FALSE      0

/* Режими роботи годинника. */
#define MODE_WORK        0
#define MODE_EDIT_SECS    1
#define MODE_EDIT_MINS    2
#define MODE_EDIT_HOURS   3

/* Ліміти для годин, хвилин і секунд у BCD*/
#define LIMIT_HOURS      0x24
#define LIMIT_MINUTES    0x60
#define LIMIT_SECONDS    0x60

/* Позиції у масиві, де зберігаються відповідні компоненти поточного часу */
#define I_SECONDS    0
#define I_MINUTES    1
#define I_HOURS      2
/* Кількість цифр на індикаторі*/
#define N_DIGITS     6
/*
* Частота процесора
* UL - беззнаковий Long [0, +4 294 967 295]
*/
#define F_CPU (4000000UL)

/* кількість зсувів праворуч для переміщення старшої тетради у молодшу */
#define HI_NIBBLE_SHIFT       0x04

/* Маска для виділення молодшої тетради */
#define LO_NIBBLE_MASK        0x0F    //0b00001111 == 0x0F
```

```
/* Константа для виконання корекції BCD - числа при досягенні межі 0xXA*/
#define BCD_CORRECTION        0x06

/* Маска для виділення стану піна, до якого підключено кнопку +1*/
#define PLUS_BUTTON_MASK      0x40

/* Затримка для утримання кнопки +1 перед початком автоінкремента*/
#define AUTOINC_DELAY         2

/* Маска для ініціалізації порта E */
#define PE_INIT_MASK          0x3F    //0b00111111 == 0x3F
/*
* Маска для гасіння індикатора, враховано два старших біта,
* біти що використовуються для включення підтяжки ніжок PE6 і PE7
*/
#define INDICATOR_DESELECT_MASK    0xC0    // 0b11000000 == 0xC0

/* Частота оновлення індикатора у Гц. */
#define    REFRESH_RATE    100

/* Коефіцієнт передподільника TC1. */
#define    PRESCALER       64

/* Сегменти індикатора. */
#define A       1     // 0b00000001 == 0x01
#define B       2     // 0b00000010 == 0x02
#define C       4     // 0b00000100 == 0x04
#define D       8     // 0b00001000 == 0x08
#define E       16    // 0b00010000 == 0x10
#define F       32    // 0b00100000 == 0x20
#define G       64    // 0b01000000 == 0x40
#define DP      128   // 0b10000000 == 0x80

/* Таблиця перекодування у "семисегментний" код. */
const unsigned char LED_CODE[] = {                                    //1.3
   A|B|C|D|E|F,    // 0x3F -> '0'
   B|C,            // 0x06 -> '1'
   A|B|D|E|G,      // 0x5B -> '2'
   A|B|C|D|G,      // 0x4F -> '3'
   B|C|F|G,        // 0x66 -> '4'
   A|C|D|F|G,      // 0x6D -> '5'
   A|C|D|E|F|G,    // 0x7D -> '6'
   A|B|C,          // 0x07 -> '7'
   A|B|C|D|E|F|G,  // 0x7F -> '8'
```

```
    A|B|C|D|F|G     // 0x6F -> '9'
};

/* Граничне значення TC1, за якого індикатор (повністю) оновлюється
REFRESH_RATE разів у секунду. */
const unsigned int TIMER_MAX = F_CPU / PRESCALER / REFRESH_RATE /
N_DIGITS;

/*
 * Число оновлень індикатора, після якого відбувається оновлення маски
 * миготіння, простіше – період миготіння.
 */
const unsigned int BLINK_MAX = 150;

/* Лічильник оновлень. */
unsigned char updateCounter = 0;                                        //1.4

/* Маска миготіння. */
unsigned char blinkMask = 0xFF;

/* Нова маска миготіння. */
unsigned char newBlinkMask = 0xFF;

/* Масив для зберігання поточного часу (у BCD). */
unsigned char time[3] = {
    0,    // секунди
    0,     // хвилини
    0    // години
};

/* Остання оновлена цифра. */
unsigned char lastRefreshedDigit = 0;

/* Режим. */
unsigned char mode = MODE_EDIT_HOURS;

/* Лічильник утримання кнопки. */
unsigned char holdCounter = 0;

/*
 * Функція інкремента заданого значення у масиві.
 *     index – індекс у масиві часу
 *     limit – максимальне значення (у BCD).
 * Повертає TRUE, якщо у поточній позиції відбулося переповнення, яке може
```

```
 * означати, що необхідно оновити також інші позиції.
 */
unsigned char incTimePosition(unsigned char index, unsigned char limit)      //2.2.2.0
{                                                                            //5.2.1.0

   if (index < N_DIGITS >> 1) {                                              //2.2.2.1; 5.2.1.1

      time[index]++;                                                         //2.2.2.2; 5.2.1.2

      /*
       * Лічба відбувається в BCD. Якщо досягли значення 0xXA, то необхідно
       * додати 0x06, щоб отримати 0xY0, де Y == X + 1.
       */

      if ((time[index] & LO_NIBBLE_MASK) == 0x0A) {                          //2.2.2.3; 5.2.1.3
         /*
          * Наприклад, 0x39 + 0x01 = 0x3A, а потрібно 0x40, тому
         * 0x3A + 0x06 = 0x40.
         */

         time[index] += BCD_CORRECTION;                                      //2.2.2.4; 5.2.1.4
      }

      /* У випадку досягнення граничного значення лічби -> почати з 0. */
      if (time[index] == limit) {                                            //2.2.2.5; 5.2.1.5

         time[index] = 0;                                                    //2.2.2.6; 5.2.1.6

         return TRUE;
      }
   }

   return FALSE;                                                             //2.2.2.7; 5.2.1.7

}

/*
 * Оновлює цифри на індикаторі.
 */
void refreshDigit()                                                          //4.1.0
{

   /* lastRefreshedDigit / 2 -> індекс у масиві time[]. */
   unsigned char index = lastRefreshedDigit >> 1;                            //4.1.1
```

```
    /*
     * Старша чи молодша тетрада?
     * Якщо старша тетрада, то виконати зсув на 4 біта праворуч,
     * щоб правильно виконати виведення у порт E.
     * За допомогою конструкції (lastRefreshedDigit & 0x01) виконується перевірка
на парність.
     */

    unsigned char shift = (lastRefreshedDigit & 0x01) ? HI_NIBBLE_SHIFT : 0x00;
//4.1.2

    /* Визначення, чи потрібна крапка (у крайній правій позиції – не потрібна). */

    unsigned char point = (shift == 0 && index > 0) ? DP : 0x00;              //4.1.3

    /* Гасіння індикатора. */

    PORTE &= INDICATOR_DESELECT_MASK;                                          //4.1.4

     /* Виведення чергової цифри в порт D. */

    PORTD = LED_CODE[(time[index] >> shift) & LO_NIBBLE_MASK] | point;
//4.1.5

    /* Вибір позиції на індикаторі з урахуванням маски миготіння. */

    PORTE |= (1 << lastRefreshedDigit) & blinkMask;                            //4.1.6

    /* Після оновлення останньої цифри -> почати з початку. */

    if (++lastRefreshedDigit == N_DIGITS) {                                    //4.1.7

        /*
         * Оновлення маски миготіння на останній цифрі, інакше можливе
         * несинхронне миготіння цифр, коли одну із них
         * ще не видно, а інша використовує нову маску і її видно,
         * та навпаки.
         */

        blinkMask = newBlinkMask;                                              //4.1.8

        lastRefreshedDigit = 0;                                                //4.1.9
    }
```

```
}                                                                    //4.1.10

/*
 * Функція інкременту значення годин.
 */
void incHours()                                                      //5.2.3.3.0
{

   incTimePosition(I_HOURS, LIMIT_HOURS);                            //5.2.3.3.1
}

/*
 * Функція інкременту значення хвилин.
 */
void incMinutes()                                                    //5.2.3.0
{

   /* Переповнено значенння хвилин → інкремент годин. */
     if (incTimePosition(I_MINUTES, LIMIT_MINUTES)) {               //5.2.3.1
                                                                    //5.2.3.2

      incHours();                                                   //5.2.3.3
   }
}                                                                   //5.2.3.4


/*
 * Функція інкременту значення секунд.
 */
void incSeconds()
                                                                    //5.2.0
{

   /* Переповнено значенння секунд → інкремент хвилин. */
   if (incTimePosition(I_SECONDS, LIMIT_SECONDS)) {                 //5.2.1 - 5.2.2
      incMinutes();                                                 //5.2.3
   }
}

/*
 * Функція інкременту показань годин у режимі налаштування.
 */
void setupIncTime()                                                  //2.2.0
{
```

```
    if (mode == MODE_EDIT_HOURS) {                                  //2.2.1
        incTimePosition(I_HOURS, LIMIT_HOURS);                      //2.2.2
    } else if (mode == MODE_EDIT_MINS) {                            //2.2.3
        incTimePosition(I_MINUTES, LIMIT_MINUTES);                  //2.2.4
    } else {
        incTimePosition(I_SECONDS, LIMIT_SECONDS);                  //2.2.5
    }
}                                                                   //2.2.6

/*
 * Обробник переривань від асинхронного таймера TC0.
 */
ISR(TIMER0_OVF_vect)                                                //5.0
{
    if (!mode) {                                                    //5.1
        /* Якщо не у режимі налаштування → інкрементувати час. */
        incSeconds();                                               //5.2
    }
}                                                                   //5.3

/*
 * Обробник переривань від T/C1.
 */
ISR(TIMER1_COMPA_vect)                                              //4.0
{

    /* Оновити цифру на індикаторі. */
    refreshDigit();                                                 //4.1

    if (mode == 0) {                                                //4.2
        /* Якщо в рабочому режимі → більше нічого не робити. */
        return;
    }
/* Інкремент лічильника оновлень */
    updateCounter++;                                                //4.3

    if (updateCounter == BLINK_MAX) {                               //4.4
        /*
         * Якщо лічильник оновлень досяг значенння BLINK_MAX ->
         * оновити маску видимості.
         */

        /* Кнопку  "+" натиснуто? */
```

```
        if (PINE & PLUS_BUTTON_MASK) {                          //4.5
            /* Не натиснуто. */

            /*
             * Інвертувати 2 розряди у масці => 00110000b – години,
             * 00001100b – хвилини, 00000011b – секунди.
             */
            newBlinkMask ^= ((3 << ((mode - 1)<<1)));           //4.6

            /* Скинути лічильник утримання кнопки "+1". */
            holdCounter = 0;                                    //4.7
        } else {
            /* Натиснуто. */

            /*
             * Якщо кнопка "+1" утримується 4 періоди миготіння →
             * почати інкремент часу.
             */
            if (holdCounter == AUTOINC_DELAY) {                 //4.8
                setupIncTime();                                 //4.9
            } else {
                holdCounter++;                                  //4.10
            }

            /* У режимі автоінкременту відключити миготіння. */
            newBlinkMask = 0xFF;                                //4.11
        }

        /* Скидання лічильника оновлень. */
        updateCounter = 0;                                      //4.12
    }
}                                                               //4.13

/*
 * Обробник переривань від кнопки "Setup".
 */
ISR(INT7_vect)                                                  //3.0
{

    if (mode == MODE_WORK) {                                    //3.1
        /* Перейти до налаштування значення годин. */
        mode = MODE_EDIT_HOURS;                                 //3.2
    } else {
        /* Перейти до налаштування хвилин або секунд. */
```

```
        mode--;                                                    //3.3
    }



    /*
     * Скинути маску миготіння, щоб минулий налаштований розряд не
     * залишився невидимим.
     */
    newBlinkMask = 0xFF;                                           //3.4
        /* Обнулити лічильник миготіння. */
    updateCounter = 0x00;                                          //3.5
    /*
     * Після завершення налаштування реініціалізуати таймер,
     * щоб перша секунда була повною, інакше можливий варіант,
     * коли після виходу із режиму налаштування значення секунд
     * інкрементується одразу.
     */
    if(mode == MODE_WORK)                                          //3.6
    {
        initAsyncTimer();                                          //3.7
    }
}                                                                  //3.8

/*
 * Обробник переривань від кнопки "+1".
 */
ISR(INT6_vect)                                                     //2.0
{

    if (mode == MODE_WORK) {                                       //2.1
        /* Якщо у робочому режимі → нічого не робити. */
        return;
    }

    /* Інкрементувати показання у поточній позиції. */
    setupIncTime();                                                //2.2
    /* Не блимаємо. */
    newBlinkMask = 0xFF;                                           //2.3
    /* Якщо кнопку натиснуто → обнулити лічильник миготіння. */
    updateCounter = 0x00;                                          //2.4
}                                                                  //2.5
/* Налаштування портів ВВ */
void initPorts()                                                   //1.5.1.2.0
```

```
{

    /* Налаштування порту D на виведення. */
    DDRD  = 0xFF;                                                    //1.5.1.2.1
    PORTD = 0xFF;

    /* Налаштування порту E на виведення, окрім PE6 та PE7. */
    DDRE  = PE_INIT_MASK;                                            //1.5.1.2.2
    PORTE = 0xFF;
}                                                                    //1.5.1.2.3
/* Ініціалізація асинхронного таймера */
void initAsyncTimer()                                               //1.5.1.3.0; 3.7.0
{

    /* Заборона переривань від TC0. */
    TIMSK &= ~((1 << TOIE0) | (1 << OCIE0));                         //1.5.1.3.1; 3.7.1

    /* Дозвіл асинхронного режиму. */
    ASSR |= (1 << AS0);                                             //1.5.1.3.2; 3.7.2

    /* Почати лічбу з нуля. */
    TCNT0 = 0x00;                                                   //1.5.1.3.3; 3.7.3

    /* Встановити коефіцієнт дільника частоти 128. */
    TCCR0 |= (1<<CS00)|(1<<CS02);                                   //1.5.1.3.4; 3.7.4

    /* Очікування оновлення регістрів. */
    while (ASSR & 0x07)                                             //1.5.1.3.5; 3.7.5
        ;

    /* Дозволити переривання після переповнення TC0. */
    TIMSK |= (1 << TOIE0);                                          //1.5.1.3.6; 3.7.6
}                                                                   //1.5.1.3.7; 3.7.7
/* Налаштування зовнішніх переривань для кнопок */
void initExternalInt()                                              //1.5.1.4.0
{

    /* Кнопки пов'язано з перериваннями INT6 та INT7. */
    EIMSK |= (1 << INT7) | (1 << INT6);                             //1.5.1.4.1

    /* Переривання за фронтом. */
    EICRB |= (1<<ISC71)|(1<<ISC70)|(1<<ISC61)|(1<<ISC60);   //1.5.1.4.2
}                                                                   //1.5.1.4.3
/* Налаштування T/C1 */
```

```
void initTimer1()                                                    //1.5.1.5.0
{

    /* WGM11, WGM10 <- 0: режим скидання таймера при співпадінні для T/C1. */
    TCCR1A &= ~(1<<WGM11)|(1<<WGM10);                                //1.5.1.5.1

    /* WGM13 <- 0, WGM12 <- 1, CS <- 3: джерело опорної == CLKio / 64. */
    TCCR1B |= 0x0B;                                                  //1.5.1.5.2

    /* Почати лічбу з нуля. */
    TCNT1 = 0x00;                                                    //1.5.1.5.3

    /* Граничне значення лічби таймера. */
    OCR1A = TIMER_MAX;                                               //1.5.1.5.4

    /* Дозвіл переривання від блоку порівняння A. */
    TIMSK |= (1 << OCIE1A);                                          //1.5.1.5.5
}                                                                    //1.5.1.5.6
/* Ініціалізація периферії */
void init()                                                          //1.5.1.0
{

    /* Вимкнення Watchdog (можливо не знадобиться). */
    WDTCR = 0x10;                                                    //1.5.1.1

    /* Налаштування портів ВВ/ВИВ. */
    initPorts();                                                     //1.5.1.2

    /* Переведення TC0 в асинхронний режим. */
    initAsyncTimer();                                                //1.5.1.3

    /* Налаштування зовнішніх переривань для кнопок. */
    initExternalInt();                                               //1.5.1.4

    /* Налаштування T/C1. */
    initTimer1();                                                    //1.5.1.5
}
/* Головна програма*/
int main()                                                           //1.5.0
{

    /* Ініціалізація периферії */
    init();                                                          //1.5.1
```

```
   /* Дозволити переривання. */
   sei();                                                    //1.5.2

 /* Нічого не робити: усе інше – в обробниках переривань. Нескінченний цикл */
   while (1)                                                 //1.5.3
      ;

   return 0;
}
```

9.6.4. Опис роботи програми

Початок роботи програми

Робота програми починається одразу після скидання мікроконтролера. Обробник переривання скидання за адресою 0x0000 пам'яті програм передає керування процедурі ініціалізації – RESET (блок 1.0 схеми алгоритму). Вона скидає вміст регістра прапорців, заповнює покажчик стека адресою кінця СОЗП, ініціалізує всі змінні їх початковими значеннями, використовуючи оголошені за допомогою .EQU константи. Крім того, виконується заповнення масиву LED_CODE, бітових образів цифр, призначених для виведення на індикатор. Після завершення ініціалізації керування передається процедурі main (блок 1.5.0 схеми алгоритму). Ця процедура виконує ініціалізацію всіх периферійних модулів, які задіяно в роботі, викликаючи процедуру init (блок 1.5.1.0 схеми алгоритму). Init виконує виклики процедур відповідальних за ініціалізацію портів введення/виведення (initPorts – блок 1.5.1.2.0 схеми алгоритму), асинхронного таймера (initAsyncTimer – блок 1.5.1.3.0 схеми алгоритму), зовнішніх переривань (initExternalInt – блок 1.5.1.4.0 схеми алгоритму) та таймера-лічильника 1 (initTimer1 – блок 1.5.1.5.0 схеми алгоритму).

Ініціалізація портів введення-виведення

Порт D налаштовується на виведення, початкове значення 0xFF. Порт E – на виведення, крім двох старших біт, які налаштовуються на введення.

Ініціалізація асинхронного таймера

Таймер-лічильник T0 програмується в асинхронний режим. Початкове значення таймера – нуль. Дільник частоти налаштовується на коефіцієнт 128, що з урахуванням максимального значення 255 і частоти кварцу 32768, дає в результаті період в одну секунду. Після налаштування необхідно очікувати оновлення асинхронних регістрів. Коли ці регістри оновлено, виконується дозвіл переривання від таймера-лічильника T0.

Ініціалізація зовнішніх переривань

Програмується дозвіл зовнішніх переривань INT6, INT7 та режим спрацювання за фронтом.

Ініціалізація таймера/лічильника 1

Для T/C1 програмується режим скидання за збігом. В якості опорної використовується основна частота, яку поділено на 64. Початкове значення встановлюється в нуль, а максимальне налаштовується таким чином, щоб забезпечити виклик переривання 600 раз в секунду, що при 6 розрядах дає швидкість відновлення індикатора в 100 гц. Завершується налаштування дозволом відповідного переривання.

Закінчується процес ініціалізації переходом основного потоку виконання в нескінченний цикл.

Після завершення всіх ініціалізацій керування повертається у процедуру main, де відбувається встановлення прапорця глобального дозволу переривань.

Після завершення процесу ініціалізації годинник знаходиться в режимі налаштування значення годин, про що свідчать миготливі розряди годин.

З цього моменту виконання будь-якої процедури викликається лише за перериваннями.

У програмі передбачені обробники наступних переривань:

– EXT_INT6 (блок 2 схеми алгоритму) – обробник натискання на кнопку «+1» (викликається при її відпусканні). Цей обробник перевіряє режим роботи програми, якщо це не режим налаштування, подальші дії не виконуються і відбувається вихід з обробника. Якщо годинник знаходиться в одному з режимів налаштування, то викликається процедура setupIncTime (блок 2.2.0 схеми алгоритму). Ця процедура перевіряє режим налаштування і в залежності від нього інкрементує відповідне значення (годин, хвилин або секунд). На завершення обробник скидає маску миготіння, що викликає негайне відображення "миготливих" цифр на індикаторі (це значить, що якщо миготливі цифри в даний момент миготіння були невидимі, то вони стануть видимі не чекаючи кінця циклу миготіння);
– EXT_INT7 (блок 3 схеми алгоритму) – обробник натискання на кнопку «Setup» (викликається при її відпусканні). Виконує циклічну зміну режимів: робочий режим -> налаштування годин -> налаштування хвилин -> налаштування секунд -> робочий режим. Виконується скидання маски миготіння, щоб при переході від налаштування одного розряду часу до налаштування іншого, попередній миготливий розряд не залишився невидимим. На завершення процедура перевіряє чи відбувся перехід в

робочий режим, і, якщо це так, то виконується повторна ініціалізація асинхронного таймера. Це виконується для того щоб перша секунда після переходу в робочий режим була повною. Інакше можливо, що перехід в робочий режим відбудеться якраз перед переповненням таймера і час буде меншим, ніж секунда. Такий підхід дозволяє запускати відлік часу переходом в робочий режим, точно знаючи що відлік почнеться з 00 секунд;

– TIM1_COMPA (блок 4 схеми алгоритму) – обробник переривання за переповненням від блоку порівняння A таймера/лічильника 1. Обробник починає свою роботу з виклику процедури refreshDigit (блок 4.1 схеми алгоритму), що виконує оновлення чергової цифри на індикаторі. Далі виконується інкремент і перевірка на досягнення максимального значення лічильника миготіння. Цей лічильник призначено для того, щоб виконувати миготіння певним розрядом індикатора. Максимальне значення лічильника розраховано для досягнення прийнятної частоти миготіння, кратною частоті оновлення. При переповненні лічильника миготіння виконується інверсія бітів в масці миготіння відповідно до режиму роботи. Для прикладу, в режимі налаштування секунд в масці будуть циклічно інвертуватися нульовий та перший біти, приймаючи значення 00 і 11. У процедурі оновлення індикатора refreshDigit виконується операція AND маски миготіння і маски вибору розряду для оновлення. Це дозволяє відключати оновлення певних розрядів, роблячи їх невидимими половину циклу миготіння, і видимими іншу половину циклу. Наприклад, маска вибору розряду для оновлення розраховується циклічним зсувом одиниці на n позицій вліво, і приймає значення від 00000001b до 00100000b для вибору від першого до шостого розряду відповідно. При накладенні на цю маску маски миготіння виходить (для прикладу маска миготіння 11111100b – розряд секунд приховано): 00000001b AND 11111100b = 00000000b. Аналогічно і для маски вибору 00000010b, та на інші маски вибору, маска миготіння не впливає. Якщо обробник виявляє переповнення лічильника миготіння він формує нову маску миготіння, і зберігає її у змінній newBlinkMask, яка використовується в процедурі refreshDigit для виконання описаної вище операції. Маска не змінюється безпосередньо в обробнику переривання, оскільки це може викликати несинхронне миготіння розрядів на індикаторі, коли один з розрядів використовує стару маску та його бачимо, а інший нову та він невидимий, і навпаки. Маска оновлюється тільки після оновлення всього вмісту індикатора. Крім операцій з маскою обробник також виконує облік часу утримання кнопки «+1» і при перевищенні певного ліміту виконує автоматичний інкремент часу;

– TIM0_OVF (блок 5 схеми алгоритму) – обробник переривання за переповненням асинхронного таймера. Виконує перевірку чи знаходиться

годинник в режимі роботи. Якщо це так, виконує інкремент часу на одну секунду. Це робиться шляхом виклику процедури інкременту секунд (incSeconds блок 5.2.0 схеми алгоритму), яка при переповненні значення секунд (60) викличе процедуру інкременту хвилин (incMinutes блок 5.2.3.0 схеми алгоритму), яка в свою чергу при переповненні (60) викличе інкремент годин (incHours блок 5.2.3.3.0 схеми алгоритму). Всі ці процедури використовують процедуру incTimePosition (блок 2.2.2.0 схеми алгоритму), передаючи в неї відповідне значення ліміту і індекс в масиві значення часу, за який необхідно провести інкремент.

Контрольні запитання та завдання

1. Для чого може використовуватись таймер/лічильник?
2. Які регістри відповідають за дозвіл/заборону переривань від таймерів/лічильників?
3. Яким чином задається режим роботи таймера/лічильника T0 (T2)?
4. Чим відрізняється режим роботи CTC від режиму Normal?
5. Для рішення яких завдань краще використання режиму Phase Correct PWM ніж режиму Fast PWM?
6. Які таймери/лічильники можуть працювати в асинхронному режимі?
7. Що може виступати в якості задавача частоти при роботі таймера/лічильника в асинхронному режимі?
8. Для чого можуть використовуватись 16-розрядні таймери/лічильники?
9. При виникненні яких подій таймери/лічильники T1, T3, T4, T5 можуть генерувати переривання?
10. За допомогою яких регістрів виконується керування таймером/лічильником?
11. Які сигнали можуть використовуватись в якості тактового сигналу f_{clkn} для таймерів/лічильників T1, T3, T4 і T5?
12. Яким чином здійснюється вибір джерела тактового сигналу, а також запуск і зупинка таймерів/лічильників?
13. Як увімкнути режим роботи Normal таймера/лічильника T3? Для чого може використовуватись даний режим роботи?
14. Що таке роздільна здатність лічильника таймера?
15. У чому полягає різниця між роботою 8-розрядних та 16-розрядних таймерів/лічильників у режимі Fast PWM?
16. Завдяки чому в режимі Phase and Frequency Correct PWM кожен період сигналу є повністю симетричним?
17. У чому полягає функція вартового таймера?
18. Чим відрізняється розширений вартовий таймер від звичайного?

10. ПРОЕКТУВАННЯ МОДУЛЯ АНАЛОГО-ЦИФРОВОГО ПЕРЕТВОРЮВАЧА

10.1. Загальні відомості

До складу більшості моделей AVR-мікроконтролерів входить модуль 10-розрядного аналого-цифрового перетворювача (АЦП) послідовного наближення [9]. Основні параметри модуля [1; 7]:

– абсолютна похибка: ±2 молодшого значущого розряду (МЗР);

– інтегральна нелінійність: ±0.5 МЗР;

– швидкодія: до 15 тис. вибірок/с.

В зарубіжній літературі МЗР позначається як LSB (Least Significant Bit – молодший значущий розряд). Аналогічно, MSB (Most Significant Bit – старший значущий розряд) відповідає позначенню СЗР.

На вході модуля є 8/16-канальний аналоговий мультиплексор, що визначає кількість аналогових каналів перетворення. Деякі мікроконтролери, наприклад ATmega8x, мають тільки 6 каналів перетворення, два з яких – ADC4 та ADC5 є 8-розрядними.

У більшості моделей входи АЦП можуть також об`єднуватися попарно для формування каналів з диференціальним входом.

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

Як джерело опорної напруги для АЦП може використовуватись напруга живлення мікроконтролера та внутрішнє або зовнішнє джерело опорної напруги.

Модуль АЦП може функціонувати у двох режимах:

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

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

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

10.2. Функціонування модуля

На рис. 10.1 наведено функціональну схему модуля АЦП. В деяких моделях мікроконтролерів елементи і пов'язані з ними сигнали, які виділено на цьому рисунку сірим кольором, відсутні, а не інвертуючий вхід компаратора з пристроєм

вибірки-зберігання підключено безпосередньо до виходу мультиплексора (показано пунктирною лінією).

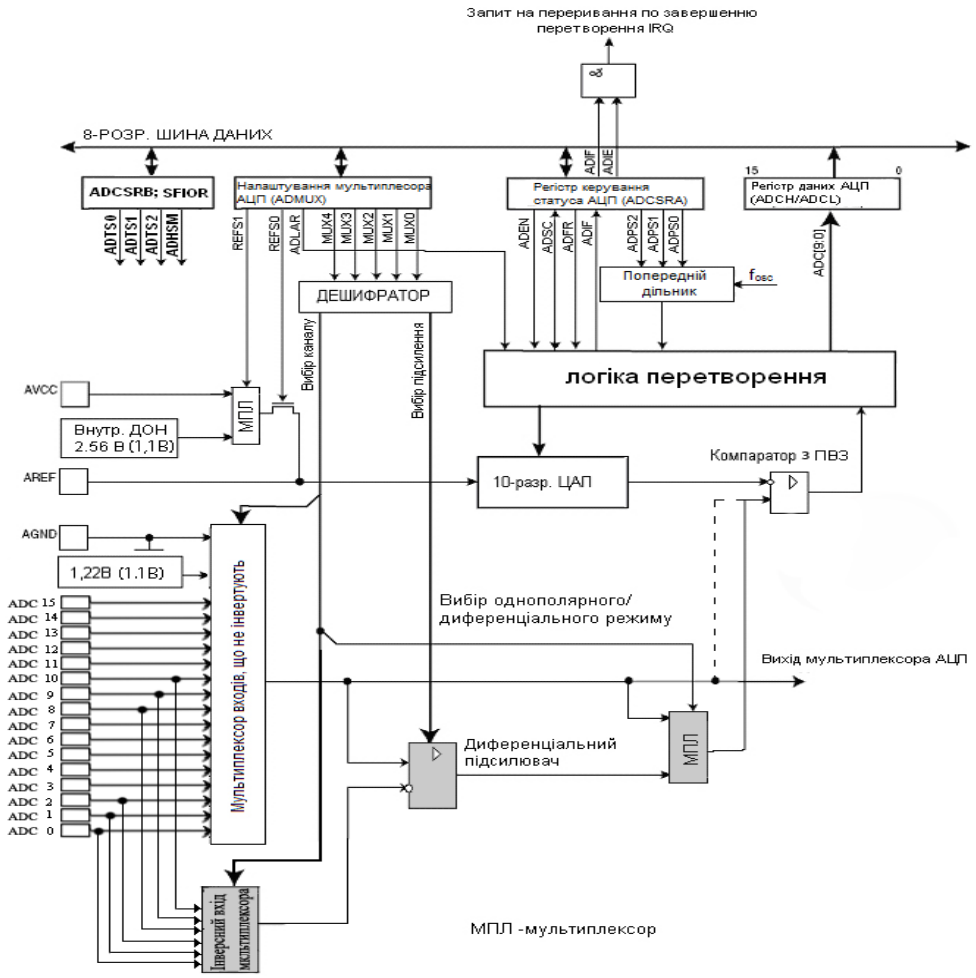


Рис. 10.1. Функціональна схема модуля аналого-цифрового перетворювача

Вхідний аналоговий сигнал перед перетворюванням зберігається у ПВЗ. АЦП працює за принципом послідовного наближення [9]. В кінці перетворення у регістрі даних АЦП (ADCH та ADCL) буде десятирозрядний двійковий код, який еквівалентний вхідній напрузі.

АЦП може обробляти вхідні сигнали у двох режимах: однополярному та диференціальному. При однополярному режимі вхідний сигнал поступає на один із входів ADC0…ADC15. При диференціальному режимі використовуються шість входів мультиплексора (ADC0, ADC1, ADC2, ADC8, ADC9, ADC10). Різниця сигналів між цими входами підсилюється за допомогою диференціального підсилювача і поступає на вхід компаратора.

10.3. Програмування модуля

В табл. 10.1 наведено адреси регістрів, що використовуються для керування модулем АЦП деяких моделей сімейства. Формати регістрів ADCSRA (ADCSR) і ADMUX наведено на рис. 10.2 та 10.3, а опис функцій їх розрядів – у табл. 10.2 і 10.3 відповідно.

Таблиця 10.1. **Адреси регістрів керування модулем АЦП**

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

	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADFR	ADIF	ADIÉ	ADPS2	ADPS1	ADPSO	ATmega8x ATmega128x
Зчитування(R)/Запис(W)	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Початкове значення	0	0	0	0	0	0	0	0	
	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIÉ	ADPS2	ADPS1	ADPSO	Інші моделі
Зчитування(R)/Запис(W)	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Початкове значення	0	0	0	0	0	0	0	0	

ATmega 8x – ADCSR
Інші моделі – ADCSRA

Рис. 10.2. Формат регістра ADCSRA (ADCSR)

	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADFR	ADIF	ADIÉ	ADPS2	ADPS1	ADPSO	ATmega8x ATmega128x
	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ATmega8x ATmega48x/88x/168x
Зчитування(R)/Запис(W) Початкове значення	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	
	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	Інші моделі
Зчитування(R)/Запис(W) Початкове значення	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	

Рис. 10.3. Формат регістра ADMUX

Таблиця 10.2. **Опис розрядів регістра ADCSRA (ADCSR*)**

Розряд	Назва	Опис
7	ADEN	Дозвіл АЦП (1 – увімкнено, 0 – вимкнено)
6	ADSC	Запуск перетворення (1 – почати перетворення)
5	ADATE (ADFR *)	Вибір режиму роботи АЦП (0 – одиночне, 1 – безперервне перетворення)
4	ADIF	Прапорець завершення АЦП
3	ADIE	Дозвіл переривання від завершення АЦП
2...0	ADPS2:ADPS0	Вибір частоти перетворення (див. таблицю 10.6)
* В моделях ATmega8x, ATmega128x		

Таблиця 10.3. **Опис розрядів регістра ADMUX**

Розряд	Назва	Опис	Модель
7...6	REFS1:REFS0	Вибір джерела опорної напруги (див. таблицю 10.5)	Всі моделі
5	ADLAR	Ліве вирівнювання результату (див. таблицю 10.11)	Всі моделі
4	–	Зарезервовано	ATmega8x
	MUX4	Вибір вхідних каналів (див. таблицю 10.9)	Всі моделі крім ATmega8x
3...0	MUX3...MUX0	Вибір вхідних каналів і частоти перетворення (див. таблиці 10.8, 10.9)	Всі моделі

Формат регістрів ADCSRB і SFIOR наведено рис. 10.4 і 10.5 відповідно (не використовувані розряди регістра SFIOR позначено як «–»).

	7	6	5	4	3	2	1	0	
	—	—	—	—	—	ADTS2	ADTS1	ADTSO	ATmega64x
Зчитування(R)/Запис(W)	R	R	R	R	R	R/W	R/W	R/W	
Початкове значення	0	0	0	0	0	0	0	0	

	7	6	5	4	3	2	1	0	
	—	ACME	—	—	—	ADTS2	ADTS1	ADTSO	ATmega48x/88x/168x ATmega164x/324x/644x ATmega165x/325x/3250x ATmega645x/6450x ATmega1281x/2561x
Зчитування(R)/Запис(W)	R	R/W	R	R	R	R/W	R/W	R/W	
Початкове значення	0	0	0	0	0	0	0	0	

	7	6	5	4	3	2	1	0	
	—	ACME	—	—	MUX5	ADTS2	ADTS1	ADTSO	ATmega640x/1280x/2560x
Зчитування(R)/Запис(W)	R	R/W	R	R	R/W	R/W	R/W	R/W	
Початкове значення	0	0	0	0	0	0	0	0	

Рис. 10.4. Формат регістра ADCSRB

	7	6	5	4	3	2	1	0	
	ADTS2	ADTS1	ADTSO	—	X	X	X	X	ATmega8535x ATmega16x/32x
Зчитування(R)/Запис(W)	R	R	R	R	R	R/W	R/W	R/W	
Початкове значення	0	0	0	0	0	0	0	0	

Рис. 10.5. Формат регістра SFIOR

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

У більшості моделей модуль АЦП може використовуватись як мультиплексор аналогових сигналів для аналогового компаратора. Для цього необхідно встановити розряд ACME (Analog Comparator Multiplexer Enable) в одиницю (див. розділ 12). Інверсний вхід аналогового компаратора може бути з’єднано, або з його входом AIN1, або з будь-яким із входів АЦП в залежності від значення розряду ADEN (одиниця – AIN1, нуль – один із входів АЦП).

В моделях ATmega8x та ATmega128x режим роботи АЦП визначається станом розряду ADFR (табл. 10.2, див. вище). Якщо його встановлено в одиницю, АЦП працює в режимі безперервного перетворення. У цьому режимі запуск кожного наступного перетворення здійснюється автоматично після закінчення поточного перетворення.

Якщо розряд ADFR скинуто в нуль, АЦП працює в режимі одиночного перетворення та запуск кожного перетворення здійснюється командою користувача.

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

Якщо розряд ADATE скинуто в нуль, АЦП працює в режимі одиночного перетворення. Якщо розряд ADATE встановлено в одиницю, функціонування АЦП визначається вмістом розрядів ADTS2...0 відповідно до табл. 10.4.

Таблиця 10.4. **Джерело сигналу для запуску перетворення**

ADTS2	ADTS1	ADTS0	Джерело сигналу запуску АЦП
0	0	0	Режим безперервного перетворення
0	0	1	Переривання від аналогового компаратора
0	1	0	Зовнішнє переривання INT0
0	1	1	Переривання за подією "Збіг A" таймера/лічильника T0
1	0	0	Переривання за переповненням таймера/лічильника T0
1	0	1	Переривання за подією "Збіг B" таймера/лічильника T1
1	1	0	Переривання за переповненням таймера/лічильника T1
1	1	1	Переривання за подією "Захоплення" таймера/лічильника T1

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

Запуск перетворення за перериванням здійснюється при встановленні в одиницю прапорця обраного переривання. Розряд ADSC регістра ADCSRA при цьому апаратно встановлюється в одиницю. Запуск перетворення у цих режимах також може бути здійснено встановленням в одиницю розряду ADSC регістра ADCSRA.

Для модуля можуть використовуватись різні джерела опорної напруги (ДОН). Вибір конкретного ДОН здійснюється за допомогою розрядів REFS1:REFS0 регістра ADMUX (табл. 10.5).

Згідно з табл. 10.5, до виводу AREF мікроконтролера може бути підключено внутрішнє ДОН. У цьому випадку для підвищення завадозахищеності до виводу AREF треба підключити зовнішній фільтруючий конденсатор.

Таблиця 10.5. **Вибір джерела опорної напруги**

REFS1	REFS0	Джерело опорної напруги[1)]	Модель
0	0	До виводу AREF підключено зовнішнє ДОН; внутрішнє ДОН відключено	Всі моделі
0	1	Напруга живлення AVcc[2)]	Всі моделі
1	0	Внутрішнє ДОН напругою 1.1 В[2)]	ATmega164x/324x/644x ATmega640x/1280x/1281x ATmega2560x/2561x
		Зарезервовано	Решта моделей
1	1	Внутрішнє ДОН напругою 2.56 В[2]	ATmega48x/88x/168x ATmega165/325x/3250x ATmega645x/6450x
		Внутрішнє ДОН напругою 1.1 В[2)]	Решта моделей
[1)] При роботі з підсиленням 10x чи 200x в якості внутрішнього ДОН можна використовувати тільки ДОН, яке підключено при REFS1:0 = 11. [2)] Якщо до виводу AREF підключено зовнішнє джерело напруги, дані варіанти використовуватись не можуть.			

Канал однополярного або диференціального аналогового введення та каскад диференціального підсилення обираються шляхом програмування розрядів MUXn (n = 0, 1…5) у регістрі ADMUX (табл. 10.9, 10.10). В якості однополярного аналогового входу АЦП в залежності від моделі мікроконтролера може бути обрано один із входів ADC0...ADC15, внутрішнє джерело опорної напруги 1,22 (1,1) В, або 0В (GND). В режимі диференціального введення є можливість вибору входів диференціального підсилювача, що інвертують або не інвертують.

Якщо обрано диференціальний режим введення, то підсилювач буде помножувати різницю напруг між обраною парою входів на заданий коефіцієнт підсилення. Якщо обрано однополярний режим введення, то каскад підсилення не використовується.

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

АЦП генерує 10-розрядний результат, що міститься в парі регістрів даних АЦП: ADCH і ADCL. У початковому стані результат перетворення розміщується в молодших 10-ти розрядах 16-розрядного слова (вирівнювання вправо), але може бути розміщено у старших 10-ти розрядах (вирівнювання вліво) шляхом встановлення в одиницю розряду ADLAR у регістрі ADMUX (табл. 10.3, див. вище, та табл. 10.11, див. нижче).

Коли досить точності 8-розрядного значення, то рекомедовано подання результату перетворювання з вирівнюванням вліво. В цьому випадку необхідно читати тільки регістр ADCH. В іншому ж випадку необхідно першим читати вміст регістра ADCL, а потім ADCH, що гарантує, що обидва байти є результатом того самого перетворення. Як тільки виконано читання ADCL блокується доступ до регістрів даних з боку АЦП. Це означає, що якщо зчитано ADCL і перетворення завершується перед читанням регістра ADCH, то жоден з регістрів не може модифікуватися та результат перетворення губиться. Після читання ADCH доступ до регістрів ADCH і ADCL знову дозволяється.

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

Одиночне перетворення запускається шляхом запису одиниці у розряд запуску перетворення АЦП – ADSC. Даний розряд залишається в одиничному стані у процесі перетворення та скидається за завершенням перетворення. Якщо в процесі перетворення перемикається канал аналогового введення, то перш ніж це зробити, АЦП завершить поточне перетворення.

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

10.4. Формування тактового сигналу

Для формування тактового сигналу модуль АЦП має попередній дільник, який формує похідні частоти відносно частоти синхронізації мікроконтролера (рис. 10.5).

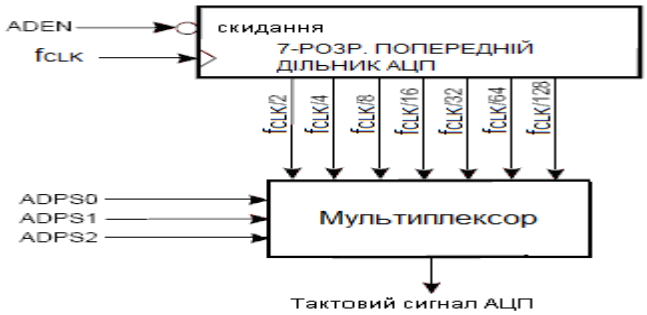


Рис. 10.5. Схема попереднього дільника АЦП

Коефіцієнт ділення встановлюється за допомогою розрядів ADPS2; ADPS1 та ADPS0 у регістрі ADCSRA (табл. 10.6).

Таблиця 10.6. **Задання коефіцієнта ділення попереднього дільника АЦП**

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

З моменту включення АЦП попередній дільник починає лічбу та працює доки розряд ADEN = 1. Якщо потрібно забезпечити максимальну роздільну здатність (10 розрядів), то частота на вході схеми послідовного наближення повинна бути в діапазоні 50...200 кГц [1; 7]. Якщо достатньо точності менше 10 розрядів, то частота на вході АЦП може бути більше 200 кГц.

10.5. Часові діаграми роботи

Часові діаграми роботи АЦП в різних режимах наведено на рис. 10.6…10.9.

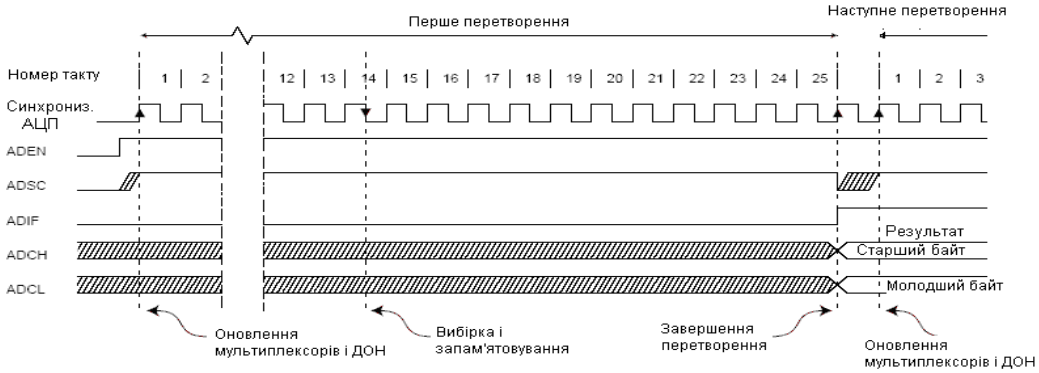


Рис. 10.6. Часові діаграми роботи АЦП при першому одиночному перетворенні в режимі одиночного перетворення

Якщо встановленням розряду ADSC у регістрі ADCSRA ініціюється перше одиночне однополярне перетворення, то воно починається з наступного наростаючого фронту тактового (синхро) сигналу АЦП (рис. 10.6).

Нове одиночне перетворення може бути запущено відразу ж після скидання розряду ADSC (до збереження результату поточного перетворення). Однак реально цикл перетворення почнеться не раніше ніж через один такт після закінчення поточного перетворення.

Рис. 10.7. Часові діаграми роботи АЦП в режимі наступного одиночного перетворення

Рис. 10.8. Часові діаграми роботи АЦП в режимі запуску за перериванням

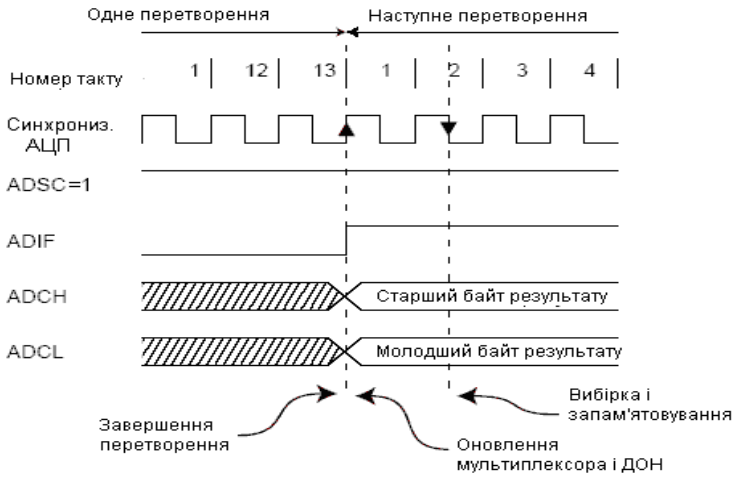


Рис. 10.9. Часові діаграми роботи АЦП в режимі автоматичного перезапуску для однополярного перетворення

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

Наступне одиночне перетворення вимагає 13 тактів синхронізації АЦП. Після його початку на вибірку-зберігання затрачується 1,5 такти синхронізації, а після початку першого перетворення на вибірку-зберігання затрачується 13,5 тактів. За завершенням перетворення результат зберігається в регістрах даних АЦП і встановлюється прапорець ADIF. В режимі одиночного перетворення одночасно скидається розряд ADSC. Розряд ADSC може бути знову встановлено програмно і нове перетворення буде ініційовано першим наростаючим фронтом тактового сигналу АЦП.

В режимі безперервного перетворення (автоматичного перезапуску) нове перетворення починається одразу за завершенням попереднього. При цьому біт ADSC залишається у високому стані. Час перетворення для різних режимів представлено в табл. 10.7.

Таблиця 10.7. **Час перетворення АЦП**

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

Якщо використовується запуск за перериванням, то цикл перетворення починається за першим наростаючим фронтом тактового сигналу після встановлення прапорця обраного переривання. При встановленні цього прапорця здійснюється скидання попереднього дільника модуля АЦП, що забезпечує фіксовану затримку між генерацією запиту на переривання та початком циклу перетворення. Перетворення запускається при встановленні відповідного прапорця, тобто навіть тоді, коли перетворення встановленням прапорця ADSCA (ADSC) не ініціалізувалося.

Прапорець ADIF, як і прапорці інших переривань, скидається апаратно при запуску підпрограми обробки переривання від АЦП або програмно, записом у нього одиниці. Дозвіл переривання здійснюється встановленням в одиницю розряду ADIE регістра ADCSR при встановленому прапорці I регістра SREG.

Диференціальне перетворення синхронізується з внутрішнім генератором тактових імпульсів CKADC2, частота якого дорівнює половині частоти синхронізуючого пристрою АЦП. Це виконується автоматично таким чином, щоб вибірка-зберігання відбувалась у певній фазі CKADC2.

Диференціальне перетворення, що ініційовано користувачем, коли CKADC2 має низький рівень, виконується за той же час, що й одиночне перетворення – 13 тактів. Диференціальне перетворення, що ініційовано користувачем у той час, коли CKADC2 мало високий рівень, через специфіку механізму синхронізації буде виконано за 14 тактів.

Відповідно при диференціальному перетворенні на вибірку-зберігання затрачується 1,5 або 2,5 такти.

Використання диференціальних вхідних каналів має деякі особливості.

У режимі автоматичного перезапуску нове перетворення ініціюється відразу за завершенням попереднього. Оскільки в цей момент CKADC2 дорівнює одиниці, всі перетворення, що були автоматично перезапущені (тобто всі, крім першого), будуть тривати 14 тактів синхронізації АЦП.

Про використання режиму запуску за перериванням одночасно з диференціальними каналами між перетвореннями необхідно вимикати АЦП, щоб уникнути некоректних вимірювань, пов’язаних із невизначеністю моменту скидання попереднього дільника АЦП. В результаті вимикання і вмикання АЦП між перетвореннями виконуватимуться лише «довгі» перетворення, результати яких завжди будуть коректними.

Підсилювальний каскад оптимізовано під частотний діапазон до 4 кГц для будь-яких коефіцієнтів підсилення. Підсилення сигналів більш високих частот буде нелінійним [1; 7]. Якщо вхідний сигнал містить частотні складові вище частотного діапазону підсилювального каскаду, необхідно встановити зовнішній фільтр низьких частот.

10.6. Керування вхідним мультиплексором

Виводи мікроконтролера, які підключено до входу АЦП, визначаються станом розрядів MUX3...MUX0; MUX4...MUX0 регістра ADMUX та регістра ADCSRB відповідно до табл. 10.8, 10.9.

Частина мікроконтролерів Mega мають шістнадцять однополярних входів [1; 7]. В цьому випадку для керування вхідним мультиплексором використовуються шість керуючих розрядів: MUX5...MUX0.

Для каналів з диференціальним входом зазначені розряди визначають також коефіцієнт попереднього підсилення вхідного сигналу. Попередній підсилювач, має вбудовану схему корекції зсуву. Величину зсуву, що залишилася після

корекції, може бути враховано програмним способом. Для цього треба обидва входи диференціального підсилювача підключити до того самого вивода мікроконтролера (табл. 10.9), а потім віднімати отримане значення з результату наступних перетворень. Таким чином, помилку зсуву можна знизити до величини, меншої ніж одиниця молодшого значущого розряду.

Таблиця 10.8. **Керування вхідним мультиплексором у моделях ATmega8x, ATmega48x/88x/168x**

MUX3...MUX0	Однополярний (несиметричний) вхід
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6[1]
0111	ADC7[1]
1000...1101	Зарезервовано
1110	1.22 В (1,1 В[2])
1111	0В (GND)
[1] Є тільки в корпусах TQFP-32 і MLF-32. [2] В моделях ATmega48x/88x/168x	

Програмування розрядів MUXn (n = 0, 1, …, 5) та REFS1:0 у регістрах ADMUX та ADCSRB підтримується буферизацією через тимчасовий регістр. Цим гарантується, що нові налаштування каналу перетворення та опорного джерела набудуть чинності в безпечний момент для перетворення.

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

Особливі заходи необхідно вжити при зміні диференціального каналу. Як тільки здійснено вибір диференціального каналу підсилювальному каскаду потрібно 125 мкс для стабілізації нового значення. Отже, протягом перших 125 мкс після перемикання диференціального каналу перетворення не повинно стартувати. Якщо ж у цей період перетворення все-таки виконувалися, то їхній результат необхідно ігнорувати.

Таблиця 10.9. **Керування вхідним мультиплексором у моделях ATmega16x/164x/32x/8535x/64x/128x/164x/165x/325x/3250x/645x/6450x/ 1281x/2561x**

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

[1] У моделях ATmega165x/325x/3250x/645x/6450x/1251x/2561x

Таку ж затримку на встановлення необхідно ввести при першому диференціальному перетворенні після зміни опорного джерела АЦП за рахунок зміни розрядів REFS1:0 в ADMUX.

Якщо дозволено роботу інтерфейсу JTAG, то функції каналів АЦП на виводах портів, що використовуються для модуля JTAG, відміняються [7].

У деяких мікроконтролерів є можливість відключення вхідних цифрових буферів на виводах ADC0...ADC15 у випадку, якщо відповідні виводи використаються тільки для зчитування аналогових сигналів [7]. При відключених цифрових буферах зменшується загальний струм споживання мікроконтролера, а відповідні розряди регістрів PINX завжди читаються як нуль.

Відключення цифрових буферів на входах ADC0...ADC7 здійснюється записом одиниці у розряди ADC0D...ADC7D регістра DIDR0, який розташовано за адресою ($7E), а відключення буферів на входах ADC8...ADC15 здійснюється записом одиниці у розряди ADC8D...ADC15D регістра DIDR2, розташованого за адресою ($7D) (рис. 10.10).

	7	6	5	4	3	2	1	0	
	ADC7D	ADC6D	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D	ATmega48x/88x/168x ATmega165x ATmega325x/3250x ATmega645x/6450x ATmega640x/1280x/1281x ATmega2560x/2561x
Зчитування(R)/Запис(W)	R/W	R/W	R/W	R/W		R/W	R/W	R/W	
Початкове значення	0	0	0	0		0	0	0	

	7	6	c	4	3	2	1	0	
	ADC15D	ADC14D	ADC13D	ADC12D	ADC11D	ADC10D	ADC09D	ADC08D	ATmega640x/1280x/2560x
Зчитування(R)/Запис(W)	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Початкове значення	0	0	0	0	0	0	0	0	

Рис. 10.10. Формат регістрів DIR0 (а) та DIR2 (б)

10.7. Збереження результату перетворення

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

У початковому стані результат перетворення вирівнюється вправо (старші 6 розрядів регістра ADCH – не є значущими). Однак він може вирівнюватися також вліво (молодші 6 розрядів регістра ADCL будуть не значущими).

Для керування вирівнюванням результату перетворення призначено розряд ADLAR регістра ADMUX. Якщо цей розряд встановлено в одиницю, результат перетворення вирівнюється за лівою границею 16-розрядного слова, якщо скинутий в нуль – за правою границею.

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

У табл. 10.10 наведено приклади вирівнювання результату вліво та вправо.

Звернення до регістрів ADCH і ADCL для отримання результату перетворення повинно виконуватися в певній послідовності: спочатку необхідно прочитати регістр ADCL, а потім ADCH. Ця вимога пов'язана з тим, що після звернення до регістра ADCL процесор блокує доступ до регістрів даних з боку АЦП доти, поки не буде прочитано регістр ADCH. Завдяки цьому можна бути впевненим, що при читанні регістрів ADCH, ADCL у них будуть перебувати складові того самого результату. Відповідно, якщо чергове перетворення

завершиться до звернення до регістра ADCH, результат перетворення буде загублено. З іншого боку, якщо результат перетворення вирівнюється вліво й досить точності 8-розрядного значення, для отримання результату можна прочитати тільки вміст регістра ADCH.

Таблиця 10.10. **Вирівнювання результату АЦП**

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

10.8. Рекомендації з перемикання вхідних каналів

При перемиканні вхідних каналів необхідно врахувати деякі рекомендації.

В режимі одиночного (несиметричного) перетворення програмування перемикання каналів необхідно виконувати перед початком перетворення. Перемикання каналів відбувається протягом одного такту синхронізації АЦП після запису одиниці в регістр ADSC. Тому перед вибором нового каналу треба дочекатися завершення перетворення.

В режимі автоматичного перезапуску канал необхідно вибирати перед початком першого перетворення. Перемикання каналів відбувається також протягом одного такту синхронізації АЦП після запису одиниці в регістр ADSC.

При перемиканні на диференціальний канал перше перетворення буде характеризуватися поганою точністю через перехідний процес у схемі автоматичного регулювання зсуву. Отже, перший результат такого перетворення рекомендується ігнорувати.

10.9. Особливості підключення джерела опорної напруги

Джерело опорної напруги $U_{ДОН}$ АЦП визначає діапазон перетворення АЦП. Якщо рівень однополярного сигналу понад $U_{ДОН}$, то результатом перетворення буде 0x3FF. В якості $U_{ДОН}$ можуть виступати AVcc, внутрішнє ДОН 2,56 В (1,1 В)

або зовнішнє ДОН, що підключено до виводу AREF. AVcc підключається до АЦП через пасивний ключ. Внутрішня опорна напруга 2,56 В (1,1 В) генерується внутрішнім еталонним джерелом VBG, що буферизовано внутрішнім підсилювачем. Зовнішній вивід AREF зв'язано безпосередньо з АЦП. Підключенням конденсатора між виводом AREF і спільним виводом можна знизити вплив шумів на опорне джерело [1; 7]. Напругу $U_{ДОН}$ на виводі AREF можна виміряти вольтметром з високим вхідним опором. Сигнал $U_{ДОН}$ є високоомним джерелом, тому зовні до нього можна підключити тільки ємнісне навантаження.

Якщо користувач використовує зовнішнє опорне джерело, що підключено до виводу AREF, то не допускається одночасно використання іншого опорного джерела, тому що це приведе до шунтування зовнішньої опорної напруги. Якщо до виводу AREF не прикладено зовнішню напругу, то користувач може обрати, як опорне джерело, AVcc або 2,56 В (1,1 В). Результат першого перетворення після перемикання опорного джерела може характеризуватися низькою точністю, тому користувачеві рекомендується його ігнорувати.

10.10. Результат перетворення

Для каналів з однополярним входом результат перетворення визначається виразом:

$$ADC = \frac{1023 * U_{IN}}{U_{REF}}, \qquad (10.1)$$

де U_{IN} – значення вхідної напруги в мілівольтах, а U_{REF} – величина опорної напруги в мілівольтах.

Коефіцієнт передачі АЦП визначається формулою:

$$K_{ПЕР} = 1023/U_{REF}\,[МЗР/мВ]. \qquad (10.2)$$

На рис. 10.11 представлено функцію перетворення АЦП в однополярному режимі.

Код 0x000 відповідає рівню аналогової землі, а 0x3FF – рівню напруги ДОН мінус один крок квантування за напругою.

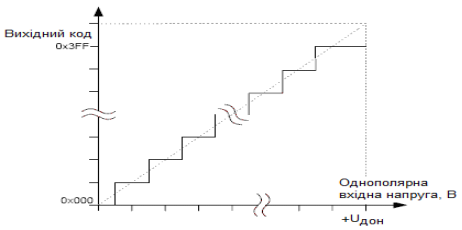


Рис. 10.11. Функція перетворення АЦП при зміні однополярного сигналу

Зв'язок між вхідним сигналом та вихідними кодами для однополярного режиму відображає табл. 10.11.

Таблиця 10.11. **Зв'язок між вхідним і вихідним кодами**

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

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

Приклад. Нехай ADMUX = 0x00…0x07 (будь-який однополярний вхід). Напруга на цьому вході дорівнює 1000 мВ. $U_{ДОН}$ = 2,56 В. Тоді:

Код АЦП = 1023 * 1000 / 2560 = 400 = 0x190.

Для каналів з диференціальним входом результат перетворення визначається виразом:

$$ADC = 512 \cdot K_{П} \cdot \frac{U_{POS} - U_{NEG}}{U_{REF}}, \qquad (10.3)$$

де U_{P0S} – величина напруги на додатному (не інвертуючому) вході, U_{NEG} – величина напруги на від'ємному (інвертуючому) вході, а Кп – коефіцієнт підсилення.

В цьому випадку результат перетворення представляється у додатковому коді, а його значення лежить в діапазоні: $200 (-512)...$1FF (+511).

Приклад. Нехай ADMUX = 0xED (пари входів ADC3 – ADC2, Кп = 10, $U_{дон}$ = 2.56В, результат з лівостороннім вирівнюванням). Напруга на вході ADC3 = 300 мВ, а на вході ADC2 = 500 мВ. Тоді:

$$КОД_{АЦП} = 512 \cdot 10 \cdot \frac{300 - 500}{2560} = -400 = 0x270.$$

З урахуванням обраного формату розміщення результату (лівосторонній) ADCL = 0x00, а ADCH = 0x9C. Якщо ж обрано правосторонній формат (ADLAR = 0), то ADCL = 0x70, ADCH = 0x02.

За завершенням перетворення (ADIF = 1) результат може бути зчитано з пари регістрів результату перетворення АЦП (ADCL, ADCH). При необхідності

визначити полярність результату треба опитати старший розряд результату перетворення (ADC9 в ADCH). Якщо даний розряд дорівнює одиниці, то результат від'ємний, якщо ж нуль, то додатний.

10.11. Підвищення точності перетворення

Рекомендації, що дозволяють підвищити точність перетворення та найбільшою мірою використати можливості АЦП, наведено у [1; 7].

10.12. Відносна похибка

Однією з основних похибок АЦП є похибка від квантування за рівнем, яка виникає через ділення (квантування) вхідної напруги на постійні інтервали ΔU [9]. Зміні вхідної напруги на ±ΔU відповідає зміна вихідного двійкового коду на ±1 молодшого значущого розряду (МЗР).

Абсолютна похибка квантування за рівнем дорівнює ±ΔU/2 або ±0,5 МЗР.

Відносна похибка квантування за рівнем

$$\delta_{ВІДН} = \frac{50}{N_Д - 1}\%,$$

де $N_Д$ – число дискретних значень вихідного коду, включаючи нульове.

У свою чергу

$$N_Д = 2^{N_{РДК}},$$

де $N_{РДК}$ – число розрядів вихідного двійкового коду.

Для нашого АЦП $N_{РДК}$ = 10, $N_Д$ = 2^{10} = 1024.

Тоді

$$\delta_{ВІДН} = \frac{50}{1024-1}\% = \frac{50}{1023}\% = 0{,}05\ \%.$$

10.13. Моделювання модуля АЦП у пакеті PROTEUS

10.13.1. Опис моделі

Схему моделювання модуля АЦП у пакеті PROTEUS 8.6 зображено на рис. 10.12.

Ця схема складається з наступних елементів:

- U2 – AVR-мікроконтролер ATMega32;
- R3… R6 – резистори опором 400Ом;
- RV1 – резистор змінного опору на 10кОм;
- BAT2 – джерело напруги 5В;
- VCC – джерело живлення мікроконтролера;
- U1 – семисегментний чотирипозиційний індикатор;

– V1 – вольтметр для вимірювання вхідної напруги.

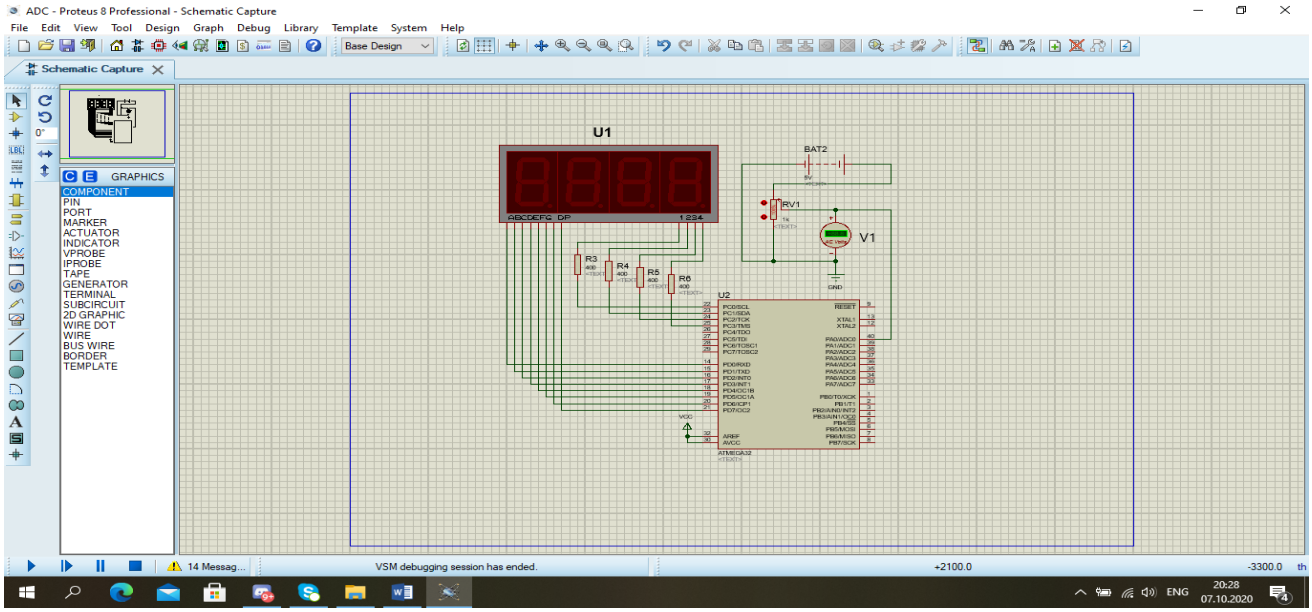


Рис. 10.12. Схема моделювання модуля АЦП в PROTEUS 8.6

Вхідна аналогова напруга подається на лінію PA0 (ADC0) порту A з виходу дільника напруги +5В (RV1). За допомогою резистора RV1, можна змінювати величину вхідної напруги від 0В до +5В.

Після перетворення її значення у цифровий еквівалент, відповідна величина виводиться у порти мікроконтролера для відображення на індикаторі. Індикатор підключено до ліній PC0…PC3 порту C та ліній PD0…PD7 порту D.

АЦП програмується у режим безперервного перетворення з використанням каналів з однополярним входом. Вище для однополярного режиму представлено функцію перетворення АЦП (рис. 10.11), а табл. 10.11 відображає зв'язок між вхідним сигналом та вихідними кодами.

Для каналів з однополярним (несиметричним) входом результат перетворення визначається виразом (10.1), а коефіцієнт передачі – виразом (10.2), які наведено вище.

При U_{REF} = 5В, $K_{ПЕР}$ = 1023/5000 = 0,2046 [*МЗР/мВ*].

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

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

Нижче на рис. 10.13 наведено схему моделі після запуску процесу моделювання. Якщо за допомогою дільника RV1 встановити на вольтметрі V1 напругу +5В, то на індикаторі U1 побачимо число 1023, що відповідає наведеному вище розрахунку.

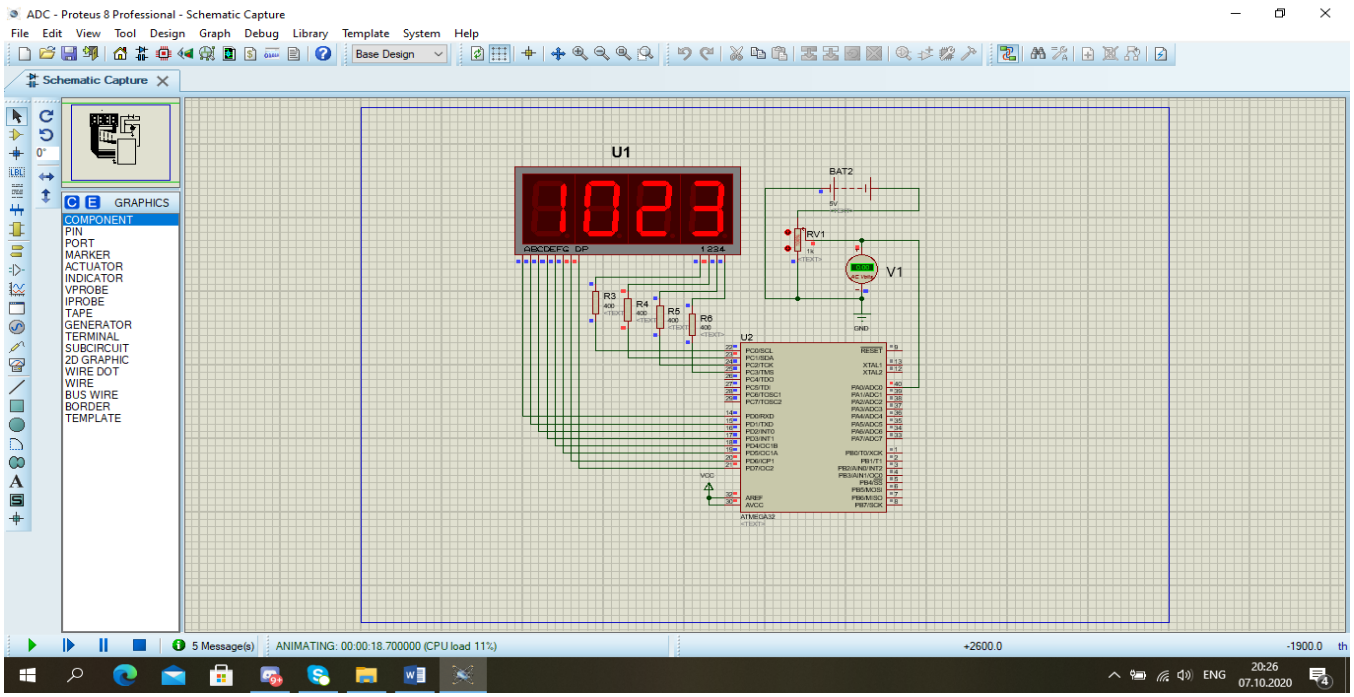


Рис. 10.13. Схема моделі АЦП після запуску процесу моделювання

Для налаштування параметрів АЦП у програмі виконуються такі дії: у регістрі ADCSRA встановлюються в логічну одиницю біти: ADEN – дозвіл АЦП, ADSC – запуск перетворення, ADATE – безперервний режим роботи, ADPS2 та ADPS1 – переддільник на 64, ADIE – дозвіл переривань від АЦП. Щоб обрати зовнішнє джерело опорної напруги біти REFS1 і REFS0 у регістрі ADMUX скидаються у логічний нуль.

Для відображення отриманого десяткового коду перетворення АЦП на семисегментних індикаторах використовується таймер/лічильник T2 мікроконтролера. При його переповненні відбувається переривання та результат аналого-цифрового перетворення відображається на чотирьохпозиційному семисегментному індикаторі. Для цього у порт C подається число, в якому в логічну одиницю встановлено біт для вибору активного індикатора. Перший індикатор, якщо лічити зліва направо, обирається розрядом PC0 = 1 (інші біти порту C дорівнюють нулю). При виборі другого індикатора PC0 = 2, інші біти порту C дорівнюють нулю, і т. д. Після вибору індикатора, за допомогою математичної операції отримання остачі, визначається відповідна цифра, яка є індексом в масиві SEG (див. робочу програму) та відповідає своєму семисегментному аналогу. Оскільки окремі сегменти індикатора, який використано у моделі, активуються нульовим сигналом, то відповідне значення масива SEG у програмі перед виведенням в порт D інвертується.

Для дозволу переривань від таймера/лічильника T2 використовується регістр TIMSK (Timer/Counter InterruptMaSK Register – регістр маски переривань

від таймерів/лічильників) [1]. Для дозволу переривання необхідно встановити в одиницю розряд TOIE2 цього регістра, а також встановити в одиницю прапорець глобального дозволу переривань – I регістра SREG.

В якості тактового сигналу f_{clk2} таймера/лічильника T2 використовується масштабований системний тактовий сигнал: $f_{clk2} = f_{clkI/0/}N$, де N – коефіцієнт ділення попереднього дільника. Програмування N здійснюються за допомогою розрядів CS22...CS20 регістра керування таймером – TCCR2. В моделі обрано N = 8. Для програмування цього значення згідно з [1] треба задати: CS22 = CS20 = 0, CS21 = 1.

10.13.2. Схема алгоритму роботи

Схему алгоритму роботи моделі наведено на рис. 10.14.

Після виконання блоку ініціалізації (блок 1) АЦП запускається в режим безперервного перетворення. Паралельно починає роботу таймер/лічильник T2, який при кожному переповненні викликає підпрограму його обробки. Задачею цієї підпрограми є виведення на дисплей результату перетворення від АЦП, який формується у десятковому коді. До завершення першого перетворення АЦП на дисплей виводиться нульове значення. Коли АЦП закінчує перше перетворення, встановлюється відповідний прапорець, виконання програми переривається та викликається підпрограма обробки цього переривання.

При черговому переповненні таймер отримує результат від АЦП та відповідна підпрограма виводить його на дисплей.

Далі АЦП буде безперервно виконувати наступні перетворення, результат яких за допомогою таймера також буде виводитись на дисплей.

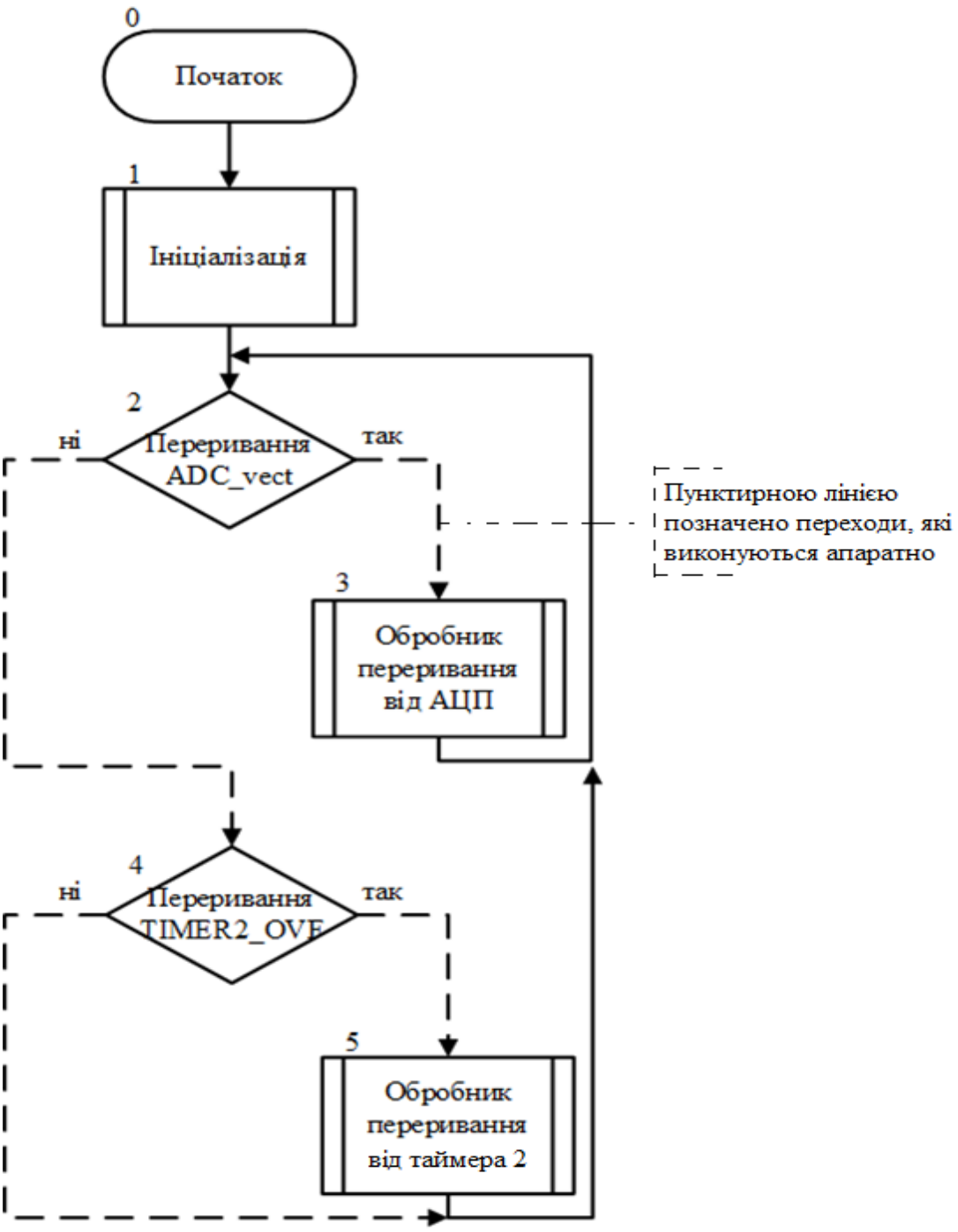


Рис. 10.14. Схема алгоритму роботи модуля

Схему алгоритму ініціалізації модуля наведено на рис. 10.15.

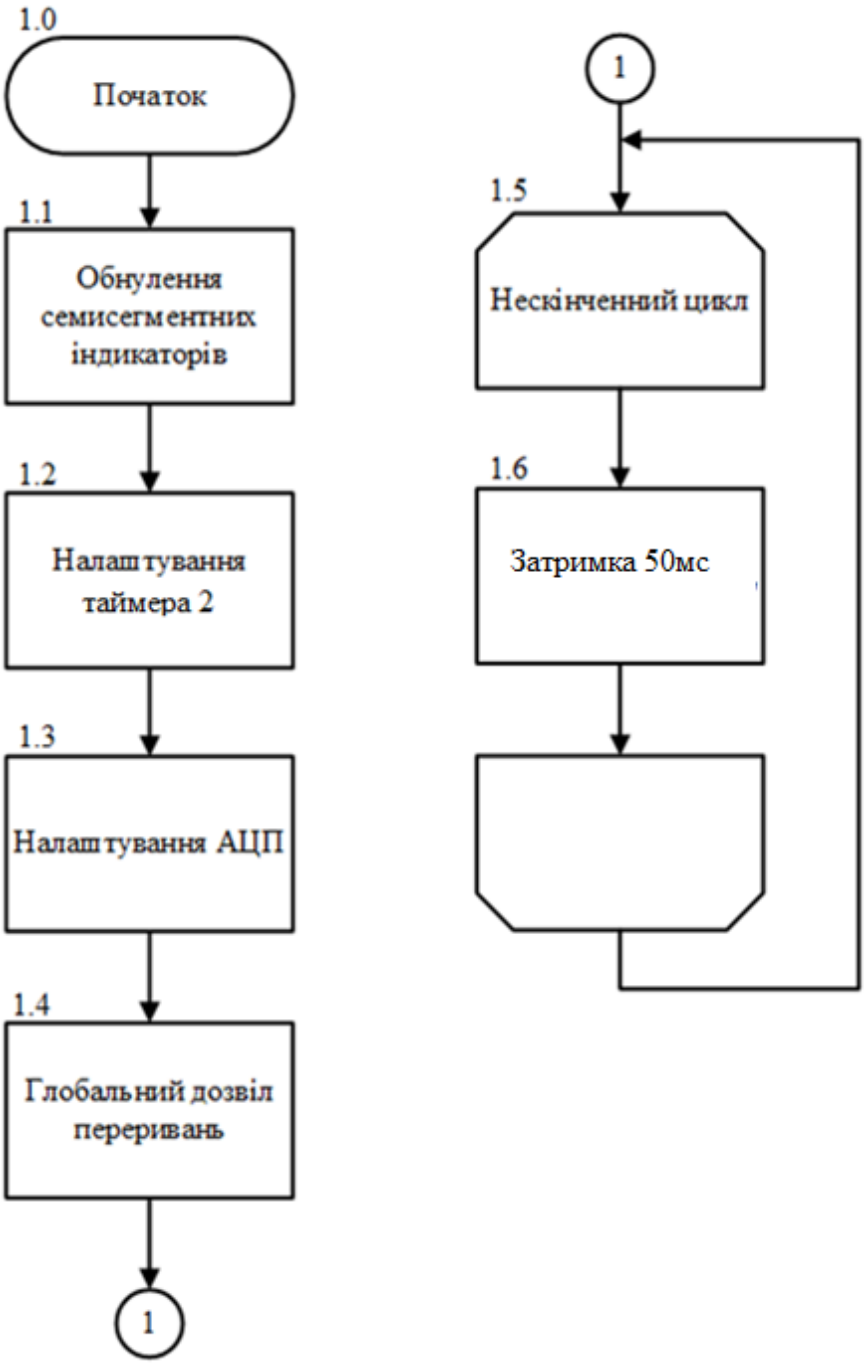


Рис. 10.15. Схема алгоритму ініціалізації модуля

Схему алгоритму обробки переривання від таймера Т2 наведено на рис. 10.16.

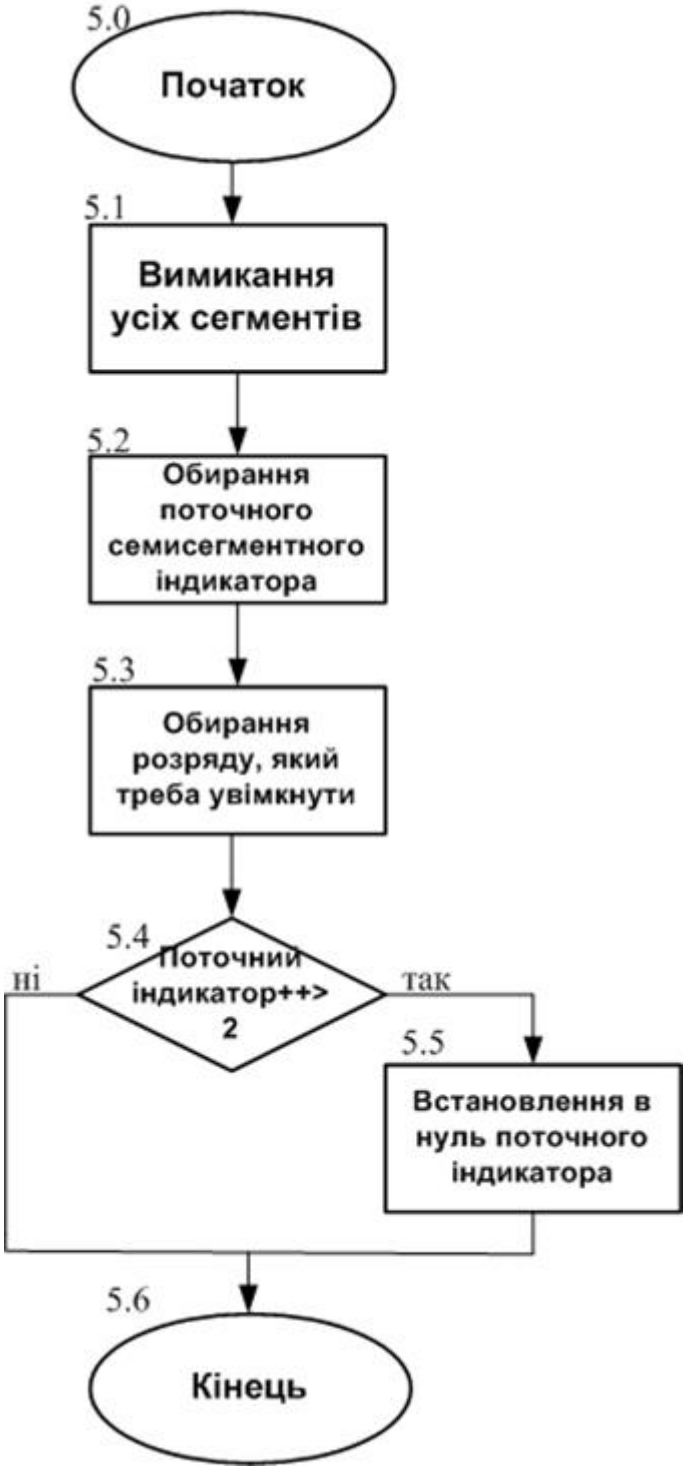


Рис. 10.16. Схема алгоритму обробки переривання від таймера Т2

Схему алгоритму обробки переривання від завершення перетворення АЦП наведено на рис. 10.17.

Рис. 10.17. Схема алгоритму обробки переривання від завершення

10.13.3. Робоча програма мовою С

```
// Підключення файлів бібліотек
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
// Масив значень для відображення цифр на семисегментних індикаторах
//——0———1———2———3———4———5———6———7———8———9——dp
char SEG[] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F, 0x80};
// Ініціалізації глобальних змінних
volatile unsigned char current_indicator = 0;
volatile unsigned int display = 0;
int main (void) //0
{
// Блок 1.0 – Ініціалізація програми
   // Блок 1.1 – Обнулення портів семисегментних індикаторів
   DDRC = 0xFF;
   PORTC = 0x00;
   DDRD = 0xFF;
   PORTD = 0x00;
   // Блок 1.2 – Налаштування таймера T2
   TIMSK |= (1 << TOIE2);  // Дозвіл переривання від таймера T2
   TCCR2 |= (1 << CS21);   // Переддільник на 8
   // Блок 1.3 – Налаштування АЦП
   ADCSRA |= (1 << ADEN)| // Дозвіл АЦП
```

```
    |(1 << ADSC)|  // Запуск перетворення
    |(1 << ADATE)|  // Безперервний режим роботи АЦП
    |(1 << ADPS2)|(1 << ADPS1)| // Переддільник на 64
    |(1 << ADIE);  // Дозвіл переривань від АЦП
    ADMUX &= (~(1 << REFS1))&(~(1 << REFS0));  // Зовнішнє ДОН
        // Блок 1.4 – Глобальний дозвіл переривань
    sei();
    // Блок 1.5 – Основний цикл
    while(1)
    {
        _delay_ms(50);  // Блок 1.6 – Затримка 50 мс
    }
}
// Блок 2 – Умова переривання від АЦП
ISR (ADC_vect)
{ // Блок 3.0 – Обробник переривань від АЦП
    Display = ADC; // Блок 3.1 – Присвоєння глобальній змінній «display» поточне
значення АЦП
}
// Блок 4 – Умова переривання від таймера T2
ISR (TIMER2_OVF_vect)
{ // Блок 5.0 – Обробник переривань від таймера T2
    PORTD = 0xFF;  // Блок 5.1 – Вимикання всіх сегментів
    PORTC = (1 << current_indicator); // Блок 5.2 – Обирання поточного індикатора
(починаючи зліва направо)
            // Блок 5.3
    switch (current_indicator)
    {
        case 0:
        PORTD = ~(SEG[display % 10000 / 1000]); // Вмикання цифри тисяч
        break;
        case 1:
        PORTD = ~((SEG[display % 1000 / 100])); // Вмикання цифри сотень
        break;
        case 2:
        PORTD = ~(SEG[display % 100 / 10]); // Вмикання цифри десятків
        break;
        case 3:
```

```
    PORTD = ~(SEG[display % 10 / 1]); // Вмикання цифри одиниць
    break;
  }
    if ((current_indicator++) > 2) // Блоки 5.4, 5.5 – Перехід на наступний
                                   //індикатор, якщо current_indicator не більше двох
    current_indicator = 0; // Блок 5.6 – Обнулення current_indicator, якщо він
                                   //більше двох
  }
```

Нижче наведено пояснення фрагменту програми виведення на чотири семисегментних індикатори результату перетворення АЦП – ADC, який надходить у десятковому коді згідно з формулою 10.1, що наведено вище. Для виведення ADC використовується математична операція «остача», яка в мові програмування «С» обчислюється оператором «%».

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

1023%10000/1000 = 1023/1000 = 1,023. Цифра 1 виводиться на перший індикатор (зліва направо).

Далі відбуваються наступні дії:

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

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

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

Наприклад, ADC = 686, тоді:

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

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

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

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

10.14. Моделювання у пакеті PROTEUS цифрового вольтметра

10.14.1. Особливості архітектури модуля АЦП у складі мікроконтролера ATmega32

Основним елементом цифрового вольтметра є аналого-цифровий перетворювач, в якості якого використовується відповідний модуль мікроконтролера ATmega32. Архітектуру цього модуля розглянуто вище у підрозділах 10.1…10.11.

10.14.2. Опис моделі цифрового вольтметра

Робочу модель цифрового вольтметра у пакеті PROTEUS 8.6 показано на рис. 10.18.

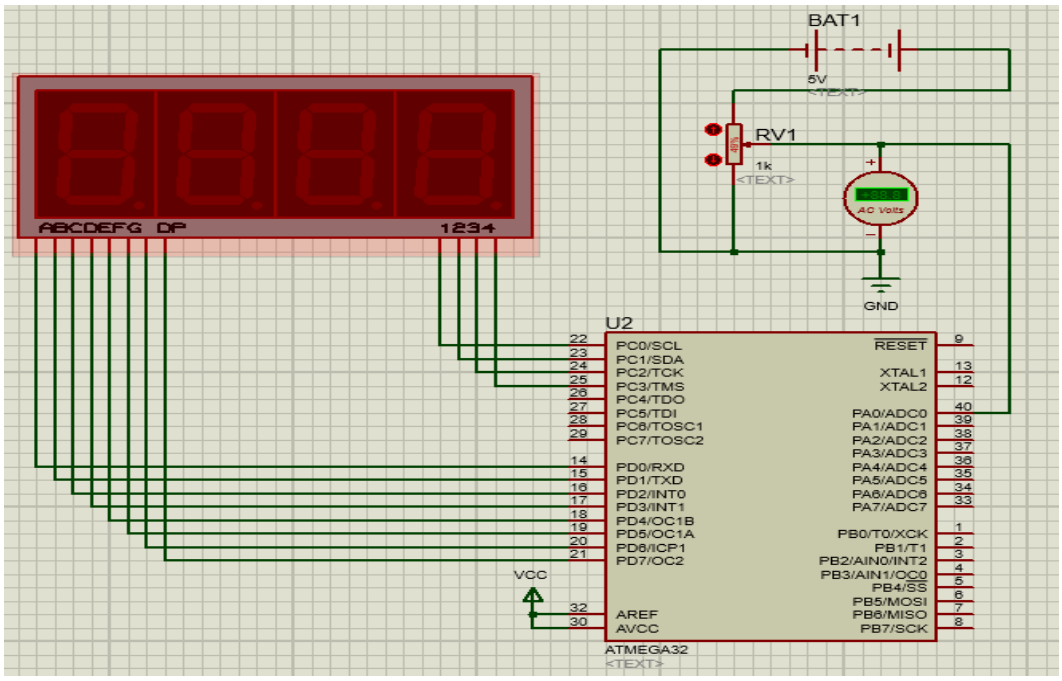


Рис. 10.18. Схема моделі цифрового вольтметра у пакеті PROTEUS 8.6

З лівого боку моделі зображено 7-сегментний чотирьохпозиційний цифровий дисплей, який з’єднано з портами C та D мікроконтролера ATMega32 відповідними лініями зв’язку. У верхній частині рисунку знаходиться батарея BAT1 та резистор RV1, напругу з виходу якого можна змінювати. Ця вхідна аналогова напруга подається на лінію PA0 (ADC0) порту A мікроконтролера та перетворюється модулем АЦП у цифровий еквівалент. Особливості керування цифровим дисплеєм описано у попередньому підрозділі. Різниця між виведенням результату перетворення АЦП, який описано вище, полягає в тому, що раніше на дисплей виводилася цифрова інформація у чотирьохрозрядному десятковому коді, а в моделі цифрового вольтметра відображається абсолютне значення вхідної напруги у вольтах з точністю до сотих долей вольта.

На рис. 10.19 наведено зовнішній вигляд 7-сегментного індикатора, який входить до складу чотирьохпозиційного цифрового дисплею. Для отримання на індикаторі чисел потрібно керувати сегментами індикатора A, B, C, D, E, F, G та точкою Dp згідно до табл. 10.12; 10.13.

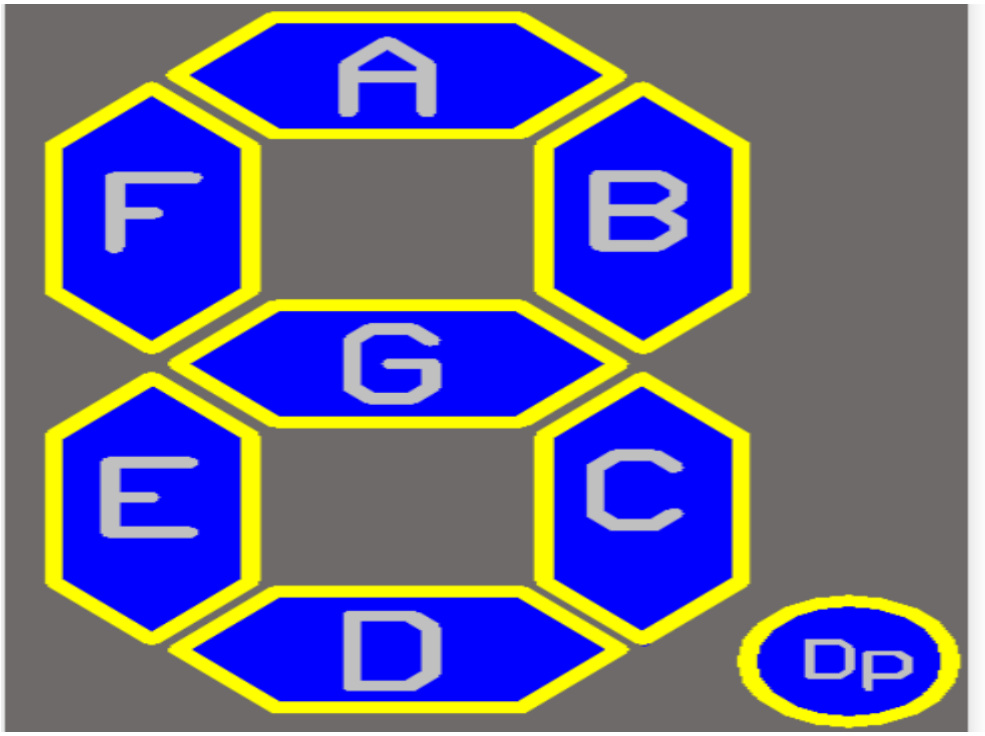


Рис. 10.19. 7-сегментний індикатор

Таблиця 10.12. **Зв'язок між цифрами на індикаторі та значенням керуючих сигналів без використання точки**

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

Таблиця 10.13. **Зв'язок між цифрами на індикаторі та значенням керуючих сигналів з використанням точки**

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

Нижче на рис. 10.20, 10.21 наведено декілька прикладів роботи моделі.

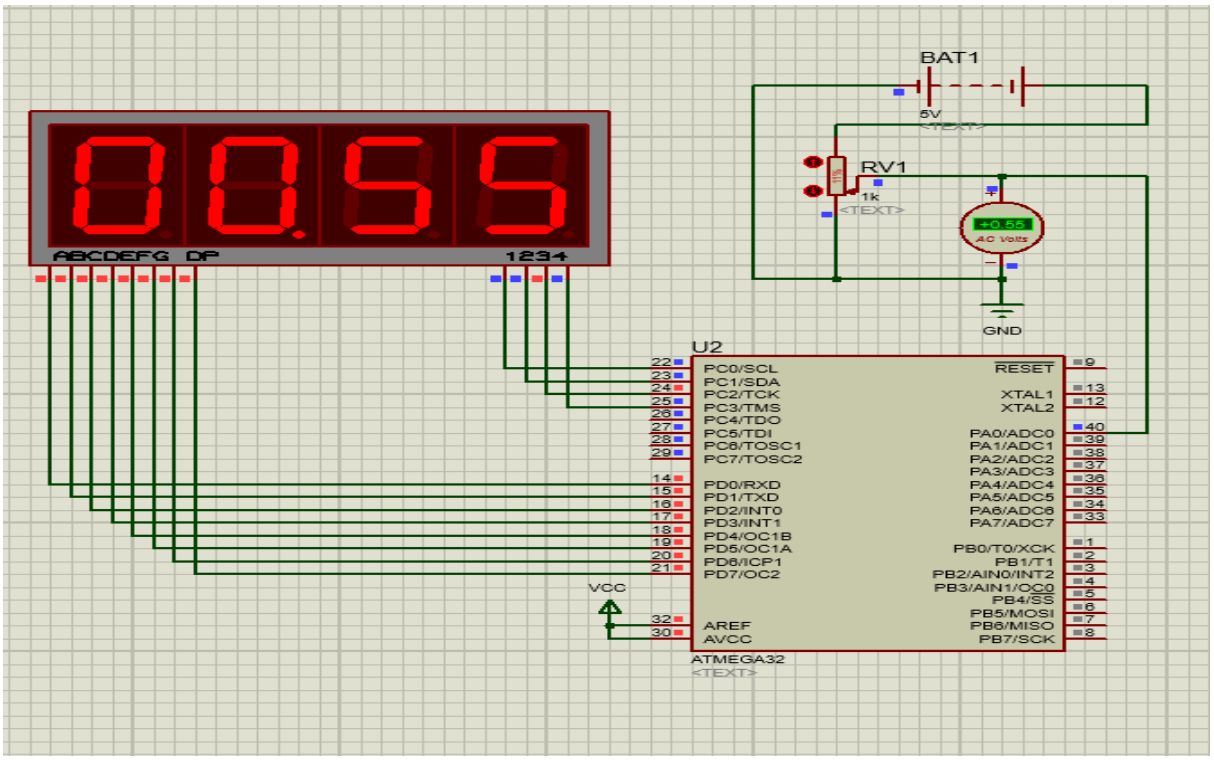


Рис. 10.20. Робота цифрового вольтметра при $U_{ВХ.АЦП}$ = 0,55В

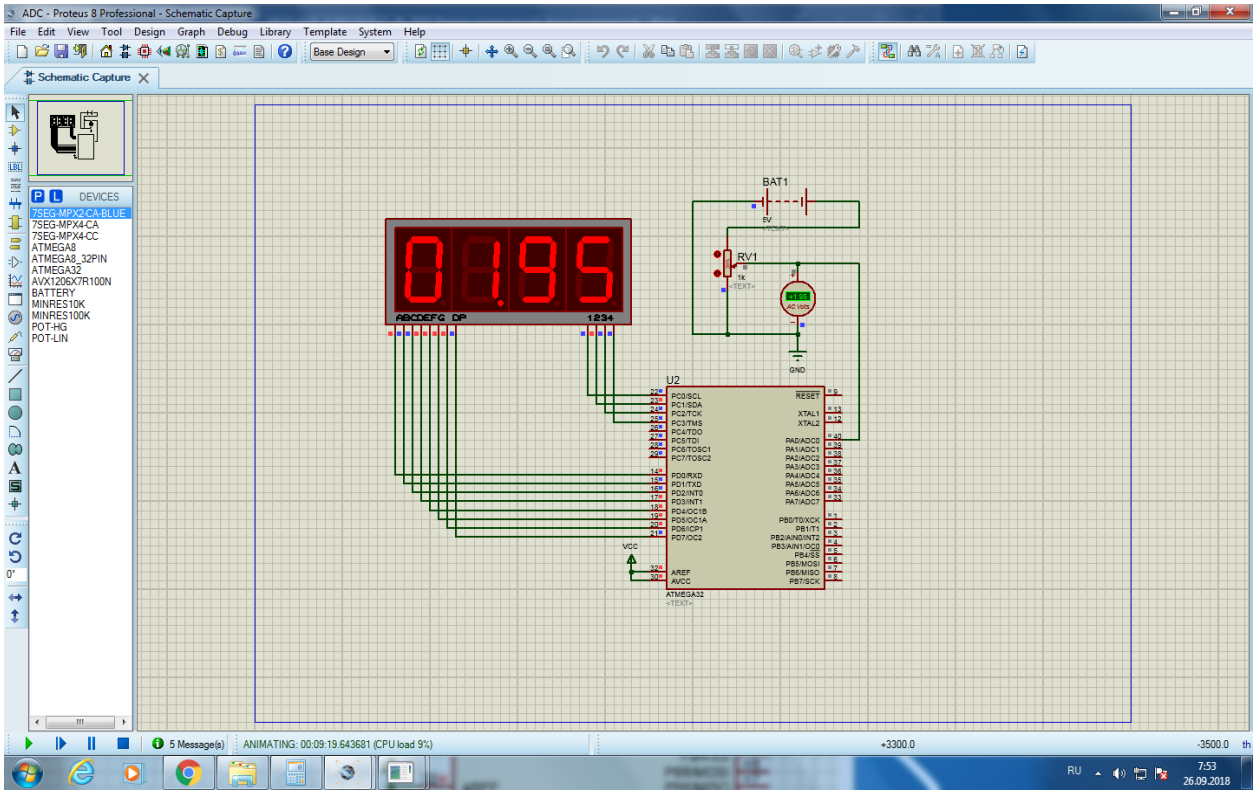


Рис. 10.21. Робота цифрового вольтметра при $U_{ВХ.АЦП}$ = 1,95В

10.14.3. Схема алгоритму роботи

Схема алгоритму роботи цифрового вольтметра подібна схемі алгоритму роботи модуля АЦП, яку наведено вище на рис. 10.14…10.17.

Ініціалізація модуля АЦП, таймера Т2 мікроконтролера та робота моделі у більшості виконується аналогічно описанному у підрозділі 10.13. Різниця полягає у виведенні результату моделювання на дисплей.

Нижче наведено пояснення перетворення значення ADC, яке отримується після завершення роботи модуля АЦП у десятковому коді, у значення вхідної напруги у вольтах для його виведення на дисплей.

Як відмічено у підрозділі 10.10 для каналів з однополярним (несиметричним) входом результат перетворення АЦП визначається виразом:

$$ADC = 1023 \cdot U_{IN}/U_{REF},$$

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

Коефіцієнт передачі АЦП

$$K_{ПЕР} = 1023/U_{REF}\,[МЗР/мВ].$$

При U_{REF} = 5В, $K_{ПЕР}$ = 1023/5000 = 0,2046 [*МЗР/мВ*].

Значення вхідної напруги

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

Наприклад, якщо на вхід АЦП було подано напругу 0,55В, тоді згідно з результатом моделюванням АЦП (див. підрозділ 10.13) ADC = 113.

Перетворення значення ADC_value = ADC в змінну «display», виконується згідно з виразом: display = (ADC_value) • (5/1023) •100. Тоді при ADC = 113 display = 0113 • (5/1023) • 100 = 55.

Тобто, при U_{IN} = 0,55В значення «display» згідно з робочою програмою дорівнює 55. Для виведення цього значення використовується математична операція «остача», яка в мові програмування «С» обчислюється оператором «%».

Це значення програма обробляє наступним чином:

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

Наприклад, якщо на вхід АЦП було подано напругу 1,95В, тоді згідно з моделюванням АЦП (див. підрозділ 10.13) ADC = 399.

Значення «display = 0399 • (5/1023) • 100 = 195.

Це значення програма обробляє наступним чином:

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

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

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

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

10.14.4. Робоча програма мовою С

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

// Масив значень для відображення цифр на семисегментних індикаторах
char SEG[] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F};
```

```
char SEG_with_dot[] = {0xBF, 0x86, 0xDB, 0xCF, 0xE6, 0xED, 0xFD, 0x87, 0xFF,
0xEF};

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

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

   // Блок 1.2 – Налаштування таймера T2
   TIMSK |= (1 << TOIE2); // Дозвіл переривання від таймера T2
   TCCR2 |= (1 << CS21);  // Переддільник на 8

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

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

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

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

```
}
// Блок 2 – Умова переривання від АЦП
ISR (ADC_vect)
{// Блок 3 – Обробник переривань від АЦП
   ADC_value = ADC; // Блок 3.1 – Присвоювання глобальній змінній поточне
значення АЦП
}

// Блок 4 – Умова переривання від таймера Т2
ISR (TIMER2_OVF_vect)
{  // Блок 5 – Обробник переривань від таймера Т2
   PORTD = 0xFF; // Блок 5.1 – Вимикання всіх сегментів
   PORTC = (1 << current_indicator); // Блок 5.2 – Обирання поточного індикатора
                                     //починаючи зліва направо)
   display = (ADC_value)*(5/1023)*100; // Обрахування значення напруги, яка
                                       //виводиться на дисплей у вольтах
   //Блок 5.3
   switch (current_indicator)
   {
      case 0:
      PORTD = ~(SEG[display % 10000 / 1000]); // Вмикання цифри десятків
      break;
      case 1:
      PORTD = ~(SEG_with_dot[display % 1000 / 100]); // Вмикання цифри
                                                     //одиниць
      break;
      case 2:
      PORTD = ~(SEG[display % 100 / 10]); // Вмикання цифри десятих
      break;
      case 3:
      PORTD = ~(SEG[display % 10 / 1]); // Вмикання цифри сотих
      break;
   }
   if ((current_indicator++) > 2) // Блоки 5.4; 5.5 – Перехід на наступний
                                  //індикатор, якщо current_indicator не більше двох
   current_indicator = 0; // Блок 5.6 – Обнулення current_indicator, якщо він
                          //більше двох
}
```

Контрольні запитання та завдання

1. До складу яких моделей AVR-мікроконтролерів входить модуль АЦП? Назвіть його розрядність.
2. Назвіть основні параметри АЦП.
3. Назвіть основні джерела опорної напруги для АЦП.
4. Назвіть режими роботи АЦП.
5. Яку роль у функціональній схемі модуля АЦП виконують: дешифратор, мультиплексор та пристрій вибірки та зберігання?
6. У якому регістрі зберігається результат перетворення?
7. Поясніть як саме відбувається перетворення вхідного сигналу у двійковий код?
8. Яку роль у функціональній схемі модуля АЦП виконує компаратор ?
9. Який принцип перетворення використовується в АЦП? Поясніть особливості цього принципу.
10. Який регістр відповідає за налаштування мультиплексора АЦП?
11. Наведіть та поясніть формати регістрів стану і керування АЦП.
12. Який розряд регістра ADCSRA відповідає за дозвіл роботи АЦП?
13. Як обрати джерело опорної напруги?
14. Як вибрати режим роботи АЦП?
15. За якими перериваннями може відбуватися запуск АЦП?
16. Яку функцію виконує попередній дільник?
17. Як формується тактовий сигнал АЦП?
18. Який регістр використовується для налаштування вхідного мультиплексора?
19. Поясніть часові діаграми роботи АЦП при першому одиночному перетворенні в режимі одиночного перетворення.
20. Поясніть часові діаграми роботи АЦП в режимі наступного одиночного перетворення.
21. Поясніть часові діаграми роботи АЦП в режимі запуску за перериванням.
22. Поясніть часові діаграми роботи АЦП в режимі автоматичного перезапуску для однополярного перетворення.
23. Скільки тактів синхронізації АЦП витрачається на перетворення в режимах: першого одиночного перетворення та наступного одиночного перетворення?
24. Скільки однополярних входів має вхідний мультиплексор модуля АЦП?
25. Як можна програмно врахувати величину зсуву?
26. Чому після перемикання диференціального каналу підсилювача перетворення не повинно стартувати не раніше, ніж через 125 мкс?
27. Назвіть способи вирівнювання результату АЦП.

28. Яке вирівнювання слід використовувати якщо досить точності 8-розрядного значення?
29. Які особливості перемикання каналів при одиночному перетворенні?
30. Які особливості перемикання каналів при перетворенні у режимі автоматичного перезапуску?
31. Що може виступати у ролі джерела опорної напруги?
32. Який вихідний опір повинно мати джерело вхідного сигналу?
33. Яким буде результат перетворення для каналів з однополярним входом?
34. Яким буде результат перетворення для каналів з диференціальним входом?
35. Якою формулою визначається коефіцієнт передачі АЦП?
36. Поясніть функцію перетворення АЦП при зміні однополярного сигналу.
37. Поясніть зв'язок між вхідним сигналом та вихідними кодами для однополярного режиму.
38. Як можна відключити вхідні цифрові буфери на виводах ADC0...ADC15 у випадку, якщо відповідні виводи використаються тільки для зчитування аналогових сигналів?
39. Назвіть методи підвищення точності перетворення АЦП.
40. Чому дорівнює похибка зсуву та як її можна програмно врахувати?
41. Що таке інтегральна нелінійність?
42. Що таке диференціальна нелінійність?
43. Опишіть відмінності в програмах при моделюванні модуля АЦП та цифрового вольтметра.
44. Опишіть особливості програмування мовою С виведення на дисплей результату роботи модуля АЦП та цифрового вольтметра.
45. Як розраховується відносна похибка АЦП від квантування за рівнем?
46. Опишіть схему моделювання модуля АЦП в пакеті PROTEUS 8.6.
47. В який режим програмується модуль АЦП при його моделюванні?
48. Чому дорівнює коефіцієнт передачі модуля АЦП при його моделюванні?
49. В якому вигляді виводиться на індикацію результат моделювання модуля?
50. За допомогою якої математичної операції визначається відповідна цифра при її виведенні на індикатор? Відповідь пояснити.
51. Який таймер/лічильник мікроконтролера використовується при виведенні на семисегментні індикатори?
52. Який сигнал використовується в якості тактового для таймера 2? Відповідь пояснити.
53. Чим відрізняється виведення результату моделювання цифрового вольтметра від моделювання модуля АЦП?

11. ПРОЕКТУВАННЯ МОДУЛЯ ЦИФРО-АНАЛОГОВОГО ПЕРЕТВОРЮВАЧА

11. 1. Характеристика модуля ЦАП у складі мікроконтролерів XMega

Частина AVR-мікроконтролерів, наприклад, XMega мають модуль цифро-аналогового перетворювача (ЦАП). Нижче наведено основні характеристики цього модуля [1, 18]:

- 12-розрядна роздільна здатність;
- частота перетворення до 1 МГц;
- гнучкий діапазон перетворення;
- кілька джерел запуску;
- в якості виходу ЦАП може бути один неперервний вихід для одноканального режиму роботи, або два окремих виходи зі схемою вибірки-зберігання для двоканального режим роботи.
- вбудоване калібрування зміщення і коефіцієнта передачі;
- внутрішнє або зовнішнє джерело опорної напруги (ДОН);
- можливість використання в якості входу для аналогового компаратора та АЦП;
- наявність енергозберігаючого режиму роботи;
- висока навантажувальна здатність – внутрішній опір становить 300 Ом для ATXMegaA3, ATXMegaA4 та 850 Ом для ATXMegaA1.

11.2. Принцип роботи модуля

Модуль ЦАП призначено для перетворення цифрового коду в аналогову напругу.

Значення вихідного сигналу залежить від опорної напруги U_{ref}.

Вихідна напруга та коефіцієнт передачі каналу ЦАП обчислюються за наступними виразами:

$$U_{\text{цап}} = \frac{CHnDATA}{4096} \cdot U_{ref},$$

$$K_{\text{цап}} = U_{ref}/4096 \text{ [мВ/МЗР]},$$

де $CHnDATA$ – десятковий еквівалент числа, яке зберігається в регістрі даних, n = 0/1 – номер регістра, U_{ref} – опорна напруга.

На рис. 11.1 зображено функціональну схему модуля ЦАП.

Схема функціонує наступним чином. Вхідні дані для перетворення записуються у вхідні регістри даних CH0DATA (в одноканальному режимі роботи) або CH0DATA та CH1DATA (у двоканальному режимі роботи).

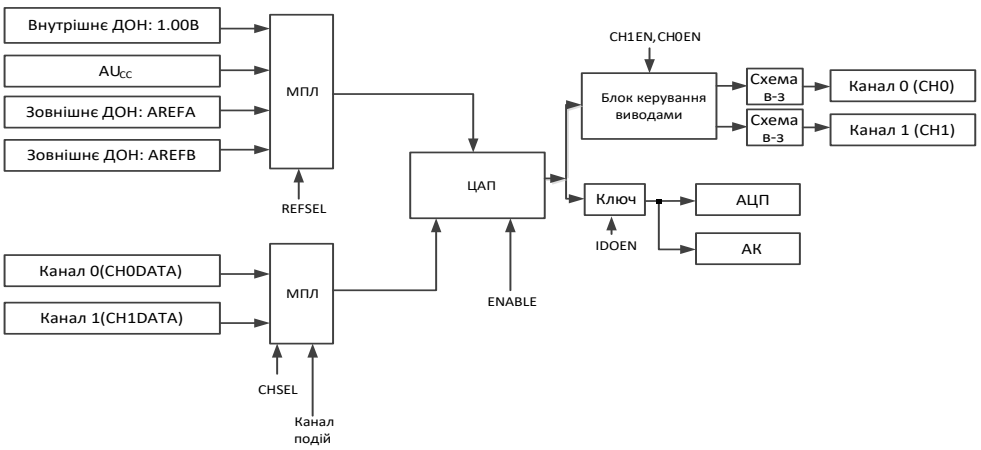


Рис. 11.1. Функціональна схема модуля ЦАП

Після початку перетворення, дані з регістрів крізь мультиплексор потрапляють безпосередньо на схему ЦАП, де за допомогою джерела опорної напруги – U_{ref}, обраного відповідними бітами регістра керування, виконується перетворення. Регістри керування задають режим живлення, вибір каналу, джерело запуску, дані калібрування тощо. Аналоговий сигнал, отриманий під час перетворення, в залежності від обраного режиму роботи подається через блок керування виводами в канал 0 та канал 1, та/або на вхід аналогового компаратора та АЦП, в залежності від значення розряду IDOEN регістра CTRLA.

11.3. Джерела опорної напруги

Для ЦАП у складі XMega можуть використовуватися наступні джерела опорної напруги:

- внутрішнє ДОН зі значенням 1.00В;
- зовнішнє джерело AU_{CC};
- зовнішня напруга, що подається на вивід AREF порту A та/або B.

11.4. Вихідні канали

В якості виходу ЦАП можуть служити або один аналоговий вихід (канал 0), або два окремих виходи зі схемами вибірки-зберігання (в-з). Виходи схеми вибірки-зберігання можуть працювати повністю незалежно, дозволяючи генерувати два аналогових сигнали, що розрізняються як за амплітудою, так і за частотою.

Для кожного з виходів схем вибірки-зберігання передбачено окремі регістри даних та регістри керування. Вихідна напруга ЦАП може бути подана на вхід аналогового компаратора (АК), або АЦП. (рис. 11.1).

11.5. Режими роботи

Модуль ЦАП може працювати в одноканальному, або двоканальному режимі роботи.

Одноканальний режим роботи

В одноканальному режимі роботи регістр даних CH0DATA через мультиплексор з'єднується з входом ЦАП (рис. 11.2).

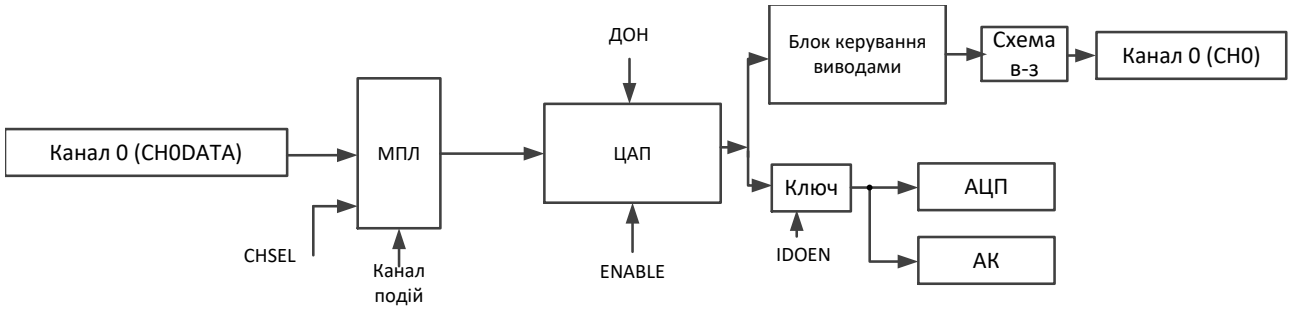


Рис. 11.2. Функціональна схема роботи ЦАП в одноканальному режимі

Нижче наведено послідовність програмування ЦАП для одноканального режиму роботи:

– записом 0x00 в розряди CHSEL[1:0] регістра керування CTRLB, встановити одноканальний режим роботи;

– скинути розряд CH0TRIG в регістрі керування CTRLB для запуску перетворення за записом у регістр даних;

– записом 01 в розряди REFSEL[1:0] регістра керування CTRLC обрати в якості ДОН аналогову напругу AU_{CC};

– встановити в регістрі керування CTRLA розряд дозволу роботи каналу 0 - CH0EN;

– встановити в регістрі керування CTRLA розряд дозволу роботи модуля ЦАП - ENABLE;

– записати 12-розрядне значення (з правим вирівнюванням) в регістр даних каналу 0 - CH0DATA для запуску перетворення. Молодші біти записуються в першу чергу.

Двоканальний режим роботи

У двоканальному режимі роботи (рис. 11.3) ЦАП по черзі перетворює дані з CH0DATA та CH1DATA. Блоки вибірки-зберігання використовуються для збереження значень між перетвореннями.

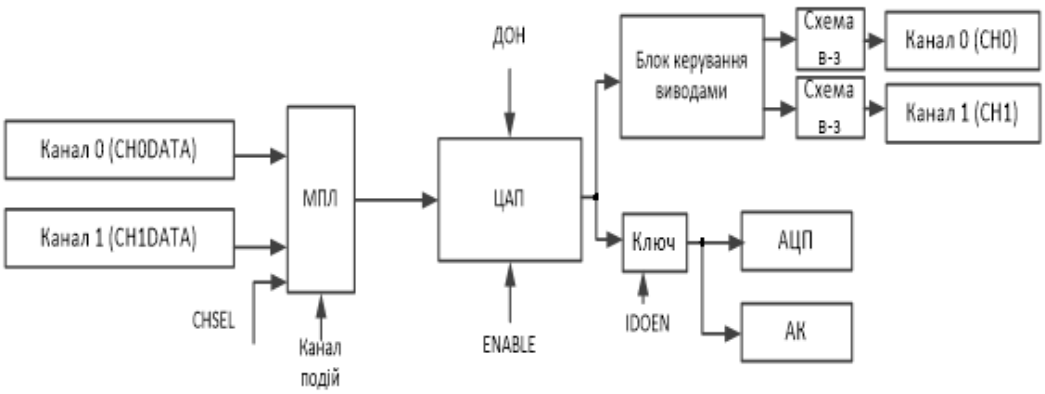


Рис. 11.3. Функціональна схема роботи ЦАП у двоканальному режимі

Для коректної видачі вихідного значення на два виходи ЦАП повинен регулярно оновлювати канали.

Нижче наведено послідовність програмування ЦАП для двоканального режиму роботи:

– записом 0x01 в розряди CHSEL[1:0] регістра керування CTRLB встановити двоканальний режим роботи;

– для запуску перетворення за записом в регістр даних, а не за подією скинути розряд CH0TRIG в регістрі керування CTRLB;

– встановити в регістрі керування CTRLC розряди REFSEL в 0x01 для того, щоб обрати аналогове ДОН;

– встановити в регістрі керування CTRLA розряди дозволу роботи каналу 0 (CH0EN) та каналу 1 (CH1EN);

– встановити розряди вибірки каналу CONINTCAL[2:0] регістра керування таймером TIMCTRL в 0x04 для встановлення інтервалу між перетвореннями в 24 такти (інтервал, що дорівнює 3 мкс при периферійній частоті 8 МГц);

– встановити розряди REFRESH[3:0] регістра керування часом TIMCTRL в 0x06 (інтервал оновлення величиною в 1024 такти, що дорівнює 128 мкс при периферійній частоті 8 МГц);

– встановити розряд ENABLE в регістрі керування CTRLA для дозволу роботи модуля ЦАП;

– записати 12-розрядне значення (з правим вирівнюванням) в регістр даних відповідного каналу для запуску перетворення. При відсутності змін в регістрах даних виходи будуть оновлюватися кожні 128 мкс.

11.6. Тактування модуля

Модуль ЦАП тактується сигналом синхронізації від зовнішнього тактового генератора. Інтервал перетворень і частота оновлення в двоканальному режимі задаються кратними періоду сигналу синхронізації.

11.7. Обмеження часових характеристик

Для коректної роботи модуля ЦАП необхідно дотримуватися ряду часових обмежень, які задаються кратними періоду сигналу синхронізації. Недотримання обмежень може погіршити точність перетворення.

Час оновлення ЦАП – це інтервал часу між оновленнями каналів у двоканальному режимі. Величина цього інтервалу не повинна перевищувати 30 мкс. Цей параметр впливає на те, щоб забезпечувати підтримку стабільного вихідного сигналу на два виходи у двоканальному режимі роботи. Ця необхідність виникає тому що, схема буде втрачати амплітуду сигналу з часом, як конденсатор втрачає напругу завдяки під'єднаному до нього резистору. Більш висока частота оновлення спричиняє більш високе енергоспоживання.

Інтервал вибірки та перетворення ЦАП – це проміжок часу від моменту початку перетворення в каналі до запуску нового перетворення. Цей проміжок не повинен бути менше 1 мкс в одноканальному режимі і 1,5 мкс у двоканальному режимі (режимі вибірки-зберігання). Фактично це час, необхідний для перетворення та зберігання значення для аналогового виходу. Це аналогічно процесу зарядки конденсатора, якщо часовий інтервал завеликий, ви можете втратити інформацію з сигналу з більш високою швидкістю наростання вхідної напруги. Якщо інтервал вибірки та перетворення менший ніж інтервал оновлення, то канали будуть оновлені у визначений час, навіть якщо зайві (повторні) перетворення було виконано між інтервалами, що викликано ручним оновленням даних в регістрі.

На рис. 11.4 наведено часові діаграми оновлення та запитів перетворення каналів 0 та 1, якщо ЦАП запрограмовано на роботу у двоканальному режимі.

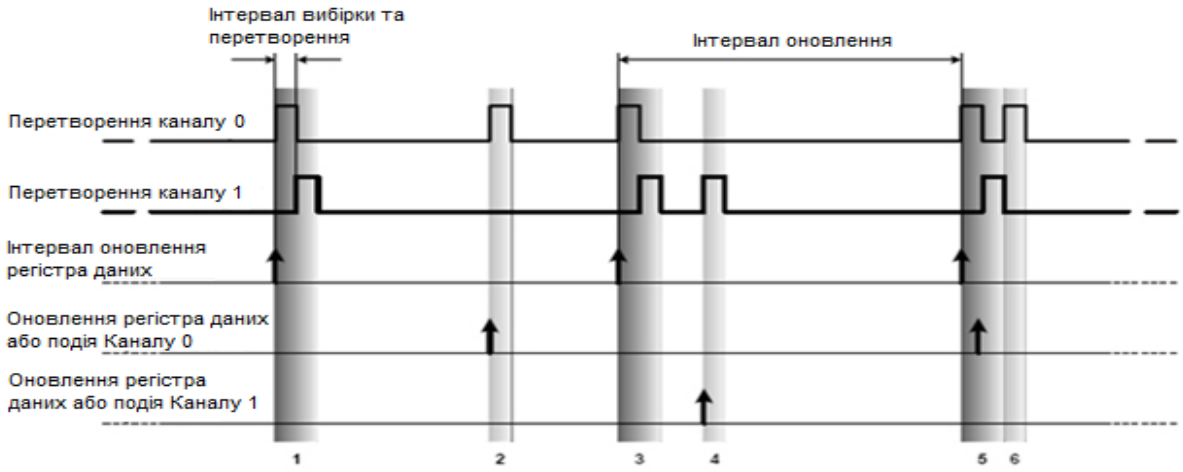


Рис. 11.4. Оновлення та запити перетворення каналів

Цифрами від 1 до 6 відмічено окремі моменти часу, в які відбуваються події які описано нижче:

1) Одночасно з інтервалом вибірки і перетворення починається інтервал оновлення каналу 0. Після закінчення першого інтервалу вибірки та перетворення настає інтервал вибірки та перетворення каналу 1.
2) Запит на перетворення (оновлення регістра даних або подія) для каналу 0, ініціює вибірку і перетворення тільки каналу 0, навіть якщо сигнал з'являється в середині інтервалу оновлень.
3) Початок чергового інтервалу оновлення, аналогічно п.1.
4) Запит перетворення (оновлення регістра даних або подія) для каналу 1, ініціює вибірку та перетворення тільки каналу 1, навіть якщо обидва канали були тільки що оновлені.
5) Початок чергового інтервалу оновлення, аналогічно п.1. Запит перетворення каналу 0 відкладається до моменту описаному в п.6.
6) Відкладений запит перетворення в п.5 ініціює чергову вибірку та перетворення по каналу 0 відразу після закінчення вибірки та перетворення каналу 1.

Зверніть увагу на те, що якщо частота запитів перетворення досягає частоти оновлення, частота перетворення може стати неточною, оскільки в момент надходження запиту перетворення може виконуватись звичайне оновлення. Такий запит відкладається до завершення оновлення обох каналів. При надходженні запитів перетворення для декількох каналів до виконання першого з них всі, крім першого запиту, буде проігноровано.

Якщо для конкретної програми це є проблемою, можна вимкнути автоматичне оновлення і натомість виконувати оновлення з достатньо високою частотою вручну. Зауважимо, що вимкнути автоматичне оновлення тільки для одного каналу неможливо.

11.8. Режим енергозбереження

При необхідності зниження модулем ЦАП споживаного струму під час перетворень, його можна перевести в економічний режим роботи. У цьому режимі між виконанням перетворень модуль переходить у відключений стан. Робота в цьому режимі супроводжується збільшенням часу перетворення при запуску нового перетворення.

11.9. Система подій

Система подій мікроконтролерів XMega − це набір функцій, який дозволяє периферійним модулям взаємодіяти один з одним без втручання центрального

процесора. Деякі периферійні модулі можуть генерувати події, часто за тими ж умовами, що і переривання. Ці події проходять через систему маршрутизації подій до споживачів подій, де споживачами подій можуть бути ініційовані певні дії. Центральний процесор не бере участі в цьому процесі, за винятком етапу налаштування. Наприклад, можна ініціювати захоплення входу таймера/лічильника при натисканні користувачем на кнопку, або почати аналого-цифрове перетворення при переповненні таймера/лічильника.

11.10. Програмування модуля

11.10.1. Регістри даних

На рис. 11.5…11.8 наведено опис двох регістрів: CHnDATAH і CHnDATAL (де n = 0/1) − відповідно старша і молодша частини 12-розрядного значення CHnDATA, яке перетворюється в аналоговий сигнал. За замовчуванням 12-розрядів поділяються на 8 розрядів в CHnDATAL і 4 CHnDATAH з позиції молодшого значущого розряду − вирівнювання вправо. Для деяких програм краще використовувати лівоспрямовані розряди. Вибрати дані з вирівнюванням вліво можна, встановлюючи розряд LEFTADJ у регістрі CTRLC.

	розряд	7	6	5	4	3	2	1	0
Вирівнювання вправо	+0x18	CHDATA[7:0]							
Вирівнювання вліво	+0x18	CHDATA[3:0]				-	-	-	-
Вирівнювання вправо	Читання/Запис	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Вирівнювання вліво	Читання/Запис	R/W	R/W	R/W	R/W	R	R	R	R
Вирівнювання вправо	Початкове значення	0	0	0	0	0	0	0	0
Вирівнювання вліво	Початкове значення	0	0	0	0	0	0	0	0

Рис. 11.5. CH0DATAL − молодший байт регістра даних 0

	розряд	7	6	5	4	3	2	1	0
Вирівнювання вправо	+0x19	-	-	-	-	CHDATA[11:8]			
Вирівнювання вліво	+0x19	CHDATA[11:4]							
Вирівнювання вправо	Читання/Запис	R	R	R	R	R/W	R/W	R/W	R/W
Вирівнювання вліво	Читання/Запис	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Вирівнювання вправо	Початкове значення	0	0	0	0	0	0	0	0
Вирівнювання вліво	Початкове значення	0	0	0	0	0	0	0	0

Рис. 11.6. CH0DATAH − старший байт регістра даних 0

	розряд	7	6	5	4	3	2	1	0
Вирівнювання вправо	+0x1A	CHDATA[7:0]							
Вирівнювання вліво	+0x1A	CHDATA[3:0]				-	-	-	-
Вирівнювання вправо	Читання/Запис	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Вирівнювання вліво	Читання/Запис	R/W	R/W	R/W	R/W	R	R	R	R
Вирівнювання вправо	Початкове значення	0	0	0	0	0	0	0	0
Вирівнювання вліво	Початкове значення	0	0	0	0	0	0	0	0

Рис. 11.7. CH1DATAL − молодший байт регістра даних 1

	розряд	7	6	5	4	3	2	1	0
Вирівнювання вправо	+0x1B	-	-	-	-	CHDATA[11:8]			
Вирівнювання вліво	+0x1B	CHDATA[11:4]							
Вирівнювання вправо	Читання/Запис	R	R	R	R	R/W	R/W	R/W	R/W
Вирівнювання вліво	Читання/Запис	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Вирівнювання вправо	Початкове значення	0	0	0	0	0	0	0	0
Вирівнювання вліво	Початкове значення	0	0	0	0	0	0	0	0

Рис. 11.8. CH1DATAH − старший байт регістра даних 1

11.10.2. Регістри керування

CTRLA-регістр керування А (рис. 11.9)

Bit	7	6	5	4	3	2	1	0	
+0x00	-	-	-	IDOEN	CH1EN	CH0EN	.	ENABLE	CTRLA
Read/Write	R	R	R	R/W	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рис. 11.9. CTRLA-регістр керування А

– Bits 7:5 – зарезервовано;

– Bit 4 − IDOEN: дозвіл виведення в АЦП та АК. Встановлення цього розряду в одиницю направляє внутрішній вихід ЦАП до АЦП та АК;

– Bit 3 − CH1EN: дозвіл виведення каналу 1. Встановлення цього розряду в одиницю виводить результат перетворення на вихід мікросхеми, інакше канал доступний тільки для внутрішнього використання.

– Bit 2 − CH0EN: дозвіл виведення каналу 0. Встановлення цього розряду в одиницю виводить результат перетворення на вихід мікросхеми, інакше канал доступний тільки для внутрішнього використання;

– Bit 1 – зарезервовано;

– Bit 0 − ENABLE: дозвіл ЦАП. Встановлення цього розряду в одиницю дозволяє функціонування ЦАП.

CTRLB-регістр керування B (рис. 11.10)

Bit	7	6	5	4	3	2	1	0	
+0x01	-	CHSEL[1:0]		-	-	-	CH1TRIG	CH0TRIG	CTRLB
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рис. 11.10. CTRLB-регістр керування B

– Bit 7 – зарезервовано;

– Bits 6:5 – CHSEL [1:0]: вибір режиму роботи ЦАП (табл. 11.1). Ці розряди програмують режими роботи ЦАП: одно- або двоканальний;

– Bits 4:2 – зарезервовано;

– Bit 1 – CH1TRIG: режим автоматичного перетворення каналу 1. Якщо розряд встановлено в одиницю, то подія, яку встановлено у регістрі EVCTRL, ініціює перетворення за умови, що дані у регістрі CH1DATA ще не перетворено;

– Bit 0 – CH0TRIG: режим автоматичного перетворення каналу 0. Якщо розряд встановлено в одиницю, то подія, яку встановлено у регістрі EVCTRL, ініціює перетворення за умови, що дані у регістрі CH0DATA ще не перетворено.

Таблиця 11.1. **Вибір режиму роботи ЦАП**

CHSEL [1:0]	Режим роботи
00	Одноканальний (працює тільки канал 0)
01	Зарезервовано
10	Двоканальний (вибірка/зберігання для каналів 0 та 1)
11	Зарезервовано

CTRLC-регістр керування C (рис. 11.11)

Bit	7	6	5	4	3	2	1	0	
+0x02	-	-	-	REFSEL[1:0]		-	-	LEFTADJ	CTRLC
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	

Рис. 11.11. CTRLC-регістр керування C

– Bits 7:5 – зарезервовано;

– Bits 4:3 − REFSEL [1:0] : вибір ДОН. Джерело опорної напруги і, відповідно, діапазон перетворення ЦАП вибирається згідно табл. 11.2;

– Bit 2:1 – зарезервовано;

– Bit 0 – LEFTADJ: лівоспрямовані дані; якщо розряд встановлено в одиницю, то регістри CH0DATA і CH1DATA – лівоспрямовані.

Таблиця 11.2. **Вибір джерела опорної напруги ЦАП**

REFSEL [1:0]	**Джерело опорної напруги**
00	Вбудоване джерело 1.00В
01	AU_{cc}
10	Вивід AREF порту A
11	Вивід AREF порту B

Регістр керування подіями (рис.11.12)

Bit	7	6	5	4	3	2	1	0	
+0x03	-	-	-	-	-	EVSEL[2:0]			EVCTRL
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рис. 11.12. EVCTRL-регістр керування подіями

Bits 7:3 – зарезервовано;

– Bits 2:0 – обирають маршрут подій (табл. 11.3).

Табл*иця 11.3.* **Вибір маршруту подій ЦАП**

EVSEL[2:0]	**Маршрут подій**
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Регістр керування часовими інтервалами (рис. 11.13)

Bit	7	6	5	4	3	2	1	0	
+0x04	-	CONINTVAL[2:0]			REFRESH[3:0]				TIMCTRL
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	

Рис. 11.13. TIMCTRL-регістр керування часовими інтервалами

– Bit 7 – зарезервовано;

– Bits 6:4 – CONINTVAL[2:0]: інтервал між двома завершеними перетвореннями. Інтервал повинно бути встановлено відповідно периферійному тактовому генератору, щоб переконатися, що нові перетворення не почнуться до тих пір, поки результат попереднього перетворення не запишеться. Інтервал перетворення ЦАП не повинен бути менше ніж 1 мкс в одноканальному режимі, і не менше ніж 1,5 мкс у двоканальному режимі (табл. 11.4);

– Bits 3:0 – REFRESH [3:0]: керування часом оновлення результатів. Ці розряди контролюють часовий інтервал між оновленням результату в двоканальному режимі. Інтервал повинно бути встановлено відповідно з периферійним тактовим генератором, щоб уникнути втрати точності конвертованого значення (табл. 11.5).

Таблиця 11.4. **Доступні установки інтервалу між перетвореннями**

CONINTVAL[2:0]	**Кількість тактів між перетвореннями в одноканальному режимі**	**Кількість тактів між перетвореннями у двоканальному режимі**
0	1	1
1	2	3
10	4	6
11	8	12
100	16	24
101	32	48
110	64	96
111	128	192

Регістр статусу ЦАП (рис. 11.14)

Bit	7	6	5	4	3	2	1	0	
+0x05	-	-	-	-	-	-	CH1DRE	CH0DRE	STATUS
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рис. 11.14. STATUS-регістр

– Bits 7:2 – зарезервовано;

– Bit 1 – CH1DRE: регістр даних для каналу 1 порожній. Якщо розряд скинуто у нуль, запис у регістр даних може призвести до втрати результату перетворення. Цей розряд може використовуватись для запитів DMA [18];

– Bit 0 – CH0DRE: регістр даних для каналу 0 порожній. Якщо розряд скинуто у нуль, запис у регістр даних може призвести до втрати результату перетворення. Цей розряд може використовуватись для запитів DMA[18].

Таблиця 11.5. **Частота оновлення результатів**

REFRESH[3:0]	**Кількість тактів між оновленням результату**
0	16
1	32
10	64
11	128
100	256
101	512
110	1024
111	2048
1000	4096
1001	8192
1010	16384
1011	32768
1100	65536
1101	Зарезервовано
1110	Зарезервовано
1111	Оновлення вимкнуто

11.10.3. Регістри калібрування

Для досягнення високої точності перетворення, треба калібрувати коефіцієнти передачі і зміщення модуля ЦАП. Калібрувальні значення для корегування коефіцієнту передачі і зміщення є 7-розрядними.

Найкращі результати досягаються при калібруванні в тих же умовах, в яких планується використовувати ЦАП, тобто за одних і тих же U_{REF}, вихідному каналі, часі перетворення і інтервалі оновлення.

З урахуванням похибок, теоретичну передатну функцію ЦАП можна записати наступним чином:

$$U_{DACn} = gain \cdot U_{ref} \cdot \frac{CHnDATA}{4096} + offset,$$

де *gain* - коефіцієнт передачі, *offset* - коефіцієнт зсуву.

У ідеального ЦАП коефіцієнт передачі дорівнює 1, а коефіцієнт зміщення дорівнює 0. Калібрування відбувається за допомогою регістрів GAINCAL та OFFSETCAL (рис. 11.15, 11.16) і не залежить від вибору режиму.

Регістр калібрування підсилення ЦАП (рис. 11.15)

Bit	7	6	5	4	3	2	1	0	
+0x08	-	GAINCAL[6:0]							GAINCAL
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	

Рис. 11.15. GAINCAL-регістр калібрування підсилення ЦАП

– Bit 7 – зарезервовано;

– Bits 6:0 – GAINCAL [6:0] – корегування коефіцієнта передачі ЦАП. Ці розряди використовуються, щоб компенсувати помилку підсилення ЦАП.

Регістр калібрування зсуву ЦАП (рис. 11.16)

Bit	7	6	5	4	3	2	1	0	
+0x09	-	OFFSETCAL[6:0]							OFFSETCAL
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	

Рис. 11.16. OFFSETCAL-регістр калібрування зсуву ЦАП

– Bit 7 – зарезервовано;

– Bits 6:0 – OFFSETCAL [6:0] – корегування коефіцієнта зсуву. Ці розряди використовуються щоб компенсувати помилку зсуву ЦАП.

У табл. 11.6 наведено загальну таблицю регістрів модуля ЦАП

11.11. Характеристики ЦАП у складі мікроконтролерів AVR

Характеристики ЦАП у складі дяких мікроконтролерів AVR наведено у табл. 11.7.

Таблиця 11.6. **Загальна таблиця регістрів модуля ЦАП**

Адреса	Регістр	Розряд 7	Розряд 6	Розряд 5	Розряд 4	Розряд 3	Розряд 2	Розряд 1	Розряд 0
+0x00	**CTRLA**	-	-	-	IDOEN	CH1EN	CH0EN	-	ENABLE
+0x01	**CTRLB**	-	CHSEL[1:0]		-	-	-	CH1TRIG	CH0TRIG
+0x02	**CTRLC**	-	-	-	REFSEL[1:0]		-	-	LEFTADJ
+0x03	**EVCTRL**	-	-	-	-	-	EVSEL[2:0]		
+0x04	**TIMCTRL**	-	CONINTVAL[2:0]			REFRESH[3:0]			
+0x05	**STATUS**	-	-	-	-	-	-	CH1DRE	CH0DRE
+0x06	**Зарезервовано**	-	-	-	-	-	-	-	-
+0x07	**Зарезервовано**	-	-	-	-	-	-	-	-
+0x08	**GAINCAL**	-	GAINCAL[6:0]						
+0x09	**OFFSETCAL**	-	OFFSETCAL[6:0]						
Адреса	**Регістр**	Розряд 7	Розряд 6	Розряд 5	Розряд 4	Розряд 3	Розряд 2	Розряд 1	Розряд 0
+0x10	**Зарезервовано**	-	-	-	-	-	-	-	-
+0x11	**Зарезервовано**	-	-	-	-	-	-	-	-
+0x12	**Зарезервовано**	-	-	-	-	-	-	-	-
+0x13	**Зарезервовано**	-	-	-	-	-	-	-	-
+0x14	**Зарезервовано**	-	-	-	-	-	-	-	-
+0x15	**Зарезервовано**	-	-	-	-	-	-	-	-
+0x16	**Зарезервовано**	-	-	-	-	-	-	-	-
+0x17	**Зарезервовано**	-	-	-	-	-	-	-	-
+0x18	**CH0DATAL**	CHDATA[7:0]							
+0x19	**CH0DATAH**	-	-	-	-	CHDATA[11:8]			
+0x1A	**CH1DATAL**	CHDATA[7:0]							
+0x1B	**CH1DATAH**	-	-	-	-	CHDATA[11:8]			

Таблиця 11.7. **Характеристики ЦАП у складі мікроконтролерів AVR**

МК	Каналів	Розрядність	МК	Каналів	Розрядність
ATxmega64A1	4	12bit	ATxmega32A4	2	12bit
ATxmega128A1	4	12bit	ATxmega64A4	2	12bit
ATxmega192A1	4	12bit	ATxmega128A4	2	12bit
ATxmega256A1	4	12bit	AT90PWM3B	1	10bit
ATxmega384A1	4	12bit	AT90PWM2B	1	10bit
ATxmega64A3	2	12bit	AT90PWM81	1	10bit
ATxmega128A3	2	12bit	AT90PWM216	1	10bit
ATxmega192A3	2	12bit	AT90PWM316	1	10bit
ATxmega256A3	2	12bit	ATmega16M1	1	10bit
ATxmega256A3B	2	12bit	ATmega32M1	1	10bit
ATxmega16A4	2	12bit	ATmega64M1	1	10bit

11.12. Опис роботи та розрахунок цифро-аналогових перетворювачів на основі резисторної матриці R-2R з підсумовуванням напруг

11.12.1. Загальна характеристика

У даному підрозділі розглянуто принцип роботи та розрахунок ЦАП на основі резисторної матриці R-2R з підсумовуванням напруг, який використовується в моделі ЦАП мікроконтролерів сімействах XMega [9, 18].

В цьому перетворювачі операційний підсилювач, що підсумовує напруги, працює в режимі роботи, близькому до холостого ходу (рис. 11.17).

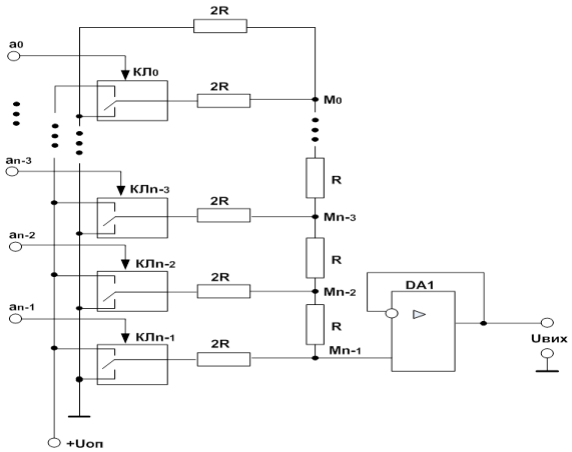


Рис. 11.17. n-розрядний ЦАП з матрицею R-2R та з підсумовуванням напруг

11.12.2. Принцип дії ЦАП з матрицею R-2R та підсумовуванням напруг

В перетворювачі в порівнянні з ЦАП, в яких виконується підсумовування струмів, використовується зворотне включення входу і виходу матриці R-2R [9].

На входи $a_0, a_1, a_2 ... a_{n-1}$ надходять цифрові сигнали, які відповідають значенню і-го розряду вхідного двійкового коду (i = 0, 1,…, n-1). Якщо на вході і-го розряду присутня логічна одиниця, то відповідний ключ КЛі переключається у верхнє положення та опорна напруга $U_{ОП}$ через резистори матриці R-2R з визначеним коефіцієнтом ділення подається на не інвертуючий вхід операційного підсилювача (ОП) DA1, де відбувається підсумовування напруг.

Якщо на вхід і-го розряду надходить логічний нуль, то ключ переключається в нижнє положення, і дана гілка матриці R-2R підключається до спільної шини.

Оскільки матриця резисторів є лінійним ланцюгом, її роботу можна проаналізувати методом суперпозиції, тобто внесок у вихідну напругу від кожного джерела (розряду) розрахувати незалежно один від одного. Внески від кожного

розряду підсумовуються на не інвертуючому вході ОП і на виході отримуємо результат у вигляді напруги.

11.12.3. Розрахунок ЦАП

Якщо, наприклад, у старшому розряді вхідного ДК присутня логічна одиниця, а в інших розрядах – логічні нулі, то ключ $Кл_{n-1}$ знаходиться у верхньому положенні і підключає гілку резисторної матриці (РМ) з резистором 2R до джерела опорної напруги $U_{ОП}$. Інші ключі знаходяться у нижньому положенні і підключають інші гілки РМ (резистори 2R) до спільної шини.

Еквівалентну схему ЦАП для цього випадку наведено на рис. 11.18*а*.

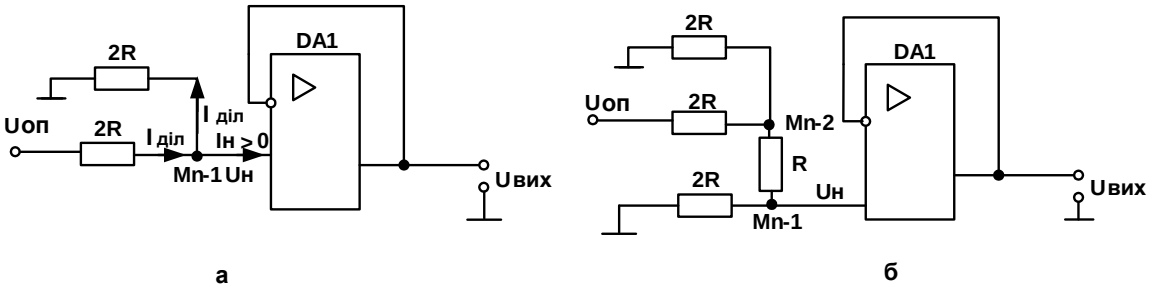


Рис. 11.18. Еквівалентні схеми ЦАП: а – при перетворенні коду 100…0В;
б – при перетворенні коду 010…0В

Еквівалентний опір РМ вище вузла M_{n-1} дорівнює 2R. Вхідний опір ОП великий та він працює в режимі, близькому до холостого ходу. Тому струм, створюваний джерелом $U_{ОП}$ протікає через два однакових резистори 2R, що утворюють дільник напруги $U_{ОП}$. У цьому випадку напруга на виході дільника визначається з виразу:

$$U_{діл} = U_{Н} = \frac{U_{ОП} \cdot 2R}{2R + 2R} = \frac{U_{ОП}}{2}. \qquad (11.1)$$

Якщо, наприклад, на вхід схеми надходить комбінація ДК: 010…0В, то в цьому випадку ключ $Кл_{n-2}$ увімкнений у верхнє положення, а інші ключі – у нижнє. Еквівалентна схема ЦАП, прийме вигляд, представлений на рис. 11.18*б*.

Резистори R і 2R, які розташовано нижче вузла M_{n-2}, включено послідовно $(R_{ВХ.DA1} \to \infty)$. Тоді замінюємо їх еквівалентним опором:

$$R + 2R = 3R. \qquad (11.2)$$

Напруга в точці M_{n-2} визначається виразом:

$$U_{M_{n-2}} = \frac{U_{\text{оп}} \cdot 2\text{R} \| 3\text{R}}{2\text{R} + 2\text{R} \| 3\text{R}} = \frac{U_{\text{оп}} \cdot {}^{6}\!/_{5} \cdot R}{2\text{R} + {}^{6}\!/_{5} \cdot R} = \frac{U_{\text{оп}} \cdot 3}{8}. \qquad (11.3)$$

Напруга у вузлі М $_{n\text{-}1}$:

$$U_{M_{n-1}} = U_{\text{н}} = \frac{U_{M_{n-2}} \cdot 2\text{R}}{\text{R} + 2\text{R}} = \frac{U_{\text{оп}}}{4}. \qquad (11.4)$$

При подачі на вхід ЦАП двійкового коду: 001…0В напруга на не інвертуючому вході ОП буде дорівнювати:

$$U_{\text{Н}} = \frac{U_{\text{ОП}}}{8}. \qquad (11.5)$$

При надходженні коду: 00...01 В напруга:

$$U_{\text{Н}} = \frac{U_{\text{ОП}}}{2^n}. \qquad (11.6)$$

Оскільки коефіцієнт передачі підсумовуючого операційного підсилювача $K_{\text{U.ІМС ОП}} = 1$[9], то вираз для визначення сумарної вихідної напруги від дії одиниць у всіх розрядах вхідного ДК прийме вигляд:

$$U_{\text{ВИХ max}} = U_{\text{ОП}}\left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + ... + \frac{1}{2^n}\right) = \frac{\text{U}_{\text{ОП}}}{2^n}\sum_{i=0}^{n-1} 2^i. \qquad (11.7)$$

Якщо позначити значення і-х розрядів вхідного ДК як a_i, де a_i дорівнює 0 чи 1, то останній вираз перетвориться до вигляду:

$$U_{\text{ВИХ}} = \frac{\text{U}_{\text{ОП}}}{2^n}\sum_{i=0}^{n-1} a_i \cdot 2^i. \qquad (11.8)$$

Співмножник $\sum_{\text{i}=0}^{\text{n}-1} \text{a}_\text{i} \cdot 2^\text{i}$ є десятковим еквівалентом вхідного двійкового коду (представляє десятковий еквівалент вхідного цифрового коду).

Розглянутий перетворювач називають помножуючим, тому що вихідна напруга пропорційна добутку значення опорного сигналу $U_{\text{ОП}}$ на значення вхідного цифрового коду.

Коефіцієнт передачі, тобто розрахункове збільшення вихідної напруги при зміні вхідного коду на одиницю молодшого розряду (ціна молодшого значущого розряду) складає:

$$K_{\text{ЦАП}} = \frac{U_{\text{ОП}}}{2^{n}} \left[\frac{\text{В}}{\text{МЗР}} \right]. \qquad (11.9)$$

Якщо в якості $U_{ОП}$ для модуля ЦАП мікроконтролера AVR використати зовнішню напругу, що подається на вивід AREF порту A або B та дорівнює 4,96В = 4096mВ, то коефіцієнт передачі

$$K_{\text{ЦАП}} = 4096/2^{10} = 4096/1024 = 4 \text{ mВ/МЗР},$$

а максимальне значення напруги на виході ЦАП, коли у всіх розрядах вхідного коду присутні одиниці:

$$U_{\text{ВИХ.max}} = 1023*4 = 4092\text{mВ} = 4{,}092\text{В}.$$

Контрольні запитання та завдання

1. Яку роздільну здатність має ЦАП у складі мікроконтролерів XMega?
2. Які джерела опорної напруги можуть бути використані для модуля ЦАП?
3. Наведіть вираз, що пов'язує вихідну напругу ЦАП зі значенням, яке записано у регістр даних.
4. Де може бути використано аналоговий сигнал, отриманий у результаті перетворення?
5. Для чого може бути використано режим запуску мікроконтролера за настанням події?
6. Завдяки якому пристрою забезпечується двоканальне функціонування ЦАП?
7. Які часові обмеження накладаються на роботу ЦАП?
8. Як досягається режим енергозберігання модулю ЦАП і які наслідки має використання цього режиму?
9. Навіщо і як виконується калібрування ЦАП?
10. Чому дорівнює коефіцієнт передачі ЦАП?
11. Чому дорівнює максимальне значення напруги на виході ЦАП, коли у всіх розрядах вхідного коду присутні одиниці?
12. Чому дорівнює коефіцієнт передачі операційного підсилювача у схемі ЦАП з підсумовуванням напруг?

12. ПРОЕКТУВАННЯ МОДУЛЯ АНАЛОГОВОГО КОМПАРАТОРА

12.1. Загальні відомості

Компаратором називається електронний пристрій, який призначено для порівняння двох напруг. В залежності від форми представлення порівнюваних сигналів компаратори поділяються на:

– аналогові компаратори (АК);
– цифрові компаратори (ЦК) [9].

Нижче описано АК, які призначено для порівняння двох аналогових напруг, одна з яких виконує функцію еталонної – U_{ET} , а інша порівнюється з еталонною – U_X .

Зазвичай АК включає власне аналоговий компаратор (ВАК) і схему формування рівнів (СФР) (рис. 12.1).

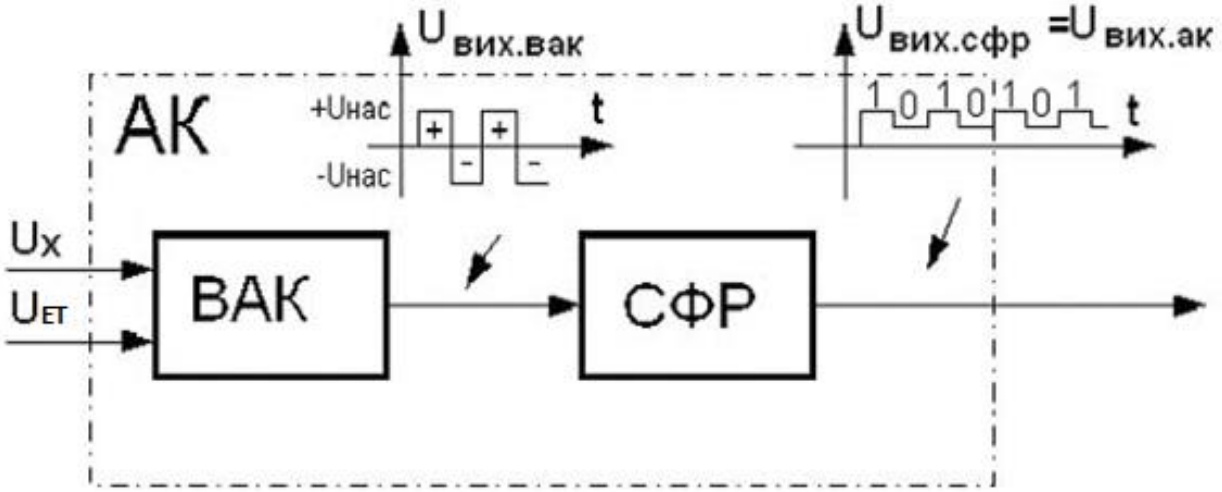


Рис. 12.1. Структура аналогового компаратора

Власне аналоговий компаратор виконує порівняння двох напруг: U_{ET} і U_X і формує на виході дискретний сигнал, що приймає одне з двох значень: $+U_{НАС}$; $-U_{НАС}$ [9].

Схема формування рівнів перетворює значення: $+U_{НАС}$; $-U_{НАС}$ у рівні цифрових сигналів ТТЛШ/КМОН-схем (ТТЛШ – транзисторно-транзисторна логіка з переходами Шоттки, КМОН – комплементарний метал-оксид-напівпровідник).

Значення $+U_{НАС}$ перетворюється в рівень логічної одиниці, а значення -$U_{НАС}$ – у рівень логічного нуля.

В електроніці застосовуються два варіанти схемного виконання АК:

– на інтегральній мікросхемі операційного підсилювача (ІМС ОП);
– на спеціалізованій мікросхемі аналогового компаратора.

12.2. Аналогові компаратори на інтегральній мікросхемі операційного підсилювача

Розглянемо схему АК, що виконує порівняння двох додатних напруг (рис. 12.2*а*) [9]. Одна напруга є еталонною: $(U_{ET} = const)$, а друга: (U_X) – повільно змінюється за законом, представленим на рис. 12.2*в*.

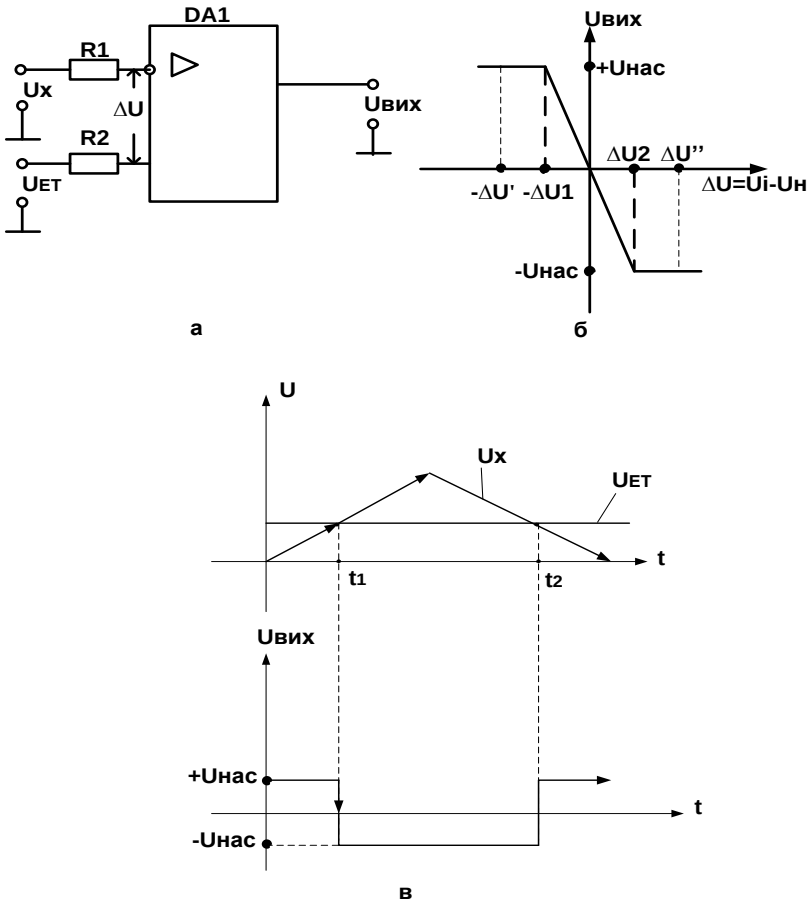


Рис. 12.2. АК для порівняння однополярних напруг: а – схема; б – передатна характеристика; в – часові діаграми

До моменту часу $t = t_1$ напруга $U_X < U_{ET}$. Потенціал входу ІМС ОП, що не інвертує (*Н)*, більш додатний, ніж потенціал входу, що інвертує (*І)*.

У цьому випадку різниця напруг між входами *І* та *Н*

$$\left|-\Delta U'\right| = \left|(U_I - U_H)\right| > \left|-\Delta U_1\right|. \qquad (12.1)$$

Відповідно до передатної характеристики ІМС ОП (рис. 12.2*б*) у цьому випадку $U_{\text{ВИХ}} = +U_{\text{НАС}}$. Після моменту часу $t = t_1$ вхідний сигнал $U_{\text{Х}} \geq U_{\text{ЕТ}}$. З'являється різниця напруг між входами

$$\Delta U'' = (U_I - U_H) > \Delta U_2 . \qquad (12.2)$$

Відповідно до передатної характеристики ІМС ОП (рис. 12.2*б*) у цьому випадку $U_{\text{ВИХ}} = -U_{\text{НАС}}$.

На рис. 12.2*в* переключення АК відбувається миттєво, що є ідеальним випадком. Реально має місце невелика затримка між моментом досягнення рівності сигналів $U_{\text{Х}}$ та $U_{\text{ЕТ}}$ і моментом, коли вихідний сигнал $U_{\text{ВИХ}}$ починає зменшуватися. Крім того, лінія, що відображає зменшення $U_{\text{ВИХ}}$, йде не перпендикулярно вісі часу, а під невеликим нахилом.

Після моменту часу $t = t_2$ напруга $U_{\text{ЕТ}}$ знову стає більшою ніж $U_{\text{Х}}$, вихідна напруга змінюється і приймає значення: $+U_{\text{НАС}}$.

12.3. Аналоговий компаратор у складі AVR-мікроконтролерів

12.3.1. Загальні відомості

До складу мікроконтролерів сімейства AVR, входить модуль аналогового компаратора [1; 7]. Будучи увімкненим, компаратор дає можливість порівнювати значення напруг на двох відповідних виводах мікроконтролера. Результатом порівняння є логічне значення нуль або одиниця, що може бути прочитано з програми. За результатом порівняння може бути згенероване переривання, а також відбутися захоплення стану таймера/лічильника T1. Остання функція дозволяє вимірювати тривалості аналогових сигналів. Виводи, що використовуються компаратором, є контактами портів введення/виведення загального призначення (табл. 12.1).

Щоб вони могли використовуватися аналоговим компаратором, їх треба запрограмувати як входи (відповідний розряд регістра DDRx скинути у нуль). Крім того, записом нуля у відповідний розряд регістра PORT*x*n необхідно відключити внутрішні підтягуючі резистори.

У деяких мікроконтролерів є можливість відключення вхідних цифрових буферів у випадку, якщо контакти мікроконтролера, що відповідають виводам AIN0 та AIN1, використовується тільки для введення аналогових сигналів [1; 7]. При відключених цифрових буферах зменшується загальний струм споживання мікроконтролера, а відповідні розряди регістра PINx завжди читаються як нуль.

Таблиця 12.1. **Виводи, що використовуються аналоговим компаратором**

Назва	**ATmega8x**	**ATmega8515x/8535x**	**ATmega16x/32x**	**ATmega64x/128x/165x**	**ATmega162x/163x**	**ATmega48x/88x/168x**	**ATmega164x/324x/644x**	**ATmega325x/3250x/645x /6450x**	**ATmega640x/1280x/1281x /2560x/2561x**	**Призначення**
AIN0	PD6	PB2	PB2	PE2	PB2	PD6	PB2	PE2	PE2	Не інвертуючий вхід
AIN1	PD7	PB3	PB3	PB3	PB3	PD7	PB3	PB3	PE3	Інвертуючий вхід

Відключення цифрових буферів на входах AIN0 та AIN1 здійснюється записом одиниці відповідно у розряди AIN0D та AIN1D регістра DIDR1, розташованого за адресою ($7F). Формат цього регістра наведено на рис. 12.3.

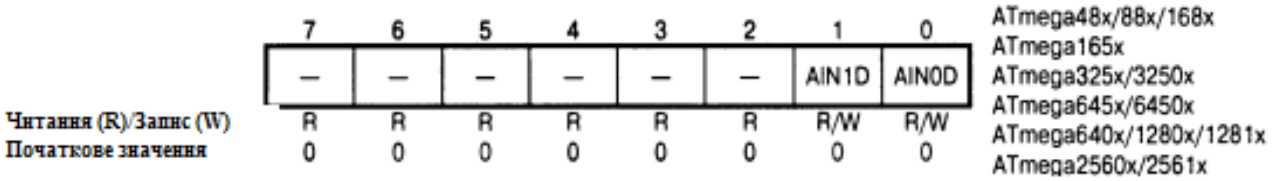


Рис. 12.3. Формат регістра DIDR1

12.3.2. Функціонування та програмування компаратора

Структурну схему аналогового компаратора приведено на рис. 12.4.

Керування компаратором і контроль його стану здійснюється за допомогою регістра ACSR, який в залежності від моделі розташовано за адресою $08 ($28) або $30 ($50) (табл. 12.2). Формат цього регістра приведено на рис. 12.5, а призначення розрядів коротко описано в табл. 12.3.

Модуль АК AVR-мікроконтролера є звичайним компаратором. Якщо напруга на виводі AIN0 (не інвертуючий вхід) більш додатна (менш від'ємна) напруги на виводі AIN1 (інвертуючий вхід), то результат порівняння буде

дорівнювати одиниці. У іншому випадку результат порівняння буде дорівнювати нулю. Цей результат (стан виходу компаратора) зберігається в розряді ACO регістра ACSR.

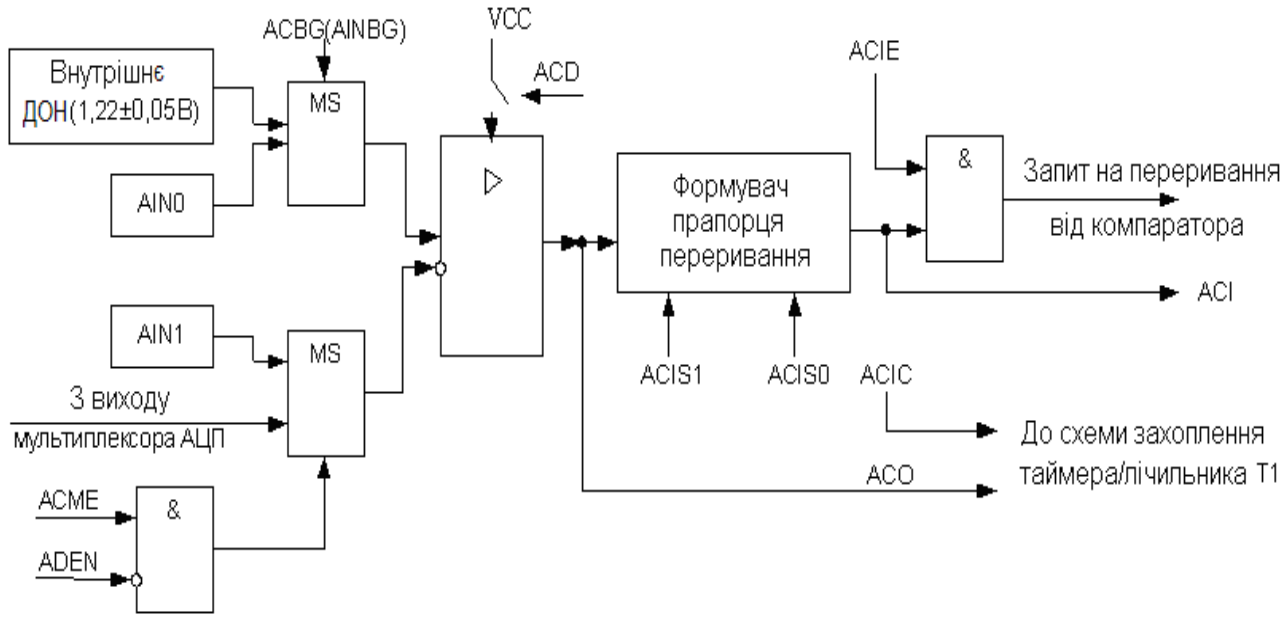


Рис. 12.4. Структурна схема аналогового компаратора

	7	6	5	4	3	2	1	0	
	ACD	AINBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	ATmega161x
Зчитування(R)/Запис(W)	R/W	R	R	R/W	R/W	R/W	R/W	R/W	
Початкове значення	0	0	N/A	0	0	0	0	0	

	7	6	5	4	3	2	1	0	
	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	Інші моделі
Зчитування(R)/Запис(W)	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Початкове значення	0	0	N/A	0	0	0	0	0	

Рис. 12.5. Формат регістра ACSR

Таблиця 12.2. **Адреса регістра ACSR**

Модель	**Адреса регістра**
ATmega8515x/8535x	$08 ($28)

Закінчення табл. 12.2.

Модель	**Адреса регістра**
ATmega8x/16x/32x/64x/128x	
ATmega48x/88x/168x	$30 ($50)
ATmega162x	$08 ($28)
ATmega164x/324x/644x	$30 ($50)
ATmega165x	
ATmega325x/3250x/645x/6450x	
ATmega640x/1280x/1281x/2560x/2561x	

Таблиця 12.3. **Призначення розрядів регістра ACSR**

Розряд	**Назва**	**Опис**
7	ACD	Вимикання компаратора (нуль – ввімкнено, одиниця – вимкнено)
6	ACBG (AINBG*)	Підключення до не інвертуючого входу компаратора внутрішнього ДОН (нуль – не підключено, одиниця – підключено)
5	ACO	Результат порівняння (вихід компаратора)
4	ACI	Прапорець переривання від компаратора
4	ACIE	Дозвіл переривання від компаратора
2	ACIC	Підключення компаратора до схеми захоплення таймера/лічильника T1 (одиниця – підключено, нуль – не підключено)
1, 0	ACIS1:ACIS0	Умова виникнення переривання від компаратора
*В моделях ATmega161x		

За вмикання і вимикання компаратора відповідає розряд ACD. Оскільки при подачі напруги живлення всі розряди регістра ACSR скидаються в нуль, компаратор вмикається автоматично. Для вимикання компаратора розряд ACD слід встановити в одиницю. При зміні стану цього розряду переривання від компаратора слід заборонити.

Якщо стан виходу компаратора (розряд ACO) змінився заданим чином, встановлюється прапорець переривання ACIF регістра ACSR і генерується запит на переривання, якщо воно дозволено.

Як і для інших переривань, цей прапорець скидається апаратно при запуску підпрограми обробки переривання або програмно, записом у нього одиниці. Для дозволу переривання необхідно встановити в одиницю розряд ACI регістра ACSR і прапорець I регістра SREG. Яка саме зміна стану виходу компаратора викликає переривання, визначається станом розрядів ACIS1:ACIS0 регістра ACSR відповідно до табл. 12.4.

Таблиця 12.4. **Умови генерації запиту на переривання від компаратора**

ACIS1	ACIS0	Умова
0	0	Будь-яка зміна стану виходу компаратора
0	1	Зарезервовано
1	0	Зміна стану виходу компаратора з одиниці на нуль
1	1	Зміна стану виходу компаратора з нуля на одиницю

При програмуванні цих розрядів переривання від компаратора повинно бути заборонено.

Крім генерації переривання, компаратор може керувати схемою захоплення таймера/лічильника T1. Для цього необхідно встановити в одиницю розряд ACIC регістра ACSR. В результаті вихід компаратора підключиться до схеми захоплення замість виводу ICP1 мікроконтролера. Якщо розряд ACIC скинуто у нуль, компаратор відключено від блоку захоплення таймера/лічильника T1. Компаратор може порівнювати сигнали, не тільки на виводах AIN0 і AIN1. Замість виводу AIN0 мікроконтролера до не інвертуючого входу компаратора може бути підключено внутрішнє джерело опорної напруги (ДОН) величиною 1.22 (±0.05) В. Для цього необхідно встановити в одиницю розряд ACBG (AINBG у моделях ATmega161x) регістра ACSR.

На інвертуючий вхід компаратора може надходити сигнал з виходу мультиплексора модуля АЦП. Для цього треба встановити в одиницю розряд ACME, який розташовано, в залежності від моделі (табл. 12.5), або в регістрі спеціальних функцій SFIOR (3-й розряд регістра), або в регістрі керування АЦП ADCSRB (6-й розряд) [1; 7]. Модуль АЦП при цьому має бути вимкнено (розряд ADEN регістра ADCSRA треба скинути в нуль).

Таблиця 12.5. **Підключення мультиплексора АЦП до компаратора**

Модель	**Регістр**	**Адреса**
ATmega8515x/8535x	SFIOR	$30 ($50)
Модель	Регістр	Адреса
ATmega8x/16x/32x/64x/128x	SFIOR	
ATmega48x/88x/168x		$20 ($40)
ATmega162x	ADCSRB	($7B)
ATmega164x/324x/644x		
ATmega165x		
ATmega325x/3250x/645x/6450x		
ATmega640x/1280x/1281x/2560x/2561x		

Основні параметри аналогового компаратора представлено в [7].

Контрольні запитання та завдання

1. Який електронний пристрій називається компаратором?
2. На які види поділяються компаратори в залежності від форми представлення порівнюваних сигналів?
3. Який електронний пристрій виконує функцію власне аналогового компаратора?
4. Поясніть роботу АК, який виконано на ІМС ОП.
5. Поясніть призначення схеми формування рівнів.
6. Наведіть та поясніть структуру модуля аналогового компаратора.
7. Наведіть та поясніть формати керуючих регістрів модуля АК.
8. Яким чином можна підключити мультиплексор АЦП до компаратора?
9. Від чого залежить стан виходу компаратора?
10. Де зберігається стан виходу компаратора?
11. Назвіть умови генерації запиту на переривання від компаратора.
12. Як повинні бути сконфігуровані виводи, які використовуються аналоговим компаратором?
13. Які виводи мікроконтролера використовуються компаратором?
14. Який розряд відповідає за вмикання та вимикання компаратора?
15. Як вихід компаратора підключити до схеми захоплення таймера/лічильника T1?

13. СПЕЦІАЛЬНІ РЕЖИМИ РОБОТИ МІКРОКОНТРОЛЕРА

13.1. Тактування

13.1.1. Загальні відомості

Нижче розглянуто виконання тактування AVR-мікроконтролерів на прикладі сімейства Mega [1; 7]. Тактові сигнали формує пристрій синхронізації, спрощену структуру якого наведено на рис. 13.1. Блок попереднього дільника і можливість підключення зовнішнього RC-ланцюга є не у всіх моделях.

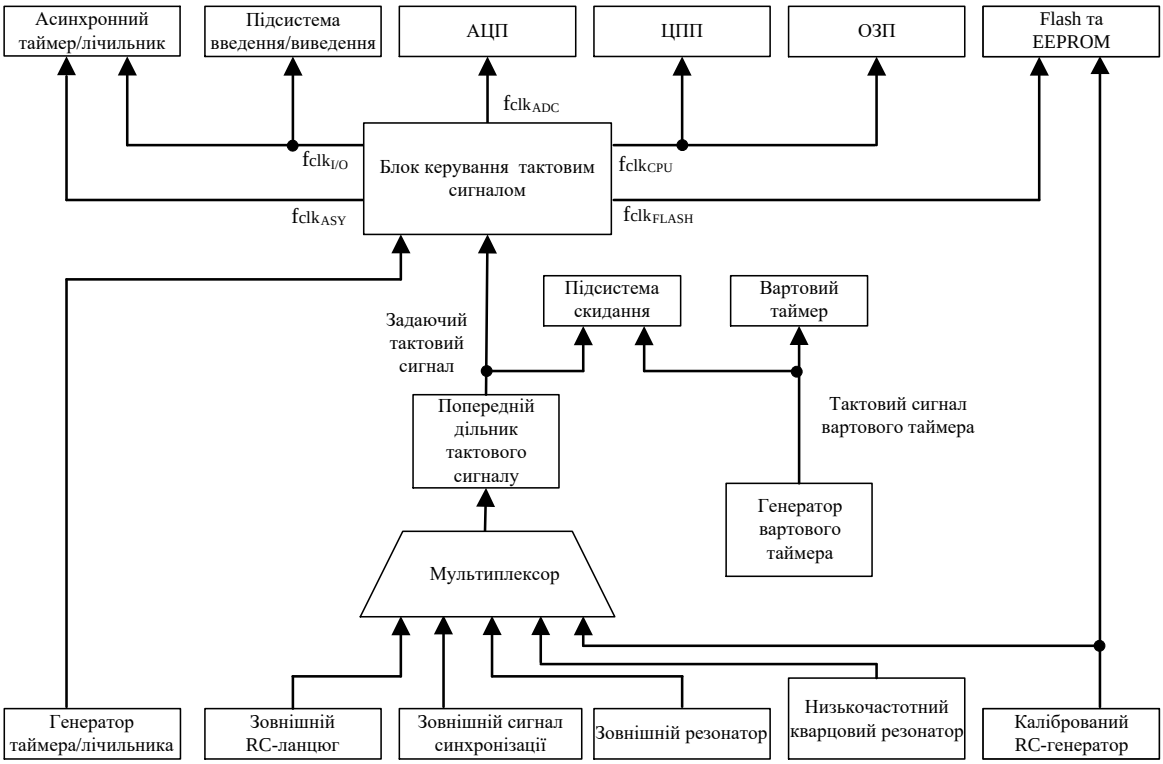


Рис. 13.1. Спрощена структура пристрою синхронізації

Для тактування можуть використовуватися різні джерела тактового сигналу. Перш за все, це вбудований генератор з під’єднуваним зовнішнім кварцовим/керамічним резонатором. В якості тактового може використовуватися найпростіший RC-генератор – як з внутрішнім (каліброваним), так і з зовнішнім RC-ланцюгом. Крім того, в якості тактового може використовуватися сигнал від зовнішнього джерела.

Можливість використання того чи іншого джерела тактового сигналу залежить від моделі мікроконтролера (табл. 13.1).

Таблиця 13.1. **Джерела тактового сигналу**

Джерело тактового сигналу	**ATmega8x/16x/32x/ 64x/128x**	**ATmega8515x**	**ATmega8535x**	**ATmega48x/88x/168x**	**ATmega162x**	**ATmega164x/324x/ 644x**	**ATmega165x**	**ATmega325x/3250x, ATmega645x/6450x**	**ATmega640x, ATmega1280x/1281x**
Кварцовий генератор	•	•	•	•	•	•	•	•	•
Генератор з зовнішнім RC-ланцюгом	•	•	•						
Внутрішній RC-генератор	•	•	•	•	•	•	•	•	•
Зовнішній сигнал синхронізації	•	•	•	•	•	•	•	•	•

На базі системного тактового сигналу формуються додаткові сигнали, що використовуються для тактування різних модулів і блоків мікроконтролера (рис. 13.1, див. вище):

– f_{clkCPU} – тактовий сигнал центрального процесора. Використовується для тактування блоків мікроконтролера, що відповідають за роботу з ядром мікроконтролера: регістровий файл, пам’ять даних і т. ін.. При відключенні цього сигналу центральний процесорний пристрій (ЦПП) зупиняється і відповідно припиняються всі обчислення;

– $f_{clkI/O}$ – тактовий сигнал підсистеми введення/виведення. Використовується більшістю периферійних пристроїв, таких як таймери/лічильники та інтерфейсні модулі. Цей сигнал використовується також підсистемою зовнішніх переривань, проте ряд зовнішніх переривань може генеруватися і за його відсутності;

– $f_{clkFLASH}$ – тактовий сигнал для Flash-пам’яті програм. Як правило, цей сигнал активується та деактивується одночасно з тактовим сигналом центрального процесора f_{clkCPU};

– f_{clkASY} – тактовий сигнал асинхронного таймера/лічильника. Тактування здійснюється безпосередньо від зовнішнього кварцового резонатора, частотою 32768 Гц. Наявність цього сигналу дозволяє використовувати відповідний таймер/лічильник в якості годинника реального часу, навіть при знаходженні мікроконтролера у «Сплячому» режимі;

– f_{clkADC} – тактовий сигнал для модуля АЦП. Наявність цього тактового сигналу дозволяє здійснювати перетворення при зупинених ЦПП і підсистемі введення/виведення. При цьому значно зменшується рівень завад, що генеруються мікроконтролером, і відповідно збільшується точність перетворення.

Оскільки архітектура мікроконтролерів повністю статична, мінімально допустиму частоту нічим не обмежено (аж до покрокового режиму роботи), а максимальна робоча частота визначається конкретною моделлю мікроконтролера.

Вибір режиму роботи тактового генератора здійснюється програмуванням конфігураційних комірок (FUSE Bits) CKSEL3...0. Необхідні значення для кожного режиму роботи наведено у [1; 7].

13.1.2. Генератор із зовнішнім резонатором

Резонатор підключається до виводів XTAL1 і XTAL2 мікроконтролерів, як показано на рис. 13.2.

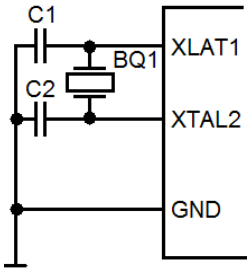


Рис. 13.2. Підключення кварцового чи керамічного резонатора

Ці виводи є відповідно входом і виходом інвертуючого підсилювача тактового генератора, який вбудовано у мікроконтролер.

Конденсатори C1 і C2, які призначено для підвищення стабільності роботи генератора, підключаються між виводами резонатора та спільним проводом. Їх ємності залежать від типу резонатора. Для кварцових резонаторів ємності цих конденсаторів зазвичай знаходяться в межах 12...22 пФ, а для керамічних повинні вибиратися відповідно до рекомендацій виробників резонаторів. Конфігурування тактового генератора із зовнішнім резонатором в різних моделях розглядається у [1; 7].

13.1.3. Низькочастотний кварцовий генератор

Цей режим призначено для використання низькочастотного кварцового резонатора, так званого «годинникового кварцу», частотою 32768 Гц. Як і інші зовнішні резонатори, він підключається до виводів TOSC1 та TOSC2 мікроконтролерів.

У деяких моделях є внутрішні конденсатори, які можна підключити між виводами резонатора й спільним проводом [1; 7]. Ємність кожного з конденсаторів

становить 10 пФ. В інших моделях необхідно використовувати зовнішні конденсатори.

13.1.4. Зовнішній сигнал синхронізації

Сигнал від зовнішнього джерела подається на вивід XTAL1, як показано на рис. 13.3.

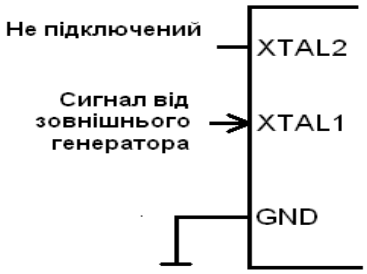


Рис. 13.3. Підключення зовнішнього джерела тактового сигналу

Цей сигнал повинен задовольняти вимогам мікроконтролера за частотою, шпаруватістю та рівнями напруги. Вивід XTAL2 залишають непідключеним.

У деяких моделях між виводом XTAL1 і спільним проводом можна включити внутрішній конденсатор ємністю 36 пФ [1; 7].

13.1.5. Генератор з зовнішнім RC-ланцюгом

При реалізації додатків, які не потребують високої часової точності, можна використовувати найпростіший RC-генератор. При цьому зовнішній RC-ланцюг підключається до виводу XTAL1, як показано на рис. 13.4.

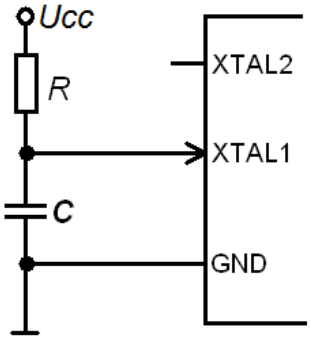


Рис. 13.4. Підключення зовнішнього RC-ланцюга

Ємність конденсатора ланцюга повинна бути не менше 22 пФ, а опір резистора рекомендується вибирати з діапазону 3.3...100 кОм. Орієнтовно частоту сигналу генератора можна розрахувати за формулою [1; 7]:

$$f = \frac{1}{3 \cdot \mathrm{RC}}.$$

Зовнішній конденсатор в цих моделях можна виключити, задіявши внутрішній, ємністю 36 пФ. Як і у випадку кварцового генератора, при використанні зовнішнього RC-ланцюжка тактовий генератор може працювати в чотирьох різних режимах, кожен з яких оптимізовано для певного діапазону частот.

13.1.6. Внутрішній калібрований RC-генератор

Використання вбудованого RC-генератора з внутрішнім часозадаючим RC-ланцюгом (внутрішнього RC-генератора) є найбільш економічним рішенням, так як при цьому не потрібні ніякі зовнішні компоненти.

Номінальні частоти внутрішнього RC-генератора для деяких моделей наведено в табл. 13.2.

Таблиця 13.2. **Номінальні частоти внутрішнього RC-генератора**

Модель	Частота [МГц]	Зауваження
ATmega8515x/8535x	1.0; 2.0; 4.0; 8.0	При Ucc = 5.0 В, T = +25°C
ATmega8x/16x/32x/64x/128x	1.0; 2.0; 4.0; 8.0	
ATmega48x/88x/168x	8.0	При Ucc = 5.0 В, T = +25°C
ATmegal62x	8.0	
ATmega 164x/324x/644x	8.0	
ATmega 165x/325x/3250x/645x/6450x	8.0	
ATmega640x/1280x/1281x/2560x/2561x	8.0	

Як видно з таблиці, в деяких моделях внутрішній RC-генератор може працювати на декількох фіксованих частотах. Робоча частота генератора цих моделей визначається вмістом конфігураційних комірок CKSEL3...0 [1; 7].

13.1.7. Керування тактовою частотою

У деяких моделях сімейства є можливість програмного зменшення частоти сигналу, що надходить від тактового генератора. Зрозуміло, що одночасно зі зменшенням тактової частоти зменшуються частоти сигналів f_{clkCPU}, $f_{clkI/0}$, $f_{clkFLASH}$, f_{c1kADC}, тобто сповільнюється робота всіх периферійних пристроїв мікроконтролера. Якщо асинхронний таймер/лічильник працює в синхронному режимі, то відповідним чином змінюється і частота сигналу f_{clkASY}.

Для керування попереднім дільником тактового сигналу використовується один з регістрів введення/виведення. Назва цього регістра, його адреса, та призначення окремих розрядів для різних моделей наведено в [1; 7].

13.2. Режим зниженого енергоспоживання

13.2.1. Загальні відомості

AVR-мікроконтролери мають декілька режимів зниженого енергоспоживання, що називають SLEEP-режимом. Кожен з цих режимів дозволяє знизити енергоспоживання мікроконтролера за рахунок відключення виконання окремих робочих функцій. Вхід в будь-який зі «сплячих» режимів виконується командою SLEEP. При виході мікроконтролера зі SLEEP-режиму виконання програми продовжується з місця зупинки [1; 7].

Режими зниженого енергоживлення надають користувачеві широкі можливості щодо зменшення струму, який споживається кристалом. При переході у SLEEP-режим виконання програми припиняється, а поновлюється при настанні певних подій. У нових моделях, крім того, передбачено зниження енергоспоживання кристала відключенням тактових сигналів незадіяних периферійних модулів.

13.2.2. Керування режимами зниженого енергоспоживання

Мікроконтролери сімейства MEGA підтримують від 3 до 6 режимів зниженого енергоспоживання (табл. 13.3).

Режими відрізняються числом периферійних пристроїв мікроконтролера, функціонуючих під час «сну», і відповідно ступенем зменшення енергоспоживання.

Залежно від моделі для керування «сплячим» режимом використовується різне число регістрів введення/виведення, які наведено в табл. 13.4.

Формати цих регістрів наведено у [1; 7].

Для керування «сплячим» режимом в мікроконтролерах сімейства, в залежності від моделі, використовується два або чотири розряди керуючих регістрів. Призначення цих розрядів наведено в табл. 13.5

Перехід в будь-який з режимів зниженого споживання здійснюється командою SLEEP. При цьому прапорець SE повинний бути встановлений в одиницю. Щоб уникнути ненавмисного перемикання мікроконтролера у «сплячий» режим рекомендується встановлювати цей прапорець безпосередньо перед виконанням команди SLEEP. Режим, в який перейде мікроконтролер після виконання команди SLEEP, визначається станом розрядів SM2...SM0.

Відповідність між вмістом цих розрядів і режимом зниженого енергоспоживання наведено в табл. 13.6.

Таблиця 13.3. **Режими зниженого енергоспоживання**

Режими зниженого енергоспоживання	**ATmega8515x**	**ATmega8535x**	**ATmega8x**	**ATmegal6x/32x/64x/128x**	**ATmega48x/88x/168x**	**ATmegal62x**	**ATmegal64x/324x/644x**	**ATmegal65x, ATmega325x/3250x, ATmega645x/6450x**	**ATmega640x, ATmegal280x/1281x, ATmega2560x/2561x**
Idle	•	•	•	•	•	•	•	•	•
ADC Noise Reduction		•	•	•	•		•	•	•
Power Down	•	•	•	•	•	•	•	•	•
Power Save		•	•	•	•	•	•	•	•
Standby	•	•	•	•	•	•	•	•	•
Extended Standby		•		•		•	•		•

Таблиця 13.4. **Регістри для керування «сплячим» режимом**

Назва	**Опис**	**Адреса**	**Модель**
MCUCR	Регістр керування мікроконтролера	$35($55)	ATmega8x/16x/32x/64x/128x, ATmega8535x
MCUCR	Регістр керування мікроконтролера	$35($55)	ATmega8515x, ATmegal62x
MCUCSR	Регістр керування і стану мікроконтролера	$34($54)	
EMCUCR	Додатковий регістр керування мікроконтролера	$36($56)	
SMCR	Регістр керування сплячим режимом	$33($53)	ATmega48x/88x/168x, ATmega 164x/324x/644x/l 65x, ATmega325x/3250x/645x/6450x, ATmega640x, ATmega1280x/1281x/2560x/2561x

.

Таблиця 13.5. **Призначення розрядів регістрів для керування «сплячим» режимом**

Розряд	Опис
SE	**Дозвіл переходу в режим зниженого енергоспоживання**. Встановлення цього розряду в одиницю дозволяє переведення мікроконтролера в режим зниженого енергоспоживання. Перемикання здійснюється за командою SLEEP. При скинутому розряді SE виконання команди не робить ніяких дій.
SM2...SM0	**Вибір режиму зниженого енергоспоживання**. Стан цих розрядів визначає в який режим перейде мікроконтролер після виконання команди SLEEP (таблиця 13.6).

Таблиця 13.6. **Вибір режиму зниженого енергоспоживання**

SM2	SM1	SM0	Режими
0	0	0	Idle
0	0	1	ADC Noise Reduction
0	1	0	Power Down
0	1	1	Power Save
1	0	0	Зарезервовано
1	0	1	Зарезервовано
1	1	0	Standby[1)]
1	1	1	Extended Standby[1)]
[1)] Ці режими можна використовувати тільки при роботі із зовнішнім резонатором			

Наявність того чи іншого режиму в конкретній моделі можна визначити за табл. 13.3, див. вище. При відсутності в конкретної моделі мікроконтролера будь-якого з режимів значення розрядів SM2...SM0, відповідні цьому режиму, є зарезервованими.

Вихід зі «сплячого» режиму може бути здійснено:

- у результаті переривання. При генерації переривання мікроконтролер переходить у робочий режим, зупиняється на 4 такти, виконує підпрограму обробки переривання і відновлює виконання програми з інструкції, наступної за командою SLEEP. Вміст РЗП, ОЗП і РВВ при цьому не змінюється;
- у результаті скидання. Після переходу мікроконтролера в робочий режим керування передається за адресою вектора скидання.

Особливості окремих режимів зниженого енергоспоживання та рекомендації по їх застосуванню розглянуто у [1; 7].

13.3. Скидання

Переведення роботи мікроконтролера у початковий стан виконується шляхом так званого «скидання». Останнє може бути викликано наступними подіями:

- включення напруги живлення мікроконтролера;
- подача сигналу НИЗЬКОГО рівня на вивід RESET (апаратне скидання);
- тайм-аут вартового таймера;
- падіння напруги живлення нижче заданої величини;
- скидання за інтерфейсом JTAG.

При настанні будь-якої з перерахованих вище подій у всі регістри введення/виведення заносяться їх початкові значення, а в лічильник команд завантажується значення адреси вектора скидання. За цією адресою має знаходитись одна з команд безумовного переходу RJMP або JMP. Значення адреси вектора скидання визначається станом конфігураційних комірок BOOTSZ1 і BOOTSZ0 [1; 7].

Структурну схему підсистеми скидання наведено на рис. 13.5.

У деяких моделей відсутні елементи, які на рисунку виділено пунктиром.

Схема скидання працює наступним чином. При настанні події, що приводить до скидання мікроконтролера, формується внутрішній сигнал скидання. Одночасно запускається таймер формування затримки скидання. Після закінчення певного проміжку часу внутрішній сигнал скидання знімається і починається виконання програми.

Всі мікроконтролери сімейства дозволяють визначити подію, в результаті якої відбулося скидання пристрою. В залежності від моделі (табл. 13.7) для цього використовується або регістр керування і стану мікроконтролера MCUCSR, який розташовано за адресою $34 ($54), або регістр стану мікроконтролера MCUSR, розташований за тією ж адресою.

Наведені в табл. 13.7 регістри містять набір прапорів, стан яких залежить від події, що викликала скидання пристрою. Формат регістрів MCUSR і MCUCSR, опис прапорців, які використовуються для визначення джерела скидання та опис окремих видів скидання наведено в [1; 7].

13.4. Підсистема переривань

13.4.1. Загальні відомості

Для виконання пріоритетної задачі, що обумовлена внутрішньою або зовнішньою подією, у МПС використовується підсистема переривань, яку мають всі сучасні мікроконтролери.

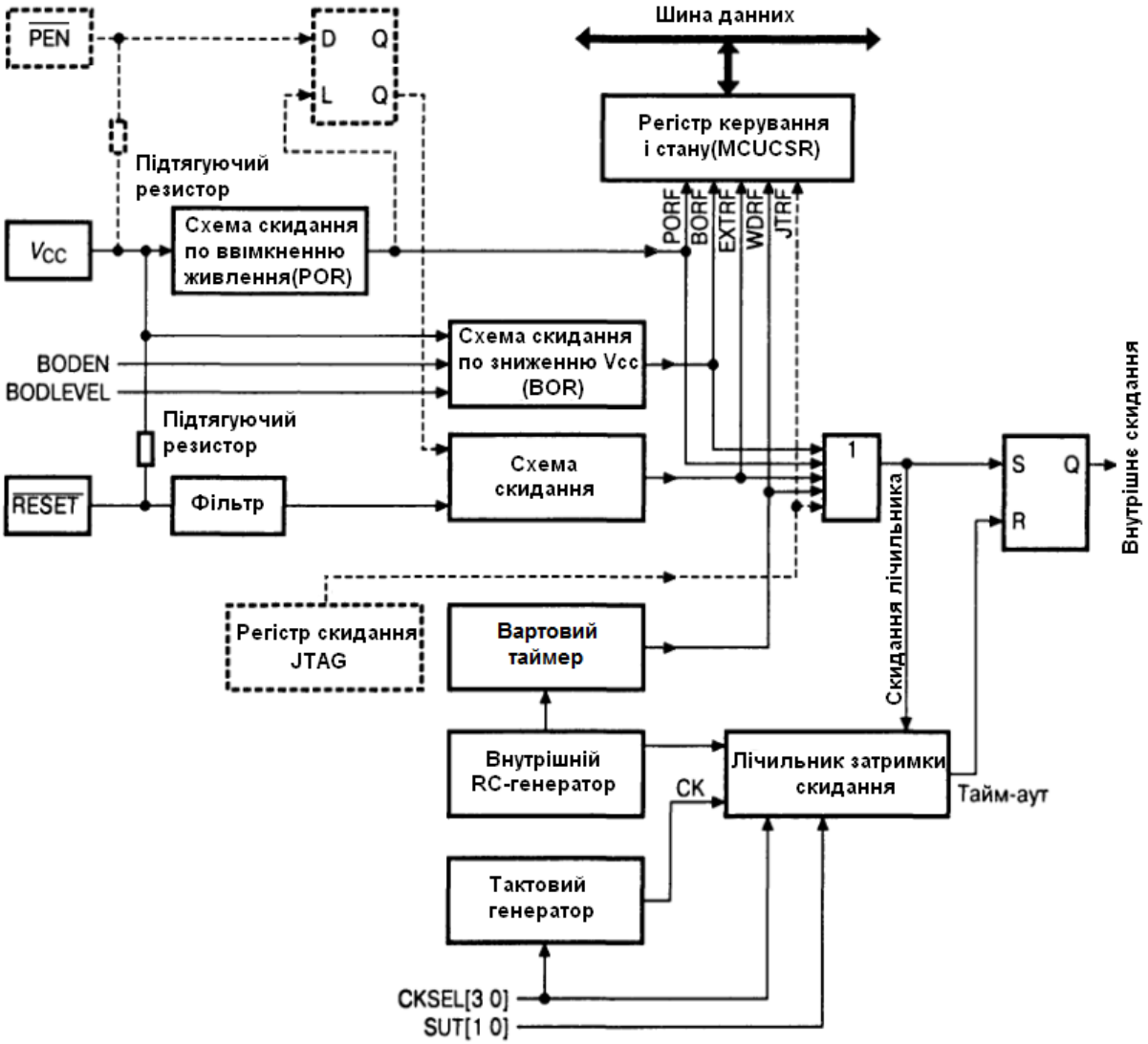


Рис. 13.5. Структурна схема підсистеми скидання

Таблиця 13.7. **Регістри для визначення джерела скидання**

Назва	**Опис**	**Адреса**	**Модель**
MCUCSR	Регістр керування і стану мікроконтролера	$34($54)	ATmega8515x/8535x, ATmega8x/16x/32x/64x/128x, ATmegal62x
MCUSR	Регістр стану мікроконтролера	$34 ($54)	ATmega48x/88x/168x, ATmegal64x/324x/644x, ATmegal65x/325x/3250x/645x/6450x, ATmega640x/1280x/1281x/2560x/2561x

При виникненні переривання мікроконтролер зберігає у стеку адресу наступної команди (вміст лічильника команд PC) і завантажує в нього адресу відповідної підпрограми обробки переривання (вектора переривання). За цією адресою знаходиться команда відносного переходу до підпрограми обробки переривання. В кінці цієї підпрограми знаходиться команда RETI, що забезпечує повернення в основну програму і відновлення стану попередньо збереженого у стеку лічильника команд.

Оскільки основними джерелами переривань є зовнішні переривання або різні периферійні пристрої мікроконтролерів, то кількість переривань залежить від конкретної моделі мікроконтролера.

13.4.2. Таблиця векторів переривань

AVR-мікроконтролери мають багаторівневу систему пріоритетних переривань [1; 7]. Молодші адреси пам’яті програм, починаючи з адреси $0001 або $0002 відведено під таблицю векторів переривань. В цій таблиці кожному перериванню відповідає своя адреса, яка при виникненні переривання завантажується в лічильник команд. Положення вектора в таблиці визначає також і пріоритет відповідного переривання: чим менше адреса, тим вище пріоритет переривання. Розмір вектора переривання залежить від обсягу пам’яті програм мікроконтролера і становить 1 слово для моделей з об'ємом пам’яті менше 16 Кбайт і 2 слова для інших моделей. Для переходу до підпрограми обробки переривання використовуються команда RJMP, або – JMP.

У більшості мікроконтролерів сімейства Mega положення таблиці векторів переривань може бути змінено. Таблиця може розташовуватися не тільки на початку пам’яті програм, але також і на початку області завантажувача. Переміщення таблиці може бути здійснено безпосередньо в ході виконання програми [1; 7].

Для керування розміщенням таблиці використовується регістр керування мікроконтролера MCUCR, який розташовано за адресою $35 ($55), або загальний регістр керування перериваннями GICR, розташований за адресою $3B ($5B) (табл. 13.8).

Для керування таблицею переривань в цих регістрах використовуються два молодших розряди: IVSEL (1-й розряд) і IVCE (0-й розряд). Стан прапорця IVSEL визначає положення таблиці в пам’яті програм. Якщо прапорець скинуто в нуль, то таблиця векторів переривань розташовується на початку пам’яті програм, якщо встановлено в одиницю – на початку області завантажувача [1; 7].

Таблиця 13.8. **Регістри для керування розміщенням таблиці переривань**

Назва	Опис	Адреса	Модель
MCUCR	Регістр керування мікроконтролера	$35 ($55)	ATmega64x/128x, ATmega88x/168x, ATmega164x/324x/644x, ATmega165x/325x/3250x/645x/6450x, ATmega640x/1280x/1281x/2560x/2561x
GICR	Загальний регістр керування перериваннями	$3B ($5B)	ATmega8515x/8535x, ATmega8x/l6x/32x, ATmega162x

Розмір таблиці векторів переривань залежить від моделі мікроконтролера і становить від 16 до більш ніж 56 векторів. Розподіл адрес таблиці векторів переривань для різних мікроконтролерів сімейства наведено у [7]. При розміщенні векторів переривань в області завантажувача до значень, наведених у таблицях, слід додати значення початкової адреси області завантажувача.

Якщо переривання в роботі мікроконтролера не передбачаються, то на місці таблиці векторів переривань може бути розміщена частина основної програми.

13.4.3. Обробка переривань

Для дозволу виконання відповідної підпрограми обробки переривання треба встановити в одиницю прапорець глобального дозволу переривань – I регістра SREG та встановити відповідні розряди регістрів масок переривань [1; 7].

Обробка переривань здійснюється наступним чином:

- при виконанні умов, необхідних для генерації переривання, прапорець, що відповідає цьому перериванню, встановлюється в одиницю, а прапорець I апаратно скидається, забороняючи тим самим обробку наступних переривань. Для дозволу вкладених переривань в підпрограмі обробки переривання цей прапорець треба знову встановити в одиницю;
- якщо переривання дозволено (прапорець дозволу переривання встановлено) у лічильник команд завантажується адреса вектора відповідного переривання. При цьому прапорець переривання апаратно скидається. Ряд прапорців переривань може бути також скинуто записом одиниці в розряд регістра, який відповідає прапорцю;
- виконується підпрограма обробки переривання;

– виконується команда повернення з переривання RETI, при цьому прапорець I апаратно встановлюється в одиницю, дозволяючи обробку наступних переривань;
– автоматично зі стеку відновлюється вміст лічильника команд.

Потім основна програма продовжує своє виконання з того місця, де її було перервано.

При виклику підпрограм обробки переривань вміст регістра стану SREG у стеку не зберігається. Якщо це необхідно, користувач повинен самостійно запам'ятовувати вміст цього регістра при вході в підпрограму обробки переривання і відновлювати його значення перед викликом команди RETI.

Для переривань, які викликано динамічними подіями (наприклад, для переривання, яке генерується при рівності вмісту лічильного регістра і регістра порівняння таймера), прапорець переривання встановлюється тільки в момент виникнення події. Якщо прапорець переривання скинуто, а умови генерації переривання присутні, прапорець буде встановлено тільки в момент виникнення наступної події.

Для зовнішніх переривань, які генеруються за рівнем, прапорці не передбачено, тому інформація про ці переривання буде зберігатися до наявності події, яка викликає переривання.

AVR-мікроконтролери підтримують чергу переривань. Вона працює наступним чином: якщо умови генерації одного або більше переривань виникають у той час, коли прапорець глобального дозволу переривань скинуто (усі переривання заборонено), відповідні прапорці встановлюються в одиницю і залишаються в цьому стані до встановлення прапорця глобального дозволу переривань. Після цього виконується їхня обробка відповідно до пріоритету.

Час відгуку для будь-якого переривання залежить від моделі МК та становить 5 або 4 такти. Протягом цього часу відбувається збереження лічильника команд у стеку. Протягом наступних двох або трьох тактів виконується команда переходу до підпрограми обробки переривання. Якщо переривання відбудеться під час виконання команди, яка триває кілька циклів, то генерація переривання відбудеться тільки після виконання цієї команди. Якщо ж переривання відбудеться під час перебування мікроконтролера у «сплячому» режимі, то час відгуку збільшується ще на 4 або 5 тактів.

Повернення в основну програму займає 4 або 5 тактів, протягом яких відбувається відновлення лічильника команд зі стека. Після виходу з переривання процесор завжди виконує одну команду основної програми, перш ніж обслужити будь-яке відкладене переривання.

13.4.4. Зовнішні переривання

В мікроконтролерах сімейства Mega є два різновиди зовнішніх переривань. Переривання першого типу (так звані «звичайні») генеруються при появі на вході зовнішнього переривання заданого сигналу. Ці переривання присутні у всіх мікроконтролерах сімейства (конкретне число таких переривань залежить від моделі). Переривання другого типу генеруються при будь-якій зміні стану певних виводів мікроконтролера. Наявність тих чи інших зовнішніх переривань в різних моделях показано в [1; 7].

На певних входах зовнішніх переривань виявлення фронтів сигналів відбувається асинхронно, тобто не вимагає наявності тактового сигналу $f_{clkI/O}$. Наприклад, зовнішні переривання за низьким рівнем і переривання за зміною стану виводів завжди реєструються асинхронно. Відповідно, всі ці переривання можуть використовуватися для виведення мікроконтролера зі «сплячих» режимів, відмінних від режиму Idle.

Для дозволу/заборони «звичайних» зовнішніх переривань в залежності від моделі використовується або загальний регістр керування перериваннями GICR, або регістр маскування зовнішніх переривань EIMSK. Для дозволу/заборони переривань за зміною стану виводів використовуються або ті ж самі регістри, або окремий регістр PCICR. Формати регістрів GICR, EIMSK і PCICR різних моделей та описи їх розрядів наведено у [1; 7].

Для індикації настання «звичайних» зовнішніх переривань в мікроконтролерах Mega використовується або загальний регістр прапорців переривань GIFR, або регістр прапорців зовнішніх переривань EIFR. Для індикації переривань за зміною стану виводів використовуються або ті ж самі регістри, або окремий регістр PCIFR [1; 7].

Всі зовнішні переривання генеруються навіть у тому випадку, якщо відповідні виводи сконфігуровані як виходи. Ця особливість мікроконтролерів дозволяє генерувати переривання програмно.

Крім зовнішніх, більшість переривань є внутрішніми і сигналізують про зміну стану окремих периферійних модулів. Кожний модуль може підтримувати одне або декілька переривань і кожне з цих переривань може бути окремо дозволено або заборонено. Запит на переривання генерується в тому випадку, якщо в будь-який момент після його дозволу виявляється відповідна йому умова. У кожного переривання є окремий вектор переривання [7].

13.4.5. Особливості використання модуля переривань у мікроконтролерах XMega

Мікроконтролери XMega мають програмований багаторівневий контролер переривань – PMIC-контролер, який відповідає за обробку запитів переривань з урахуванням їх пріоритетів і рівнів. Після того, як запит переривання підтверджується PMIC-контролером, у лічильник програми завантажується адреса вектора переривання, а потім виконується процедура обробки переривання. Більш детально робота та програмування підсистеми переривань мікроконтролерів XMega описано у [1; 18].

Контрольні запитання та завдання

1. Як називається пристрій, який формує тактові сигнали для МК?
2. Які джерела тактового сигналу можуть використовуватись мікроконтролерами сім'ї Mega?
3. Як здійснюється підключення кварцового резонатора?
4. Для чого призначено зовнішні конденсатори C1 і C2 в схемі підключення кварцового резонатора?
5. Як здійснюється підключення зовнішнього RC-ланцюга?
6. Як здійснюється підключення зовнішнього джерела тактового сигналу?
7. Як вирахувати необхідний опір резистора при використанні генератора з зовнішнім RC-ланцюгом?
8. Поясніть особливості використання внутрішнього каліброваного RC-генератора.
9. Від чого залежить можливість використання того чи іншого джерела тактового сигналу?
10. Які пристрої мікроконтролера використовують тактовий сигнал $f_{clkI/O}$? Як він формується?
11. Чим обмежено мінімально допустиму тактову частоту мікроконтролера?
12. За якою командою здійснюється перехід у «сплячий» режим?
13. Скільки режимів зниженого енергоспоживання підтримують мікроконтролери сімейства Mega?
14. Чим відрізняються між собою режими зниженого енергоспоживання?
15. Як здійснюється вибір режиму зниженого енергоспоживання?
16. Які біти використовуються для керування «сплячим» режимом?
17. За рахунок чого можна знизити енергоспоживання мікроконтролера?
18. Як може бути здійснено вихід зі «сплячого» режиму?

19. Які види скидання можна використовувати у мікроконтролерах сімейства?
20. Що має знаходитись за адресою вектора скидання?
21. Наведіть структуру схеми скидання та поясніть її роботу.
22. Яке значення завантажується в лічильник команд під час скидання?
23. Як можна визначити подію, в результаті якої відбулося скидання пристрою?
24. Що таке «тайм-аут» вартового таймера? Що відбувається за цією подією?
25. Для чого призначено підсистему переривань?
26. Назвіть основні види переривань.
27. Від чого залежить кількість переривань у мікроконторолері?
28. Як відбувається маскування переривань?
29. Назвіть основні джерела переривань.
30. Які регістри потрібні для програмування та керування роботою підсистеми переривань?
31. Які адреси відведено під таблицю векторів переривань в AVR-мікроконтролерах?
32. Від чого залежить розмір таблиці векторів переривань?
33. Чим визначається пріоритет відповідного переривання?
34. Які дії необхідно виконати для дозволу виконання відповідної підпрограми обробки переривання?
35. Як виконується команда RETI?
36. Як відбувається повернення з підпрограми обробки переривання до основної програми?
37. Яку роль відіграє стек під час проектування підсистеми переривань?
38. Опишіть послідовність обробки переривань у МК типу AVR.
39. Назвіть особливості використання модуля переривань у мікроконтролерах XMega.
40. Чи потрібно конфігурувати виводи мікроконтролера, які використовуються для зовнішніх переривань?
41. Від чого залежить кількість зовнішніх переривань в AVR-мікроконтролерах?
42. Чи зберігається у стеку вміст регістра стану SREG при виклику підпрограм обробки переривань?
43. Чи підтримують AVR-мікроконтролери чергу переривань?

14. ПРОЕКТУВАННЯ CAN-МОДУЛЯ

14.1. Загальні відомості

CAN-протокол було розроблено фірмою “Bosch” для систем керування вузлами автомобіля. CAN (Controller Area Network – мережа контролерів) є асинхронним послідовним протоколом високошвидкісної та високонадійної передачі даних у широкомовному режимі в мультимайстерному середовищі. Положення стандарту, які закріплено у використовуваній на сьогоднішній день специфікації 2.0 A/B фірми “Bosch” та міжнародному стандарті ISO 11898, відповідають двом початковим рівням (фізичному та канальному) семирівневої моделі взаємодії відкритих систем ISO/OSI [8].

Основними перевагами цього стандарту є:

- зрілість стандарту. Протокол вже багато років використовується в мікроконтролерних мережах. На сучасному ринку представлено тисячі різних CAN-виробів;
- апаратна підтримка протоколу;
- високонадійний алгоритм передачі та обробки помилок, механізм, що дозволяє від’єднувати несправний віддалений вузол і тим самим не допускати блокування всієї мережі;
- хороша підтримка систем реального часу та систем, керованих подіями.

Controller Area Network має асинхронну послідовну структуру шини з одним логічним сегментом мережі. CAN-мережа може складатися з двох або більше вузлів, з можливістю під’єднання та від`єднання вузлів від шини без переналагоджування інших пристроїв. Логіка шини працює за механізмом «монтажне І», в якому «рецесивний» біт відповідає логічній одиниці, а «домінантний» логічному нулю. Якщо жоден вузол не формує нульовий біт, шина перебуває в одиничному стані. Поява нульового біта від будь-якого вузла створює нульовий стан шини.

У CAN-мережі інформація передається парою скручених дротів (кручена пара). Лінії шини називаються CANH та CANL. Через диференціальний характер лінії зв'язку CAN-шина малочутлива до електромагнітних завад. Для підвищення надійності у високошвидкісних режимах роботи застосовують екранування шини.

Двійкова інформація кодується NRZ-кодом. Код NRZ (Non Return to Zero – без повернення до нуля) – це найпростіший двійковий код, що передається звичайним цифровим сигналом. Логічному нулю відповідає високий рівень напруги в кабелі, логічній одиниці – низький рівень напруги (або навпаки, що не принципово). Протягом бітового інтервалу (bit time, BT), тобто часу передачі одного біта ніяких змін рівня сигналу в кабелі не відбувається.

До переваг NRZ-коду відносяться його досить проста реалізація (вихідний сигнал не треба спеціально кодувати на передавальному кінці та декодувати на приймальному кінці). В порівнянні з іншими кодами потрібна мінімальна пропускна здатність лінії зв'язку.

Для синхронізації даних всіма вузлами використовується додатковий наповнюючий біт (виконується біт-стаффінг). При послідовній передачі п'яти біт однакової полярності, передавач вставляє один додатковий біт протилежної полярності. Приймач також перевіряє полярність та знищує додаткові біти (виконує дебіт-стаффінг).

Структуру типової CAN-мережі у складі мікропроцесорної системи керування показано на рис. 14.1.

Кожне повідомлення, яке передається мережею, має оригінальний 11-бітний або 29-бітний ідентифікатор.

При груповому доступі до шини використовується неруйнівний арбітраж з опитуванням стану шини. Перед початком передачі даних вузол перевіряє стан шини (відсутність активності на шині). Коли починається передача повідомлення, вузол стає керуючим для шини, а всі інші вузли переходять у режим прийому. Кожен вузол видає підтвердження прийому, перевіряє ідентифікатор повідомлення, обробляє або знищує прийняті дані. Якщо два або більше вузли починають передачу одночасно, порозрядний арбітраж дозволяє уникнути конфлікту на шині. Кожен вузол видає на шину свій ідентифікатор і контролює її стан.

Якщо вузол посилає «рецесивний» біт, а читає «домінантний», значить арбітраж втрачено і вузол перемикається в стан прийому. Таким чином, вузол з більш високим пріоритетом виграє арбітраж без необхідності повторювати передачу повідомлення. Всі інші вузли будуть намагатися передати повідомлення після того, як шина звільниться.

Даний механізм не дозволяє передавати одночасно повідомлення з однаковим ідентифікатором, оскільки можуть виникати помилки. Оригінальна специфікація – CAN 2.0A визначає довжину ідентифікатора 11 біт. Такі повідомленнями називаються стандартними. При цьому можливо використовувати лише 2048 різних ідентифікаторів, з яких 16 ідентифікаторів з найменшим пріоритетом: 2032…2047 – зарезервовано. CAN-модулі версії 2.0A можуть тільки передавати або приймати дані.

Рис.14.1. Мікропроцесорна система керування

У новій версії CAN 2.0B довжина ідентифікатора може бути 11 або 29 біт (536 мільйонів варіантів). Версія 2.0B дозволяє активним вузлам робити віддалений стандартний або розширений запит та отримувати дані від потрібного пристрою, використовуючи відповідно стандартне або розширене повідомлення.

CAN-модуль апаратно реалізує шостий рівень та PLS- , MDI- та PMA-підрівні еталонної ISO/OSI-моделі взаємозв'язку відкритих систем OSI (Open System Interconnection) [8].

Шостий (канальний) рівень – має LCC- та MAC-підрівні. LCC-підрівень забезпечує: фільтрацію повідомлень; повідомлення про перевантаження; керування пошуком помилок. MAC-підрівень забезпечує: отримання повідомлень; виконання арбітражу; перевірку помилок; передачу повідомлень та сигналів помилки; видачу висновку про несправність.

PLS-підрівень забезпечує: синхронізацію біт; кодування біт; загальну синхронізацію. PMA-підрівень є мікросхемою CAN-трансивера, а MDI-підрівень – роз'єм для під’єднання до шини.

14.2. Основні характеристики CAN-протоколу

До основних характеристик CAN-протоколу можна віднести [8]:

- пріоритетність повідомлень;
- гарантований час відгуку;
- гнучкість конфігурації;
- груповий прийом із синхронізацією часу;
- систему несуперечності даних;
- роботу у мультимайстерному режимі;
- виявлення помилок та сигналізація про їх наявність;
- автоматичну ретрансляцію пошкоджених повідомлень, як тільки шина знову стане вільною;
- можливість відрізняти нерегулярні помилки та постійні відмови вузлів, а також автономне вимикання дефектних вузлів.

Інформація на шині подається у вигляді фіксованих повідомлень різної, але обмеженої довжини. Коли шина вільна, будь-який підключений до неї вузол може почати передавати нове повідомлення.

В CAN-мережах немає ніякої інформації щодо конфігурації системи (наприклад, адреса вузла). Вузли можуть бути додані до CAN-мережі без зміни програмного забезпечення та апаратних засобів будь-якого вузла або прикладного рівня.

Зміст повідомлення визначається ідентифікатором. Ідентифікатор не вказує адресата повідомлення, але описує значення даних, так, щоб всі вузли в мережі

мали здатність фільтрацією вирішити, чи повинні вони реагувати на дане повідомлення чи ні. Будь-яке число вузлів може отримувати та одночасно реагувати на одне і теж повідомлення. В середині CAN-мережі гарантується, що повідомлення одночасно буде прийнято або всіма вузлами або жодним вузлом.

Швидкість CAN може бути різною в різних системах. Однак у конкретній мережі швидкість передачі інформації повинна бути фіксована. Ідентифікатор визначає статичний пріоритет повідомлення протягом доступу до шини.

Посилаючи кадр віддаленого запиту даних, вузол може виконувати запит даних у іншого вузла, які той повинен передати у вигляді відповідного кадру даних. Кадр віддаленого запиту даних та кадр даних мають однаковий ідентифікатор.

Коли шина вільна, будь-який вузол може починати передавати повідомлення. Вузлу з повідомленням більш високого пріоритету, буде передане право на доступ до шини.

Якщо два або більше вузлів починають одночасно передавати повідомлення, конфлікт доступу до шини вирішується порозрядним арбітражем з використанням ідентифікатора. Якщо кадр даних і кадр віддаленого запиту даних з однаковим ідентифікатором починають передаватися одночасно, то кадр даних має перевагу над кадром віддаленого запиту даних. Протягом арбітражу кожен передавач порівнює рівень переданого біта з рівнем, що спостерігається на шині. Якщо ці рівні однакові, вузол може продовжувати передачу. Коли передано одиничний рівень, а спостерігається нульовий рівень, це означає, що вузол втратив право арбітражу і не повинен посилати більше жодного біта.

Для досягнення безпеки передачі даних у кожному CAN-вузлі є потужні засоби самоконтролю, виявлення та повідомлення про помилки.

Для виявлення помилок застосовуються наступні засоби:

- поточний контроль (передавачі порівнюють рівні бітів, що передаються, з рівнями, виявленими на шині);
- контроль правильності прийнятих повідомлень із використанням надлишкового циклічного коду;
- контроль формату переданого кадру.

Механізми виявлення помилок дозволяють виявляти [8]:

- всі помилки глобального характеру;
- всі локальні помилки передавача;
- до 5 випадкових помилок у повідомленні;
- послідовну групу помилок завдовжки до 15 біт;
- будь-які помилки непарності.

Помилки передач будуть виявлятися з досить високою ймовірністю за допомогою вбудованого механізму виявлення помилок CAN-протоколу [8].

Пошкоджені кадри позначаються кожним вузлом, що виявив помилку. Такі кадри перериваються та автоматично передаються заново. CAN-вузли здатні відрізнити короткі неполадки від постійних відмов. Дефектні вузли від'єднуються від шини CAN-модуля.

Послідовна лінія зв'язку CAN є шиною, до якої теоретично може бути під'єднано будь-яку кількість вузлів. Фактично загальне число вузлів буде обмежено часовою затримкою та (або) електричним навантаженням на лінії шини.

Шина складається з каналу обміну, що передає біти. Із цих даних може бути отримана інформація про пересинхронізацію. В якості каналу обміну може бути один дріт (плюс земля), два диференціальних дроти, оптичне скловолокно і т. ін.

Шина може мати одне із двох логічних значень: "домінантне" або "рецесивне". При одночасній передачі "домінантного" та "рецесивного" бітів, значення, що виникає на шині буде "домінантне". Вважається, що "домінантний" рівень еквівалентний нульовому рівню, а "рецесивний" рівень еквівалентний одиничному рівню.

Всі приймачі перевіряють несуперечність отриманого повідомлення, підтверджують несуперечність повідомлення та позначають суперечливі повідомлення.

Для зменшення кількості енергії, яка споживається від джерела живлення, CAN-вузол може бути переведений у режим „сну" без внутрішньої активності та з відключеними формувачами шини. Вихід із цього режиму відбувається при появі будь-якої активності на шині або внутрішньому стані системи. При пробудженні, здійснюється внутрішнє перезавантаження. MAC-підрівень буде чекати стабілізації генератора системи, і потім буде очікувати самосинхронізації із сигналами на шині (перевіряючи, одинадцять послідовних одиничних біт), поки вихідні формувачі знову не встановляться в стан „під'єднано до шини".

Вимоги до синхронізації дозволяють використовувати керамічні резонатори в системах зі швидкостями передачі до 125 Кбіт/сек. Для більш високої швидкості CAN-шини, потрібен кварцовий резонатор.

14.3. Структура повідомлень CAN-мережі

14.3.1. Загальні відомості

Повідомлення, що передаються CAN-шиною, називаються кадрами (фреймами). Існують чотири типи кадрів:

1) Data Frame – кадр даних.
2) Remote Frame – кадр запиту даних.
3) Error Frame – кадр помилки.
4) Overload Frame – кадр перевантаження.

Є два формати повідомлень, які передаються, що відрізняються довжиною поля ідентифікатора:

- кадри з 11-розрядним ідентифікатором, які називаються стандартними кадрами;
- кадри, що містять 29-розрядні ідентифікатори та називаються розширеними кадрами.

Кадр даних передає дані від передавача до приймача. Кадр віддаленого запиту даних передається вузлом, щоб запросити передачу кадру даних від іншого вузла з тим же самим ідентифікатором. Кадр помилки передається будь-яким вузлом при виявленні помилки на шині. Кадр перевантаження використовується, щоб забезпечити додаткову затримку між попереднім і наступним кадром даних або кадром віддаленого запиту даних. Кадри даних і кадри віддалених запитів даних можуть використовуватися у стандартному та розширеному форматах. Вони відокремлюються від попередніх кадрів міжкадровим простором.

14.3.2. Кадр даних

Кадр даних (Data Frame) складається із семи різних полів: «початок кадру» (start of frame); "поле арбітражу" (arbitration field); "поле керування" (control field); "поле даних" (data field); "поле CRC" (CRC field); "поле підтвердження" (ACK field) та "кінець кадру" (end of frame). Поле даних може мати нульову довжину. На рис. 14.2 та рис. 14.3 наведено структури, відповідно стандартного та розширеного повідомлень.

Нижче наведено опис окремих полів цих повідомлень.

Початок кадру (стандартний або розширений формат)

Початок кадру (Start of Frame) відмічає початок кадру даних або кадру віддаленого запиту даних. Це поле складається з одиночного нульового біта. Вузлу дозволено почати передачу, коли шина вільна (див. міжкадровий простір). Всі вузли повинні синхронізуватися за фронтом, який викликаний передачею поля "початок кадру" (див. апаратна синхронізація) вузлом, що почав передачу першим.

Поле арбітражу

Формат поля арбітражу (Arbitration field) відрізняється для стандартного та розширеного форматів:

- у стандартному форматі поле арбітражу, складається з 11-розрядного ідентифікатора та RTR-біта. Біти ідентифікатора позначені ID10...ID0;

– у розширеному форматі поле арбітражу складається з 29-розрядного ідентифікатора, SRR-біта, IDE-біта, та RTR-біта. Біти ідентифікатора позначені ID28...ID0.

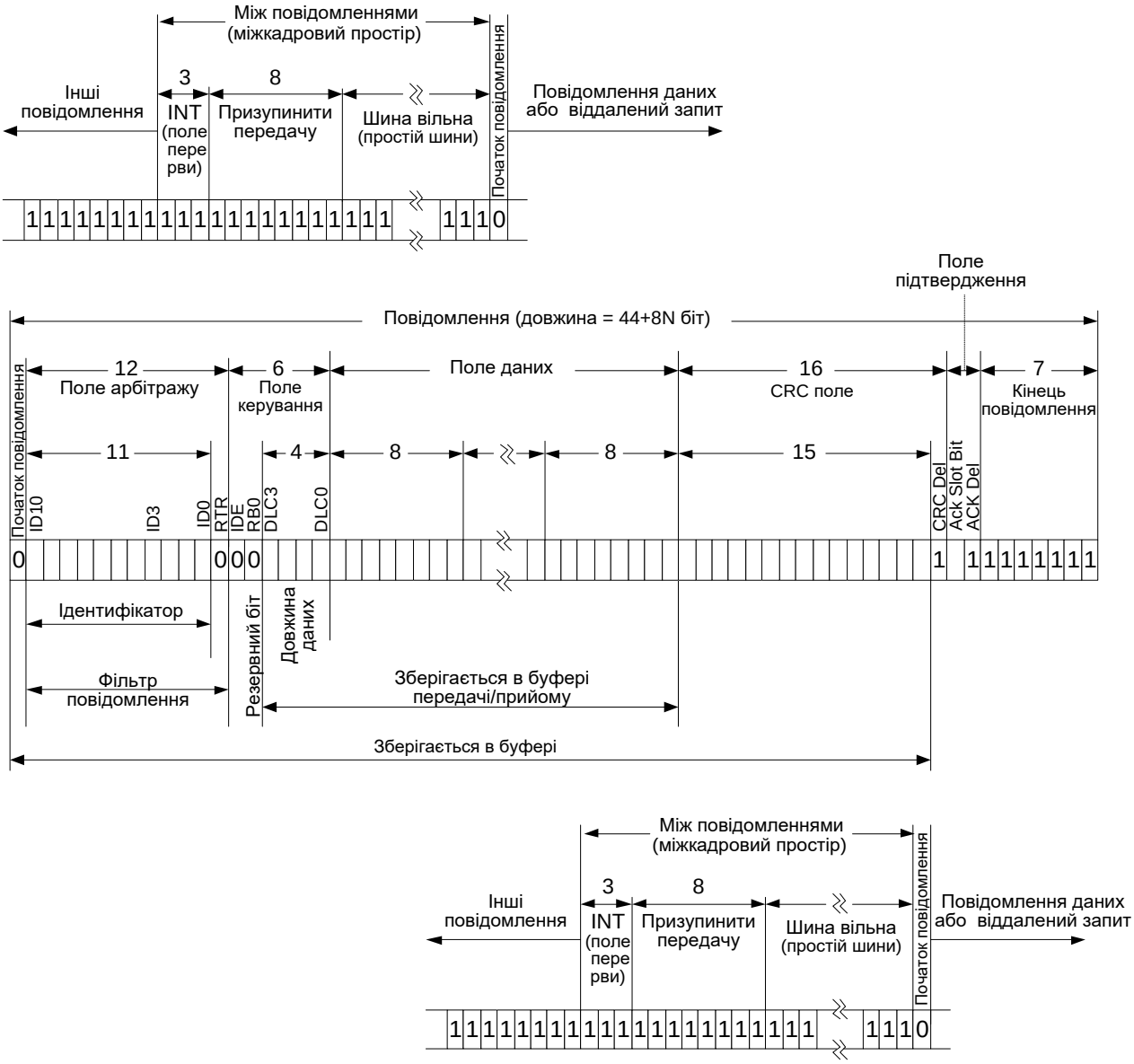


Рис. 14.2. Стандартне повідомлення

Ідентифікатор стандартного повідомлення

Довжина ідентифікатора (Identifier) стандартного повідомлення – 11 біт і відповідає BASE ID у розширеному форматі (рис. 14.3). Ці біти передаються в порядку ID10...ID0. Молодший біт – ID0. Сім старших бітів (ID10…ID4) не повинні бути одиничними бітами. У стандартному кадрі ідентифікатор супроводжується RTR-бітом, який дорівнює нулю.

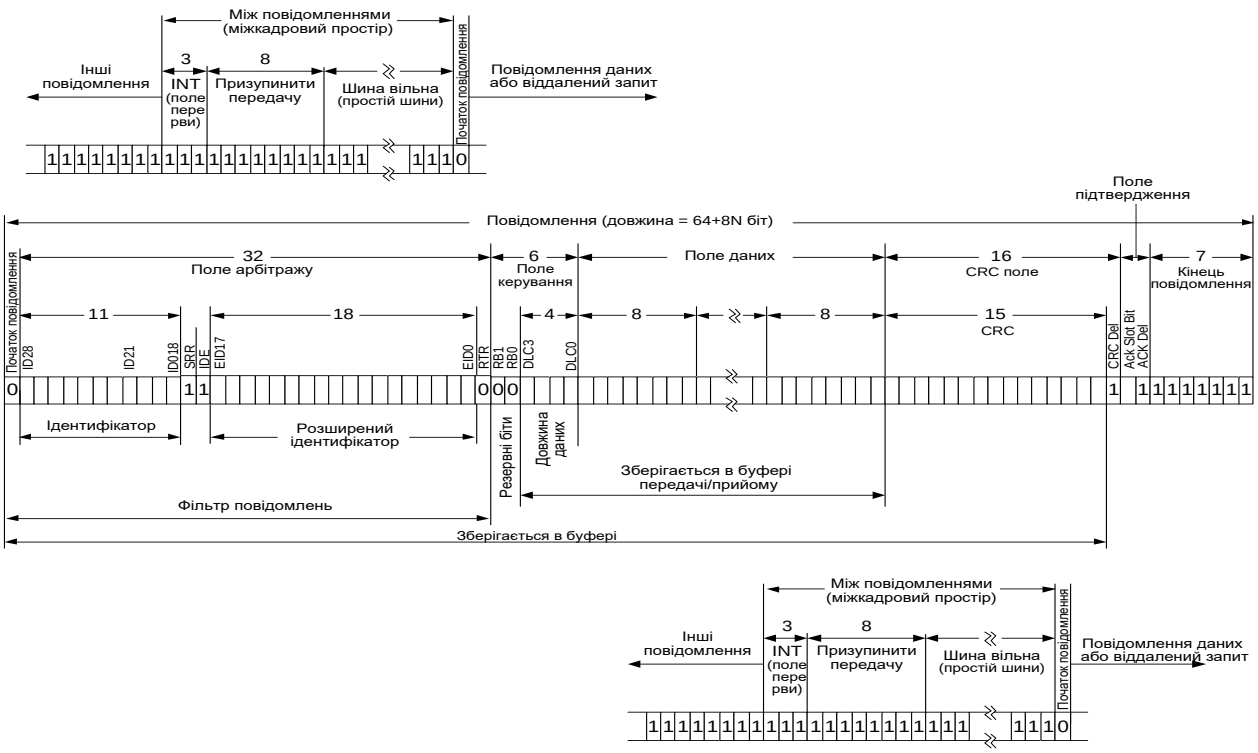


Рис. 14.3. Розширене повідомлення

Ідентифікатор розширеного повідомлення

На відміну від стандартного ідентифікатора, розширений ідентифікатор (рис. 14.3) складається з 29 біт. Його формат містить дві секції:

1) Base ID – 11 біт.
2) Extended ID – 18 біт.

Base ID

Base ID складається з 11 біт. Ця секція передається в порядку від ID28 до ID18. Це еквівалентно формату стандартного ідентифікатора. Base ID визначає базовий пріоритет розширеного кадру.

Extended ID

Extended ID складається з 18 біт. Ця секція передається в порядку від ID17 до ID0. У розширеному кадрі ідентифікатор супроводжується RTR-бітом, який дорівнює нулю.

Біт віддаленого запиту RTR (стандартний і розширений формат)

В кадрах даних біт віддаленого запиту – RTR повинен бути переданий нульовим рівнем. У кадрі віддаленого запиту даних RTR-біт повинен бути

одиничним. У розширеному кадрі спочатку передається Base ID, а наступними бітами передаються SRR та IDE. Extended ID передається після IDE-біта.

Біт SRR (розширений формат)

В розширених кадрах біт SRR дорівнює одиниці та замінює RTR-біт стандартного кадру. При одночасній передачі стандартного кадру та розширеного кадру, Base ID якого збігається з ідентифікатором стандартного кадру, стандартний кадр має перевагу над розширеним кадром.

Біт IDE (розширений формат)

Біт розширеного ідентифікатора IDE належить:

- полю арбітражу для розширеного формату;
- полю керування для стандартного формату. IDE-біт у стандартному форматі передається нульовим рівнем, а у розширеному форматі – одиничним рівнем.

Поле керування (стандартний і розширений формат)

Поле керування (Control field) складається із шести біт. Формат поля керування відрізняється для стандартного та розширеного формату. Кадри в стандартному форматі включають: код довжини даних DLC, біт IDE, що передається нульовим рівнем (див. вище), та зарезервований біт RB0. Кадри в розширеному форматі включають код довжини даних і два зарезервованих біти RB1 та RB0. Зарезервовані біти повинні мати нульовий рівень.

Код довжини даних

Число байт у полі даних стандартного та розширенного форматів визначається кодом довжини даних – Data length code. Цей код має довжину 4 біти та передається у середині поля керування (табл. 14.1). Допустиме число байт даних, що передаються: 0; 1; ... 8. Інші величини використовуватися не можуть.

Поле даних

Поле даних (Data field) стандартного та розширенного форматів складається з даних, які будуть передані в середині кадру даних. Воно може містити від 0 до 8 байт, кожен з яких містить 8 біт, які передаються, починаючи зі старшого значущого розряду (СЗР).

Таблиця 14.1. **Кодування довжини даних**

Число байт даних	**Код довжини даних**			
	DLC3	**DLC2**	**DLC1**	**DLC0**
0	d	d	d	d
1	d	d	d	r
2	d	d	r	d
3	d	d	r	r
4	d	r	d	d
5	d	r	d	r
6	d	r	r	d
7	d	r	r	r
8	r	d	d	d
d – “домінантний” r – “рецесивний”				

Поле CRC

Поле CRC стандартного та розширеного форматів (CRC field) містить послідовність CRC та CRC-роздільник.

Послідовність CRC

При обчисленні послідовністі контролю кадру (CRC Sequence) береться початковий поліном, коефіцієнти якого задано послідовністю біт, які складаються з полів: "початок кадру", "поле арбітражу", "керуюче поле", "поле даних" (якщо воно є) і для 15 молодших коефіцієнтів – нулів. Цей поліном ділиться на поліном:

$$X^{15} + X^{14} + X^{10} + X^{8} + X^{7} + X^{4} + X^{3} + 1.$$

Залишок від цього поліноміального ділення і є послідовність CRC, яка передається шиною.

Приклад розрахунку поля CRC

Нижче наведено приклад розрахунку п`ятнадцяти бітів поля CRC для кадру даних, в якому передається повідомлення з ідентифікатором – 00000000001 з одним байтом даних – 00001111 (рис.14.4) [8].

Процес поліноміального ділення для обчисення поля CRC показано на рис. 14.5, а повідомлення з розрахованим полем CRC – на рис. 14.6.

Початок повідомлення	Ідентифікатор	RTR	IDE	RB0	Довжина данних	Поле даних	CRC поле	CRC Del	ACK Slot Bit	ACK Del	Кінець кадру
0	0 0 0 0 0 0 0 0 0 0 0 1	0	0	0	0 0 0 1	0 0 0 0 1 1 1 1	x x x x x x x x x x x x x x x x	1		1	1 1 1 1 1 1 1

Рис. 14.4. Повідомлення, яке потрібно передати

```
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 1
                          1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 1                                         | 1 1 1 1 1 0 1 0 1 0 0 0 0 1 1
                          0 1 0 0 0 1 0 0 1 0 0 1 0 1 1 0 0
                            1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 1
                            0 1 0 0 1 1 0 0 1 0 1 1 0 1 0 1 0
                              1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 1
                              0 1 0 1 1 1 0 0 1 1 1 1 0 0 1 1 0
                                1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 1
                                0 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 0
                                  1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 1
                                  0 0 1 1 1 1 0 1 0 1 1 0 0 1 1 1 0 0
                                      1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 1
                                      0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0
                                          1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 1
                                          0 0 0 0 0 1 0 1 1 0 0 0 1 1 0 1 0 0 0 0 0
                                                  1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 1
                                                  0 1 1 1 0 1 0 0 0 0 1 1 1 0 0 1 0
                                                    1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 1
                                                    0 0 1 0 1 1 0 1 1 1 1 0 1 0 1 1 0 - значення поля CRC
```

Рис. 14.5. Поліноміальне ділення

Початок повідомлення	Ідентифікатор	RTR	IDE	RB0	Довжина данних	Поле даних	CRC поле	CRC Del	ACK Slot Bit	ACK Del	Кінець кадру
0	0 0 0 0 0 0 0 0 0 0 0 1	0	0	0	0 0 0 1	0 0 0 0 1 1 1 1	1 0 1 1 0 1 1 1 1 0 1 0 1 1 0	1		1	1 1 1 1 1 1 1

Рис. 14.6. Повідомлення з розрахованим полем CRC

Роздільник CRC (стандартний та розширений формати)

Послідовність CRC супроводжується роздільником CRC (CRC-Delimiter), що складається з одного одиничного біта.

Циклічні CRC-коди названо так тому, що в них частина комбінацій коду або всі комбінації можуть бути отримано шляхом циклічного зсуву однієї або декількох комбінацій коду. Циклічний зсув здійснюється справа наліво, причому крайній лівий символ щоразу переноситься в кінець комбінації. Практично всі циклічні коди відносяться до систематичних кодів – у них контрольні та інформаційні розряди розташовано на чітко визначених місцях. Крім того, циклічні коди відносяться до числа блочних кодів. Кожен блок кодується самостійно. Ідея побудови циклічних кодів базується на використанні многочленів, що не

приводяться в полі двійкових чисел. Многочлени, що не приводяться – це такі многочлени, які діляться без залишку тільки на себе і на одиницю.

Поле підтвердження (стандартний та розширений формати)

Поле підтвердження (ACK-field) має довжину два біти й містить: біт – "Область підтвердження" і роздільник підтвердження. У поле підтвердження вузол, що передає, посилає два біти з одиничним рівнем. Приймач, що отримав повідомлення правильно, сповіщає про це передавачу, посилаючи біт з нульовим рівнем протягом прийому біта "область підтвердження".

Біт "Область підтвердження"

Всі вузли, що отримали повідомлення з відповідною послідовністю CRC, сповіщають про це під час прийому біта "Область підтвердження" (ACK Slot Bit) шляхом заміни біта з одиничним рівнем на біт з нульовим рівнем.

Роздільник підтвердження

Роздільник підтвердження (ACK-delimiter) – другий біт поля підтвердження, який має одиничний рівень.

Кінець кадру (стандартний та розширений формати)

Кожен кадр даних і кадр віддаленого запиту даних закінчується послідовністю прапорців, яка складається із семи одиничних біт (End of Frame).

14.3.3. Кадр віддаленого запиту даних

Вузол, який виконує прийом даних, може запросити передачу відповідних даних від потрібних вузлів, посилаючи кадр віддаленого запиту даних (Remote Frame), (рис. 14.7).

Кадр віддаленого запиту даних існує у стандартному та розширеному форматах. В обох випадках він складається із шести бітових полів: "початок кадру (повідомлення)" (Start of Frame), "поле арбітражу" (Arbitration-field), "поле керування " (Control-field), "поле CRC" (CRC-field), "поле підтвердження" (ACK-field), "кінець кадру" (End of Frame). На відміну від кадру даних, RTR-біт кадру віддаленого запиту даних – одиничний. У цьому кадрі поле даних відсутнє. Значення коду довжини даних відповідає коду довжини даних кадру даних, який запитується. RTR-біт вказує – чи є переданий кадр кадром даних чи кадром віддаленого запиту.

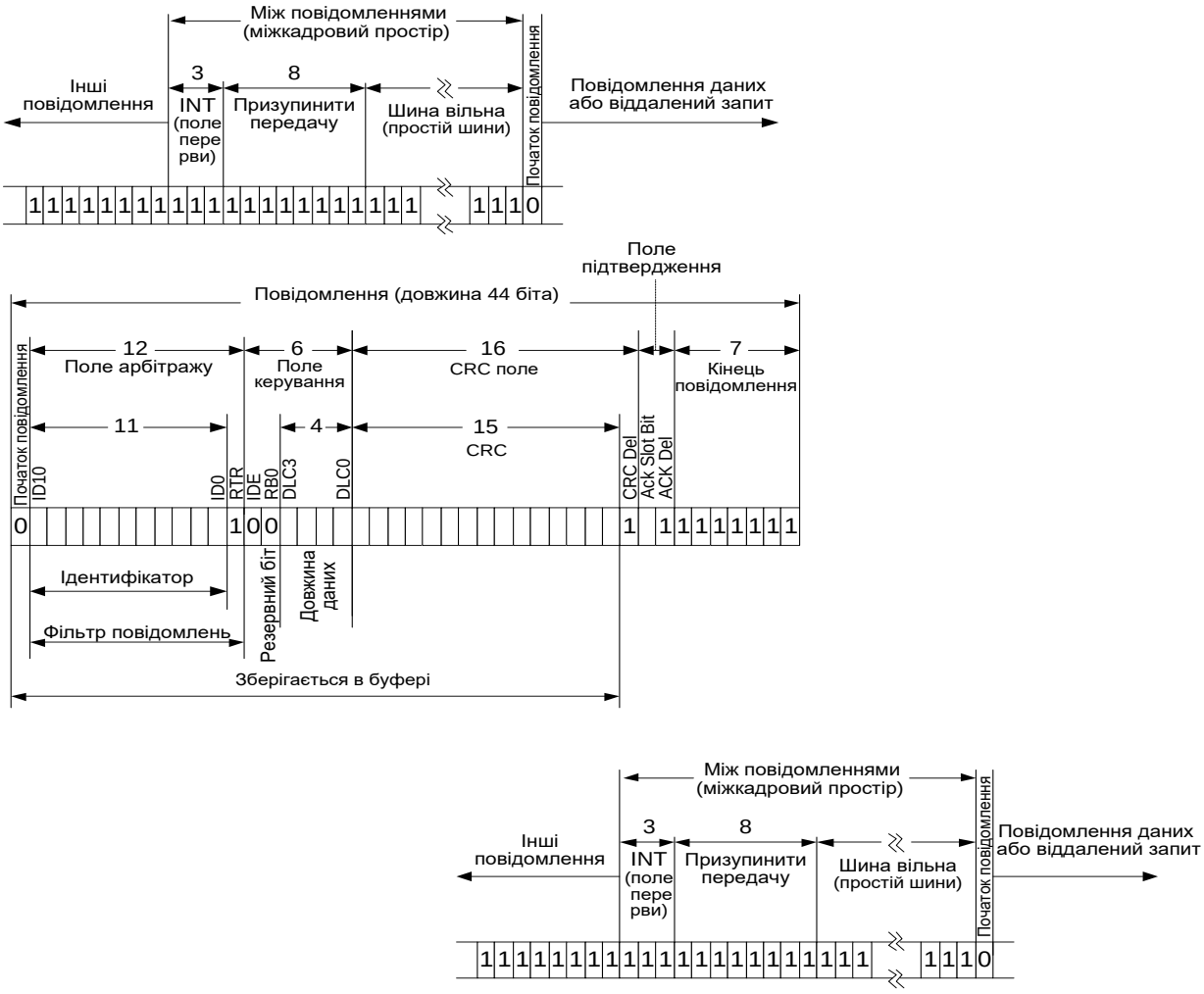


Рис.14.7. Кадр віддаленого стандартного запиту даних

В табл. 14.2 показано вплив бітів RTR, SRR та IDE на тип повідомлення, що передається.

Прапорець помилки

Є два види прапорця помилки (Error flag): прапорець активної помилки та прапорець пасивної помилки.

Прапорець активної помилки складається із шести послідовних нульових біт.

Прапорець пасивної помилки складається із шести послідовних одиничних біт, якщо вони не перезаписані нульовими бітами інших вузлів.

Таблиця 14.2. **Вплив бітів RTR, SRR та IDE на тип повідомлення, що передається**

№ п.п.	Значення біта			Тип повідомлення
	RTR в стандартному повідомленні (SRR в розширеному повідомленні)	IDE	RTR в розширеному повідомленні	
1	0	0	–	Стандартне повідомлення
2	1	1	0	Розширене повідомлення
3	1	0	–	Віддалений стандартний запит
4	1	1	1	Віддалений розширений запит

Вузол у стані "активної помилки", виявивши умову помилки, сповіщає про це передачею прапорця активної помилки. Форма прапорця помилки порушує закон заповнення біта (біт-стаффінг), який застосовується до всіх полів від поля "початок кадру" до роздільника CRC, або руйнує фіксовану форму поля підтвердження або поля "кінець кадру". Як наслідок, всі інші вузли виявляють умову помилки і у свою чергу починають передачу прапорця помилки. Таким чином, послідовність "домінантних" бітів, що фактично може з'явитися на шині, отримується суперпозицією різних прапорців помилки, які передано окремими вузлами. Сумарна довжина цієї послідовності змінюється від шести до дванадцяти біт. Вузол, у стані "пасивної помилки", виявивши умову помилки, робить спробу сповістити про це передачею прапорця пасивної помилки. Він очікує появи шести послідовних однакових одиничних біт, які починають прапорець пасивної помилки. Прапорець пасивної помилки завершується після виявлення цих 6 біт.

Роздільник помилки

Роздільник помилки (Error Delimiter) складається з восьми "рецесивних" бітів. Після передачі прапорця помилки кожен вузол посилає "рецесивні" біти та перевіряє шину, поки не виявляє "рецесивний" біт. Далі він починає передачу ще семи "рецесивних" бітів.

14.3.4. Кадр перевантаження

Кадр перевантаження (Overload Frame) містить два бітових поля: прапорець перевантаження та роздільник перевантаження [8].

Є три види перевантаження, які приводять до передачі прапорця перевантаження:

- внутрішній стан приймача, який потребує затримки наступного кадру даних або кадру віддаленого запиту даних;
- виявлення "домінантного" біта при передачі першого та другого бітів поля перерви;
- якщо вузол виявляє "домінантний" біт на восьмому біті (останньому біті) роздільника помилки або роздільника перевантаження, це спричиняє передачу кадру перевантаження (а не кадру помилки). Лічильники помилок не будуть збільшені.

14.3.5. Міжкадровий простір

Кадри даних і кадри віддаленого запиту даних відокремлюються від попередніх кадрів будь-якого типу (кадр даних, кадр віддаленого запиту даних, кадр помилки, кадр перевантаження) бітовим полем, яке називається міжкадровим простором (Interframe Space) [8]. На відміну від них, кадрам перевантаження й кадрам помилки не передує міжкадровий простір (кратні кадри перевантаження також не відокремлюються інтервалами).

Для вузла, що був джерелом попереднього повідомлення, міжкадровий простір містить бітові поля: перерва та простій шини. Для вузла в стані "пасивної помилки", що був джерелом попереднього повідомлення, міжкадровий простір містить додаткове поле «призупинити передачу». Це поле має бути між полями «перерва» і «простій» шини.

14.4. CAN-модуль мікроконтролера AT90CAN128

14.4.1.Загальні відомості

У складі сім`ї AVR є, наприклад, мікроконтролер AT90CAN128, який має CAN-модуль. Він апаратно підтримує протоколи CAN-2.0A та CAN-2.0B.

Нижче наведено основні характеристики цього мікроконтролера:

- стандартний і розширений типи повідомлень;
- довжина даних у повідомленні від 0 до 8 байт;
- програмована швидкість передачі інформації до 1 Мбіт/с;
- підтримка віддаленого запиту даних;
- 15 повних об'єктів повідомлень з окремими ідентифікаторами і масками;

– можливість указання пріоритету та аварійного припинення передачі;
– можливість пробудження з SLEEP-режиму;
– гнучка система переривань від CAN-модуля;
– низьке енергоспоживання в режимі SLEEP.

14.4.2. Структура CAN-модуля

Структуру CAN-модуля мікроконтролера AT90CAN128 наведено на рис. 14.8.

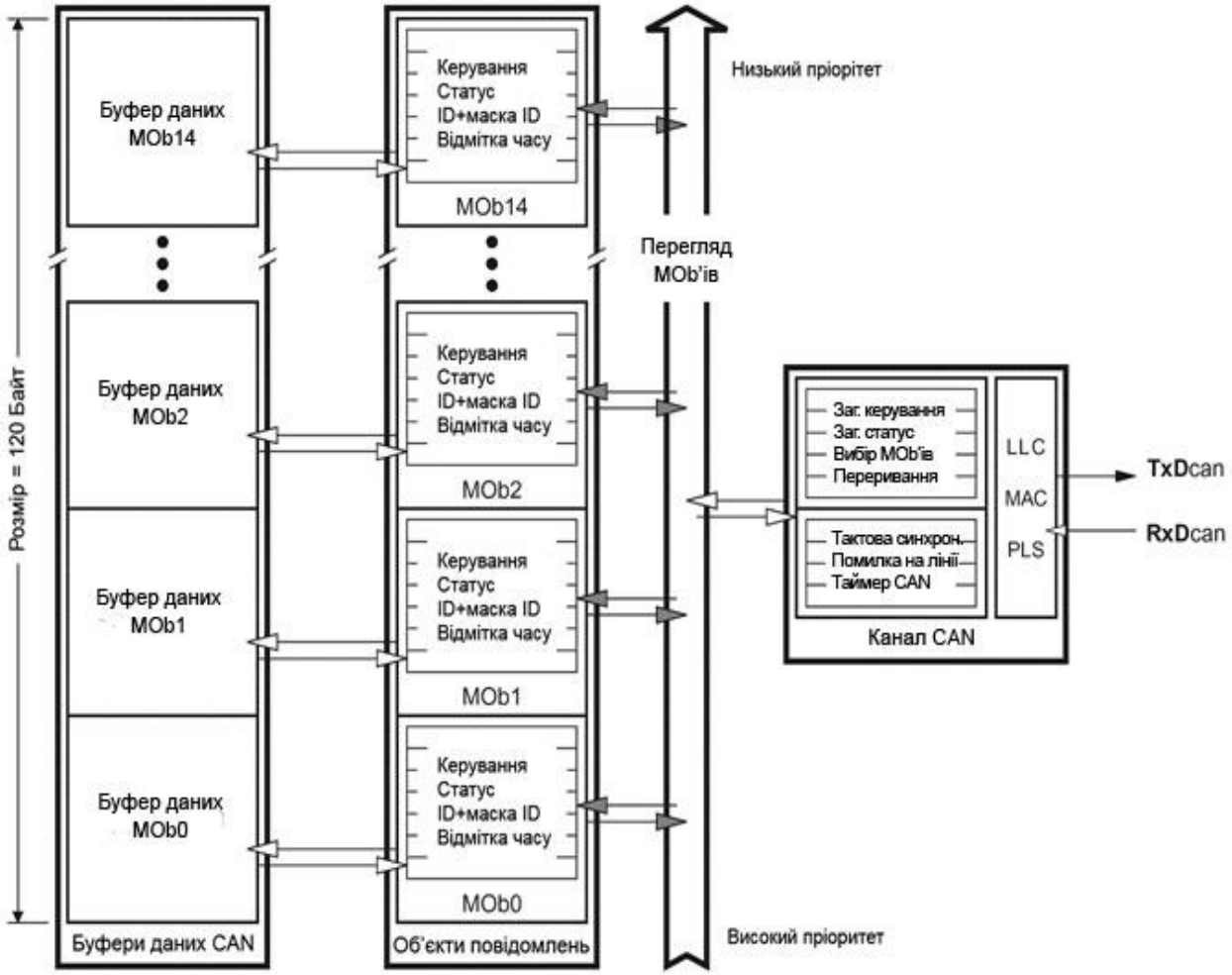


Рис. 14.8. Структура CAN-модуля мікроконтролера AT90CAN128

Мікроконтролер забезпечує повну апаратну підтримку фільтрації повідомлень та керування ними. Для кожного повідомлення, яке повинно бути передано або отримано, CAN-модуль містить так званий об'єкт повідомлення – MOb (Message Objects). Останній було розроблено для опису CAN-кадру, як об'єкта. MOb є дескриптором CAN-кадру та містить всю інформацію для роботи з ним.

14.4.3. Організація керуючих регістрів

Склад та організацію керуючих регістрів CAN-модуля, наведено на рис. 14.9.

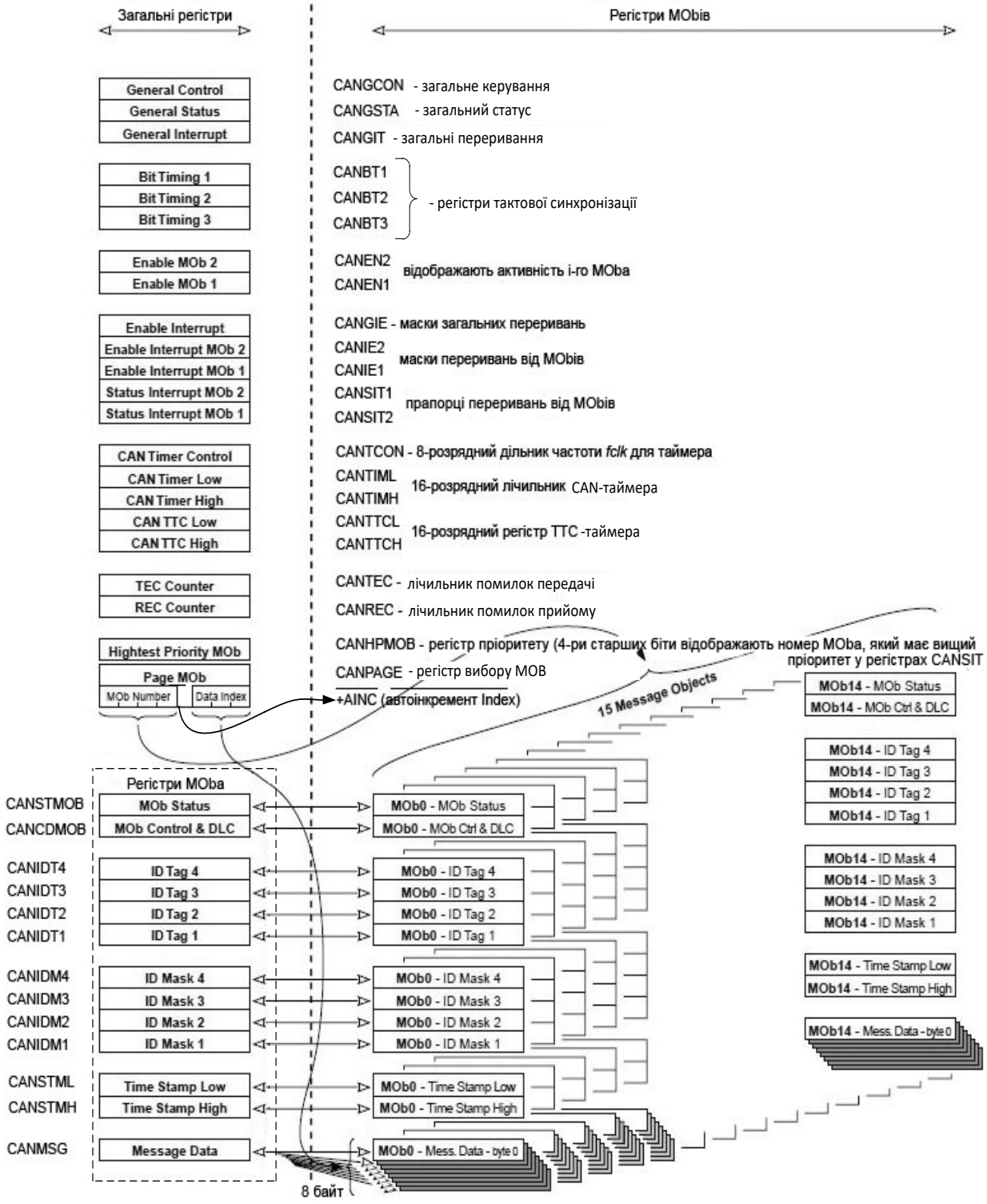


Рис. 14.9. Організація керуючих регістрів CAN-модуля

Ці регістри поділяються на дві групи: загальні регістри та регістри MOb [8].

Для програмування CAN-модуля в цілому використовують загальні регістри, а для програмування об`єктів повідомлень – регістри MOb, які описано в [8].

14.4.4. Режими роботи

Загальна характеристика

Кожен MOb має свої власні поля, щоб керувати поточним режимом. Після перезавантаження мікроконтролера жоден з MOb не має вибраного режиму роботи.

Перед активацією периферійних пристроїв, під’єднаних до CAN-модуля, кожен MOb повинен бути налаштований відповідним чином на поточний режим (виняток лише для дезактивованого режиму – біти CONMOB0; CONMOB1 регістра CANCDMOB дорівнюють нулю) [8].

Налаштування об'єктів повідомлень виконується згідно табл. 14.3.

Таблиця 14.3. **Налаштування об’єктів повідомлень**

<table>
<tr><th colspan="2">Конфігурація MOb</th><th>Відповідь</th><th rowspan="2">RTR Tag</th><th rowspan="2">Поточний режим</th></tr>
<tr><th>CONMOB0</th><th>CONMOB1</th><th>Біт RPLV регістра CANCDMOB</th></tr>
<tr><td>0</td><td>0</td><td>x</td><td>x</td><td>MOb дезактивований</td></tr>
<tr><td rowspan="2">0</td><td rowspan="2">1</td><td>x</td><td>0</td><td>Передача кадру даних</td></tr>
<tr><td>x</td><td>1</td><td>Передача кадру віддаленого запиту</td></tr>
<tr><td rowspan="3">1</td><td rowspan="3">0</td><td>x</td><td>0</td><td>Прийом кадру даних</td></tr>
<tr><td>0</td><td rowspan="2">1</td><td>Прийом кадру віддаленого запиту</td></tr>
<tr><td>1</td><td>Прийом кадру віддаленого запиту та автоматична відповідь</td></tr>
<tr><td>1</td><td>1</td><td>x</td><td>x</td><td>Прийом кадрів в буфер (прийом мультикадрів)</td></tr>
</table>

CAN-модуль визначає, які повідомлення повинні передаватися, а які прийматися. Коли ідентифікатор отриманого повідомлення збігається з одним із запрограмованих ідентифікаторів, дозволених до прийому повідомлень, об’єкт повідомлення зберігається і програма інформується перериванням. Також на повідомлення віддаленого запиту можуть автоматично надсилатися відповідні дані.

Дезактивований режим

У цьому режимі MOb залишається "вільним" (табл. 14.3).

Передача кадру даних або віддаленого запиту (Tx-конфігурація)

Робота CAN-модуля в цьому режимі відбувається у наступній послідовності:

1. Перед відправкою повідомлення або кадру віддаленого запиту наведені нижче біти керуючих регістрів MOb повинні бути ініціалізовані [8]:

- IDT (регістри CANIDT1…CANIDT4) – 11-бітний або 29-бітний ідентифікатор;
- IDE (регістр CANCDMOB) – біт ознаки розширеного ідентифікатора;
- RTRTAG (регістр CANIDT4) – біт ознаки віддаленого запиту на передачу;
- DLC (регістр CANCDMOB) – розмір поля даних;
- RBnTAG (регістр CANIDT4) – зарезервовані поля;
- MSG (регістри CANMSG, CANPAGE) – поле даних.

2. Після встановлення Tx-конфігурації (табл. 14.3) та ініціалізації регістрів MOb готовий відіслати кадр даних або віддаленого запиту.

3. Далі CAN-модуль переглядає (сканує) всі MOb в Tx-конфігурації. Можлива ситуація, коли декілька MOb очікують передачі:

- в наслідок втрати арбітражу шини або невдалої попередньої передачі;
- ініціювання передачі декількох наступних MOb до завершення передачі поточного;
- активування CAN-модуля за попереднім налаштуванням на передачу декількох MOb.

В цьому випадку CAN-модуль знаходить MOb, що має найбільший пріоритет і намагається надіслати його.

4. Коли передачу завершено, встановлюється біт TXOK регістра CANSTMOB. В цьому випадку можливе переривання, якщо воно не замасковано.

5. Всі параметри і дані залишаються доступними в MOb до нової ініціалізації.

У [8] наведено схему алгоритму підготовки MOb до передачі повідомлення чи віддаленого запиту та керучу програму мовою C.

Прийом кадру даних або віддаленого запиту (Rx-конфігурація)

Робота CAN-модуля в цьому режимі відбувається у наступній послідовності:

1. Перед прийомом повідомлення наведені нижче поля керуючих регістрів MOb повинні бути ініціалізовані:
   - IDT – 11-бітний або 29-бітний ідентифікатор (регістри CANIDT1…CANIDT4);
   - IDMSK – маска IDT (регістри CANIDM1…CANIDM4);
   - IDE – біт ознаки розширеного ідентифікатора (регістр CANCDMOB);
   - IDEMSK – маска розширеного ідентифікатора IDE (регістр CANIDM4);
   - RTRTAG – біт ознаки віддаленого запиту на передачу (регістр CANIDT4);
   - RTRMSK – біт маски RTRTAG (регістр CANIDM4);
   - DLC – розмір поля даних (регістр CANCDMOB);
   - RBnTAG – резервні поля (регістр CANIDT4).
2. Після встановлення Rx-конфігурації (табл. 14.3, див. вище) та ініціалізації регістрів MOb готовий прийняти кадр даних або віддаленого запиту.
3. Коли з CAN-мережі отримано ідентифікатор кадру, CAN-модуль переглядає всі MOb намагаючись знайти відповідний MOb з найбільшим пріоритетом.
4. IDT-, IDE-, та DLC-поля знайденого відповідного MOb оновлюються значеннями із прийнятого кадру.
5. Як тільки прийом закінчено, байти даних отриманого повідомлення зберігаються (не для кадру віддаленого запиту) в буфері даних відповідного MOb (рис. 14.8) і встановлюється біт RXOK регістра CANSTMOB. В цьому випадку можливе переривання, якщо воно не замасковано.
6. Всі параметри і дані залишаються доступними в MOb до наступної ініціалізації.

У [8] наведено схему алгоритму підготовки MOb до прийому повідомлення чи віддаленого запиту та керучу програму мовою С.

Автоматична відповідь

Робота CAN-модуля в цьому режимі відбувається у наступній послідовності:

1. Кадр даних у відповідь на кадр віддаленого запиту може бути автоматично відправлено після прийому очікуваного кадру віддаленого запиту, якщо біт RPLV регістра CANCDMOB встановлено у одиницю (табл. 14.3, див. вище).

2. Для цього режиму наведені нижче поля керуючих регістрів MOb повинні бути ініціалізовані:

- IDT (регістри CANIDT1…CANIDT4) – 11-бітний або 29-бітний ідентифікатор;
- IDE (регістр CANCDMOB) – біт ознаки розширеного ідентифікатора;
- RTRTAG (регістр CANIDT4) – біт ознаки віддаленого запиту на передачу;
- DLC (регістр CANCDMOB) – розмір поля даних;
- RBnTAG (регістр CANIDT4) – резервні поля;
- MSG (регістри CANMSG, CANPAGE) – поле даних.

3. MOb, обраний запитом, готовий відразу відповісти без додаткових налаштувань, тому що не треба заповнювати його CAN-буфер даних з кадру запиту. IDT-, IDE-, DLC- та деякі інші поля прийнятого кадру віддаленого запиту використовуються для автоматичної відповіді.

4. Коли передачу відповіді завершено, встановлюється біт TXOK регістра CANSTMOB. При цьому можливе переривання, якщо воно не замасковано.

5. Всі параметри і дані залишаються доступними в MOb до наступної ініціалізації.

У [8] наведено схему алгоритму роботи CAN-модуля у цьому режимі та керучу програму мовою С.

Прийом мультикадрів

Цей режим використовується для прийому в буфер відповідного набору кадрів – отримання мультикадрів. Керувати вхідними кадрами такого типу дозволяє наявність пріоритетету між MOb. У цьому режимі налаштовується лише один набір MOb, включаючи непослідовні MOb. Коли всі MOb набору отримають належні кадри буде встановлено біт завершення прийому мультикадрів в буфер – BXOK регістра CANGIT. При цьому можливе переривання, якщо воно не замасковано.

Робота CAN-модуля в цьому режимі відбувається у наступній послідовності:

1. MOb повинні бути ініціалізовані як MOb в Rx-конфігурації.

2. MOb будуть готові отримувати кадри даних, коли встановлено їх відповідну конфігурацію (табл. 14.3, див. вище).

3. Коли ідентифікатор кадру отримано з CAN-мережі, CAN-модуль перебирає всі MOb, намагаючись знайти відповідний MOb з найбільшим пріоритетом.

4. IDT-, IDE- та DLC-поля відповідного MOb оновлюються значеннями з прийнятого кадру.

5. Як тільки прийом закінчено, байти даних отриманого повідомлення зберігаються (не для кадру віддаленого запиту) в буфері даних відповідного MOb і встановлюється біт RXOK регістра CANSTMOB. В цьому випадку можливе переривання, якщо воно не замасковано.

6. Коли прийом для останнього MOb з набору завершено, встановлюється біт завершення прийому кадрів в буфер BXOK (регістр CANGIT). В цьому випадку можливе переривання, якщо воно не замасковано.

7. Біт BXOK може бути очищено тоді, коли біти CONMOB0, 1 регістра CANCDMOB для всіх MOb набору було перезавантажено.

8. Всі параметри і дані залишаються доступними в MOb до наступної ініціалізації.

У [8] наведено схему алгоритму роботи CAN-модуля у цьому режимі та керучу програму мовою С.

14.4.5. Структура блоку фільтрації

Одним з важливих вузлів CAN-модуля є блок фільтрації повідомлень (рис. 14.10).

У цьому блоці ідентифікатор вхідного повідомлення побітно підсумовується за модулем два з регістром фільтру, який складається з 13/32 бітів регістрів CANIDT1, CANIDT2, CANIDT3, CANIDT4 (залежно від формату кадру – версія 2.0 A/B) і біта IDE регістра CANCDMOB. Після цього результат інвертується і на нього накладається маска з регістрів масок ідентифікатора CANIDM1, CANIDM2, CANIDM3, CANIDM4. Якщо після цих операцій ми отримуємо послідовність одиничних біт, то повідомлення відповідає вимогам фільтрації та записується у відповідний об'єкт повідомлення (MOb).

Маска застосовується для того, щоб вказати які біти фільтра будуть використовуватися для перевірки ідентифікатора вхідного повідомлення. Якщо біт маски дорівнює нулю, то відповідний біт ідентифікатора повідомлення буде прийнято незалежно від значення біта фільтра.

Нижче наведено два приклади роботи блоку фільтрації для прийому стандартних повідомлень.

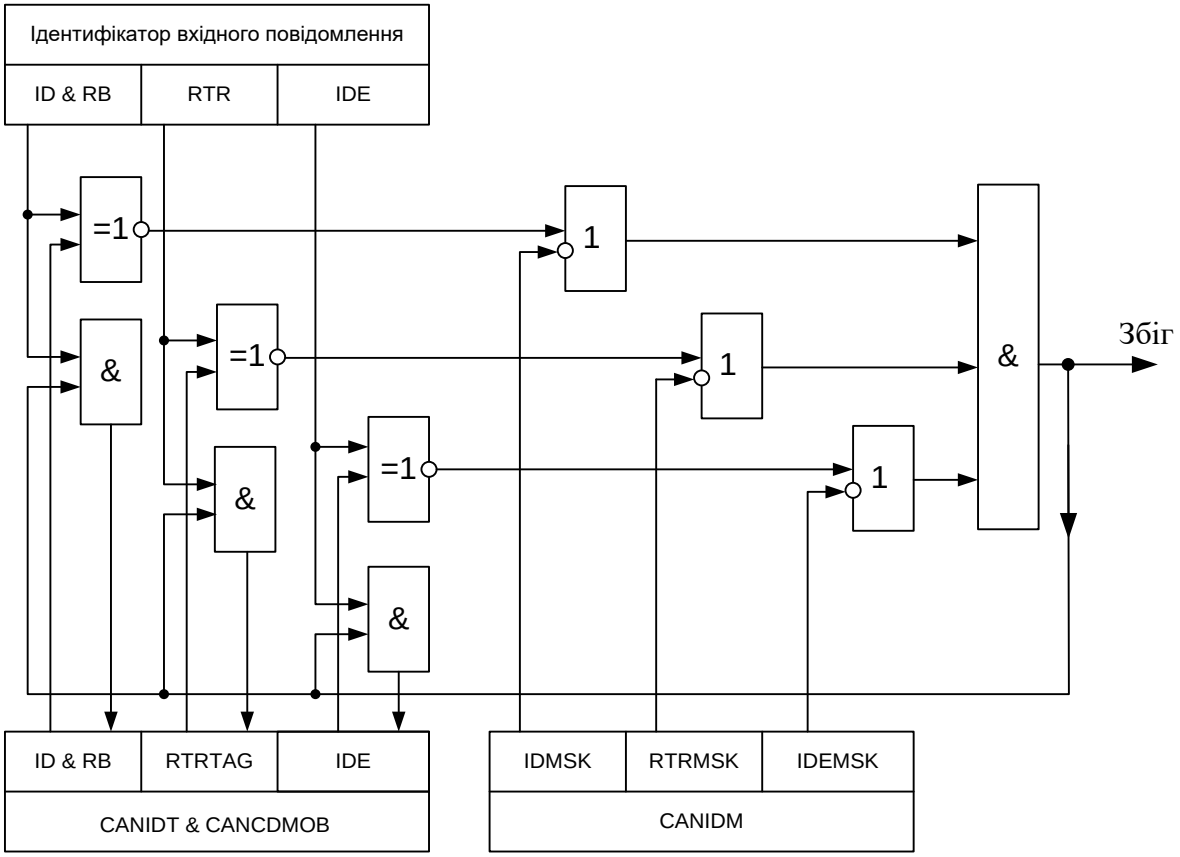


Рис. 14.10. Структура блоку фільтрації повідомлень

Приклад1: Повна фільтрація

Прийом тільки ID = 0x317:
ID MSK = 111 1111 1111 b;
ID TAG = 011 0001 0111 b.

Приклад2: Часткова фільтрація

Прийом ID від 0x310 до 0x317:
ID MSK = 111 1111 1000 b;
ID TAG = 011 0001 0xxx b.

14.4.6. Структура переривань від CAN-модуля

CAN-модуль підтримує кілька різних джерел переривань. Підпрограма обробки CAN-переривання від різних джерел має адресу 0x0024, а обробки переривання від переповнення CAN-таймера – адресу 0x0026. Всі переривання визначаються індивідуальними бітами дозволу. Щоб дозволити переривання, відповідний біт дозволу треба встановити в одиницю разом з бітом глобального дозволу переривань у регістрі статусу.

У CAN-модулі мікроконтролера AT90CAN128 можлива обробка переривань як для всіх повідомлень, так і для кожного конкретного об'єкта повідомлень (MOb).

Структуру переривань представлено на рис. 4.11.

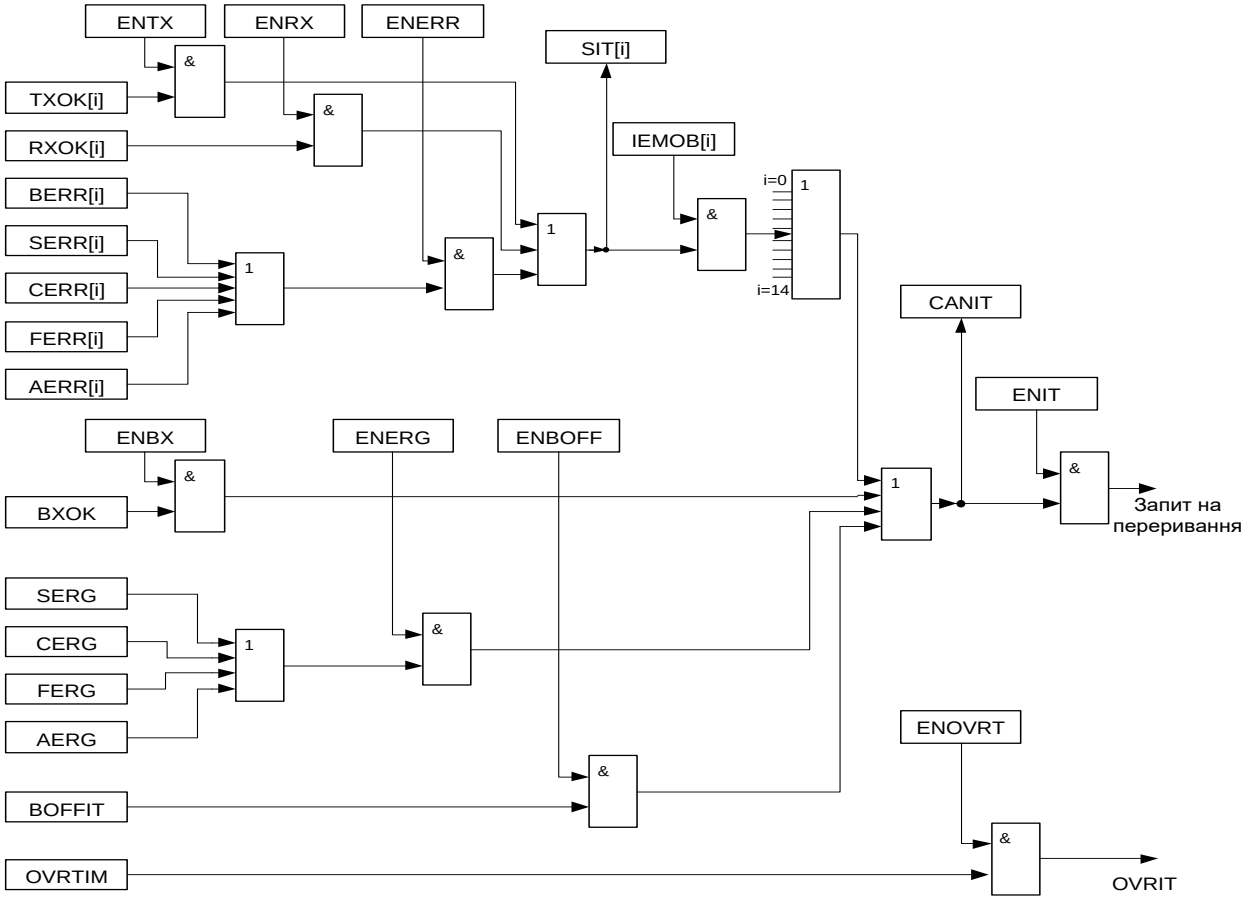


Рис. 14.11. Структура переривань мікроконтролера AT90CAN128

Існують такі переривання:

- успішне отримання повідомлення;
- успішне відправлення повідомлення;
- виявлено помилку;
- заповнення буфера зберігання кадрів;
- встановлення стану «відключення від шини»;
- переповнення лічильника CAN-таймера.

На виході логічного елемента І, який розташовано у правій частині рисунку, формується «Запит на переривання» від різних джерел. Цей запит має адресу в пам`яті програм – 0x0024.

Нижче наведено пояснення умовних позначень, які використано на рис. 14.11.

Регістр статусу об'єкта повідомлення – CANSTMOB:

- CANSTMOB.6 (TXOK) – успішна передача;
- CANSTMOB.5 (RXOK) – успішний прийом;
- CANSTMOB.4 (BERR) – помилка біта;
- CANSTMOB.3 (SERR) – помилка наповнення;
- CANSTMOB.2 (CERR) – помилка CRC;
- CANSTMOB.1 (FERR) – помилка форми;
- CANSTMOB.0 (AERR) – помилка підтвердження.

Загальний регістр переривань – CANGIT:

- CANGIT.7 (CANIT) – загальний прапорець переривань (відображає наявність будь-якого переривання крім OVRTIM);
- CANGIT.6 (BOFFIT) – прапорець переривання відключення від шини;
- CANGIT.5 (OVRTIM) – прапорець переповнення лічильника CAN-таймера;
- CANGIT.4 (BXOK) – прапорець отримання відповідного набору кадрів в буфер;
- CANGIT.3 (SERG) – прапорець помилки наповнення;
- CANGIT.2 (CERG) – прапорець CRC-помилки;
- CANGIT.1 (FERG) – прапорець помилки форми;
- CANGIT.0 (AERG) – прапорець помилки підтвердження.

Загальний регістр дозволу переривань – CANGIE:

- CANGIE.7 (ENIT) – дозвіл всіх переривань (крім переривання від переповнення CAN-таймера);
- CANGIE.6 (ENBOFF) – дозвіл переривання від’єднання від шини;
- CANGIE.5 (ENRX) – дозвіл переривання прийому;
- CANGIE4 (ENTX) – дозвіл переривання передачі;
- CANGIE.3 (ENERR) – дозвіл переривань від “помилки MOb”;
- CANGIE.2 (ENBX) – дозвіл переривання від отримання відповідного набору кадрів в буфер (BXOK);
- CANGIE.1 (ENERG) – дозвіл переривань від загальних помилок;
- CANGIE.0 (ENOVRT) – дозвіл переривання від переповнення лічильника CAN-таймера.

Регістри дозволу переривань від MOb – CANIE1 і CANIE2:

– відповідні біти (IEMOB[i]), які встановлено в одиницю, показують, для яких MOb дозволено переривання. Регістри мають 15 біт, що відповідає кількості об'єктів повідомлення.

При виникненні загального переривання встановлюється в одиницю відповідний прапорець (біт) в регістрі CANGIT. Якщо також в одиницю встановлено "дозволяючий біт" в регістрі CANGIE, який відповідає прапорцю загального переривання, то встановлюється в одиницю загальний прапорець переривань регістра CANGIT – CANGIT.7 (CANIT).

При виникненні переривання від одного з MOb встановлюється в одиницю відповідний біт в регістрі CANSTMOB. Встановлення в одиницю прапорця переривань CANIT в цьому випадку відбудеться тільки при встановленому "дозволяючому біті" для даного переривання в регістрі CANGIE (CANGIE.3; CANGIE.4; CANGIE.5) та встановленому біті, що дозволяє переривання від даного MOb, в регістрах CANIE1 або CANIE2.

Для правильного визначення переривань від MOb та загальних переривань спочатку повинні бути очищені відповідні біти регістрів CANSTMOB та CANGIT.

14.4.7. Структура блоку CAN-таймера

CAN-модуль має блок таймера, структуру якого наведено на рис. 14.12.

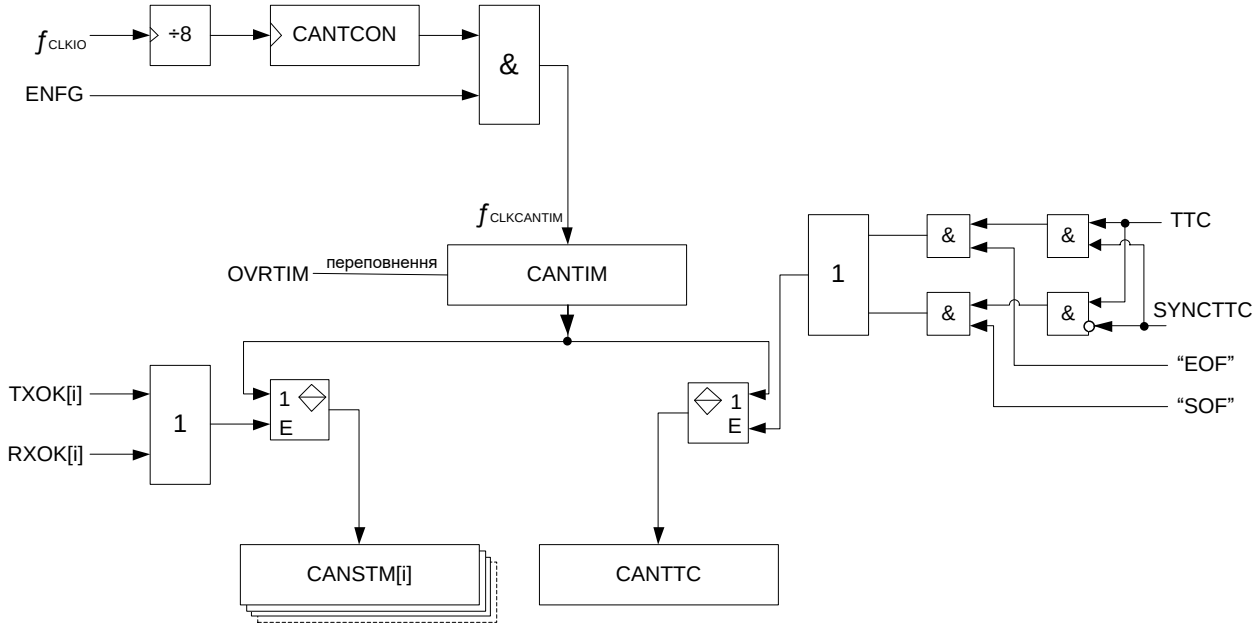


Рис. 14.12. Структура блоку CAN-таймера

Нижче описано призначення основних вузлів блоку.

Основним елементом блоку є 16-бітний лічильник – CANTIM. При встановленні в одиницю біта ENA/$\overline{STB}$ регістра CANGCON через деякий час CAN-модуль переходить в активний стан, що відображає встановлення в одиницю біта ENFG регістра CANGSTA. Після цього на вхід лічильника CANTIM починає надходити послідовність імпульсів тактової частоти від підсистеми введення/виведення МК. Ця частота ділиться апаратно на вісім, а потім проходить через восьмирозрядний програмований дільник – CANTCON. В результаті роботи цих дільників період частоти на вході CANTIM встановлюється відповідно до наступного виразу:

$$T_{CLKCANTIM} = T_{CLKIO} \cdot 8 \cdot \left(CANTCON\left[7:0\right] + 1 \right),$$

де T_{SCKIO} – період тактової частоти підсистеми введення/виведення МК [1; 7].

Під впливом цієї частоти CANTIM починає рахувати зі значення 0x0000. Коли відбувається його переповнення і перехід зі значення 0xFFFF на 0x0000, генерується переривання – OVRTIM = 1. Підпрограма обробки цього переривання має адресу 0x0026.

Кожен MOb має 16-розрядний регістр часових відміток – CANSTM, в якому за подіями: RXOK – успішний прийом або TXOK – успішна передача (регістр CANSTMOB) відбувається збереження значення CANTIM.

У CAN-мережі можлива взаємодія за стандартом TTCAN (Time-Triggered CAN – CAN, керована часом). Цей стандарт дозволяє підвищити ефективність використання пропускної здатності шини та організувати взаємодію вузлів в мережі таким чином, що зникає невизначеність у часі передачі повідомлень, яка існує у CAN. Наприклад, повідомлення з низьким пріоритетом може бути надіслано не з першого разу через втрату арбітражу вузлом, що намагається його передати і завчасно невідомо, чи це трапиться. У режимі TTCAN кожен вузол передає повідомлення у певний виділений для нього часовий інтервал, інші вузли в цей час не намагатимуться передавати дані. Це дозволяє уникнути колізій і втрати арбітражу. Також у цьому режимі передавач не намагається автоматично повторно передати повідомлення у разі виникнення помилки, оскільки тоді він буде використовувати не свій часовий інтервал, що у цьому режимі неприпустимо.

Режим TTC програмується встановленням в одиницю біта TTC регістра CANGCON. У цьому режимі кадр надсилається один раз, навіть якщо відбулася помилка.

Для TTC передбачено два режими синхронізації:

– синхронізація за початком кадру (SYNCTTC = 0, SOF = 1);

– синхронізація за кінцем кадру (SYNCTTC = 1, EOF = 1).

Якщо встановлено режим TTC (TTC = 1), а біт SYNCTTC = 0, то застосовується синхронізація за початком кадру – SOF (Start of Frame)) і при виявленні відповідного поля, яке складається з одного нульового біта, значення з регістра таймера CANTIM записується в регістр CANTTC.

Якщо встановлено режим TTC (TTC = 1), а біт SYNCTTC = 1, то застосовується синхронізація за кінцем кадру – EOF (End of Frame)) і при виявленні останнього одиничного біта поля "кінець кадру", яке складається з 7 одиничних біт, значення з регістра таймера CANTIM записується в регістр CANTTC.

Для того, щоб мікроконтролер міг брати участь у роботі CAN-шини, де використовується режим TTCAN, необхідно додатково програмно реалізовувати відповідні алгоритми взаємодії [19].

14.4.8. Обробка помилок

Існує 5 різних типів помилок, які можуть виникати одночасно.

Помилка біта

Вузол, що передає дані на шину, одночасно її контролює. Помилка біта виникає, якщо значення біта на шині відрізняється від значення, що передається. Виняток становить поле арбітражу, поле підтвердження та передачі додаткового 6-го біта протилежної полярності, після передачі п’яти біт однакової полярності.

Передавач, що посилає прапорець пасивної помилки і виявляє “домінантний” біт, не вважає, що це помилка біта.

Помилка заповнення

Помилка заповнення виникає, якщо при прийомі між початком повідомлення та роздільником CRC виявлено підряд 6 біт однакової полярності.

Помилка CRC

Послідовність CRC обчислюється передавачем і передається в кадрі повідомлення, а також обчислюється при прийомі повідомлення. Помилка CRC виникає, якщо значення, що обчислено приймачем, не збігається із прийнятим від передавача.

Помилка форми

Помилка форми виявляється, якщо бітове поле фіксованого формату (кінець повідомлення, роздільники CRC і підтвердження) містять один або більше

заборонених біти. “Домінантний” біт протягом останнього біта “кінець кадру” не розглядається, як помилка форми.

Помилка підтвердження

Помилка підтвердження виявляється передавачем щораз, коли він не виявляє “домінантний” біт в області підтвердження, що говорить про те, що жоден з вузлів не одержав повідомлення правильно. При цьому повідомлення передається повторно.

Лічильники помилок

В CAN-модулі для підрахунку помилок використовуються 8-розрядні лічильники помилок: CANTEC – лічильник помилок передачі та CANREC – лічильник помилок прийому.

В залежності від кількості помилок CAN-модуль може знаходитись в одному з трьох станів:

- активна помилка;
- пасивна помилка;
- відключення від шини (рис. 14.13).

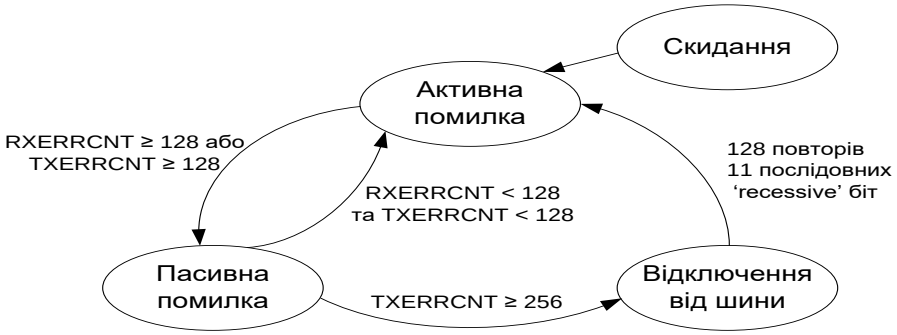


Рис. 14.13. Стан лічильників помилок

При обміні CAN-мережею стан лічильників змінюється згідно з правилами, які описано в [8].

14.4.9. Бітова синхронізація

Номінальна швидкість передачі та номінальна тривалість біта

Номінальна швидкість передачі інформації визначається числом бітів у секунду, які передаються ідеальним передавачем при відсутності пересинхронізації [8].

Номінальний час передачі біта = 1/номінальну швидкість передачі інформації в бітах в секунду.

Номінальний час передачі біта можна розділити на кілька ділянок, що не перекриваються (рис. 14.14):

- сегмент синхронізації (SYNC_SEG);
- сегмент часу розповсюдження (PROP_SEG);
- фазовий сегмент 1 (PHASE_SEG1);
- фазовий сегмент 2 (PHASE_SEG2).

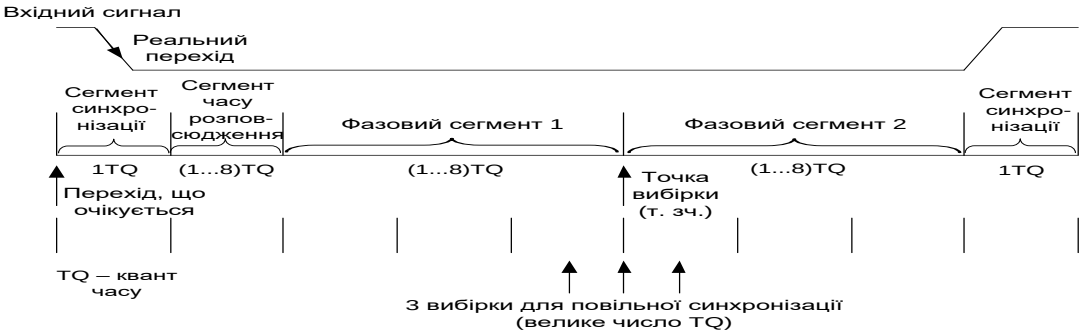


Рис. 14.14. Номінальний час біта

Кожен сегмент складається із цілого числа відрізків часу, які називаються квантами часу – TQ. Номінальний час передачі біта коливається від 8TQ до 25TQ (включно). При швидкості передачі 1 Мбіт/с мінімальний номінальний час передачі біта дорівнює 1мкс.

Тривалість одного кванта часу – T_{SCL} визначається тактовою частотою підсистеми введення/виведення мікроконтролера і значенням коефіцієнта ділення цієї частоти, що може змінюватися програмно:

$$T_{SCL} = (BRP[5:0] + 1)\cdot T_{CLKI/O},$$

де $T_{CLKI/O}$ – період тактової частоти підсистеми введення/виведення МК, BRP[5:0] – десятковий еквівалент шести біт регістра CANBT1.

Сегмент синхронізації (SYNC_SEG)

Сегмент синхронізації (SYNC_SEG) має довжину один квант часу – TQ та використовується для синхронізації різних вузлів на шині. Початок передачі визначає від`ємний фронт вхідного сигналу, що повинен перебувати в межах сегмента синхронізації.

Сегмент часу розповсюдження

Сегмент часу розповсюдження використовується, щоб компенсувати час фізичного запізнювання в мережі. Цей час дорівнює подвоєній сумі часу розповсюдження сигналу на лінії шини, затримки вхідного компаратора і затримки

вихідного формувача. Значення цього сегмента може програмуватися від 1TQ до 8TQ.

Фазові сегменти 1, 2

Ці сегменти використовуються, щоб компенсувати помилки фази (фазові зсуви) при прийомі. Фазовий сегмент 1 може бути подовжено, а фазовий 2 – вкорочено пересинхронізацією (ресинхронізацією) [8].

Фазові сегменти призначено для оптимального розташування точки вибірки отриманого біта у межах часу біта, що передається. Точка вибірки розташовується між фазовим сегментом 1 і фазовим сегментом 2 (рис. 14.14, див. вище).

Фазовий сегмент 1 визначає точку вибірки в межах біта, що передається. Фазовий сегмент 2 забезпечує затримку до початку наступного біта. Тривалість обох цих сегментів може програмуватися від 1TQ до 8TQ.

Точка вибірки

Точка вибірки (точка зчитування – т. зч.), (Sample point) – це момент часу, при якому рівень на шині читається та інтерпретується, як значення відповідного біта. Вона розташована наприкінці фазового сегмента 1 (рис. 14.14, див. вище).

Якщо біт має велику кількість TQ, то можна задати багаторазовий вибірковий контроль стану шини. В цьому випадку CAN-модуль з періодичністю TQ/2 робить вибірку три рази для кожного біта, який прийнято. Правильним вважається значення, яке отримано за мажоритарним принципом – два або три рази прийнято нуль, або одиницю. Увімкнення багаторазової вибірки здійснюється встановленням спеціального програмованого біта SMP регістра CANBT3.

Синхронізація

Для компенсації зсуву фази між частотами генераторів різних вузлів шини та при зміні ведучого при арбітражі, кожен CAN-модуль повинен синхронізуватися до переходу рівня вхідного сигналу з одиниці на нуль (рис. 14.14, див. вище). Коли перехід виявлено, то схема синхронізації порівнює розміщення переходу з переходом, що очікується при прийомі, та виконує налаштування значень фазових сегментів 1 та 2.

Є два механізми синхронізації:

- апаратна синхронізація;
- синхронізація з відновленням тактових інтервалів (пересинхронізація/ ресинхронізація).

Апаратна синхронізація

Апаратна синхронізація виконується при переході від “рецесивного” до “домінантного” біта протягом холостого стану шини, що вказує на початок повідомлення. При апаратній (жорсткій) синхронізації тактові інтервали (тривалість сегментів) не змінюються протягом усього повідомлення.

Після апаратної синхронізації внутрішній бітовий час кожного вузла перезапускається з сегмента синхронізації SYNC_SEG (рис. 14.14, див. вище).

Синхронізація з відновленням тактових інтервалів

Синхронізація з відновленням тактових інтервалів (пересинхронізація/ресинхронізація) призначена для зменшення фазових спотворень при прийомі та виконується автоматичним подовженням фазового сегмента 1 або скороченням фазового сегмента 2. Максимальне значення зміни фазових сегментів коливається в межах від 1TQ до 4TQ. Синхронізація виконується при переходах від “рецесивного” до “домінантного” біта протягом прийому кадру. Фіксоване значення максимального числа послідовних біт однакової полярності (“біт-стаффінг“) гарантує своєчасне відновлення синхронізації. В межах бітового інтервалу допускається тільки один тип пересинхронізації. Фронт сигналу буде використовуватися для пересинхронізації тільки тоді, якщо значення, яке виявлено при попередній точці зчитування (попереднє значення на шині), відрізняється від значення на шині відразу після фронту.

Фазове спотворення (помилка фази) – e визначається в квантах TQ як різниця часу точки зчитування (tт.зч.) бітового інтервалу приймача, протягом якого з’явився перехід вхідного сигналу з одиниці в нуль, та часу появи цього переходу: $t_{\downarrow}$ (рис. 14.4).

Значення та знак e визначається в такий спосіб:

- e = 0, якщо фронт вхідного сигналу (перехід з одиниці в нуль) перебуває в межах сегмента синхронізації SYNC_SEG (ресинхронізація не проводиться);
- e > 0, якщо фронт вхідного сигналу (перехід з одиниці в нуль) не перебуває в межах SYNC_SEG та знаходиться перед точкою зчитування бітового інтервалу приймача, протягом якого з’явився перехід вхідного сигналу з одиниці в нуль;
- e < 0, якщо фронт сигналу (перехід з одиниці в нуль) не перебуває в межах SYNC_SEG та знаходиться після точки зчитування бітового

інтервалу приймача, протягом якого з'явився перехід вхідного сигналу з одиниці в нуль.

Ефект від пересинхронізації – такий самий як і у випадку апаратної синхронізації, коли величина помилки фази е менше або дорівнює значенню ширини періоду пересинхронізації, що програмується.

Коли величина помилки фази е більша, ніж ширина переходу пересинхронізації, що програмується, тоді:

- якщо помилка фази е – додатна, то фазовий сегмент 1 подовжується на ширину періоду пересинхронізації, що програмується;
- якщо помилка фази е – від'ємна, то фазовий сегмент 2 скорочується на ширину періоду пересинхронізації, що програмується.

На рис. 14.15, 14.16 наведено два приклади пересинхронізації.

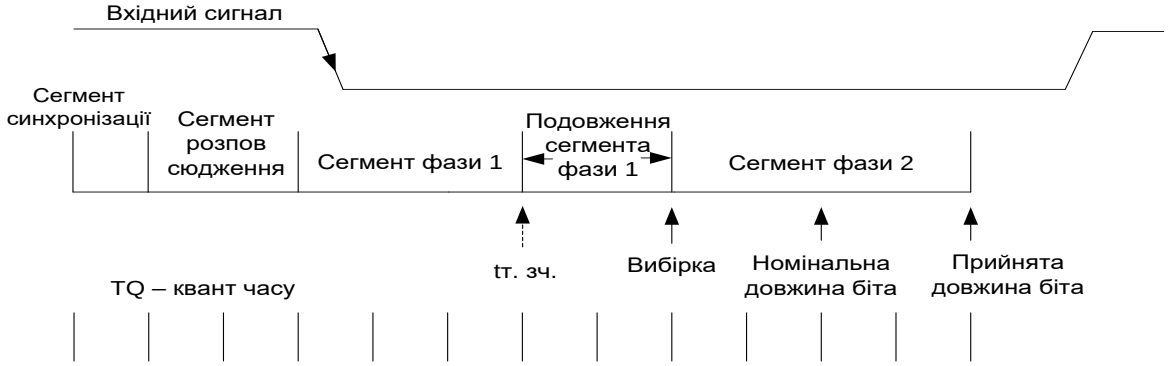


Рис. 14.15. Синхронізація з відновленням тактових інтервалів за рахунок автоматичного подовження фазового сегмента 1

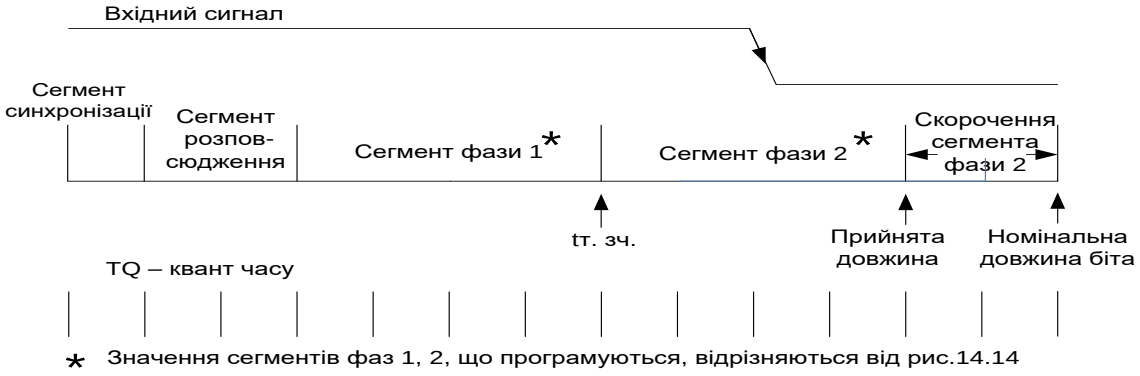


Рис. 14.16. Синхронізація з відновленням тактових інтервалів за рахунок автоматичного скорочення фазового сегмента 2

Приклад програмування швидкості обміну інформацією CAN-мережею

Зміст завдання

Написати мовою Асемблер фрагмент програми, який забезпечує потрібну швидкість обміну інформацією CAN-мережею для AVR-мікроконтролера AT90CAN128, розрахувати тривалість одного біта $t_{\text{біта}}$ та тривалість одного кванта часу $T_{SCL} = T_{TQ}$.

Вихідні дані:

– тактова частота підсистеми введення/виведення МК:

$$f_{CLKIO} = 10 \text{ МГц};$$

– довжина періоду ресинхронізації: $t_{\text{п.р.}} = T_{SJW} = 1T_{SCL}$;

– тривалість сегменту часу розповсюдження: $t_{\text{с.р.}} = T_{PRS} = 5T_{SCL}$;

– довжина фазового сегмента 2: $T_{PHS2} = 5T_{SCL}$;

– довжина фазового сегмента 1: $T_{PHS1} = 5T_{SCL}$;

– використовувати одноразову вибірку при прийомі кожного біта;

– швидкість обміну $V_{\text{пд}} = 10$ Кбіт/с.

Рішення завдання

Розрахуємо тривалість одного біта: $t_{\text{біта}} = \frac{1}{V_{\text{пд}}} = \frac{1}{10000} = 100$ мкс.

Згідно рис. 14.14 та вихідними даними, які наведено вище, номінальний час одного біта має наступне число квантів часу:

$$n_{\text{кв.ч}} = 1TQ + 5TQ + 5TQ + 5TQ = 16TQ = 16T_{SCL}.$$

Тоді потрібна тривалість одного кванта часу: $T_{SCL} = \frac{\text{t}_{\text{біта}}}{\text{n}_{\text{кв.ч}}} = \frac{100}{16} = 6.25$ мкс.

Тривалість одного кванта часу: T_{SCL} = (BRP[5:0] + 1)·T_{CLKIO},
де T_{CLKIO} – період тактової частоти підсистеми введення/виведення МК,
BRP[5:0] – десятковий еквівалент шести біт регістра CANBT1 [8].

Відповідно останній формулі розраховуємо потрібне значення:

$$\text{BRP}[5:0] = \frac{T_{SCL}}{T_{CLKIO}} - 1 = \frac{6.25}{0.1} - 1 = 62.5 - 1 = 61.5,$$

$$\text{де } T_{CLKIO} = \frac{1}{f_{CLKIO}} = \frac{1}{10*10^6 \text{ Гц}} = 0{,}1_{\text{МКС.}}$$

Приймаємо BRP[5: 0] = 61 = 111101b.

Нижче наведено керуюче слово КС1 (8 біт), яке потрібно завантажити у регістр CANBT1 для отримання розрахованного вище значення BRP[5: 0]:

7р	6р	5р	4р	3р	2р	1р	0р
–	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0	–
0	1	1	1	1	0	1	0

Для програмування потрібних значень tп.р. = T_{SJM}; tс.р. = T_{PRS}; $t_{фс2}$ = T_{PHS2} та $t_{фс1}$ = T_{PHS1} використовуються регістри CANBT2 та CANBT3 [8], в які треба завантажити керуючі слова КС2 та КС3, які наведено нижче.

КС2:

7р	6р	5р	4р	3р	2р	1р	0р
–	SJW1	SJW0	–	PRS2	PRS1	PRS0	–
0	0	0	0	1	0	0	0

КС3:

7р	6р	5р	4р	3р	2р	1р	0р
–	PHS22	PHS21	PHS20	PHS12	PHS11	PHS10	SMP
0	1	0	0	1	0	0	0

При формуванні КС2 та КС3 треба враховувати, що потрібні значення T_{SJM}, T_{PRS}, T_{PHS2} та T_{PHS1} зв'язані з відповідними розрядами регістрів CANBT2 та CANBT3 виразами, які наведено нижче [8]:

$$T_{SJW} = T_{SCL}(\mathrm{SJW}[1:0] + 1) = \mathrm{t}_{\text{п.р.}};$$

$$T_{PRS} = T_{SCL}(\mathrm{PRS}[2:0] + 1) = t_{c.p.};$$

$$T_{PHS2} = T_{SCL}(\mathrm{PHS2}[2:0] + 1) = \mathrm{t}_{\text{фс2}};$$

$$T_{PHS1} = T_{SCL}(\mathrm{PHS1}[2:0] + 1) = \mathrm{t}_{\text{фс1}}.$$

Біт SMP має значення нуль, тому що в завданні було вказано, що треба використовувати одноразову вибірку при прийомі кожного біта. Якщо в завданні буде вказано, що значення кожного біта при прийомі зчитується тричі, один раз в точці вибірки і по одному разу на відстані 1/2 тривалості T_{SCL}, тоді біт SMP треба встановити у одиницю.

Згідно [8] регістри CANBT1, CANBT2 та CANBT3 мають відповідно наступні адреси у просторі адрес регістрів введення/виведення МК AT90CAN128: $00E2; $00E3 та $00E4.

Нижче наведено фрагменти програми мовою Асемблер, які програмують потрібну швидкість обміну CAN-мережею.

Програмування регістра CANBT1:

```
LDI  R18, $7A;        R18 <- КС1 = 01111010b = $7A;
LDI  R27, $00;        R27 <- СБ адреси  CANBT1;
LDI  R26, $E2;        R26 <- МБ адреси CANBT1;
ST  X, R18;           CANBT1 <- R18 = $7A;
```

Програмування регістра CANBT2:

```
LDI  R17, $08;        R17 <- КС2 = 00001000b = $08;
LDI  R27,$00;         R27 <- СБ адреси  CANBT2;
LDI  R26, $E3;        R26 <- МБ адреси  CANBT2;
ST  X, R17;           CANBT2 <- R17 = $08.
```

Програмування регістра CANBT3:

```
LDI  R16, $48;        R16 <- КС3 = 01001000b = $48;
LDI  R27, $00;        R27 <- СБ адреси  CANBT3;
LDI  R26, $E4;        R26 <- МБ адреси  CANBT3;
ST  X, R16;           CANBT3 <- R16 = $48.
```

14.5. Фізичний рівень CAN-протоколу

14.5.1. Загальні відомості

Фізичний рівень (Physical Layer) CAN-протоколу визначає опір кабелю, рівень електричних сигналів у мережі і т. ін. Існує кілька фізичних рівнів CAN-протоколу: ISO 11898, ISO 11519, SAE J2411 і т. ін. [1; 19]. Серед останніх часто використовується фізичний рівень, який описано у стандарті ISO 11898. Цей рівень в якості середовища передачі визначає двопровідну диференціальну лінію з резисторами-термінаторами на кінцях зі значенням 120 Ом. Допускається коливання цього значення в межах від 108 Ом до 132 Ом.

Швидкість передачі CAN-мережею залежить від довжини кабелю. Це пов'язано з кінцевою швидкістю світла та механізмом побітового арбітражу. Під час останнього всі вузли мережі повинні отримувати поточний переданий біт практично одночасно, тобто сигнал в мережі повинен встигнути поширитися по всьому кабелю за один квант часу.

Залежність між швидкістю передачі та максимальною довжиною кабелю відображено у табл. 14.4.

Таблиця 14.4. **Залежність швидкості передачі даних від довжини шини**

Швидкість передачі	Час передачі біта	Довжина лінії зв'язку
1 Мбіт/с	1 мкс	30 м
800 Кбіт/с	1,25 мкс	50 м
500 Кбіт/с	2 мкс	100 м
250 Кбіт/с	4 мкс	250 м
125 Кбіт/с	8 мкс	500 м
62,5 Кбіт/с	20 мкс	1000 м
20 Кбіт/с	50 мкс	2500 м
10 Кбіт/с	100 мкс	5000 м

При довжині кабелю 30 метрів швидкість передачі даних CAN-мережею буде максимальною та дорівнює 1Мбіт/с.

При необхідності передавати дані на більші відстані рекомендується використовувати вузли-ретранслятори.

14.5.2. Трансивери для CAN-мережі

Для організації дротового з'єднання на CAN-шині найбільш широке поширення отримали два типи приймачів/передавачів (трансиверів): «High Speed» (ISO 11898-2) та «Fault Tolerant» [21…23].

Трансивери, які виконано у відповідності зі стандартом «High Speed» (ISO 11898-2), найбільш прості, дешеві та надають можливість передавати дані зі швидкістю до 1 Мбіт/с. «Fault Tolerant» приймачі-передавачі не дуже чутливі до ушкоджень на шині та дозволяють побудувати високонадійну мережу, що мало споживає, зі швидкостями передачі даних не вище 125 Кбіт/с.

Організація фізичного середовища передачі даних повинна задовольняти набору обов'язкових вимог, які продиктовано способами арбітражу та синхронізації для CAN-інтерфейсу. Фізичне середовище передачі даних повинно мати можливість перебувати у двох станах – «рецесивному» та «домінантному». CAN-шина перебуває в «рецесивному» стані, якщо жоден вузол не передає «домінантний» біт,

і в «домінантному» стані, якщо хоча б один вузол встановив «домінантний» біт. Іншими словами на шині не повинен відбуватися конфлікт, якщо одночасно кілька пристроїв виставили різні логічні рівні. На шині в цьому випадку повинен встановлюватися «домінантний» стан.

Максимальне число вузлів, підключених до шини, фактично визначається здатністю навантаження застосованих трансиверів. Наприклад, при використанні мікросхеми PCA82C250 від компанії NXP (цю мікросхему було розроблено також компанією Philips Semiconductors) воно дорівнює 110. Цей трансивер відноситься до стандарту «High Speed».

Широке поширення отримали трансивери «High Speed» стандарту ISO 11898-2. Прикладом такого трансивера є мікросхема PCA82C250. Його з`єднання із CAN-шиною показано на рис. 14.17.

Напруга живлення цього трансивера становить 5 В. Рівні напруг, що відповідають «рецесивному» й «домінантному» для стандарту ISO 11898-2 показано на рис. 14.18.

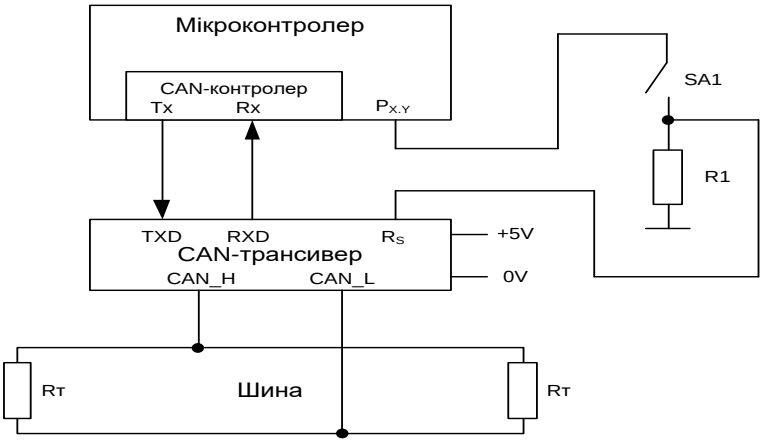


Рис. 14.17. З'єднання із шиною «High Speed»-трансивера

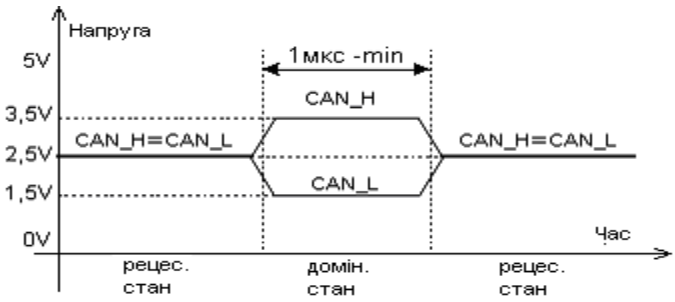


Рис. 14.18. Фізичні рівні "high speed"-трансиверів

Логічному нулю відповідає високий рівень сигналу (приблизно 3,5 В) на лінії CAN-H і низький рівень (приблизно 1,5 В) на лінії CAN-L, тобто між лініями

CAN-H і CAN-L присутня різниця потенціалів приблизно 2В. Логічній одиниці відповідають рівні напруги (приблизно 2,5 В) на обох лініях, тобто між лініями CAN-H і CAN-L присутня різниця потенціалів приблизно 0В.

Є три режими роботи трансивера, які налаштовуються ззовні через вивід Rs (рис. 14.17):

1. Високошвидкісний режим – High-Speed.
2. Режим регулювання швидкості наростання сигналу, що передається в CAN-шину – Slope-Control.
3. Режим очікування – Standby.

Високошвидкісний режим вибирається підключенням виводу Rs до VSS – 0В. У цьому режимі вихідні драйвери мають швидкий час наростання і спаду, що забезпечує швидкість передачі до 1 Мбіта/с.

Якщо потрібно зменшити випромінювані драйвером електромагнітні завади, PCA82C250 можна встановити в режим регулювання швидкості наростання сигналу, що передається в CAN-шину. Для цього треба підключити резистор R1 від виводу Rs на спільний мінус (рис. 14.17) [22]. Зменшення швидкості наростання вихідної напруги призводить до зменшення швидкості передачі даних CAN при заданій довжині шини, або до скорочення довжини шини при заданій швидкості передачі даних.

Режим очікування (режим сну – Sleep) встановлюється під’єднанням виводу Rs до VDD – +5В. На рис. 14.17 для цього використано перемикач SA1 та лінія PX.Y одного із вільних портів введення-виведення МК. У режимі сну передавач від’єднано, а приймач працює в режимі зниженого енергоспоживання. Вивід RxD, який приймає, як і раніше функціонує, але на нижчій швидкості.

Режим очікування можна використовувати для встановлення пристрою в режим низького енергоспоживання і від’єднання передавача у випадку, якщо CAN- модуль несправний і видає на шину непередбачувані дані.

Інший тип трансивера – «Fault Tolerant» орієнтовано на застосування в легкових автомобілях, де не потрібна висока швидкість передачі даних, але велике значення мають такі якості шини, як висока надійність і мінімальне енергоспоживання. Трансивер «Fault Tolerant» може бути виконано, наприклад, на мікросхемі Motorola MC33388 [23].

14.5.3. Роз'єми для CAN-мережі

Міжнародна організація CiA (The CAN in Automation international users and manufacturers group) рекомендує у своєму стандарті CiA DS-102 для з’єднання окремих вузлів з CAN-шиною використовувати наведену на рис. 14.19 розводку

виводів 9-контактного роз'єму D-Sub. Дану угоду використовують деякі протоколи більш високих рівнів CAN (CANopen, Smart Distributed System).

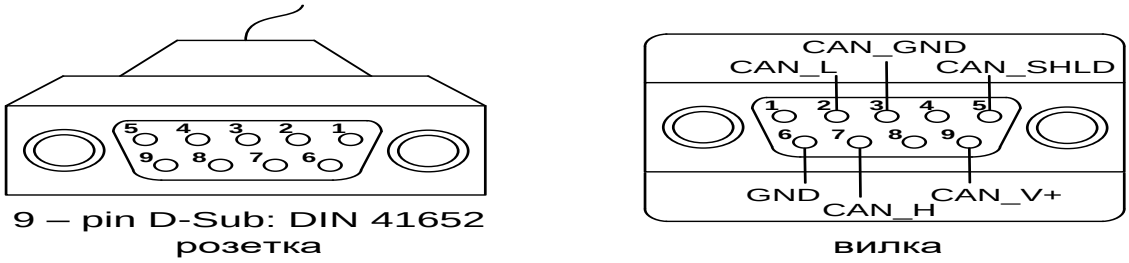


Рис.14.19. CAN-роз'єм згідно зі стандартом CiA DS-102

Призначення виводів CAN-роз'єму згідно зі стандартом CiA DS-102 наведено у табл. 14.5.

Таблиця 14.5. **Призначення виводів CAN-роз'єму згідно зі стандартом CiA DS-102**

Контакт	Ланцюг	Примітка
1	–	–
2	CAN_L	Низький рівень – домінантний стан
3	CAN_GND	Земля
4	–	–
5	(CAN_SHLD)	Екран (необов'язковий)
6	GND	Земля (необов'язковий)
7	CAN_H	Високий рівень – рецесивний стан
8	–	–
9	(CAN_V+)	Живлення (необов'язковий)

14.6. Принципова схема вузла CAN-мережі

Принципову схему вузла CAN-мережі, наведено на рис. 14.20. Основним елементом схеми є сучасний мікроконтролер AT90CAN128, до складу якого входить CAN-модуль, що підтримує протокол CAN 2.0A/B.

Мікроконтролер AT90CAN128 є досить потужним пристроєм, здатним виконувати безліч різноманітних функцій, пов'язаних з обробкою та керуванням обміном даними в різних протоколах з безліччю периферійних (зовнішніх) пристроїв.

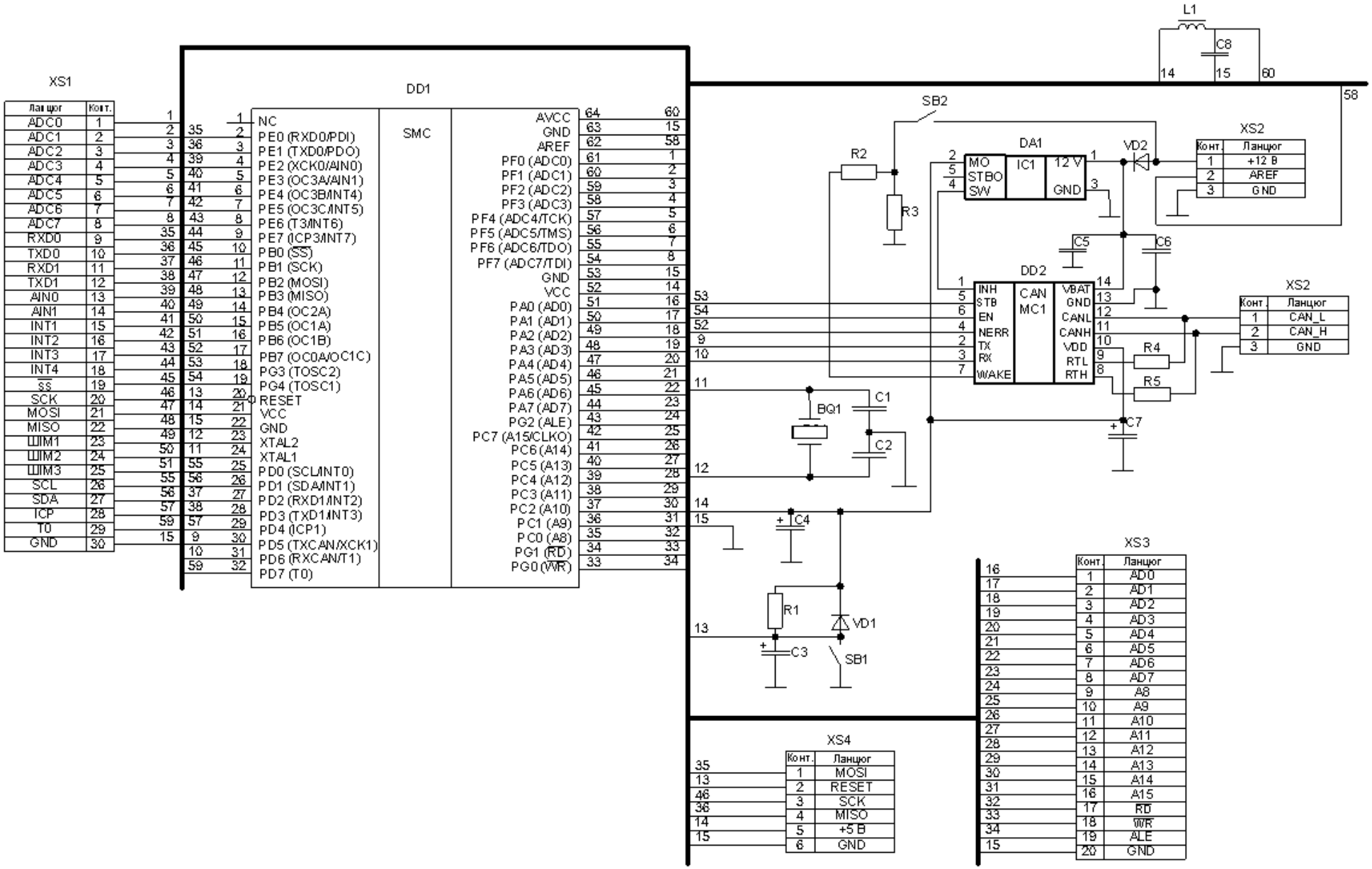


Рис.14.20. Принципова схема вузла CAN-мережі

Лінії TXCAN й RXCAN мікроконтролера є відповідно виходом та входом CAN-модуля мікроконтролера. Вони підключаються до відповідних ліній трансивера.

Для керування передачею даних CAN-мережею на фізичному рівні, використовується трансивер типу «Fault Tolerant», в якості якого в роботі обрано мікросхему Motorola MC33388 (DD2).

Мікросхема може керувати зовнішнім стабілізатором напруги (DA1) через вивід INH. Це дозволяє перемикати пристрій в режим зниженого енергоспоживання. Вихід з цього стану відбувається при виявленні активності на шині у «сплячому» режимі або здійснюється за допомогою ключа SB2.

Трансивер має чотири режими роботи, що програмно задаються: нормальний, тільки прийом, живлення від батарейки (Stanby VBAT) і сну (Sleep). Перехід в один з названих режимів здійснюється подачею керуючих сигналів на виводи STB та EN (DD2), які пов'язано з виходами PG3, PG4 порту G МК (DD1).

У нормальному режимі всі функції трансивера доступні та вихід NERR трансивера, який зв'язано із входом PB7 МК, сигналізує про пошкодження шини. У випадку відсутності пошкодження на виході NERR присутній високий рівень напруги (логічна одиниця).

В режимі «Тільки прийом» частина, яка передає, відключена, тому пристрій не керує шиною, а функції прийому здійснюються як у нормальному режимі.

В режимі «Sleep» функції передачі та прийому відключено. Вивід CANL через опір R4 і внутрішній опір трансивера підключено до VBAT. Зовнішній стабілізатор напруги (DA1) низьким рівнем напруги, що з'являється на виході INH (DD2), вимикається, і на вхід VDD трансивера (DD2) не подається напруга живлення.

MC33388 відслідковує активність на шині та сигнал на вході WAKE, що може бути сформований за допомогою зовнішнього перемикача SB2. Якщо виконуються умови виходу зі стану «Sleep», пристрій переходить у режим Stanby VBAT, схожий на режим «Sleep», але на виході INH підтримується високий рівень, відповідно забезпечується зовнішня напруга живлення для трансивера: +5В.

Згідно з принциповою схемою, яка розглядається, окрім CAN-модуля, можна використовувати інші перефирійні модулі МК:

- аналого-цифровий перетворювач (АЦП). Вхідні сигнали на АЦП подаються з контактів 1…8 роз’єму XS1 на входи ADC0…ADC7 МК;
- універсальний асинхронний приймач-передавач1 (УАПП1), (лінії RXD0, TXD0);

– універсальний асинхронний приймач-передавач2 (УАПП2), (лінії RXD1, TXD1);
– аналоговий компаратор (АК), (лінії AIN0, AIN1);
– чотири зовнішніх переривання (лінії INT4, INT5, INT6, INT7);
– синхроний послідовний інтерфейс (SPI), (лінії $\overline{SS}$, SCK, MOSI, MISO);
– трьохканальний широтно-імпульсний модулятор (ШІМ), (лінії OC2A, OC1A, OC1B);
– синхроний послідовний інтерфейс (I2C), (лінії SCL, SDA);
– таймер T1, який використовується у режимі “Захоплення” (лінія ICP1);
– таймер T0, який використовується для підрахунку зовнішніх подій (лінія T0);
– лінії AD0… AD7, A8… A15, ALE, $\overline{RD}$, $\overline{WR}$ можуть використовуватися для підключення до МК зовнішньої пам’яті даних, об’ємом 64 Кбайт.

До виводів XTAL1, XTAL2 МК-ра підключається кварцовий резонатор, що задає тактову частоту роботи мікроконтролера f_{CLK}, яка дорівнює 16 МГц.

Для підвищення стабільності частоти, яка формується генератором тактових імпульсів, у схемі використовуються два конденсатори однакової ємності. Значення конденсаторів обрано відповідно до рекомендацій, які наведено в [1] і становлять 30 пФ.

Одним з функціональних елементів розглянутої принципової схеми є вузол скидання, що формує зовнішнє скидання ($\overline{RESET}$), який подається на вхід RESET МК у нормальному режимі роботи.

Основним елементом цього вузла є інтегруючий RC-ланцюг, виконаний на резисторі R1 і конденсаторі C3. В початковому стані (до подачі напруги живлення) ємність C3 розряджена. При увімкненні напруги VDD = +5В, ця ємність не може миттєво зарядитися і на вхід RESET (СКИДАННЯ) мікроконтролера надходить низький рівень напруги. Через заряджання конденсатора C3 через резистор R1 напруга на вході RESET збільшується. В середині МК на вході лінії RESET включено пороговий елемент (тригер Шмітта), що спрацьовує після того, як напруга на конденсаторі досягне порогу спрацьовування цього тригера. У момент спрацьовування тригера дія сигналу «$\overline{RESET}$» припиняється і мікроконтролер переходить у нормальний режим роботи.

Для виконання операції «СКИДАННЯ» МК при увімкненій напрузі живлення VDD необхідно короткочасно натиснути кнопку SB1. При цьому ємність C3 швидко розрядиться та при відпусканні кнопки повторно відбудеться скидання МК. Діод VD1 служить для швидкого розрядження ємності C3 при

вимиканні напруги живлення та розімкнутій кнопці SB1. Це розрядження здійснюється через внутрішній опір джерела живлення.

14.7. Тлумачний словник основних загальноприйнятих термінів CAN-технології

Тлумачний словник основних загальноприйнятих термінів CAN-технології наведено у [20].

14.8. Схема алгоритму роботи CAN-модуля

Схему алгоритму роботи CAN-модуля наведено на рис. 14. 21.

14.9. Моделювання CAN-модуля

Опис моделі CAN-модуля наведено у [8].

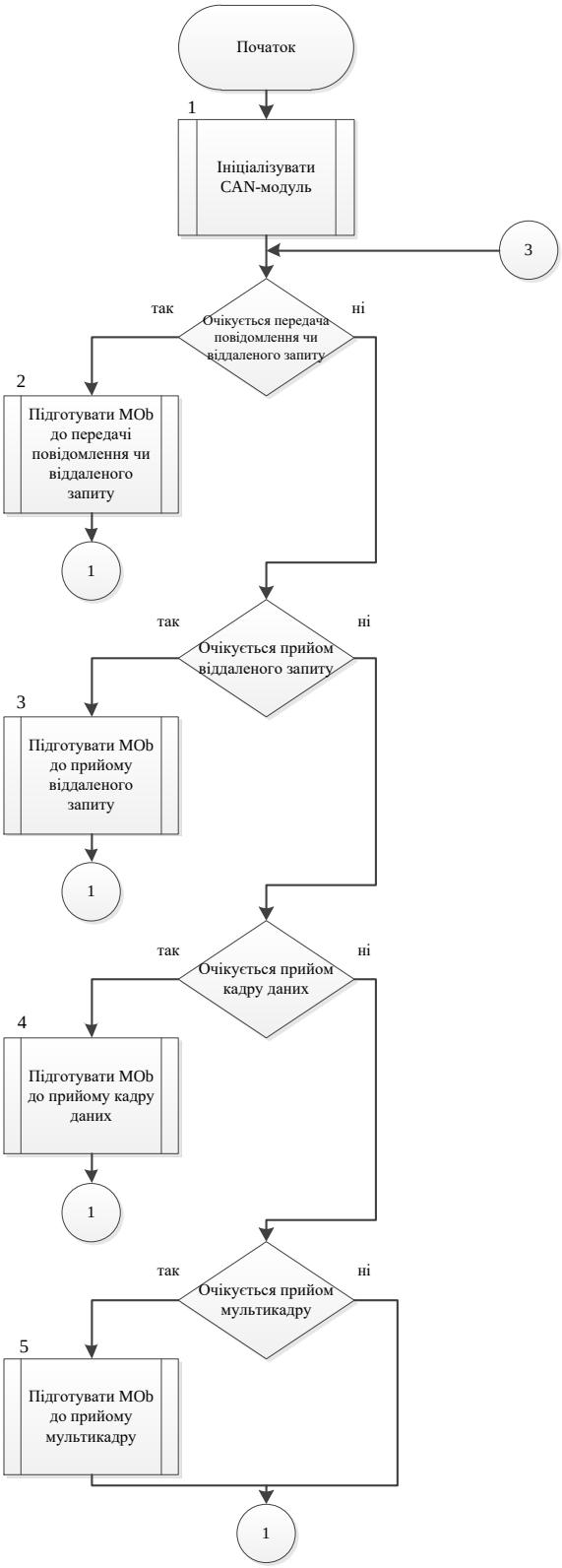


Рис. 14. 21. Схема алгоритму роботи CAN-модуля

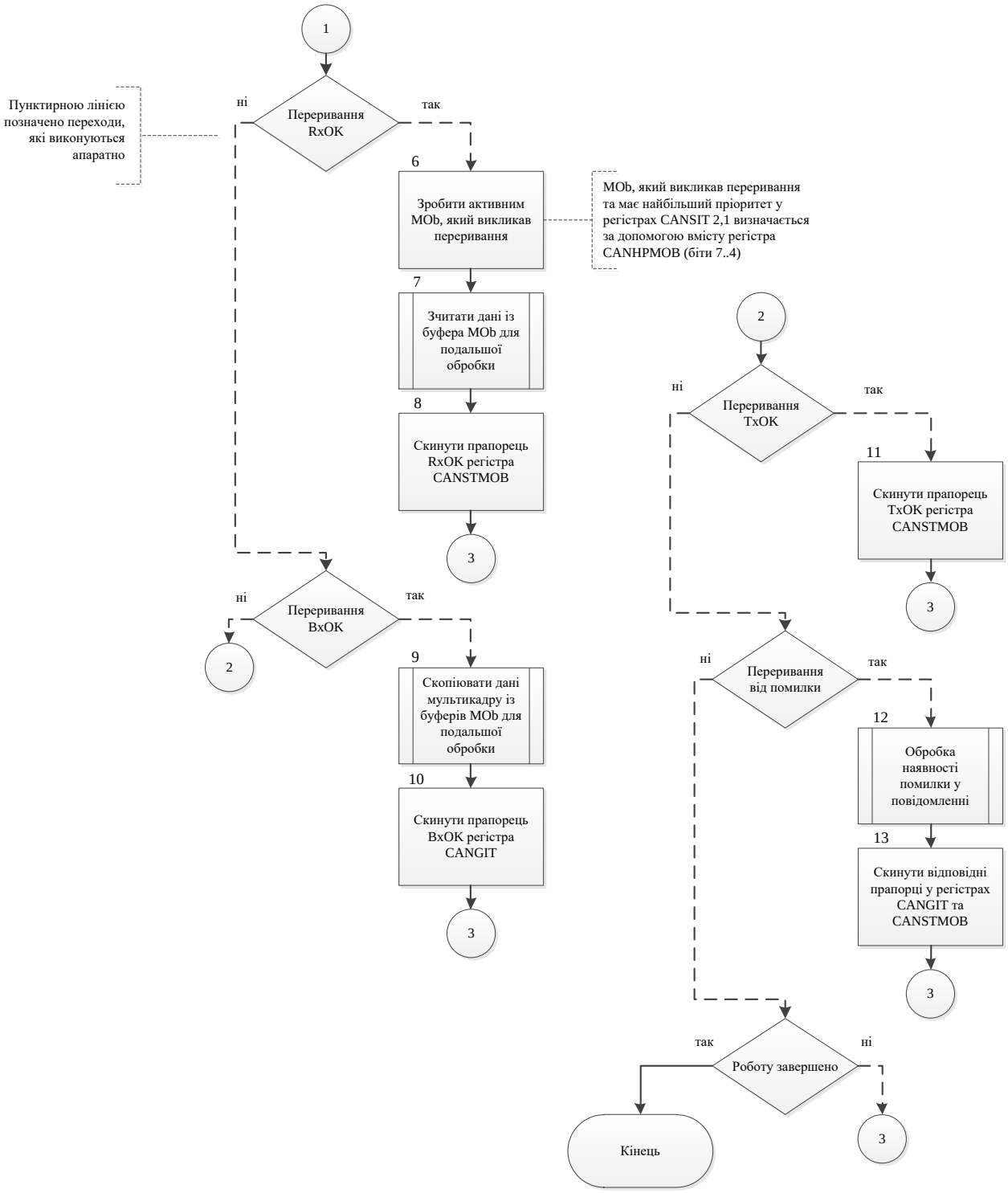


Рис. 14.21. (Продовження). Див. також с. 446

Схема алгоритму ініціалізації CAN-модуля

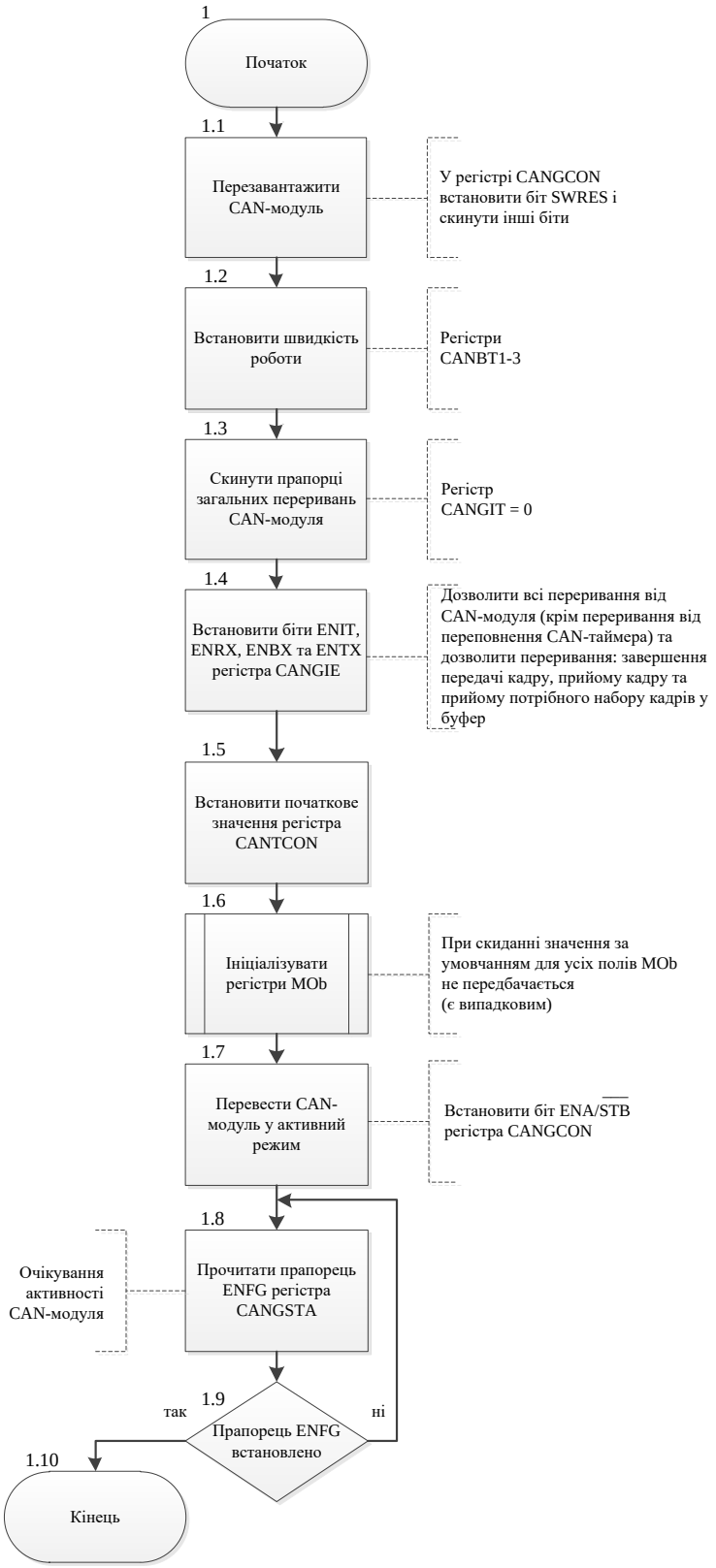


Рис. 14.21. (Продовження). Див. також с. 446, 447

Схема алгоритму підготовки MOb до передачі повідомлення чи віддаленого запиту

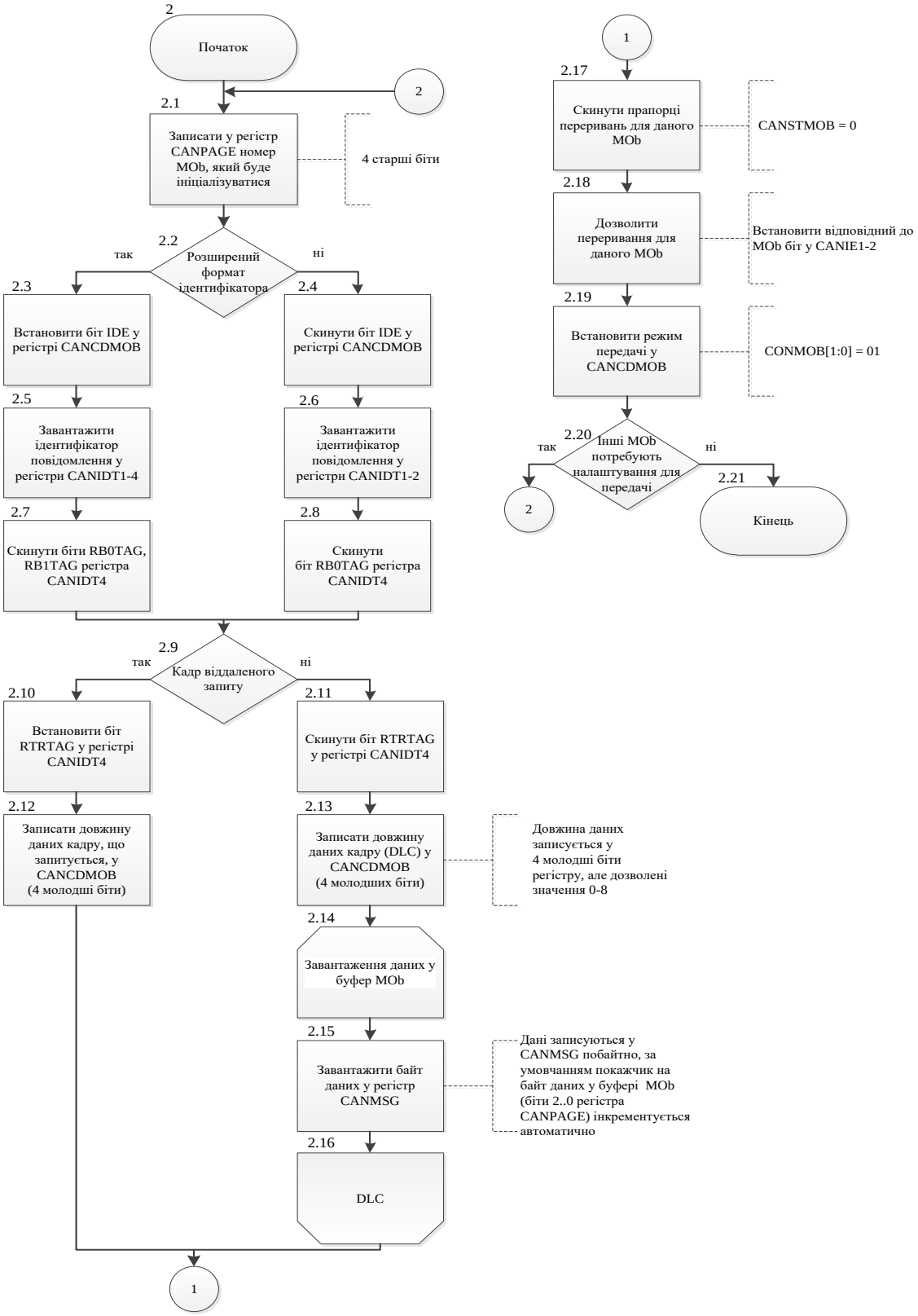


Рис. 14.21. (Продовження). Див. також с. 446…448

Схема алгоритму підготовки MOb до прийому віддаленого запиту та автоматичної відповіді

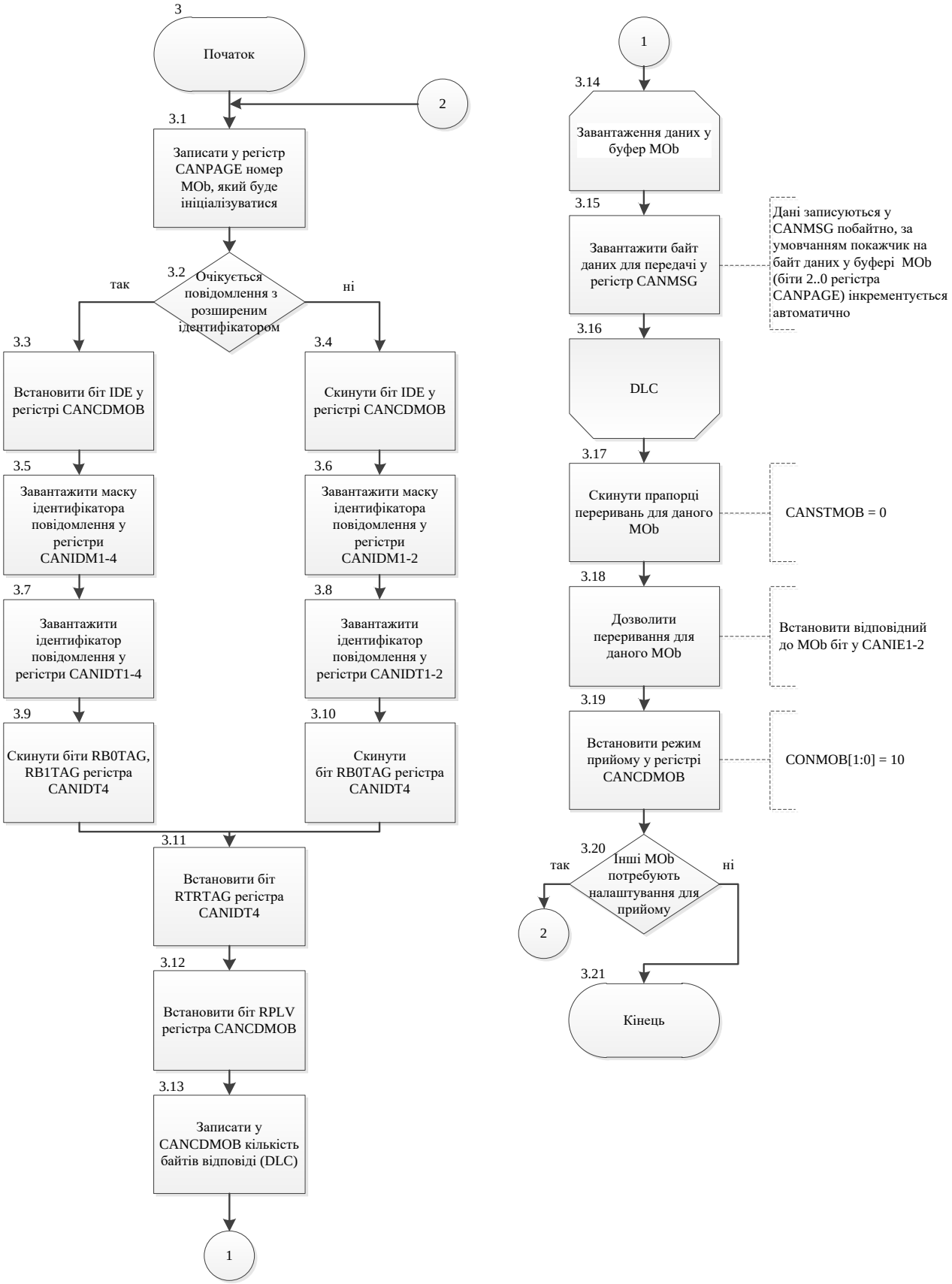


Рис. 14.21. (Продовження). Див. також с. 446…449

Схема алгоритму підготовки MOb до прийому кадру даних

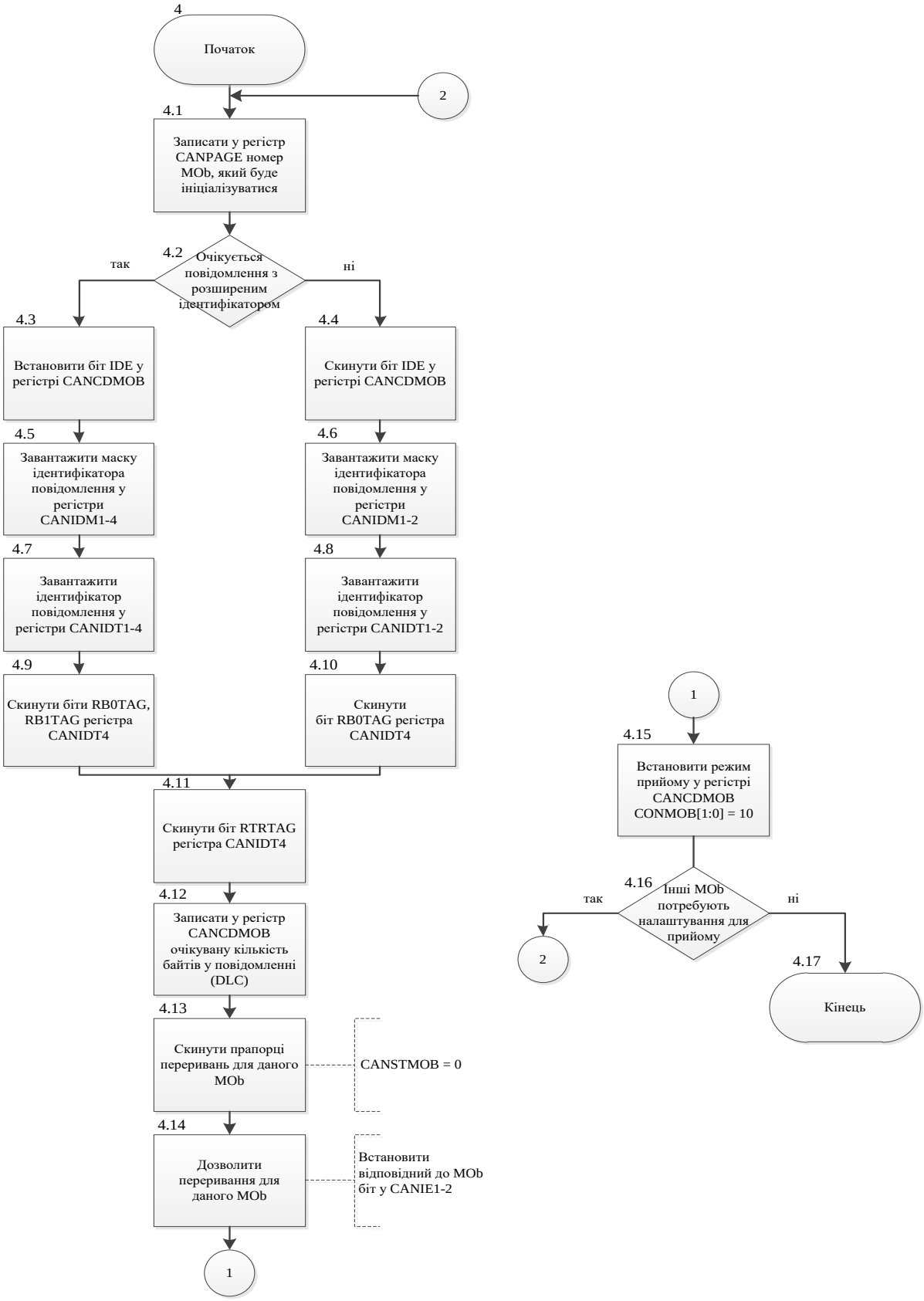


Рис. 14.21. (Продовження). Див. також с. 446…450

Схема алгоритму підготовки MOb до прийому мультикадру (відповідного набору кадрів)

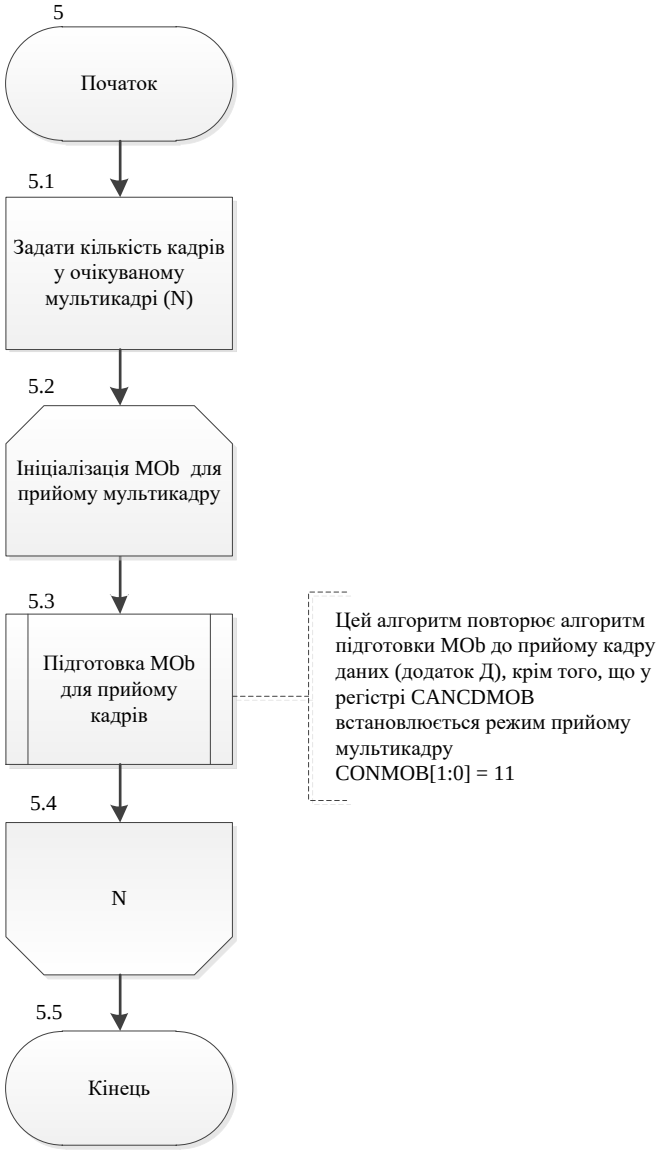


Рис. 14.21. (Продовження). Див. також с. 446…451

Схема алгоритму зчитування даних із буфера MOb в пам’ять даних МК

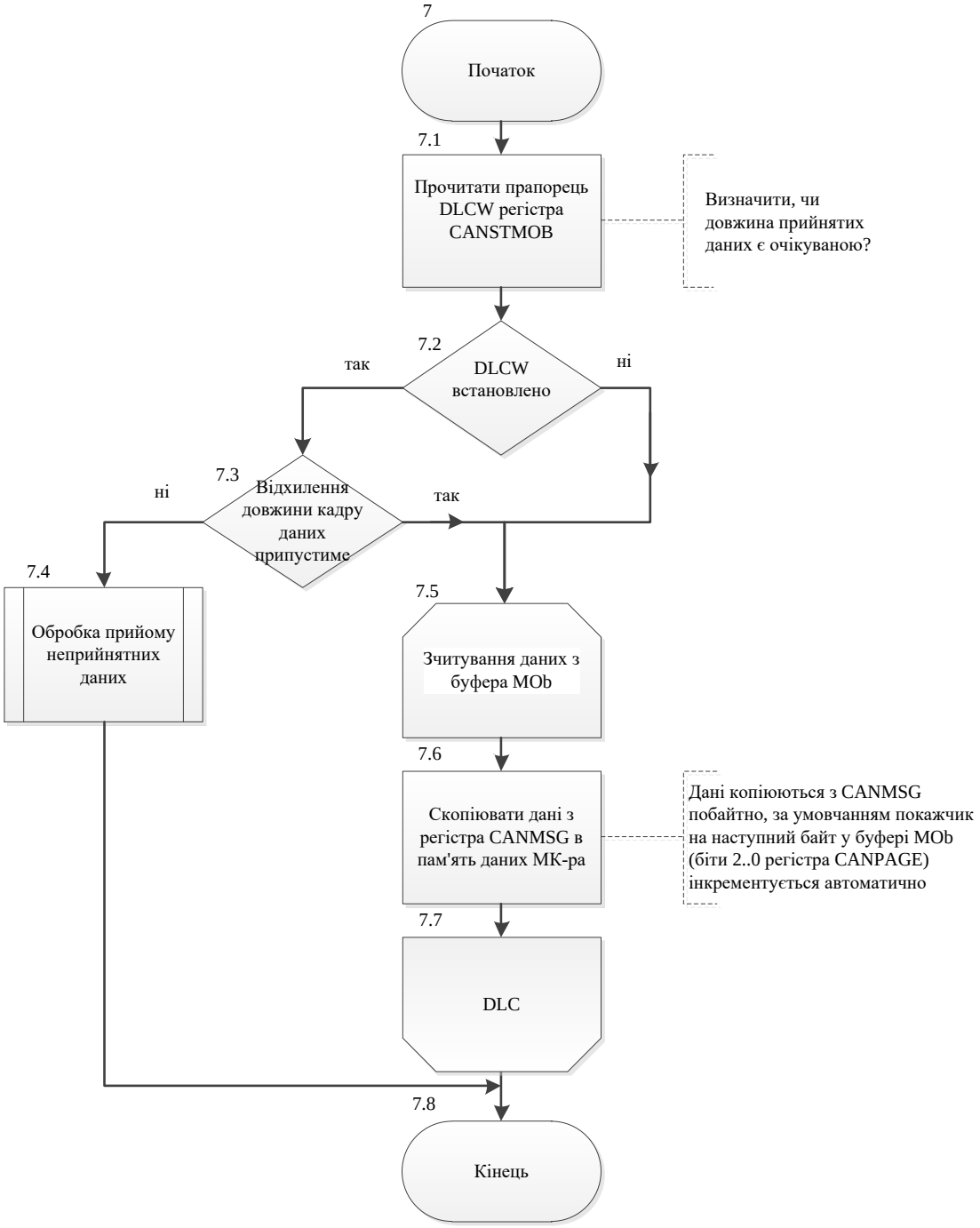


Рис. 14.21. (Продовження). Див. також с. 446…452

Схема алгоритму зчитування даних мультикадру із буферів MOb в пам’ять даних МК

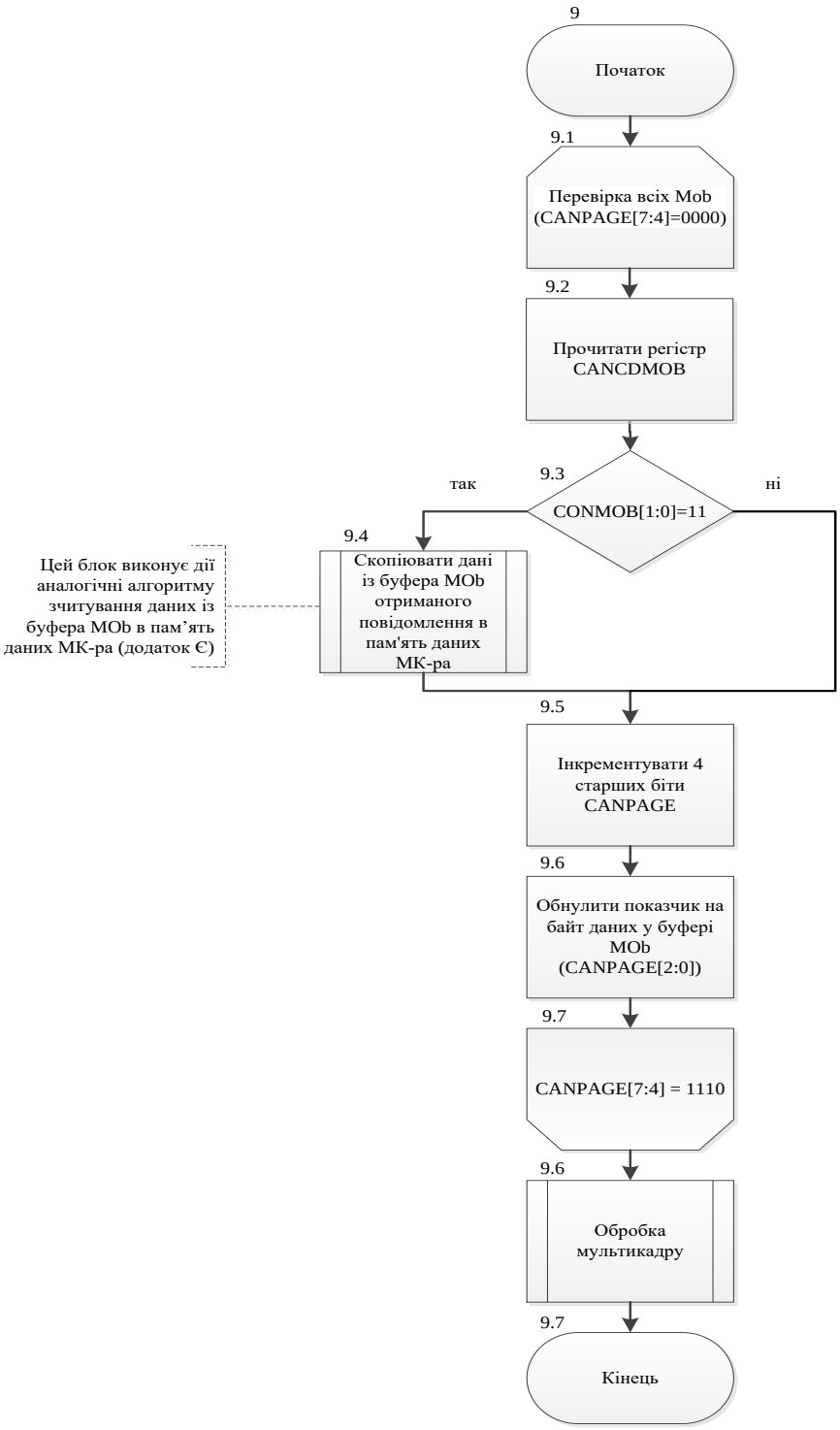


Рис. 14.21. (Закінчення). Див. також с. 446…453

Контрольні запитання та завдання

1. Дайте характеристику CAN-протоколу. Назвіть основні переваги протоколу. Опишіть структуру CAN-мережі.
2. Який завадостійкий код використовується у CAN-мережі? Дайте характеристику коду.
3. Опишіть схему мікропроцесорної системи керування із використанням CAN-мережі.
4. Опишіть еталонну модель ISO/OSI CAN-шини.
5. Поясніть основні характеристики CAN-модуля.
6. Перерахуйте типи помилок, що можуть бути виявлені при передачі CAN-мережею. Наведіть механізми виявлення помилок.
7. Назвіть рівні сигналів при передачі CAN-шиною.
8. Дайте визначення поняттю «фрейм». Типи фреймів. Формати фреймів.
9. Як визначається номінальний час передачі біта?
10. Для чого потрібна синхронізація? Назвіть та опишіть види синхронізації.
11. Наведіть основні характеристики CAN-модуля мікроконтролера AT90CAN128.
12. Опишіть структуру CAN-модуля мікроконтролера AT90CAN128.
13. Дайте характеристику режимів роботи CAN-модуля.
14. Опишіть послідовність програмування режиму передачі кадру (повідомлення) або віддаленого запиту (Tx-конфігурація).
15. Опишіть послідовність програмування режиму прийому кадру даних або віддаленого запиту (Rx-конфігурація).
16. Опишіть програмування режиму прийому мультикадрів в буфер.
17. Опишіть структуру блоку фільтрації повідомлень.
18. Перерахуйте переривання від CAN-модуля. Опишіть структуру переривань.
19. Опишіть структуру блоку таймера CAN-модуля.
20. Опишіть організацію регістрів CAN-модуля.
21. Дайте характеристику фізичному рівню (Physical Layer) CAN-протоколу.
22. Вкажіть максимальну швидкість передачі даних CAN-мережею.
23. Опишіть призначення виводів CAN-роз’єму.
24. Наведіть та поясніть схему з’єднання «High-Speed» трансиверів із шиною.
25. Опишіть принципову схему вузла CAN-мережі.
26. Опишіть алгоритми роботи вузла CAN-мережі в різних режимах.

ПРЕДМЕТНИЙ ПОКАЖЧИК

Інтерфейс :
USART, 129
Виводи МК-ра, 117
Керуючі регістри :
паралельні порти, 119
інтерфейс USART, 129
інтерфейс SPI, 223
інтерфейс I^2C, 162
таймери, 249
система переривань, 393
аналоговий компаратор, 377
аналого-цифровий перетворювач, 322
цифро-аналоговий перетворювач, 359
SLEEP-режим, 390
скидання, 393
системний генератор, 385
Коди статусу модуля I^2C :
ведучий передавач, 185
ведучий приймач, 190
ведений передавач, 193
ведений приймач, 199
Передача даних :
паралельна, 117
послідовна, 117
синхронна, 167, 226
асинхронна, 147
Переривання :
зовнішні, 398
внутрішні, 399
джерела, 393
обробка, 396
дозвіл, 393
пріоритети, 396
Режими роботи :
паралельні порти, 119
інтерфейс USART, 139
інтерфейс SPI, 229
інтерфейс I^2C, 185
таймери, 262
аналоговий компаратор, 380
АЦП, 322
ЦАП, 361
знижене енергоспоживання, 390
скидання, 393
системний генератор, 385
арбітраж шини I^2C, 204
Стани модуля I^2C :
ведучий передавач, 166
ведучий приймач, 166
ведений передавач, 185
ведений приймач, 185
під час арбітражу, 204
Схеми алгоритмів робити модуля I^2C :
ведучий передавач, 189
ведучий приймач, 198
ведений передавач, 202
ведений приймач, 192
вектори переривань, 395
статичне завдання пріоритетів переривань, 396
динамічне завдання пріоритетів переривань, 396

Скидання :
апаратне, 393
джерела, 393
від вартового таймера, 393
за включенням живлення, 439

Структурні схеми :
порти введення/виведення, 121
модуль USART/UART, 130
блок синхронізації USART, 131
формування синхрочастот для USART, 133
мережа з інтерфейсом I^2C, 162
інтерфейс I^2C (TWI), 173
блок контролю адреси (I^2C), 167
модуль SPI, 223
з'єднання мікроконтролерів інтерфейсом SPI, 226
підключення декількох пристроїв до інтерфейсу ведучого SPI, 229
модуль USI, 236
виявлення стану СТАРТ (USI), 236
попередній дільник таймера/лічильника, 253
8-розрядні таймери, 254
модулятор, 275
16-розрядні таймери, 255
блок захоплення таймера, 261
вартовий таймер, 275
розширений вартовий таймер, 275
аналоговий компаратор, 380
попередній дільник АЦП, 329
пристрій синхронізації МК-ра, 392
підсистема скидання, 397
n-розрядний ЦАП з матрицею R-2R та з підсумовуванням напруг, 373
Функціональна схема :
модуля АПЦ, 322
модуля ЦАП, 359
одноканальний режим роботи ЦАП, 364
двоканальний режим роботи ЦАП, 365
Часові діаграми роботи :
зчитування стану розряду PIN_{xn}, 126
формат кадру USART, 138
синхронний режим роботи USART, 142
розпізнавання розрядів UART, 141
розпізнавання стоп-біта та наступного старт-біта UART, 141
передача даних в режимі MSPI, 150
передача даних UART, 139
прийом даних UART, 142
обмін даними шиною I^2C, 167
обмін даними інтерфейсом SPI, 226
вплив сигналу $\overline{SS}$ на початок обміну даними (SPI), 233
модуль USI, 236
режим CTC (8-й таймер), 254
режим Fast PWM (8-й таймер), 254
режим Phase Correct PWM (8-й таймер), 254
модулятор, 254
режим CTC (16-й таймер), 2645
режим Fast PWM (16-й таймер), 267
режим Phase Correct PWM (16-й таймер), 270
режим Phase and Frequency Correct PWM (16-й таймер), 273
АЦП, 330
скидання по ввімкненню живлення, 396
апаратне скидання, 396

скидання від вартового таймера, 396
скидання по зниженню напруги живлення, 396
оновлення та запити перетворення каналів ЦАП, 367
стани шини I^2C, 167
синхронізація сигналу SCL (I^2C), 171
розподіл пріоритетів I^2C, 171
формати пакетів I^2C, 172, 173
взаємодія програми з модулем I^2C (TWI), 181

СПИСОК ЛІТЕРАТУРИ

Базова

1 Проектування мікропроцесорних систем: Проектування мікропроцесорних систем на базі AVR-мікроконтролерів: Периферійні модулі AVR-мікроконтролерів: Навчальний посібник для студентів напряму підготовки 6.050201 «Системна інженерія» кафедри Автоматики та управління у технічних системах / Укл.: А.О. Новацький– К: НТУУ „КПІ”, 2012. – 470 с. : ил.

2 Навчальний посібник з дисципліни «Проектування мікропроцесорних систем», розділ «Програмування мікроконтролерів родини AVR» для студентів напряму підготовки 6.050201 «Системна інженерія» кафедри Автоматики та управління у технічних системах / Укл.: А.О. Новацький, Є.В. Глушко – К: НТУУ „КПІ”, 2013. – 109 с. : ил.

3 Мікропроцесорні та мікроконтролерні системи : підручник. У 2 ч. Ч. 1. Мікропроцесорні системи [Електронний ресурс] / А. О. Новацький. – Електронні текстові дані (1 файл: 43,8 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, Вид-во «Політехніка», 2019. – 367 с. : ил.

4 Мікропроцесорні та мікроконтролерні системи : лаб. практикум [Електронний ресурс] : навч. посіб. для студ. освітньої програми «Інтегровані інформаційні системи» спец. 126 «Інформаційні системи та технології» / Уклад. А. О. Новацький. – Електронні текстові дані (1 файл: 18,97 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2018. – 415 с.: ил.

5 Проектування мікропроцесорних систем: Проектування мікропроцесорних систем на базі мікроконтролерів сімейства MCS-51: Периферійні модулі мікроконтролерів сімейства MCS-51 :навч. посіб. для студ. напряму підготовки 6.050201 «Системна інженерія» кафедри Автоматики та управління у технічних системах / А. О. Новацький. – Київ : НТУУ «КПІ», 2016. – 399 с.: ил.

6 Евстифеев А.В. Микроконтроллеры AVR семейств Tiny и Mega фирмы ATMEL – М.: Додэка, 2005 – 560 с.: ил.

7 Евстифеев А. В. Микроконтроллеры AVR семейства Mega. Руководство пользователя. – М.:Издательский дом «Додэка-XXI», 2007. – 592 с.: ил.
8 Проектування CAN-мережі: Навчальний посібник для студентів спеціальності 8.050201.01 «Комп'ютеризовані системи управління та автоматика» кафедри Автоматики та управління у технічних системах / Автор: А.О. Новацький К: НТУУ „КПІ", 2011- 169 с.
9 Комп'ютерна електроніка [Електронний ресурс] : підручник для студ. спеціальності 126 «Інформаційні системи та технології», спеціалізації «Інтегровані інформаційні системи» / А.О. Новацький ; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 80.9 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2018. – 468 с.

Допоміжна

10 Голубцов М.С., Кириченкова А.В. Микроконтроллеры AVR: от простого к сложному. – М.: Солон-Пресс, 2005.
11 Трамперт Вольфган. AVR-RISC микроконтроллеры. – Перевод с немецкого. – Киев.: МК – Пресс, 2006.
12 Баранов В.Н. Применения микроконтроллеров AVR: схемы, алгоритмы, программы. – М.: Додэка, 2004.
13 Ревич Ю. В. Практическое программирование микроконтроллеров Atmel AVR на языке ассемблера. – 2-е изд., испр. – СПб.:БХВ-Петербург,2011.
14 Кравченко А. В. 10 практических устройств на AVR-микроконтроллерах. Книга 1 – М.: Издательский дом «Додэка-XXI», К. «МК-Пресс», 2008.
15 Кравченко А. В. 10 практических устройств на AVR-микроконтроллерах. Книга 2 – К. «МК-Пресс», СПб.: «КОРОНА-ВЕК», 2009.
16 Кравченко А. В. 10 практических устройств на AVR-микроконтроллерах. Книга 3 – К. «МК-Пресс», СПб.: «КОРОНА-ВЕК», 2011.
17 Александров Е. К., Грушвицкий Р. И. Микропроцессорные системы: Учебное пособие для вузов. – СПб.: Политехника, 2002.
18 Официальное описание микроконтроллеров XMEGA http://www.gaw.ru/html.cgi/txt/ic/Atmel/micros/avr_xmega/start.htm

19 Опис CAN-протоколу – http://www.itt-ltd.com/reference/ref_can.html
20 CAN-словарь CAN-dictionaryv2_ru PDF-формат – http://www.can-cia.org
21 Схеми та пояснення роботи CAN контролерів та трансиверів – http://atmel.com
22 CAN-трансивер – https://www.nxp.com/docs/en/data-sheet/PCA82C250.pdf
23 CAN-трансивер – https://www.nxp.com/docs/en/data-sheet/MC33388.pdf
24 Фьюзі-біти – http://easyelectronics.ru/avr-uchebnyj-kurs-konfiguraciya-fuse-bit.html

Навчальне видання

Новацький Анатолій Олександрович

МІКРОПРОЦЕСОРНІ ТА МІКРОКОНТРОЛЕРНІ СИСТЕМИ

Частина 2

ПРОЕКТУВАННЯ МІКРОПРОЦЕСОРНИХ СИСТЕМ

Підручник

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Свідоцтво про державну реєстрацію: серія ДК № 5354 від 25.05.2017 р.
просп. Перемоги, 37,
м. Київ, 03056

Формат 60×84 1/16. Папір офс. Гарнітура Times.
Ум. друк. арк. 27,20. Обл.-вид. арк. 31,84.
Поз. 20-1-1-008.

Видавництво «Політехніка», КПІ ім. Ігоря Сікорського
вул. Політехнічна, 14, корп. 15
м. Київ, 03056
тел. (044) 204-81-78