МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

А.О.Новацький

Електроніка та мікропроцесорна техніка. Частина 2. Мікропроцесорні системи

Курс лекцій

Навчальний посібник

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського як навчальний посібник для здобувачів ступеня бакалавра за освітньою програмою «Інтегровані інформаційні системи» спеціальності 126 «Інформаційні системи та технології»

Електронне мережне навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2025 р.

Автор: Новацький А.О. кандидат техн. наук, доцент, КПІ ім. Ігоря Сікорського, кафедра Інформаційних систем та технологій
Рецензент: Селіванов Віктор Львович, кандидат техн. наук, доцент, КПІ ім. Ігоря Сікорського, кафедра Обчислювальної техніки
Відповідальний редактор: Полторак Вадим Петрович, кандидат техн. наук, доцент, КПІ ім. Ігоря Сікорського, кафедра Інформаційних систем та технологій

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол №5 від 6.03.2025 р.) за поданням Вченої ради факультету інформатики та обчислювальної техніки (протокол № 8 від 24.02.2025 р.)

Електронне мережне навчальне видання

Стисло викладено основи систем числення, кодування та двійкової арифметики. Розглянуто характеристику мікропроцесорної системи, її структуру, програмну модель та характеристику команд типового мікропроцесора та мікроконтролера. Наведено матеріал про архітектури мікроконтролерних мереж та основних периферійних модулів мікроконтролерів. Розглянуто моделювання окремих модулів мікроконтролера у пакеті «PROTEUS».

Навчальний посібник призначено для здобувачів ступеня бакалавра за освітньою програмою «Інтегровані інформаційні системи» спеціальності 126 «Інформаційні системи та технології», які вивчають проектування та використання мікропроцесорних систем.

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

Реєстр. № НП 24/25-371. Обсяг 13,9 авт. арк. Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
проспект Перемоги, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

ЗМІСТ

ВСТУП ……. 14
ЛЕКЦІЯ 1. ОСНОВНІ ПОНЯТТЯ МІКРОПРОЦЕСОРНОЇ тЕХНІКИ.
ЕЛЕКТРИЧНІ СХЕМИ ТИПОВИХ МІКРОПРОЦЕСОРНИХ СИСТЕМ 16
1. ОСНОВНІ ПОНЯТТЯ МІКРОПРОЦЕСОРНОЇ тЕХНІКИ 16
1.1. Мікропроцесор 16
1.2. Мікроконтролер 16
1.3. Мікропроцесорна/мікроконтролерна система 17
1.4. Основні характеристики МП/МК 17
1.5. Системи числення та коди 17
1.5.1. Загальна характеристика систем числення та кодів 17
1.5.2. Десяткова система числення 18
1.5.3. Двійкова система числення 18
1.5.4. Шістнадцяткова система числення 19
1.5.5. Двійково-десяткові числа (коди) 21
1.5.6. Представлення десяткових чисел в коді ASCII 21
1.6. Формати подання чисел 22
1.6.1. Подання чисел зі знаком 22
1.6.2. Числа з нефіксованою крапкою 25
1.7. Двійкова арифметика 25
1.7.1. Двійкове додавання 25
1.7.2. Двійкове віднімання 26
1.7.3. Двійкове множення 26
1.7.4. Двійкове ділення 26
1.8. Двійково-десяткова арифметика 26
2. ЕЛЕКТРИЧНІ СХЕМИ ТИПОВИХ МІКРОПРОЦЕСОРНИХ СИСТЕМ 27
2.1. Структурна схема типової локальної мікроконтролерної системи керування .. 27
2.2. Функціональна схема мікропроцесорної системи керування 29
Контрольні запитання та завдання 33
ЛЕКЦІЯ 2. МОДУЛЬНА СТРУКТУРА МПС.
СТРУКТУРНІ СХЕМИ ТИПОВОГО МП ТА МК 35
1. МОДУЛЬНА СТРУКТУРА МПС 35
1.1. Загальна характеристика модульної структури МПС 35
1.2. Модуль мікропроцесора 35
1.3. Модуль пам’яті 37
1.4. Модуль введення/виведення 37
1.5. Контролер прямого доступу до пам’яті 37
1.6. Контролер переривань 37

1.7. Модуль таймера 38
1.8. Системна шина 38
1.9. Підключення МП до системної шини МПС 39
2. СТРУКТУРНІ СХЕМИ ТИПОВОГО МП ТА МК 40
2.1. Структура восьмирозрядного мікропроцесора 40
2.2. Структура шістнадцятирозрядного мікропроцесора 44
2.3. Структура восьмирозрядного мікроконтролера 49
2.3.1. Загальна характеристика восьмирозрядного мікроконтролера 49
2.3.2. Структура ядра AVR-мікроконтролерів 49
Контрольні запитання та завдання 50
ЛЕКЦІЯ 3. ОСОБЛИВОСТІ АРХІТЕКТУРИ ПАМ'ЯТІ МПС. ОРГАНІЗАЦІЯ ПАМ'ЯТІ МПС НА ОСНОВІ МІКРОПРОЦЕСОРА 52
1. ОСОБЛИВОСТІ АРХІТЕКТУРИ ПАМ'ЯТІ МПС 52
1.1. Призначення та місце модуля пам'яті в мікропроцесорній системі 52
1.2. Класифікація пам'яті 52
1.2.1. Основна та зовнішня пам'ять 52
1.2.2. Пам'ять з довільним та послідовним доступом 53
1.2.3. Енергозалежна та енергонезалежна пам'ять 53
1.2.4. Статична та динамічна пам'ять 53
1.2.5. Основні характеристики пам'яті 53
1.2.6. Призначення та організація стека 54
1.2.7. Режим прямого доступу до пам'яті 54
2. ОРГАНІЗАЦІЯ ПАМ'ЯТІ МПС НА ОСНОВІ МІКРОПРОЦЕСОРА 55
2.1. Організація пам'яті на основі 16-розрядного МП 55
2.1.1. Фізична та логічна організація пам'яті 55
2.1.2. Вибір типу сегмента пам'яті під час обчислення фізичної адреси 58
2.1.3. Розподіл пам'яті на банки 59
Контрольні запитання та завдання 62
ЛЕКЦІЯ 4. ОРГАНІЗАЦІЯ ПАМ'ЯТІ МПС НА ОСНОВІ МІКРОКОНТРОЛЕРА. ПРОГРАМУВАННЯ FLASH- ТА EEPROM-ПАМ'ЯТІ 63
1. ОРГАНІЗАЦІЯ ПАМ'ЯТІ НА ОСНОВІ МІКРОКОНТРОЛЕРА 63
1.1. Загальна характеристика пам'яті на основі 8-розрядного мікроконтролера 63
1.2. Організація пам'яті програм AVR-мікроконтролерів 64
1.3. Організація пам'яті даних AVR-мікроконтролерів 65
1.3.1. Загальна характеристика пам'яті даних AVR-мікроконтролерів 65
1.3.2. Статична пам'ять даних 65
1.3.2.1. Регістри загального призначення 65
1.3.2.2. Стек 67

1.3.2.3. Регістри введення/виведення 68
1.4. Використання зовнішнього оперативного запам’ятовувального пристрою..... 70
1.5. Енергонезалежна пам’ять даних EEPROM..... 70
1.6. Особливості проектування пам’яті великого об’єму 71
2. ПРОГРАМУВАННЯ FLASH- ТА EEPROM-ПАМ’ЯТІ..... 71
2.1. Режими програмування..... 71
2.2. Сторінкова організація пам’яті програм і даних..... 71
2.3. Програмування пам’яті за послідовним каналом 72
Контрольні запитання та завдання 74
ЛЕКЦІЯ 5. ПРОГРАМНА МОДЕЛЬ МП/МК. ХАРАКТЕРИСТИКА КОМАНД МП/МК 75
1. ПРОГРАМНА МОДЕЛЬ МП/МК 75
1.1. Послідовність розробки робочої керувальної програми..... 75
1.2. Програмна модель 8-розрядного мікропроцесора 75
1.3. Програмна модель 16-розрядного мікропроцесора..... 76
1.4. Програмна модель мікроконтролера 80
2. ХАРАКТЕРИСТИКА КОМАНД МП/МК..... 80
2.1. Код операції команди 80
2.2. Мнемоніка команди та мнемокод 80
2.3. Машинний код команди..... 82
2.4. Операнди 82
2.5. Коментар..... 82
2.6. Формати команд..... 82
2.6.1. Команди 16-розрядного мікропроцесора..... 82
2.6.2. Команди 8-розрядного мікроконтролера сімейства AVR..... 84
2.7. Формати даних..... 87
2.7.1. Загальна характеристика форматів даних..... 87
2.7.2. Формати (типи) даних 8-розрядного мікропроцесора/мікроконтролера..... 87
2.7.3. Формати (типи) даних 16-розрядного мікропроцесора..... 88
2.8. Довжина команд у байтах і їх розміщення в пам’яті програм..... 88
2.9. Вплив команд на прапорці..... 89
2.10. Час виконання команд..... 89
Контрольні запитання та завдання 90
ЛЕКЦІЯ 6. СПОСОБИ АДРЕСАЦІЇ ОПЕРАНДІВ МП/МК. 91
КОМАНДИ МІКРОПРОЦЕСОРІВ/МІКРОКОНТРОЛЕРІВ 91
1. СПОСОБИ АДРЕСАЦІЇ ОПЕРАНДІВ МП/МК..... 91
1.1. Загальні відомості про способи адресації 91
1.2. Восьмирозрядний мк сім’ї AVR..... 91
1.2.1. Загальні відомості про способи адресації AVR-мікроконтролерів 91

1.2.2. Неявна адресація 91
1.2.3. Безпосередня адресація 91
1.2.4. Пряма адресація 91
1.2.5. Пряма адресація одного регістра загального призначення 92
1.2.6. Пряма адресація двох регістрів загального призначення 92
1.2.7. Пряма адресація рвв 93
1.2.8. Пряма адресація СПД 93
1.2.9. Непряма адресація 93
1.2.9.1. Проста непряма адресація 94
1.2.9.2. Відносна непряма адресація 94
1.2.9.3. Непряма адресація з попереднім декрементом (переддекрементом) 95
1.2.9.4. Непряма адресація з постінкрементом 95
1.2.9.5. Непряма адресація пам'яті програм 96
1.2.9.6. Непряма адресація констант в пам'яті програм 96
1.2.10. Відносна адресація пам'яті програм 97
2. КОМАНДИ МІКРОПРОЦЕСОРІВ/МІКРОКОНТРОЛЕРІВ 97
2.1. Команди шістнадцятирозрядного мікропроцесора 97
2.2. Команди восьмирозрядного МК сімейства AVR 98
2.2.1. Базовий набір команд 98
2.2.2. Нові команди AVR-мікроконтролерів 98
Контрольні запитання та завдання 105
ЛЕКЦІЯ 7. ОРГАНІЗАЦІЯ ПІДСИСТЕМИ ПЕРЕРИВАНЬ У МПС 107
1. ОРГАНІЗАЦІЯ ПІДСИСТЕМИ ПЕРЕРИВАНЬ У МПС 107
1.1. Загальні відомості про підсистему переривань 107
1.2. Організація підсистеми переривань мікропроцесора 107
1.2.1. Види переривань 107
1.2.2. Особливості обробки зовнішніх переривань 109
1.3. Організація підсистеми переривань мікроконтролера 109
1.3.1. Основні властивості підсистеми переривань мікроконтролера 109
1.3.2. Обробка переривань 110
1.3.3. Зовнішні переривання 111
1.3.4. Внутрішні переривання 112
1.3.5. Особливості використання модуля переривань у МК XMega 112
1.4. Маскування та призначення пріоритетів переривань 112
1.4.1. Маскування переривань 112
1.4.2. Призначення пріоритетів переривань 113
1.4.3. Визначення адреси підпрограми обробки переривання 113
Контрольні запитання та завдання 113

ЛЕКЦІЯ 8. ФОРМУВАННЯ ІНТЕРВАЛІВ ЧАСУ ТА ПІДРАХУНОК ЗОВНІШНІХ ПОДІЙ У МПС 115
1. ФОРМУВАННЯ ІНТЕРВАЛІВ ЧАСУ ТА ПІДРАХУНОК ЗОВНІШНІХ ПОДІЙ У МПС 115
1.1. Способи формування інтервалів часу та підрахунок 115
зовнішніх подій у МПС 115
1.2. Таймери мікроконтролерів 116
1.2.1. Загальні відомості про таймери мікроконтролерів 116
1.2.2. Попередні дільники таймерів 116
1.2.3. Восьмирозрядні таймери сімейства AVR 118
1.2.4. 16-розрядні таймери сімейства AVR 118
1.2.4.1. Опис структурної схеми таймера 118
1.2.4.2. Керування тактовим сигналом 120
1.2.4.3. Режими роботи 120
1.2.5. Вартовий таймер 129
1.2.6. Моделювання модуля таймера AVR-мікроконтролерів 130
1.3. Застосування для формування інтервалів часу мікросхеми програмованого таймера 130
Контрольні запитання та завдання 130
ЛЕКЦІЯ 9. ПРОЕКТУВАННЯ ПРИСТРОЮ КЕРУВАННЯ ДВИГУНОМ ПОСТІЙНОГО СТРУМУ 132
1. ПРОЕКТУВАННЯ ПРИСТРОЮ КЕРУВАННЯ ДВИГУНОМ ПОСТІЙНОГО СТРУМУ 132
1.1. Завдання 132
1.2. Розробка та опис робочої моделі 132
1.2.1. Робоча модель пристрою 132
1.2.2. Опис моделі 133
1.2.3. Опис окремих елементів моделі 138
1.2.3.1. Мікроконтролер 138
1.2.3.2. Модуль таймера 139
1.2.3.3. Двигун 139
1.2.3.4. Драйвер ШІМ 139
1.2.3.5. Захисні діоди 141
1.3. Розрахунок ШІМ-модуля та його програмування мовою Асемблер 141
1.4. Схема алгоритму роботи пристрою 143
1.5. Розробка окремих фрагментів програми мовою Асемблер 144
1.5.1. Зупинка таймера 144
1.5.2. Завантаження регістра TCCR1A 144

1.5.3. Програмування ліній PB5, PB1, PB0 на виведення 145
1.5.4. Завантаження регістра ICR1 145
1.5.5. Завантаження регістра OCR1A 146
1.5.6. Програмування $К_{ДІЛ}$ = N = 8, режиму роботи номер 8 та запуск таймера T/C1 146
1.6. Керувальна програма мовою С 147
1.6.1. керувальна програма мовою С, яка відповідає завданню 147
1.6.2. Компіляція робочої програми, отримання hex-файлу та його завантаження у пам’ять комп`ютера 149
1.7. Розробка та опис структурної схеми пристрою керування двигуном постійного струму ……. ……. ……. ……. ……. …… … 149
Контрольні запитання та завдання 151
ЛЕКЦІЯ 10. МОДЕЛЮВАННЯ ПРИСТРОЮ КЕРУВАННЯ ДВИГУНОМ ПОСТІЙНОГО СТРУМУ В ПАКЕТІ PROTEUS.
ОРГАНІЗАЦІЯ ОБМІНУ ДАНИМИ В МПС 152
1. МОДЕЛЮВАННЯ ПРИСТРОЮ КЕРУВАННЯ ДВИГУНОМ ПОСТІЙНОГО СТРУМУ В ПАКЕТІ PROTEUS 152
2. ОРГАНІЗАЦІЯ ОБМІНУ ДАНИМИ У МПС 154
2.1. Призначення та місце модуля введення/виведення у МПС 154
2.2. Способи обміну даними в МПС 155
2.3. Адресація пристроїв введення/виведення 157
2.4. Підключення ПВВ до системної шини та зовнішніх пристроїв 157
2.5. Порти введення/виведення AVR-мікроконтролерів 160
2.5.1. Загальна характеристика портів ведення/виведення 160
2.5.2. Звернення до портів введення/виведення 161
2.5.3. Структура портів введення/виведення 161
2.5.4. Конфігурування виводів портів введення/виведення 163
Контрольні запитання та завдання 164
ЛЕКЦІЯ 11. АРХІТЕКТУРА ПОСЛІДОВНОГО ІНТЕРФЕЙСУ 165
1. АРХІТЕКТУРА ПОСЛІДОВНОГО ІНТЕРФЕЙСУ 165
1.1. Загальна характеристика інтерфейсу 165
1.2. Опис структури модулів УАПП/УСАПП 165
1.3. Швидкість прийому-передачі даних 167
1.4. Формат кадру 169
1.5. Передача даних 170
1.6. Прийом даних 172
1.7. Обмін даними через інтерфейс УСАПП у МК-мережі 176

1.8. Розрахунок швидкості передачі інформації, тривалості одного біта та часу передачі одного байта 177
1.9. Моделювання модуля УСАПП у пакеті PROTEUS..... 177
Контрольні запитання та завдання 178
ЛЕКЦІЯ 12. ПРОЕКТУВАННЯ ПРИСТРОЮ ОБМІНУ ІНФОРМАЦІЄЮ МІЖ МОДУЛЕМ УСАПП ТА МІКРОКОНТРОЛЕРОМ 179
1. ПРОЕКТУВАННЯ ПРИСТРОЮ ОБМІНУ ІНФОРМАЦІЄЮ МІЖ МОДУЛЕМ УСАПП ТА МІКРОКОНТРОЛЕРОМ 179
1.1. Завдання..... 179
1.2. Розробка та опис робочої моделі в пакеті PROTEUS 179
1.2.1. Робоча модель пристрою 179
1.2.2. Опис робочої моделі..... 180
1.3. Розрахунок модуля УСАПП 181
1.3.1. Розрахунок швидкості обміну..... 181
1.3.2. Програмування регістра UBRR контролера швидкості передачі 181
1.3.3. Програмування регістра UCSRA 182
1.3.4. Програмування регістра UCSRC 182
1.3.5. Програмування регістра UCSRB 184
1.3.6. Програмування регістра даних UDR..... 185
1.4. Схема алгоритму роботи пристрою..... 185
1.5. Керувальна програма мовою Асемблер 186
1.5.1. Лістинг керувальної програми мовою Асемблер..... 186
1.5.2. Компіляція робочої програми та отримання hex-файлу..... 188
1.6. Моделювання пристрою в пакеті PROTEUS 188
1.7. Розробка та опис структурної схеми пристрою 194
ЛЕКЦІЯ 13. ЗВ’ЯЗОК МП/МК З АНАЛОГОВИМ ОБ’ЄКТОМ КЕРУВАННЯ. ЗВ’ЯЗОК МП/МК З МОДЕМОМ 198
1. ЗВ’ЯЗОК МІКРОПРОЦЕСОРІВ/МІКРОКОНТРОЛЕРІВ З АНАЛОГОВИМ ОБ’ЄКТОМ КЕРУВАННЯ..... 198
1.1. Особливості введення/виведення аналогової інформації у МПС 198
1.2. Застосування АЦП під час введення аналогової інформації у МПС 199
1.2.1. Загальні відомості про АЦП 199
1.2.2. Модуль АЦП AVR-мікроконтролерів..... 199
1.2.2.1. Функціональна схема модуля..... 199
1.2.2.2. Програмування модуля 200
1.2.2.3. Формування тактового сигналу 201
1.2.2.4. Часові діаграми роботи..... 202
1.2.2.5. Результат перетворення 203

1.2.2.6. Моделювання модуля АЦП та цифрового вольтметра 204
1.3. Застосування ЦАП під час виведення аналогової інформації у МПС 205
1.3.1. Загальна характеристика ЦАП 205
1.3.2. Особливості архітектури модуля ЦАП в складі AVR-мікроконтролерів 205
2. ЗВ'ЯЗОК МП/МК З МОДЕМОМ 205
2.1. Структурна схема інтерфейсу RS-232 205
2.2. Формат даних інтерфейсу RS-232 206
2.3. Пристрій перетворення рівнів 207
2.4. Роз'єм RS-232 207
Контрольні запитання та завдання 208
ЛЕКЦІЯ 14. МОДЕЛЮВАННЯ МОДУЛЯ АЦП. МОДЕЛЮВАННЯ ЦИФРОВОГО ВОЛЬТМЕТРА 210
1. МОДЕЛЮВАННЯ МОДУЛЯ АЦП 210
1.1. Схема моделі та її опис 210
1.2. Схема алгоритму роботи моделі 212
1.3. Робоча програма 215
2. МОДЕЛЮВАННЯ МОДУЛЯ ЦИФРОВОГО ВОЛЬТМЕТРА 218
2.1. Схема моделі та її опис 218
2.2. Схема алгоритму роботи моделі 219
2.3. Робоча програма 222
Контрольні запитання та завдання 225
ЛЕКЦІЯ 15. МІКРОКОНТРОЛЕРНА МЕРЕЖА RS-485. МІКРОКОНТРОЛЕРНА МЕРЕЖА НА ОСНОВІ ІНТЕРФЕЙСУ SPI 226
1. МІКРОКОНТРОЛЕРНА МЕРЕЖА RS-485 226
1.1. Загальні відомості про мікроконтролерні мережі 226
1.2. Мікроконтролерна мережа RS-485 226
1.2.1. Загальна характеристика мережі 226
1.2.2. Кількість вузлів мережі 228
1.2.3. Швидкість та дальність передачі даних 228
1.2.4. Протоколи та роз'єми для передачі 228
1.2.5. Узгодження «відкритого» кінця кабелю 228
1.2.6. Рівні сигналів у мережі 229
1.2.7. Зсув на сигнальних ланцюгах 229
1.2.8. Реалізація інтерфейсу RS-485 230
2. МІКРОКОНТРОЛЕРНА МЕРЕЖА НА ОСНОВІ ІНТЕРФЕЙСУ SPI 233
2.1. Загальна характеристика інтерфейсу SPI 233
2.2. Характеристика модуля SPI мікроконтролерів AVR 233
2.2.1. Опис структурної схеми модуля 233

2.2.2. Обмін даними між двома мікроконтролерами 236
2.2.3. Структура SPI-мережі 237
2.2.4. Програмування модуля 238
2.2.5. Режими передачі даних SPI-інтерфейсом 239
2.2.6. Програмування швидкості передачі даних 242
2.2.7. Використання виводу $\overline{SS}$ 242
2.2.8. Використання інтерфейсу SPI для програмування пам'яті 243
2.2.9. Периферійні пристрої з SPI-інтерфейсом 244
2.2.10. Моделювання модуля SPI 244
Контрольні запитання та завдання 244
ЛЕКЦІЯ 16. МІКРОКОНТРОЛЕРНА МЕРЕЖА НА ОСНОВІ ІНТЕРФЕЙСУ I^2C. ....
МІКРОКОНТРОЛЕРНА МЕРЕЖА 1-WIRE 246
1. МІКРОКОНТРОЛЕРНА МЕРЕЖА НА ОСНОВІ ІНТЕРФЕЙСУ I^2C 246
1.1. Особливості архітектури інтерфейсу I^2C 246
1.2. Модуль I^2C мікроконтролерів AVR 249
1.2.1. Загальна характеристика модуля I^2C (TWI) мікроконтролерів AVR 249
1.2.2. Керування модулем TWI 251
1.2.3. Взаємодія прикладної програми з модулем TWI 252
2. МЕРЕЖА 1-WIRE 255
2.1. Особливості архітектури мережі 1-WIRE 255
2.2. Основні характеристики інтерфейсу та мережі 1-WIRE 255
2.3. Структура мережі 1-WIRE 256
2.4. Загальна характеристика датчика температури 257
2.5. Організація пам'яті датчика температури 258
2.5.1.Загальна характеристика пам'яті датчика 258
2.5.2. Регістр температури 258
2.5.3. Формування стану «Аварія» 259
2.5.4. Конфігураційний регістр 259
2.5.5. Контрольна сума циклічного надлишкового коду 260
2.5.6. Формат коду датчика 260
2.6. Команди мікроконтролера 260
2.6.1. Загальна характеристика команд 260
2.6.2. ROM-команди 261
2.6.3. Функціональні команди 261
2.7. Схема алгоритму роботи системи 262
2.8. Моделювання мережі 1-WIRE 262
2.8.1 Схема моделі 262
2.8.2. Опис роботи моделі 262

Контрольні запитання та завдання 263
ЛЕКЦІЯ 17. CAN-МЕРЕЖІ 266
1. CAN-МЕРЕЖІ 266
1.1. Загальні відомості про CAN-мережі 266
1.2. Основні характеристики CAN-протоколу 268
1.3. Структура повідомлень CAN-мережі 269
1.3.1. Загальна характеристика повідомлень 269
1.3.2. Кадр даних 270
1.3.3. Кадр віддаленого запиту даних 271
1.3.4. Кадр помилки 272
1.3.5. Кадр перевантаження 272
1.4. CAN-модуль AVR-мікроконтролера 272
1.4.1. Загальні відомості про CAN-модуль 272
1.4.2. cСтруктура CAN-модуля 273
1.4.3. Організація керувальних регістрів 273
1.4.4. Режими роботи CAN-модуля 274
1.4.5. Структура блока фільтрації 278
1.4.6. Структура переривань від CAN-модуля 279
1.4.7. Структура блока CAN-таймера 281
1.4.8. Обробка помилок 283
1.4.9. Бітова синхронізація 284
1.4.10. Фізичний рівень CAN-протоколу 287
Контрольні запитання та завдання 288
ЛЕКЦІЯ 18. МОДУЛЬ АНАЛОГОВОГО КОМПАРАТОРА. СПЕЦІАЛЬНІ РЕЖИМИ РОБОТИ МІКРОКОНТРОЛЕРА. 289
1. МОДУЛЬ АНАЛОГОВОГО КОМПАРАТОРА 289
1.1. Загальні відомості про компаратор 289
1.2. Аналоговий компаратор у складі AVR-мікроконтролерів 289
1.3. Функціонування та програмування компаратора 289
2. СПЕЦІАЛЬНІ РЕЖИМИ РОБОТИ МІКРОКОНТРОЛЕРА 291
2.1. Тактування мікроконтролера 291
2.1.1. Загальні відомості про тактування 291
2.1.2. Генератор із зовнішнім резонатором 293
2.1.3. Зовнішній сигнал синхронізації 293
2.1.4. Генератор із зовнішнім та внутрішнім RC-ланцюгом 294
2.1.5. Керування тактовою частотою 294
2.2. Режими зниженого енергоспоживання 294
2.2.1. Загальні відомості про режими зниженого енергоспоживання 294
2.2.2. Керування режимами зниженого енергоспоживання 295

2.3. Скидання 295
2.5. Самопрограмування AVR-мікроконтролерів 296
2.4.1. Загальні відомості про самопрограмування 296
2.4.2. Керування процесом самопрограмування 298
2.4.3. Зміна вмісту пам’яті програм 299
2.4.4. Типові процедури оновлення Flash-пам’яті 303
Контрольні запитання та завдання 305
СПИСОК ЛІТЕРАТУРИ 307

ВСТУП

Конспект орієнтовано на підготовку фахівців за освітньою програмою «Інтегровані інформаційні системи» спеціальності 126 «Інформаційні системи та технології», які вивчають проектування та використання мікропроцесорних систем.

Для успішного опановування навчального матеріалу необхідно засвоїти принципи побудови та використання мікропроцесорних пристроїв, на основі яких розробляються та функціонують мікропроцесорні та мікроконтролерні системи.

Конспект складається із 18 лекцій та охоплює теоретичний матеріал та практичні завдання, які необхідні для вивчення базової частини дисципліни «Електроніка та мікропроцесорна техніка».

В конспекті розглянуто такі питання: основні поняття та особливості мікропроцесорної техніки; структурна та функціональна схеми МПС; структури типового мікропроцесора (МП) та мікроконтролера (МК); програмні моделі МП та МК; мова Асемблера та її використання у МПС; характеристика команд МП та МК; способи адресації операндів і команди типового МП та МК; організація підсистеми переривань; формування інтервалів часу та підрахунок зовнішніх подій у МПС; організація пам'яті та модуля введення/виведення; інтерфейси: I^2C, SPI, CAN, 1-WIRE, RS-485 та RS-232; зв'язок МП та МК з аналоговим об'єктом керування; зв'язок МП/МК з модемом і т. ін.

Матеріал викладено так, що кожна наступна лекція є логічним продовженням попередньої. В кінці кожної лекції наведено контрольні запитання та завдання для самоконтролю.

Тематика лекцій відповідає робочій програмі дисципліни «Електроніка та мікропроцесорна техніка», 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 вміти: розраховувати, програмувати та моделювати мікропроцесорні та мікроконтролерні пристрої та системи; розробляти структурні, функціональні та принципові схеми та схеми алгоритмів роботи.

Конспект написано на основі досвіду викладання відповідних дисциплін згідно з програмами підготовки бакалаврів і магістрів спеціальності

126 «Інформаційні системи та технології» на кафедрі інформаційних систем та технологій у Національному технічному університеті України «Київський політехнічний інститут імені Ігоря Сікорського».

Курс забезпечується такими дисциплінами: математика, фізика, програмування, основи системної інженерії, комп'ютерні мережі, теорія інформації та кодування, комп'ютерна електроніка.

Зі свого боку, курс забезпечує засвоєння студентами більшості наступних дисциплін спеціальності, оскільки в ньому розглянуто архітектурні особливості сучасних мікропроцесорних та мікроконтролерних пристроїв та систем, які є основою сучасних інтегрованих інформаційних систем.

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

ЛЕКЦІЯ 1. ОСНОВНІ ПОНЯТТЯ МІКРОПРОЦЕСОРНОЇ ТЕХНІКИ. ЕЛЕКТРИЧНІ СХЕМИ ТИПОВИХ МІКРОПРОЦЕСОРНИХ СИСТЕМ

1. ОСНОВНІ ПОНЯТТЯ МІКРОПРОЦЕСОРНОЇ ТЕХНІКИ

1.1. Мікропроцесор

Проце́сор (англ. *processor*) – електронна схема, призначена для виконання арифметичних і логічних операцій над даними, здійснення введення та виведення даних і т. ін.

*Мікропроцесо*р (МП) – це процесор в мікромініатюрному інтегральному виконанні, що реалізований в одній великій інтегральній схемі та обробляє дані методом виконання команд із певного набору команд процесора.

В мікропроцесорних системах МП не може функціонувати без інших інтегральних схем, які виконують функції пам’яті, введення/виведення, формування часових інтервалів, аналого-цифрового та цифро-аналогового перетворювання і т. ін. Мікропроцесор реалізує функції, зазвичай покладені на центральний процесор електронної обчислювальної машини та виконує арифметичні і логічні операції, аналізує і приймає рішення, що змінюють процес обчислення, керує процесом обміну інформацією і т. ін.

1.2. Мікроконтролер

Мікроконтро́лер (МК) (англ. microcontroller), або однокристальний мікрокомп'ютер – виконаний у вигляді інтегральної мікросхеми спеціалізований комп’ютер, що включає мікропроцесор, оперативну та постійну пам’ять для збереження виконуваного коду програм і даних, і блоки зі спеціальними функціями (лічильники, компаратори, АЦП, ЦАП і т. ін.). По суті, мікроконтролер – це однокристальний комп’ютер, здатний виконувати прості завдання під час для керування електронними пристроями. Використання однієї мікросхеми значно знижує розміри, енергоспоживання і вартість пристроїв, побудованих на базі мікроконтролерів.

Мікроконтролери можна зустріти в багатьох сучасних приладах, таких як телефони, пральні машини, телевізори і т. ін.; вони відповідають за роботу двигунів і систем гальмування сучасних автомобілів; з їх допомогою створюються системи контролю і системи збору інформації. На основі мікроконтролерів проєктують та створюють вимірювальні пристрої, системи керування об’єктами та процесами, вони є основою охоронних, протипожежних систем, домофонів, сигналізацій і т. ін. Значна кількість процесорів, що випускаються у світі, це мікроконтролери.

1.3. Мікропроцесорна/мікроконтролерна система

Мікропроцесорна/мікроконтролерна система (МПС/МКС) – це система обробки даних, контролю та керування, побудована на базі одного або кількох мікропроцесорів/мікроконтролерів.

1.4. Основні характеристики МП/МК

До основних характеристик МП і МК можна віднести: довжину слова даних (розрядність); кількість комірок пам'яті, що адресуються, і швидкодію.

Довжина слова даних визначається максимальною кількістю розрядів оброблюваних даних, що розглядаються апаратною частиною МП/МК, як єдине ціле. У значної кількості вживаних на теперішній час МП/МК довжина слова даних становить 8, 16 та 32 біти.

Кількість адресованих комірок пам'яті залежить від числа адресних виводів МП/МК. Наприклад, 8-розрядний МП і8080 має 16 адресних ліній (виводів), що дозволяє адресувати 2^{16} = 65536 комірок пам'яті, завдовжки 8 біт (1 байт), тобто 65536 байт. У МП-техніці для вказівки об'єму пам'яті використовують скорочення 1К = 1024. Тоді МП і8080 адресує 64 Кбайт пам'яті. 16-розрядний МП і8086 має 20 адресних виводів, тобто здатен адресувати 2^{20} = 1 Мбайт пам'яті.

8-розрядний МК типу AT89C51 містить 16 адресних виводів, що забезпечує адресацію 64 Кбайт пам'яті.

Швидкодія (продуктивність) МП/МК визначається часом виконання окремих команд і програм. Для оцінювання швидкодії МП/МК можна використовувати:

а) максимальне значення тактової частоти ($f_{такт}$) або мінімальне значення періоду тактової частоти (тривалість такту машинного циклу);

б) час виконання найпростішої операції, наприклад, пересилання із регістра в регістр;

в) час виконання тестових програм.

Потужність МП/МК визначається названими раніше параметрами: довжиною слова даних, кількістю адресованих комірок пам'яті та швидкодією.

1.5. Системи числення та коди

1.5.1. Загальна характеристика систем числення та кодів

Система числення (англ. numeral system або system of numeration) – символічний метод запису чисел за допомогою письмових знаків. В мікропроцесорній техніці використовуються *позиційні* системи числення – двійкова, як основна, десяткова, шістнадцяткова і вісімкова, як допоміжні [1; 2].

Алфавіт системи числення – це набір цифр (символів), який використовують для запису чисел. *Основа системи числення* – це кількість цифр (символів) в алфавіті.

Позиційні системи числення мають *три важливі характеристики*: кількість цифр (символів) системи дорівнює її основі; найбільша цифра на одиницю менше основи; для *визначення значення кожної цифри* остання помножується на *вагу позиції* (*основа в степені*, значення якої визначається положенням цифри). Цілу і дробову частини, у записі дробового числа, завичай, розділяють символом «коми». Далі, в записі дробових чисел, будемо дотримуватися такого позначення, пам'ятаючи, що в комп'ютерних системах для цих речей використовується розділювач «крапка».

Для представлення дискретної інформації застосовуються *коди* с тими ж назвами, як у системах числення: двійковий; шістнадцятковий і т. ін. Про коди кажуть у тих випадках, коли в тій чи іншій системі числення представлені *дискретні повідомлення* (дискретна інформація), що називаються *даними*.

1.5.2. Десяткова система числення

В повсягденному житті використовується система числення з основою 10 – десяткова, в якій числа утворюються за допомогою цифр від 0 до 9. Слово «цифра» перекладається від латинського *digitus*, що означає «палець». Така кількість цифр – десять, пояснюється тим, що у нас десять пальців. Десяткова система є прикладом *позиційної системи числення*, коли положення (позиція, розряд) кожної цифри в числі визначає її значення, або вагу.

Наприклад, число 374.29 є скороченим записом виразу:

$$(3\times10^{2})+(7\times10^{1})+(4\times10^{0})+(2\times10^{-1})+(9\times10^{-2}).$$

Крапка в позиційному представленні десяткових чисел називається *десятковою крапкою* і використовується для відділення цілої частини числа від дробової частини. Тому можна записати, що

$$N = N_I + N_F, \qquad (1)$$

де N_I та N_F – відповідно ціла і дробова частини числа.

Положення будь-якої цифри визначає степінь числа з основою 10 (*вага позиції/розряду числа*), на яке ця цифра помножується. В даному прикладі цифра 3 знаходиться в розряді сотень (10^2), 7 – десятків (10^1), 4 – одиниць (10^0), 2 – десятих долей (10^{-1}) і 9 – сотих долей (10^{-2}). В нашому прикладі N_I = 374, а N_F = 0,29.

1.5.3. Двійкова система числення

Найпростішою позиційною системою числення є двійкова, яка має основу 2. Для представлення чисел використовуються дві десяткові цифри – 0 і 1. Наприклад, число 1011.1101_2 еквівалентно десятковому числу:

$$(1\times 2^3)+(0\times 2^2)+(1\times 2^1)+(1\times 2^0)+(1\times 2^{-1})+(1\times 2^{-2})+(0\times 2^{-3})+(1\times 2^{-4})=11{,}8125_{10}.$$

Записуючи числа в тій чи іншій системі числення, *підрядкові символи числа* визначають основу системи числення. Їх зазвичай опускають, якщо відомо, про яку основу йдеться. Окрім підрядкових символів для вказівки системи числення, в якій записано число, можуть використовуватися латинські букви: *B* – для двійкової (бінарної); *D* – для десяткової; *H* – для шістнадцяткової. Наприклад: 0110*B*; 7548*D*; 7598*H*; ABBA*H*.

Необхідно відзначити, що в наведеному прикладі крапка в позиційному (двійковому) представленні числа відокремлює його цілу і дробову частину, як і в випадку десяткової крапки. При поданні інформації у двійковому коді (ДВК) *праворуч* записують значення *молодшого* (нульового) значущого розряду *(МЗР)*, *ліворуч – значення* старшого (*Np*–1)-го) значущого розряду *(СЗР)*, де N_p – число розрядів ДК.

Одним із методів переведення чисел із десяткової системи числення у двійкову є метод *ділення-множення* [1; 2].

Лічба у двійковій системі числення менш складна, ніж в інших системах, оскільки тут використовується усього дві цифри – 0 і 1. Якщо трапляється деяка послідовність подій, то *першій* події відповідає цифра 1. Однак вже для *другої* події цифри два у двійковій системі не має. У цьому випадку відбувається *перенесення* в наступний розряд, що дає дворозрядне двійкове число. Скажімо, у десятковій системі, коли число у поточному розряді сягає десяти, то розряд обнуляється і одиниця додається до старшого. Наприклад: 9 + 1 = 10, 44 + 7 = 51.

Аналогічним чином у двійковій системі: коли число в розряді сягає двох – розряд обнуляється і одиниця додається до старшого розряду. Тобто: 1 + 1 = 10. Якщо перевести це число в десяткову систему числення, то отримаємо бажаний результат: $10_2=(1\times 2^1)+(0\times 2^0)=2_{10}$.

У табл. 1 представлено послідовність двійкових чисел, яка відповідає десятковим числам від 0 до 9.

Згідно таблиці *перенесення одиниці у наступний* – більш старший розряд відбувається *під час зображення усіх парних* десяткових чисел.

Під час зображення десяткових чисел 2, 4, 8, 16 і т. д. відбуваються відповідно перенесення із розряду двійок (2^1), четвірок (2^2), вісімок (2^3) і т. д.

1.5.4. Шістнадцяткова система числення

Для запису двійкових чисел *великої розрядності* використовують їх більш компактне представлення у шістнадцятковій системі. В цій системі числення числа представляються за степенями основи 16 ($16 = 2^4$), що еквівалентно кількості комбінацій чотирирозрядного двійкового числа.

Таблиця 1. Представлення чисел у двійковій та десятковій системах числення

Двійкове число	**Десяткове число**
0	0
1	1
10	2
11	3
100	4
101	5
110	6
111	7
1000	8
1001	9

В шістнадцятковій системі числення використовуються наступні символи: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, *A*, *B*, *C*, *D*, *E*, *F* (табл. 2).

Таблиця 2. Представлення чисел у десятковій, двійковій та шістнадцятковій системах числення

Десяткове число	**Двійкове число**	**Шістнадцяткове число**
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Переведення із шістнадцяткової системи в інші системи числення здійснюється аналогічно двійковій і десятковій. Аналогічно виконуються арифметичні дії над шістнадцятковими числами.

1.5.5. Двійково-десяткові числа (коди)

Є розбіжність між *машинним представленням* чисел (*двійкова* система числення) і представленням чисел у нашому *повсякденному* житті (*десяткові* числа). Перетворення між ними, у випадку великого об'єму вхідних даних і вихідних результатів, приводить до помітних втрат процесорного часу.

Разом з тим, є велике коло задач, що характеризуються значними об'ємами числових даних і порівняно простою їх обробкою (економічні задачі, статистичні та бухгалтерські розрахунки і т. ін.). *У обчислювальній техніці* існує форма представлення чисел, в якій *поєднувалися* двійкова і десяткова системи числення. Така форма має назву *двійково-десяткового кодування* (Binary-Coded Decimal) або BCD-кодування. Загальна властивість цієї форми виявляється в тому, що *за основу береться десяткове число* і кожна його цифра зображується тим чи іншим *двійковим еквівалентом.* Використовуються *дві форми* BCD-кодування: двійково-десяткові числа в запакованому і незапакованому форматах.

У запакованому форматі байт (вісім біт) містить дві десяткові цифри. Менша цифра займає праву тетраду (біти 3…0), старша – ліву тетраду (біти 7…4). Обидві цифри представлені своїми двійковими еквівалентами та називаються також кодом 8421 – за двійковими вагами окремих розрядів тетради.

У незапакованому форматі десятковим числам відповідають *двійкові набори* від 0000 0000 (цифра 0) до 0000 1001 (цифра 9), або в шістнадцятковій системі від 00H до 09H. Власне значення десяткової цифри займає *молодшу тетраду* і представлено її двійковим еквівалентом, а *в старшій тетраді* знаходиться комбінація *0000*.

Приклади розміщення десяткових цифр у байті та розміщення багаторозрядного BCD-числа в пам'яті у разі використання BCD-кодів у запакованому та незапакованому форматах наведено у [1; 2].

1.5.6. Представлення десяткових чисел в коді ASCII

Для кодування *алфавітно-цифрової* інформації або символів використовується *7-розрядний ASCII-код* (американський стандартний код для обміну інформацією. У [1; 2] наведено приклади кодування у цьому коді десяткових цифр та латинських символів.

Дана форма під час кодування десяткових цифр відрізняється від незапакованого формату тим, що *старша тетрада байта* замість $0000B = 0D$ містить $0011B = 3D$. Тобто у цьому коді *десятковим цифрам* відповідають набори у

двійковій системі числення: від 00110000 (цифра 0) до 00111001 (цифра 9), або в шістнадцятковій системі: від 30H до 39H.

1.6. Формати подання чисел

1.6.1. Подання чисел зі знаком

Всі системи числення допускають використання як додатних, так і від'ємних чисел, для позначення яких, як відомо, застосовують знаки плюс (+) або мінус (–). Число в такому поданні називається *числом зі знаком*.

Оскільки ЕОМ побудовані на схемах двійкової логіки та для подання двійкових станів застосовуються символи нуль і одиниця, тому ці ж символи використовуються і для позначень знака числа. Додатковий розряд для подання знака числа, називається *знаковим розрядом* і розміщується з лівого боку найбільшого значущого розряду числа. Для позначення *додатних і від'ємних значень використовують відповідно: нуль та одиниця.*

У МП-техніці існують *три способи подання двійкових чисел зі знаком*, які відомі як прямий, зворотний і додатковий код числа [1; 2]. Найбільш часто в МП-техніці використовується *додатковий* код числа.

Додатковий код (англ. «two's complement», іноді «twos-complement») дозволяє замінити операцію віднімання операцією додавання і зробити операції додавання і віднімання однаковими для знакових і беззнакових чисел, що спрощує архітектуру МПС. В англомовній літературі «додатковий код» називають «two's complement» («доповненням до двійок»). Додатковий код *додатного* числа збігається із прямим кодом. *Додатковий код для від'ємного* числа *можна отримати* інвертуванням його двійкового модуля та додаванням до інверсії одиниці, або віднімання числа з нуля.

Перевагами подання чисел у вигляді додаткового коду є:

– операції додавання і віднімання на практиці виконуються просто;
– є єдине подання нуля.

Діапазон десяткових чисел зі знаком, які можна представити у додатковому ДВК, *залежить від кількості розрядів* останнього.

У табл. 3 наведено два типові для МП способи використання ДВК: як двійкових чисел зі знаком, так і без знака. *Лівий стовпчик* містить двійкові числа від 0000 0000 до 1111 1111, *правий стовпчик* – їх десяткові еквіваленти від 0 до 255, отримані з припущення, що розглядаються числа без знака. *У центральному стовпчику* знаходяться десяткові еквіваленти двійкових чисел лівого стовпчика, отримані з припущення, що *від'ємні числа* записувалися у *додатковому коді*. Тут *додатним* двійковим числам (від 0000 0000 до 0111 1111) відповідають десяткові числа від 0 до +127, а тим, що вміщують одиницю в восьмому розряді

(сьомому, якщо рахувати від нуля) *від'ємним* двійковим числам (від 1000 0000 до 1111 1111) – десяткові від: –128 до – 1.

Таблиця 3. Десяткові еквіваленти двійкових чисел

Двійковий еквівалент	**Десятковий еквівалент**	
Восьмирозрядні двійкові числа 7р 0р	Двійкові числа зі знаком (від'ємні числа у додатковому коді)	Двійкові числа без знака
0000 0000	+0	0
0000 0001	+1	1
0000 0010	+2	2
0000 0011	+3	3
.	.	.
.	.	.
.	.	.
0111 1100	+124	124
0111 1101	+125	125
0111 1110	+126	126
0111 1111	+127	127
1000 0000	–128	128
1000 0001	–127	129
1000 0010	–126	130
1000 0011	–125	131
.	.	.
.	.	.
1111 1100	–4	252
1111 1101	–3	253
1111 1110	–2	254
1111 1111	–1	255

Як видно з табл. 3 за допомогою *8 біт* можна подати у двійковій формі десяткові числа зі знаком від: *–128 до +127*, включаючи 0.

Якщо кількість розрядів ДВК *дорівнює 16*, тоді діапазон десяткових чисел: *– 32768 …+ 32767.*

Нижче наведено *приклад* формування додаткового ДВК від числа – 3_{10}:

– число 3_{10} у двійковій формі: 0000 011;
– зворотний код числа 3_{10}: 1111 100;
– додавання до зворотного коду: 1;
– число: -3_{10} у додатковому коді: 111 1101.

Відповідно до табл. 3 арифметичні операції над двійковими числами без знака нічим не відрізняються від подібних операцій над двійковими числами зі знаком, від'ємні з яких подані своїми доповненнями. Це істотно спрощує апаратну реалізацію подібних операцій у МП/МК. Однак слід звернути увагу на те, з якими числами ви маєте справу в даний момент: без знака чи зі знаком.

Наприклад, у разі додавання двох чисел без знака результат – число без знака подається у вигляді деякої послідовності бітів, котру можна інтерпретувати і як від'ємне число у додатковому коді. В загальному випадку у разі додавання або віднімання чисел зі знаком результат є число зі знаком. У цьому разі, якщо біт старшого розряду дорівнює одиниці, то результат – від'ємне число у додатковому коді.

Якщо потрібно визначити *абсолютне значення* (величину) *від'ємного* результату, останній необхідно представити у зворотному коді, а потім додати одиницю.

Є простіший спосіб визначення *абсолютного значення від'ємного* двійкового числа: необхідно додати ваги двійкових розрядів, які вміщують нулі, і до суми додати одиницю.

В простоті операцій над від'ємними числами у вигляді додаткового ДВК можна переконатися *на прикладі операції віднімання*, яка виконується наступним чином: визначається додатковий код *від'ємника* і виконується додавання цього коду зі *зменшуваним*. Якщо різниця – число додатне (біт старшого розряду дорівнює нулю), то біт перенесення необхідно відкинути: отримана послідовність бітів і є ДВК. Якщо різниця – число від'ємне (біт старшого розряду дорівнює одиниці), то вона представлена в додатковому ДВК. Вище вказувалося, що необхідно зробити для визначення абсолютної величини від'ємного числа, поданого в такому вигляді.

Нижче наведено приклад виконання операції віднімання в МП/МК.

Обчислити різницю чисел 26 – 34:

а) визначення додаткового коду числа 34

0 0 1 0 0 0 1 0 число 34_{10}
1 1 0 1 1 1 0 1 зворотний код числа 34_{10}
<u>0 0 0 0 0 0 0 1</u> одиниця, що додається до зворотного коду
1 1 0 1 1 1 1 0 додатковий код числа 34_{10} (-34_{10})

б) обчислення різниці

Десяткова арифметика	Двійкова арифметика	
26	0 0 0 1 1 0 1 0	число 26_{10}
–	+	
34	1 1 0 1 1 1 1 0	додатковий код числа 34_{10} (-34_{10})
– 08	1 1 1 1 1 0 0 0	різниця: -08_{10}

в) визначення абсолютного значення різниці

1 1 1 1 1 0 0 0 різниця у додатковому коді

0 0 0 0 0 1 1 1 зворотний код різниці

0 0 0 0 0 0 0 1 одиниця, що додається до зворотного коду

0 0 0 0 1 0 0 0 абсолютне значення різниці (8_{10}).

Другий спосіб визначення абсолютного значення від'ємного числа 11111000: $2^2 + 2^1 + 2^0 + 1 = 4 + 2 + 1 + 1 = 8$.

1.6.2. Числа з нефіксованою крапкою

Окрім розглянутих вище форматів чисел з фіксованою крапкою, в МП-техніці застосовуються також формати з нефіксованою крапкою, розглянуті в [1; 2].

1.6.3. Угруповання біт

Окремі двійкові біти (розряди) можуть об'єднуватися в групи, що мають назви:

- тетрада (група з 4-х біт);
- байт (група з 8-ми біт);
- слово (група з 16-ти біт);
- подвійне слово (група з 32-х біт).

1.7. Двійкова арифметика

1.7.1. Двійкове додавання

Двійкове *додавання* виконується за наступними правилами:

а) 0 + 0 = 0;

б) 0 + 1 = 1;

в) 1 + 1 = 0 плюс перенесення у наступний розряд;

г) 1 + 1 + 1 = 1 плюс перенесення у наступний розряд.

Розглянемо, як приклад, додавання чисел 1110 і 1100:

1 1 1 0	перший доданок (14_{10});
+1 1 0 0	другий доданок (12_{10});
1 1 0 1 0	сума (26_{10});.

1.7.2. Двійкове віднімання

Двійкове віднімання виконується за наступними правилами:

а) 0 – 0 = 0;

б) 1 – 0 = 1;

в) 1 – 1 = 0;

г) 0 – 1 = 1 – позика одиниці зі старшого розряду.

Розглянемо, як приклад, віднімання числа 1100_2 із 10110_2.

1 0 1 1 0 зменшуване ($+22_{10}$);

– 1 1 0 0 від'ємник ($+12_{10}$);

0 1 0 1 0 різниця ($+10_{10}$).

В МП/МК операція віднімання замінюється додаванням зменшуваного з додатковим кодом від'ємника (п. 1.6.1).

1.7.3. Двійкове множення

Під час двійкового множення частковий добуток зсувається на один розряд вліво для обробки кожного наступного розряду множника.

Множення чисел здійснюється за наступними правилами: а) 0 х 0 = 0; б) 0 х 1 = 0; в) 1 х 0 = 0; г) 1 х 1 = 1.

Розглянемо, як приклад, множення 5_{10} х 3_{10} = 15_{10}:

$0\,1\,0\,1_2$ ($+5_{10}$);

х $0\,0\,1\,1_2$ ($+3_{10}$);

$0\,1\,0\,1_2$

$+\,0\,1\,0\,1_2$

$0\,1\,1\,1\,1_2$ ($+15_{10}$).

1.7.4. Двійкове ділення

Операція ділення є зворотною відносно множення і реалізується схожими циклічними діями. Алгоритм ділення наведено у [1].

1.8. Двійково-десяткова арифметика

Вище були розглянуті два формати представлення двійково-десяткових чисел: *запакований і незапакований*. У сучасних МП/МК немає команд, що дійсно оперують названими числами. Виконання операцій над двійково-десятковими числами здійснюється *за два етапи:*

– *на першому* етапі операнди обробляються як цілі двійкові числа командами двійкової арифметики;

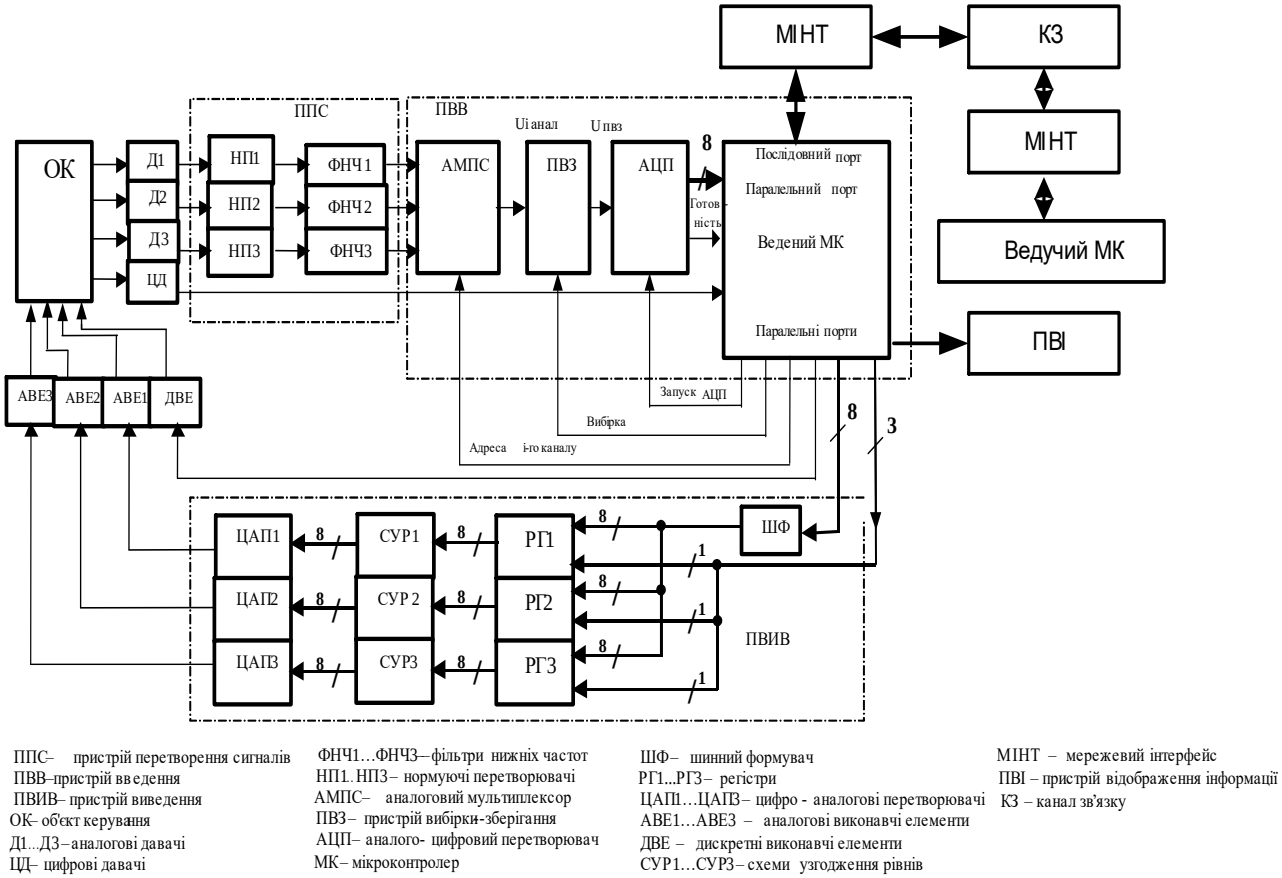
– *на другому* – спеціальні *команди корекції* перетворюють проміжний результат у двійково-десятковий формат.

Багато команд корекції для обох форматів двійково-десяткових чисел представлені, наприклад, у шістнадцятирозрядному МП i8086 [1].

2. ЕЛЕКТРИЧНІ СХЕМИ ТИПОВИХ МІКРОПРОЦЕСОРНИХ СИСТЕМ

2.1. Структурна схема типової локальної мікроконтролерної системи керування

Розглянемо приклад типової *гіпотетичної локальної* мікроконтролерної системи керування (ЛМКСК), структурну схему якої наведено на рис. 3 [2].



ППС– пристрій перетворення сигналів
ПВВ–пристрій введення
ПВИВ– пристрій виведення
ОК- об'єкт керування
Д1…ДЗ – аналогові давачі
ЦД– цифрові давачі

ФНЧ1…ФНЧЗ– фільтри нижніх частот
НП1..НПЗ– нормуючі перетворювачі
АМПС– аналоговий мультиплексор
ПВЗ– пристрій вибірки-зберігання
АЦП– аналого- цифровий перетворювач
МК– мікроконтролер

ШФ– шинний формувач
РГ1...РГЗ– регістри
ЦАП1…ЦАПЗ– цифро - аналогові перетворювачі
АВЕ1…АВЕЗ – аналогові виконавчі елементи
ДВЕ – дискретні виконавчі елементи
СУР1…СУРЗ– схеми узгодження рівнів

МІНТ – мережевий інтерфейс
ПВІ – пристрій відображення інформації
КЗ – канал зв'язку

Рис. 3. Структурна схема типової ЛМКСК

ЛМКСК керує визначеним об’єктом керування (агрегатом) за декількома параметрами, наприклад, температурою, тиском, кутом повороту, переміщенням і т. ін. ЛМКСК підтримує кожний з конкретних параметрів на заданому рівні. Система названа локальною, оскільки керування виробляється і здійснюється на нижньому (локальному) рівні складної розподіленої системи керування, що включає декілька різних агрегатів (об’єктів керування). Основним елементом ЛМКСК є мікроконтролер (МК), який називається *веденим*, оскільки передбачається, що в складній розподіленій системі мається декілька подібних ведених МК, які керують окремими агрегатами на локальному рівні.

На вищому рівні ієрархії *розподіленої системи керування* може знаходитися *ведучий* МК, який на основі інформації про стан окремих агрегатів виробляє необхідні значення відповідних керувальних впливів для ведених МК.

Ведучий і ведений МК можуть бути зв'язані між собою *мікроконтролерною мережею* [2].

Розглянемо роботу системи за наведеною вище структурною схемою. Інформація про поточне значення параметрів контролю знімається з *аналогових давачів* (Д1...Д3) і проходить через *нормувальні перетворювачі* (НП1...НП3), які перетворюють діапазон зміни електричних сигналів, що знімаються з давачів, до діапазону вхідних сигналів, що відповідає обраному АЦП.

Оскільки *інформаційні сигнали* в більшості систем керування – *низькочастотні*, то для придушення високочастотних завад використовуються фільтри нижніх частот (*ФНЧ)*.

АМПС по черзі підключає до АЦП один з трьох аналогових електричних сигналів, які відображають поточні значення контрольованих параметрів. У випадку, якщо за час перетворення АЦП, зміна вхідного сигналу відповідає зміні вихідного ДВК на виході АЦП більше, ніж на одиницю МЗР, то для зменшення так званої «апертурної» похибки, яка виникає у цьому разі, в систему *включають ПВЗ* [1]. ПВЗ запам'ятовує миттєві значення вхідних аналогових сигналів в момент часової вибірки і підтримує їх постійними на вході АЦП протягом часу перетворення останнього. *З виходу АЦ*П інформація у паралельному ДВК надходить у *ведений МК*, який порівнює поточне значення контрольованого параметра з заданим значенням і виробляє керувальний вплив відповідно до сигналу розузгодження та обраним законом керування (П, ПІ, ПІД). Сигнали керування, наприклад, для трьох АВЕ знімаються з виходу одного з *паралельних портів МК* та запам'ятовуються у зовнішніх *регістрах РГ1...РГ3.* Для *підвищення навантажувальної* здатності виходів паралельного порту МК в системі використано ШФ. Виходи РГ1…РГ3 через СУР1…СУР3 зв'язано з *входами ЦАП1…ЦАП3,* що формують аналогові керувальні впливи, спрямовані на усунення сигналу розузгодження і призначені для відпрацьовування АВЕ1…АВЕ3. *СУР1…СУР3 необхідні* в тих випадках, коли рівні одиничних логічних сигналів, що знімаються з виходів регістрів, не відповідають необхідним рівням одиничних сигналів на входах ЦАП [1; 2; 5]. В якості СУР, як правило, використовують *логічні елементи з відкритим колектором* [5].

В загальному випадку, ЛМКСК окрім аналогових давачів і виконавчих елементів можуть містити *цифрові давачі і дискретні виконавчі елементи*, які через паралельні порти та відповідні інтерфейси поєднуються з МК. Багато сучасних давачів можуть на своїх виходах формувати сигнали у *форматах інтерфейсів*: SPI, I^2C, RS-485, RS-232, 1-WIRE і т. ін. [4; 7; 8]. В цьому випадку для їх обробки в МК останній повинен мати *відповідний модуль*, або використовувати підпрограми.

У випадку, коли ведучий та ведені МК знаходяться на відстані, наприклад, сотень метрів, для їх звязку використовують мікроконтролерні *мережі:* CAN, RS-485, 1-WIRE і т. ін. [2; 4: 7; 8]. За більшої відстані застосовуються модеми.

У разі необхідності в ЛМПСК також використовуються *ПВІ*. Для цього можуть застосовуватися семисегментні світлодіодні або рідкокросталічні *індикатори,* рідкокристалічні дисплеї і т. ін. У [1] наведено приклади реалізації деяких вузлів ЛМКСК, структуру якої наведено вище.

2.2. Функціональна схема мікропроцесорної системи керування

Роботу і застосування будь-якої *МПС необхідно розглядати* як на апаратному, так і на програмному рівні, оскільки ці дві властивості МПС нероздільні одна від одної [1; 2]. Аналіз МПС *на апаратному* рівні починають з вивчення *структурної* схеми, яку розглянуто вище.

Розглянемо приклад *функціональної* схеми *гіпотетичної* мікропроцесорної системи керування (МПСК) з використанням восьмирозрядного МП типу і8080, яку наведено на рис. 4.

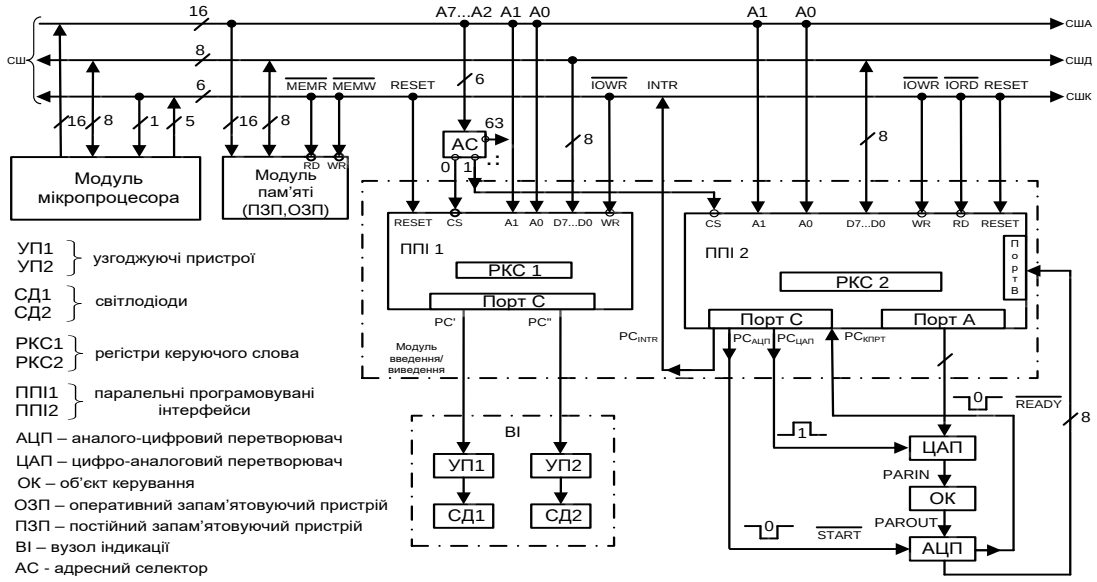


Рис. 4. Функціональна схема гіпотетичної МПСК

МПСК орієнтована на *восьмирозрядний МП*, наприклад і8080, і містить *наступні складові*: модуль МП; модуль пам'яті: постійний запам'ятовувальний пристрій (ПЗП); оперативний запам'ятовувальний пристрій (ОЗП); модуль введення/виведення (МВВ/ВИВ) – два програмовані паралельні інтерфейси (ППІ1 та ППІ2); ЦАП; АЦП; ОК та ВІ.

В ПЗП зберігається *робоча програма*, під керуванням якої працює МПСК, а в ОЗП зберігаються дані, а також знаходиться стек. МВВ/ВИВ , що складається з двох ППІ, забезпечує обмін інформацією між МП, ОК та індикацією.

АЦП та ЦАП призначені для поєднання аналогового об'єкта керування з цифровими вузлами МПСК.

Цифрові вузли МПС зв'язані між собою *системною шиною* (СШ), що складається із трьох окремих системних шин: системної шини адреси (США); системної шини даних (СШД) та системної шини керування (СШК).

Системною шиною адреси передається адреса від МП до пристрою, з яким у даний момент часу здійснюється обмін інформацією: пам'яттю або ПВВ. *Системною шиною даних* МП обмінюється інформацією (даними) між пам'яттю і ПВВ. В системі, що розглядається, *із пам'яті в МП по СШД* передаються команди робочої програми, яка керує роботою МПСК.

Крізь ППІ1 від МП *до вузла індикації по СШД* передається інформація, що відображає стан ОК. Від ППІ2 у МП *по СШД передається інформація про поточне значення контрольованого параметра* PAROUT. *Від МП до ППІ2 по СШД* пересилається керувальна інформація для підтримки контрольованого параметра ОК на заданому рівні.

Системною шиною керування передаються керувальні сигнали, які виробляються модулем МП. В розглянутому прикладі таких сигналів п'ять: RESET – системне скидання (здійснює початкове налагодження (ініціалізацію) ППІ1 і ППІ2); $\overline{\text{MEMR}}$ – читання пам'яті (керує читанням програми з ПЗП або даних з ОЗП); $\overline{\text{MEMW}}$ – запис у пам'ять (керує записом до ОЗП); $\overline{\text{IORD}}$ – читання пристрою введення/виведення (керує введенням інформації від зовнішнього пристрою (ЗВПР) до МП); $\overline{\text{IOWR}}$ – запис у пристрій введення/виведення (керує виведенням інформації з МП у зовнішній пристрій).

Один керувальний *сигнал (INTR*) передається за СШК в МП і дозволяє організувати обмін даними через ППІ2 за перериванням. В реальних системах кількість керувальних сигналів, що передаються за СШК, може бути більше названих п'яти.

В конкретний момент часу МП виконує передачу інформації *в одному з двох напрямів*: із МП у пам'ять (для МПСК, що мають ОЗП) або МВВ/ВИВ; із пам'яті або МВВ/ВИВ у МП.

У першому випадку кажуть, що відбувається *запис* (виведення, передача), а у другому – *читання* (введення, прийом). *В обох* випадках інформація передається *за СШД*.

Окрім того, *необхідно підкреслити*, що МП не може одночасно працювати з пам'яттю і МВВ/ВИВ, а взаємодіє тільки з одним із них, який обраний адресним селектором і керувальним сигналом $\overline{\text{MEMRD}}$ / $\overline{\text{MEMW}}$ або $\overline{\text{IORD}}$/$\overline{\text{IOWR}}$.

Оскільки передбачається, що в прикладі (рис. 4) *ОК – аналоговий*, він містить аналогові давач і виконавчий елемент. Тоді *для зв'язку такого ОК* з цифровими

вузлами МПС використовуються: *АЦП у разі введення* в МПСК значення контрольованого параметра від давача, та *ЦАП у разі виведення* керувального впливу на виконавчий елемент.

До складу ЦАП на рис. 4 умовно включено: власне ЦАП, що містить резисторну матрицю R-2R та операційний підсилювач; буферний регістр, в який у ДВК переписується керувальний вплив із порту А ППІ2 в момент надходження синхроімпульсу, що програмно сформований на лінії порту С ППІ2 ($РС_{ЦАП}$). *В реальній системі* буферний регістр може не використовуватися, а його функцію виконує регістр порту А ППІ2.

Для запуску АЦП на лінії $РС_{АЦП}$ порту С ППІ2 програмно формується керувальний імпульс ($\overline{START}$) низького рівня. *Після завершення* аналого-цифрового перетворення АЦП формує сигнал «Готовність» ($\overline{READY}$), який також має низький рівень та поступає в систему за лінією $РС_{КПРТ}$ порту С ППІ2. *Дані від АЦП в МП* передаються на СШД через порт В ППІ2.

Існує *два способи* введення даних у МП: *програмно-керований* та *за перериванням*. *У першому випадку* програма постійно перевіряє значення сигналу $РС_{КПРТ}$ і, у разі виявлення логічного нуля, керує введенням даних у МП. Очевидно, що в цьому разі МП використовується не дуже ефективно, оскільки витрачає свій машинний час на постійне опитування біта $РС_{КПРТ}$, доки АЦП не закінчить перетворення. *У другому випадку* МП може виконувати необхідну роботу по обробленню даних до надходження сигналу «INTR – запит переривання», що формується в момент закінчення перетворення АЦП і за СШК передається на однойменний вивід МП. У цьому разі *відбувається переривання* основної програми, і черговий байт вводиться в МП (у нашому прикладі через порт В ППІ2). *Після введення* чергового байта формується сигнал «*RESET» для АЦП*, що передається тією ж лінією, що і сигнал «$\overline{START}$», але має *інверсне до нього* одиничне значення.

Зображений на рис. 4 *адресний селектор* може бути реалізовано за допомогою *двійкового дешифратора*, що перетворює вхідний *паралельний шестирозрядний* ДВК (розряди А7…А2 США) у *багатопозиційний унітарний* код (число позицій дорівнює $2^6 = 64$) з активними нульовими вихідними сигналами. Активний логічний нуль буде з'являтися на тому виході дешифратора, номер якого є десятковим еквівалентом вхідного ДВК.

В розглянутому прикладі *використовуються тільки два виходи з 64-х* (0 і 1), що у потрібний момент обирають один із двох ППІ. *Інші виходи* в реальній системі можуть застосовуватися для адресації інших зовнішніх пристроїв.

В реальній системі функції МВВ/ВИВ можуть бути виконані за допомогою *тільки одного ППІ*. В наведеній вище схемі (рис. 4) не використовуються чотири

розряди порту С ППІ2, два з яких можна використати для керування вузлом індикації.

Робота МПСК зводиться до підтримування аналогового вихідного сигналу об'єкта керування, позначеного PAROUT, в межах заданого діапазону (рис. 5):

$$PARMIN \leq PAROUT < PARMAX, \qquad (2)$$

де PARMIN і PARMAX – відповідно мінімальне і максимальне значення регульованого параметра.

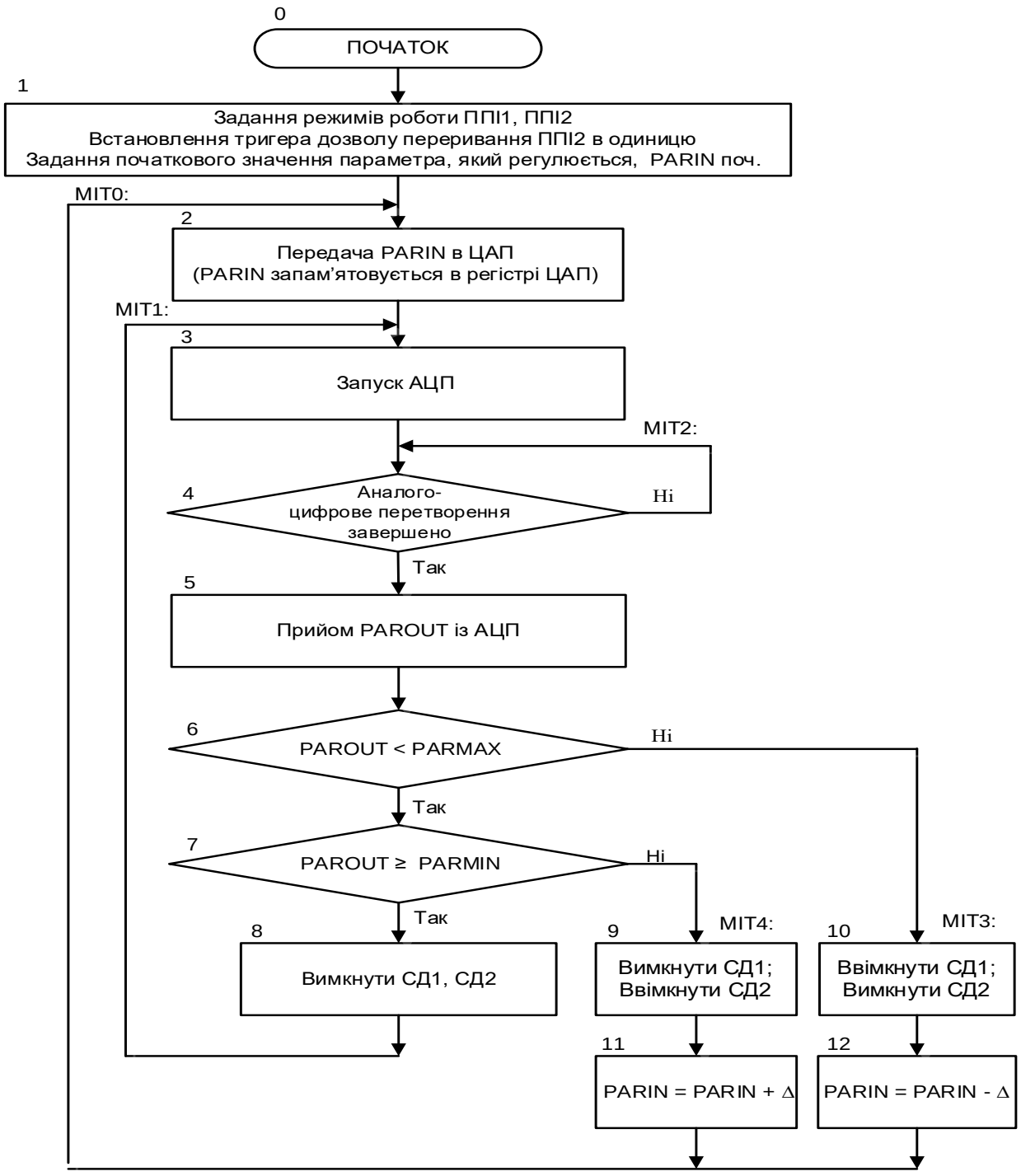


Рис. 5. Схема алгоритму роботи МПСК

На початку роботи для виведення ОК на робочий режим через порт А ППІ2 у ЦАП передається початкове значення регульованого параметра $PARIN_{поч}$.

Якщо в процесі роботи системи значення PAROUT *знаходиться в межах* зазначеного вище діапазону, то значення $PARIN_{поч.}$ не змінюється.

Якщо PAROUT вийшов за межі діапазону з боку мінімуму чи максимуму, то поточне значення PARIN змінюється з кроком ±Δ, розмір якого може програмно змінюватися. Ця зміна буде вироблятися доти, поки PAROUT знову не опиниться в межах заданого діапазону.

Порушення чи не порушення границь діапазону у розглянутій системі відображуються вузлом індикації (двома світлодіодами СД1 і СД2).

Індикацією відображуються три стани регульованого об'єкта:

- норма (PAROUT знаходиться в межах допуску) – обидва світлодіоди вимкнено;
- PAROUT ≥ PARMAX (вихід за межі допуску з боку максимуму) – вмикається СД1 і вимикається СД2;
- PAROUT < PARMIN (вихід за межі допуску з боку мінімуму) – вимикається СД1 і вмикається СД2.

Розробку програмного забезпечення звичайно починають зі схеми алгоритму роботи системи (рис. 5). *Після цього розробляється* і налагоджується керувальна програма мовою Асемблер або С обраного МП/МК. *Після налагодження* ця програма компілюється – транслюється в машинні коди процесора і створюється об'єктний модуль, що на спеціальному пристрої – *програматорі завантажується* в ПЗП МПСК.

Контрольні запитання та завдання

1. Що таке МП та МК?
2. Що таке системна шина керування? Наведіть приклад керувальних сигналів.
3. Від чого залежить кількість адресованих комірок пам'яті МП або МК?
4. За значеннями яких параметрів можна оцінювати швидкодію МП?
5. Що таке система числення? Назвіть відомі вам системи числення.
6. Що таке позиційна система числення? Наведіть приклад позиційної системи числення.
7. Запишіть беззнакове число:
   - 010111012 у десятковій системі числення;
   - 110011102 у шістнадцятковій системі числення;
   - 72310 у двійковій системі числення;
   - 59610 у шістнадцятковій системі числення;
   - 8C16 у двійковій системі числення;
   - F5A16 у десятковій системі числення;
   - 73510 у запакованому BCD-форматі;

– 57110 у незапакованому BCD-форматі.

8. Запишіть у восьмирозрядному додатковому коді число зі знаком: –35.
9. Знайдіть суму та різницю беззнакових чисел: 010011012 і 000001112.
10. Чим відрізняється мікроконтролер від мікропроцесора?
11. Яка інформація зберігається в постійному запам’ятовуючому пристрої (ПЗП) та оперативному запам’ятовуючому пристрої (ОЗП)?
12. Яку функцію у гіпотетичній мікропроцесорній системі керувіння виконують паралельні програмовані інтерфейси, аналого-цифровий перетворювач (АЦП) та цифро-аналоговий перетворювач (ЦАП)?
13. Назвіть складові системної шини МПСК та опишіть їх призначення.
14. Назвіть сигнали, які передаються у гіпотетичній мікропроцесорній системі керування системною шиною, та опишіть їх призначення.
15. Як називають процес передачі інформації із МП в пам’ять або в зовнішній пристрій у мікропроцесорній системі керування?
16. Як називають процес передачі інформації із пам’яті або в зовнішнього пристрою в МП у мікропроцесорній системі керування?
17. Пояснити, як у реальній мікропроцесорній системі керування замінити одним інтерфейсом два паралельних інтерфейси, які використовуються у гіпотетичній МПСК?
18. Чи може МП у мікропроцесорній системі керування одночасно працювати з пам’яттю і зовнішнім пристроєм? Відповідь пояснити.
19. Назвіть та поясніть призначення керувальних сигналів МПСК для АЦП.
20. Назвіть та поясніть способи введення даних від об’єкта керування у МП.
21. Наведіть та поясніть схему алгоритму роботи гіпотетичної МПСК.

ЛЕКЦІЯ 2. МОДУЛЬНА СТРУКТУРА МПС. СТРУКТУРНІ СХЕМИ ТИПОВОГО МП ТА МК

1. МОДУЛЬНА СТРУКТУРА МПС

1.1. Загальна характеристика модульної структури МПС

На рис. 1 наведено модульну структуру МПС, яку побудовано на базі 8-розрядного мікропроцесора (МП), наприклад, і8080 [1].

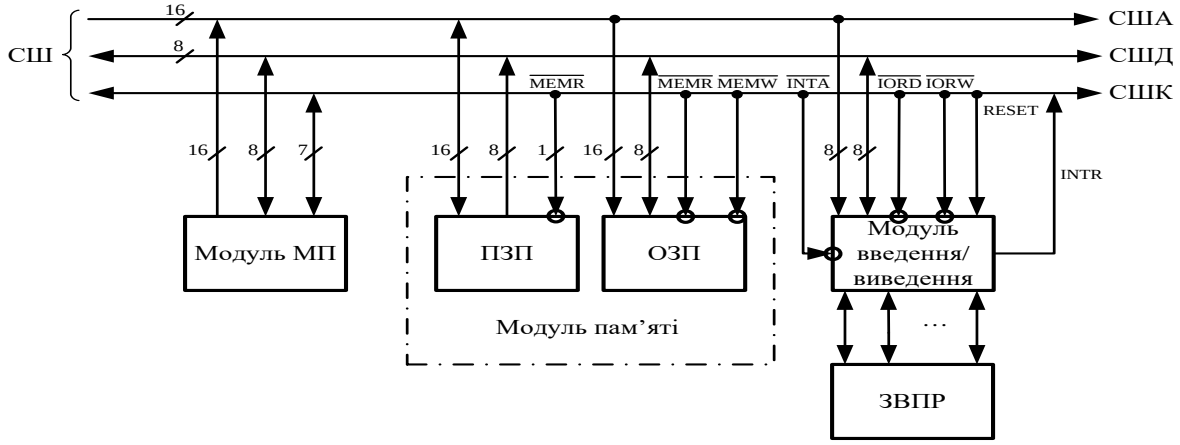


Рис. 1. Модульна структура МПС

До складу МПС входять *три основні модулі*: МП, пам'яті та модуль введення/виведення (МВВ/ВИВ). Окрім цього, до МПС можуть входити модулі: переривань, таймерів, АЦП, ЦАП, прямого доступу до пам'яті тощо.

1.2. Модуль мікропроцесора

На рис. 2, як приклад, наведено функціональну схему модуля МП на основі *шістнадцятирозрядного МП*, наприклад, МП і8086.

Цей модуль вирішує наступні *задачі*: розподіл (*демультиплексування*) шини адреси/даних (ШАД), *буферування* шини адреси і шини даних, а також *формування системних керувальних сигналів* для блоків пам'яті і зовнішніх пристроїв.

Перша задача вирішується, наприклад, за допомогою інтегральних мікросхем і8282, що виконують функції регістрів-защіпок. Зображені на рис. 2 два восьмибітових регістри і8282 запам'ятовують 15 молодших розрядів адреси (А0…А14). Це дозволяє адресувати $2^{15} = 32$ Кбайт комірок пам'яті. Оскільки сигнал $\overline{BHE}$ формується в тому ж інтервалі часу, що і адресні сигнали, то він також запам'ятовується у защіпці. Для доступу до пам'яті максимальної ємності 1 Мбайт необхідно під'єднати ще один регістр-защіпку, на який подаються старші розряди адреси МП, що залишилися: виходи А15, А16…А19.

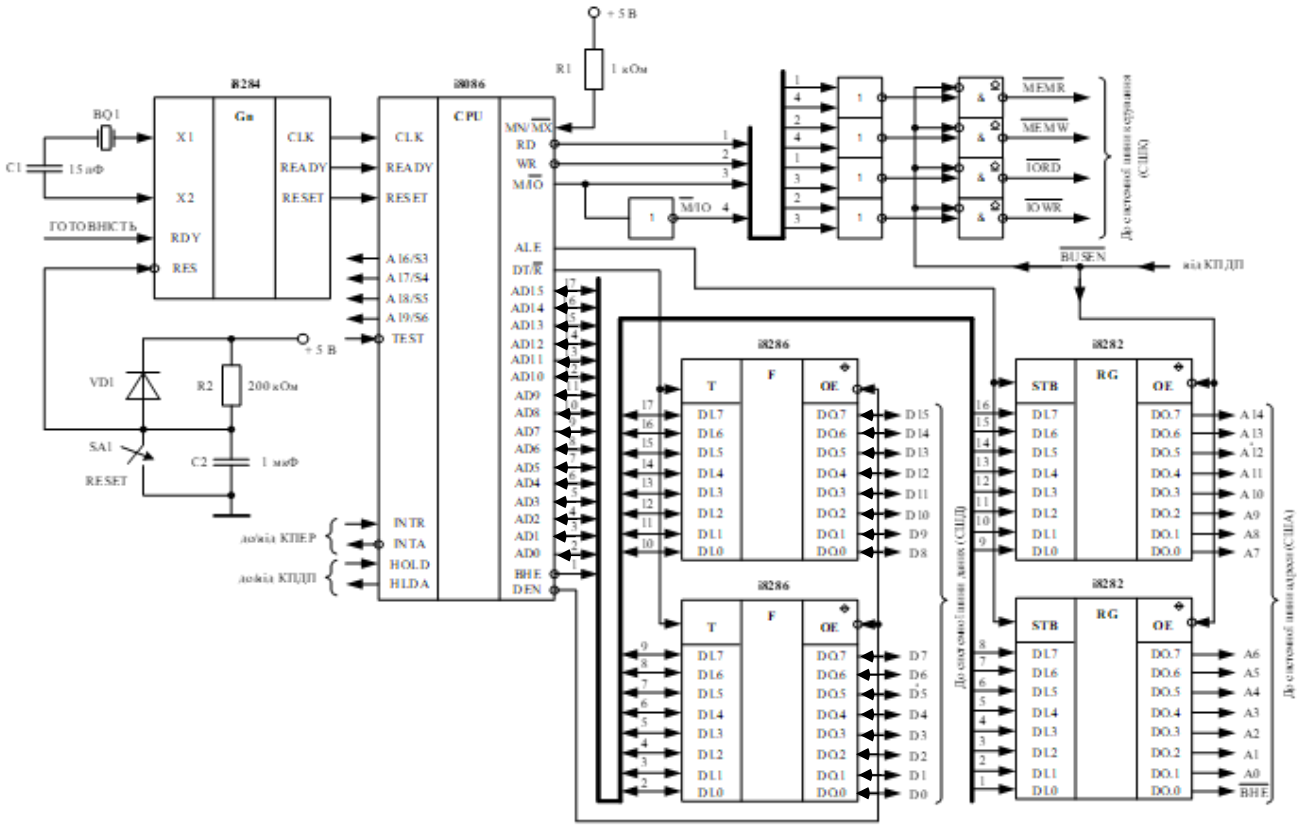


Рис. 2. Функціональна схема модуля мікропроцесора i8086

Друга задача вирішується за допомогою двоспрямованих восьмибітових шинних формувачів i8286, які підсилюють сигнали для передачі на системну шину даних.

Третя задача може бути вирішена, наприклад, за допомогою комбінаційних цифрових логічних схем, які на основі сигналів: $\overline{RD}$, $\overline{WR}$ та $M/\overline{IO}$, що виробляються мікропроцесором, формують необхідні керувальні сигнали: $\overline{MEMW}$, $\overline{MEMR}$, $\overline{IORD}$ та $\overline{IOWR}$. Названі сигнали керують запам'ятовувальними та зовнішніми пристроями подібно тому, як це робиться в системах, побудованих на основі МП i8080 [1].

Якщо в МПС введення/виведення організовано із відображенням на пам'ять, то сигнал $M/\overline{IO}$ не використовується і на ЗВПР подаються сигнали $\overline{RD}$ та $\overline{WR}$ після підсилення.

Шинні формувачі та регістри повинні забезпечувати *три стани* вихідних сигналів, щоб можна було організувати прямий доступ до пам'яті (ПДП), якщо він використовується в системі. Виходи регістрів та шинних формувачів *переходять у третій стан* за зняттям (встановленням у одиницю) сигналу $\overline{\text{ДОЗВІЛ ШИН}}$ ($\overline{BUSEN}$), що поступає *від контролера прямого доступу до пам'яті* (КПДП).

Після цього функцію формування керувальних сигналів виконує КПДП. Керувальні сигнали для СШК знімаються з *логічних елементів з відкритим колектором/стоком.* Це зроблено для того, щоб поєднати ці сигнали з аналогічними

сигналами, які формуються КПДП. У цьому разі кожен з аналогічних керувальних сигналів *від модуля МП і КПДП* поєднуються та за допомогою зовнішнього резистора підключаються до напруги живлення: +5В. Таке з'єднання зветься *монтажне «АБО» для нульових* вихідних сигналів, або *монтажне «І» для одиничних* вихідних сигналів [1]. Якщо захоплення шин і обмін даними за режимом ПДП не передбачено, то необхідності в такому поєднанні немає.

1.3. Модуль пам'яті

Модуль пам'яті призначено для збереження програм і даних. До його складу входять постійний запам'ятовуючий пристрій (ПЗП) та оперативний запам'ятовуючий пристрій (ОЗП). ПЗП призначений для збереження програм і даних, що беруть участь в операціях. Інформація до ПЗП заноситься на етапі виготовлення або перепрограмування системи. ПЗП – це енергонезалежна пам'ять. ОЗП призначений для збереження даних, що оброблюються, і для збереження програм, які зазвичай змінюються. Більш детально організація пам'яті розглядається у лекції 3.

1.4. Модуль введення/виведення

МВВ/ВИВ – забезпечує взаємозв'язок МП із зовнішніми пристроями (ЗВПР). До його складу *можуть в*ходити: ППІ і УАПП – універсальний асинхронний програмований приймач/передавач, які виконують відповідно паралельний та послідовний обмін інформацією між ЗВПР та МП. Додатково до складу МПС можуть входити також *наступні модулі*: контролер переривань (КПЕР), КПДП, таймер, АЦП, ЦАП і т. ін.

1.5. Контролер прямого доступу до пам'яті

Для реалізації *режиму ПДП* необхідно забезпечити безпосередній зв'язок КПДП і пам'яті МПС. Для цього можна було б використовувати спеціально виділені шини адреси, керування та даних, які пов'язують КПДП з пам'яттю. Проте це приведе до значного ускладнення системи, особливо у разі підключення декількох зовнішніх пристроїв. КПДП підключається до пам'яті за допомогою системної шини МПС. Сумісне використання системної шини МП та КПДП описано у підрозд. 1.2.

1.6. Контролер переривань

На технічні характеристики МПСК вагомо впливають засоби обміну інформацією між обчислювальним ядром і різноманітними периферійними пристроями. Суттєво *підвищити ефективність* роботи засобів підтримки обміну інформацією у мультимодульних системах можна, у випадку надання права ініціації обміну кожному модулю системи. Для цього обчислювальне ядро системи повинно мати можливість *приймати запити на переривання від інших модулів*, переривати

обробку поточного процесу і переходити на обслуговування запитів з обміну інформацією.

Засоби, які реалізують у МПС *обмін за перериваннями*, називаються *підсистемою переривань* і реалізуються у вигляді окремих ВІС (*контролери переривань* – КПЕР), *або* входять *у склад мікроконтролерів*. У МПС на основі мікропроцесорів і8080 та і8086, використовується, наприклад, спеціальна мікросхема і8259, за допомогою якої можна реалізувати багаторівневу систему переривань з можливістю врахування пріоритетів і маскування входів [1].

У мікроконтролерах запити на переривання можуть надходити ззовні, або від окремих периферійних модулів, які входять у склад МК.

1.7. Модуль таймера

Модуль таймера у МПС зазвичай *використовуєтьс*я для формування інтервалів часу та підрахунку зовнішніх подій.

У МПС існують *наступні способи* формування інтервалів часу (часових затримок, одиночних імпульсів, послідовностей імпульсів і т. ін.): апаратний; програмний та апаратно-програмний.

Перший спосіб (апаратний) реалізується використанням спеціалізованих електронних пристроїв, наприклад, ліній затримки, одновібраторів, мультивібраторів і т. ін. [5].

У разі *другого способу* (програмний), використовуючи систему команд конкретного МП чи МК і з огляду на те, що виконання кожної команди займає деякий час, можна розробити підпрограми, що формують часові затримки, одиночні імпульси, послідовності імпульсів і т. ін.

Апаратно-програмний спосіб реалізується застосуванням програмованих таймерів. Восьми- і шістнадцятирозрядні МП не містять вбудованих таймерів і використовують зовнішні мікросхеми, наприклад, і8253 [1].

Мікроконтролер може містити декілька внутрішніх восьми- та шістнадцятирозрядних програмованих таймерів [1; 2].

Окрім формування часових інтервалів, таймери *можуть виконувати* підрахунок імпульсів, що ідентифікують настання зовнішніх подій, тобто працювати як лічильники зовнішніх подій з апаратним чи програмним запуском. Таймери сучасних МК, окрім цього, *можуть працювати у режимах*: захоплення, порівняння та широтно-імпульсного модулятора (ШІМ) [1; 2].

1.8. Системна шина

Окремі модулі МПС об'єднані між собою внутрішньосистемним інтерфейсом і взаємодіють за адресним принципом – усі підлеглі пристрої та їх складові частини

мають адреси, що не повторюються, за якими до них звертаються пристрої, які виконують функції керування.

Внутрішньосистемний інтерфейс *реалізується* найчастіше на *основі єдиної системної шини (СШ),* якою передаються адреси, дані, команди та сигнали керування. У цьому разі для передачі даних і команд використовується *системна шина даних* (СШД). Адреси передаються окремою *системною шиною адреси* (США), яка керується мікропроцесором.

Системна шина керування (СШК) призначена для обміну керувальними сигналами між МП, пам'яттю або ЗВПР. Основні сигнали СШК: $\overline{MEMR}$ ($\overline{ЧТП}$) – читання з пам'яті; $\overline{MEMW}$ ($\overline{ЗПП}$) – запис у пам'ять; $\overline{IORD}$ ($\overline{ЧТВВ}$) – читання з пристрою введення/виведення; $\overline{IOWR}$ ($\overline{ЗПВВ}$) – запис у пристрій введення/виведення; *INT* – сигнал переривання до МП; $\overline{INTA}$ ($\overline{ППЕР}$) – підтвердження переривання із МП до КПЕР.

Процесор *обробляє інформацію трьох типів*: дані, адреси та команди програми. *Над даними* виконуються арифметичні та логічні операції, реалізовані процесором. *Обробка адреси* визначається способом збереження і доступу до даних і команд, та також ґрунтується на виконанні обчислювальних операцій. *Обробка команд* складається з *перетворення коду операції* команди (КОП) в послідовність керувальних впливів (*мікрооперацій*) відповідно до алгоритму виконання команди. Кожна мікрооперація (або їх сукупність – мікрокоманда) виконується у фіксовані інтервали часу (такти), а вся команда – за термін повного циклу (командного циклу) і утворює *мікропрограму*. Таким чином, *обробка ко*манд складається в їх представленні (інтерпретації) *у формі мікропрограм*. Під керуванням мікропрограми виконується обробка даних і адрес, а також керування іншими пристроями МПС через внутрішньосистемний інтерфейс.

1.9. Підключення МП до системної шини МПС

У разі використання, наприклад, МП i8080/i8086 використовується архітектура *з трьома системними шинами*: США, СШД і СШК (рис. 8).

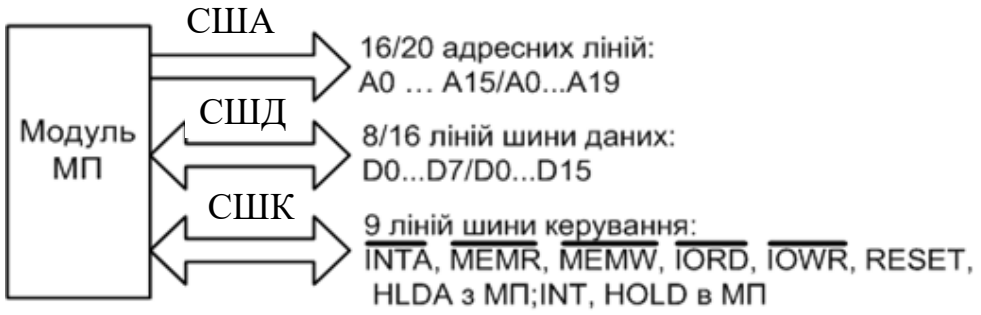


Рис. 8. Структурна схема системи з трьома шинами для МП i8080/i8086

США є *однонапрямною*, тому сигнали передаються лише від зовнішніх виводів МП. США в МПС пов'язана з безліччю адресних входів, що підключені паралельно, тому *сумарний струм для адресних ліній* МПС може за величиною перевершувати припустиме значення струму вихідних ліній МП. Отже, *для збільшення навантажувальної здатності* вихідних адресних ліній МП їх необхідно буферизувати. Для цього може використовуватися *формувач шини адреси* – мікросхема, виходи якої *мають три стани*, що дозволяє відключати США від МП у режимі прямого доступу до пам'яті.

У разі підключення СШД до МП необхідно враховувати *можливість перевантаження виводів МП* і те, що *СШД є двонапрямною*. Передача даних може вестися *в обох напрямках*. Однак у кожний момент часу дані передаються лише в одному напрямку. Тому СШД підключається до МП через *двонаправлений буфер* шини даних, функції якого може виконувати, наприклад, мікросхема і8286.

Системна шина керування є по суті *однонапрямною*, тому за одними лініями шини сигнали надходять у МП, а за другими від нього, тому, необхідності організації двостороннього обміну по одним і тим самим лініям немає. СШК підключається до МП через формувач керувальних сигналів (ФКС), який виробляє п'ять *основних* сигналів: $\overline{\text{INTA}}$, $\overline{\text{MEMR}}$, $\overline{\text{MEMW}}$, $\overline{\text{IORD}}$, $\overline{\text{IOWR}}$. ФКС може бути виконано на твердій логіці, або з використанням спеціалізованого *системного контролера* типу, наприклад, і8228, що виконує також роль шинного формувача.

2. СТРУКТУРНІ СХЕМИ ТИПОВОГО МП ТА МК

2.1. Структура восьмирозрядного мікропроцесора

Структурну схему типового восьмирозрядного МП, наприклад, і8080, зображено на рис. 9 [1]. Нижче наведено описання його основних вузлів.

Арифметико-логічний пристрій

Арифметико-логічний пристрій (АЛП) *є центральним вузлом* кожного МП і призначений для виконання арифметичних, логічних операцій, а також операцій зсуву. Вихідні дані для виконання операцій можуть передаватися *з акумулятора і регістрів загального призначення* або через буфер шини даних *з пам'яті*, причому *один з операндів міститься в акумуляторі*. Результат виконаної операції повертається в акумулятор. Вбудований у МП *блок десяткової корекції* дозволяє виконувати операцію додавання чисел у запакованому BCD-форматі.

Регістри

МП має *декілька регістрів*, які виконують наведені нижче функції.

Акумулятор (А) – регістр, який є найуніверсальнішим і таким, що часто використовується у командах. У ньому завжди *міститься один з опе*рандів, що

беруть участь в арифметичних, логічних операціях або операціях зсуву, а також результат операцій, що виконуються в АЛП.

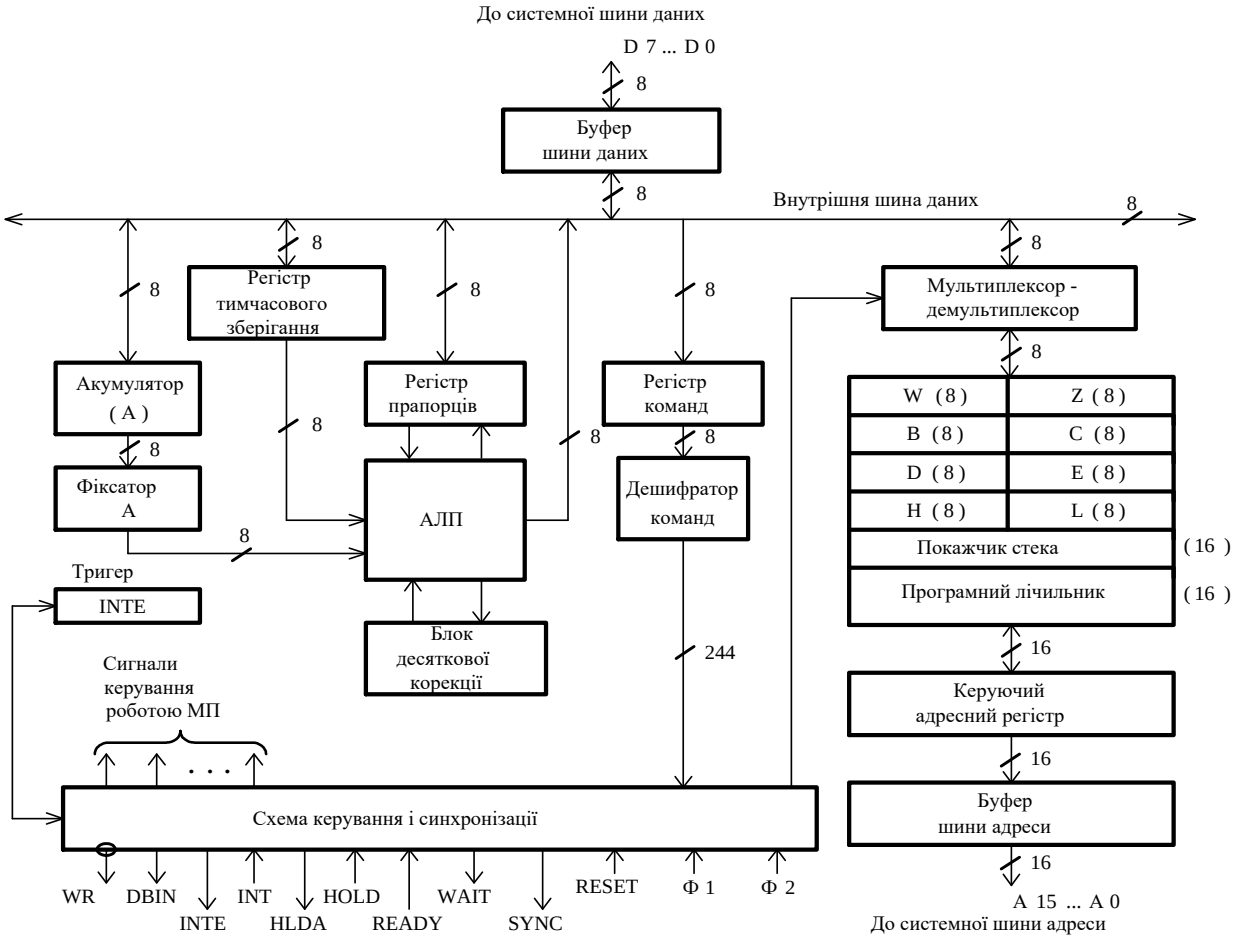


Рис. 9. Структурна схема МП i8080

Обмін інформацією із зовнішніми пристроями в цьому МП виконується *тільки через акумулятор*, що *є його певним недоліком.*

Регістри загального призначення (РЗП) – призначені для забезпечення швидкого доступу до даних (операндів), що в них зберігаються, оскільки знаходяться всередині МП і, по суті, є швидкодіючим *надоперативним запам'ятовувальним пристроєм* (НОЗП). Вони позначаються буквами B, C, D, E, H, L і, як видно з рис. 9, об'єднані попарно BC, DE, HL, що дозволяє обробляти операнди завдовжки як 8 біт, так і 16 біт. Остання можливість використовується, якщо необхідно *організувати збереження адреси* пам'яті або виконати *обчислення над двобайтовими операндами.*

Програмний лічильник (ПЛ) – шістнадцятирозрядний регістр, призначений для *збереження адреси поточної* виконуваної програмою команди.

Регістр команд (РК) – регістр, призначений *для зберігання КОП* поточної виконуваної команди. КОП завжди міститься у *першому байті* машинного коду команди. *Програмно не доступний*.

Регістр стану (прапорців) – регістр, що містить прапорці – *тригери*, значення яких залежить від результату виконання деяких команд в АЛП. У наведеному МП використовується *п'ять* прапорців, а інші три біти восьмибітного регістра прапорців не використовуються. Формат регістра стану наведено на рис. 10.

$7_р$	6	5	4	3	2	1	$0_р$
S	Z	0	AC	0	P	1	C

Рис. 10. Формат регістра стану (прапорців) МП i8080

Нижче наведено *опис прапорців*:

– S – прапорець *знака*, встановлюється в одиницю, якщо результат виконання команди має від'ємне значення. Прапорець дорівнює *старшому розряду коду* результату;

– Z – прапорець *нуля*, який встановлюється *в одиницю*, якщо після виконання операції усі розряди акумулятора *мають нульове* значення;

– C – прапорець *перенесення/запозичення*, встановлюється в одиницю, якщо у разі виконання операції додавання відбулося перенесення з 7-го розряду в уявний 8-ий або у разі віднімання відбулася позика одиниці з уявного 8-го розряду;

– AC – прапорець *допоміжного перенесення*, встановлюється в одиницю, якщо під час виконання команди відбулося перенесення з 3-го в 4-й розряд або позика з 4-го в 3-й розряд. Цей прапорець використовується *спеціальною командою* під час корекції результату додавання чисел у запакованому BCD-форматі;

– P – прапорець *парності*, встановлюється *в одиницю*, якщо результат виконання операції *містить парну* кількість одиниць.

Покажчик стека (ПС) – шістнадцятирозрядний регістр, який призначений для адресації комірок стекової пам'яті. Для цього мікропроцесора у ньому зберігається адреса *верхівки стека*, останньої комірки стекової пам'яті, куди записана корисна інформація. Запис чергового байта в стек ведеться з декрементом регістра-покажчика стека.

Регістри тимчасового зберігання: фіксатор A, регістр тимчасового зберігання та регістри W і Z (рис. 9) є *програмно недоступні* та використовуються для проміжного збереження операндів і адрес під час вибірки команд з пам'яті та їх виконання.

Буферні регістри: буфер шини даних (двонапрямний) та буфер шини адреси (однонапрямний) використовуються *для узгодження МП з* СШ, програмно недоступні.

Керувальний адресний регістр складається з: регістра адреси пам'яті (РАП) та схеми інкремента-декремента адреси (ІДА). РАП – шістнадцятирозрядний, який приймає і зберігає адресу від кожного шістнадцятирозрядного регістра. Вихід РАП

через схему ІДА зв'язано з буфером шини адреси та зі входом кожного шістнадцятирозрядного регістра, який містить адресу. *Схема ІДА* пов'язана з виходом РАП та призначена для зміни вмісту регістра адреси пам'яті на ± (для формування поточних адрес пам'яті програм і стека). В процесі вибірки КОП команди з пам'яті вміст регістрів ПЛ і РАП збігається. *Після декодування КОП поточної* команди у програмний лічильник *записується адреса КОП наступної команди*, а вміст РАП змінюється згідно з виконуваною командою.

Блок десяткової корекції призначений для корекції результату після додавання чисел у запакованому BCD-форматі. Для корекції використовується *команда DAA.*

Мультиплексор забезпечує *введення* інформації з декількох паралельних каналів в один послідовний під час передачі даних, наприклад, з блоку РЗП на внутрішню шину даних.

Демультиплексор служить *для прийому* інформації з одного послідовного каналу, наприклад з внутрішньої шини даних, і виведення її на один з паралельних каналів, наприклад в один з РЗП.

Дешифратор команд призначений *для декодування КОП команди*, який знаходиться у першому байті машинного коду команди. За результатом декодування *виробляється потрібна послідовність сигналів керування*, що призводить до зчитування наступних байтів команди для її виконання.

Схема керування і синхронізації є однією з найважливіших вузлів МП, що забезпечує правильну послідовність подій у МП. Після зчитування команди з пам'яті та її декодування пристрій керування *генерує послідовність сигналів, необхідних для виконання команди.* Робота схеми *програмується* під час виготовлення МП. Схема керування містить *маленький МП всередині мікропроцесора*. Ще однієї з основних функцій схеми керування є *організація зв'язку* МП з зовнішніми пристроями та системною шиною.

Часові співвідношення в МП та МК

Час, який необхідний для зчитування команди з пам'яті та її виконання, називається *командним циклом* (КЦ). КЦ для МП типу i8080 складається *з декількох машинних циклів* (МЦ), точна кількість яких залежить від складності виконуваної команди і дорівнює *кількості звернень МП до пам'яті або одного з ПВВ/ВИВ* (рис. 11).

Машинний цикл складається з деякої кількості елементарних дій, що називаються станами *(тактами*), і виконуються за один період системного тактувого сигналу.

Тому для визначення повного часу виконання команди потрібно знати, яка кількість тактів міститься в КЦ і чому дорівнює період сигналу тактування, а потім їх перемножити. Часові співвідношення у типовому МП та МК описано у [1; 2].

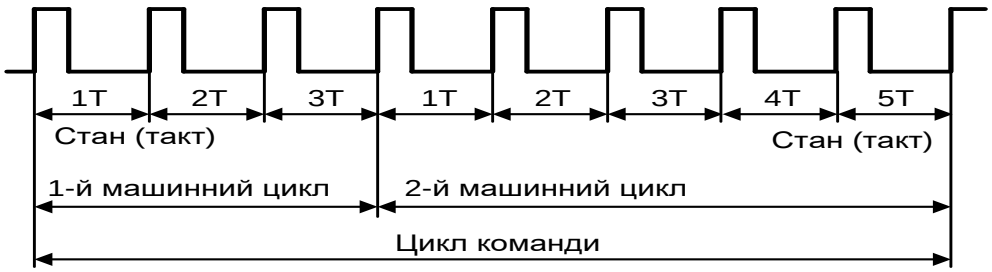


Рис. 11. Приклад часової структури командного циклу

2.2. Структура шістнадцятирозрядного мікропроцесора

В якості прикладу шістнадцятирозрядного МП в роботі розглянуто *мікросхему* i8086. Під час розробки цього МП було застосовано нові у порівнянні з МП i8080 архітектурні рішення, до яких, зокрема, належить *розділення функцій сполучення з шиною та виконання команд.* Відповідно до цього структуру МП i8086 можна умовно розділити на *дві частини*: виконавчий блок (ВБ) і блок сполучення з шиною (БСШ) (рис. 12) [1].

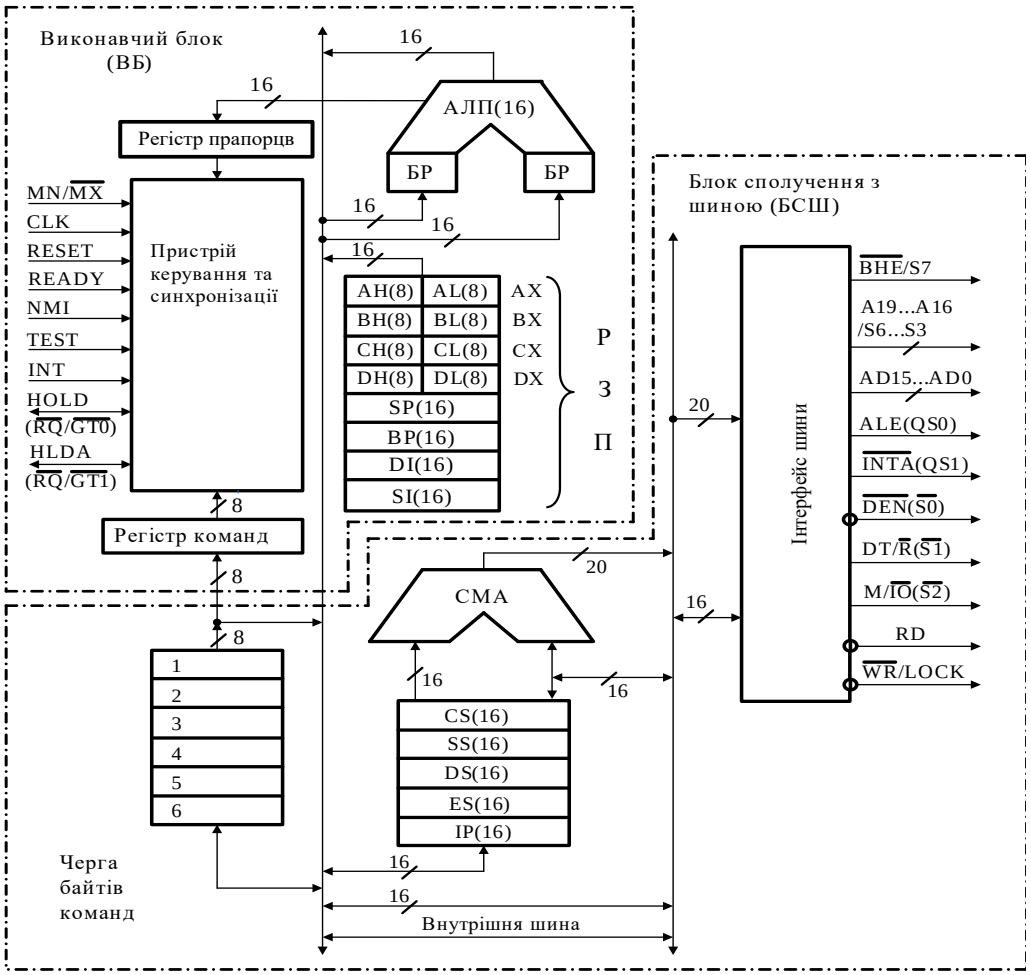


Рис. 12. Структура МП i8086

Виконавчий блок призначений для зберігання даних, що обробляються, і виконання операцій над ними. Він включає вісім шістнадцятирозрядних регістрів загального призначення (РЗП), шістнадцятирозрядний АЛП, регістр прапорців (ознак) – RF, регістр команд та пристрій керування і синхронізації.

РЗП належать до *надоперативних запам'ятовувальних пристроїв* (НОЗП). Вони повністю доступні програмісту, можуть використовуватися довільно, але в багатьох командах вони мають специфічне призначення. РЗП поділяються на дві групи.

Перша група РЗП: *AX, BX, CX, DX* в арифметичних і логічних командах може використовуватися для зберігання операндів і результатів виконання операцій. Переважно для цієї мети застосовується регістр *AX – акумулятор*. У регістрі BX – *регістрі бази*, як правило, зберігається початкова адреса структури даних, що обробляється, яка розташована у сегменті даних *DS*. *Регістр CX* у ряді команд використовується як лічильник під час організації циклічних обчислень. *Регістр даних DX* у командах множення і ділення зберігає половину 32-розрядних операндів, а в командах введення/виведення може містити адресу зовнішніх пристроїв.

В процесорі передбачена *можливість роботи з восьмирозрядними* «половинками» названих регістрів. *Регістр AX* розпадається на два: регістр молодшого байта акумулятора *AL* і регістр старшого байта акумулятора *AH*. Аналогічно *регістр BX* – розпадається на два регістри *BL* і *BH*, регістр *CX* – на два регістри *CL* і *CH*, а *регістр DX* – на регістри *DL* і *DH*.

Друга група РЗП складається з двох шістнадцятирозрядних регістрів-покажчиків: *SP* і *BP* та двох шістнадцятирозрядних індексних регістрів: *SI* і *DI*. *Регістри-покажчики* використовуються для роботи зі стековою пам'яттю (стековим сегментом пам'яті): *покажчик стека SP* містить адресу верхівки стека, а *покажчик бази BP* зберігає адресу початкового елемента структури даних у стековому сегменті SS. *Індексні регістри* використовуються для доступу до елементів рядків даних. *Під час читання* елемента рядка-джерела регістр індексу *SI* зберігає *зсув адреси в сегменті даних DS*, а *у разі запису* елемента рядка-приймача регістр індексу *DI* містить *зсув адреси в екстра-сегменті ES*.

Арифметико-логічний пристрій (АЛП) призначений для виконання арифметичних, логічних операцій та операцій зсуву над восьми та шістнадцятирозрядними операндами – даними, що беруть участь в операціях. *Операнди надходять* в АЛП з регістрів загального призначення і/або пам'яті.

Результат операції повертається в регістр загального призначення чи в пам'ять. АЛП окрім схем для виконання операцій містить *два програмно недоступних* буферних регістри (БР) для тимчасового збереження операндів.

Регістр прапорців (рис. 13) містить 9 *ознак (прапорців),* що *характеризують стан програми*, виконуваної МП, та керують його роботою.

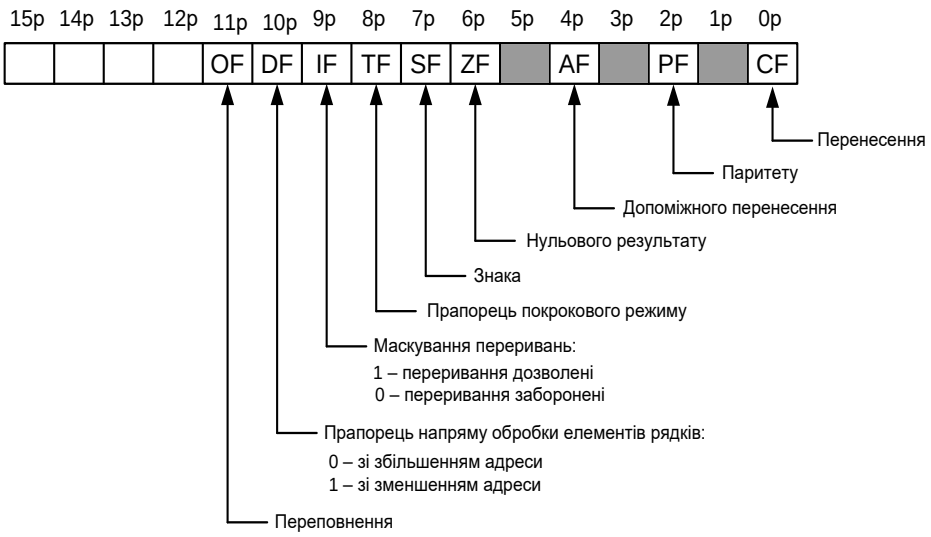


Рис. 13. Формат регістра прапорців МП і8086

Шість прапорців встановлюються відповідно до результату виконаної в АЛП чергової операції. Якщо розрядність оброблюваних кодів позначити *n*, а нумерацію розрядів здійснити справа наліво від 0 до (*n* – 1), то:

- *прапорець перенесення CF* встановлюється в одиницю, якщо в результаті виконання операції виникло перенесення з (*n*–1)-го розряду результату в уявний більш старший *n*-й розряд або була потрібна позика з уявного *n*-го розряду;
- *прапорець паритету PF доповнює* код результату *до непарного* числа одиниць, тому встановлюється в одиницю, якщо кількість одиниць у коді результату парна;
- *прапорець допоміжного перенесення AF* встановлюється в одиницю, якщо під час виконання операції виникло перенесення з 3-го розряду результату в 4-й або виконалася позика з 4-го розряду в 3-й. Прапорець AF зазвичай використовується для виконання команд десяткової корекції результату під час виконання операцій над BCD-числами;
- *прапорець нульового результату ZF* приймає одиничне значення, якщо всі розряди результату нульові. Якщо хоча б один розряд результату відмінний від нуля, то прапорець *ZF* скидається в нуль;
- *прапорець знаку SF* встановлюється рівним старшому (*n*–1)-му розряду коду результату. Під час виконання операцій над числами зі знаком прапорець *SF* відповідає знаку результату. Якщо результат більший за нуль або дорівнює нулю (додатний), то *SF* скидається в нуль, якщо результат менший за нуль (від'ємний), то *SF* буде встановлено в одиницю;
- *прапорець переповнення OF* встановлюється в одиницю, якщо в результаті виконання арифметичних операцій *над числами зі знаком* відбувається *переповнення розрядної сітки*, тому результат *не укладається в діапазон, виділений n-розрядному числу зі знаком* (перенесення з (*n*–2)-го розряду в

(*n*–1)-й не збігається з перенесенням з (*n*–1)-го розряду в уявний *n*-й. Наприклад, якщо *n* = 8, то *діапазон чисел зі знаком*: –128…+127, якщо *n* = 16, то *діапазон чисел зі знаком*: –32768…+32767.

Три прапорці, що залишилися, характеризують *стан пристрою керування МП:*

- *прапорець напряму передачі DF* визначає спосіб зміни адрес джерел/приймачів операндів в *командах роботи з рядками даних*. Якщо *DF* = 0, то адреси збільшуються (автоінкремент адреси), якщо *DF* = 1, то адреси зменшуються (декремент адреси);
- *прапорець дозволу переривання IF*, що скидається *в нуль*, *забороняє (маскує*) обробку зовнішнього переривання, а встановлений *в одиницю, дозволяє* його;
- *прапорець покрокового режиму TF*, що знаходиться в одиничному стані, задає спеціальний режим виконання програми, в якому *після обробки кожної команди генерується програмне переривання.*

В регістр команд вміщується КОП команди (8 біт), що підлягає виконанню (*програмно недоступний*). КОП деяких команд *має довжину більше, ніж 8 біт*. Ці *додаткові біти* зберігаються у другому байті машинного коду команди (*постбайті*). Вісім біт КОП *декодуються дешифратором*, що дозволяє разом з додатковими бітами *ідентифікувати* тип виконуваної команди.

Пристрій керування і синхронізації забезпечує функціонування окремих вузлів і МП в цілому. Він аналізує вхідні і генерує вихідні керувальні сигнали, керує обміном інформацією в самому МП та виконанням команд. *Після декодування* КОП чергової виконуваної команди пристрій керування формує послідовність керувальних сигналів, що забезпечують виконання цієї команди.

Блок сполучення з шиною забезпечує обмін інформацією між МП і зовнішніми відносно нього приймачами або джерелами даних (пам’ять, ПВВ/ВИВ і т. ін.). До складу *БСШ* входять: група сегментних регістрів, регістр адреси команди, черга байтів команд, суматор адреси та інтерфейс шини.

Сегментні регістри призначено для збереження початкових (базових) *логічних* адрес сегментів пам’яті.

Процесор містить *чотири* шістнадцятирозрядних *сегментних* регістри, що відповідає *чотирьом поточним* сегментам пам’яті:

- *регістр сегмента команд CS:* визначає сегмент пам’яті, в якому зберігається виконувана програма (містить логічну адресу початку поточного сегмента коду);
- *регістр сегмента даних DS:* задає сегмент пам’яті, що містить оброблювані дані (містить логічну адресу початку поточного сегмента даних);

– *регістр сегмента стека SS:* визначає сегмент пам’яті, доступ до елементів якого можливий за допомогою стекової адресації (містить логічну адресу початку поточного сегмента стека);
– *регістр екстра-сегмента ES:* задає додатковий сегмент, в якому також можуть зберігатися оброблювані дані (містить логічну адресу початку поточного додаткового сегмента даних).

Регістр адреси команд IP (покажчик команд) зберігає шістнадцятирозрядну адресу (*зміщення*) чергової команди, що повинна витягатися із *сегмента команд*, тому *є зсувом відносно початку поточного сегмента команд*.

Шість восьмирозрядних регістрів черги команд складають внутрішню проміжну пам’ять програми зі швидким доступом, яку призначено для збереження послідовності команд, що виконуються.

Суматор адреси (СМА) призначено для формування *20-розрядної фізичної* адреси зовнішньої пам’яті. Для цього використовуються *дві шістнадцятирозрядні логічні* адреси: початку сегмента та зміщення в сегменті (зсуву відносно початку сегмента).

Інтерфейс шини забезпечує обмін інформацією із системною шиною і формує керувальні сигнали.

У МП і8086 передбачена *асинхронна паралельна робота* ВБ та БСШ. В процесі виконання програми БСШ вибирає з пам’яті послідовність команд і розміщує їх у черзі байтів команд (до шести байт). Після завершення виконання попередньої операції у ВБ чергова команда вибирається з молодших регістрів черги і направляється на обробку в ВБ.

Виконавчий блок виконує запропоновану операцію, а БСШ завантажує чергу новою командою чи командами, залежно від кількості регістрів черги, що звільнилися. Якщо ВБ потрібно прочитати дані з пам’яті або записати їх у пам’ять, то процес вибірки команд переривається і БСШ виконує цикл читання чи запису даних. Після цього відновлюється вибірка команд, що продовжується до заповнення черги.

Таке *перекриття етапів вибірки і виконання команд* значно зменшує час читання команд із сумарного часу виконання програми, підвищуючи тим самим продуктивність МП. Якщо ВБ виконує команду, що передає керування іншій комірці пам’яті команд (наприклад, команди CALL, JMP), то стара черга команд стирається та БСШ знову встановлює чергу команд, починаючи заповнювати її з нового значення.

У більшості випадків *ВБ може відразу* одержувати команду з черги, а не чекати, доки вона буде вибрана з пам’яті.

2.3. Структура восьмирозрядного мікроконтролера

2.3.1. Загальна характеристика восьмирозрядного мікроконтролера

Одним з типових восьмирозрядних МК є AT89C51, який належить до сімейства МК-51 (MCS-51). Ці МК були створені на базі «Інтелоподібних МП». Тому вони мали відповідну архітектуру та *значний час знаходили широке застосування* у МПС. Але *одним з недоліків* цих МК є *наявність регістра-акумулятора*, який обов'язково приймав участь під час виконання частини команд. Це *частково зменшувало* час виконання робочої програми.

Цей недолік було усунуто в AVR-мікроконтролерах, які сьогодні знаходять досить широке застосування серед восьмирозрядних МК.

«Традиційні» AVR-мікроконтролери, що починали свій розвиток з серії Classic (позначення AT90S), потім розділилися *на три сімейства*: Tiny (позначення ATtiny), Mega (позначення ATmega) і XMega (позначення ATXMega).

Розробники AVR-мікроконтролерів поклопоталися про інженерів, що застосовують дану платформу, зробивши МК різних серій *повивідно сумісними* [2]. Таке рішення забезпечує свободу вибору кристала і можливість подальшого переходу між різними серіями. Слід зазначити, що для AVR-мікроконтролерів також характерна *програмна сумісність «знизу вверх»,* що забезпечується збереженням найменувань регістрів спеціального призначення в різних серіях і практично єдиним набором команд.

На додаток відзначимо, що платформа AVR, як і раніше, продовжує привертати увагу розробників збалансованим за функціональністю і вартістю набором МК у поєднанні з доступністю програмних і апаратних засобів підтримки розробок для них.

Сьогодні широко використовуються МК *сімейства Mega* [2], які є 8-бітними МК, призначеними для використання у вбудованих системах [4]. Вони виготовляються за малоспоживаючою *КМОН-технологією*, яка у поєднанні з удосконаленою RISC-архітектурою дозволяє досягти найкращого співвідношення вартість/швидкодія/енергоспоживання.

Сьогодні AVR-мікроконтролери широко використовуються у складі апаратної обчислювальної платформи «Arduino» (Ардуіно), основними компонентами якої є *плата мікроконтролера*, яка має можливість зв'язку зі значною кількість сучасних периферійних пристроїв для використання у вбудованих системах (англ. embedded system).

2.3.2. Структура ядра AVR-мікроконтролерів

Ядро AVR-мікроконтролерів виконано за вдосконаленою RISC-архітектурою (англ. enhanced RISC) (рис. 14), в якій використовується ряд рішень, спрямованих на

підвищення швидкодії МК. *Арифметико-логічний пристрій*, що виконує всі обчислення, *підключений безпосередньо до 32 робочих регістрів*, що об'єднані 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*.

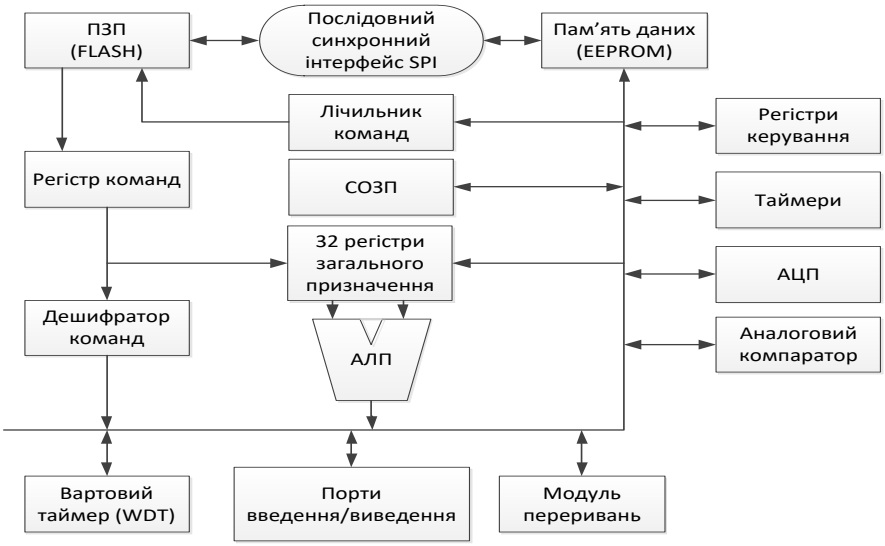


Рис. 14. Структура ядра AVR-мікроконтролерів

Крім того, *практично кожна з команд* (за винятком команд, в які входить 16-бітна адреса операнда) *займає одну комірку пам'яті програм*.

У [2] наведені структури деяких AVR-мікроконтролерів. *Наприклад*, до моделей ATmega640x/1280x/2560x входять:

- 11 портів введення/виведення (порти A...F, H, J...L – 8-бітні, порт G – 6-бітний);
- два 8-бітних (T0, T2) та чотири 16-бітних (T1, T3, T4, T5) таймери;
- 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.

Контрольні запитання та завдання

1. Назвіть та опишіть три основні модулі, які має мікропроцесорна система.
2. Наведіть та опишіть функціональну схему модуля МП на основі шістнадцятирозрядного МП, наприклад, МП i8086.
3. Як організовано режим ПДП в модулі МП i8086?
4. Назвіть види пам'яті, які має модуль пам'яті МПС, та опишіть їх призначення.
5. Опишіть склад та призначення модуля введення/виведення МПС.

6. Як контролер ПДП підключається до пам’яті?

7. Назвіть призначення підсистеми переривань та опишіть як її реалізують у МПС?

8. Для чого використовують модуль таймера у МПС?

9. Назвіть призначення та опишіть склад системної шини МПС.

10. Наведіть та опишіть структурну схему МПС з трьома системними шинами для МП i8080/i8086.

11. Наведіть та опишіть структурну схему типового 8-розрядного МП.

12. Назвіть та опишіть призначення регістра прапорців МП i8080.

13. Наведіть та опишіть структурну схему типового 16-розрядного МП.

14. Наведіть та опишіть формат регістра прапорців МП i8086.

15. Які функції виконує регістр команд МП i8080/i8086 та чим він відрізняється від лічильника команд?

16. Для чого призначено сегментні регістри у МП i8086?

17. Як формується 20-розрядна фізична адреси зовнішньої пам’яті у МПС на основі шістнадцятирозрядного i8086.

18. Дайте визначення командного циклу у МП i8080.

19. Що потрібно знати для визначення повного часу виконання команди?

20. Що являє собою арифметико-логічний пристрій у МП та МК?

21. Для чого служить лічильник команд у МП та МК?

22. Який основний недолік мають мікроконтролери сімейства МК-51 (MCS-51) та як його було ліквідовано у МК сімейства AVR.

23. Наведіть та опишіть структуру ядра МК сімейства AVR.

24. За якою технологією виготовляються МК сімейства AVR та які переваги має ця технологія?

25. На які сімейства останнім часом розділилися AVR-МК?

ЛЕКЦІЯ 3. ОСОБЛИВОСТІ АРХІТЕКТУРИ ПАМ'ЯТІ МПС. ОРГАНІЗАЦІЯ ПАМ'ЯТІ МПС НА ОСНОВІ МІКРОПРОЦЕСОРА

1. ОСОБЛИВОСТІ АРХІТЕКТУРИ ПАМ'ЯТІ МПС

1.1. Призначення та місце модуля пам'яті в мікропроцесорній системі

Під час вивчення модульної структури МПС відзначалося, що *одним з основних її модулів є модуль пам'яті* (лекція 2).

В пам'яті, яку також називають *запам'ятовувальним пристроєм* (ЗП), зберігаються програми і дані, що використовуються для керування роботою МПС [1; 2]. Пам'ять *поділяється* на: ОЗП та ПЗП. *Керувальні програми* повинні розміщуватися в енергонезалежному ПЗП, щоб зберігатися у разі вимикання живлення, тому його іноді називають *пам'яттю програм*. У ОЗП *зберігаються дані*, що беруть участь у виконанні операцій, тому його називають *пам'яттю даних*. Це робиться дуже умовно, оскільки в процесі роботи системи (великої, складної) в ОЗП із зовнішнього носія інформації може завантажуватися керувальна програма. Під час роботи системи *ПЗП* допускає головним чином *читання* записаної в нього інформації, а *в ОЗП* можна *спочатку записати* поточні дані, а *потім їх прочитати*.

В сучасних МК *є можливість змінювати вміст* ПЗП під час роботи системи. Для цього використовується *додатковий модуль та команда самопрограмування* [2].

1.2. Класифікація пам'яті

1.2.1. Основна та зовнішня пам'ять

Пам'ять МПС зазвичай *розділяють* на основну і зовнішню.

Кожен МП/МК може адресувати деяке число комірок пам'яті, що утворюють *єдиний адресний простір МПС*, реалізованої на основі даного МП/МК. Ці комірки утворюють *основну (внутрішню)* пам'ять МПС. Будь-яку комірку цієї пам'яті можна адресувати в конкретній системі для читання або для запису інформації. Наприклад, якщо США має кількість ліній $n = 16$, то *основна пам'ять* МПС може мати до $2^{16} = 65\,536$ комірок. Основна пам'ять *МПС на МК поділяється стосовно мікроконтролера* на: *внутрішню* пам'ять програм/даних (ВПП/ВПД) і *зовнішню* пам'ять програм/даних (ЗПП/ЗПД).

Зовнішню пам'ять *мікропроцесорних систем утворюють* пристрої тривалого збереження інформації, наприклад, диски, магнітні стрічки, дискети тощо.

В даному підрозділі *розглядається основна пам'ять*, яка виконується переважно на напівпровідникових ВІС.

1.2.2. Пам’ять з довільним та послідовним доступом

Залежно *від способу звернення* до окремих комірок пам’яті розрізняють: пам’ять з довільним доступом та пам’ять з послідовним доступом.

У разі *довільного* доступу (вибірки) допускається *будь-який порядок надходження адрес*, що лежать в адресному просторі даного процесора. В разі адресації комірок пам’яті *другого типу* зміна адрес можлива *тільки у визначеному порядку* – збільшення чи зменшення адрес. Прикладом такого доступу є запис і читання інформації на/з диску. *Основна* пам’ять МПС є пам’яттю *із довільним* доступом.

1.2.3. Енергозалежна та енергонезалежна пам’ять

Залежно від того, як впливає *наявність джерела живлення* на збереження інформації, пам’ять поділяється на: енергозалежну та енергонезалежну.

В енергозалежній пам’яті *дані у разі відсутності живлення руйнуються*, а в *енергонезалежній – ні. Прикладом енергозалежної* пам’яті є статичний оперативний запам’ятовувальний пристрій (*СОЗП*) – SRAM, а прикладом *енергонезалежної* – FLASH-пам’ять програм та EEPROM-пам’ять даних.

1.2.4. Статична та динамічна пам’ять

Основна пам’ять даних (оперативний запам’ятовуючий пристрій (ОЗП)) МПС будується здебільшого на МОН (КМОН)-транзисторах і поділяється на *статичну і динамічну*. ОЗП динамічного типу зазвичай виконуються *на МОН-транзисторах*. Особливістю цих транзисторів є дуже високий вхідний опір затвору транзистора, що дозволяє використовувати як елемент пам’яті конденсатор, функції якого виконує вхідна паразитна ємність МОН-транзистора. В результаті виходить так звана динамічна пам’ять. Динамічну пам’ять використовують у ПК для зберігання досить великих обсягів інформації [1].

У *статичних ОЗП* кожен елемент пам’яті, об’ємом 1 біт, являє собою симетричний *тригер*, складений із двох транзисторних ключів на МОН-транзисторах з перехресними зв’язками [1; 5].

Для *зберігання да*них в МПС здебільшого використовуються *статичні ОЗ*П та енергонезалежна *EEPROM*-пам’ять даних, а для зберігання програм – FLASH-пам’ять.

1.2.5. Основні характеристики пам’яті

Основними характеристиками пам’яті є: об’єм; розрядність; швидкодія; споживана потужність; габарити; вартість; технологія виготовлення та програмування. *Об’єм* пам’яті вимірюється кількістю комірок, що вона містить. *Розрядність* відображає кількість біт інформації, що містяться в одній комірці

пам’яті. З метою *зменшення споживаної потужності* від джерел живлення, зменшення габаритів та вартості, сучасні ВІС пам’яті виконуються за МОН (КМОН)-технологією. Основним параметром, що характеризує *швидкодію* пам’яті, є час доступу до пам’яті. *Розрізняють* час доступу у разі читання та запису. Час, необхідний для виведення інформації з пам’яті на шину даних після одержання адреси потрібної комірки, називається *часом доступу у разі читання*. *Час доступу у разі запису* – час необхідний для запису даних в область, що адресується.

1.2.6. Призначення та організація стека

Стеком називається *спеціальна область оперативної пам’яті*, що головним чином створюється і використовується під час організації та виконання підпрограм. Стек у МПС можна *умовно поділити* на: апаратний та програмний. Наявність виду стека залежить від типу МП або МК. В МПС, в яких стек реалізовано апаратно, відсутні команди запису в стек або читання зі стека (PUSH/POP), і стек використовується тільки для зберігання адрес повернення із підпрограм. Більшість МПС мають можливість роботи зі стеком командами PUSH/POP. Для адресації комірок стека МП/МК містять спеціальні *регістри-покажчики стека – SP*. До виконання команди роботи зі стеком SP адресує «верхівку» стека – останню комірку стекової пам’яті, в яку записано інформацію, або першу вільну комірку пам’яті (залежить від типу МП/МК).

Для запису даних у стек є команда з мнемонікою «PUSH», а для читання – «POP». Механізм виконання цих команд в «інтелоподібних» МП дуже схожий. *По-перше*, стек заповнюється в бік менших адрес. *По-друге*, обмін зі стеком відбувається словами (2 байти). У разі запису у стек спочатку записується старший байт слова, а потім – молодший, а у разі читання – навпаки. *По-третє*, виконується правило обміну зі стеком – першим увійшов (був записаний) в стек – останнім вийшов (був прочитаний) зі стека, тобто *стек типу FILO* («first in last out»).

У восьмирозрядних МП для стека виділяється спеціальна частина загального адресного простору пам’яті способом «карти пам’яті». *У 16-розрядних* МП для стека використовується окремий сегмент, максимальний об’єм якого складає 64 Кбайт.

Під час виконання команд PUSH і POP в мікроконтролерах, наприклад, *типу* AVR стек заповнюється в сторону менших адрес та обмін відбувається байтами. Принцип «першим увійшов, останнім вийшов» в МК зберігається.

1.2.7. Режим прямого доступу до пам’яті

Один зі способів обміну даними в МПС є *режим ПДП*, коли обмін даними між зовнішнім пристроєм (ЗВПР) і пам’яттю МПС відбувається без участі МП. Обміном в режимі ПДП *керують зовнішні* стосовно МП електронні схеми, а не програма, що виконується МП [1]. Ці схеми розташовуються в спеціальному пристрої, що

називається контролером ПДП (КПДП). КПДП підключається до пам'яті за допомогою системних шин МПС. Режим ПДП забезпечує необхідну швидкість обміну, під час якого час на обмін одним байтом обмежується швидкодією пам'яті.

Від КПДП на вхід «HOLD» МП передається керувальний сигнал «*Вимога прямого доступу до пам'яті (ВПДП)*». Процесор, отримавши сигнал «ВПДП», припиняє виконання програми, видає з виходу «HLDA» на СШК керувальний сигнал контролеру «*Надання прямого доступу до пам'яті (НПДП)*» і відключається від США, СШД і СШК. Для відключення МП від системної шини використовується можливість переведення його виходів, які під'єднано до цих шин, у *третій (високоімпедансний) стан.* Керувальні сигнали від КПДП до СШК підключаються за допомогою *елементів з відкритим колектором (стоком)* [5]. З цього моменту системна шина МПС передається у розпорядження КПДП. Останній *обмінюється блоками даних* з пам'яттю, а потім, знявши сигнал «ВПДП», повертає керування системною шиною мікропроцесору.

2. ОРГАНІЗАЦІЯ ПАМ'ЯТІ МПС НА ОСНОВІ МІКРОПРОЦЕСОРА

2.1. Організація пам'яті на основі 16-розрядного МП

2.1.1. Фізична та логічна організація пам'яті

Під *фізичною* розуміється організація пам'яті *на апаратному* рівні, а під *логічною – на програмному*. У МПС на основі, наприклад, 8-розрядного МП i8080 ОЗП і ПЗП *логічно сполучено* (для звернення до їхніх комірок використовуються ті самі команди). *Фізично* ОЗП і ПЗП *розділено*, оскільки використовують різні мікросхеми, що адресуються способом «карти пам'яті» [1].

На рис. 1 наведено склад та організацію адресного простору пам'яті МП i8086.

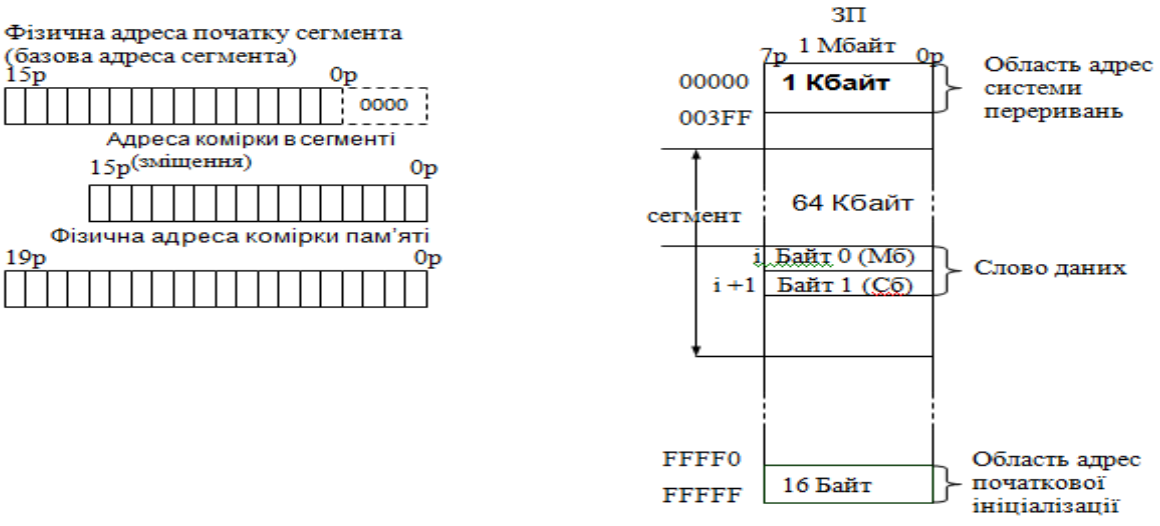


Рис. 1. Організація адресного простору пам'яті МП i8086

МП i8086 має 20-розрядну адресну шину, що дозволяє адресувати 2^{20} = 1 048 576 комірок пам'яті [1]. Розрядність комірки прийнято рівною 8 біт (1 байт), тобто МП *доступно 1 Мбайт* пам'яті. Схемні рішення, що будуть розглянуті нижче, забезпечують обмін з пам'яттю байтами або словами (16-розрядними даними) з автоматичним вибором необхідного формату. 20-розрядна фізична адреса може змінюватися *від 00000H до FFFFFH*.

Дві частини простору пам'яті, які розміщено в молодших адресах 00000…003FFH (1 Кбайт) і старших адресах FFFF0…FFFFFH (16 байт), використовуються відповідно для обслуговування переривань і початкової ініціалізації системи. *Слова даних у пам'яті* розміщуються в порядку збільшення номерів байтів: молодші байти за меншими адресами, старші байти – за більшими адресами.

Логічно весь адресний простір пам'яті розбито на сегменти, кожен з яких може мати *довжину* від 16 до 65 536 байт. Відповідно адресний простір МП (1 Мбайт) може бути розділено на *кількість сегментів* від 65 536 до 16.

Для адресації конкретної комірки в сегменті використовується 16-розрядна адреса – зміщення відносно початку сегмента, що змінюється від 0000H до FFFFH.

У конкретний момент часу МП доступні тільки *чотири сегменти*: коду, даних, стека і додатковий сегмент даних, що називаються *поточними*.

Для адресації комірок пам'яті окремих сегментів використовуються *дві 16-розрядні логічні адреси*: *перша* визначає початок сегмента і зберігається в одному з чотирьох сегментних регістрів: CS, DS, SS чи ES, а *друга* адресує комірку в обраному сегменті (є зміщенням відносно початку сегмента).

Формування 20-розрядної фізичної адреси комірки пам'яті за двома 16-розрядними логічними адресами: початку сегмента та зсуву в сегменті ілюструє наведений вище рис. 2. У разі обчислення 20-розрядної фізичної адреси комірки пам'яті 16-розрядна логічна адреса початку сегмента *зсувається вліво на чотири розряди*, у молодші біти записуються чотири нулі та до отриманої таким чином 20-розрядної *фізичної адреси початку сегмента* додають другу 16-розрядну логічну адресу (*зсув/зміщення у сегменті*).

В якості зсуву (зміщення) можуть бути:

- вміст регістра-покажчика команд (IP) у разі витягування (читання) команд із кодового сегмента;
- вміст регістра SP у разі виконання команд роботи зі стеком (PUSH/POP);
- ефективна (виконавча) адреса операнда EA (BA), що обчислюється під час виконання команди залежно від способу адресації операндів.

Більш детально це питання розглянуто у [1].

Розміщення сегмента в пам'яті задається 20-розрядною *фізичною адресою початку сегмента* (базовою адресою сегмента), молодша шістнадцяткова цифра

якого повинна дорівнювати 0H. Тобто *базова адреса сегмента* має вигляд XXXX0H, де X – будь-яка шістнадцяткова цифра (рис. 2).

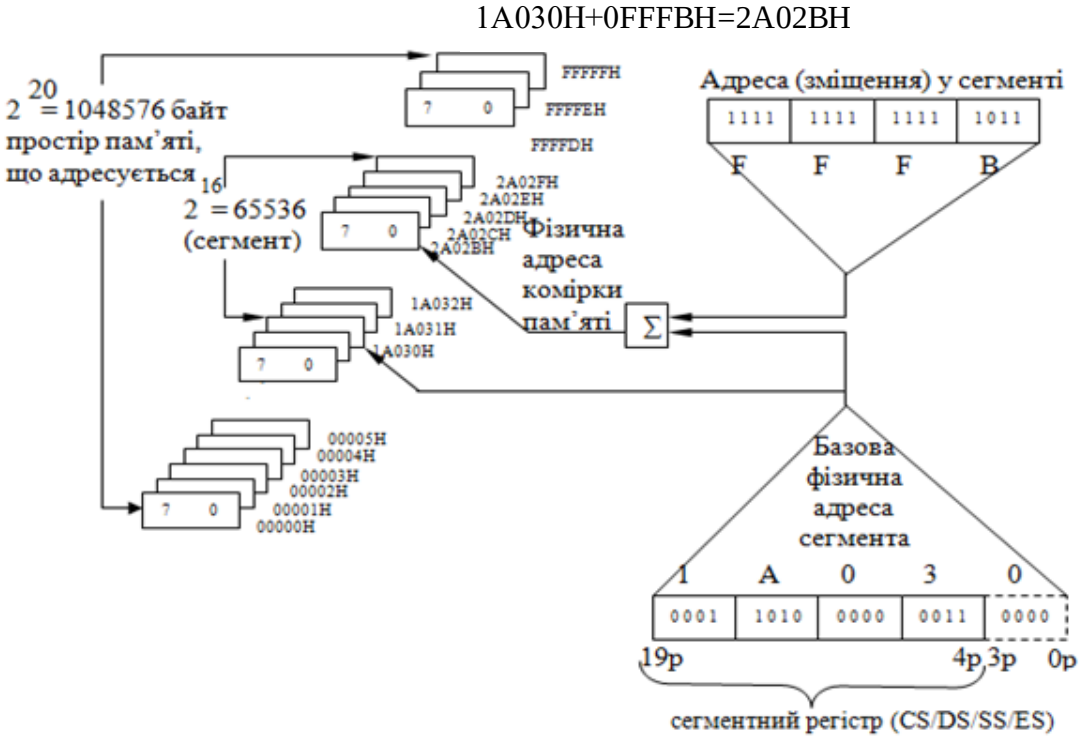


Рис. 2. Формування фізичної адреси комірки пам'яті

Шістнадцять старших значущих розрядів 20-розрядної фізичної базової адреси сегмента зберігаються в сегментних регістрах та називаються *логічною адресою початку сегмента* (логічною базовою адресою сегмента, логічною адресою сегмента).

У конкретний момент часу МП доступні тільки чотири сегменти пам'яті, що називаються *поточними*:

- поточний сегмент команд (адреса початку сегмента визначається регістром CS);
- поточний сегмент даних (адреса початку сегмента задається регістром DS),
- поточний сегмент стека (адреса початку сегмента визначається регістром SS);
- поточний додатковий (екстра) сегмент даних (адреса початку сегмента задається регістром ES).

Сегменти можуть *збігатися*, якщо збігається вміст сегментних регістрів, можуть *перекриватися*, якщо вміст регістрів відрізняється менш чим на 1000H, чи бути цілком *різними*, якщо коди в регістрах сегментів відрізняються більш ніж на 1000H.

У розглянутому на рис. 2 прикладі у сегментному регістрі міститься код 1A03H. Адреса комірки у сегменті (зміщення) дорівнює FFFBH. Повна 20-розрядна *фізична адреса комірки* утворюється в МП як сума вмісту сегментного регістра, що зсунуто вліво на 4 двійкових розряди та доповнено з правого боку кодом 0000B, і адреси в сегменті (зміщення):

1A030H + 0FFFBH = 2A02BH.

Таким чином, у прикладі обирається комірка пам'яті з фізичною адресою 2A02BH.

Сегментна організація пам'яті має свої *переваги та недоліки*. До її переваг належить модульність програм (у них чітко виділені області даних, стека і власне коди команд), просте переміщення програм у просторі пам'яті (що є важливим у *мультипрограмному середовищ*і), просте перемикання з однієї програми на іншу. Один з *недоліків* сегментної організації пам'яті полягає у труднощах маніпуляції фізичними адресами, наприклад, у разі порівняння двох адрес *нерівність логічних* адрес *не свідчить* про *нерівність фізичних* адрес.

2.1.2. Вибір типу сегмента пам'яті під час обчислення фізичної адреси

Оскільки в кожен момент часу *програмі доступні чотири поточних* сегментних регістри: CS, DS, SS чи ES, то існує *відповідність* між джерелом адреси в сегменті (зміщенням) і типом сегмента.

У разі відсутності додаткових вказівок прийняті *наступні призначення* сегментних регістрів (рис. 3), що задають *правила*, відповідно до яких МП для обчислення фізичної адреси комірки пам'яті обирає *сегментний регістр* за відомим джерелом *зміщення* в сегменті.

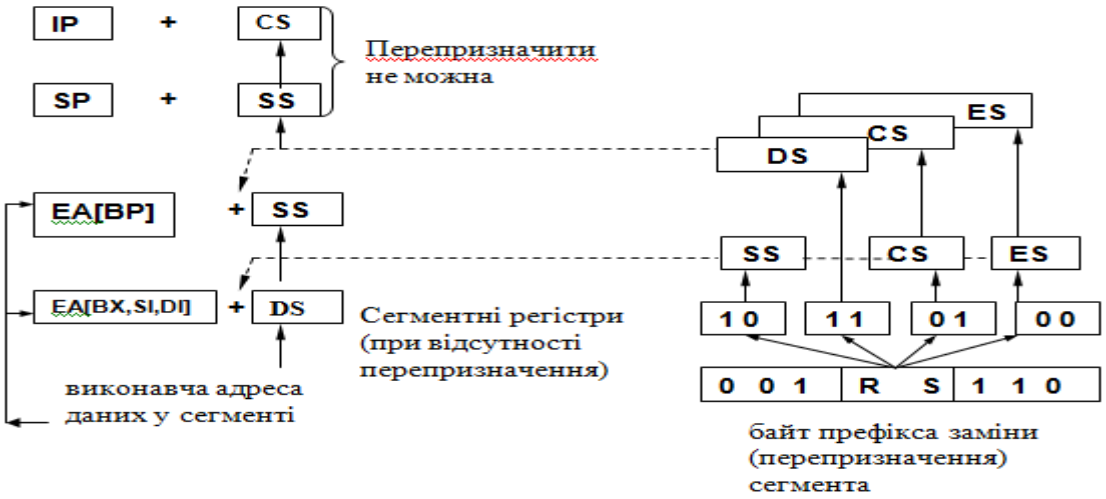


Рис. 3. Вибір типу сегмента пам'яті під час обчислення фізичної адреси

Під час *читання команд із пам'яті* зміщення витягається з регістра-покажчика команд (програмного лічильника) IP. У цьому випадку в разі обчислення фізичної адреси використовується сегментний регістр CS.

Якщо відбувається *звернення до стека* командами PUSH чи POP, то значення зміщення в сегменті залежить від вмісту регістра SP, а положення стекового сегмента в пам'яті визначається вмістом сегментного регістра SS.

Під час звернення до даних, які розміщено у поточних сегментах даних чи стека, МП формує *виконавчу (ефективну)* 16-розрядну адресу операнда, що є зміщенням відносно початку обраного сегмента.

Ефективна адреса являє собою *ціле беззнакове* число. *Значення ефективної* адреси залежно від зазначеного у машинному коді команди способу адресації операнда може визначатися вмістом одного з регістрів BX, BP, SI, DI, їхніми комбінаціями, та/або безпосередніми значеннями.

Якщо для обчислення ефективної адреси використовувався регістр-покажчик бази *BP*, то отримана ефективна адреса є зміщенням у сегменті стека, положення якого в пам'яті визначається вмістом сегментного регістра *SS*.

Якщо для обчислення ефективної адреси використовувалися регістри *BX, SI, DI*, або ефективна адреса *дорівнює 16-розрядному значенню*, яке входить у машинний код команди, то ефективна адреса є зміщенням у сегменті даних, положення якого в пам'яті визначається вмістом сегментного регістра *DS*.

Перші два призначення: CS:IP і SS:SP *змінити неможливо*.

Перепризначити можна лише сегментні регістри, а також сегменти пам'яті під час розрахунку фізичної адреси за *ефективною адресою*.

Це робиться за допомогою *однобайтного префікса* заміни сегмента (рис. 3), що повинен передувати машинному коду команди, яка звертається до нестандартного сегмента пам'яті.

Наприклад, мнемокоду команди mov [cs:codeByte], bh; M(CS·16+offset codeByte)<-bh у машинному коді *передує префікс* заміни сегмента: 2Eh=00101110b, який *перепризначає* сегмент DS на сегмент CS.

2.1.3. Розподіл пам'яті на банки

Коли *об'єм однієї комірки* пам'яті складає *8 біт*, а *СШД* має більшу кількість ліній, наприклад для МП i8086 – *16 біт*, пам'ять *фізично розділено на два банки* по 512 Кбайт (рис. 4).

Комірки пам'яті *обох банків адресуються паралельно 19-розрядними* адресами – A19…A1 (2^{19} = 512 Кбайт), а *доступ до банків* здійснюється *сигналами*: $\overline{BHE}$ – передавання за СШД старшого байта та $\overline{A0}$ – передавання за СШД молодшого байта. Один банк називається *молодшим* і містить *комірку з парними* адресами (обирається сигналом $\overline{A0}$), а *другий* банк (*старший*) – *з непарними* адресами (вибирається сигналом $\overline{BHE}$).

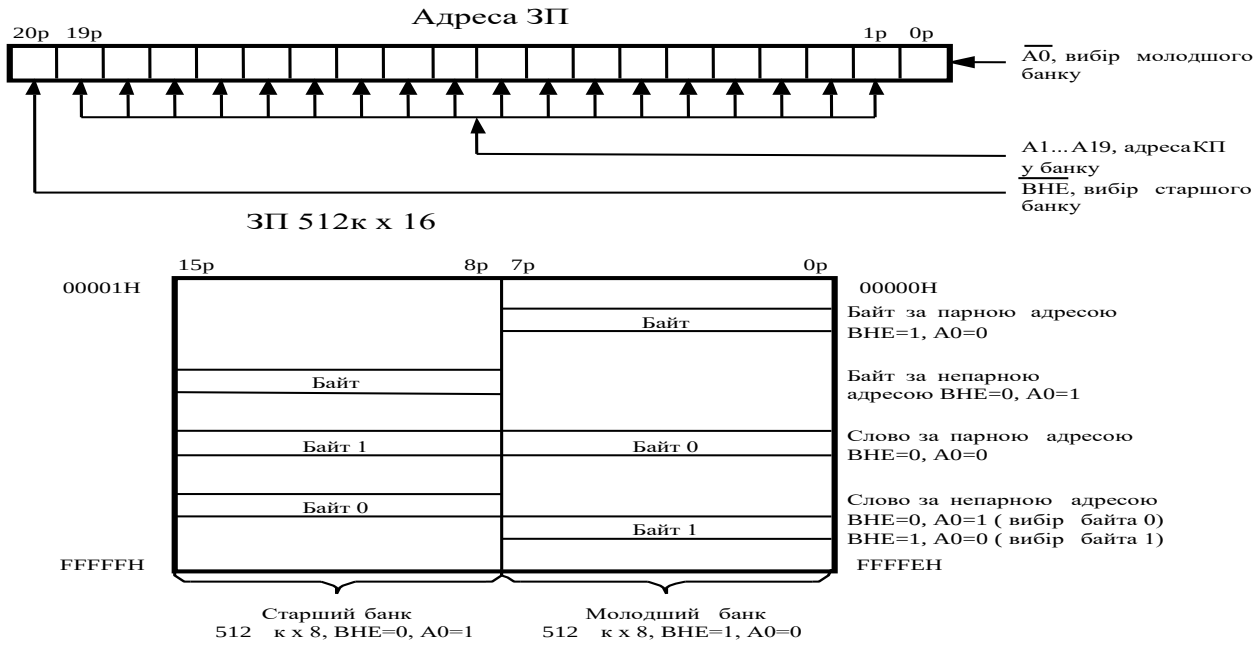


Рис. 4. Розподіл пам’яті МП i8086 на банки

Якщо 16-розрядне слово розташовано за парною адресою, то звернення до нього відбувається за один машинний цикл МП.

На рис. 5 наведено структурну схему модуля пам’яті *МПС мінімальної конфігурації* на основі МП i8086, яка реалізує розподіл пам’яті на банки, що описаний вище.

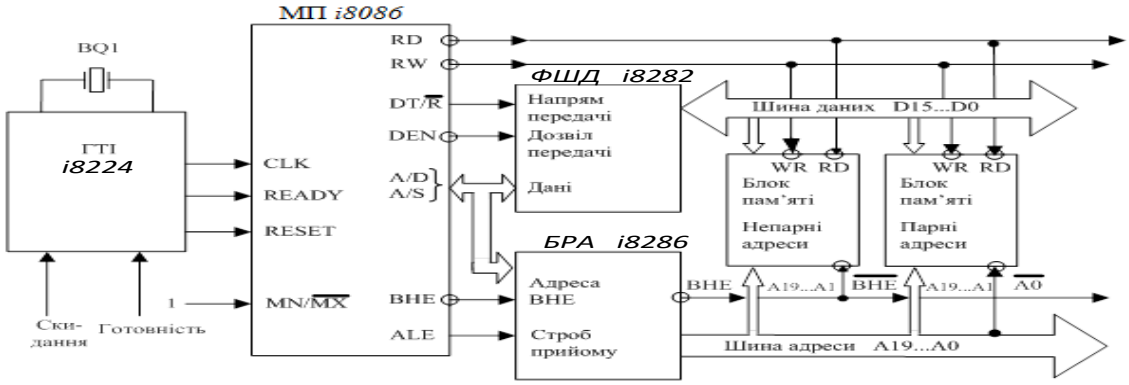


Рис. 5. Структурна схема модуля пам’яті МПС на основі МП i8086

Оскільки в МП використовуються *мультиплексовані лінії адреси/даних* і адреси/стану, значення адреси, що видаються по цих лініях на початку машинного циклу, необхідно запам’ятовувати.

Цю функцію виконують буферні регістри адреси (БРА), що запам’ятовують 20 розрядів адреси та значення сигналу $\overline{BHE}$ і видають їх на вихід стабільними на весь період обміну. Підключення ліній системної шини даних D15…D0 до МП здійснюється за допомогою формувачів шини даних (ФШД).

Сигнал CLK, що синхронізує роботу МП, надходить від генератора тактових імпульсів (ГТІ), що синхронізує також зовнішні сигнали готовності READY і скидання RESET.

Кожен цикл обміну з пам'яттю складається з чотирьох тактів Т1, Т2, Т3 і Т4, тривалість яких збігається з періодом ГТІ – CLK. Якщо *пам'ять знімає сигнал готовності* READY, то між тактами Т3 і Т4 вставляються такти чекання $T_{чек}$, під час яких МП чекає більш повільного партнера за обміном. Часову діаграму роботи МП i8086 для машинних циклів читання і запис наведено в [1].

Нижче наведено *функціональну сх*ему модуля пам'яті МПС на основі МП i8086 (рис. 6).

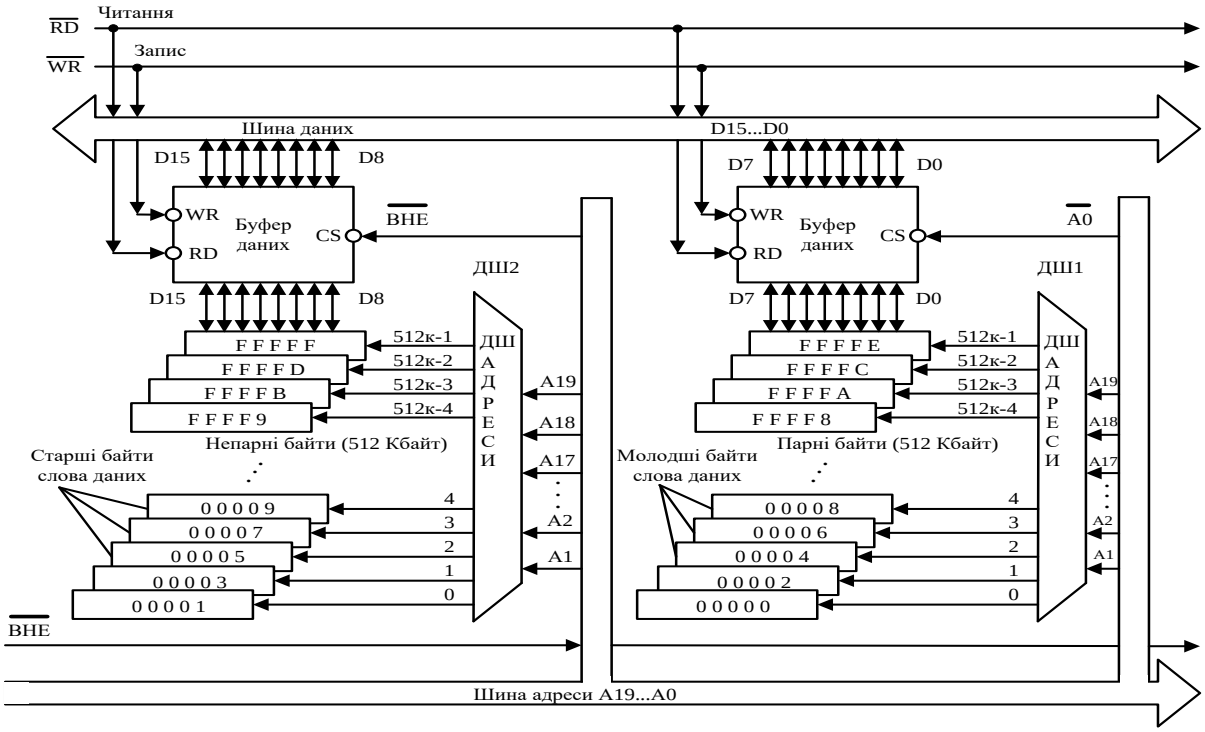


Рис. 6. Функціональна схема модуля пам'яті МПС на основі МП i8086

У банку пам'яті з *парними* адресами комірок (молодший банк) зберігаються дані, що передаються через буфер даних *за вісьма молодшими лініями СШД*: D7…D0. В банку пам'яті *з непарними* адресами (старший банк) знаходяться дані, що обмінюються з процесором *за вісьма старшими лініями СШД*: D15…D8.

Буфер даних непарного банку пам'яті підключається сигналом $\overline{BHE}$, *парного* – сигналом $\overline{A0}$.

Контрольні запитання та завдання

1. Що зберігається у ПЗП?
2. Для чого призначено ОЗП?
3. Обґрунтуйте необхідність використання режиму ПДП у МПС.
4. Назвіть основну функцію КПДП.
5. Опишіть особливості взаємодії МП та КПДП.
6. Опишіть особливості «Гардварської архітектури» МК.
7. Яку розрядність має одна комірка пам’яті програм AVR-МК?
8. Опишіть особливості підсистеми введення/виведення МК.
9. Опишіть призначення та місце модуля пам’яті у МПС.
10. Опишіть підключення модуля пам’яті до системної шини.
11. Наведіть класифікацію пам’яті.
12. Чим відрізняються енергонезалежна та енергозалежна пам’ять?
13. Чим відрізняються статична та динамічна пам’ять? Наведіть приклади.
14. Наведіть та поясніть основні характеристики пам’яті.
15. Дайте визначення FLASH ПЗП.
16. Поясніть фізичну та логічну організацію пам’яті в МПС на основі i8086.
17. Як адресуються комірки пам’яті в МПС на основі МП i8086?
18. Поясніть призначення та організацію стека у МПС на основі МП i8086 і МК типу AVR.

ЛЕКЦІЯ 4. ОРГАНІЗАЦІЯ ПАМ'ЯТІ МПС НА ОСНОВІ МІКРОКОНТРОЛЕРА. ПРОГРАМУВАННЯ FLASH- ТА EEPROM-ПАМ'ЯТІ

1. ОРГАНІЗАЦІЯ ПАМ'ЯТІ НА ОСНОВІ МІКРОКОНТРОЛЕРА

1.1. Загальна характеристика пам'яті на основі 8-розрядного мікроконтролера

Нижче розглядається організація пам'яті МПС на основі 8-розрядного AVR-мікроконтролера. В цих мікроконтролерах (МК) 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; 2]. 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. 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. 1. Карта пам'яті деяких мікроконтролерів сімейства Mega

Примітка: позначкою «•» на рис. 1 відмічено моделі, до яких *можна підключити* зовнішній СОЗП.

Оскільки AVR-мікроконтролери мають *16-бітну систему команд*, об’єм пам’яті програм на рис. 1 вказано *не у байтах*, а *в 16-бітових словах*. Символ «$» є ознакою шістнадцяткової системи числення.

1.2. Організація пам’яті програм AVR-мікроконтролерів

Вище у лекції 3 було розглянуто пам’ять програм «інтелоподібних» МП/МК, що мають вісьмирозрядну організацію. Пам’ять програм AVR-мікроконтролерів має *шістнадцятибітну організацію*, для якої довжина кожної команди кратна одному слову – 16 біт.

Наприклад, *об’єм пам’яті програм* МК сімейства *Mega* складає від 2К (2·1024) до більш ніж 128К (128·1024) шістнадцятибітних слів.

У частини моделей сімейства пам’ять програм *логічно поділено* на 2 рівні частини: область прикладної програми і область завантажувача [2].

В області завантажувача може розташовуватися *спеціальна програма – завантажувач*, що дозволяє МК самостійно керувати завантаженням та вивантаженням прикладних програм. Використання цієї області та реалізація програми-завантажувача розглянуто окремо у [2]. Якщо самопрограмування МК 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) МК виконання програми починається з цієї адреси, де повинна розміщуватися команда переходу до основної частини ініціалізації програми. Починаючи з адреси $0001/$0002 розташовується *таблиця векторів переривань*. Розмір цієї області залежить від моделі МК [2].

У разі *виникнення переривання* після збереження у стеку поточного значення лічильника команд відбувається виконання команди, розташованої за адресою відповідного вектора. Ці команди *виконують перехід* 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 МК положення вектора скидання і таблиці векторів переривань *може бути змінено*. Вони можуть розташовуватися не лише на початку пам’яті програм, 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*.

1.3. Організація пам’яті даних AVR-мікроконтролерів

1.3.1. Загальна характеристика пам’яті даних AVR-мікроконтролерів

AVR-мікроконтролери мають пам’ять даних *двох видів*: енергозалежну статичну пам’ять даних (СПД) і енергонезалежну EEPROM-пам’ять.

Статична пам’ять даних включає *регістрову* пам’ять та статичний оперативний запам’ятовуючий пристрій (*СОЗП*). Регістрова пам’ять має *32 регістри* загального призначення (РЗП), об’єднаних у *регістровий файл*, і *64 основні службові* регістри введення/виведення (РВВ). В моделях з розвиненою периферією є також область *додаткових (англ. extended*) РВВ (ДРВВ). *Під основні* РВВ в пам’яті МК відводиться *64 байти*, а *під ДРВВ − 160 або 416 байт* (залежно від моделі).

В обох областях РВВ розташовуються різні *службові регістри*: регістри керування МК, регістри стану і т. ін., а також регістри керування периферійними пристроями, що входять до складу МК. Загальна *кількість* РВВ і ДРВВ *залежить* від конкретної моделі МК.

Для зберігання змінних окрім РЗП використовується також СОЗП. Ряд МК сімейства, крім того, мають *можливість підключення зовнішнього СОЗП*, об’ємом *до 64 Кбайт.*

Для *довготривалого зберігання* різної інформації, яка може змінюватися в процесі функціонування готової системи (калібрувальні константи, серійні номери, ключі і т. ін.), в МК 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.

1.3.2. Статична пам’ять даних

В AVR-мікроконтролерах використовується *лінійна організація* СПД (рис. 2).

До складу СПД *входять РЗП, РВВ, а також СОЗП*. *Максимальний об’єм* СПД складає *65536 байт* та залежить від конкретної моделі сімейства.

1.3.2.1. Регістри загального призначення

32 регістра загального призначення об’єднано в *регістровий файл* швидкого доступу, структуру якого показано на рис. 3.

В AVR-мікроконтролерах *всі РЗП безпосередньо доступні АЛП*, на відміну від 8-бітових МК деяких інших фірм, в яких є лише один такий регістр − робочий регістр A/W (акумулятор).

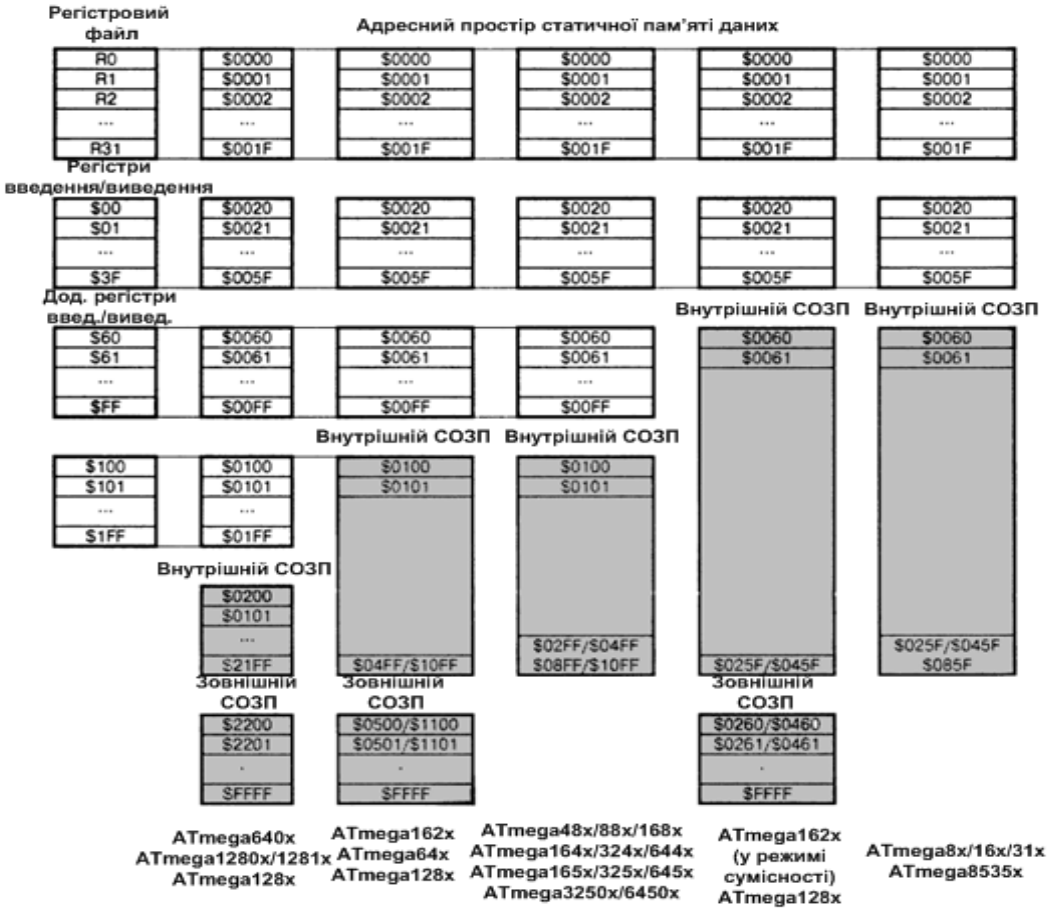


Рис. 2. Організація СПД

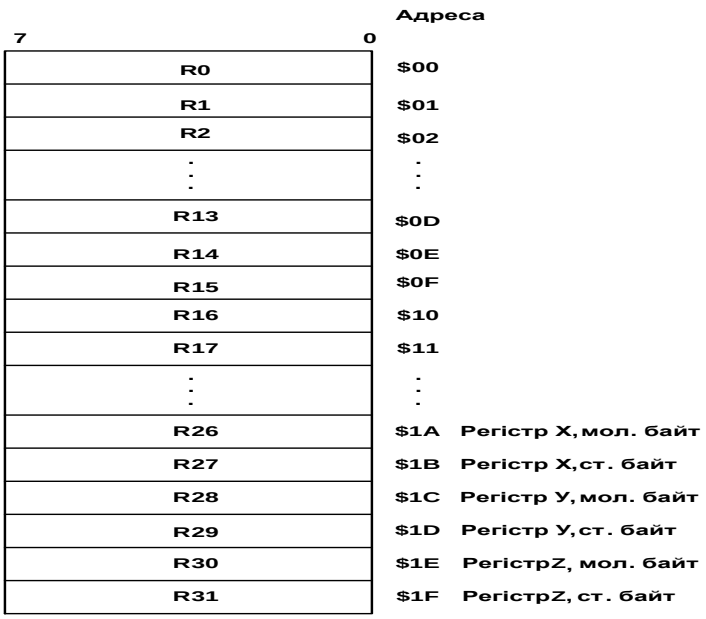


Рис. 3. Структура РЗП (регістрового файлу)

Як показано на рис. 3, кожен *регістр файлу має* свою власну адресу в просторі СПД. Тому до них можна відповідними командами звертатися *двома способами* – як до регістрів та як до СПД в цілому. Таке рішення є ще однією відмінною

особливістю архітектури AVR, що *підвищує ефективність* роботи МК та його *продуктивність*.

Останні 6 регістрів файлу (R26...R31) можуть об'єднуватися у *три 16-бітних регістри:* X, Y та Z (рис. 4), що використовуються *як покажчики* під час *непрямої адресації* СПД.

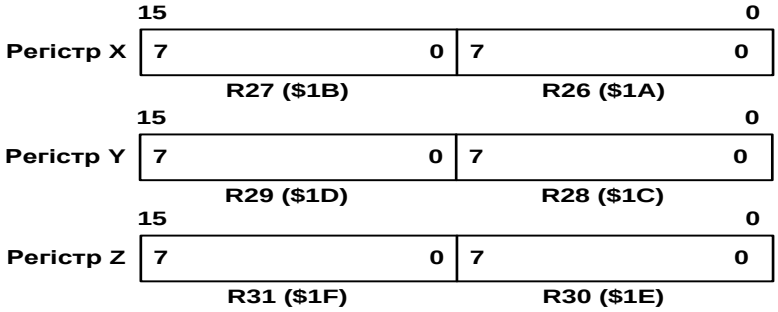


Рис. 4. Регістри-покажчики X,Y та Z

1.3.2.2. Стек

Призначення та організацію стека в МПС описано у лекції 3. AVR-мікроконтролери *мають обидва різновиди* стека (в залежності від моделі) апаратний і програмний. Стек *розміщується* у СОЗП, і його глибина визначається тільки розміром вільної області цієї пам'яті.

В залежності від ємності СОЗП, *як покажчик стека* використовується або один РВВ: 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 РВВ і, відповідно, програмно доступні. В наборі команд AVR-МК є команди занесення в стек – *PUSH* і зчитування зі стека – *POP*, що дозволяє програмі використовувати стек для своїх потреб. На початку програми покажчик стека 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 *верхівку* стека – першу вільну комірку стекової пам'яті.

У разі програмного виклику підпрограм, або виклику підпрограм за перериванням, *адреса команди*, розташованої за командою виклику, *зберігається у стеку*. *Значення покажчика* стека у цьому разі зменшується на 2, тому що для збереження лічильника команд потрібно 2 байти. Під час *повернення з підпрограми* ця адреса відновлюється зі стека і завантажується в лічильник команд. Значення покажчика стека відповідно *збільшується на* 2.

1.3.2.3. Регістри введення/виведення

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

У AVR-мікроконтролерах частина РВВ розташовуються в так званому *основному просторі* введення/виведення, розміром 64 байти. У більшості моделей сімейства є також *простір додаткових* РВВ розміром 160 або 416 байт. Введення додаткових РВВ пов'язане з тим, що для підтримки всіх периферійних пристроїв, наявних в цих моделях, *адрес* перших основних 64-х РВВ *недостатньо*.

Кількість та розподіл адрес простору введення/виведення (як основного, так і додаткового) залежить від конкретної моделі МК або від складу і можливостей периферійних пристроїв даної моделі [2].

У разі вказівкі 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. Якщо адреса, наприклад, регістра SREG вказана *як $3F ($5F),* то це означає що *за адресою $3F* можна відповідними командами звернутися до РВВ основного простору РВВ, 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 основними РВВ знаходяться *32 РЗП*.

Приклади РВВ деяких Mega-AVR-мікроконтролерів наведено у [2].

До РВВ, 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. Окрім адресації РВВ 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 МК під час виконання програми. Більшість прапорців у разі настання 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* МК вони скидаються *в нуль*. Формат регістра SREG наведено на рис. 5, а опис його його окремих розрядів – в табл. 1.

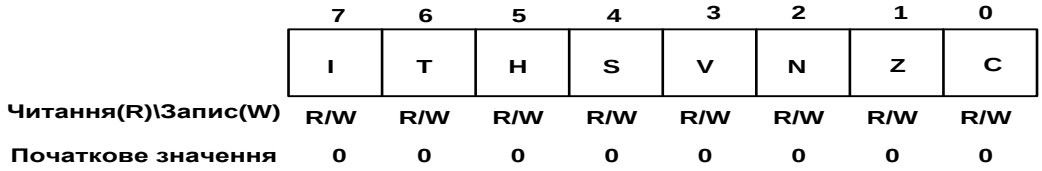


Рис. 5. Формат регістра стану SREG

Таблиця 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 I 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.** *Розряд регістра РЗП* 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 у разі виконання 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 у разі *переповнення розрядної сітки* знакового результату. Використовується під час роботи зі знаковими 8-ми розрядними 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 над додатними числами відбувся *вихід за межі байта*

1.4. Використання зовнішнього оперативного запам'ятовувального пристрою

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, що використовуються для підключення зовнішнього ОЗП, схема його підключення до МК, конфігурація, часові діаграми звернення до зовнішнього ОЗП та програмна реалізація *описані* у [2].

1.5. Енергонезалежна пам'ять даних EEPROM

Частина МК сімейства Tiny та всі МК сімейств Mega та Xmega *мають* енергонезалежну пам'ять даних (EEPROM-пам'ять). Цю пам'ять *розміщено у власному* адресному просторі, а її об'єм становить від 60 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 регістрах введення/виведення: EEARH:EEARL. Адреси, формати та опис окремих розрядів регістрів керування EEPROM-пам'яті наведено у [2].

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

1. *Для визначення готовності* пам'яті до запису даних треба 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 (Store Program Memory Control Register).
3. *Завантажити байт* даних у регістр EEDR, а *необхідну а*дресу – у регістр EEAR.
4. Встановити *в одиницю прапорець* EEMPE (EEMWE) регістра EECR.
5. *Протягом чотирьох* 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 МК ніколи не змінює вміст пам'яті програм, другий крок можна пропустити.

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 [2].

За закінченням циклу запису розряд EEPE (EEWE) апаратно скидається, після чого програма може почати запис наступного байта.

Під час запису в EEPROM можуть виникнути деякі проблеми, які викликані перериваннями, тому рекомендовано *забороняти всі переривання* (скидати біт I регістра SREG) на час виконання пунктів 2…5 описаної вище послідовності [2].

Два приклади реалізації *функції запису* в EEPROM-пам’ять та *процедуру читання* наведено у [2].

1.6. Особливості проектування пам’яті великого об’єму

Пам’ять великих об’ємів звичайно *поділяється на кілька модулів (сторінок),* кожна з яких має об’єм, що обирається виходячи з можливостей реалізації на ВІС, і виконується у вигляді автономного модуля. *Під час сторінкової* адресації спочатку обирається сторінка, а потім комірка на сторінці.

Сторінки можна обирати, наприклад, *за допомогою дешифратора*. У цьому випадку використовується *дешифратор* ДВК в унітарний. На вхід дешифратора подаються *старші розряди адреси* пам’яті, що задають *номер сторінки*. З виходів дешифратора знімається *унітарний код*, що обирає одну зі сторінок, адреса якої у ДВК надійшла на вхід дешифратора. Молодші розряди адреси *одночасно* надходять на всі сторінки. *Проте активним сигналом з виходу дешифратора в кожний момент часу буде обрано лише одну зі сторінок. Виходи* мікросхем пам’яті *не обраних* сторінок будуть знаходитися у високоімпедансному стані.

2. ПРОГРАМУВАННЯ FLASH- ТА EEPROM-ПАМ’ЯТІ

2.1. Режими програмування

Одним з питань, яке треба вирішувати під час розробки мікроконтролерної системи (МКС), є програмування FLASH- та EEPROM-пам’яті.

AVR-мікроконтролери підтримують *такі режими* програмування [2]:

−режим послідовного програмування (за SPI-інтерфейсом);

−режим паралельного програмування за високої напруги;

−режим програмування за JTAG-інтерфейсом.

2.2. Сторінкова організація пам’яті програм і даних

В AVR-мікроконтролерах використовується сторінкова організація FLASH-пам’яті програм та EEPROM-пам’яті даних [2].

Під час програмування FLASH-пам’яті весь її об’єм розбивається на сторінки, розмір і кількість яких залежить від об’єму пам’яті програм МК (табл. 2).

Таблиця 2. Параметри сторінкової організації пам’яті програм

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

Дані спочатку завантажуються в *буфер сторінки* і тільки потім заносяться безпосередньо в пам’ять програм. Прошивка всіх комірок сторінки у цьому разі відбувається одночасно.

Аналогічно організовано EEPROM-пам’ять. Розмір сторінок, а також їх кількість наведено в табл. 3.

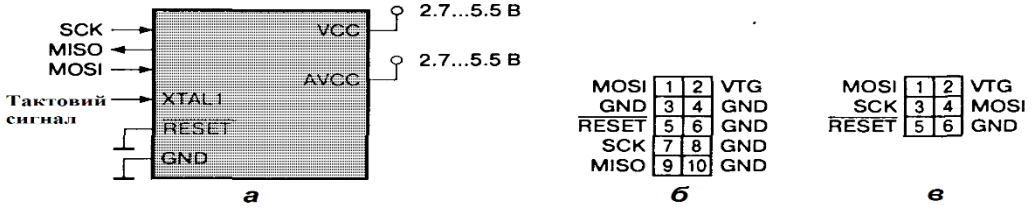
Таблиця 3. Параметри сторінкової організації EEPROM-пам’яті

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

У багатьох моделях *сторінкова* організація EEPROM-пам’яті використовується тільки під час програмування в *паралельному* режимі, а програмування *за послідовним* каналом відбувається *побайтно*.

2.3. Програмування пам’яті за послідовним каналом

Для програмування або перепрограмування пам’яті програм і даних *безпосередньо у пристрої* використовують послідовний *SPI-інтерфейс* (рис. 6).



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

Рис. 6. Включення МК у режимі програмування за послідовним каналом

На рис. 6*а* наведено *схему включення мікроконтролера* у цьому режимі. На рис. 6*б, в* показано *два варіанти розводки колодки* для підключення програматора, які рекомендовано компанією Atmel.

Зокрема, перший варіант розводки (рис. 6*б*) використовується в програматорі AVRISP і налагоджувальних платах STK200/300 компанії Atmel.

Другий варіант розводки (рис. 6*в*) використовується в більш нових пристроях компанії – програматорі AVRISP mkll і платі STK500.

Як видно з рис. 6*а*, для обміну даними між програматором і пристроєм використовується *три лінії SPI-інтерфейсу*: SCK – лінія тактового сигналу, MOSI – вхід даних і MISO – вихід даних. *Відповідність* між цими лініями інтерфейсу і контактами портів введення/виведення деяких МК наведено в табл. 4.

Таблиця 4. Виводи, які використовуються під час програмування 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

У деяких моделях мікроконтролерів виводи, що використовуються для програмування, *не збігаються* з виводами портів введення/виведення *модуля SPI*.

Часові діаграми, параметри сигналів, команди та послідовність програмування *наведено* у [2].

Контрольні запитання та завдання

1. Опишіть організацію пам’яті AVR-мікроконтролерів.
2. За якою архітектурою виконано організацію пам’яті AVR-мікроконтролерів сімейства Mega?
3. Наведіть структурну схему карти пам’яті AVR-мікроконтролерів сімейства Mega. Опишіть структуру і призначення кожного виду пам’яті.
4. Опишіть структуру регістрового файлу AVR-мікроконтролерів.
5. Де в AVR-мікроконтролерах розташовуються РВВ? Який з них використовується найчастіше?
6. Який можливий максимальний розмір ОЗП AVR-мікроконтролерів?
7. Що таке енергонезалежна пам’ять даних EEPROM? Які регістри використовуються для керування EEPROM?
8. Які режими програмування підтримують МК сімейства Mega?
9. Опишіть формат та призначення окремих розрядів регістра SREG.
10. Що означає символ «$» у разі наведення адрес пам’яті?
11. Для чого використовується лічильник команд (Program Counter – PC) та від чого залежить його розмір?

ЛЕКЦІЯ 5. ПРОГРАМНА МОДЕЛЬ МП/МК. ХАРАКТЕРИСТИКА КОМАНД МП/МК

1. ПРОГРАМНА МОДЕЛЬ МП/МК

1.1. Послідовність розробки робочої керувальної програми

Якщо МПС виконана *на основі мікропроцесора*, то керувальна програма знаходиться у зовнішній пам'яті програм (комірках ПЗП), а якщо основним вузлом МПС *є мікроконтролер*, то програма може знаходитися у резидентній пам'яті програм і/або зовнішній пам'яті програм. *Без керувальної програми* жодна МПС працювати не буде.

Основні етапи створення керувальної програми:

- розробка схеми алгоритму роботи, який повинна реалізувати керувальна програма;
- якщо задача складна, то програма може бути реалізована і відлагоджена мовою високого рівня або мовою Асемблера;
- компіляція – переведення програми з мови високого рівня або Асемблера в машинні коди конкретного МП/МК (створення об'єктного модуля);
- введення керувальної програми у ПЗП за допомогою програматора;
- початкова ініціалізація системи, в процесі якої у програмний лічильник завантажується початкова адреса програми.

Керувальна програма може включати *підпрограми* – частини основної програми, які можуть бути викликані *відповідною командою* з основної програми або *за перериванням*.

Однією з основних характеристик архітектури будь-якого МП/МК є *програмна (програмістська*) модель, що включає *частину структури*, що доступна програмісту за допомогою системи команд [1; 2].

1.2. Програмна модель 8-розрядного мікропроцесора

На рис. 1 наведено програмну (програмістську) модель *8-розрядного МП*, наприклад, i8080.

До програмістської моделі МП належать:

- регістри загального призначення: B, C, D, E, H, L, які є надоперативним ОЗП;
- слово-стану програми – Program Status Word (рис. 2), яке складається з акумулятора A і регістра прапорців (англ. RF): A – старший байт, RF – молодший байт.

Регістр прапорців є *набором тригерів*, які показують стан програми після виконання команд, які впливають на прапорці (див. лекцію 4).

Рис. 1. Програмна (програмістська) модель МП i8080

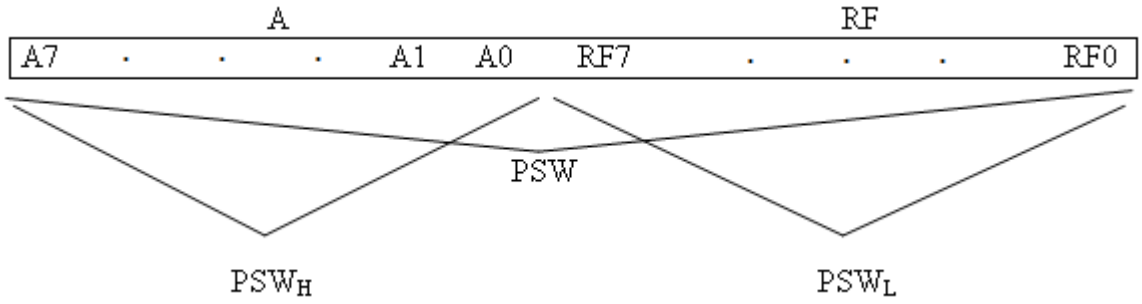


Рис. 2. Слово-стану програми (PSW) МП i8080

1.3. Програмна модель 16-розрядного мікропроцесора

В програмній моделі 16-розрядного МП, наприклад, i8086, яку показано на рис. 3, є кілька груп регістрів з відповідним функціональним призначенням.

Мікропроцесор має вісім 16-розрядних *регістрів загального призначення (РЗП)*, для вибору яких в машинних кодах команд досить трьох бітів.

Чотири РЗП: AX, BX, CX і DX допускають в командах програми вказувати їх старшу (H – High), або молодшу (L – Low) половини. Такий подвійний характер цих регістрів дозволяє багатьом командам оперувати байтами і словами. Інші чотири РЗП можна використовувати тільки словами.

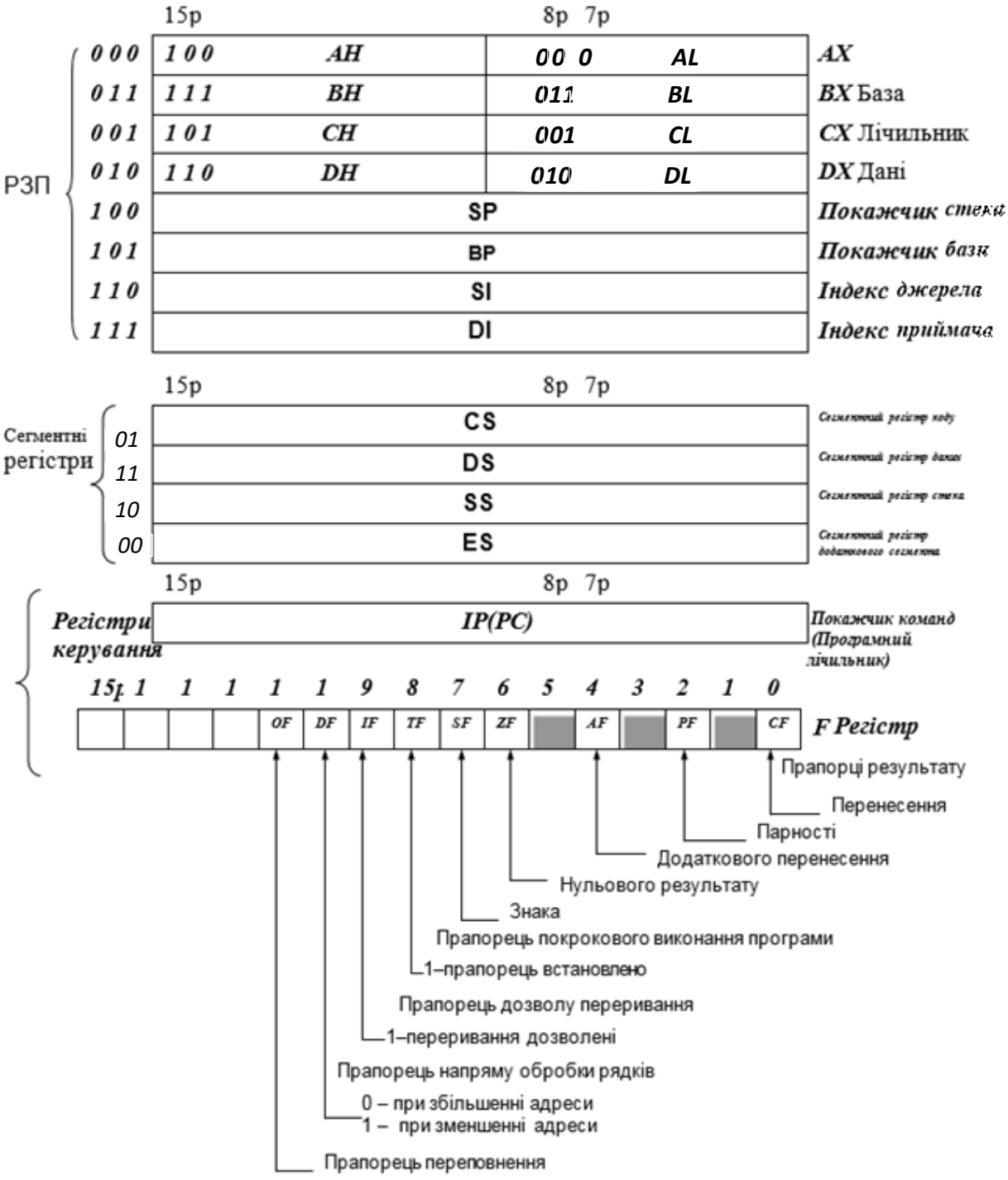


Рис. 3. Програмна модель МП i8086

Регістри AX…DX беруть участь в арифметичних і логічних операціях, але у деяких командах вони *спеціалізовані*, що відбито в їхніх назвах. Регістр AX використовується в операціях множення, ділення, введення/виведення слів і у деяких операціях з рядками. Регістр AL бере участь в аналогічних операціях з байтами, в операціях перетворення і десяткової арифметики. Регістр BX широко застосовується для адресації *структур даних* у сегменті даних за замовчуванням.

Регістр CX виконує функції лічильника повторень у програмних циклах і в операціях з рядками. Регістр CL служить *лічильником зсувів*. Регістр DX бере участь в операціях *множення і ділення* слів. Окрім того, він зберігає *адресу порту* в командах введення/виведення з непрямою адресацією.

Регістри BP, SP, SI і DI утворюють групу *покажчиків та індексних* регістрів. Вони можуть використовуватися для збереження адрес, забезпечуючи непряму адресацію пам'яті, а також беруть участь в обчисленнях ефективної адреси (див. лекцію 3). Однак ці регістри можуть залучатися для арифметичних і логічних операцій так само, як і *регістри загального призначення*. Покажчик *стека SP* використовується для адресації *верхівки стека* в сегменті стека SS. Обидві стекові операції (PUSH і POP) автоматично модифікують SP. За допомогою *покажчика бази* BP забезпечується доступ до даних, що знаходяться в сегменті стека (*не тільки у верхівці стека*). Зазвичай регістр BP залучається *для адресації параметрів*, що передаються через стек підпрограмам. *Індексні регістри* SI і DI застосовуються для адресації даних, а *також елементів рядків* в командах роботи з рядками.

В нижній частині рис. 3 розміщено *регістр прапорців (ознак).* Шість прапорців CF, PF, AF, ZF, SF і OF фіксують визначені ознаки результату відповідної арифметичної чи логічної операції (перші п'ять прапорців аналогічні прапорцям МП i8080):

– прапорець перенесення CF фіксує значення біта перенесення, що виникає під час додавання/віднімання байтів чи слів, а також значення біта, що висувається в операціях зсуву. Він також показує особливість результату операцій множення та ділення. Прапорець перенесення відіграє важливу роль у разі написання програм;
– прапорець паритету (парності) PF переходом *в одиницю* реєструє наявність *парного* числа одиниць в восьми молодших бітах результату операції. Він застосовується для контролю достовірності під час передачі даних;
– прапорець допоміжного перенесення AF аналогічний прапорцю CF, але фіксує перенесення з молодшої тетради результату (чи позики). Цей прапорець використовується в операціях двійково-десяткової арифметики;
– прапорець нуля ZF сигналізує про одержання *нульового* результату операції;
– прапорець знаку SF *повторює* значення *старшого біта* результату, що у додатковому коді відповідає знаку числа;
– прапорець переповнення OF відзначає втрату старшого біта результату операції додавання чи віднімання *над знаковими* числами. Переповнення виникає, коли значення перенесень у старший біт і зі старшого біта не збігаються та результат операції не відповідає діапазону чисел зі знаком.

Прапорець OF показує також зміну старшого (знакового) біта у разі арифметичних зсувів вліво.

Три останніх прапорці керують деякими діями МП. Програміст може відповідними командами задати стан кожного з них:

– прапорець напрямку DF визначає сканування (перегляд) *елементів рядків* від менших адрес до більших (DF = 0) чи навпаки (DF = 1);

– прапорець переривання IF задає реакцію МП на запит переривання на вході INT. Якщо IF = 0, запит переривання ігнорується, а якщо IF = 1, МП розпізнає і відповідно реагує на нього. Прапорець IF не впливає на сприйняття переривань на вході NMI, які не маскуються, і внутрішніх переривань;

– прапорець трасування (покрокового виконання програми) TF під час встановлення в одиницю переводить МП у покроковий режим роботи, в якому МП автоматично генерує внутрішнє переривання після виконання кожної команди.

Покажчик команд IP виконує функції програмного лічильника PC (далі використовується назва – IP). В процесі вибірки команд із програмної пам’яті відбувається відповідна модифікація IP для того, щоб він адресував наступну команду, яка повинна виконуватись.

Чотири сегментних регістри: коду – CS, даних – DS, стеку – SS і додаткових даних – ES використовуються під час адресації пам’яті.

Мікропроцесор має 20-розрядну шину адреси пам’яті, але в програмній моделі немає жодного регістра завдовжки 20 біт. Всередині МП фізична адреса пам’яті подана двома 16-бітовими словами, одне з яких називається *логічною базовою (початковою) адресою сегмента*, а друге – *внутрішньосегментним зсувом*. Ці два слова являють собою *32-бітову логічну адресу* комірки пам’яті. Пристрій перетворення адрес в шинному інтерфейсі МП у разі кожного звертання до пам’яті перетворює логічну адресу у фізичну. Програма використовує *чотири поточних сегменти пам’яті* з *максимальним розміром* 64 Кбайт.

Реальний розмір сегмента необов’язково має бути максимальним. *Мінімальний розмір* сегмента дорівнює 16 байтам. На розміщення сегментів в просторі 1 Мбайт накладається тільки одне обмеження: базова *20-бітова фізична* адреса сегмента має бути *кратна 16*, тому його *4 молодших біти* мають бути *нульовими*. Інакше кажучи, *фізична базова адреса сегмента* має вигляд: XXXX0H. Для *логічної базової адреси* сегмента досить 16 біт. Такі *«урізані»* логічні базові адреси сегментів знаходяться у регістрах CS, DS, SS і ES.

Максимальний робочий простір, до якого МП має доступ у будь-який момент часу, складається з *64 Кбай*т для коду (власне програми), *64 Кбайт* для стека і

128 Кбайт для даних. Якщо програмі потрібний більший робочий простір, вона повинна *модифікувати вміст* сегментних регістрів.

Для адресації конкретного байта в сегменті служить другий компонент логічної адреси – *зсув.* Він є *16-бітовим цілим беззнаковим числом* та показує відстань цього байта від початку сегмента. Отже, *для утворення фізичної адреси* з пари: початок сегмента – зсув необхідно зсунути логічну 16-бітову базову адресу сегмента вліво на 4 біти (у цьому разі молодша тетрада адреси отримає нулі) і додати зсув.

1.4. Програмна модель мікроконтролера

На рис. 4 наведено програмну модель *одного з AVR*-мікроконтролерів. Ця модель має регістри загального призначення, регістри введення/виведення, статичний ОЗП, який у технічній літературі англійською мовою називають SRAM, ємністю 128 байт, EEPROM-пам’ять даних, ємністю 512 Кбайт та ПЗП FLASH-типу, ємністю 2 К слів (4 Кбайт). Регістри загального призначення, регістри введення/виведення та статичний ОЗП називають *статичною пам’яттю даних* (СПД), тому що вони є енергозалежними. Відповідно EEPROM-пам’ять даних є *енергонезалежно*ю. Опис окремих складових моделі наведено у лекції 4.

2. ХАРАКТЕРИСТИКА КОМАНД МП/МК

2.1. Код операції команди

Код операції команди – комбінація бітів (не більше восьми для 8-розрядних МП/МК), що знаходяться на початку машинного коду команди та визначають *тип операції,* що підлягає виконанню в конкретний момент часу. Код операції витягається з пам’яті та розміщується в програмно недоступний регістр команд. Потім він декодується і визначається тип команди, яка має бути виконана. Для 16-розрядних МП код операції частково може знаходитися у другому байті машинного коду команди (постбайті). Окрім кодів операції в машинний код команди можуть входити *адреси та операнди*.

2.2. Мнемоніка команди та мнемокод

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

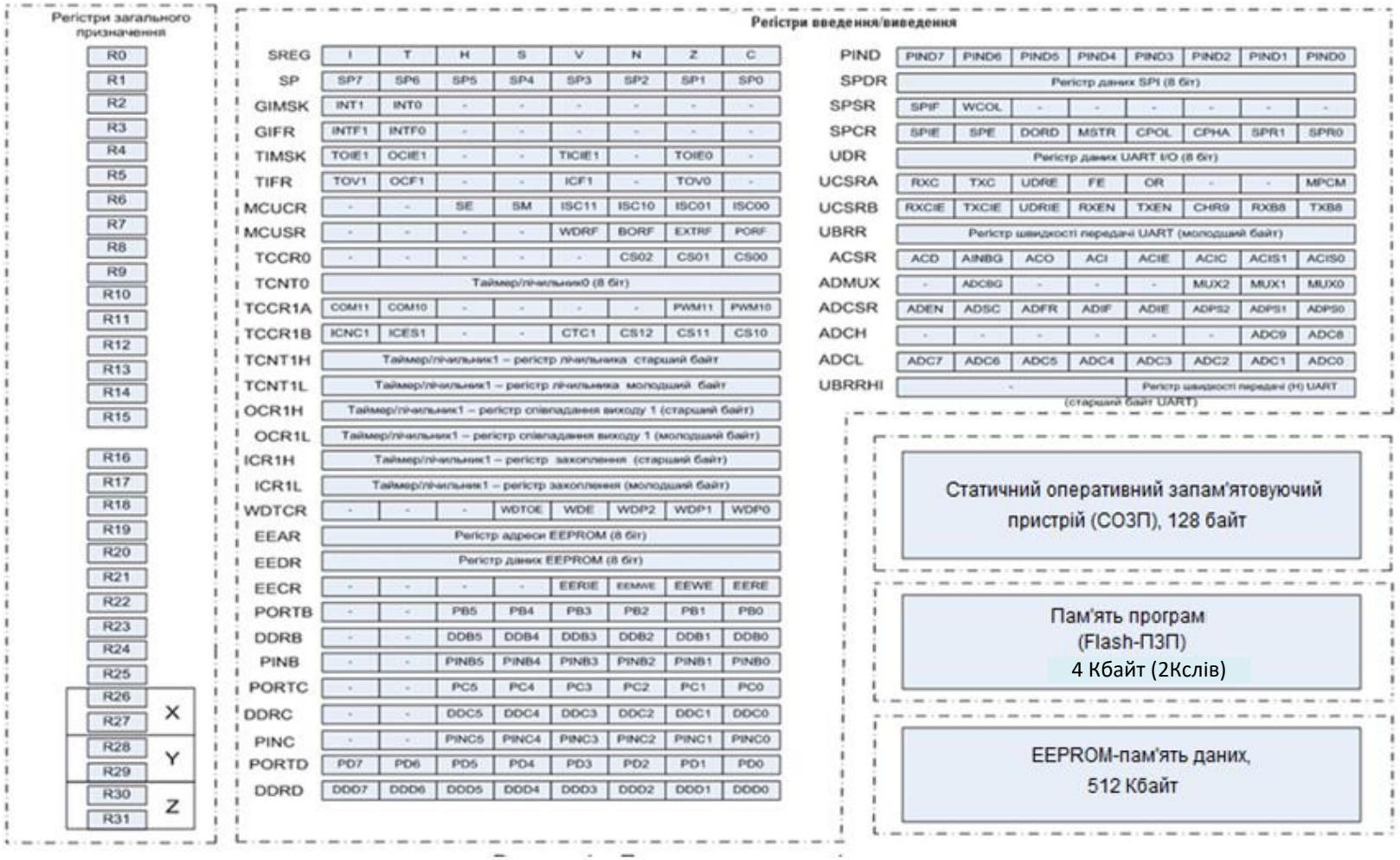


Рис. 4. Програмна модель одного з AVR-мікроконтролерів

Мнемоніки введені для полегшення написання програм і складають *основу мови Асемблера.*

Мнемокод включає мнемоніку команди та описання операндів, які беруть участь в операції.

2.3. Машинний код команди

Машинний код команди є двійковим кодом команди, який складається з одного або декількох байтів залежно від типу команди конкретного МП/МК.

2.4. Операнди

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

2.5. Коментар

Коментар використовується для полегшення читання програми. Компілятор (Асемблер) пропускає коментарі, що показані в програмі через ";" після або до мнемоніки команди, не генеруючи у цьому разі ніякого машинного коду.

2.6. Формати команд

2.6.1. Команди 16-розрядного мікропроцесора

Основні формати команд 16-розрядного МП, наприклад, i8086, наведено на рис. 5.

Команди *розміщуються у пам'яті* побайтово в комірках з послідовно зростаючими адресами. Вони вибираються з пам'яті словами (по два байти) і завантажуються у чергу команд блока сполучення із системною шиною МП.

Перший байт команди завжди містить код операції, причому в окремих командах його *нульовий розряд* несе інформацію про довжину операндів (байт чи слово), а *перший* розряд задає тип джерела і/чи приймача інформації.

Другий байт у більшій частині команд може бути трьох типів:

–постбайт, що визначає, звідки брати операнд (операнди) і/чи куди направляти результат операції [1]. У деяких командах *розряди 5…3 постбайта* (поле reg) використовуються для розширення коду операції команди;

–адреса порту введення/виведення, що у двобайтових командах задає пряму адресу порту у діапазоні 00H…FFH;

–8-розрядний зсув, що в командах умовної передачі керування визначає відносну адресу переходу.

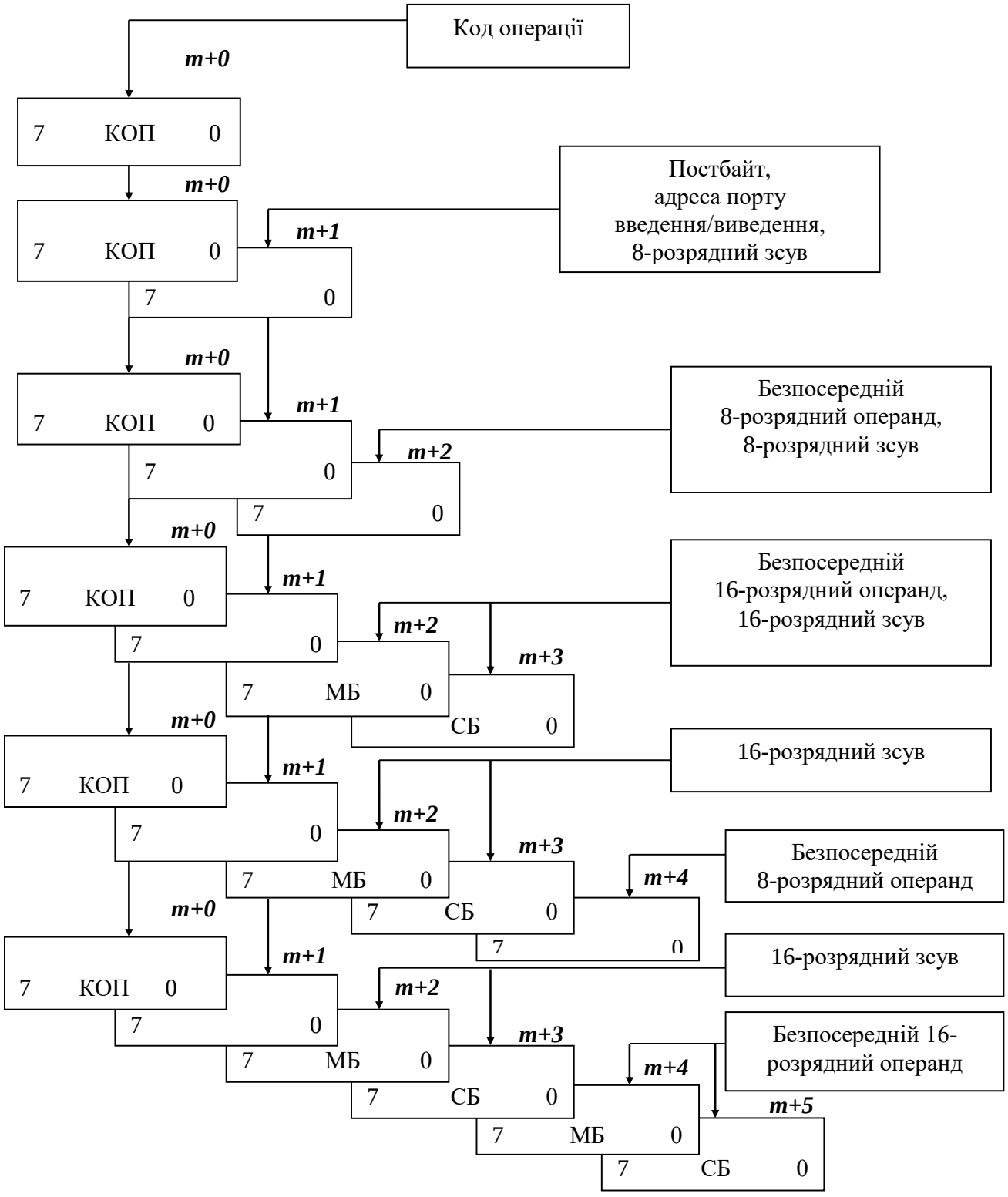


Рис. 5. Формати команд 16-розрядного МП

Наступні байти в довгих командах містять 8-розрядний чи 16-розрядний зсув і/чи безпосередній операнд, також 8-розрядний чи 16-розрядний.

Якщо в командах є зсув і безпосередній операнд, то спочатку розміщується зсув, а потім операнд. Для 16-розрядного зсуву, і для 16-розрядного операнда завжди *першим* розміщується *молодший байт*, а другим – старший (адреса старшого байта на одиницю більше адреси молодшого). *8-розрядний зсув* під час формування ефективної адреси *розширюється* до 16-розрядного *значенням старшого (знакового) розряду*. Наприклад, якщо задано байт зсуву 7EH = 01111110B, то він розширюється до 007EH. Якщо задано байт A3H = 10100011B, то він розширюється до FFA3H. *Зсув сприймається* МП як ціле двійкове число зі знаком у додатковому коді, що перебуває у діапазоні: –128…+127 для 8-розрядного чи: –32768…+32767 для 16-розрядного зсуву.

2.6.2. Команди 8-розрядного мікроконтролера сімейства AVR

Більшість МК сімейства AVR мають *14 форматів (типів) базового набору* команд, наведених на рис. 6.

На рис. 6 використано такі *умовні позначення*: КОП – код операції; K – константа даних; k – адресна константа; b – номер біта в регістрі введення/виведення або регістрового файлу (регістр загального призначення) – 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 крім 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 двоопернадні команди, в яких один операнд: K6 = 6 біт (K = 0…63) входить у 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 двоопернадні команди, наприклад, SBCI Rd*, K; ORI Rd*, K, в яких Rd* – один із шістнадцяти регістрів загального призначення (R16…R31), а другий K – 8-бітна константа (безпосередній операнд): $0 \le K \le 255$.

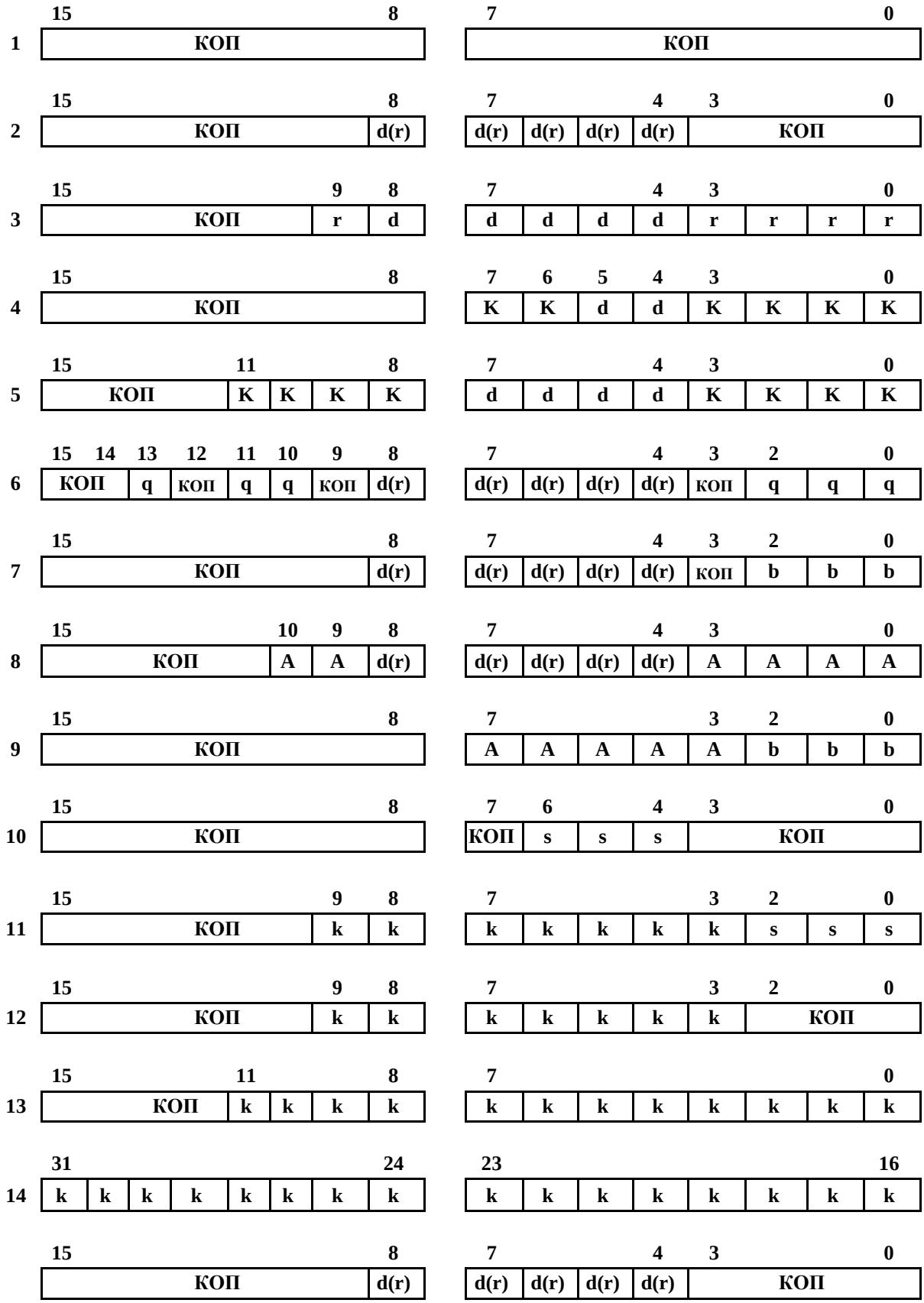


Рис. 6. Формати базового набору команд МК сім'ї AVR

До шостого типу належать команди, в яких *один з двох* операндів може бути *регістром загального призначення* (Rd – приймач, Rr – джерело), а *другий* міститься у статичній пам'яті даних 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 регістра загального призначення. 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 регістра загального призначення.

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

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

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 відносного переходу* залежно 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 ($-64 \le k \le 63$) під час переходу додається до адреси наступної команди.

Дванадцятий тип мають команди *умовного відносного переходу* залежно від значень окремих прапорців регістра стану (*прапорці адресуються неявно*). Наприклад, команди BREQ k; BRCC k і т. ін., де k = 7 біт ($-64 \le k \le 63$) під час переходу 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 *регістр* загального призначення (Rd – приймач, Rr – джерело), а *другий* міститься у *статичній* пам'яті даних (регістри загального призначення, регістри введення/виведення та статичний ОЗП). Оскільки k = 16 біт, то *кількість комірок* статичної пам'яті даних становитиме $2^{16} = 65536$.

2.7. Формати даних

2.7.1. Загальна характеристика форматів даних

Під час роботи з МП або МК необхідно знати не тільки формати команд, які керують роботою МП/МК, але й *формати (типи) даних (операндів*), що беруть участь у виконанні тієї або іншої команди (операції).

2.7.2. Формати (типи) даних 8-розрядного мікропроцесора/мікроконтролера

Формати даних 8-розрядного МП/МК наведено на рис. 7.

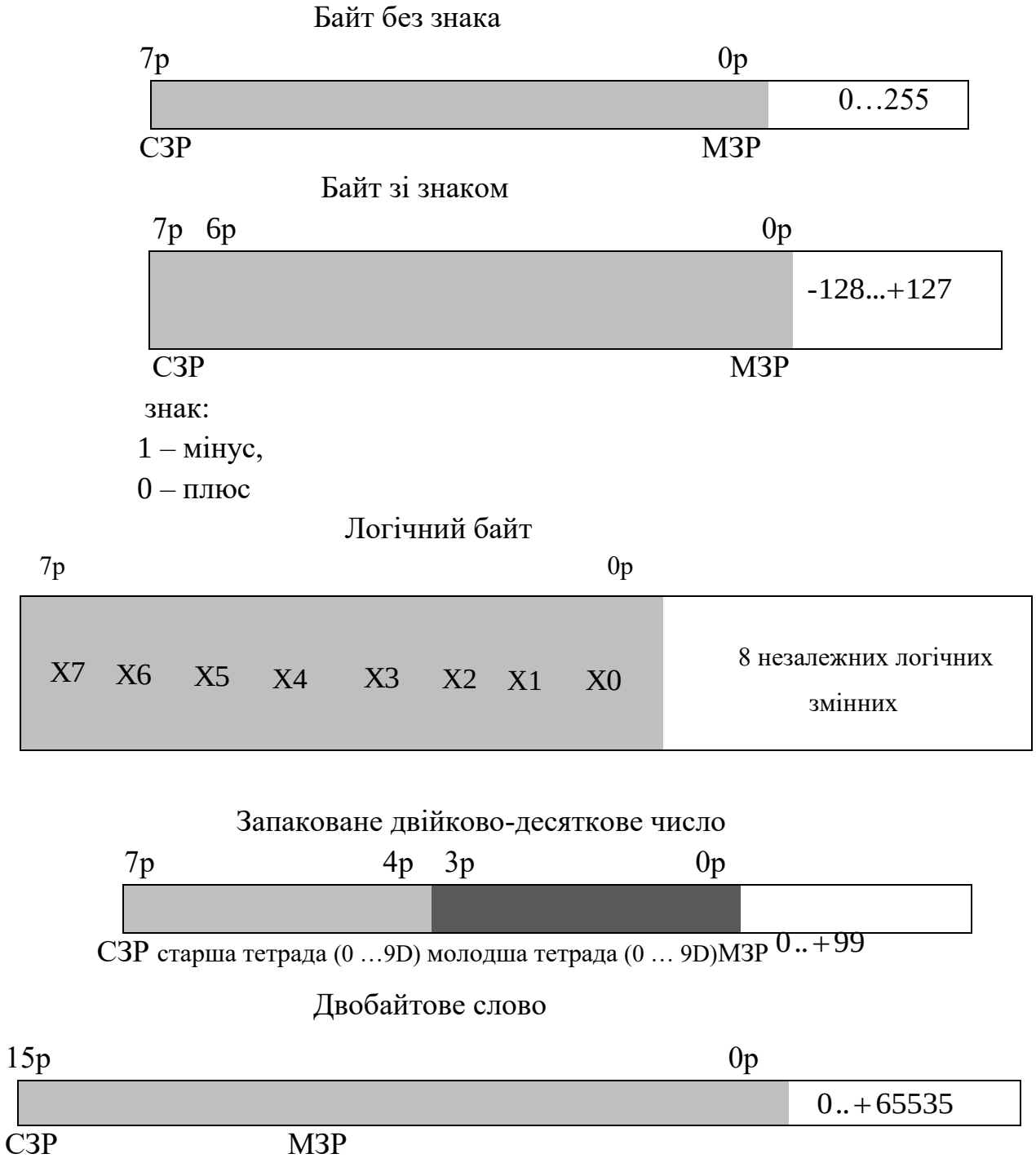


Рис. 7. Формати (типи) даних 8-розрядного МП/МК

2.7.3. Формати (типи) даних 16-розрядного мікропроцесора

Формати даних 16-розрядного МП наведено на рис. 8.

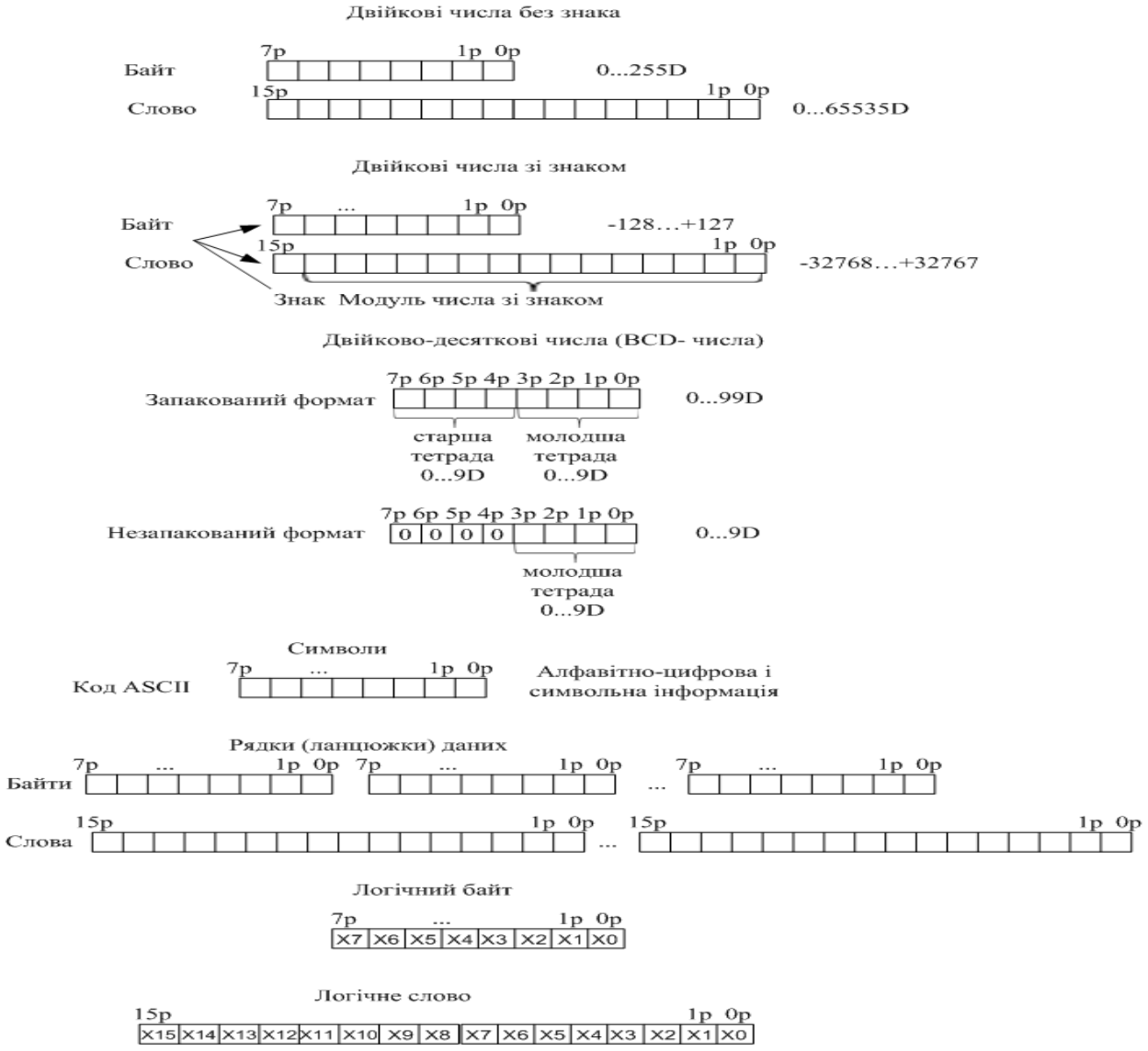


Рис. 8. Типи та формати даних МП i8086

2.8. Довжина команд у байтах і їх розміщення в пам'яті програм

Довжина команд 16-розрядного МП i8086 складається *від одного до шести байт* (рис. 5), які розміщуються в пам'яті так само, як у 8-розрядному МП. Наприклад, байти найдовшої команди (*6 байт*) розміщуються *в такому порядку*: 1-й байт – код операції; 2-й байт – постбайт (використовується для адресації операндів); 3-й байт – молодший байт зсуву, який бере участь в обчисленні ефективної адреси операнда, яка формується згідно з форматом постбайта; 4-й байт – старший байт зсуву; 5-й байт – молодший байт безпосереднього операнда; 6-й байт – старший байт безпосереднього операнда.

Відповідно до наведеного рис. 6 команди МК сімейства AVR мають довжину –

одне слово (2 байти), за винятком 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 МК на основі прочитаного КОП, 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 у разі кожного звернення до ЗП.

2.9. Вплив команд на прапорці

Як відзначалося раніше, *одним з основних* регістрів МП/МК є *регістр прапорців* (ознак). Прапорці регістра ознак встановлюються під час виконання ряду команд МП/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. Якщо команда не змінює прапорець, то ставлять *риску*, якщо змінює, то ставлять *плюс* або *х*, а якщо встановлює у будь-який стан, то пишуть *нуль (0) або одиницю (1*). Як правило, команди пересилань не змінюють прапорці, окрім команд, які пересилають дані у регістр прапорців. *Частіше* прапорці *змінюють* арифметичні, логічні команди і команди порівняння.

Для AVR-мікроконтролерів 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 (лекція 6).

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.

Вплив команд на прапорці *у 16-му* МП i8086 показано у [1].

2.10. Час виконання команд

В одній з колонок таблиць, що описують систему команд, записують *кількість тактів* тактової частоти МП/МК (f_T), що необхідно для виконання команди. Знаючи число тактів (n_T) і значення тактової частоти можна визначити час виконання команди $t_K = n_T/f_T$.

В AVR-МК окремі команди виконуються за 1, 2, 3 або 4 такти. *Тривалість одного такту* дорівнює одному *періоду* тактової частоти f_{BQ}. Для шістнадцятирозрядного МП, наприклад, i8086 одна з колонок таблиць, в яких приводяться команди та їх характеристики, містить інформацію про час виконання команд у кількості тактів [1]. *В цій колонці може стояти* або *конкретне число*, що відображає час виконання даної команди в тактах, *або* приводиться *вираз*: $n + T_{BA}$, де n – конкретне значення; T_{BA} – додаткове число тактів, необхідне для

обчислення ефективної (виконавчої) адреси операнда в пам’яті. Це значення визначається за відомим способом адресації операнда, що застосовується у відповідній команді [1].

Контрольні запитання та завдання

1. Назвіть основні етапи створення керувальної програми.
2. Наведіть та опишіть програмну модель шістнадцятирозрядного МП.
3. Що являє собою регістр прапорців?
4. Назвіть регістри загального призначення МП i8086.
5. Назвіть індексні регістри МП i8086.
6. Назвіть сегментні регістри МП i8086.
7. Для чого використовується регістр SP?
8. Що показує прапорець OF?
9. Яку функцію виконує покажчик команд IP?
10. Що називається логічною базовою адресою сегмента?
11. Як із пари: початок сегмента – зміщення у сегменті утворити фізичну адресу комірки пам’яті?
12. Дайте визначення мові асемблера.
13. Назвіть та дайте визначення основних керувальних програм, які використовуються у разі створення програм мовою асемблера.
14. Поясніть наступні терміни: мнемоніка команди та мнемокод; код операції команди (КОП); операнди та коментар.
15. Наведіть та поясніть формати команд 8-х МП/МК і 16-го МП.
16. Поясніть, що таке прапорці та яку роль вони відіграють в роботі МПС.
17. Як можна обчислити час виконання окремих команд в МК та МП?
18. Що таке підпрограми і як вони використовуються під час створення асемблерних програм? Наведіть приклад.
19. Наведіть та опишіть програмну модель AVR-мікроконтролерів.
20. Наведіть та поясніть формати даних (операндів), що беруть участь у виконанні тієї або іншої команди (операції).
21. Як визначається довжина та час виконання команд AVR-МК.

ЛЕКЦІЯ 6. СПОСОБИ АДРЕСАЦІЇ ОПЕРАНДІВ МП/МК. КОМАНДИ МІКРОПРОЦЕСОРІВ/МІКРОКОНТРОЛЕРІВ

1. СПОСОБИ АДРЕСАЦІЇ ОПЕРАНДІВ МП/МК

1.1. Загальні відомості про способи адресації

Тип звернення (адресації) до операндів (даних, що беруть участь в операції) називають *способом адресації*. До способів адресації операндів *також відносять* те, як в командах передачі керування дається *вказівка на адресу переходу*.

В шістнадцятирозрядному МП, наприклад, і8086 *застосовуються наступні* способи адресації операндів: неявна, регістрова, безпосередня, пряма, непряма, базова, індексна, базово-індексна, стекова, відносна, адресація рядків та адресація портів введення/виведення [1].

1.2. Восьмирозрядний МК сім'ї AVR

1.2.1. Загальні відомості про способи адресації AVR-мікроконтролерів

AVR-мікроконтролери підтримують наведені *нижче способи* адресації операндів: неявна; безпосередня; пряма та непряма адресації.

Деякі способи адресації *мають кілька різновидів* в залежності від того, до якої області пам'яті виконується звернення під час прямої адресації, або які додаткові дії виконуються над індексним регістром під час непрямої адресації.

1.2.2. Неявна адресація

У разі *неявної* адресації в команді відсутня адреса операнда в явному вигляді. Інформацію про потрібну адресу операнда МК отримує з КОП 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.

1.2.3. Безпосередня адресація

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

1.2.4. Пряма адресація

У разі прямої 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.

1.2.5. Пряма адресація одного регістра загального призначення

Цей спосіб адресації використовується в командах, що *оперують з одним* РЗП. У цьому разі адреса регістра-операнда (його номер) міститься в розрядах 8...4 (5 біт) машинного коду команди (рис. 1).

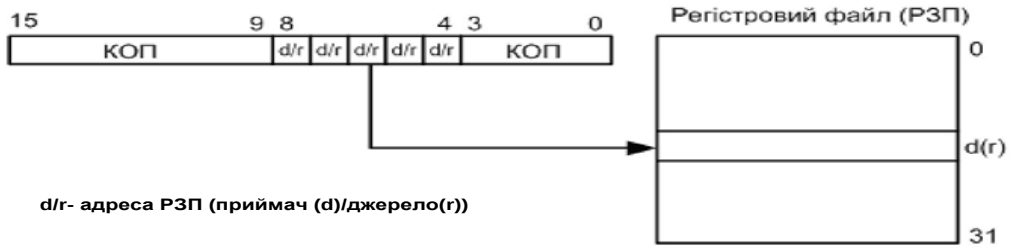


Рис. 1. Пряма адресація одного РЗП

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.

1.2.6. Пряма адресація двох регістрів загального призначення

Цей спосіб адресації використовується в командах, що *оперують одночасно двома* РЗП. У цьому разі 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. 2).

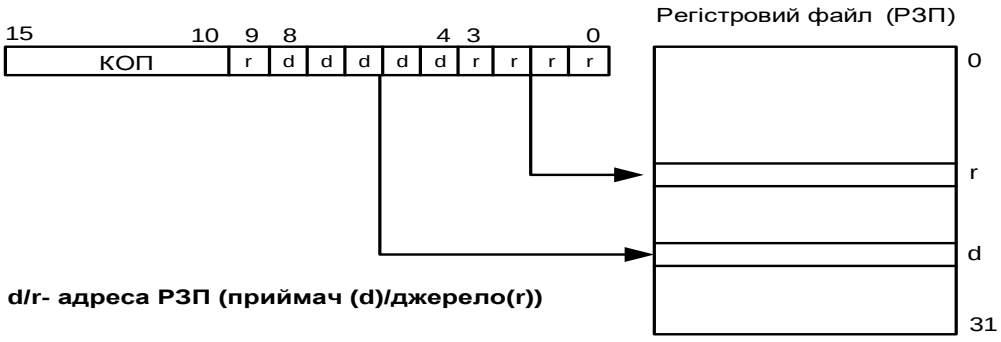


Рис. 2. Пряма адресація двох регістрів загального призначення

До команд, що використовують цей спосіб адресації, *належить* команди *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.

1.2.7. Пряма адресація РВВ

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

Рис. 3. Пряма адресація РВВ

1.2.8. Пряма адресація СПД

Цей спосіб *використовується* під час звернення до всього адресного простору СПД, яка включає: РЗП, РВВ та СОЗП (рис. 4).

Рис. 4. Пряма адресація СПД

За адресами $00...$1F у СПД розташовано *РЗП*, *за адресами* $20...$5F – *основні РВВ*, а *за адресами* $60...$FF – *додаткові 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 КОП та адреса РЗП (у розрядах з 8-го по 4-й). У *другому слові* міститься адреса комірки пам'яті, до якої відбувається звернення.

1.2.9. Непряма адресація

У разі непрямої адресації адреса комірки СПД 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 та непряма адресація з постінкрементом.

1.2.9.1. Проста непряма адресація

У разі використання простої непрямої адресації *звернення* виконується *до СПД* за адресою, що міститься в одному з індексних регістрів: X, Y або Z (рис. 5). Жодних дій із вмістом індексного регістра у цьому разі не виконується.

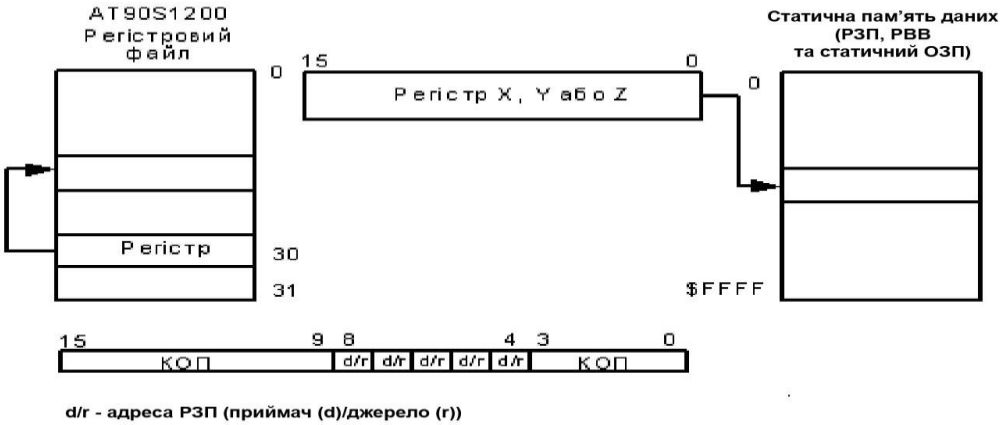


Рис. 5. Проста непряма адресація

МК *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 у СПД). Адреса РЗП міститься в розрядах 8...4 машинного коду команди.

1.2.9.2. Відносна непряма адресація

У командах відносної непрямої адресації адреса комірки СПД, до якої виконується звернення, обчислюється *додаванням вмісту індексного* регістра Y, або Z і *константи*, що міститься в машинному коді команді (рис. 6).

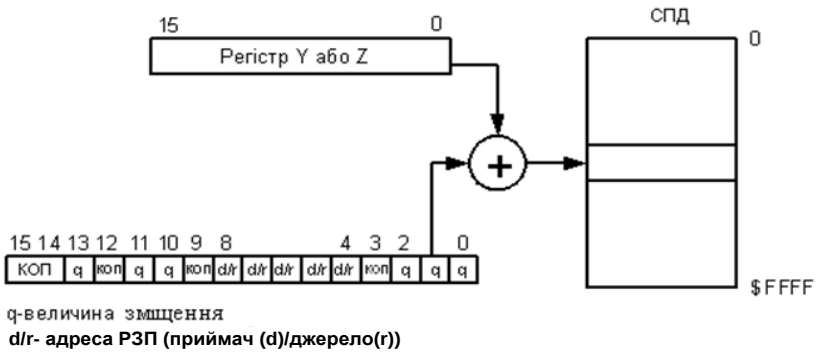


Рис. 6. Відносна непряма адресація

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 РЗП 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 ($0 \le q \le 63$).

1.2.9.3. Непряма адресація з попереднім декрементом (переддекрементом)

Під час виконання команд непрямої адресації з переддекрементом вміст індексного регістра *спочатку зменшується на одиницю*, а потім виконується звернення за отриманою адресою (рис. 7).

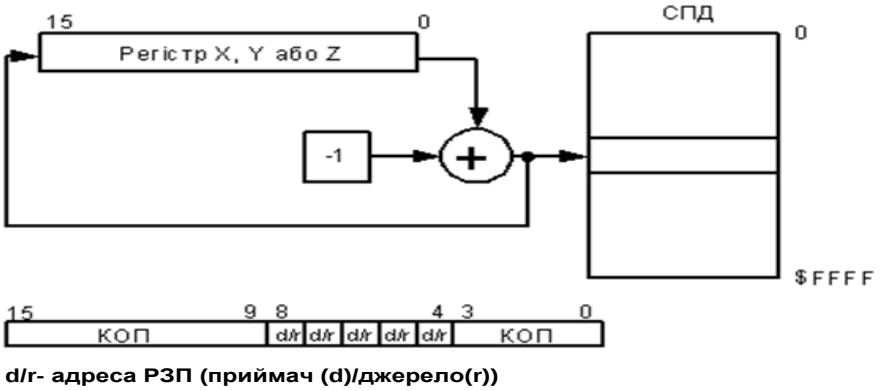


Рис. 7. Непряма адресація з переддекрементом

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

1.2.9.4. Непряма адресація з постінкрементом

Під час виконання команд непрямої адресації з постінкрементом (наступним інкрементом) *після звернення за адресою*, що міститься в індексному регістрі, *вміст* індексного регістра *збільшується на одиницю* (рис. 8).

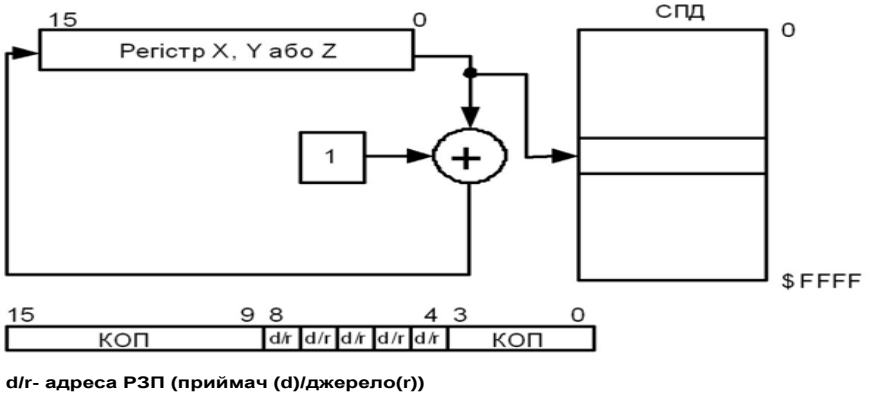


Рис. 8. Непряма адресація з постінкрементом

МК 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 РЗП міститься в розрядах 8...4 машинного коду команди, а регістри X/Y/Z *адресуються неявно*.

1.2.9.5. Непряма адресація пам’яті програм

Такий спосіб адресації використовується *в командах* IJMP, ICALL (рис. 9).

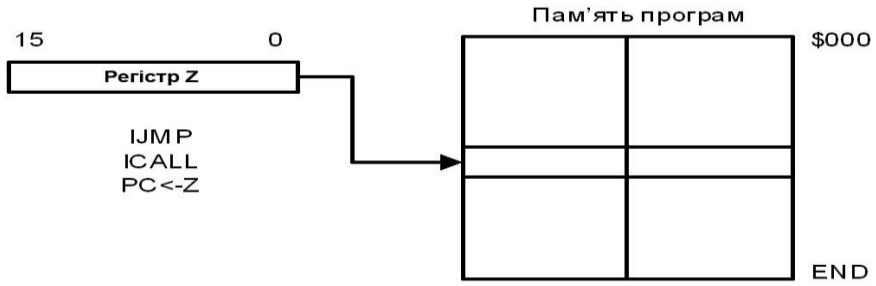


Рис. 9. Непряма адресація пам’яті програм

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. На відміну від команд відносного переходу ця команда *має більший діапазон переходів*. 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.

1.2.9.6. Непряма адресація констант в пам’яті програм

Цей спосіб адресації *використовується в команді* LPM, яка завантажує один байт із пам’яті програм в регістр загального призначення R0 (рис. 10).

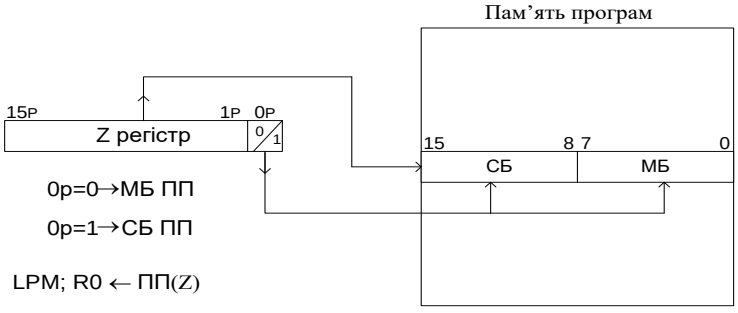


Рис. 10. Непряма адресація констант в пам’яті програм

Адреса комірки пам’яті, до якої відбувається звернення, міститься в індексному регістрі Z. У цьому разі *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).

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.

1.2.10. Відносна адресація пам'яті програм

Цей спосіб адресації використовується *в командах* RJMP, RCALL (рис. 11).

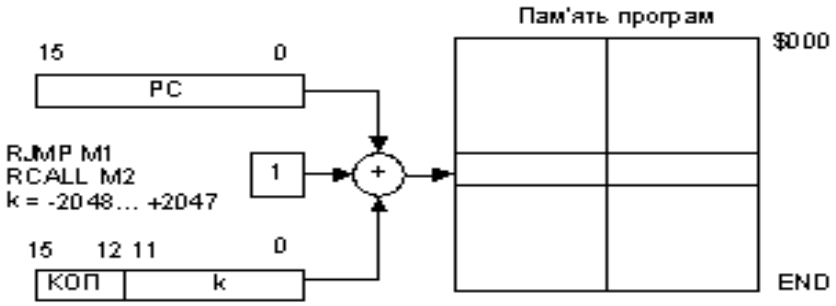


Рис. 11. Відносна адресація пам'яті програм

У разі виконання команди до поточного вмісту лічильника команд (адреси наступної команди) додається *дванадцятирозрядне число зі знаком* – k, яке представлено *у додатковому* ДВК. 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).

В 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.

2. КОМАНДИ МІКРОПРОЦЕСОРІВ/МІКРОКОНТРОЛЕРІВ

2.1. Команди шістнадцятирозрядного мікропроцесора

Всі команди шістнадцятирозрядного МП, наприклад, i8086, залежно від їх функціонального призначення, умовно *поділено на такі групи*: пересилання; робота зі стеком; введення/виведення; обмін; трансляція; завантаження; пересилання прапорців; арифметичні; корекція; порівняння; логічні; зсув; обробка рядків; умовні і безумовні переходи; організація циклів; виклик і повернення з підпрограм; програмні переривання та керування мікропроцесором [1].

2.2. Команди восьмирозрядного МК сімейства AVR

2.2.1. Базовий набір команд

AVR-мікроконтролери мають RISC-архітектуру. Практично всі команди *займають одну комірку* пам'яті, розрядність якої дорівнює 16. Виняток становлять команди, в яких одним з операндів є 16-розрядна адреса СПД (РЗП, РВВ та СОЗП). Більшість команд виконується *за один машинний цикл* (1 такт).

Всі команди *базового набору* можна розділити на *декілька груп*: логічні операції; арифметичні операцій та команди зсуву; операції з бітами; команди пересилання даних; команди передачі керування та команди керування системою. Нижче скорочено описано ці команди (табл. 1). Детальнішу інформацію наведено у [2].

В табл. 1 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 РВВ; P* – адреса РВВ, 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 РЗП/РВВ (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).

2.2.2. Нові команди AVR-мікроконтролерів

Архітектура AVR-мікроконтролерів постійно *оновлювалася*. Це оновлення стосується як структури МК, так і його системи команд. Кожне сімейство, послідовно успадковувало набір команд попереднього сімейства. Нижче наведено опис команд, які останнім часом з'явилися в AVR-мікроконтролерах: Mega та XMega (табл. 2) [2].

Таблиця 1. Базовий набір команд AVR-мікроконтролерів

№ з/п	Мнемоніка	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. 1

№	Мнемо-ніка	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. 1

№	Мнемоніка	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 постінкрементом	(Z) ← Rr, Z ← Z + 1	-	2	2
24	ST	-Z, Rr	Непряме збереження з переддекрементом	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 біт регістра	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. 1

№ з/п	Мнемоніка	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 МК							
1	NOP		Порожня операція	__	–	1	1
2	SLEEP		Переведення у режим сну	__	–	1	3
3	WDR		Скидання вартового таймера	__	–	1	1

Таблиця 2. Нові команди МК AVR

Команда	Операнди	Опис	Дія	Прапорці	Такти
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
MUL	Rd, Rr	Множення беззнакових чисел	R1:R0 ← RdxRr	Z,C	2

Продовження табл. 2

Команда	**Операнди**	**Опис**	**Дія**	**Прапорці**	**Такти**
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
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
BREAK		Зупинка процесора після виконання команди	PC ← PC + 1, зупинка виконання програми	–	1
EIJMP		Безумовний непрямий перехід у пам’яті програм ємністю 4 Мслів	PC← (EIND:Z)	–	2

Продовження табл. 2

Команда	Операнди	Опис	Дія	Прапорці	Такти
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
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
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

Закінчення табл. 2

Команда	Операнди	Опис	Дія	Прапорці	Такти
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

Контрольні запитання та завдання

1. Поясніть, що означає поняття: способ адресації операндів.
2. Дайте визначення та наведіть приклад команди з неявною адресацією.
3. Дайте визначення та наведіть приклад команди з безпосередньою адресацією.
4. Дайте визначення та наведіть приклад команд з прямою адресацією.
5. Дайте визначення та наведіть приклади команд з непрямою адресацією.
6. Дайте визначення та наведіть приклади команд з відносною адресацією.
7. Як компілятор обчислює величину зміщення відносно адреси наступної команди в програмі, в якій присутні мітки. Наведіть приклад.
8. Назвіть та опишіть логічні команди МП та МК.
9. Поясніть використання логічних команд для: встановлення в одиницю, скидання в нуль, інвертування потрібних бітів операнда-джерела, у цьому разі інші біти повинні не змінюватись.
10. Назвіть та опишіть команди роботи з бітами AVR- мікроконтролерів.
11. Які чотири основні булеві функції реалізуються за допомогою логічних команд?
12. Опишіть особливості виконання зсувів: логічного, циклічного, арифметичного.
13. Назвіть основні групи команд з базового набору AVR-мікроконтролерів. Дайте коротку характеристику кожній з груп.
14. Що спільного та які вімінності у разі виконання команд JMP та CALL?
15. Опишіть призначення та відмінності команд RET та RETI.
16. Поясніть особливості виконання команди RJMP.
17. Як компілятор використовує мітки в командах умовних переходів у разі формування машинних кодів команд?
18. Чим відрізняється алгоритми виконання команд арифметичних та логічних зсувів? Наведіть приклади їх використання.
19. Який діапазон десяткових чисел відображає 8-розрядне число зі знаком?

20. Чим відрізняються мнемоніка та мнемокод команд?
21. Для чого використовується регістр PC/IP?
22. Опишіть роботу конвеєра у разі виконання програми у AVR-мікроконтролерах.
23. Для чого використовуються регістри-покажчики X,Y і Z?

ЛЕКЦІЯ 7. ОРГАНІЗАЦІЯ ПІДСИСТЕМИ ПЕРЕРИВАНЬ У МПС

1. ОРГАНІЗАЦІЯ ПІДСИСТЕМИ ПЕРЕРИВАНЬ У МПС

1.1. Загальні відомості про підсистему переривань

У лекції 2 було надано коротку характеристику підсистемі переривань у МПС. У цій лекції це питання буде розглянуто більш детально.

Підсистема переривань характеризуються такими *властивостями*:

- кількістю джерел (рівнів/векторів/адрес) переривань;
- видом переривань;
- можливістю програмного маскування і керування пріоритетами;
- порядком обслуговування та обчисленням адреси підпрограми обробки переривання і т. ін.

1.2. Організація підсистеми переривань мікропроцесора

1.2.1. Види переривань

Переривання *поділяються* на внутрішні або зовнішні.

Розглянемо організацію підсистеми переривань у МПС на прикладі МП i8086. Мікропроцесор *має векторну (адресну*) підсистему переривань із зовнішніми та внутрішніми джерелами запитів. Кожне джерело *має свій «тип*» – *номер* «входу» у таблицю адрес обробників переривань (рис. 1), за яким МП знаходить відповідну підпрограму обслуговування переривання.

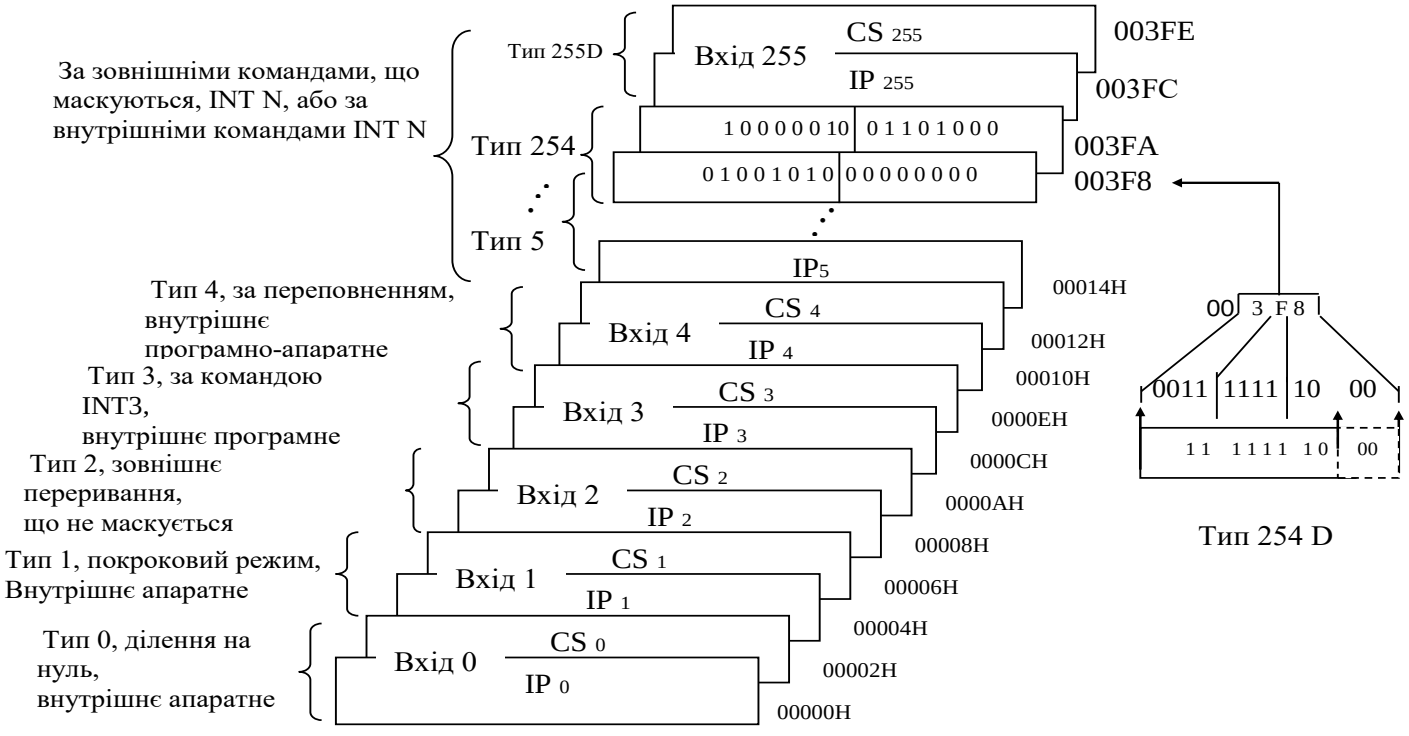


Рис. 1. Організація переривань у мікропроцесорі i8086

Внутрішні джерела переривань поділяються на апаратні, програмно-апаратні та програмні. До *апаратних* відносяться переривання від помилки ділення на нуль (тип 0) і відлагоджувальні (тип 1). Запит *типу 0* генерується апаратно, якщо під час виконання команд DIV, IDIV дільник дорівнює нулю. Запит *типу 1* виникає після виконання чергової команди за умови виконання програми у покроковому режимі (прапорець TF = 1). До *програмно-апаратного* належить переривання за переповненням (*тип 4*), що настає, якщо виконується команда INTO і прапорець переповнення розрядної сітки після виконання арифметичних команд над числами зі знаком: OF = 1. *Програмні переривання* виникають під час виконання *однобайтної* команди INT3 (тип 3) чи *двобайтової* команди INT type, що імітує запит переривання будь-якого типу. Програмні переривання *не маскуються*.

Зовнішні запити переривання надходять за двома входами МП: NMI та INT. Немасковані запити NMI викликають переривання *типу 2* і сприймаються за фронтом (зростаючим рівнем) сигналу на вході NMI.

Запити, *що маскуються*, надходять за входом INT і сприймаються процесором тільки тоді, коли переривання дозволені (не замасковані) – *прапорець* IF = 1. В іншому випадку, коли IF = 0, *запит відкидається*. Варто звернути увагу, що під час запиту за входом INT *не обов'язково* буде обслуговуватися переривання типу 3, як у разі виконання команди INT3. Зовнішній пристрій, що виставив запит, повинен за СШД передати у МП *байт*, що дозволяє ідентифікувати тип переривання.

У разі одночасного надходження декількох запитів діє *система пріоритетів*, що у МП i8086 виглядає так (від вищого рівня до нижчого): переривання під час ділення на нуль (вищий рівень); переривання за командами INT3, INT type, INTO; переривання за входом NMI; переривання за входом INT; переривання за прапорцем TF (покроковий режим виконання програми).

Обробка будь-якого прийнятого запиту *починається* із запам'ятовування вмісту регістра прапорців RF у стеку. Прапорці IF і TF *скидаються* в нуль, забороняючи наступні зовнішні переривання і покроковий режим роботи. У стек *пересилається* вміст регістрів CS і IP (запам'ятовується місце повернення в основну програму). За ідентифікованим типом переривання з *таблиці витягається* адреса входу у підпрограму обробки запиту: *перше слово* пересилається у регістр IP, *друге* – у регістр CS, після чого підпрограма виконується. Завершується вона *командою* IRET (повернення з переривання). У цьому разі зі стека у зворотному порядку *відновлюється* вміст регістрів CS, IP та RF і керування передається перерваній програмі. У цій програмі *необхідно відновити* прапорці IF = 1 і TF = 1, що дозволить обробку нових зовнішніх запитів і покроковий режим роботи.

1.2.2. Особливості обробки зовнішніх переривань

Джерелами зовнішніх переривань у МПС є пристрої введення/виведення; пристрої, що задають час; аварійні ситуації (наприклад, аварія живлення) тощо.

Зовнішні пристрої (ЗВПР) *можуть видавати сигнал* на обслуговування переривань у будь-який момент часу, подаючи сигнал рівня «логічна одиниця» на *вхід INT* мікропроцесора (апаратний запит переривань).

Мікропроцесор, *отримавши* сигнал запиту *INT* від ЗВПР, *передає* системною шиною керування *сигнал* $\overline{INTA}$ – підтвердження переривання, що інформує зовнішній пристрій про те, що МП припинив виконання основної програми і перейшов до режиму обслуговування переривання.

Після завершення виконання поточної команди основної програми, *МП приймає* від ЗВПР системною шиною даних *восьмирозрядний номер* – логічну адресу переходу на виконання підпрограми обслуговування переривання. Мікропроцесор *зберігає* поточний стан виконуваної програми у стеку та переходить до підпрограми обслуговування переривання. *Завершується* підпрограма обслуговування переривання *командою IRET*, що відновлює стан перерваної основної програми зі стека та продовжує її виконання.

1.3. Організація підсистеми переривань мікроконтролера

1.3.1. Основні властивості підсистеми переривань мікроконтролера

Розглянемо, *як приклад*, основні властивості підсистеми переривань МК *типу* AVR. У разі виникнення переривання МК *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 МП, то кількість переривань залежить від конкретної моделі МК.

AVR-мікроконтролери мають *багаторівневу систему пріоритетних* переривань [2]. Молодші адреси пам’яті програм, починаючи з адреси $0001 або $0002 відведено під *таблицю векторів* переривань. *Вектор переривання* – це закріплений за пристроєм номер, який ідентифікує відповідний обробник переривань. В цій таблиці кожному перериванню *відповідає своя адреса*, яка у разі виникнення 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 МК 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 МК 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 [2].

Розмір таблиці векторів переривань залежить від моделі МК і становить від 16 до більш ніж 56 векторів. *Розподіл адрес* таблиці векторів переривань для різних МК сімейства наведено у [2]. *У разі розміщення* векторів переривань в області завантажувача до значень, наведених у таблицях, слід додати значення початкової адреси області завантажувача.

Якщо переривання в роботі МК *не передбачаються*, то на місці таблиці векторів переривань може бути розміщена частина основної програми.

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 *периферійних* модулів [2].

1.3.2. Обробка переривань

Обробка переривань здійснюється наступним чином:

- у разі виконання 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) вміст лічильника команд зберігається у стеку, після чого у лічильник команд завантажується адреса вектора відповідного переривання. У цьому разі 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, у цьому разі 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. У разі виклику підпрограм обробки переривань вміст регістра стану

SREG у стеку не зберігається. Якщо це необхідно, користувач повинен самостійно запам'ятовувати вміст цього регістра під час входу в підпрограму обробки переривання і відновлювати його значення перед викликом команди RETI.

Для переривань, які викликано *динамічними подіями* (наприклад, для переривання, яке генерується під час 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 доти наявна подія, 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 МК 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.

1.3.3. Зовнішні переривання

В AVR-мікроконтролерах, наприклад, сімейства Mega є два різновиди зовнішніх переривань. Переривання *першого типу* (так звані «звичайні») генеруються у разі появи на вході зовнішнього переривання заданого сигналу. Ці переривання присутні у всіх МК сімейства (конкретне число таких переривань залежить від моделі). Переривання *другого типу* генеруються у разі зміни стану певних виводів МК.

Зовнішні переривання *за низьким рівнем* і переривання *за зміною стану*

виводів реєструються асинхронно, тобто не вимагають наявності тактового сигналу $f_{clkI/O}$. Відповідно, ці переривання можуть використовуватися *для виведення* МК зі «сплячих» режимів.

Для дозволу/заборони та індикації настання зовнішніх переривань використовуються РВВ мікроконтролерів. Їх формати та описання окремих розрядів наведено у [2].

Всі зовнішні переривання генеруються навіть у тому випадку, якщо відповідні *виводи сконфігуровані як виходи*. Ця особливість МК дозволяє генерувати переривання програмно.

1.3.4. Внутрішні переривання

Крім зовнішніх, більшість переривань у AVR-мікроконтролерах 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. Окрім цього, для дозволу виконання відповідної підпрограми обробки переривання, треба встановити в одиницю прапорець глобального дозволу переривань I регістра SREG. Запит на переривання генерується в тому випадку, якщо в будь-який момент після його дозволу встановленням відповідних прапорців виявляється відповідна йому умова. У кожного переривання є *окремий вектор* переривання [2].

1.3.5. Особливості використання модуля переривань у МК 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 [9].

1.4. Маскування та призначення пріоритетів переривань

1.4.1. Маскування переривань

Переривання МП/МК можуть *програмно маскуватися*. Для цього використовуються *прапорці* (тригери), які приймають два значення: нуль, переривання заборонено; одиниця, переривання дозволено.

Існують *два види прапорців маскування* переривань: прапорець *глобального* (загального) маскування всіх переривань (як правило, знаходиться в регістрі прапорців); прапорці маскування переривань від окремих *периферійних модулів* МК.

1.4.2. Призначення пріоритетів переривань

У разі одночасного надходження переривань від декількох джерел може діяти *система пріоритетів*, особливості якої залежать від типу конкретного МП/МК. Загальний підхід призначення пріоритетів переривань полягає в тому, що, як правило, існують *дві ступені розподілу* пріоритетів.

Перша ступінь реалізована апаратно і дозволяє за умови одночасного запиту від декількох джерел обрати одне з вищим пріоритетом.

Друга ступінь реалізується програмно. У цьому разі для кожного джерела переривання у МК використовується окремий прапорець (тригер), який програмно або встановлюється в одиницю, що задає цьому перериванню високий пріоритет, або складається в нуль, що задає низький пріоритет. Очевидно, що переривання із вищим програмно заданим пріоритетом перерве підпрограму обслуговування переривання із нижчим програмно встановленим пріоритетом.

Для МК сімейства МК51, наприклад AT89C51, це питання розглянуте у [1]. Пріоритети зовнішніх переривань для МПС на МП типу i8086, можуть задаватися у разі програмування *контролера переривань*.

1.4.3. Визначення адреси підпрограми обробки переривання

Для кожного можливого переривання під час проектування МПС розробляється *відповідна підпрограма*, яка зберігається у пам’яті МПС. Для того, щоб викликати потрібну підпрограму обробки переривання треба визначити її адресу. Як це зробити залежить від типу конкретного МП/МК, який використовується у МПС. Визначення адрес для МП типу i8086 описано у п. 1.2.1.

Для *визначення адрес* підпрограми обробки переривання у МК використовують *два підходи*:

- всі підпрограми обробки переривань починаються *з однієї адреси*, а відповідь на запитання, яке конкретне переривання потрібно обробити, дається за допомогою аналізування підпрограмою прапорців наявності переривань;
- за кожною підпрограмою обробки переривань *закріплюється конкретна адреса* пам’яті, яка вказується в описанні архітектури МК, наприклад AVR-МК, підсистема переривань яких описана вище.

Контрольні запитання та завдання

1. Для чого призначена підсистема переривань?
2. Назвіть основні види переривань.
3. Як відбувається маскування переривань?

4. Назвіть основні джерела переривань.
5. Які регістри потрібні для програмування та керування роботою підсистеми переривань?
6. Якими основними командами реалізується обробка переривань?
7. Як виконується команда LCALL?
8. Як виконується команда RETI?
9. Як виконується команда RET?
10. Як відбувається повернення з підпрограми обробки переривання до основної програми?
11. Яку роль відіграє стек під час проектування підсистеми переривань?
12. Як у МП i8086 обчислюється адреса підпрограми обробки переривання?
13. Де зберігаються підпрограми обробки переривань у МК?
14. Яку двадцятирозрядну фізичну адресу пам’яті у шістнадцятковому коді буде сформовано у МП i8086 під час отримання, наприклад, запиту зовнішнього переривання типу 165?
15. Опишіть послідовність обробки переривань у мікроконтролері.
16. Чому під час виходу із підпрограми обробки зовнішнього переривання у МК треба використовувати команду RETI, а не RET?
17. Назвіть джерела переривань і адреси підпрограм, що їх обслуговують, в AVR-мікроконтролерах.

ЛЕКЦІЯ 8. ФОРМУВАННЯ ІНТЕРВАЛІВ ЧАСУ ТА ПІДРАХУНОК ЗОВНІШНІХ ПОДІЙ У МПС

1. ФОРМУВАННЯ ІНТЕРВАЛІВ ЧАСУ ТА ПІДРАХУНОК ЗОВНІШНІХ ПОДІЙ У МПС

1.1. Способи формування інтервалів часу та підрахунок зовнішніх подій у МПС

У МПС часто треба формувати *інтервали часу*: часові затримки, одиночні імпульси, послідовності імпульсів і т. ін.

Для цього існують наступні *способи*: апаратний; програмний та апаратно-програмний. *Перший спосіб* реалізується використанням спеціалізованих електронних пристроїв, наприклад, ліній затримки, одновібраторів, мультивібраторів і т. ін [5]. У *другому способі*, використовуючи систему команд конкретного МП/МК і з огляду на те, що виконання кожної команди займає деякий час, можна розробити підпрограми, що формують часові затримки, одиночні імпульси, послідовності імпульсів і т. ін. *Апаратно-програмний спосіб* реалізується застосуванням програмованих таймерів (ПРТ). Окрім формування часових інтервалів, таймери можуть виконувати *підрахунок імпульсів*, що ідентифікують настання зовнішніх подій, тобто працювати *як лічильники зовнішніх подій*.

Ідея використання таймерів/лічильників (Т/Л) полягає в наступному. Таймери/лічильникі мають підсумовуючий двійковий лічильник. В режимі *лічильника* переключення таймера відбувається зовнішніми імпульсами, які ідентифікують настання зовнішніх подій. В режимі *таймера* лічильник переключається імпульсами, які формує системний тактовий генератор МПС. Цю частоту можна *програмно ділити*, тобто змінювати її період. На початку програмування пристрою за відомою тактовою частотою МК *встановлюється коефіцієнт її ділення*, що задає потрібний період імпульсів на вході Т/Л. Потім у двійковий лічильник програмно *записується початкове значення* (це може бути нульове значення). Початкове *значення обирається* таким чином, щоб *від початку лічення до переповнення* на вхід лічильника надійшла потрібна кількість імпульсів з відомим періодом. Факт *переповнення відображається* встановленням прапорця, що буде вказувати на те, що з моменту початку лічення до переповнення надійшла потрібна кількість імпульсів, тобто формується потрібний часовий інтервал. Аналогічно може *програмуватися підрахунок* потрібної кількості імпульсів, які відбражають настання зовнішніх подій.

Таймери сучасних МК можуть працювати також в режимах: захоплення, порівняння та широтно-імпульсного модулятора. В режимі *захоплення* модуль таймера МК, окрім лічильника, містить додатковий регістр, в який під час

надходження зовнішнього імпульсу на відповідний вхід МК захоплюється (пересилається у відповідний регістр) поточний стан лічильника. У разі захоплення *встановлюється прапорець* та викликається *підпрограма* обробки цього стану, якщо вона дозволена. Наявність цього режиму розширює можливості програмування часових інтервалів.

В режимі *порівняння (збігу*) модуль таймера МК окрім лічильника містить *декілька додаткових регістрів*, в які програмно записуються значення, з якими буде порівнюватися поточний стан лічильника. *В момент збігу* встановлюється прапорець та може викликатися підпрограма, якщо вона дозволена. Окрім цього для кожного каналу збігу виділяється *окремий вивід* МК, на якому в момент збігу формується відповідний сигнал.

На основі останнього режиму *на виділених в*иводах МК може формуватися послідовність прямокутних імпульсів. *Частота* цих імпульсів підтримується програмно *на постійному* рівні, але може *змінюватися їх шпаруватість*. Тобто таймер може формувати *декілька широтно-імпульсно-модульованих сигналів* (ШІМ-сигналів), які можуть використовуватися, наприклад, для керування двигунами постійного струму.

Восьми та шістнадцятирозрядні *мікропроцесори* не містять вбудованих таймерів і для формування інтервалів часу використовують зовнішні мікросхеми, наприклад, і8253 [1]. *Мікроконтролери* можуть містити декілька внутрішніх восьми та шістнадцятирозрядних таймерів [2].

1.2. Таймери мікроконтролерів

1.2.1. Загальні відомості про таймери мікроконтролерів

В цій роботі розглянуто, *як приклад*, МК сімейства AVR, які залежно від моделі, мають у своєму складі *до шести* таймерів загального призначення, з яких два – восьмирозрядні, чотири – шістнадцятирозрядні [2].

Ці таймери можуть працювати *в наступних режимах*: таймера; лічильника зовнішніх подій; захоплення; порівняння; ШІМ; вартового таймера та асинхронному режимі.

1.2.2. Попередні дільники таймерів

Таймери містять *блок попередніх дільників*, які призначено для формування тактових сигналів *для двійкового лічильника таймера*: f_{clkTn}, де n – номер таймера. Спрощену структурну схему блоку попереднього дільника таймерів, що *не мають асинхронного* режиму роботи, наведено на рис. 2.

На рис. 3 наведено структурну схему блоку попереднього дільника таймерів, що мають можливість роботи в *асинхронному* режимі в якості «годинника реального часу».

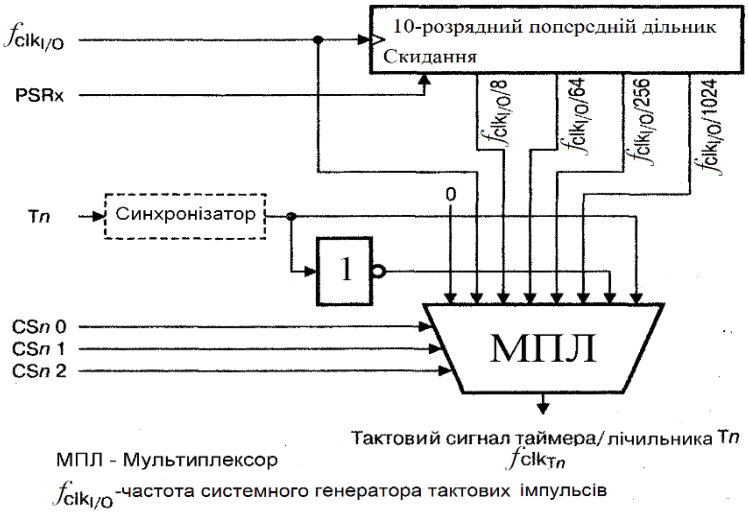


Рис. 2. Блок попереднього дільника таймера без асинхронного режиму

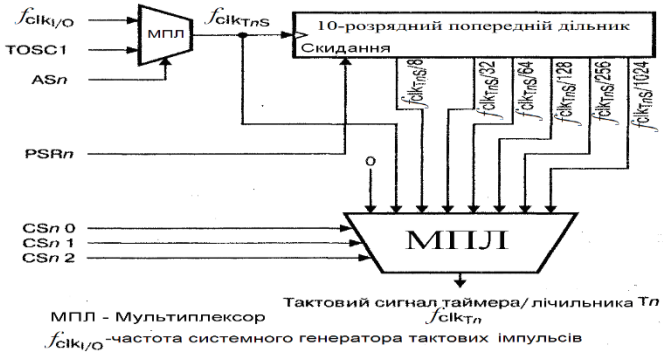


Рис. 3. Блок попереднього дільника таймера з асинхронним режимом

До складу *блоку попередніх дільників* 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 TOSC1. За схемою, яку наведено на рис. 3 виконано, наприклад, *попередній дільник таймера* T0.

Всі таймери, що не мають асинхронного режиму роботи, *використовують той самий* 10-розрядний попередній дільник. У цьому разі керування тактовим сигналом кожного таймера здійснюється *індивідуально*.

Всі МК сімейства дозволяють здійснювати *скидання попередніх дільників*, а *деякі* моделі дозволяють також здійснювати їхню *зупинку*. Для цього використовуються *спеціальні регістри*, формати яких наведено у [2] .

В якості тактового сигналу таймерів, що не мають асинхронного режиму роботи (рис. 2), може використовуватися *також зовнішній сигнал*, який надходить на вхід T*n (n* = 0...5) мікроконтролера.

1.2.3. Восьмирозрядні таймери сімейства AVR

Більшість МК сімейства AVR мають *два восьмирозрядних* таймери: T0 та T2, які мають *три варіанти* виконання. Приклади структурних схем всіх трьох варіантів, функції, які вони можуть виконувати, регістри керування та режими роботи описано у [2]. Опис основних режимів роботи восьмирозрядних таймерів буде наведено нижче під час розгляду шістнадцятирозрядних таймерів.

Восьмирозрядні таймери можуть працювати в асинхронному режиму роботи (*мають блок RTC*), який у разі відповідного налаштування викликає переривання з періодом, який дорівнює 1 секунді. На відміну від основного кварцу МК, блок RTC використовує *додатковий кварц*, який має частоту 32768 Гц. Це дозволяє під час ділення цієї частоти: 32768/128/256 отримати рівно одну секунду та використовувати таймер як *«годинник реального часу»* [4].

1.2.4. 16-розрядні таймери сімейства AVR

1.2.4.1. Опис структурної схеми таймера

Більшість моделей МК сімейства AVR мають *чотири шістнадцятирозрядних* таймери: T1, T3, T4 та T5. Вони можуть використовуватися в усіх режимах, які описано у п. 1.2.1, та відрізняються кількістю блоків порівняння і відповідно кількістю каналів генерації ШІМ-сигналів [2]. Спрощену структурну схему одного з таймерів, наприклад, моделі ATmega64x/ATmega128x, наведено на рис. 4.

До складу таймера входять наступні РВВ (де n = 1, 3, 4, 5): 16-розрядний *лічильний* регістр TCNTn; 16-розрядний регістр *захоплення* ICRn; три 16-розрядних регістри *порівняння* OCRnA, OCRnB, OCRnC та *три 8-розрядних* регістри керування: TCCRnA, TCCRnB та TCCRnC.

Фізично кожен із 16-розрядних регістрів *розміщено у двох 8-розрядних* РВВ, 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 та їх формати наведено у [2].

Таймери можуть генерувати переривання під час *виникнення наступних подій*: у разі переповнення лічильного регістра; за рівністю лічильного регістра і регістра порівняння (по одному перериванню на кожен блок порівняння) та під час збереженнія вмісту лічильного регістра у регістрі захоплення [2].

Лічильний регістр таймера/лічильника TCNTn в залежності від режиму роботи модуля *скидається*, *інкрементується*, або *декрементується* з кожним

імпульсом тактового сигналу таймера/лічильника f_{clkn}. *Після подачі напруги живлення* в регістрі TCNTn знаходиться *нульове* значення.

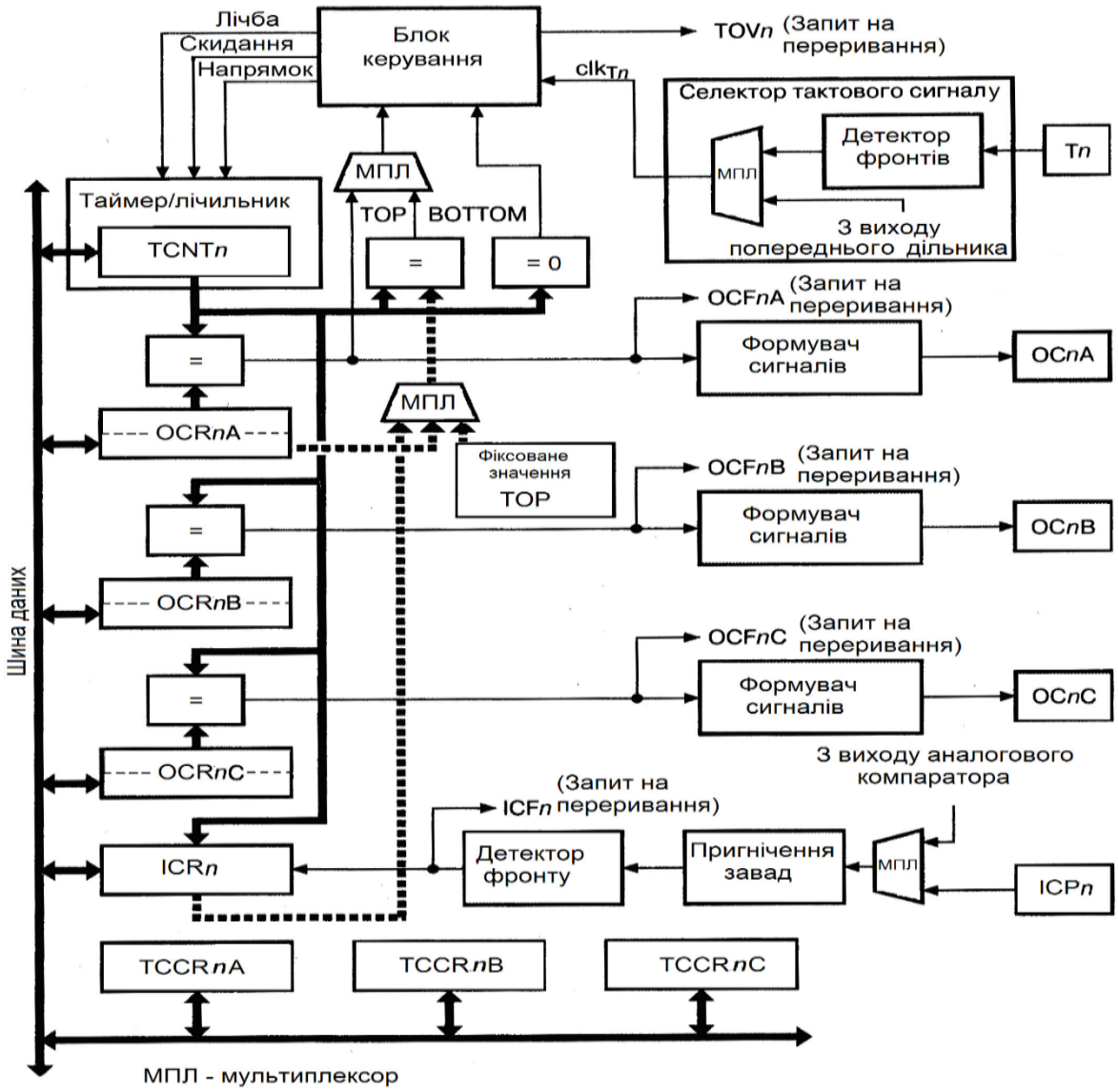


Рис. 4. Структурна схема 16-розрядних таймерів/лічильників T1, T3…T5

Регістри OCRnA/OCRnB/OCRnC входять до складу блоків порівняння. Під час роботи таймера в *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. У разі виникнення цієї події *може змінюватися стан виводу* OCnA/OCnB/OCnC мікроконтролера. Щоб таймер 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 *у разі досягнення* лічильником *максимального або мінімального* значення.

Для керування схемою захоплення використовуються два розряди регістра TCCRnB: ICNCn та 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. Якщо ж цей розряд встановлено в *одиницю*, то з *появою активного* фронту виконується *4 вибірки з частотою*, рівною тактовій частоті МК. 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.

1.2.4.2. Керування тактовим сигналом

Формування тактового сигналу для таймерів – f_{clkTn} (n = 1, 3, 4, 5) здійснюється блоком попереднього дільника, що був розглянутий у п. 1.2.2. *В якості тактового* сигналу f_{clkTn} таймерів більшості МК 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) та зовнішній сигнал, що надходить на вхід Tn мікроконтролера ($f_{clkTn} = f_{clk\ EXT}$). *Вибір джерела* тактового сигналу, а також *запуск і зупинка* таймерів здійснюються за допомогою розрядів CSn2...CSn0 *регістра* керування таймером *TCCRnB* відповідно до табл. 1.

1.2.4.3. Режими роботи

Режим роботи таймерів визначається станом відповідних розрядів *регістра TCCRnA* (табл. 2).

Таблиця 1. Вибір джерела тактового сигналу таймерів/лічильників Tn

CSn2	CSn1	CSn0	Джерело тактового сигналу	
			Т3 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.

Режим Normal

Режим Normal є *найпростішим* режимом роботи таймерів. У ньому лічильний регістр функціонує як звичайний лічильник, що підсумовує. З кожним імпульсом тактового сигналу f_{clkTn} здійснюється *інкремент* лічильного регістра.

У разі переходу 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.

Поведінка виходів OCnA/OCnB/OCnC кожного з блоків порівняння таймерів/лічильників Tn визначається станом розрядів COMnx1:COMnx0 регістрів TCCRnA, як показано в табл. 3.

Стан виходу будь-якого блоку порівняння також може бути змінено *примусово*, *записом одиниці* у розряд FOCnA/FOCnB/FOCnC регістра TCCRnC або регістра TCCRnA. *Переривання* у цьому разі *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. 2*).

Таблиця 2. Режими роботи 16-розрядних таймерів

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

Таблиця 3. 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.

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. 2*).

Після досягнення максимального значення TOP, відбувається скидання лічильного регістра та 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 наведено на рис. 5.

У разі досягнення лічильником максимального значення встановлюється прапорець: OCFnA (реж. 4) або ICFnA (реж. 12). Одночасно із встановленням прапорця може змінюватися стан виводу OCnx МК. Поведінка виводу визначається станом розрядів COMnx1:COMnx0 регістрів TCCRnA (табл. 3).

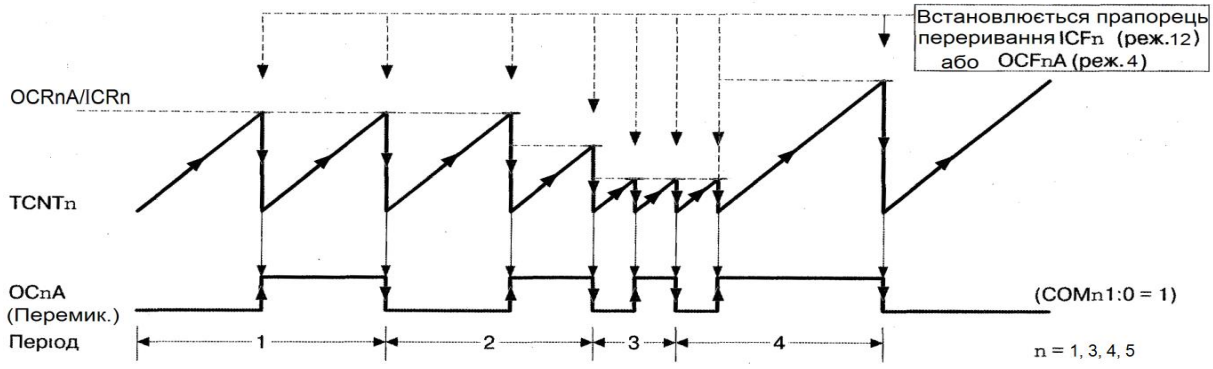


Рис. 5. Часові діаграми для режиму «Скидання за збігом» (Chop on Timer Coincidence)

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:

$$f_{OCnx} = f_{clkI/0}/2 \cdot N(1+Y), \quad (1)$$

$$t_{імп} = N \cdot (1+Y) \cdot T_{clkI/0}, \quad (2)$$

$$T_{OCnx} = 2\, t_{імп} = 2 \cdot N \cdot (1+Y) \cdot T_{clkI/0}, \quad (3)$$

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. Як і в режимі Normal, *стан виводу* OCnx у разі необхідності може бути *змінено примусово* записом одиниці у розряд FOCnx регістра TCCRnC або регістра TCCRnA. *Переривання* у цьому разі *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 [2]. Відмінність полягає тільки в тому, що *максимальне значення, до якого буде лічити лічильник, може програмно змінюватися (табл. 2).* 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 керувальних регістрів таймерів/лічильників (табл. 2).

Роздільна здатність R визначається виразом:

$$R = \log(TOP + 1)/\log 2 , \qquad (4)$$

де TOP – модуль лічби (табл. 4).

Таблиця 4. 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.

У разі досягнення лічильником максимального значення встановлюється прапорець переривання TOVn. Одночасно з ним встановлюється прапорець ICFn (реж. 14) або OCFn (реж. 15).

У разі 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. 5).

Таблиця 5. 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 у разі рівності лічильного регістра і відповідного регістра порівняння. Встановлюється в одиницю під час досягнення 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.

Часові діаграми для випадку, коли модуль лічби визначається вмістом регістра OCRnA, показано на рис. 6.

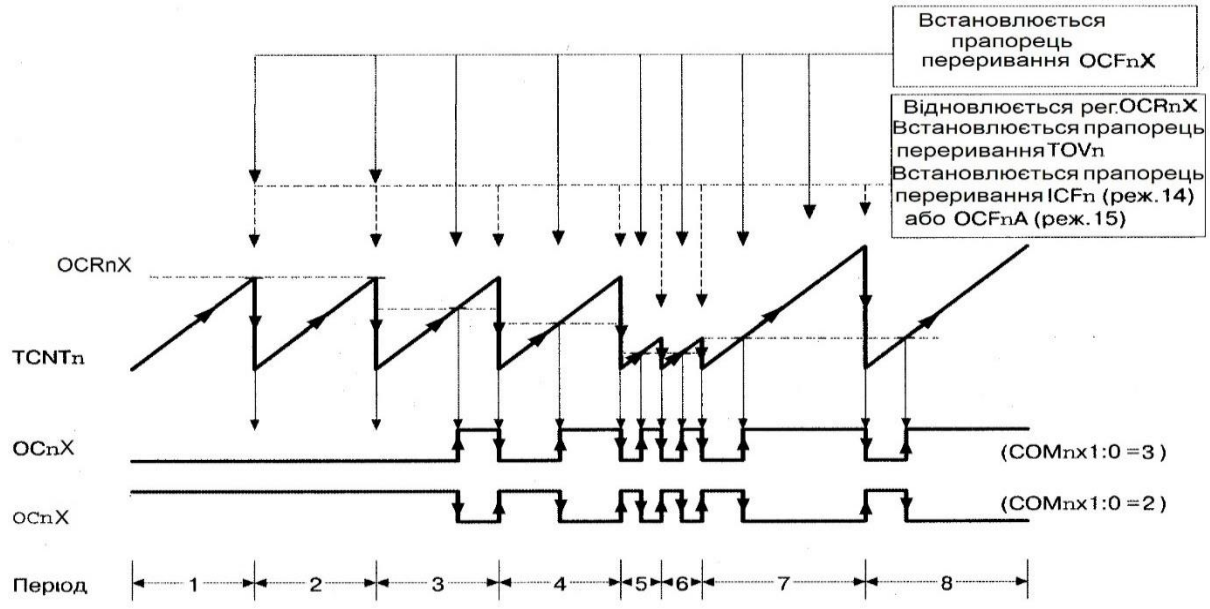


Рис. 6. Формування ШІМ-сигналу в режимі Fast PWM

У разі роботи з *фіксованим* значенням модуля лічильника *для задання модуля лічби* рекомендується використовувати регістр захоплення *ICRn*. У цьому разі *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 [2].

Якщо *вміст регістра порівняння дорівнює модулю лічби*, то вихід відповідного блоку порівняння переключиться в стійкий стан, обумовлений станом розрядів COMnx1:COMnx0 (табл. 5).

Частота сигналу, тривалість імпульсу та період сигналу, які генеруються, визначаються виразами:

$$f_{OCnx} = f_{clkI/0}/N\cdot(TOP+1), \quad (5)$$

$$t_{імп} = T_{clkI/0}\cdot N\cdot(1 + OCRnx), \text{ неінвертований ШІМ-сигнал} \quad (6)$$

$$t_{імп} = T_{clkI/0} - (T_{clkI/0}\cdot N\cdot(1 + OCRnx)), \text{ інвертований ШІМ-сигнал} \quad (7)$$

$$T_{OCnx} = T_{clkI/0}\cdot N\cdot(TOP+1), \quad (8)$$

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.

У разі необхідності *блок порівняння «А»* у цьому режимі може також використовуватися для генерації *сигналу «Меандр».* Для цього необхідно записати до розрядів COMnA1:COMnA0 значення «01», що задає перемикання стану виходу OCnA під час виникнення 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 ШІМ-сигналів*. 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 керувальних регістрів таймера [2].

У разі досягнення 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. У разі досягнення лічильником мінімального *значення (\$0000*) також відбувається зміна напряму лічби та одночасно встановлюється прапорець переривання TOVn. Під час *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. 5).

У разі зміни модуля лічби під час роботи таймера 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. У цьому разі *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 відносно середини періоду модуляції* на виході блоків порівняння, як показано на рис. 7 (*3-й період сигналу*).

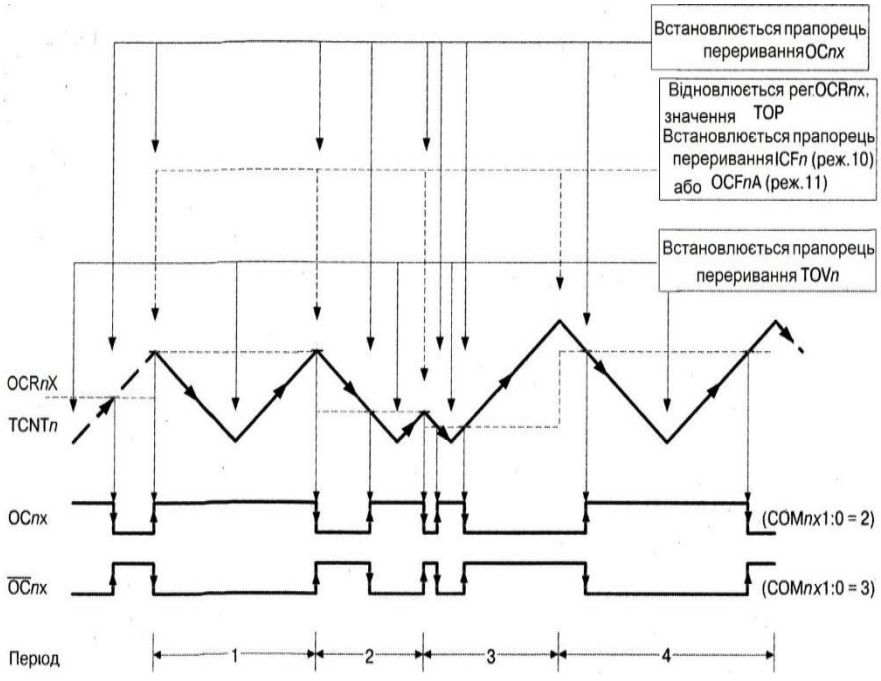


Рис. 7. Формування ШІМ-сигналу в режимі Phase Correct PWM

У разі частої зміни 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*.

Частота, тривалість імпульсу та період сигналу, що генерується, визначаються виразами:

$$f_{OCnx} = f_{clkI/0}/2 \cdot N \cdot (TOP+1), \quad (9)$$

$$t_{імп} = T_{clkI/0} \cdot 2 \cdot N \cdot (1 + OCRnx), \text{ неінвертований ШІМ-сигнал} \quad (10)$$

$$t_{імп} = T_{clkI/0} - (T_{clkI/0} \cdot 2 \cdot N \cdot (1 + OCRnx)), \text{ інвертований ШІМ-сигнал} \quad (11)$$

$$T_{OCnx} = T_{clkI/0} \cdot 2 \cdot N \cdot (TOP+1), \quad (12)$$

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. 8).

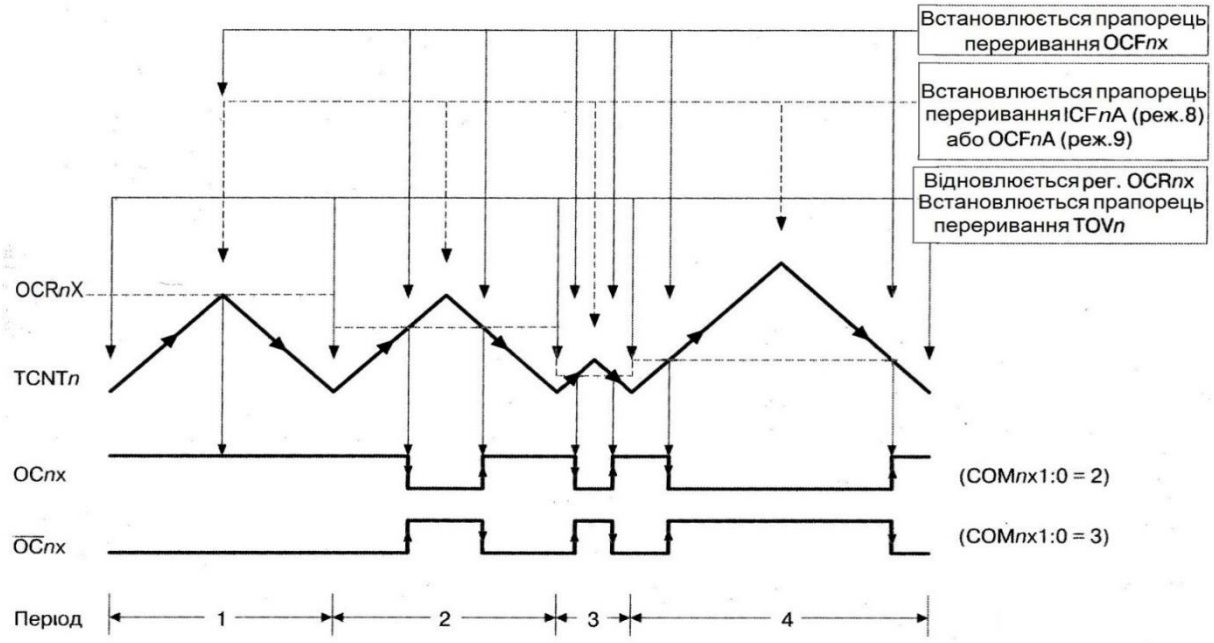


Рис. 8. Формування ШІМ-сигналу в режимі Phase and Frequency Correct PWM

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 середини періоду модуляції у* і, відповідно, частота сигналу, що генерується, залишається сталою.

Як і в інших режимах, при роботі з *фіксованими значеннями модуля лічби*, що змінюються *рідко*, для його задання рекомендується використовувати регістр

захоплення *ICRn*. У цьому разі 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.

У разі 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). У разі досягненнія лічильником мінімального значення ($0000) напрям лічби знову змінюється. У цьому разі встановлюється прапорець переривання TOVn відповідного регістра і відбувається відновлення вмісту регістра порівняння.

У разі 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. 5).

Частота, тривалість імпульсу та період сигналу, що генерується, визначаються виразами:

$$f_{OCnx} = f_{clkI/0}/2 \cdot N \cdot (TOP+1), \quad (13)$$

$$t_{імп} = T_{clkI/0} \cdot 2 \cdot N \cdot (1 + OCRnx), \text{ неінвертований ШІМ-сигнал} \quad (14)$$

$$t_{імп} = T_{clkI/0} - (T_{clkI/0} \cdot 2 \cdot N \cdot (1 + OCRnx)), \text{ інвертований ШІМ-сигнал} \quad (15)$$

$$T_{OCnx} = T_{clkI/0} \cdot 2 \cdot N \cdot (TOP+1), \quad (16)$$

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.

1.2.5. Вартовий таймер

Всі МК 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 МК сімейства можна зустріти вартовий таймер у *двох виконаннях*: стандартному та розширеному [2]. Структурну схему вартового таймера у *стандартному* режимі приведено на рис. 9.

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 [2].

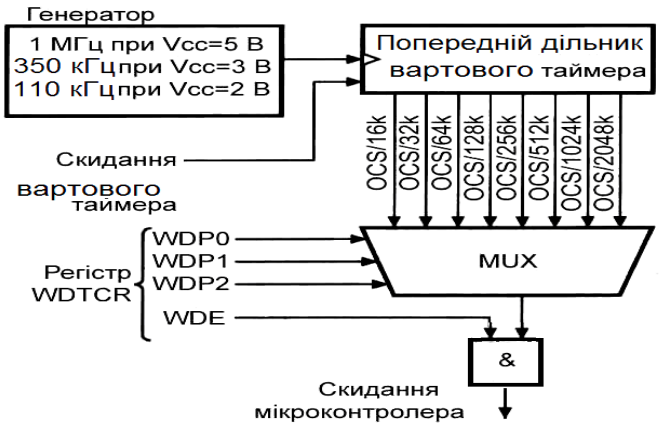


Рис. 9. Структурна схема вартового таймера

1.2.6. Моделювання модуля таймера AVR-мікроконтролерів

У [2; 4] описано моделювання в пакеті PROTEUS *модуля таймера AVR-мікроконтролера для керування двигуном постійного струму та годинником реального часу*. Наведено моделі, їх описання, схеми алгоритмів роботи, робочі програми та «скріншоти», які пояснюють роботу моделей.

1.3. Застосування для формування інтервалів часу мікросхеми програмованого таймера

Для реалізації генераторів тактових імпульсів необхідної частоти, формування часових затримок, імпульсів визначеної тривалості, стробувальних імпульсів, а також підрахунку зовнішніх подій у МПС може використовуватися, наприклад, програмований *таймер (ПРТ) і8253*.

У [1] наведено умовне позначення цієї мікросхеми на електричних схемах, структурну схему, зв’язок таймера із системною шиною даних МПС, описано роботу в шести режимах, розглянуто програмування таймера та приклад використання.

Контрольні запитання та завдання

1. Назвіть способи формування інтервалів часу у МПС.
2. Який цифровий електронний пристрій складає основу ПРТ?
3. Чим відрізняється режим таймера від режиму підрахунку зовнішніх подій?
4. Для чого можуть використовуватись таймери/лічильники?
5. Опишіть режими роботи 16-розрядних таймерів.
6. Для рішення яких завдань краще використання режиму Phase Correct PWM ніж режиму Fast PWM?
7. У разі виникнення яких подій таймери T1, T3, T4, T5 можуть генерувати переривання?
8. За допомогою яких регістрів виконується керування таймером?

9. Які сигнали можуть використовуватись в якості тактового сигналу f_{clkn} для таймерів T1, T3, T4 і T5?
10. Яким чином здійснюється вибір джерела тактового сигналу, а також запуск і зупинка таймерів?
11. Завдяки чому в режимі Phase and Frequency Correct PWM кожен період сигналу є повністю симетричним?
12. Як обчислюється шпаруватість ШІМ-сигналу?
13. Як впливає значення шпаруватості на швидкість обертання двигуна постійного струму?
14. Наведіть та поясніть модель, схему алгоритму роботи пристрою керування двигуном постійного струму та робочу програму.
15. Наведіть та поясніть робочу модель годинника реального часу.
16. Наведіть та поясніть схему алгоритму роботи та робочу програму керування моделлю годинника реального часу.
17. Опишіть структуру, режими роботи та призначення виводів програмованого таймера i8253.
18. Як таймер i8253 підключається до системної шини?
19. Опишіть особливості програмування таймера i8253.

ЛЕКЦІЯ 9. ПРОЕКТУВАННЯ ПРИСТРОЮ КЕРУВАННЯ ДВИГУНОМ ПОСТІЙНОГО СТРУМУ

1. ПРОЕКТУВАННЯ ПРИСТРОЮ КЕРУВАННЯ ДВИГУНОМ ПОСТІЙНОГО СТРУМУ

1.1. Завдання

Виконати проектування пристрою керування двигуном постійного струму:

1) Розробити, навести та описати робочу модель в пакеті PROTEUS.
2) Зробити розрахунок ШІМ-модуля.
3) Розробити схему алгоритму роботи пристрою.
4) Розробити керувальну програму мовою Асемблер.
5) Розробити керувальну програму мовою С.
6) Виконати моделювання пристрою в пакеті PROTEUS. Навести «скріншоти», які пояснюють роботу моделі та підтверджують правильність виконаних розрахунків, роботи схеми алгоритму та робочої керувальної програми.
7) Розробити та описати структурну схему пристрою.

Вихідні дані:

– тип МК-ра: AT mega 128;
– номер таймера: T/C1;
– частота вихідного ШІМ-сигналу: fшім = 25 кГц;
– частота тактового сигналу підсистеми введення/виведення: $f_{CLK\ I/0}$ = 16 МГц;
– шпаруватість ШІМ-сигналу: Qшім = 2;
– режим роботи модуля ШІМ: Phase and Frequency Correct PWM;
– лінія порту мікроконтролера, на якій формується ШІМ-сигнал, – PB5;
– вид вихідного ШІМ-сигналу – *інвертований*.

Розрахувати:

– коефіцієнт ділення переддільника: $К_{діл.} = N$;
– модуль лічби: TOP;
– період ШІМ-сигналу: T_{PB5};
– тривалість імпульсу ШІМ-сигналу: $t_{імпPB5}$;
– шпаруватість: Q.

1.2. Розробка та опис робочої моделі

1.2.1. Робоча модель пристрою

Робочу модель пристрою керування двигуном постійного струму наведено на рис. 1.

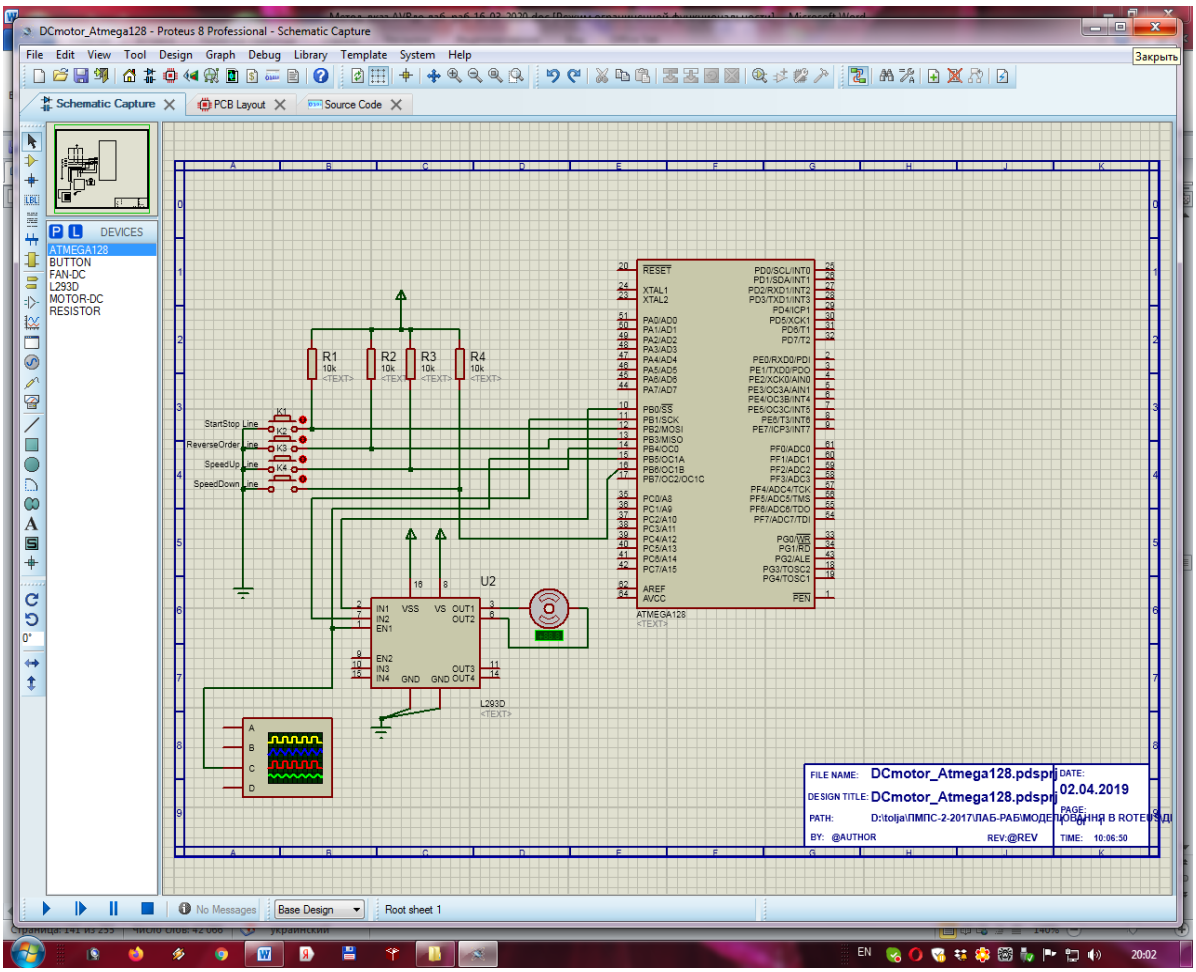


Рис. 1. Схема робочої моделі пристрою керування двигуном постійного струму

1.2.2. Опис моделі

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 за допомогою ШІМ-сигналу мікроконтролер керує двигуном. В *якості мікроконтролера* обрано мікросхему ATmega128, яка знаходиться у правому куті моделі. *Зовнішні резистори* використовуються через те, що на вхідних лініях мікроконтролера не підключені внутрішні підтягуючі резистори.

На рис. 2 наведено *збільшений вигляд* кнопок керування та порядок їх розташування.

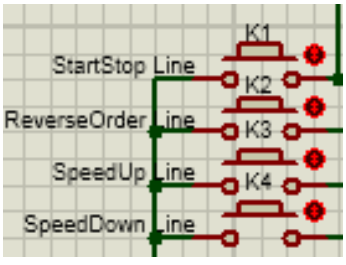


Рис. 2. Порядок розташування кнопок керування

У вихідному стані кнопки *є нормально розімкненими*. В цьому випадку на відповідні лінії порту B (PB) мікроконтролера подаються *високі рівні* напруги, тобто логічні одиниці. Коли *кнопки замикаються*, на входи мікроконтролера подаються низькі 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*. Він не може зупинитися миттєво та одразу почати обертатися в інший бік. Після зміни напрямку програмно, у нього витрачається певний період часу, щоб фізично зупинитись, та почати обертатись у іншу сторону. Кнопку K2 підключено до *13-го виводу* ATmega128, а саме до PB3, який попередньо *запрограмовано як вхід*.

Кнопка K3 («SpeedUp Line») відповідає за *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. 3).

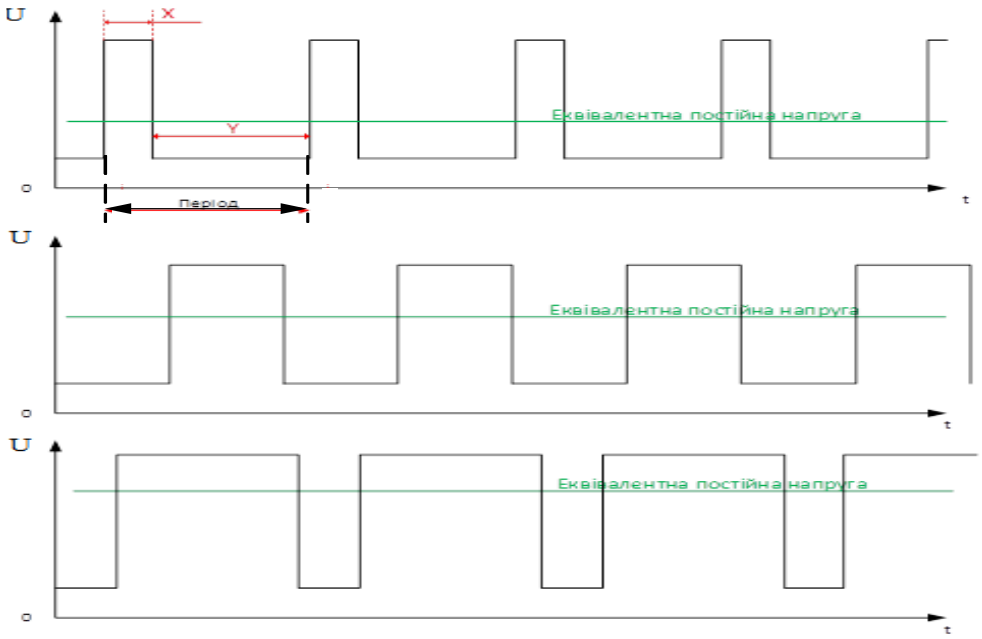


Рис. 3. Формування імпульсного ШІМ-сигналу

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 (рис. 3).

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. 4 наведено підключення до мікроконтролера драйвера ШІМ-сигналу – мікросхеми L293D (U2).

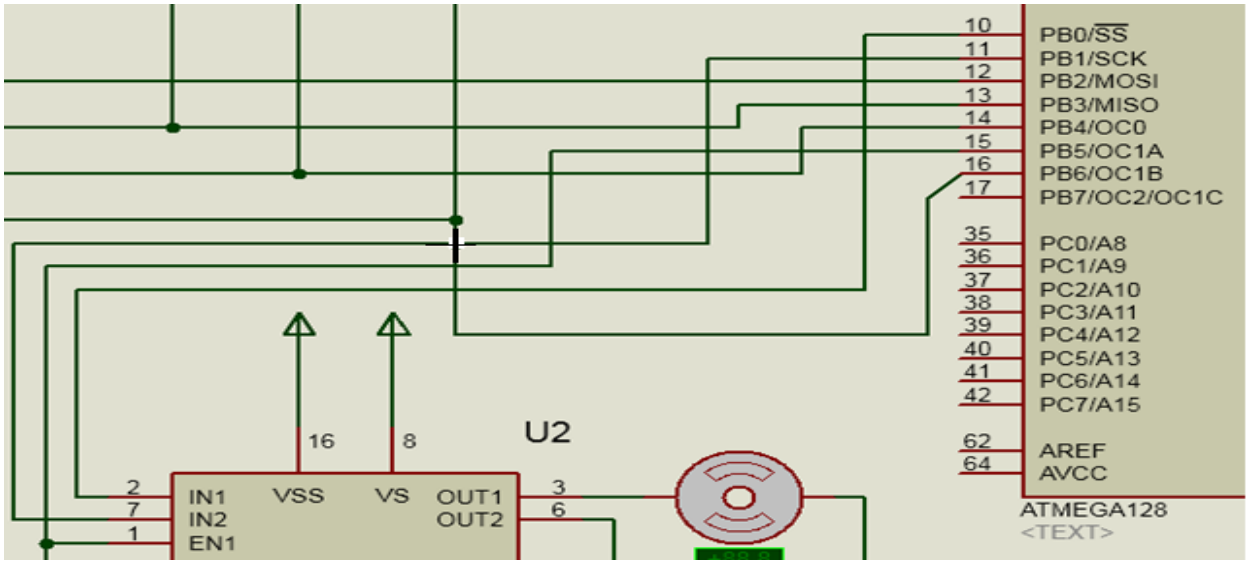


Рис. 4. Підключення мікросхеми L293D та мікроконтролера ATmega128

До входів IN1 та IN2 цієї мікросхеми підключені виводи 10 та 11 мікроконтролера: *PB0 та PB1, які запрограмовані як виходи. В залежності від комбінації* сигналів керування на цих входах – нуль або одиниця, двигун буде або стояти, або 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*. Вхід EN1 відповідає за роботу *першої мостової схеми* у складі мікросхеми. Завдяки *зміни шпаруватості* ШІМ-сигналу на цьому вході можна *змінювати швидкість* обертання двигуна.

Два входи GND мікросхеми L293D заземлено. На *вхід VSS* подається напруга *живлення самої мікросхеми*, а на вхід *VS – напруга живлення двигуна*.

На рис. 5 зображено підключення до мікросхеми L293D двигуна.

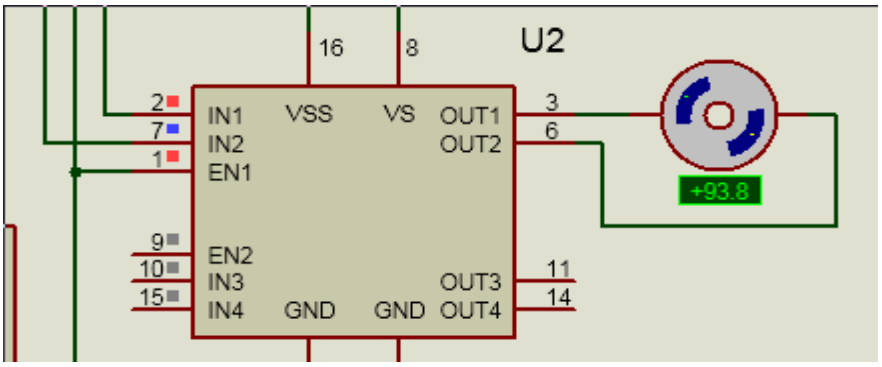


Рис. 5. Підключення до мікросхеми L293D двигуна

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.

Як видно з рис. 5 в моделі сигнал на вході IN1 позначено *червоним кольором*, тобто від мікроконтролера подано високий рівень (*логічну одиницю*). Сигнал на вході IN2 позначено *синім* кольором, тобто подається низький рівень – *логічний нуль.* За такою комбінацією на входах IN1 та IN2 двигун почав обертатися *за годинниковою стрілкою*. *У разі протилежної* комбінаії двигун буде обертатися *проти годинникової стрілки*. *За однакових* сигналах на входах IN1 та IN2 *двигун обертатися не буде*.

На рис. 6…9 наведено деякі фрагменти, які демонструють роботу моделі.

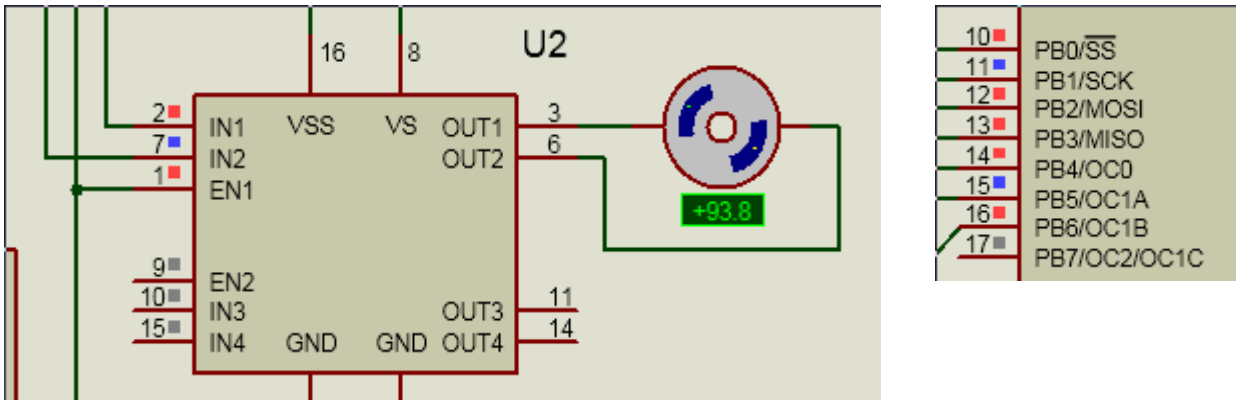


Рис. 6. Стан моделі після натискання кнопки «старт/стоп»

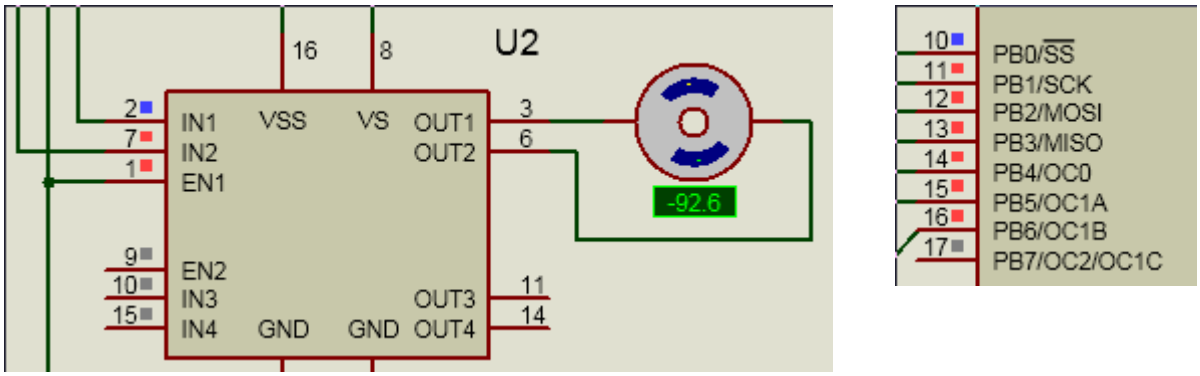


Рис. 7. Стан моделі після активації режиму реверсу

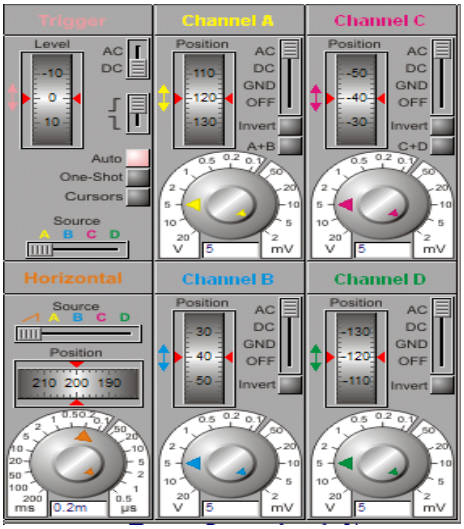
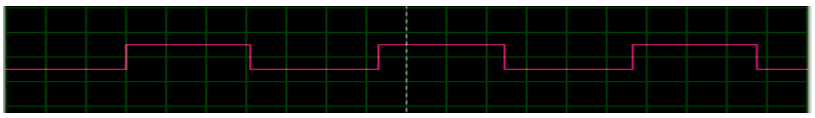
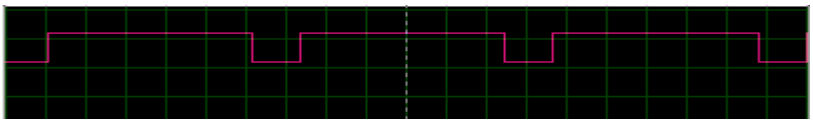


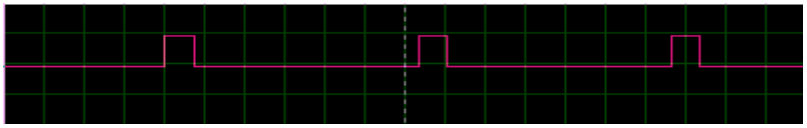
Рис. 8. Налаштування осцилографа



q = 2



q < 2



q > 2

Рис. 9. ШІМ-сигнал, що подається на двигун (шпаруватість q = 2, q < 2, q > 2)

На рис. 10 наведено стан усієї системи *після запуску сценарію*. Шпаруватість ШІМ-сигналу дорівнює 2, але *двигун не обертається*, тому що IN1 = IN2 = 0.

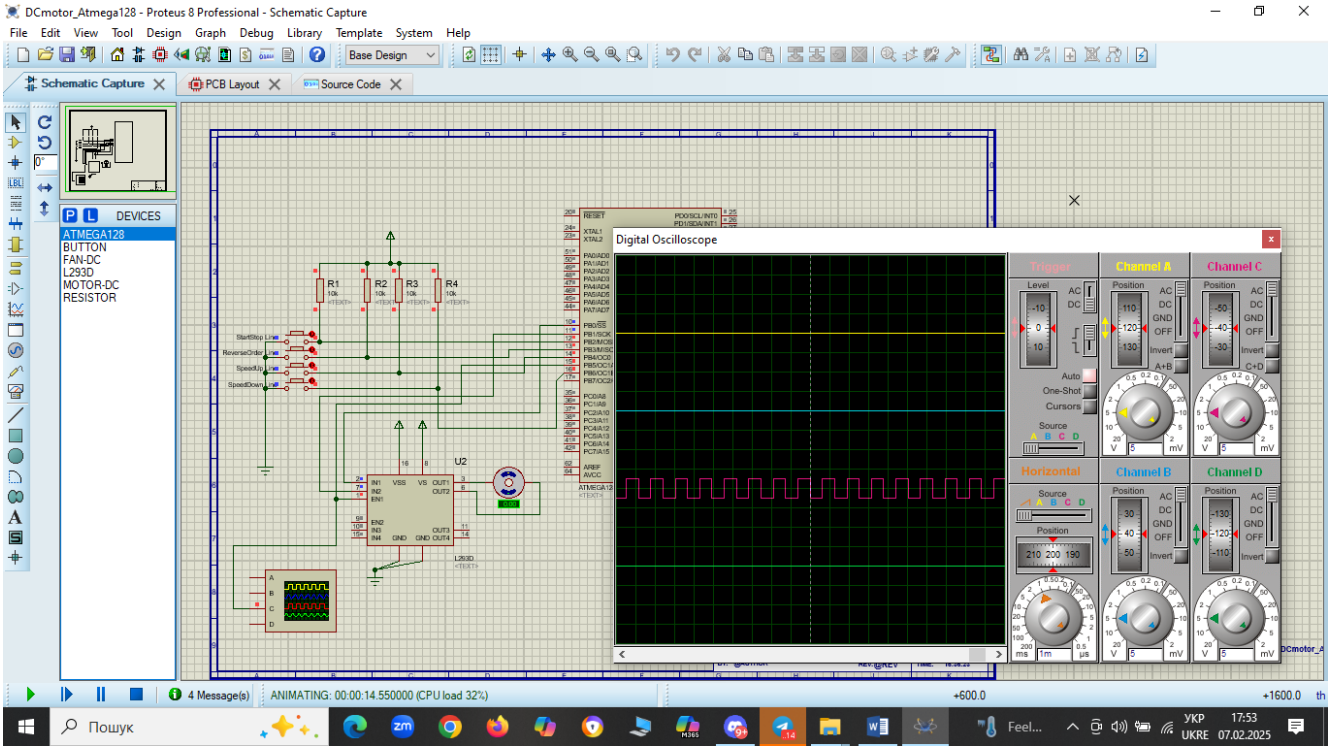


Рис. 10. Стан усієї системи після запуску сценарію

1.2.3. Опис окремих елементів моделі

1.2.3.1. Мікроконтролер

Для контролю та виконання всіх функцій, які покладено на модель, було обрано МК сім`ї AVR – *ATmega 128*.

На рис. 11 наведено *зовнішній вигляд* та позначення виводів цього мікроконтролера.

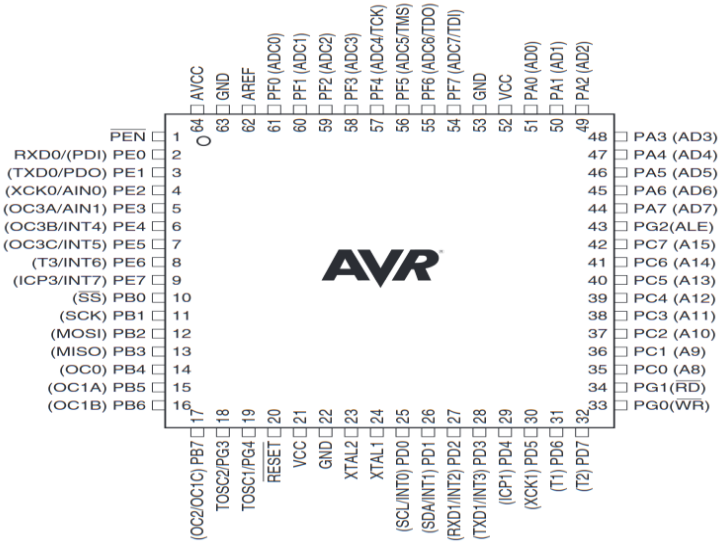


Рис. 11. Зовнішній вигляд та позначення виводів мікроконтролера ATmega 128

Мікроконтролер є малопотужним 8-розрядним *КМОН*-мікроконтролером, який має розширену RISC-архітектуру. Його основні характеристики та опис виводів розглянуто у [2].

1.2.3.2. 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 керувальних імпульсів, що у свою чергу викликає зміну постійної складової імпульсного сигналу (рис. 3). *Чим менше шпаруватість, а відповідно більше величина відношення «тривалість імпульсу/період», тим більша напруга надходить на двигун, який буде обертатись із більшою швидкістю.*

1.2.3.3. Двигун

В моделі використовується *електродвигун постійного струму*. Регулювання швидкості обертання двигуна зазвичай забезпечується за допомогою формування керувальних імпульсів у вигляді сигналів з широтно-імпульсною модуляцією. Подібний підхід дозволяє *регулювати середню величину потужності*, що надходить на двигун. Частота сигналу з широтно-імпульсною модуляцією повинна перевищувати 20 кГц, що дозволяє виключити звукові ефекти, пов’язані з формуванням звукових сигналів самим двигуном під час зміни магнітного поля, яке в ньому утворюється [1].

1.2.3.4. Драйвер ШІМ

Двигун постійного струму підключається до мікроконтролера за допомогою відповідної *мостової схеми – драйвера*, яку зображено на рис.12.

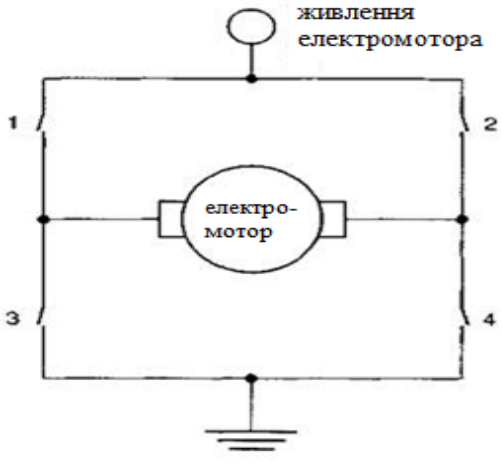


Рис. 12. Мостова схема для підключення двигуна постійного струму до мікроконтролера

Подібну схему мають драйвери двигунів, які використовуються для *перетворення* керувальних сигналів *малої потужності від мікроконтролера* у сигнали, *потужність яких достатня для керування двигунами*.

Окрім підсилення потужності керувальних сигналів *драйвери керують зміною напрямку обертання* двигунів. *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.13.

Рис. 13. Зовнішній вигляд мікросхеми L293D

Мікросхема L293D включає *два драйвери* та може керувати *двома електродвигунами відповідної потужності*. Також мікросхема забезпечує *розділене живлення для самої мікросхеми та для керованих нею двигунів*, що дозволяє підключати електродвигуни з більшою напругою живлення, чим у мікросхеми. Розділення живлення мікросхеми і електродвигунів необхідне також для зменшення завад, які викликано стрибками напруги, що пов’язані з роботою двигунів.

Обидва драйвери, що входять у склад мікросхеми мають *ідентичні принципи роботи*, тому нижче розглянуто принцип роботи лише одного з них, спрощений вид якого наведено на рис. 14.

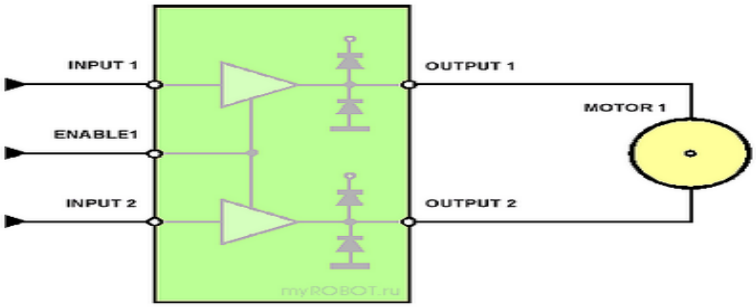


Рис. 14. Спрощений вид драйвера мікросхеми L293D

До виходів OUTPUT1 та OUTPUT2 підключається двигун постійного струму MOTOR1. *Вхід ENABLE1*, який відповідає за ввімкнення драйвера, підключають до додатного полюсу джерела живлення: *+5V*. Якщо у разі цього відповідні керувальні сигнали на входи INPUT1 та INPUT2 не подаються, 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 на керувальні входи INPUT1 та INPUT2 подати сигнали *одного рівня*, тобто під'єднати обидва входи до додатного полюсу джерела живлення або до від`ємного, то *двигун обертатися не буде*.

Якщо прибрати керувальний сигнал зі входу ENABLE1, то за будь-яких варіантах сигналів на входах INPUT1 та INPUT2 двигун також обертатися не буде. *На вхід ENABLE1 подається імпульсний ШІМ-сигнал*, який формується модулем таймера мікроконтролера. *Змінюючи шпаруватість* цього сигналу ми *змінюємо постійну складову* імпульсного сигналу, що в свою чергу *змінює швидкість* обертання двигуна.

Нумерацію, позначення та опис виводів мікросхеми L293D наведено у [2].

1.2.3.5. Захисні діоди

Як видно з рис. 14 схема драйверів містить захисні діоди. Ці діоди призначено *для захисту транзисторних ключів* у складі драйверів від додатних і від'ємних паразитних імпульсів досить високої амплітуди, які з'являються на обмотках двигуна *під час комутації обмоток*. *Додатні* імпульси виникають у разі запирання (*вимикання*) ключів, а від'ємні – *під час включення*. Механізм виникнення цих імпульсів описано у [1].

1.3. Розрахунок ШІМ-модуля та його програмування мовою Асемблер

На рис.15 наведено формування таймером мікроконтролера ШІМ-сигналу в *режимі Phase and Frequency Correct PWM [2]*.

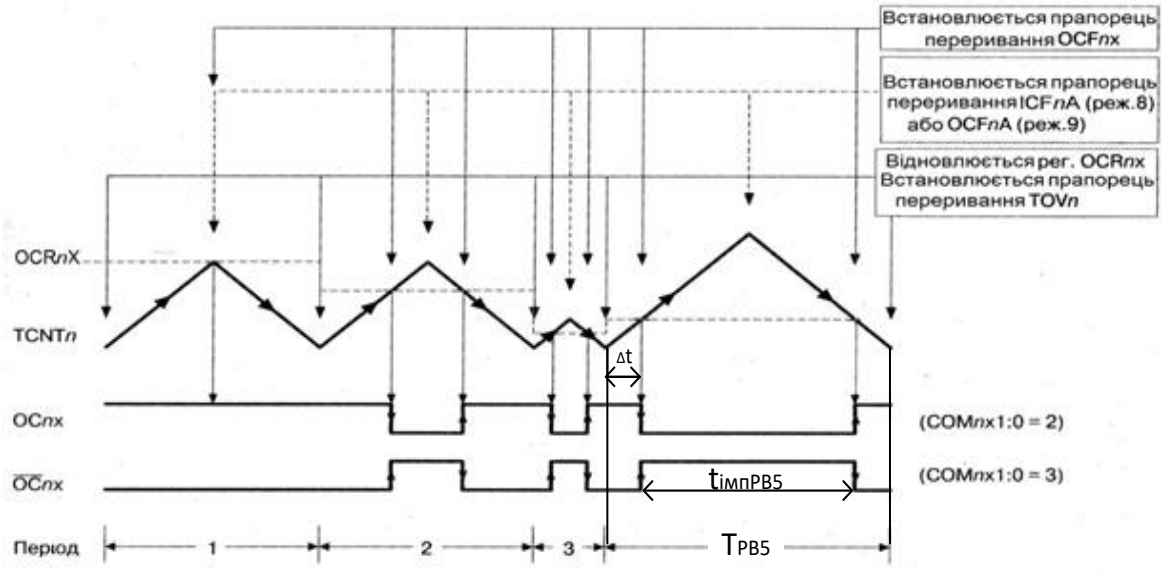


Рис. 15. Формування таймером ШІМ-сигналу в режимі Phase and Frequency Correct PWM

Величина частоти ШІМ-сигналу f_{PB5} визначається виразом [2]:

$$f_{OCnx} = f_{clkI/0}/2 \cdot N \cdot TOP, \quad (1)$$

де N – коефіцієнт ділення попереднього дільника таймера; TOP – модуль лічби; $f_{clkI/0}$ – частота генератора тактових імпульсів підсистеми введення/виведення мікроконтролера.

Обираємо *восьмий режим* роботи таймера (лекція 8). В цьому режимі значення модуля лічби *TOP визначається* вмістом регістра *ICR1*. Згідно (1) за *N = 8* та значенні f_{PB5}, яке задано в завданні та дорівнює *25кГц*, розраховуємо значення ICR1 = TOP:

$$TOP = ICR1 = \frac{f_{CLK\,I/O}}{2N \cdot f_{PB5}} = \frac{16 \cdot 10^6}{2 \cdot 8 \cdot 25 \cdot 10^3} = 40 = 00101000B = \$0028.$$

Період ШІМ-сигналу: $T_{PB5} = \frac{1}{f_{PB5}} = \frac{1}{25000} = 40$ мкс.

У разі шпаруватості ШІМ-сигналу $Q_{PB5} = 2$ потрібна тривалість імпульсу

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

Згідно з рис. 15

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

$$\Delta\,t = \frac{T_{PB5} - t_{імпPB5}}{2} = \frac{40-20}{2} = 10\ мкс. \quad (3)$$

Відповідно до роботи таймера в режимі Phase and Frequency Correct PWM, інвертований ШІМ-сигнал (рис. 15)

$$\Delta\,t = T_{clkI/0} \cdot N \cdot OCR1A, \quad (4)$$

де OCR1A – регістр порівняння каналу А таймера 1.

Із виразу (4)

$$OCR1A = \frac{\Delta\,t}{T_{CLK\,I/0} \cdot N} = \frac{10 \cdot 10^{-6}}{\frac{1}{16} \cdot 10^{-6} \cdot 8} = 20 = 00010100B = \$0014.$$

Тоді шпаруватість:

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

$$= \frac{1}{1 - 2 * \frac{1}{f_{CLKI/O}} * N * OCR1A * f_{PB5}} \quad . \quad (5)$$

За останньою формулою *за OCR1A = 20* розрахуємо значення шпаруватості Q, яке повинно *дорівнювати 2*:

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

Отриманий результат говорить про те, що попередні розрахунки зроблено вірно.

Тепер обчислимо значення Q *у разі збільшення OCR1A на 5 та у разі зменшення на 5:*

$$Q_{(OCR1A=25)} = \frac{1}{1-2\cdot\frac{1}{16\cdot10^6}\cdot8\cdot25\cdot25\cdot10^3} = 2{,}667,$$

$$Q_{(OCR1A=15)} = \frac{1}{1-2\cdot\frac{1}{16\cdot10^6}\cdot8\cdot15\cdot25\cdot10^3} = 1{,}6.$$

1.4. Схема алгоритму роботи пристрою

На рис. 16 наведено схему алгоритму роботи пристрою.

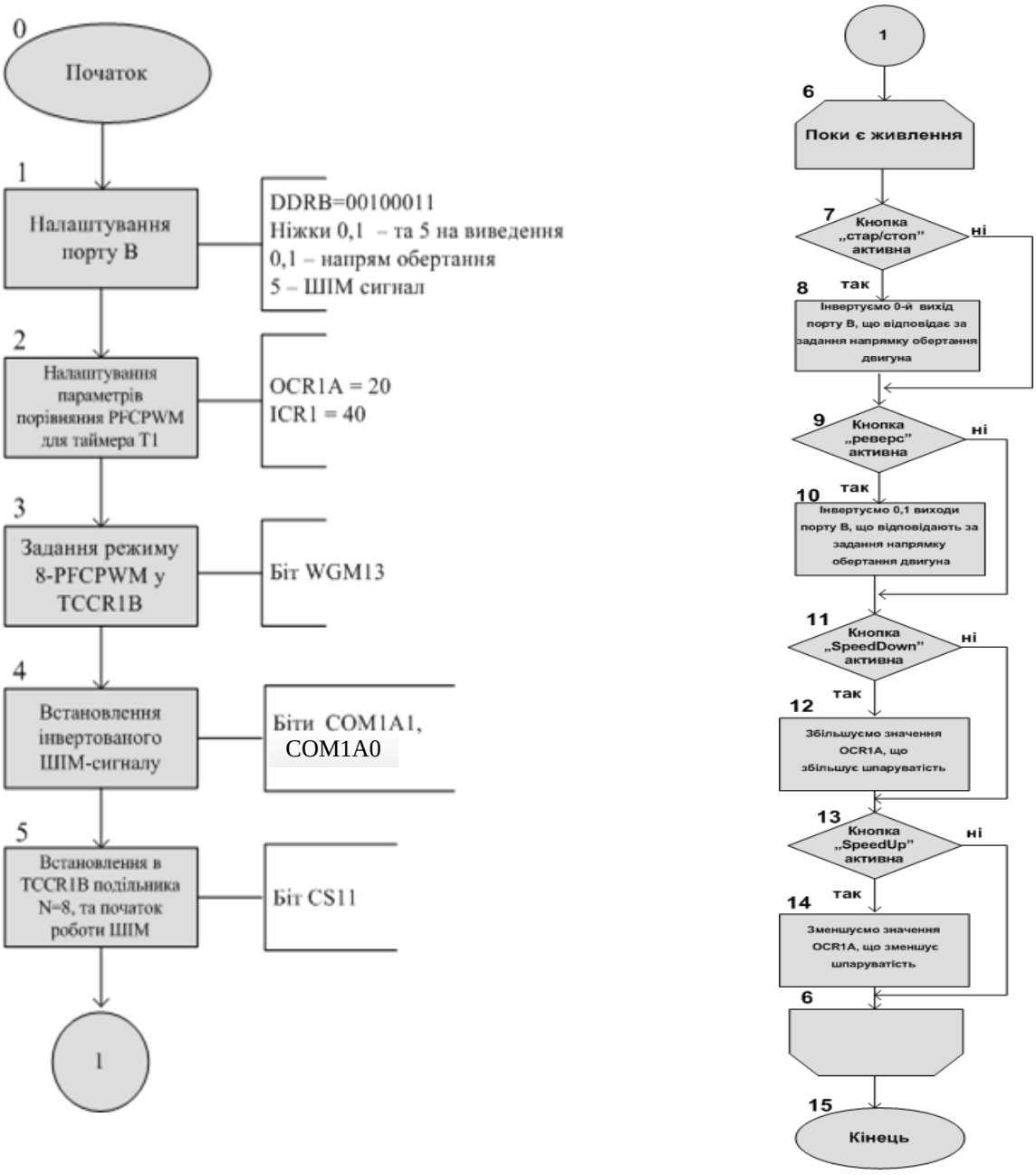


Рис. 16. Схема алгоритму роботи пристрою

1.5. Розробка окремих фрагментів програми мовою Асемблер

1.5.1. Зупинка таймера

Для зупинки таймера треба записати в регістр керування *TCCR1B* (рис. 17), *адреса* якого дорівнює *$004E* [2], наведене нижче керувальне слово
КС1 = 00000000B = $00.

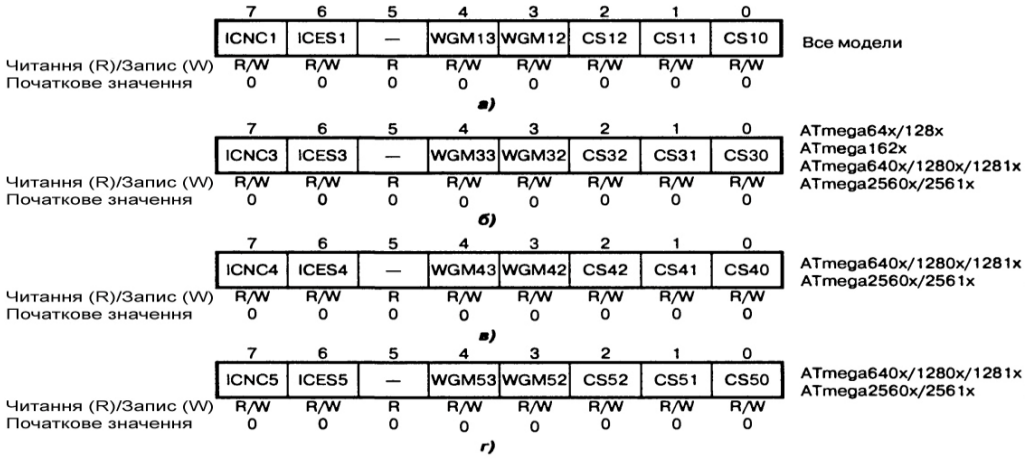


Рис. 17. Формат регістрів TCCR1B (а), TCCR3B (б), TCCR4B (в), TCCR5B (г)

Тоді програма *зупинки таймера* має вигляд:

LDI R18, $00; R18← КС1 = $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)

LDI R27, $00; R27 ← $00
LDI R26, $4E; R26 ← $4E } X(R27, R26) ← $004E;
ST X, R18; TCCR1B ← R18 = $00, зупинка таймера.

1.5.2. Завантаження регістра TCCR1A

Формат регістра TCCR1A наведено у [2]. Для нашої задачі для формування *інвертованого ШІМ-сигналу* згідно табл. 1 необхідно встановити біти COM1A1 = COM1A0 = 1.

Для програмування *режиму роботи 8* (лекція 8) необхідно запрограмувати біти: *WGM11 = 0; WGM10 = 0*. Інші біти регістра TCCR1A у нашому прикладі не використовуються. Тому запишемо в них нулі.

Таким чином формат регістра *керувального слова (КС2*), яке завантажується у регістр TCCR1A мікроконтролера AT Mega 128, має вигляд:

7 р.	6р.	5р.	4р.	3р.	2р.	1р.	0р.	
COM1A1	COM1A0	COB1B1	COM1BO	COM1C1	COM1C0	WGM11	WGM10	КС2
1	1	0	0	0	0	0	0 B=$C0	

Таблиця 1. Поведінка виводу OCnA/OCnB/OCnC в режимі Phase and Frequency Correct PWM

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

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

Адреса регістра *TCCR1A = $004F* [2]. Тоді програма завантаження регістра TCCR1A має вигляд:

LDI R17, $C0; R17← KC2 = $C0;
LDI R29, $00; R27 ← $00
LDI R28, $4F; $26 ← $4F } Y(R29, R28) ← $004F;
ST Y, R17; TCCR1A←R17 = $C0.

1.5.3. Програмування ліній PB5, PB1, PB0 на виведення

Для програмування ліній *PB5*, *PB1*, *PB0 на виведення* необхідно встановити *5-й, 1-й та 0-й розряди регістра DDRB в одиницю*. *Інші* розряди регістра залишити *без змін*. Згідно [2] *адреса* регістра *DDRB: $0037*.

Тоді програма програмування ліній PB5, PB1, PB0 на виведення має вигляд:

LDI R31, $00; } Z(R31, R30) ← $0037
LDI R30, $37; R30 ← $37 ;
LD R19, Z; R19← DDRB
ORI R19, $23; R19← R19 ∨ 00100011B; R19.5←1, R19.1←1, R19.0←1, інші біти без змін;
ST Z, R19; DDRB←R19; DDRB. 5 ←1, DDRB. 1 ←1, DDRB. 0 ←1.

1.5.4. Завантаження регістра ICR1

16-розрядний регістр *ICR1 має адресу*: *$0047 (СБ) : $0046 (МБ)* [2].

В ICR1 треба завантажити: *TOP* = 40 = 00101000B = *$0028* (підрозд. 1.3).

Тоді програма завантаження регістра ICR1 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.
```

1.5.5. Завантаження регістра OCR1A

16-розрядний регістр *OCR1A* має *адресу*: *$004B (СБ): $004A (МБ)* [2]. В OCR1A треба завантажити: 20 = 00010100B = $0014 (підрозд. 1.3).

Тоді програма завантаження регістра OCR1A 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.
```

1.5.6. Програмування $К_{діл}$ = N = 8, режиму роботи номер 8 та запуск таймера T/C1

Вище у разі завантаження 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 (лекція 8).*

Для програмування *Кділ = 8* (табл. 2) та запуску таймера також використовують регістр TCCR1B (рис. 18), в який треба завантажити керувальне 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)	

Таблиця 2. Вибір джерела тактового сигналу таймерів/лічильників Tn

CSn2	CSn1	CSn0	Джерело тактового сигналу	
			T3 в моделях ATmega162x	Інші
0	0	0	Таймер/лічильник	Таймер/лічильник зупинений
0	0	1	$f_{clkI/0}$	$f_{clkI/0}$
0	1	0	$f_{clkI/0}/8$	$f_{clkI/0}/8$
0	1	1	$f_{clkI/0}/64$	$f_{clkI/0}/64$
1	0	0	$f_{clkI/0}/256$	$f_{clkI/0}/256$
1	0	1	$f_{clkI/0}/1024$	$f_{clkI/0}/1024$
1	1	0	$f_{clkI/0}/16$	Вивід Tn, лічба виконується за спадаючим фронтом
1	1	1	$f_{clkI/0}/32$	Вивід Tn, лічба виконується за наростаючим фронтом

Примітка. n = 1, 3, 4 або 5.

Адреса регістра *TCCR1B* дорівнює *$004E* [2]. Тоді програма програмування *Кділ = N = 8, режиму роботи номер 8 та запуску таймера T/C1має вигляд:*

LDI R17, $12; R17← $12;
LDI R29, $00; R29← $00;
LDI R28, $4E; R28← $4E;
ST Y, R17; TCCR1B ← R17 = $12.

1.6. Керувальна програма мовою С

1.6.1. Керувальна програма мовою С, яка відповідає завданню

```
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 #define 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; //змінна delay, яка використовується у разі зміни
//шпаруватості
16 // Init Phase and frequency correct PWM
17 TCCR1B = _BV(WGM13);  // 3
18 TCCR1A = 0; // 4
19 TCCR1A |= _BV(COM1A0); // 4
20 TCCR1A |= _BV(COM1A1);
21 TCCR1B |= _BV(CS11);  // 5
22 //об’явлення змінних-прапорців
23 char flag_start_stop=0,flag_reverse=0,flag_speedup=0,flag_speeddown=0;
24 while(1) // 6
25 {
26// 7: Start–Stop button action
27 if(!(PINB&4))
28 { flag_start_stop = 1; _delay_ms(10); }
29 if(( flag_start_stop==1 )&&(PINB&4))
30{PORTB^=1; flag_start_stop = 0; }   // 8
31// 9: Reverse button action
32 if(!(PINB&8))
33{ flag_reverse = 1; _delay_ms(10); }
34 if(( flag_reverse==1 )&&(PINB&8))
35{PORTB^=3; flag_reverse = 0; }  // 10
36 // 11: Speed –  button action
37 if(!(PINB&64))
38{ flag_speeddown=1; _delay_ms(10); }
```

```
39 if(( flag_speeddown==1 )&&(PINB&64))
40{ if (OCR1A!=40) OCR1A+=delay; flag_speeddown = 0; } // 12
41 // 13: Speed + button action
42 if(!(PINB&16))
43{ flag_speedup = 1; _delay_ms(10); }
44 if(( flag_speedup==1 )&&(PINB&16))
45{
46 if (OCR1A!=0)OCR1A– =delay; // 14
47flag_speedup = 0;
48}
49} // 6: End while
50} // 15: End of proram
```

1.6.2. Компіляція робочої програми, отримання hex-файлу та його завантаження у пам’ять комп`ютера

У [3] описано послідовність створення робочого проекту в Atmel Studio, компіляцію програми, створення hex-файлу та його *завантаження в пам’ять мікроконтролера.*

1.7. Розробка та опис структурної схеми пристрою керування двигуном постійного струму

Нижче на рис. 18 наведено схему електричну структурну пристрою керування двигуном постійного струму. Основним вузлом структури є мікроконтролер, який через вхідний паралельний порт отримує сигнали керування двигуном від чотирьох кнопок: запуск/зупинка двигуна; зміна напрямку обертання двигуна; збільшення швидкості обертання; зменшення швидкості обертання.

Через вихідний паралельний порт МК видає сигнали керування, які запускають чи зупиняють двигун та змінюють напрямок його обертання.

Для зміни швидкості обертання двигун використовується модуль таймера мікроконтролера, який формує імпульсний ШІМ-сигнал зі змінною шпаруватістю.

Для збільшення потужності сигналу, який подається в обмотку збудження двигуна використовується драйвер ШІМ (ДРШІМ)

Кварцовий резонатор (КР) визначає частоту високочастотного генератора, схема якого вбудована у МК.

Схема скидання (ССКИД) формує сигнал «RESET» (Скидання), який запускає виконання робочої програми.

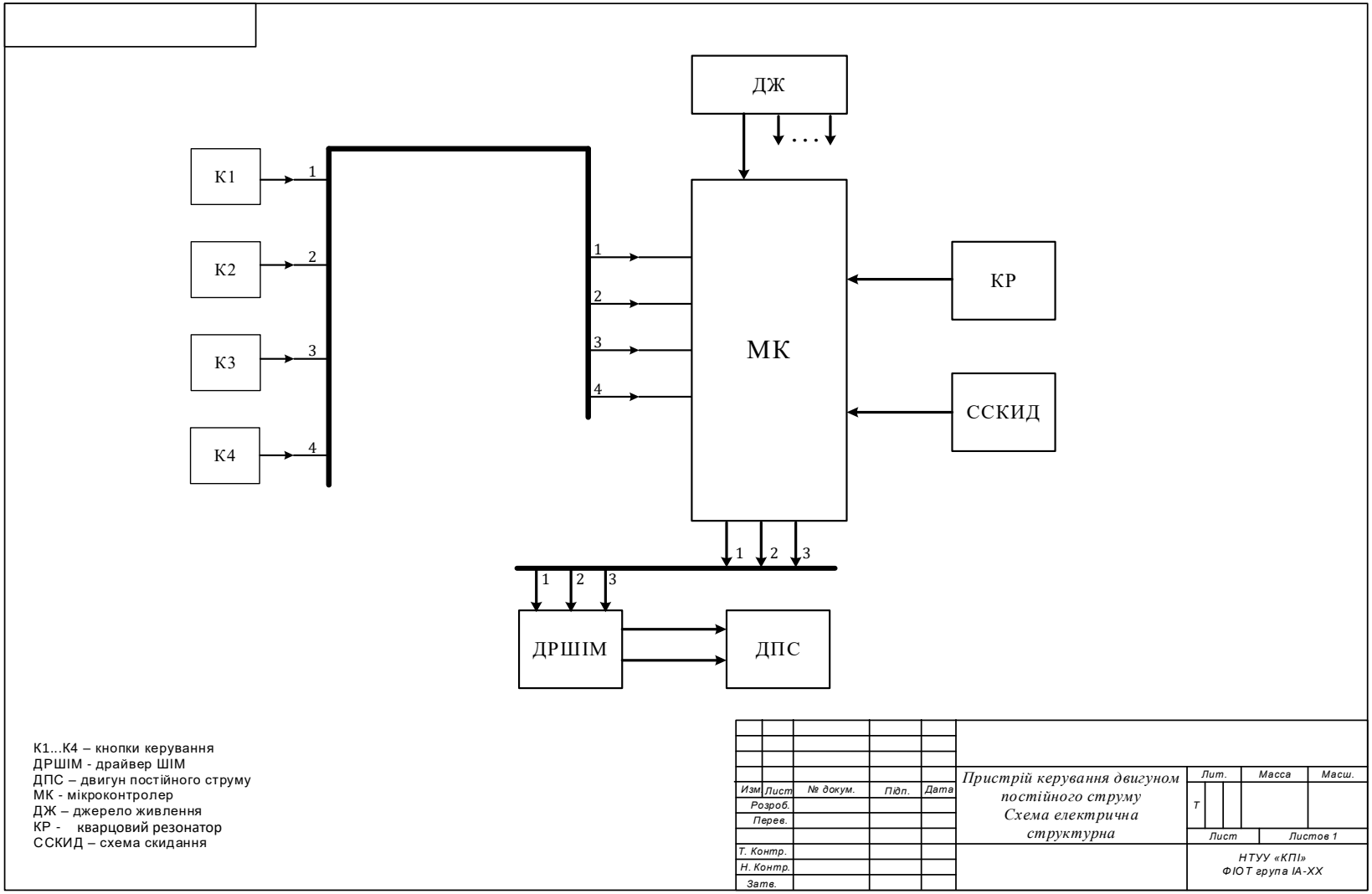


Рис. 18. Схема електрична структурна пристрою

Контрольні запитання та завдання

1. Для чого може використовуватись таймер/лічильник?
2. Для рішення яких завдань краще використання режиму Phase Correct PWM ніж режиму Fast PWM?
3. Для чого можуть використовуватись 16-розрядні таймери/лічильники?
4. Під час виникнення яких подій таймери/лічильники T1, T3, T4, T5 можуть генерувати переривання?
5. За допомогою яких регістрів виконується керування таймером/лічильником?
6. Які сигнали можуть використовуватись в якості тактового сигналу fclkn для таймерів/лічильників T1, T3, T4 і T5?
7. Яким чином здійснюється вибір джерела тактового сигналу, а також запуск і зупинка таймерів/лічильників?
8. Завдяки чому в режимі Phase and Frequency Correct PWM кожен період сигналу є повністю симетричним?
9. Поясніть, що таке ШІМ-сигнал?
10. Як обчислюється шпаруватість ШІМ-сигналу?
11. Як впливає значення шпаруватості на швидкість обертання двигуна?
12. Наведіть та поясніть схему алгоритму роботи моделі.
13. Наведіть та поясніть робочу програму на мові Асемблер, що керує моделлю.
14. Наведіть та поясніть робочу програму на мові СІ, що керує моделлю.

ЛЕКЦІЯ 10. МОДЕЛЮВАННЯ ПРИСТРОЮ КЕРУВАННЯ ДВИГУНОМ ПОСТІЙНОГО СТРУМУ В ПАКЕТІ PROTEUS.

ОРГАНІЗАЦІЯ ОБМІНУ ДАНИМИ В МПС

1. МОДЕЛЮВАННЯ ПРИСТРОЮ КЕРУВАННЯ ДВИГУНОМ ПОСТІЙНОГО СТРУМУ В ПАКЕТІ PROTEUS

1. Для початку моделювання необхідно *запустити модель* на виконання (рис. 1).

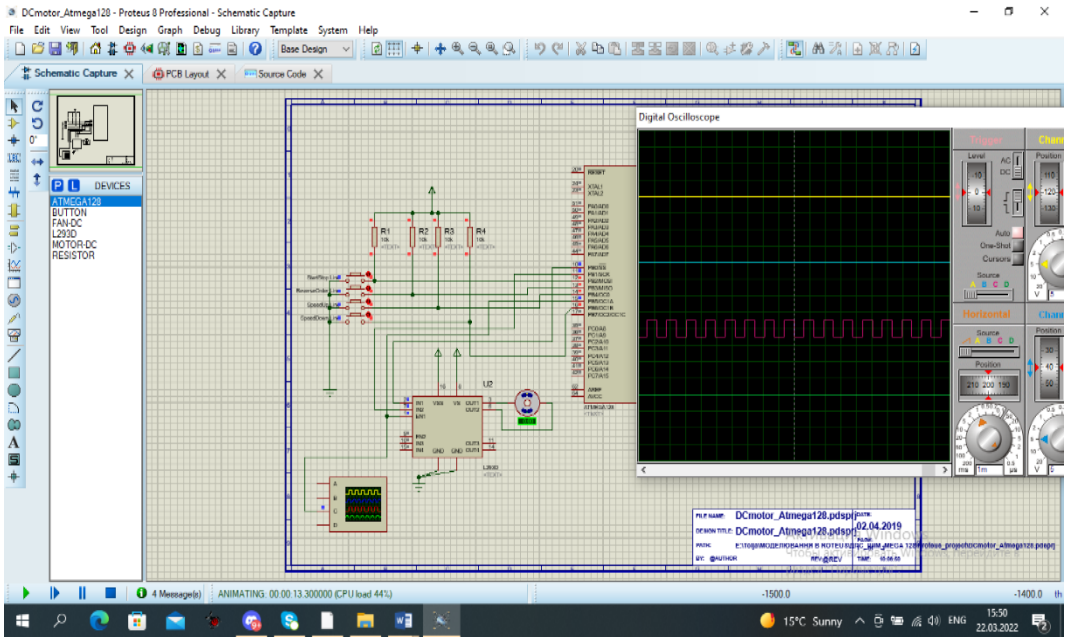


Рис. 1. Стан моделі після початку моделювання

Ми бачимо, що з лінії PB5 мікрококонтролера на осцилограф виводиться послідовність прямокутних імпульсів зі шпаруватістю 2, але *двигун не обертається*, тому що на лініях IN1 та IN12 присутні нулі.

2. Для запуску двигуна короткочасно натискаємо та відпускаємо кнопку «StartStop Line». Двигун починає обертатися *за годинниковою стрілкою* (рис. 2), тому що на лінію IN1 від мікроконтролера подається логічна одиниця, а на лінію IN2 – логічний нуль.

3. Для зупинки двигуна треба ще раз натиснути та відпустити кнопку «StartStop Line». Через інерційність двигуна він *зупиняється не миттєво*.

4. Для зміни напрямку обертання двигуна натискаємо та відпускаємо кнопку «ReverseOrder Line». Дигун починає обертатися *проти годинникової стрілки* (рис. 3), тому що на лінію IN1 від мікроконтролера подається логічний нуль, а на лінію IN2 – логічний одиниця.

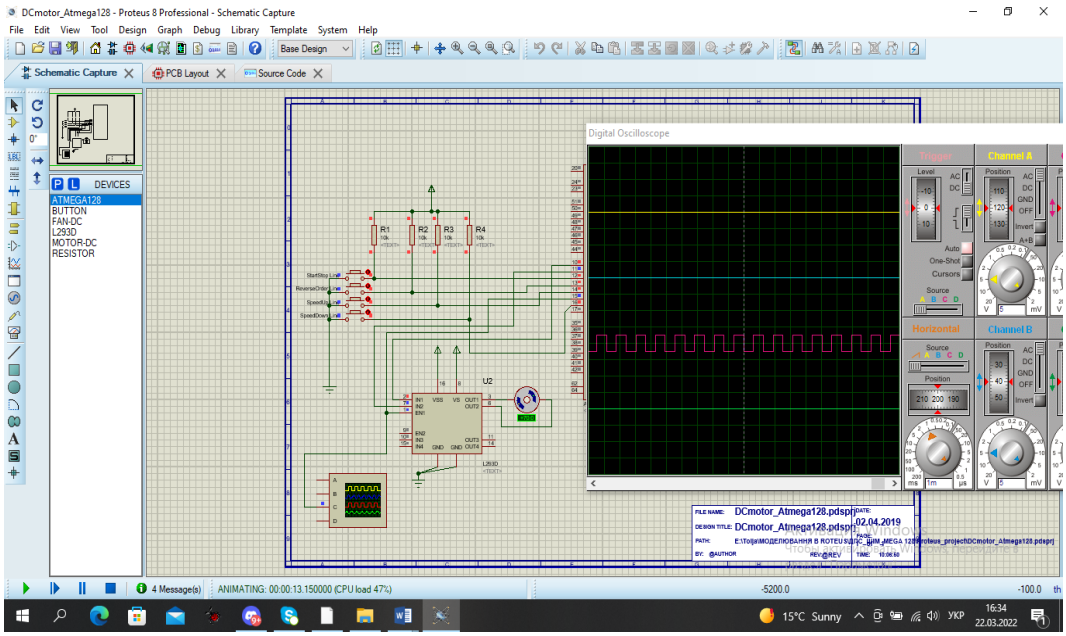


Рис. 2. Стан моделі після натискання та відпускання кнопки «StartStop Line»

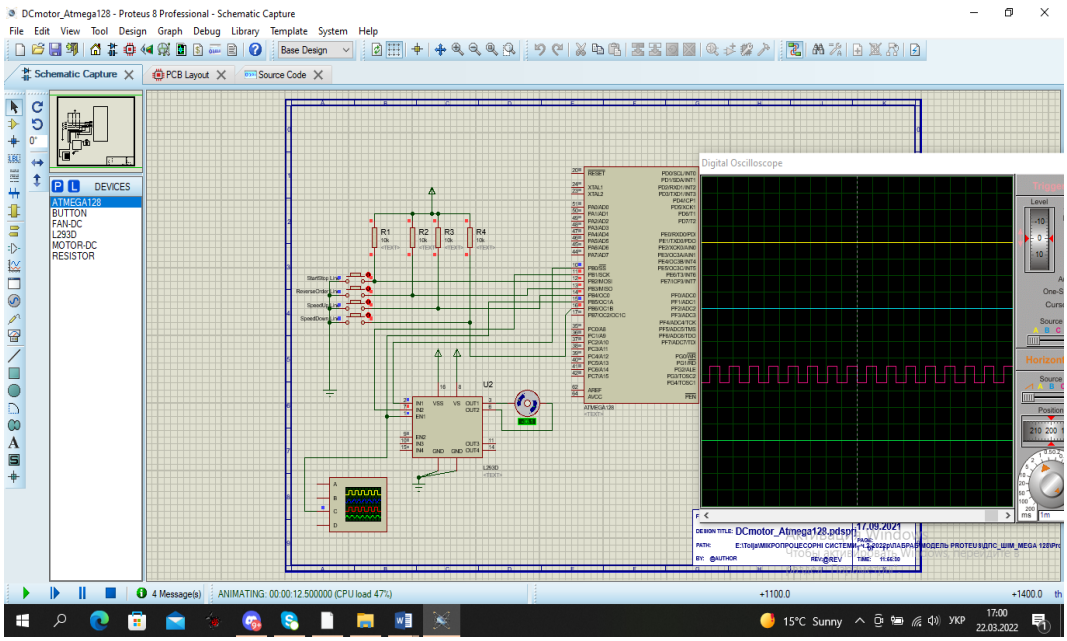


Рис. 3. Стан моделі після натискання та відпускання кнопки «ReverseOrder Line»

5. Для збільшення швидкості обертання двигуна треба декілька разів короткочасно натискати та відпускати кнопку «SpeedUp Line». *Шпаруватість* керуючих імпульсів *зменшується* (рис. 4), тому двигун починає обертатися швидше.

6. Для зменшення швидкості обертання двигуна треба декілька разів короткочасно натискати та відпускати кнопку «SpeedDown Line». *Шпаруватість* керувальних імпульсів *збільшується* (рис. 5), тому двигун починає обертатися повільніше.

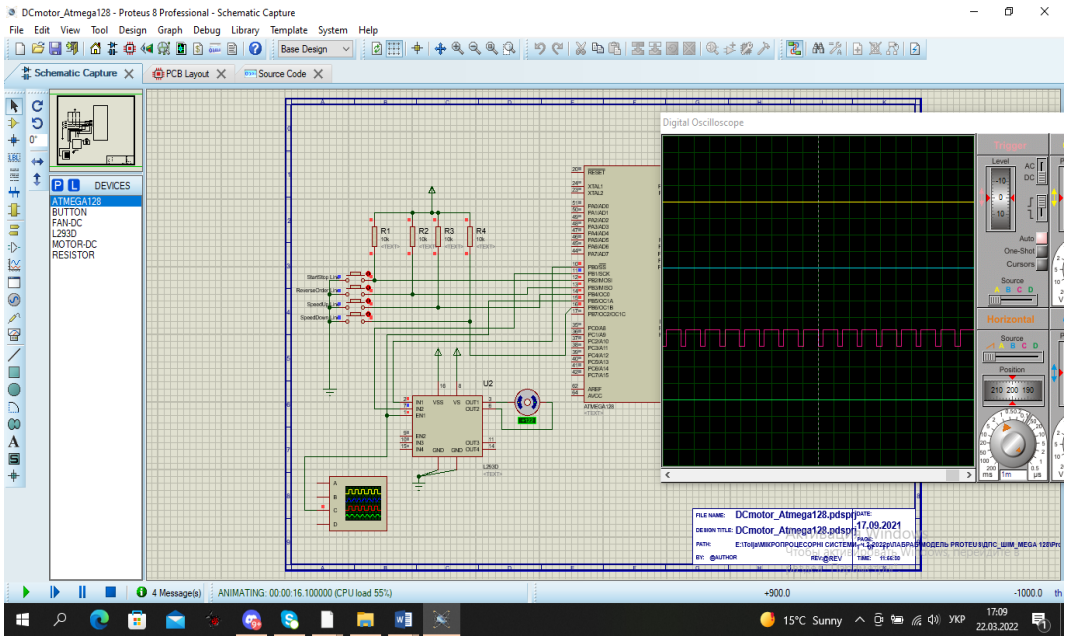


Рис. 4. Стан моделі після декількох натискань та відпускань кнопки «SpeedUp Line»

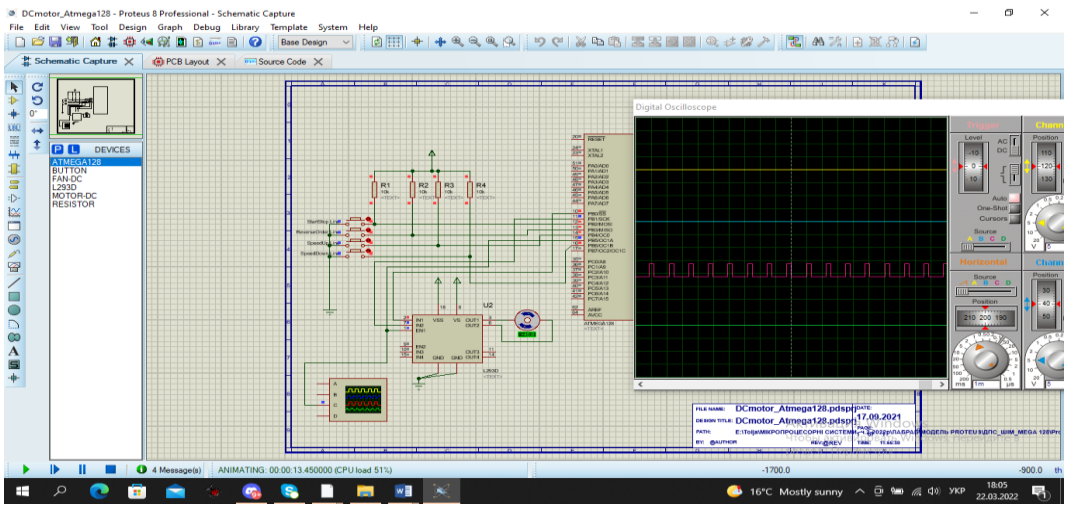


Рис. 5. Стан моделі після декількох натискань та відпускань «SpeedDown Line»

2. ОРГАНІЗАЦІЯ ОБМІНУ ДАНИМИ У МПС

2.1. Призначення та місце модуля введення/виведення у МПС

Під час вивчення модульної структури МПС (лекція 2) було відзначено, що *одним із трьох її основних* модулів є модуля введення/виведення (МВВ) що забезпечує взаємозв'язок МП із ЗВПР. Функціональні можливості мікропроцесорів досить обмежені, оскільки вони зазвичай *не містять* пам'ять достатнього об'єму і порти введення/виведення для зв'язку із зовнішніми пристроями. Тому МПС на основі МП *містять модулі пам'яті та введення/виведення*. Введення (читання, прийом) *відповідає потоку* даних від ЗВПР у МП/МК, а виведення (запис, передача) – *потоку даних* із МП/МК до ЗВПР.

Під час розгляду зв'язків між окремими елементами МПС зазвичай використовують *поняття інтерфейс*, що є границею між декількома пристроями, наприклад, між МП та ЗВПР.

Під інтерфейсом розуміють сукупність уніфікованих технічних і програмних засобів, необхідних для підключення пристроїв до системи чи однієї системи до іншої. *Серед властивостей інтерфейсу* можна відзначити розв'язування задач синхронізації, вибору напряму передачі даних, а також приведення у відповідність рівнів та форм сигналів. *Основу інтерфейсу* МК складають порти введення/виведення інформації, які виконано на основі регістрів.

Окрім наявності необхідних апаратних засобів *МВВ є програмованим*, тобто має відповідні регістри для запису *керувальних с*лів і регістри, які відображають *стан* інтерфейсу. Сучасні МП/МК *мають команди обміну* даними з необхідною периферією.

До *складу МВВ можуть входити*, наприклад, паралельний програмований інтерфейс (ППІ) і послідовний програмований універсальний синхронно/асинхронний приймач/передавач (УСАПП). Ці пристрої необхідні для *організації паралельного та послідовного обміну* інформацією між ЗВПР і МП.

Окрім поняття «інтерфейс» в МПС також використовують *термін «система (підсистема)* введення/виведення», в який включають, насамперед, *порти* введення/виведення та *їхню програмну підтримку* [1; 2].

Окремі МК, наприклад, сімейства AVR, *мають* внутрішню (резидентну) пам'ять і *порти введення/виведення*, кількості яких може бути недостатньо, тому вони *використовують також зовнішні* пристрої обміну та пам`ять.

2.2. Способи обміну даними в МПС

В залежності від особливостей програмування обміну даними в МПС використовуються *два способи*: під керуванням програми (програмно-керований) та за перериваннями.

Програмно-керований обмін виконується з ініціативи МП і під його керуванням. МП постійно опитує стан ЗВПР за допомогою *читання* відповідного *прапорця готовності*. Якщо ЗВПР *готовий* до обміну, тобто встановлено у логічну *одиницю прапорець* готовності, виконується операція обміну: введення чи виведення даних. Якщо ЗВПР *не готовий* до обміну, то МП *очікує готовності*, постійно виконуючи відповідні команди програми і періодично аналізуючи готовність ЗВПР. Такий *спосіб не завжди є ефективним*, особливо у випадку, коли треба довго чекати та програма нічого не робить окрім постійної перевірки готовності ЗВПР до обміну.

Більш ефективним є *обмін за перериванням*, коли у разі необхідності обміну, викликається підпрограма, яка керує обміном. Після закінчення обміну, передається

керування перерваній програмі. У [1] *розглянуто ці види обміну* на прикладі використання мікросхеми ППІ і8255.

В залежності від виду існують *два способи* обміну: паралельний, коли одночасно передаються всі або декілька бітів слова даних, або послідовний, коли біти слова даних пересилаються по черзі, починаючи, наприклад, з його молодшого розряду. У разі *паралельного* обміну ЗВПР зв'язується з МП лініями зв'язку, довжина яких обмежена і складає кілька метрів.

У разі *послідовного* обміну даними та використанні, наприклад, інтерфейсів: CAN, RS-485, модемів, і т ін., довжину ліній зв'язку можна суттєво збільшити.

Обмін інформацією всередині МП/МК здійснюється *в паралельній* формі. Якщо використовується *послідовний обмін* даними, необхідно: *у разі передачі* даних від МПС до ЗВПР – перетворити дані з паралельної форми в послідовну, а *у разі прийому* інформації від ЗВПР та введенні її у МПС – перетворити з послідовної форми в паралельну. Процес *перетворення даних* з паралельної форми в послідовну наведено на рис. 6.

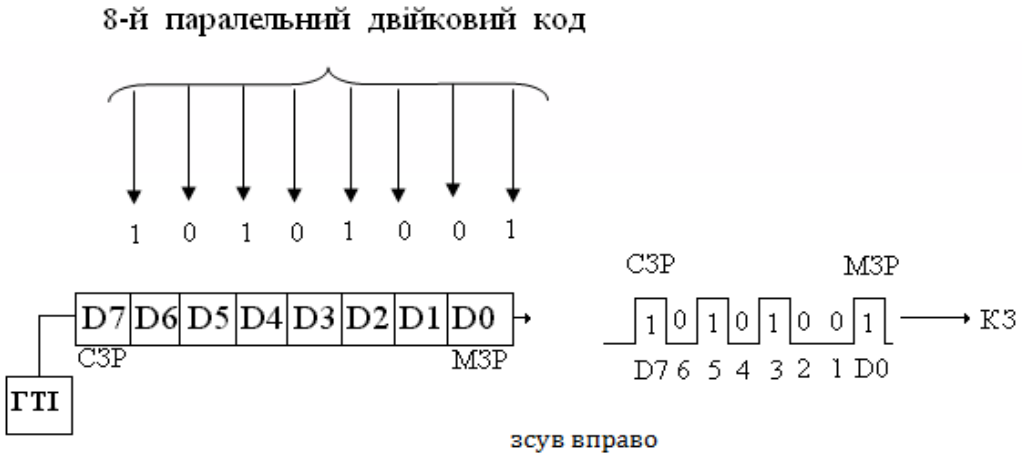


Рис. 6. Перетворення даних з паралельної форми в послідовну

Інформація в паралельному ДВК завантажується в *регістр зсуву*. Його вміст під впливом тактових імпульсів від генератора тактових імпульсів (ГТІ) послідовно *зсувається* на один розряд, наприклад, *праворуч*. Дані *на виході* такого регістра будуть мати послідовну форму. Переважно під час послідовної передачі в канал зв'язку (КЗ) *першим передається молодший біт* (розряд) слова даних (МЗР), останнім – старший біт (СЗР).

Для зворотного перетворення даних з послідовної форми в паралельну необхідно виконати дії, зворотні стосовно 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 в МП.

Подібні перетворення виконуються у мікроконтролері у разі обміну даними через його послідовний порт (рис. 7).

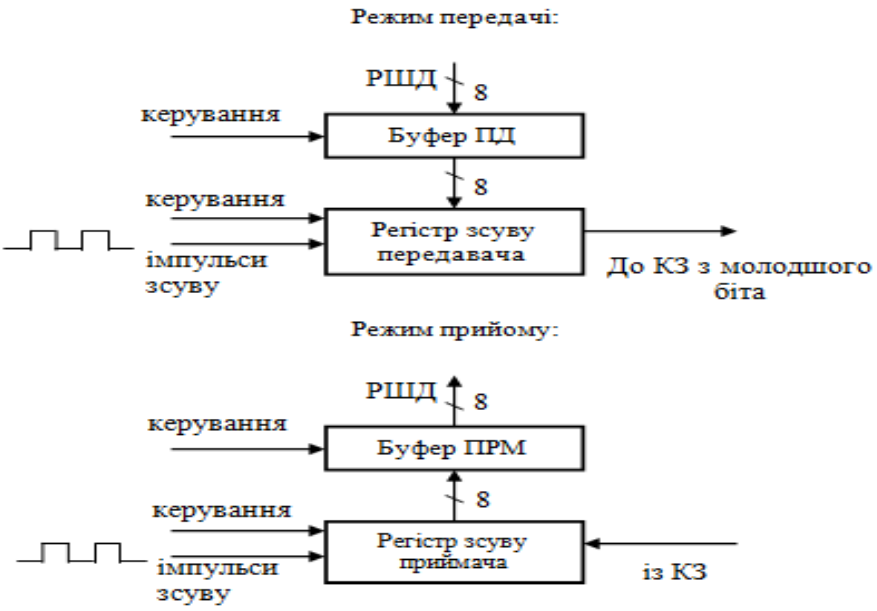


Рис. 7. Спрощена структура послідовного порту МК

2.3. Адресація пристроїв введення/виведення

В МПС на основі мікропроцесора адресація ПВВ виконується одним із *двох способів: командами* введення (IN) чи виведення (OUT), або *командами звернення до ЗВПР, як комірок пам'яті.*

У другому випадку МП розглядає порти введення/виведення як звичайні комірки пам'яті. Заздалегідь *з'ясовується діапазон пам'яті*, адреси якого привласнюються портам. Це дозволяє використовувати всі операції над вмістом комірок пам'яті для роботи з портами введення/виведення. Наприклад, з'являється *можливість виконувати з портами* арифметичні чи логічні операції. *Недоліками* цього способу є те, що, по-перше, у низці випадків може *збільшитися час доступу* до порту, а, по-друге, *частина адресного простору пам'яті* передається портам.

У МПС на основі мікроконтролера, наприклад, AVR, *порти*, за допомогою яких відбувається обмін інформацією із ЗВПР, *входять до складу МК*. *Для звернен*ня до портів використовують *команди з прямою* адресацією (див. п. 1.2.7).

2.4. Підключення ПВВ до системної шини та зовнішніх пристроїв

Це питання розглянуто у лекції 2 на прикладі обміну даними у *гіпотетичній* мікропроцесорній системі керування з використанням «інтелоподібного» мікропроцесора i8080 та паралельного програмованого інтерфейсу (ППІ) – i8255.

Спрощену *структуру* ППІ наведено на рис. 8.

Інтерфейс має 2 виводи для подачі живлення, 8 – для зв'язку із системною шиною даних, 6 – для подачі керувальних сигналів і 24 виводи, що можуть окремо програмуватися по *дві групи з дванадцяти виводів* і використовуватися в *трьох режимах* роботи: 0, 1 і 2.

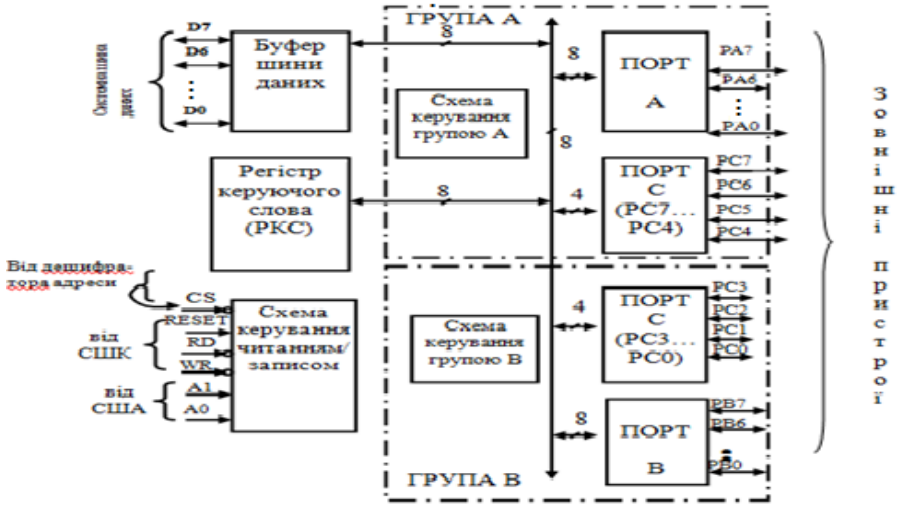


Рис. 8. Спрощена структурна схема ППІ і8255

У режимі 0 кожну групу з дванадцяти виводів може бути запрограмовано на просте (*нестробоване) введення чи виведення.*

У режимі 1 вісім виводів кожної групи програмуються на *стробоване* введення чи виведення, а інші використовуються для керування програмним обміном чи обміном за перериванням.

У режимі 2 мікросхема *в основному* використовується в якості двонапрямного 8-розрядного каналу обміну даними через порт А, який керується сигналами п’яти виводів порту С.

Режим обміну задається програмно пересиланням у регістр керувального слова (РКС) інтерфейсу *керувального слова* відповідного формату.

Стан інтерфейсу також контролюється програмою за допомогою зчитування *слова стану* (порт С).

ППІ *складається* з портів введення/виведення: А, В і С, схем керування групами ліній А і В, буфера шини даних, регістра керувального слова і схеми керування читанням/записом. *Порти А, В і С призначено* для прийому і збереження інформації, що пересилається між МП і зовнішніми пристроями ((ЗВПР).

Порт А містить вхідний і вихідний 8-розрядні регістри з формувачами та може використовуватися для введення чи виведення інформації *у всіх трьох режимах. Порт В* складається з 8-розрядного регістра введення/виведення з формувачами та може працювати на введення чи виведення даних *у режимах 0 і 1*. *Порт С* має два 4-розрядних регістри з формувачами. Ці регістри можуть застосовуватися для введення чи виведення інформації *в режимі 0*. Під час роботи портів А і В у режимах 1 чи 2 більша частина виводів порту С використовуються для прийому і видачі керувальних сигналів обміну, а *регістри порту С виконують функцію регістрів стану. Особливістю порту С* є можливість програмного встановлення/скидання його окремих розрядів. *Буфер шини даних* (БШД) є тристабільним, двонапрямним і призначений для підключення внутрішньої шини

даних ППІ до системної шини мікропроцесорної системи. *Передача інформації* (D7...D0) через буфер виробляється *за командами* мікропроцесора.

МП передає інтерфейсу дані для зовнішнього пристрою чи керувальні слова для програмування ППІ, а *приймає* інформацію від зовнішніх пристроїв чи слова стану інтерфейсу. *Схема керування читанням/записом* керує усіма внутрішніми та зовнішніми пересиланнями інформації. Для цього використовуються *сигнали*: $\overline{CS}$ – вибір кристала ППІ, що звичайно надходить від селектора (дешифратора) адреси; $\overline{RD}$ – читання, що дозволяє передачу на системну шину даних інформацію від зовнішнього пристрою чи слово стану ППІ (режим введення); $\overline{WR}$ – запис, керує передачею даних чи керувального слова із системної шини даних (режим виведення); A1, A0 – значення молодших розрядів системної адресної шини, що необхідні для вибору одного з портів ППІ чи РКС; RESET – скидання, що *обнуляє усі* внутрішні *регістри* інтерфейсу і перемикає його порти *в режим 0 для введення* інформації.

Вплив зовнішніх керувальних сигналів на роботу ППІ пояснює табл. 1.

Таблиця 1. Напрям і вид обміну даними між МП і ЗВПР

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A1	A0	Напрям та вид обміну
					МП→ЗВПР (виведення, запис, передача)
0	1	**0**	**0**	**0**	Запис даних до порту А
0	1	**0**	**0**	1	Запис даних до порту В
0	1	**0**	1	**0**	Запис даних до порту С
0	1	**0**	1	1	Запис даних до РКС
					ЗВПР → МП (введення, читання, прийом)
0	**0**	1	**0**	**0**	Читання порту А
0	**0**	1	**0**	1	Читання порту В
0	**0**	1	1	**0**	Читання порту С
1	x	x	x	x	Схему відключено (високоімпедансний стан)
0	**0**	1	1	1	Заборонений стан

Керувальні сигнали для системної шини виробляє мікропроцесор. *Вибір режиму роботи* інтерфейсу здійснюється записом керувального слова до РКС у будь-якому місці програми.

На рис. 9 наведено *підключення* ППІ до системної шини.

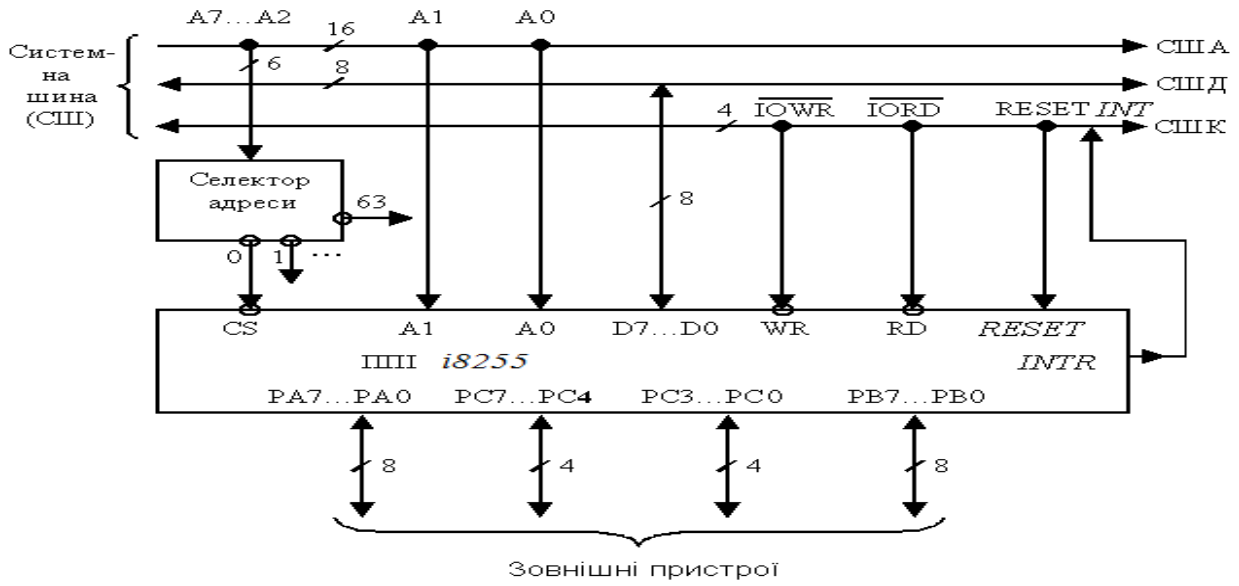


Рис. 9. Підключення ППІ до системної шини

Більш детально роботу ППІ в різних режимах та його програмування описано у [1].

2.5. Порти введення/виведення AVR-мікроконтролерів

2.5.1. Загальна характеристика портів ведення/виведення

Мікроконтролери сімейства AVR *мають модуль введення/виведення (МВВ)*, *апаратну частину якого* виконано на базі портів введення/виведення (регістрів). Кожен *порт має певне число виводів*, через які МК може приймати або передавати цифрові сигнали [2].

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*.

В портах введення/виведення МК сімейства реалізовано *функцію* «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 окремих моделей сімейства *можуть мати різну* кількість портів і, відповідно, контактів введення/виведення [2]. В усіх МК сімейства *переважна більшість контактів введення/виведення мають додаткові (альтернативні функції)* та використовуються відповідними периферійними пристроями.

2.5.2. Звернення до портів введення/виведення

*Звернення до порт*ів виконується через регістри введення/виведення (РВВ). За деякими винятками, під кожен порт в адресному просторі введення/виведення зарезервовано *по три адреси* [2]. За цими адресами розміщуються *три регістри*: *регістр даних* порту PORTx, *регістр напряму* даних DDRx і *регістр виводів* порту PINx. Дійсні назви регістрів *отримуються підста*новкою 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*. *У разі виведення* у порт відбувається запис у відповідний регістр даних порту PORTx. *Під час введення* виконується читання регістра виводів порту PINx (сигналів, які присутні на виводах порту). *Під час читання* регістра PORTx у МК вводяться дані з регістра-защіпки порту. У разі знаходження МК *у стані скидання* виводи всіх портів перебувають у третьому (високоімпедансному) стані – Hi-Z [1]. *Адреси регістрів*, що відносяться до портів введення/виведення, наведено у [2].

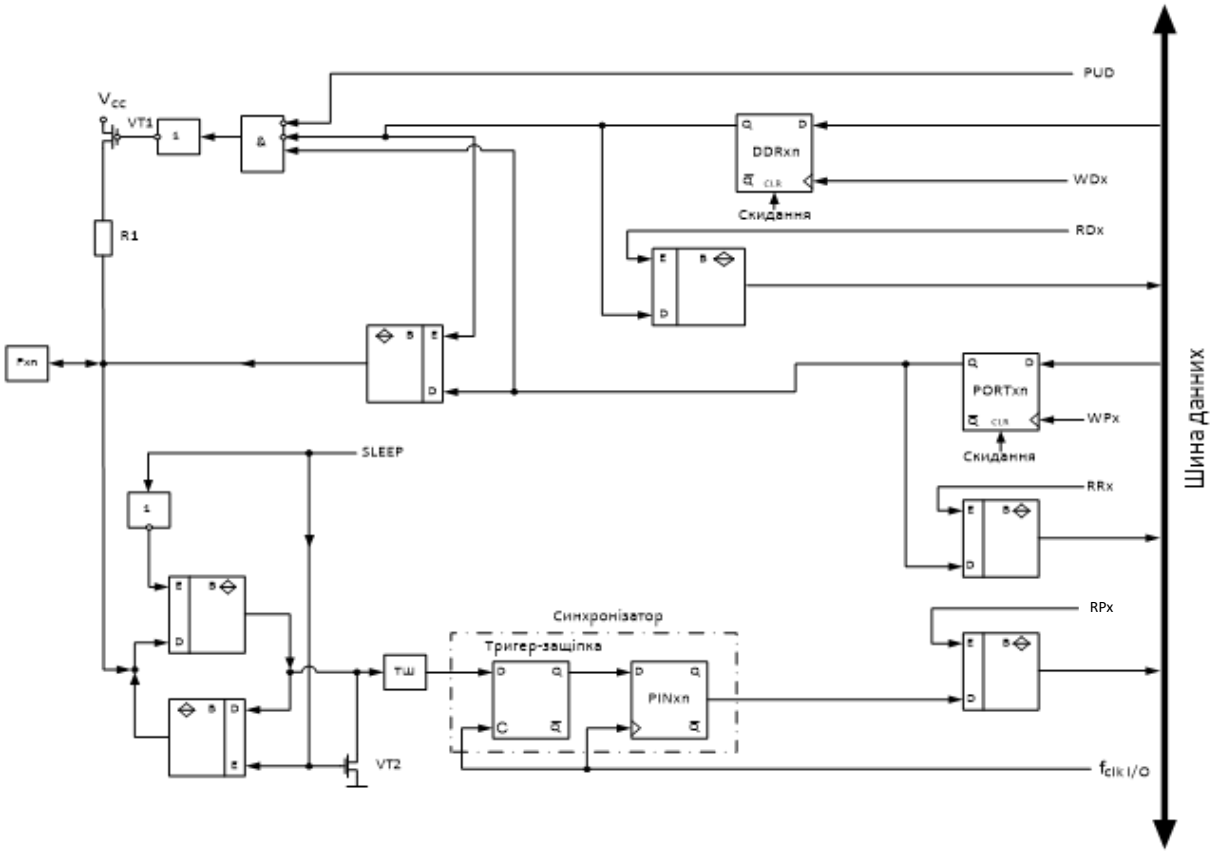
2.5.3. Структура портів введення/виведення

Спрощену структурну схему одного з каналів порту введення/виведення Pxn під час його роботи в якості цифрового входу/виходу загального призначення наведено на рис. 10.

Кожному каналу порту відповідають *розряди трьох регістрів* введення/виведення: PORTxn, DDRxn і PINxn. 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 вище 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 одиничного керувального сигналу – «E» *буде активним*. *Виходи інших* (не обраних) буферів на час передачі переводяться *в третій* стан.

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*.



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 МК.

Рис. 10. Структурна схема контакту введення/виведення

Розряд PORTxn регістра PORTx виконує *подвійну функцію*. Якщо вивід функціонує *як вихід* (DDRxn = 1), цей розряд визначає сигнал на виводі порту. Якщо розряд регістра PORTx 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. У разі встановлення *розряду* PORTxn в *одиницю* підтягуючий резистор підключається між виводом МП і живленням, за умови, що біт PUD = 0.

Керування підтягуючими резисторами для всіх виводів портів здійснюється розрядом *PUD* регістра спеціальних функцій *SFIOR* або регістра керування MCUCR [2]. 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 МК.

2.5.4. Конфігурування виводів портів введення/виведення

Різні *можливості конфігурування* виводів портів введення/виведення наведено у табл. 2.

Таблиця 2. Конфігурації виводів портів

DDRxn	PORTxn	PUD*	Функція виводу	Резистор	Примітки
0	0	X	Вхід	Відключений	Третій стан (Hi-Z)**
0	1	0	Вхід	Підключений	У разі підключення 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 МК (*незалежно від розряду* DDRxn) може бути отримано шляхом *читання розряду PINxn* регістра PINx. У цьому разі *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. 9, *з тригера-защіпки* та *розряду PINxn*.

Значення сигналу на виводі МК *фіксується* тригером-защіпкою *за високим* рівнем тактового сигналу і *переписується* потім у тригер PINxn за *наростаючим фронтом* тактового сигналу $f_{CLKI/O}$. Відповідно, *величина затримки* може становити від 0,5 до 1,5 періоду системного тактового сигналу.

З цієї ж причини *між операціями зміни і повторного зчитування станів виводів* необхідно вставляти команду NOP. У [2] наведено *приклад конфігурування* одного з портів МК.

Переважна більшість контактів введення/виведення МК сімейства *мають додаткові функції* і можуть використовуватися різними периферійними пристроями. В цьому випадку користувач повинен або *самостійно задавати конфігурацію* виводу, *або – вивід конфігурується автоматично* під час включення відповідного периферійного пристрою [2].

Контрольні запитання та завдання

1. Поясніть призначення та місце ПВВ у МПС.
1. Дайте визначення поняттю «інтерфейс».
2. Що являє собою послідовний обмін даними між зовнішніми пристроями та МП/МК?
3. Чим відрізняються паралельний обмін від послідовного?
4. Які перетворення даних відбуваються у послідовному інтерфейсі?
5. Поясніть особливості обміну під керуванням програми та за перериванням.
6. Як адресуються ЗВПР у МПС?
7. Поясніть особливості програмування обміну даними у МПС.
8. Які способи адресації ПВВ є у МПС на основі МП i8086?
9. Як відбувається звернення до паралельних портів AVR-МК?
10. Наведіть та опишіть спрощену структурну схему одного з каналів паралельного порту введення/виведення AVR-МК?
11. Скільки регістрів та як використовуються для програмування паралельних порти AVR-МК?
12. Як здійснюється керування підтягуючими резисторами портів.
13. Наведіть приклад конфігурування виводів портів введення/виведення.

ЛЕКЦІЯ 11. АРХІТЕКТУРА ПОСЛІДОВНОГО ІНТЕРФЕЙСУ

1. АРХІТЕКТУРА ПОСЛІДОВНОГО ІНТЕРФЕЙСУ

1.1. Загальна характеристика інтерфейсу

Більшість МК 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 [2].

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 синхронно/асинхронний приймач-передавач.

Всі AVR-МК *сімейства Mega* мають модулі УСАПП, які, у разі роботі в асинхронному режимі сумісні з модулями УАПП як за розміщенням розрядів керувальних регістрів, так і за функціонуванням.

Всі модулі УАПП/УСАПП підтримують *дуплексний* обмін за послідовним каналом. У цьому разі *швидкість* передачі даних *може змінюватись* у доволі широких межах. У модулях УАПП пакет даних 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 та некоректний старт-біт. В модулях є також корисна функція – *фільтрація завад*.

Для програмування модулів є *3 переривання*, запит на генерацію яких формується під час виникнення наступних подій: передачу завершено; регістр даних передавача пустий та прийом завершено.

Виводи МК, що використовуються модулями УАПП/УСАПП, *є лініями портів* введення/виведення загального призначення. *Як приклад*, виводи деяких МК, що використовуються модулями УСАПП сімейства Mega, та їх функції *наведено* у [2].

1.2. Опис структури модулів УАПП/УСАПП

Спрощену структурну схему модуля УАПП/УСАПП наведено на рис. 1.

Елементи, які виділені на рисунку *сірим кольором*, є тільки у складі модулів УСАПП. Модулі мають *три частини*: блок тактування, блок передавача та блок приймача. *Блок тактування* включає схему синхронізації, яка використовується під

час роботи у синхронному режимі, і контролер швидкості передачі. В модулях УАПП блок тактування складається *тільки з контролера* швидкості передачі.

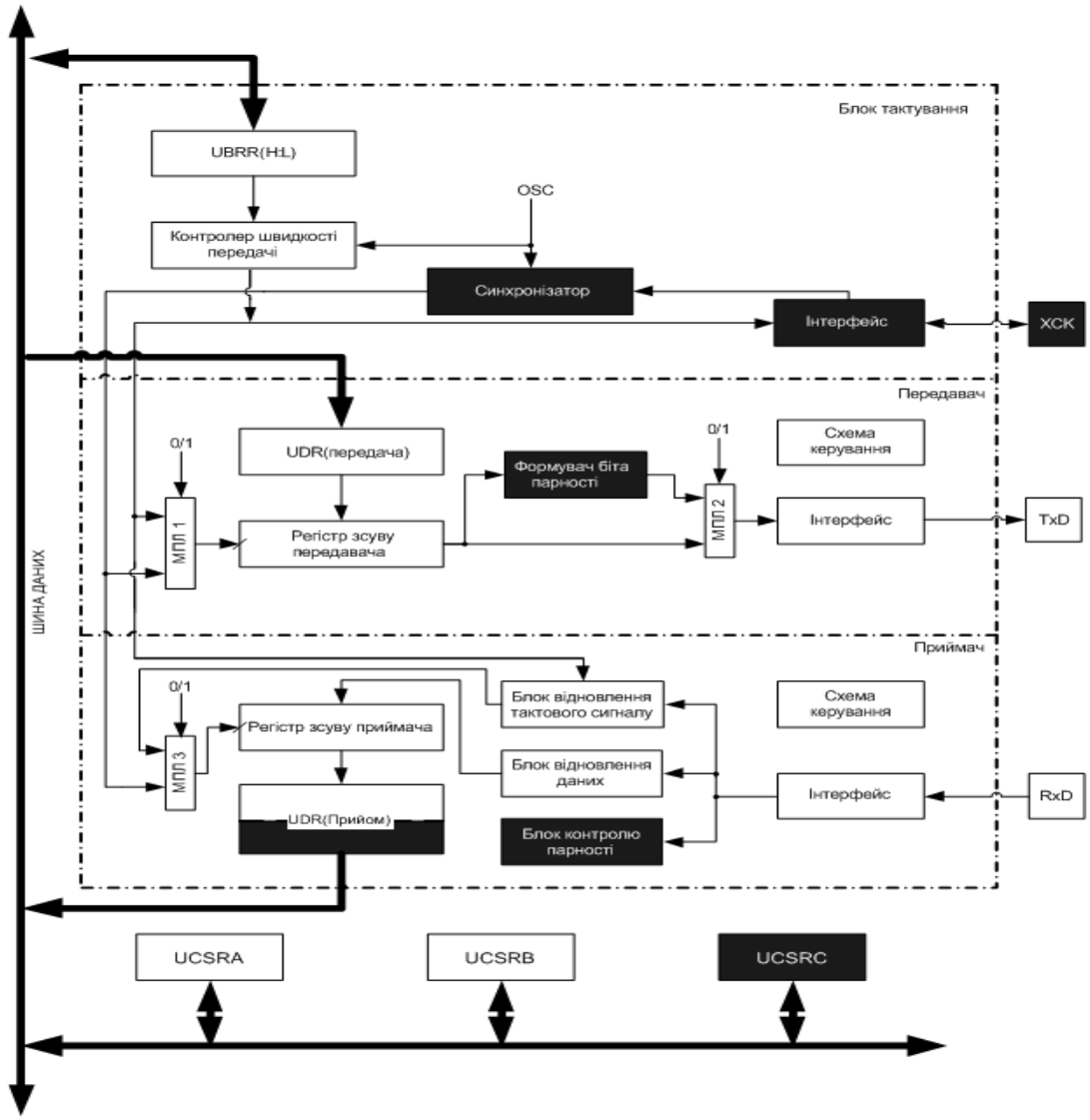


Рис. 1. Спрощена структурна схема модуля УАПП/УСАПП

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 (тільки УСАПП); *буфер* UDR: *дворівневий* в УСАПП та однорівневий в УАПП; регістр зсуву, а також схему керування.

Буферні регістри приймача і передавача розміщуються за однаковою адресою простору РВВ і позначаються *як регістр даних UDR* (Universal Data Register – UDRn, де n = 0, 1, 2 або 3). 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 МК наведено у [2].

Дворівневий буфер приймача у модулях УСАПП є *FIFO-буфером* (First In First Out, першим зайшов, першим вийшов). *Зміна* стану буфера *відбувається* за будь-яких звернень 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.

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 [2].

1.3. Швидкість прийому-передачі даних

На рис. 2 в якості прикладу наведено структурну схему *блоку синхронізації* модуля УСАПП мікроконтролера Mega 128.

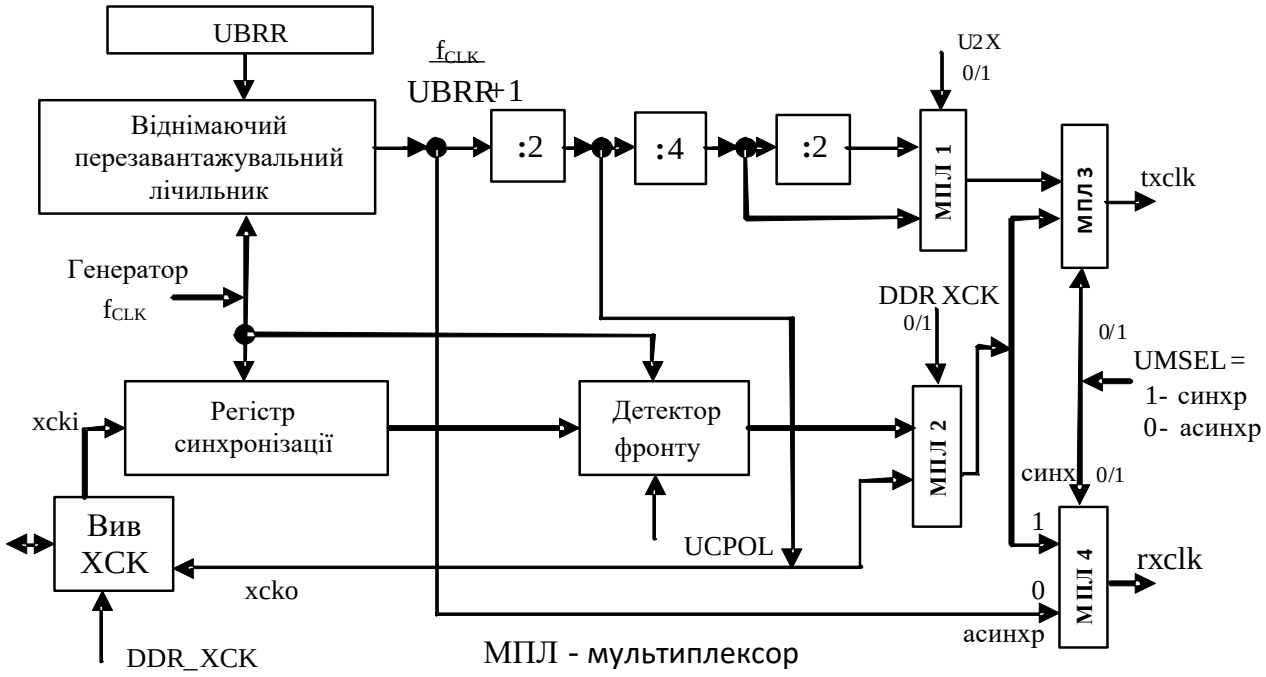


Рис. 2. Структурна схема блоку синхронізації модуля УСАПП

На схемі використовуються *наступні сигн*али: 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.

В асинхронному режимі, а також у *синхронному* режимі під час роботи в 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 передавач*а – через додатковий дільник, коефіцієнт ділення якого: 2, 8 чи 16 залежить від режиму роботи модуля УАПП/УСАПП.

Для програмування асинхронного чи синхронного режиму роботи призначено *біт UMSEL* в 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 [2].

Під час використання *синхронного* режиму біт DDR_XCK в регістрі напряму даних для виводу XCK задає, *чи буде синхронізація внутрішньою*: DDR_XCK = 0 (ведений режим) *чи зовнішньою*: DDR_XCK = 1 (ведучий режим). Вивід XCK *активний тільки* у разі використання 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, назви та особливості використання цих регістрів наведено у [2].

Під час роботи *в асинхронному* режимі швидкість обміну визначається не тільки вмістом регістра UBRR, але й *станом розряду* U2X (U2Xn) регістра UCSRA (UCSRnA) [2]. Якщо *цей розряд* встановлено *в одиницю*, коефіцієнт ділення попереднього дільника зменшується у два рази, а швидкість обміну відповідно *подвоюється*. Під час роботи в *синхронному* режимі цей розряд має бути *скинуто*.

Швидкість обміну *визначається* наступними формулами:

– звичайний асинхронний режим (U2Xn = 0):

$$BAUD = fclк/(16(UBRR+1)); \quad (1)$$

– пришвидшений асинхронний режим (U2Xn = 1):

$$BAUD = fclк/(8(UBRR+1)); \quad (2)$$

– синхронний ведучий режим:

$$BAUD = fclк/(2(UBRR+1)), \quad (3)$$

де BAUD – *швидкість* передачі у *бодах*, fclк – тактова частота МК а, UBRR – *вміст* регістра контролера швидкості передачі: *0...4095*.

В *якості прикладу* у [2] наведено значення регістра UBRR, що дозволяють отримати стандартні для асинхронного режиму швидкості передачі у разі використання деяких резонаторів, а також значення похибок, що отримуються, відносно стандартних швидкостей.

1.4. Формат кадру

Сукупність одного байта даних і додаткової інформації під час передачі називається *кадром* (рис. 3).

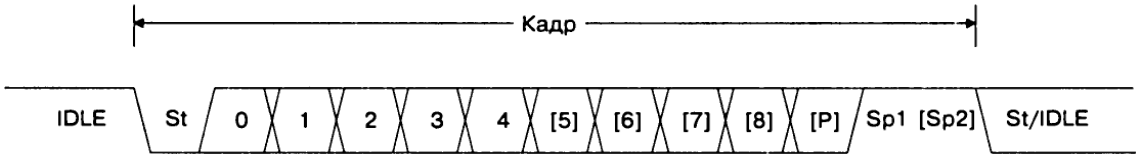


Рис. 3. Формат кадру

На цьому рисунку використано наступні скорочення: St – старт-біт, завжди має низький рівень; 0…8 – номера бітів даних; P – біт паритету: парність або непарність; Sp1, Sp2 – стоп-біти, завжди мають високий рівень; IDLE – стан очікування, в якому призупинено обмін лінією RxD/TxD.

У *стані очікування* (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 в *асинхронному режимі* використовується схемою синхронізації приймача для *підлаштування фази* синхрочастоти приймача під отримувані посилки (біти). Далі передаються біти слова даних, починаючи з *молодшого* розряду. 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) [2]. Розмір слова даних в модулях USART визначається відповідно до табл. 1.

Таблиця 1. Визначення розміру слова даних в 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

В *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 кадру 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. 2.

Таблиця 2. Керування контролем парності

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

Примітка: n = 0, 1, 2 чи 3.

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

$$P_{EVEN} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1,$$
$$P_{ODD} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0. \qquad (4)$$

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

1.5. 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. 4*а*).

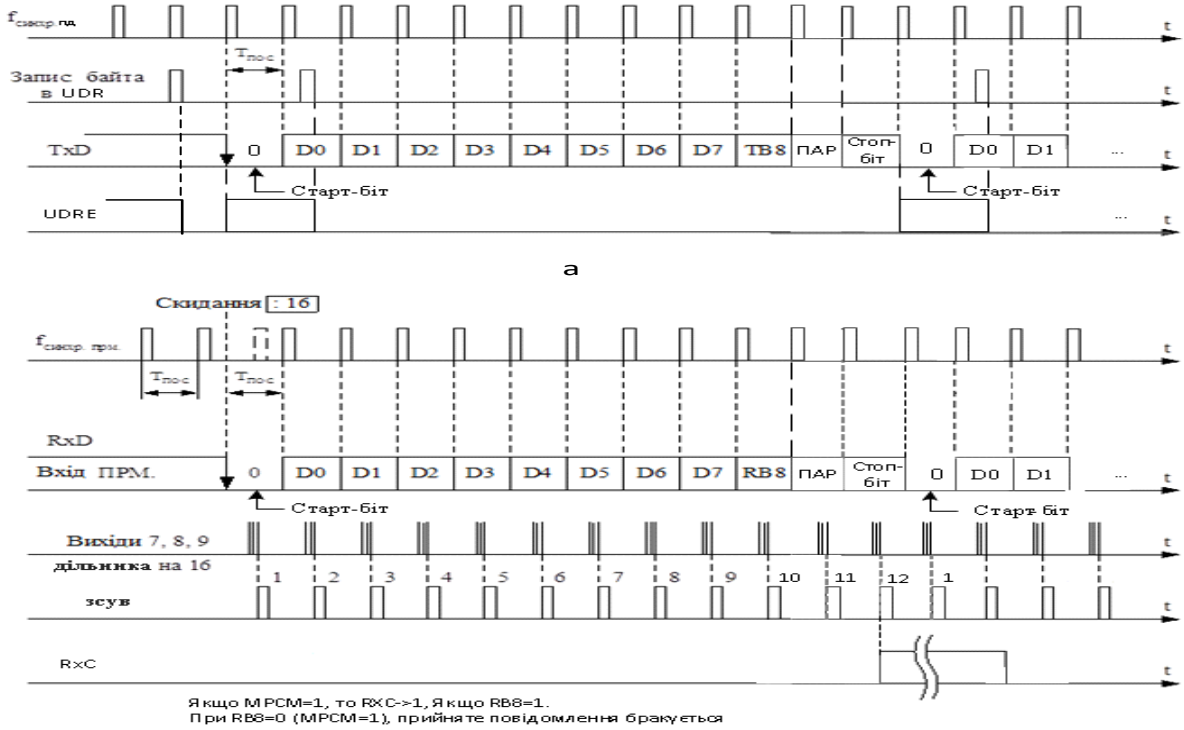


Рис. 4. Часові діаграми роботи інтерфейсу в асинхронному режимі: а – передача; б – прийом даних

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 в розряд TXB8 (TXB8n) *до запису байта* в регістр даних.

Можливі *два варіанти реакції на запис* даних в 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 в регістр зсуву одразу ж після запису в регістр UDR; якщо запис в регістр UDR здійснюється під час передачі, то в цьому випадку дані пересилаються в регістр зсуву після передачі останнього стоп-біта поточного кадру.

Після пересилання слова даних в 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 байта в 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 підрозд. 1.4. Швидкість зсуву визначається налаштуванням контролера тактових сигналів. У разі роботи в *синхронному* режимі *зміна стану виводу* TXD (TXDn) відбувається за одним з фронтів сигналу XCK (XCKn) (рис. 5).

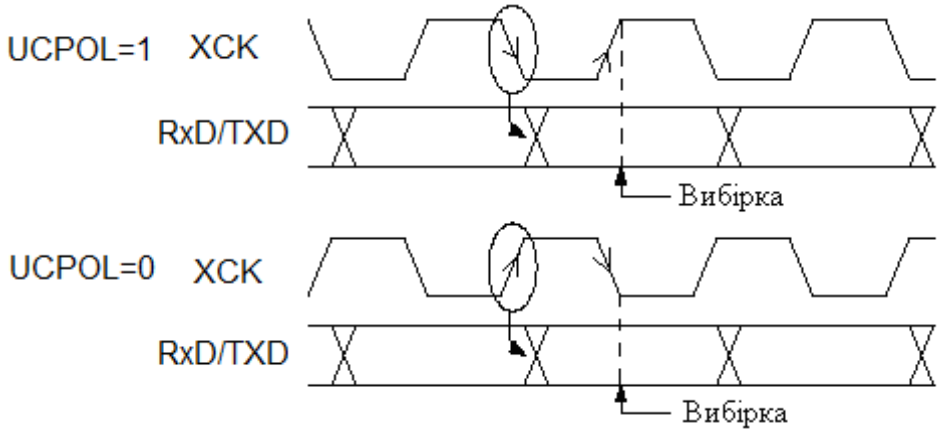


Рис. 5. Часові діаграми для синхронного режиму роботи модуля УСАПП

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.

Якщо під час передачі попереднього слова в 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 під час входу в підпрограму обробки переривання або програмно, записом в цей розряд одиниці.

Відключення передавача здійснюється скиданням розряду TXEN (TXENn) регістра UCSRB (UCSRnB). *Якщо в 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. *У разі вимкненого передавача* вивід TXD (TXDn) може використовуватися, як лінія введення/виведення загального призначення.

1.6. Прийом даних

Для дозволу робота приймача треба встановити розряд RXEN (RXENn) регістра UCSRB (UCSRnB) [2]. 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 XCK (XCKn).

Прийом даних починається після виявлення приймачем коректного старт-біта (рис. 4*б*). Кожен розряд кадру зчитується з частотою, яка визначається контролером швидкості передачі або тактовим сигналом ХСК (ХСКn). Отримані *розряди даних послідовно розміщуються* в 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. *У разі використання* 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). В *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 під час читання 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 в 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. У разі *вимкненого контролі* парності прапорець UPE (UPEn) завжди читається *як нуль*.

В 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) [2]. 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 у разі *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. В модулях УАПП прапорець помилки кадрування FE (FEn) має бути прочитано 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 під час наступного звернення 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. В модулях УАПП цей *прапорець* скидається після читання регістра даних, а в модулях УСАПП – під час звільнення буфера (після зчитування всіх даних, які знаходяться у ньому).

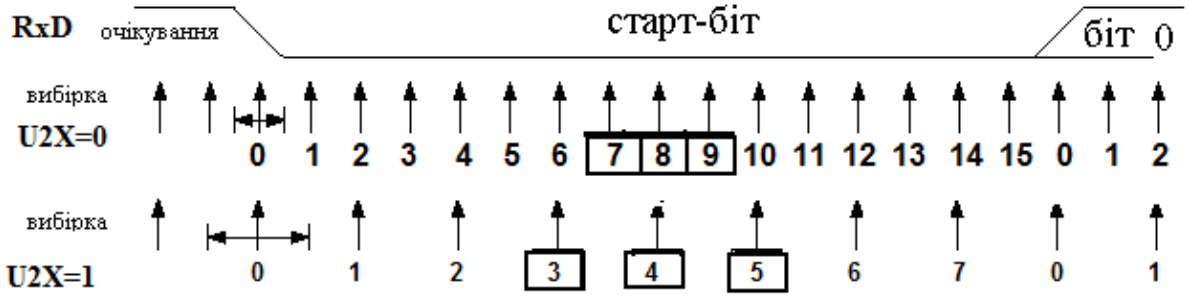
Вимкнення приймача здійснюється скиданням розряду RXEN (RXENn) регістра UCSRB (UCSRnB). На відміну від передавача, приймач *вимикається відразу* ж після скидання розряду, тобто *кадр, який приймається* у цей момент, *втрачається*. *У разі вимкнення* приймача очищується його буфер, тобто втрачаються також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

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

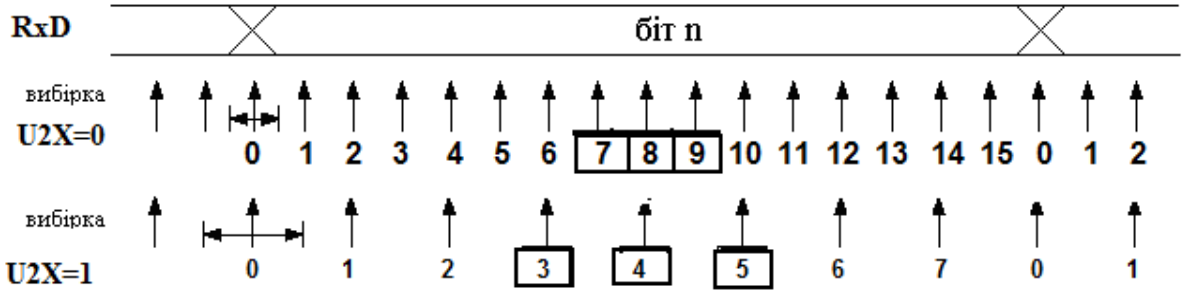
У разі роботи модуля УСАПП у *синхронному* режимі стан виводу RXD (RXDn) читається за одним з фронтів сигналу ХСК (ХСКn). Якщо розряд UCPOL (UCPOLn) регістра UCSRC (UCSRnC) скинуто *в нуль*, зчитування стану виводу RXD відбувається *за спадаючим* фронтом сигналу ХСК (ХСКn), якщо ж встановлено в *одиницю – за наростаючим* фронтом сигналу. Іншими словами, зчитування даних з виводу RXD (RXDn) і їх видача на вивід TXD (TXDn) відбуваються за протилежними фронтами (рис. 5). У разі прийому 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. Вона *опитує* вхід приймача *з метою* визначення старт-біта кадру (рис. 4*б*).

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 (за U2X = 1) *– у 8 разів*. Горизонтальні стрілки на рис. 6 ілюструють *можливий відхід синхронізації* в 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» (рис. 4*б*; рис. 6*а*).



а



б

Рис. 6. Розпізнавання розрядів кадру: а – старт-біт; б – інші розряди

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-ту для кожного біта.

В *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*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. Коли це відбувається, вважається, що виявлено старт-біт нової послідовності даних, з яким синхронізується внутрішній тактовий сигнал приймача. Після цього починає знову 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*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.

1.7. Обмін даними через інтерфейс УСАПП у МК-мережі

У МК-мережах *відбувається обмін* інформацією між одним ведучим та одним з декількох ведених МК. Кожен ведений МК мережі має свою унікальну адресу. Якщо *ведучий МК хоче що-небудь передати*, то він посилає адресний байт, який визначає, до якого з ведених МК він збирається звернутися. Коли *один з ведених* МК розпізнав свою адресу, він переходить в *режим прийому* даних і відповідно приймає подальші байти даних. *Решта ведених* МК *ігнорують* байти даних, що приймаються, до посилки ведучим нового адресного байта.

Програмування режиму прийому кадрів даних/адреси, що приймаються, здійснюється встановленням в одиницю розряду MPCM (MPCMn) регістра UCSRA (UCSRnA) [2].

В мікроконтролері, який виконує *роль ведучого*, треба встановити режим передачі *9-розрядних* даних. У разі *передачі адресного* байта старший – 8-й розряд, за умови, що під час двійкового кодування перший (молодший) розряд, має номер – нуль, *повинен встановлюватися в одиницю*, а під час *передачі б*айтів даних він повинен скидатися в *нуль*. У ведених МК механізм прийому залежить від режиму роботи приймача. У разі прийому 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 МК-мережі виконуються наступні дії:

1. *У всіх ведених* МК встановлюється в одиницю розряд MPCM (MPCMn) регістра UCSRA (UCSRnA).
2. Всі МК програмуються *в режим* передачі дев'ятирозрядних кадрів.
3. Ведучий МК посилає *адресний кадр з TB8 = 1* (регістр UCSRB (UCSRnB)), а всі ведені МК його приймають. У кожному з ведених МК *встановлюється прапорець* RXC (RXCn) регістра UCSRA (UCSRnA).
4. Кожен *ведений* МК *читає вміст* регістра даних (адресний кадр) та *порівнює його із своєю мережевою адресою.* МК, адреса якого *співпала* з адресою, посланою ведучим, *скидає в нуль розряд* MPCM (MPCMn).
5. *Адресований* МК починає приймати кадри, що містять *дані*, оскільки їх *дев'ятий розряд* дорівнює *нулю*. *У решти не адресованих* ведених МК 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 МК *встановлює* в *одиницю* розряд MPCM (MPCMn) і знову чекає прихід кадру з адресою. Процес повторюється з пункту 3.

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

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

$$V_{ПД} = \frac{f_{CLK}}{16 \cdot (UBRR + 1)}, \qquad (5)$$

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)

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

У лінію зв'язку передаються наступні біти: старт-біт, 8 інформаційних біт, 9-й службовий біт – ТВ8, додатковий перевірочний біт на парність і один стоп-біт. Тривалість цих 12 посилок однакова і за $V_{ПД}$ = 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 $12 \cdot t_{посилок}$, що за тривалістю $t_{ПОС}$ = 25 мксек становить $t_Б = 25 \cdot 12 = 300$ мксек.

1.9. Моделювання модуля УСАПП у пакеті PROTEUS

У [2] *описано моделювання* в пакеті PROTEUS модуля УСАПП у складі МК *ATmega8515*.

Наведено схему алгоритму роботи моделі, робочу програму та «скріншоти», які підтвердили їх працездатність.

Контрольні запитання та завдання

1. Дайте визначення поняттю «інтерфейс».
2. Що являє собою послідовний обмін даними між зовнішніми пристроями і мікроконтролером?
3. Чим відрізняються паралельний обмін від послідовного?
4. Які перетворення даних відбуваються у послідовному інтерфейсі?
5. Поясніть особливості обміну під керуванням програми та за перериванням.
6. Як адресуються ЗВПР у МПС?
7. Поясніть особливості програмування обміну даними у МПС.
8. Які способи адресації ПВВ є у МПС на основі МП і8086?
9. Що таке УСАПП (USART) та УАПП (UART)?
10. З яких трьох частин складається структура модуля УСАПП/УАПП? Які елементи містить кожна з них?
11. Які регістри використовуються для керування модулями УАПП та УСАПП?
12. Поясніть структурну схему блоку синхронізації модуля УСАПП мікроконтролера Mega 128.
13. Як визначається швидкість обміну в модулі УСАПП?
14. Як визначається розмір слова даних в модулях УСАПП?
15. Як працює схема контролю парності в модулях УСАПП?
16. Як проходить передача даних в модулях УСАПП?
17. Як проходить прийом даних в модулях УСАПП?
18. Опишіть послідовність дій для здійснення обміну даними у МК-мережі.
19. Опишіть моделювання модуля УАПП в пакеті PROTEUS.

ЛЕКЦІЯ 12. ПРОЕКТУВАННЯ ПРИСТРОЮ ОБМІНУ ІНФОРМАЦІЄЮ МІЖ МОДУЛЕМ УСАПП ТА МІКРОКОНТРОЛЕРОМ

1. ПРОЕКТУВАННЯ ПРИСТРОЮ ОБМІНУ ІНФОРМАЦІЄЮ МІЖ МОДУЛЕМ УСАПП ТА МІКРОКОНТРОЛЕРОМ

1.1. Завдання

Виконати проектування пристрою обміну інформацією між модулем УСАПП мікроконтролера AT mega 8515 та віртуальним терміналом:

1) Розробити, навести та описати робочу модель в пакеті PROTEUS.
2) Зробити розрахунок модуля УСАПП.
3) Розробити схему алгоритму роботи пристрою.
4) Розробити керувальну програму мовою Асемблер.
5) Виконати моделювання пристрою в пакеті PROTEUS. Навести «скріншоти», які пояснюють роботу моделі та підтверджують правильність виконаних розрахунків, схеми алгоритму та робочої керувальної програми. *Обгрунтувати*, що отримана швидкість обміну відповідає завданню. *Навести* ASCII-код символа, який отримано від віртуального терміналу – Virtual Terminal.
6) Розробити та описати структурну схему пристрою.
7) Вихідні дані:

– швидкість передачі: $V_{\text{пд}}$= 9600 бод;
– тактова частота кварцового резонатора: f_{CLK} = 8МГц;
– для обміну використовувати модуль USART;
– режим передачі – асинхронний;
– розмір слова даних – 8 розрядів;
– використовувати перевірку на непарність;
– кількість стоп-бітів – 1;
– байт для передачі знаходиться в регістрі R22 та дорівнює: 10110101b;
– символ для прийому від віртуального терміналу – 3.

1.2. Розробка та опис робочої моделі в пакеті PROTEUS

1.2.1. Робоча модель пристрою

Робочу модель дослідження універсального асинхронного приймача-передавача (УАПП) в пакеті Proteus наведено на рис. 1.

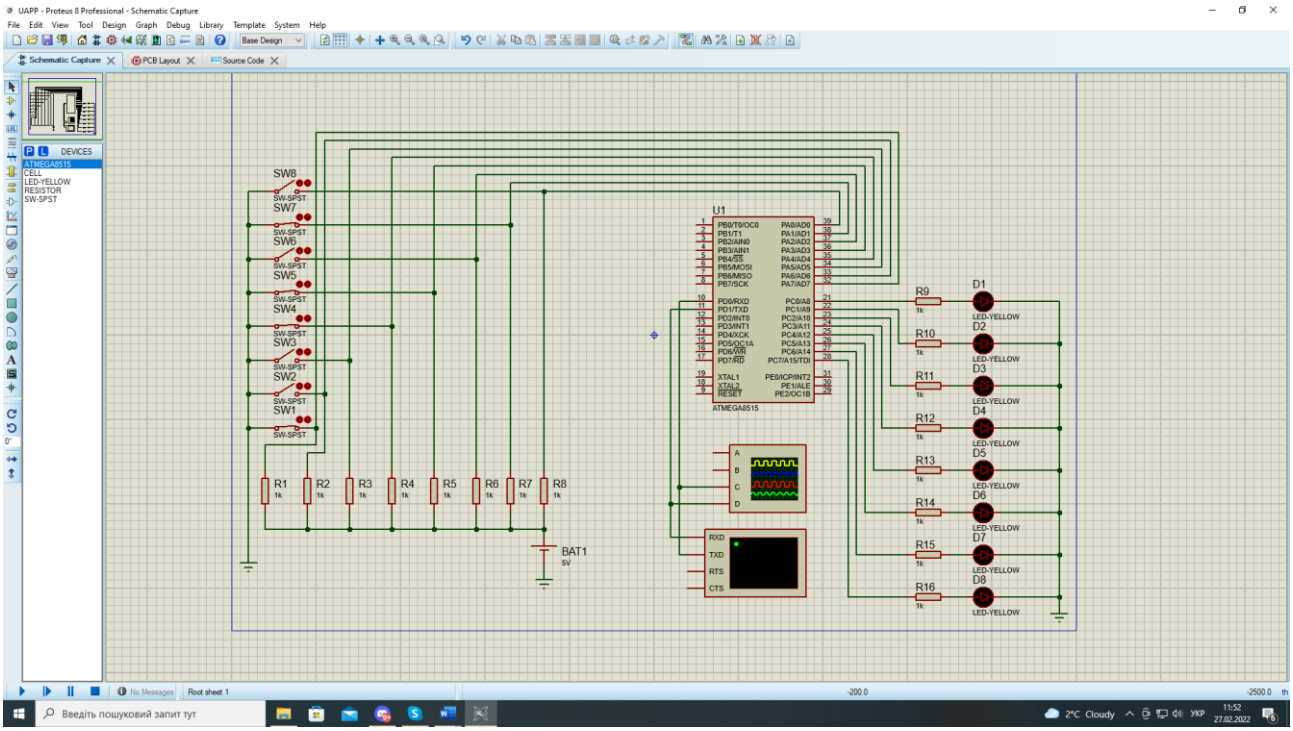


Рис. 1. Схема робочої моделі

1.2.2. Опис робочої моделі

Зліва знаходяться *вісім кнопок*, за допомогою яких задається байт (8 біт) даних, який передається через модуль УАПП.

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

До ліній TxD – вихід передавача УСАПП та RxD – вхід приймача підключено *осцилограф* та віртуальний термінал «*Virtual Terminal*», який вибрано з віртуальних інструментів програми Proteus. На них відображаються *повідомлення*, які вводяться та виводяться через модуль УАПП в/з мікроконтролера.

До виводів XTAL1 та XTAL2 в реальних схемах підключають кварцовий резонатор частотою, яка визначає тактову частоту генератора мікроконтролера. В нашій роботі вона дорівнює f_{BQ} = 8МГц. Також підключають конденсатори С1 та С2, ємністю 30пФ, які призначені для підвищення стабільності роботи системного генератора. *Резонатор та конденсатори* потрібні у *практичній* схемі. В модель Proteus їх можна не вводити.

В якості мікроконтролера в моделі використано AVR-мікроконтролер *ATmega 8515*. *Для задання тактової частоти* треба *двічі лівою кнопкою* миші клацнути на мікроконтролері та в опції Clock Frequency вказати відповідне значення.

1.3. Розрахунок модуля УСАПП

1.3.1. Розрахунок швидкості обміну

У разі роботи *в асинхронному* режимі швидкість обміну визначається вмістом *регістра UBRR* та станом *розряду U2X (U2Xn)* регістра UCSRA. Якщо цей *розряд встановлено в «1»*, коефіцієнт ділення попереднього дільника зменшується у два рази, а *швидкість* обміну відповідно *подвоюється*. Під час роботи у синхронному режимі цей розряд має бути скинуто.

Швидкість обміну *для асинхронного* режиму визначається наступними формулами (лекція 11):

– асинхронний режим (звичайний, U2Xn = 0):

$$BAUD = f_{CLK}/(16(UBRR+1);$$

– асинхронний режим (пришвидшений, U2Xn = 1):

$$BAUD = f_{CLK}/(8(UBRR+1)),$$

де BAUD – швидкість передачі в бодах, f_{CLK} – тактова частота мікроконтролера, UBRR – вміст регістра контролера швидкості передачі (0…4095).

У [2] наведено *значення* регістра *UBRR*, що дозволяє отримати стандартні для асинхронного режиму швидкості передачі у разі використання різних резонаторів, а також величини значень похибок відносно стандартних швидкостей, що отримуються у цьому випадку. *Умовам завдання задовольняють* значення UBRR = 51 за U2X = 0, або UBRR = 103 за U2X = 1. В обох випадках похибка дорівнює 0,2%. Обираємо UBRR = 51 = $0033 та U2X = 0.

1.3.2. Програмування регістра UBRR контролера швидкості передачі

Для нашого МК-ра *регістр UBRR є 12-розрядним* та фізично розміщений у двох регістрах введення/виведення [2].

Ці регістри мають *адреси*: UBRRH = $0040; UBRRL = $0029. Під час завантаження *регістра UBRR біт URSEL* повинен дорівнювати нулю, що відбувається після скидання мікроконтролера [2].

Згідно з розрахунком, наведеним вище, UBRR = 51 = $0033.

Тоді робоча програма має вигляд:

```
LDI R18, $00; R18←$00;
LDI R31, $00; R31←$00;
LDI R30, $40; R30←$40; R31, R30 (Z)←$0040 (адреса UBRRH);
ST Z, R18;  UBRRH ←00 (старший байт UBRR).
```

LDI R19, $33; R19←$33;
LDI R27, $00; R27←$00;
LDI R26, $29; R26←$29; R27, R26 (X)←$0029 (адреса UBRR0L);
ST X, R19; UBRR0L = $33 (молодший байт UBRR).

1.3.3. Програмування регістра UCSRA

Формат регістра UCSRA наведено на рис. 2, а опис його окремих розрядів – у [2].

	7	6	5	4	3	2	1	0
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn
Зчитування(R)/Запис(W)	R	R/W	R	R	R	R	R/W	R/W
Початкове значення	0	0	1	0	0	0	0	0

Рис. 2. Формат регістрів UCSRA (UCSRnA)

Для прикладу, який розглядається, у разі програмування регістра потрібно скинути біт U2X та встановити біт UDRE в одиницю. Після скидання мікроконтролера біт UDRE автоматично встановлюється у одиницю, а біт U2X скидається в нуль. Інші біти регістра мають початкові нульові значення.

Керувальне слово КС1 для програмування регістра UCSRA має вигляд:

	7р	6р	5р	4р	3р	2р	1р	0р	
UCSRA	0	0	1	0	0	0	0	0	КС1=$20

Регістр UCSRA в просторі резидентної пам’яті даних МК-ра AT mega8515 має *адресу*: $002B [2].

Тоді програма має вигляд:

LDI R16, $20; R16←КС1 = $20,
LDI R27, $00; R27←$00,
LDI R26, $2B; R26←$2B; R27, R26 (X) ←$002B,
ST X, R16; UCSR0A ←R16 = КС1 = $20.

1.3.4. Програмування регістра UCSRC

Формат регістра UCSRC наведено на рис. 3, а опис його окремих розрядів – у [2].

Для МК-ра AT mega 8515 розряд *UMSEL* програмує режим роботи модуля USART. Оскільки у завданні задано *асинхронний* режим, то цей біт повинен мати *нульове* значення.

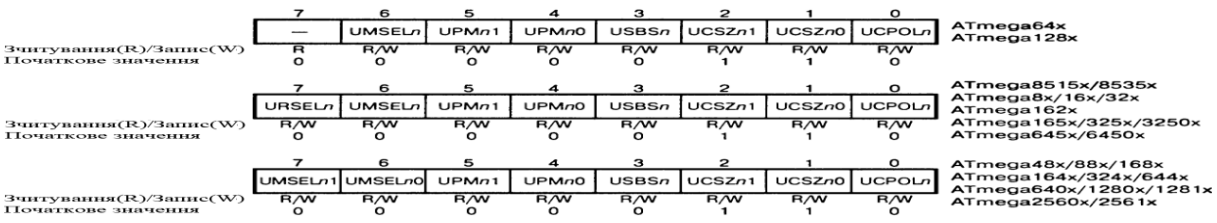


Рис. 3. Формат регістрів UCSRC (UCSRnC)

Біти UPM1 та UPM0 згідно з табл. 1 керують *контролем парності*. В нашому випадку необхідно запрограмувати: UPM1 = 1; UPM0 = 1 (перевірка на непарність).

Таблиця 1. Керування контролем парності

UPM1 (UPMnl)	UPM0 (UPMn0)	Режим роботи
0	0	Виключений
0	1	Зарезервовано
1	0	Включений, перевірка на парність (even parity)
1	1	Включений, перевірка на непарність (odd parity)

Примітка: n = 0, 1, 2 чи 3.

Значення біта парності отримується шляхом виконання операції «виключне АБО» над усіма розрядами слова даних, яке передається. Якщо використовується перевірка *на парність* (even parity), то отриманий результат і*нвертується*:

$$P_{EVEN} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1,$$
$$P_{ODD} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0. \quad (2)$$

Біт *USBS* скидається в *нуль*, тому що ми повинні передавати *1 стоп-біт*.

Біти UCSZ1, UCSZ0 разом з бітом *UCSZ2 регістра UCSRB* програмують розмір слова даних (табл. 2).

Для нашого прикладу треба запрограмувати: UCSZ1 = 1; UCSZ0 = 1, а UCSZ2 = 0 (передаються 8 розрядів). Розряд UCPOL в асинхронному режимі не використовуються, тому записуємо в нього нуль.

Біт URSEL= 1, тому, що запис виконується в регістр UCSRC.

Тоді керувальне слово КС2 для програмування регістра UCSRC має вигляд:

	7р	6р	5р	4р	3р	2р	1р	0р	
UCSRC	1	0	1	1	0	1	1	0	КС2=$B6

Таблиця 2. Визначення розміру слова даних у модулях USART

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.

Регістр UCSRC в просторі резидентної пам'яті даних МК-ра AT mega8515 має адресу: $0040 [2].

Тоді програма має вигляд:

LDI R17, $B6; R17← $B6,
LDI R29, $00; R29←$00,
LDI R28, $40; R28←$40, R29, R28 (Y) ←$0040,
ST Y , R17; UCSR0C ←R17 = КС2 = $B6.

1.3.5. Програмування регістра UCSRB

Формат регістра UCSRB наведено на рис. 4, а опис його розрядів – у [2].

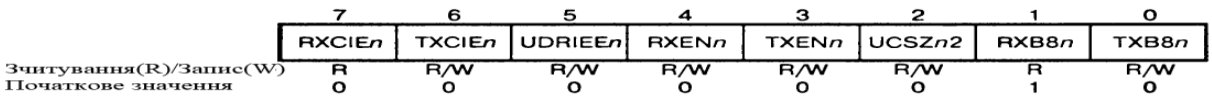


Рис. 4. Формат регістрів UCSRB (UCSRnB)

Для нашого прикладу необхідно запрограмувати: RXCIE = TXCIE = = UDRIE = 0; UCSZ02 = 0 (п. 1.3.4) та TXEN = RXEN = 1. Інші біти не використовуються, тому пишемо в них нулі.

Тоді керувальне слово КС3 для програмування регістра UCSRB має вигляд:

	7р	6р	5р	4р	3р	2р	1р	0р	
UCSRB	0	0	0	1	1	0	0	0	КС3=$18

Регістр UCSRB в просторі резидентної пам’яті даних МК-ра AT mega8515 має адресу: $002A [2].

Тоді програма має вигляд:

LDI R19, $18; R19← КС3=$18,
LDI R29, $00; R29←$00,
LDI R28, $2A; R28←$2A; R29, R28 (Y)←$002A,
ST Y, R19; UCSRB ←R19 = КС3= $18.

1.3.6. Програмування регістра даних UDR

Регістр UDR в просторі резидентної пам’яті даних МК-ра AT mega8515 має адресу: $000C [2]. Тоді програма має вигляд:

LDI R31, $00; R31← $00;
LDI R30, $0C; R30←$0C; R31, R30 (Z) ← $000C;
ST Z, R22; UDR ← R22 (завантаження байта для передачі).

1.4. Схема алгоритму роботи пристрою

Схему алгоритму роботи пристрою наведено на рис. 5.

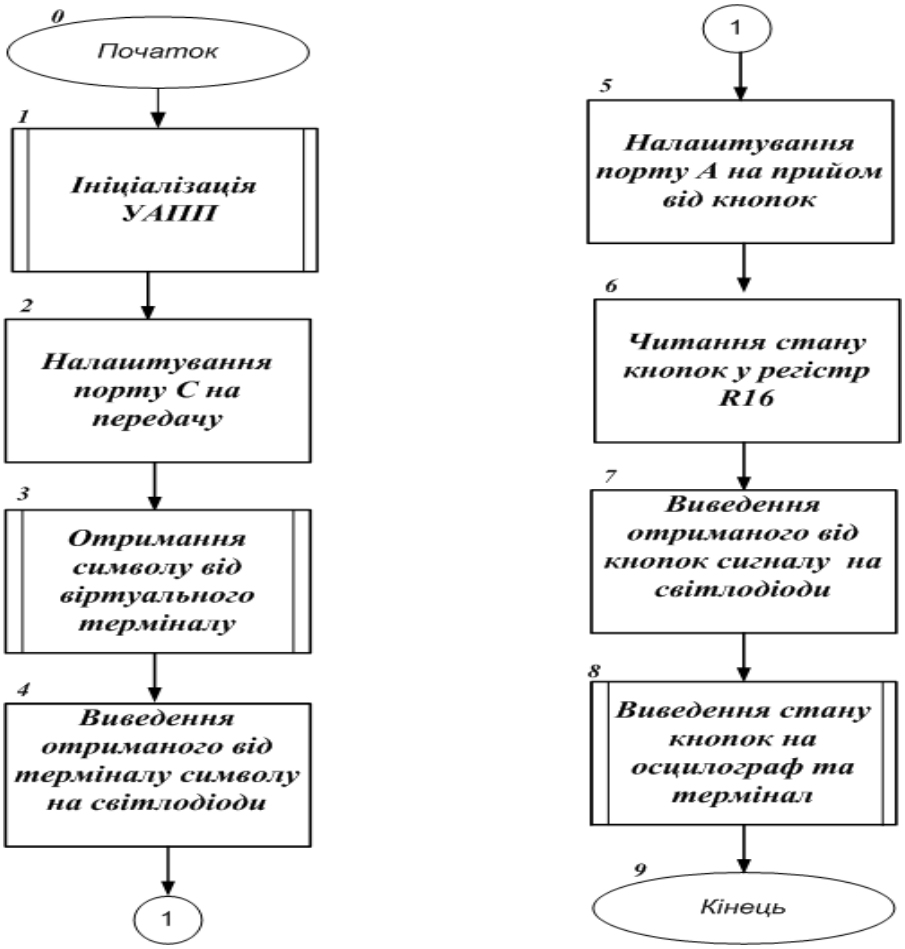


Рис. 5. Схема алгоритму роботи пристрою

1.5. Керувальна програма мовою Асемблер

1.5.1. Лістинг керувальної програми мовою Асемблер

```
.nolist ; директива відключає генерацію коду в лістинг,
        ; тобто далі у файлі *.lss не буде фіксуватися асемблерний код
.include "m8515def.inc" ; підключення стандартного заголовочного
                        ; файлу для ATmega8515
.list     ; директива включає генерацію коду в лістинг,
        ; тобто далі у файлі *.lss буде фіксуватися асемблерний код
.equ f_CK = 8000000 ; частота в герцах
.equ BAUD = 9600 ; швидкість для УСАПП у бодах
.equ UBRR_value = (f_CK/(BAUD ·16)) –1; розрахунок значення для регістра
                        ;UBRR
.cseg      ; директива визначення, що далі іде код програми
.org 0      ; директива визначення, що  код програми записано
; у FLASH-пам’яті та  буде розміщено з нульової адреси
; КОД ОСНОВНОЇ ПРОГРАМИ
Start: ; (блок 0)
; ініціалізація з мітки init_USART
rcall init_USART ; (блок 1) відносний виклик підпрограми ініціалізації
; налаштування порту С на передачу
LDI R16, 0xFF  ; (блок 2) R16← 0xFF
OUT DDRC, R16   ; (блок 2) DDRC← R16
; отримання того, що відправив віртуальний термінал (блок 3)
rcall USART_recieve; відносний виклик підпрограми з мітки
     ; USART_recieve
; (блок 4) виведення отриманого від терміналу символу на світлодіоди
OUT PORTC, R16 ; (блок 4) PORTC←R16
```

```
; (блок 5) налаштування порту А на прийом від кнопок
LDI R16, 0x00   ; (блок 5) R16←0x00
OUT DDRA, R16   ; (блок 5) DDRA← R16
; (блок 6) читання стану кнопок в регістр R16
in R16, PINA  ; (блок 6) R16← PINA
; (блок 7) виведення отриманого від кнопок на світлодіоди
OUT PORTC, R16 ; (блок 7) PORTC←R16
; (блок 8)  виведення стану кнопок на осцилограф та термінал
; (підпрограма з міткою USART_send)
rcall USART_send ; відносний виклик підпрограми з мітки
; USART_ send
; (блок 1) ПІДПРОГРАМА ІНІЦІАЛІЗАЦІЇ МОДУЛЯ USART
init_USART:
; програмування швидкості обміну 9600бод
ldi   R16, high(UBRR_value) ; (блок 1) R16←старший байт UBRR_value
out   UBRRH, R16 ; (блок 1) UBRRH←R16
ldi   R16, low(UBRR_value)   ; (блок 1) R16←молодший байт UBRR_value
out   UBRRL, R16 ; (блок 1) UBRRL←R16
; дозвіл роботи приймача-передавача модуля USART
ldi   R16, (1<<RXEN)|(1<<TXEN)|(0<<RXCIE)|(0<<TXCIE)|(0<<UDRIE)
out   UCSRB, R16 ; (блок 1) UCSRB←R16
; програмування передачі восьми біт, з перевіркою на непарність,
; в асинхроному режимі, з одним стоп-бітом
ldi   R16, (1<< URSEL)|(1<<UPM1)|(1<<UPM0)|(1<< UCSZ1)|(1<< UCSZ0)
out   UCSRC, R16 ; (блок 1) UCSRC← R16
ret  ; (блок 1) повернення з підпрограми ініціалізації
; передача через модуль USART
```

```
USART_send:
out   UDR, R16  ; (блок 8) регістр даних UDR← R16
sending:
; чекання 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 пропускається наступна команда,
; інакше наступна команда
rjmp  USART_recieve; (блок 3) безумовний перехід на мітку  USART_recieve
in  R16, UDR   ; (блок 3) R16←UDR (завантаження в
                    ; регістр  R16 отриманого байта від термінала
ret ; (блок 3) повернення з підпрограми
END Start  ; (блок 9)
```

1.5.2. Компіляція робочої програми та отримання hex-файлу

Для програмування мікроконтролерів AVR може використовуватися, наприклад, Microchip Studio 7; Atmel Studio 7 – інтегроване середовище розробки для програмування та відлагодження програм для мікроконтролерів AVR та AVR32 в операційних системах Windows. Після шостої версії середовище може працювати як і з AVR-мікроконтролерами, так і з системами з ARM-архітектурою. У [3; 4] описано послідовність створення робочого проекту в цьому середовищі, компіляцію програми створення hex-файлу та його *завантаження в пам'ять мікроконтролера.*

1.6. Моделювання пристрою в пакеті PROTEUS

Налаштування віртуального терміналу

Для налаштування віртуального терміналу (Terminal) треба *двічі натиснути на нього лівою кнопкою миші*, та зробити налаштування згідно завдання. Нижче на рис. 6 наведено приклад для випадку, коли зі швидкістю 9600 біт/сек потрібно

передавати наступну інформацію: 8 біт даних та один стоп-біт з перевіркою на непарність.

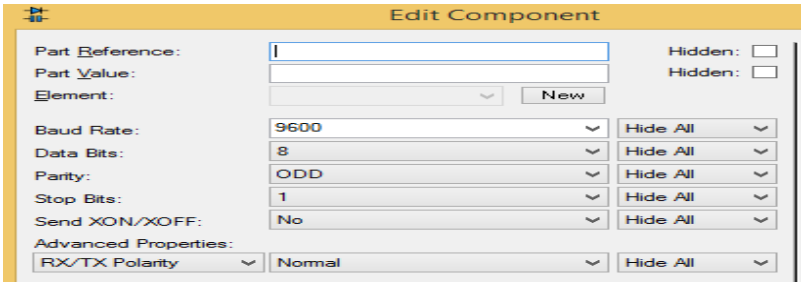


Рис. 6. Налаштування віртуального термінала

Для налаштовування *перевірки на парність або непарність* треба використати *опцію Parity*.

Налаштування свілодіодів на цифровий режим

Для налаштувати свілодіодів на цифровий режим треба *двічі клікнути на них лівою кнопкою миші* та обрати необхідний цифровий режим (рис. 7).

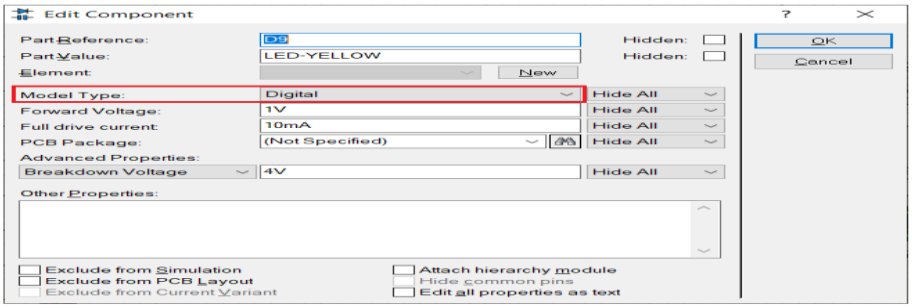


Рис. 7. Налаштування світлодіодів на цифровий режим

Запуск процесу моделювання

Для запуску процесу моделювання треба натиснути «Start VSM Debugging» (рис. 8).

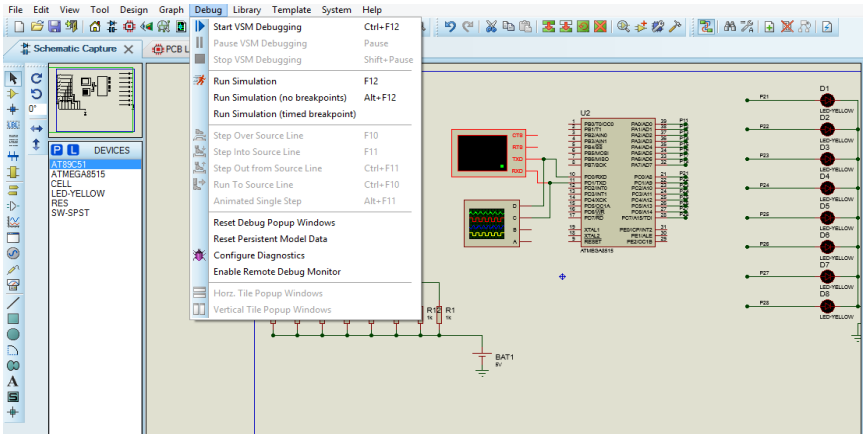


Рис. 8. Запуск процесу моделювання

Встановлення двох точок зупинки під час виконання програми

Для встановлення двох точок зупинки під час виконання програми *натискається кнопка* «Pause VSM Debugging». У вкладці «Source Code» встановлюються дві точки зупинки. Для цього потрібно *лівою кнопкою миші двічі клацнути з лівої сторони двох команд*, як показано на рис. 9.

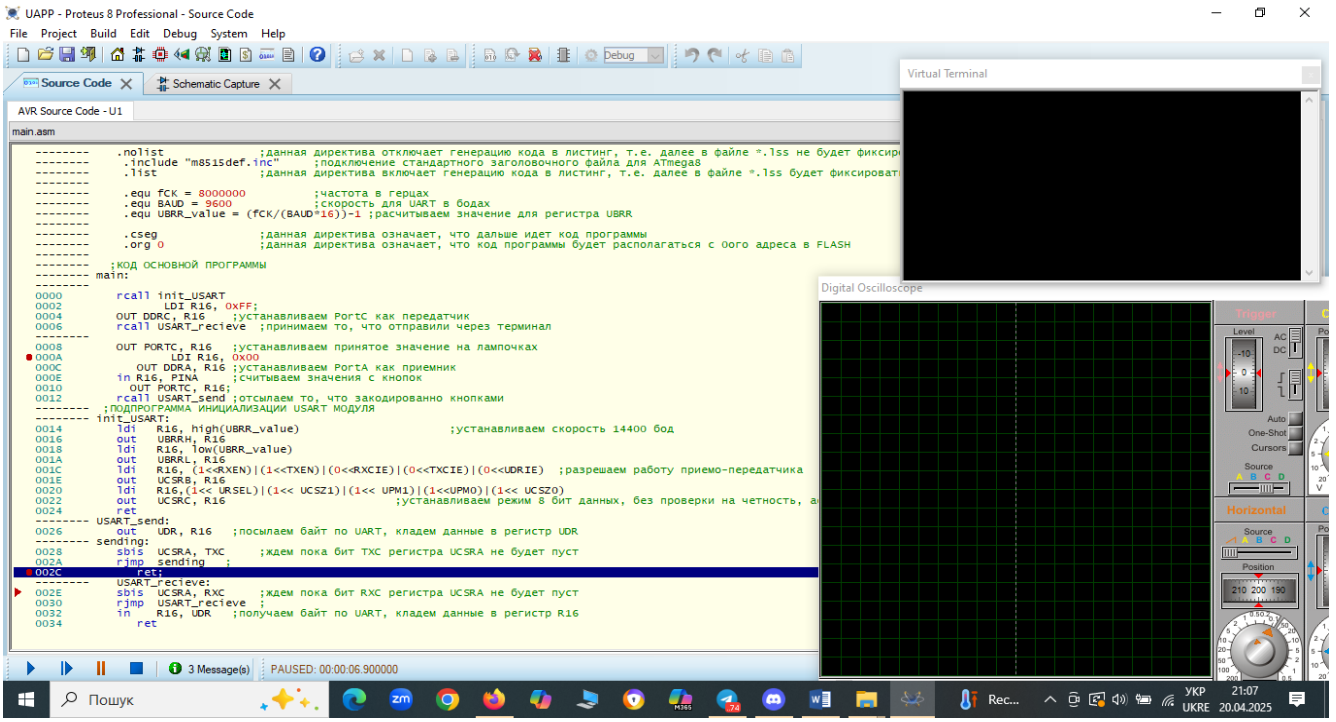


Рис. 9. Встановлення двох точок зупинки

Введення символу, який надсилається від віртуального терміналу

У разі введення символу від віртуального терміналу потрібно виконати наступне:

1) У вікні «Virtual Terminal» натисканням правої кнопки миші обираються: «Hex Display Mod» та «Echo Typed Characters» (рис. 10). Якщо вікно Virtual Terminal закрито, то його можна *відкрити із вкладки Debug*.

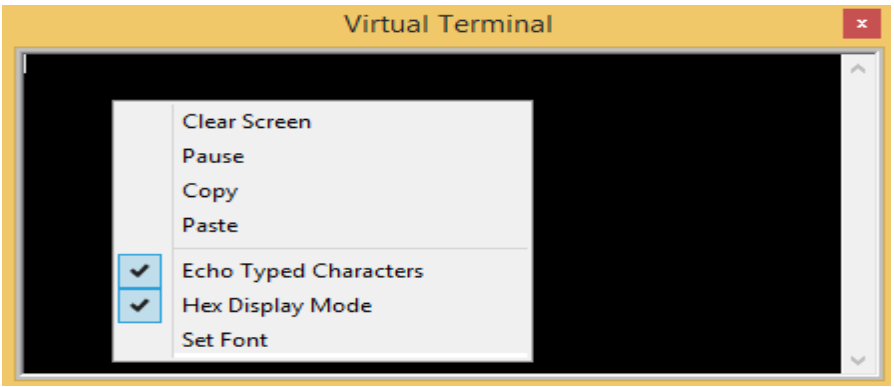


Рис. 10. Встановлення опцій віртуального термінала

2) Виконати налаштування осцилографа згідно рис. 11 (натиснути *кнопку* «One-Shot» (*один кадр*)).

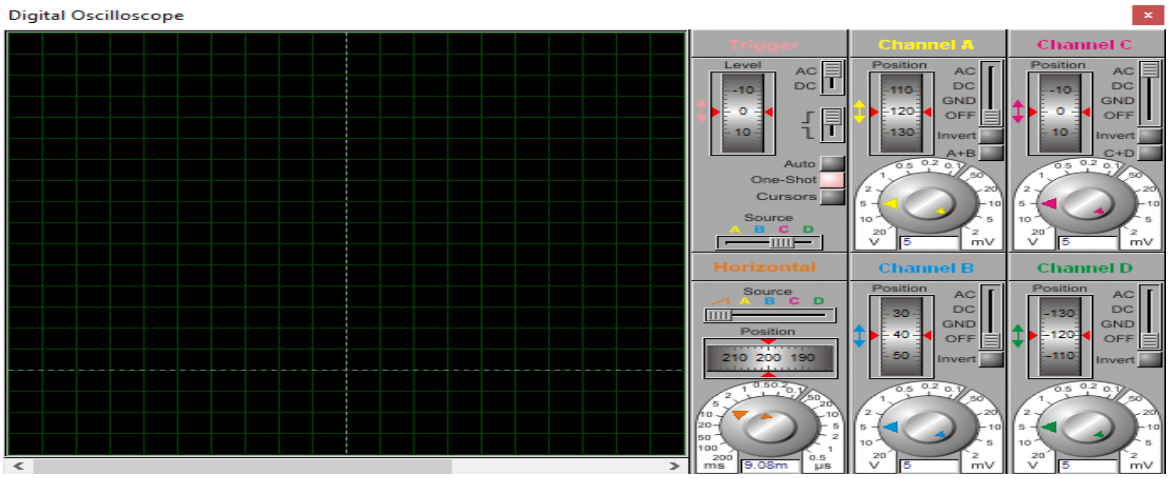


Рис. 11. Налаштування осцилографа

3) Натиснути «Run Simulation» (F12**)**.

4) Перед введенням символу у вікні терміналу *повинен стояти курсор.*

5) Ввести символ, який надсилається від віртуального терміналу до мікроконтролера через УАПП. Символ, який вводиться з клавіатури, віртуальний термінал перетворює в ASCII-код згідно таблиці, яку наведено у [1].

Наприклад, вводиться цифра 7, ASCII-код якої 37 (hex) = 0011 0111 (bin), *в таблиці, яку наведено у [1], виділено жовтим кольором*.

Програма перейде на рядок, де встановлено першу точку зупину.

У вкладці зі схемою *Schematic Capture* на світлодіодах та осцилографі відображається символ, який прийнято від термінала через виходи *порту С* (рис. 12).

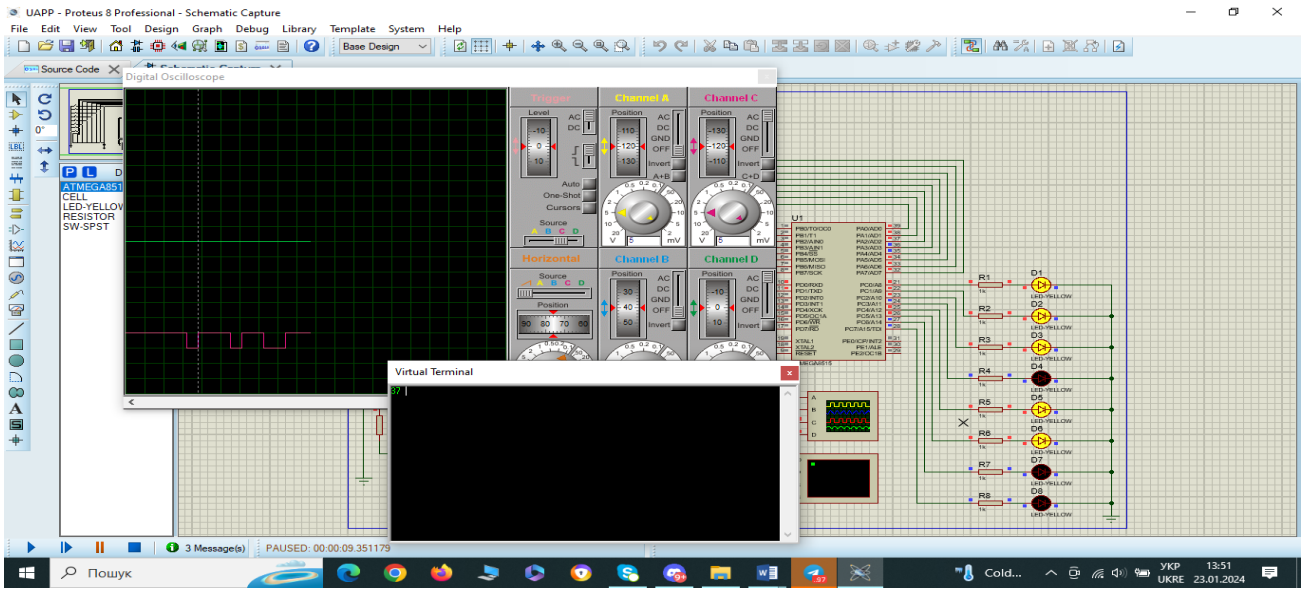


Рис. 12. Відображення на світлодіодах та осцилографі символу 7, який прийнято від терміналу

Світлодіоди відображають бінарний код символу, який ми прийняли від терміналу (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. Після останнього другого нульового біта перед одиничним стоп-бітом іде також одиничний біт, тому що сума одиниць у символі непарна та згідно (2) – $P_{0DD} = 1$. Якщо ми, наприклад, від термінала передаємо цифру 9, ASCII-код якої 39 (hex) = 00111001 (bin), то після другого нульового біта перед одиничним стоп-бітом іде третій нульовий біт, тому що сума одиниць у символі парна та згідно (2) – $P_{0DD} = 0$ (рис. 13).

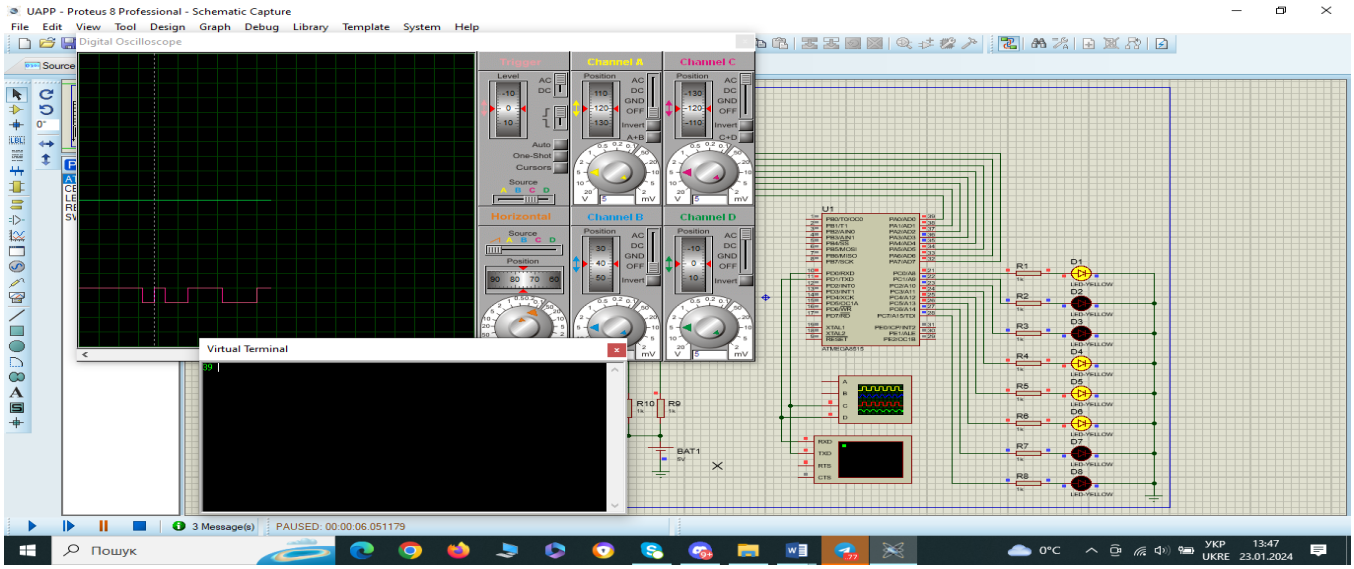


Рис. 13. Відображення на світлодіодах та осцилографі символу 9, який прийнято від терміналу

Введення символу, який надсилається через УАПП від мікроконтролера

1) Число для передачі мікроконтролером через УАПП кодується кнопками: замкнена кнопка – нуль, а розімкнена – одиниця (рис. 14).

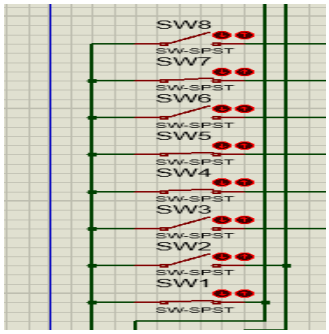


Рис. 14. Кодування кнопками символу для передачі

2) Щоб побачити переданий сигнал на осцилографі, *потрібно натиснути* кнопку на опції One-Shot (один кадр) (рис. 11).

3) Для продовження виконання програми знову натискається «Run Simulation». Програма переходить до другої точки зупину (рис. 15).

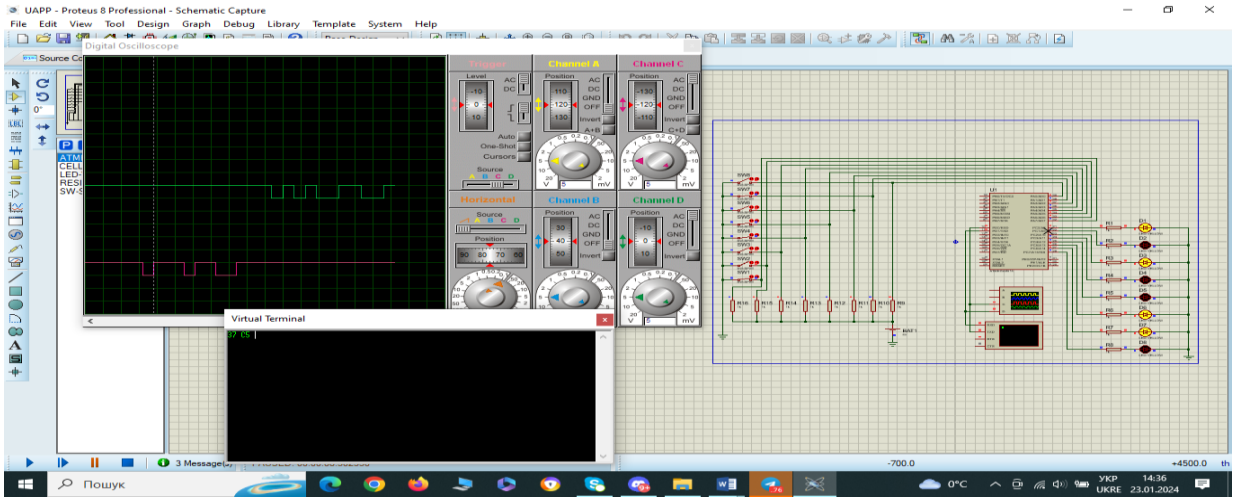


Рис. 15. Відображення сигналів на осцилографі: від терміналу – червоний; від МК (кнопок) – зелений

4) Нижче пояснюються сигнали, що відображаються на осцилографі: осцилограф відображає сигнали за двома променями: верхній (зелений) – сигнал, який передається від кнопок, нижній (червоний) – сигнал, який приймається *від терміналу*: 37(hex).

Перший перепад сигналу, який передається *від кнопок*, із високого рівня у низький – старт-біт. Потім знаходяться вісім інформаційних бітів, починаючи з молодшого розряду, які відображені на осцилографі, як 01100101 (bin) = 65 (hex). Перед одиничним стоп-бітом іде другий нульовий біт, тому що сума у символі парна та згідно (2) – $P_{0DD} = 0$ (рис. 15). Далі розглянуто ще один приклад, коли на кнопках було закодовано число 67(hex) = 01100111(bin) (рис. 16, рис. 17).

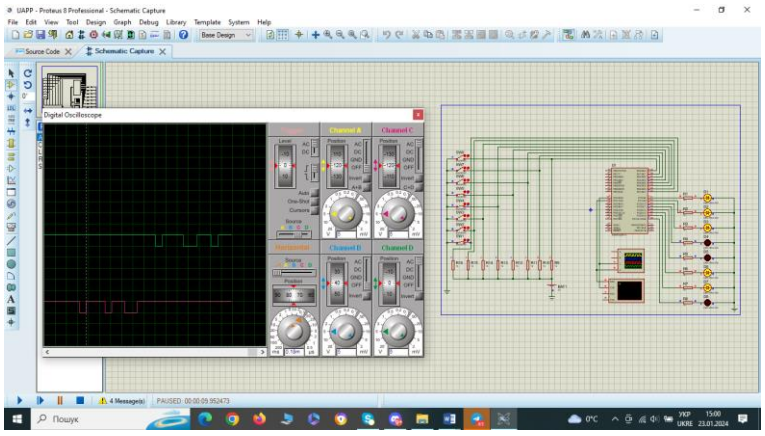


Рис. 16. Відображення сигналів на осцилографі: від терміналу – червоний, від МК (кнопок) – зелений

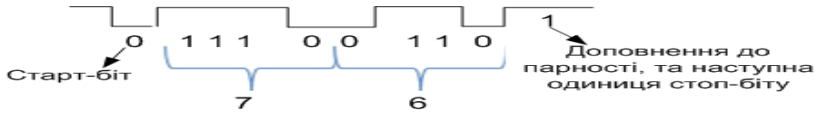


Рис. 17. Передача числа: 67 (hex) = 01100111 (bin)

На верхньому промені осцилографа відображено: перший нуль – старт-біт, потім три одиниці та один нуль – це число 7 на кнопках, далі йде нуль, дві одиниці та нуль – це число 6 на кнопках, потім після нульового біта перед одиничним стоп-бітом іде також одиничний біт, тому що сума одиниць у символі непарна та згідно (2) – $P_{0DD} = 1$ (рис. 17).

Визначення отриманої швидкості передачі

Для визначення отриманої швидкості передачі, яку в даному прикладі було задано – 9600 біт/сек = 9600 пос/сек = 9600 бод, налаштовується така розгортка сигналу на осцилографі, щоб *сумістити бічні сторони імпульсу із вертикальними лініями сітки осцилографа* (рис. 18).

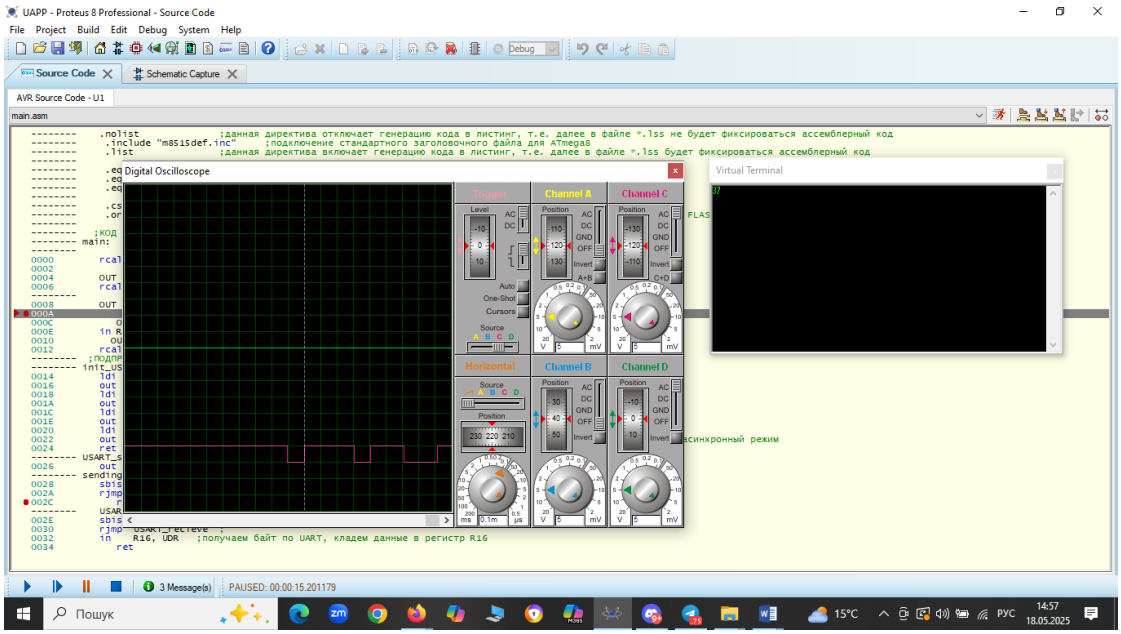


Рис. 18. Визначення отриманої швидкості передачі на осцилографі

Як видно з рис. 18, тривалість одного імпульсу дорівнює приблизно 0,104 мс. Якщо розділити один біт на цей час, та помножити на 1000, отримується задана швидкість

$$V = \frac{1}{0.1} \cdot 1000 = 9600 \ \frac{\text{біт}}{\text{сек}}.$$

1.7. Розробка та опис структурної схеми пристрою

Нижче на рис. 19 наведено схему електричну структурну пристрою обміну інформацією між модулем УАПП МК ATmega 8515 та віртуальним терміналом.

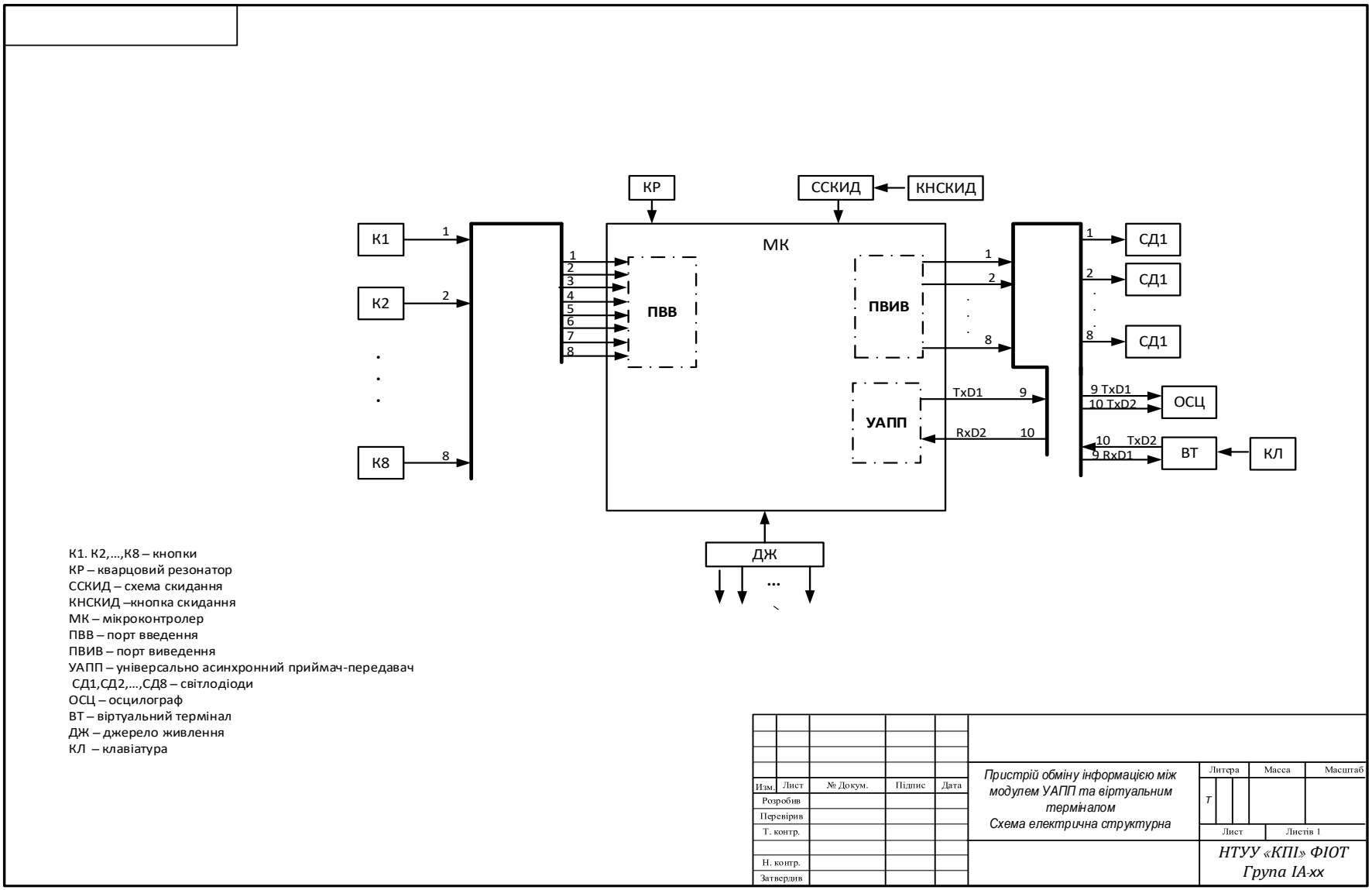


Рис. 19. Схема електрична структурна пристрою обміну інформацією між модулем УАПП мікроконтролера ATmega 8515 та віртуальним терміналом

Основним вузлом структури є мікроконтролер, який керує обміном інформації.

Через паралельний порт введення мікроконтролер отримує сигнали від восьми кнопок, якими кодується байт для передачі. Цей байт через порт виведення передається до світлодіодів. Через модуль УАПП мікроконтролера байт, який закодовано кнопками, *з виходу TxD1 передається на вхід RxD1 віртуального* термінала у послідовному асинхронному старт-стопному режимі. Цей байт також відображається на екрані осцилографа.

З виходу TxD2 віртуального терміналу у послідовному асинхронному старт-стопному режимі *на вхід RxD2 модуля УАПП* подається ASCII-код символу, який вводиться з клавіатури персонального комп`ютера. Цей код також відображається на екрані осцилографа.

Кварцовий резонатор (КР) визначає частоту високочастотного генератора, схему якого вбудовано у мікроконтролер.

Схема скидання (ССКИД) формує сигнал «RESET» (Скидання), який запускає виконання робочої програми спочатку. Цей сигнал формується автоматично під час включення живлення, або від зовнішньої кнопки «КНСКИД».

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. Як визначається швидкість обміну в модулі УСАПП?
6. Як визначається розмір слова даних у модулях УСАПП?
7. Як працює схема контролю парності?
8. Як проходить передача даних в модулях УСАПП?
9. Як проходить прийом даних в модулях УСАПП?
10. Як визначити швидкість передачі даних для асинхронного звичайного режиму?
11. Як визначити тривалість передачі кадру для асинхронного прискоренного режиму?
12. Як в моделі підключено кнопки, які задають байт для передачі?
13. Як в моделі підключено світлодіоди, які відображають передану в моделі інформацію?
14. Як в моделі визначається тривалість переданого біта?
15. Опишіть формат та призначення розрядів регістра UCSRC.
16. Як в програмі відбувається налаштування порту А на передачу?
17. Як в програмі визначається час завершення передачі та прийому чергового байта?
18. Дайти характеристику ASCII-коду.
19. Поясніть як в моделі підключено світлодіоди та в якому випадку вони будуть світитися?
20. Як працюють схеми відновлення тактового сигналу і даних при асинхронному прийомі?

ЛЕКЦІЯ 13. ЗВ'ЯЗОК МП/МК З АНАЛОГОВИМ ОБ'ЄКТОМ КЕРУВАННЯ. ЗВ'ЯЗОК МП/МК З МОДЕМОМ

1. ЗВ'ЯЗОК МІКРОПРОЦЕСОРІВ/МІКРОКОНТРОЛЕРІВ З АНАЛОГОВИМ ОБ'ЄКТОМ КЕРУВАННЯ

1.1. Особливості введення/виведення аналогової інформації у МПС

Для зв'язку МП/МК з аналоговим об'єктом керування використовується модуль АЦП-ЦАП [1…5]. Під час проектування модуля АЦП-ЦАП потрібно вирішувати наведені нижче задачі.

На апаратному рівні:

- вибір розрядності двійкового коду за заданою похибкою дискретизації АЦП;
- визначення необхідного часу перетворення АЦП;
- вибір величини дискретизації за часом за теоремою Котельникова;
- визначення необхідності застосування і, якщо це необхідно, то вибір пристрою вибірки-зберігання;
- якщо МПС проектується *на основі мікропроцесора*, то вибір мікросхем АЦП і ЦАП, що забезпечують потрібну похибку, швидкодію і споживану потужність;
- вибір схем включення, що забезпечують необхідний діапазон зміни вхідних і вихідних напруг;
- якщо МПС проектується *на основі мікроконтролера*, то вибір МК, який має відповідний модуль АЦП та ЦАП;
- розробка принципової схеми.

На програмному рівні. Якщо МПС проектується *на основі МП*, то:

- формування імпульсу вибірки для ПВЗ;
- формування сигналу запуску АЦП («СТАРТ»);
- перевірка готовності даних на виході АЦП (аналіз виходу «READY» або обмін за перериванням);
- після визначення готовності АЦП введення даних у МП;
- формування сигналу «СКИДАННЯ» для АЦП;
- після завершення обробки даних, отриманих від АЦП, виведення керувальних впливів у порт, до якого під'єднано ЦАП. Як правило, порт виведення має буферний регістр, який зберігає байт до наступного виводу. ЦАП перетворює цей байт в аналогову напругу, яка подається до виконавчого пристрою.

Якщо МПС проектується *на основі МК*, то:

- виконання ініціалізації відповідного модуля;

– запуск процесу аналого-цифрового перетворення;
– визначення часу закінчення перетворення в АЦП;
– обробка результату перетворення в МП/МК;
– виведення керувального впливу до ЦАП;
– передача аналогової напруги до виконавчого пристрою.

1.2. Застосування АЦП під час введення аналогової інформації у МПС

1.2.1. Загальні відомості про АЦП

АЦП – це пристрої, що *перетворюють* вхідні аналогові сигнали у відповідні їм цифрові сигнали, які придатні для роботи з МК/МП і іншими цифровими пристроями. АЦП широко *застосовуються*: у пристроях дискретної автоматики; цифрових системах керування для перетворення аналогових сигналів від датчиків у цифрову форму; в системах відображення інформації для цифрової індикації; в системах передачі даних і багатьох інших галузях техніки.

У [5] описано принцип роботи АЦП, наведено його розрахунок, розглянуто необхідність застосування АЦП, виконано аналіз роботи АЦП послідовного наближення.

1.2.2. Модуль АЦП AVR-мікроконтролерів

1.2.2.1. Функціональна схема модуля

До складу більшості моделей AVR-мікроконтролерів входить модуль *10-розрядного АЦП послідовного наближення*. На рис. 1 наведено функціональну схему модуля АЦП одного з AVR-мікроконтролерів, аналоговий мультиплексор якого має *16 аналогових входів*.

В деяких моделях МК елементи і пов’язані з ними сигнали, які виділено на цьому рисунку *сірим кольором, відсутні*, а неінвертуючий вхід компаратора з пристроєм вибірки-зберігання підключено безпосередньо до виходу мультиплексора (показано пунктирною лінією).

У більшості моделей входи АЦП *можуть об’єднуватися попарно* для формування каналів з диференціальним входом. Для деяких каналів є можливість 10- та 200-кратного *попереднього підсилення* вхідного сигналу.

Як *джерело опорної напруги* для АЦП може використовуватись напруга живлення МК 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*.

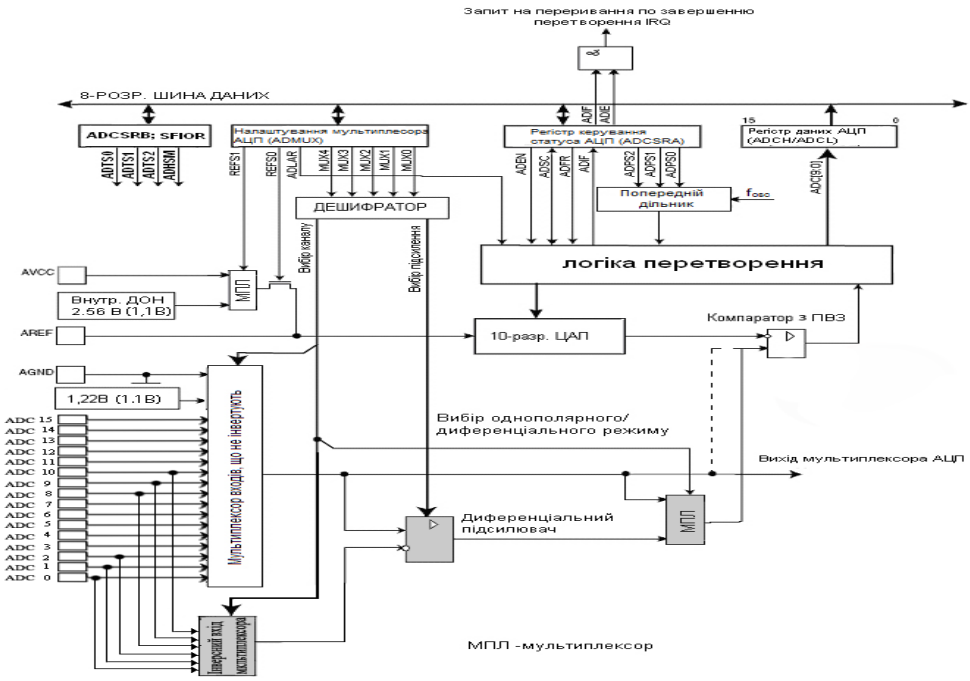


Рис. 1. Функціональна схема модуля АЦП

Модуль АЦП *містить ПВЗ*, що зберігає вхідний аналоговий сигнал та під час перетворення підтримує напругу на вході безпосередньо АЦП на постійному рівні.

АЦП може обробляти вхідні сигнали у *двох режимах*: однополярному та диференціальному. У разі *однополярного режиму* вхідний сигнал поступає на один із входів ADC0…ADC15. У разі *диференціального режиму* 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.

1.2.2.2. Програмування модуля

У [2], як приклад, *наведено*: адреси регістрів, що використовуються для керування модулем АЦП деяких моделей сімейства Mega; формати цих регістрів та описання окремих розрядів. *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* [2].

Режим роботи АЦП визначається *станом розряду* ADFR. Якщо його встановлено в *одиницю*, АЦП працює в режимі *безперервног*о 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. *Рекомендації по вибіру* джерела опорної напруги (ДОН), каналу аналогового введення та каскаду диференціального підсилення наведено у [2].

Запуск АЦП можливий не тільки командою користувача, але й *перериванням* від деяких периферійних пристроїв, наявних у складі МК.

В кінці перетворення у регістрі даних АЦП (ADCH та ADCL) буде *десятирозрядний ДВ*К, який еквівалентний вхідній напрузі. *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 в регістрі ADMUX. *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. За завершенням перетворення АЦП *генерує запит на переривання*.

Одиночне перетворення *запускається* шляхом запису *одиниці* у розряд запуску перетворення АЦП – *ADSC*. Даний розряд залишається в одиничному стані в 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.

1.2.2.3. Формування тактового сигналу

Для формування тактового сигналу модуль АЦП має *попередній дільник*, який формує похідні частоти відносно частоти синхронізації МК (рис. 2). Коефіцієнт *ділення встановлюється* за допомогою розрядів ADPS2; ADPS1 та ADPS0 в регістрі

ADCSRA [2]. З моменту включення АЦП попередній дільник починає лічбу та працює доки розряд ADEN = 1.

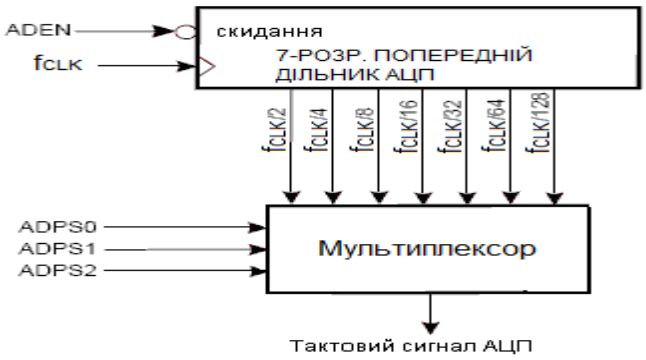


Рис. 2. Схема попереднього дільника АЦП

1.2.2.4. Часові діаграми роботи

Часові діаграми роботи АЦП в різних режимах наведено на рис. 3, 4.

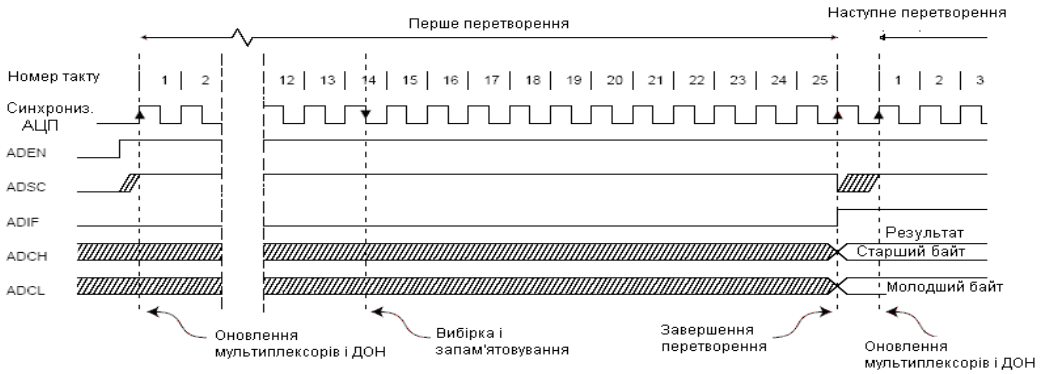


Рис. 3. Часові діаграми роботи АЦП в режимі першого одиночного перетворення

Рис. 4. Часові діаграми роботи АЦП в режимі наступного одиночного перетворення

Перше одиночне *однополярне* перетворення ініціюється встановленням в *одиницю розряду* ADSC у регістрі ADCSRA та виконується *за 25 тактів*

синхронізації. *Починається* перетворення з наступного наростаючого фронту тактового (синхро) сигналу АЦП (рис. 3). *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.

Наступне одиночне перетворення вимагає *13 тактів* синхронізації АЦП (рис. 4). *За завершенням* перетворення результат зберігається в регістрах даних АЦП і встановлюється прапорець ADIF. В режимі одиночного перетворення після його завершення *скидається* розряд ADSC. Його *знову встановити* можна *програмно* і *нове перетворення буде* ініційовано першим наростаючим фронтом тактового сигналу АЦП.

Часові діаграми роботи АЦП *у разі запуску за перериванням та в режимі безперервного перетворення* наведено у [2].

В табл. 1 наведено час для різних типів перетворення.

Таблиця 1. **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

1.2.2.5. Результат перетворення

Для каналу з *однополярним входом* результат перетворення (десятковий еквівалент коду на виході АЦП) визначається виразом:

$$ADC = \frac{1023 * U_{IN}}{U_{REF}}, \quad (1)$$

де U_{IN} – значення вхідної напруги в мілівольтах, а U_{REF} – величина опорної напруги в мілівольтах.

Коефіцієнт передачі

$$K_{ПЕР} = 1023/U_{REF}\,[МЗР/мВ]. \quad (2)$$

На рис. 5 наведено *функцію перетворення* АЦП в *однополярному* режимі.

Код 0x000 відповідає рівню аналогової землі, а 0x3FF – рівню джерела опорної напруги (ДОН) мінус один крок квантування за рівнем [5].

Зв’язок між вхідним сигналом та вихідними кодами для однополярного режиму відображає табл. 2.

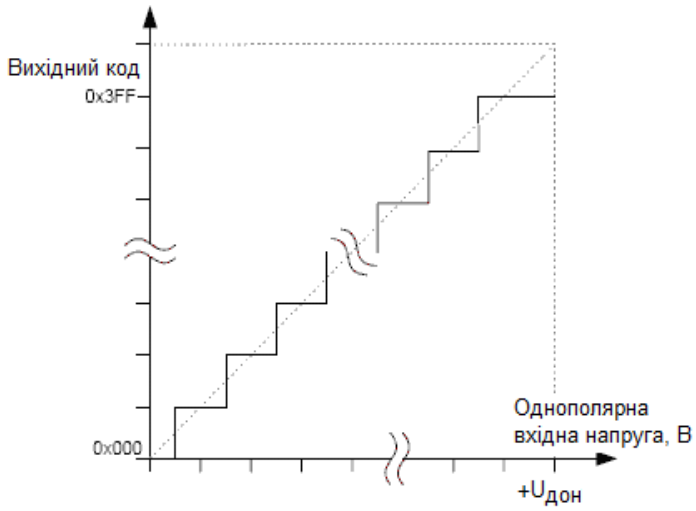


Рис. 5. Функція перетворення АЦП для однополярного режиму

Таблиця 2. Зв’язок між вхідним і вихідним кодами для однополярного режиму

$U_{АЦПn}$*	**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 десятковий/шістнадцятковий еквівалент коду на виході АЦП:

$$ADC = 1023 \cdot 1000/2560 = 400 = 0x190.$$

Для *каналів з диференціальним* входом результат перетворення визначається виразом, який наведено у [2].

1.2.2.6. Моделювання модуля АЦП та цифрового вольтметра

У [4] *описано моделювання* модуля АЦП та цифрового вольтметра в пакеті PROTEUS. Наведено схеми алгоритмів роботи моделей та робочі програми мовою С. Отримані результати моделювання підтверджують працездатність цих алгоритмів та робочих програм.

1.3. Застосування ЦАП під час виведення аналогової інформації у МПС

1.3.1. Загальна характеристика ЦАП

Цифро-аналогові перетворювачі *призначені* для перетворення цифрових сигналів в аналогові. Вони служать для сполучення цифрових і аналогових пристроїв та широко використовуються для керування аналоговими пристроями за допомогою МП/МК *у таких галузях техніки*, як: системи керування технологічними процесами; виконавчі пристрої програмованих верстатів та роботів; дискретна автоматика; вимірювальна автоматика тощо. Серед різних схемних виконань ЦАП застосування знаходять *перетворювачі* з *резисторною матрицею* (РМ) R-2R із підсумовуванням струмів або підсумовування напруг [5; 6].

Нижче наведено вирази для розрахунку вихідної напруги, максимальної вихідної напруги та коефіцієнту передачі ЦАП, в яких використовується резисторна матриця R-2R із *підсумовуванням напруг*:

$$U_{\text{вих}} = \frac{\text{U}_{\text{ОП}}}{2^8} \cdot N_B \quad , \tag{3}$$

$$U_{\text{вих.max}} = \frac{\text{U}_{\text{ОП}}}{2^8} \sum_{i=0}^{7} 2^i, \tag{4}$$

$$K_{\text{ЦАП}} = \frac{U_{\text{ОП}}}{2^8} \left[\frac{\text{В}}{\text{МЗР}} \right], \tag{5}$$

де $N_B = \sum_{i=0}^{7} a_i \cdot 2^i$ – десятковий еквівалент значення вхідного ДВК; $\text{U}_{\text{ОП}}$ – значення опорної напруги.

Якщо, наприклад, $\text{U}_{\text{ОП}}$ = 5,12 В, тоді $K_{\text{ЦАП}}$ = 20 mВ/МЗР.

1.3.2. Особливості архітектури модуля ЦАП в складі AVR-мікроконтролерів

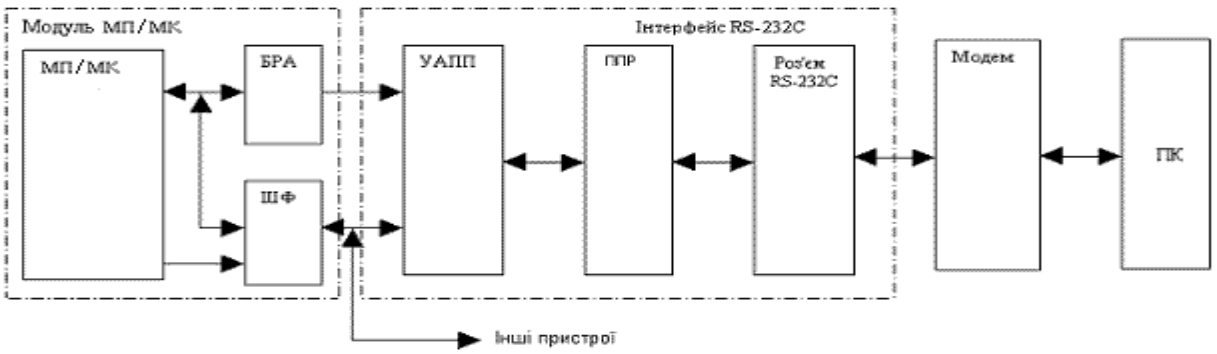
Частина AVR-мікроконтролерів, наприклад, XMega мають модуль ЦАП.

Його виконано за КМОН-технологією з використанням резисторної матрицею R-2R із *підсумовуванням напруг*. Основні характеристики модуля ЦАП, функціональні схеми, режими роботи, їх опис, та програмування розглянуто у [2].

2. ЗВ'ЯЗОК МП/МК З МОДЕМОМ

2.1. Структурна схема інтерфейсу RS-232

Обмін інформацією між МП/МК і модемом може здійснюватися через *інтерфейс* RS-232 [2; 4]. Структурну схему сполучення МП/МК з модемом за допомогою інтерфейсу RS-232 *наведено на рис. 6*.



МП - мікропроцесор; БРА – буферний регістр адреси
ШФ – шинний формувач; УАПП – універсальний асинхронний послідовний програмований приймач – передавач;
ППР – пристрій перетворення рівнів; ПК - персональний комп'ютер

Рис. 6. Структурна схема сполучення МП/МК з модемом за допомогою інтерфейсу RS-232

Схема *містить*: універсальний асинхронний послідовний програмований приймач-передавач (УАПП); пристрій перетворення рівнів (ППР); роз'єм RS-232; буферний регістр адреси (БРА) та шинний формувач (ШФ). УАПП *перетворює* дані з паралельного формату в послідовний під час передачі (виведення) з МП/МК і з послідовного формату в паралельний під час прийому (введення) у МП/МК. В якості УАПП може використовуватись, *наприклад, мікросхема* TL16550. *Обмін* інформацією у МПС з використанням УАПП ведеться *в асинхронному послідовному старт-стопному режимі.*

2.2. Формат даних інтерфейсу RS-232

Формат даних інтерфейсу RS-232, які передаються в канал зв'язку наведено на рис. 7.

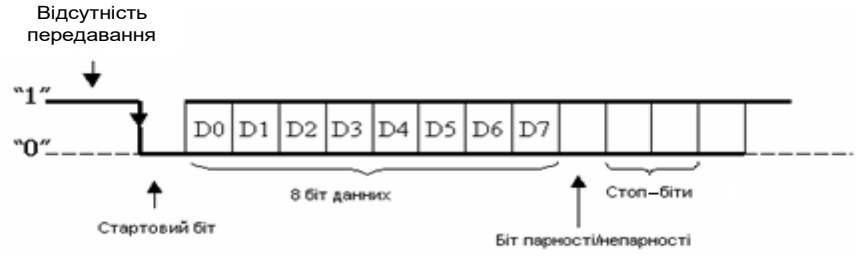


Рис. 7. Формат даних інтерфейсу RS-232

Власне *дані* (5, 6, 7 чи 8 біт) супроводжуються *стартовим нульовим* бітом, бітом *парності/непарності* (якщо такий контроль програмно передбачено) і *стоповим одиничним* сигналом, що включає 1; 1,5 чи 2 стоп-біти. Одержавши стартовий біт, приймач вибирає з лінії біти даних через визначені інтервали часу. Дуже важливо, щоб *тактові частоти приймача і передавача* були *однаковими* та *стабільними*. Швидкість передачі за RS-232 може обиратися згідно інформації, яку наведено у [2].

2.3. Пристрій перетворення рівнів

Всі сигнали RS-232 передаються/приймаються *спеціально обраними рівнями*, що забезпечують високу завадостійкість зв'язку (рис. 8).

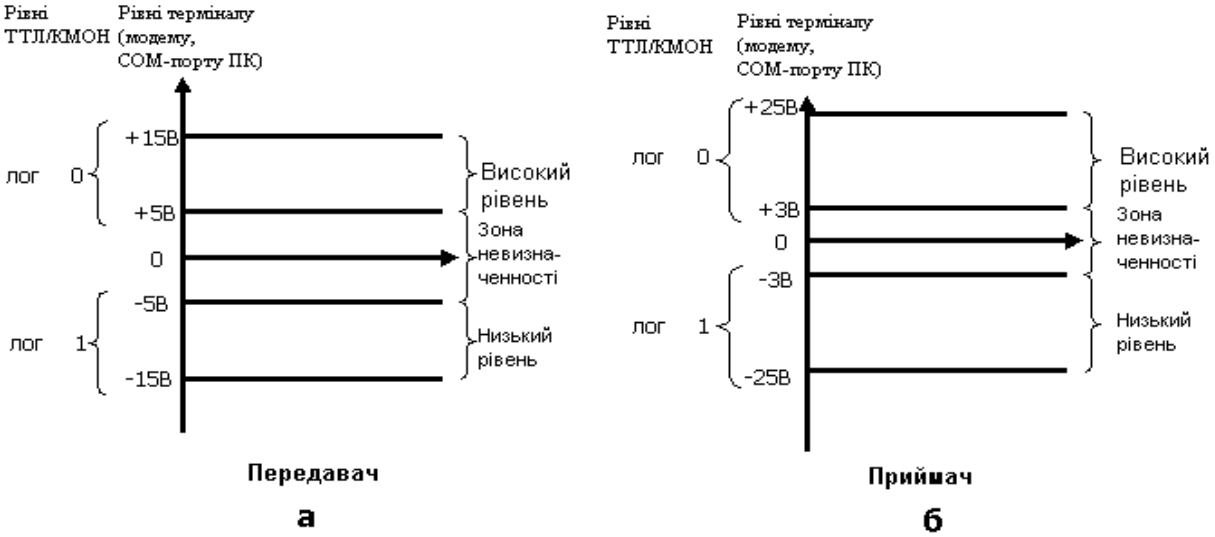


Рис. 8. Рівні сигналів RS-232 на передавальній (*а*) та на приймальній (*б*) кінцях лінії зв'язку

Дані передаються/приймаються *в інверсному вигляді*: логічній *цифровій одиниці* відповідає *низький* рівень, а логічному *нулю* – *високий* рівень. Як видно з рис. 8 під час передачі цифрового *логічного нуля* на виході інтерфейсу формується *високий* рівень напруги в діапазоні: +5 В…+15 В, а під час передачі цифрової *логічної одиниці* – низький рівень напруги в діапазоні: –5В…–15В.

Під час прийому на вхід інтерфейсу надходить *високий рівень* напруги в діапазоні: +3 В…+25 В, що несе інформацію *про цифровій логічний нуль*, чи *низький* рівень напруги в діапазоні: –3 В…–25 В, що відображає *цифрову логічну одиницю*.

Пристрої перетворення рівнів використовують *для узгодження* рівнів ТТЛШ/КМОН-сигналів, що діють у МПС, з рівнями сигналів послідовного інтерфейсу, що передаються у лінію зв'язку або приймаються з лінії зв'язку.

Різні *варіанти схемної реалізації* ППР розглянуто в [4], одним із яких є застосування мікросхеми фірми MAXIM: MAX232A.

2.4. Роз'єм RS-232

Для зв'язку інтерфейсу RS-232 із зовнішнім терміналом (модемом) або ПК можуть використовуватися, наприклад, 9-контактні *роз'єми* (рис. 9).

Нижче описано *призначення контактів роз'єма*: SG – сигнальне заземлення, нульовий провід; TxD – дані, що передаються МП/МК у послідовному коді (від'ємна логіка); RxD – дані, що приймаються МП/МК у послідовному коді (від'ємна логіка); DCD – виявлення несучої даних (детектування сигналу, що

приймається МП/МК); DTR – запит передавача термінала (модему); DSR – готовність передавача термінала (модему); RTS – запит приймача термінала (модему); CTS – готовність приймача термінала (модему); RI – індикатор виклику.

У [1] *наведено функціональні* схеми двох модулів структурної схеми інтерфейсу (рис. 6), характеристику окремих регістрів мікросхеми УАПП TL16550 та рекомендації по їх програмуванню.

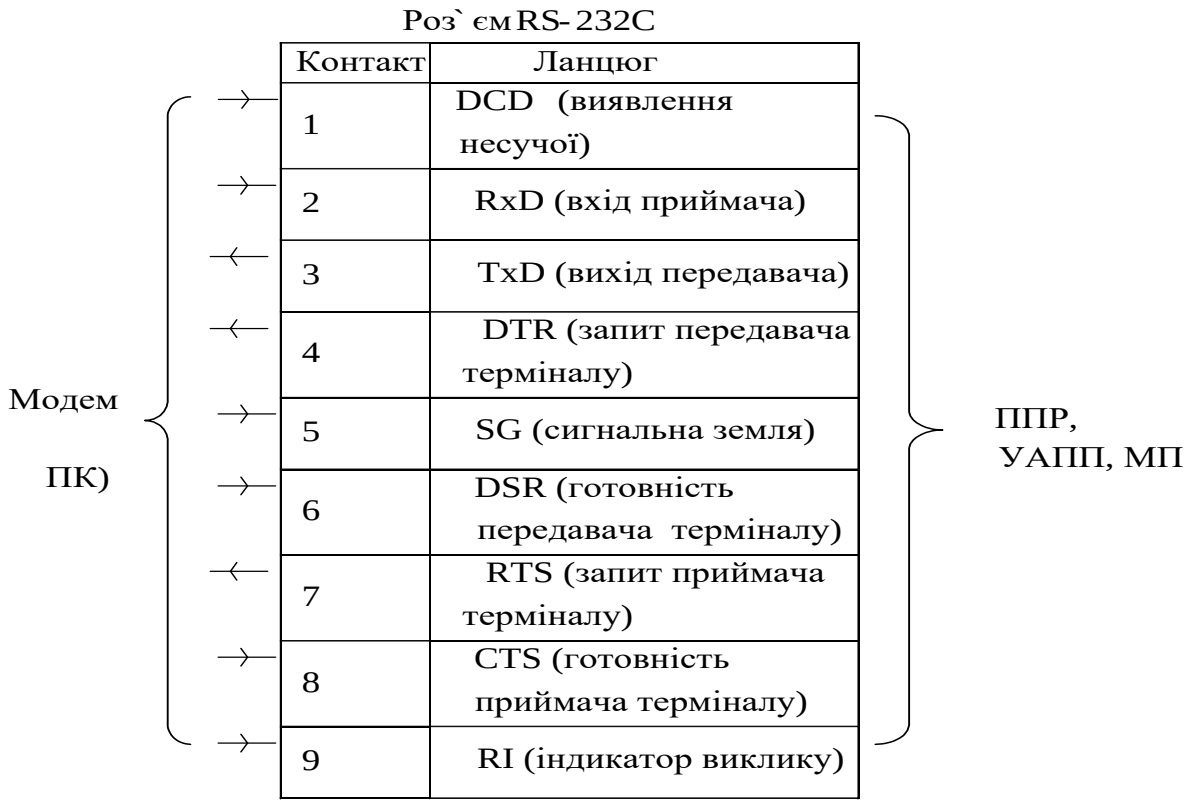


Рис. 9. 9-контактний роз'єм RS-232

Контрольні запитання та завдання

1. Опишіть особливості введення/виведення аналогової інформації у МПС.
2. Поясніть принцип роботи АЦП послідовного наближення.
3. Обґрунтуйте необхідність застосування ПВЗ у МПС.
4. Як розрахувати абсолютну та відносну похибки АЦП від квантування за рівнем?
5. З яких міркувань обирається число розрядів АЦП?
6. Як обирається величина кроку квантування за часом?
7. До складу яких моделей AVR-мікроконтролерів входить модуль АЦП? Назвіть його розрядність.
8. Назвіть основні параметри АЦП.
9. Назвіть основні джерела опорної напруги для АЦП.
10. Назвіть режими роботи АЦП.

11. Яку роль у функціональній схемі модуля АЦП AVR-МК виконують: дешифратор, мультиплексор та пристрій вибірки та зберігання?
12. В якому регістрі зберігається результат перетворення АЦП?
13. Який регістр відповідає за налаштування мультиплексора АЦП?
14. Наведіть та поясніть формати регістрів стану і керування АЦП.
15. За якими перериваннями може відбуватися запуск АЦП?
16. Яку функцію виконує попередній дільник АЦП?
17. Як формується тактовий сигнал АЦП?
18. Поясніть часові діаграми роботи АЦП.
19. Назвіть способи вирівнювання результату АЦП.
20. Яке вирівнювання слід використовувати якщо досить точності 8-розрядного значення?
21. Яким буде результат перетворення для каналів з однополярним входом?
22. Якою формулою визначається коефіцієнт передачі АЦП?
23. Поясніть функцію перетворення АЦП у разі зміни однополярного сигналу.
24. Назвіть методи підвищення точності перетворення АЦП.
25. Опишіть відмінності в програмах у разі моделювання модуля АЦП та цифрового вольтметра.
26. Як розраховується відносна похибка АЦП від квантування за рівнем?
27. Чим відрізняється виведення результату моделювання цифрового вольтметра від моделювання модуля АЦП?
28. Поясніть призначення УАПП.
29. Назвіть та поясніть призначення ППР.
30. Наведіть та поясніть структурну схему сполучення МП/МК та модему за допомогою інтерфейсу RS-232.

ЛЕКЦІЯ 14. МОДЕЛЮВАННЯ МОДУЛЯ АЦП. МОДЕЛЮВАННЯ ЦИФРОВОГО ВОЛЬТМЕТРА

1. МОДЕЛЮВАННЯ МОДУЛЯ АЦП

1.1. Схема моделі та її опис

Схему моделювання модуля АЦП у пакеті PROTEUS 8.6 зображено на рис. 1.

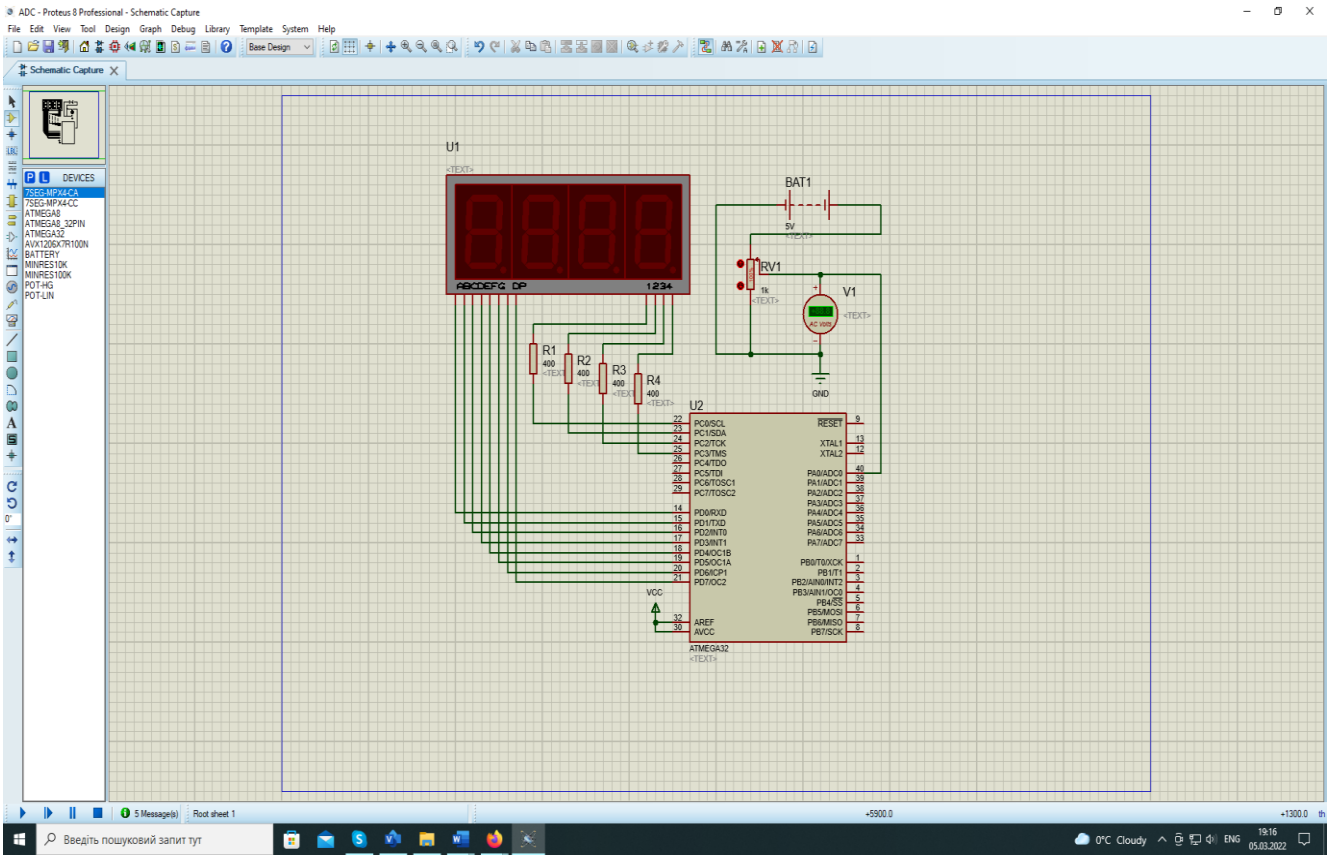


Рис. 1. Схема моделювання модуля АЦП у пакеті PROTEUS 8.6

Схема складається з наступних елементів:

– U2 – AVR-мікроконтролер ATMEGA32;

– R1…R4 – резистори опором 400Ом;

– RV1 – резистор змінного опору 10кОм;

– BAT1 – джерело напруги 5В;

– VCC – джерело живлення мікроконтролера;

– U1 – семисегментний чотирипозиційний індикатор;

– V1 – вольтметр для вимірювання вхідної аналогової напруги.

Вхідна аналогова напруга *подається на лінію PA0* (ADC0) порту A мікроконтролера з виходу дільника напруги +5В (RV1). За допомогою резистора RV1 можна *змінювати* величину вхідної напруги від 0В до +5В.

Після перетворення значення вхідної напруги в цифровий еквівалент, відповідна величина для відображення на індикаторі виводиться в порти C та D мікроконтролера. *Індикатор підключено* до ліній PC0…PC3 порту C та ліній PD0…PD7 порту D.

АЦП *програмується* у режим *безперервного перетворення* з використанням каналу з *однополярним* входом. У лекції 13 представлено функцію перетворення АЦП для однополярного режиму, яка відображає зв'язок між вхідним сигналом та вихідним двійковим кодом.

Для каналів з однополярним (несиметричним) входом результат перетворення визначається виразом:

$$ADC = 1023 \cdot U_{IN}/U_{REF}, \quad (1)$$

де ADC – десятковий еквівалент двійкового коду (десятковий код) на виході АЦП у кількості молодших значущих розрядах (МЗР); U_{IN} – значення вхідної напруги в мілівольтах, а U_{REF} – величина опорної напруги в мілівольтах.

Відповідно коефіцієнт передачі

$$K_{ПЕР} = 1023/U_{REF}\,[МЗР/мВ]. \quad (2)$$

За U_{REF} = 5В, $К_{ПЕР}$ = 1023/5000 = 0,2046 [МЗР/мВ].

Якщо, наприклад, Uвх = 5В = 5000мВ, то десятковий код на виході буде дорівнювати:

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

На рис. 2 наведено схему моделі після запуску процесу моделювання.

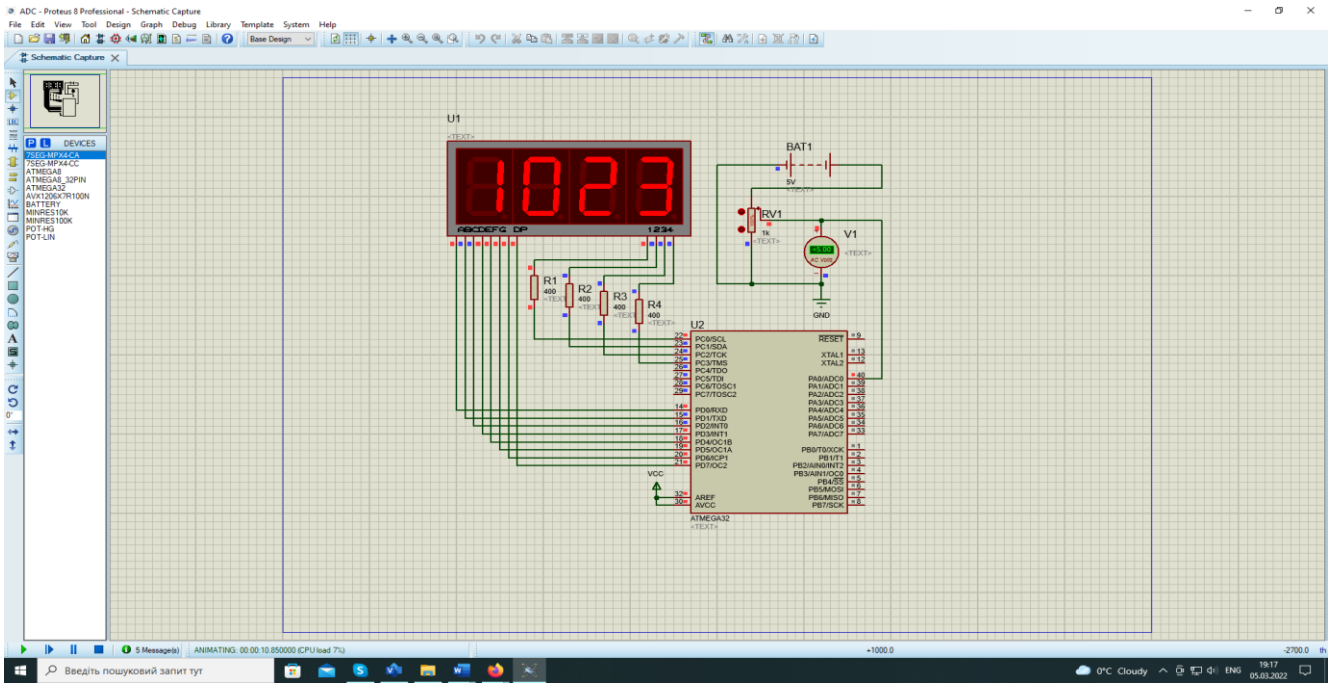


Рис. 2. Схема моделі АЦП після запуску процесу моделювання

За допомогою дільника RV1 на вольтметрі V1 встановлено *напругу +5В*, на індикаторі U1 бачимо *число 1023*, що відповідає наведеному вище розрахунку.

1.2. Схема алгоритму роботи моделі

Схему алгоритму роботи моделі наведено на рис. 3.

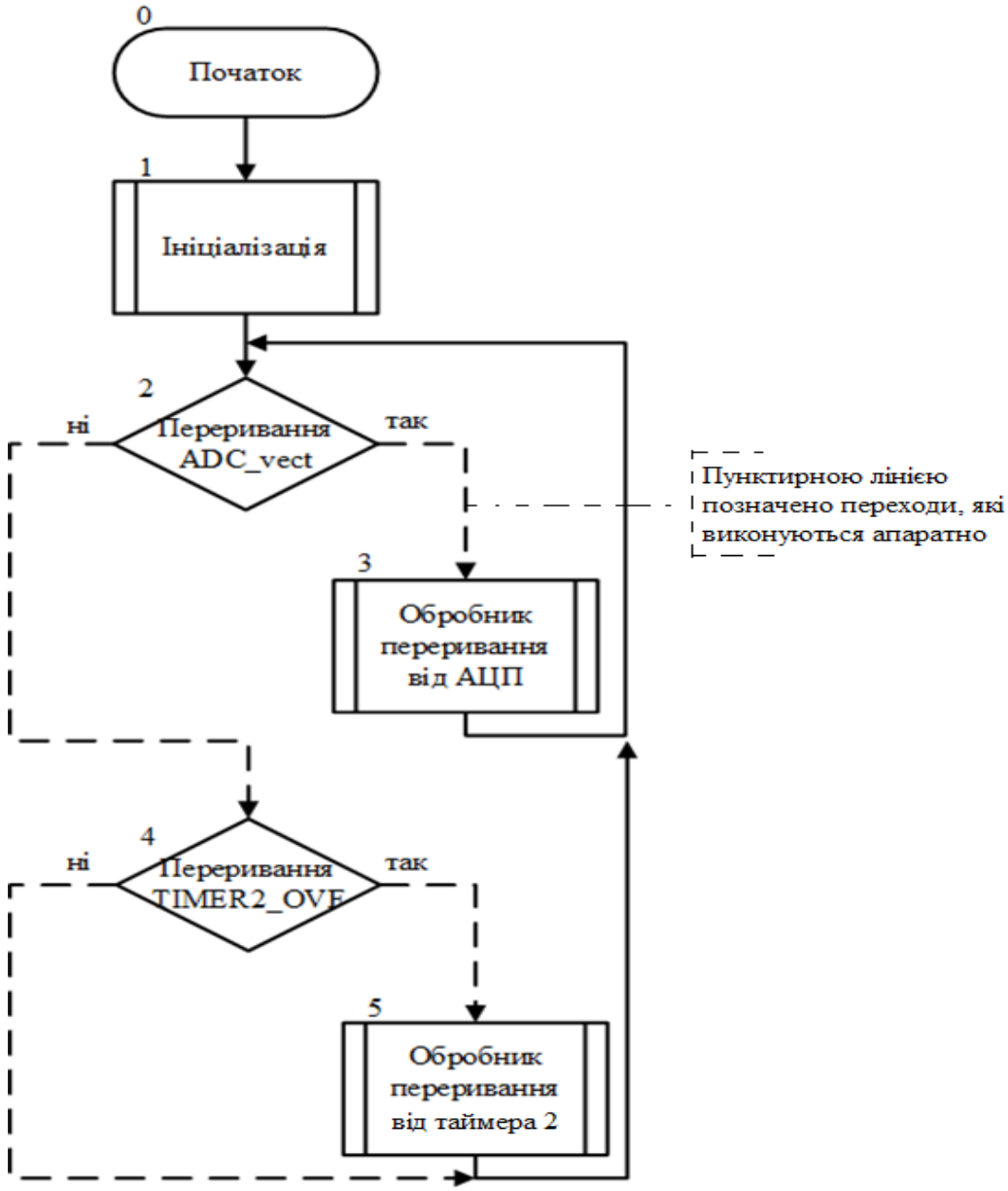


Рис. 3. Схема загального алгоритму роботи моделі

Після виконання ініціалізації (блок 1) АЦП запускається в режим *безперервного перетворення*. *Паралельно починає роботу* таймер/лічильник Т2, *під час кожного переповнення* якого викликається відповідна підпрограма. 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.

*У разі чергов*ого переповнення 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.

Схему алгоритму ініціалізації наведено на рис. 4.

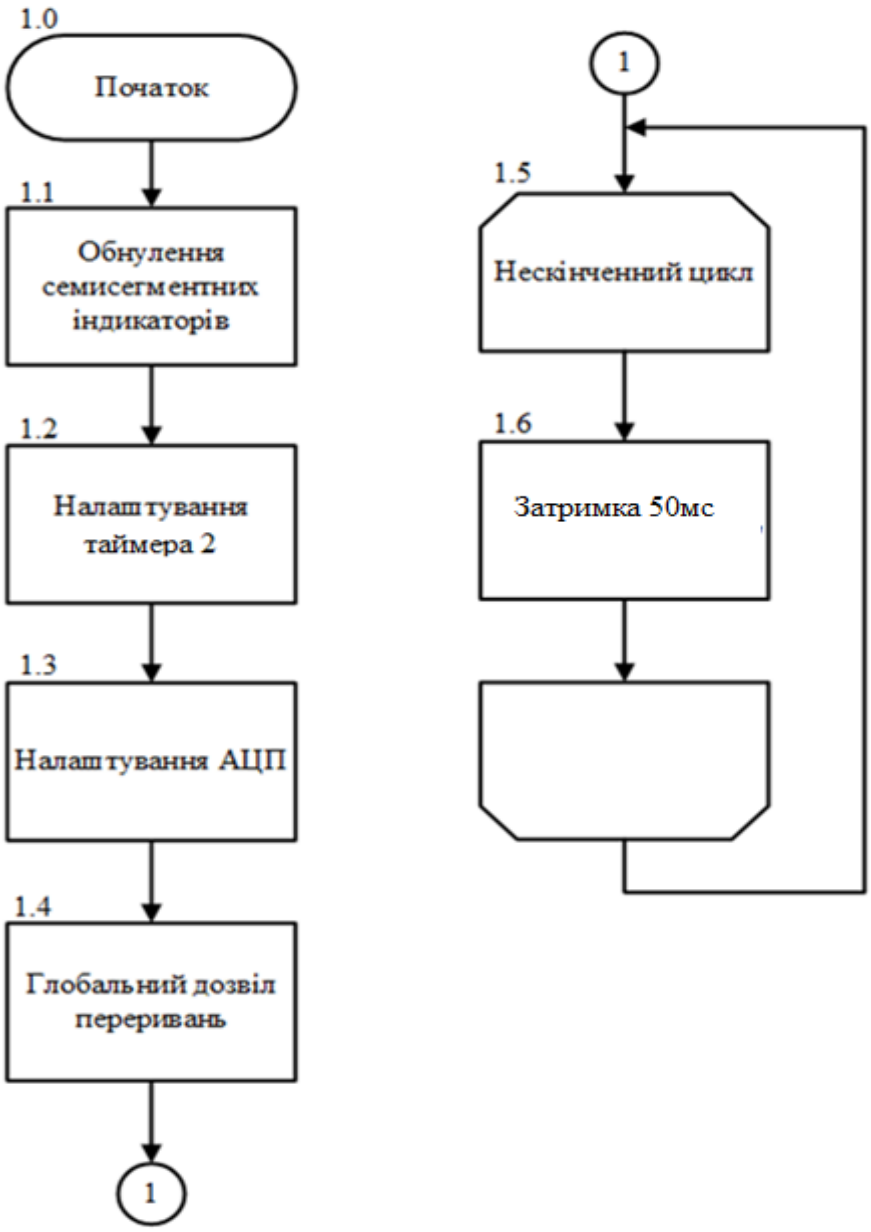


Рис. 4. Схема алгоритму ініціалізації

Схему алгоритму обробки переривання від таймера 2 наведено на рис. 5.

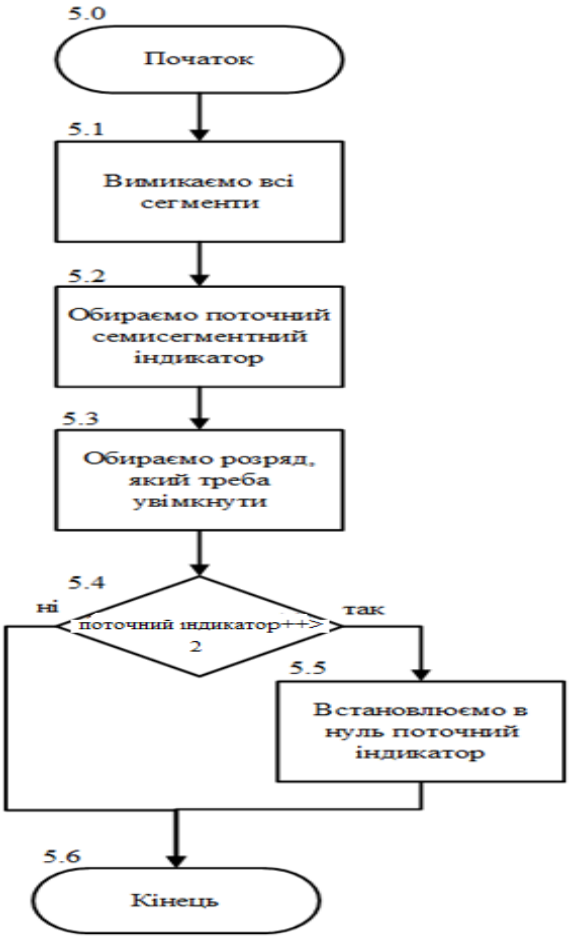


Рис. 5. Схема алгоритму обробки переривання від таймера 2

Схему алгоритму обробки переривання завершення перетворення АЦП наведено на рис. 6.

Рис. 6. Схема алгоритму обробки переривання завершення перетворення АЦП

1.3. Робоча програма

Нижче наведено робочу програми мовою С, яку розроблено згідно з алгоритмом, який наведено на рис. 3…6.

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

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

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

// Блок 1 - Ініціалізація програми
int main (void)
{
     // Блок 1.1 - Налаштування портів семисегментних індикаторів на
     // виведення та їх обнулення
    DDRC = 0xFF;
    PORTC = 0x00;
    DDRD = 0xFF;
    PORTD = 0x00;
    // Блок 1.2 - Налаштування таймера 2
    TIMSK |= (1 << TOIE2); // Дозвіл переривання від таймера 2
    TCCR2 |= (1 << CS21); // Коефіцієнт ділення попереднього
                          //дільника = 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) // Коефіцієнт ділення переддільника
                               // АЦП = 64
    |(1 << ADIE); // Дозвіл переривань від АЦП
    ADMUX &= (~(1 << REFS1))&(~(1 << REFS0)); // Зовнішнє ДОН

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

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

```
}

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

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

    //Блок 5.3
    switch (current_indicator)
    {
        case 0:
        PORTD = ~(SEG[display % 10000 / 1000]); // Вмикаємо цифру
                                                // ТИСЯЧ
        break;
        case 1:
        PORTD = ~((SEG[display % 1000 / 100])); // Вмикаємо цифру
                                                // СОТЕН
        break;
        case 2:
        PORTD = ~(SEG[display % 100 / 10]); // Вмикаємо цифру
                                            //ДЕСЯТКІВ
        break;
        case 3:
        PORTD = ~(SEG[display % 10 / 1]); // Вмикаємо цифру ОДИНИЦЬ
        break;
    }
    if ((current_indicator++) > 2) // Блок 5.4 – Переходимо на
                                   //наступний індикатор
    current_indicator = 0; // Блок 5.5 – Обнуляємо, якщо
                           //current_indicator за межами
                           //чотирипозиційного індикатора
}
```

Для налаштування параметрів АЦП в робочій програмі виконуються такі дії: в регістрі ADCSRA в логічну одиницю встановлюються біти: ADEN – дозвіл АЦП; ADSC – запуск перетворення; ADATE – безперервний режим роботи; ADPS2 та ADPS1 – коефіцієнт ділення переддільника: 64; ADIE – дозвіл переривань від АЦП.

Щоб обрати зовнішнє джерело опорної напруги біти REFS1 і REFS0 в регістрі ADMUX скидаються у логічний нуль.

Для відображення отриманого десяткового коду перетворення АЦП на семисегментних індикаторах використовується таймер/лічильник Т2 мікроконтролера. У разі його переповнення відбувається переривання та результат аналого-цифрового перетворення відображається на чотирипозиційному семисегментному індикаторі. Для цього в порт С 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 С дорівнюють нулю). Під час вибору *другого* індикатора PC1 = 1, інші біти порту С дорівнюють нулю, і т. д. Після вибору відповідного індикатора за допомогою математичної операції *отримання остачі*, визначається 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 в програмі перед виведенням в порт D інвертується.

Для дозволу переривань від таймера/лічильника Т2 використовується регістр TIMSK (Timer/Counter Interrupt MaSK Register – регістр маски переривань від таймерів/лічильників) [2]. *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 Т2 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 треба задати: CS22 = CS20 = 0, CS21 = 1 [2].

Нижче наведено пояснення фрагменту програми виведення на чотири семисегментних індикатори результату перетворення АЦП – ADC, який надходить у десятковому коді згідно (1), яку наведено вище.

Для виведення ADC використовується *математична операція «остача»*, яка в мові програмування «С» обчислюється оператором «%».

Наприклад, у разі $U_{вх.АЦП}$ = 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, якщо 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.

2. МОДЕЛЮВАННЯ МОДУЛЯ ЦИФРОВОГО ВОЛЬТМЕТРА

2.1. Схема моделі та її опис

Робоча модель цифрового вольтметра та її опис співпадає з моделлю АЦП (див. розділ 1).

Основним елементом цифрового вольтметра є аналого-цифровий перетворювач, в якості якого в роботі використовується відповідний модуль мікроконтролера ATmega32. Архітектуру цього модуля розглянуто в лекції 13.

Різниця між виведенням результату перетворення АЦП, який описано у підрозд. 1.3, полягає в тому, що раніше на дисплей виводилася цифрова інформація у чотирирозрядному десятковому коді, а в моделі цифрового вольтметра відображається абсолютне значення вхідної напруги у вольтах з точністю до сотих долей вольта.

На рис. 7 наведено *зовнішній вигляд 7-сегментного* індикатора, який входить до складу чотирипозиційного цифрового дисплею.

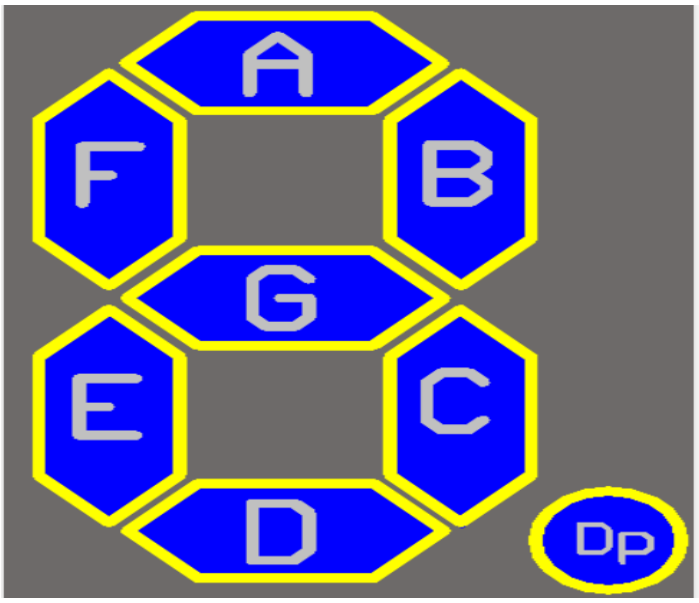


Рис. 7. Зовнішній вигляд 7-сегментного дисплею

Для отримання на індикаторі чисел потрібно керувати сегментами індикатора A, B, C, D, E, F, G та точкою Dp згідно до табл. 1, 2.

Таблиця 1. Зв'язок між цифрами на дисплеї та значенням керувальних сигналів без використання точки

Цифра на дисплеї	Dp	G	F	E	D	C	B	A	Значення керувальних 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 2. Зв'язок між цифрами з точкою на дисплеї та значенням керувальних сигналів з використанням точки

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

На рис. 8, 9 наведено декілька прикладів роботи моделі.

2.2. Схема алгоритму роботи моделі

Схема алгоритму роботи моделі цифрового вольтметра подібна схемі алгоритму роботи моделі модуля АЦП, яку наведено вище на рис. 3…6.

Ініціалізація модуля АЦП, таймера T2 мікроконтролера та робота моделі у більшості виконується аналогічно описанному у підрозд. 1.2. *Різниця полягає* у виведенні результату моделювання на дисплей.

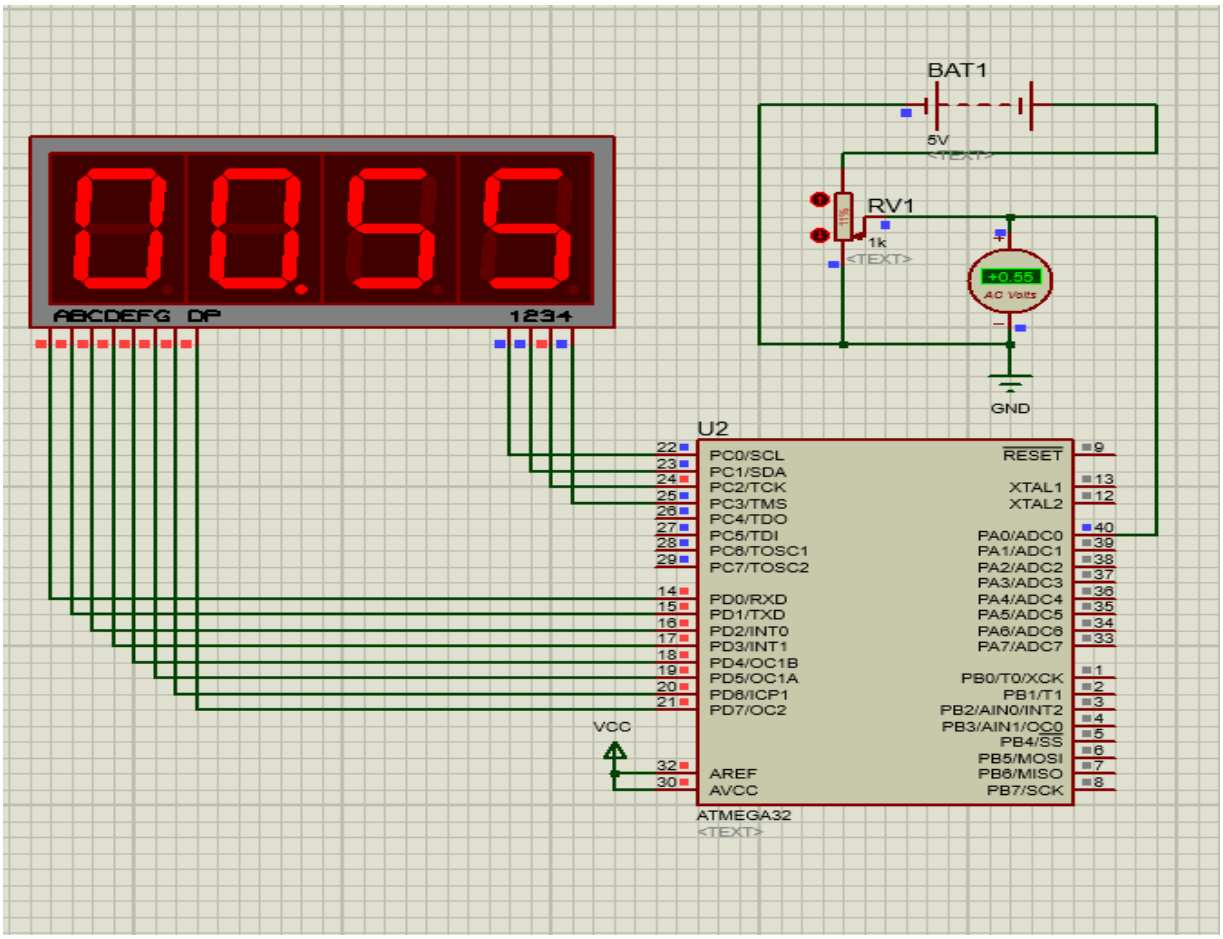


Рис. 8. Робота цифрового вольтметра за $U_{BX} = 0{,}55В$

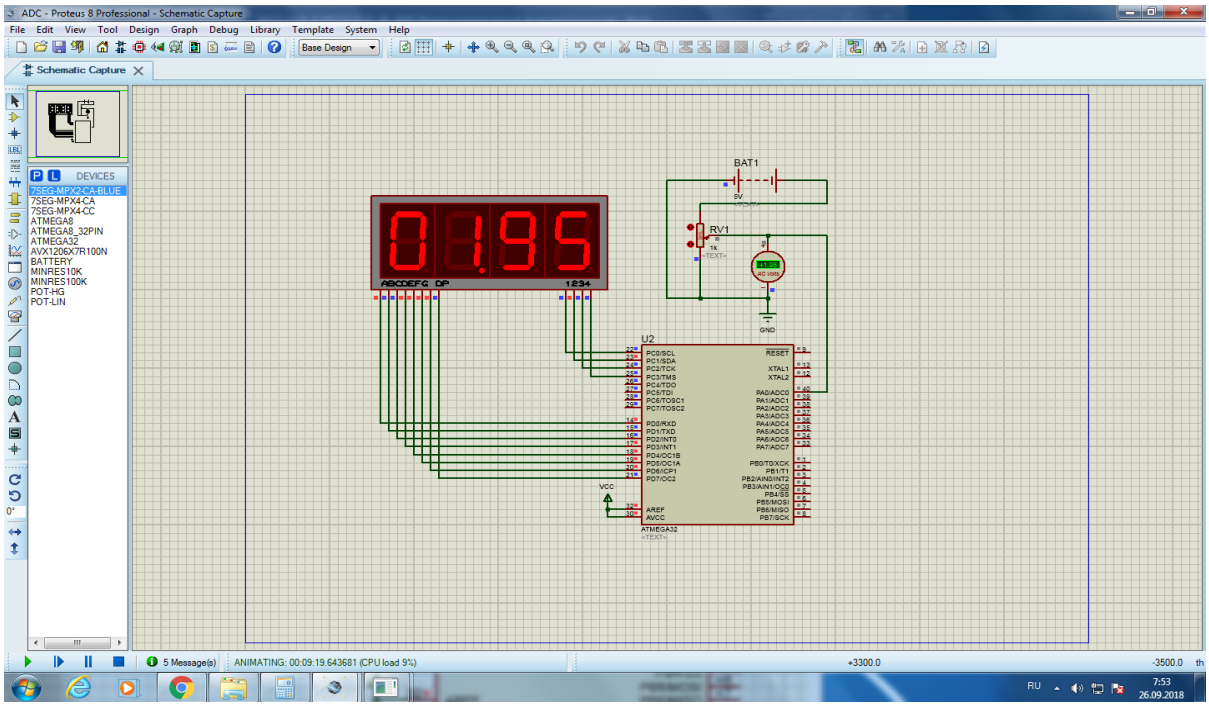


Рис. 9. Робота цифрового вольтметра за $U_{BX} = 1{,}95В$

Нижче наведено пояснення перетворення значення ADC, яке отримується після завершення роботи модуля АЦП в десятковому коді, в значення вхідної напруги у вольтах для його виведення на дисплей.

Як відмічено у підрозд. 1.1 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

$$К_{ПЕР} = 1023/U_{REF} \text{ [МЗР/мВ]}.$$

За U_{REF} = 5В, $К_{ПЕР}$ = 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: ADC = 113.

Перетворення значення ADC_value = ADC у змінну «display», виконується згідно з виразом:

$$display = (ADC\_value) \cdot (5/1023) \cdot 100.$$

Тоді за ADC = 113

$$display = 0113 \cdot (5/1023) \cdot 100 = 55.$$

Тобто, за 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:

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: ADC = 399.

Значення «display = 0399 · (5/1023) · 100 = 195.

Це значення програма обробляє наступним чином:

1) 195%10000/1000 = 195/1000 = 0,195. Цифра 0 виводиться на перший індикатор;
2) 195%1000/100 = 195/100 = 1,95. Цифра 1 з десятковою крапкою виводиться на другий індикатор;
3) 195%100/10 = 95/10 = 9,5. Цифра 9 виводиться на третій індикатор;
4) 195%10/1 = 5/1 = 5. Цифра 5 виводиться на четвертий індикатор.

2.3. Робоча програма

```
// 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 – Налаштування та обнулення портів C та D керування
// семисегментними індикаторами
   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 поточного значення АЦП
    }
    // Блок 4 – Умова переривання від таймера T2
    ISR (TIMER2_OVF_vect)
    {   // Блок 5 – 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)
        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 з десятковою крапкою
            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. Опишіть особливості введення/виведення аналогової інформації у МПС.
2. Назвіть основні джерела опорної напруги для АЦП.
3. Назвіть режими роботи АЦП.
4. Поясніть часові діаграми роботи АЦП.
5. Яким буде результат перетворення для каналів з однополярним входом?
6. Якою формулою визначається коефіцієнт передачі АЦП?
7. Поясніть функцію перетворення АЦП у разі зміни однополярного сигналу.
8. Опишіть відмінності в програмах у разі моделювання модуля АЦП та цифрового вольтметра.
9. Як розраховується відносна похибка АЦП від квантування за рівнем?
10. Чим відрізняється виведення результату моделювання цифрового вольтметра від моделювання модуля АЦП?
11. Опишіть схему моделювання модуля АЦП в пакеті PROTEUS 8.6.
12. В який режим програмується модуль АЦП у разі його моделювання?
13. Чому дорівнює коефіцієнт передачі модуля АЦП під час його моделювання?
14. За допомогою якої математичної операції визначається відповідна цифра під час її виведення на індикатор? Відповідь пояснити.
15. Який таймер/лічильник мікроконтролера використовується під час виведення на семисегментні індикатори?
16. Який сигнал використовується в якості тактового для таймера 2? Відповідь пояснити.
17. Чим відрізняється виведення результату моделювання цифрового вольтметра від моделювання модуля АЦП?

ЛЕКЦІЯ 15. МІКРОКОНТРОЛЕРНА МЕРЕЖА RS-485. МІКРОКОНТРОЛЕРНА МЕРЕЖА НА ОСНОВІ ІНТЕРФЕЙСУ SPI

1. МІКРОКОНТРОЛЕРНА МЕРЕЖА RS-485

1.1. Загальні відомості про мікроконтролерні мережі

Сьогодні у МПС використовуються мікроконтролерні мережі (МК-мережі) на базі інтерфейсів: RS-485; RS-232; SPI; I^2C (TWI); 1-WIRE; CAN; LIN-мережі і т. ін. [2; 4]. В цій лекції буде розглянуто мережі на базі інтерфейсів: RS-485; RS-232 та SPI.

1.2. Мікроконтролерна мережа RS-485

1.2.1. Загальна характеристика мережі

В мережах RS-485 використовується стандарт передачі даних *за двопровідним напівдуплексним багатоточковим послідовним каналом з'язку* (КЗ): RS-485/EIA-485 (RS485 – англ. Recommended Standard 485, EIA-485 – англ. Electronic Industries Alliance-485).

СтандартRS-485 *розроблений* спільно двома *асоціаціями*: Асоціацією електронної промисловості (EIA – Electronics Industries Association) та Асоціацією промисловості засобів зв'язку (TIA – Telecommunications Industry Association). *Раніше EIA* маркувала всі свої стандарти префіксом «RS» (англ. Recommended Standard – рекомендований стандарт). Багато інженерів продовжують використовувати це позначення, проте EIA/TIA *з метою полегшити* ідентифікацію походження своїх стандартів офіційно *замінив «RS» на «EIA/TIA»*. Нині різні розширення стандарту RS-485 охоплюють широке розмаїття програм. Цей стандарт став основою для створення цілого *сімейства обчислювальних мереж*, які широко використовуються у *промисловій автоматизації*.

Стандарт RS-485 це назва популярного інтерфейсу, що використовується в промислових АСК ТП для з'єднання контролерів та іншого обладнання. Головна *відмінність* RS-485 *від* також широко *розповсюдженого* RS-232 – можливість *об'єднання декількох пристроїв у мережу*.

У стандарті RS-485 для передачі і прийому даних часто використовується *вита пара дротів.* Передача даних здійснюється за допомогою *диференціальних сигналів. Різниця напруг між дротами* однієї полярності означає логічну одиницю, різниця іншої полярності – нуль.

Таким чином, інтерфейс RS-485 *передає і приймає лише послідовності* логічних рівнів, не піклуючись про логічні подробиці передачі даних у вигляді поділу на байти, видачі старт-стопних сигналів, корекції помилок, керування

напрямком і черговістю передачі даних. Усе це *має бути реалізовано окремо*, наприклад, за допомогою використання модуля універсального асинхронного передавача/приймача (УАПП). Останній входить до *інтерфейсу* RS-232, або до більшості сучасних мікроконтролерів.

Для передачі сигналів формувач активного передавача (ПД) генерує *дві комплементарні* (логічно протилежні) *напруги*: високий рівень на виході А, низький рівень на виході В, або високий рівень на В, низький рівень на А. *Усі інші* ПД у цей час, щоб уникнути конфлікту і спотворення сигналу перебувають *у третьому* (високоімпедансному) стані [2; 4].

На рис. 1 показано, як стандарт EIA-RS-485 визначає напруги Uoa, Uob і Uo.

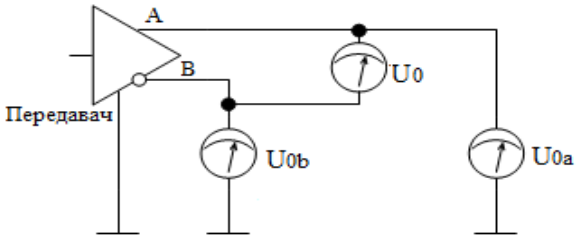
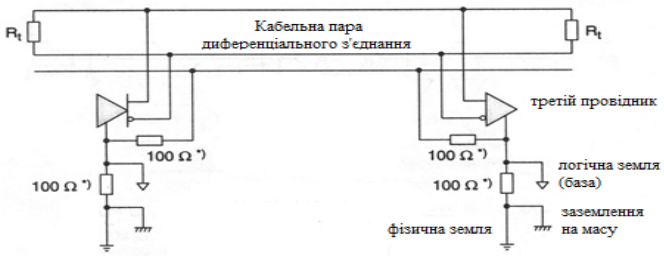


Рис. 1. Вимірювання напруг U_{0a}, U_{0b} і U_0

Коли рівень напруги U_{0a} – низький, тоді U_{0b} – високий, а коли U_{0a} – високий, тоді U_{0b} – низький. *Формувачі сигналів мережі RS-485 повинні використовувати спільну землю*, тому термін «двопровідна лінія» непридатний до RS-485. У загальному випадку, необхідно використовувати *трипровідну* лінію (рис. 2).



Примітка. *) – резистори, включені послідовно з лініями повернення сигналу, обмежують циркулюючі струми.

Рис. 2. Схема підключення інтерфейсу RS-485/ EIA-485 до трипровідної лінії

Третім проводом може бути високоякісне реальне заземлення з низьким власним опором. Іноді рекомендують логічну «землю» заземлювати на реальну землю (на шасі) через резистор у 100 Ом для обмеження струму.

Приймачі (ПРМ) мережі розроблено так, щоб реагувати *на різницю напруг* $U_0 = A - B$, *більшу ніж 200 Мв* (напруги вимірюються щодо спільного потенціалу

(землі)). *За досить довгих лініях зв'язку* та наявності струму через третій провід (за різних значень падіння напруги в локальних заземлюючих ланцюгах) можлива поява значного постійного потенціалу, який додається до напруги в сигнальних проводах A і B. Саме тому приймачі RS-485 працюють *у діапазоні напруг*: –7…+12 В, а не 0…+5 В (діапазон вихідної напруги ПД).

Загальні рекомендації щодо використання RS-485 наведено у [2; 4].

1.2.2. Кількість вузлів мережі

Максимальна кількість вузлів у мережі *не може перевищувати 32*, але під час використання повторювачів (*ретрансляторів*) можна з'єднувати разом фактично необмежену кількість вузлів, але одночасно зі зростанням діаметра топології мережі *збільшуються і затримки*, а *швидкість* передачі даних може *стати неприйнятно низькою*.

1.2.3. Швидкість та дальність передачі даних

Мережа RS-485 забезпечує передачу даних *зі швидкістю до 10 Мбіт/с. Максимальна дальність* залежить від швидкості. *Наприклад,* під час швидкості 10 Мбіт/с максимальна довжина лінії – 120 м, а у разі швидкості 100 Кбіт/с – 1200 м.

1.2.4. Протоколи та роз'єми для передачі

Стандарт не нормує формат інформаційних кадрів і протокол обміну. *Найбільш часто для передачі* байтів даних використовується протокол інтерфейсу RS-232: стартовий біт, біти даних, біт паритету (якщо потрібно), стоповий біт. Тобто у мікроконтролерній мережі *фізичний інтерфейс* RS-485 використовується *разом з інтерфейсом RS-232*, який програмується.

Протоколи обміну працюють за принципом – «ведучий–ведений». Один пристрій на магістралі *є ведучим* (master) і ініціює обмін *з одним з ведених* пристроїв (slave), які розрізняються логічними адресами. *Одним з популярних* протоколів є протокол Modbus RTU [1; 4].

Тип з'єднувачів (роз'ємів) та їх розпаювання також *не обумовлюються* стандартом. Трапляються з'єднувач DB9, клемні з'єднувачі т. ін.

Під час підключення *слід правильно приєднати* сигнальні ланцюги, що зазвичай звуться A і B.

1.2.5. Узгодження «відкритого» кінця кабелю

Для узгодження «відкритого» кінця кабелю з рештою лінії використовують *резистори-термінатори*, які забезпечують *усунення відбиття0 сигналу*.

Номінальний опір резисторів відповідає хвильовому опору кабелю, і для кабелів на основі витої пари *зазвичай становить*: 100…120 Ом. Наприклад, широко поширений кабель UTP-5, використовуваний для прокладки Ethernet, має імпеданс 100 Ом. Для іншого типу кабелю може знадобитися інший номінал.

1.2.6. Рівні сигналів у мережі

Інтерфейс RS-485 використовує *диференційну схему* передачі сигналу. Це означає, що *рівні напруг на сигнальних* ланцюгах A і B змінюються в протифазі, як показано на рис. 3.

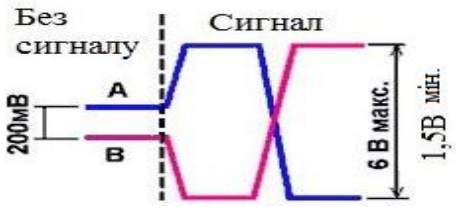


Рис. 3. Рівні сигналів в мережі

Передавач повинен забезпечувати мінімальний рівень сигналу 1,5 В за максимального навантаження – 32 стандартних входи і 2 резистора-термінатора, і не більше 6 В на холостому ході [4]. Рівні напруг вимірюють диференціально, один сигнальний дріт відносно іншого. *Пороговий діапазон* прийнятого сигналу RS-485: ±200 мВ.

1.2.7. Зсув на сигнальних ланцюгах

За відсутності сигналу в мережі на сигнальних ланцюгах *повинен бути* невеликий зсув. Цей зсув призначений *для захисту приймачів* (ПРМ) від помилкових спрацьовувань. *Рекомендується створювати* зсув трохи більше 200 мВ (зона недостовірності вхідного сигналу відповідно до стандарту), у цьому разі ланцюг A «підтягують» до додатного полюса джерела, а ланцюг B – до «спільного» (рис. 4).

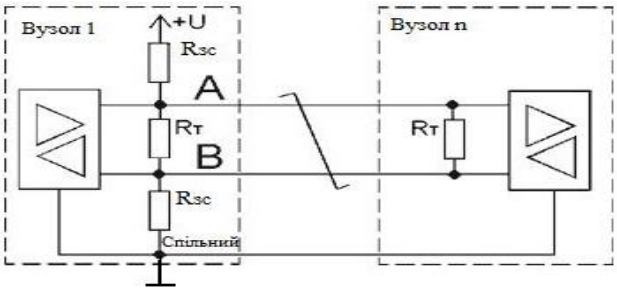


Рис. 4. Схема реалізації ланцюга зсуву на сигнальних ланцюгах

Номінали резисторів зсуву розраховують, виходячи з *необхідного зсуву*: U_{AB} > 200 мВ і напруги джерела живлення: +5 В. Якщо приймемо $R_{зс}$ = 560 Ом, тоді

U_A = 2,63В, а напруга U_B = 2,38В [4]. Різниця напруг $U_{AB} = U_A - U_B$ = 2,63 В – 2,38 В= = 0,25 В = 250 мВ, тобто це відповідає умові $U_{AB} > 200$ мВ.

Якщо на лінії знаходиться *багато ПРМ*, то номінал $R_{зс}$ має бути менше. В довгих лініях передачі необхідно також *враховувати опір крученої пари*, який може «з'їдати» частину зміщуючої різниці потенціалів для віддалених від місця підтяжки пристроїв. *Для довгої лінії краще ставити* два комплекти підтягуючих резисторів в обидва віддалені кінці поруч із термінаторами.

1.2.8. Реалізація інтерфейсу RS-485

Інтерфейс RS-485 може бути реалізований на основі мікросхем фірми MAXIM [1; 4]. Ці мікросхеми *є ПРМ/ПД (трансиверами*) для взаємодії за протоколами RS-485 і RS-422 (табл.1).

Таблиця 1. Параметри RS-485 мікросхем MAXIM

Найменування	Дуплекс/ напівдуплекс	Швикість передачі (Мбіт/с)	Обмеження наростання вихідної напруги	Режим мікро–спожи–вання	Дозвіл приймача/ передавача	Струм спокою (мкА)	Число пристроїв на шині	Число виводів
MAX481	Напів–дуплекс	2.5	Ні	Так	Так	300	32	8
MAX483	Напів–дуплекс	0.25	Так	Так	Так	120	32	8
MAX485	Напів–дуплекс	2.5	Ні	Ні	Так	300	32	8
MAX487	Напів–дуплекс	0.25	Так	Так	Так	120	128	8
MAX488	Дуплекс	0.25	Так	Ні	Ні	120	32	8
MAX489	Дуплекс	0.25	Так	Ні	Так	120	32	14
MAX490	Дуплекс	2.5	Ні	Ні	Ні	300	32	8
MAX491	Дуплекс	2.5	Ні	Ні	Так	300	32	14
MAX1487	Напів–дуплекс	2.5	Ні	Ні	Так	230	128	8

Кожна мікросхема включає в себе один вихідний формувач і один приймач. Всі мікросхеми живляться від однополярної напруги: +5 В. Приймачі розглянутих мікросхем працюють в діапазоні: – 7 ... +12 В.

Мікросхеми MAX487 і MAX1487 мають вчетверо більше навантаження на лінію, що дозволяє підключити до 128 приймачів на спільну шину. Мікросхеми MAX488…MAX491 забезпечують *повнодуплексну* передачу даних, а мікросхеми MAX481, MAX483, MAX485, MAX487 та MAX1487 розроблено для *напівдуплексного* режиму (рис. 5).

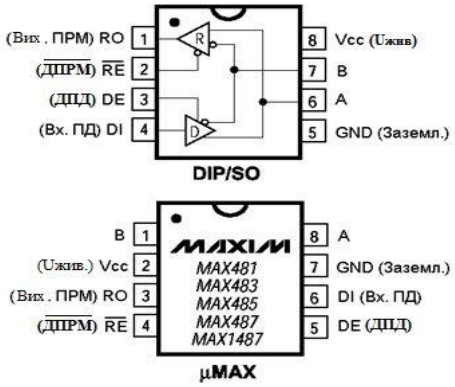


Рис. 5. Призначення виводів напівдуплексних мікросхем RS-485 фірми MAXIM

Нижче наведено *приклад* типового використання мікросхем RS-485 у напівдуплексній (рис. 6) та в дуплексній мережах (рис. 7).

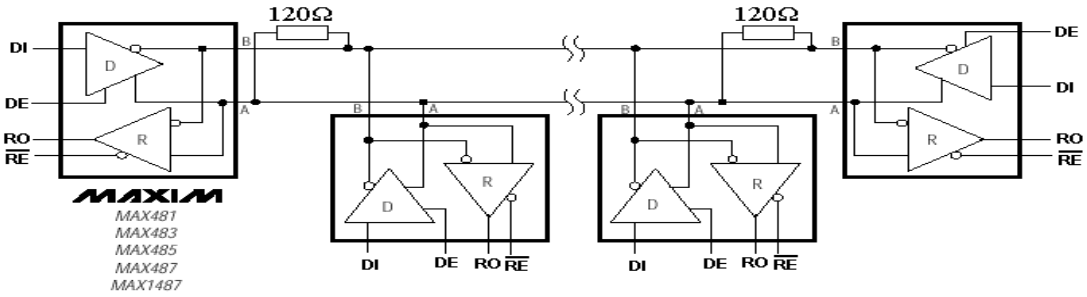
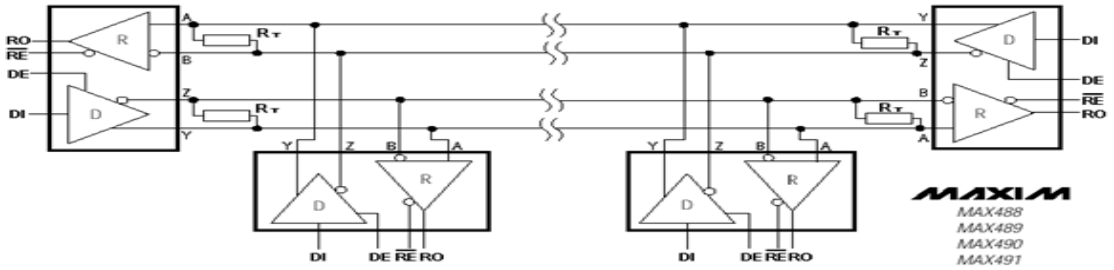


Рис. 6. Типове використання мікросхем RS-485 в напівдуплексній мережі



Примітка. $\overline{RE}$ і DE тільки для MAX489/MAX491. R_T – термінатор: 120 Ω.

Рис. 7. Типове використання мікросхем RS-485 у дуплексній мережі (мікросхеми MAX488…MAX491)

Розглянуті мікросхеми можуть працювати не тільки у мережі, а й у простому *з'єднанні типу «точка-точка»* (рис. 8).

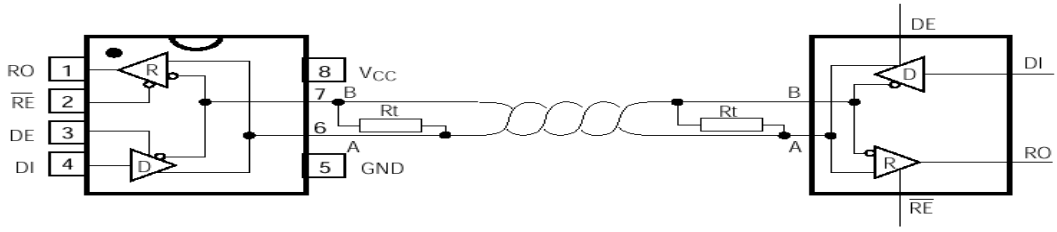


Рис. 8. Приклад простого з'єднання мікросхем типу точка-точка (напівдуплекс)

Нижче наведено приклад використання мікросхеми MAX485 *спільно з мікроконтролером* AT89C51 (рис. 9).

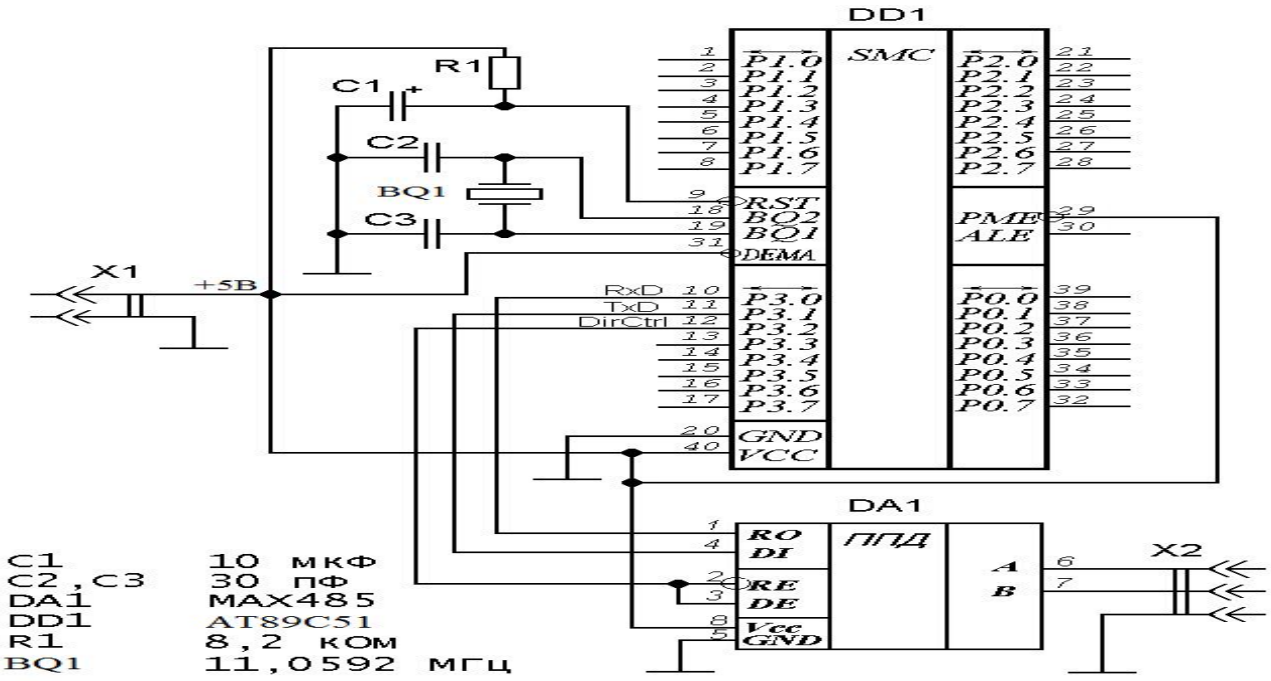


Рис. 9. Принципова схема підключення МК AT89C51 до мікросхеми RS-485

Розглянутий вище інтерфейс *RS-485 є фізичним* інтерфейсом, який забезпечує обмін даними у розглянутих вище мережах. Цей інтерфейс часто використовується *разом з програмованим (логічним)* інтерфейсом RS-232 для зв'язку з комп'ютером, або мікроконтролером.

RS-232 (англ. Recommended Standard 232, інша назва EIA232 [2] – *стандарт для асинхронного* інтерфейсу: UART (універсальний асинхронний приймач-передавач). Пристрій, який підтримує цей стандарт, широко відомий як *послідовний порт* персональних комп'ютерів. Історично стандарт мав широке поширення у телекомунікаційному устаткуванні. Нині він *використовується для підключення до комп'ютерів широкого спектра обладнання*, невибагливого до швидкості обміну, особливо у разі значного віддалення його від комп'ютера та відхилення умов застосування від стандартних. У комп'ютерах, зайнятих офісними та розважальними програмами, практично *витіснений інтерфейсом USB*.

Стандарт RS-23 часто використовується для взаємодії мікроконтролерів різних архітектур, які мають у своєму складі інтерфейс UART, з іншими цифровими пристроями та периферією.

RS-232 *забезпечує передачу даних* та деяких *спеціальних сигналів* між терміналом (англ. Data Terminal Equipment, DTE) та комунікаційним пристроєм (англ. Data Communications Equipment, DCE) *на відстань* до 15 метрів на

максимальній швидкості – 115200 бод. Оскільки цей інтерфейс відомий не лише простотою програмування, але й невибагливістю, у реальних умовах *ця відстань збільшується у багато разів із приблизно пропорційним зниженням швидкості.*

На рис. 10 навено *функціональну схему мікроконтролерної мережі RS-485* в мікропроцесорній системі з розподіленим керуванням.

У [4] розглянуто *приклади двох моделей* мережі з використанням інтерфейсів *RS-485 та RS-232*, наведено: схеми алгоритмів роботи, робочі програми мовою «С» та описано їх моделювання в пакеті »PROTEUS».

2. МІКРОКОНТРОЛЕРНА МЕРЕЖА НА ОСНОВІ ІНТЕРФЕЙСУ SPI

2.1. Загальна характеристика інтерфейсу SPI

У МПС крім задачі передачі неперервного потоку інформації, достатньо часто необхідно передавати *окремі цифрові пакети чи команди керування*. Для передачі *такого виду* інформації призначено *синхронний послідовний периферійний інтерфейс (*англ. Serial Peripheral Interface – SPI) [2; 4].

Стандарт SPI було розроблено *компанією Motorola* для мікроконтролерів серії 68000. Завдяки *простоті* та популярності цього стандарту, багато інших виробників вже *багато років його використовують*.

Усі МК сімейства Mega та XMega мають інтерфейс SPI.

За його допомогою може здійснюватися *обмін* даними на відстані *до трьох метрів* зі швидкістю *до 2 Мбіт/с між окремими мікроконтролерами* або *мікроконтролерами* і різними *периферійними* пристроями, *такими, як* цифрові потенціометри, ЦАП/АЦП, FLASH-ПЗП і т. ін.

Крім того, через інтерфейс SPI може бути здійснено програмування МК (*режим послідовного програмування*).

Під час обміну даними інтерфейсом SPI в мікроконтролерній мережі AVR-мікроконтролер може працювати *як ведучий* (режим «Master»), або *як ведений* (режим «Slave»). Користувач може задавати *сім програмованих значень швидкості* передачі та *формат передачі* (від молодшого розряду до старшого або навпаки).

Додатково під час надходження даних за допомогою інтерфейсу SPI можна *виводити МК з режиму* зниженого енергоспоживання *Idle.*

2.2. Характеристика модуля SPI мікроконтролерів AVR

2.2.1. Опис структурної схеми модуля

Структурну схему модуля SPI наведено на рис. 11. Модуль використовує *чотири виводи* МК, які є *лініями портів введення/виведення загального призначення* (табл. 2).

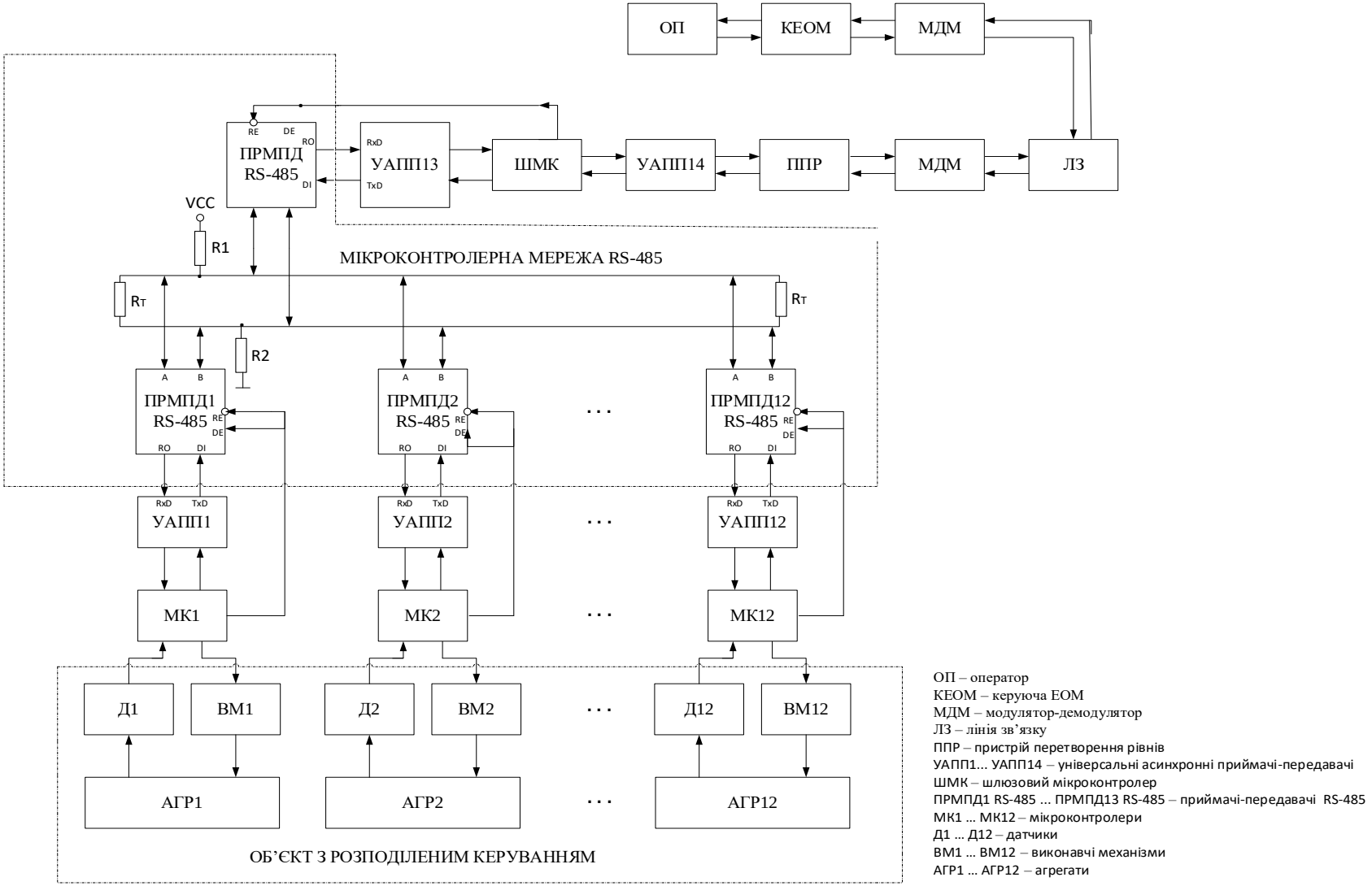


Рис. 10. Функціональна схема мікроконтролерної мережі RS-485 для мікропроцесорної системи з розподіленим керуванням

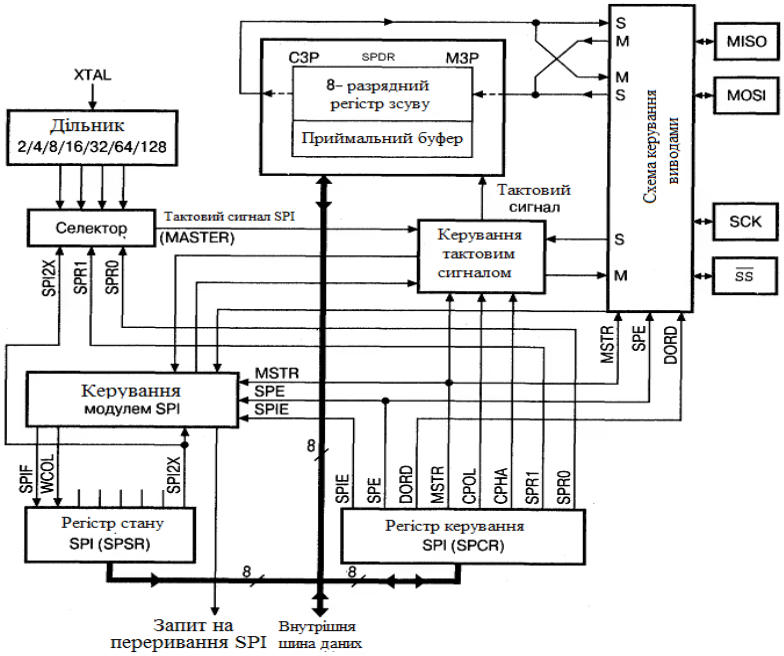


Рис. 11. Структурна схема модуля SPI

Таблиця 2. 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

Режим роботи зазначених виводів (напрям передачі даних) у разі включеного модуля SPI перевизначається згідно табл. 3.

Таблиця 3. 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

Вивід SCK в режимі Master працює як вихід, тому відповідна лінія порту ВВ/ВИВ загального призначення *має бути запрограмована користувачем на виведення*. Напрямок обміну даними: передача (вихід) або прийом (вхід) визначається або програмуванням відповідних розрядів регістра DDRB, або задається схемою керування модулем залежно від режиму роботи «Master»/«Slave». У цьому разі зберігається можливість керування внутрішніми підтягуючими резисторами виводів, що працюють як входи.

2.2.2. Обмін даними між двома мікроконтролерами

З'єднання двох мікроконтролерів (ведучий-ведений) інтерфейсом SPI показано на рис. 12.

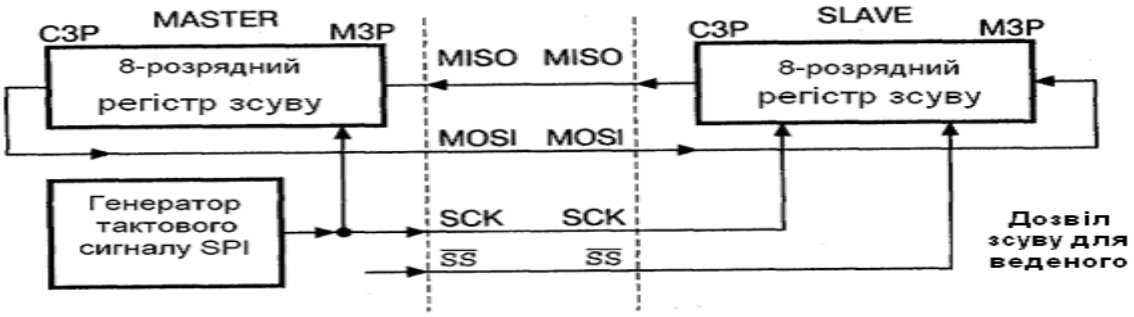


Рис. 12. З'єднання двох МК інтерфейсом SPI

Вивід SCK ведучого МК є *виходом* тактового сигналу, а *веденого* МК – *входом*. *Перед входами 8-розрядних регістрів зсуву* ведучого та веденого МК знаходяться *два синхронних тригери (буфери*), які на *рисунку не показано*. Їх призначення буде описано у п. 2.2.5. 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*, МК працює в режимі «Master», якщо скинуто *в нуль* – у режимі «Slave».

Передача даних здійснюється таким чином. *У разі запису* в регістр даних SPI ведучого МК *запускається генератор* тактового сигналу модуля SPI. Після цього *дані починають порозрядно видаватися* на вивід MOSI ведучого і, відповідно, надходити на вивід MOSI веденого МК. *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 PCR встановлено в одиницю), *генерується запит на переривання*.

Можлива наступна передача залежить від режиму роботи (п. 2.2.5).

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 МК, так і у веденому.* Прийняті байти *зберігаються в приймальних буферах* для подальшого використання.

У модулі реалізовано *одинарну буферизацію під час передачі і подвійну під час прийому.* Це означає, що готовий для передачі байт даних не може бути записано у регістр даних SPI до *закінчення попереднього циклу обміну.*

У разі спроби 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.

Під час прийому даних прийнятий байт треба прочитати з регістра даних до того, як в регістр зсуву надійде останній розряд наступного байта. В іншому випадку попередній байт буде загублено.

2.2.3. Структура SPI-мережі

Структуру SPI-мережі наведено на рис. 13.

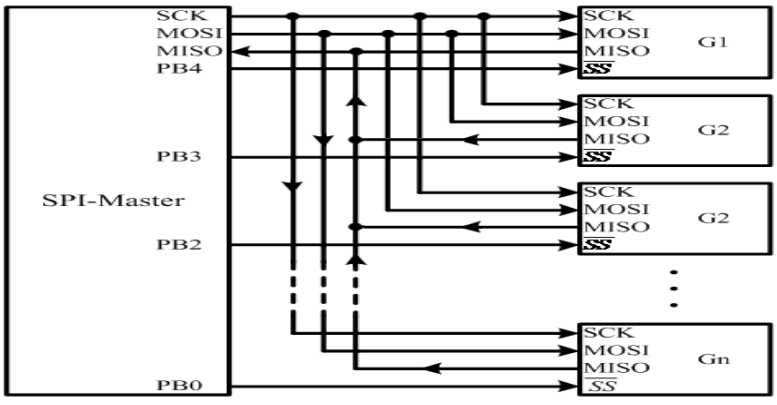


Рис. 13. Структура SPI-мережі

Один з ведених пристроїв з точки зору ведучого пристрою, може бути, наприклад, *тільки блоком передачі.* В якості такого веденого можна навести, наприклад, *АЦП з інтерфейсом SPI*. *Іншим веденим* може бути, наприклад, *ЦАП*, який буде використовуватись *як блок прийому*.

2.2.4. Програмування модуля

Для програмування модуля SPI призначено регістр керування SPCR (рис. 14). Опис його розрядів наведено в табл. 4.

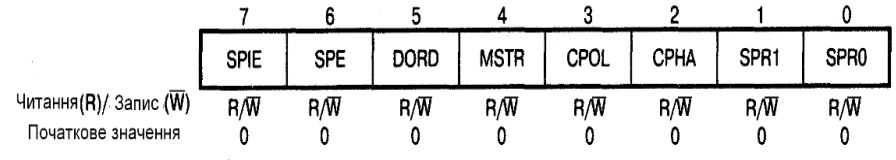


Рис. 14. Формат регістра SPCR

Таблиця 4. Опис 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 (табл. 6)
2	CPHA	Фаза тактового сигналу – визначає момент зчитування сигналу, табл. 6
1, 0	SPR1:SPR0	Програмування швидкості передачі (табл. 7)

Контроль стану модуля, а також додаткове керування швидкістю обміну здійснюється за допомогою регістра SPSR. Розряди з сьомого по перший цього регістра доступні тільки для читання, а нульовий розряд – як для читання, так і для запису. Формат цього регістра наведено на рис. 15, а опис його розрядів – у табл. 5.

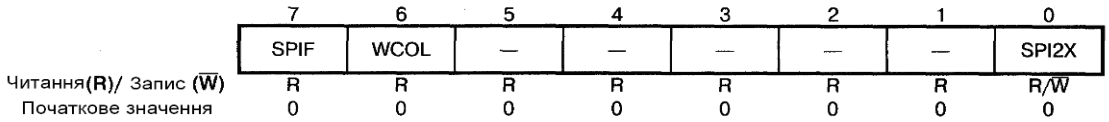


Рис. 15. Формат регістра SPSR

Дані, що *передаються*, записуються в регістр даних SPDR, а дані, які *приймаються*, зчитуються з цього регістра. Запис у цей регістр *ініціює початок передачі,* а *під час його читання* зчитується вміст приймального буфера регістра зсуву.

Таблиця 5. 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 під час переведення МК з режиму «Master» у режим «Slave» за допомогою виводу $\overline{SS}$, див. п. 2.2.7. Прапорець скидається апаратно під час старту підпрограми обробки переривання, або після читання регістра стану SPI з наступним зверненням до регістра даних SPDR
6	WCOL	**Прапорець конфлікту запису** Цей 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 даних SPDR
5...1	–	Зарезервовано, читаються як «0»
0	SPI2X	**Подвоєння швидкості обміну** У разі встановлення цього розряду в одиницю та роботі МК у режимі «Master» подвоюється частота сигналу SCK

Тобто *регістр даних під час читання* служить буфером між регістровим файлом МК і регістром зсуву модуля SPI. *Під час передачі* додаткового буфера немає, а *функцію регістра даних* виконує регістр зсуву.

Адреси регістрів, які використовуються під час програмування модуля наведено у [2].

2.2.5. Режими передачі даних SPI-інтерфейсом

Є *чотири режими передачі* даних SPI-інтерфейсом: режими 0…3 (рис. 16, 17).

Ці режими *розрізняються* відповідністю *між фазою тактового сигналу* SCK – розряд CPHA регістра SPCR (визначає момент зчитування (вибірки) сигналу), його *полярністю* – розряд CPOL регістра SPCR та *даними*, які передаються. Є *чотири комбінації розрядів CPHA і CPOL* регістра SPCR (табл. 6), які визначають відповідний режим роботи модуля.

В режимах 0, 1 (CPHA = 0) *початок обміну визначається* встановленням сигналу на виводі $\overline{SS}$ веденого в активний стан – *логічний нуль*. У цьому разі *ведений виставляє* на лінію MISO старший розряд даних, якщо першим передається СЗР, або молодший – якщо першим передається МЗР (рис. 16).

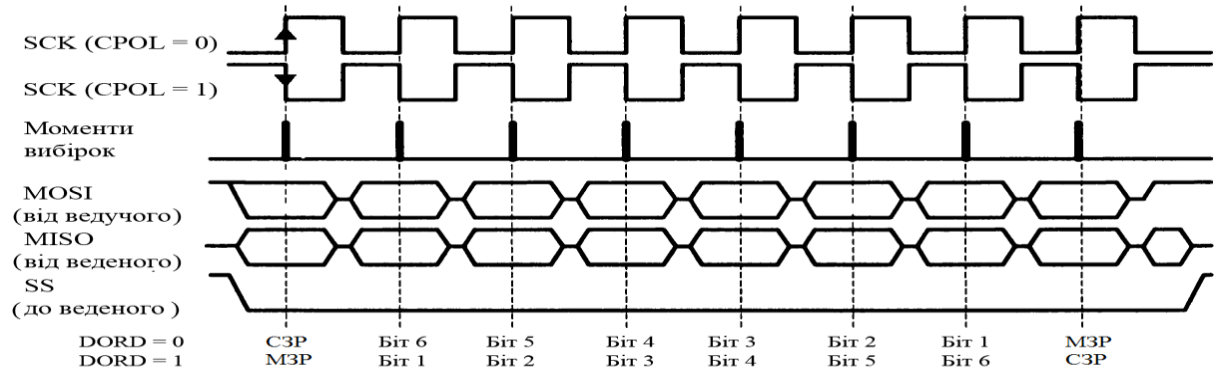


Рис. 16. Передача даних за CPHA = «0» (режими 0, 1)

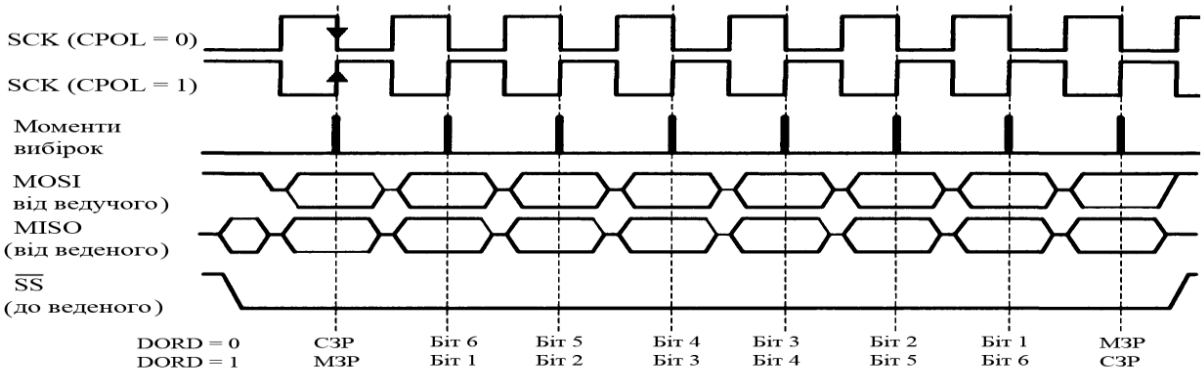


Рис. 17. Передача даних за CPHA = «1» (режими 2, 3)

Таблиця 6. Задання режиму передачі даних

Розряд	**Опис**
CPOL	**Полярність тактового сигналу** «0» – генеруються імпульси додатної полярності, у разі відсутності імпульсів на виводі присутній низький рівень; «1» – генеруються імпульси від'ємної полярності, у разі відсутності імпульсів на виводі присутній високий рівень (рис. 16, 17)
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. 16); «1» – обробка даних виконується за заднім фронтом імпульсів сигналу SCK (для CPOL = «0» – за спадаючим фронтом, а для CPOL = «1» – за наростаючим фронтом) (рис. 17)

Ведучий видає на лінію MOSI СЗР/МЗР на $0{,}5 \cdot T_{SCK}$ раніше, ніж на лінії SCK з'являється перший імпульс, T_{SCK} – період синхроімпульсів.

Перед входами 8-розрядних регістрів зсуву ведучого та веденого МК (рис. 12) знаходяться два синхронних тригери (буфери), які на рисунку не показано. В ці буфери

першим перепадом сигналу на лінії SCK *записується* значення сигналу, яке присутнє на їх інформаційному вході. *Цей момент* на рис. 16 відмічено на лінії «*Моменти вибірок*».

Другим перепадом імпульсів на лінії SCK *відбувається зсув* інформації вліво відповідно до рис. 12. У цьому разі *стан буферів переписується* в молодші розряди регістра зсуву, а *черговий вихідний біт* з регістра зсуву виставляється на лінії MOSI/MISO та *з затримкою записується* у буфери.

Вказаний зсув у часі між моментом видачі чергового розряду в лінію зв'язку між МК і моментом фіксації цього біта в буфері *дозволяє компенсувати часові затримки* під час 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.

У режимах 0, 1 після обміну байтом треба подати на вхід $\overline{SS}$ веденого МК напругу *високого* рівня та перевести останній у стан *очікування*. *Після цього*, щоб почати передавати наступний байт, треба подати на вхід $\overline{SS}$ веденого МК напругу *низького* рівня (рис. 18*а*).

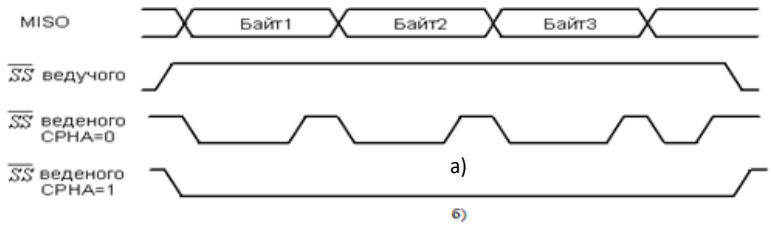


Рис. 18. Вплив сигналу $\overline{SS}$ для веденого на початок обміну даними: *а* – режими 0, 1; *б* – режими 2, 3

У режимах 2, 3 сигнал вибору веденого $\overline{SS}$ може залишатися в активному стані – логічний нуль протягом передачі декількох байт інформації (рис. 18*б*). Це трохи спрощує логіку програмування модуля SPI. *Під час передачі* даних від ведучого до веденого та у зворотному напрямку *всі непарні перепади* SCK викликають *видачу* чергового біта посилки з регістра зсуву ПД *на лінію*. Кожний *парний перепад* використовується *для запису* цього біта *в буфер* перед регістром зсуву.

Таким чином, *під час передачі* біти даних *висуваються* з виходів регістрів зсуву *від'ємними або додатними фронтами* сигналу SCK, а *фіксуються в буферах* на входах регістрів зсуву *додатними або від'ємними фронтами* синхросигналу SCK. Ці фронти *зсунуто відносно моментів зміни бітів* на виході регістрів зсуву на половину періоду (рис. 16, 17). *Це гарантує достатній час на встановлення даних на входах відповідних регістрів зсуву перед висуванням.*

2.2.6. 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. 7).

Таблиця 7. 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 МК.

Для пристрою, який перебуває *в режимі «Slave*», стан цих розрядів *байдужий*. Нормальне функціонування МК у режимі «Slave» гарантується на частотах, менших або рівних $f_{CLK}/4$ [2].

2.2.7. Використання виводу $\overline{SS}$

Цей вивід призначено *для вибору активного веденого* пристрою та в режимі «Slave» завжди є входом (рис. 13). У разі подачі на нього напруги *низького рівня* модуль SPI активується і вивід MISO перемикається *в режим виведення* даних (*якщо це задано користувачем*, див. табл. 3). *Інші виводи* модуля SPI в цьому режимі є входами. 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 МК перебуває *в режимі «Master»* (розряд MSTR регістра SPCR встановлено в одиницю), *напрямок* передачі даних через вивід $\overline{SS}$ визначається *користувачем* (табл. 3). Якщо вивід сконфігуровано *як вихід*, він використовується для керування виводом $\overline{SS}$ МК, 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 заборонено [2].

2.2.8. Використання інтерфейсу SPI для програмування пам'яті

У більшості МК для програмування (перепрограмування) пам'яті безпосередньо у пристрої може використовуватись *послідовний інтерфейс SPI* (рис. 19) [2].

Рис. 19. Включення МК в режимі програмування послідовним каналом

Як видно з цього рисунку для підключення програматора до пристрою використовуються *три лінії SPI-інтерфейсу*: SCK (тактовий сигнал), MOSI (вхід даних) і MISO (вихід даних). *Відповідність між лініями інтерфейсу і контактами портів ВВ/ВИВ* деяких AVR-мікроконтролерів наведено в табл. 8.

Таблиця 8. Виводи, що використовуються під час програмування 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 МК, наприклад, ATmega64x та ATmega128x виводи, які використовуються *для програмування пам'яті* (табл. 8), *не збігаються* з виводами, призначеними *для штатної роботи інтерфейсу SPI.*

Часові діаграми сигналів під час програмування МК у розглянутому режимі та *значення параметрів* сигналів наведено в [2].

2.2.9. Периферійні пристрої з SPI-інтерфейсом

Для спрощення вибору елементної бази під час проектування електронних схем у [2] наведено деякі мікросхеми: АЦП; ЦАП; давачі й т. ін., що містять інтерфейс SPI.

2.2.10. Моделювання модуля SPI

Моделювання модуля SPI *мікроконтролера ATmega8* у пакеті PROTEUS 8.6 описано у [4], де наведено схему алгоритму роботи моделі та робочу програми мовою С. Отримані результати моделювання підтвердили працездатність цього алгоритму та робочої програми.

Контрольні запитання та завдання

1. Дайте загальну характеристику інтерфейсу RS-485/EIA-485.
2. Наведіть та опишіть схему підключення інтерфейсу RS-485/EIA-485 до трьохпровідної лінії.
3. Яку кількість вузлів можна підключити в мережу RS-485/EIA-485?
4. Якою може бути швидкість обміну та відстань в мережі RS-485/EIA-485?
5. Наведіть та поясніть схему підключення інтерфейсів RS-485 до локальної мережі.
6. Назвіть загальні рекомендації щодо використання інтерфейсу RS-485.
7. Наведіть та поясніть схему реалізації ланцюга зсуву на сигнальних ланцюгах інтерфейсу RS-485.
8. Назвіть параметри інтерфейсу RS-485 для мікросхем фірми MAXIM.
9. Опишіть призначення виводів напівдуплексних мікросхем RS-485 фірми MAXIM.
10. Наведіть та опишіть схему типу точка-точка (напівдуплекс) з’єднання двох мікросхем RS-485.
11. Наведіть та опишіть схеми типового використання мікросхем RS-485 в напівдуплексній та дуплексній мережах.
12. Наведіть та опишіть схему підключення мікроконтролера AT89C51 до інтерфейсу RS-485.
13. Дайте загальну характеристику інтерфейсу RS-232.
14. Наведіть та опишіть формат даних інтерфейсу RS-232.
15. Наведіть та опишіть 9-контактний роз'єм RS-232.
16. Наведіть та опишіть схему робочої моделі мережі RS-485-232-1.
17. Наведіть та опишіть схему робочої моделі мережі RS-485-232-2.
18. Наведіть та опишіть схеми алгоритмів роботи та робочі програми моделей: RS-485-232-1 та RS-485-232-2.

19. Виконайте порівняння моделей RS-485-232-1 та RS-485-232-2.
20. Який режим обміну використовується в модулі SPI: асинхронний чи синхронний? Відповідь пояснити.
21. На яку відстань і з якою швидкістю можна здійснювати обмін даними через інтерфейс SPI?
22. Поясніть структурну схему модуля SPI.
23. В яких режимах може працювати МК під час обміну даними інтерфейсом SPI?
24. Скільки та які виводи МК використовує модуль SPI?
25. Як виконується перепризначення режиму роботи виводів модуля SPI?
26. Поясніть схему з'єднання двох МК інтерфейсом SPI.
27. Які пристрої знаходяться перед входами 8-розрядних РГ зсуву ведучого та веденого МК? Яку функцію вони виконують?
28. Який регістр призначено для керування модулем SPI?
29. Який регістр використовується для контролю стану модуля, а також для додаткового керування швидкістю обміну?
30. Як задати режим роботи МК у режимі «Master»?
31. В який спосіб здійснюється передача даних у модулі SPI?
32. Як визначити кінець передачі байта?
33. За якої умови під час передачі даних від ведучого до веденого можлива передача у зворотному напрямку?
34. Що означає те, що у модулі реалізовано одинарну буферизацію під час передачі та подвійну під час прийому?
35. Поясніть структуру SPI-мережі МК.
36. Як обирається потрібний ведений МК у SPI-мережі?
37. Поясніть вплив сигналу $\overline{SS}$ на початок обміну даними в різних режимах роботи модуля.
38. Чи можна до інтерфейсу підключати декілька периферійних пристроїв?
39. Назвіть кількість режимів передачі даних та принцип їх роботи.
40. Що є джерелом тактового сигналу під час роботи модуля? Як формується цей сигнал?
41. З яких етапів складається обмін у режимі SPI?
42. Поясніть схему включення МК у режимі програмування послідовним каналом під час використання інтерфейсу SPI.

ЛЕКЦІЯ 16. МІКРОКОНТРОЛЕРНА МЕРЕЖА НА ОСНОВІ ІНТЕРФЕЙСУ I²C. МІКРОКОНТРОЛЕРНА МЕРЕЖА 1-WIRE

1. МІКРОКОНТРОЛЕРНА МЕРЕЖА НА ОСНОВІ ІНТЕРФЕЙСУ I²C

1.1. Особливості архітектури інтерфейсу I²C

Інтерфейс I²C (InterIC, або IIC) є *двопровідним послідовним синхронним* інтерфейсом, який розроблено фірмою «Philips Corporation», і призначено для зв'язку між *інтегральними мікросхемами або модулями*. Є ціла група I²C-сумісних пристроїв для різних додатків: ЦАП, АЦП, мікросхеми пам'яті, давачі, мікроконтролери, що містять модуль I²C і т. ін. [2; 4]. Інтерфейс I²C в AVR-мікроконтролерах має назву TWI.

Шина інтерфейсу I²C *складається* із двох ліній: двонаправленої лінії даних (англ. SDA); лінії тактових синхроімпульсів (англ. SCL).

На рис. 1 наведено структурну схему типової мікроконтролерної мережі, що використовує для обміну даними інтерфейс (шину) I²C.

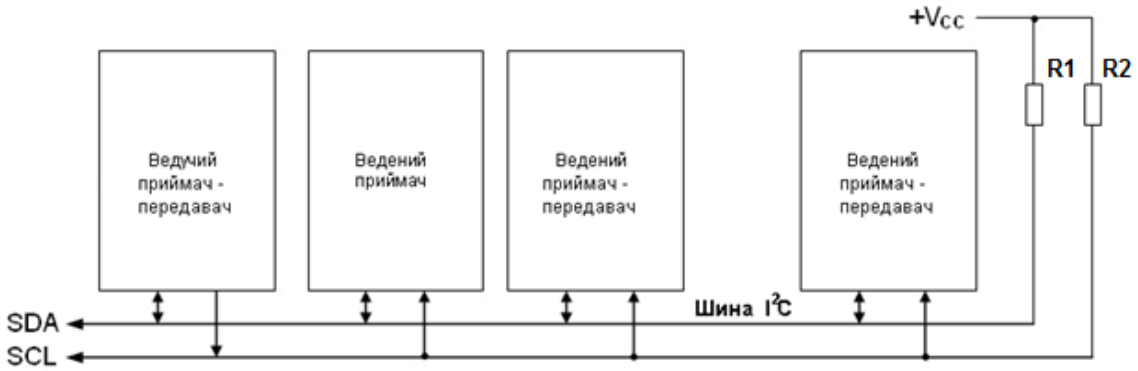


Рис. 1. Структурна схема мережі на основі інтерфейсу I²C

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 I²C, 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 напрямку $R/\overline{W}$, який визначає, чи буде ведучий *читати дані від веденого* ($R/\overline{W}=1$), чи буде *передавати дані веденому* ($R/\overline{W}=0$). *Приймач* (ПРМ) після одержання адреси або даних видає на шину SDA *біт підтвердження* (логічний нуль).

Інтерфейс I²C може використовувати *два формати* адреси: 7-бітну, або 10-бітну адресу.

В AVR-мікроконтролерах, наприклад, *сімейства Mega* використовується *7-бітна* адреса. На рис. 2 наведено *формат адреси*, на якому використано такі позначення: S – умова «СТАРТ»; $R/\overline{W}$ – біт «*читання* / $\overline{запис}$»; $\overline{ACK}$ – біт підтвердження.

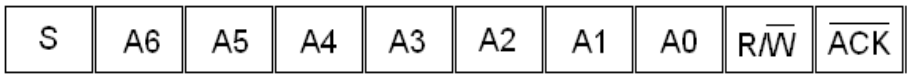
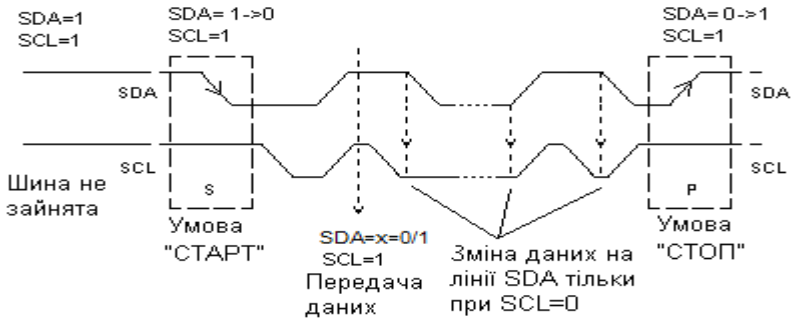
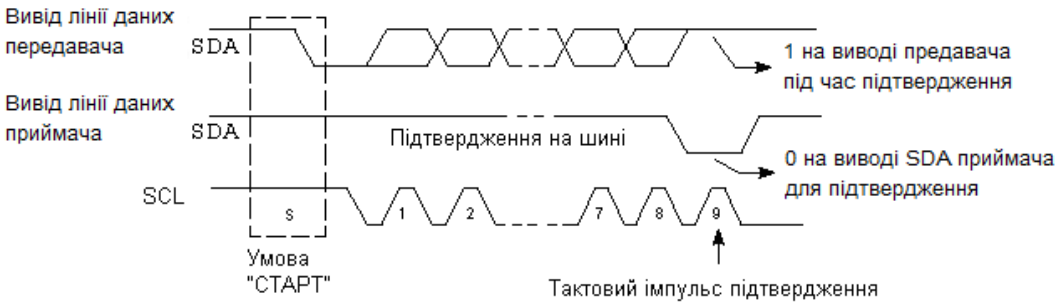


Рис. 2. Формат 7-бітної адреси

На рис. 3 наведено часові діаграми, які відображають стани на шині (рис. 3*а*), а також пояснюють формування сигналу підтвердження (рис. 3*б*).



а



б

Рис. 3. Часові діаграми: *а* – передача даних; *б* – формування сигналу підтвердження

Наведені діаграми відображають такі *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 (англ. SDA) з одиниці в нуль за одиничним значенням сигналу на лінії синхронізації (англ. SCL);

– припинення передачі (умова «СТОП» – Р): зміна сигналу на лінії даних з нуля в одиницю за одиничним значенням сигналу на лінії синхронізації;
– коректність даних: щоб не сформувати невірну умову «СТАРТ» або «СТОП» стан лінії даних не повинен змінюватися за одиничним значенням сигналу на лінії синхронізації. *Дані можна змінювати*, якщо на лінії синхронізації є 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 даних або адреси. *Активний передавач* (ПД) (ведений або ведучий) після передачі чергового байта формує на лінії SDA сигнал високого рівня. *Ведучий* пристрій (приймач (ПРМ) або ПД) формує на лінії SCL тактовий імпульс, а ПРМ (ведучий або ведений) видає на SDA *сигнал підтвердження* низького рівня. *Приймач,* що генерує біт підтвердження, підключає лінію SDA до *низького* рівня й *утримує* її в цьому стані *доти,* поки тактовий імпульс лінії SCL не переключиться у стан низького рівня. Для *припинення обміну* ПРМ повинен залишити останній прийнятий байт без підтвердження, що автоматично викликає формування активним ПД умови «СТОП».

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 свідчить 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 свідчить 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 та біт напрямку $R/\overline{W}=0$, що свідчить 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 дані на лінію SDA. *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 напрямку $R/\overline{W}=1$, що свідчить про те, що ведучий буде здійснювати прийом. Дані послідовно передаються лінією SDA *від веденого передавача* тоді, 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 у цьому разі 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 в головному* (ведучому) режимі, то відповідний пристрій I^2C переключається у підпорядкований режим і може розпізнавати свою власну підпорядковану адресу.

На рис. 4 наведено часові діаграми, які відображають обмін даними шиною I^2C.

1.2. Модуль I^2C мікроконтролерів AVR

1.2.1. Загальна характеристика модуля I^2C (TWI) мікроконтролерів AVR

Мікроконтролери AVR сімейства Mega мають модуль двопровідного синхронного послідовного інтерфейсу TWI (Two Wire (Serial) Interface). У деяких моделях функцію TWI може виконувати модуль USI [2; 4].

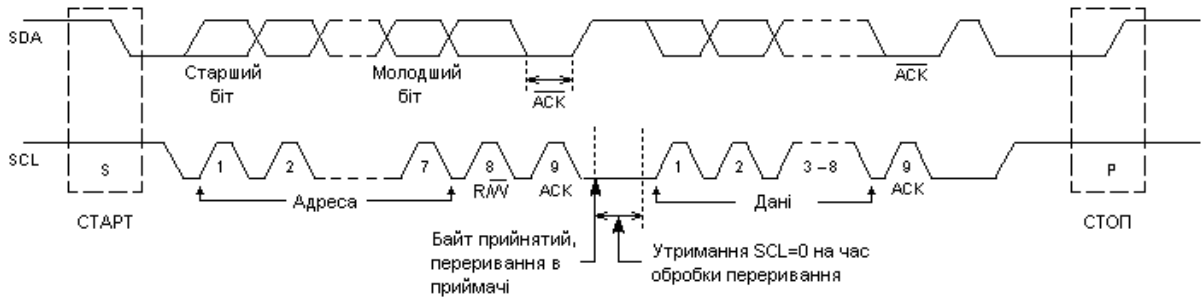


Рис. 4. Часові діаграми обміну даними шиною I²C

Інтерфейс TWI є повним *аналогом* базової версії інтерфейсу *I²C фірми «Philips»* (підрозд. 1.1). Він дозволяє об'єднати разом *до 128 різних пристроїв* за допомогою двонаправленої шини, яка складається із двох ліній: лінії SCL і лінії SDA. *Додатково* для реалізації шини використовуються *два підтягуючі резистори*: *R*1, *R*2, по одному на кожну лінію (рис. 5).

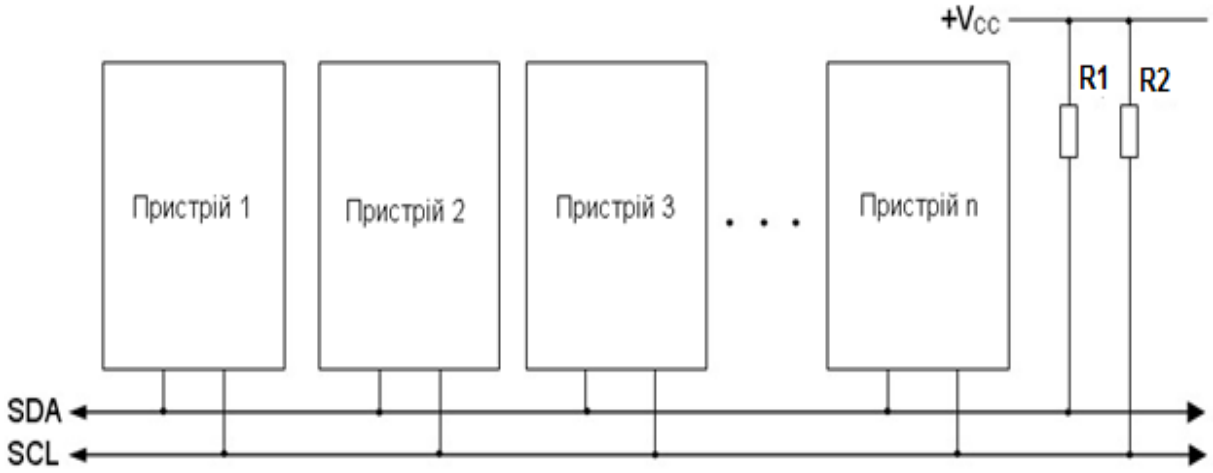


Рис. 5. Використання підтягуючих резисторів в мережі TWI

Шинні формувачі *на вихідних лініях* виконуються за схемою *з відкритим стоком,* що дозволяє реалізувати функцію «монтажне АБО/І» [5].

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 [2].

Швидкість передачі даних для основного режиму роботи дорівнює: *100 Кбіт/с.,* а для режиму із заниженою швидкістю – 10 Кбіт/с.

Протокол інтерфейсу TWI дозволяє підключати до шини кілька ведучих пристроїв *(режим Multi-Master*) [2; 4].

Під час 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*. Формати цих пакетів відповідають наведеним на рис. 4 та у [4].

На практиці кожен цикл обміну шиною TWI складається з таких етапів (рис. 4):

- формування стану «СТАРТ»;
- передача адресного пакета $SLA + R/\overline{W}$;
- передача одного або декількох пакетів даних;
- формування стану «СТОП».

Швидкість обміну шиною TWI *задається ведучим*, оскільки 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 (рис. 4). *На тривалість тактових* імпульсів це не впливає.

1.2.2. Керування модулем TWI

Керування модулем TWI здійснює блок керування (Control Unit) відповідно до значень *регістра керування TWCR* та інформації, яка надходить до нього від інших блоків модуля [2; 4]. У разі настання певних подій, зазначених нижче, блок керування формує в *регістрі 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 у разі виникнення таких 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. 6, а опис його розрядів наведено в табл. 1.

Формат регістра TWSR показано на рис. 7, а опис його розрядів наведено в табл. 2.

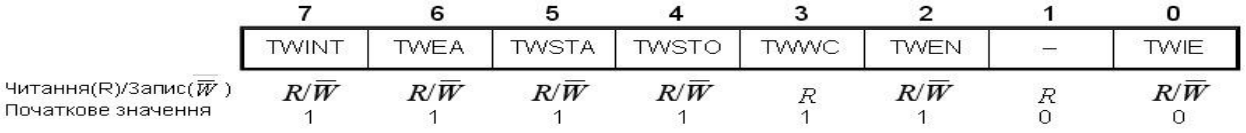


Рис. 6. Формат регістра TWCR

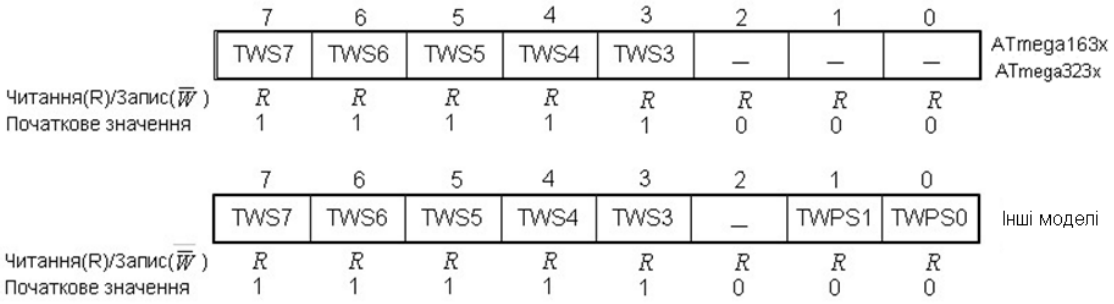


Рис.7. Формат регістра TWSR

1.2.3. Взаємодія прикладної програми з модулем TWI

На рис. 8 показано взаємодію прикладної програми з модулем TWI на прикладі передачі одного байта даних від ведучого до веденого.

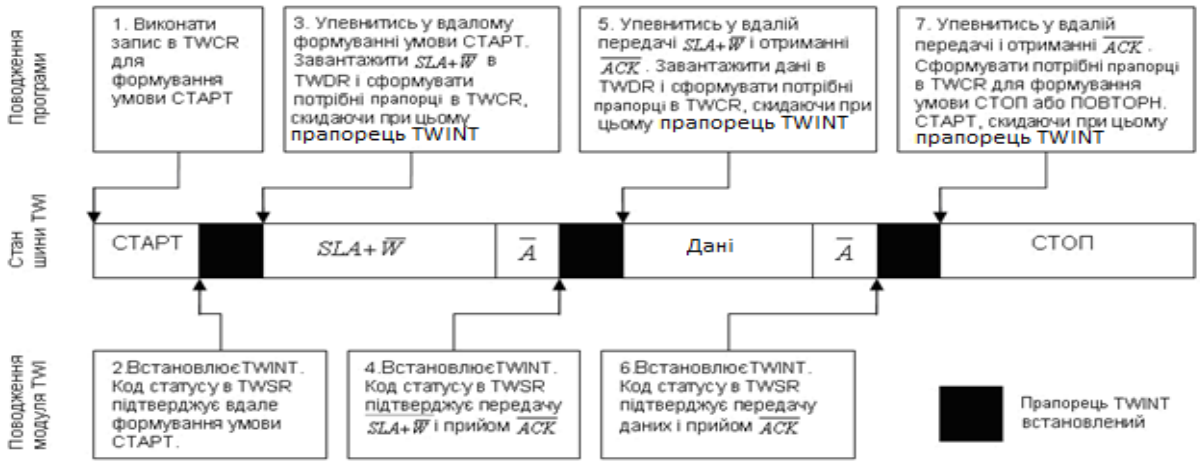


Рис. 8. Приклад взаємодії програми з модулем TWI

Взаємодія прикладної програми з модулем TWI ґрунтується 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 яких-небудь 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.

Таблиця 1. 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. У разі скидання розряду в нуль пристрій *віртуально* відключається від шини TWI, тобто біт підтвердження не формується
5	TWSTA	**Прапорець стану «СТАРТ»** Під час запису 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 неадресованого 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 у разі спроби запису в регістр даних TWDR, коли прапорець переривання TWINT скинуто. Прапорець скидається під час запису в регістр TWDR, коли прапорець переривання TWINT встановлено
2	TWEN	**Дозвіл роботи модуля TWI** Розряд TWEN керує функціонуванням модуля TWI. Під час запису в цей розряд одиниці модуль TWI включається і бере на себе керування контактами ВВ/ВИВ мікроконтролера, які відповідають виводам SCL і SDA. У разі скидання 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

Таблиця 2. Опис розрядів РС 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 описано під час подальшого розгляду 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 [2; 4].

Встановлення прапорця TWINT регістра TWCR означає, що *модуль TWI закінчив* виконання чергової операції та *очікує реакції програми*, у цьому разі *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. Передача даних відбувається у послідовності, яку на рис. 8 відмічено номерами.

Взаємодія прикладної програми з модулем TWI складається із таких трьох частин.

1. Після завершення модулем виконання чергової операції, він встановлює у регістрі TWCR прапорець TWINT, записує у регістр TWSR код статусу й 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. У разі завантаження в регістр нового значення необхідно *скинути прапорець* TWINT *записом* у нього *одиниці*. Після скидання прапорця модуль почне виконання операції, обумовленої вмістом регістра TWCR.

У [2; 4] *для режимів роботи мережі*: ведучий передавач (Master Transmitter); ведучий приймач (Master Receiver); ведений передавач (Slave Transmitter) та ведений приймач (Slave Receiver) наведено: керувальні слова, які треба записувати в регістр TWCR для програмування цього режиму; коди статусу в регістрі TWSR; часові діаграми роботи; схему алгоритму роботи модуля та його програмну реалізацію.

2. МЕРЕЖА 1-WIRE

2.1. Особливості архітектури мережі 1-WIRE

Мережі 1-WIRE проектуються на основі однойменного інтерфейсу, який розроблений фірмою Dallas Semiconductor. Інтерфейс був рекомендований розробниками для застосування в багатьох сферах, однією з яких є

Як середовище для передачі інформації за *однопровідною* система автоматизації (технологія1-WIRE-мереж) [2; 4; 7] *лінією* найчастіше використовується звичайний *телефонний кабель* і, як наслідок, швидкість обміну в цьому випадку невелика. Однак, якщо ретельно проаналізувати об'єкти, які потребують автоматизації, то для більшості з них *гранична швидкість* обслуговування у *16,4 Кбіт/с є достатньою*.

Переваги 1-WIRE-технології: просте вирішення адресування; нескладний протокол; проста структура лінії зв'язку; проста зміна конфігурації мережі; дешевизна всієї технології в цілому показують раціональність та високу ефективність цього інструмента *під час вирішення задач комплексної автоматизації в різних областях діяльності.*

2.2. Основні характеристики інтерфейсу та мережі 1-WIRE

Мережа 1-WIRE є мережею з *централізованим керуванням* та *шинною топологією.* Для здійснення зв'язку використовується *одна лінія даних (DATA*) та *один зворотний провід*. Таким чином, для реалізації середовища обміну цієї мережі можуть бути застосовані доступні кабелі, які мають *неекрановану виту пару* будь-якої категорії, а також, наприклад, *звичайний телефонний провід*. Такі кабелі під час їх прокладання не потребують наявності спеціального обладнання, а обмеження *максимальної довжини* однопровідної лінії регламентовано розробниками на рівні *300 м*.

Основою архітектури 1-WIRE-мереж є *топологія спільної шини*, коли кожен з пристроїв підключено безпосередньо до єдиної магістралі без каскадних з'єднань або розгалужень, при цьому як базова використовується *структура мережі з одним ведучим та багатьма веденими.*

Конфігурація будь-якої 1-WIRE-мережі *може змінюватися* в процесі її роботи. Ця можливість досягається завдяки *наявності в протоколі* 1-WIRE-інтерфейсу спеціальної *команди пошуку ведених* пристроїв, яка дозволяє швидко визначити нових учасників інформаційного обміну. *Стандартна швидкість* виконання такої команди дозволяє обробити приблизно *75 вузлів мережі за секунду*.

Кожен пристрій, який має 1-WIRE-інтерфейс, має *унікальну* індивідуальну адресу. Така мережа має *практично необмежений* адресний простір. Кожен з

однопровідних приладів одразу *готовий до використання* в 1-WIRE-мережі без будь-яких додаткових апаратно-програмних модифікацій.

Передача сигналів для 1-WIRE-інтерфейсу *асинхронна* та *напівдуплексна*, а вся інформація, що циркулює в мережі, сприймається абонентами або як *команди,* або як *дані*. *Команди* мережі *генеруються ведучим* та забезпечують різні варіанти пошуку та адресації ведених пристроїв, визначають активність на лінії окремих компонентів, керують обміном даними в мережі і т. ін.

Стандартна *швидкість* роботи 1-WIRE-мережі складає *16,4 Кбіт/с*.

Під час реалізації однопровідного інтерфейсу *використовуються стандартні* КМОН/ТТЛШ логічні рівні сигналів. *Живлення* пристроїв може здійснюватися *від зовнішнього джерела* з робочою напругою в діапазоні від 2,8 до 6,0 В, або застосуванням *механізум «паразитного живлення*», дія якого полягає у використанні кожним з ведених компонентів 1-WIRE-лінії електричної енергії імпульсів, які передаються шиною даних [2; 7].

Фізична *реалізація* інтерфейсу 1-WIRE достатньо *проста*. Шина 1-WIRE має бути *підтягнута окремим резистором* R1 до напруги живлення пристроїв. *Передача* будь-якої інформації можлива тільки *видачею низького рівня в лінію* – замиканням її на спільний провід, а у високий логічний рівень лінія повернеться сама завдяки наявності зовнішнього резистора, який його підтягує.

Обмін інформацією ведеться *тайм-слотами*: один тайм-слот служить для обміну одним бітом інформації; дані передаються побайтно, біт за бітом, починаючи з молодшого біта. *Достовірність* переданих/прийнятих даних (перевірка відсутності спотворень) *гарантується* за допомогою підрахунку циклічної контрольної суми [2; 7].

2.3. Структура мережі 1-WIRE

На рис. 9 наведено структурну схему 1-WIRE-мережі в системі *вимірювання температури.*

Система призначена для вимірювання температури в приміщенні, де необхідно вимірювати температуру в декількох точках. Система є набором датчиків температури, які можуть бути розміщено *на відстані до 300 метрів*.

Система *виконує індикацію* температури в самому приміщенні (виведення на LCD-дисплей) і *передачу даних через модем* на віддалений пункт обробки та керування.

У цій системі як ведучий в мережі 1-WIRE використовується, наприклад, МК ATmega8535 – 8-розрядний КМОН-мікроконтролер, заснований на розширеній AVR-RISC-архітектурі.

Завдяки виконанню більшості інструкцій за один машинний цикл ATmega8535 досягає продуктивності 1 млн операцій за секунду, що дозволяє

проектувальникам систем оптимізувати співвідношення енергоспоживання і швидкодії [2].

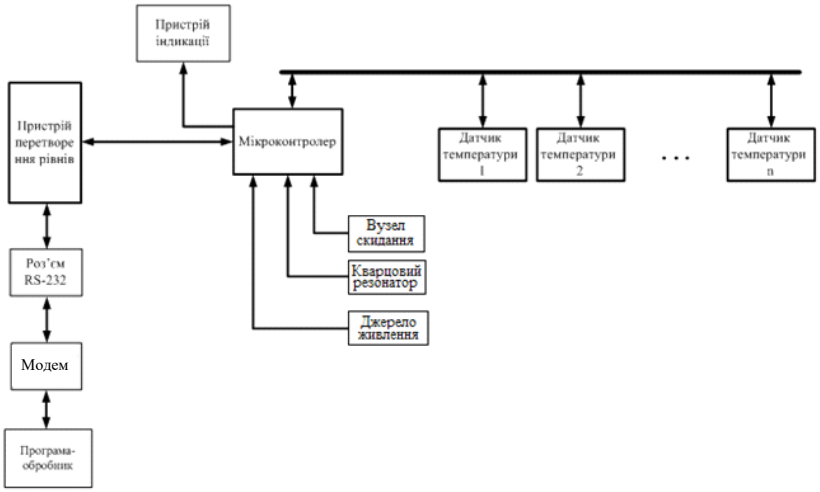


Рис. 9. Структурна схема 1-WIRE-мережі в системі вимірювання температури

2.4.Загальна характеристика датчика температури

В якості ведених пристроїв 1-WIRE-мережі (рис. 9) можуть бути датчики температури, наприклад, DS18B20 з програмованим значенням *розрядності* вихідного коду від 9 до 12 біт, яку можна зберігати в EEPROM-пам'яті приладу [2; 4]. Датчик DS18B20 обмінюється даними 1-WIRE-шиною і при цьому може бути як єдиним пристроєм на лінії, так і працювати в групі. Всіма процесами на шині керує центральний МК. Датчик має 3 виводи та може живитися напругою лінії даних («паразитне живлення») за відсутності зовнішнього джерела напруги.

Кожен датчик DS18B20 *має унікальний 64-бітовий послідовний двійковий код* адреси, який дозволяє спілкуватися з багатьма датчиками DS18B20, які встановлено на одній шині. Такий принцип дозволяє використовувати один МК, щоб контролювати багато датчиків DS18B20, розміщених на деякій відстані. Вигоду з цієї особливості мають системи контролю температури в будівлях, устаткуванні чи машинах.

На рис. 10 наведено структурну схему датчика DS18B20.

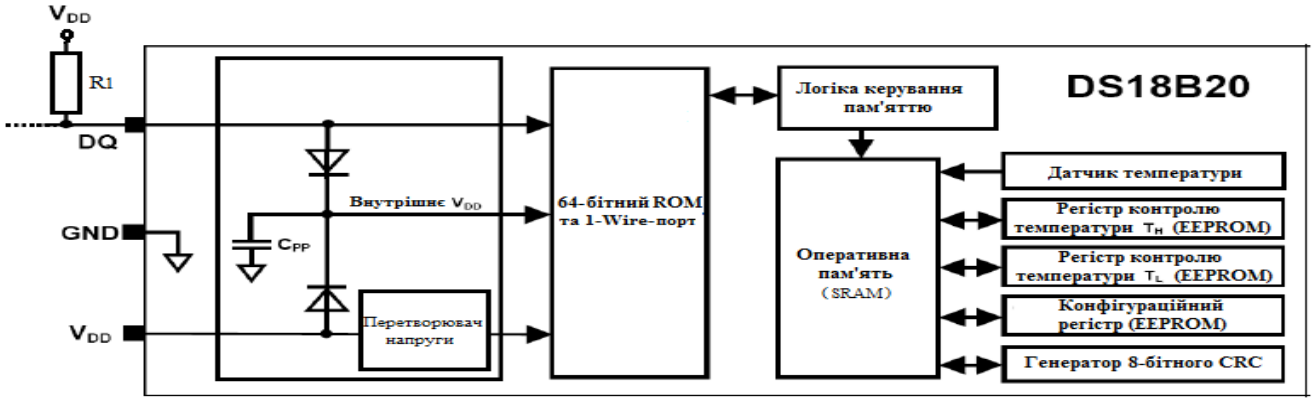


Рис. 10. Структурна схема датчика DS18B20

Датчик має *64-бітовий постійний запам'ятовуючий пристрій* (англ. ROM), який зберігає унікальний двійковий код приладу. *Оперативна пам'ять* (SRAM) містить *двобайтовий* регістр температури, який зберігає значення температури по закінченню процесу її вимірювання.

У пам'яті EEPROM є два однобайтових *регістра контролю* температури: TH і TL та *конфігураційний* регістр. Цей регістр дозволяє користувачеві встановлювати розрядність цифрового перетворювача температури: 9, 10, 11 або 12 біт, що впливає на час конвертації температури.

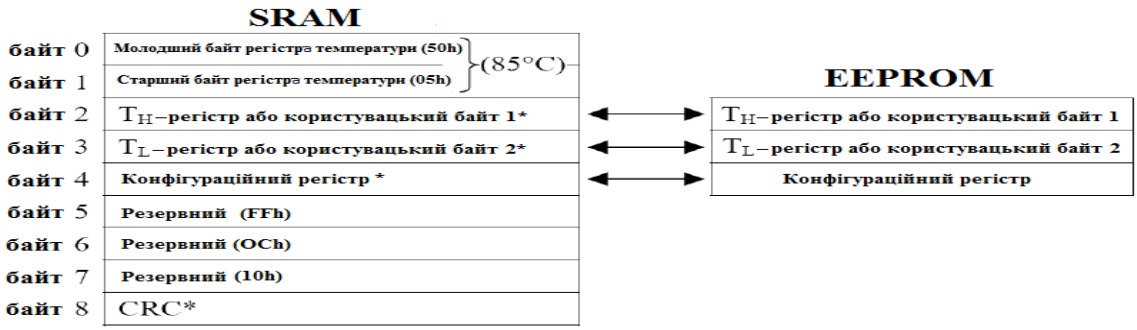
Оскільки регістри TH, TL і регістр конфігурації є комірками енергонезалежної пам'яті даних EEPROM, то вони зберігають дані, коли прилад вимкнено.

Ведучий МК *ідентифікує і звертається* до температури, використовуючи 64-бітовий код приладу. Оскільки кожен прилад має унікальний код, *кількість* приладів, до яких можна звернутися на одній шині, *фактично необмежено*.

2.5. Організація пам'яті датчика температури

2.5.1.Загальна характеристика пам'яті датчика

Організацію пам'яті датчика DS18B20 показано на рис. 11.



Примітка. * *– Стан після включення* живлення залежить від значень, збережених в EEPROM

Рис. 11. Організація пам'яті датчика DS18B20

2.5.2. Регістр температури

Молодший та старший байти регістра температури *зберігають поточну* вимірювану температуру. Під час подачі живлення *початкове значення* температури становить: +85 градусів. Формат регістра подано на рис. 12, де S – знак температури (0 – додатна температура, 1 – від'ємна), біти від 11-го до 4-го – ціла частина значення температури, біти від 3-го до 0-го – дробова частина.

	біт 7	біт 6	біт 5	біт 4	біт 3	біт 2	біт 1	біт 0
Молодший байт	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
	біт 15	біт 14	біт 13	біт 12	біт 11	біт 10	біт 9	біт 8
Старший байт	S	S	S	S	2^7	2^6	2^5	2^4

*Рис. 12.*Формат регістра температури

2.5.3. Формування стану «Аварія»

Після того, як датчик DS18B20 виконає температурне перетворення, *поточне* температурне значення *порівнюється* зі значеннями, які записано в регістрах TH і TL. На рис. 13 наведено формат цих регістрів.

біт 7	біт 6	біт 5	біт 4	біт 3	біт 2	біт 1	біт 0
S	2^6	2^5	2^5	2^5	2^2	2^1	2^0

Рис. 13. Формат регістрів TH та TL

Біт знака S вказує на додатне чи від'ємне значення температури: для додатних чисел S = 0, а для від'ємних – S = 1. Регістри TH і TL є комірками енергонезалежної пам'яті даних EEPROM. Таким чином, вони зберігають дані, коли пристрій знеструмлено. До TH і TL можна звернутися через другий та третій байти SRAM.

Для порівняння використовуються тільки біти 4…11 регістра температури (*ціле значення* температури).

Якщо вимірювана температура нижче або дорівнює TL або вище або дорівнює TH, *формується стан* «Аварія» і встановлюється прапорець «Аварія». Цей прапорець оновлюється після кожного наступного температурного перетворення. Якщо у цьому разі стан «Аварія» зникне, то прапорець «Аварія» буде скинуто.

Ведучий пристрій може перевірити стан «Аварія» всіх датчиків DS18B20 на шині, подаючи команду «Пошук Аварії» [ECh]. Будь-який датчик DS18B20 з встановленим прапорцем «Аварія» відповість на цю команду. Таким чином, ведучий пристрій точно може визначити, які датчики DS18B20 перебувають у стані «Аварія». Якщо змінюються значення регістрів TH або TL, то необхідно запустити нове температурне перетворення, щоб виконалась перевірка умови контролю температури, яку задано в регістрах TH або TL.

2.5.4. Конфігураційний регістр

Четвертий байт пам'яті SRAM – є конфігураційним регістром. У ньому задається *температурна розрядність* датчика DS18B20. Початкове значення регістра залежить від вмісту EEPROM-пам'яті. Формат конфігураційного регістра подано на рис. 14, де R1, R0 – біти, що задають *температурну розрядність* термометра, від якої залежить максимальний час перетворення температури (табл. 3).

біт 7	біт 6	біт 5	біт 4	біт 3	біт 2	біт 1	біт 0
0	R1	R0	1	1	1	1	1

Рис. 14. Формат конфігураційного регістра

Таблиця 3. Залежність температурної розрядності та максимального часу перетворення від значень бітів R1, R0

R1	R0	Розрядність	Максимальний час перетворення
0	0	9 біт	93,75 мс
0	1	10 біт	187,5 мс
1	0	11 біт	375 мс
1	1	12 біт	750 мс

2.5.5. Контрольна сума циклічного надлишкового коду

Контрольна сума (CRC) – *це байт*, який зберігається в пам'яті SRAM останнім (*восьмим*). Він обчислюється за спеціальним алгоритмом на основі значення всіх *семи попередніх* байтів [2; 7].

Алгоритм підрахунку такий, що якщо всі байти передані-прийняті без спотворень, прийнятий байт контрольної суми обов'язково збігається зі значенням, розрахованим у мікроконтролері, або пристрої. *Алгоритм підрахунку* CRC має бути *однаковим* як для МК, так і для будь-якого пристрою.

2.5.6. Формат коду датчика

Кожен датчик DS18B20 має *унікальний 64-бітовий* код, збережений в ROM-пам'яті. Формат коду подано на рис. 15.

Рис. 15. Формат коду датчика

Примітка. LSB – Least Significant Bit (молодший значущий розряд); MSB – Most Significant Bit (старший значущий розряд).

Молодші 8 біт коду містять *код сімейства* 1-WIRE. Для датчика DS18B20 це 28h. Наступні 48 біт містять *унікальний серійний* номер датчика. Старші 8 біт містять CRC-байт, який обчислено *від перших 56 біт* коду ROM-пам'яті. Саме завдяки *унікальності коду* забезпечується адресація пристроїв 1-WIRE.

2.6. Команди мікроконтролера

2.6.1. Загальна характеристика команд

Як було зазначено вище, кожен пристрій 1-WIRE має *унікальний ідентифікаційний* 64-бітовий номер, який було запрограмовано на етапі

виробництва мікросхеми. *Унікальний* – це означає, що фірма-виробник гарантує, що не знайдеться двох мікросхем з однаковим ідентифікаційним номером (принаймні протягом кількох десятків років за наявних темпів виробництва).

Під час розгляду протоколу обміну будемо вважати, що на шині 1-WIRE є більше одного пристрою. У цьому випадку перед МК буде *дві задачі*: визначення кількості наявних пристроїв і вибір (адресація) одного конкретного з них для обміну даними.

Команди МК поділяються *на дві групи*:

- ROM-команди – команди для роботи з адресами пристроїв (пошук адреси, зчитування адреси, вибір адреси тощо);
- функціональні команди – команди, наявність яких вимагає виконання відповідних дій з боку кінцевого пристрою.

Послідовність операцій доступу до датчика DS18B20 така:

- ініціалізація;
- подача ROM-команди;
- подача функціональної команди DS18B20.

2.6.2. ROM-команди

У табл. 4 наведено деякі з важливих загальних ROM-команд.

Опис виконання цих команд наведено у [7].

Таблиця 4. ROM-команди пристроїв 1-WIRE

Команда	Значення байта	Опис
SEARCH ROM	0xF0	Пошук адрес – використовується під час універсального алгоритму визначення кількості та адрес підключених пристроїв
READ ROM	0x33	Зчитування адреси пристрою – використовується для визначення адреси єдиного пристрою на шині
MATCH ROM	0x55	Вибір адреси – використовується для звернення до конкретної адреси пристрою з багатьох підключених
SKIP ROM	0xCC	Ігнорування адреси – використовується для звернення до всіх або єдиного пристрою на шині, у цьому разі адреса пристрою ігнорується (можна звертатися до невідомого пристрою)

2.6.3. Функціональні команди

Функціональні команди дозволяють мікроконтролеру *читати і записувати* інформацію в оперативну пам’ять датчика DS18B20, *запускати* температурне *перетворення* датчика, *визначати* його режим живлення. Опис виконання цих команд наведено у [7].

2.7. Схема алгоритму роботи системи

Схему алгоритму роботи системи та програму, яка реалізує цей алгоритм мовою С, наведено у [7]. *До схеми алгоритму* належать: алгоритм ініціалізації МК, що міститься у системі; алгоритм ініціалізації LCD-пристрою; алгоритм ініціалізації УАПП; алгоритм роботи головної підпрограми, що виконується під час роботи системи; алгоритм пошуку пристроїв на шині; алгоритм пошуку адреси пристрою; алгоритм процедури отримання значень температури від датчиків; алгоритми перевірки CRC та розрахунку контрольної суми.

2.8. Моделювання мережі 1-WIRE

2.8.1 Схема моделі

Для моделювання мережі 1-WIRE використано пакет програмного забезпечення Proteus 8.6. Схему моделі наведено на рис. 16.

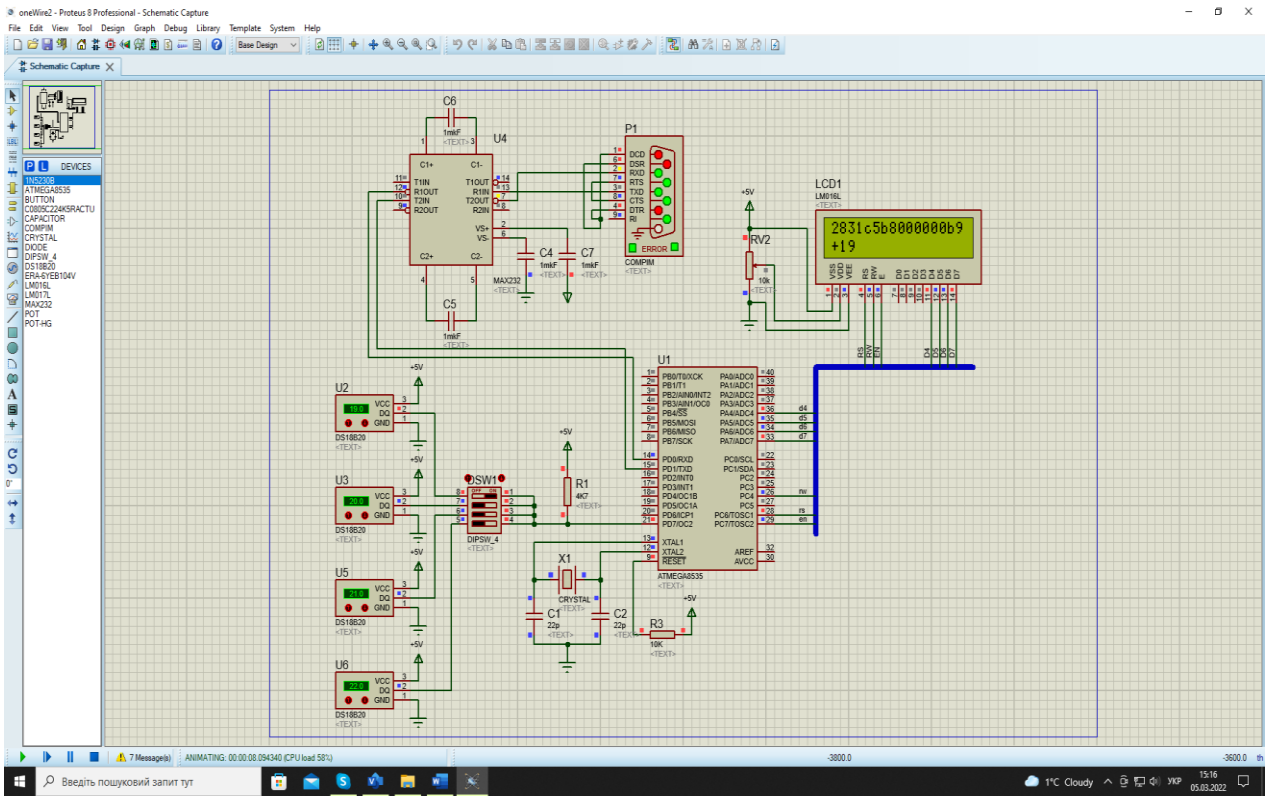


Рис. 16. Стан моделі під час підключення одного давача, який вимірює температуру

2.8.2. Опис роботи моделі

Датчики температури підключено до МК за допомогою опору R1. Кнопки імітують динамічне підключення датчиків до шини.

На старті роботи системи контролер опитує шину на наявність підключених датчиків. Якщо підключено хоча б один датчик, контролер починає опитувати датчики та виводити значення температури від кожного з них на LCD-дисплей.

З виходу послідовного порту МК інформація подається на перетворювач рівнів MAX232, а далі надходить на роз'єм RS-232. Потім, наприклад, через модем інформація подається на віддалений пункт керування, де виконується подальша обробка значень температури програмою високого рівня.

Для зміни в моделі температури на кожному з датчиків, використовуються кнопки під дисплеєм встановлення температури (рис. 17).

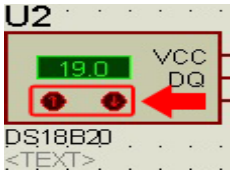


Рис. 17. Кнопки встановлення температури

Для режиму роботи із «паразитним живленням» необхідно VCC з'єднати зі спільним проводом (рис. 18).

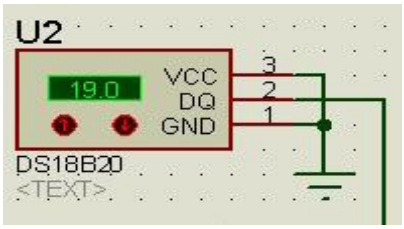


Рис. 18. Приклад датчика з «паразитним живленням»

Оскільки на схемі присутні кілька датчиків, то їх підключення або відключення здійснюється через «світч», який має 2 положення – «ON» та «OFF» відповідно. На рис. 19 наведено «світч», де всі 4 датчики підключено.

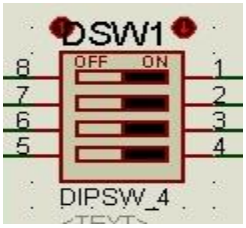


Рис. 19. Приклад «світча» з увімкнутими датчиками

Контрольні запитання та завдання

1. Якою фірмою 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 мають бути підключені 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 чотири можливі режими обміну даними інтерфейсом I^2C (TWI).
7. Поясніть призначення станів «СТАРТ» і «СТОП» шини TWI. Яким чином вони формуються?
8. У чому полягає задача розподілу пріоритетів під час підключення до шини TWI декількох ведучих пристроїв? Як вона вирішується?
9. Опишіть формат адресного пакету та пакету даних.
10. Наведіть та поясніть структурну схему типового модуля TWI.
11. У разі виникнення яких подій блок керування здійснює формування запиту на переривання?
12. Назвіть та опишіть використання регістрів керування TWI-модулем.
13. На чому ґрунтується взаємодія прикладної програми з модулем TWI?
14. З яких трьох частин складається будь-який етап взаємодії прикладної програми з модулем TWI? Поясніть цю взаємодію.
15. Назвіть деякі пристрої, які підтримують інтерфейс TWI.
16. Опишіть роботу модуля TWI у режимі «Ведучий передавач».
17. Опишіть роботу модуля TWI у режимі «Ведучий приймач».
18. Опишіть роботу модуля TWI у режимі «Ведений приймач».
19. Опишіть роботу модуля TWI у режимі«Ведений передавач».
20. Які можливості надає використання стану «ПОВСТАРТ»?
21. З якою швидкістю може виконуватися обмін інформацією в I^2C (TWI)-мережі?
22. На яку відстань може передаватися інформація в I^2C (TWI)-мережі?
23. Скільки ведучих пристроїв може одночасно бути в I^2C (TWI)-мережі?
24. Опишіть схему моделювання інтерфейсу TWI.
25. Опишіть схему алгоритму роботи моделі.
26. Опишіть схему алгоритмів запису/читання в/з EEPROM-пам’яті.
27. На яку відстань можуть передаватися дані в мережах 1-WIRE?
28. Скільки ведених пристроїв може бути в мережі 1-WIRE?
29. Назвіть основні застосування інтерфейсу 1-WIRE.
30. Як ведеться передача сигналів у мережі 1-WIRE?
31. Опишіть, як підключається інтерфейс 1-WIRE в мережу?
32. Опишіть основні правила передачі даних в мережі 1-WIRE.
33. Опишіть обмеження застосування 1-WIRE-мереж.
34. Наведіть та опишіть структурну схему системи вимірювання температури з використанням давачів 1-WIRE.
35. Дайте загальну характеристику датчикам температури DS18B20.

36. Поясніть використання унікального 64-бітового послідовного двійкового коду адреси датчика DS18B20.
37. Наведіть та опишіть структурну схему датчика DS18B20.
38. Опишіть організацію пам’яті датчика DS18B20.
39. В якому діапазоні може вимірювати температуру датчик DS18B20?
40. Як формується контрольна сума циклічного надлишкового коду?
41. Наведіть та поясніть формат коду датчика DS18B20.
42. На які групи поділяються команди датчика DS18B20?
43. Наведіть та опишіть загальні ROM-команди.
44. Опишіть послідовність пошуку пристроїв у мережі 1-WIRE.
45. Наведіть та опишіть функціональні команди.
46. Опишіть послідовність операцій для доступу до датчика DS18B20.
47. Наведіть та опишіть схему моделювання мережі 1-WIRE.

ЛЕКЦІЯ 17. CAN-МЕРЕЖІ

1. CAN-МЕРЕЖІ

1.1. Загальні відомості про CAN-мережі

CAN (англ. Controller Area Network) – мережа контролерів є *асинхронним послідовним* протоколом високошвидкісної та високонадійної передачі даних. CAN-протокол було розроблено фірмою “Bosch” для систем керування вузлами автомобіля [4; 8].

Логіка CAN-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 (англ. Non Return to Zero – без повернення до нуля) – це найпростіший двійковий код, що передається звичайним цифровим сигналом. Логічному нулю відповідає низький рівень напруги в кабелі, логічній одиниці – високий 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 належить його досить проста реалізація (вихідний сигнал не треба спеціально кодувати на передавальному кінці та декодувати на приймальному кінці). Порівняно з іншими кодами потрібна мінімальна пропускна здатність лінії зв’язку.

Для синхронізації даних всіма вузлами використовується додатковий наповнюючий біт (*виконується біт-стаффінг*). Під час послідовної 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 МПС керування показано на рис. 1.

Кожне повідомлення, яке передається мережею, має *оригінальний 11-бітний або 29-бітний ідентифікатор.*

У разі групового доступу 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 керувальним для шини, а всі інші вузли переходять у режим прийому.

Рис 1. Мікропроцесорна система керування

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 у 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*. У цьому разі 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*.

У версії CAN-2.0B довжина ідентифікатора може бути 11 або 29 біт (536 мільйонів варіантів). Вона дозволяє активним вузлам додатково робити *віддалений стандартний або розширений запит* та отримувати дані від потрібного пристрою, використовуючи відповідно стандартне або розширене повідомлення.

1.2. Основні характеристики CAN-протоколу

Коли шина вільна, будь-який підключений до неї вузол може почати передавати нове повідомлення.

У CAN-мережах *немає* ніякої інформації щодо конфігурації системи (наприклад, *адреси вузла*). Вузли *можуть бути додані до CAN-мережі* без зміни програмного забезпечення та апаратних засобів будь-якого вузла.

Зміст повідомлення визначається *ідентифікатором*. Ідентифікатор не вказує адресата повідомлення, але *описує значення дан*их так, щоб всі вузли в мережі мали здатність фільтрацією вирішити, чи повинні вони реагувати на це повідомлення чи ні. Будь-яка кількість 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.

Посилаючи кадр *віддаленого запиту даних*, вузол може виконувати запит даних від 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.

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 такі 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.

Помилки передач виявляються з досить високою ймовірністю за допомогою вбудованого *механізму виявлення помилок* CAN-протоколу.

Пошкоджені кадри позначаються кожним вузлом, що виявив помилку. Такі кадри перериваються та автоматично передаються заново. Дефектні вузли від'єднуються від шини CAN-модуля.

Шина складається з каналу обміну, що передає біти. В якості цього каналу можуть бути *два диференціальних* дроти, оптичне скловолокно і т. ін.

Шина може мати одне із двох логічних значень: «домінантне» або «рецесивне». Вважається, що «домінантний» рівень еквівалентний *нульовому* рівню, а «рецесивний» рівень еквівалентний *одиничному* рівню.

У разі одночасної передачі «домінантного» та «рецесивного» бітів значення, що виникає на шині, буде «домінантне».

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.

Для забезпечення достатньо високої швидкості обміну CAN-шиною потрібен *кварцовий* резонатор.

1.3. Структура повідомлень CAN-мережі

1.3.1. Загальна характеристика повідомлень

Повідомлення, що передаються CAN-шиною, називаються *кадрами* (*фреймами*). *Є чотири типи кадрів* [8]:

1. Data Frame – кадр даних.

2. Remote Frame – кадр запиту даних.
3. Error Frame – кадр помилки.
4. Overload Frame – кадр перевантаження.

Є *два формати даних* (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 у разі виявлення 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.

1.3.2. 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) [8]. *Поле даних може мати* нульову довжину.

Кількість байт у полі даних стандартного та розширеного форматів визначається кодом довжини даних – Data length code. Цей код має довжину *4 біти* та передається всередині поля керування (табл. 1).

Таблиця 1. Кодування довжини даних

Кількість 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»				

Допустиме *кількість байт* даних, що передаються, починаючи зі старшого значущого розряду: *0; 1; ... 8*. Інші величини використовуватися не можуть.

Поле CRC стандартного та розширеного форматів (CRC field) містить *послідовність CRC та CRC-роздільник*.

Під час обчислення послідовності контролю кадру (CRC Sequence) береться початковий поліном, коефіцієнти якого задано послідовністю біт, які *складаються з полів:* «початок кадру», «поле арбітражу», «керувальне 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* CRC розрахунку п'ятнадцяти бітів поля CRC для кадру даних, в якому передається повідомлення з ідентифікатором – 00000000001 з одним байтом даних, – 00001111 наведено *у [8]*. Після розрахунку поле CRC = 101101111010110.

Послідовність CRC супроводжується *роздільником CRC* (CRC-Delimiter), що складається з одного *одиничного біта*.

Поле підтвердження (ACK-field) має довжину *два біти* й містить: перший біт – «Область підтвердження» і другий біт – роздільник підтвердження. У поле підтвердження передавальний вузол 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».*

Роздільник підтвердження (ACK-delimiter) – другий біт поля підтвердження, який має *одиничний* рівень.

Кожен кадр даних і кадр віддаленого запиту даних *закінчується* послідовністю прапорців, яка складається із *семи одиничних* біт (Endof Frame).

1.3.3. Кадр віддаленого запиту даних

Вузол, який виконує прийом даних, *може запросити* передачу відповідних даних від потрібних вузлів, посилаючи *кадр віддаленого запиту* даних (Remote Frame) [8].

Кадр віддаленого запиту даних *є у стандартному та розширеному* форматах. В обох випадках він складається із *шести бітових п*олів: «початок кадру» (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.

1.3.4. Кадр помилки

Кадр *помилки* (Error frame) складається із *двох різних полів*. Перше поле є суперпозицією *прапорців помилки*, отриманих від вузла, який передає кадр помилки, та інших вузлів. Є два види прапорця помилки: прапорець активної помилки та прапорець пасивної помилки [8]. Наступне поле – *роздільник* помилки (Error Delimiter).

1.3.5. 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. *Основним з них є* внутрішній стан приймача, який потребує затримки наступного кадру даних або кадру віддаленого запиту даних

У табл. 2 показано вплив бітів RTR, SRR та IDE на тип повідомлення, що передається.

Таблиця 2. Вплив бітів RTR, SRR та IDE на тип повідомлення, що передається

№ з/п	Значення біта			Тип повідомлення
	RTR у стандартному повідомленні (SRR у розширеному повідомленні)	IDE	RTR у 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

1.4. CAN-модуль AVR-мікроконтролера

1.4.1. Загальні відомості про CAN-модуль

У сімействі AVR є, наприклад, МК AT90CAN128, який має CAN-модуль. Він апаратно підтримує протоколи CAN-2.0A та CAN-2.0B.

Нижче наведено *основні характеристики* цього мікроконтролера:

- стандартний та 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 у SLEEP-режимі.

1.4.2. Структура CAN-модуля

Структуру CAN-модуля МК AT90CAN128 наведено на рис. 2.

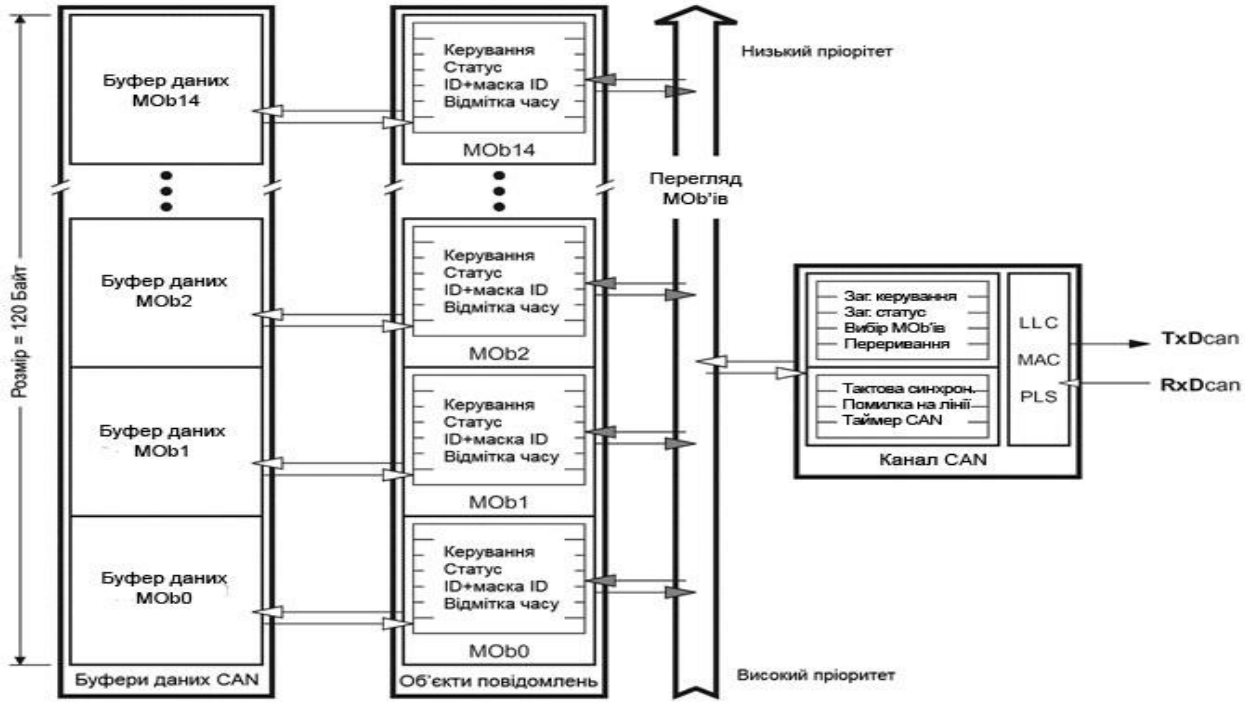


Рис. 2. Структура CAN-модуля МК AT90CAN128

Мікроконтролер *забезпечує* повну апаратну підтримку *фільтрації* повідомлень та *керування* ними. *Для кожного повідомлення*, яке має бути передано або отримано, CAN-модуль містить *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.

1.4.3. Організація керувальних регістрів

Склад та організацію керувальних регістрів CAN-модуля наведено на рис. 3. *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.

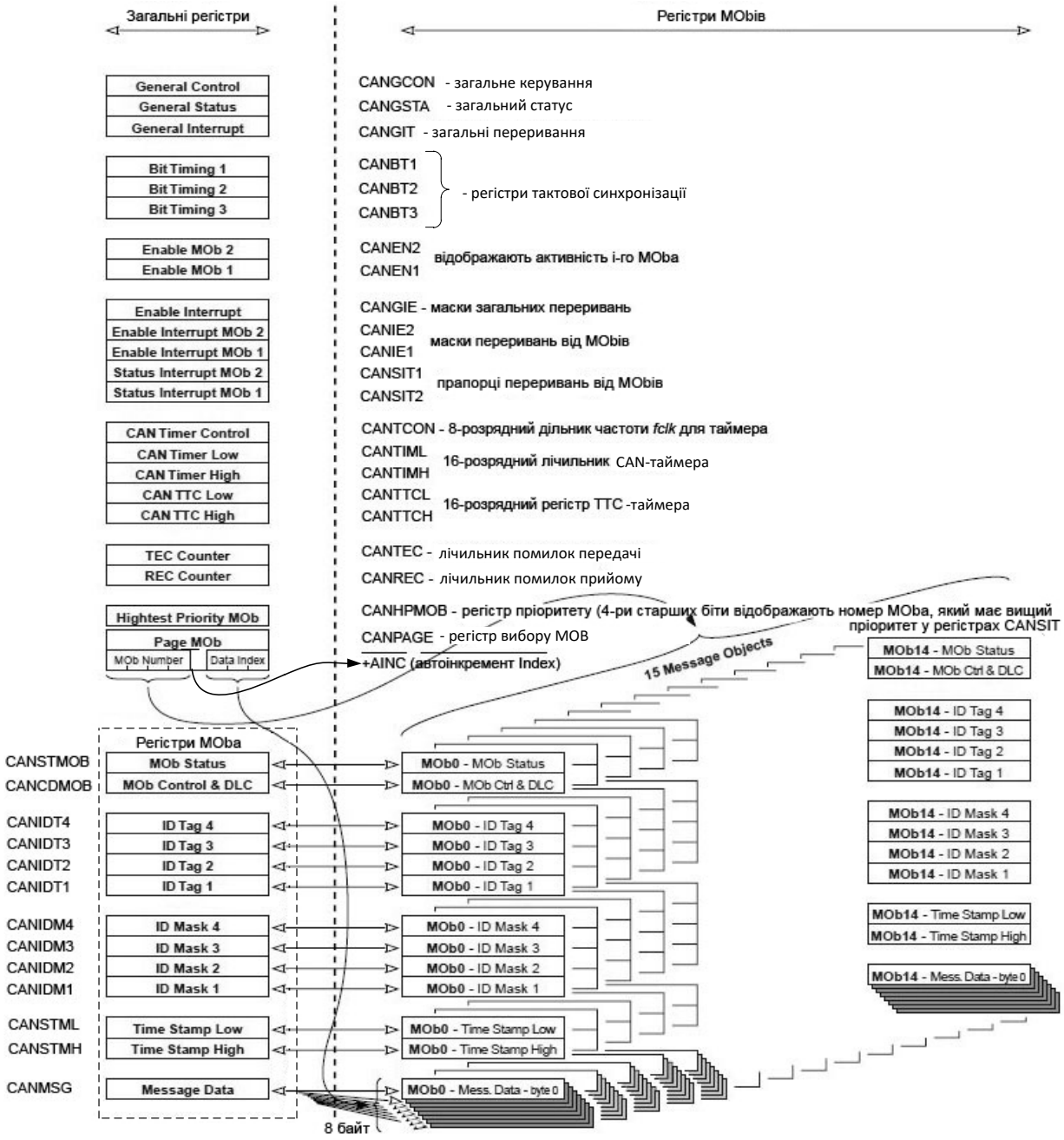


Рис.3. Організація керувальних регістр*ів CAN-модуля*

1.4.4. Режими роботи CAN-модуля

Загальна характеристика режимів роботи

Кожен MOb має свої власні біти, щоб керувати поточним режимом. *Після перезавантаження* МК жоден з MOb не має вибраного режиму роботи.

Перед активацією периферійних пристроїв, під’єднаних до CAN-модуля, кожен MOb *має 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 з табл. 3.

Таблиця 3. Налаштування об'єктів повідомлень

Конфігурація MOb		**Відповідь**	**RTR Tag**	**Поточний режим**
CONMOB0	**CONMOB1**	**Біт RPLV регістра CANCDMOB**		
0	0	x	x	MOb дезактивований
0	1	x	0	Передача кадру даних
		x	1	Передача кадру віддаленого запиту
1	0	x	0	Прийом кадру даних
		0	1	Прийом кадру віддаленого запиту
		1		Прийом кадру віддаленого запиту та автоматична відповідь
1	1	x	x	Прийом кадрів в буфер (прийом мультикадрів)

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».

Передача кадру даних або віддаленого запиту (Tx-конфігурація)

Робота CAN-модуля в цьому режимі відбувається у такій послідовності:

1. Перед відправкою повідомлення або кадру віддаленого запиту мають *бути ініціалізовані* наведені нижче біти керуючих регістрів MOb [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. 3) та ініціалізації регістрів відповідний MOb готовий відіслати кадр даних або віддаленого запиту.
3. Далі CAN-модуль переглядає (сканує) всі MOb у 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.

У цьому випадку CAN-модуль *знаходить MOb, що має найбільший пріоритет* і намагається надіслати його.

4. Коли передачу завершено, встановлюється біт TXOK регістра CANSTMOB. У цьому випадку можливе переривання, якщо воно не замасковано.
5. Усі параметри і дані залишаються доступними в MOb до нової ініціалізації.

Схему алгоритму підготовки MOb до передачі повідомлення чи віддаленого запиту та керувальну програму мовою С наведено у [8].

Прийом кадру даних або віддаленого запиту (Rx-конфігурація)

Робота CAN-модуля в цьому режимі відбувається у такій послідовності:

1. Перед прийомом повідомлення мають бути ініціалізовані наведені нижче поля керуючих регістрів MOb:
   – 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. 3) та ініціалізації регістрів відповідний 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. 3) і встановлюється біт RXOK регістра CANSTMOB. У цьому випадку можливе переривання, якщо воно не замасковано.
6. Усі параметри і дані залишаються доступними в MOb до наступної ініціалізації.

Схему алгоритму підготовки MOb до прийому повідомлення чи віддаленого запиту та керувальну програму мовою С наведено у [8].

Автоматична відповідь

Робота CAN-модуля в цьому режимі відбувається у такій 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 в одиницю (табл. 3).

2. Для цього режиму мають бути ініціалізовані наведені нижче поля керуючих регістрів MOb:
   - 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, оскільки 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, у цьому разі можливе переривання, якщо воно не замасковано.

5. Усі параметри і дані залишаються доступними в MOb до наступної ініціалізації.

Схему алгоритму роботи CAN-модуля у цьому режимі та керувальну програму мовою С наведено у [8].

Прийом мультикадрів

Цей режим використовується для прийому в буфер *відповідного набору кадрів* – отримання *мультикадрів*. Керувати вхідними кадрами такого типу дозволяє

наявність пріоритету 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 у буфер – BXOK регістра CANGIT, у цьому разі можливе переривання, якщо воно не замасковано.

Робота CAN-модуля в цьому режимі відбувається у наступній послідовності:

1. MOb мають бути ініціалізовані як MOb у Rx-конфігурації.
2. MOb будуть готові отримувати кадри даних, коли встановлено їх відповідну конфігурацію (табл. 3).
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. У цьому випадку можливе переривання, якщо воно не замасковано.
6. Коли прийом для останнього MOb з набору завершено, встановлюється біт завершення прийому кадрів у буфер BXOK (регістр CANGIT). У цьому випадку можливе переривання, якщо воно не замасковано.
7. Біт BXOK може бути очищено тоді, коли біти CONMOB0, CONMOB1 регістра CANCDMOB для всіх MOb набору було перезавантажено.
8. Усі параметри і дані залишаються доступними в MOb до наступної ініціалізації.

Схему алгоритму роботи CAN-модуля у цьому режимі та керувальну програму мовою С наведено у [8].

1.4.5. Структура блока фільтрації

Одним з важливих вузлів CAN-модуля є *блок фільтрації* повідомлень (рис. 4).

У цьому блоці *ідентифікатор вхідного* повідомлення побітно *підсумовується за модулем два* з регістом 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 отримуємо послідовність одиничних біт, то повідомлення відповідає вимогам фільтрації та записується у відповідний об’єкт повідомлення (MOb).

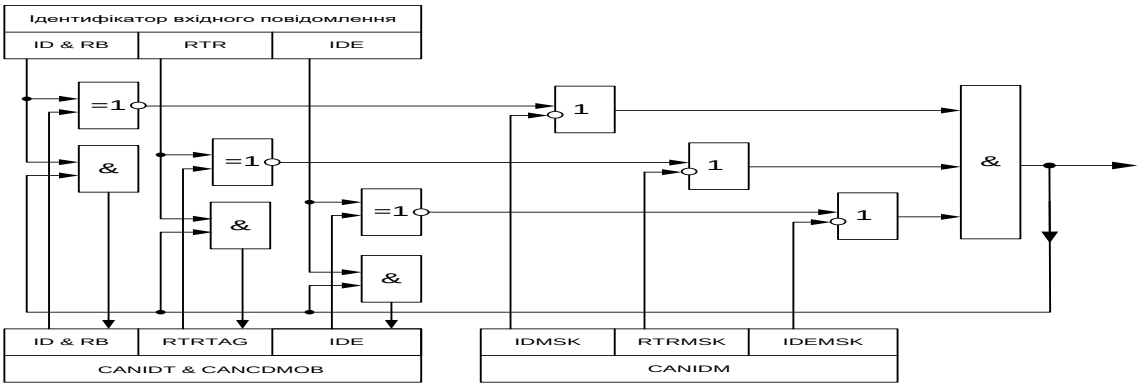


Рис. 4. Структура блока фільтрації повідомлень

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.

Декілька прикладів роботи блока фільтрації для прийому стандартних повідомлень наведено у [8].

1.4.6. 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*. Усі 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 МК AT90CAN128 можлива обробка переривань як *для всіх повідомлень*, так і для *кожного конкретного об'єкта* повідомлень (MOb).

Структуру переривань зображено на рис. 5.

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 розміщено у правій частині рис. 5, формується «Запит на переривання» від різних джерел. Цей запит має адресу в пам'яті програм – 0x0024.

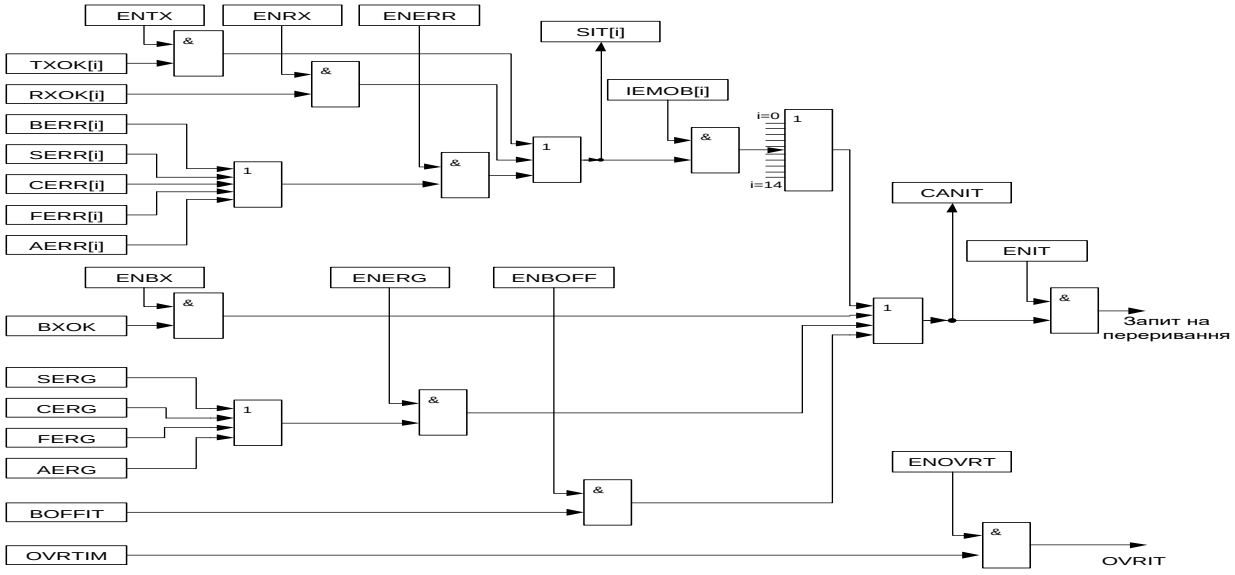


Рис. 5. Структура переривань МК AT90CAN128

Нижче наведено пояснення умовних позначень, які використано на рис. 5.

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 у 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 усіх 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 у 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.

Під час виникнення загального переривання встановлюється в одиницю відповідний прапорець (біт) у регістрі CANGIT. Якщо також в одиницю встановлено «дозволяючий біт» у регістрі CANGIE, який відповідає прапорцю загального переривання, то встановлюється в одиницю загальний прапорець переривань регістра CANGIT – CANGIT.7 (CANIT).

Під час виникнення переривання від одного з MOb встановлюється в одиницю відповідний біт у регістрі CANSTMOB. Встановлення в одиницю прапорця переривань CANIT у цьому випадку відбудеться тільки у разі встановленого «дозволяючого біта» для цього переривання в регістрі CANGIE (CANGIE.3; CANGIE.4; CANGIE.5) та встановленого біта, що дозволяє переривання від цього MOb, в у регістрах CANIE1 або CANIE2.

Для правильного визначення переривань від MOb та загальних переривань спочатку мають бути очищені відповідні біти регістрів CANSTMOB та CANGIT.

1.4.7. Структура блока CAN-таймера

CAN-модуль має блок таймера, структуру якого наведено на рис. 6.

Основним елементом блока є *16-бітний лічильник* – CANTIM. Під час *встановлення в одиницю біта* ENA/$\overline{\text{STB}}$ регістра CANGCON через деякий час CAN-модуль переходить в активний стан, що відображає встановлення в *одиницю* біта *ENFG* регістра CANGSTA.

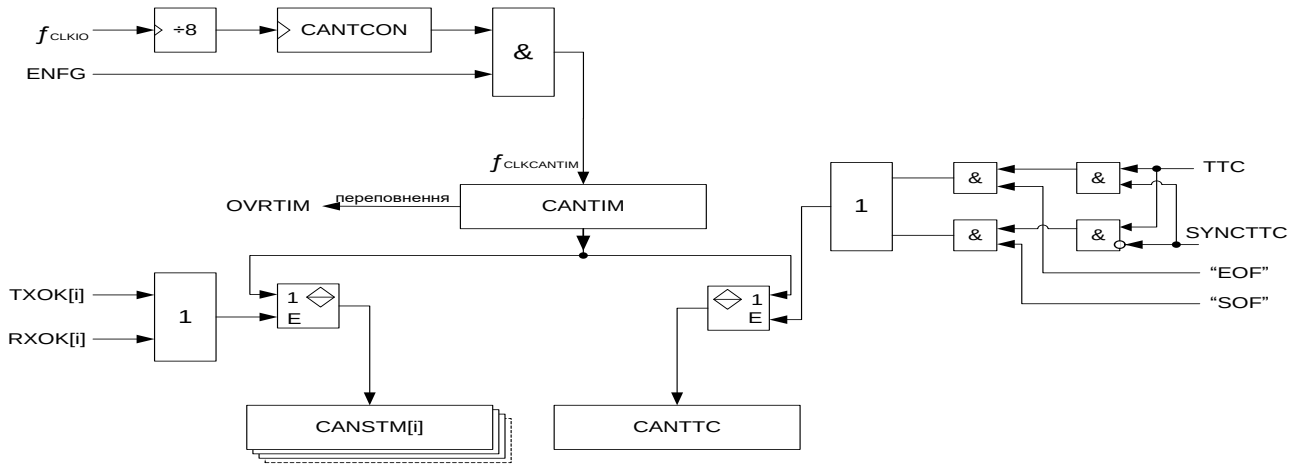


Рис. 6. Структура блока CAN-таймера

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 *8-розрядний програмований дільник* – CANTCON. У результаті роботи цих дільників *період частоти на вході* CANTIM встановлюється відповідно до такого виразу:

$$T_{CLKCANTIM} = T_{CLKIO} \cdot 8 \cdot \left(CANTCON[7:0]+1\right),$$

де T_{CLKIO} – період тактової частоти підсистеми введення/виведення МК [2].

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 поточного 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 є в 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 у разі виявлення 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 у разі виявлення *останнього* одиничного біта поля «кінець кадру», яке складається з 7 одиничних біт, значення з регістра таймера CANTIM записується в регістр CANTTC.

Для того, щоб МК міг брати участь у роботі CAN-шини, де використовується режим TTCAN, необхідно *додатково* програмно реалізовувати *відповідні алгоритми* взаємодії [8].

1.4.8. Обробка помилок

Є *п'ять* різних *типів* помилок, які можуть виникати одночасно: *помилка біта* – виникає, якщо значення біта на шині *відрізняється* від значення, *що передається*. *Виняток* становить поле арбітражу, поле підтвердження та передача додаткового шостого біта протилежної полярності, після передачі п'яти біт однакової полярності; *помилка наповнення виникає*, якщо під час прийому між початком повідомлення та роздільником CRC виявлено підряд 6 біт однакової полярності; *помилка CRC* виникає, якщо значення, що обчислено приймачем, не збігається із прийнятим від передавача; *помилка форми* виявляється, якщо бітове поле фіксованого формату (кінець повідомлення, роздільники CRC і підтвердження) містять один або більше заборонених бітів; *помилка підтвердження* виявляється передавачем щоразу, коли він не виявляє «домінантний» біт в області підтвердження, що свідчить про те, що жоден з вузлів не одержав повідомлення правильно, у цьому разі повідомлення передається повторно.

Для підрахунку помилок використовуються 8-розрядні лічильники помилок: CANTEC – лічильник помилок передачі та CANREC – лічильник помилок прийому. Стан лічильників змінюється згідно з правилами, які описано в [8].

Залежно від кількості помилок CAN-модуль може перебувати в одному з трьох станів: *активна* помилка; *пасивна* помилка та *відключення* від шини.

Вузол у стані "активної помилки" передає кадр (повідомлення) з *активним* прапорецем помилки. Вузол у стані "пасивної помилки" передає кадр (повідомлення) з *пасивним* прапорецем помилки [8].

1.4.9. Бітова синхронізація

Час передачі біта *розділяють на кілька ділянок*, що не перекриваються (рис. 7): сегмент *синхронізації* (SYNC_SEG); сегмент *часу розповсюдження* (PROP_SEG); *фазовий* сегмент *1* (PHASE_SEG1); *фазовий* сегмент *2* (PHASE_SEG2).

Рис. 7. Час передачі біта

Кожен сегмент складається із цілого числа відрізків часу, які називаються *квантами часу* – TQ. *Час передачі біта* коливається *від 8 TQ до 25 TQ* (включно). За *швидкості* передачі *1 Мбіт/с час передачі біта* дорівнює *1 мкс*.

Тривалість *одного кванта* часу – T_{SCL} (T_{TQ}) 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 [8].

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 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 CAN-вузла. Значення цього сегмента може програмуватися *від 1 TQ до 8 TQ*.

Фазові сегменти 1, 2 використовуються, щоб *компенсувати* помилки фази (*фазові зсуви*) під час прийому. Фазовий *сегмент 1* може бути *подовжено*, а фазовий *сегмент* 2 – *вкорочено пересинхронізацією* (ресинхронізацією) [8].

Фазові сегменти призначено *для оптимального розміщення точки вибірки отриманого біта в межах часу біта*, що передається. *Точка вибірки* (Sample point) розміщується *між фазовим сегментом 1 і фазовим сегментом 2* (рис. 7).

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 1 TQ до 8 TQ.*

Якщо біт *має значну 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 прийнятого біта. 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.

Для компенсації зсуву фази між частотами генераторів різних вузлів шини та під час зміни ведучого при арбітражі, кожен CAN-модуль повинен *синхронізуватися до переходу рівня вхідного сигналу з одиниці на нуль* (рис. 7). *Коли* перехід *виявлено*, то схема синхронізації *порівнює розміщення переходу* з переходом, що очікується під час прийому, та виконує *налаштування значень фазових сегментів 1 та 2*.

Є два механізми синхронізації: *апаратна* синхронізація та *синхронізація з відновленням тактових інтервалів* (пересинхронізація/ресинхронізація).

Апаратна синхронізація виконується під час переходу від «рецесивного» до «домінантного» біта протягом холостого стану шини, що вказує на початок повідомлення. *Під час апаратної* (жорсткої) 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. 7).

Синхронізація з відновленням тактових інтервалів (пересинхронізація/ресинхронізація) призначена для зменшення фазових спотворень під час прийому та *виконується автоматичним подовженням фазового сегмента 1 або скороченням фазового сегмента 2. Максимальне значення зміни* фазових сегментів коливається в межах *від 1 TQ до 4 TQ*. Синхронізація *виконується під час переходів* від *«рецесивного» до «домінантного»* біта протягом прийому кадру. Фіксоване значення максимальної кількості послідовних бітів 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 під час попередньої точки 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 вибірки (зчитування) бітового інтервалу приймача, протягом якого з'явився перехід вхідного сигналу з одиниці в нуль, та часу появи цього переходу – $t_{↘}$ (рис. 7).

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 вибірки (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 «e»* менше або дорівнює значенню *ширини періоду пересинхронізації, що програмується.*

Коли величина *помилки фази «e»* більша, ніж ширина переходу пересинхронізації, що програмується, тоді:

- якщо *помилка фази «e» додатна*, то фазовий *сегмент 1 подовжується* на ширину періоду пересинхронізації, що програмується;
- якщо *помилка фази «e» від'ємна*, то фазовий *сегмент 2 скорочується* на ширину періоду пересинхронізації, що програмується.

На рис. 8, 9 наведено *два приклади* пересинхронізації.

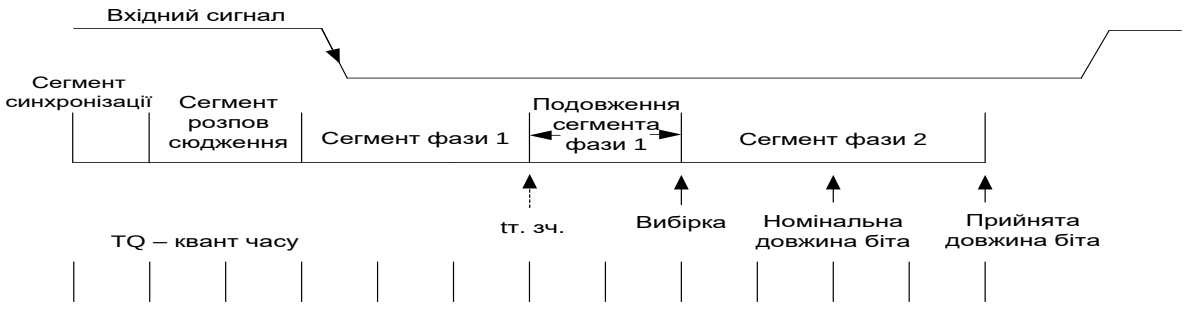


Рис. 8. Синхронізація з відновленням тактових інтервалів завдяки автоматичному подовженню фазового сегмента 1

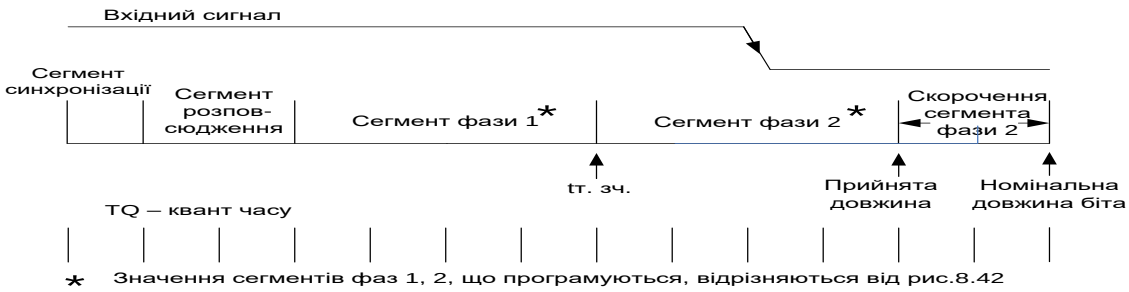


Рис. 9. Синхронізація з відновленням тактових інтервалів завдяки автоматичному скороченню фазового сегмента 2

У [8] наведено приклад програмування швидкості обміну інформацією CAN-мережею.

1.4.10. Фізичний рівень CAN-протоколу

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, SAEJ2411 і т. ін. [8]. У переважній більшості випадків використовується фізичний рівень CAN, який описаний у стандарті ISO 11898 Останній в якості середовища передачі визначає двопровідну диференціальну лінію з імпедансом (термінатори) 120 Ом (допускається коливання імпедансу 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. 4.

Таблиця 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

За довжини кабелю *30 метрів* швидкість передачі даних CAN-мережею буде максимальною та дорівнює *1 Мбіт/с*.

У разі необхідності передавати дані на більші відстані рекомендовано використовувати вузли-ретранслятори.

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» [8].

Для з'єднання окремих вузлів з CAN-шиною можуть використовуватися 9-контактні роз'єм D-Sub [8].

Приклад принципової схеми вузла CAN-мережі на мікроконтролері AT90CAN128 наведено у [8].

Контрольні запитання та завдання

1. Дайте характеристику CAN-протоколу. Назвіть його основні переваги.
2. Опишіть структуру CAN-мережі.
3. Який завадостійкий код використовується у CAN-мережі? Дайте характеристику цьому коду.
4. Поясніть основні характеристики CAN-модуля.
5. Перерахуйте типи помилок, що можуть бути виявлені під час передачі CAN-мережею. Наведіть механізми виявлення помилок.
6. Назвіть рівні сигналів під час передачі CAN-шиною.
7. Дайте визначення поняттю «фрейм». Типи фреймів. Формати фреймів.
8. Для чого потрібна синхронізація? Назвіть та опишіть види синхронізації.
9. Наведіть основні характеристики CAN-модуля МК AT90CAN128.
10. Опишіть структуру CAN-модуля мікроконтролера AT90CAN128.
11. Дайте характеристику режимів роботи CAN-модуля.
12. Опишіть послідовність програмування режиму передачі кадру (повідомлення) або віддаленого запиту (Tx-конфігурація).
13. Опишіть послідовність програмування режиму прийому кадру даних або віддаленого запиту (Rx-конфігурація).
14. Опишіть програмування режиму прийому мультикадрів у буфер.
15. Опишіть структуру блока фільтрації повідомлень.
16. Перерахуйте переривання від CAN-модуля. Опишіть структуру переривань.
17. Опишіть структуру блока таймера CAN-модуля.
18. Опишіть організацію регістрів CAN-модуля.
19. Дайте характеристику фізичному рівню (Physical Layer) CAN-протоколу.
20. Вкажіть максимальну швидкість передачі даних CAN-мережею. Від чого вона залежить?
21. Опишіть алгоритми роботи вузла CAN-мережі в різних режимах.

ЛЕКЦІЯ 18. МОДУЛЬ АНАЛОГОВОГО КОМПАРАТОРА. СПЕЦІАЛЬНІ РЕЖИМИ РОБОТИ МІКРОКОНТРОЛЕРА.

1. МОДУЛЬ АНАЛОГОВОГО КОМПАРАТОРА

1.1. Загальні відомості про компаратор

Компаратором називається електронний пристрій, який призначено для порівняння двох напруг. В залежності від форми представлення порівнюваних сигналів компаратори поділяються на: аналогові компаратори (АК) та цифрові компаратори (ЦК) [5].

1.2. Аналоговий компаратор у складі AVR-мікроконтролерів

До складу МК сімейства AVR, входить модуль АК [2]. Будучи увімкненим, компаратор дає можливість порівнювати значення напруг на двох відповідних виводах МК. *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 Т1. Остання функція дозволяє *вимірювати тривалості* аналогових сигналів. Виводи, що використовуються компаратором, є *контактами портів* введення/виведення загального призначення [2].

Щоб вони могли використовуватися АК, 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 МК є можливість *відключення вхідних цифрових* буферів у випадку, якщо контакти МК, що відповідають виводам аналогового компаратора: AIN0 та AIN1, використовується тільки для введення аналогових сигналів [2]. У разі відключення цифрових буферів *зменшується загальний струм* споживання МК, а відповідні розряди регістра PINx завжди читаються як нуль.

1.3. Функціонування та програмування компаратора

На рис. 1 наведено структурну схему АК.

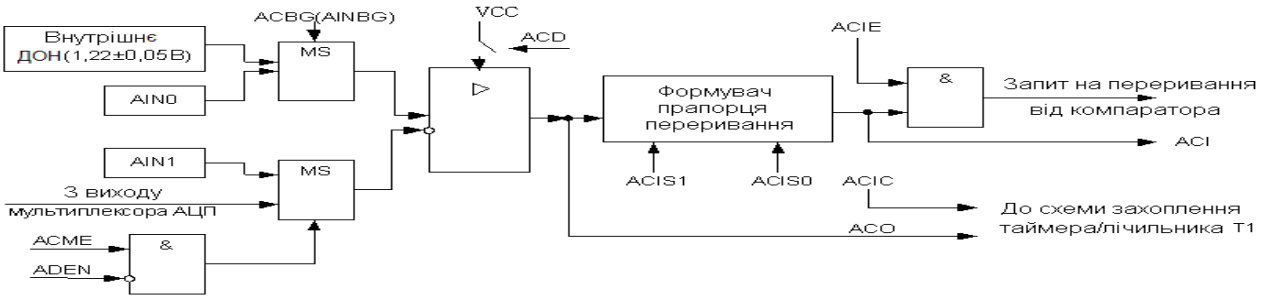


Рис. 1. Структурна схема АК

Керування компаратором і контроль його стану здійснюється за допомогою регістра ACSR [2]. Призначення розрядів цього регістра описано в табл. 1.

Таблиця 1. Призначення розрядів регістра ACSR

Розряд	**Назва**	**Опис**
7	ACD	Вимикання компаратора (нуль – ввімкнено, одиниця – вимкнено)
6	ACBG (AINBG*)	Підключення до неінвертуючого входу компаратора внутрішнього джерела опорної напруги (ДОН) (нуль – не підключено, одиниця – підключено)
5	ACO	Результат порівняння (вихід компаратора)
4	ACI	Прапорець переривання від компаратора
3	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		

Модуль АК AVR-мікроконтролерів є *звичайним аналоговим* компаратором. Якщо напруга на виводі AIN0 (неінвертуючий 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 у *розряді ACO* регістра ACSR. За *вмикання і вимикання* компаратора відповідає *розряд* ACD. Оскільки *під час подачі напруги живлення* всі розряди регістра ACSR скидаються в нуль, компаратор *вмикається автоматично*. Для *вимикання* компаратора розряд ACD слід встановити в *одиницю*. У разі зміни 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 під час запуску* підпрограми обробки переривання або *програмно записом у нього одиниці*.

Для *дозволу переривання* необхідно встановити в одиницю розряд ACIE 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. 2.

Таблиця 2. 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

У разі програмування 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 неінвертуючого входу компаратора може бути підключено внутрішнє джерело опорної напруги (ДОН) величиною 1,22 (±0,05)В. Для цього необхідно встановити *в одиницю розряд* ACBG 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, або в регістрі спеціальних функцій SFIOR (3-й розряд регістра), або в регістрі керування АЦП ADCSRB (6-й розряд) [2]. *Модуль* АЦП у цьому разі має *бути вимкнений* (розряд ADEN регістра ADCSRA треба скинути в нуль).

Основні параметри АК наведено у [2].

2. СПЕЦІАЛЬНІ РЕЖИМИ РОБОТИ МІКРОКОНТРОЛЕРА

2.1. Тактування мікроконтролера

2.1.1. Загальні відомості про тактування

Нижче розглянуто виконання тактування AVR-мікроконтролерів на прикладі сімейства Mega [2]. Тактові сигнали формує *пристрій синхронізації*, спрощену структуру якого наведено на рис. 2.

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*.

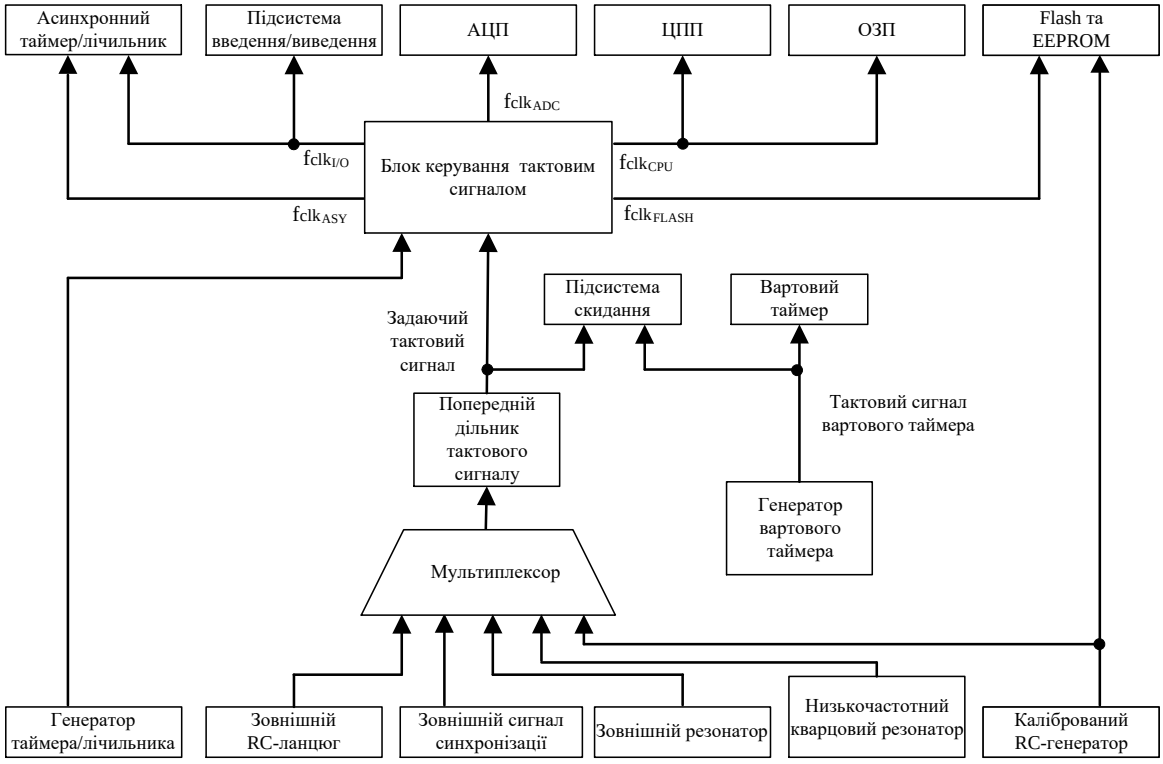


Рис. 2. Спрощена структура пристрою синхронізації

Можливість використання того чи іншого джерела тактового сигналу залежить від моделі МК [2].

На базі системного тактового сигналу формуються додаткові сигнали, що використовуються для тактування різних модулів і блоків МК (рис. 1):

– f_{clkCPU} – тактовий сигнал центрального процесора. Використовується для тактування блоків МК, що відповідають за роботу з ядром МК: регістровий файл, пам’ять даних і т. ін. У разі відключення цього сигналу центральний процесорний пристрій (ЦПП) зупиняється і відповідно всі обчислення припиняються;

– $f_{clkI/O}$ – тактовий сигнал підсистеми введення/виведення. Використовується більшістю периферійних пристроїв, таких як таймери/лічильники та інтерфейсні модулі. Цей сигнал використовується також підсистемою зовнішніх переривань, проте деякі зовнішні переривання можуть 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 у разі знаходження МК у «Сплячому» режимі;

– f_{clkADC} – тактовий сигнал для модуля АЦП. Наявність цього тактового сигналу дозволяє здійснювати перетворення під час зупинених ЦПП і підсистеми введення/виведення. У цьому разі значно зменшується рівень завад, що генеруються МК, і відповідно збільшується точність перетворення.

Оскільки архітектура МК повністю статична, *мінімально допустиму частоту* нічим не обмежено (аж до покрокового режиму роботи), а *максимальна робоча* частота визначається конкретною моделлю МК.

Вибір режиму роботи тактового генератора здійснюється *програмуванням конфігураційних комірок (*FUSE Bits) CKSEL3...0. Необхідні значення для кожного режиму роботи наведено у [2].

2.1.2. Генератор із зовнішнім резонатором

Резонатор підключається до виводів XTAL1 і XTAL2 МК, як показано на рис. 3.

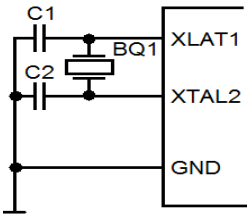


Рис. 3. Підключення кварцового чи керамічного резонатора до МК

Ці виводи є відповідно *входом і виходом інвертуючого підсилювача* тактового генератора, який вбудовано у МК. Конденсатори С1 і С2 призначено для *підвищення стабільності* роботи генератора. Їх ємності залежать від типу резонатора [2]. *Низькочастотний кварцовий резонатор*, так званий «годинниковий кварц», частотою 32768 Гц підключається до виводів TOSC1 та TOSC2 мікроконтролера (лекція 8).

2.1.3. Зовнішній сигнал синхронізації

Сигнал від зовнішнього джерела подається на вивід XTAL1, як показано на рис. 4.

Рис. 4. Підключення зовнішнього джерела тактового сигналу

Цей сигнал повинен задовольняти вимогам МК за частотою, шпаруватістю та рівнями напруги. Вивід XTAL2 залишають непідключеним.

2.1.4. Генератор із зовнішнім та внутрішнім RC-ланцюгом

У разі реалізації додатків, які *не потребують високої часової точності*, можна використовувати найпростіший RC-генератор. У цьому разі зовнішній RC-ланцюг підключається до виводу XTAL1, як показано на рис. 5.

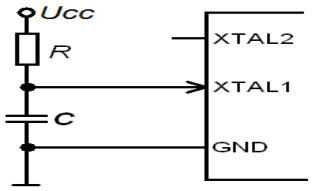


Рис. 5. Підключення зовнішнього RC-ланцюга

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 [2]:

$$f = \frac{1}{3 \cdot \mathrm{RC}}.$$

Використання вбудованого RC-генератора з внутрішнім часозадаючим RC-ланцюгом є *найбільш економічним* рішенням, тому що у цьому разі не потрібні ніякі зовнішні компоненти. Номінальні частоти внутрішнього RC-генератора для деяких моделей наведено у [2].

2.1.5. 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 МК. Якщо асинхронний таймер/лічильник працює в асинхронному 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 у [2].

2.2. Режими зниженого енергоспоживання

2.2.1. Загальні відомості про режими зниженого енергоспоживання

AVR-мікроконтролери мають *декілька режимів зниженого* енергоспоживання, що називають SLEEP-режимом [2]. Кожен з цих режимів дозволяє знизити енергоспоживання МК за рахунок *відключення виконання окремих робочих функцій*. Вхід в будь-який зі «сплячих» режимів виконується *командою* SLEEP. Під час переходу у SLEEP-режим *виконання програми припиняється*, а поновлюється у разі настання певних подій. Під час *виходу* МК зі SLEEP-режиму

виконання програми продовжується з місця зупинки. У нових моделях, крім того, передбачено зниження енергоспоживання кристала *відключенням тактових сигналів* незадіяних периферійних модулів.

2.2.2. Керування режимами зниженого енергоспоживання

Мікроконтролери сімейства MEGA підтримують *від 3 до 6 режимів* зниженого енергоспоживання: Idle; ADC Noise Reduction; Power Down; Power Save; Standby та Extended Standby [2]. Режими *відрізняються числом периферійних* пристроїв МК, функціонуючих під час «сну», і відповідно ступенем зменшення енергоспоживання. Залежно від моделі для керування «сплячим» режимом використовується різне число регістрів введення/виведення. Формати цих регістрів наведено у [2]. Перехід в будь-який з режимів зниженого споживання здійснюється командою SLEEP. У цьому разі *прапорець* SE регістра MCUCR повинний бути встановлений в *одиницю*. Щоб уникнути ненавмисного перемикання МК у «сплячий» режим рекомендується встановлювати цей прапорець *безпосередньо перед виконанням* команди SLEEP. *Режим*, в який перейде МК після виконання команди SLEEP, визначається *станом розрядів* SM2...SM0 регістра MCUCR.

Вихід зі «сплячого» режиму може бути здійснено:

- у результаті переривання. Під час генерації переривання МК 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 у цьому разі не змінюється;
- у результаті скидання. Після переходу МК в робочий режим керування передається за адресою вектора скидання.

Особливості окремих режимів зниженого енергоспоживання та рекомендації по їх застосуванню розглянуто у [2].

2.3. Скидання

Переведення роботи мікроконтролера у початковий стан виконується шляхом так званого «Скидання». Останнє може бути викликано *наступними подіями*: включення напруги живлення МК; подача сигналу *низького рівня* на вивід «RESET» (апаратне скидання); тайм-аут вартового таймера; падіння напруги живлення нижче заданої величини та скидання за інтерфейсом JTAG [2]. Під час настання 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 [2].

Структурну схему підсистеми скидання наведено на рис. 6.

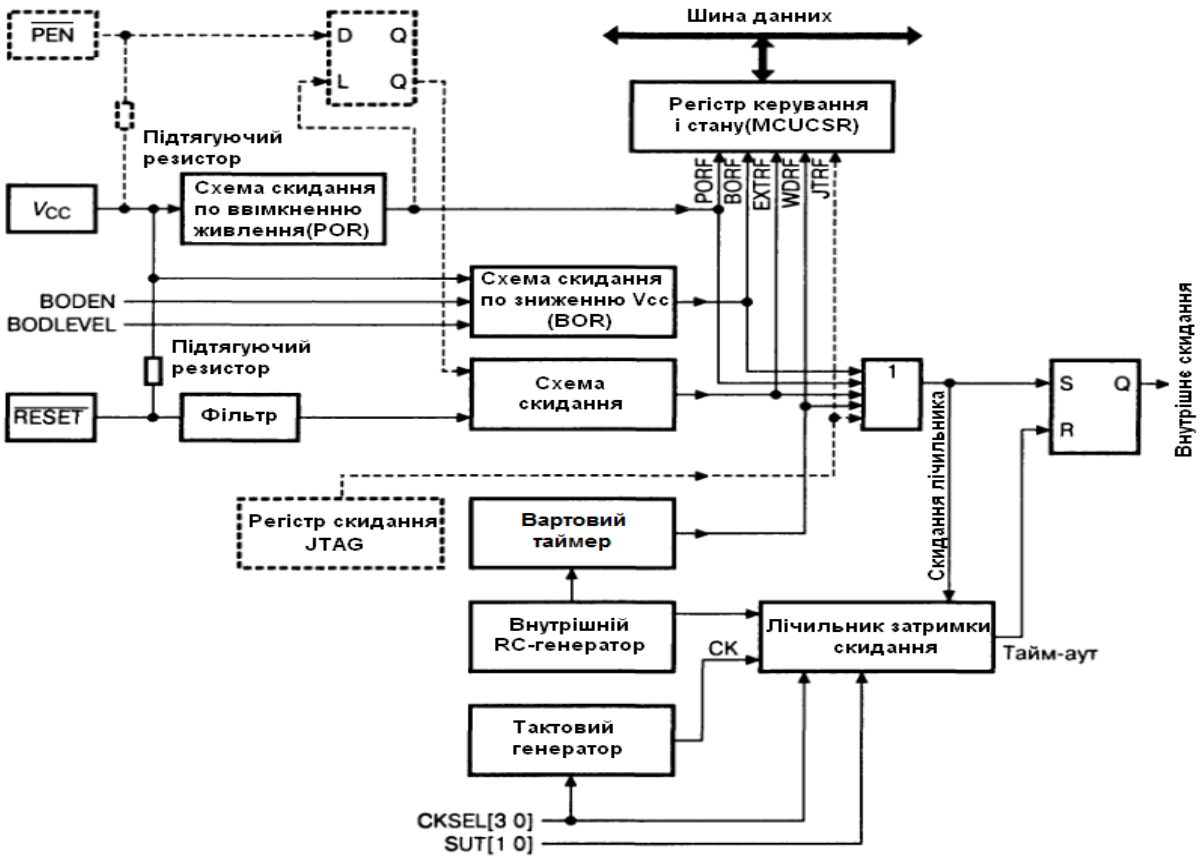


Рис. 6. Структурна схема підсистеми скидання

У *деяких моделей відсутні* елементи, які на рис. 6 виділено *пунктиром*. Схема скидання працює наступним чином. У разі настання події, що приводить до скидання МК, формується внутрішній сигнал «Скидання». Одночасно запускається *таймер формування затримки скидання.* Після закінчення певного проміжку часу внутрішній сигнал скидання знімається і починається виконання програми.

2.5. Самопрограмування AVR-мікроконтролерів

2.4.1. Загальні відомості про самопрограмування

Пам’ять мікропроцесорних систем *з початку виникнення* останніх поділялася на: *постійний* запам’ятовуючий пристрій (ПЗП), або ROM – англ. Read-Only Memory – пам’ять тільки для читання та *оперативний* запам’ятовуючий пристрій (ОЗП), або RAM – англ. Random Access Memory – пам’ять з довільним доступом. В ПЗП зберігалися робочі програми, а в ОЗП – дані, які приймали участь в операціях. Постійний запам’ятовуючий пристрій відносився до *енергонезалежної* пам’яті, яка програмувалася на етапі створення МПС та в процесі роботи не змінювалася, тобто *програма тільки читалася*. Оперативний запам’ятовуючий пристрій відносився до *енергозалежної* пам’яті, яка виконувалася *на тригерах* [5]. Якщо зникало живлення такої пам’яті, то її зміст губився.

В сучасних МПС *в якості ОЗП крім* RAM використовується енергонезалежна EEPROM-пам’ять. Крім того з’явилася *можливість змінювати* ПЗП під час роботи МПС. Для цього сучасні AVR-мікроконтролери, наприклад, всі МК сімейства Mega та XMega мають *відповідний модуль* та *можливість самопрограмування*, тобто можуть самостійно змінювати вміст своєї пам’яті програм під час виконання програми. Ця особливість дозволяє створювати на їхній основі дуже гнучкі системи, *алгоритм роботи яких буде змінюватися* самим МК залежно від відповідних внутрішніх умов або зовнішніх подій [2].

Для підтримки процесу самопрограмування *всю область пам’яті програм* логічно розділено на *дві секції* – секцію *прикладної* програми (Application Section) і секцію *завантажувача* (Boot Loader Section) (рис. 7).

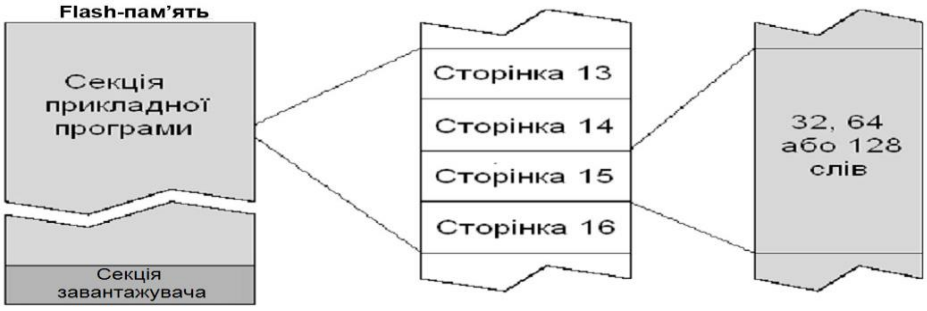


Рис. 7. Розділення пам’яті програм на дві секції

Програмування пам’яті програм здійснюється *програмою-завантажувачем*, яку розміщено в *однойменній* секції.

Розмір секції завантажувача, її розміщення в пам’яті програм та розмір секції прикладної програми в більшості МК визначається *двома конфігураційними* комірками BOOTSZ1:BOOTSZ0 [2].

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 або TWI, наявний у конкретному МК.

Завантажувач *може змінювати вміст обох секцій*, що дозволяє модифікувати власний код та видаляти себе з пам’яті, якщо потреби в ньому не буде. 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 МК. *Положення вектора скидання* визначається станом конфігураційної комірки BOOTRST. Якщо в ній зберігається одиниця, вектор скидання розміщується на початку пам’яті програм за адресою $0000. Для запрограмованої комірки, коли в ній зберігається нуль, вектор скидання розміщується на початку секції завантажувача.

Пам’ять програм *поділяють 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 [2] наведено розміри цих областей для деяких AVR-мікроконтролерів сімейства Mega та описано їх використання під час програмування МПС.

2.4.2. Керування процесом самопрограмування

Керування процесом програмування здійснюється *командою* SPM (англ. Store Program Memory) у разі використання *регістрів* введення/виведення SPMCR/SPMCSR (Store Program Memory Control/Status Register). Формати цих регістрів для різних моделей сімейства та опис їх розрядів наведено у [2].

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.

Для адресації пам’яті програм *у разі використання* 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 *додатковий регістр* введення/виведення RAMPZ. *Дані* записуються в *регістрову пару* R1:R0.

Оскільки пам’ять програм в мікроконтролерах сімейства Mega має *сторінкову організацію* (табл. 3), лічильник команд можна *умовно розділити* на дві частини.

Перша частина (*молодші розряди*) адресують комірки на сторінці, а *друга частина* визначає сторінку (рис. 8, табл. 4).

Рис. 8. Адресація пам’яті програм під час використання команди SPM

Під час самопрограмування регістр Z використовується *для адресації окремих слів* в ПП, тому молодший біт регістра Z: Z0 = 0, а інші біти адресують слова.

Як приклад, у [2] 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 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	$2^7 \cdot 2^6 = 8K$	$2^7 = 128$	$2^6 = 64$
ATmega32x ATmega324x ATmega325x ATmega3250x	$2^8 \cdot 2^6 = 16K$	$2^8 = 256$	$2^6 = 64$
ATmega64x ATmega640 ATmega645x ATmega6450x	$2^8 \cdot 2^7 = 32\ K$	$2^8 = 256$	$2^7 = 128$
ATmega128x ATmega1280x ATmega1281x	$2^9 \cdot 2^7 = 64K$	$2^9 = 512$	$2^7 = 128$
ATmega2560x ATmega2561x	$2^{10} \cdot 2^7 = 128\ K$	$2^{10} = 1024$	$2^7 = 128$

2.4.3. Зміна вмісту пам’яті програм

Загальні відомості про зміну вмісту пам’яті програм

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.

Таблиця 4. Адресація пам’яті програм під час використання 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 розміщуються у регістрі RAMPZ

Слід зазначити, що *очищення сторінки може виконуватися* як після заповнення буфера, так і перед його заповненням. Однак, якщо необхідно змінити тільки частину сторінки, наведений вище 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. 8), записати значення «x0000011» у регістр SPMCR і протягом чотирьох машинних циклів виконати команду SPM. Вміст регістрів R1 та R0 у цьому разі ігнорується. В мікроконтролерах з числом сторінок 512 або 1024 для адресації використовуються *додаткові молодші біти* регістра RAMPZ: Z16 = RAMPZ0, Z17 = RAMPZ1 [2].

Для занесення слова команди в буфер слід завантажити адресу комірки в регістр Z (секція ZWORD) (рис. 8), а слово команди – у регістри R1:R0. Після цього необхідно записати значення «x0000001» в 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 наступним чином: в регістр Z (секція ZPAGE) заноситься адреса сторінки, в регістр SPMCR записується значення «x0000101» і протягом чотирьох машинних циклів виконується команда SPM. Вміст регістрів R1 та R0 у цьому разі 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. Під час заповнення 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. 9.

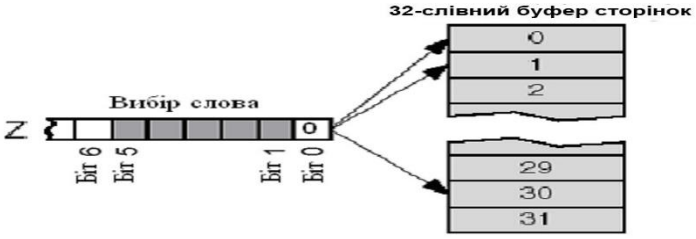


Рис. 9. Запис слова у буфер сторінок

Мікроконтролери, що мають більші розміри сторінки для вибору слів використовують більше біт. Наприклад, ATmega128 *для адресації слів* використовує молодші біти Z1...Z7 (табл. 4), тому що одна сторінка містить $2^7 = 128$ слів/$2^8 = 256$ байт (табл. 3).

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. 8). 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 в 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 в регістрі SPMCR, і протягом чотирьох машинних циклів необхідно виконати команду SPM. Вміст регістрів R1:R0 у цьому разі ігнорується. *Як приклад*, використання регістра Z для мікроконтролера, сторінки якого містять 32 слова (64 байти) показано на рис. 10.

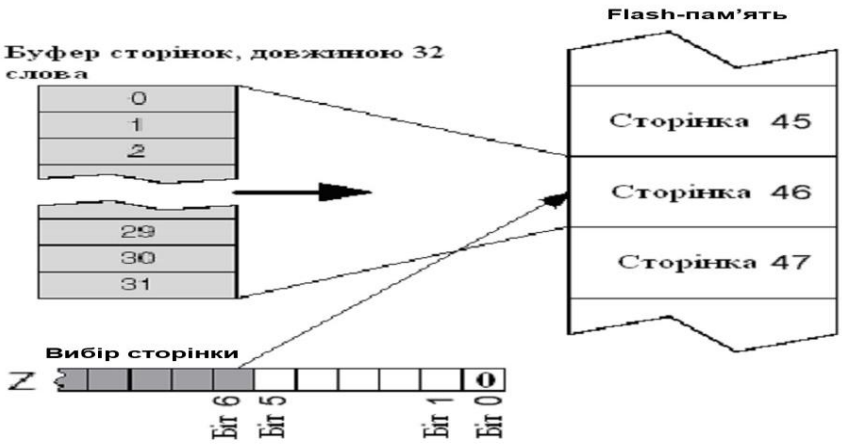


Рис. 10. Запис сторінки у Flash-пам’ять

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 [2], програмуванням яких *можна заборонити доступ* із

прикладної програми до програми завантажувача та *забезпечити завантажувачу* можливість доступу до секції прикладної програми.

SPM-переривання

У всіх мікроконтролерів, що підтримують режим самопрограмування, процес оновлення Flash-пам’яті можна контролювати за допомогою переривань. Встановлення біта SPMIE в 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.

Режими захисту Flash- та EEPROM-пам’яті

Вміст Flash- та EEPROM-пам’яті *можна захистити* від запису та читання програмуванням комірок захисту (Lock Bits) LB1 та LB2. Ці біти можна змінити тільки за допомогою послідовного або паралельного програмування [2].

Читання конфігураційних комірок та комірок захисту

Крім програмування мікроконтролера, завантажувач *може також зчитувати* вміст конфігураційних комірок та комірок захисту [2].

2.4.4. Типові процедури оновлення Flash-пам’яті

На рис. 11 показано *дві стандартні схеми* алгоритмів оновлення Flash-пам’яті.

Схема ліворуч (рис. 11*а*) описує алгоритм «зчитування-модифікації-запису» невеликих частин Flash-пам’яті, наприклад, констант. Схема праворуч (рис. 11*б*) описує алгоритм запису сторінки *без попереднього зчитування її вмісту*, наприклад, виконується запис даних, що надійшли від UART.

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 (виконати переактивізацію RWW-секції) [2].

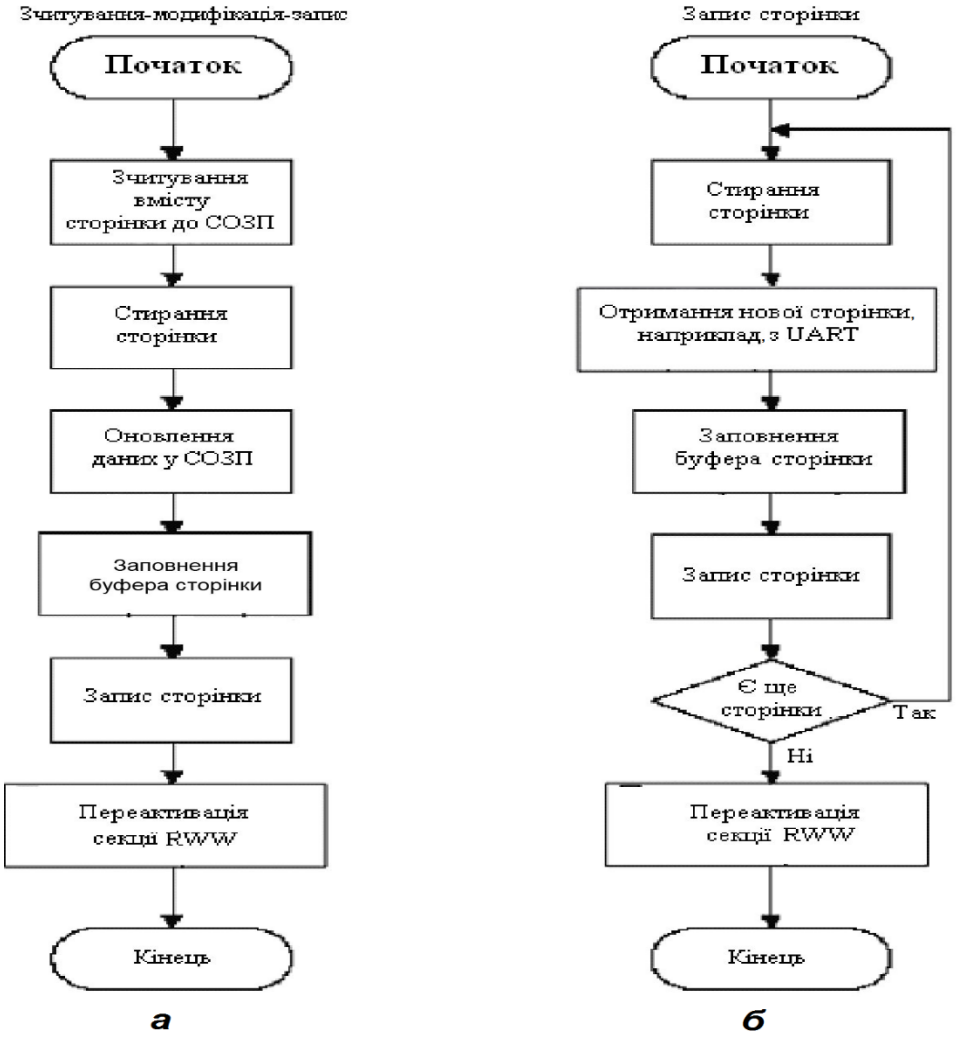


Рис. 11. Схеми алгоритмів оновлення Flash-пам’яті

Примітка: Необхідні дії для переактивації секції RWW описано вище у [2].

Контрольні запитання та завдання

1. Як називається пристрій, який формує тактові сигнали для МК?
2. Які джерела тактового сигналу можуть використовуватись у AVR-МК?
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 у разі використання 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 fclkI/O? Як він формується?
11. Чим обмежено мінімально допустиму тактову частоту МК?
12. За якою командою здійснюється перехід у «сплячий» режим?
13. Скільки режимів зниженого енергоспоживання підтримують МК 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 МК?
18. Як може бути здійснено вихід зі «сплячого» режиму?
19. Які види скидання можна використовувати у МК 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. Який електронний пристрій називається компаратором?
25. На які види поділяються компаратори в залежності від форми представлення порівнюваних сигналів?
26. Який електронний пристрій виконує функцію власне аналогового компаратора?
27. Поясніть роботу АК, який виконано на ІМС ОП.
28. Поясніть призначення схеми формування рівнів.
29. Наведіть та поясніть структуру модуля аналогового компаратора.
30. Наведіть та поясніть формати керувальних регістрів модуля АК.

31. Яким чином можна підключити мультиплексор АЦП до АК?
32. Від чого залежить стан виходу компаратора?
33. Де зберігається стан виходу компаратора?
34. Назвіть умови генерації запиту на переривання від компаратора.
35. Які виводи МК використовуються АК та як вони повинні бути сконфігуровані?
36. Який розряд відповідає за вмикання та вимикання компаратора?
37. Як вихід компаратора можна підключити до схеми захоплення таймера/лічильника Т1?
38. Поясніть, що таке самопрограмування МК та для чого воно використовується.
39. Як логічно розділено всю область пам’яті програм для підтримки самопрограмування?
40. Яким чином відбувається керування процесом самопрограмування МК?
41. Опишіть процес очищення сторінки пам’яті.
42. Поясніть, у чому полягає особливість використання переривань під час самопрограмування.
43. Як може виконуватись завантаження нового вмісту пам’яті програм у мікроконтролер?
44. Чим визначається розмір секції завантажувача та розмір секції прикладної програми?
45. Опишіть призначення окремих розрядів регістра SPMCR.
46. Як відбувається заповнення тимчасового буфера сторінки новим вмістом?
47. Яким чином новий вміст пам’яті програм під час самопрограмування потрапляє в МК?
48. Поясніть призначення та використання SPM-переривання.
49. Наведіть та поясніть алгоритми оновлення Flash-пам’яті під час самопрограмування.

СПИСОК ЛІТЕРАТУРИ

Базова

1. 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.
2. Електроніка та мікропроцесорна техніка : підручник. У 2 ч. Ч. 2. Мікропроцесорні системи [Електронний ресурс]/А. О. Новацький. – Електронні текстові дані (1 файл: 20,2 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, Вид-во «Політехніка», 2023. – 489с.
3. Мікропроцесорні та мікроконтролерні системи: Частина 2. Проектування мікропроцесорних систем: Лабораторний практикум [Електронний ресурс] : навч. посіб. для студ. освітньої програми «Інтегровані інформаційні системи» спеціальності 126 «Інформаційні системи та технології» / А.О. Новацький ; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 22,38 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2021. – 268 с.
4. Проектування вбудованих систем: Лабораторний практикум [Електронний ресурс] : навч. посіб. для студ. освітньої програми «Інтегровані інформаційні системи» спеціальності 126 «Інформаційні системи та технології» / А.О. Новацький, В.М. Шимкович; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 34,22 Кбайт). – Київ : КПІ ім. Ігоря Сікорського, 2022.
5. 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.
6. Електроніка і мікропроцесорна техніка. Частина 1. Комп’ютерна електроніка: Лабораторний практикум [Електронний ресурс] : навч. посіб. для здобувачів ступеня бакалавра за освітньою програмою «Інтегровані інформаційні системи» спеціальності 126 «Інформаційні системи та технології» / А.О. Новацький, Ю. М. Бердник; Електронні текстові дані (1 файл: 20,8 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2024. – 275 с.

Допоміжна

7. Проектування та програмування мікропроцесорних систем і мереж: Проектування мережі 1–WIRE: Навчальний посібник для студентів спеціальностей 7.05020101, 8.05020101 «Комп’ютеризовані системи управління

та автоматика» кафедри автоматики та управління в технічних системах / Автор: А.О. Новацький– К: НТУУ „КПІ”, 2014.
8. Навчальний посібник з дисципліни «Проектування мікропроцесорних систем та мереж», розділ «Проектування CAN-мережі» 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”, 2016.
9. Официальное описание микроконтроллеров XMEGA http://www.gaw.ru/html.cgi/txt/ic/Atmel/micros/avr_xmega/start.htm.
10. Електроніка та мікропроцесорна техніка: Курсова робота [Електронний ресурс] : навч. посіб. для студ. освітньої програми «Інтегровані інформаційні системи» спеціальності 126 «Інформаційні системи та технології» / А.О. Новацький, КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 6,2 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2023. – 101с.