Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

МІКРОПРОЦЕСОРНІ ПРИСТРОЇ

Навчальний посібник

Київ
«Кафедра»
2017

УДК 004.318:621.3.049.77](075.8)
М59

Гриф надано Вченою радою КПІ ім. Ігоря Сікорського
(Протокол № 06 від 12 червня 2017р.)

Рецензенти:
Новський В. О., д. т. н., п. н. с., Інститут елекродинаміки НАН України;
Ільїна Н. О., д. т. н., проф. кафедри промислової та біомедичної електроніки, Національний технічний університет «Харківський політехнічний інститут».

Відп. редактор:
Хохлов Ю. В., к. т. н., доц., Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського».

Терещенко Т. О.

М59 **Мікропроцесорні пристрої**: навч. посібник для студентів зі спец-ті «Електроніка» **/ Т. О. Терещенко, В. А. Тодоренко, Л. М. Батрак, Ю. С. Ямненко**. – К.: Кафедра, 2017. – 244 с.
ISBN 978-617-7301-37-9

У посібнику наведено принципи побудови мікропроцесорних систем, архітектура і характеристики AVR-мікроконтролерів, організація пам`яті програм та даних. Докладно і з численними прикладами розглянуто периферійні пристрої – порти вводу/виводу, таймери-лічильники, аналогово-цифровий перетворювач мікроконтролера ATmega16, аналоговий компаратор, система переривань, універсальний асинхронний прийомопередавач UART. Розглянуто режими роботи та система команд. Багато уваги приділено організації взаємодії мікропроцесора з об`єктом керування та оператором.

Видання призначено для студентів технічних вузів, може бути використано також студентами середніх професійних навчальних закладів при вивченні сучасних мікроконтролерів. Цікавим є для інженерно-технічних працівників, що займаються проектуванням мікропроцесорної техніки.

УДК 004.318:621.3.049.77](075.8)

ISBN 978-617-7301-37-9

ЗМІСТ

ВСТУП

Високі функціональні можливості і експлуатаційні характеристики мікропроцесоров і мікроконтролерів, такі як малі габарити, маса, потужність, що споживається, значно поширили області їхнього застосування. Перевагою мікропроцесорних пристроїв керування і обробки інформації є їх гнучкість: систему, розроблену для виконання конкретної задачі керування, легко пристосувати для вирішення інших задач шляхом зміни програмного забезпечення.

Розширення функцій мікропроцесорних пристроїв призвело до необхідності вдосконалення знань спеціалістів різних профілей в цьому напрямку. Тому вивчення основ побудови мікропроцесорних пристроїв є необхідною складовою підготовки спеціалістів з електронної техніки.

Мета даного навчального посібника є освоєння однокристального мікроконтролера (МК) та проектування цифрових та мікропроцесорних систем на його основі. В якості МК обрані сучасні мікроконтролери серії AVR. Оскільки AVR Flash мікроконтролери доступні в різних версіях (у 8..100-вивідних корпусах), то вони чудово підходять в різні додатки, починаючи від управління побутовими пристроями до складних систем керування електроприводами Мікроконтролери AVR все частіше використовуються замість спеціалізованих інтегральних схем в різних галузях техніки. Вони містять всі необхідні функції і в більшості випадків забезпечують більш високу гнучкість і меншу вартість.

Більш швидке просування цих МК на ринок при мінімальних витратах забезпечується:

- Гнучкістю: Flash пам'ять дозволяє використовувати один і той же мікроконтролер в декількох додатках і досить просто виконати оновлення програмного забезпечення в процесі експлуатації.
- Масштабованістю: сумісність за програмним кодом дозволяє адаптувати існуюче програмне забезпечення для іншого представника сімейства AVR, відповідного вимогам додатка.

- Технічною підтримкою: рекомендації щодо застосування, опорна розробка і канали кваліфікованої технічної підтримки забезпечують безперешкодність проектування

- Прийнятним набором функцій: Серед безлічі мікроконтролерів, починаючи від ATtiny до ATMega, розробник зможе знайти оптимальне рішення під конкретні вимоги до системи управління.

Посібник виконаний у відповідності з навчальною та робочою програмою курса «Мікропроцесорні пристрої керування та обробки інформації», що читається в Національному тенхнічному унівеститеті України НТУУ «КПІ ім. І. Сікорського» на факультеті електроніки. Під час вивчення курсу студенти набувають знань про архітектуру, принципи побудови, функціонування та програмування мікропроцесорів і мікроконтролерів. Як результат студент повинні змогти виконати розробку мікроконтролерної системи, що реалізує функції збору інформації в системі, вводу дискретних та аналогових величин, формування опорних аналогових сигналів, формування статичних та імпульсних цифрових сигналів, виводу даних на семисегментний індикатор, роботи із клавіатурою, роботи з послідовними інтерфейсами.

З метою кращого сприйняття матеріалу у підручнику наведено приклади складання програм та проектування мікропроцесорних систем, після кожного підрозділу – контрольні питання.

Підручник розраховано на студентів бакалаврату “Електроніка”, “Електротехніка” та інших бакалавратів, програмою яких передбачено вивчення дисципліни “ Мікропроцесорні пристрої”. Книга також може бути корисною для спеціалістів, що працюють у галузі електроніки.

Відгуки і побажання прохання надсилати за адресою:

03056, Київ-56, вул. Політехнічна, 16, к.313

Тел. (044) 441-13-53

Факс (044) 236-96-76

E-mail: *tereshhenko50@bk.ru*

РОЗДІЛ 1. ПРИНЦИПИ ПОБУДОВИ МІКРОПРОЦЕСОРНИХ СИСТЕМ

1.1. Основні поняття та визначення мікропроцесорної техніки. Загальна характеристика і класифікація мікропроцесорних комплектів

Основні поняття і визначення

Мікропроцесор (МП) – це пристрій, який здійснює приймання, обробку і видачу інформації. Конструктивно МП містить одну або кілька інтегральних схем і виконує дії за програмою, записаною в пам'яті.

Мікропроцесорна система – обчислювальна, контрольно-вимірювальна або керуюча система, в якій основним пристроєм обробки інформації є МП. Мікропроцесорна система будується з набору великих інтегральних схем (ВІС).

Мультимікропроцесорна, або мультипроцесорна система – система, яка утворюється шляхом об'єднання деякої кількості універсальних або спеціалізованих МП, завдяки чому забезпечується паралельна обробка інформації і розподілене керування.

Мікропроцесорний комплект (МПК) – сукупність ВІС, сумісних за електричними, інформаційними та конструктивними параметрами і призначених для побудови електронно-обчислювальної апаратури і мікропроцесорних систем керування. Типовий склад МПК: ВІС МП (один чи кілька корпусів інтегральних схем); ВІС оперативних запам'ятовувальних пристроїв (ОЗП); ВІС постійних запам'ятовувальних пристроїв (ПЗП); інтерфейси або контролери зовнішніх пристроїв; службові ВІС (тактовий генератор, регістри, шинні формувачі, контролери шин, арбітри шин, тощо).

Задачі мікропроцесорних систем керування електронними аппаратами можна поділити на два типи.

1. Традиційні, які вирішувалися і системами керування на жорсткій логіці:

- задачі регулювання та стабілізації - реалізація функцій регуляторів (пропорційного (П), пропорційно-інтегрального (ПІ), пропорційно-інтегрально-диференційног ПІД);
- задачі формування сигналів керування напівпровідниковими елементами, їх розподілу та синхронізації з мережею або задавальним генератором;
- задачі формування сигналів з імпульсною модуляцією.

2. специфічні задачі, які не вирішувалися традиційними системами:

- задачі обчислювального модуля в колі зворотного зв‘язку перетворювача;
- комбіновані системи з вирішенням задач обчислення та зміни параметрів регулятора (прогнозування, ідентифікация параметрів навантаження та перетворювачів, реалізація виконання умов оптимального за деяким критерієм керування);
- діагностика стану перетворювачів та навантеження, виявлення передаварійних станів, реалізація ресурсозберігаючого керування;
- створення мультимікропроцесорних систем, які вирішують задачу узгодженого керування перетворювачами в рамках електротехнологічного об‘єкту;
- приймання та передавання сигналів керування іншим мікропроцесорним системам або ПК вищего рівня ієрархії;
- організація зв‘язку з оператором (підкючення клавіатури, різних типів індикаторів та дисплеїв, зуммерів).

Переваги та недоліки у порівнянні із системами керування на жорсткій логіці

Перевагами є велика гнучкість систем, тобто легкість внесення корректив шляхом модифікації програмного забезпечення, та можливість вирішувати нові задачі керування, наприклад, обчислення прогнозів, реалізація оптимального керування, керування з ідентифікацією навантаження, організації взаємодії із системою керування вищего ступеню керування або іншими системами того ж рівня.

Недоліками є обмеження реалізації мікропроцесорних алгоритмів за часом, який визначається частотою переключеннявентилей перетворювача, а також внесення затримки в коло зворотного з‘язку перетворювача та наявність дискретизації за рівнем керуючого впливу. Останній фактор перетворює лінійну характеристику керування на нелінійну, рис. 1.1. Це негативно впливає на стійкість та керованість й спостежлевість. Показано, що область стійкої роботи широтно-імпульсного перетворювача з ПІ регулятором на базі 8-розрядного мікропроцесорра зменшилася майже в три рази. Із збільшенням розрядності мікропроцесора негативний вплив дискретизаціїї значно поменшується – при реалізації ПІД-регулятора на 16-розрядних мікропроцесорах вплив дискретизації практично відсутній.

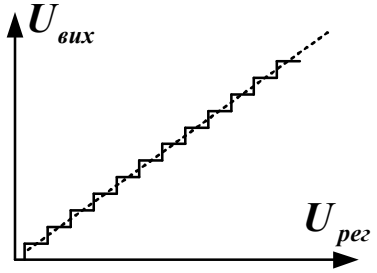


Рис. 1.1. Вплив дискретизації за рівнем на регулювальну характеристику напівпровідникового перетвоювача

Особливостю розробки мікропроцесорних систем керування напівпровідниковими перетворювачами є необхідність системного підхіда

до проектування системи розробки в цілому. Розробка апаратної частини мікропроцесорної системи, як правило, не викликає труднощів, оскільки система будується по жортско заданим правилам - принципам магістральності та модульності. Найбільш трудомісткім є етап проектування програмного забезпечення (розроблення схемотехніки повинні опанувати знання програмування на мовах низького рівня – асеемблері).

Якщо цільова функція мікропроцесорної системи керування (МСК) сформульована, тобто задача на розробку поставлена, то для одержання тексту вихідної програми необхідно виконати ряд послідовних дій:

1) докладний опис задачі;

2) аналіз задачі;

3) інженерну інтепретацію задачі, бажано з залученням того або іншого апарата формалізації (граф автомата, мережі Петрі, матриці станів і зв’язністю і т.п.);

4) розробку загальної блок-схеми алгоритму (БСА) роботи контролера;

5) розробку деталізованих БСА окремих процедур, виділених на основі модульного принципу упорядкування програм;

6) детальну проробку інтерфейсу контролера і внесення виправлень у загальну і деталізовані БСА;

7) розподіл робочих регістрів і пам'яті МСК;

8) формування тексту вихідної програми.

У результаті роботи пунктам 1-3 даного переліку одержують так називану функціональну специфікацію прикладної програми МСК, у котрої основну увагу приділяється деталізації способів формування вхідної і вихідної інформації.

На мові схем алгоритмів розроблювач описує метод, обраний для рішення поставленої задачі. Досить часто буває, що та сама задача може бути вирішена різними методами. Спосіб рішення задачі, обраний на етапі

її інженерної інтерпретації, на основі якого формується БСА, визначає не тільки якість розроблювальної прикладної програми, але і якісні показники кінцевого виробу.

Класифікація мікропроцесорів та мікропроцесорних комплектів

Мікропроцесори та мікропроцесорні комплекти класифікують за такими ознаками: галуззю застосування, призначенням; розрядністю даних; кількістю ВІС; типом архітектури; типом системи команд; кількістю ядер.

За ***галуззю застосування*** МП поділяють на МП електронно-обчислювальних систем (МП з архітектурою IA-32, IA-64, як-то Pentium, Itanium) та МП вбудованих систем керування (MCS51, AVR мікроконтролери).

За ***призначенням*** МП поділяють на універсальні та спеціалізовані. *Універсальні мікропроцесори* – МП загального призначення, які дають змогу розв‘язати широкий клас задач – обчислення, обробки та керування. *Спеціалізовані мікропроцесори* призначені для розв’язання задач лише певного класу. До них належать сигнальні, медійні, мультимедійні МП, трансп’ютери.

Сигнальні процесори призначені для цифрової обробки сигналів у реальному масштабі часу (наприклад, фільтрації сигналів, обчислення згортки, обчислення кореляційної функції, підсилення, обмеження і трансформації сигналу, пряме та зворотне перетворення Фур‘є). До сигнальних процесорів належать процесори фірм Texas Instruments – TMS320C80, Analog Devices – ADSP2106x, Motorola – DSP560xx та DSP9600x. Зазначимо, що DSP– процесори називають ще процесорами сигналів на відміну від CISC, RISC – мікропроцесорів - процесорів подій

Медійні і мультимедійні процесори призначені для обробки аудіо сигналів, графічної інформації, відео зображень, а також для розв’язання ряду задач у мультимедіа комп‘ютерах, іграшкових приставках, побутовій

техніці. До медійних і мультимедійних належать процесори фірм MicroUnity – Mediaprocessor, Philips – Trimedia, Cromatic Reserch – Mpact Media Engine, Nvidia – NV1, Cyrix – MediaGX.

Трансп'ютери призначені для масових паралельних обчислень і роботи у мультипроцесорних системах. Для них характерним є наявність внутрішньої пам'яті і вбудованого міжпроцесорного інтерфейсу, тобто каналів зв'язку з іншими ВІС МП. До трансп'ютерів належать процесори фірм Inmos – T-2, T-4, T-8, T9000.

За ***розрядністю даних*** виділяють 8 розрядні (i8080), 16-розрядні (i8086, i80286, i80386, i80486, Pentium) та 64-розрядні (Itanium, AMD64) процесори.

За ***кількістю ВІС у МПК*** розрізняють багатокристальні МПК і однокристальні мікроконтролери (ОМК). Багатокристальні комплекти – це МПК з однокристальними і секційними МП.

Однокристальний МП, або *мікропроцесор із фіксованою розрядністю даних* – це конструктивно завершений виріб у вигляді однієї ВІС. До цього типу належать процесори фірм Intel – Pentium (P5, P6, P7), AMD – K5, K6, Cyrix – 6x86, Diqital Equipment – Alpha 21064, 21164A, Silicon Graphics – MIPS R10000, Motorola – Power PC 603, 604, 620, Hewlett Packard – PA-8000, Sun Microsystems – Ultra SPARC II.

У *секційних МП* в одній ВІС реалізується лише деяка функціональна частина (секція) процесора. Інша назва секційних МП – *розрядно-модульні мікропроцесори* або *мікропроцесори з нарощенням розрядності.* Секційність ВІС МП обумовлює значну гнучкість МПС, можливість нарощення розрядності даних, створення специфічних технологічних команд із набору мікрокоманд. До секційних належать МП серій К589, К1804.

Однокристальний мікроконтролер – пристрій, що конструктивно виконаний в одному корпусі ВІС і містить основні складові частини МПК. До таких мікроконтролерів належать ОМК фірм Intel – MCS-196/296,

MicroChip – PIC17C4x PIC17C75x, Mitsubishi Electric – M3820, Motorola – MC33035, MC33039, STMicroelectronics – ST7Lite2, Advanced RISC Machines – ARM7TDMI.

За ***типом архітектури***, або принципом побудови, розрізняють МП з *фоннеймановською* архітектурою (наприклад, MCS-51) і МП з *гарвардською* архітектурою (AVR-мікроконтролери).

За ***типом системи команд*** розрізняють процесори з повним набором команд – *CISC* (Complete Instruction Set Computing) і процесори зі зменшеним набором команд – *RISC* (Reduced Instruction Set Computing). Як правило, *CISC*-процесори виконуються за фоннеймановською архітектурою, а *RISC*-процесори – за гарвардською.

За ***кількістю ядер*** МП поділяють на одноядерні та багатоядерні. В *одноядерних* МП є можливість псевдо-одночасного виконання різних задач - в такому випадку процесор по черзі виконує обчислення, але при такому підході можливий варіант вичерпування ресурсів процесору, що призводить до зниження обчислювальної потужності МП. *Багатоядерні* МП (наприклад, Intel Core 2 Duo) забезпечують можливість одночасного виконання різних обчислень, оскільки кожне ядро має власну кеш-пам’ять, тому операційна система має досить ресурсів для паралельного виконання обчислень, що потребують великої обчислювальної потужності.

Слід зазначити, що багато МПК підпадають під різні класифікаційні ознаки, оскільки здатні вирішувати задачі різних класів.

Так, існують універсальні МП з мультимедійним розширенням наборів команд, наприклад, Pentium MMX, Pentium П, Cyrix 6x86MX, AMD K6, Ultra SPARC. У CISC-процесорах Pentium PRO реалізовано ядро з RISC-архітектурою.

Напрямки розвитку архітектури процесорів: RISC та CISC архітектури

Поняття *архітектури* мікропроцесора визначає його складові частини та зв’язки та взаємодію між ними. Архітектура містить: 1) структурну схему самого МП; 2) програмну модель МП (описання функцій регістрів); 3) інформацію про організацію пам'яті (ємність пам’яті та способи її адресації); 4) опис організації процедур введення-виведення.

Існують два основних типи архітектури – фоннеймановська та гарвардська. *Фоннеймановську* архітектуру (рис. 1.2,а) запропонував 1945 року американський математик Джо фон Нейман. Особливістю цієї архітектури є те, що програма і дані знаходяться у спільній пам’яті, доступ до якої здійснюється по одній шині даних і команд.

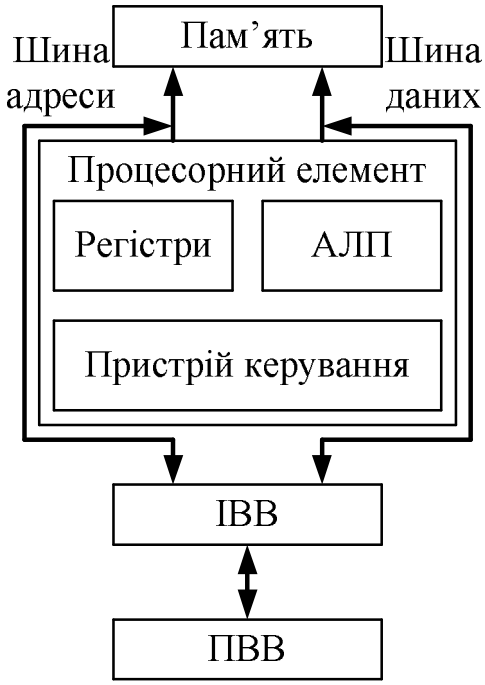


Рис. 1.2 Основні типи архітектури: а – фоннеймановська; б – гарвардська

Гарвардську архітектуру вперше реалізовано у 1944 році в релейній обчислювальній машині Гарвардського університету (США). Особливістю цієї архітектури є те, що пам’ять даних і пам’ять програм розділені і мають окремі шину даних і шину команд (рис. 1.2,б), що дозволяє підвищити швидкодію МП системи.

Структурні схеми МП обох архітектур містять: процесорний елемент, пам’ять, інтерфейси введення/виведення (ІВВ) і ПВВ. Пам’ять і ІВВ для різних типів МП можуть бути як внутрішніми, тобто розміщуватися на тому ж кристалі, що і процесорний елемент, так і зовнішніми. Процесорний елемент містить регістри, арифметично-логічний пристрій (АЛП), пристрій керування і виконує функції оброблення даних та керування процесами обміну інформацією. Пам’ять забезпечує зберігання кодів команд програми і даних. Інтерфейси призначені для зв’язку з ПВВ (наприклад, з клавіатурою, дисплеєм, друкувальними пристроями, датчиками інформації). Усі елементи структурної схеми з’єднані за допомогою шин.

Обґрунтування вибору базового сімейства процесорів

Вибір мікропроцесорів залежить від вимог по швидкодії (в залежності від частоти роботи перетворювача, обсягу обчислень при виробленні керуючого впливу, необхідними апаратними ресурсами мікропроцесора або мікроконтролера (наявністю вбудованих АЦП, мультиплексорів, компараторів, ШІМ, таймерів, регістрів захоплення подій, інтерфейсів зв‘язку та кількості ліній портів для введення-виведення сигналів.

1.2. Принципи побудови мікропроцесорних систем. Загальний вигляд структурної схеми мікропроцесорної системи. Організація шин. Склад системної шини

В основу побудови мікропроцесорних систем (МПС) покладено три принципи: 1) магістральності; 2) модульності; 3) мікропрограмного керування.

Принцип магістральності визначає характер зв'язків між функціональними блоками МПС – усі блоки з'єднуються з єдиною системною шиною.

Шина – це інформаційний канал, який об'єднує всі функціональні блоки МПС і забезпечує обмін даними у вигляді двійкових чисел. Конструктивно шина являє собою n провідників та один спільний провідник (землі). Дані по шині передаються у вигляді слів, що є групою бітів.

У паралельній шині n бітів передаються по окремих лініях одночасно, у послідовній шині – по єдиній лінії послідовно у часі. Паралельні шини виконують у вигляді плоского кабелю, а послідовні – у вигляді коаксіального або волоконно-оптичного кабелю. Коаксіальний кабель використовують при передачі даних на відстань до 100 метрів, узгоджуючи передавальні і приймальні каскади із хвильовим опором лінії. Волоконно-оптичний кабель використовують для передачі на більші відстані.

Усі основні блоки МПС з'єднують з єдиною паралельною шиною, яка називається системною шиною SB (System Bus). Системна шина містить три шини: адреси, даних і керування.

Шина адреси AB (Address Bus) є однонапрямленою. Вона призначена для передавання адреси комірки пам'яті або пристрою введення-виведення. Напрям передавання по шині адреси – від МП до зовнішніх пристроїв. Варіанти умовних позначень однонапрямленої паралельної шини показано на рис.1.3, на якому стрілка вказує напрям передавання.

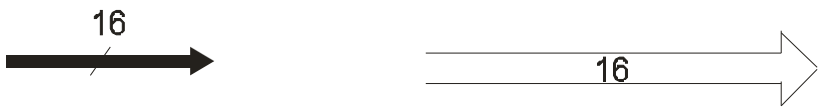


Рис. 1.3 Варіанти умовних позначень однонапрямленої паралельної 16-розрядної шини

Число 16 на рис 1.3 позначає розрядність шини. Зазначимо, що допускається позначення шин і без наведення розрядності.

Шина даних DB (Data Bus) є двонапрямленою. Вона призначена для передавання даних між блоками МПС. Інформація по одних і тих самих лініях DB може передаватися у двох напрямах – як до МП, так і від нього. Варіанти умовних позначень двонапрямленої шини показано на рис.1.3.

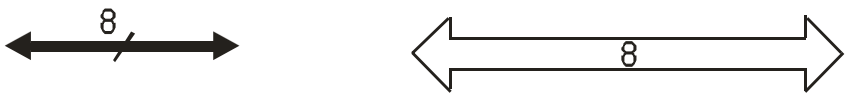


Рис. 1.4 Варіанти умовних позначень двонапрямленої паралельної 8-розрядної шини

Шина керування CB (Control Bus) призначена для передавання керуючих сигналів. Шина керування позначається так само, як і шина адреси (див. рис. 1.3). Хоча напрям керуючих сигналів може бути різним, однак шина керування не є двонапрямленою, оскільки для сигналів різного напряму використовуються окремі лінії.

Як приклад на рис. 1.5 показано структурну схему передавання інформації між *m* регістрами по внутрішній *n*-розрядній шині даних з урахуванням прийнятих позначень, а на рис. 1.6 – розширену структурну схему.

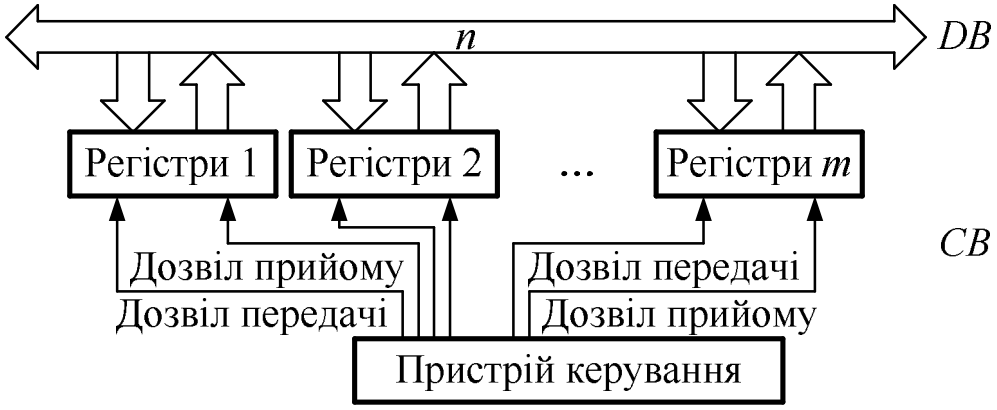


Рис. 1.5. Структурна схема передавання інформації між *m* регістрами по внутрішній *n*-розрядній шині даних

Дані по шині з *n* ліній передаються в режимі розподілу часу. Пристрій керування в кожний момент часу визначає адресу регістра, який передає інформацію, та регістра, який приймає інформацію. Для цього пристрій керування генерує сигнали *Дозвіл передачі* і *Дозвіл прийому*, що передаються по лініях шини керування *СВ*. Лінії шини і сигнали керування мають назви *Дозвіл передачі* і *Дозвіл прийому*. У кожний момент часу передавати інформацію в шину може тільки один регістр. Це означає, що у разі надходження сигналу *Дозвіл передачі* до шини приєднується тільки один модуль (рис. 1.6).

Вхідні лінії регістрів з'єднані безпосередньо з відповідними лініями шини. Тому під час подання сигналу *Дозвіл прийому*, який надходить по окремій лінії для кожного регістра, дані передаються по шині у відповідний регістр. Вихідні лінії регістрів з'єднуються з відповідними лініями шини через ключі *S,* що допускають монтажну логіку. Сигнал *Дозвіл передачі* надходить на ключі від пристрою керування по окремій для кожного регістра лінії.

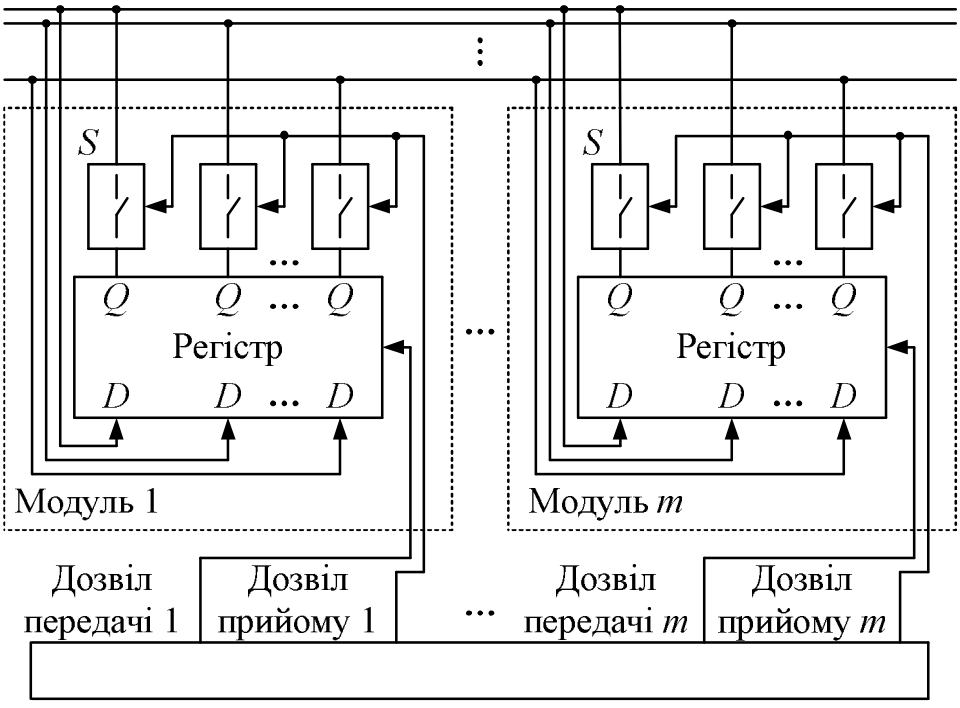


Рис. 1.6. Розширена структурна схема передавання інформації між *m* регістрами по внутрішній *n*-розрядній шині даних

Модуль, який братиме участь в обміні інформацією, визначається за одним з таких способів: 1) через відповідні лінії шини керування, окремі для кожного модуля; 2) за допомогою k ($k = \log_2 m$) ліній шини адреси, по яких передається ідентифікаційний код, що ставиться у відповідність кожному модулю й однозначно його визначає; 3) використанням одних і тих самих ліній шини даних для передавання адрес і даних.

Дані по шині можуть передаватися у двох режимах: синхронному й асинхронному. У синхронному режимі пристрій керування визначає модулі, що беруть участь в обміні інформацією, синхронізує роботу модулів та керує процесом обміну, виробляючи відповідні сигнали керування і синхронізації. В асинхронному режимі модулі, готові до обміну, ініціюють процес передавання та прийняття інформації, виробляючи відповідні сигнали готовності.

Принцип модульності полягає в тому, що система будується на основі обмеженої кількості типів конструктивно і функціонально завершених модулів. Кожний модуль МПС системи має вхід керування третім (високоімпедансним) станом. Цей вхід називається $\overline{CS}$ *(Chip Select)* – вибір кристала або $\overline{OE}$ (*Output Enable*) – дозвіл виходу.

Дію сигналу $\overline{CS}$ для тригера показано на рис.1.7. Вихідний сигнал тригера Q з'явиться на виводі лише за активного (у цьому випадку – нульового) рівня сигналу $\overline{CS}$. Якщо $\overline{CS} = 1$, тригер переводиться у високоімпедансний стан.

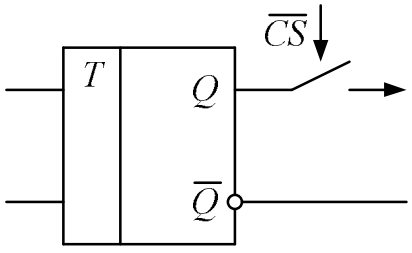


Рис. 1.7 Дія сигналу $\overline{CS}$ для тригера

Вихід тригера є *тристабільним*, тобто може знаходитися у одному з трьох станів: логічної одиниці, логічного нуля або у високоімпедансному.

У кожний момент часу до системної шини МПС під’єднано лише два модулі – той, що приймає, і той, що передає інформацію. Інші знаходяться у високоімпедансному стані.

Принципи магістральності і модульності дозволяють нарощувати керуючі і обчислювальні можливості МП через під’єднання інших модулів.

Принцип мікропрограмного керування полягає у можливості здійснення елементарних операцій – мікрокоманд (зсуву, пересилки інформації, логічних операцій). Певною комбінацією мікрокоманд можна створити набір команд, який максимально відповідатиме призначенню системи, тобто створити *технологічну мову*. У секційних процесорах набір мікрокоманд можна змінити, використовуючи інші мікросхеми пам’яті мікрокоманд.

Узагальнену структурну схему МПС фоннеймановського типу зображено на рис. 1.8. До складу МПС входять: 1) центральний процесор (ЦП), 2) ПЗП, 3) ОЗП; 4) система переривань, 5) таймер, 6) ПВВ. Пристрої введення-виведення під’єднані до системної шини через інтерфейси введення-виведення.

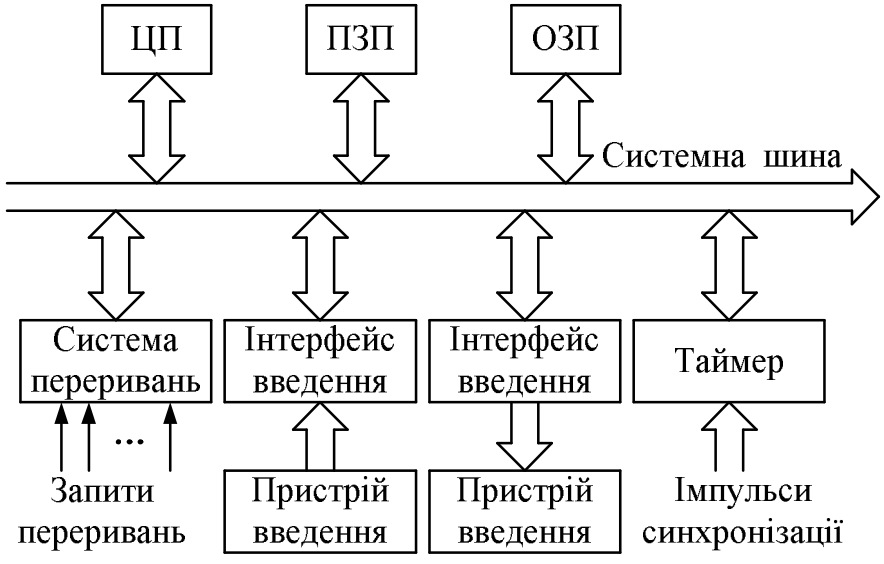


Рис.1.8 Узагальнена структурна схема мікропроцесорної системи керування

Постійний та оперативний запам’ятовувальні пристрої складають *систему пам’яті,* яка призначена для збереження інформації у вигляді двійкових чисел. Постійний запам’ятовувальний пристрій призначений для збереження програм керування таблиць, констант, а операційний запам’ятовувальний пристрій – для збереження проміжних результатів обчислень. Пам’ять організовано у вигляді масиву комірок, кожна з яких має свою адресу і містить байт або слово. Байтом називається група із 8 біт, а слово може мати будь-яку довжину в бітах. Під словом найчастіше розуміють двійкове число завдовжки два байти.

Для звернення до комірки пам’яті необхідно видати її адресу на шину адреси, на шині керування сформувати сигнали читання чи запису у пам‘ять, а по шині даних – ввести або вивести вміст комірки пам‘яті із заданою адресою. На рис. 1.9 зображено структуру пам’яті з 8 однобайтових комірок, де кожній адресі відповідає певний вміст комірки. Так, комірка з адресою 000 має вміст $01011111_2 = 5F_{16}$.

Адреса	Дані
000	01011111
001	00010011
010	01110111
011	00001100
100	00000000
101	11111111
110	10101010
111	11110000

Рис. 1.9 Структура пам’яті з 8 однобайтових комірок

Модуль *центрального процесора* здійснює оброблення даних і керує усіма іншими модулями системи. Центральний процесор, крім ВІС МП, містить схеми синхронізації та інтерфейсу із системною шиною. Він

вибирає коди команд з пам’яті, дешифрує їх і виконує. Впродовж часу виконання команди – командного циклу, ЦП виконує такі дії:

− виставляє адресу команди на шину адреси АВ;

− отримує код команди з пам’яті та дешифрує його;

− обчислює адреси операнда і зчитує дані;

− виконує операцію, визначену командою;

− сприймає зовнішні керуючі сигнали, (наприклад, запити переривань);

− генерує сигнали стану і керування, необхідні для роботи пам’яті та ПВВ.

Пристрої введення-виведення, або *зовнішні пристрої,* – це пристрої, призначені для введення інформації у МП або виведення інформації з нього. Прикладами ПВВ є дисплеї, друкувальні пристрої, клавіатура, цифро-аналоговий та аналого-цифровий пристрої, реле, комутатори. Для з’єднання ПВВ із системною шиною їх сигнали повинні відповідати певним стандартам. Це досягається за допомогою інтерфейсів введення-виведення.

Інтерфейси введення-виведення, або *контролери* чи *адаптери*, виконують функцію узгодження пристроїв введення-виведення із сигналами системної шині МПС. Мікропроцесор звертається до інтерфейсів за допомогою спеціальних команд введення-виведення. При цьому МП виставляє на шину адреси АВ адресу інтерфейсу, а по шині даних DB зчитує дані з пристрою введення або записує у пристрій виведення. На рис. 1.5 показано один інтерфейс введення і один інтерфейс виведення.

Система переривань дозволяє МПС реагувати на зовнішні сигнали – запити переривань, джерелами яких можуть бути: сигнали готовності від зовнішніх пристроїв, сигнали від генераторів, сигнали з виходів датчиків. З появою запиту переривання ЦП перериває основну програму і переходить до виконання підпрограми обслуговування запиту переривання, а після

обробки запиту повертається в основну програму. Для побудови системи переривань МПК містять ВІС спеціальних програмованих контролерів переривань.

Таймер призначений для реалізації функцій, пов'язаних з відліком часу. Після того, як МП завантажує в таймер число, яке задає частоту, затримку або коефіцієнт ділення, таймер реалізує необхідну функцію.

1.3. Подання чисел у мікропроцесорах

Системи числення поділяють на позиційні і непозиційні. Серед позиційних систем розрізняють системи з безпосереднім або кодованим поданням чисел. *Розрядом* цифри називається місце (або позиція) цифри в позиційній системі. *Основою p* системи числення називають кількість різних цифр, які застосовуються для написання чисел: 0, 1,..., p – 1. На рис. 1.10 наведено класифікацію систем числення. Курсивом виділено подання десяткового числа 11 у різних системах числення.

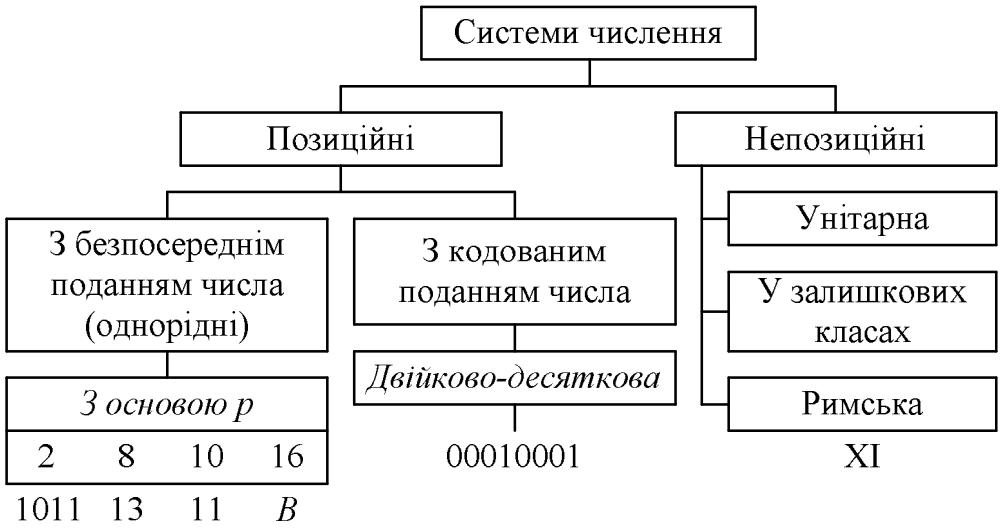


Рис. 1.10. Класифікація систем числення

До непозиційних систем числення належать: римська, унітарна і система залишкових класів.

У римській системі числення використовують такі цифри: $I = 1$, $V = 5$, $X = 10$, $L = 50$, $C = 100$, $D = 500$, $M = 1000$. У цій системі число 2016 записується як *MMXVI*. В унітарній системі число подають загальною сукупністю однорідних об'єктів. У системі залишкових класів числа подають остачами від ділення на прості числа. У цій системі всі операції можуть виконуватися окремо для цифр кожного розряду.

До позиційних систем числення відносять системи, у яких кожна цифра займає певне положення (розряд або позицію) у ряду цифр, що зображують число. Щоб одержати значення числа, потрібно кожну цифру розряду помножити на число, яке називається *вагою розряду*. Ваги окремих розрядів являють собою геометричну прогресію зі знаменником, що дорівнює основі системи числення p. Наприклад, розряди десяткового числа 1327,45 мають ваги: $10^3 = 1000$; $10^2 = 100$; $10^1 = 10$; $10^0 = 1$; $10^{-1} = 0{,}1$; $10^2 = 0{,}01$. Позиційні системи числення, в яких цифри всіх розрядів набувають значення 0,1,…, p–1, а основа p є однаковою для всіх розрядів, називають *однорідними*.

Подання числа X в однорідній позиційній системі числення з основою p має вигляд:

$$X = \sum_{s=1}^{n} x^{(s)} p^{n-s} = x^{(1)} p^{n-1} + x^{(2)} p^{n-2} + \ldots + x^{(n)} p^0 = x^{(1)} x^{(2)} \ldots x^{(n)}. \quad (1.1)$$

Число X, що містить n розрядів цілої частини і k розрядів дробу, можна виразити формулою

$$X = \sum_{s=1}^{n+k} x^{(s)} p^{n-s} = x^{(1)} p^{n-1} + x^{(2)} p^{n-2} + \ldots + x^{(n)} p^0 + x^{(n+1)} p^{-1} + \\ + x^{(n+2)} p^{-2} + \ldots + x^{(n+k)} p^{-k} = x^{(1)} x^{(2)} \ldots x^{(n)}, \; x^{(n+1)} x^{(n+2)} \ldots x^{(n+k)}. \quad (1.2)$$

Десяткова система або система з основою 10 ($p = 10$) оперує з 10 цифрами (від 0 до 9). У системах числення з основою більше 10 використовують 10 цифр для молодших значень цифр розрядів і латинські літери A, B, C... – для старших.

Якщо необхідно позначити основу системи числення, то використовують числові індекси або латинські літери: для двійкового числа індекс 2 або літера *B (Binary),* для десяткового – індекс 10 або літера *D (Decimal),* для шістнадцяткового – індекс 16 або літера *H (Hexadecimal).*

Приклад 1.1. Записати число 11_{10} у двійковій системі числення.

У двійковій системі числення основа $p = 2$, а цифри розрядів можуть набувати значень 0 або 1. Згідно з формулою (1.1) число 11_{10} у двійковій системі числення записують як

$$X = \sum_{s=1}^{4} x^{(s)} 2^{n-s} = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 1011_2 .$$

Приклад 1.2. Записати число 11_{10} у вісімковій системі числення.

У вісімковій системі числення основа p = 8, а розрядні компоненти можуть набувати значень {0, 1, 2, 3, 4, 5, 6, 7}. Згідно з формулою (1.1) число 11_{10} у вісімковій системі числення можна записати так:

$$X = \sum_{s=1}^{2} x^{(s)} 8^{n-s} = 1 \cdot 8^1 + 3 \cdot 8^0 = 13_8 .$$

Приклад 1.3. Записати число 11_{10} у шістнадцятковій системі числення.

Для шістнадцяткової системи p = 16, $x^{(i)}$ = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, *A*, *B*, *C*, *D*, *E*, *F*}. Згідно з формулою (1.1)

$$X = B \cdot 16^1 = 0B_{16} = 0BH .$$

***Приклад* 1.4.** Подати число $1327{,}45_{10}$ у вигляді полінома (1.2).

Значення ваг та цифр розрядів числа 1327,45 у десятковій системі числення ілюструє табл. 1.1 згідно з формулою (1.2):

$$1327{,}45 = 1 \cdot 10^3 + 3 \cdot 10^2 + 2 \cdot 10^1 + 7 \cdot 10^0 + 4 \cdot 10^{-1} + 5 \cdot 10^{-2}.$$

Таблиця 1.1.

Значення ваг та цифр розрядів у десятковій системі

Степінь основи	3	2	1	0	–1	–2	
Значення ваг позицій	1000	100	10	1	0,1	0,01	
Цифра розряду	1	3	2	7	4	5	
Число	1000	+ 300	+ 20	+ 7	+ 0,4	+ 0,05	$= 1327{,}45_{10}$

Розглянуті у прикладах системи, у яких кожній цифрі розряду відповідає окремий символ, називаються *системою з безпосереднім поданням чисел*.

Систему, в якій кількість символів менша, ніж кількість цифр, а кожну цифру кодують певною комбінацією кількох символів, називають *системою з кодованим поданням чисел*. Такою є, наприклад, двійково-десяткова система числення, яка містить десять цифр, але тільки два символи, причому кожна з десяти цифр кодується числами двійкової системи: 0 = 0000, 1 = 0001,…, 9 = 1001.

Приклад 1.5. Записати число 11_{10} у двійково-десятковій системі числення.

Для кожної цифри десяткового числа запишемо її двійковий еквівалент, який займає 4 розряди ($1_{10} = 0001_2$). Отже, число 11_{10} у двійково-десятковій системі буде подано таким чином:

$$11_{10} = 0001\ 0001_{2\text{-}10}.$$

Двійкова система числення

Двійкова система числення, або система з основою 2, використовує цифри 0 і 1. Такі цифри називаються *бітами* (***Bi****nary Digit****s***). Фізично в цифрових електронних системах значення 0 відповідає напрузі низького рівня (*L*-рівня), а значення 1 – напрузі високого рівня (*H*-рівня).

У табл. 1.2 наведено значення ваг перших чотирьох двійкових позицій і показано відповідність між двійковим числом 1001_2 та його десятковим еквівалентом 9_{10}.

Таблиця 1.2.

Значення позицій двійкових чисел

Степінь основи	3	2	1	0
Значення ваг позицій	8	4	2	1
Двійкове число	Старший біт 1	0	0	Молодший біт 1
Десяткове число	8	+ 0	+ 0	+ 1 = 9_{10}

Розряд, якому відповідає значення ваги позиції 1, називається *молодшим бітом*, а розряд, якому відповідає найбільше значення ваги позиції (у цьому разі 8), – *старшим бітом.*

У табл. 1.3 подано десяткові числа від 0 до 15 та їх двійкові еквіваленти.

Таблиця 1.3.

Двійкові еквіваленти десяткових чисел від 0 до 15

Числа											
десяткові		Двійкові				десяткові		Двійкові			
Значення ваг позицій											
10^1	10^0	2^3	2^2	2^1	2^0	10^1	10^0	8	4	2	1
0	0	0	0	0	0	0	8	1	0	0	0
0	1	0	0	0	1	0	9	1	0	0	1
0	2	0	0	1	0	1	0	1	0	1	0
0	3	0	0	1	1	1	1	1	0	1	1
0	4	0	1	0	0	1	2	1	1	0	0
0	5	0	1	0	1	1	3	1	1	0	1
0	6	0	1	1	0	1	4	1	1	1	0
0	7	0	1	1	1	1	5	1	1	1	1

Перетворення двійкового числа на десятковий еквівалент

Під кожним бітом двійкового числа записують десяткові значення кожної позиції. Десяткові числа підсумовують. Приклад перетворення двійкового числа $1011\ 0110_2$ на десятковий еквівалент наведено у табл. 1.4.

Таблиця 1.4.

Перетворення двійкового числа на десятковий еквівалент

Степінь основи	7	6	5	4	3	2	1	0	
Значення ваг позицій	128	64	32	16	8	4	2	1	
Двійкове число	1	0	1	1	0	1	1	0	
Десяткове число	128	+ 0	+ 32	+ 16	+ 0	+ 4	+ 2	+ 0	$= 182_{10}$

Перетворення десяткового числа на двійковий еквівалент

Десяткове число ділять на 2. Остачу у вигляді 0 або 1 записують у молодший розряд двійкового числа. Частку від ділення знов ділять на 2, остачу (0 або 1) записують у наступний після молодшого розряд. Ці дії

виконують доти, доки частка від чергового ділення не дорівнюватиме 1. Одиницю записують у старший розряд двійкового числа. Приклад перетворення десяткового числа 155_{10} на двійковий еквівалент 10011011_2 наведено на рис. 1.11.

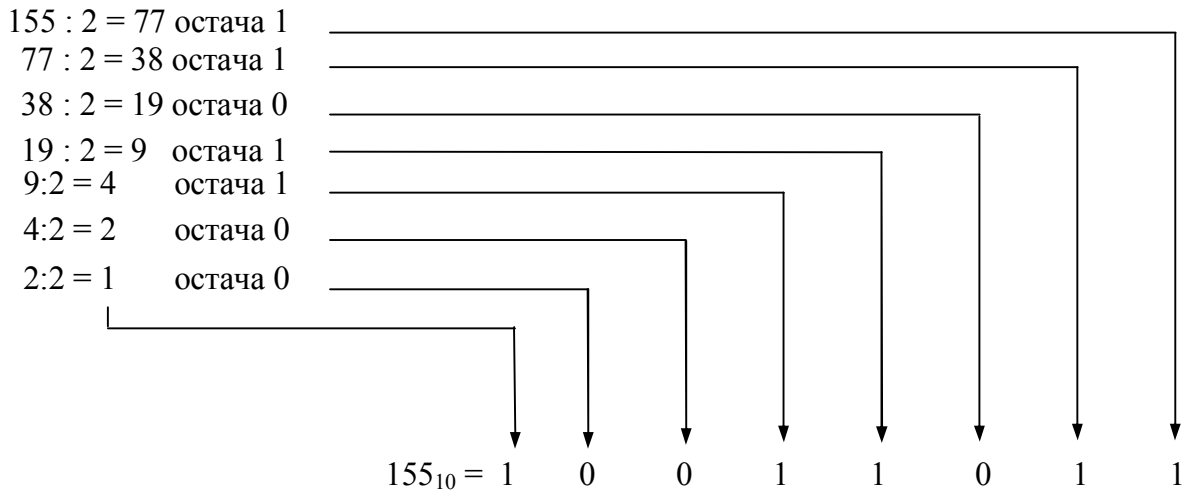


Рис. 1.11. Перетворення десяткового числа 155_{10} на двійковий еквівалент

Для перетворення цілого числа *X*, записаного в системі числення з основою *p*, на його еквівалент у системі числення з основою *q* слід ділити *X* на *q* до отримання цілої остачі, меншої від *q* (метод послідовного ділення). Число *X* у системі числення з основою *q* подається послідовністю остач ділення в порядку, зворотному їхньому одержанню, причому старшу цифру в числі *X* дає остання остача.

Для чисел, що мають як цілу, так і дробову частину, переведення з однієї системи числення в іншу здійснюється окремо для цілої і дробової частин. Для перетворення правильного дробу *X*, записаного у системі числення з основою *p,* на його еквівалент в системі числення з основою *q* слід послідовно множити *X* на *q,* причому множити слід тільки дробові частини (метод послідовного множення). Еквівалент *X* у системі числення з основою *q* подається у вигляді послідовності цілих частин результатів множення у порядку їхнього одержання, причому старший розряд є цілою частиною першого результату. Якщо необхідно виконати перетворення з точністю q^{-k}, то кількість послідовних множень дорівнює *k.*

***Приклад* 1.6.** Записати десятковий дріб $0{,}366_{10}$ з точністю 2^{-8} у двійковій системі числення.

Дробові частини заданого числа та чисел, що утворюються у результаті множення, 8 разів послідовно множимо на 2 (рис. 1.12).

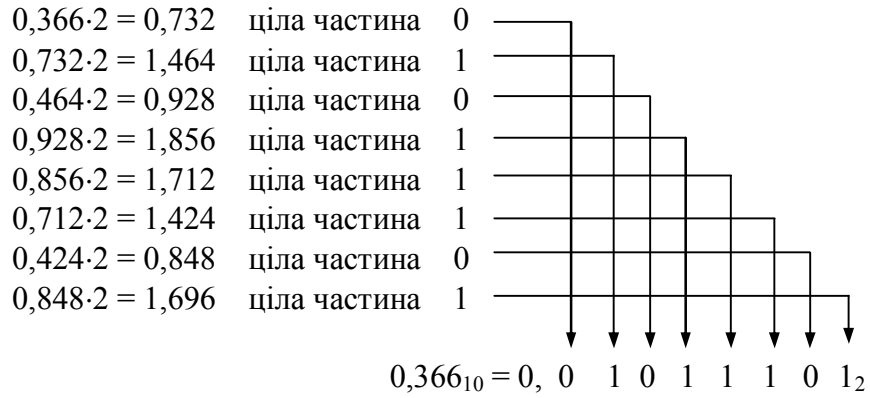


Рис. 1.12. Перетворення десяткового числа $0{,}366_{10}$ на двійковий еквівалент з точністю 2^{-8}

У табл. 1.5 наведено перетворення одержаного двійкового числа на десятковий еквівалент, де під кожним бітом двійкового числа записані десяткові значення кожної позиції, які підсумовують.

Отримане число $0{,}36328125_{10}$ наближує задане число 0,366 з точністю $2^{-8} = 0{,}0039$.

Таблиця 1.5.

Перетворення двійкового числа $0{,}01011101_2$ на десятковий еквівалент

Степінь основи	0	–1	–2	–3	–4	–5	–6	–7	–8
Значення ваг позицій	1	0,5	0,25	0,125	0,0625	0,03125	0,015625	0,0078125	0,00390625
Двійкове число	0	0	1	0	1	1	1	0	1
Десяткове число	0+0+0,25+0+0,0625+0,03125+0,015625+0+0,00390625 = 0,36328125								

Двійкова арифметика

Додавання, віднімання або множення двійкових чисел виконують за такими самими правилами, як і в арифметиці десяткових чисел. Правила додавання однорозрядних двійкових чисел наведено у табл. 1.6. Правила 1 та 2 очевидні; правило 3 ілюструє перенесення одиниці у старший розряд:

1 + 1 = 10. Правило 4 показує, що результатом додавання трьох 1 є число 11.

Таблиця 1.6.

Правила двійкового додавання

Номер правила	1	2	3	4
1-й доданок	0	0	1	1 (Перенесення з молодшого розряду) + 1
2-й доданок	+ 0	+ 1	+ 1	1
Сума	0	1	10	11
			Перенесення 1 до старшого розряду	Перенесення 1 до старшого розряду

Приклад 1.7. Додати два 8-розрядних двійкових числа: 10100010_2 і 01110101_2.

Виконуємо додавання двох чисел, використавши правила двійкового додавання (табл. 1.6) для кожного з розрядів. У цьому прикладі результат додавання 8-розрядних чисел є 8-розрядним:

```
  1   111   – рядок перенесень
  10100011
+ 00110101
----------
  11011000
```

Правильність додавання перевіримо в десятковій системі (табл. 1.7).

Таблиця 1.7.

Додавання у двійковій та десятковій системах

Значення ваг розрядів / Числа		2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1-й доданок	Двійкове	1	0	1	0	0	0	1	1
	Десяткове	128		+ 32				+ 2	+ 1 = 163_{10}
2-й доданок	Двійкове	0	0	1	1	0	1	0	1
	Десяткове			32	+ 16		+ 4		+ 1 = 53_{10}
Результат	Двійкове	1	1	0	1	1	0	0	0
	Десяткове	128	+ 64		+ 16	+ 8			= 216_{10}

Приклад 1.8. Додати два двійкових числа: 10100011_2 та 01110101_2.

Виконаємо додавання у двійковому вигляді. Результат є 9-розрядним числом:

$$\begin{array}{cccccccccc} 1 & 1 & 1 & & & & & 1 & 1 & \\ & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & \\ + & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & \\ \hline 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & \end{array}$$

Результат перевіримо додаванням у десятковому вигляді:

$$\begin{array}{rr} & 163 \\ + & 117 \\ \hline & 280 \end{array}$$

Якщо для позначення кожного двійкового числа відводиться вісім розрядів, то одержання 9-розрядного результату додавання двох чисел призводить до некоректної роботи мікропроцесора. У прикладі 1.8 у восьми молодших розрядах результату додавання у двійковому вигляді знаходиться число $00011000_2 = 24_{10}$. При цьому відбувається перенесення 1 у дев'ятий розряд, вага якого $2^8 = 256$. Це перенесення можна врахувати програмно і скоректувати неправильний результат, наприклад, розміщенням перенесення у додатковому регістрі. Тоді двобайтове число буде містити 256 + 24 = 280, тобто правильний результат.

У табл. 1.8 наведено правила віднімання однорозрядних двійкових чисел. Правила 1 – 3 аналогічні правилам десяткового віднімання. Правило 4 потребує позики зі старшого розряду так, що зменшуваним є число 10, від'ємником – 1 і різницею – 1.

Таблиця 1.8.

Правила двійкового віднімання

Номер правила	1	2	3	4
Зменшуване	0	1	1	1 0 ← 0 (Позика зі старшого розряду)
–	–	–		
Від'ємник	0	0	1	1
Різниця	0	1	0	1

***Приклад* 1.9.** Відняти від двійкового числа 10100011_2 число 00110101_2.

Виконуємо віднімання двох чисел, використавши правила двійкового віднімання (див. табл. 1.8):

$$\begin{array}{rll} 11111 & \text{– рядок позик} & \\ 10100011 & = & 163_{10} \\ -\ 00110101 & = & 53_{10} \\ \hline 01101110 & = & 110_{10} \end{array}$$

Відзначимо, що віднімання більшого числа від меншого приводить до некоректного результату, оскільки є потреба у позиці з дев'ятого розряду. Результат віднімання у цьому випадку треба скоректувати програмно.

У табл. 1.9 подано правила множення однорозрядних двійкових чисел.

Таблиця 1.9.

Правила двійкового множення

Номер правила	1	2	3	4
Множник	0	1	0	1
	×	×	×	×
Множник	0	0	1	1
Добуток	0	0	0	1

У перших трьох випадках принаймні один множник дорівнює 0, тому добуток дорівнює 0. У четвертому випадку множниками є одиниці, тому добутком є 1.

Приклад 1.10. Перемножити два двійкових числа: 1101_2 і 101_2.

Знайдемо добуток, використавши правила (див. табл. 1.9):

$$\begin{array}{rl} & 1101 \\ \times & 101 \\ \hline & 1101 \\ & 0000 \\ + & 1101 \\ & 1000001 \end{array}$$

Для перевірки подамо множники у десятковому вигляді: $1101_2 = 13_{10}$, $101_2 = 5_{10}$, результат множення $13 \times 5 = 65_{10} = 1000001_2$. Відзначимо, що розрядність добутку дорівнює сумі розрядів множників.

Шістнадцяткова система числення

Шістнадцяткова система числення є системою з основою 16 та містить 16 символів: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9*, A*, *B*, *C*, *D*, *E*, *F*. У табл. 1.15 наведено двійкові та шістнадцяткові еквіваленти 16 перших десяткових чисел.

Таблиця 1.15.

Двійкові та шістнадцяткові еквіваленти десяткових чисел

Десяткове число		Шістнадцятковий еквівалент	Двійковий еквівалент			
Значення ваг позицій						
10^1	10^0	16^0	2^3	2^2	2^1	2^0
0	0	0	0	0	0	0
0	1	1	0	0	0	1
0	2	2	0	0	1	0
0	3	3	0	0	1	1
0	4	4	0	1	0	0
0	5	5	0	1	0	1
0	6	6	0	1	1	0
0	7	7	0	1	1	1
0	8	8	1	0	0	0
0	9	9	1	0	0	1
1	0	*A*	1	0	1	0
1	1	*B*	1	0	1	1
1	2	*C*	1	1	0	0
1	3	*D*	1	1	0	1
1	4	*E*	1	1	1	0
1	5	*F*	1	1	1	1

Кожну шістнадцяткову цифру подають єдиною комбінацією чотирьох двійкових цифр. Так, шістнадцятковим еквівалентом двійкового числа 10011110_2 є число $9E_{16}$. Це означає, що старшу тетраду (4 старші розряди) 1001 двійкового числа записують як 9_{16}, а молодшу тетраду 1110 – як E_{16}.

Приклад **1.16.** Перетворити двійкове число 111010_2 на шістнадцятковий еквівалент.

Для перетворення двійкового числа на шістнадцятковий еквівалент треба поділити його на тетради, починаючи з наймолодшого розряду, а потім кожну тетраду

замінити еквівалентною шістнадцятковою цифрою. Значення молодшої тетради $1010_2 = A_{16}$, старшої – $0011_2 = 3_{16}$. Отже, $111010_2 = 3A_{16}$.

***Приклад* 1.17.** Перетворити шістнадцяткове число $7F_{16}$ на двійковий еквівалент.

Для перетворення шістнадцяткового числа на двійковий еквівалент кожну шістнадцяткову цифру слід замінити на двійковий еквівалент – тетраду (див. табл. 1.15). Еквівалентом шістнадцяткової цифри 7_{16} є двійкове число 0111_2, а цифри F_{16} – число 11112_2. Отже, $7F_{16} = 11110111_2$.

***Приклад* 1.18.** Перетворити шістнадцяткове число $2C6E_{16}$ на десятковий еквівалент.

Перетворення виконують згідно з табл. 1.16. Кожну цифру шістнадцяткового числа множать на відповідну вагу позиції. Сума цих добутків дає десяткове число 11374_{10}.

Таблиця 1.16.

Перетворення шістнадцяткового числа на десяткове

Значення позицій	16^3	16^2	16^1	16^0
Значення ваг позицій	4096	256	16	1
Шістнадцяткове число	2	3	6	*E*
Десяткове число	2×4096	$+3 \times 256$	$+ 6 \times 16$	$+1 \times 14 = 11374_{10}$

***Приклад* 1.19.** Перетворити десяткове число 15797 на шістнадцятковий еквівалент.

При перетворенні десяткове число 15797_{10} ділять на 16, що дає частку 987_{10} і остачу $5_{10} = 5_{16}$. Таким чином, молодший розряд шістнадцяткового числа має значення 5. Частка 987_{10} стає діленим і її знову ділять на 16, що дає частку 61_{10} і остачу $11_{10} = B_{16}$, яка стає значенням другого розряду шістнадцяткового числа. Ділення 61_{10} на 16 дає частку 3_{10} і остачу $13_{10} = D_{16}$. Ділення 3_{10} на 16 дає частку 0 і остачу $3_{10} = 3_{16}$. Оскільки остання частка дорівнює 0, то цифра 3_{16} стає значенням старшого розряду шістнадцяткового числа:

$15797_{10} : 16 = 987_{10}$ остача $5_{10} = 5_8$
$987_{10} : 16 = 61_{10}$ остача $11_{10} = B_{16}$
$61_{10} : 16 = 3_{10}$ остача $13_{10} = D_{16}$
$3_{10} : 16 = 0$ остача $3_{10} = 3_{16}$

3 *D B* 5

Отже, 1579710 = 3DB516.

Двійково-десяткова система числення

Двійково-десяткова система числення – система, у якій кожну десяткову цифру від 0 до 9 подають 4-розрядним двійковим еквівалентом. Така система числення дозволяє скоротити програмні та апаратні витрати при перетворенні двійкових чисел, які використовують під час обробки інформації у процесорі, на десяткові, що виводять на пристрої відображення. Ця система є ефективною і при перетворенні десяткових чисел на двійкові. Двійково-десяткові числа записують з індексом 2-10 або ДДК (двійково-десятковий код), наприклад $01001001_{2\text{-}10}$, $0100_{ДДК}$.

***Приклад* 1.20.** Перетворити десяткове число 3691_{10} на двійково-десятковий код.

При перетворенні кожну цифру десяткового числа перетворюють на двійковий 4-розрядний еквівалент:

Десяткове число	3	6	9	1
Двійково-десяткове число	0011	0110	1001	0001

Отже, $3691_{10} = 0011\ 0110\ 1001\ 0001_{2\text{-}10}$.

***Приклад* 1.21.** Перетворити двійково-десяткове число $1000000001110010_{2\text{-}10}$ на десятковий еквівалент.

Кожна тетрада двійково-десяткового числа перетворюється на десятковий еквівалент:

Двійково-десяткове число	1000	0000	0111	0010
Десяткове число	8	0	7	2

Отже, $1000\ 0000\ 0111\ 0010_{2\text{-}10} = 8072_{10}$.

Подання чисел у мікропроцесорах

У регістрах або комірках пам’яті МП інформацію розміщено у вигляді двійкових чисел, причому для кожного розряду числа відведено окрему комірку, що зберігає один біт інформації. Сукупність комірок, призначених для розміщення одного двійкового числа, називають

розрядною сіткою. Кількість комірок у розрядній сітці обмежена і залежить від конструктивних особливостей МП.

Залежно від способу обробки бітів, розміщених у розрядній сітці, розрізняють два види кодів: паралельний, коли в кожний момент часу всі розряди сітки доступні для обробки, і послідовний, коли в кожний момент часу доступний один розряд сітки. Числа, подані паралельним кодом, доступні за один такт, а числа, подані послідовним кодом, – за n тактів, де n – розрядність сітки. Якщо розрядність числа перевищує довжину сітки, то його обробка ведеться по частинах.

Натуральним кодом називають подання числа як цілого беззнакового у двійковій системі числення. Діапазон чисел у натуральному коді для n-розрядної сітки становить від 0 до $2^n - 1$, тобто для 8-розрядної сітки діапазон чисел у натуральному коді – від 0 до 255. Наприклад, натуральний код числа 53_{10} у 8-розрядній сітці наведено на рис. 1.13.

Натуральний код числа 53_{10} у 8-розрядній сітці

Розряди	*D*7	*D*6	*D*5	*D*4	*D*3	*D*2	*D*1	*D*0
Вага розрядів	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
53_{10}	0	0	1	1	0	1	0	1

Рис. 1.13. Натуральний код числа 53_{10} у 8-розрядній сітці

Для подання цілих знакових чисел використовують прямий, обернений і додатковий коди. Старший розряд сітки є знаковим. Значення цього розряду дорівнює 0 для додатних чисел і 1 – для від'ємних. В інших розрядах розміщується модуль числа. Додатні числа подаються однаково у всіх трьох кодах. Так, додатне число $+53_{10}$ має вигляд, поданий на рис. 1.13. Але оскільки старший розряд є знаковим, діапазон додатних чисел становить від 0 до $2^{n-1} - 1$. Наприклад, для 8-розрядної сітки діапазон додатних чисел – від 0 до +127. Подання від'ємних чисел залежить від використання того чи іншого коду.

Подання від'ємного числа *у прямому коді* здійснюється так. У старшому, знаковому, розряді розміщується 1, а в інших розрядах – модуль числа. Діапазон від'ємних чисел у прямому коді становить від 0 до $-(2^{n-1}-1)$. Для прикладу на рис. 1.14 наведено прямий код числа -53_{10} у 8-розрядній сітці, для якої діапазон від'ємних чисел – від 0 до –127 = 11111111.

Розряди	*D*7	*D*6	*D*5	*D*4	*D*3	*D*2	*D*1	*D*0
Вага розрядів	–	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	Знак числа	0	1	1	0	1	0	1
	1							

Рис. 1.14. Прямий код числа -53_{10} у 8-розрядній сітці

Перевагами прямого коду є простота виконання арифметичних операцій та однаковий діапазон значень додатних і від'ємних чисел.

Недоліками прямого коду є те, що операції додавання і віднімання чисел з різними знаками потребують додаткових операцій для визначення більшого за модулем числа та знака результату. Крім того, наявність знака при поданні числа 0 (+0 = 000000 і –0 = 1000000) потребує виконання зайвих операцій.

Подання від'ємного числа *у оберненому коді* здійснюється обчисленням числа, яке доповнює додатне число з тим самим модулем до найбільшого беззнакового числа, яке може бути розташоване в даній розрядній сітці. Одержання оберненого коду від'ємного числа зводиться до порозрядного інвертування розрядів додатного числа, включаючи знаковий розряд. Діапазон від'ємних чисел у оберненому коді становить від 0 до $-(2^{n-1}-1)$. Як приклад на рис. 1.15 показано обернений код числа -53_{10} у 8-розрядній сітці, для якої діапазон від'ємних чисел – від 0 до -127_{10}, при цьому оберненим кодом найменшого числа -127_{10} є число 10000000_2.

Розряди	*D*7	*D*6	*D*5	*D*4	*D*3	*D*2	*D*1	*D*0
Ваги розрядів	–	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	Знак числа	Модуль числа						
	1	1	0	0	1	0	1	0

Рис. 1.15. Обернений код числа -53_{10} у 8-розрядній сітці

Розглянемо віднімання двох чисел в оберненому коді на прикладі чисел $+163_{10}$ і $+53_{10}$. Подамо різницю у вигляді:

$$125_{10} - 53_{10} = 125_{10} + (255_{10} - 53_{10}) - 255_{10}. \qquad (1.3)$$

Вираз $(255_{10} - 53_{10})$ є оберненим кодом від'ємного числа -53_{10}, тобто доповненням до найбільшого числа $255_{10} = 11111111_2$, що можна розмістити у 8-розрядній сітці. З виразу (1.3) випливає, що операцію віднімання можна замінити операцією додавання в оберненому коді з подальшим коригуванням результату (відніманням 255_{10}). Виконання операції додавання числа 125_{10} та числа -53_{10}, поданого в оберненому коді, показано на рис. 1.16.

Рядок перенесень	1	1	1	1	1				
$+125_{10}$	+	0	1	1	1	1	1	0	1
$255_{10}-53_{10}$		1	1	0	0	1	0	1	0
$+71_{10}$		0	1	0	0	0	1	1	1

Рис. 1.16. Виконання операції додавання числа 125_{10} та числа -53_{10}, поданого в оберненому коді

Під час додавання виникає перенесення у неіснуючий дев'ятий розряд, тобто переповнення розрядної сітки. Це зменшує отриманий результат на 256_{10}. Оскільки згідно з виразом (1.3) результат треба зменшити на 255_{10}, для правильного результату потрібно до одержаної суми додати одиницю. Після такого коригування відповідь становитиме $+71_{10} + 1_{10} = +72_{10} = +01001000_2$, що відповідає виразу (1.3). Коригування результату досягається апаратними засобами – реалізацією додавання до молодшого біта значення перенесення, яке виходить за розрядну сітку. У цьому прикладі має місце перенесення одиниці в неіснуючий дев'ятий

розряд. Нульове значення знакового розряду результату означає, що різниця чисел додатна.

Перевагами оберненого коду є простота операцій одержання та додавання чисел із різними знаками, недоліками – два подання нуля: +0 = 00000000 і –0 = 111111111, а також потреба в апаратній реалізації коригування результату.

Подання від'ємного числа *у додатковому коді* здійснюється обчисленням числа, яке доповнює додатне число з тим самим модулем до найбільшого беззнакового числа, з подальшим додаванням 1 до результату. Інакше кажучи, додатковий код отримують додаванням 1 до оберненого коду.

Додатковий код можна одержати за таким формальним правилом: цифри прямого коду додатного числа необхідно інвертувати послідовно зліва направо до останньої одиниці, не включаючи її. Останню праву одиницю і наступні за нею (праворуч) нулі слід залишити без зміни. Діапазон від'ємних чисел у додатковому коді становить від 0 до -2^{n-1}. Як приклад на рис. 1.17 показано додатковий код числа -53_{10} у 8-розрядній сітці, для якої діапазон від'ємних чисел – від 0 до -128_{10}, при цьому додатковим кодом найменшого числа -128_{10} є число 10000000_2.

Розряди	*D*7	*D*6	*D*5	*D*4	*D*3	*D*2	*D*1	*D*0
Ваги розрядів	–	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	Знак числа	Модуль числа						
	1	1	0	0	1	0	1	1

Рис. 1.17. Додатковий код числа -53_{10} у 8-розрядній сітці

Розглянемо віднімання чисел у додатковому коді. Подамо різницю у вигляді:

$$125_{10} - 53_{10} = 125_{10} + (256_{10} - 53_{10}) - 256_{10}. \qquad (1.4)$$

Вираз ($256_{10} - 53_{10}$) є додатковим кодом від'ємного числа -53_{10}. З виразу (1.4) випливає, що операцію віднімання можна замінити операцією

додавання у додатковому коді. Виконання операції додавання числа 125_{10} та числа -53_{10}, поданого у додатковому коді, ілюструє рис. 1.18.

Рядок перенесень	1	1	1	1	1	1	1	1	
$+125_{10}$	+	0	1	1	1	1	1	0	1
$256_{10} - 53_{10}$		1	1	0	0	1	0	1	1
$+72_{10}$		0	1	0	0	1	0	0	0

Рис. 1.18. Виконання операції додавання числа 125_{10} та числа -53_{10}, поданого у додатковому коді

Унаслідок додавання відбулося переповнення, тобто перенесення у неіснуючий дев'ятий розряд, що зменшило результат на 256_{10}. Отже, відповідно до формули (1.4) отримана сума і буде остаточним результатом.

Перевагами додаткового коду є простота операцій одержання та додавання чисел із різними знаками, а також те, що нуль має єдине подання: 0 = 00000000. Завдяки цим перевагам додатковий код використовують найчастіше.

Кодування, захищене від завад

Унаслідок дій завад під час передачі, обробки і збереження двійкових кодів у мікропроцесорних системах можуть статися помилки, наприклад прийом 1 замість 0 або навпаки. Це може призвести до неправильного результату роботи мікропроцесорної системи.

Неправильний прийом, обробка або збереження одного або декількох бітів називається *помилкою.* Кількість неправильно прийнятих, оброблених або збережених бітів називається *кратністю помилки.* Для визначення діапазону послідовності бітів, який містить неправильні біти, використовується поняття *пакет помилок*. Довжина пакета помилок визначається кількістю символів між першим і останнім неправильним бітом, включаючи ці біти. При цьому на кількість правильних бітів, розміщених між неправильними, накладається обмеження.

Одним із найбільш ефективних шляхів захисту інформації у мікропроцесорних системах є кодування, стійке до завад, яке здійснюється введенням у кодові комбінації додаткових бітів, призначених або для виявлення і виправлення помилок, або тільки для виявлення помилок. Відповідно до цього коди, стійкі до завад, поділяють на коректувальні коди, які виявляють і виправляють помилки, та коди, які тільки виявляють помилки.

Можливість виявлення помилок за наявності додаткових бітів обумовлена тим, що для передачі інформації використовуються не всі можливі комбінації в кількості $N = 2^n$, а лише деяка частина з них – $N_0 < N$. Комбінації з безпомилковою інформацією є дозволеними, інші $N - N_0$ комбінацій є забороненими. Поява заборонених комбінацій розглядається як помилка. Можливі такі помилки, за яких одна дозволена комбінація переходить у іншу. У цьому разі помилки не виявляються. Загальна кількість можливих помилок визначається добутком $N \cdot N_0$. З цієї кількості помилок буде виявлено $N_0(N - N_0)$. Відношення кількості виявлених до загальної кількості можливих помилок називають *коефіцієнтом виявлення коду*:

$$K_{\text{вияв}} = \frac{N_0(N - N_0)}{N \cdot N_0} = 1 - \frac{N_0}{N}.$$

Розглянемо можливість виправлення помилок під час використання коду з додатковими бітами. Для забезпечення виправлення помилок множина кодових комбінацій N розбивається на N_0 підмножин, що не перетинаються. Кожній з цих підмножин відповідає одна з дозволених комбінацій. При цьому виправляються помилки, що не переводять передану комбінацію в інші підмножини. Помилка буде виправлена у $N - N_0$ випадках, тоді як загальна кількість помилок – $N \cdot N_0$. Відношення кількості виправлених помилок до кількості виявлених називають *коефіцієнтом виправлення коду*:

$$K_{\text{випр}} = \frac{N - N_0}{N_0(N - N_0)} = \frac{1}{N_0}.$$

Ступінь відмінності двох кодових комбінацій характеризується кодовою відстанню *d,* тобто кількістю бітів, які є різними для двох комбінацій. Кодова відстань може набувати значень від 1 до *n,* де *n* – довжина кодової комбінації. Кодову відстань можна визначити, обчисливши суму двох комбінацій за модулем 2 і підрахувавши кількість одиниць у цій сумі. Якщо $d = 1$, то всі кодові комбінації є дозволеними, а виявлення і виправлення помилок неможливе. При $d = 2$ одноразові помилки переводять дозволені комбінації в заборонені, тому такий код може виявляти всі одноразові помилки, а також помилки непарної кратності (помилки одночасно у трьох, п'яти, семи і так далі розрядах). Помилки парної кратності (помилки одночасно у двох, чотирьох, шести і так далі розрядах) таким кодом не виявляються. Якщо необхідно виявляти помилки кратності *r*, то мінімальна кодова відстань $d_{\min}$ між комбінаціями коду має бути хоча б на одиницю більше від кратності помилки $d_{\min} \geq r + 1$.

Коректувальна здатність коду забезпечується введенням *k* додаткових перевірних бітів. Тоді загальна довжина кодової комбінації буде $n + k$, а загальна кількість можливих комбінацій $N = 2^{n+k}$, що дозволяє визначити необхідну кількість перевірних бітів для побудови кодів, стійких до завад.

Одним з поширених кодів, стійких до завад, які використовуються у мікропроцесорній техніці, є *код з контролем на парність*. У цьому коді до інформаційних бітів праворуч додається один контрольний біт ($k = 1$). Якщо кількість одиниць в інформаційних бітах є парною, то значення контрольного біта дорівнює 0, в протилежному випадку – 1. Отже, у будь-якому випадку кількість одиниць у повній послідовності $n + 1$ біт є парною. Якщо при перевірці після передачі кількість одиниць є непарною, то це означає, що має місце помилка. Код із контролем на парність

дозволяє виявляти всі помилки непарної кратності і не дозволяє виявляти помилки парної кратності.

***Приклад* 1.22.** Знайти значення контрольного біта *k* в кодах із контролем на парність: 1) 11000; 2) 11100.

Значення контрольного біта визначимо з умови парності кількості одиниць у послідовності із заданого коду і контрольного біта. Таким чином, у прикладі

N	*k*
11000	0
11100	1

Контрольні питання

1. Дайте визначення поняттю мікропроцесор
2. Що таке мікропроцесорна система? Чим вона відрізняється від мультимікропроцесорної система?
3. Дайте визначення мікропроцесорного комплекту.
4. Назвіть складові частини МПК.
5. За якими признаками класифікують мікропроцесорні комплекти?
6. Поясніть різницю між універсальними та спеціалізованими МП.
7. Для вирішення яких задач використовують сигнальні процесори?
8. Для вирішення яких задач використовують мультимедійні та медійні процесори?
9. На які типи розділяють МПК? Наведіть приклади.
10. Поясніть різницю між однокристальним і багатокристальним МПК
11. Яке призначення та які складові частини системної шини?
12. Вкажіть принципи передачі інформації по паралельним і послідовним шинам:.
13. Як конструктивно виконується послідовна шина?
14. Як конструктивно виконується паралельна шина?

15. Дайте визначення системної шини.

16. Назвіть призначення шини даних.

17. Які можливі напрямки передавання даних по шині даних?

18. Назвіть призначення шини керування.

19. Назвіть принципи побудови МПС і охарактеризуйте їх.

20. Поясніть призначення функціональних модулів типової структури МПС.

21. Поясніть призначення входу керування третім станом.

22. Яку назву мають входи керування третім (високоімпедансним) станом модулів МП?

23. Як діє сигнал CS на виходи ВІС з трьома станами?

24. Назвіть модулі, що входять до складу типової МПС.

25. Перерахуйте складові системи пам'яті МПС.

26. Назвіть призначення оперативного запам'ятовуючого пристрою.

27. Назвіть призначення постійного запам'ятовуючого пристрою.

28. Дайте визначення байта інформації.

29. Як звернутися до певної комірки пам'яті?

30. Назвіть призначення модуля ЦП.

31. Назвіть призначення пристроїв введення-виведення.

32. Як здійснюється введення і виведення даних у пристрої введення-виведення?

33. Дайте визначення архітектури МП.

34. Яка відмінність між гарвардською та фоннеймановською архітектурою?

35. Які функції виконує пристрій керування?

36. Подайте число -75_{10}: а) у прямому коді; б) у оберненому коді; в) у додатковому коді.

37. Вкажіть діапазон значень додатних і від'ємних чисел у додатковому коді при $n = 8$.

РОЗДІЛ 2. АРХІТЕКТУРА І ХАРАКТЕРИСТИКИ AVR-МІКРОКОНТРОЛЕРІВ

2.1. Характеристики AVR-мікроконтролерів

Перші мікроконтролери AVR були розроблені в дослідницькому центрі Atmel в Норвегії групою інженерів до складу якої входили Alf Bogen та Vergard Wollan. Ініціали їх імен та посилання на тип архітектури (Risc architecture) і сформували назву "AVR". Перший AVR мікроконтролер AT90S1200 був випущений у 1996 – 1997 р.

Сьогодні AVR мікроконтролери за показниками «ціна – швидкодія - енергоспоживання» вважають одними з найкращих серед 8 бітових контролерів з RISC архітектурою. Об'єм їх продажу подвоюється щороку. По інформації, що наведена в огляді фірми Atmel Product Selection Guide, Volume 11, 2015, відзначається, що на сьогоднішній день випущено біля 7 мільярдів AVR мікроконтролерів.

Сфери застосування AVR мікроконтролерів надзвичайно широкі – від найпростіших приладів до складних систем збору обробки інформації та керування, що застосовуються у побуті та промисловості.

Перший офіційний каталог мікроконтролерів фірми Atmel було випущено в 1997 році. До складу цього каталогу ввійшли мікроконтролери першого сімейства Classic. А вже в 1999 році у другому випуску каталогу були приведені дані мікроконтролерів трьох сімейств: Classic, Mega, Tiny.

З 2008 р. фірма Atmel почала серійний випуск нового сімейства XMega 8 бітових AVR мікроконтролерів.

Сьогодні фірмою Atmel виробляються 8, 8/16 та 32 розрядні AVR мікроконтролери. Всього випускається 294 різних типів AVR мікроконтролерів.

Сімейства AVR мікроконтролерів мають наступні загальні властивості:

- це 8 (16- або 32) розрядні RISC мікроконтролери загального призначення з Гарвардською архітектурою з фізично та логічно розділеними шинами даних та адресними просторами пам’яті програм та даних;
- мають вбудовану пам'ять програм Flash – типу, статичну SRAM пам’ять даних та енергонезалежну EEPROM пам’ять даних. Підвищена ємність вбудованої пам’яті даних дозволяє використовувати для розробки прикладних програм мову високого рівня;
- продуктивність мікроконтролера сягає 1 MIPS на частоті 1МГц;
- мають 32 регістри загального призначення, які утворюють файловий регістр. За участю цих регістрів виконуються операції в арифметично логічному пристрої. Таке збільшення кількості регістрів загального призначення дозволяє використовувати їх як для зберігання змінних, так і для виконання операцій над ними, що суттєво підвищує ефективність програм;
- в AVR мікроконтролерах використовується 1-рівневий конвеєр по роботі з пам’яттю програм, який дозволяє на 1 циклі мікроконтролера виконувати поточну команду та робити вибірку коду наступної команди;
- передбачена можливість як зовнішнього так і внутрішньо схемного програмування мікроконтролера;
- пам'ять програм розподілена на два сектори – для прикладної програми (application sector) та для програми завантажувача (boot sector) мікроконтролера. Існує можливість перерозподілу ємності цих сегментів. Сектор завантаження дозволяє реалізувати функцію самопрограмування;
- наявність спеціальних засобів (фюзесів) програмування структури мікроконтролера.

Сучасна класифікація сімейств 8-розрядних AVR мікроконтролерів представлена у каталогах трьома напрямками Tiny, Mega, XMega. Проте на офіційному сайті фірми Atmel визначено більшу кількість сімейств:

- tinyAVR- 38 різних 8 розрядних мікроконтролерів;
- megaAVR – 105 різних 8 розрядних мікроконтролерів;
- AVR XMEGA – 46 різних 8/16 розрядних мікроконтролерів;
- Battery Management MCUs – 5 різних 8 розрядних мікроконтролерів;
- Automotive AVR MCUs – 38 різних 8 розрядних мікроконтролерів;
- 32 біт AVR UC3 - 62 різних 32 розрядних мікроконтролерів.

Вбудоване ядро AVR мікроконтролерів використовується також спільно з FPGA програмованими матрицями в ASIC платформах.

Також існує можливість підключення ядра AVR мікроконтролера при розробці систем на кристалі SoC на базі FPGA матриць. Існує можливість отримання опису ядра з використанням ресурсів сайту www.opencores.org.

Досить давно випускають як спеціалізовані мікросхеми, так і засоби для емуляції таких систем, наприклад ATasicICE POD.

Огляд максимальних ресурсів AVR мікроконтролерів

До ресурсів мікроконтролерів зазвичай включають дані про фізичні характеристики корпусів, діапазони напруги живлення та робочої температури, максимальну робочу частоту, кількість розрядів шини даних, об'єми пам'яті програм, статичної та енергонезалежної пам'яті даних. Вказують на кількість ліній вводу/виводу та дають опис кількості вбудованих контролерів, таких як таймери-лічильники, контролер PTC ємнісної Touch панелі, периферійних контролерів USART, SPI, TWI, USB, PWM, Ethernet, CAN. Вказують дані по групі аналогових контролерів компаратора, аналогово-цифрового та цифро-аналогового перетворювачів. Максимальні ресурси мікроконтролерів змінюються у залежності від типу сімейства.

У табл.2.1 відображені максимальні діапазони по ресурсам всіх AVR мікроконтролерів.

Таблиця 2.1

№	Назва ресурсу	Значення, діапазон
1.	Розрядність процесорного ядра (CPU)	8 / 16 / 32
2.	Кількість виводів корпусу мікроконтролера	6 – 144
3.	Максимальна робоча частота	1-84 МГц
4.	Об'єм вбудованої пам'яті програм FLASH	0,5 – 512 кБайт
5.	Об'єм статичної пам'яті даних SRAM	0,03 - 128 кБайт
6.	Об'єм енергонезалежної пам'яті даних EEPROM	0 – 4096 Байт
7.	Максимальна кількість ліній вводу/виводу	4 – 123
8.	Напруга живлення	0,7 – 25 В
9.	Діапазон робочих температур	-40 - +150 ºС
10.	Кількість каналів Touch Channels (контролер PTC)	0 – 56
11.	Кількість таймерів лічильників (8 / 16 біт)	1 – 10
12.	Сторожовий таймер (WDT)	1
13.	Незалежний RTC генератор 32кГц	0 – 1
14.	Аналоговий компаратор	1 – 8
15.	Температурний сенсор	0 – 1
16.	Розрядність аналогово - цифрового перетворювача	8 – 10 - 12
17.	Кількість каналів аналогово-цифрового перетворювача	4 – 28
18.	Цифро аналоговий перетворювач DAC	0 – 4
19.	Контролер UART	0 – 8
20.	Контролер SPI	0 – 12
21.	Контролер TWI	0 – 4
22.	Контролер USB	0 –Device - OTG
23.	Контролер PWM	0 – 36
24.	Контролер Ethernet	0 – 1
25.	Контролер CAN	0 – 2

Сімейство AVR мікроконтролерів CLASSIC

До першої групи AVR мікроконтролерів, які започаткували сімейство CLASSIC, входило 4 мікроконтролера. Найбільш цікавим був мікроконтролер AT90S8515. Фірма Atmel створювала цей мікроконтролер як перехідний від сімейства MCS-51 до AVR. Типологічно, за розміщенням ліній портів, живлення, скидання та генератора, він повністю повторював

розповсюджений мікроконтролер AT89C51. Це дозволило розробникам без зміни печатних вузлів переключитися на новий тип сімейства мікроконтролерів. Перші моделі мікроконтролерів характеризувалися значною кількістю помилок, які відзначалися в Errata - листах технічної документації. Поступово у нових версіях мікроконтролерів ці помилки виправлялися. Для позначення мікроконтролерів цього сімейства використовувався префікс «AT90». Розвиток цього сімейства характеризувався появою нових мікросхем з розширеними можливостями, в тому числі і спеціалізованих, наприклад для роботи з CAN шиною (AT90CAN), USB шиною (AT90USB), PWM формувачем імпульсів керування трифазових інверторів напруги (AT90PWM).

В цьому сімействі створювалися мікроконтролери як з розширеними можливостями (наприклад введено аналогово цифровий перетворювач в AT90S4433, AT90S8535), так і з обмеженими можливостями, наприклад, AT90S2323, AT90S2343 випущено в корпусі DIP8. По завершенню розвитку цього напрямку до сімейства входило 17 різних типів мікроконтролерів.

Поступово фірма Atmel відійшла від напрямку розвитку сімейства CLASSIC та почала розвивати два інших сімейства - з обмеженими апаратними можливостями AT tiny та розширеними можливостями ATmega. При цьому мікроконтролери сімейства CLASSIC розподілили між новими сімействами.

Сімейство AVR мікроконтролерів tinyAVR

Мікроконтролери сімейства tinyAVR розроблені для систем де суттєвими є розміри та вартість. В початкових мікросхемах цього сімейства зменшували кількість вбудованих ресурсів, у порівнянні з сімейством ATmega. Проте на певному етапі була проведена розробка мікросхем ресурсні показники яких перевищили показники деяких мікроконтролерів ATmega.

Базовими рисами цього сімейства є компактність та потужність споживання. Мікроконтролери мають таку само продуктивність, як і мікроконтролери сімейства ATmega. Мікроконтролери можуть працювати в широкому діапазоні напруги живлення 0,7 – 5,5 В та температури зовнішнього середовища від -40 до 125°С, з робочою частотою до 20 МГц та продуктивністю до 20MIPS. Випускаються у корпусах які мають 6 – 32 виводи. Мають вбудовану FLESH пам'ять програм 0,5-8 кБайт, статичну SRAM та енергонезалежну EEPROM пам'ять даних, супервізор напруги живлення. Характеризуються високим ступенем інтеграції. Кожен контакт мікроконтролера, за винятком ліній подачі живлення, має значну кількість альтернативних функцій. До складу навіть найпростіших мікроконтролерів (TINY5, TINY10), що випускаються у корпусі з 6 виводами (SOT23-6) входять такі складні вузли як аналогово-цифровий перетворювач з 4 входами.

Проте деякі типи мікроконтролерів мають надзвичайно малу ємність SRAM пам'яті даних (TINY4, TINY5), що не дозволяє ефективно використовувати мову Сі для програмування. Розробку прикладних програм у цьому випадку доцільно проводити з використанням мови асемблеру. Деякі мікроконтролери мають у своєму складі підвищуючий стабілізатор напруги, який дає можливість працювати з напругою джерела живлення до 0,7В. У табл. 2.2 відображені максимальні діапазони значень для ресурсів AVR мікроконтролерів tinyAVR.

Таблиця 2.2

№	Назва ресурсу	Значення, діапазон
1.	Розрядність процесорного ядра (CPU)	8
2.	Кількість виводів корпусу мікроконтролера	6 – 32
3.	Максимальна робоча частота	4-20 МГц
4.	Об’єм вбудованої пам’яті програм FLASH	0,5 – 16 кБайт
5.	Об’єм статичної пам’яті даних SRAM	0,03 - 1 кБайт
6.	Об’єм енергонезалежної пам’яті даних EEPROM	0 – 512 Байт
7.	Максимальна кількість ліній вводу/виводу	4 – 28
8.	Напруга живлення	0,7 – 5,5 В

		Продовження табл.. 2.2
9.	Діапазон робочих температур	-40 - +125 ºС
10.	Кількість каналів Touch Channels (контролер PTC)	0 – 12
11.	Кількість таймерів лічильників (8 / 16 біт)	1 – 3
12.	Сторожовий таймер (WDT)	1
13.	Незалежний RTC генератор 32кГц	0 - 1
14.	Аналоговий компаратор	1 - 2
15.	Температурний сенсор	0 - 1
16.	Розрядність аналогово - цифрового перетворювача	8 – 10
17.	Кількість каналів аналогово-цифрового перетворювача	4 - 28
18.	Контролер UART	0 - 2
19.	Контролер SPI	0 - 2
20.	Контролер TWI	0 - 1
21.	Контролер PWM	0 - 9

У табл. 2.3 відображені характеристики деяких мікроконтролерів сімейства tinyAVR.

Таблиця 2.3

Тип	**Виводи**	**Flash (КВ)**	**SRAM (В)**	**EEPROM(В)**	**I/Q**	**ADC**	**Корпус**
TINY 4	4/6	0.5	32	N/A	4		SOT23-6,UDFN-8
TINY 5	4/6	0.5	32	N/A	4	4 x 8bit	SOT23-6,UDFN-8
TINY 9	4/6	1	32	N/A	4		SOT23-6,UDFN-8
TINY 10	4/6	1	32	N/A	4	4 x 8bit	SOT23-6,UDFN-8
TINY 13A	8/10/12	1	64	64	6	4 x 10bit	PDIP-8, SOIC-8, QFN-10,QFN-20
TINY 20	14/15/20	2	128	N/A	10/12	4 x 10bit	SOIC-14, TSSOP-14, WCCSP-12, UFBGA-15, VQFN-20
TINY 24A	14/15/20	2	128	128	12	12 x 10bit	SOIC-14, PDIP-14, UFBGA-15, QFN-20 MLF-20, VQFN-20
TINY 25	8/20	2	128	128	6	4 x 10bit	PDIP-8, SOIC-8, QFN-20,QFN-20
TINY 40	20	4	256	N/A	18	8 x 10bit	SOIC-14, TSOP-20, VQFN-20
TINY 44A	14	4	256	256	12	8 x10 bit	SOIC-14, PDIP-14, UFBGA-15, QFN-20 MLF-20, VQFN-20
TINY 45	8/20	4	256	256	6	4 x 10bit	PDIP-8, SOIC-8, TSOP-8,QFN-20
TINY 48	28/32	8	256	64	24/28	10bit	PDIP-28,QFN-28, QFP-32,QFN-32
TINY 84A	14	8	512	512	12	8 x 10bit	SOIC-14, PDIP-14, UFBGA-15, QFN-20,MLF-20, VQFN-20

Сімейство AVR мікроконтролерів megaAVR

На іншому полюсі AVR, мікроконтролерів за рівнем інтеграції і можливостей, знаходиться сімейство megaAVR**.** Для мікроконтролерів цієї групи характерні наступні властивості:

- великий об'єм Flash пам'яті програм від 4 до 256 Кбайт);
- режим самопрограмування, забезпечений можливістю програмування з Boot-сектору пам’яті програм за допомогою програми-завантаження;
- вбудований апаратний помножувач, що підтримує множення 8 розрядних чисел;
- розширені набори вбудованої периферії;
- широкий набір спеціальних мікроконтролерних функцій, у тому числі: до шести режимів енергозбереження і можливість програмної установки тактової частоти;
- розширення системи команд до 130 – 133;
- організація в нових моделях інтерфейсу граничного сканування (IEEE 1149. 1/ JTAG), що підтримує вбудоване налагодження і забезпечує ще один шлях програмування Flash і EEPROM пам'яті, фюзесів конфігурації мікроконтролера та бітів блокування вмісту пам’яті;
- спеціальні мікроконтролерні функції, що забезпечують високу усталеність роботи апаратних і програмних засобів при випадкових змінах напруги живлення;
- ємність вбудованої пам’яті програм знаходиться у межах від 4 до 256 кБайт.

Мікроконтролери цього сімейства характеризуються типовою продуктивністю виконання програм 1,0 MIPS/МГц можуть працювати на частоті генератора до 20МГц.. Області застосування мікроконтролерів цієї групи – в системах загального призначення для реалізації функцій збору інформації, її обробки та керування периферійним обладнанням.

У табл. 2.4 відображені максимальні діапазони значень для вбудованих ресурсів AVR мікроконтролерів megaAVR.

Таблиця 2.4

№	Назва ресурсу	Значення, діапазон
1.	Розрядність процесорного ядра (CPU)	8
2.	Кількість виводів корпусу мікроконтролера	20 - 100
3.	Максимальна робоча частота	16-20 МГц
4.	Об'єм вбудованої пам'яті програм FLASH	4 – 256 кБайт
5.	Об'єм статичної пам'яті даних SRAM	0,25 - 16 кБайт
6.	Об'єм енергонезалежної пам'яті даних EEPROM	256 – 4096 Байт
7.	Максимальна кількість ліній вводу/виводу	19 - 86
8.	Напруга живлення	1,8 – 5,5 В
9.	Діапазон робочих температур	-40 - +125 ºС
10.	Кількість каналів Touch Channels (контролер PTC)	8 - 32
11.	Кількість таймерів лічильників (8 / 16 біт)	1 - 6
12.	Сторожовий таймер (WDT)	1
13.	Незалежний RTC генератор 32кГц	0 - 1
14.	Аналоговий компаратор	1 - 4
15.	Температурний сенсор	0 - 1
16.	Розрядність аналогово - цифрового перетворювача	10
17.	Кількість каналів аналогово-цифрового перетворювача	8 - 16
18.	Цифро аналоговий перетворювач DAC	0 - 1
19.	Контролер UART	0 - 4
20.	Контролер SPI	1 - 5
21.	Контролер TWI	0 - 2
22.	Контролер USB	0 –Device - OTG
23.	Контролер PWM	3 - 15
24.	Контролер CAN	0 - 1

У табл. 2.5 відображені характеристики деяких мікроконтролерів сімейства megaAVR.

Таблиця 2.5

Серія	Виводи	Flash (KB)	SRAM (KB)	EEPROM (KB)	I/O	ADC	Корпус
MEGA 16A	40/44	16	1	0.5	32	8 x 10bit	PDIP-40, MLF-44, QFP-44
MEGA 32A	40/44	32	2	1	32	8 x 10bit	PDIP-40, MLF-44, QFP-44

Продовження табл.. 2.5

MEGA 48PB	32	4	0.5	0.25	27	8 x 10bit	MLF-32, QFP-32
MEGA 64A	64	64	2	N/A	53	8 x 10bit	MLF-64, QFP-64
MEGA 88PB	32	8	1	0.5	23	8 x 10bit	MLF-32, QFP-32
MEGA 162	40/44	16	1	0.5	35	N/A	PDIP-40,MLF-44, QFP-44
MEGA 164PA	40/44/49	16	1	0.5	32	8 x 10bit	PDIP-40,MLF-44, QFP-44
MEGA 168PB	32	16	1	0.5	27	8 x 10bit	MLF-32, QFP-32
MEGA 169PA	64	16	1	0.5	54	8 x 10bit	MLF-64,QFP-64, FN-64
MEGA 324PB	44	32	2	1	39	8 x 10bit	MLF-44, QFP-44
MEGA 328PB	32	32	2	1	27	8 x 10bit	MLF-32, QFP-32
MEGA 329PA	64	32	2	1	54	8 x 10bit	MLF-64, QFP-64
MEGA 640	100	64	8	4	86	16 x 10bit	BGA-100, QFP-100
MEGA 344PA	40/44	64	4	2	32	8 x 10bit	MLF-64, QFP-64
MEGA 349P	64	64	4	2	54	8 x 10bit	PDIP-40, MLF-44, QFP-44
MEGA 1280	100	128	8	4	86	16 x 10bit	BGA-100, QFP-100
MEGA 1281	64	128	8	4	54	16 x 10bit	MLF-64, QFP-64
MEGA 1284	40/44	128	16	4	32	8 x 10bit	PDIP-40, MLF-44, QFP-44
MEGA 2560	100	256	8	4	86	16 x 10bit	BGA-100, QFP-100
MEGA 2561	64	256	8	4	54	8 x 10bit	MLF-64, QFP-64
MEGA 3290P	100	32	2	1	69	8 x 10bit	QFP-100
MEGA 6490P	100	64	4	2	69	8 x 10bit	QFP-100
MEGA 8515	40/44	8	0.5	0.5	35	N/A	PDIP-40,MLF-44, QFP-44
MEGA 8535	40/44	8	0.5	0.5	32	8 x 10bit	PDIP-40,MLF-44, QFP-44

Сімейство AVR мікроконтролерів Xmega

Мікроконтролери AVR сімейства Xmega відносяться до 8/16 розрядних та характеризуються вищою складністю та ступенем інтеграції,

у порівнянні з megaAVR. Їх доцільно використовувати в системах загального призначення для яких суттєвими є складність системних вимог та швидкість математичної обробки сигналів.

Для цих мікроконтролерів характерні наступні властивості:

- великий обсяг вбудованої FLESH пам'яті програм від 16 до 384 кБайт;
- типова продуктивність 1,0 MIPS/МГц та максимальна робоча частота до 32 МГц;
- наявність аналогових пристроїв обробки інформації з підвищеною точністю (аналогово цифровий та цифро аналоговий перетворювачі 12 бітні) та швидкодією. Для аналогово цифрового перетворювача максимальна швидкодія досягає 4 MSPS;
- система обробки подій спрощує взаємодію периферійних ресурсів мікроконтролера. Всі периферійні пристрої можуть використовувати механізм прямого доступу до пам'яті (DMA);
- високий рівень інтеграції, наприклад, можлива реалізація до 32 каналів PWM, до 8 вбудованих контролерів UART, можливість використання USB контролера;
- високий рівень кількості виконуваних паралельно задач. Система переривань охоплює до 122 джерел;
- значна кількість вбудованих 16 бітних таймерів (до 8);
- наявність вбудованих криптоблоків алгоритмів AES (Advanced Encryption Standard) та DES (Data Encryption Standard);
- на відміну від контролерів megaAVR забезпечена сумісність різних мікроконтролерів по програмному коду. Це дає можливість створення бібліотек функцій та використання їх в різноманітних проектах.

Мікроконтролери сімейства Xmega у залежності від складності поділяються на 5 серій – A, B, C, D, E. Найбільш складними та насиченими апаратними ресурсами є мікроконтролери A групи. Їх

доцільно використовувати у складних пристроях які характеризуються найбільш жорсткими функціональними вимогами та швидкодією. Мікроконтролери B серії більш прості проте до їх складу входить інтегрований контролер рідкокристалічного дисплею. Серія C включає мікроконтролери початкового рівня, проте до їх складу входить повношвидкісний USB порт. Серія D включає мікроконтролери початкового рівня які характеризуються малим енергоспоживання. Вони рекомендуються фірмою Atmel для технічних рішень критичних з точки зору споживання енергії. Серія E характеризується найменшим корпусом. Мікроконтролери цієї серії призначені для використання в системах із жорсткими вимогами до габаритів. Загальні характеристики A-E серій Xmega мікроконтролерів приведено у табл.2.6.

Таблиця 2.6

Серія	**Виводи**	**Flash (KB)**	**SRAM (KB)**	**EEPROM (KB)**	**I/O**	**ADC**	**Корпус**
A1U	100	64/128	4/8	2	78	2-16 x 12bit 2msps	TQFP-100, BGA-100, VFBGA-100
A2U	64	64/128 192/256	4/8/16	2/4	50	2-16 x 12bit 2msps	TQFP-64, QFN-64
A4U	44	16/32 64/128	2/4/8	1/2	34	12 x 12bit 2msps	QFP-44, QFN-44, VFBGA-49
B1	100	64/128	4/8	2	53	2-8 x 12bit	TQFP-100, VFBGA-100
B3	64	64/128	4/8	2	36	1-8 x 12bit	TQFP-64, QFN-64, DRQFN-64
C3	64	32/64 128/192 256/384	2/4/8/16	4/8/16	50	1-16 x 12bit	TQFP-64, QFN-64
C4	44	16/32	2/4	1	34	1-16 x 12bit	TQFP-44, QFN-44, VFBGA-49
D3	64	32/64 128/192 256/38	4/8 16/32	1/2/4	50	1-16 x 12bit	TQFP-64, QFN-64
D4	44	16/32 64/128	1/2/4/8	1/2/4	34	1-12 x 12bit	TQFP-44, VFBGA-49
E5	32	8/16/32	1/2/4	0.5/1	26	1-16 x 12bit	QFP-44, QFN-44,UQFN-49

У табл. 2.7 приведено параметри деяких мікроконтролерів Xmega.

Таблиця 2.7.

Тип	Status (b)	Flash (KB)	Boot code (Bytes)	EEPROM (KB)	SRAM (KB)	I/O pins	16-bit Timers	PWM (channel)	SPI	TWI(I2C)	USART	12-bit ADC (ch.)	12-bit DAC (ch.)	Analog Comp.	Interrupts	Interrupts Ext.
ATxmega 64A1	I	64	4	2	4	78	8	24	4	4	8	2x8	2x2	4	122	78
ATxmega 128A1	I	128	8	2	8	78	8	24	4	4	8	2x8	2x2	4	122	78
ATxmega 192A1	I	192	8	4	16	78	8	24	4	4	8	2x8	2x2	4	122	78
ATxmega 256A1	I	256	8	4	16	78	8	24	4	4	8	2x8	2x2	4	122	78
ATxmega 384A1	I	384	8	4	34	78	8	24	4	4	8	2x8	2x2	4	102	78
ATxmega 64A3	I	64	4	2	4	50	7	22	3	2	7	2x8	1x2	4	102	50
ATxmega 128A3	I	128	8	2	8	50	7	22	3	2	7	2x8	1x2	4	102	50
ATxmega 192A3	I	192	8	4	16	50	7	22	3	2	7	2x8	1x2	4	102	50
ATxmega 256A3B	I	256	8	4	16	49	7	22	2	2	6	2x8	1x2	4	102	49
ATxmega 256A3	I	256	8	4	16	50	7	22	3	2	7	2x8	1x2	4	102	50
ATxmega 16A4	F	16	4	1	2	34	5	16	2	2	5	1x12	1x2	2	77	34
ATxmega 32A4	F	32	4	1	4	34	5	16	2	2	5	1x12	1x2	2	77	34
ATxmega 64A4	F	64	4	2	4	34	5	16	2	2	5	1x12	1x2	2	77	34
ATxmega 128A4	F	128	4	2	8	34	5	16	2	2	5	1x12	1x2	2	77	34
ATxmega 64D3	F	64	4	2	4	50	5	18	2	1	3	1x16		2	67	50
ATxmega 128D3	F	128	8	2	8	50	5	18	2	1	3	1x16		2	67	50
ATxmega 192D3	F	192	8	2	16	50	5	18	2	1	3	1x16		2	67	50
ATxmega 256D3	F	256	8	4	16	50	5	18	2	1	3	1x16		2	67	50
ATxmega 16D4	F	16	4	1	2	34	4	14	2	1	2	1x12		2	55	34
ATxmega 32D4	F	32	4	1	4	34	4	14	2	1	2	1x12		2	55	34

Всі МК містять 16 розрядний RTC, окрім ATxmega 256A3B, який має 32 розрядний; Vcc Range – 16 - 32V, Clock Speed - 32 MHz. МК ATxmegaxxAx мають 8 каналів Event і 4 канали DMA, ATxmegaxxDx – 4 канали Event і не мають DMA.

Сімейство AVR мікроконтролерів Battery Management MCUs

Мікроконтролери цього сімейства доцільно використовувати в системах які працюють спільно з літій-іонними батареями. Вони дозволяють визначати рівень заряду акумуляторної батареї, яка має від 1 до 4 елементів. На базі цих мікроконтролерів може бути створена система з лічильником кулонів, захистом від короткого замикання та перегріву. Можливе балансування рівнів заряду окремих акумуляторів в батареї. Особливістю мікроконтролерів є те, що вони можуть працювати в широкому діапазоні робочої напруги від 1,8 до 25 В. Мають досить велику ємність пам’яті програм від 8 до 40 кБайт, типову для AVR мікроконтролерів продуктивність 1,0 MIPS/МГц та максимальну робочу частоту до 8 МГц. Мікроконтролери мають засоби, які дозволяють проводити аутентифікацію акумуляторів, що унеможливлює використання в системах не оригінальних батарей у випадках такої необхідності.

Сімейство AVR мікроконтролерів Automotive AVR MCUs

Це сімейство 8 бітових AVR мікроконтролерів розроблено для використання в автомобільній електроніці. Мікроконтролери можуть працювати в розширеному діапазоні температур від – 40 до 150°C, вирізняються широким набором периферійних функцій та характеризуються підвищеною надійністю. Мають типову для AVR мікроконтролерів продуктивність 1,0 MIPS/МГц та максимальну робочу частоту до 32 МГц.

Фірма пропонує готові рішення по технології SIP (System-in-Package, "система в корпусі"), які поєднують AVR мікроконтролер, інтерфейси LIN

та CAN, стабілізатор напруги та супервізор напруги, блок що дозволяє проводити обчислення з плаваючою комою (FPU), механізм захисту коду FlashVault™, інтерфейси Ethernet, USB OTG. До складу цього сімейства входять мікроконтролери решти сімейств, наприклад, AT90CAN32, ATtiny44, ATmega88, ATxmega 6403, AT32UC3C0512.

На базі таких мікроконтролерів можливе створення компонентів автомобіля для кузова та салону, захисту та доступу до автомобіля, інформаційно-розважальної системи на базі сенсорних рішень, компонентів для моторного відсіку.

2.2. Архітектура AVR мікроконтролера Mega 16

Узагальнена архітектура AVR мікроконтролера Mega 16 наведена на рис. 2.1.

Мікроконтролер містить:

- Flash Program Memory – флеш-пам‘ять програм (ПЗП);
- Program Counter - програмний лічильник;
- Instruction Register – регістр інструкцій;
- Instruction Decoder – пристрій декодування інструкцій;
- Control Lines – шина керування мікроконтролера;
- Data Bus 8-bit – шина даних мікроконтролера;
- EEPROM – енергонезалежна пам‘ять даних з байтовим доступом;
- Data SRAM – ОЗП статичного типу;
- ALU - арифметико-логічний пристрій (АЛП);
- 32x8 General Purpose Register - регістри загального призначення які формують файловий регістр;
- Status and Control – регістр статусу мікроконтролера (SREG);
- Interrupt unit - модуль переривань;
- SPI (Serial Peripheral Interface) - послідовний периферійний

інтерфейс, та послідовний інтерфейс програмування;

- WDT (Watchdog Timer) - сторожjвий таймер;
- Analog Comparator - аналоговий компаратор;
- I/O lines – 8 бітові порти введення-виведення;
- I/O module 1 - I/O module n - модулі вводу виведення інформації до яких належать таймери, аналогово цифровий перетворювач (АЦП), контролер TWI, UART універсальний асинхронний приймач-передавач.

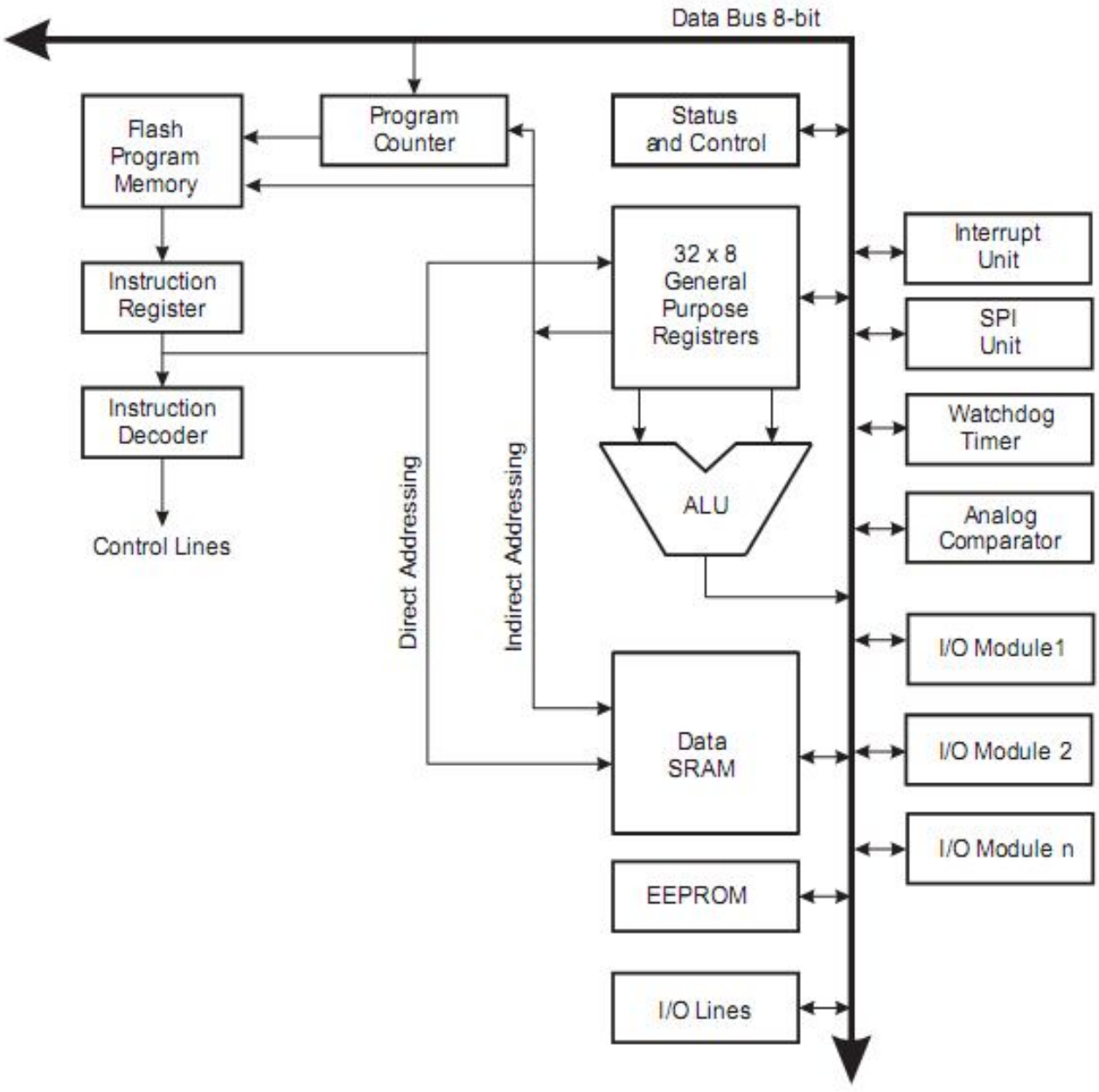


Рис.2.1 Архітектура *AVR* мікроконтролера Mega 16

Основою архітектури ядра AVR мікроконтролера є Гарвардський процесор, регістрова пам’ять (регістри загального призначення та регістри введення-виведення), пам'ять програм та пам'ять даних. Склад та кількість периферійних пристроїв (таймерів, портів введення-виведення, послідовних інтерфейсів, АЦП) залежить від конкретної моделі мікроконтролера.

Гарвардський процесор реалізує повний логічний і фізичний розподіл не тільки адресних просторів, але й інформаційних шин для звертання до пам'яті програм і до пам'яті даних, причому способи адресування і доступу до цих масивів пам'яті також різні. Подібна побудова забезпечує істотне підвищення продуктивності мікроконтролера. Процесор працює одночасно як із пам'яттю програм, так і з пам'яттю даних; розрядність шини пам'яті програм розширена до 16 біт. В AVR мікроконтролерах використовується технологія однорівневої конвеєризації, унаслідок чого цикл "вибірка - виконання" команди помітно скорочений.

Послідовність виконання команд в конвеєрі та обробка їх а АЛП наведено на рис. 2.2 та 2.3 відповідно.

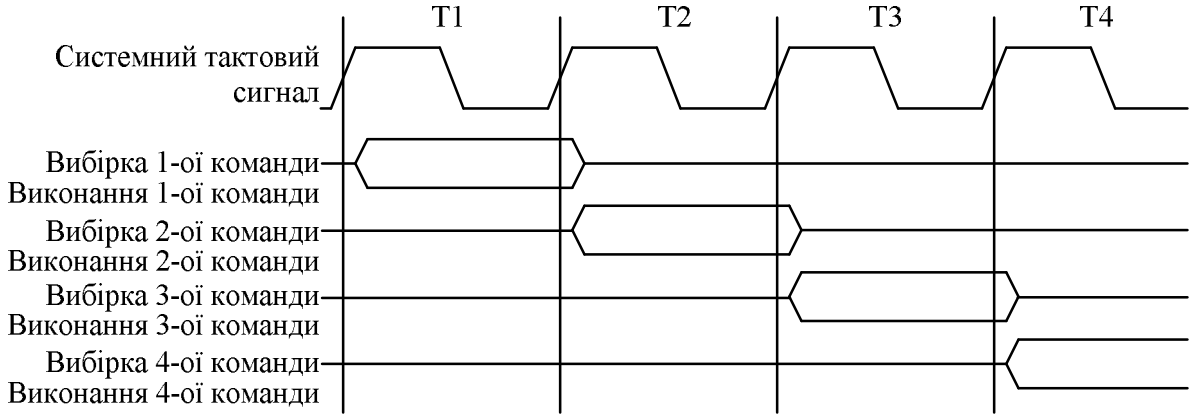


Рис.2.2 Послідовність виконання команд в конвеєрі

Під час першого машинного циклу T1 (рис. 2.2) відбувається вибірка команди з пам'яті програм і її декодування. Під час другого циклу T2 ця

команда виконується, а паралельно відбувається вибірка й декодування другої команди, і так далі.

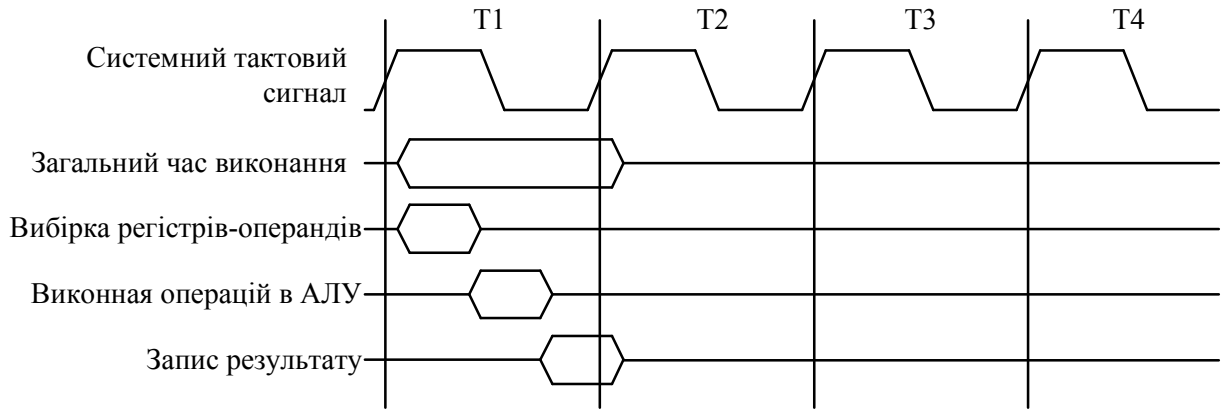


Рис.2.3 Функціонування АЛП

Завдяки підключенню АЛП безпосередньо до регістрового файлу він виконує одну команду (читання вмісту двох регістрів, виконання операції й запис результату в регістр - приймач) за один такт, як показано на рис. 2.3.

У результаті фактичний час виконання кожної команди виходить рівним одному машинному циклу. Таке рішення дозволяє досягати продуктивності до 1 MIPS на МГц. Продуктивність AVR мікроконтролера суттєво підвищується також за рахунок того, що АЛП може виконувати операції з вмістом 32 регістрів загального призначення. Ці регістри можуть виконувати функції як регістрів загального призначення, так і регістрів ОЗП. Це суттєво скорочує часові втрати які пов’язані з пересиланням інформації.

Для порівняння, у мікроконтролерів сімейства MCS-51 коротка команда виконується за 12 тактів генератора (1 машинний цикл), протягом якого процесор послідовно зчитує код операції і виконує її. У PIC-контролерах фірми Microchip, де вже реалізований конвеєр, коротка команда виконується протягом 8 періодів тактової частоти (2 машинних цикли).

Розмір лічильника команд становить від 9 до 12 розрядів, у залежності від обсягу пам'яті. При цьому лічильник команд недоступний програмі безпосередньо (як регістр). При нормальному виконанні програми вміст лічильника команд автоматично збільшується на 1 (або на 2, залежно від команди) у кожному машинному циклі. Цей порядок порушується при виконанні команд переходу, виклику й повернення з підпрограм, а також при виникненні переривань.

Програмна модель AVR мікроконтролера

Програмна модель зображена на рис. 2.4.

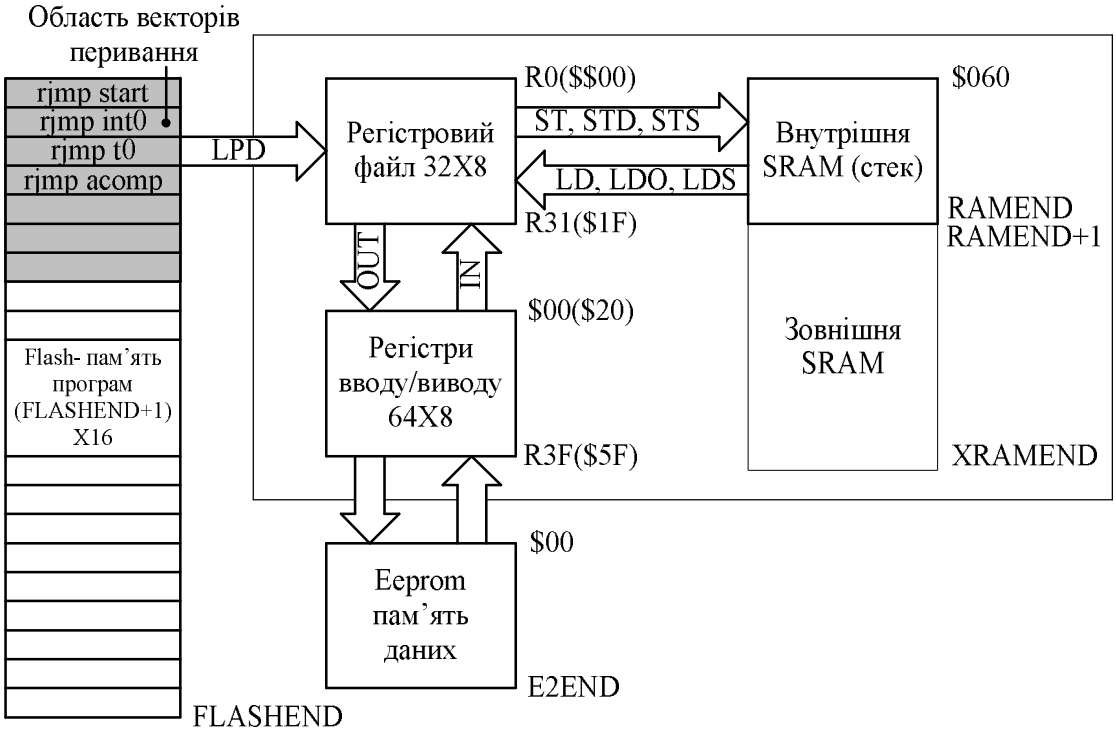


Рис.2.4 Програмна модель AVR-мікроконтролерів

Являє собою сукупність програмно доступних ресурсів, які використовуються наступним чином:

- для збереження робочої програми та програми завантаження мікроконтролера (FLASH пам'ять програм, Application та Boot сектори);

- для збереження динамічних змінних (внутрішня та зовнішня SRAM);
- для збереження енергонезалежних змінних (EEPROM пам'ять даних);
- регістрів вводу/виводу, за допомогою яких здійснюється керування вбудованими ресурсами мікроконтролера;
- регістрового файлу, що використовується для обробки інформації в АЛП.

Арифметико-логічний пристрій

До складу мікроконтролера ATmega16 входить 8-бітовий арифметико-логічний пристрій (АЛП), який безпосередньо пов’язаний з 32 регістрами, що входять до складу регістрового файлу. Арифметико-логічний пристрій здатен виконувати три типи операцій: арифметичні, логічні та бітові. Операції можуть виконуються між елементами регістрового файлу, або цими регістрами та константами. АЛП може виконувати операції над числами представленими у трьох форматах: бітовому, байтовому та двобайтовому (операції додавання та віднімання). Більшість операцій виконуються за один такт генератора. Особливістю АЛП є присутність вбудованого апаратного помножувача байтових чисел.

В результаті виконання кожної операції в АЛП модифікується вміст регістру статусу SREG. Регістр SREG відноситься до групи регістрів введення/виведення та доступний для читання та запису інформації.

На рис.2.5 приведено бітову упаковку регістру SREG.

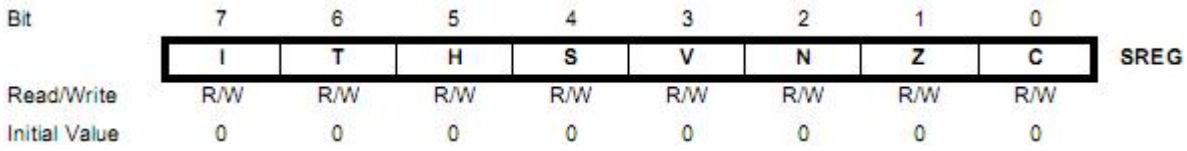


Рис.2.5

Біти цього регістру мають наступне призначення:

- I (SREG.7) – загальний дозвіл переривань.

Мікроконтролер має дворівневу систему дозволів переривань. Індивідуальні дозволи переривань встановлюються у відповідних регістрах масок. Для встановлення загального дозволу роботи системи переривань використовується біт «I». Для дозволу переривань необхідно встановити цей біт «I = 1». У разі виконання підпрограми переривань біт апаратно скидається «I = 0». Апаратне поновлення (I = 1) відбувається після виконання команди повернення (RETI) з підпрограми переривань. Такий алгоритм маніпуляцій бітом «I» забороняє використання алгоритмів з вкладеними перериваннями. Проте можливий програмний дозвіл вбудованих переривань, за рахунок повторного примусового програмного встановлення дозволу (I = 1) у програмах переривання;

- T (SREG.6) – комірка зберігання біту, що копіюється.

Існує дві інструкції, які дозволяють оперативно зберегти (BST) або передати (BLD) біти до будь якого файлового регістру;

- H (SREG.5) – прапор половинного переносу.

Цей біт встановлюється (H = 1), якщо відбувся перенос, або позика при операціях з тетрадами. Цей біт дозволяє працювати з двійково-десятково (BCD) упакованими числами;

- S (SREG.4) – прапор знаку.

Цей прапор визначається як результат операції XOR між прапорами від'ємного результату «N» та переповнення розрядної сітки знакового результату «V». Зокрема прапор встановлюється (S=1), якщо результат арифметичної операції менше нуля;

- V (SREG.3) – прапор переповнення додаткового коду.

Прапор встановлюється (V = 1) при переповненні розрядної сітки знакового результату. Використовується при роботі з числами із знаком;

- N (SREG.2) – прапор від'ємного результату.

Прапор встановлюється (N = 1) якщо результат арифметичної або логічної операції над числами із знаком негативний. Тобто «7» розряд результату операції дорівнює «1»;

- Z (SREG.1) – прапор нуля.

Прапор встановлюється (Z = 1) якщо результат арифметичної або логічної операції дорівнює нулю;

- C (SREG.0) – прапор переносу.

Прапор встановлюється (C = 1) якщо результат арифметичної або логічної операції виходить за межі байту.

Особливістю регістру статусу є те, що він автоматично не зберігається в стеку при виконанні звичайних підпрограм та підпрограм переривань. Якщо в таких підпрограмах використовуються операції з АЛП, то вміст регістру статусу необхідно зберігати з використанням програмних засобів..

Ознаки результату операції використовуються для виконання подальших арифметико-логічних операцій або команд умовних переходів.

Після включення живлення, а також після скидання мікроконтролера, в залежності від налаштування фюзесів, до лічильника програм автоматично завантажується початкова адреса Application сектору (0x000) або Boot сектору (розміщення та об’єм можуть модифікуватися за допомогою фюзесів). За цією адресою розташовується команда переходу до фрагменту ініціалізації (Application сектор), або програми завантаження (Boot сектор).

При виникненні переривання в лічильник команд завантажується адреса відповідного вектора переривання. Область векторів переривань може розташовуватися, в залежності від налаштувань мікроконтролера, як в Application, так і в Boot секторах. Якщо в програмі використовуються переривання, за адресами векторів повинні розміщатися команди переходу до відповідних підпрограм обробки переривань.

2.3. Схеми синхронізації та скидання мікроконтролера

Система формування тактових імпульсів мікроконтролера

Структурна схема системи формування тактових імпульсів мікроконтролера ATmega16 зображена на рис. 2.6.

У мікроконтролера ATmega16 може формуватися 5 тактових послідовностей:

- CLK_{CPU} – базова тактова послідовність, що забезпечує роботу арифметико-логічного пристрою та роботу з пам'яттю мікроконтролера;
- CLK_{IO} – тактова послідовність, що використовується модулями вводу/виводу, таймерами, послідовними інтерфейсними блоками, системою переривань мікроконтролера;
- CLK_{FLASH} – тактова послідовність, що використовується для керування роботою Flesh інтерфейсу;
- CLK_{ADC} – тактова послідовність, що використовується аналогово-цифровим перетворювачем (АЦП). Введення окремої лінії тактування дозволяє заблокувати решту тактових послідовностей для зменшення впливу завад при роботі АЦП;
- CLK_{ASY} – асинхронна тактова послідовність, що формується таймером/лічильником Т2 та дозволяє виконувати ряд програмних завдань в режимі реального часу, навіть при заблокованій послідовності CLK_{CPU};

Існує можливість блокування тактових послідовностей, що забезпечує зменшення енергоспоживання мікроконтролера.

У мікроконтролера mega16 існує 7 джерел тактових послідовностей. П'ять з цих джерел формують базові послідовності, що використовуються безпосередньо при виконанні програми. До них відносяться наступні джерела:

- зовнішній RC- генератор;
- зовнішня тактова послідовність;

- кристалічний резонатор;
- низькочастотний кристалічний резонатор;
- внутрішній калібрований RC-генератор.

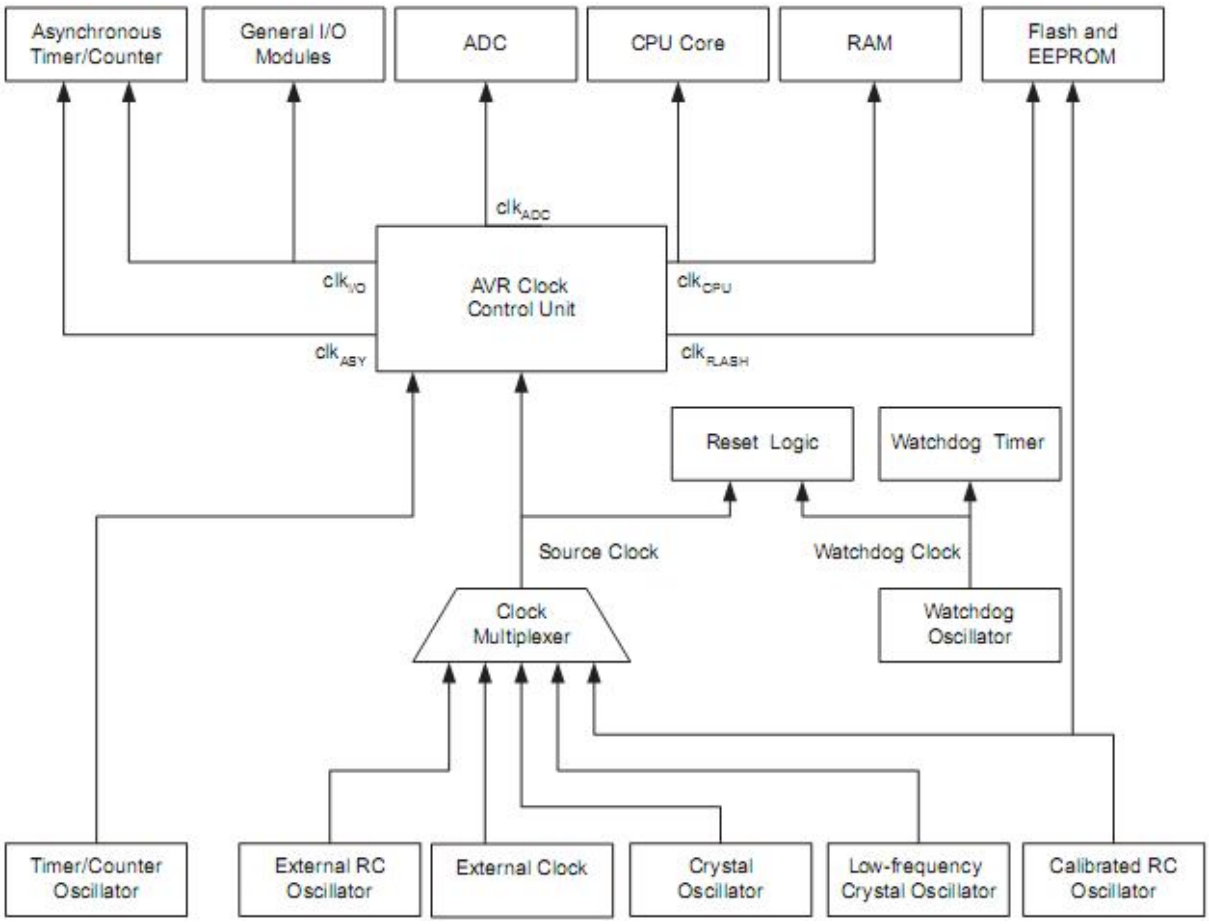


Рис.2.6 Система формування тактових імпульсів мікроконтролера ATmega16

Додаткові два джерела є асинхронними, по відношенню до генераторів базових послідовностей, та можуть застосовуватись для формування вторинних функцій мікроконтролера:

- асинхронний генератор тактової послідовності таймера/лічильника T2. Використовує незалежний кварцовий резонатор. Цей генератор адаптовано для частоти резонатора 32768Гц;
- незалежний асинхронний генератор сторожового таймера.

Дистрибутив мікроконтролера поступає від виробника налаштованим для роботи з внутрішнім каліброваним RC- генератором частотою 1МГц.

Підключення певних джерел тактових сигналів проводиться на етапі програмування мікроконтролера за рахунок встановлення фюзесів CKSEL0 – CKSEL3 та SUT0 – SUT1. На рис. 2.7 приведено заставку програматора PonyProg, що відповідає режиму програмування біт конфігурації та захисту.

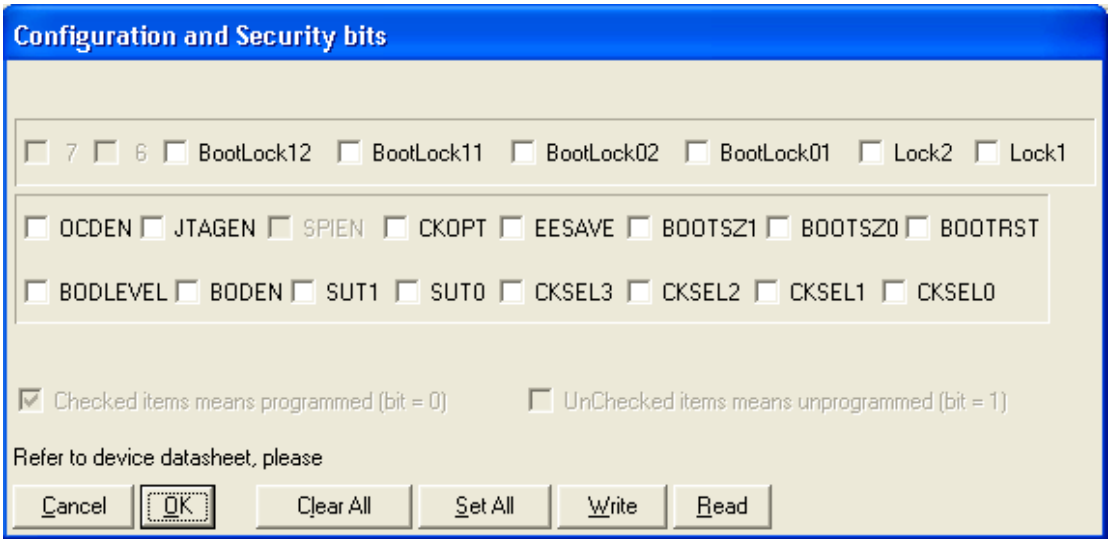


Рис.2.7

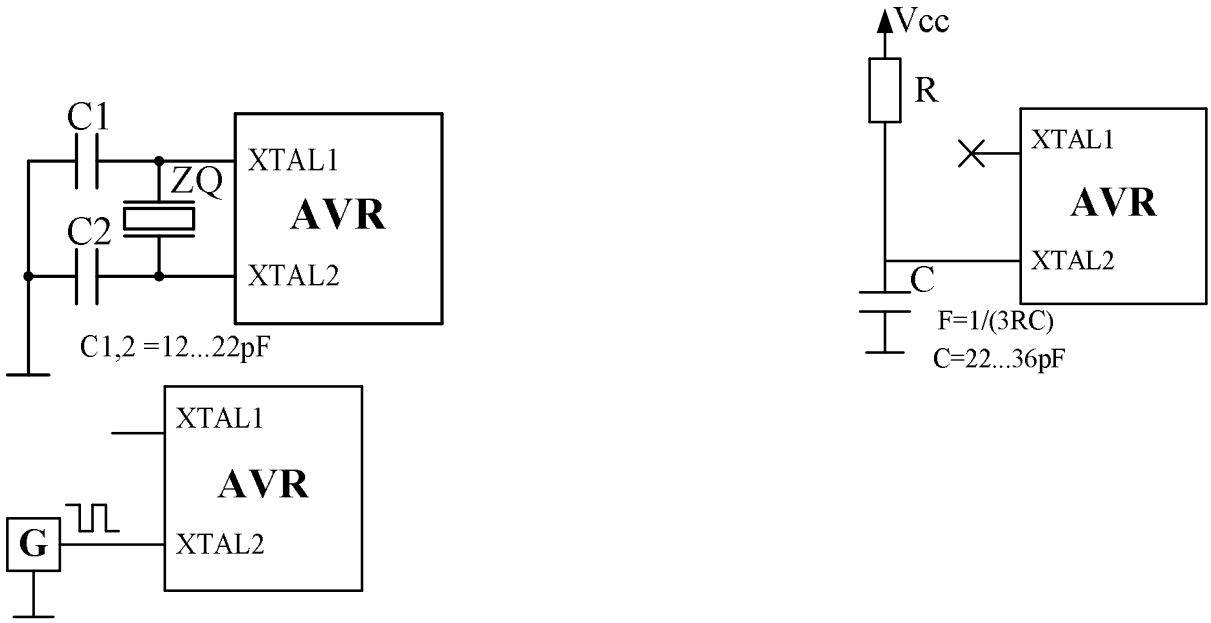
У табл. 2.8 приведено відповідність фюзесів програмування CKSEL0 – CKSEL3 відповідним джерелам тактових сигналів.

Таблиця 2.8

ДЖЕРЕЛА ТАКТОВИХ СИГНАЛІВ	CKSEL3...0
Зовнішній кристалічний / керамічний резонатор	1111 - 1010
Зовнішній низькочастотний кварц	1001
Зовнішній *RC* генератор	1000 - 0101
Внутрішній *RC* генератор	0100 - 0101
Зовнішнє джерело сигналів	0000

Схеми підключення джерел тактових імпульсів наведено на рис. 2.8. Конкретні значення фюзесів CKSEL0 – CKSEL3 та SUT0 – SUT1 визначаються у технічному описі мікроконтролера, згідно з обраним джерелом тактових імпульсів та режимом роботи.

У разі вибору зовнішнього керамічного/кристалічного резонатора значення фюзесів CKSEL0 – CKSEL3 обирають згідно табл. 2.9.



а) б) с)

Рис. 2.8. Схеми синхронізції (а - зовнішній кварцовий резонатор, б –RC ланцюг, в - зовнішній тактовий генератор

Таблиця 2.9

CKOPT	**CKSEL3...1**	**Діапазон частот (МГц)**	**Діапазон номіналів C1 та C2 (пФ)**
1	101[1]	0.4 – 0.9	-
1	110	0.9 – 3.0	12 – 22
1	111	0.3 – 8.0	12 – 22
0	101, 110, 111	1.0 ≤	12 – 22

Біти SUT0 – SUT1 та CKSEL0 обираються згідно табл. 2.10.

Таблиця 2.10

CKSEL0	**SUT1…0**	**Час запуску після виключення або режиму енергозбереження**	**Додаткова затримка після перезапуску Vcc = 5.0 V**
0	00	258 CK[1]	4.1 ms
0	01	258 CK[1]	65 ms
0	10	1K CK[2]	-
0	11	1K CK[2]	4.1 ms
1	00	1K CK[2]	65 ms
1	01	16K CK	-
1	10	16K CK	4.1 ms
1	11	16K CK	65 ms

При роботі з внутрішнім *RC*-генератором номінальне значення частоти встановлюється у відповідності з даними табл. 2.11.

Таблиця 2.11

CKSEL3...0	Номінальне значення частоти (МГц)
101[1]	1.0.
0010	2.0
0011	4.0
0100	8.0

За допомогою калібрувального регістру OSCCAL встановлене значення частоти внутрішнього *RC*-генератора можна корегувати. На рис. 2.9 приведено бітову упаковку цього регістру, а у табл. 2.11 вказані орієнтовні значення вмісту цього регістру для ряду робочих частот.

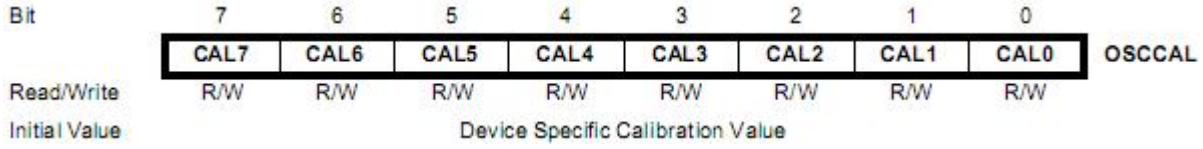


Рис.2.9

Таблиця 2.11

Значення OSCCAL	Мінімальна частота у відсотках від номінальної частоти (%)	Максимальна частота у відсотках від номінальної частоти (%)
$00	50	100
$7F	75	150
$FF	100	200

Система скидання

Система скидання забезпечує початкове дистрибутивне налаштування основних вбудованих ресурсів мікроконтролера.

Розрізняють два типа скидання мікроконтролера – холодне та гаряче.

При реалізації холодного скидання регістри керування мікроконтролера переводяться в дистрибутивний стан. У вказівник пам'яті програми заноситься адреса вектора скидання. У регістри SRAM пам'яті даних заноситься довільне значення. Такий тип скидання використовується при включенні мікроконтролера.

При реалізації гарячого скидання мікроконтролер знаходиться у ввімкненому стані. Як і в попередньому випадку регістри керування мікроконтролера переводяться в дистрибутивний стан. Проте зберігається інформація у регістрах SRAM пам'яті. Структурну схему системи скидання зображено на рис.2.10.

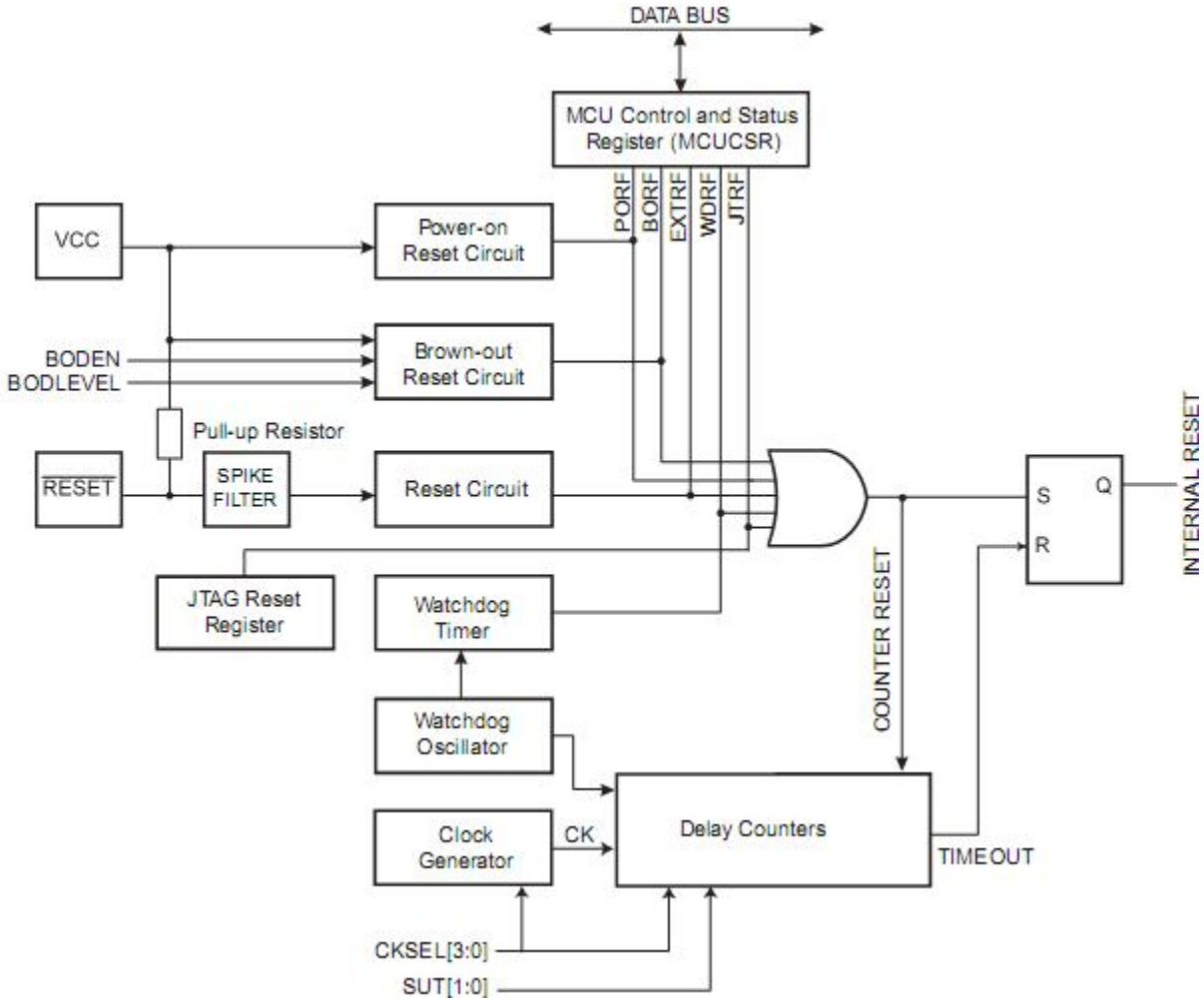


Рис.2.10

Адреса вектора скидання визначається значенням фюзесу BOOTRST (рис. 2.7). Якщо цей фюзес не запрограмований, то вектор скидання

розташовується на початку пам’яті програм (за адресою 0x0000). У випадку його програмування розташування вектора скидання визначається двома фюзесами, що визначають розміри BOOT- сектору – BOOTSZ0 та BOOTSZ1. Можливі чотири варіанти налаштування мікроконтролера Mega16 з розташуванням вектора скидання за адресами 0x1C00, 0x1E00, 0x1F00, 0x1F80.

Джерела скидання

Мікроконтролер ATmega16 має 5 незалежних джерел скидання:

- скидання при подачі напруги джерела живлення (Power-on-Reset);
- зовнішнє скидання;
- скидання, що обумовлене дією сторожового таймера;
- скидання, що обумовлене дією супервізора напруги живлення;
- скидання по інтерфейсу JTAG.

Скидання при подачі напруги живлення

На рис.2.11 зображено часові діаграми, що пояснюють дію такого типу скидання.

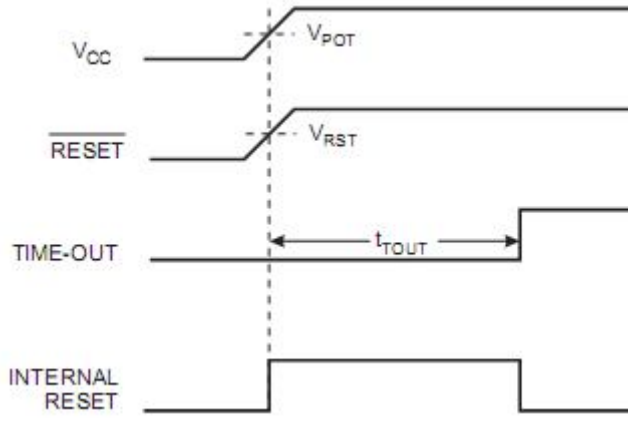


Рис.2.11

При подачі напруги живлення проводиться аналіз величини напруги. У разі досягнення цією напругою рівня U_{POT} вмикається внутрішній

генератор затримки, який формує тривалість імпульсу скидання (t_{TOUT}). Сигнал внутрішнього скидання формується на інтервалі часу, що відповідає напрузі живлення меншій за U_{POT}, та додатковому інтервалі, який формує генератор затримки. Тривалість інтервалу затримки можна регулювати за рахунок зміни фюзесів програмування SUT0 – SUT1 (табл.2.10). Таке налаштування необхідне для узгодження динамічних властивостей джерела живлення мікроконтролера та системі скидання.

Зовнішнє скидання

На рис.2.12 зображено часові діаграми, що пояснюють дію зовнішнього типу скидання.

Зовнішнє скидання викликається внаслідок подачі на вивід *RST* мікроконтролера сигналу низького рівня. Після досягнення позитивним фронтом сигналу скидання рівня U_{RST} вмикається внутрішній генератор затримки, який формує додаткову тривалість імпульсу скидання (t_{TOUT}).

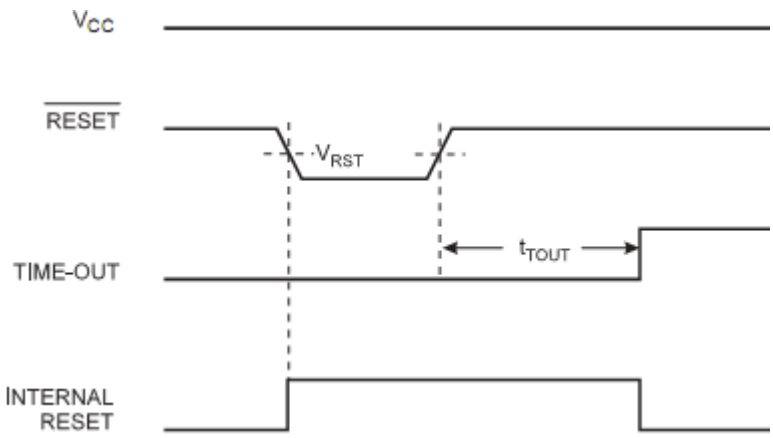


Рис.2.12

Схема формування сигнала зовнішнього скидання приведена на рис. 2.13. Зазначимо, що хоча в МК **AVR** є своя внутрішня схема скидання, а сигнал **RESET** підтянутий резистором в 100кОм к **Vcc**, але внаслідок чутливості такої схеми до завад.. рекомендується **RST** подтягнути до лінії живлення резистором в 10к.

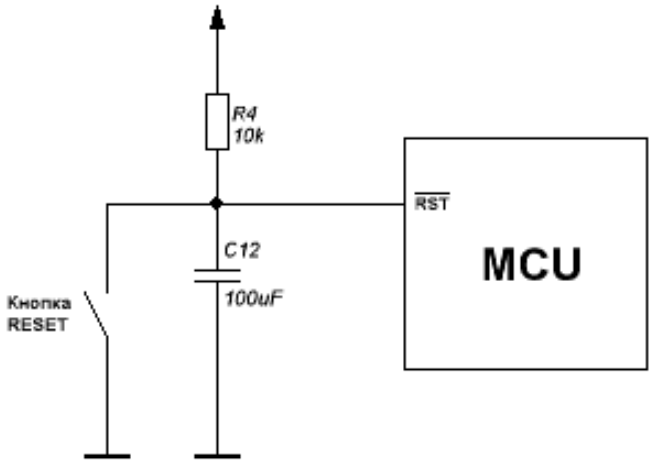


Рис. 2.13

Скидання під дією супервізора напруги джерела живлення

Супервізор напруги використовується для того, щоб уникнути некоректної роботи мікроконтролера у разі провалів напруги живлення. У мікроконтролера використовується вбудований супервізор напруги (BOD-Brown-Out-Detection) який при програмуванні фюзесу BODEN (рис. 2.7) може вмикатися. За рахунок відповідного програмування фюзесу BODLEVEL можлива зміна рівня напруги спрацювання супервізора. Для не програмованого фюзесу (BODLEVEL=1) напруга спрацювання складає 2,7В. У разі його програмування (BODLEVEL=0) напруга спрацювання складає 4,2В. На рис.2.14 зображено часові діаграми, що пояснюють дію супервізора напруги.

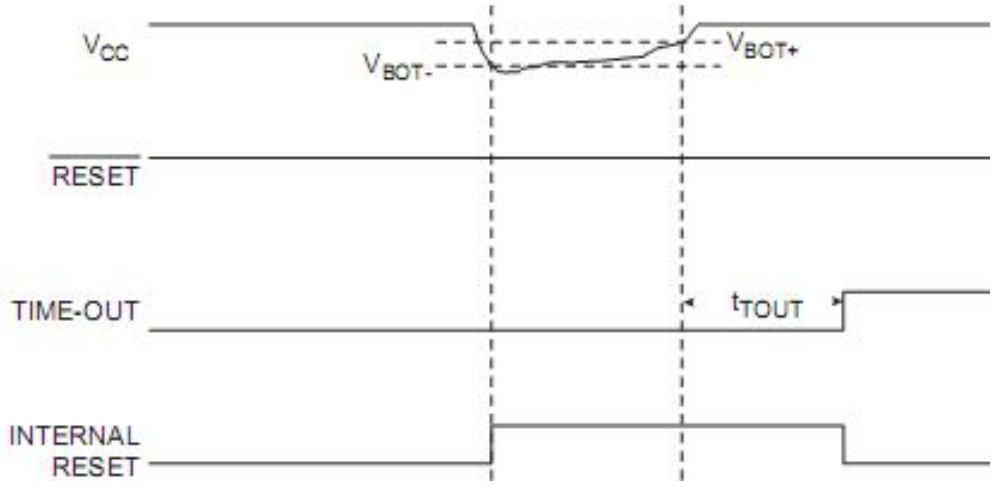


Рис.2.14

У разі зменшення напруги живлення до рівня U_{BOT-} починає формуватися імпульс внутрішнього скидання. Якщо напруга живлення підвищується до рівня U_{BOT+} включається внутрішній генератор затримки, який формує додаткову тривалість імпульсу скидання (t_{TOUT}).

Такий варіант скидання мікроконтролера застосовується у разі використання внутрішньої енергонезалежної EEPROM - пам'яті даних. Він забезпечує захист від спотворення даних.

Скидання під дією сторожового таймера

Сторожовий таймер WDT є неодмінним атрибутом усіх сучасних мікроконтролерів. Він використовується для захисту від збоїв програми. При включенні сторожового таймеру через запрограмовані інтервали часу формуються імпульси скидання мікроконтролера. Формування таких імпульсів уникають за рахунок програмного скидання лічильника інтервалу часу. Сторожовий таймер має свій власний *RC*-генератор, що працює навіть під час знаходження мікроконтролера в режимі Power Down. Типове значення частоти дорівнює 1 МГц, при напрузі живлення Vcc = 5.0 В, і 350 кГц, при Vcc = 3.0 В.

Структурну схему сторожового таймера зображено на рис. 2.15, а на рис.2.16 зображено часові діаграми, що пояснюють дію цього типу скидання.

У разі спрацювання системи скидання сторожового таймера формується короткий імпульс запуску ($WDT_{TIME-OUT}$) внутрішнього генератора затримки та починає формуватися внутрішній імпульс скидання. Цей імпульс скидання завершується в кінці інтервалу додаткової тривалості (t_{TOUT}). Тривалість інтервалу повторних спрацювань сторожового таймера встановлюється за допомогою регістра керування WDTCR (рис. 2.17) згідно з даними табл. 2.12.

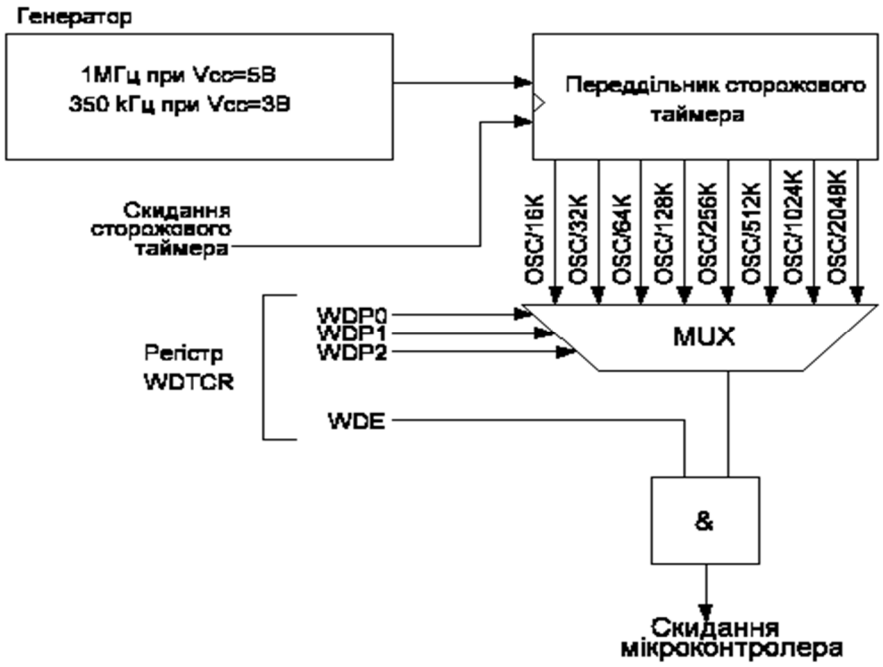


Рис. 2. 15.Структурна схема сторожового таймера

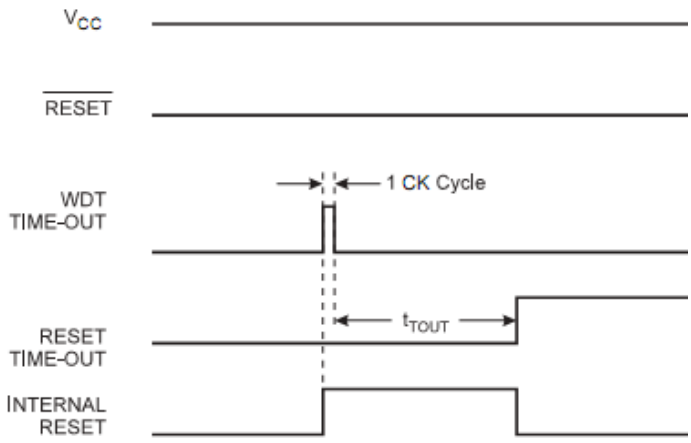


Рис.2.16

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

Рис.2.17

Для вмикання/вимикання сторожового таймера використовують два розряди регістра WDTCR – WDE І WDTOE. Якщо розряд WDE встановлений в «1», сторожовий таймер ввімкнено, якщо скинуто в «0» -

вимкнений. Безпосередньо перед ввімкненням таймера рекомендується також виконувати його скидання командою WDR. Вимкення сторожового таймера (скидання розряду WDE) можна здійснити тільки при встановленому розряді WDTOE.

Причому через 4 машинні цикли після установки в «1» цей розяд апаратно скидається, завдяки чому фактично зникає можливість випадкового вимкнення сторожового таймера.

Для виключення сторожового таймера рекомендується наступна послідовність дій:

- однією командою записати логічну «1» в розряди WDE I WDTOE;
- протягом наступних чотирьох машинних циклів записати «0» в розряд WDE.

Період затримки спрацювання сторожового таймера задається з допомогою розрядів WDP2…..WDP0 регістру WDTCR згідно з табл. 2.12.

Щоб уникнути випадкового скидання мікроконтролера при зміні періоду сторожового таймера, необхідно перед записом розрядів WDP2:WDP0 або заборонити роботу сторожового таймера або скинути його.

Таблиця 2.12

WDP2	**WDP1**	**WDP0**	**Кількість періодів генератора WDT**	**Час спрацьовування WDT при Vcc=3.0V**	**Час спрацьовування WDT при Vcc=5.0V**
0	0	0	16K (16,384)	17.1 ms	16.3 ms
0	0	1	32K (32,768)	34.3 ms	32.5 ms
0	1	0	64K (65,536)	68.5 ms	65 ms
0	1	1	128K (131,072)	0.14 s	0.13 s
1	0	0	256K (262,144)	0.27 s	0.26 s
1	0	1	512K (524,288)	0.55 s	0.52 s
1	1	0	1,024K (1,048,576)	1.1 s	1.0 s
1	1	1	2,048K (2,097,152)	2.2 s	2.1 s

У системі команд асемблеру AVR мікроконтролерів використовується команда «wdr» для скидання лічильника сторожового таймера в початковий стан. В інтегрованому середовищі *Image Craft ICC v*7, для цієї мети використовується макрос _WDR().

Нижче, у прикладі 2.1 приведено функцію вимикання сторожового таймера.

***Приклад* 2.1**

```
      //---------- Функція вимикання сторожового таймера
void WDT_OFF(void)
{
      _WDR();  // скидання лічильника сторожового таймера
WDTCR |= (1<<WDTOE) | (1<<WDE);
      WDTCR = 0x00;
      }
//---------------------------
```

Контрольні питання

1. Загальна характеристика мікропроцесорів сімейства AVR.
2. Огляд максимальних ресурсів мікропроцесорів.
3. Різновиди процесорів AVR.
4. Архітектура AVR-мікроконтролерів Mega 16. Структурна схема, основні блоки та сигнали.
5. Що являє собою арифметико-логічний пристрій ATmega16
6. Яке призначення бітів регістра стану SREG?
7. Які існують джерела тактових послідовностей у мікроконтролері mega16.
8. Наведіть схему приєднання тактового генератора.
9. Настроювання джерела тактового сигналу?
10. Принцип роботи системи скидання.
11. Які джерела скидання передбачені в мікроконтролері ATmega16?
12. Поясніть дію скидання при подачі напруги живлення.
13. Поясніть дію зовнішнього скидання.

РОЗДІЛ 3. ОРГАНІЗАЦІЯ ПАМ`ЯТІ AVR МІКРОКОНТРОЛЕРІВ

3.1. Організація пам`яті програм та даних

Пам’ять мікроконтролерів AVR організована за Гарвардською архітектурою, в якій розділені адресні простори пам’яті програм та пам’яті даних, а також і шини доступу до них. Загальна карта розподілу пам’яті мікроконтролера відповідає програмній моделі, яку зображено на рис. 2.4.

Пам’ять програм

Пам’ять програм (FLASH) призначена для зберігання команд, що керують роботою мікроконтролера. В пам’яті програм зберігаються також різні константи, що не змінюються під час роботи програми.

Усі AVR мікроконтролери мають вбудовану флеш-пам'ять програм, що може бути завантажена як за допомогою звичайного паралельного програматора, так і за допомогою SPI-інтерфейсу, безпосередньо на цільовій платі. Число циклів перезапису залежить від типу мікроконтролера та знаходиться у діапазоні від 1000 до 25000. Час зберігання інформації (Retention Time) також залежить від типу мікроконтролера і для сучасних мікросхем доходить до 25 років.

Пам’ять програм виконана на базі пам’яті флеш-типу з електричним стиранням інформації. Оскільки довжина команди складає 16 біт (деяких – 32 біта), пам’ять програм має 16-розрядну організацію 8К*16 біт. Для адресації пам’яті програм використовується лічильник команд (PC – Program Counter). Його розрядність відповідає об’єму пам’яті програм, для мікроконтролера Atmega16 складає 13 біт. Лічильник команд вказує на адресу двобайтових слів. Відповідно мітки у програмах, які написані мовою асемблеру, також розраховані на таке пакування слів команд. Записи констант в таблиці мають байтове пакування. Тому при завантаженні констант, що розміщуються в таблицях, необхідно подвоювати адресу мітки, щоб переходити від адреси двобайтових слів до

фізичної адреси байта. При написанні програм мовою Сі ця особливість враховується автоматично при компіляції.

На рис.3.1 зображено діаграму розподілу пам’яті програм.

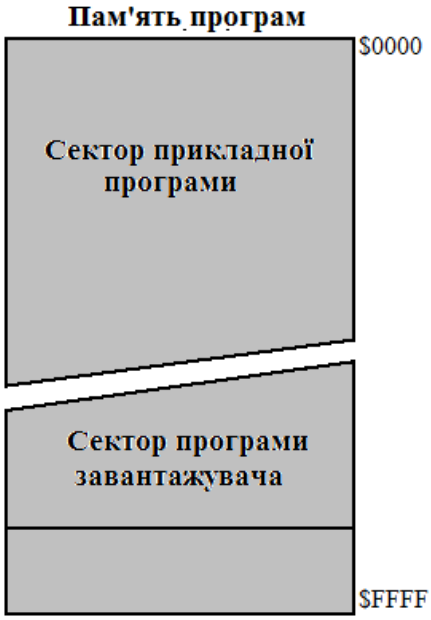


Рис.3.1

Пам'ять програм розділена на дві частини – сектор програми завантаження (Boot), та сектор прикладної програми. У Boot – секторі може розташовуватися програма, що дозволяє проводити самопрограмування, як області прикладної програми, так і частин програми завантаження мікроконтролера. Перерозподіл розмірів секторів відбувається під час програмування, за рахунок встановлення фюзесів BOOTSZ0, BOOTSZ1.

За адресою $000 пам’яті програм знаходиться вектор скидання. Після ініціалізації (скидання) мікроконтролера виконання програми починається з цієї адреси. Існує можливість переносу адреси вектора скидання на початок Boot-сектора (встановлюється фюзес програмування BOOTRST). Для мікроконтролерів з об’ємом пам’яті програм що перевищує 8 кБайт, за цією адресою розміщують команду переходу rjmp до частини ініціалізації

програми). Починаючи з адреси $002 у мікроконтролера ATmega16 розташовується таблиця векторів переривань.

Використовують наступні способи ініціалізації констант у пам'яті програм.

Якщо адреса розміщення неважлива, то застосовують варіанти, що наведені у прикладі 3.1.

***Приклад* 3.1**

```
const int  Con1  = 1;
const char Con_Mas1[] = {1,2,3,4,5,6,7,8,9,10};
```

Якщо необхідно жорстко визначити адресу, то використовують варіант опису, який приведено у прикладі 3.2. Недоліком такого варіанту є можливість перетину фрагментів основної програми з кодами команд та описом констант. У випадку такої події у вікні повідомлень деяких Сі-компіляторів може бути відсутня інформація про помилку. У пам'яті програм розміщуються ті коди, які визначені останніми.

***Приклад* 3.2**

```
#pragma abs_address:0x1B0
const char Titul[] = "Designed by _______09.2016";
#pragma abs_address:0x1E0
const char Name[] = "Serial number";
#pragma abs_address:0x1F0
const unsigned int Serial_number =0x0000;
#pragma end_abs_address
```

3.2. Пам'ять даних AVR мікроконтролерів

Пам'ять даних складається з двох областей:

- статичний запам'ятовуючий пристрій (SRAM);
- енергонезалежна пам'ять даних на основі EEPROM.

Кожна з цих областей розміщена в своєму адресному просторі, мають власні вказівники та шини адреси.

На рис. 3.2 зображено структуру адресного простору SRAM – області. У складі області виділяють наступні сегменти:

- регістровий файл (R0 – R31);

- регістри вводу/виводу (РВВ) 64 - 128, у залежності від моделі мікроконтролера;
- регістри оперативного запам’ятовуючого пристрою (ОЗП).

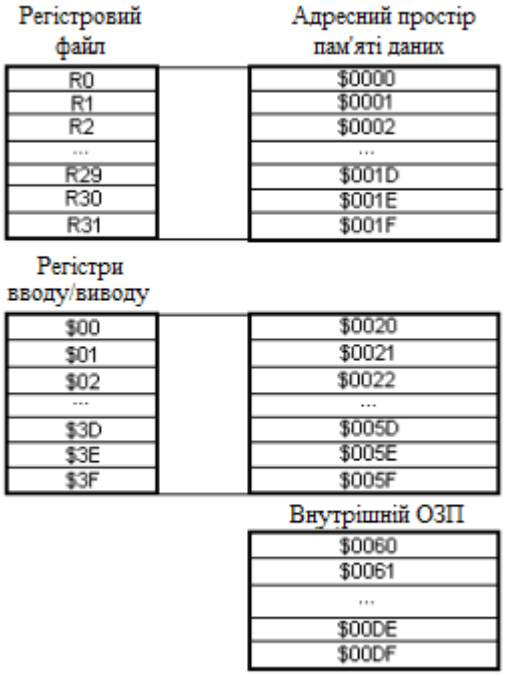


Рис. 3.2.

Статичний запам’ятовуючий пристрій (*SRAM*)

Регістровий файл включає 32 регістра загального призначення, що мають символічні імена R0 – R31.

В області регістрів вводу/виводу розташовані службові регістри, а також регістри керування пристроями мікроконтролера. Розміри області РВВ залежать від типу мікроконтролера та складають від 64 до 128 байт. Для мікроконтролера Atmega16 РВВ має об’єм у 64 байти.

Для зберігання змінних разом з регістрами файлового регістру також можуть використовуватися регістри ОЗП. Цей сегмент пам’яті використовується також для створення стекової області.

Для зберігання даних, які можуть змінюватися в процесі налагодження та функціонування готової системи, може використовуватися енергонезалежна пам’ять даних - EEPROM. Ця пам’ять

розташована в окремому адресному просторі. Доступ до неї здійснюється за допомогою спеціальних РВВ.

Регістри загального призначення (регістровий файл) та регістри ОЗП SRAM області можуть використовуватися для зберігання змінних.

Крім того регістри SRAM області використовуються для створення стеку до якого записуються адреси точок повернення у разі використання функцій та підпрограм. До стеку можуть також записуватися значення змінних.

Шість із 32-х регістрів файлу можуть використовуватися як три 16-розрядних покажчики адреси при непрямому адресуванні даних (рис. 3.3). Один із цих покажчиків (*Z Pointer*) застосовується також для доступу до даних, записаних у пам'яті програм мікроконтролера. Використання трьох 16-бітних покажчиків (*X, Y* і *Z Pointers*) істотно підвищує швидкість пересилання даних при роботі прикладної програми.

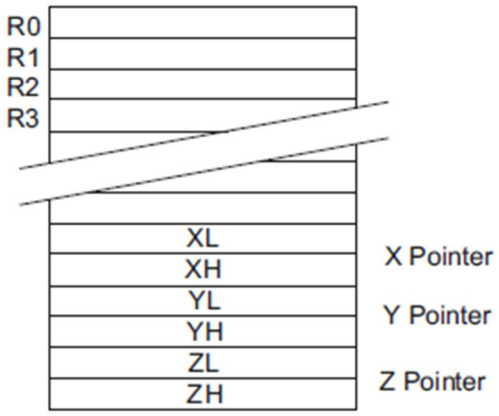


Рис.3.3. Регістровий файл

У мікроконтролерах AVR усі 32 РЗП безпосередньо доступні АЛП на відміну від мікроконтролерів інших фірм, в яких є лише один такий регістр – акумулятор. Завдяки цьому будь-який РЗП може використовуватись в усіх командах і як операнд-джерело, і як операнд-передавач. Виняток складають арифметичні та логічні команд, що

виконують дії між константою та регістром (SBCI, SUBI, CPI, ANDI, ORI), команда завантаження константи у регістр (LDI) та команди операцій з бітами (SBR, CBR). Ці команди можуть звертатися лише до другої половини файлового регістру (*R*16…*R*32). Команди 16-розрядного додавання з константою ADIW і віднімання константи SBIW у якості першого операнду використовують тільки регістри *R*24, *R*26, *R*28, *R*30.

Для резервування місця для змінної необхідно вказати її тип та ім'я.

Ініціалізація дистрибутивного значення змінної відбувається після присвоєння їй певного значення. Якщо операція ініціалізації у програмі відсутня, то змінним автоматично присвоюється нулеві значення (компілятор ImageCraft ICC).

***Приклад* 3.3**

```
unsigned char  var1;     //резервування місця для змінної
unsigned int   var2=9;   //ініціалізація змінної
```

Місце розташування змінної у файловому регістрі чи SRAM-сегменті залежить від її типу модифікаторів доступу до змінної (register, volatile).

Якщо при визначенні змінної тип не вказано, або використано модифікатор register, то резервується місце у сегменті файлового регістру.

У разі застосування модифікатору volatile використовуються файлові регістри R0/R1/R2/R3/R4/R5/R6/R7/R8/R9/R24/R25/R26/R27/R30/R31. Зміст такої змінної може змінюватися у функції, що викликається. Така змінна не запам`ятовується та не поновлюється. Розташована змінна у SRAM-сегменті.

У мікроконтролері Atmega16 регістри вводу/виводу розташовуються у просторі розміром 64 байта.

Для обміну інформацією з регістрами вводу виводу використовують способи, що наведені у прикладі 3.4.

***Приклад* 3.4**

```
unsigned char TMP;  //визначення змінної TMP
```

```
PORTA=4;              //запис у регістр PORTA числа «4»
TMP=SREG;             //читання регістру вводу/виводу SREG
```

Енергонезалежна пам’ять даних

Пам’ять EEPROM розташована в своєму адресному просторі та має лінійну організацію. Спеціальних команд звертання до EEPROM немає.

Ініціалізація області EEPROM відбувається з використанням pragma-визначень (компілятор ImageCraft ICC). Для ініціалізації EEPROM визначають символічні імена відповідних змінних та їх вміст. У прикладі 3.5 проілюстрована ініціалізація EEPROM.

***Приклад* 3.5**

```
#pragma data:eeprom
unsigned char ee_var1 = 10;
unsigned int  ee_var2 = 0x100;
#pragma data:data
```

Після компіляції проекту у цьому випадку генерується вихідний файл <output file>.eep.

Читання й запис комірок EEPROM виконується за допомогою регістрів введення/виведення - EEAR (регістр адреси), EEDR (регістр даних) і EECR (регістр керування).

До регістру адреси завантажується адреса комірки, до якої будуть звертатися. У залежності від ємності цього сегменту пам’яті регістр адреси має різну кількість байт. Так для мікроконтролера ATmega16 регістр адреси складається з двох байт – EEARH та EEARL (рис. 3.4.)

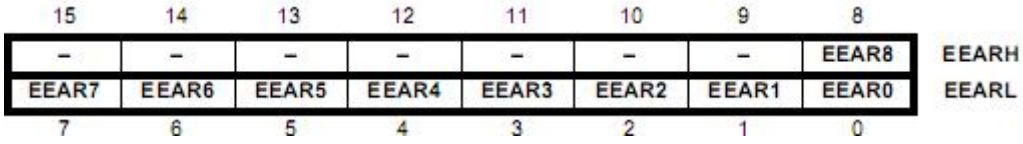


Рис. 3.4.

Регістр даних EEPROM - пам’яті, називається EEDR. При запису в цей регістр завантажуються дані, які повинні розміщатися в EEPROM за адресою, що знаходиться в регістрах EEARH:EEARL. При зчитуванні в

цей регістр поступають дані записані в EEPROM за адресою, що знаходиться в регістрах EEARH:EEARL (рис. 3.5).

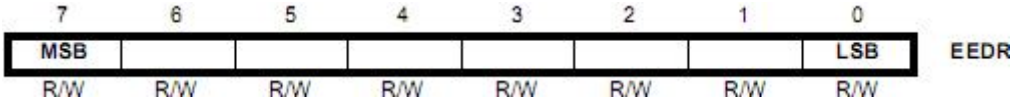


Рис. 3.5.

Регістр керування використовується для управління доступом до EEPROM - пам’яті. Цей регістр називається EECR (рис. 3.6).

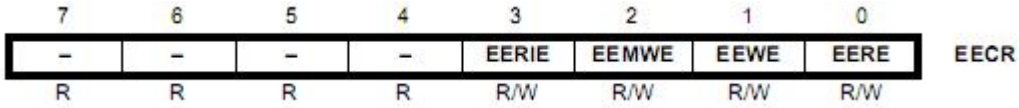


Рис. 3.6.

Окремі біти цього регістру мають наступне значення:

- EERE - дозвіл читання з EEPROM;
- EEWE - дозвіл запису до EEPROM;
- EEMWE – керування дозволом запису до EEPROM;
- EERIE - дозвіл переривання від EEPROM.

Для звертання до змінних, що розташовані у EEPROM області, можна використовувати бібліотечні функції та макроси, або необхідно розробити власні функції.

Якщо використовуються бібліотечні функції, то для їх виклику необхідно підключити до проекту вкладений файл “eeprom.h” (компілятор ImageCraft ICC). Для запису/читання змінних int-типу використовують функції:

- EEPROM_WRITE(int location, object) – запис двох байт.

Наприклад,

```
unsigned int i;
EEPROM_WRITE(0x1, i);    // запис  змінної “i”за адресою 0x1
```

- EEPROM_READ(int location, object) – читання двох байт.

Наприклад,

```
unsigned int i;
EEPROM_READ(0x1, i);     // читання 2 байт у змінну "i"
```

Для запису/читання змінних char-типу використовують внутрішні функції:

- unsigned char EEPROMread(int location) – читання байту з EEPROM за адресою location;
- int EEPROMwrite(int location, unsigned char byte) – запис байту за адресою location та повернення «0» у разі успіху.

У деяких випадках можлива розробка власних функцій читання/запису для роботи з EEPROM пам'яттю. Наприклад, бібліотечні функції не підтримують роботу з змінними, формат яких складає 4 байти. У прикладах 3.6, 3.7 приведені функцій запису/читання байту, які описані в технічному описі мікроконтролера ATmega8.

***Приклад* 3.6**

```
;-----------------------------------------------------
;Функція запису у EEPROM
;-----------------------------------------------------
void EEPROM_write (unsigned int uiAddress, unsigned char
ucData)
{
while (EECR & (1<<EEWE));     //очікування закінчення
//попереднього звертання
     EEAR = uiAddress;        //встановлення адреси EEPROM
     EEDR = ucData;           //встановлення даних EEPROM
     EECR |= (1<<EEMWE);      //запуск майстер - стробу
     EECR |= (1<<EEWE));      //запуск запису
}
```

***Приклад* 3.7**

```
;-----------------------------------------------------
;Функція читання з EEPROM
;-----------------------------------------------------
unsigned char EEPROM_read (unsigned int uiAddress)
{
while (EECR & (1<<EEWE));     //очікування закінчення
//попереднього звертання
EEAR = uiAddress;        //встановлення адреси EEPROM
EECR |= (1<<EERE);       //дозвіл читання
```

```
return EEDR;                //повернення даних
}
```

У прикладі 3.8 приведено програму в якій проводиться ініціалізація константи у FLASH - сегменті та змінних у SRAM-області і EEPROM-сегменті. Для розробки програми використовувався засіб Application Builder інтегрованого середовища ImageCraft ICC.

***Приклад* 3.8**

У наведеному нижче прикладі проводиться така ініціалізація змінних:

- ee_var1 та ee_var2 визначаються у EEPROM-сегменті;
- c_var1 визначається у FLASH (ROM) - сегменті;
- tmp_SRAM визначається у SRAM-області.

З використанням бібліотечних функцій видобуваються значення змінних, що розташовані у EEPROM-сегменті .

Проводиться математична обробка змінних у відповідності з наступним виразом

```
result = c_var1*(tmp_ee1+tmp_ee2)-tmp_SRAM.
```

```
//-------------------------------------------------------------
--
//ICC-AVR application builder: 11.02.2016 11:45:40
// Target: M16
// Crystal: 10.000Mhz
//-------------------------------------------------------------
--
#include <iom16v.h>
#include <macros.h>
#include <eeprom.h>
//-------------------------------------------------------------
--//ініціалізація змінних у EEPROM
#pragma data:eeprom
unsigned char ee_var1 = 100;
unsigned int  ee_var2 = 0x100;

#pragma data:data
//-------------------------------------------------------------
//ініціалізація константи  у FLASH (ROM)
const char c_var1 = 2;
//-------------------------------------------------------------
void port_init(void)
{
PORTA = 0xC0;               //ініціалізація портів
DDRA  = 0xF0;
PORTB = 0x00;
DDRB  = 0x00;
```

```
PORTC = 0x00;
DDRC  = 0x00;
PORTD = 0x00;
DDRD  = 0x00;
}
//-----------------------------------------------------------
--
Void init_devices(void)
{
CLI();                      //disableallinterrupts
port_init();                //ініціалізація портів
MCUCR = 0x00;
GICR  = 0x00;
TIMSK = 0x00;               //timerinterruptsources
SEI();                      //re-enableinterrupts
}
//-----------------------------------------------------------
--
//Головна функція
//------------------
void main(void)
{
//--------------------------
init_devices();          //Ініціалізація мікроконтролера

//--------------------------
while(1)             //Головний цикл
 {
unsigned int result,tmp_ee1,tmp_ee2;
unsigned int tmp_SRAM = 10;
tmp_ee1 = EEPROMread(ee_var1);  //читання char-змінної ee_var1
EEPROM_READ((int)&ee_var2,tmp_ee2);//читання        int-змінної
ee_var2
result=c_var1*(tmp_ee1+tmp_ee2)-tmp_SRAM;         //математична
обробка
};
//--------------------------
}

//-----------------------------------------------------------
```

Контрольні питання

1. Організація пам`яті AVR-мікроконтролерів.
2. Карта розподілу пам`яті.
3. Особливості організації пам’яті програм AVR мікроконтролерів.
4. Яким чином відбувається адресація пам’яті програм?

5. Особливості організації пам’яті статичного запам’ятовуючого пристрою.

6. Призначення та склад регістрів введення виведення. Команди роботи з ними.

7. Призначення та склад файлового регістру. Приклади команд роботи з ним.

8. Від чого залежить місце розташування змінної у файловому регістрі чи SRAM- сегменті?

9. Енергонезалежна пам'ять даних. Приклади команд роботи з нею.

10. Як відбувається ініціалізація області EEPROM?

11. Організація стеку. Наведіть фрагмент програми ініціалізації стеку.

12. Які регістри призначені для читання та запису комірок EEPROM?

13. Призначення регістру адреси EEPROM?

14. Призначення регістру даних EEPROM пам’яті?

15. Призначення регістру керування EEPROM - пам’яті?

16. Особливості використання бібліотечних функції та макросів при звертанні до змінних, що розташовані у EEPROM.

РОЗДІЛ. 4. ПЕРІФЕРІЙНІ ПРИСТРОЇ

4.1 Порти вводу/виводу

В залежності від типу МК число ліній портів введення/виведення може бути від 4 до 123. Мікроконтролер ATMega16 має 4 восьмибітові порти (*A*, *B*, *C*, *D*) та відповідно 32 лінії вводу/виводу. Кожна лінія портів може бути використана незалежно як для прийому, так і передачі цифрових сигналів.

Вхідні буфери портів побудовані за схемою тригера Шмідта. Для ліній, що мають конфігурацію вхідних, можливе підключення внутрішнього резистора опором 35…120 кОм між входом та шиною живлення V_{DD}.

Вихідні драйвери портів забезпечують однакові значення максимального струму навантаження для обох значень рівня вихідної напруги ($I_{OL} = I_{OH} = 40$мА).

У мікроконтролері реалізовано алгоритм роботи з портами «читання/модифікація/запис», що дозволяє виконувати ряд операцій пов’язаних з виводом безпосередньо на лініях портів.

Більшість ліній портів, окрім безпосередніх функцій вводу\виводу цифрових сигналів, мають також альтернативні функції. Лінії портів використовуються в альтернативному режимі у випадках застосування відповідних вбудованих контролерів, наприклад таких як TWI, ADC.

Регістри портів

Звертання до портів відбувається через регістри вводу/виводу. В адресному просторі вводу/виводу для кожного порту відведено по 3 регістри:

- DDRx - напрямку роботи (передавач/приймач);
- PORTx – вихідних даних;

- PINx – вхідних даних.

Наприклад, для порту A - DDRA, PORTA, PINA.

Розряди цих регістрів мають назви:

- DDx7… DDx0 – для регістрів DDRx;
- Px7…Px0 – для регістрів PORTx;
- PINx7… PINx0 – для регістрів PINx.

Регістри PINx дозволяють здійснити доступ до фізичних значень сигналів на виводах порту. Відповідно вони доступні лише для читання. Регістри PORTx та DDRx доступні як для читання, так і для запису. Після рестарту мікроконтролера в регістри PORTx та DDRx записуються початкові нулеві значення. Це відповідає режиму приймача.

Структура одного розряду порту введення/виведення наведена на рис. 4.1.

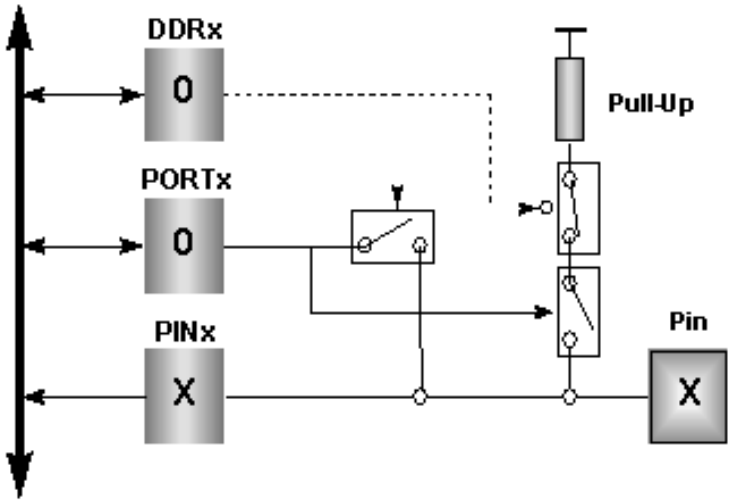


Рис 4.1. Структура одного розряду порту введення/виведення AVR мікроконтролера

Запис у порт означає запис необхідного стану для кожного виводу порту у відповідний регістр даних порту PORTx. А читання стану порту виконується читанням або регістра даних порту PORTx, або регістра виводів порту PINx. При читанні регістра виводів порту PINx відбувається зчитування логічних рівнів сигналів, що присутні на виводах порту. А при

читанні регістра даних порту відбувається зчитування даних, що знаходяться у регістрі PORTx.

Регістри портів розташовані в сегменті регістрів вводу/виводу (32 регістру на початку сегмента), для якого можливі бітові маніпуляції. Це суттєво спрощує роботу з окремими лініями портів. Можливе виконання команд бітових маніпуляцій (наприклад, для порту А):

- встановлення «1» значення на лінії порту PA1

```
PORTA |=(1 << PA1);
```

- встановлення «0» значення на лінії порту PA1

```
PORTA &=~(1 << PA1);
```

- інверсія вихідного значення на лінії порту PA1

```
PORTA ^=(1 << PA1),
```

та розгалуження:

- виконання операцій, якщо на лінії порту PA1 «0» значення

```
if (!(PINA & (1 << PA1))) {};
```

- виконання операцій, якщо на лінії порту PA1 «1» значення

```
if (PINA & (1 << PA1)) {}.
```

Конфігурування портів

Під час конфігурування портів встановлюють такі властивості:

- задають напрямки передачі даних ліній портів (вхід або вихід);
- підключають/відключають внутрішні підтягуючі резистори.

Напрямок передачі даних визначається вмістом регістра передачі даних DDRx. Якщо розряд DDxn цього регістра встановлено в «1», відповідний *n*-вивід порту є виходом. У протилежному випадку (встановлено «0»), відповідний *n*-вивід порту є входом.

Керування підтягуючим резистором здійснюється за допомогою регістра даних PORTx. Якщо розряд Pxn регістра PORTx встановлено в «1» і відповідний вивід порту є входом, між цим виводом та шиною живлення підключається підтягуючий резистор. Для того щоб відключити

підтягуючий резистор, необхідно або скинути відповідний розряд регістра PORTx, або зробити вивід порту виходом.

Приклад ініціалізації порту

Початкова ініціалізація портів є надзвичайно важливою процедурою. Ініціалізація повинна проводитися одразу після рестарту мікроконтролера. Затримка з її проведенням у деяких випадках може привести до виникнення аварійної ситуації у об'єкті керування. Визначення типів ліній портів та їх початкового стану проводиться на підставі аналізу схеми пристрою. Нижче приведено приклад ініціалізації порту А з конфігурацією ліній, що наведено у табл. 4.1. У відповідності до цієї таблиці визначаються константи ініціалізації ini_DDRA та ini_PORTA.

Таблиця 4.1

Лінії порту	**PA7**	**PA6**	**PA5**	**PA4**	**PA3**	**PA2**	**PA1**	**PA0**
Передавач	так	так	так	так	ні	ні	ні	ні
Вихідний рівень	0	1	0	0	-	-	-	-
Приймач	ні	ні	ні	ні	так	так	так	так
Підтяжка	ні	ні	ні	ні	ні	ні	так	так

У функції ініціалізації портів доцільно використовувати символічні імена констант. Це суттєво спрощує модифікацію програми при її адаптації до розробленої печатної плати. Для символічного опису використовують директиву *#define ім'я значення.* Під час опису констант ініціалізації портів доцільно вживати бінарний формат чисел, що дає змогу більш оперативно оцінювати бітовий стан порту.

При розробці схеми мікроконтролера зазвичай користуються шинною технологією опису зв'язків, що суттєво спрощує зовнішній вигляд схеми електричної принципової. Лініям зв'язку виводів портів мікроконтролера з периферійними елементами доцільно давати назви на латиниці, без використання заборонених при програмуванні мовою Сі

символів. При проведенні опису символічних імен, що використовуються у програмі, вказуються такі імена ліній портів. Використання символічних імен ліній дає можливість більш оперативно узгоджувати програмний продукт з друкованою платою як на етапі її розробки, так і модифікації.

***Приклад* 4.1**

Провести ініціалізацію порту А згідно конфігурації ліній порту, яку приведено в табл. 4.1, після чого встановити лінію PA7 в одиничний стан.

```
//ICC-AVR application builder : 13.05.2016 22:28:05
//Target : M16
//Crystal: 10.000Mhz

#include <iom16v.h>
#include <macros.h>
//---------------------------
#define    ini_DDRA    0b11110000 //константи ініціалізації
#define    ini_PORTA   0b01000011 //згідно з даними табл.2.2
#define    LED1        PA7        //символічне ім'я лінії
схеми
//---------------------------
//---------- Функція ініціалізації портів
void port_init(void)              //ініціалізація порту А
{
DDRA  = ini_DDRA;
PORTA = ini_PORTA;
}
//---------------------------
//---------- Функція ініціалізації мікроконтролера
void init_devices(void)
{
CLI();                          //заборона переривань
port_init();                    //ініціалізація портів
SEI();                          //дозвіл переривань
}
//---------------------------
//---------- Головна функція програми
void main(void)
{
init_devices();          //Ініціалізація мікроконтролера
while (1)                //нескінченний цикл
      {
      PORTA |=(1 << LED1); //встановлення «1» на лінії PA7
      };
}
//---------------------------
```

4.2. Таймери – лічильники AVR мікроконтролерів

Загальні відомості

Мікроконтролери з AVR ядром мають у своєму складі від 1 до 10 8 /16-розрядних таймерів лічильників. Мікроконтролери megaAVR, у залежності від моделі, мають у своєму складі від одного до трьох таймерів/лічильників загального призначення.

Таймер/лічильник T0 8-розрядний, у більшості моделей може використовуватися лише для відліку чи вимірювання часових інтервалів, або як лічильник зовнішніх імпульсних сигналів. При переповненні лічильного регістру таймеру генерується запит на переривання.

Два інших таймера (16-розрядний T1 та 8-розрядний T2) мають додаткові по відношенню до T0 функції. Обидва таймери можуть генерувати запит на переривання не тільки при переповненні лічильного регістру, а й при здійсненні ряду інших подій.

Таймери можуть також працювати в якості широтно-імпульсних модуляторів. Таймер T2, крім того, може працювати в асинхронному (відносно тактового сигналу мікроконтролера) режимі. У складі усіх мікроконтролерів сімейства є також сторожовий таймер. Цей таймер використовується для захисту від зациклювання програми.

Побудова таймерів/лічильників

Кожний таймер/лічильник використовує один або більше лінії портів вводу/виводу загального призначення мікроконтролера. Функції що реалізовані під час роботи з таймерами/лічильниками на цих виводах, являються альтернативними функціями портів.

Таймер/лічильник T0

Структурну схему таймера/лічильника T0 зображено на рис. 4.2.

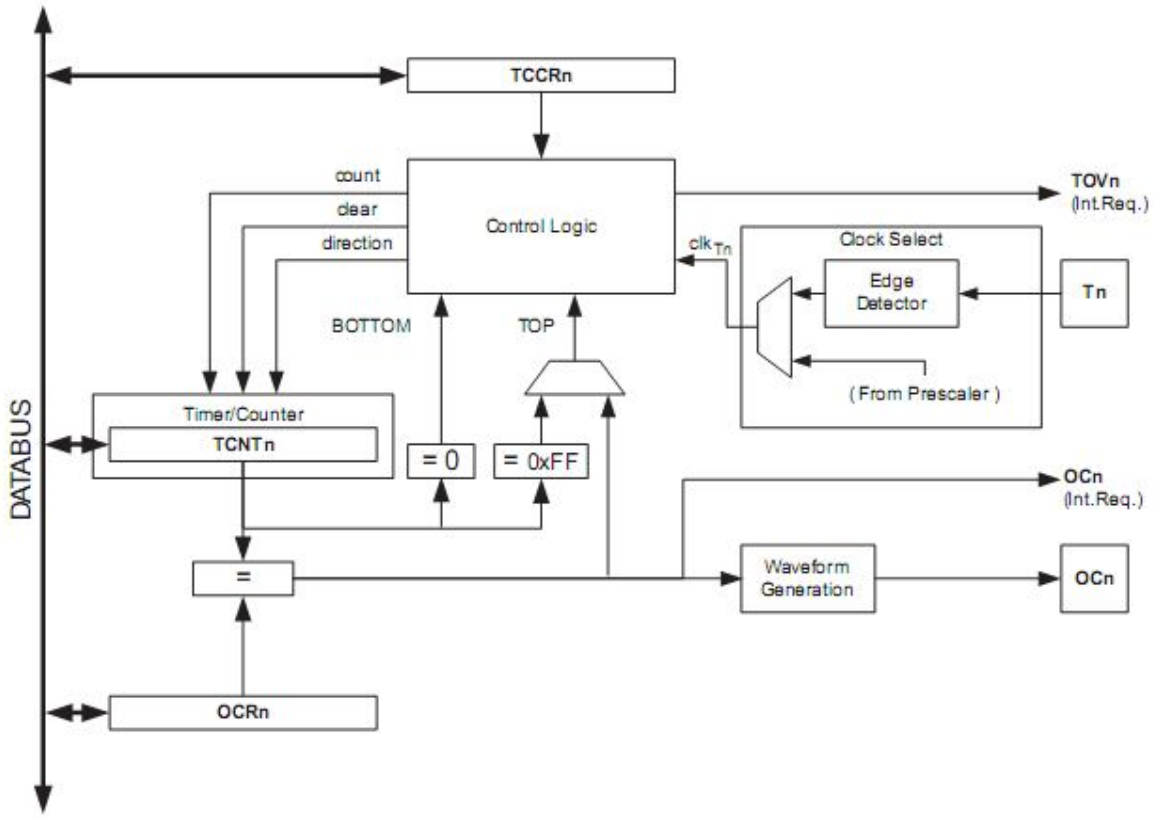


Рис. 4.2 Структурна схема таймера/лічильника T0

До його складу входять: блок керування з регістром керування TCCR0, а також 2 регістри - лічильний регістр TCNT0, та вихідний регістр порівняння OCR0. Блок керування забезпечує селекцію тактових імпульсів, що поступають на лічильний регістр. У табл. 4.2 приведено можливі варіанти конфігурації джерел тактових імпульсів.

Таблиця 4.2

CS02	CS01	CS00	Опис
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8(From prescaler)
0	1	1	$clk_{I/O}$/64(From prescaler)
1	0	0	$clk_{I/O}$/256(From prescaler)
1	0	1	$clk_{I/O}$/1024(From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge
1	1	1	External clock source on T0 pin. Clock on rising edge

Згідно з цією таблицею визначаються три основних режими вибору джерела тактових сигналів:

- вимкнений стан таймера/лічильника;
- робота в якості таймера з попереднім подільником тактової частоти генератора (коефіцієнти ділення 1/1, 1/8, 1/64, 1/256, 1/1024);
- робота в якості лічильника зовнішніх імпульсів, що поступають на лінію порту T0 (реєстрація по наростаючому або спадаючому фронту).

Для керування режимами роботи використовується регістр TCCR0. На рис. 4.3. приведено бітовий опис цього регістру.

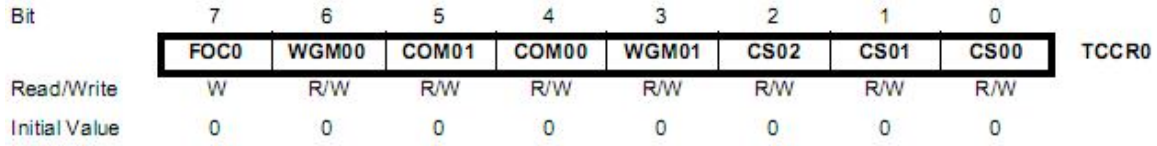


Рис. 4.3.

Біти WGM01 та WGM00 використовуються для вибору базових режимів роботи таймера/лічильника: Normal, CTC, Phase Correct PWM, Fast PWM. У табл. 4.3 вказані відповідні значення біт WGM01 та WGM00 при виборі цих режимів роботи.

Таблиця 4.3

Mode	WGM01 (CTC0)	WGM00 (PWM0)	Timer/Counter Mode of Operation	TOP	Update of OCR0	TOV Flag Set-on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM,Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Immediate	MAX
3	1	1	Fast PWM	0xFF	TOP	MAX

Біти COM01 та COM00 використовуються для встановлення способу зміни вихідного сигналу на виводі OC0 мікроконтролера. Описи відповідних реакцій для режимів роботи таймера/лічильника Normal (CTC), Fast PWM та Phase Correct PWM практично не відрізняються. Для

режиму роботи Fast PWM у табл. 4.4 вказані значення біт COM01 та COM00 та відповідні реакції на виводі OC0.

Таблиця 4.4

COM01	COM00	Description
0	0	Нормальна робота порту, OC0 відключено
0	1	Зарезервовано
1	0	Скидання OC0 при співпаданні, встановлення OC0 при TOP
1	1	Встановлення OC0 при співпаданні, встановлення OC0 при TOP

Таймер/лічильник T0 може формувати два запита на переривання – в результаті переповнення регістру лічильника або при порівнянні значень в лічильному регістрі TCNT0 та вихідному регістрі порівняння OCR0. Прапори переривань (TOV0, OCF0) знаходиться в регістрі прапорів переривань від таймерів TIFR. Дозвіл або заборона відповідних переривань здійснюється установкою/скиданням прапорів TOIE0 або OCIE0 регістра TIMSK.

Режим роботи Normal

У режимі Normal таймер/лічильник працює як інкрементний накопичувач. Після накопичення максимальної 8-бітової величини 0xFF відбувається переповнення і в лічильному регістрі TCNT0 встановлюється значення 0x00. Переповнення супроводжується встановленням прапору переповнення TOV0. Цей прапор автоматично скидається у разі виконання відповідної підпрограми переривань. Частота переповнень у цьому режимі визначається за наступною формулою

$$F_{OC} = \frac{f_{CLK} \cdot N}{0xFF + 1 - N_{TCNT0}},$$

де: f_{CLK} - тактова частота генератора мікроконтролера; N- коефіцієнт ділення попереднього подільника (1/1, 1/8, 1/64, 1/256, 1/1024); N_{TCNT0} – константа завантаження лічильного регістру.

Недоліком цього режиму є необхідність постійного поновлення після переривань дистрибутивних значень регістру лічильника TCNT0 для забезпечення сталої частоти переповнень.

Режим роботи CTC

У режимі CTC таймер/лічильник працює як інкрементний накопичувач. Накопичення ведеться від нульового значення лічильного регістру TCNT0 до значення, записаного у вихідному регістрі порівняння OCR0. Після приходу наступного тактового імпульсу в лічильному регістрі TCNT0 встановлюється значення 0x00 та встановлюється прапор переповнення OCF0. Цей прапор автоматично скидається у разі виконання відповідної підпрограми переривань. Частота переповнень у цьому режимі визначається за наступною формулою

$$F_{OC} = \frac{f_{CLK} \cdot N}{1 + N_{OCR0}},$$

де N_{OCR0} – константа завантаження регістру порівняння.

На рис. 4.4 зображено часові діаграми роботи таймера/лічильника у CTC режимі.

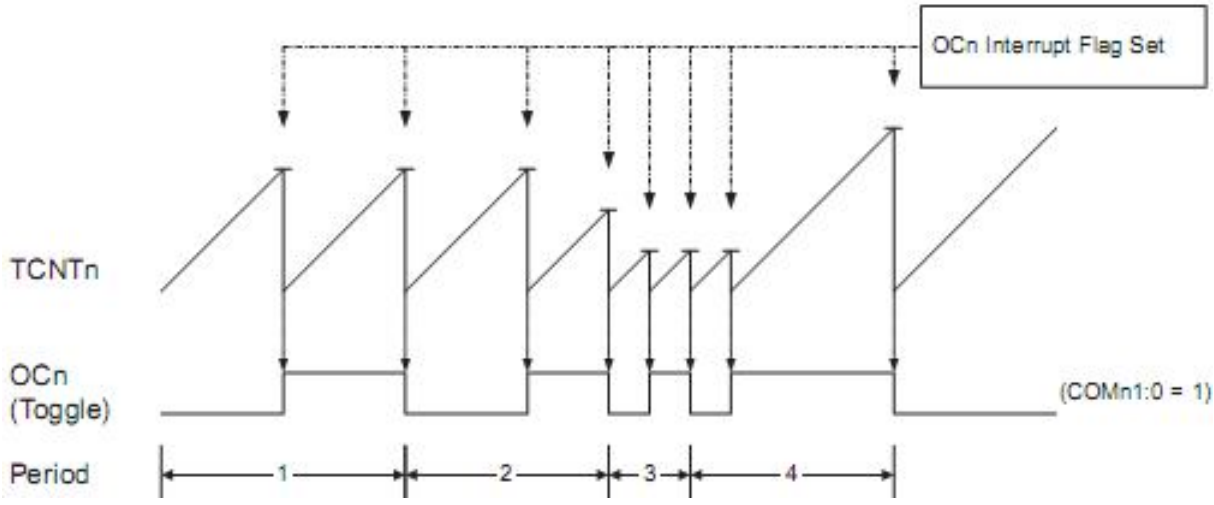


Рис. 4.4 Часові діаграми роботи таймера T0 у CTC режимі

Перевагою цього режиму роботи, відносно режиму Normal, є те, що для забезпечення сталої частоти переповнень нема необхідності в поновленні дистрибутивних значень регістру лічильника TCNT0, що спрощує програмний продукт.

Режим роботи Fast PWM

У режимі Fast PWM таймер/лічильник працює як інкрементний накопичувач значень регістру TCNT0. Накопичення ведеться від нульового значення лічильного регістру до величини 0xFF. Після переповнення в лічильному регістрі TCNT0 встановлюється значення 0x00. Переповнення супроводжується встановленням прапору переповнення TOV0. Частота переповнень у цьому режимі визначається за наступною формулою

$$F_{OC} = \frac{f_{CLK} \cdot N}{256} \ .$$

Ця частота є частотою PWM сигналу.

Тривалість імпульсів PWM сигналу визначає значення, що записане у вихідному регістрі порівняння OCR0. При порівнянні значень у регістрах TCNT0 та OCR0 на виході мікроконтролера OC0 починає формуватися вихідний імпульс, полярність якого та присутність визначається бітами керування COM01 та COM00 (табл. 4.4). Одночасно формується прапор переривань OCF0. Закінчується формування вихідного імпульсу при переповненні лічильного регістру TCNT0.

На рис. 4.5 зображено часові діаграми роботи таймера/лічильника у режимі Fast PWM.

Режим Fast PWM має наступні особливості:

- зміна частоти PWM імпульсів можлива за рахунок зміни коефіцієнту ділення попереднього подільника;
- при зміні тривалості імпульсів PWM сигналу нове значення завантажується у вихідний регістр порівняння OCR0 не в момент, що

визначається програмою, а лише після завершення повного циклу генерації чергового періоду - при переповненні лічильного регістру TCNT0.

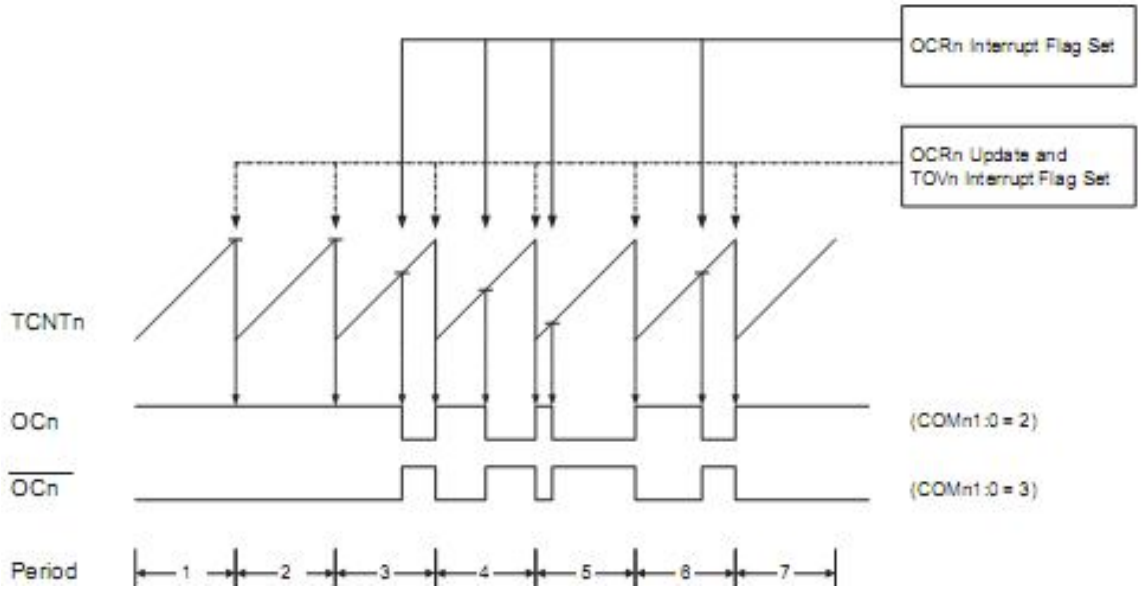


Рис. 4.5 Часові діаграми роботи таймера T0 у режимі Fast PWM.

Режим роботи Phase Correct PWM

У режимі Phase Correct PWM таймер/лічильник на періоді повторення працює спочатку як інкрементний накопичувач значень регістру TCNT0, а після досягнення максимального значення 0xFF переходить у декрементний режим. Після встановлення в лічильному регістрі TCNT0 значення 0x00 завершується цикл формування повного періоду PWM сигналу, що супроводжується встановленням прапору переповнення TOV0. Частота PWM сигналу в цьому режимі визначається за наступною формулою

$$F_{OC} = \frac{f_{CLK} \cdot N}{510} \ .$$

Як і в режимі Fast PWM тривалість імпульсів визначає значення, що записане у регістрі OCR0. Відмінність полягає у тому, що імпульс починає формуватися при порівнянні значень регістрів TCNT0 та OCR0 у інкрементному режимі роботи, а завершується при порівнянні у декрементному режимі. Полярність та присутність цього імпульсу, визначається бітами керування COM01 та COM00 (табл. 4.4). Прапор

переривань порівняння OCF0 формується двічі за період, у моменти початку та закінчення імпульсу.

На рис. 4.6 зображено часові діаграми роботи таймера/лічильника у режимі Phase Correct PWM.

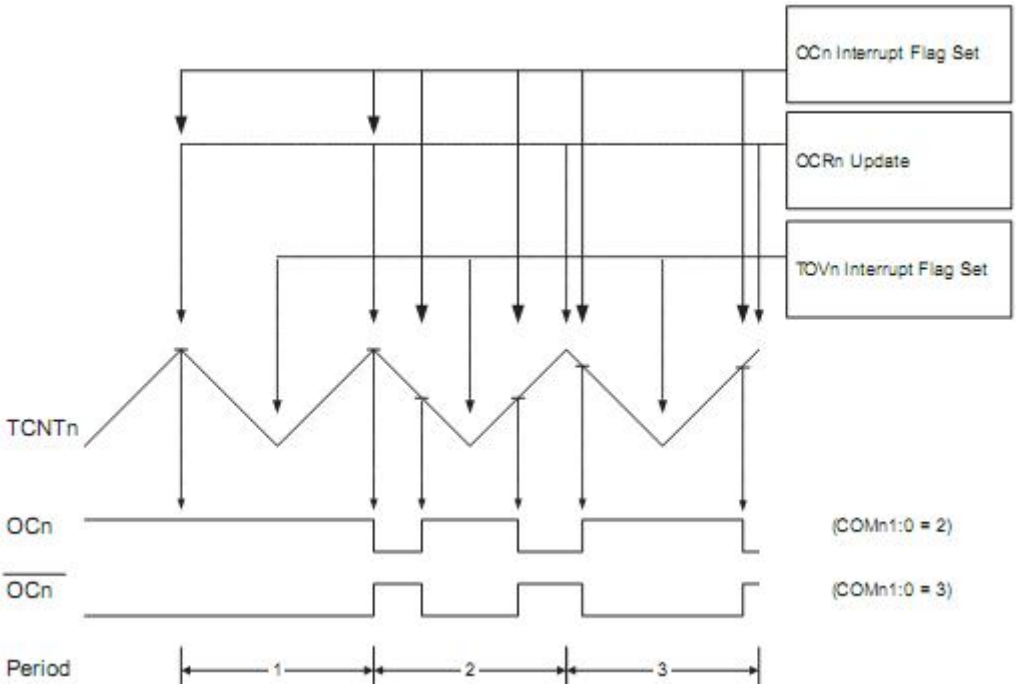


Рис. 4.6 Часові діаграми роботи таймера T0 у режимі Phase Correct PWM

Режим Phase Correct PWM має наступні особливості:

- як і в режимі Fast PWM зміна частоти PWM послідовності можлива за рахунок зміни коефіцієнту ділення попереднього подільника;
- при зміні тривалості імпульсів PWM сигналу нове значення завантажується у вихідний регістр порівняння OCR0, в момент переходу
- таймера/лічильника з інкрементного в декрементний режим.

Таймер/лічильник T2

Структурну схему таймера/лічильника T2 зображено на рис. 4.7.

Цей лічильник відрізняється від T0 наявністю елементів, що забезпечують асинхронний режим роботи. Додатково введено генератор до якого через виводи мікроконтролера TOSC1 та TOSC2 може підключатися зовнішній низькочастотний резонатор (32768 кГц). Це дозволяє створити

на базі мікроконтролера незалежний асинхронний генератор реального часу.

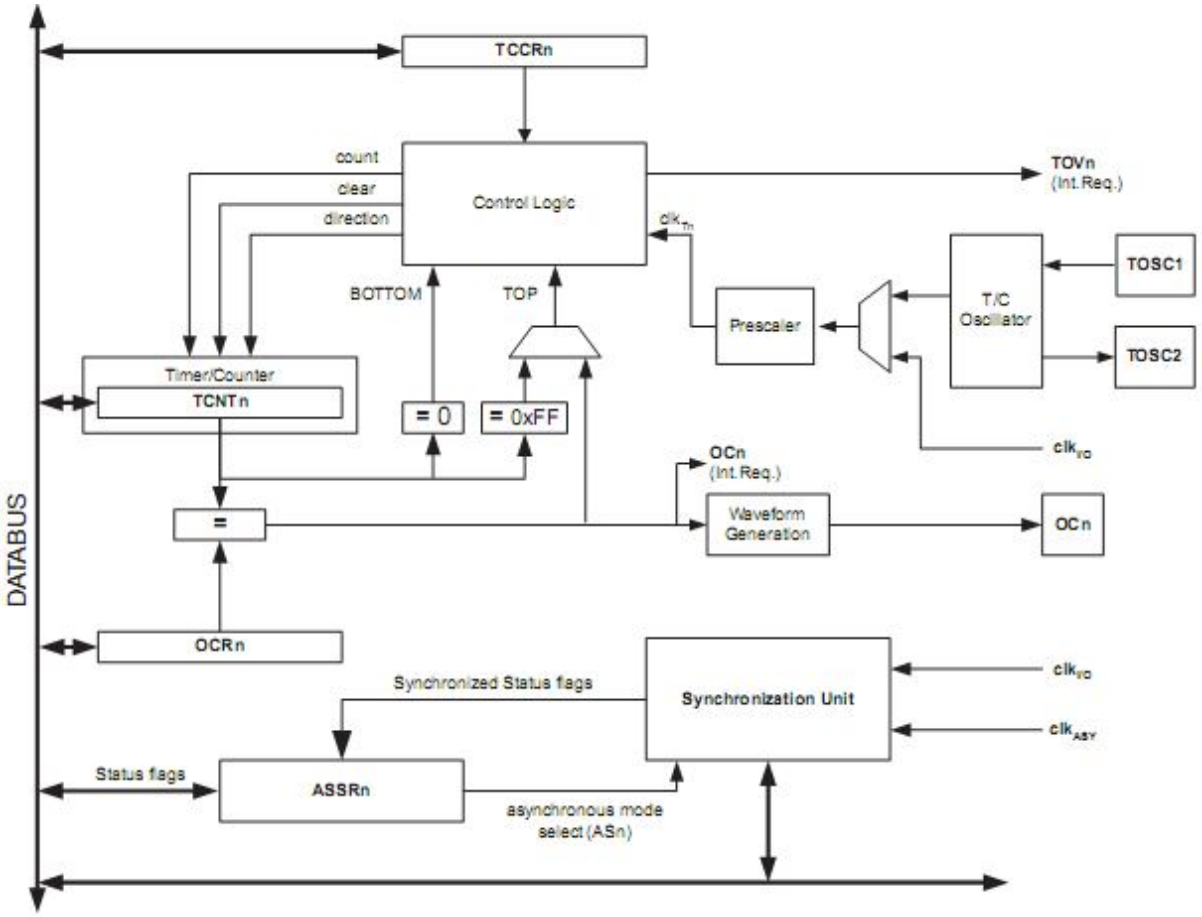


Рис. 4.7 Структурна схема таймера/лічильника T2

Для забезпечення керування асинхронним режимом введено регістр ASSR2 (рис. 4.8).

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	AS2	TCN2UB	OCR2UB	TCR2UB	ASSR
Read/Write	R	R	R	R	R/W	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Рис. 4.8

Біт AS2 використовується для вибору асинхронного режиму роботи таймера/лічильника T2. Коли до цього біту записано «0» таймер/лічильник працює з тактовою послідовністю, що формується генератором

мікроконтролера. У разі запису «1» до таймера/лічильника підключається вихід асинхронного генератора і тактова частота визначається резонатором, який підключено до виводів TOSC1 та TOSC2.

Біти TCN2UB, OCR2UB та TCR2UB використовуються для коректної модифікації регістрів TCNT2, OCR2 та TCCR2 при роботі таймера/лічильника в асинхронному режимі. Запис в цих бітах «1» відповідає тому, що передача попередньо модифікованих даних ще не відбулась і нова модифікація може викликати помилки у роботі. Після такої передачі у ці біти записуються значення «0».

Решта функцій таймера/лічильника Т2 повністю відповідає функціям таймера/лічильника Т0.

Таймер/лічильник Т1

Структурну схему таймера/лічильника Т1 зображено на рис. 4.9. До складу таймера/лічильника Т1 входять чотири 16-розрядні регістри (лічильний регістр TCNT1H:TCNT1L, регістр захвату ICR1H:ICR1L та регістри порівняння OCR1AH:OCR1AL та OCR1BH:OCR1BL), 16-розрядний компаратор, два 8-розрядні регістри керування TCCR1A та TCCR1B, а також блок керування таймером.

Усі прапори стану таймера/лічильника (переповнення, збігу та захвату)
знаходяться в регістрі прапорів переривань від таймерів TIFR, а дозвіл / заборона переривань від таймера здійснюється установкою / скиданням відповідних прапорів регістра TIMSK.

Режим таймера

У цьому режимі роботи за кожним імпульсом, що надходить на тактовий вхід таймера/лічильника, відбувається інкремент вмісту лічильного регістру CNT1. При переході таймера/лічильника зі стану

«$FFFF» у стан «$0000» встановлюється прапор TOV1 регістра TIFR та генерується запит на переривання.

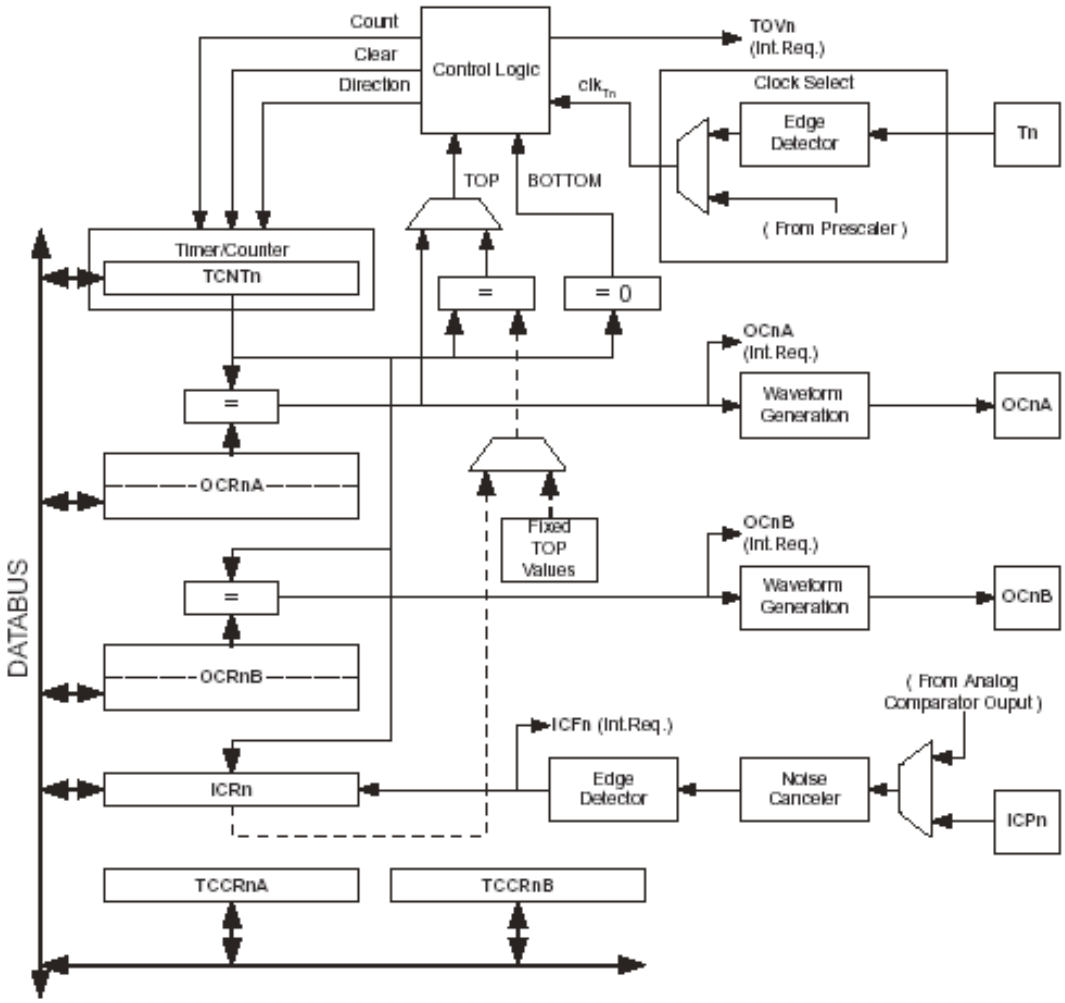


Рис. 4.9 Структурна схема таймера T1

Дозвіл переривання здійснюється установкою в «1» розряду TOIE1 регістра TIMSK (рис. 4.11) (прапор I загального дозволу переривань регістра SREG також повинен бути встановлений в «1»).

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рис. 4.10

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рис. 4.11.

Функція захвату (Capture)

Структурну схему таймера/лічильника при реалізації функції захвату зображено на рис. 4.12.

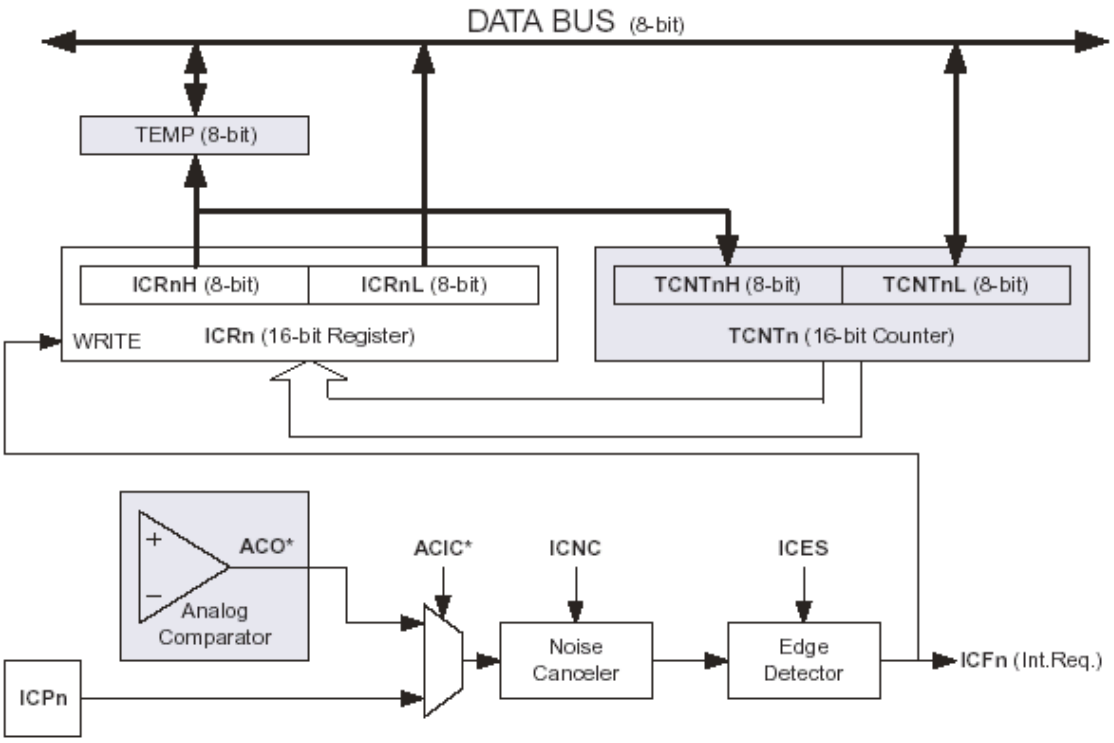


Рис. 4.12. Структурна схема таймера T1 при реалізації функції захвату

Під даною функцією розуміють збереження у визначений час стану таймера/лічильника в регістрі захвату ICR1 (рис. 4.13). Ця дія може виконуватися або за активним фронтом сигналу на виводі ICP мікроконтролера, або за сигналом від компаратору. При цьому встановлюється прапор ICF1 регістра TIFR та генерується запит на переривання. Дозвіл переривань здійснюється встановленням в «1» розряду TICIE1 регістра TIMSK.

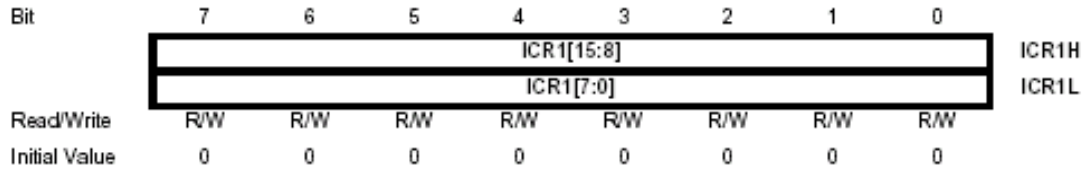


Рис. 4.13

Для керування схемою захвату використовується два розряди регістра TCCR1B: ICNC1 та ICES1. Розряд ICNC1 керує схемою знешкодження завад. Якщо цей розряд встановлений в «0», схема знешкодження завад вимкнена та захват відбувається за першим активним фронтом на виводі ICP мікроконтролера. Якщо цей розряд встановлений в «1», то при надходженні активного фронту на вивід ICP виконуються 4 вибірки з частотою, рівною тактовій частоті мікроконтролера. Захват виконується якщо всі вибірки мають рівень, що відповідає активному фронту сигналу («1» для зростаючого та «0» для спадаючого).

Активний фронт сигналу визначається станом розряду ICES1. Якщо цей розряд скинутий в «0», то активним є спадаючий фронт. Якщо ж цей розряд встановлений в «1», то активним є зростаючий фронт. Вихід ICP повинен мати конфігурацію вхідного виводу, тобто розряд керування портом DDRx, що відповідає даному виводу, має бути скинутим в «0».

Фізично регістр захвату ICR1 розміщений в двох регістрах ICR1H:ICR1L які доступні лише для читання.

Оскільки регістр захвату є 16-розрядним, при його читанні використовується спеціальний тимчасовий регістр TEMP. При читанні регістра ICR1L вміст цього регістра пересилається, а вміст регістра ICR1H зберігається в регістрі TEMP. При читанні регістра ICR1H повертається значення, що збережене в регістрі TEMP. Отже, при читанні регістра ICR1 першим повинен бути прочитаний регістр ICR1L. Переривання на час звертання до регістра ICR1 мають бути забороненими.

Функція порівняння (Compare)

Ця функція передбачає безперервне порівняння вмісту лічильного регістру таймера/лічильника з числом, що знаходиться в регістрі порівняння. При рівності вмісту в цих регістрах встановлюється прапор відповідного переривання, а також можуть виконуватися інші дії.

Якщо стан таймера/лічильника стане рівним числу, що знаходиться в регістрі порівняння, то в наступному машинному циклі встановлюється відповідний прапор переривань в регістрі TIFR та генерується запит на переривання. Дозвіл переривань здійснюється встановленням в «1» відповідних прапорців регістра TIMSK.

Разом з установкою прапора в регістрі TIFR, при рівності вмісту лічильного регістра та регістра порівняння, можуть виконуватися й інші дії:

- скидання таймера/лічильника;
- зміна стану визначеного виводу мікроконтролера.

Поведінка мікроконтролера визначається відповідними бітами регістрів керування TCCR1A та TCCR1B.

Кожен регістр порівняння фізично розташований у двох регістрах вводу/виводу, які доступні для запису/читання: OCR1A – OCR1AH:OCR1AL та OCR1B – OCR1BH:OCR1BL.

Регістри керування таймера/лічильника Т1 - TCNT1, OCR1A, OCR1B та ICR1

Лічильний регістр TCNT1 реалізує операцію додавання (у режимі ШІМ – додавання/віднімання) та доступний в будь-який момент часу як для читання, так і для запису. Під час запису в регістр TCNT1, при роботі таймера, рахунок буде продовжений за наступним за операцією запису імпульсом тактового сигналу таймера/лічильника. Після подачі напруги живлення в регістрі TCNT1 буде знаходитися нульове значення.

Фізично регістр TCNT1 розміщений в двох регістрах TCNTH1:TCNTL1. Щоб при звертанні до цих регістрів запис або читання обох байтів відбувався одночасно, використовується спеціальний регістр TEMP (цей регістр використовується лише процесором та програмно недосяжний). Цей же регістр використовується і при звертанні до решти 16-розрядних регістрів таймера/лічильника T1: OCR1A, OCR1B та ICR1. Переривання на час звертання до цих регістрів має бути забороненим.

Запис у регістр TCNT1

При запису старшого байту значення в регістр TCNT1H він розміщується в регістрі TEMP. Далі, при запису молодшого байту в регістр TCNT1L він об'єднується з вмістом регістра TEMP та обидва байти записуються в регістр TCNT1 одночасно.

Читання регістра TCNT1

При читанні TCNT1L (молодший байт) вміст регістра TCNT1L пересилається в регістр TEMP. А при наступному читанні регістра TCNT1H повертається значення, що збереглося в регістрі TEMP.

Регістри керування TCCR1A та TCCR1B

Керування таймером/лічильником T1 здійснюється за допомогою двох регістрів керування TCCR1A та TCCR1B. На рис. 4.14 наведено бітову структуру цих регістрів.

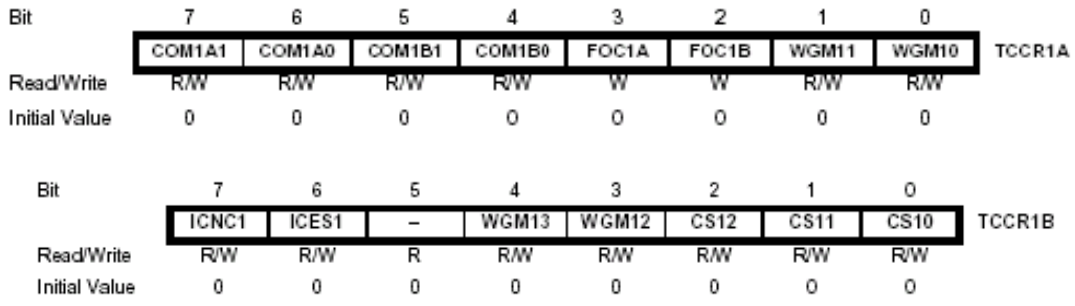


Рис. 4.14

У табл. 4.5 наведено можливі режими роботи таймера/лічильника T1 та відповідні значення керуючих біт.

Таблиця 4.5.

Режим	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Режим Timer/Counter	Top	Update of OCR1x	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM,8-bit	0x00FF	TOP	TOP
6	0	1	1	0	Fast PWM,9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	-	-	-
14	1	1	1	0	Fast PWM	ICR1	TOP	TOP
15	1	1	1	1	Fast PWM	OCR1A	TOP	TOP

Вибір джерела тактового сигналу

По відношенню до тактового сигналу таймер/лічильник може працювати в двох режимах:

- Режим таймера. В цьому режимі на вхід таймера/лічильника надходять імпульси тактового сигналу мікроконтролера (безпосередньо або через подільник);
- Режим рахування кількості сигналів. В цьому режимі інкремент вмісту лічильного регістру відбувається за активним фронтом сигналу на вході T1 мікроконтролера.

Вибір джерела тактового сигналу, а також запуск або зупинка таймера/лічильника здійснюється за допомогою розрядів CS12…CS10 регістра керування таймером TCCR1B. Повний опис конфігурацій приведено в табл. 4.6.

Таблиця 4.6

Регістр TCCR1B			Джерело тактового сигналу
CS12	CS11	CS10	
0	0	0	Таймер/лічильник зупинено
0	0	1	СК – тактовий сигнал мікроконтролера
0	1	0	СК/8
0	1	1	СК/64
1	0	0	СК/256
1	0	1	СК/1024
1	1	0	Лінія порту T1, інкремент по спадаючому фронту сигналу
1	1	1	Лінія порту T1, інкремент по зростаючому фронту сигналу

При використанні зовнішнього тактового сигналу необхідно пам’ятати, що він синхронізується з частотою тактового генератора мікроконтролера. Тому для забезпечення коректної роботи таймера від зовнішнього сигналу проміжок часу між сусідніми імпульсами повинен бути більше періоду тактового сигналу мікроконтролера.

Приклади ініціалізації таймерів/лічильників

У приведених прикладах продемонстровані функції ініціалізації таймера/лічильника T1 на режими роботи “Normal” (приклад 4.1), та “Fast PWM” (приклад 4.2). При їх розробці використовувався засіб Application Builder інтегрованого середовища розробки програмного забезпечення Image Craft v7.

***Приклад* 4.1**

У першому прикладі забезпечується ініціалізація таймера/лічильника T1 на режим “Normal”.

```
//-----------------------------
//ICC-AVR application builder : 14.05.2016 1:47:18
//Target : M16
//Crystal: 10.000Mhz
```

```
//---------- Функція ініціалізації TC1
void init_timer(void)
{
TIMSK  = 0x00;           //заборона переривань TC1
TCNT1  = 0;              //завантаження регістрів лічильника
TCCR1B = 0;              //виключення TC1
}
//---------------------------
```

Приклад 4.2

У другому прикладі забезпечується формування на лінії порту PWM (PD4 – OC1B) періодичної послідовності позитивних імпульсів з алгоритмом утворення “Fast PWM”. Використана секція “B” таймера TC1, попередній подільник 1/256. Реалізовано режим “Fast PWM”, 8 розрядів, порт встановлюється в «1» на інтервалі формування імпульсу.

Параметри імпульсів:

- Частота імпульсів 152,6 Гц. Частота визначається з формули

$$F_{OC} = f_{CLK} / (N * TOP),$$

де f_{CLK} – частота генератора мікроконтролера (10МГц); N – коефіцієнт ділення попереднього подільника; TOP – модуль рахування.

Для 8-бітової “Fast PWM” величина TOP складає 256; тривалість – 3,25мс; розрядність (точність) PWM – 8 біт.

```
//---------------------------
//ICC-AVR application builder : 14.05.2016 2:26:40
// Target : M16
// Crystal: 10.000Mhz

//---------- Функція ініціалізації портів
void port_init(void)
{
DDRD  = 0b00010000; // лінія порту PWM (PD4 – OC1B) - передавач
PORTD = 0x00;
}
//---------- Функція ініціалізації TC1
//TIMER1 initialize - prescale:256
//WGM: 5) PWM 8bit fast, TOP=0x00FF
void timer1_init(void)
{
TCCR1B = 0x00;           //зупинка таймера
TCNT1  = 0;              //завантаження періоду
OCR1BL = (0x00ff-127);   //завантаження тривалості імпульсів
TCCR1A = (1<<COM1B1)|(1<<COM1B0)|(1<<WGM10); //режим роботи
TCCR1B = (1<<CS12)|(1<<WGM12);               //старт таймера
}
//---------------------------
```

4.3. Аналогово-цифровий перетворювач мікроконтролера ATmega16

Мікроконтролери сімейства megaAVR мають в своєму складі 10-розрядний АЦП послідовного наближення з одним ядром перетворення. Число каналів залежить від моделі та знаходиться у межах від 8 до 16.

Мікроконтролер ATmega16 має наступні параметри вбудованого АЦП:

- одне ядро перетворення. Розрядність 10 біт;
- максимальна продуктивність АЦП досягає 15кSPS;
- час перетворення знаходиться у межах від 65 до 260 мкс. Можливе зменшення часу перетворення за рахунок втрати точності;
- 8 мультиплексованих однополярних вхідних канали. Можливість використання 7 диференційних вхідних канали. Існує можливість зміни коефіцієнту підсилення аналогової напруги (10х, 200х) для двох з них;
- форматування результату вимірювань. Вирівнювання результату до однобайтового або двобайтового формату;
- наявність вбудованого джерела еталонної напруги 2,56В. Можливість підключення зовнішнього еталонного джерела, або джерела живлення АЦП мікроконтролера;
- режими одноразового перетворення та циклічних перетворень;
- режим автоматичного запуску перетворення (автотрігерінг) по ряду подій у вбудованих контролерах – компараторі, таймерах-лічильниках…);
- можливість використання переривань по події завершення циклу перетворення;
- наявність схеми вибірки-зберігання аналогової інформації на вході АЦП;
- інтегральна не лінійність вимірювань не перевищує 0,5 LSB;
- абсолютна точність вимірювань ±2 LSB.

В якості входів модуля АЦП в моделі ATmega16 використовуються виводи порту А.

Структурна схема аналогово-цифрового перетворювача

Структурну схему модуля АЦП мікроконтролера ATmega16 зображено на рис. 4.15.

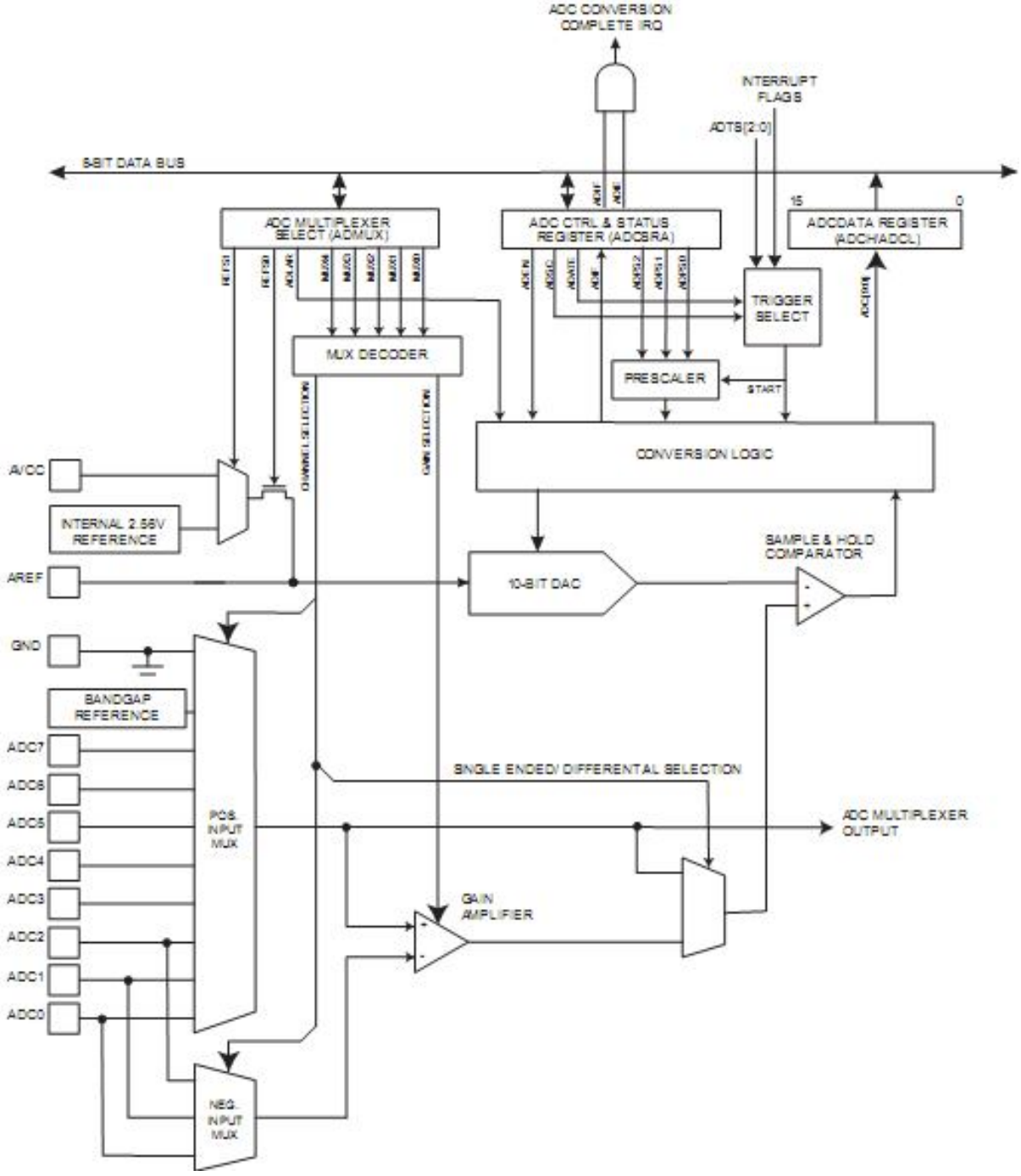


Рис. 4.15. Структурна схема модуля АЦП мікроконтролера ATmega16

До складу схеми входять наступні блоки:

- АЦП послідовного наближення;
- блок комутації джерела еталонної напруги;
- блок комутаціїї вхідних сигналів;
- схему керування;
- програмно доступні регістри ADCSRA (регістр керування, статусу АЦП), ADMUX (регістр керування мультиплексором) та ADCH:ADCL (регістр результату).

Для живлення модуля АЦП у мікроконтролері передбачені виводи: зовнішнього джерела опорної напруги. Напруга на цьому виводі повинна AVCC (напруга живлення) та AGND (аналогова «земля»). Напруга на виводі AVCC не повинна відрізнятися від напруги живлення мікроконтролера більше ніж на ± 0,3В, а аналогова «земля» має бути з'єднана з цифровою.

Мікроконтролер має вивід AREF для підключення до АЦП знаходитися в діапазоні 0…*V*cc.

АЦП може працювати у двох режимах:

- режим одинарного перетворення. У цьому режимі запуск кожного перетворення ініціюється користувачем;
- режим безперервного перетворення. У цьому режимі запуск перетворень виконується безперервно через певні інтервали часу, або від певного джерела, яке визначається налаштування АЦП.

Регістри керування аналогово-цифрового перетворювача

Керування модуля АЦП та контроль його стану здійснюється за допомогою регістра ADCSRA та ADMUX. Результат перетворення розміщується в регістрах ADCH:ADCL.

Режими роботи АЦП встановлюються в основному за допомогою регістру ADCSRA (рис. 4.16).

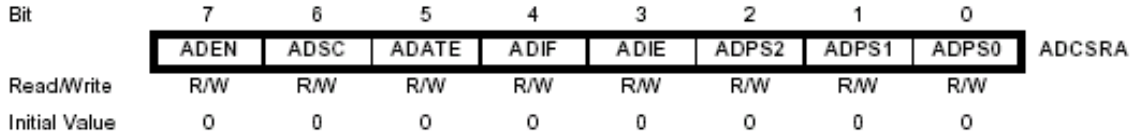


Рис. 4.16

Біти регістру ADCSRA мають наступне призначення («1» - активна):

- ADEN - дозвіл АЦП;
- ADSC - запуск перетворення;
- ADATE - визначення режиму перетворень (одноразові/циклічні);
- ADIF - прапор переривань АЦП;
- ADIE - дозвіл переривань;
- ADPS2, ADPS1, ADPS0 – визначення частоти перетворень.

Перед початком використання АЦП необхідно дозволити його роботу. Для цього необхідно записати «1» в розряд ADEN регістра ADCSRA, а для заборони відповідно «0». Якщо АЦП буде заблокований на час циклу перетворення, то перетворення не буде закінченим (в регістр даних АЦП залишиться результат попереднього перетворення).

Режим роботи АЦП визначається станом розряду ADATE. Якщо він встановлений в «1», АЦП працює в режимі безперервного перетворення. У цьому режимі запуск кожного наступного перетворення здійснюється автоматично після закінчення поточного. Якщо ж біт ADATE скинутий в «0», АЦП працює в режимі одинарного перетворення та запуск кожного перетворення здійснюється за командою користувача.

Запуск перетворення здійснюється установкою в «1» розряду ADSC регістра ADCSRA, а сам цикл перетворення починається за першим зростаючим фронтом тактового сигналу після установки цього розряду. Тривалість циклу складає 13 тактів; вибірка та запам’ятовування вхідного сигналу здійснюється на протязі перших 1,5 тактів. Через 13 тактів перетворення закінчується, розряд ADSC апаратно скидається в «0» (в

режимі одинарного перетворення), та результат перетворення зберігається в регістрі даних АЦП. Одночасно встановлюється прапор переривання ADIF регістра ADCSRA та генерується запит на переривання. Цей прапор скидається апаратно при запуску підпрограми обробки переривань від АЦП або програмно – записом в нього логічної «1». Дозвіл переривання здійснюється установкою в «1» розряду ADIE регістра ADCSRA (прапор I регістра SREG також повинен бути встановлений в «1»).

Якщо АЦП працює в режимі безперервного перетворення, новий цикл почнеться одразу ж після запису результату. В режимі одинарного перетворення нове перетворення можна запустити одразу 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. При написанні програми потрібно враховувати одну особливість: для першого після включення АЦП перетворення необхідно на 12 тактів більше, ніж для усіх наступних. Це пов'язане з тим, що при запуску першого перетворення спочатку виконується одне «холосте» перетворення, що ініціалізує АЦП. Розряд ADSC в цьому випадку скидається лише після закінчення робочого перетворення.

В режимі безперервного перетворення запуск відбувається або по закінченню процесу попереднього перетворення (по встановленню прапора ADIF) або через певний час від джерел запуску. Джерела запуску у цьому випадку визначаються бітами ADTS2:ADTS0 регістра спеціальних функцій SFIOR (рис. 4.17).

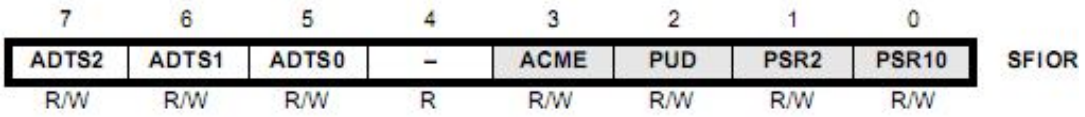


Рис. 4.17

Існують такі можливості для запуску перетворення АЦП в режимі безперервного перетворення:

- режим вільного запуску;
- запуск по встановленню прапора переривань аналогового компаратора;
- запуск по встановленню прапора переривань по входу INT0;
- запуск по встановленню прапора переривань по події переповнення (режим Normal) таймера T0;
- запуск по встановленню прапора переривань по події співпадіння (CTC режим) таймера T0;
- запуск по встановленню прапора переривань по події переповнення (режим Normal) таймера T1;
- запуск по встановленню прапора переривань по події співпадіння (CTC режим) таймера T1;
- запуск по встановленню прапора переривань по захвату таймера T1.

У табл. 4.7 приведено опис налаштування біт ADTS2:ADTS0.

Таблиця 4.7

ADTS2	ADTS1	ADTS0	ДЖЕРЕЛО ЗАПУСКУ
0	0	0	Режим вільного запуску
0	0	1	Аналоговий компаратор
0	1	0	Зовнішнє переривання 0
0	1	1	По співпадінню, таймер/лічильник T0
1	0	0	По переповненню, таймер/лічильник T0
1	0	1	По співпадінню, таймер/лічильник T1
1	1	0	По переповненню, таймер/лічильник T1
1	1	1	По захвату, таймер/лічильник T1

Коли завершується чергове перетворення у режимі вільного запуску, або встановлюється відповідний прапор переривань, скидається

попередній подільник схеми керування АЦП і запускається чергове перетворення.

У даному режимі АЦП виконує послідовні перетворення, незалежно від того - скинутий прапор переривання ADIF чи ні.

Тактовий сигнал модуля АЦП формується подільником, на вхід якого, в свою чергу, надходить тактовий сигнал мікроконтролера. Коефіцієнт ділення цього подільника та, відповідно, тривалість перетворення визначається станом розрядів ADPS2…ADPS0 регістра ADCSRA. Значення коефіцієнту ділення у залежності від стану бітів ADPS2…ADPS0 приведено у табл. 4.8.

Таблиця 4.8

ADPS2	ADPS1	ADPS0	Значення коефіцієнту ділення
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Найбільша точність перетворення досягається, якщо тактова частота модуля АЦП знаходиться в діапазоні: 50…200 кГц. Відповідно коефіцієнт ділення рекомендується вибирати таким чином, щоб тактова частота модуля АЦП знаходилася у вказаному діапазоні.

Результат перетворення зберігається в регістрі даних АЦП. Оскільки АЦП – 10-розрядний, цей регістр фізично розташований в двох регістрах вводу/виводу ADCH:ADCL, доступних лише для читання. Ці регістри при підключенні живлення містять значення «$0000». Звертання до цих регістрів (для отримання результату перетворення) повинно виконуватися в певній послідовності: спочатку необхідно прочитати регістр ADCL, а потім ADCH. Ця вимога пов’язана з тим, що після звернення до регістру

ADCL процесор блокує доступ до регістрів даних зі сторони АЦП до тих пір, поки не буде прочитаний регістр ADCH. Завдяки цьому можна бути впевненим, що при читанні регістрів в них будуть знаходитися складові одного й того ж результату. Відповідно, якщо чергове перетворення закінчиться до звернення до регістру ADCH, результат перетворення буде втраченим.

Керування вхідним мультиплексором модуля АЦП здійснюється за допомогою регістра ADMUX.

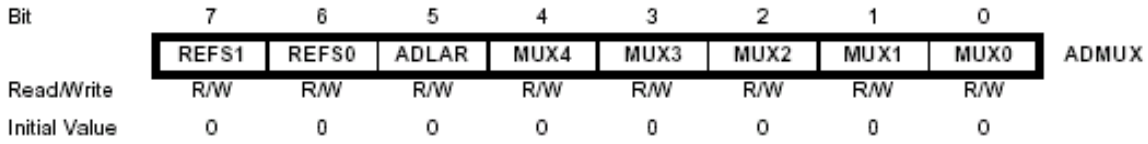


Рис. 4.18

Розряди MUX4…MUX0 цього регістра визначають номер активного каналу (номер аналогового входу, що підключений до входу АЦП). Стан цих розрядів можна змінити в будь-який час, однак, якщо це буде зроблено під час циклу перетворення, зміна каналу відбудеться лише після завершення перетворення. Завдяки цьому в режимі безперервного перетворення можна легко реалізувати сканування каналів. Під цим терміном в даному випадку розуміють послідовне перетворення сигналів декількох каналів.

Розряди REFS1-REFS0 відповідають за вибір джерела опорної напруги. У табл. 4.9 приведено можливі варіанти конфігурування.

Таблиця 4.9

REFS2	REFS0	Вибір джерела еталонної напруги
0	0	AREF зовнішне
0	1	AVCC із зовнішнім конденсатором на виводі AREF
1	0	Зарезервовано
1	1	Внутрішня еталонна напруга з зовнішнім конденсатором на виводі AREF

Розряд ADLAR регістру керування ADMUX відповідає за тип вирівнювання результату перетворення в регістрі даних АЦП.

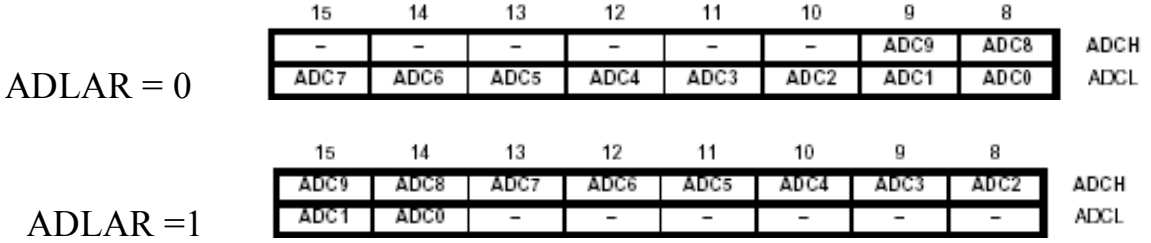


Рис. 4.19

Вирівнювання результату «вліво» (ADLAR = 0) використовується у випадку, коли необхідно отримати 10-бітний результат. Якщо достатньо 8-бітного результату, то доцільно використати режим «правої» упаковки (ADLAR=1).

Аналогово-цифровий перетворювач мікроконтролера Mega16 дозволяє працювати як з диференційними, так і з однополярними входами. При роботі з диференційними входами є можливість підсилення сигналів у 10 або 200 раз. У табл. 4.10 наведені константи налаштування бітів регістру ADMUX для реалізації різних режимів роботи зі входами аналогово-цифрового перетворювача.

Розряди MUX4…MUX0 цього регістра визначають номер активного каналу (номер аналогового входу, що підключений до входу АЦП). Стан цих розрядів можна змінити в будь-який час, однак, якщо це буде зроблено під час циклу перетворення, зміна каналу відбудеться лише після завершення перетворення. Завдяки цьому в режимі безперервного перетворення можна легко реалізувати сканування каналів. Під цим терміном в даному випадку розуміють послідовне перетворення сигналів декількох каналів.

Слід зауважити, що при швидкій зміні каналів на входи яких подаються різні напруги, внаслідок інерційності пристрою вибірки/зберігання, може виникати суттєва похибка в перетворенні. У

цьому випадку в програму сканування каналів необхідно вводити додаткову затримку після комутації нового каналу.

Таблиця 4.10

MUX 4..0	Недиференціальний вхід	Неінвертуючий диференціальний вхід	Інвертуючий диференціальний вхід	Коэффициент усиления, GAIN
00000	ADC0	Диференціальних входів і підсилення немає		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000	Недиференціальних входів немає	ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010[1]		ADC0	ADC0	200x
01011[1]		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x
01110[1]		ADC2	ADC2	200x
01111[1]		ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x
10010		ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x
10111		ADC7	ADC1	1x
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x
11101		ADC5	ADC2	1x
11110	1.22 В (VBG)	Диференціальних входів і підсилення немає		
11111	0 В (GND)			

Приклад використання АЦП

У прикладі 4.3 приведено програму вимірювання напруги, яка подається на вхід ADC0.

У прикладі проводилася ініціалізація АЦП на наступний режим роботи:

- АЦП включено (ADEN = 1);
- режим одиночних перетворень (ADATE = 0);
- дозволені переривання від АЦП (ADIE = 1);
- частота тактового сигналу АЦП (частота генератора 10МГц) – 156,2 кГц (1/64 – ADPS2 = ADPS1 = 1);
- підключено внутрішнє джерело опорної напруги (REFS1=REFS0=1);
- визначено нормальний порядок упаковки результату перетворення (ADLAR = 0, праве вирівнювання);
- підключено вхідний канал з несиметричним входом ADC0.

Для забезпечення цього режиму роботи необхідно провести ініціалізацію порту А та регістрів ADCSRA, ADMUX.

***Приклад* 4.3**

```
//----------------------------
//ICC-AVR application builder : 14.05.2016 1:47:18
//Target : M16
//Crystal: 10.000MHz
#include <iom16v.h>
#include <macros.h>
//----------------------------
unsigned int input; // змінна для зберігання результату
//----------------------------
//---------- Функція ініціалізації портів
void port_init(void)
{
PORTA = 0;               // початковий стан порту А
DDRA  = 0;               // налаштування порту А на прийом
}
//----------------------------

//---------- Функція переривань АЦП
#pragma interrupt_handler adc_isr:15
```

```
void adc_isr(void)
{
input = ADC;
}
//----------------------------
//---------- Функція ініціалізації АЦП
void adc_init(void)
{
ADCSR = 0b11101110;
//біти ADEN, ADSC, ADATE, ADIE, ADPS2, ADPS1 - «1»
ADMUX =(1<<REFS1)|(1<<REFS0);      //біти REFS1,REFS0 - «1»
}
//----------------------------
//---------- Функція ініціалізації мікроконтролера
void init_devices(void)
{
CLI();
port_init();         //ініціалізація портів
adc_init();          //ініціалізація АЦП
SEI();          //загальний дозвіл переривань
}
//----------------------------
//---------- Головна функція програми
void main(void)
{
init_devices();     //ініціалізація мікроконтролера

while(1)            // нескінченний цикл
     {};
}
//----------------------------
```

4.4. Аналоговий компаратор

Модуль аналогового компаратора включено до складу всіх Mega AVR та tinyAVR мікроконтролерів. Усі мікроконтролери сімейства Classic, за винятком моделей AT90S/LS2323, AT90S/LS2343 і AT90C8534 містять у своєму складі аналоговий компаратор.

Компаратори AVR мікроконтролерів дозволяють порівнювати значення аналогової напруги на неінвертуючому (AIN0) та інвертуючому (AIN1) виводах. У випадку перевищення напруги на неінвертуючому вході величини напруги на інвертуючому вході на виході компаратора (ACO) встановлюється високий рівень. На жаль не існує прямого фізичного виходу компаратора, є лише програмний - біт ACO у регістрі керування та

статусу. Проте аналоговий компаратор має власний прапор переривань. Встановлення цього прапору може викликатися наступними подіями:

- наростаючим фронтом сигналу на виході ACO компаратора;
- падаючим фронтом сигналу на виході ACO компаратора;
- будь якою зміною стану виходу ACO компаратора.

Компаратор також може використовуватись для автотрігерінгу в АЦП, або для спрацювання захвату в таймері лічильнику Т1.

Аналоговий компаратор, який включено до складу мікроконтролера ATmega16 має наступні властивості:

- це компаратор групи загального використання;
- вхідна напруга відсічки не перевищує 40мВ;
- вхідні струми інвертуючого та неінвертуючого входів лежать у межах від -50 до +50 нА;
- типове значення часу затримки спрацювання компаратора залежить від напруги живлення та лежить у межах від 750 до 500 нс.

Структурна схема компаратора мікроконтролера ATmega16

Структурну схема компаратора приведено на рис .4.20.

Базовим вузлом структурної схеми є безпосередньо аналоговий компаратор. У зв'язку з досить великим енергоспоживанням передбачено можливість його відключення за рахунок програмного конфігурування біта керування ACD.

Неінвертуючий вхід компаратора може бути підключеним до вхідного виводу AIN0 мікроконтролера, або до вбудованого джерела еталонної напруги за рахунок конфігурування біта керування ACBG. Інвертуючий вхід компаратора може бути підключеним до вхідного виводу AIN1 мікроконтролера, або до виходу мультиплекора АЦП за рахунок конфігурування бітів керування ACME та ADEN. Для використання вхідних виводів мікроконтролера для введення аналогової напруги

необхідно налаштувати відповідні регістри DDRx на режим приймача, а у регістрах PORTx відключити внутрішні підтягувальні резистори.

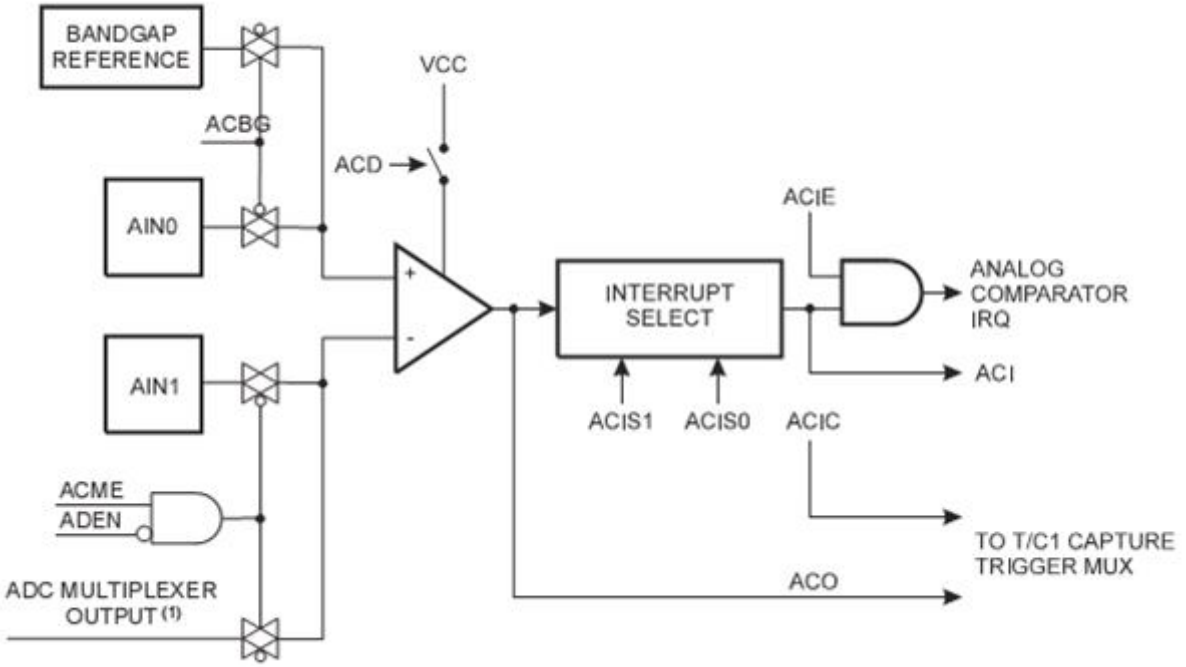


Рис. 4.20. Структурна схема компаратора мікроконтролера ATmega16

Вихід компаратора відображається у біт ACO регістра керування та статусу. Відповідно до стану цього біту працює блок формування прапору переривань ACI. Логіку роботи цього блоку можна змінювати за рахунок налаштування двох біт керування ACIS0, ACIS1. Для компаратора існує власний ресурс дозволу переривань – біт ACIE. Лише у разі дозволу переривань від компаратора та наявності прапора переривань ACI може бути сформований запит переривань.

Для реєстрації вихідного стану компаратора можна використовувати переривання, або біт вихідного стану компаратора AC0 (регістр ACSR).

Регістри керування компаратором

Для керування режимами роботи компаратора використовують регістри ACSR та SFIOR.

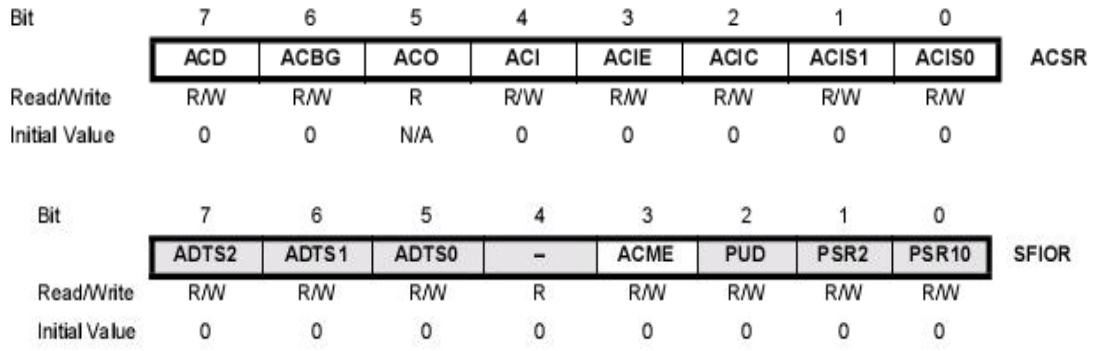


Рис. 4.21

Біти регістру ACSR мають наступне призначення («1» - активна):

- ACD - заборона компаратора;
- ACBG - підключення неінвертуючого входу компаратора до джерела опорної напруги;
- AC0 - вихід компаратора;
- ACI - прапор переривань компаратора;
- ACIE - дозвіл переривань від компаратора;
- ACIC - підключення компаратора до схеми захвату таймера TC1;
- ACIS1, ACIS0 – визначення типу сигналу, що викликає переривання. Нижче у табл. 4.11 приведено відповідність режимів роботи системи переривань від стану бітів ACIS1, ACIS0.

Таблиця 4.11

ACIS1	ACIS0	Режим роботи системи переривань
0	0	Comparator Interrupt on Output Toggle
0	1	Reserved
1	0	Comparator Interrupt on Falling Output Edge
1	1	Comparator Interrupt on Falling Output Edge

У регістрі SFIOR для керування режимами роботи компаратора використовується лише один біт ACME. У залежності від стану цього біту до неінвертуючого входу компаратора підключаються вивід AIN1 мікроконтролера, або вхідні виводи аналогово-цифрового перетворювача,

які визначаються регістром ADMUX. Слід зауважити, що у цьому випадку звичайно використовуються режими роботи з одно полярними сигналами.

У табл. 4.12 приведено інформацію про відповідність бітів керування стану входів компаратора.

Таблиця 4.12

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

Важливу роль при цьому відіграє також стан біту ADEN регістру ADCSRA, який активізує АЦП. У разі використання для компаратора входів мультиплекора АЦП необхідно відключити (ADEN = 0).

Приклад використання компаратора

У прикладі 4.4 приведено програму, де компаратор використовує для порівняння напругу вбудованого джерела та напруги, що заводиться на нульовий канал АЦП (ADC0). Реалізується принцип генерації переривань по будь-якому фронту на виході компаратора (режим Output Toggle - OT). У підпрограмі переривань проводиться аналіз вихідного стану компаратора (біт ACO регістру ACSR) та у залежності від цього формується сигнал на виході LED1.

При ініціалізації компаратора були використані наступні налаштування:

- компаратор включено (ACD=0) ;

- інвертуючий вхід підключено до внутрішнього джерела опорної напруги (ACBG = 1);
- неінвертуючий вхід підключено до виводу мікроконтролера AIN1;
- дозволені переривання від компаратора (ACIE = 1);
- переривання виникають у разі будь якої зміни напруги на виході компаратора (ACIS0=ACIS1=0).

***Приклад* 4.4**

```
//----------------------------
//ICC-AVR application builder : 17.11.2016 1:10:21
//Target : M16
//Crystal: 10.000MHz

#include <iom16v.h>      //файл визначень ATmega16
#include <macros.h>
//---------- Визначення ліній портів
#define LED1 6           //визначення виходу LED1
//----------------------------
//---------- Функція ініціалізації портів
void port_init(void)
{
PORTA = 0b01000000;      //початковий стан на виходіLED1
DDRA  = 0b01000000;      //лінія LED1 - передавач
               //вхід АЦП (PA0) - приймач, підтяжка відсутня
}
//----------------------------
//---------- Функція ініціалізації компаратора
void comparator_init(void)
{
ACSR = (1<<ACBG) | (1<<ACIE); //підключено внутрішню Uref,
                              //дозволені переривання
SFIOR  = (1<<ACME);           //дозволена робота з каналами АЦП
CSRA = 0x00;                  //АЦП виключено
ADMUX  = 0x00; //підключено до входу компаратора лінію АЦП ADC0
}
//----------------------------
//---------- Функція ініціалізація мікроконтролера
void init_devices(void)
{
CLI();                        //заборона переривань
port_init();                  //ініціалізація портів
comparator_init();            //ініціалізація компаратора
SEI();                        //дозвіл переривань
}
//----------------------------
//---------- Функція переривань компаратора
#pragma interrupt_handler ana_comp_isr:17
```

```
void ana_comp_isr(void)
{
if     (ACSR & (1<<ACO))       //перевірка вихідного стану
       {PORTA |= (1<<LED1);}   //"1" на виході LED1
else
       {PORTA &= ~(1<<LED1);}; //"0" на виході LED1
}
//----------------------------
//---------- Головна Функція програми
void main(void)
{
init_devices();               //ініціалізація мікроконтролера
while(1)                      //нескінченний цикл
{};
}
//----------------------------
```

4.5. Система переривань

Мікроконтролери AVR мають систему переривань із одним фізичним рівнем пріоритету. Проте існує можливість логічної організації багаторівневої системи. Кількість джерел переривань залежить від типу мікроконтролера. Пріоритетність цих джерел визначається адресами їх векторів переривання. Чим менша адреса вектора, тим вищий пріоритет переривання. Це означає, що у разі одночасної генерації двох запитів першим буде оброблятися запит із меншою адресою вектора переривання.

Джерела переривань поділяються на два типи. Джерела першого типу формують запити переривань тригерного типу. Прапори переривань цих запитів зберігаються до моменту виконання підпрограми переривань, або їх програмного скидання. Джерела другого типу формують запит переривання у разі існування умов. Відсутня пам'ять цих запитів. Якщо зовнішній сигнал що викликає запит переривання пропадає, то пропадає і запит.

Кожне переривання має власний біт керування дозволом. Крім того в регістрі SREG є біт загального дозволу переривань (I).

У разі виконання підпрограми переривань мікроконтролер записує в стек вміст програмного лічильника (PC) та замість нього розміщує адресу

відповідного вектора переривань. Якщо переривання викликається запитом тригерного типу, то скидається прапор запиту цього переривання. За адресою вектора переривання в пам’яті програм має бути розташована команда переходу на початок підпрограми переривань (jmp addr). На час виконання цієї підпрограми апаратно скидається біт загального дозволу переривань, що забороняє обробку інших переривань. Завершується підпрограма переривань командою *RETI* за якою програмний лічильник завантажується зі стеку адресою точки виходу в переривання та поновлюється загальний дозвіл переривань.

Переривання бувають внутрішні і зовнішні. Прикладами внутрішніх переривання є переповнення таймеру, закінчення передачі (прийому) по UART, тощо. Зовнішні переривання викликаються подіями на виводах мікроконтролера. Так у мікроконтролера ATmega16 є три зовнішніх переривання - INT0, INT1 і INT2 (лише для мікроконтролерів у корпусі PDIP). Ці переривання жорстко «прив'язані» до ліній портів PB2, PD2, PD3 і перепризначити їх до інших ліній портів неможливо.

Для дозволу або заборони зовнішніх переривань призначений керуючий регістр GICR (General Interrupt Control Register).

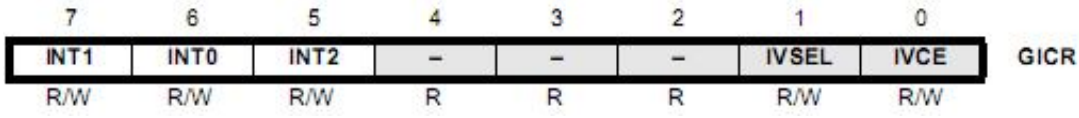


Рис. 4.21

Установка бітів INT1, INT0 або INT2 дозволяє переривання при виникненні запрограмованих типів подій на відповідних виводах мікроконтролера AVR, а скидання - забороняє.

Зовнішні переривання можуть відбуватися за такими типами подій:

- по низькому рівню на виводах INT0, INT1;
- по будь-якій зміні логічного рівня на виводах INT0, INT1;
- по падаючому фронту сигналу на виводах INT0, INT1, INT2;

- по наростаючому фронту сигналу на виводах INT0, INT1, INT2.

Умови генерації переривань встановлюються за допомогою конфігураційних регістрів. Для INT0, INT1 - це регістр MCUCR (MCU Control Register) (рис. 4.22). Для INT2 - MCUCSR (MCU Control and Status Register) (рис. 4.23).

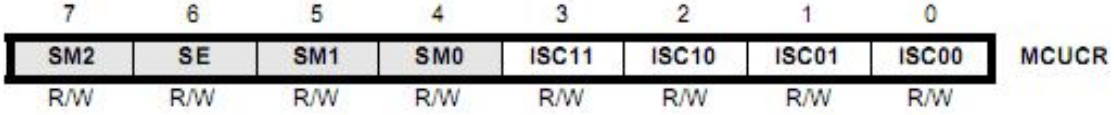


Рис. 4.22

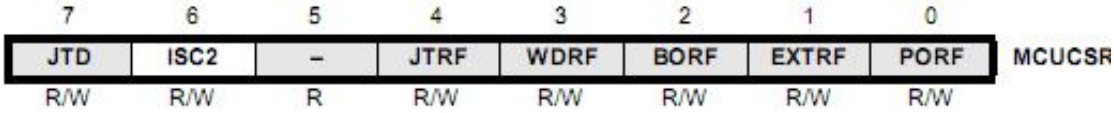


Рис. 4.23

У табл. 4.13 наведені можливі значення розрядів ISC01, ISC00 і відповідні їм умови генерації зовнішнього переривання по входу INT0.

Таблиця 4.13

ISC01	ISC00	Умова генерації зовнішнього переривання INT0
0	0	по низькому рівню виводу INT0
0	1	по будь-якій зміні логічного рівня на виводі INT0
1	0	по падаючому фронту сигналу на виводі INT0
1	1	по наростаючому фронту на виводі INT0

Для переривання по входу INT1 таблиця виглядає аналогічно, тільки керуючі розряди інші – це ISC11 і ISC10. Переривання INT2 може відбуватися тільки по фронту сигналу, тому для установки умов використовується лише один біт – це біт ISC2 регістра MCUCSR (табл. 4.14).

Таблиця 4.14

ISC2	Умова генерації зовнішнього переривання. INT2
0	по падаючому фронту сигналу на виводі INT2
1	по наростаючому фронту на виводі INT2

Останній регістр, має відношення до зовнішніх переривань, - це регістр прапорів переривань GIFR (General Interrupt Flag Register) (рис. 4.24).

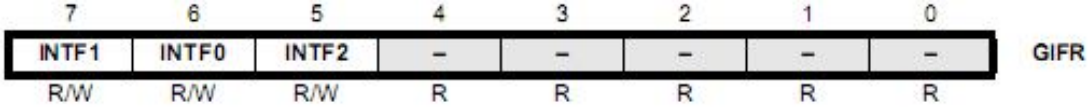


Рис. 4.24

У ньому містяться прапори INTF0, INTF1, INTF2, що встановлюються в разі формування запитів на зовнішні переривання.

Таблиця векторів переривань

Для мікроконтролера ATmega16 номери векторів переривань, символічні імена, та умови їх генерації приведено в табл. 4.15.

Таблиця 4.15

№	Адреса	Джерело	Визначення переривання
1	$000	RESET	Скидання мікроконтролера
2	$002	INT0	Зовнішнє переривання по входу INT0
3	$004	INT1	Зовнішнє переривання по входу INT1
4	$006	TIMER2COMP	Співпадання по таймеру/лічильнику T2
5	$008	TIMER2OVF	Переповнення таймера/лічильника T2
6	$00A	TIMER1CAPT	Захват таймера/лічильника T1
7	$00C	TIMER1COMPA	Співпадання по таймеру/лічильнику T1A
8	$00E	TIMER1COMPB	Співпадання по таймеру/лічильнику T1B
9	$010	TIMER1OVF	Переповнення таймера/лічильника T1
10	$012	TIMER0OVF	Переповнення таймера/лічильника T0
11	$014	SPI, STC	Завершення обміну по SPI
12	$016	USART, RXC	Прийом по USART закінчено
13	$018	USART, UDRE	Вивільнення регістру даних USART
14	$01A	USART, TXC	Передачу по USART закінчено
15	$01C	ADC	Завершення перетворення ADC
16	$01E	EE_RDY	Готовність EEPROM
17	$020	ANA_COMP	Аналоговий компаратор

Продовження табл.. 4.15			
18	$022	TWI	Переривання від контролера TWI
19	$024	INT2	Зовнішнє переривання по входу INT2
20	$026	TIMER0COMP	Співпадання по таймеру/лічильнику T0
21	$028	SPM_RDY	Завершення запису в пам'ять програм

Звичайно ця таблиця розташовується за вказаними адресами. Проте існує можливість програмного переносу адрес векторів переривання на початок BOOT- сектору. У зв’язку з тим, що існує можливість зміни розмірів BOOT- сектору, відповідно можливі і різні варіанти розташування векторів переривання. У залежності від значень фюзесів BOOTSZ1:BOOTSZ0 можливі 4 варіанти розташування початку цієї секції.

Для програмної зміни адрес векторів переривання використовуються біти керування IVSEL та IVCE регістру керування GICR. Біт IVSEL визначає місце розташування таблиці в області прикладної програми (IVSEL = 0) або на початку BOOT- сектора (IVSEL = 1). Біт IVCE виконує функцію допоміжного майстер-біту. Для зміни положення таблиці необхідно спочатку встановити IVCE = 1, а потім протягом 4 циклів мікроконтролера змінити значення біту IVSEL. Після такої операції майстер-біт автоматично скидається.

Організація стеку

При виконанні звичайних підпрограм та підпрограм переривання для апаратного запису адреси повернення в базову програму та програмних записів значень змінних використовується стек. У мікроконтролерів Mega AVR стек реалізується в SRAM області вбудованої пам’яті даних. Зона стеку SRAM області не поділяється на ділянки для апаратних та програмних записів. Хоча в деяких інтегрованих середовищах розробки програмних продуктів мовою Сі, наприклад Image Craft, такий розподіл відбувається на рівні організації startup сегменту програми. Для адресації

використовується дотична адресація на область пам'яті стеку. Для мікроконтролера Mega16 вказівник складається з двох регістрів SPH:SPL. При рестарті мікроконтролера в ці регістри записуються нулеві значення.

При роботі з стеком використовується механізм заповнення «зверху-вниз», тобто з старших адрес до молодших. Використовується FILO організація стеку. Дані, що заносяться до стеку першими зчитуються останніми.

При запису до стеку відбувається переддекрементна модифікація вмісту вказівника. Тобто в регістрах зберігається адреса останньої занятої комірки пам'яті стеку. Після виходу з підпрограм відбувається постінкрементна модифікація вмісту вказівника. У зв'язку з цим, у разі використання підпрограм, необхідно провести ініціалізацію вказівника SPH : SPL. У ці регістри необхідно занести адресу кінцевої комірки SRAM області. Звичайно в файлі опису мікроконтролера, призначеному для програмування мовою асемблеру (наприклад, для мікроконтролера ATmega16 в AVR Studio це m16def.inc, або в ImageCraft ICC це aiom16.s), визначається значення останньої адреси SRAM області як RAMEND.

4.6. Універсальний асинхронний прийомопередавач UART

Загальні відомості

Мікроконтролери AVR мають у своєму складі модуль контролера універсального асинхронного прийомопередавача UART. Ряд мікроконтролерів мають модифікований прийомопередавач USART, який може працювати як в синхронному, так і асинхронному режимах. Через такий контролер здійснюється прийом і передача інформації, представленої послідовним кодом, з зовнішніми пристроями. При роботі в асинхронному режимі контролери USART повністю сумісні з контролерами UART по розміщенню керуючих біт у USART регістрах,

генерації швидкості обміну, протоколам обміну, операціям передачі та прийому.

У сімействі megaAVR мікроконтролерів є мікроконтролери до складу яких входять два таких модулі.

Властивості USART контролера мікроконтролера ATmega16

До складу мікроконтролера ATmega16 входить універсальний синхронно асинхронний прийомопередавач. Він має наступні властивості:

- повний дуплексний режим роботи, який забезпечується незалежними послідовними регістрами прийому та передачі;
- може працювати як в синхронному, так і асинхронному режимах роботи;
- в синхронному режимі може працювати в режимах генерації синхроімпульсів як Master пристроєм, так і Slave;
- генератор швидкості обміну з високою роздільною здатністю;
- можливість роботи з протоколами обміну з 5, 6, 7, 8, 9 бітовими даними та 1 або 2 стоп бітами;
- вбудований генератор біт контролю парності/непарності та засоби їх контролю;
- засоби контролю помилки фрейму та перевищення розрядності даних;
- три незалежних переривання по завершенню прийому, передачі та по спустошенню регістру передачі;
- наявність мультипроцесорного режиму роботи.

На рис. 4.24 зображено структурну схему USART контролера мікроконтролера ATmega16.

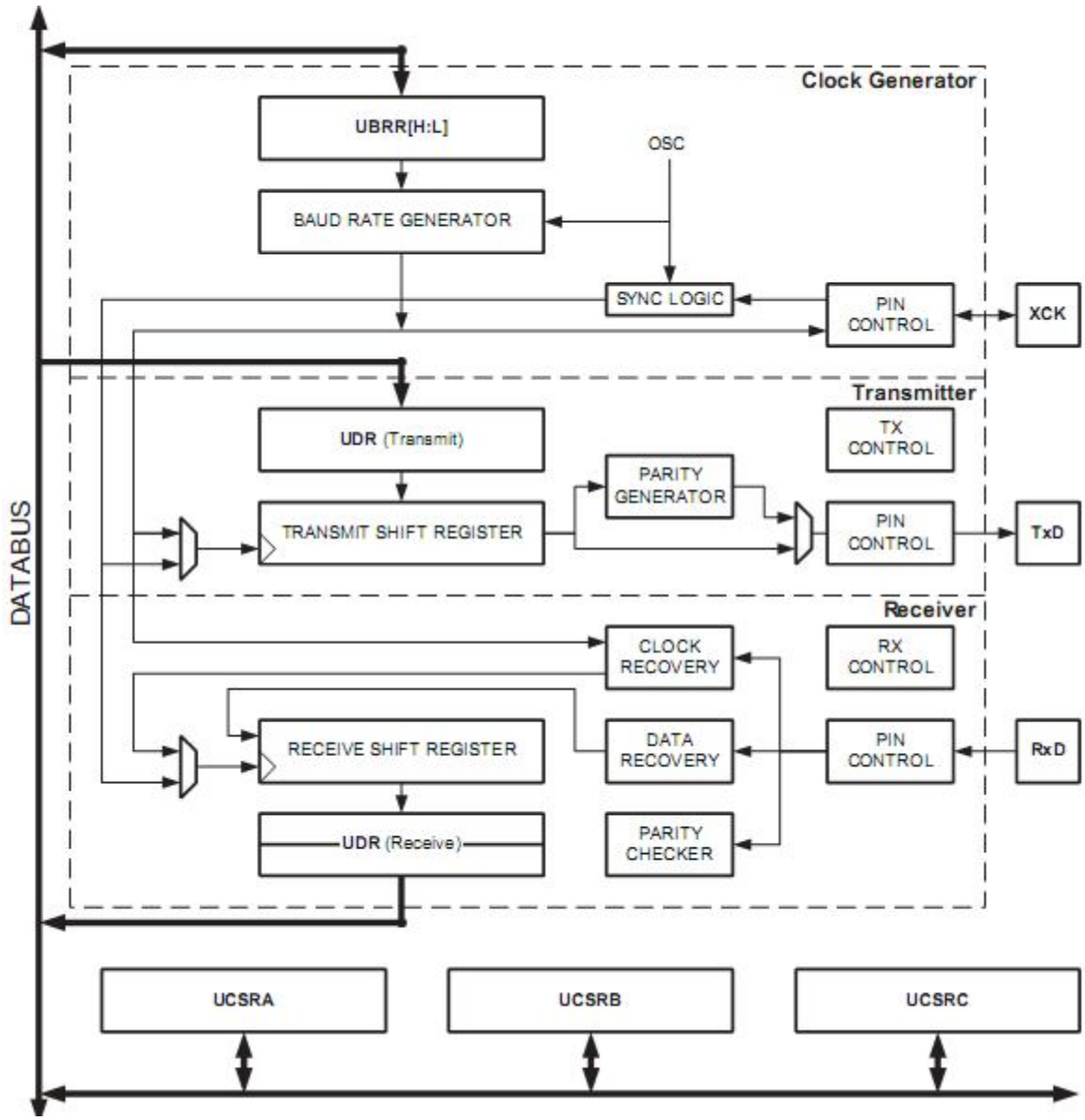


Рис. 4.24 Структурна схема USART контролера ATmega16

На цій схемі присутні наступні елементи:

- Clock Generator – блок тактування з регістром завдання швидкості обміну UBRR;
- Transmitter – передавач з регістром керування UDR (Transmit);
- Receiver – приймач з регістром керування UDR (Receive);
- UCSRA, UCSRB, UCSRC - регістри керування.

Для зв’язку з блоками мікроконтролера використовується внутрішня шина даних DataBus. Зв'язок з периферійними пристроями реалізується через виводи мікроконтролера XCK (порт PB0), TxD (порт PD1), RxD

(порт PD0). Лінія XCK використовується для передачі/прийому імпульсів синхронізації у синхронному режимі обміну. Через лінію RxD заводяться сигнали з зовнішнього передавача. Лінія мікроконтролера TxD використовується для передачі сигналів.

До складу блоку тактування входять:

- Baud Rate Generator – генератор швидкості обміну робота якого визначається регістром керування UBRR;
- Pin Control – драйвер, що формує вхідні/вихідні сигнали лінії XCK мікроконтролера;
- Sync Logic – блок синхронізації.

До складу блоку передавача входять:

- регістр даних UDR (Transmit), до якого програмних шляхом заводяться дані на передачу;
- Transmit Shift Register – регістр зсуву передавача, який забезпечує побітову передачу даних на лінію TxD у відповідності до протоколу обміну;
- Parity Generator – генератор біта парності/непарності;
- Pin Control – драйвер, що формує вихідні сигнали лінії TxD;
- Tx Control – блок керування передавача.

До складу блоку приймача входять:

- регістр даних UDR (Receiver), з якого програмних шляхом вилучаються прийняті дані;
- Receive Shift Register – регістр зсуву приймача, який забезпечує побітовий прийом даних з лінії RxD у відповідності до протоколу обміну;
- Parity Checker – схема контролю парності/непарності прийнятих даних;
- Pin Control – драйвер, що формує вхідні сигнали лінії RxD;
- Rx Control – блок керування приймача.

Потік даних, переданих по каналу USART, являє собою сукупність посилок або кадрів (фреймів). Структуру одного кадру зображено на рис. 4.25.

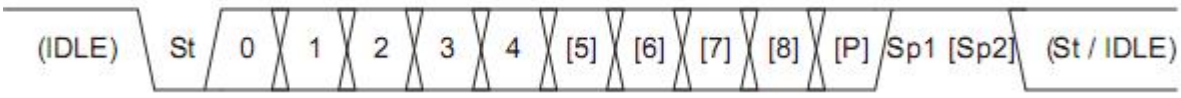


Рис. 4.25 Структура одного кадру

Кожен кадр містить стартовий біт (St), від 5 до 8 бітів даних, у разі обміну з 9 бітами - біт паритету (P) і 1 або 2 стопових біта (Sp1, Sp2). Стартовий біт має рівень логічного 0, стоповий - рівень логічної 1.

Швидкість передачі даних може варіюватися в широких межах, причому високі швидкості передачі можуть бути досягнуті навіть при відносно низькій тактовій частоті мікроконтролера.

Прийняті і передані дані (вісім розрядів) зберігаються в регістрі UDR. Дев’ятий біт при передачі поступає з біту TXB8 регістру UCSRB. При прийомі дев’ятий біт поступає у біт RXB8 регістру UCSRB.

Регістри USART мікроконтролера Atmega16

У мікроконтролера Atmega16 для роботи з USART використовуються наступні регістри вводу/виводу:

- UDR - регістри даних передавача та приймача. Це два регістри, які мають спільне ім’я, але розділені на логічному рівні за рахунок використання різних команд при читанні та запису інформації;
- UCSRA - регістр керування та контролю;
- UCSRB - регістр керування та контролю;
- UCSRC - регістр керування та контролю;
- UBRRH : UBRRL – регістри швидкості обміну.

Регістр даних UDR

Фізично регістр даних складається з двох окремих регістрів, один з яких використовується для передачі даних, інший - для прийому.

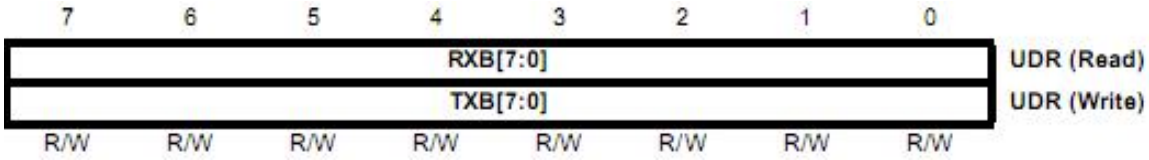


Рис. 4.26

При читанні регістра UDR виконується звертання до регістру приймача, при запису - до регістра передавача. В цьому регістрі зберігаються молодші 8 біт даних обміну. В режимах обміну з 5, 6, 7 бітами при передачі ці біти ігноруються, а при прийомі – у відповідні розряди регістра UDR приймача записується «0».

Регістр керування та контролю UCSRA

Нижче приведено бітову структуру регістра керування та контролю UCSRA.

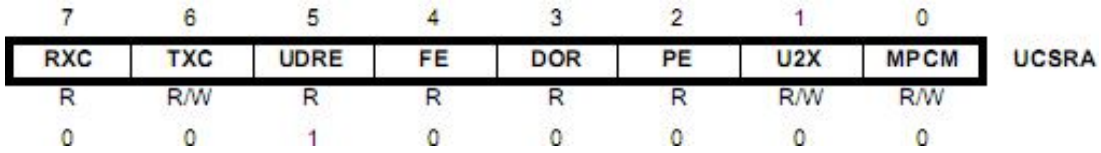


Рис. 4.27

Біти цього регістра мають наступне призначення:

- Біт 7, RXC (Receive Complete) – прийом завершено.

Даний біт встановлюється в стан 1 при пересиланні прийнятого символу з зсувного регістру прийому в UDR незалежно від помилок прийому кадру. Біт RXC очищується при зчитуванні UDR. При прийомі даних ініційованому перериванням, підпрограма обробки переривання повинна забезпечити зчитування даних з регістра UDR, з тим, щоб

очистити RXC, інакше після закінчення підпрограми обробки переривання відбудеться нове переривання;

- біт 6, TXC (Transmit Complete) – передачу завершено.

Даний біт встановлюється в стан 1 коли всі дані (включаючи стоповий біт) виведені із зсувного регістру передачі і в UDR не записані нові дані. Цей прапор використовується в напівдуплексному режимі обміну, коли обладнання передачі має встановити режим прийому і звільнити комунікаційну шину відразу після завершення передачі. Прапор TXC очищується апаратно при виконанні обробки відповідної підпрограми переривання. Очистити біт TXC можна також програмно - записом в біт логічної 1;

- біт 5, UDRE (UART Data Register Empty) – регістр даних порожній.

Даний біт встановлюється в стан 1, якщо символ, записаний в UDR, пересилається в зсувний регістр передачі. Установка цього біта означає, що передавач готовий до отримання нового символу для передачі. Коли біт UDRIE в UCSRB встановлений, та встановлений UDRE, виконується підпрограма переривання по завершенню передачі UART. Біт UDRE очищується при запису нових даних в регістр UDR. Під час скидання мікроконтролера біт UDRE встановлюється в стан 1 з тим, щоб відображати готовність передавача до обміну інформацією;

- біт 4, FE (Framing Error) – помилка кадру.

Даний біт встановлюється в стан 1 при виявленні умов помилкового прийому кадру, тобто коли стоповий біт знаходиться у стані 0. Біт FE очищається при прийомі стопового біта з логічним рівнем 1;

- біт 3, DOR (Data Over Run) – переповнення даних.

Біт DOR встановлюється в стан 1 при виявленні умов переповнення, тобто коли символ, що вже знаходиться в регістрі UDR, не зчитаний перед пересиланням нового символу із зсувного регістру прийому. Біт DOR буферизований, що означає, що він буде залишатися встановленим поки не

будуть зчитані правильні дані з UDR. Біт DOR очищається (скидається в 0) коли дані прийняті і переслані в UDR;

- біт 2, PE (Parity Error) – прапор помилки контролю парності.

Біт встановлюється в стан 1, якщо в даних, які знаходяться у буферному регістрі приймача, виявлено помилку парності. У разі відключення контролю парності цей біт знаходиться в 0 стані;

- біт 1, U2X – біт подвоєння швидкості обміну;
- біт 0, MPCM – біт дозволу режиму мультипроцесорного обміну. У разі встановлення цього біту в 1 стан, SLAVE мікроконтролер очікує прийому кадру з адресою пристрою. Кадри які не мають адресної інформації ігноруються.

Регістр керування та контролю UCSRB

Нижче приведено бітову структуру регістра керування та контролю UCSRB.

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

Рис.4.28

Біти цього регістра мають наступне призначення:

- біт 7, RXCIE (RX Complete Interrupt Enable) - дозвіл переривання по завершенню прийому.

При встановленому в стан 1 біті RXCIE і встановленому дозволі глобального переривання установка біта RXC в регістрі UCSRA призведе до виконання переривання по завершенню прийому;

- біт 6, TXCIE (TX Complete Interrupt Enable) - дозвіл переривання по завершенню передачі.

При встановленому в стан 1 біті TXCIE і встановленому дозволі глобального переривання установка біта TXC в регістрі UCSRA призведе до виконання переривання по завершенню передачі;

- біт 5, UDRIE (UART Data Register Empty Interrupt Enable) - дозвіл переривання при звільненні регістру даних.

При встановленому в стан 1 біті UDRIE і встановленому дозволі глобального переривання установка біта UDRE в регістрі USR призведе до виконання переривання при звільненні регістру даних UART;

- біт 4, RXEN (Receiver Enable) – дозвіл приймача.

Встановлений у стан 1 біт RXEN дозволяє роботу приймача UART. При цьому автоматично визначається режим роботи лінії відповідного порту мікроконтролера. Якщо приймач заборонений, то прапори статусу TXC, DOR і FE стають недійсними;

- біт 3, TXEN (Transmitter Enable) – дозвіл передавача.

Встановлений у стан 1 біт TXEN дозволяє передавач UART. При забороні передавача під час передачі символу, передавач не буде заблокований поки не буде повністю переданий символ із зсувного регістру та наступний символ , що знаходиться в UDR;

- біт 2, UCSZ2, або (9 Bit Characters) - режим 9 розрядного обміну.

При встановленому біті UCSZ2 в стан 1 передаються і приймаються 9-розрядні символи та стартовий і стоповий біти. Дев'яті біти читаються і записуються з використанням бітів RXB8 і TXB8 (відповідно) регістра UCSRB. Дев'ятий біт даних може використовуватися як додатковий стоповий біт або біт контролю парності. Функціонування контролеру у залежності від стану цього біту розглянуто при описі регістра керування UCSRC;

- біт 1, RXB8 (Receive Data Bit 8) - 8-розряд прийнятих даних при 9 бітовому обміну.

При встановленому в стан 1 біті CHR9 біт RXB8 є восьмим бітом даних прийнятого символу;

- біт 0, TXB8 (Transmit Data Bit 8) - 8-розряд даних що передаються при 9 бітовому обміну.

При встановленому в стан +1 біті CHR9 біт TXB8 є дев'ятим бітом даних кадру, що передається.

Регістр керування та контролю UCSRC

На рис .4.29. приведено бітову структуру регістра керування та контролю UCSRC.

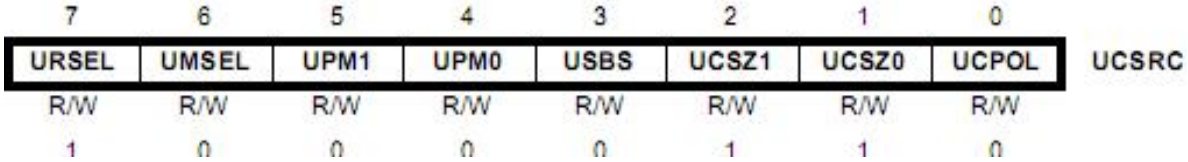


Рис. 4.29

Біти цього регістра мають наступне призначення:

- біт 7, URSEL – біт селекції регістрів UCSRC та UBRRH. У табл. 4.16 описано логіку вибору регістрів UCSRC та UBRRH;
- біт 6, UMSEL – біт вибору синхронного/асинхронного режимів роботи.

Таблиця 4.16

Значення біту	Режим роботи
URSEL = 0	Звернення до регістра UBRRH
URSEL = 1	Звернення до регістра UCSRC

У табл. 4.17 описано логіку налаштування цих режимів роботи;

Таблиця 4.17

Значення біту	Режим роботи
UMSEL = 0	Асинхронний режим обміну
UMSEL = 1	Синхронний режим обміну

• біти 4-5, UPM0, UPM1 – біти налаштування блоку контролю парності. У табл.4.18 описано логіку налаштування цих режимів роботи;

Таблиця 4.18

Значення бітів		Режим роботи
UPM1	UPM0	
0	0	Заборона роботи блоку контролю парності
0	1	Зарезервовано
1	0	Контроль парності
1	1	Контроль непарності

• біт 3, USBS – біт вибору кількості стоп бітів у кадрі даних. У табл. 4.19 описано логіку налаштування формату кадру.

Таблиця 4.19

Значення біту	Режим роботи
USBS = 0	1 стоп біт
USBS = 1	2 стоп біта

• біти 1-2, UCSZ0, UCSZ1, UCSZ2 – налаштування кількості біт у кадрі даних. У табл. 4.20 описано логіку налаштування формату даних у кадрі.

Таблиця 4.20

Значення бітів			Формат даних
UCSZ2	UCSZ1	UCSZ0	
0	0	0	5 біт
0	0	1	6 біт
0	1	0	7 біт
0	1	1	8 біт
1	0	0	Зарезервовано
1	0	1	Зарезервовано
1	1	0	Зарезервовано
1	1	1	9 біт

- біт 0, UCPOL – біт вибору моменту передачі та прийому даних в синхронному режимі обміну. Імпульси синхронізації формуються на лінії XCK мікроконтролера. В асинхронному режимі обміну цей біт повинен бути скинутим в стан логічного 0. У табл. 4.21 описано логіку налаштування цього біту.

Таблиця 4.21

Значення біту	Передача	Прийом
UCPOL = 0	Наростаючий фронт XCK	Падаючий фронт XCK
UCPOL = 1	Падаючий фронт XCK	Наростаючий фронт XCK

Регістр швидкості обміну UBRR (UART Baud Rate Register)

При формуванні протоколу обміну звичайно приділяють увагу двом складовим:

- забезпеченню необхідного формату кадру;
- забезпеченню необхідної швидкості обміну.

Швидкість обміну визначається в одиницях [біт/сек.], або [бод]. Для використання асинхронного методу обміну необхідно щоб протоколи обміну абонентів були узгодженими. Для цього формати кадрів передавача та приймача повинні бути тотожними. Допускається досить мала розбіжність встановлених швидкостей обміну, яка не повинна перевищувати декількох процентів. Виробники мікроконтролера ATmega16 рекомендують налаштовувати швидкість USART контролера з похибкою, що не перевищує 0,5%. У цьому випадку забезпечується краща захищеність каналу передачі від завад. У разі встановлення швидкостей абонентів з великою розбіжністю (7-10%, у залежності від формату кадру) можливе порушення режиму обміну, яке буде приводити до помилки фрейму. Величина помилки визначається наступним чином

$$Err\ [\%] = (BR_{real} - BR)\cdot 100\% / BR,$$

де: BR_{real} – дійсне значення швидкості обміну; BR – бажане значення швидкості.

В асинхронному режимі та в синхронному режимі обміну у якості ведучого швидкість обміну встановлюється за допомогою налаштування регістрів швидкості UBRR (має 2 регістри UBRRH та UBRRL) та біту подвоєння швидкості U2X (регістр UCSRA). Використовуються 12 біт регістра UBRR. Нижче наведено бітовий опис регістрів UBRR.

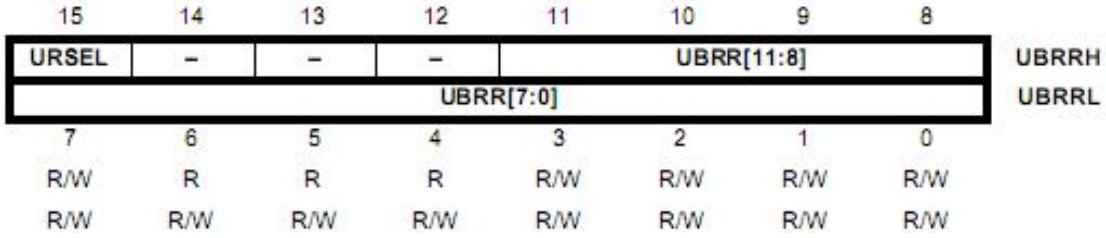


Рис .4.30

Регістри UBRRH та UCSRC мають спільну адресу у просторі регістрів вводу/виводу мікроконтролера ATmega16. Їх селекція забезпечується з використанням біту URSEL у відповідності з табл.2.30.

Для запису інформації у регістр UBRRH необхідно встановити значення біту URSEL = 0. Читання даних з цього регістру відбувається конвеєрним методом. За першою командою читання поступають дані з регістра UBRRH, за другою – з регістра UCSRC.

Швидкість обміну розраховується наступним чином:

- у загальному випадку в режимах асинхронного обміну

$$BR = f_{CLK} \cdot 2^{U2X} / 16(UBRR + 1);$$

- звичайний режим асинхронного обміну (U2X = 0)

$$BR = f_{\mathrm{CLK}} / 16(UBRR + 1);$$

- режим асинхронного обміну з подвоєною швидкістю (U2X = 1)

$$BR = f_{\mathrm{CLK}} / 8(UBRR + 1);$$

- режим синхронного обміну у якості ведучого

$$BR = f_{\mathrm{CLK}} / 2(UBRR + 1),$$

де *BR* — швидкість обміну, f_{CLK}- тактова частота мікроконтролера, *UBRR* - вміст регістра, U2X– розряд регістра керування та статусу UCSRA.

Передача даних

Робота передавача USART розпочинається з моменту його дозволу, тобто встановлення біту TXEN (регістр UCSRB). Це приводить до автоматичної ініціалізації лінії мікроконтролера TXD на режим роботи в якості передавача. У разі використання синхронного методу обміну також автоматично переводиться лінія XCK на режим роботи передавача.

Передача ініціюється записом байту даних до регістра UDR.

У залежності від налаштування формату кадру у зсувний регістр поступають:

- стартовий біт (формується автоматично);
- байт даних з регістра UDR;
- 9-й біт даних з біта TXB8 регістра UCSRB (залежить від налаштування);
- біт контролю парності/непарності (формується автоматично у залежності від налаштування);
- 1 або 2 стоп біти (формуються автоматично у залежності від налаштування).

Після завантаження зсувного регістра формується прапор UDRE регістра UCSRA, якій сигналізує про вивільнення регістра даних UDR та може викликати відповідне переривання.

Якщо на інтервалі передачі кадру до регістра даних UDR записується нове слово, то воно поступає в зсувний регістр лише після передачі останнього стоп біту.

Після завершення передачі встановлюється прапор TXC який може викликати відповідне переривання.

Для вимкнення передавача необхідно перевести біт TXEN у стан логічного «0». Проте передавач буде вимкнено лише після завершення поточної передачі з зсувного регістра та передачі даних з регістра UDR.

Прийом даних

Робота приймача USART розпочинається з моменту його дозволу, тобто встановлення біту RXEN (регістр UCSRB). Це приводить до автоматичної ініціалізації лінії мікроконтролера RXD на режим роботи в якості приймача. У разі використання синхронного методу обміну також автоматично переводиться лінія XCK на режим роботи приймача.

Прийом даних розпочинається після автоматичного визначення стартового біта. Проводиться потрійне опитування кожного біта даних та з використанням мажоритарного методу визначається їх значення. Швидкість опитування визначається налаштуванням мікроконтролера або сигналами синхронізації, які поступають на лінію XCK в режимі синхронного обміну.

У залежності від налаштування формату кадру мікроконтролера із зсувного регістра приймача передається наступна інформація:

- байт даних передається до регістра даних UDR приймача;
- 9-й біт даних поступає в біт RXB8 регістра UCSRB (залежить від налаштування);
- автоматично у залежності від налаштування визначається признак парності/непарності прийнятого байта даних та порівнюється з прийнятим значенням. У залежності від результату порівняння формується прапор помилки парності/непарності PE (регістр UCSRA);
- якщо стоп біт невірно визначений (логічний рівень 0) автоматично визначається помилка кадру FE (регістр UCSRA);
- якщо до моменту надходження нового кадру не зчитані попередні дані автоматично визначається помилка переповнення даних DOR (регістр UCSRA).

По завершенню прийому кадру автоматично встановлюється прапор RXC який може викликати відповідне переривання. Цей прапор автоматично скидається після зчитування даних з регістра UDR приймача.

Для вимкнення приймача необхідно перевести біт RXEN у стан логічного «0». На відміну від передавача приймач вимикається негайно.

Приклади налаштування USART контролера

Приклад 4.5. Провести ініціалізацію USART контролера мікроконтролера ATmega16 на наступний режим роботи:

- частота генератора 10 МГц;
- протокол обміну - 8 біт даних, 1 стоп біт;
- швидкість обміну 9600 бод;
- дозволені приймач та передавач;
- дозволені переривання від приймача та передавача;
- заборонено контроль парності/непарності.

Для ініціалізації USART контролера досить зручно використовувати вбудований ресурс Application Builder програмного пакету ICCAVR Image Craft.

На першому етапі у закладці CPU (рис. 4.31) проводиться визначення типу мікроконтролера та його робочої частоти.

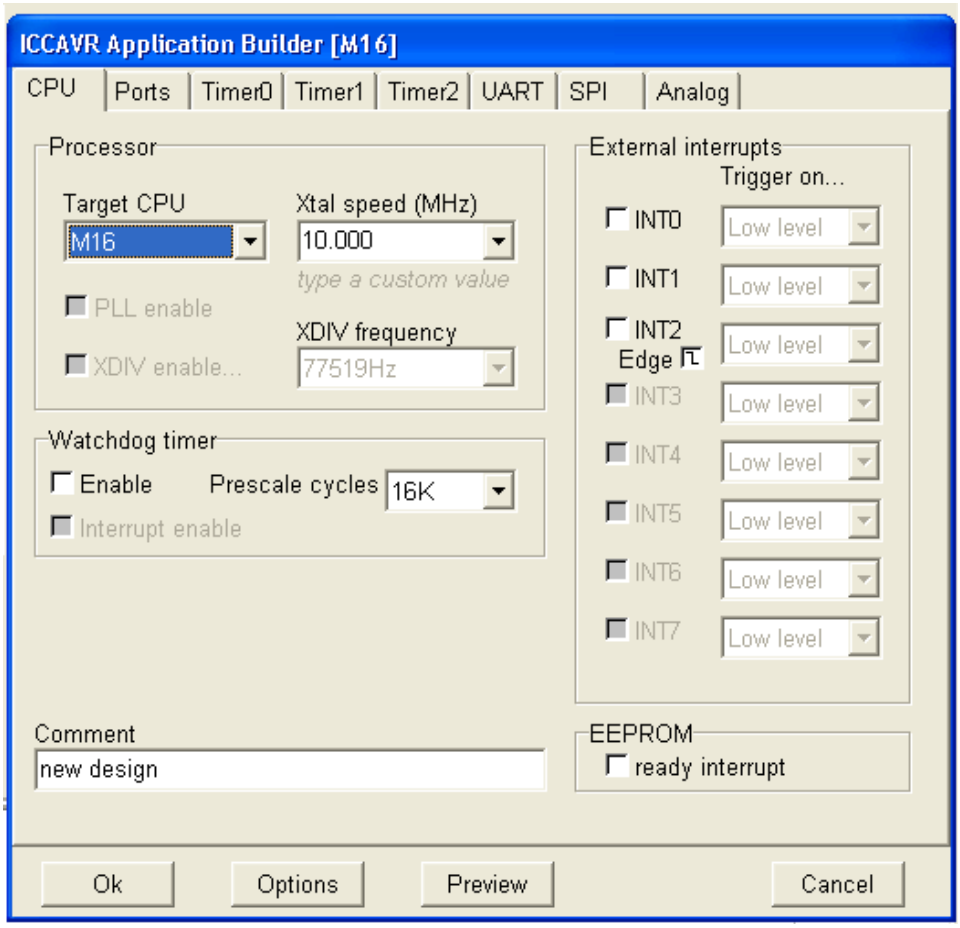


Рис. 4.31 Визначення типу мікроконтролера та робочої частоти

На другому етапі у закладці UART (рис. 4.32) визначаються налаштування параметрів контролера.

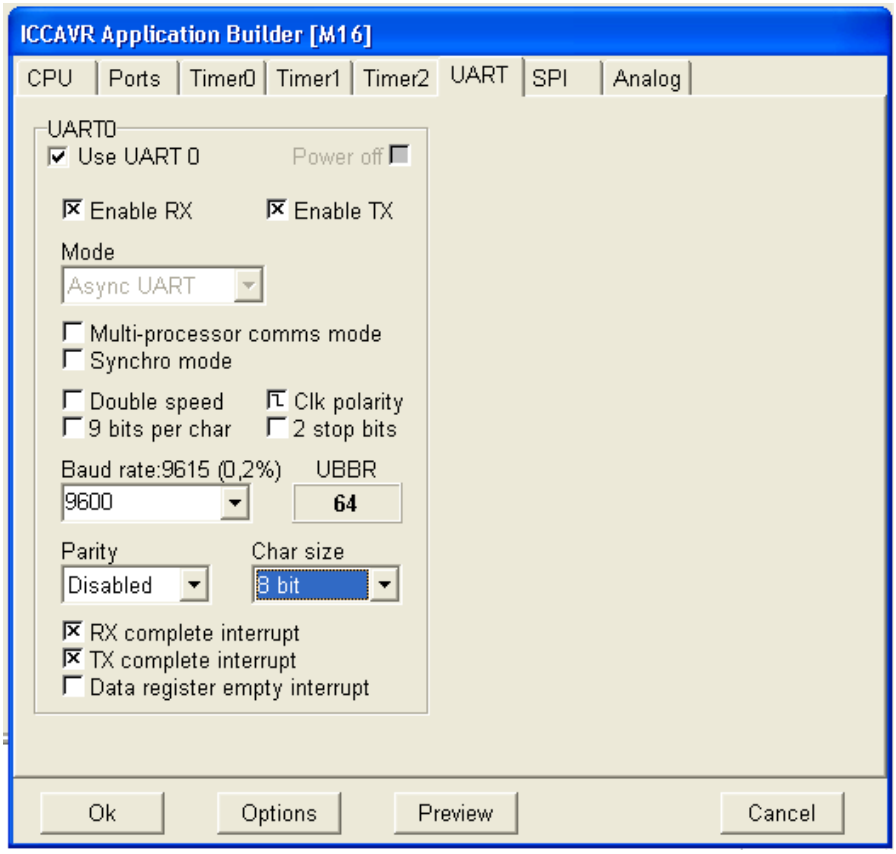


Рис. 4.32 Налаштування режиму роботи USART контролера

Після згоди з такими налаштуваннями (натиснено кнопку «OK») формується файл з функціями ініціалізації. У функції ініціалізації USART контролера вказуються параметри протоколу обміну, автоматично визначаються константи завантаження регістрів керування та швидкості обміну. Вказуються бажана та реальна швидкості обміну та величина похибки. Визначаються точки переривання та генеруються порожні функції переривання.

Автоматично генерований текст має наступний вигляд.

```
// ICC-AVR application builder : 17.05.2016 18:11:11
// Target : M16
// Crystal: 10.000Mhz

#include <iom16v.h>
#include <macros.h>
```

```
void port_init(void)
{
 PORTA = 0x00;
 DDRA  = 0x00;
 PORTB = 0x00;
 DDRB  = 0x00;
 PORTC = 0x00; //m103 output only
 DDRC  = 0x00;
 PORTD = 0x00;
 DDRD  = 0x00;
}

//UART0 initialize
// desired baud rate: 9600
// actual: baud rate:9615 (0,2%)
// char size: 8 bit
// parity: Disabled
void uart0_init(void)
{
 UCSRB = 0x00; //disable while setting baud rate
 UCSRA = 0x00;
 UCSRC = BIT(URSEL) | 0x06;
 UBRRL = 0x40; //set baud rate lo
 UBRRH = 0x00; //set baud rate hi
 UCSRB = 0xD8;
}

#pragma interrupt_handler uart0_rx_isr:12
void uart0_rx_isr(void)
{
 //uart has received a character in UDR
}

#pragma interrupt_handler uart0_tx_isr:14
void uart0_tx_isr(void)
{
 //character has been transmitted
}

//call this routine to initialize all peripherals
void init_devices(void)
{
 //stop errant interrupts until set up
 CLI(); //disable all interrupts
 port_init();
 uart0_init();

 MCUCR = 0x00;
 GICR  = 0x00;
 TIMSK = 0x00; //timer interrupt sources
 SEI(); //re-enable interrupts
 //all peripherals are now initialized
}
```

У прикладах 4.6 – 4.8 використано матеріали опису мікроконтролера ATmega16 – mega16.pdf.

***Приклад* 4.6.** Провести ініціалізацію USART контролера мікроконтролера ATmega16 на наступний режим роботи:

- швидкість обміну передається як параметр у функцію ініціалізації;
- дозволені приймач та передавач,
- протокол обміну - 8 біт даних, 2 стоп біти.

```
void USART_Init (unsigned int baud)
{
// Встановлення швидкості обміну
UBRRH = (unsigned char) (baud >>8);
UBRRL = (unsigned char) baud;
// Дозвіл приймача та передавача
UCSRB = (1<<RXEN) | (1<<TXEN);
// Встановлення формату кадру: 8 біт даних, 2 стоп біти
UCSRC = (1<<URSEL) | (1<<USBS) | (1<<UCSZ1) | (1<<UCSZ0);
}
```

***Приклад* 4.7.** З використанням контролера USART мікроконтролера ATmega16 передати байт. Передачу активізувати після спустошення буфера передавача.

```
void USART_Transmit (unsigned char data)
{
// Очікування спустошення буферу передавача
while ( !(UCSRA & (1<<UDRE) ) );
// пересилання даних
UDR = data;
}
```

***Приклад* 4.8.** З використанням контролера USART мікроконтролера ATmega16 прийняти байт. Зчитати дані з буфера приймача після завершення прийому кадру.

```
unsigned char USART_Receive (void)
{
// Очікування завершення прийому кадру
while ( !(UCSRA & (1<<RXC) ) );
// Отримання прийнятого байту з буферу та його повернення з функції return UDR;
}
```

Контрольні питання

1. Паралельні порти AVR-мікроконтролерів. Режими роботи портів.
2. Структура порту введення/виведення AVR-мікроконтролерів.

3. Регістри керування портами введення/виведення AVR-мікроконтролерів та їх призначення.

4. Конфігурування паралельних портів.

5. Початкова ініціалізація паралельних портів.

6. Таймер – рахівник 0. Структурна схема та режими роботи.

7. Таймер – рахівник 1. Структурна схема та режими роботи.

8. Таймер – рахівник 2. Структурна схема та режими роботи.

9. Робота таймерів / лічильників в режимі Normal.

10. Робота таймерів / лічильників в режимі CTC.

11. Робота таймерів / лічильників в режимі Fast PWM.

12. Робота таймерів/лічильників в режимі Phase Correct PWM.

13. Які відмінності між таймерами / лічильниками T0, T1 і T2?

14. Які переривання використовуються таймерами/лічильниками?

15. Як відбувається керування преддільником таймерів/лічильників?

16. Робота таймера/лічильника в режимі Normal.

17. Робота таймера/лічильника в режимі CTC.

18. Робота таймера/лічильника в режимі швидкого PWM.

19. Функція захвату (Capture) таймера/лічильника.

20. Функція порівняння (Compare) таймера/лічильникаю

21. Регістри керування таймера/лічильника T1.

22. Watchdog - таймер. Структурна схема.

23. Аналого-цифровий. перетворювач. Структурна схема та режими роботи.

24. Регістри керування аналогово-цифрового перетворювача.

25. Аналоговий компаратор мікроконтролера .

26. Регістри керування компаратором.

27. Система переривань мікроконтролери AVR.

28. Таблиця векторів переривань.

29. Організація стеку мікроконтролерів Mega AVR.

30. Послідовний інтерфейс UART Структурна схема та режими роботи.

31. Властивості USART контролера мікроконтролера ATmega16

32. Регістри USART мікроконтролера Atmega16

33. Регістр даних UDR

34. Регістр керування та контролю UCSRA

35. Регістр керування та контролю UCSRB

36. Регістр керування та контролю UCSRC

37. Регістр швидкості обміну UBRR.

38. Робота передавача USART.

39. Робота приймача USART.

РОЗДІЛ 5. РЕЖИМИ РОБОТИ ТА СИСТЕМА КОМАНД

5.1. Режими зменшеного енергоспоживання

Досить часто при використанні автономного живлення для економії енергії батареї живлення в МК використовуються режими зменшеного енергоспоживання. Завдяки тому, що в AVR мікроконтролерах використовується декілька лінії тактування, пов'язаних з різними блоками, за рахунок відключення цих ліній та відповідних вузлів мікроконтролерів можливе зменшення споживаного струму. У різних моделей AVR мікроконтролерів підтримується від 3 до 6 режимів зменшеного споживання.

Виділяють наступні режими зменшеного енергоспоживання:

- режим холостого ходу - *IDLE*;
- режим мікроспоживання – *PowerDown*;
- економічний режим – *PowerSave*;
- режим придушення шуму - *ADC Noise Reduction;*
- основний режим очікування –*Standby;*
- додатковий режим очікування - *Extended Standby*.

У AVR мікроконтролера ATmega16 можливе встановлення програмними шляхом всіх 6 типів режимів зменшеного енергоспоживання.

В табл. 5.1 приведено дані про активність в різних режимах енергозбереження ліній сінхроімпульсів, основного та асинхронного (виконаного на таймері ТС2) генераторів, базових блоків мікроконтролера ATmega16.

В режимі холостого ходу (*IDLE*) припиняє роботу тільки процесор і фіксується вміст пам'яті даних, а внутрішній генератор синхросигналів, таймери, система переривань і *WDT*- таймер продовжують функціонувати. Тому значної економії не виходить: споживання знижується лише на 30-

50%. Режим Idle має сенс використовувати тоді, коли загальне споживання пристрою лімітується саме МК, який при цьому обов'язково повинен знаходитися в стані постійної готовності. У всіх інших випадках слід вибирати режими "глибокого" енергозбереження, коли власне споживання МК знижується до одиниць або десятків мікроампер.

Таблиця 5.1

	Active Clock domains					Oscillators			Wake-up sources				
Sleep Mode	**clk_{CPU}**	**clk_{CPU} clk_{FLASH}**	**clk_{CPU} clk_{IO}**	**clk_{CPU} clk_{ADC}**	**clk_{CPU} clk_{ASY}**	**Main Clock Source Enabled**	**Timer Osc. Enabled**	**INT2 INT1 INT0**	**TWI Address Match**	**Timer 2**	**SPM/ EEROM Ready**	**ADC**	**Other I/O**
Idle			X	X	X	X	X(2)	X	X	X	X	X	X
ADC Noise Reduction				X	X	X	X(2)	X(3)	X	X	X	X	
Power Down								X(3)	X				
Pover Save					X(2)		X(2)	X(3)	X	X(2)			
Standby						X		X(3)	X				
Extendad Standb					X(2)	X	X(2)	X(3)	X	X(2)			

В режимі мікроспоживання (*PowerDown*) зберігається вміст регістрового файлу, але зупиняється внутрішній генератор синхросигналів. Най економніший режим. У цьому режимі зупиняється все, крім сторожового таймера (якщо його включити), зовнішніх переривань і TWI. Тільки зовнішні скидання, скидання сторожовим таймером, переривання TWI або зміна рівня на входах INT0 або INT1 може розбудити мікроконтролер. Слід врахувати, що в цьому режимі зупиняється тактовий генератор, тому щоб прокинутися мікроконтролеру може знадобитися якийсь час. При включеному *WDG*-таймері струм споживання в цьому режимі складає біля 60...80 мкА, а при виключеному - менше 1 мкА для всіх типів *AVR*. Вищенаведені значення справедливі для величини живлячої напруги 5 В.

Режим зберігання енергії (*PowerSave*) може бути реалізований тільки в тих *AVR*, що мають у своєму складі систему реального часу. У основному, режим *PowerSave* ідентичний *PowerDown*, але він допускає незалежну роботу таймера TC2. Вихід із режиму *PowerSave* можливий по перериванню, викликаному або переповненням, або спрацьовуванням блока порівняння цього лічильника. Струм споживання в цьому режимі складає 6...10 мкА при напрузі живлення 5 В на частоті 32,768 кГц.

В режимі придушення шуму при роботі аналого-цифрового перетворювача (*ADC Noise Reduction*) зупиняється робота процесора, але дозволена робота АЦП, двопровідного інтерфейсу TWI і сторожового таймера. Цей режим служить для зменшення різних наведень під час перетворення АЦП. Крім переривання після завершення перетворення АЦП, мікроконтролер з цього режиму енергозбереження може вивести зовнішнє скидання, скидання сторожовим таймером, переривання TWI, переривання таймера TC2, переривання готовності EEPROM, встановлення низького рівня на входах INT0 або INT1.

Основний режим очікування (*Standby*) відрізняється від режиму *Power Down* тим, що робота тактового генератора не припиняється. Це гарантує швидкий вихід мікроконтролера з режиму очікування усього за 6 тактів генератора. В цьому режимі заборонена робота асинхроного генератора на таймері TC2.

Додатковий режим очікування (*Extended Standby*) ідентичний режиму *Power Save,* але робота тактового генератора теж не припиняється, що гарантує швидкий вихід з режиму за 6 тактів генератора. На відміну від попереднього режиму дозволена робота асинхроного генератора на таймері TC2.

Для керування режимами пониженого енергоспоживання мікроконтролера ATmega16 використовується регістр управління MCUCR. Бітову структуру цього регістру зображено на рис. 5.1.

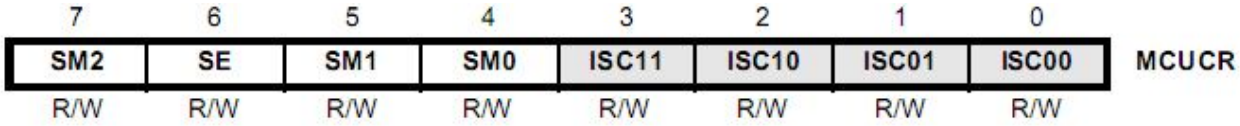


Рис. 5.1

Біт 7 – SE – дозвіл сплячого режиму. Цей біт повинен бути встановлений в "1", щоб МК зміг увійти в один з режимів пониженого енергоспоживання.

Біти 6:4 - SM2:0 забезпечують вибір певного режиму роботи.

В табл. 5.2 приведені дані про стан цих бітів в різних режимах роботи.

Таблиця 5.2

SM2	SM1	SM0	Sleep Mode
0	0	0	Idle
0	0	1	ADC Noise Reduction
0	1	0	Power-down
0	1	1	Power-save
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Standby
1	1	1	Extended Standby

Для мінімізації споживання енергії можуть також використовуватись програмні та апаратні засоби, які дозволяють відключати ряд вузлів мікроконтролера. До таких засобів відносяться:

- аналогово-цифровий перетворювач;
- аналоговий компаратор;
- вбудований супервізор напруги (Brown-out Detector)$
- сторожовий таймер (WDT);
- вбудоване джерело еталонної напруги;
- драйвери вхідних портів.

Якщо використовуються режими зменшеного енергоспоживання в яких не використовуються вхідні порти їх необхідно конфігурувати на режим із зменшеним споживанням. Для цього за допомогою встановлення біта PUD в регістрі SFIOR блокують підтяжки вхідних буферів портів.

5.2. Система команд мікропроцесора

Система команд AVR містить п'ять груп: умовного розгалуження, безумовного розгалуження, арифметичних і логічних операцій, команди пересилки даних, команди роботи з бітами. У версіях AVR мікроконтролерів сімейства Mega реалізована функція апаратного множення. Кількість команд навіть у межах одного сімейства AVR мікроконтролерів може змінюватися. Наприклад, для мікроконтролера Atmega103 до складу системи команд входить 121 інструкція.

Прості моделі AVR не мають деяких команд. Основна відмінність полягає в тому, що ті мікроконтролери (AT90S1200, Attiny10/11), у яких відсутня SRAM, не містять і відповідних команд роботи з оперативною пам'яттю. Крім того, AT90S1200 не має команд ADIW, SBIW, IJMP, ICALL, LPM, а Attiny10/11 – команд ADIW, SBIW, IJMP, ICALL, LPM, а ATtiny10/11 - команд ADIW, SBIW, IJMP, ICALL.

Мікроконтролери AVR сімейства Mega мають двобайтові, виконувані за три такти, команди абсолютних переходів JMP і CALL. В сімействах AVR мікроконтролерів Tiny і Classic ці команди не використовуються, тому що весь адресний простір обсягом до 4K слів досяжний за допомогою команд відносних переходів RJMP, RCALL. Також команда ELPM сторінкового читання FLASH пам'яті використовується лише в мікроконтролерах з великим обсягом пам'яті. Наприклад, вона входить до набору інструкцій мікроконтролера Atmega103 об’єм пам'яті програм якого має 128 Кбайт.

Спеціальна директива асемблера **device <тип AVR>** забезпечує контроль відповідності команд, використовуваних у тексті програми, типу зазначеного процесора.

Способи адресації операндів

Існує два способи адресації операндів AVR- мікроконтролерів: пряма адресація та непряма. Однак кожний спосіб адресації має кілька різновидів залежно від того, до якої області пам'яті відноситься звернення (для прямої адресації) або які додаткові дії виконуються над індексним регістром (для непрямої адресації).

При **прямій адресації** адреси операндів знаходяться безпосередньо в слові команди. У відповідності зі структурою пам'яті даних існують наступні різновиди прямої адресації: пряма адресація одного регістру загального призначення (РЗП), пряма адресація двох РЗП, пряма адресація регістру введення/виведення (РВВ), пряма адресація регістрів оперативного запам’ятовуючого пристрою (ОЗП).

Пряма адресація одного регістру загального призначення використовується в командах, що оперують із одним з регістрів загального призначення. При цьому адреса регістру - операнду (його номер) знаходиться в розрядах 8 ... 4 (5 біт) слова команди. Прикладом команд, що використовують цей спосіб адресації, є команди роботи зі стеком (PUSH, POP), команди інкременту (INC), декременту (DEC), а також деякі команди арифметичних операцій.

Пряма адресація двох регістрів загального призначення використовується в командах, що оперують одночасно із двома регістрами загального призначення. При цьому адреса регістру - джерела знаходиться в розрядах 9, 3...0 (5 біт), а адреса регістру - приймача - в розрядах 8...4 (5 біт) слова команди. До команд, що використовують цей спосіб адресації, відносяться команда пересилання даних з регістру в регістр (MOV), а також більшість команд арифметичних операцій.

Пряма адресація регістру введення/виведення (рис. 2.41) використовується в командах пересилання даних між регістром введення/виведення та регістровим файлом — IN і OUT. У цьому випадку адреса регістру введення/виведення знаходяться в розрядах 10, 9, 3...0 (6 біт), а адреса РЗП — у розрядах 8...4 (5 біт) слова команди.

Пряма адресація ОЗП використовується при зверненні до всього адресного простору пам‘яті даних (рис. 5.2). Цей спосіб адресації не підтримується в деяких простих версіях AVR- мікроконтролерів, наприклад AT90S1200.

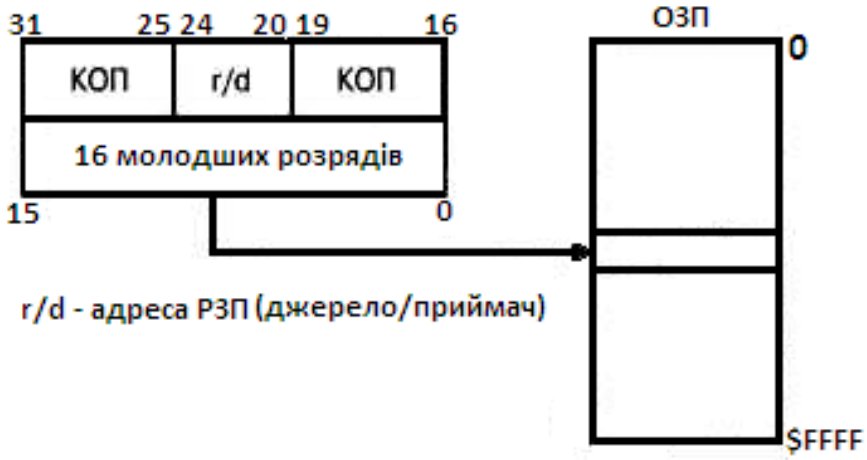


Рис. 5.2 Пряма адресація ОЗП

У системі команд мікроконтролерів є тільки дві команди, що використовують цей спосіб адресації. Це команди пересилання байта між одним з РЗП і коміркою ОЗП — LDS і STS. Кожна із цих команд займає в пам'яті програм два слова (32 біта). У першому слові міститься код операції й адреса регістру загального призначення (у розрядах з 8-го по 4-й). У другому слові знаходиться адреса комірки пам'яті, до якої відбувається звернення.

При використанні команд *простої непрямої адресації* (рис. 5.3) звернення відбувається за адресою (регістру — для AT90S1200, комірки

пам'яті — для інших моделей), яка знаходиться в індексному регістрі. Ніяких дій із умістом індексного регістру при цьому не виконується.

Мікроконтролери підтримують 6 команд (по 2 для кожного індексного регістру) простої непрямої адресації: LD Rd, X/Y/Z(пересилання байта з ОЗП в РЗП) і ST X/Y/Z, Rd (пересилання байта з РЗП в ОЗП). Адреса регістру загального призначення знаходиться в розрядах 8.. .4 слова команди.

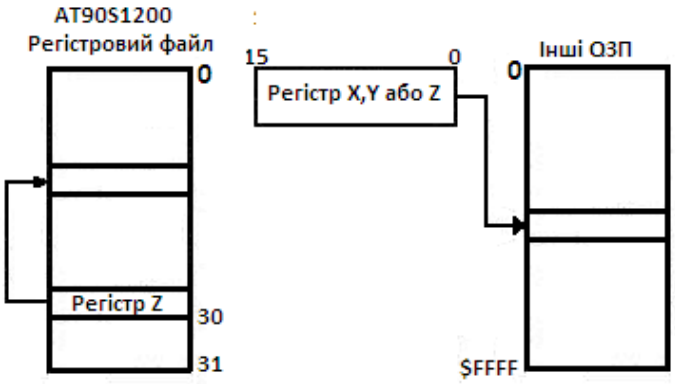


Рис. 5.3 Проста непряма адресація

При використанні команд *відносної непрямої адресації* (рис. 5.4) адреса комірки пам'яті, до якої відбувається звернення, визначається додаванням умісту індексного регістру (Y або Z) і константи, що задається в команді. Інакше кажучи, відбувається звернення за адресою, зазначеному в команді, щодо адреси, що перебуває в індексному регістрі. Відповідно мікроконтролери підтримують 4 команди відносної непрямої адресації (дві для регістру Y і дві для регістру Z): LDD Rd, Y+q/Z+q (пересилання байта з ОЗП в РЗП) і ST Y+q/Z+q, Rd (пересилання байта з РЗП в ОЗП). Адреса регістру загального призначення знаходиться в розрядах 8.. .4 слова команди, а величина зсуву — у розрядах 13, 11, 10, 2...0. Оскільки під значення зсуву приділяється тільки 6 біт, воно не може перевищувати 64.

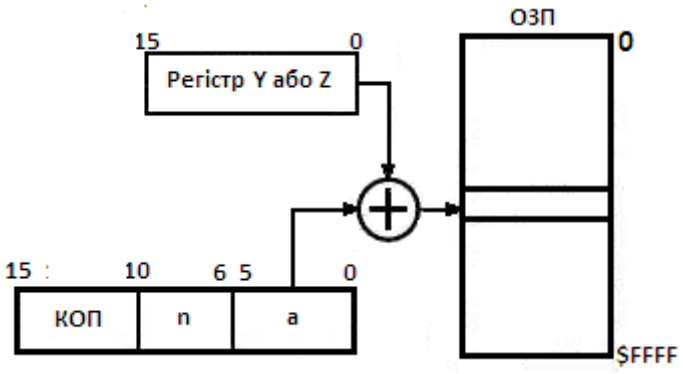


Рис 5.4 Відносна непряма адресація

При використанні команд *непрямої адресації із попереднім декрементом* (рис. 5.5) уміст індексного регістру спочатку збільшується на 1, а потім відбувається звернення по отриманій адресі. Мікроконтролери сімейства підтримують 6 команд (по 2 для кожного індексного регістру) непрямої адресації із попереднім декрементом: LD Rd, -X/-Y/-Z (пересилання байта з ОЗП в РЗП) і ST -X/-Y/-Z, Rd (пересилання байта з РЗП в ОЗП).

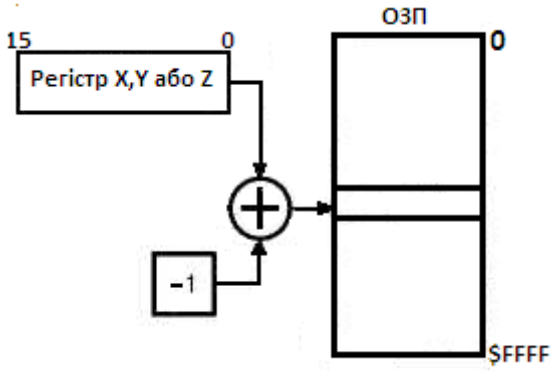


Рис. 5.5 Непряма адресація з попереднім декрементом

Адреса регістру загального призначення знаходиться в розрядах 8...4 слова команди.

При використанні команд *непрямої адресації з постінкрементом* (рис. 5.6) після звернення за адресою, яка знаходиться в індексному регістрі, вміст індексного регістру зменшується на 1. Мікроконтролери сімейства підтримують 6 команд (по 2 для кожного індексного регістру) непрямої адресації з постінкрементом: LD Rd, X + / B + / Z + (пересилання

байта з ОЗП в РЗП) і ST X + / B + / Z +, Rd (пересилання байта з РЗП в ОЗП). Адреса регістру загального призначення знаходиться в розряду 8 ... 4 слова команди.

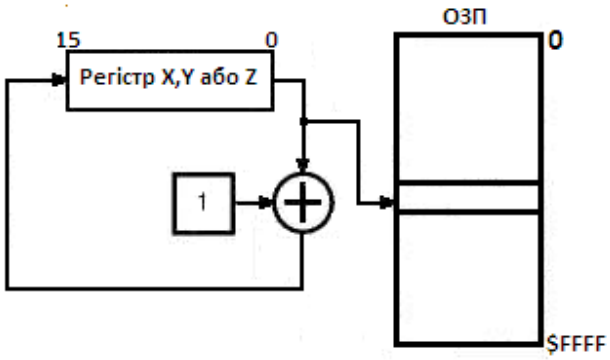


Рис. 5.6 Непряма адресація з постінкрементом

Розробка програмного забезпечення

При розробці програмного забезпечення AVR мікроконтролерів використовуються наступні мовні засоби:

- вся програма розробляється мовою асемблеру. Такий підхід застосовується при використанні мікроконтролерів з обмеженими об'ємами пам'яті програм та пам'яті даних та у разі наявності жорстких вимог щодо швидкодії мікроконтролерної системи;
- вся програма розробляється мовою Сі. У цьому випадку звичайно використовуються мікроконтролери з збільшеними об'ємами пам'яті у порівнянні з попереднім випадком. Програмне забезпечення звичайно займає більший об'єм пам'яті програм та виконується повільніше. Такий підхід до розробки програмного забезпечення виправдано при розробці складних програм обробки інформації при можливості використання тривалого часу;
- програма розробляється мовою Сі, але використовуються програмні вставки, які написані мовою асемблеру. Такий підхід використовується при розробці програмного продукту з досить простими, але швидкими драйверами периферійних пристроїв;

- програма розробляється мовою Сі з використанням окремих файлів, написаних мовою асемблеру. Такі засоби використовуються у разі необхідності створення програмного забезпечення мікроконтролерних систем, які характеризуються як підвищеною складністю алгоритмів обробки інформації, так і необхідністю використання швидких драйверів периферійних пристроїв.

Розробку програмного забезпечення AVR мікроконтролерів мовою асемблеру зазвичай проводять в інтегрованому середовищі AVR Studio. Методи створення проектів, їх компіляції та налагодження програмного забезпечення добре описані в літературі.

Для розробки проектів мовою Сі використовується декілька інтегрованих середовищ, наприклад WinAVR, Image Craft, Code Vision. Звичайно програмні продукти погано переносяться з одного середовища в інше. Проте в літературі також багато інформації про їх використання.

Досить складно знайти інформацію про використання методів асемблерних вставок та комбінованої побудови проектів на основі фрагментів програм розроблених мовою Сі та асемблеру.

Метод асемблерних вставок

В асемблерних вставках використовується синтаксис асемблера, який дозволяє компілятору самостійно використовувати регістри для операндів команд. В залежності від цього автоматично розподіляються регістри як у попередніх так і подальших функціях програми. В асемблерних вставках можливий автоматичний доступ до зовнішніх змінних.

Синтаксис асемблерних вставок суттєво залежить від типу інтегрованого середовища розробки програми мовою Сі.

Синтаксис асемблерних вставок у програму, розроблену мовою Сі в середовищі Image Craft, має наступний вигляд:

- Для одиночних вставок команд

```
asm ("nop");
```

Досить часто застосовують асемблерну вставку такому форматі

```
asm volatile("nop");
```

Модифікатор типу "volatile" забороняє оптимізатору компілятора змінювати, а у деяких випадках навіть вилучати таку інструкцію. Наприклад, при використанні оптимізатора команда "nop" буде вилучена з програми у разі її опису вставкою **asm** ("nop"). У деяких публікаціях рекомендують вживати такий стиль завжди.

- Якщо необхідно використати вставку на декілька команд можливо застосувати попередній підхід, зробивши декілька вставок

```
asm ("nop");
asm ("sei");
```

або в одній вставці описати ці команди. У цьому випадку для покращення розбірливості тексту лістингу доцільно використовувати символи переносу строчки «\n» та табуляції «\t»

```
asm ("nop\n\t" "sei\n\t").
```

- Для передачі змінної в асемблерну вставку використовується формат використання змінної у вставці «%<name>». Сама змінна має бути визначена попередньо звичайним для мови Сі шляхом. При цьому для підвищення швидкості доступу до змінної бажано описувати тип змінної з модифікатором типу *"register"*

```
register unsigned char ddd=5;
unsigned char z=0;
asm("mov %z,%ddd");
```

Слід зауважити, що необхідно відстежувати використання команд, які оперують з безпосередніми константами. Компілятор не завжди коректно використовує регістри файлового регістру і у деяких випадках можуть генеруватися повідомлення про помилки, які пов'язані з неможливістю завантаження константи у регістри R0 – R15. Такі помилки можливо відслідковувати у файлі лістингу. Наприклад, помилковим є визначення асемблерних вставок у наступному прикладі

```
register unsigned char ddd, ccc, z=0;

asm("ldi %ddd,5\n");     //використовується R16 для
завантаження
asm("ldi %ccc,10\n");    //невірно використовується R10
asm("add %ddd,%ccc\n");
```

- Допускається використовувати у вставках символічні імена R0 – R31 регістрів файлового регістру

```
asm("ldi r22,5");
asm("ldi r23,10");
asm("add r22,r23");
```

- Основним недоліком асемблерних вставок є неможливість використання макросів препроцесора, які відносяться до області регістрів вводу/виводу. Тому помилковим є опис асемблерної вставки такого типу

asm volatile ("sbi PORTB, 0x07"); //встановити біт 0x07 на лініях порту PORTB.

У цьому випадку необхідно замість символічного імені регістру вводу\ виводу використовують його адресу. Адреси регістрів можна знайти в pdf-файлі опису мікроконтролера. В середовищі Image Craft, вставки такого типу мають наступний вигляд

```
register unsigned char z;
asm("in  %z,  $3F");//введення даних з регістру SREG у
зовнішню змінну z
asm("out $3F, %z");//вивід даних з зовнішньої змінної z у
регістр SREG.
```

Метод розробки програми мовою Сі з використанням окремих підпрограм розроблених мовою асемблера

Структура суміщеної програми написаною мовою Сі з використанням файлів розроблених мовою асемблеру, як і у випадку з асемблерними вставками, суттєво залежить від типу інтегрованого середовища розробки програми мовою Сі. Розглянемо використання такого підходу для середовища Image Craft.

При застосуванні такого стилю програмування звичайно необхідно вирішувати наступні задачі:

- в програмі написаною мовою Сі об’явити асемблерну функцію;
- забезпечити передачу та повернення змінних в асемблерну функцію;
- описати файл, до якого входить асемблерна функція, та підключити його до проекту;
- описати асемблерну функцію та забезпечити в ній прийом та повернення змінних.

Для забезпечення видимості асемблерної функції в Сі-програмі її необхідно об’явити традиційними методами. У зв’язку з тим, що сама функція розміщається в іншому файлі, необхідно в об’яві використати директиву “**extern** ”**.** Застосовується така функція у Сі програмі звичайним чином. Також немає ніяких відмінностей від традиційних методів передачі та повернення параметрів.

При розробці та підключенні асемблерного файлу є ряд суттєвих вимог. Асемблерний файл має розширення “*.s”. Цей файл необхідно розмістити в папці проекту. При описі властивостей проекту необхідно вказати шлях на папку проекту. На рис. 5.7 зображено приклад визначення шляху до вкладеного асемблерного файлу.

Також необхідно перенести в папку проекту файл опису мікроконтролера *"aiom16.s"*, який розроблено спеціально для випадку використання файлів розроблених мовою асемблеру. Цей файл знаходиться в тій же папці “include” компілятора Image Craft, що і файл опису мікроконтролера для мови Сі.

Формат файлу має наступні особливості:

- починається файл з директиви визначення області тексту «`.area text`»;

- потім з використанням директиви «`.include`» підключається файл опису мікроконтролера;
- модифікується ім’я асемблерної функції об’явленої в Сі файлі – на початку імені добавляється символ підкреслення. Після імені двічі ставиться символ «::»;
- для передачі та повернення змінних в асемблерну функцію використовується регістр «`r16`». Це визначається генеральними домовленостями для компілятора програми Image Craft. У зв’язку з цим формат початку опису асемблерної функції обов’язково включає команди отримання даних з регістру «`r16`». Так само завершення опису функції повинно мати команди передачі результату що повертається у регістр «`r16`».

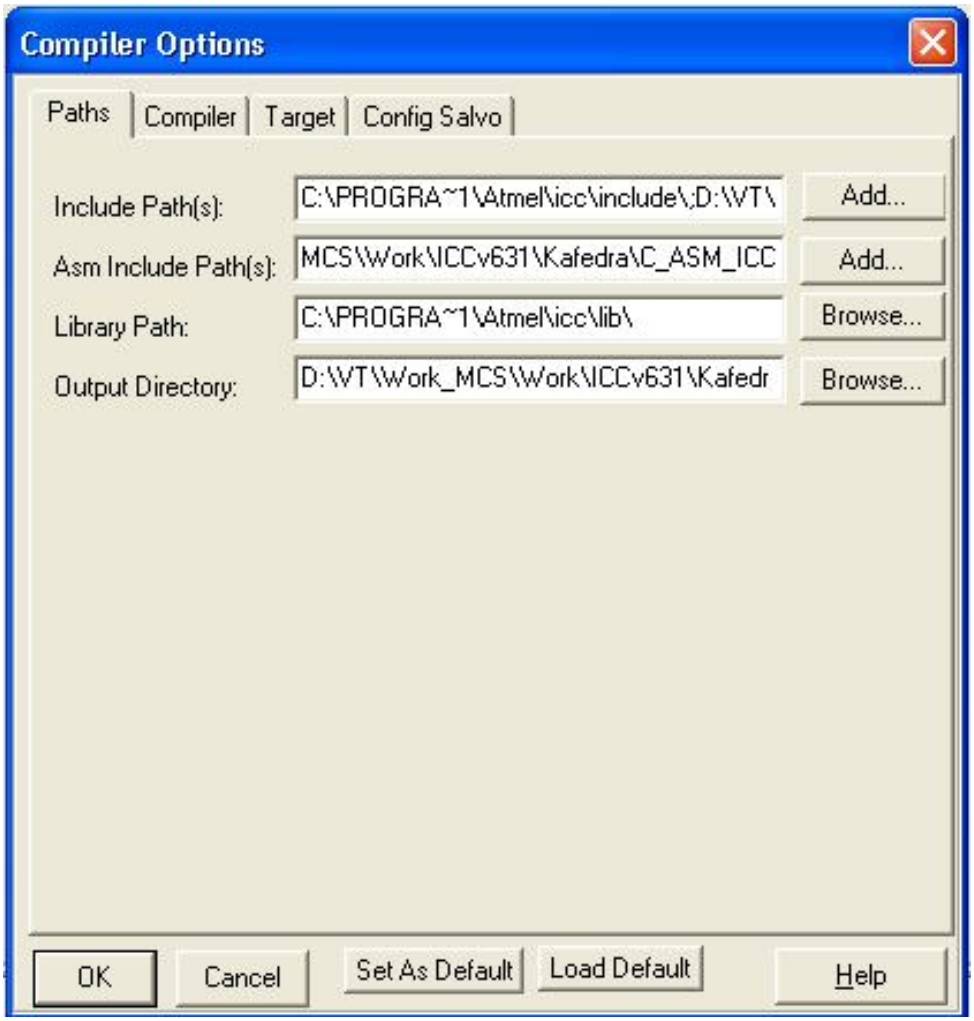


Рис. 5.7

Нижче приведено приклад комбінованого проекту, побудованого з використанням двох файлів, створених з використанням мови Сі та асемблеру.

***Приклад* 5.1.** Використовується асемблерна функція «aaa», яка отримує змінну в Сі програмі та повертає результат її обробки.

Файл створений мовою Сі має наступний вигляд

```
//-----------------------------------------------------------
-
//ICC-AVR application builder : 24.03.2016 11:40:24
// Target : M16
// Crystal: 10.000Mhz

#include <iom16v.h>
#include <macros.h>

extern unsigned char aaa (unsigned char var0);
//-----------------------------------------------
void main(void)
{
unsigned char var0, var1;

 var0=14;
 while (1)
 {
 var1 = aaa (var0);
 };
}
//-----------------------------------------------------------
-
```

Файл створений мовою асемблеру має наступний вигляд

```
;.....................................................
.area text

.include "aiom16.s"  ;see pathґs, same directory as *.h !!!
;.....................................................
_aaa::
mov R20,r16
;------ text ...
mov r16,r20
ret
;.....................................................
```

Приклади математичної обробки даних в МК

***Приклад* 5.2**. Операція додавання двох двобайтових чисел, розміщених у SRAM.

Реалізація підпрограми мовою асемблеру має наступний вигляд.

```
; [R19: R18] = [R17: R16] + [R19: R18] - сума розміщується на місці другого
; доданка R19: R18
add R18, R16 ; R18 <- R18 + R16
adc R19, R17; R19 <- R19 + R17 + C.
```

Реалізація цієї функції мовою Сі має наступний вигляд.

```
unsigned int add_16_16(unsigned int tmp1,unsigned int
tmp2)
{
unsigned int rezult;
result = tmp1+tmp2;
return rezult;
}
```

Функція повертає результат додавання двох змінних tmp1 та tmp2. Слід зауважити, що в обох варіантах результат операції не повинен перевищувати двобайтового числа.

***Приклад* 5.3**. Операція додавання багатобайтових чисел, розміщених у SRAM.

Реалізація підпрограми мовою асемблеру має наступний вигляд.

```
; [YH: YL] = [YH: YL] + [XH: XL]
; [YH: YL] - перший доданок і сума на виході (непряма адресація через YH: YL)
; [XH: XL] - другий доданок (непряма адресація побічно  через XH: XL)
; R16, R17, R18 - допоміжні регістри
; composed1 - адреса 1-го доданка і отриманої суми в ОЗП
; composed2 - адреса 2-го доданка в ОЗП
; SIZE - розмір доданків в байтах
; на виході в С знаходиться старший розряд результату
add_indirect:
ldi YH,high(composed1)   ;заносимо в покажчик Y адресу
ldi  YL,low(composed1)   ;першого доданка composed1
ldi XH,high(composed2)   ;заносимо в покажчик X адресу
        ldi  XL,low(composed2)    ;другого доданка composed2
        ldi  R16,SIZE
        clc;при першому вході в цикл C=0
```

```
ad1: ld    R17,X+
     ld R18,Y;по черзі складаємо з урахуванням перенесення
     adc R18,R17   ;всі   байти   доданок   і   заносимо
результат
     st  Y+,R18    ;за адресою першого доданка
     dec R16
     brne  ad1       ;повторюємо додавання SIZE разів
     ret
```

Реалізація цієї функції мовою Сі може мати вигляд, який приведено у першому прикладі. Проте у залежності від розрядності змінних та результату операції необхідно їх відповідно декларувати. Загальна форма декларації має наступний вигляд:

`[<storage class>] typename name,`

де: `storage class` – модифікатор доступу до змінної, `typename` – опис типу, `name` – ім'я змінної.

У табл. 5.3 наведено основні типи змінних, їх описи та діапазони можливих значень.

Таблиця 5.3

	Тип	**Варіанти**	**Опис**	**Діапазон значень**
Цілі	Char	беззнакові	unsigned char	0 – 255
		знакові	signed char	-128 - +127
	Short	беззнакові	unsigned short	0 – 65 535
		знакові	signed short	-32 768 - +32 767
	Int	беззнакові	unsigned int	0 - 65535
		знакові	signed int	-32768 - +32 767
	Long	беззнакові	unsigned long	0 – 4 294 967 295
		знакові	signed long	-2 147 483 648 - + 2 147 483 647
Плаваючі	Float	беззнакові	Float	$3.4*10^{-38}$ - $3.4*10^{38}$
	Double	беззнакові	Double	$1.7*10^{-308}$ - $1.7*10^{308}$
	long double	беззнакові	long double	$3.4*10^{-4932}$ - $3.4*10^{4932}$

Змінні можуть визначатися спільно з модифікаторами, що дозволяє надавати їм певні властивості. До переліку найбільш вживаних модифікаторів визначень змінних входять наступні ключові слова: const, volatile, static, register, extern.

Змінні можуть бути об’явлені як на зовнішньому так і внутрішньому рівнях. Якщо описи зроблено за межами усіх функцій, то об’ява вважається глобальною, або зовнішньою.

У разі визначення глобальної змінної використовують модифікатор extern. Така змінна може бути викликана з будь якої точки програми.

Визначення всередині будь якого вкладеного блоку вважається локальним, або внутрішнім. Час життя змінних, що визначаються на внутрішньому рівні розповсюджується на час виконання функції. Проте можливо модифікувати час життя такої змінної до рівня глобального, за рахунок використання модифікатору static.

У разі використання модифікатора const забороняється зміна значення змінної. Значення таких змінних (констант) записуються у пам'ять програм.

Модифікатор volatile має протилежну дію. Він вказує на те, що значення змінної може бути змінено будь якою функцією чи основною програмою.

Якщо змінна часто вживається, то для зменшення об’єму програми доцільно використовувати модифікатор register. У цьому разі компілятор резервує для змінної один з вільних файлових регістрів.

Нижче приведено варіант функції, для випадку реалізації без знакової операції додавання змінних однобайтової та двобайтової. Передбачається, що результат операції перевищує 2 байти.

```
      unsigned long add_16_16(unsigned char tmp1,unsigned int tmp2)
{unsigned long rezult;
      result = tmp1+tmp2;
      return rezult;
      }
```

Функція повертає результат додавання двох змінних tmp1 та tmp2. Слід зауважити, що в обох варіантах результат операції не повинен перевищувати двобайтового числа.

***Приклад* 5.4**. Операція додавання багатобайтових чисел, розміщених у SRAM. Реалізація підпрограми мовою асемблеру має наступний вигляд.

```
; [YH:YL] = [YH:YL] - [XH:XL]
; [YH:YL] – зменшуване при вході і різниця при
; виході (непряма адресація через YH: YL)
; [XH:XL] – від'ємник (непряма адресація через XH:XL)
; R16,R17,R18 – допоміжні регістри
; reduced - адреса зменшуваного і
; отриманої різниці в ОЗП
; subtracted - адреса від'ємника в ОЗП

; SIZE - розмір зменшуваного і від'ємника в байтах
; на виході біт C = 1 якщо [YH: YL] <[XH: XL]
sub_indirect:
     ldi YH, high(reduced)    ; Заносимо в покажчик Y адресу
     ldi  YL, low(reduced)     ; зменшуваного reduced
     ldi  XH, high(subtracted) ; Заносимо в покажчик X адресу
     ldi  XL, low(subtracted)  ; від'ємника  subtracted
     ldi  R16,SIZE
     clc      ; при першому вході в цикл C=0
sb1: ld    R17,X+
     ld    R18,Y      ; по черзі віднімаємо з урахуванням позики з
     sbc   R18,R17    ; зменшуваного всі байти від'ємника і
     st  Y+,R17     ;заносимо результат за адресою зменшуваного
     dec  R16
     brne  sb1        ; повторюємо віднімання SIZE разів
     ret
```

Для реалізація цієї функції мовою Сі необхідно використовувати рекомендації, які приведено у попередньому прикладі. У зв’язку з тим, що результат операції може бути від’ємним, необхідно використовувати знакові типи змінних. Нижче приведено

варіант функції, для випадку реалізації знакової операції віднімання 4-байтової та двобайтової змінних. Передбачається, що від'ємний результат операції може перевищувати 2 байти.

```
signed long sub_16_16(signed long tmp1,signed int tmp2)
{
unsigned int rezult;
result = tmp1-tmp2;
return rezult;
}
```

***Приклад* 5.5.** Операція множення двох двобайтових чисел, розміщених у SRAM.

Реалізація підпрограми мовою асемблеру має наступний вигляд.

Для множення 2-байтових чисел (позначимо їх як XH: XL = 2^8 * XH + XL і YH: YL = 28 * YH + YL) застосовується наступна обчислювальна схема:

XH:XL * YH:YL = (2^8*XH + XL)*(2^8*YH + YL) = 2^{16}*XH*YH + 2^8*(XH*YL + YH*XL) + XL*YL.

Визначення 32-розрядного результату зводиться до послідовності операцій однобайтових множень і подальшого складання всіх добутків з урахуванням зсуву доданків XH * YL, XL * YH на 1 байт і XH * YH на 2 байта. Підпрограма, що реалізує ці дії, наведена нижче. Нагадаємо, що добуток двох однобайтових множників, отриманий в результаті виконання команди mul Rd, Rr або будь-який інший команди множення, завжди заноситься в регістрову пару R1: R0.

```
; R23:R22:R21:R20 = R17:R16 * R19:R18
; R23:R22:R21:R20 – добуток
; R17: R16 - множене
; R19: R18 - множник
; R1, R0 - допоміжні регістри
mul16_16:
    mul   R16,R18  ;знаходимо XL*YL = R16*R18 і заносимо його в
    movw  R20,R0   ;молодші байти добутку R21:R20
    mul   R17,R19  ; знаходимо XH*YH = R17*R19 і заносимо його в
    movw  R22,R0   ;старші байти добутку R23:R22
    mul   R17,R18  ; знаходимо XH*YL = R17*R18 і додаємо його до
    clr  R17           ;байтів R23:R22:R21 добутку
    add   R21,R0
    adc   R22,R1
```

```
        adc   R23,R17
        mul   R16,R19   ; знаходимо YH*XL = R19*R16 і додаємо його
до  add   R21,R0; байтів    ;R23:R22:R21 добутку
        adc   R22,R1
        adc   R23,R17
        ret
```

Реалізація цієї функції мовою Сі має наступний вигляд.

```
unsigned long mul_16_16(unsigned int tmp1,unsigned int
tmp2)
{
unsigned int rezult;
result = tmp1*tmp2;
return rezult;
}
```

Приклад 5.6.

Операція ділення на 2 16-разрядного числа, розміщеного у SRAM.

Реалізація підпрограми мовою асемблеру має наступний вигляд.

Передбачається, що число розміщене у регістрах R17:R16:

```
lsr R17    ; R17 <- R17 >> 1 (MSB<- 0, прапор C <- LSB)
ror  R16  ; R16 <- R16 >> 1 (MSB <- C, прапор C <- LSB)
```

Реалізація цієї функції мовою Сі має наступний вигляд.

```
unsigned int mul_16_2 (unsigned int tmp1)
{
unsigned int rezult;
result = tmp1>>1;
return rezult;
}
```

Приклад 5.7. Загальний випадок ділення.

У загальному випадку найприродніший і простий спосіб розділити одне число на інше - це вирішити рівняння, що випливає безпосередньо з визначення операції ділення:

$$X = Z * Y + R,$$

$$R < Y,\ Y \neq 0,$$

де: X - ділене, Y - дільник, Z - частка, R - цілочисельний залишок від ділення.

Рівняння містить два невідомих параметра Z і R і тому не може бути явно вирішено. Для відшукання результату необхідно вдатися до ітераційного методу: віднімати з діленого дільник до тих пір, поки залишок не виявиться менше дільника. Тоді число циклів віднімання чисельно дасть частку Z, а залишок буде дорівнює цілочисельному залишку R від ділення. Приведений алгоритм має істотний недолік - швидкість його виконання безпосередньо залежить від величини частки (числа ітерацій віднімання), що робить небажаним його використання для ділення великих чисел. Застосовується він, в основному, в задачах обмежених ділення однобайтових величин. Підпрограма ділення:

```
; [R18] + {R17} = R17 / R16
; R18 – частка
; R17 – ділене при вході і цілочисельний залишок на виході
; R16 – дільник

 div8_8:
   clr  R18          ;скидаємо R18 при вході
M1:  sub  R17,R16   ;робимо віднімання R17-R16
   inc   R18
   brcc  M1          ;доти поки різниця R17-R16 > 0 (C Clear)
   dec   R18         ;коли різниця R17-R16 < 0
   add   R17,R16   ; відновлюємо R17 і коригуємо R18
   ret
```

Для чисел більшої розрядності необхідно використовувати спосіб ділення, заснований на наведеній нижче обчислювальної схемою. Для цього представимо необхідно вираз операції ділення в наступному вигляді:

$$R = X - Z * Y = X - \left(\sum_{i=0} Z_i \cdot 2^i \right) * Y = X - \sum_{i=0}^{n-1} Z_i * \left(Y * 2^i \right),$$

де: *X*- ділене, *Y*- дільник, *R* - цілочисельний залишок від ділення.

$$z = z_{n-1} \cdot 2^{n-1} + z_{n-2} \cdot 2^{n-2} + \ldots + z_1 \cdot 2^1 + z_0 \cdot 2^0 = \sum_{i=0}^{n-1} z_i \cdot 2^i$$ - частка.

Знаходження частого зводиться до відшукання коефіцієнтів z_i, які визначаються, згідно формулі, наступним чином: необхідно послідовно порівнювати X здобутками Y * 2^iі якщо X > Y * 2^i, то в цьому випадку необхідно провести віднімання Y * 2^i з X, а відповідний розряд zi встановити в 1; якщо X < Y * 2^i – віднімання пропускається і в цій ітерації $z_i = 0$. Ця процедура триває доти, поки не залишиться залишок R < Y. Добутки Y * 2^i отримують простим зсувом Y на *i* розрядів вліво. Аналогічно проводиться ділення в будь якій позиційній системі (метод ділення в стовпчик), але найпростіше в двійковій через те, що в X може міститися Y * 2^i максимум один раз (на що і вказує одиниця у відповідному розряді).

Розглянемо приклад ділення:

$$X = 0b10110011 = 179,$$
$$Y = 0b00010001 = 17.$$

10110011 - 10001	10001	перше порівняння 10110011 > 10001000 робимо віднімання і визначаємо $z_3 = 1$
00101011 - [10001]	1010	друге порівняння 101011 > 1000100 пропускаємо віднімання і визначаємо $z_2 = 0$
001010011 - 10001		третє порівняння 101011 > 100010 робимо віднімання і визначаємо $z_1 = 1$
00001001 - [10001]		третє порівняння 1001 > 10001 пропускаємо віднімання і визначаємо $z_0 = 0$
00001001		отримуємо залишок $R = 1001 \left(R < Y \right)$

$$z = z_3 \cdot 2^3 + z_2 \cdot 2^2 + z_1 \cdot 2^1 + z_0 \cdot 2^0 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 = 0b1010 = 10,$$
$$R = 0b1001 = 9.$$

Необхідно звернути увагу на те, що число ітерацій порівняння має збігатися з числом значущих розрядів частки (в даному випадку *n* = 4 для визначення z_0 ... z_3). Однак розрядність частки заздалегідь ніколи не відома і тому завжди необхідно знати

максимально можливу розрядність результату ділення. На практиці найчастіше використовують обчислення, в яких частка заздалегідь вміщається в 8, 16, 24 біта (кратно одному байту) і т.д. При цьому потрібно використовувати 8, 16, 24 ітерацій порівняння X з Y * 2^i, відповідно. З цілочисловим залишком R проблем не виникає - його розмір обмежений розрядністю Y (R < Y).

Підпрограма ділення двобайтового числа на однобайтове наведена нижче. У ній для порівняння X з Y * 2^i (i ∈ {0, ..., 15}) замість зсуву дільника Y на n розрядів вправо використовується почерговим зсув діленого на n розрядів вліво, а для економії пам'яті частка записується на теж місце, що і ділене . На початку програми здійснюється перевірка умови Y ≠ 0, без якого ділення не може бути коректно здійснено.

```
; [R17: R16] + {R18} = R17: R16 / R20
; R17: R16 - ділене при вході і частка на виході; ; R20 – дільник; ; R18 - цілочисельний залишок; ; R21, R22 - допоміжні регістри
; при виході з підпрограми в С знаходиться ознака помилки
; якщо С = 1 - сталася помилка (R20 = 0)
; якщо С = 0 - ділення успішно виконано
div16_8:
    tst R20          ; якщо R20 = 0 виходимо з підпрограми
    breq dv3         ; з ознакою помилки С = 1
    clr R18          ; очищаємо регістри R18, R19, R21 при
    clr R19          ; вході в підпрограму
    clr R21
    ldi R22,16       ; ініціалізіруємо лічильник циклів
dv1: lsl R16         ; зсув діленого R17: R16 зі
    rol R17          ; допоміжними регістрами
    rol R18          ; R19, R18 на один розряд вліво
    rol R19
    sub R18, R20     ; здійснюємо пробне віднімання
    sbc R19, R21     ; R19: R18 - R20 і якщо R19: R18> R20,
    ori R16,0x01     ; то встановлюємо z_i = 1
    brcc dv2
    add R18, R20     ; в іншому випадку відновлюємо ділене
    adc R19, R21
    andi R16,0xFE;   і встановлюємо z_i = 0
dv2: dec R22
```

```
    brne dv1          ; повторюємо цикл n = 16 раз
    clc               ; успішно завершуємо підпрограму
    ret               ; зі прапором C = 0
dv3: sec              ; виходимо через помилку
    ret               ; з прапором C = 1
```

Реалізація цієї функції мовою Сі має наступний вигляд.

```
 unsigned int div_16_8 (unsigned int tmp1, unsigned char
tmp2)
 {
 unsigned int rezult;
 result = tmp1/tmp2;
 return rezult;
 }
```

***Приклад* 5.8**. Пошук максимального елементу масиву.

Знайти найбільше (найменше) значення серед масиву чисел можна по наступним чином: треба взяти довільний елемент послідовності (скажімо найперший) і по черзі порівняти його з усіма іншими. На самому початку перший елемент вважається найбільшим числом, але якщо під час порівняння знаходиться елемент з більшим числовим значенням, то тоді вже йому присвоюється статус найбільшого і т.д. Підпрограма для знаходження максимального значення наведена нижче. Заміна в ній умови переходу brcc на brcs, дозволить отримати на виході найменший елемент списку.

```
; Підпрограма знаходження найбільшого елементу масиву чисел
; YH: YL - покажчик на елемент масиву чисел
; R16 - регістр для проміжних операцій
; R17 - найбільше елемент масиву на виході
; R18 - лічильник ітерацій порівняння
; array - адреса початку масиву чисел
; ASIZE - розмір масиву (2 ... 256)
max_item:
    ldi R18, ASIZE-1  ; ініціалізуємо лічильник ітерацій
    ldi YH, high (array)      ; заносимо в покажчик адресу початку
    ldi YL, low (array)       ; масиву чисел
    ld R16, Y +               ; в R16 заносимо перший елемент масиву
```

```
mx1: ld R17, Y +          ; в R17 заносимо і елемент масиву
   cp R16, R17            ; порівнюємо поточний найбільший елемент
   brcc mx2          ; з і елементом масиву і якщо він
   mov R16, R17      ; виявляється менше, то міняємо їх місцями
mx2: dec R18         ; декрементуємо лічильник R18 і якщо R18 = 0,
   brne mx1
   ret               ; то цикл порівнянь закінчено.
```

Реалізація цієї функції мовою Сі має наступний вигляд.

```
unsigned char mas[] = {1,2,3,3,5,2,100,3,10,2};//визначення
масиву
     unsigned char Nmax=22; //визначення довжини масиву
     unsigned char find_max(void)
     {
     unsigned char max_rezult=0; //визначення імені та типу
змінної з результатом
     unsigned char i=0; //визначення індексу елементів масиву
     for (i=0; Nmax; i++)
          {if (mas[i] > max_rezult) max_rezult = mas[i];};
     return max_rezult;
     }
```

Контрольні питання

1. Особливі режими роботи мікропроцесорів.
2. Режим холостого ходу - IDLE;
3. Режим мікроспоживання – PowerDown;
4. Економічний режим – PowerSave;
5. Режим придушення шуму - ADC Noise Reduction;
6. Основний режим очікування –Standby;
7. Додатковий режим очікування - Extended Standby.
8. Способи адресації операндів AVR- мікроконтролерів.
9. Безпосередня регістрова адресація та її структурна реалізація.
10. Мовні засоби, шо використовуються при розробці програмного забезпечення AVR мікроконтролерів.

11. Система команд мікропроцесора. Загальні відомості. Наведіть приклади команд.

12. Методи адресації. Наведіть приклади команд.

13. Команди передачі даних. Наведіть приклади команд.

14. Команди арифметичних операцій. Наведіть приклади команд.

15. Команди логічних операцій. Наведіть приклади команд.

16. Команди операцій з бітами. Наведіть приклади команд.

17. Команди розгалуження та передачі керування. Наведіть приклади команд.

18. Команди умовних переходів. Наведіть приклади команд.

19. Наведіть приклади команд, після яких встановлюється прапор перенесення.

РОЗДІЛ 6. ОРГАНІЗАЦІЯ ВЗАЄМОДІЇ МІКРОПРОЦЕСОРА З ОБ`ЄКТОМ КЕРУВАННЯ ТА ОПЕРАТОРОМ

6.1 Огляд задач мікроконтролерної системи

Типова схема мікропроцесорної системи керування напівпровідниковим перетворювачем наведена на рис. 6.1, де МК – мікроконтролер, ПС – пристрій синхронізації, ККД – контролер клавіатури та дисплею, УП – узгоджуючі пристрої (драйвери).

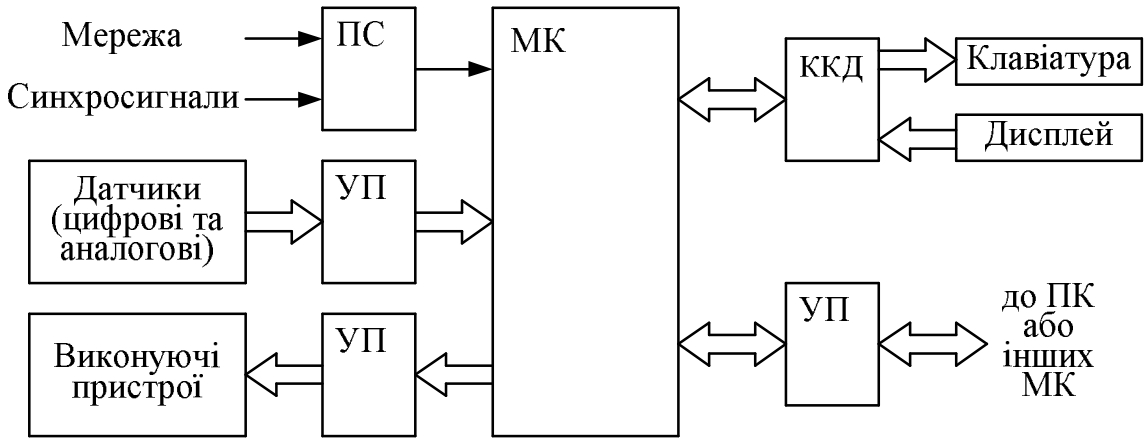


Рис. 6.1

Схема відображає приклад організації взаємодії МК з об`єктом керування, оператором, іншими мікроконтролерними системами або персональним комп’ютером. При цьому вирішуються наступні задачі:

- введення/виведення статичних та імпульсних цифрових сигналів;
- введення/виведення аналогових сигналів;
- місцеве керування оператором за допомогою клавіатури та дисплею;
- віддалений зв'язок з оператором за допомогою інших міжконтролерних пристроїв або персонального комп’ютера.

Порти вводу/виводу

Регістри портів. Звертання до портів відбувається через регістри вводу/виводу. В адресному просторі вводу/виводу для кожного порту відведено по 3 регістри:

- DDRx - напрямку роботи (передавач/приймач);
- PORTx – вихідних даних;
- PINx – вхідних даних.

Наприклад, для порту A - DDRA, PORTA, PINA.

Розряди цих регістрів мають назви:

- DDx7… DDx0 – для регістрів DDRx;
- Px7…Px0 – для регістрів PORTx;
- PINx7… PINx0 – для регістрів PINx.

Регістри PINx дозволяють здійснити доступ до фізичних значень сигналів на виводах порту. Відповідно вони доступні лише для читання. Регістри PORTx та DDRx доступні як для читання, так і для запису. Після рестарту мікроконтролера в регістри PORTx та DDRx записуються початкові нулеві значення. Це відповідає режиму приймача.

Запис у порт означає запис необхідного стану для кожного виводу порту у відповідний регістр даних порту PORTx. А читання стану порту виконується читанням або регістра даних порту PORTx, або регістра виводів порту PINx. При читанні регістра виводів порту PINx відбувається зчитування логічних рівнів сигналів, що присутні на виводах порту. А при читанні регістра даних порту відбувається зчитування даних, що знаходяться у регістрі PORTx.

Приклад ініціалізації порту

Початкова ініціалізація портів є надзвичайно важливою процедурою. Ініціалізація повинна проводитися одразу після рестарту мікроконтролера. Затримка з її проведенням у деяких випадках може привести до

виникнення аварійної ситуації у об’єкті керування. Визначення типів ліній портів та їх початкового стану проводиться на підставі аналізу схеми пристрою. Нижче приведено приклад ініціалізації порту А з конфігурацією ліній, що наведено у табл. 6.1. У відповідності до цієї таблиці визначаються константи ініціалізації ini_DDRA та ini_PORTA.

Таблиця 6.1

Лінії порту	**PA7**	**PA6**	**PA5**	**PA4**	**PA3**	**PA2**	**PA1**	**PA0**
Передавач	так	так	так	так	ні	ні	ні	ні
Вихідний рівень	1	1	0	0	-	-	-	-
Приймач	ні	ні	ні	ні	так	так	так	так
Підтяжка	ні	ні	ні	ні	ні	ні	так	так

У функції ініціалізації портів доцільно використовувати символічні імена констант. Це суттєво спрощує модифікацію програми при її адаптації до розробленої печатної плати. Для символічного опису використовують директиву *#define ім'я значення.* Під час опису констант ініціалізації портів доцільно вживати бінарний формат чисел що дає змогу більш оперативно оцінювати бітовий стан порту.

***Приклад* 6.1.**

```
//ICC-AVR application builder : 13.05.2016 22:28:05
//Target : M16
//Crystal: 10.000Mhz

#include <iom16v.h>
#include <macros.h>
//----------------------------
#define    ini_DDRA    0b11110000 //константи ініціалізації
#define    ini_PORTA   0b11000011 // константи ініціалізації
//---------- Функція ініціалізації портів
void port_init(void)//ініціалізація порту А
{
DDRA  = ini_DDRA;
PORTA = ini_PORTA;
}
//---------- Функція ініціалізації мікроконтролера
void init_devices(void)
{
CLI();                    //заборона переривань
```

```
port_init();              //ініціалізація портів
SEI();                    //дозвіл переривань
}
//---------- Головна функція програми
void main(void)
{
init_devices();
while(1)
      {
      }
}
//----------------------------
```

6.2. Виведення цифрових сигналів

Виведення статичних цифрових сигналів

Для виводу цифрових сигналів необхідно налаштувати відповідні лінії портів на режим передавача. Операції виводу проводять із регістрами PORTx (PORTA, PORTB, PORTC, PORTD).

Для встановлення лінії портів в “1”, або скидання в “0” - стан використовуються наступні команди:

- установка «1» PORTx |=(1 << bit);
- скидання в «0» PORTx &=~(1 < < bit);
- інверсія PORTx =(1 << bit),

де: “PORTx” – вихідний порт, “bit” – біт призначення.

Якщо весь порт використовується в режимі передавача, і одночасно змінюється велика кількість біт, то доцільно використовувати команди завантаження байтів

PORTx = const_1,

де: константа “const_1” попередньо визначається наступним чином

```
#define    const_1  0b11110000
```

Не допускається виконання логічних і арифметичних операцій безпосередньо на регістрах PORTx. Такі операції можливі лише з файловими регістрами.

Вивід імпульсних сигналів

Розрізняють наступні типи імпульсних сигналів:

- одиночні сигнали;
- одиночна послідовність ряду імпульсів (пакет імпульсів);
- періодичні сигнали із установленими періодом і тривалістю імпульсів;
- періодична послідовність пакетів імпульсів.

Для формування імпульсних сигналів можливе використання, як програмних засобів, так і можливостей таймерів-лічильників мікроконтролера.

Вивід імпульсних сигналів із використанням програмних засобів

Такий тип виводу використовується у випадках, коли відсутні істотні обмеження по часу виконання процедури, або при необхідності створення драйверів зовнішніх мікросхем із жорсткими вимогами до тривалості та періоду повторення імпульсів керування.

Загальний алгоритм виводу передбачає реалізацію виводу на порт певного логічного стану та необхідної програмної затримки до моменту виводу наступного стану.

У другому прикладі забезпечується формування на лінії порту LED1 періодичної послідовності двох негативних імпульсів з наступними параметрами:

- тривалість першого та другого імпульсів – 1с;
- затримка між першим і другим імпульсами – 500мс;
- період повторення пачки імпульсів – 4с.

***Приклад* 6.2**

```
//---------------------------
//ICC-AVR application builder : 20.05.2016 23:02:36
//Target : M16
//Crystal: 10.000Mhz
```

```
#include <iom16v.h>
#include <macros.h>
//---------- Символічні визначення ліній портів та макроси
#define    LED1    PA6    //лінія керування світлодіодом LED1
#define LED1_ON PORTA &=~(1<<LED1) //включення LED1
#define LED1_OFF PORTA |= (1<<LED1)//вимикання LED1
//---------- Функція ініціалізації портів
void port_init(void)
{
DDRA  = 0b11001111;      //визначення типу ліній портів
PORTA = 0b11110000;      //визначення початкового стану
}
//---------- Функція ініціалізації мікроконтролера
void init_devices(void)
{
CLI();              //заборона переривань
port_init();        //ініціалізація портів
SEI();              //дозвіл переривань
}
//--------- Функція програмної затримки на "t_mcs" мікросекунд
void DelayMcs (unsigned int t_mcs)
     {while(t_mcs--)
          {asm("nop"); asm("nop"); asm("nop"); asm("nop"); };}
//--------- Функція програмної затримки на "t_ms" мілісекунд
void DelayMs (unsigned int t_ms)
     {while(t_ms--) {DelayMcs(1000);};}
//--------- Функція програмної затримки на "t_s" секунд
void DelayS (unsigned int t_s)
     {while(t_s--) {DelayMs(1000);};}
//---------- Головна функція програми
void main(void)
{
init_devices();     //ініціалізація мікроконтролера
while(1)
     {
     LED1_ON;       //включення світлодіоду LED1
     DelayS(1);     //затримка 1 сек
     LED1_OFF;      //вимикання світлодіоду LED1
     DelayMs(500);  //Затримка 0.5 сек
     LED1_ON;       //включення світлодіоду LED1
     DelayS(1);     //Затримка 1 сек
     LED1_OFF;      //вимикання світлодіоду LED1
     DelayS(4);     //затримка 4 сек
     }
}
//---------------------------
```

Вивід імпульсних сигналів з використанням таймерів-лічильників

У разі жорстких вимог до використання часу виконання основної програми мікроконтролера для формування імпульсних сигналів застосовують таймери-рахівники.

Звичайно за допомогою таймерів вирішують наступні типи задач:

- формування тривалості затримки;
- формування імпульсів з заданою тривалістю;
- формування імпульсів з заданими тривалістю та періодом повторення.

Загальний алгоритм вирішення перших двох задач передбачає ініціалізацію обраного таймеру на реалізацію часу затримки, налаштування системи переривань та запуск таймеру.

Відмінності починаються у підпрограмі переривань. У першому випадку у підпрограмі переривань вимикається таймер. У другому випадку – виконуються також процедури маніпуляції з лініями портів, де формуються сигнали.

При вирішенні третього типу задач використовують PWM режим роботи. За рахунок ініціалізації таймера на певних портах мікроконтролера можливе безперервне формування PWM - сигналу.

Розглянемо два приклади реалізації задач другого та третього типів.

***Приклад* 6.3.** У наведеному прикладі забезпечується формування на лінії порту LED2 негативного імпульсу тривалістю 2с. Необхідна тривалість затримки забезпечується вибором коефіцієнту «попереднього подільника» 1/1024 таймера TC1 і завантаженням у робочі регістри TCNT1 відповідного початкового значення. Використано режим “Normal” таймера/рахівника.

```
//---------------------------
//ICC-AVR application builder : 14.05.2016 1:47:18
//Target : M16
//Crystal: 10.000Mhz
```

```
#include <iom16v.h>
#include <macros.h>
//---------- Символічні визначення ліній портів та константи
#define      LED2    PA7 //лінія керування світлодіодом LED2
#define      TC1_ON_1024 0b00000101     //включення TC1-1/1024
#define      TC1_OFF 0   //виключення TC1
//---------- Вектор переривання переповнення TC1
#pragma interrupt_handler  INT_TC1_OVF: iv_TIMER1_OVF
//---------- Функція ініціалізації портів
void port_init(void)
{
DDRA  = 0b11001111;      //визначення типу ліній портів
PORTA = 0b11110000;      //визначення початкового стану
}
//---------- Функція ініціалізації TC1
void init_timer(void)
{
TIMSK  = 0x04; //дозвіл переривань переповнення TC1
TCNT1H = 0xb3; //завантаження TC1 на затримку 2 секунди
TCNT1L = 0xb4;
}
//---------- Функція ініціалізації мікроконтролера
void init_devices(void)
{
CLI();                   //загальна заборона переривань
port_init();             //ініціалізація портів
init_timer();            //ініціалізація таймера TC1
SEI();                   //загальний дозвіл переривань
}
//---------- Функція переривань переповнення TC1
void INT_TC1_OVF(void)
{
PORTA |= (1<<LED2);      //вимикання світлодіода LED2
TCCR1B = TC1_OFF;        //відключення таймера TC1
}
//---------- Головна функція програми
void main(void)
{
init_devices();

PORTA &=~(1<<LED2);      //включення світлодіода LED2
TCCR1B = TC1_ON_1024;    //включення TC1 з подільником 1/1024
while(1){};
}
//---------------------------
```

***Приклад* 6.4.** У прикладі забезпечується формування на лінії порту PWM (PD4 – OC1B) періодичної послідовності позитивних імпульсів з алгоритмом утворення “Fast PWM”.

Використана секція “B” таймера TC1, попередній подільник 1/256. Реалізовано режим “Fast PWM”, 8 розрядів, порт встановлюється в «1» на інтервалі формування імпульсу.

Форму сигналів можна перевірити за допомогою осцилографа (клема PWM стенду).

Параметри імпульсів:

- частота імпульсної послідовності – 152,6 Гц. Частота визначається наступним чином

$$F_{OC} = f_{CLK} / (N * TOP),$$

де f_{CLK} – частота генератора мікроконтролера; N – коефіцієнт ділення попереднього подільника; TOP – модуль рахування. Для 8-бітової “Fast PWM” величина TOP складає 256;

- тривалість – 3,25мс;
- розрядність (точність) PWM – 8 біт.

```
//----------------------------
//ICC-AVR application builder : 14.05.2016 2:26:40
// Target : M16
// Crystal: 10.000Mhz
#include <iom16v.h>
#include <macros.h>
//----------
#define   ini_TCNT1      0    //період T
//режим   PWM 8bit fast
#define   ini_TCCR1A     (1<<COM1B1)|(1<<COM1B0)|(1<<WGM10)
#define   ini_TCCR1B     (1<<CS12)|(1<<WGM12)
#define   ini_OCR1BL     (0x00ff-127) //тривалість імпульсу T/2
//---------- Функція ініціалізації портів
void port_init(void)
{
DDRD  = 0b00010000; // лінія порту PWM (PD4 – OC1B) - передавач
PORTD = 0x00;
}
//---------- Функція ініціалізації TC1
//TIMER1 initialize - prescale:256
//WGM: 5) PWM 8bit fast, TOP=0x00FF
//desired value: 153.8462Hz
//actual value: 152,588Hz (0,8%)
//Константи ініціалізації

void timer1_init(void)
{
TCCR1B = 0x00;           //зупинка таймера
TCNT1  = ini_TCNT1;      //завантаження періоду
```

```
OCR1BL = ini_OCR1BL;     //завантаження тривалості імпульсів
TCCR1A = ini_TCCR1A;     //завантаження режиму роботи
TCCR1B = ini_TCCR1B;     //старт таймера
}
//---------- Функція ініціалізації мікроконтролера
void init_devices(void)
{
CLI();                   //загальна заборона переривань
port_init();             //ініціалізація портів
timer1_init();           //ініціалізація таймера TC1
SEI();                   //загальний дозвіл переривань
}
//---------- Головна функція програми
void main(void)
{
init_devices();          //ініціалізація мікроконтролера
while(1){};
}
//----------------------------
```

6.3. Введення цифрових сигналів

Для вводу цифрових сигналів необхідно налаштувати відповідні лінії портів на режим приймача. Для вводу використовують регістри PINx (PINA – PIND).

Побудова алгоритмів вводу залежить від типу цифрових сигналів (статичні, імпульсні), а також від присутності флуктуацій форми сигналів. Розрізняють алгоритми вводу сигналів стабільної форми, наприклад конфігураційного поля джемперів, а також сигналів із флуктуаціями. Характерним прикладом в останньому випадку може бути ввід сигналу від контактної пари, з програмною фільтрацією тремтіння контактів, де логічний рівень у процесі перемикання може декілька разів змінюватися.

При вводі імпульсних сигналів звичайно вирішують наступні задачі:

- реєстрації сигналів;
- реєстрації кількості імпульсних сигналів;
- вимірювання періоду або тривалості періодичних сигналів.

Введення статичних цифрових сигналів

В AVR- мікроконтролерів відсутні команди бітового обміну між файловими регістрами й регістрами введення/виводу. Для читання портів використовують команди завантаження байтів

$$R_d = PINx;$$

де: "R_d" - один з файлових регістрів, або змінна.

Не допускається виконання логічних і арифметичних операцій безпосередньо на регістрах PINx. Такі операції можливі лише з файловими регістрами чи змінними.

Разом з тим існує можливість реалізації програмних розгалужень на підставі аналізу стану окремих ліній портів PINx, за рахунок використання команд if(PINx & (1<<bit)){}, switch(){}.

Введення цифрових сигналів, що формуються контактною парою

Для вводу сигналів, що формуються контактною парою, часто використовують наступний алгоритм:

- проводиться первинне опитування стану порту. У разі виявлення замкненого стану контактів формується затримка, час якої перевищує час тремтіння контактів;
- проводиться вторинне опитування стану порту на підставі якого приймається рішення про стан контактної пари.

Нижче приведено приклад програми, де вводиться цифровий сигнал від кнопки S3 (порт PB4) з тривалістю тремтіння контактів < 2 мс. Замкненому стану контактів відповідає "0" , а розімкненому "1" рівень на лінії порту MODE (PB4). У програмі не використовується система переривань.

У прикладі застосована функція затримки «DelayMs (**unsigned int** t_ms)», що описана у другому прикладі. В якості реакції на натискання кнопки використане перемикання світлодіодів LED1, LED2.

***Приклад* 6.5**

```
//----------------------------
//ICC-AVR application builder : 13.05.2016 22:28:05
//Target : M16
//Crystal: 10.000Mhz

#include <iom16v.h>
#include <macros.h>
//---------- Символічні визначення ліній портів
#define    LED1    6     //лінія керування світлодіодом LED1
#define    LED2    7     //лінія керування світлодіодом LED2
#define    S3      PB4   //лінія вводу стану кнопки S1
//---------- Функція ініціалізації портів
void port_init(void)
{
DDRA  = 0b11001111;      //визначення типу ліній порту A
PORTA = 0b11110000;      //визначення початкового стану
DDRB  = 0b00000000;      //визначення типу ліній порту B
PORTB = 0b00000000;      //визначення початкового стану
}
//---------- Функція ініціалізації мікроконтролера
void init_devices(void)
{
CLI();                   //загальна заборона переривань
port_init();             //ініціалізація портів
SEI();                   //загальний дозвіл переривань
}
//--------- Функція програмної затримки на "t_mcs" мікросекунд
void DelayMcs (unsigned int t_mcs)
     {while(t_mcs--)
               {asm("nop"); asm("nop"); asm("nop"); asm("nop");
              };}
//---------- Головна функція програми
void main(void)
{
init_devices();              //ініціалізація мікроконтролера
while(1)
     {
     if (!(PINB & (1<<S3)))   // опитування стану кнопки S1
        {
        DelayMcs(4000);       //затримка 4мс
        if  (!(PINB & (1<<S3)))    //повторне опитування S1
            {
            PORTA |=(1<<LED1);     //вимикання LED1
            PORTA &=~(1<<LED2);    //включення LED2
            }
        else
            {
            PORTA |=(1<<LED2);     //вимикання LED2
```

```
            PORTA &=~(1<<LED1);    // включення LED1
            };
        };
    };
}
//---------------------------
```

6.4. Вимірювання тривалості імпульсів

Задача вимірювання тривалості імпульсних цифрових сигналів, або періоду їх повторення вирішується звичайно з використанням таймерів та системи переривань. Переривання використовують для реєстрації початку та закінчення імпульсу.

Використовується наступний алгоритм:

- після реєстрації імпульсу вмикається таймер;
- після завершення імпульсу таймер зупиняється.

Тривалість імпульсу визначається по відомій величині тривалості одного циклу, що реєструє таймер, та по зареєстрованій таймером кількості таких циклів.

Недоліком таймерів AVR - мікроконтролерів є відсутність ресурсів для зовнішнього керування операціями вмиканням/вимиканням. Можливі лише програмні реалізації, або використання інших контролерів, наприклад, вбудованого компаратора. Це обмежує точність фіксації моментів початку та закінчення імпульсів. Для підвищення точності фіксації доцільно використовувати переривання по входу INT0, що мають найвищий внутрішній пріоритет, та, відповідно, швидкість реакції. Решту переривань мікроконтролера на час вимірювань необхідно заборонити.

Для фіксації початку імпульсу потрібно налаштувати переривання по спадаючому (наростаючому) фронту INT0 – входу. Після включення таймера необхідно змінити настройку типу фронту на протилежну – падаючий (наростаючий).

У шостому прикладі розглянуто програму вимірювання тривалості імпульсу. Сигнал подається на вхід INT0 (PD2). Початок імпульсу

фіксується по спадаючому, а закінчення – по наростаючому фронту. Використовуються переривання по входу INT0. Перевіряється тривалість імпульсу. Якщо тривалість імпульсу перевищує 5120мкс –світлодіод вимикається. У цьому прикладі використано те, що на вхід INT0 поступають імпульси які формуються з напруги мережі живлення частотою 50Гц.

***Приклад* 6.6**

```
//----------------------------
//ICC-AVR application builder : 14.05.2016 2:43:20
//Target : M16
//Crystal: 10.000Mhz

#include <iom16v.h>
#include <macros.h>
//---------- Символічні визначення констант та змінних
#define TC1_ON  0b00000101    //попередній подільник 1/1024
#define TC1_OFF 0             //виключення таймера
#define ini_INT0    (1<<INT0) //дозвіл переривань по INT0
#define ini_INT0_10 (1<<ISC01)//переривання по фронту "1-0"
#define ini_INT0_01 (1<<ISC00)|(1<<ISC01)  //по фронту "0-1"
#define LED2   PA7            //визначення лінії LED2

volatile unsigned int Result=0;//змінна результату вимірювань
//----------------------------
#pragma interrupt_handler int0_isr:2 //Вектор переривання INT0
//---------- Функція ініціалізації портів
void port_init(void)
{
DDRA  = 0b11000000; //визначення приймачів - передавачів
PORTA = 0b11000000; //початковий стан ліній порту
DDRD  = 0x00;       //увесь порт - приймачі
PORTD = 0x00;       //відсутні резистори підтяжки
}
//---------- Функція ініціалізації TC1
//Режим Normal, TOP=0xffff
void timer1_init(void)
{
TCCR1B = TC1_OFF;   //зупинка таймера
TCNT1  = 0;         //очищення рахункових регістрів
}
//---------- Функція переривань INT0
void int0_isr(void)
{
if  (MCUCR == ini_INT0_10)    //розгалуження початок-кінець
    {
    TCCR1B = TC1_ON;          //включення таймера TC1
```

```
    MCUCR = ini_INT0_01;      //переривання по фронту 0-1
    }
else
    {
    TCCR1B = TC1_OFF;         //відключення таймера TC1
 Result = TCNT1;              //читання результату вимірювань
 MCUCR = ini_INT0_10;         //переривання по фронту 1-0
    TCNT1 = 0;                //очищення рахункових регістрів
    }
}
//---------- Функція ініціалізації мікроконтролера
void init_devices(void)
{
CLI();                    //загальна заборона переривань
port_init();              //ініціалізація портів
timer1_init();            //ініціалізація таймера
GICR  = ini_INT0;         //дозвіл переривань по входу INT0
MCUCR = ini_INT0_10;      //переривання по фронту "1-0"INT0
SEI();                    //загальний дозвіл переривань
}
//---------- Головна функція програми
void main(void)
{
init_devices();           //ініціалізація мікроконтролера
while(1)
   {
   if  (Result > 50) //перевірка результатів вимірювань
       {PORTA |=(1<<LED2);}   //виключити LED2, якщо тривалість
//більше 50*1024*0,1=5120мкс
   else
       {PORTA &=~(1<<LED2);}; //включити LED2, якщо менше
   };
}
//----------------------------
```

6.5. Побудова дисплеїв мікроконтролерних систем на світлодіодних індикаторах

Класифікація індикаторів

Залежно від типу фізичних процесів, що використовуються, розрізняють наступні види індикаторів - лампи накалювання; вакуумні, люмінісцентні, світлодіодні (LED), рідкокристалічні (LCD), плазмові панелі, активні та пасивні матриці.

У мікроконтролерних системах керування та обробки інформації найбільш розповсюдженими є LED та LCD індикатори.

Огляд індикаторів LED - типу

Такі індикатори використовують при побудові дисплеїв, що працюють у широкому діапазоні температур (-50 - +100°С).

Індикатори характеризуються високою надійністю, яскравістю (у окремих зразків фірми Luxeon Star до 200 Кд), малою вартістю. Фірми Kingbright та Paralight виробляють індикатори з сімома основними кольорами випромінювання.

У LED-індикаторів звичайно є лише елементи індикації, а інтерфейсні вузли відсутні.

Виробляються індикатори із такими конструкціями:

- одиночні індикатори циліндричної форми із діаметрами 1, 3, 5, 7, 10мм та кластери (рис. 6.1);

Рис. 6.1.

- мнемонічні індикатори (прямокутної, трикутної, символьної форми) (рис. 6.2.)

Рис. 6.2.

- світлові табло прямокутної та круглої формим(рис. 6.3.)

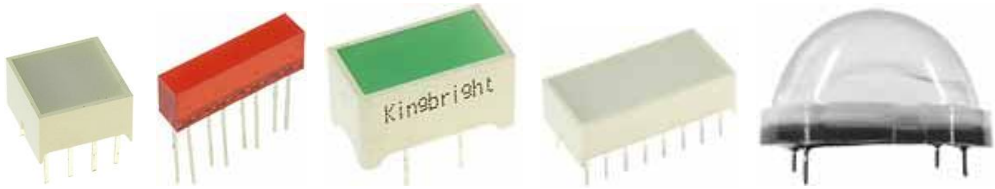


Рис. 6.3

- графічні лінійки (рис. 6.4);

Рис. 6.4.

- одиночні семисегментні індикатори та їх набори (2-10 десяткових розрядів) (рис. 6.5);
-

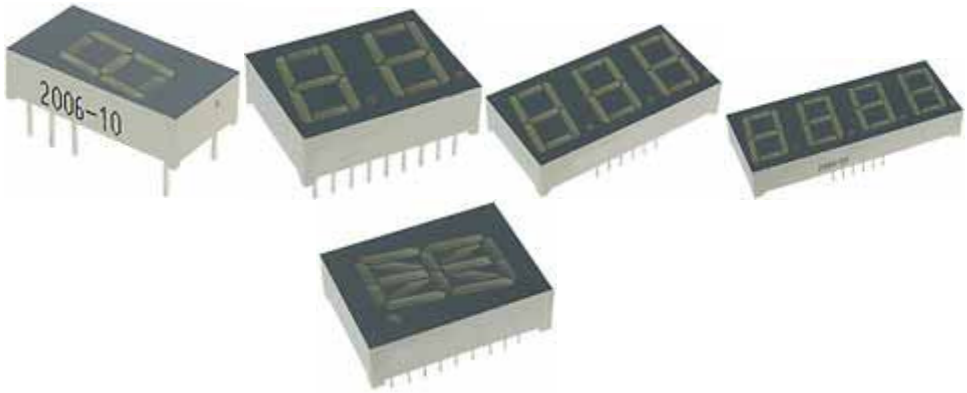


Рис. 6.5

- мозаїчні індикатори(рис. 6.6).

Рис. 6.6.

Випромінювання LED-індикатора викликається протіканням прямого струму. Величина падіння напруги на прямо зміщеному світлодіоді залежить від величини струму та використаної технології, що визначає колір випромінювання.

Одиночні індикатори звичайно мають один світлодіод. В індикаторах з великою площею поверхні (табло, семисегментні індикатори великого розміру) можуть використовуватися декілька послідовно ввімкнених світлодіодів.

У дисплеях для відтворення цифр використовують семисегментні індикатори. У залежності від електричної схеми розрізняють два типи таких індикаторів – із спільним анодом (рис. 6.7, а) та із спільним катодом (рис. 6.7, б).

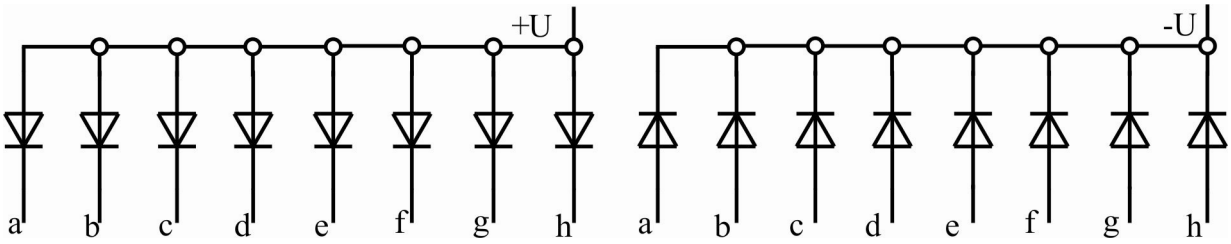
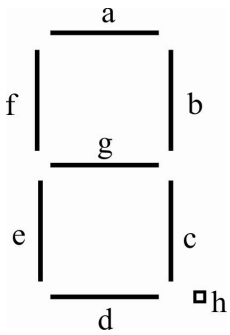


Рис. 6.7.

На вибір типу індикатора впливають особливості принципової схеми пристрою індикації та допустимі рівні вихідного струму (високого та низького рівня) буферних підсилювачів. Сегменти індикатору позначаються літерами “a – h”. Звичайно використовується варіант розміщення сегментів, що приведений на рис. 6.8.



Для побудови багаторозрядних дисплеїв можуть застосовуватися одиночні семисегментні індикатори, або їх набори. У залежності від електричної схеми розрізняють набори призначені для статичної та динамічної індикації.

Набори семисегментних індикаторів статичної індикації мають для кожного розряду числа повний комплект ліній керування (рис. 6.7).

У наборах, що використовуються у схемах динамічної індикації (рис. 6.9), лінії керування сегментами “a - h” усіх розрядів об`єднані.

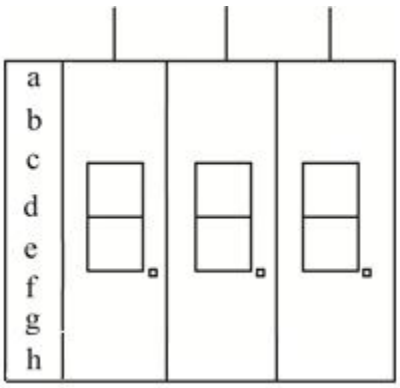


Рис. 6.9.

Розділяються лише лінії живлення розрядних груп (анодних або катодних).

Мозаїчні індикатори використовують у разі необхідності відображення графічних об’єктів, наприклад символів алфавіту. Такі індикатори побудовані на базі матриці індикаторів. Розрізняють два типи мозаїчних індикаторів – монохромні та двокольорові. Такі індикатори

звичайно використовуються для побудови дисплеїв із великою кількістю пікселей. Матриці індикаторів мають розмір 4 х 4, 8 х 8, 10 х 10 світлодіодів. Електричну схему мозаїчного індикатора розміром 2 х 2 приведено на рис. 6.10. Мозаїчні індикатори можуть використовуватися лише у дисплеях із динамічною індикацією.

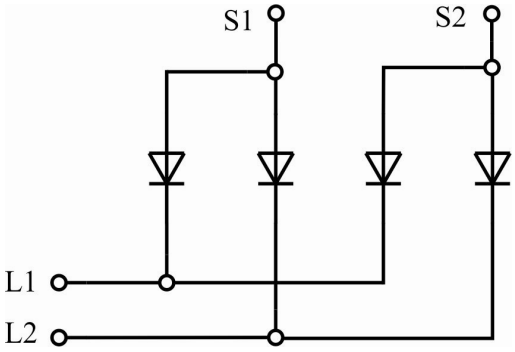


Рис. 6.10 .

Метод статичної індикації

У дисплеях, що використовують метод статичної індикації, через світлодіоди індикатора протікає безперервний робочий струм. Для забезпечення такого метода кожен одиночний семисегментний індикатор необхідно підключати через власний буферний підсилювач. В якості таких підсилювачів досить часто використовують паралельні регістри. Для передачі інформації на регістри використовується шина P8 мікроконтролера (є у декілька AVR-процесорів з апаратним контролером P8), або ця шина формується програмний шляхом.

На рис. 6.11 приведено принципову схему дворозрядного індикатора з програмним формуванням шини зв’язку з буферними регістрами.

У схемі використано два буферних регістра D2, D3, підключених до загальної паралельної шини, яка формується з використанням порту PC. Для активізації виходів регістрів на лінії керування OE заведені активні логічні рівні -«0».

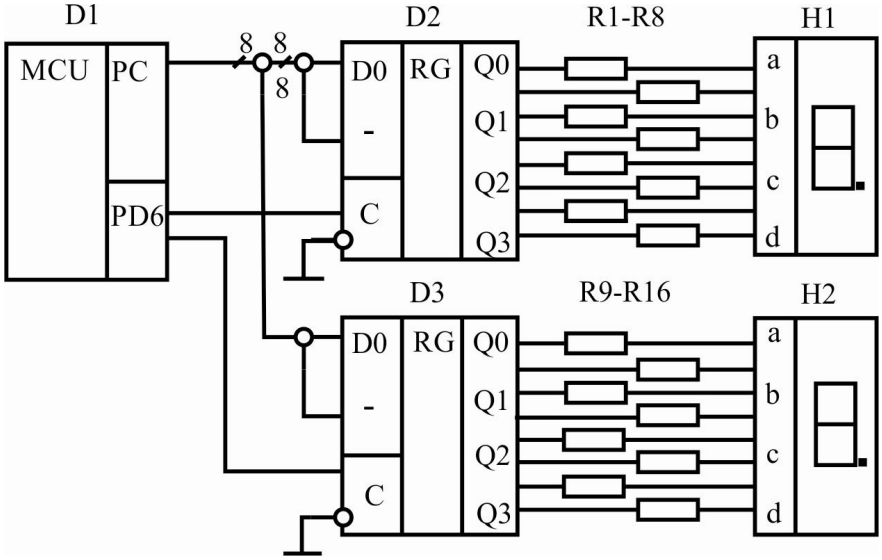


Рис. 6.11

Запис інформації у регістри одиниць та десятків відбувається за рахунок формування відповідних сигналів керування на лініях портів PD6, PD7. Резистори R1 – R16 використовуються для обмеження струму, що протікає через світлодіоди індикаторів H1, H2.

Використання метода статичної індикації спрощує програму-драйвер індикатора, але при цьому за рахунок великої кількості лінії керування та додаткових регістрів суттєво ускладнюється печатна плата.

Метод динамічної індикації

У дисплеях, що використовують динамічний метод індикації, через світлодіоди протікає імпульсний струм. Використовується спільний буферний регістр-підсилювач для керування сегментами “a - h” індикатора. Напруга живлення по черзі поступає на розряди індикатора. Синхронно із напругою живлення розрядів визначається і стан сегментів цих розрядів. Таким чином інформація по черзі виводиться у розряди сотень, десятків та одиниць. Для вилучення оптичних ефектів частота комутації кожного розряду повинна перевищувати 25 Гц. Світлодіоди індикаторів доцільно використовувати на частотах комутації до 0,5 - 4 кГц. Для забезпечення нормальної яскравості випромінювання сегментів

необхідно підвищувати імпульсний струм світлодіодів так, щоб середнє значення струму відповідало номінальному статичному струму. Так у трирозрядному індикаторі необхідно підвищувати імпульсний струм у три рази, у порівнянні з відповідним індикатором статичного типу.

Динамічний метод індикації реалізується схемотехнічно простіше, у порівнянні із статичним, але програмний продукт більш складний.

На рис. 6.12 приведено принципову схему дворозрядного індикатора, у якому використано метод динамічної індикації.

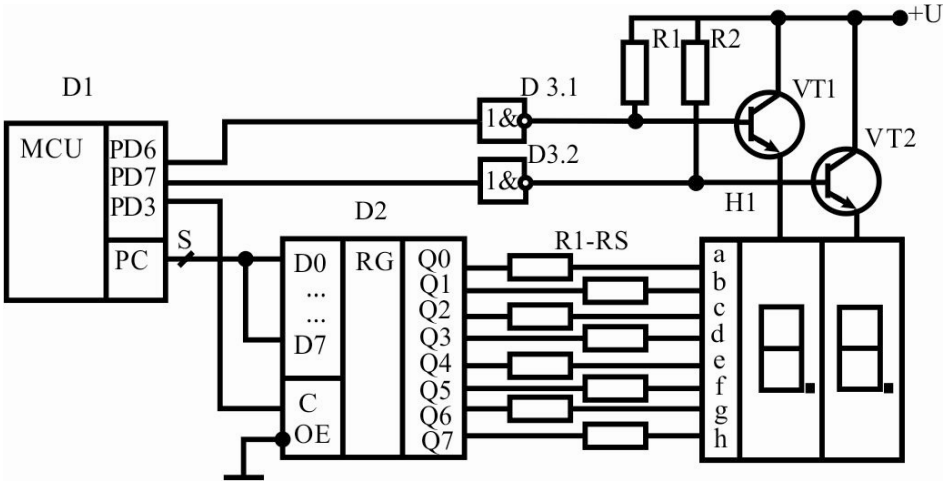


Рис. 6.12

У схемі використано один буферний регістр, який підключено до паралельної шини мікроконтролера (порт PC). Для активізації виходів регістра на лінію OE подано низький логічний рівень напруги. Активізація відповідних розрядів індикатора реалізується за допомогою ліній процесора PD6, PD7. Для запису інформації з паралельної шини у регістр використовується сигнал керування, що формується на лінії PD3 мікроконтролера. Резистори R1 – R8 використовуються для обмеження струму світлодіодів індикатора. Для нормалізації напруги, що прикладається до світлодіодів індикатора використано підсилювачі напруги D3.1, D3.2. Транзистори VT1, VT2 використовуються для комутації розрядів індикатора.

Підготовка даних для виводу на семисегментний індикатор

Для виводу цифр на семисегментний індикатор необхідно провести їх перекодування. Визначення кодів залежить від упаковки сегментів в індикаторі, упаковки паралельної шини даних та від того, який вихідний рівень порту паралельної шини є активним. У табл. 6.1 приведено приклад визначення кодів «0» та «2» для схеми розташування сегментів, що наведено на рис. 6.7 та визначеного у таблиці порядку пакування бітів порту у індикатора з спільним анодом (LOW-активних рівній керування сегментами).

Таблиця 6.1

Сегменти індикатора	h	g	f	e	d	c	b	a
Упаковка ліній шини	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
Код «0»	1	1	0	0	0	0	0	0
Код «2»	1	0	1	0	0	1	0	0

Коди цифр визначають у масиві та розміщують в пам’яті програм.

***Приклад* 6.7** У прикладі приведено фрагмент програми з визначенням масиву кодів.

```
//----------------------------
const unsigned char digits[] = {0b11000000, 0xF9, 0b10100100,
0xB0, 0x99, 0x92, 0x82, 0xF8, 0x80,  0x90};
//----------------------------
```

***Приклад* 6.8**. У прикладі приведено програму відтворення цифри «2» у розряді десятків трирозрядного LED-індикатора. Код цифри «2» передається на вихід шини даних. Для перезапису цього коду на вихід буферного регістру формується строб керування «CS_LED». Після цього активізується лінія вибірки розряду десятків індикатора.

```
//----------------------------
PORTC = digits[2];  //передача на шину даних коду цифри «1»
PORTD |= (1<<CS_LED);    //формування стробу запису коду у
PORTD &= ~(1<<CS_LED);   //буферний регістр
PORTD &=~(1<<TEN);       //активізація розряду сотень
//----------------------------
```

Звичайно необхідно передавати на дисплей багаторозрядні десяткові числа. У таких випадках використовують двійково-десяткове (BCD) перетворення з подальшим перекодуванням символів розрядів.

***Приклад* 6.9.** У прикладі приведено функцію BCD – перетворення двобайтних чисел. Передача багаторозрядного десяткового числа у функцію реалізується за допомогою змінної «out», результати перетворення (цифри, що відповідають десятковим значенням числа) розташовані у змінних «ones», «tens»,«hunds».

Визначення десяткових розрядів реалізується за рахунок виділення у числі значень сотень десятків та одиниць.

```
     //---------------------------
     unsigned char ones;   //значення одиниць
unsigned char tens;   //значення десятків
unsigned char hunds;  //значення сотень
     void transform(int out)
     {
hunds =  out / 100;
     tens  = (out % 100) / 10;
     ones  =  out % 10;
     }
     //---------------------------
```

Драйвер індикатора з статичною індикацією

У четвертому прикладі приведено програму керування трирозрядним індикатором з використанням методу статичної індикації. Фрагмент схеми приведено на рис. 6.11 (за винятком ліній керування розрядом сотень – лінія PD5).

У схемі використано буферний регістр D6 із статичним керуванням. Запис інформації відбувається при подачі на вхід “C” сигналу LED_CS (порт PD3) з високим рівнем напруги. Регістр D2 з`єднано з шиною катодів індикатора. Коди цифр визначались для семисегментного індикатора з загальним анодом (L - активні рівні напруги), для схеми розташування сегментів, що приведено на рис. 6.7, а. У першому прикладі приведено масив кодів цифр від «0» до «9».

Для активації розрядів сотень, десятків одиниць індикатора використовуються лінії керування “HUND”, “TEN”, “ONE” (активним є L-рівень напруги).

Приклад 6.10. У прикладі на індикатор у розряд десятків виводиться число «6».

```
//----------------------------
//ICC-AVR application builder : 14.05.2016 1:47:18
//Target : M16
//Crystal: 10.000Mhz
#include <iom16v.h>
#include <macros.h>
//---------- Символічні визначення ліній портів та константи
#define   TEN       PD6  //розряд десятків
#define   LED_CS    PD3  //строб запису в буферний регістр
//---------- Масив кодів цифр
const unsigned char digits[] = {0b11000000, 0xF9, 0b10100100,
0xB0, 0x99, 0x92, 0x82, 0xF8, 0x80,  0x90};
//---------- Функція ініціалізації портів
void port_init(void)
{
PORTC = 0xFF;            //початковий стан всі "1"
DDRC  = 0xFF;            //налаштування порту C на передачу
PORTD = 0b11100000;      //початковий стан ліній порту
DDRD  = 0xFF;            //налаштування порту D на передачу
}
//---------- Функція ініціалізації мікроконтролера
void init_devices(void)
{
CLI();                   //загальна заборона переривань
port_init();             //ініціалізація портів
SEI();                   //загальний дозвіл переривань
}
//---------- Головна функція програми
void main(void)
{
init_devices();          //ініціалізація мікроконтролера
//---------- Вивід числа «6» у розряд десятків індикатора
PORTC = digits[6];       //встановити код символу
PORTD |= (1<<LED_CS);    //строб запису у регістр
PORTD &= ~(1<<LED_CS);
PORTD &=~(1<<TEN);       //активація розряду десятків
//--------
while(1)
     {};                 //нескінченний цикл
}
//----------------------------
```

Драйвер трирозрядного індикатора з динамічною індикацією

Нижче приведено програму керування трирозрядним індикатором із використанням методу динамічної індикації

***Приклад* 6.11.** Фрагмент схеми приведено на рис. 6.12 (за винятком ліній керування розрядом сотень – лінія PD5). У програмі забезпечено вивід на індикатор довільного числа, значення якого не перевищує «999». Для виводу число передається як змінна функції BCD - перетворення void transform(int out).

Для забезпечення стабільної частоти індикації розрядів використано таймер рахівник та систему переривань. Індикація розрядів забезпечується програмними фрагментами, що розташовані у функції переривань.

Виділення значень десяткових розрядів забезпечується підпрограмою BCD-перетворення, яка описана у четвертому прикладі.

```
//ICC-AVR application builder : 14.05.2016 1:47:18
//Target : M16
//Crystal: 10.000Mhz
#include <iom16v.h>
#include <macros.h>
//---------- Символічні визначення ліній портів та константи
#define   ONE       PD7  //розряд одиниць
#define   TEN       PD6  //розряд десятків
#define   HUND      PD5  //розряд сотень
#define   LED_CS    PD3  //строб запису в буферний регістр
//---------- Масив кодів цифр
const unsigned char digits[] = {0b11000000, 0xF9, 0b10100100,
0xB0, 0x99, 0x92, 0x82, 0xF8, 0x80,  0x90};
//---------- Визначення змінних
unsigned char indicator  = 1;      //змінна поточних розрядів
//1 - індикація одиниць; 2 – десятків; 3 - індикація сотень
unsigned char ones;                //значення одиниць
unsigned char tens;                //значення десятків
unsigned char hunds;               //значення сотень
//---------- Функція ініціалізації портів
void port_init(void)
{
PORTA = 0;               // початковий стан
DDRA  = 0;               // налаштування порту A
PORTC = 0xFF;            // початковий стан всі "1"
DDRC  = 0xFF;            // налаштування порту C на передачу
PORTD = 0b11100000;      // початковий стан ліній порту
DDRD  = 0xFF;            // налаштування порту D на передачу
}
//---------- Функція ініціалізації таймера
void timer0_init(void)
{
```

```
TCCR0 = 0x00;           //зупинка
TCNT0 = 0xF9;           //константа відліку
OCR0  = 0xFA;           //спрацювання таймеру
TCCR0 = 0x05;           //старт таймеру
}
//---------- Функція BCD - перетворення
void transform(int out)
{
hunds = out / 100;
tens = (out % 100) / 10;
ones = out % 10;
}
//---------- Функція переривань переповнення таймера T0
#pragma interrupt_handler timer0_ovf_isr:10
void timer0_ovf_isr(void)
{
indicator++;        //модифікація поточного розряду індикації
switch (indicator)
        {
        case 1:                     //індикація одиниць
               {
               PORTC = digits[ones]; //встановити код символу
               PORTD |= (1<<LED_CS); //формування строб
               PORTD &= ~(1<<LED_CS);
               PORTD |= (1<<HUND);   //активізація розряду
               PORTD &=~(1<<ONE);
               }; break;
       case 2:                      //індикація десятків
              {
               PORTC = digits[tens];
               PORTD |= (1<<LED_CS);
               PORTD &= ~(1<<LED_CS);
               PORTD |= (1<<ONE);
               PORTD &=~(1<<TEN);
               }; break;
       case 3:                      //індикація сотень
              {
               PORTC = digits[hunds];
               PORTD |= (1<<LED_CS);
               PORTD &= ~(1<<LED_CS);
               PORTD |= (1<<TEN);
               PORTD &=~(1<<HUND);
               indicator = 0;
               }; break;
       };
TCNT0 = 0xF9;                       //завантаження таймеру
}
//---------- Функція ініціалізації мікроконтролера
void init_devices(void)
{
CLI();
port_init();   //ініціалізація портів
timer0_init(); //ініціалізація таймеру
```

```
TIMSK = 0x01;  //дозвіл переривань переповнення таймера T0
SEI();         //загальний дозвіл переривань
}
//---------- Головна функція програми
void main(void)
{
init_devices();     //ініціалізація мікроконтролера
transform (145);    //BCD- перетворення числа «145»
while(1)  {};       // нескінченний цикл
}
//---------------------------
```

6.5. Побудова дисплеїв мікроконтролерних систем на рідкокристалічних знакових індикаторах

Загальні відомості про рідкокристалічні символьні дисплеї

Алфавітно-цифрові рідкокристалічні дисплеї є недорогим і зручним рішенням, що дозволяє заощадити час і ресурси при розробці нових виробів, забезпечують відображення великого об'єму інформації.

Виробляють наступні типи рідкокристалічних дисплеїв:

- цифрові, мнемонічні, дисплеї на замовлення;
- алфавітно-цифрові знакосинтезуючі;
- графічні монохромні та кольорові.

Найбільш поширеними є алфавітно-цифрові знакосинтезуючі рідкокристалічні (РКІ) дисплеї (рис. 6.13).

Рис. 6.13

Такі дисплеї випускаються із елементами підсвічування, що дозволяє їх експлуатувати при зниженій освітленості. Існують вироби з розширеним

температурним діапазоном (-20°С ... +70°С), які можуть працювати в складних експлуатаційних умовах, зокрема в переносній апаратурі.

Значна кількість алфавітно-цифрових дисплеїв випускається з вбудованим контролером HD44780 фірми Hitachi, або йому сумісним, що значно спрощує програми-драйвери індикаторів.

По кількості рядків та знакомісць існує декілька поширених форматів дисплеїв (символів х рядків): 8 х 2, 16 х 1, 16 х 2, 16 х 4, 20 х 1, 20 х 2, 20 х 4, 24 х 2, 40 х 2, 40 х 4. Зустрічаються і менш поширені формати такі як 8 х 1, 12 х 2, 32 х 2.

В рамках одного конструктиву РКІ-модуль може мати ряд модифікацій. Зокрема, можуть застосовуватися індикатори, що відрізняються кольором фону і кольором символів. Рідкокристалічні дисплеї можуть оснащуватися заднім підсвічуванням, яке може бути реалізована декількома способами: за допомогою електролюмінісцентної панелі, люмінісцентної лампи з холодним катодам (CCFL), та на основі світлодіодної матриці (LED).

Опис виводів дисплею. Схеми підключення

В якості прикладу розглянемо дисплей WH1602B фірми Winstar. У табл. 6.2 приведено опис виводів дисплея.

Звичайно використовують дві схеми підключення дисплею – 4- та 8-бітову. На відміну від 8-бітової схеми у 4-бітовій шина даних формується лише за допомогою ліній даних DB4 - DB7. Шина керування та живлення дисплею реалізуються однаково.

На рис. 6.14 приведено 8-бітову схему включення дисплею. Ця схема дозволяє досліджувати як 8- так і 4-бітовий варіанти інтерфейсу.

Загальні відомості про вбудований контролер HD44780. До складу дисплея входять:

- рідкокристалічна матриця що має 2 рядки по 16 знакомісць. Кожне знакомісце виконано у вигляді матриці що має набір 5х11 пікселей;

- контролер HD44780;
- системи живлення, синхронізації та сканування знакомісць.

Таблиця 6.2

Контакт	Символ	Рівень	Опис
1	Vss	0В	Нуль напруги живлення
2	Vdd	5,0В	Напруга живлення для логіки
3	Vo	0 - 5В	Регулювання яскравості
4	RS	H/L	H-дані, L-код інструкції
5	RW	H/L	H-читання, L-запис
6	E	H, H/L	Строб передачі даних/команд
7	DB0	H/L	Дані, біт «0»
8	DB1	H/L	Дані, біт «1»
9	DB2	H/L	Дані, біт «2»
10	DB3	H/L	Дані, біт «3»
11	DB4	H/L	Дані, біт «4»
12	DB5	H/L	Дані, біт «5»
13	DB6	H/L	Дані, біт «6»
14	DB7	H/L	Дані, біт «7»
15	A	-	Анод світлодіода підсвічування
16	K	-	Катод світлодіода підсвічування

До складу контролера входять:

- регістри команд (РК) та даних (РД);
- пам'ять даних, що відображаються - DDRAM;
- пам'ять знакогенератора - CGRAM.

Регістр команд використовується для передачі команд керування дисплеєм, та адреси відповідних регістрів DDRAM та CGRAM пам’яті. Під час читання вмісту РК доступна інформація про стан прапора занятості контролера на час виконання чергової команди (BF) та поточний стан рахівника адреси пам’яті (AC). Для доступу до РК на шині керування задається рівень RS = 0.

За допомогою регістру даних проводиться обмін даними з DDRAM або CGRAM областями пам’яті вбудованого контролера. Для доступу до РД на шині керування задається рівень RS = 1. Запис інформації

забезпечується при рівні напруги на лінії шини керування RW = 0, а читання, відповідно, при RW = 1.

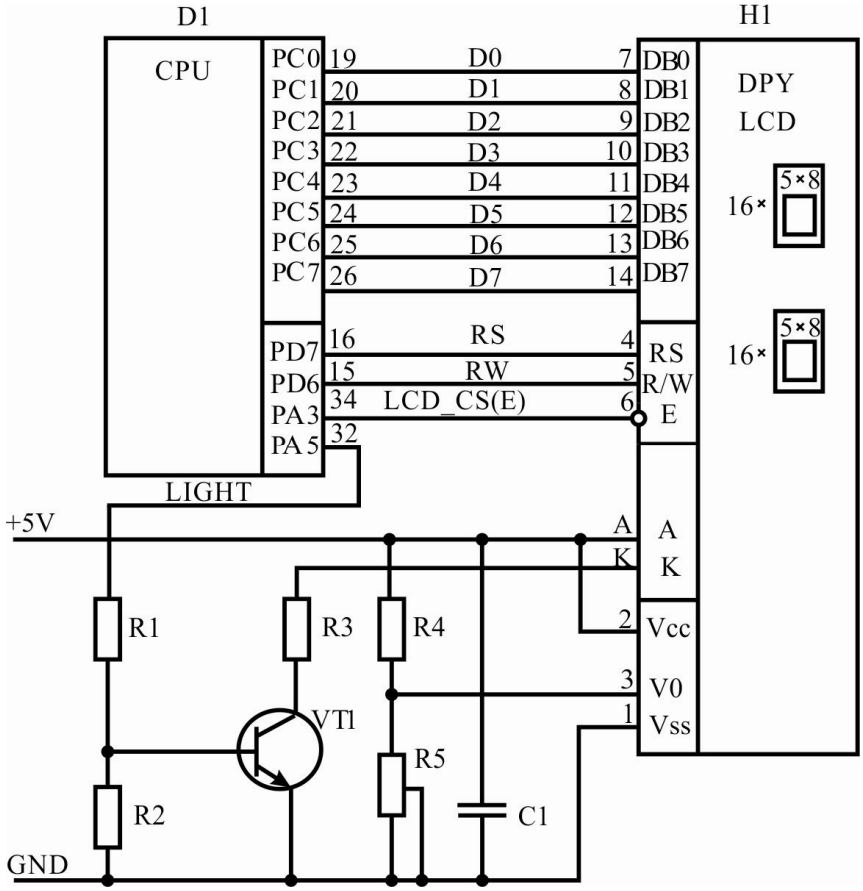


Рис. 6.14

Кожна операція обміну даними супроводжується подачею стробу (E). На рис. 6.15 зображені часові діаграми, що ілюструють запис інформації. До пам'яті даних DDRAM (Display Data RAM) записують ASCII - коди символів, що відображаються. Ємність цієї пам’яті складає 80 байт.

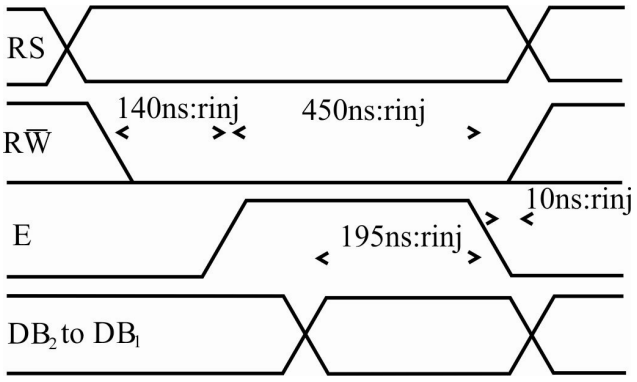


Рис. 6.15

На рис. 6.16 показано співвідношення між адресами коду в DDRAM та положенням зображення на дисплеї з двома рядками.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

Рис. 6.16

Наявність додаткових адрес DDRAM та можливості зміщувати зображення дозволяє виводити на дисплей рухомі текстові повідомлення.

Пам'ять знакогенератора CGRAM (Character Generator RAM) має дві області. У першій області сталої пам’яті (ROM) записані бітові комбінації, які дозволяють відтворювати на знакомісцях певні символи. Друга область RAM – типу. До цієї області користувач може записувати свої власні унікальні графічні зображення. Їх кількість обмежена та залежить від формату зображення. Наприклад, у разі виводу зображень форматом 5x8 пікселей можна сформувати 8 унікальних символів, які розміщують за адресами $00 - $07. У табл. 6.3 зображено приклад формування зображення символу «R» у форматі 5x8 пікселей з розміщенням за адресою CGRAM $00. Адреса та дані представлені у бінарному форматі.

Таблиця 6.3

Код символу у DDRAM	**Адреса CGRAM (RAM)**								**Дані CGRAM (RAM)**								**Зони елемента**
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
00000000b	*	*	0	0	0	0	0	0	*	*	*	1	1	1	1	0	Зона символу
	*	*	0	0	0	0	0	1	*	*	*	1	0	0	0	1	
	*	*	0	0	0	0	1	0	*	*	*	1	0	0	0	1	
	*	*	0	0	0	0	1	1	*	*	*	1	1	1	1	0	
	*	*	0	0	0	1	0	0	*	*	*	1	0	1	0	0	
	*	*	0	0	0	1	0	1	*	*	*	1	0	0	1	0	
	*	*	0	0	0	1	1	0	*	*	*	1	0	0	0	1	
	*	*	0	0	0	1	1	1	*	*	*	0	0	0	0	0	Зона курсору

У табл. 6.4 приведена відповідність зображень ASCII- кодам русифікованого дисплея для області CGRAM (ROM).

Таблиця. 6.4

H / L	LLLL	LLLH	LLHL	LLHH	LHLL	LHLH	LHHL	LHHH	HLLL	HLLH	HLHL	HLHH	HHLL	HHLH	HHHL	HHHH
LLLL	CG RAM (1)															
LLLH	CG RAM (2)															
LLHL	CG RAM (3)															
LLHH	CG RAM (4)															
LHLL	CG RAM (5)															
LHLH	CG RAM (6)															
LHHL	CG RAM (7)															
LHHH	CG RAM (8)															
HLLL	CG RAM (1)															
HLLH	CG RAM (2)															
HLHL	CG RAM (3)															
HLHH	CG RAM (4)															
HHLL	CG RAM (5)															
HHLH	CG RAM (6)															
HHHL	CG RAM (7)															
HHHH	CG RAM (8)															

У верхній строчці таблиці розміщено старшу (H), а в лівому стовпчику – молодшу тетраду (L) байту коду. Наприклад, ASCII-код цифри «0» - $30.

Система команд контролера HD44780

У табл. 6.5 приведено перелік можливих команд контролера HD44780 (для дисплея з двома рядками), станів шини даних/керування, опис результатів їх дії та часу виконання (для частоти генератора 270кГц).

Таблиця 6.5

Команда	Код команди										Опис дії команди	Час [мкс]
	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0		
Очищення дисплею	0	0	0	0	0	0	0	0	0	1	Дисплей очищається. Рахівник адреси DDRAM скидається в 0.	1530
Перехід на початок DDRAM	0	0	0	0	0	0	0	0	1	*	Скидається рахівник адреси DDRAM (адреса $0). Вміст DDRAM при цьому не змінюється.	1530
Режим вводу даних	0	0	0	0	0	0	0	1	ID	S	ID = 0 – інкремент рахівника адреси DDRAM (CGRAM) після запису; ID = 1 – декремент рахівника адреси DDRAM (CGRAM) після запису; S = 0 – заборона зсуву тексту дисплея; S = 1 – дозвіл зсуву тексту дисплея.	39
Функції дисплею	0	0	0	0	0	0	1	D	C	B	D=0 – виключити дисплей; D=1 – включити дисплей; C=0 – курсор не відображати; C = 1 – курсор відображати; B = 0 – символу з курсо-ром не мигає; B = 1 – символу з курсором мигає.	39
Змішення тексту чи курсору	0	0	0	0	0	1	SC	RL	*	*	SC = 0 – зміщення курсору; SC = 1 – зміщення тексту; RL = 0 – зміщення вліво; RL = 1 – зміщення направо.	39
Функція ініціалізації	0	0	0	0	1	DL	N	F	*	*	DL = 0 – шина даних 4 біта; DL=1 – шина даних 8 біт; N = 0 – дисплей з одним ряд-ком; N = 1 – дисплей з двома рядками; F = 0 – шрифт 5x7 пікселей; F= 1– шрифт 5x10 пікселей.	39
Адреса CGRAM	0	0	0	1	ACG						Встановлення поточної адреси CGRAM.	39
Адреса DDRAM	0	0	1	ADD							Встановлення поточної адреси DDRAM.	39
Читання флагу BF	0	1	BF	AC							Читання флагу BF і лічильника поточної адреси (AC).	1
Запис даних	1	0	Байт даних								Запис даних у DDRAM або CGRAM.	43
Читання даних	1	1	Байт даних								Читання даних із DDRAM або CGRAM.	43

Драйвери індикатора

Набір драйверів визначається типом інтерфейсу індикатора – 4- чи 8-бітовим. Для кожного типу інтерфейсу необхідно використовувати відповідні підпрограми драйверів.

Набір драйверів індикатора для 8-бітового інтерфейсу приведено у прикладі наскрізного проекту, у файлі драйверів lcd.h.

До набору драйверів індикатора входять три групи функцій.

Першу групу складають функцій, що формують сигнали даних та керування.

- формування стробу передачі коду в індикатор – «lcd_strobe». Ця функція дозволяє сформувати на шині керування імпульс запису коду з шини даних у відповідні регістри контролера індикатора та відповідну затримку, необхідну для виконання поточної команди;
- передачі байту на індикатор – «lcd_send». Ця функція дозволяє сформувати послідовність операцій необхідних для передачі у контролер індикатора байту;
- формування затримки на час виконання команди у контролері індикатора – «Delay». Ця функція дозволяє сформувати затримку на час виконання контролером індикатора поточної команди.

До другої групи входить функція ініціалізації –«InitLCD». Ініціалізація є обов'язковою процедурою, що забезпечує налаштування індикатора на певний режим роботи.

До третьої групи входять функції виводу повідомлень:

- очистки табло «LCD_Clrscr»;
- встановлення адреси знакомісця виводу «gotoz»;
- вивід символу – «OutChar». Ця функція дозволяє виводити певний символ. Для передачі параметрів використовується змінна функції «litera»;
- вивід повідомлення з 16 символів – «outtext». Для передачі параметрів використовується вказівник на масив;

- вивід повідомлення з 16 символів у першу або другу строчки, відповідно «DisplayTextL1» та «DisplayTextL2». Для передачі параметрів використовується вказівник на масив;
- вивід три розрядного числа у певний рядок, починаючи з певного місця «DisplayData». Для передачі параметрів використовується змінна «tmp».

Ініціалізація LCD

Алгоритми ініціалізації відрізняються для 4 та 8-бітного інтерфейсів.

Для 8-бітного інтерфейсу ініціалізацію проводять наступним чином:

- після подачі напруги живлення виконують затримку тривалістю не менше як 15мс, після чого до регістру команд передають команду ініціалізації 0b0011****;
- виконують затримку тривалістю не менш як 4,1мс, після чого до регістру команд ще раз передають ту само команду ініціалізації;
- виконують затримку тривалістю не менш як 100мкс, після чого до регістру команд ще раз передають ту само команду ініціалізації;
- до регістру команд передають команду встановлення функцій 0b0011NF** та виконують затримку тривалістю не менш як 39мкс;
- до регістру команд передають команду вимкнення дисплею 0b00001000 та виконують затримку тривалістю не менш як 39мкс;
- до регістру команд передають команду очистки дисплею 0b00000001 та виконують затримку тривалістю не менш як 39мкс;
- до регістру команд передають команду встановлення режиму 0b000001I/DS та виконують затримку тривалістю не менш як 39мкс.

Під час ініціалізації для формування часових затримок 15мс, 4.1мс та 100мкс не допускається використання біту стану контролера BF. Цей біт можна використовувати лише для наступних операцій керування контролером.

Приклад функції ініціалізації для 8-бітного режиму роботи індикатора «InitLCD» приведено у прикладі наскрізного проекту, у файлі драйверів індикатора lcd.h.

Підготовка текстових повідомлень

При підготовці текстових повідомлень доцільно використовувати наступні прийоми.

Якщо текст включає лише латинські символи, повідомлення доцільно оформляти у вигляді стрінгу.

У цьому випадку вивід стрінгу (наприклад, "Hello World") може бути реалізовано з використанням функції `outtext ("Hello World")`.

Слід зауважити, що драйвер виводу текстового повідомлення «OutLine» завжди виводить у строчку по 16 символів. Тому стрінг доцільно доповнювати пробілами до 16 символів.

Якщо виводяться цифри від 0 до 9, причому їх значення може змінюватися, наприклад при формуванні цифрових індикаторів.

У цьому випадку доцільно виводити ASCII - коди цифр із використанням підпрограми виводу окремих символів «outchar». Коди цифр мають зміщення на сталу величину $30. Тому перед виводом цифри код можна отримати шляхом додавання величини зміщення до значення цифри. Наприклад, код цифри «4» - $30 + 4 = $34. У функції «`DisplayData`» це враховано шляхом додавання до значення цифри

коду «0» – `outchar ('0'+tmp/100)`.

Якщо текст включає кирилізовані символи, то можлива ручна побудова масиву, з використанням табл. 6.4. Такий спосіб формування повідомлень досить важкий. У цьому випадку доцільно використовувати утиліту генерації текстових повідомлень HD44780.exe, яка розроблена для створення С-програм.

Утиліта підготовки текстових повідомлень HD44780.exe

На рис. 6. 17 зображено головне вікно програми.

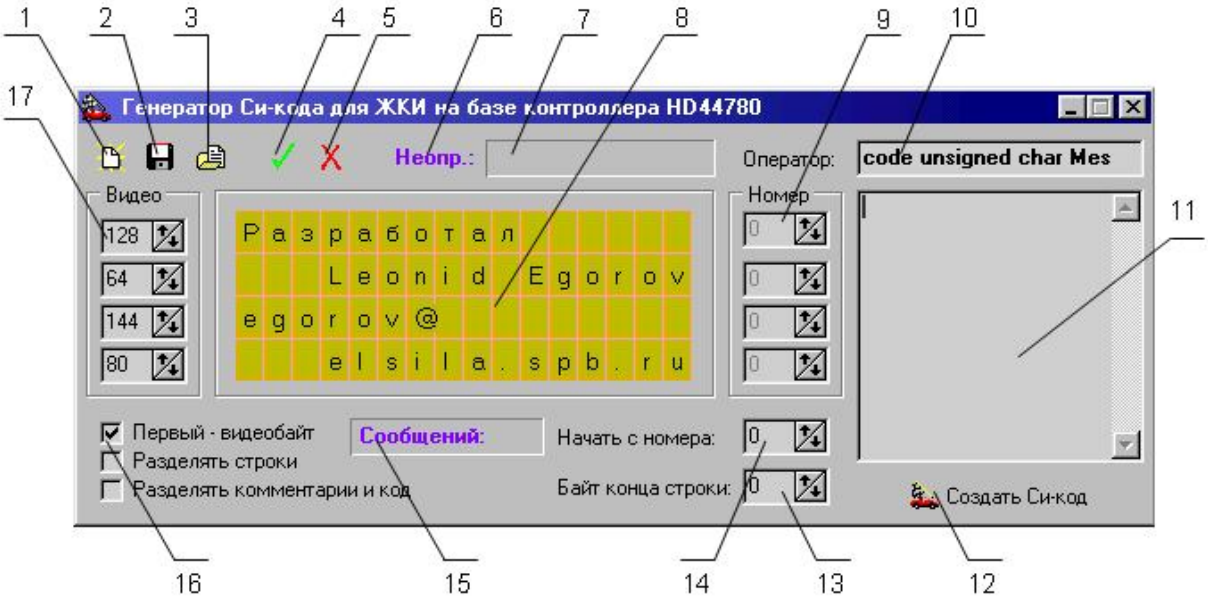


Рис. 6.17

Розглянемо інтерфейс користувача програми. У програмі використовується інтерактивна система підказок. Якщо невідомий якийсь елемент вікна програми, необхідно навести на нього курсор, що приведе до появи короткої підказки.

У програмі використовуються наступні елементи керування:

1. Кнопка прискореного виклику – «Створення нового проекту».

Очищає всі повідомлення і їх адреси початку у пам'яті програми.

2. Кнопка прискореного виклику – «Зберегти як...».

Зберігає текст або як проект *.lcd (він доступний для подальшого завантаження і правки) або як *.h – заголовочний файл.

3. Кнопка прискореного виклику – «Відкрити проект».

Відкриває проект *.lcd. Відкриваються всі відеообласті (17), положення всіх покажчиків (9), текстовий оператор (10), всі текстові повідомлення проекту (11).

4. Кнопка прискореного виклику – «Показати всі повідомлення». Використовується при переводі повідомлень (9, 8) з рядка в рядок. У протилежному випадку попереднє повідомлення не стирається з екрану.

5. Кнопка прискореного виклику – «Очистити екран». Очищає дисплей (8), всі відеообласті зберігаються.

6. Індикатор номеру поточного рядка. Якщо для поточного повідомлення, що формується у зоні (11), не задана відеоадреса, відображається слово «Неопр». Для встановлення необхідної адреси необхідно навести курсор на вікно (8) та сформувати клік кнопкою миші.

7. Індикатор номеру повідомлення. Введіть декілька рядків в поле списку повідомлень, розділяючи їх клавішею Enter, кожній строчці буде присвоєний свій номер. Якщо встановити покажчик (9) на потрібному повідомленні, перевести курсор на потрібне знакомісце дисплея (8) і клікнути кнопку миші, відбудеться позиціонування повідомлення з цим номером на екрані в задану область.

8. Заголовок повідомлення. Цей заголовок буде стояти перед кодами кожного сформованого повідомлення. Для приведеного вище фрагмента він виглядав так:

«unsigned char Mes».

Номер повідомлення MESX генерується автоматично.

9. В цій області формуються тексти повідомлень.

10. Формування кодів у буфері обміну.

Після закінчення введення всіх повідомлень збережіть проект і натисніть клавішу «Створити Сі-код». Код буде розташований в буфері обміну.

11. Байти початку повідомлення (16) та кінця повідомлення (13). Можлива корекція байтів початку повідомлення (17).

***Приклад* 6.12.** Розглянемо приклад формування повідомлення «Команда».

```
/* Date: 31.05.2010   Time: 15:29:46 */
```

```
/* Maximum length of a line: 7  byte */
/* In total byte: 8                  */
/* FileName: вК                      */
/* [0] "Команда" */  unsigned char
Mes0[]={75,111,188,97,189,227,97,0};
```

У наступному прикладі розглянуто наскрізний проект, який включає як драйвери дворозрядного індикатора, так і підготовлені дані для повідомлень.

***Приклад* 6.13.** У проекті реалізовано вивід повідомлень у першу ("Работа") та другу ("Программирование") строчки індикатора та вивід символу «2» на 1 строчку на 15 позицію. Проект включає чотири файли Lab_4.c, delay.h, lcd.h, init.h. Головний файл проекту Lab_4.c має наступний вигляд.

```
//-----------------------------------------------------------
//ICC-AVR application builder : 14.06.2016 1:27:18
//Target : M16
//Crystal: 10.000Mhz
#include <iom16v.h> //файл визначень мікроконтролера ATmega16
#include <macros.h>
//---------- Вкладені файли
#include <delay.h>  //файл програмних затримок
#include <lcd.h>    //файл драйверів індикатора
#include <init.h>   //файл ініціалізації
//---------------------------
//---------- Визначення масивів повідомлень
/* "Работа" */
unsigned char Mes1[]={80,97,178,111,191,97,0};
/* "Программирование" */
unsigned char Mes2[]={168,112,111,180,112,97,188,188,184,112,
111,179,97,189,184,101,0};
//---------------------------
//---------- Головна функція програми
void main(void)
{
InitDevices();           //Ініціалізація портів та LCD
LCD_LED_ON;              //Вмикання LED - освітлення індикатора
Lcd_Clrscr();            //очистка індикатора
//outtext ("Hello World");    //вивід тексту Hello World
DisplayTextL1(Mes1);     //вивід тексту "Работа" у 1 рядок
DisplayTextL2(Mes2);     //вивід "Программирование" у 2 рядок
DisplayData (234);  //вивід числа 234 у 2 рядок (з 3 місця)
while(1)  {};            //Нескінченний цикл
}
//-----------------------------------------------------------
```

Файл з функціями ініціалізації init.h має наступний вигляд.

```
//-------------------------------------------------------------
//Файл init.h
//-------------------------------------------------------------
//---------- Функція ініціалізації портів
void PortInit(void)
{
PORTA = 0b11001000;          //Початкове значення порту
DDRA  = 0b11111000;          //0- прийом, 1 - передача
PORTB = 0b00000000;
DDRB  = 0b00000000;
PORTC = 0;
DDRC  = 0b11111111;
PORTD = 0b11110010;
DDRD  = 0b11111010;
}
//----------------------------
//---------- Функція ініціалізації мікроконтролера
void InitDevices(void)
{
PortInit();                  //ініціалізація портів
InitLCD ();                  //ініціалізація LCD
}
//-------------------------------------------------------------
```

Файл драйверів індикатора lcd.h має наступний вигляд.

```
//-------------------------------------------------------------
// Файл lcd.h
//-------------------------------------------------------------
//---------- Символічні імена портів та ліній керування LCD
#define   LCD_PORT_DATA        PORTC
#define   LCD_PORT_DATA_DIR    DDRC
#define   LCD_PORT_DATA_OUT    0xff
#define   LCD_PORT_CONTROL     PORTD
#define   LCD_PORT_CONTROL_DIR DDRD
#define   LCD_RS               (1<<PD7)
#define   LCD_RW               (1<<PD6)
#define   LCD_E                (1<<PA3)
#define   LCD_LED              (1<<PA5)
//---------- Макрос визначення поточної позиції
#define   gotoxy(x,y)    gotoz((x)|((y)<<6)) //
//---------- Макроси керування лініями LCD
#define   lcd_set_E      (PORTA |= LCD_E)
#define   lcd_set_RS     (LCD_PORT_CONTROL |= LCD_RS)
#define   lcd_set_RW     (LCD_PORT_CONTROL |= LCD_RW)
#define   lcd_clr_E      (PORTA &=~LCD_E)
#define   lcd_clr_RS     (LCD_PORT_CONTROL &=~LCD_RS)
#define   lcd_clr_RW     (LCD_PORT_CONTROL &=~LCD_RW)
#define   LCD_LED_ON     (PORTA |=LCD_LED)
//---------- Команди керування режимами LCD (див. табл.4.5)
#define   LCD_INI         0x30     //ініціалізація
#define   LCD_CLEAR       0x01     //очистка
#define   LCD_HOME        0x02     //перехід на початок екрану
```

```
#define   LCD_MODE_SET    0x06     //режим програмування
#define   LCD_OFF         0x08     //вимикання індикатору
#define   LCD_ON          0x0C     //включення індикатору
#define   LCD_NEW_LINE    0xC0     //перехід на нову лінію
#define   LCD_FUNCTION_SET 0x38    //установка функцій
//-------------------------------------------------------

//---------- Функція формування стробу LCD
void lcd_strobe (void)
{
lcd_set_E;               //встановлення «1» на лінії стробу
DelayMcs(4);             //формування тривалості стробу
lcd_clr_E;               //встановлення «0» на лінії стробу
DelayMcs(4);             //формування затримки після стробу
}
//----------------------------
//---------- Функція передачі байту в LCD
void lcd_send (unsigned char lcddata)
{
LCD_PORT_DATA=lcddata;   //встановлення даних на шині
lcd_strobe();            //формування стробу
DelayMcs(60);            //протокольна затримка після передачі
}
//----------------------------
//---------- Функція очистки LCD
void Lcd_Clrscr (void)
{
lcd_clr_RS;              //встановлення режиму команди
lcd_clr_RW;              //встановлення режиму запису
lcd_send(LCD_CLEAR);     //передача команди очистки дісплею
DelayMs(2);              //протокольна затримка після передачі
}
//----------------------------
//---------- Функція ініціалізації LCD
void InitLCD (void)
{
LCD_PORT_CONTROL_DIR |=(LCD_RS | LCD_RW); //режим роботи порту
lcd_clr_RS;              //встановлення режиму команди
lcd_clr_RW;              //встановлення режиму запису
DelayMs(20);             //протокольна затримка
lcd_send(LCD_INI);       //передача команди ініціалізації
DelayMs(5);              //протокольна затримка
lcd_send(LCD_INI);       //передача команди ініціалізації
DelayMcs(150);           //протокольна затримка
lcd_send(LCD_INI);       //передача команди ініціалізації
lcd_send(LCD_FUNCTION_SET); //передача команди установки
lcd_send(LCD_OFF);       //передача команди вимикання
lcd_send(LCD_CLEAR);     //передача команди очистки
DelayMs(2);              //протокольна затримка
lcd_send(LCD_MODE_SET);  //передача команди установки режимів
lcd_send(LCD_ON);        // передача команди вмикання
}
//----------------------------
```

```
//---------- Функція передачі адреси знакомісця LCD
void gotoz (unsigned char z)
{
lcd_clr_RS;              //встановлення режиму команди
lcd_clr_RW;              //встановлення режиму запису
lcd_send(z | 0x80);      //передача адреси знакомісця
}
//----------------------------
//---------- Функція передачі одного символу
void outchar (unsigned char litera)
{
lcd_clr_RW;              //встановлення режиму запису
lcd_set_RS;              //встановлення режиму даних
lcd_send(litera);        //передача коду символу
}
//----------------------------
//---------- Функція передачі рядка із 16 символів
void outtext (unsigned char *text)
{
unsigned char i;
for (i = 0; text[i] && i<16; i++) outchar (text[i]);
}
//----------------------------
//---------- Функція виводу тексту у 1 рядок
void DisplayTextL1(unsigned char *tmp)
{
gotoxy(0,0);   //встановлення адреси на початку 1 рядка
outtext(tmp);  //вивід тексту
}
//----------------------------
//---------- Функція виводу тексту у 2 рядок
void DisplayTextL2(unsigned char *tmp)
{
gotoxy(0,1);   //встановлення адреси на початку 2 рядка
outtext(tmp);  //вивід тексту
}
//----------------------------
//---------- Функція виводу три розрядного числа tmp у 2 рядок
void DisplayData(unsigned int tmp)
{
gotoxy(0,1);   //встановлення адреси на початок 2 рядка
outchar ('0'+tmp/100);
outchar ('0'+(tmp%100)/10);
outchar ('0'+(tmp%100)%10);
}
//-------------------------------------------------------------
```

Файл з функціями програмної затримки delay.h має наступний вигляд.

```
//-------------------------------------------------------------
//Файл delay.h
//-------------------------------------------------------------
//--------- Функція програмної затримки на "t_mcs" мікросекунд
void DelayMcs (unsigned int t_mcs)
```

```
{while(t_mcs--)
      {asm("nop"); asm("nop"); asm("nop"); asm("nop");};}
//----------------------------
//--------- Функція програмної затримки на "t_ms" мілісекунд
void DelayMs (unsigned int t_ms)
{while(t_ms--) {DelayMcs(1000);};}
//----------------------------
//--------- Функція програмної затримки на "t_s" секунд
void DelayS (unsigned int t_s)
{while(t_s--) {DelayMs(1000);};}
//------------------------------------------------------------
```

6.6. Клавіатури мікроконтролерних систем

Класифікація клавіатур

При побудові клавіатур, у залежності від їх функціонального призначення, використовують різну кількість кнопок. Розрізняють клавіатури, що призначені:

- для вводу та редагування тексту (персональні комп'ютери);
- переважного вводу цифрової інформації, з можливістю вводу тексту (мобільні телефони);
- вводу лише цифрової інформації (калькулятори);
- епізодичного вводу параметрів мікроконтролерних пристроїв (побутова техніка, мікроконтролерні пристрої промислового призначення).

У клавіатурах із інтенсивним використанням кількість кнопок підвищено. Наприклад, розширена клавіатура персонального комп'ютера має понад 100 кнопок.

Із зменшенням інтенсивності використання клавіатури кількість кнопок зменшується. При цьому, для збереження функціональних можливостей клавіатури, кнопкам призначаються альтернативні функції та вводять можливості для визначення таких функцій (мобільні телефони).

Типова клавіатура мікроконтролера, що використовується епізодично для налаштування параметрів, має 4-6 кнопок (пральні машини, цифрові фотоапарати).

Клавіатури суттєво відрізняються по реакції на натискання кнопок. По цьому показнику розрізняють два типи клавіатур:

- з реакцією при натисканні кнопки;
- з реакцією після натискання та подальшого відпускання кнопки.

Тип реакції важливий для завдання динамічних показників та точності налаштування параметрів.

Елементна база клавіатур

У клавіатурах звичайно використовують механічні дискретні тактові кнопки (рис. 6.18) та набори кнопок (рис. 6.19).

Рис. 6. 18

Рис. 6.19

Разом з тим вже більше 15 років зусиллями компаній Atmel / Quantum та Cypress розвивається технологія безконтактних сенсорних кнопок. Технологія базується на властивості переносу заряду, та дозволяє створювати проектні рішення з реакціями на наближення та дотик. Мікросхеми драйверів кнопок можуть налаштовуватися на різну чутливість, що надає можливість проводити адаптацію клавіатур під різні матеріали панелей та їх товщину.

Цими фірмами запропоновані типові рішення по створенню безконтактних клавіатур.

Компанією Cypress запропоноване сімейство мікросхем для створення сенсорних клавіатур CapSense Express.

Компанією Quantum Research Group (ввійшла до складу фірми Atmel у 2008р.) запропоновані одноканальні драйвери QT100, QT102, датчики

наближення та дотику QT11x, багатоканальні датчики дотику QT 1103, QT 220, QT 240.

Компанія Atmel продемонструвала комплексний підхід до вирішення задач QTouch – технології. Фірмою розроблені мікросхеми драйверів для одиночних кнопок, груп з кількістю менше 10 та більше 10 кнопок. Фірма вивела на ринок технології використання одиночних кнопок QTouchTM та матриць QMatrixTM. Для більшості мікросхем розроблені відповідні плати розвитку. До складу інтегрованого пакету AVR Studio 4.18, що використовується для створення проектів AVR мікроконтролерів, введено спеціалізований розділ для розробки QTouch Studio. На офіційному сайті фірми www.atmel.com представлені бібліотеки для 8-бітових AVR-мікроконтролерів та 32-бітових ARM-мікроконтролерів AT91SAM.

На рис. 6.20 та 6.21 зображено сенсори компаній Quantum та Atmel.

Рис. 6.20

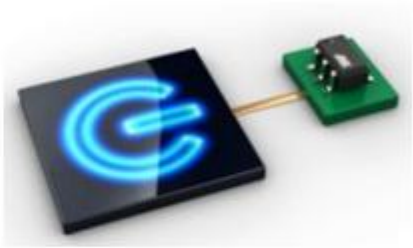


Рис. 6.21

Схемотехнічні прийоми використання окремих кнопок

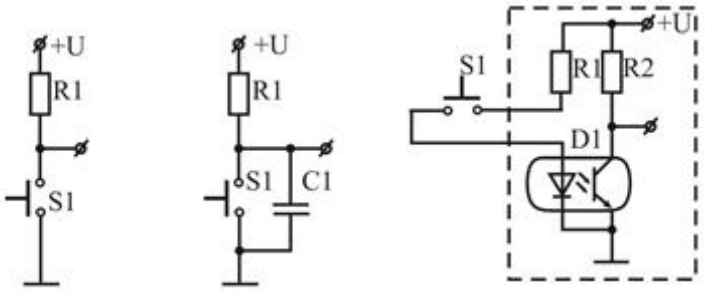
При розробці принципової схеми клавіатури необхідно вирішувати дві задачі:

- забезпечення вводу інформації від кнопок;
- узгодження клавіатури заданого об`єму з мікроконтролером.

Розглянемо схемотехніку клавіатур, що використовують механічні тактові кнопки.

Такі кнопки мають механічну контактну пару, що комутується. При визначенні режиму роботи контактної пари звичайно враховують величину мінімального струму. Для тактових кнопок, які звичайно використовуються при побудові клавіатур, цей струм складає 0,3 – 1 мА. У разі встановлення меншого робочого струму можливі відкази функціонування кнопок.

Для формування цифрових логічних рівній напруги та забезпечення мінімального струму контактів досить часто використовуються схеми, що приведені на рис. 6.22.



а) б) в)

Рис. 6.22

На вибір схемотехнічного рішення суттєво впливають віддаленість кнопки від мікроконтролера та рівень високочастотних завад. При низькому рівні завад та невеликій відстані від кнопки до портів мікроконтролера (0,1 -0,2 м) використовується схема, що приведена на рис. 6.22, а.

При підвищеному рівні завад та відстані від кнопки до мікроконтролера до 1-3 м використовують додатковий конденсатор (схема на рис. 6.22, б).

У разі підвищення рівня завад збільшують струм контактної пари до 3 - 10 мА та величину ємності конденсатора. У разі необхідності збільшення довжини лінії зв’язку між кнопкою та мікроконтролером більше 5 - 10м доцільно використовувати струмову петлю (10 - 100мА) та

оптичні розв’язки. Таку схему приведено на рис. 6.22, в. Канал оптичної розв’язки дозволяє усунути вплив падіння напруги на лінії зв’язку на рівні напруги, що формуються при включеному та виключеному станах кнопки.

Схемотехнічні прийоми побудови клавіатур

Розробку принципової схеми клавіатури проводять по заданому числу кнопок. При малому числі кнопок (1 - 10шт.) звичайно використовують безпосереднє підключення до ліній портів. Такий спосіб побудови клавіатури проілюстровано на рис. 6.23.

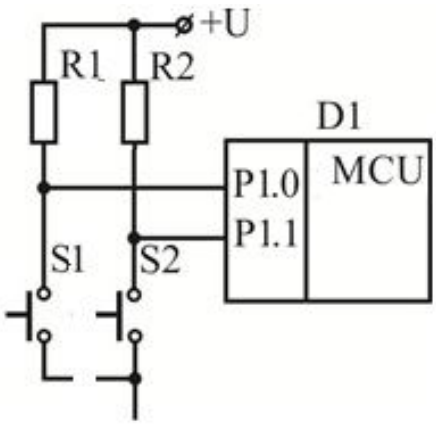


Рис. 6.23

Якщо число кнопок клавіатури знаходиться у діапазоні від 10 до 30 шт., то для вводу інформації доцільно використовувати додаткові регістри. Принципову схему такої клавіатури приведено на рис. 6.24. У цій схемі для читання інформації з регістру D2 використано паралельну шину, яка сформована на базі порту PA мікроконтролера.

Для стробування додаткових регістрів використовуються лінії порту PB. Таке схемотехнічне рішення дозволяє за допомогою двох портів мікроконтролера та 8 додаткових регістрів провести опитування клавіатури із 64 кнопками.

При розробці клавіатур із числом кнопок, що перевищує 30шт., доцільно використовувати схеми із матричною структурою.

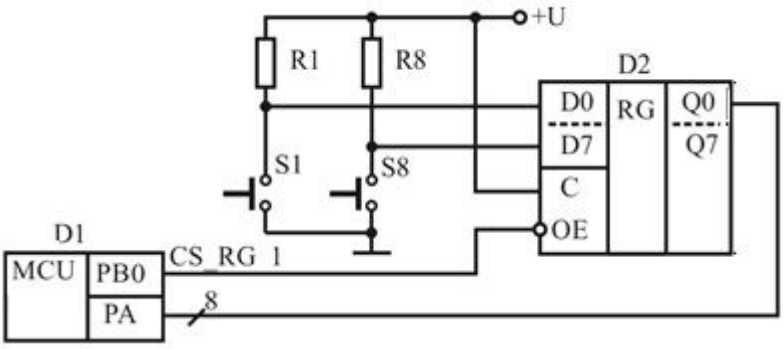


Рис. 6. 24

На рис. 6.25 приведено принципову схему, що ілюструє можливість побудови такої клавіатури.

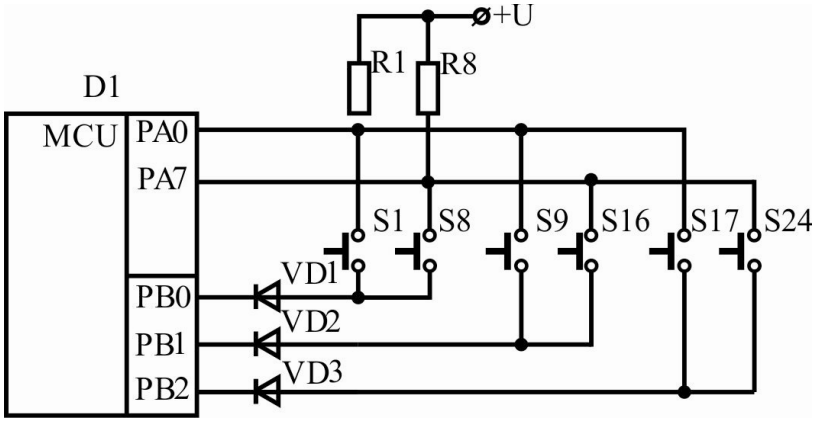


Рис. 6.25

Схема має три групи по 8 кнопок S1 - S8, S9 - S16, S17 - S24, які підключено до загальної паралельної шини вводу даних (порт PA). Групи кнопок через діоди VD1 - VD3 підключено до ліній сканування PB0 – PB2. Діоди виключають можливість короткого замикання по лініям порту PB, у разі одночасного натискання кнопок різних груп.

На лініях сканування по черзі встановлюється “0” - рівень та проводиться опит стану кнопок активізованої групи. У разі використання двох портів максимальна кількість кнопок такої матричної клавіатури досягає 64. Ця схема більш проста, у порівнянні з попереднім варіантом із додатковими регістрами.

Алгоритми ідентифікації натискання кнопки

Комутація контактної пари кнопки характеризується тремтінням контактів – багаторазовими перемиканнями із частотою механічного резонансу. Звичайно тремтіння контактів характерно для замикання контактів. Час тремтіння залежить від типу контактів та може складати від 1 до 20мс. У мікроконтролерних системах при побудові клавіатур часто використовують тактові кнопки із часом тремтіння 1 - 3мс.

Для виключення впливу багаторазового перемикання контактів на інтервалі тремтіння, необхідно формувати алгоритми ідентифікації стану кнопки із врахуванням цього явища.

Звичайно використовують три типи алгоритмів ідентифікації:

- із двома опитами та затримкою на час тремтіння контактів;
- із багаторазовими опитами та мажоритарним принципом розпізнавання стану;
- із заданим числом співпадаючих значень, при багаторазових опитах.

При використанні першого алгоритму проводиться періодичне опитування стану кнопок клавіатури. У разі реєстрації натисненого стану кнопки, для виключення впливу тремтіння контактів, проводиться затримка на час тремтіння, потім – повторне опитування. Результат повторного опитування відповідає стану кнопки.

У другому алгоритмі проводяться багаторазові опитування стану кнопки. Визначаються числа результатів експериментів, що відповідають різним станам кнопки. На підставі порівняння цих чисел, по мажоритарному принципу приймається рішення про стан кнопки. Для виключення впливу тремтіння контактів число опитувань обирають таким чином, щоб час використаний на багаторазові опитування перевищував час тремтіння контактів.

При реалізації третього алгоритму проводиться задане число експериментів. Якщо у серії опитувань отримані різні результати про стан

кнопки, то результати анулюються, а серія повторюється. Як і в попередньому алгоритмі кількість опитувань обирають у відповідності із часом тремтіння контактної пари.

Драйвер клавіатури

При написанні драйвера клавіатури задаються:

- числом кнопок;
- часом тремтіння контактів та типом алгоритму вилучення його впливу;
- часом циклу опитування стану кнопок;
- способом реєстрації натискання кнопки.

Час циклу опитування стану кнопок визначає швидкодію клавіатури. Уведення періоду опитування необхідно для узгодження швидкості роботи мікроконтролерної системи та часу реакції оператора. Звичайно період опитування обирають у діапазоні 0,3 – 1сек.

Для забезпечення заданої циклічності опитування клавіатури, у залежності від особливостей функціональних властивостей мікроконтролера, використовують два прийоми. Якщо у мікроконтролерній системі часові затримки у 0,3 - 1сек при виконанні головного циклу програми є несуттєвими, то вказану затримку реалізують у головному циклі. В іншому випадку циклічне опитування клавіатури забезпечують за рахунок використання переривань від таймера.

Для фіксації натискання кнопок використовують два алгоритми:

- подію реєструють у натисненому стані кнопки;
- подію реєструють у разі, якщо кнопка була натиснена і потім відпущена.

При використанні драйверів, у яких використовується перший алгоритм фіксації, тривале утримання кнопки у натисненому стані приводить до циклічного виконання відповідної підпрограми реакції.

Такий драйвер доцільно використовувати у разі необхідності зміни значень параметрів у широкому діапазоні.

Якщо необхідно забезпечити більш чітку модифікацію змінних, із жорсткою фіксацією їх значень по кожному натисканню кнопок, то доцільно використовувати другий алгоритм.

Приклад драйвера клавіатури

Розглянемо приклад драйвера із наступними вихідними даними:

- число кнопок – 3;
- час тремтіння контактів – 3 мс;
- алгоритм вилучення впливу тремтіння контактів – із двома опитуваннями та затримкою на час тремтіння контактів;
- спосіб та час циклічного опитування стану кнопок – підпрограми опитування кнопок та затримки (узгодження з оператором) знаходяться у головному циклі. Час затримки - 0,3 с;
- спосіб реєстрації стану кнопок – при натисканні.

Драйвер реалізує функції налаштування значення двох змінних :

- «`Voltage`» може мінятися у діапазоні «5 – 100» з кроком «5»,
- «`Current`» може мінятися у діапазоні «10 – 100» з кроком «10»;
- Перехід від однієї змінною до іншої відбувається при натисканні кнопки «`MODE`» - (S3). При цьому на LCD - індикатор виводяться текстові повідомлення, відповідно, «`Voltage`» та «`Current`»;
- Збільшення значення змінної відбувається при натисканні кнопки «UP» - (S1). Значення змінної виводяться на LCD - індикатор;
- Зменшення значення змінної відбувається при натисканні кнопки «DOWN» - (S2). Значення змінної виводяться на LCD – індикатор.

Проект реалізовано з використанням попередньої лабораторної роботи – по виводу інформації на LCD – індикатор. Проект включає п’ять файлів Lab_5.c, delay.h, lcd.h, init.h та key.h. Зміни відбулися в головному

файлі проекту Lab_5.c (включено замість Lab_4.c). Додатково включено файл функцій драйверів клавіатури key.h.

***Приклад* 6.14.** У фрагменті ініціалізації проводиться завантаження дистрибутивних значень змінних, та активізація режиму роботи «VOLTAGE».

```
     //-----------------------------------------------------
//ICC-AVR application builder : 14.06.2016 1:27:18
//Target : M16
//Crystal: 10.000Mhz

#include <iom16v.h>      //файл визначень ATmega16
#include <macros.h>
//---------- Вкладені файли
#include <delay.h>       //файл програмних затримок
#include <lcd.h>         //файл драйверів індикатора
#include <init.h>        //файл ініціалізації
#include <key.h>         //файл драйверів клавіатури
//----------------------------
//---------- Головна функція програми
unsigned char Voltage, Current, Regime;

void main(void)
{
InitDevices();           //Ініціалізація портів та LCD
     LCD_LED_ON;              //Вмикання LED - освітлення
індикатора
//---------- Встановлення дистрибутивного режиму роботи
     Regime =VOLTAGE;         //Встановлення дистрибутивного
режиму
Voltage=Voltage_MIN;     //Встановлення дистрибутивних значень
Current=Current_MIN;     //змінних
DisplayTextL1("Voltage");//вивід тексту "Voltage" у 1 рядок
DisplayData(Voltage);    //вивід значення змінної "Voltage"
//----------
while(1)
      {
      key();             //виклик функції драйвера клавіатури
      DelayMs(300);      //затримка узгодження з оператором
      };                 //Нескінченний цикл
}
//-------------------------------------------------------------
```

Файл з функціями драйверів клавіатури key.h має наступний вигляд.

```
//---------- Символічні визначення кнопок
     #define UP    (1<<PB3)   //визначення лінії кнопки UP

     #define DOWN  (1<<PB1)   //визначення лінії кнопки DOWN
     #define MODE  (1<<PB4)   //визначення лінії кнопки MODE
//---------- Символічні визначення констант
```

```
     extern unsigned char Regime;  //змінна поточного режиму
#define VOLTAGE     0          //значення змінної для режиму "U"
#define CURRENT     1          //значення змінної для режиму "I"
//----------
#define   Voltage_MIN    5    //мінімальне  значення «Voltage»
#define   Voltage_MAX    100  //максимальне значення «Voltage»
#define   Voltage_DELTA  5    //величина прирощення  «Voltage»

extern unsigned char Voltage; //змінна режиму «Voltage»
//----------
#define   Current_MIN    10   //мінімальне  значення «Current»
#define   Current_MAX    100  //максимальне значення «Current»
#define   Current_DELTA  10   //величина прирощення  «Current»
extern unsigned char Current; //змінна режиму «Current»
//----------------------------
//---------- Функція опитування клавіатури "UP"
void Key_Up(void)
{
switch (Regime)                    //розгалуження по змінним
     {
     case VOLTAGE:                       //режим VOLTAGE
          {
          if (Voltage < Voltage_MAX)    //перевірка максимуму
             {
     Voltage+=Voltage_DELTA;  //модифікація змінної
     DisplayData(Voltage);         //відображення значення
     };
}; break;
     case CURRENT:                       //режим CURRENT
          {
          if (Current < Current_MAX)    //перевірка максимуму
             {
     Current+=Current_DELTA;  //модифікація змінної
     DisplayData(Current);    //відображення значення
     };
}; break;
     };
}
//----------------------------
//---------- Функція опитування клавіатури "DOWN"
void Key_Down(void)
{
switch (Regime)                    //розгалуження по змінним
     {
     case VOLTAGE:                       //режим VOLTAGE
          {
          if (Voltage > Voltage_MIN)    //перевірка мінімуму
             {
     Voltage -= Voltage_DELTA; //модифікація змінної
     DisplayData(Voltage);    //відображення значення
     };
     }; break;
```

```
          case CURRENT:                        //режим CURRENT
          {
          if (Current > Current_MIN)    //перевірка мінімуму
             {
     Current -= Current_DELTA; //модифікація змінної
     DisplayData(Current);    //відображення значення
     };
          }; break;
     };
}
//----------------------------
//---------- Функція опитування клавіатури "MODE"
void Key_Mode(void)
{
switch (Regime)               //розгалуження по змінним
     {
     case VOLTAGE:                 //режим VOLTAGE, перехід
          {                        //режим CURRENT
          Regime =CURRENT;         //модифікація змінної режиму
          Lcd_Clrscr();            //очистка індикатора
          DisplayTextL1("Current");//вивід тексту "Current"
          DisplayData(Current);    //відображення значення
          }; break;
     case CURRENT:                 //режим CURRENT, перехід
               {                        //режим VOLTAGE
               Regime =VOLTAGE;         //модифікація змінної
режиму
               Lcd_Clrscr();            //очистка індикатора
               DisplayTextL1("Voltage");//вивід тексту
"Voltage"          DisplayData(Voltage);    //відображення
значення
               }; break;
     };
}
//----------------------------
//---------- Функція драйвера клавіатури
void key(void)
{
if (!(PINB & UP) | !(PINB & DOWN) | !(PINB & MODE) )//1
опитув.
        {
   DelayMs(3);//затримка 3 мсек, фільтрація тремтіння
контактів
   if (!(PINB & UP))     {Key_Up();};   //2 опитування кнопок
   if (!(PINB & DOWN))   {Key_Down();}; //та реакції
   if (!(PINB & MODE))   {Key_Mode();};
   };
}
//----------------------------
```

Головна функція драйвера та функції реакцій на натискання кнопок

Головна функція драйвера **void** key(**void**) реалізує опитування клавіатури та виклик відповідних функцій реакцій на натискання кнопок. У функції реалізовано алгоритм фільтрації тремтіння контактів кнопок з подвійним опитуванням та затримкою на прогнозований час тремтіння.

Зміст функцій реакцій визначається функціональними особливостями клавіатури. У розглянутому вище прикладі драйвера використано три кнопки “MODE”, “UP”, “DOWN”. Кнопка “MODE” використана для зміни режиму роботи. Кнопки “UP”, “DOWN” застосовані для зменшення та збільшення значень змінних.

Функція **void** Key_Mode(**void**) забезпечує перехід від однієї змінної до іншої. У прикладі реалізована селекція двох режимів роботи «VOLTAGE» та «CURRENT». Для визначення стану програми використано змінну режиму «Regime». Для відображення стану програми на LCD-індикатор виводяться повідомлення, «Voltage» або «Current».

У кожному із режимів можлива модифікація однієї із двох змінних із заданими кроком та діапазоном можливих значень. Дистрибутивні значення цих змінних задаються в описовій частині програми.

Функції реакції на натискання кнопок **void** Key_Down(**void**) та **void** Key_Up(**void**) використовують спільний алгоритм. У функціях проводиться розгалуження згідно з активним режимом. Потім проводиться перевірка досягнення меж границі діапазону змінної. Якщо змінна знаходиться у межах діапазону, проводиться модифікація її значення.

При визначенні початкового значення змінної, верхнього та нижнього значень меж діапазону та величини прирощення необхідно враховувати те що ці величини пов’язані між собою.

Контрольні питання та завдання

1. Введення статичних цифрових сигналів.
2. Виведення імпульсних цифрових сигналів.
3. Вивід аналогових сигналів.
4. Сформувати константи ініціалізації порта А мікроконтролера ATmega16, для режимів роботи представлених в табл 1.

Таблиця 1

	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
Режим	Прийм.	Перед.	Прийм.	Перед.	Перед.	Прийм.	Прийм.	Перед.
Підтяжка	є		немає			есть	нет	
Стан	x	1	X	0	0	x	x	1

5. Визначити тривалість інтервалу часу, який формується таймером-лічильником TC0 мікроконтролера ATmega16, якщо: частота генератора 10,24 МГц; використовується режим роботи таймера-лічильника “Normal”; коефіцієнт передачі попереднього дільника 1/1024; константа завантаження лічильного регістра TCNT0 = 0xF6 .

6. Визначити коефіцієнт передачі попереднього дільника таймера-лічильника TC1 мікроконтролера ATmega16, який формує затримку, якщо: частота генератора 10,24 МГц;

- використовується режим роботи таймера-лічильника “CTC”;
- тривалість інтервалу часу 1 мс;
- константа завантаження регістра OCR1 = 0x009F
- визначити константу завантаження регістра PORTA, необхідну для виведення на 7-сегментний індикатор символа “5”, якщо: використовується індикатор зі спільним анодом; з’єднання сегментів індикатора з виводами порта наведено в табл. 2.

Таблиця 2

a	b	g	c	d	e	f	H
PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0

СПИСОК ВИКОРИСТАННИХ ДЖЕРЕЛ

1. Схемотехника электронных систем. Том 3. Микропроцессоры и микроконтроллеры / Бойко В. І., Гуржій А. М., Жуйков В. Я., Зорі А. А., Співак В.М., Терещенко Т.О, Петергеря Ю. С. - СПб.: БХВ Петербург, 2004. – 464 с.
2. Мікропроцесорна техніка. Друге видання. Доповнене / Ю. І. Якименко, Т. О. Терещенко, Є. І. Сокол, В. Я. Жуйков, Ю. С. Петергеря. За ред. Т. О. Терещенко. – Київ, 2004. – 440 с.
3. Евстифеев А.В. Микроконтроллеры AVR семейств Tiny и Mega фирмы Atmel – 2-е изд., стер. М.: Издательский дом «Додека –XX1», 2005. – 560с.
4. Голубцов М.С. Микроконтроллеры AVR: от простого к сложному. М.: СОЛОН-Пресс, 2003. – 288с.
5. Гребнев В.В. Микроконтроллеры семейства AVR фирмы Atmel. М.: ИП РадиоСофт, 2002. – 176с.
6. Баранов В.Н. Применение микроконтроллеров AVR: схемы, алгоритмы, программы. М.: Издательский дом «Додека –XX1», 2004. – 288с.
7. Сташин В.В., Урусов А.В., Мологонцева О.Ф. Проектирование цифровых устройств на однокристальных микроконтроллерах. М.: Энергоатомиздат, 1990.
8. Белов А.В. Конструирование устройств на микроконтроллерах Наука и техника, 2005.
9. Белов А. В. Микроконтроллеры AVR в радиолюбительской практике Наука и техника, 2007.
10. Белов А. В. Создаем устройства на микроконтроллерах Наука и техника, 2007.
11. Кравченко А. В. 10 практических устройств на AVR-микроконтроллерах. Книга 1 Додэка-XXI МК-Пресс 2008
12. Кравченко А. В. 10 практических устройств на AVR-микроконтроллерах. Книга 2 КОРОНА-ВЕК МК-Пресс 2009
13. Мортон Дж. Микроконтроллеры AVR. Вводный курс Додэка-XXI 2006
14. Трамперт В. AVR-RISC Микроконтроллеры МК-Пресс 2006
15. www.atmel.com – сайт фірми Atmel.
16. www.atmel.ru - російський сайт фірми Atmel.
17. http://www.fulcrum.ru/cgi-bin/bbs/topic_sel.pl?v=p&FID=1 – конференція по використанню мікроконтролерів AVR фірми Atmel.

Навчальне видання

Тетяна Олександрівна Терещенко
Віктор Агафонович Тодоренко
Лариса Миколаївна Батрак
Юлія Сергіївна Ямненко

Мікропроцесорні пристрої

Навчальний посібник

Головний редактор Наталія Перинська
Технічний редактор Анна Ільченко

Підписано до друку: 15.09.2017
Формат 60x84/16. Гарнітура Times New Roman.
Папір офсетний. Ум.-друк. арк. 14,1. Наклад 300 прим.

Видавництво «КАФЕДРА»
04136, м. Київ, вул. Маршала Гречка 13, оф. 117
e-mail: kafedra.druk@gmail.com
www.kafedra.in.ua
тел.: (093) 521-31-21, (067) 442-98-78

Свідоцтво про внесення суб'єкта видавничої справи
до Державного реєстру видавців, виготівників
і розповсюджувачів видавничої продукції
Серія ДК № 4175 від 20.10.2011 р.

Друкарня «Гордон»
03179, м. Київ, вул. Котельникова, 95
Тел./факс (044) 501-35-69
Свідоцтво про державну реєстрацію
ДК № 1422 від 08.07.2003