МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДВНЗ «Прикарпатський національний університет
імені Василя Стефаника»

Фізико-технічний факультет

Голота В. І.

Курс лекцій з дисципліни

“ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ”

Івано-Франківськ, 2022

УДК 004.42(076.6)
Г2

Рекомендовано до друку Вченою радою фізико-технічного факультету Прикарпатського національного університету імені Василя Стефаника (протокол № 3 від 27 жовтня 2022 року)

Рецензенти:

Когут Ігор Тимофійович, професор, завідувач кафедри комп'ютерної інженерії та електроніки Прикарпатського національного університету імені Василя Стефаника, доктор технічних наук

Яремій Іван Петрович, професор кафедри матеріалознавства і новітніх технологій Прикарпатського національного університету імені Василя Стефаника, доктор фізико-математичних нвук

Г2 Голота В.І. Курс лекцій з дисципліни "Програмування мікроконтролерів": [Електронний ресурс] / *Віктор Іванович Голота* / Фізико-технічний факультет; Прикарпатський національний університет імені Василя Стефаника. – Електронні текстові дані. – Івано-Франківськ, 2022. – 275 с.

У курсі лекцій розглянуто архітектуру мікроконтролерів, структуру блоків із спеціальними функціями та особливості програмування на мові асемблера 8-розрядних мікроконтролерів родини AVR корпорації Microchip. Викладений матеріал відповідає навчальним програмам з дисципліни "Програмування мікроконтролерів".

Курс лекцій з дисципліни "Програмування мікроконтролерів" призначений для студентів галузі знань 12 "Інформаційні технології" спеціальності 123 "Комп'ютерна інженерія" та для студентів галузі знань 17 "Електроніка та телекомунікації" спеціальності 171 "Електроніка".

УДК 004.42(076.6)

ЗМІСТ

ВСТУП

Мікроконтролери (англ. microcontrollers) можна зустріти в багатьох сучасних приладах, таких як телефони, пральні машини, вони відповідають за роботу двигунів і систем гальмування сучасних автомобілів, з їх допомогою створюються системи контролю і системи збору інформації. На основі мікроконтролерів проектують та створюють вимірювальні пристрої, системи керування об'єктами та процесами, вони є основою охоронних, протипожежних систем, домофонів, сигналізацій тощо. Більшість процесорів, що випускаються у світі – мікроконтролери.

Мікроконтро́лер – це однокристальний мікрокомп'ютер, що включає мікропроцесор, оперативну та постійну пам'ять для збереження виконуваного коду програм і даних, порти вводу-виводу і блоки зі спеціальними функціями (лічильники, компаратори, АЦП та інші). Використання однієї мікросхеми значно знижує розміри, енергоспоживання і вартість пристроїв, побудованих на базі мікроконтролерів.

Одними з популярних родин мікроконтролерів, які застосовуються в системах оброблення даних, контролю і управління, є пристрої, що випускаються корпорацією Microchip та відомі під абревіатурою AVR та PIC. Мікроконтролери на ядрі AVR, мають досконалу архітектуру й забезпечують високу швидкодію та низьке енергоспоживання. Порівняно з пристроями PIC, мікроконтролери AVR мають більш розвинену систему команд, що налічує до 133 інструкцій, їхня продуктивність наближається до 1 MIPS/МГц, а flash-пам'ять програм можна внутрішньосхемно програмувати. Архітектура ядра AVR оптимізована так, що дозволяє використовувати як асемблер, так і мову високого рівня Сі. На відміну від пристроїв PIC, де одна операція виконується за 4 такти, мікроконтролери AVR можуть виконувати команди в кожному такті, тобто в 4 рази швидше на тій же тактовій частоті. А що менша частота, то менше енергоспоживання мікроконтролера.

У даному курсі лекцій розглядається програмування на мові асемблера 8-розрядних мікроконтролерів родини AVR. На даний час широко використовуються три родини мікроконтролерів AVR – “tiny”, “mega”, “XMega”, які відрізняються обсягами Flash-, EEPROM- і SRAM-пам'яті, набором периферійних пристроїв і побудовою схеми тактування.

У курсі лекцій розглянуто такі питання: характеристики мікроконтролерів родин ARM, MSP, AVR; архітектура AVR; асемблер AVR; стек і переривання; паралельні порти; Flash-, EEPROM- і SRAM-пам'яті; UASRT; SPI; TWI; аналоговий компаратор і ADC. У кінці кожної лекції наведено контрольні питання.

Курс лекцій з дисципліни “Програмування мікроконтролерів” розроблено для студентів галузі знань 12 “Інформаційні технології” спеціальності 123 “Комп’ютерна інженерія” та для студентів галузі знань 17 “Електроніка” спеціальності 123 “Електроніка”.

Курс лекцій може бути корисний студентам відповідних спеціальностей під час вивчення дисциплін, які пов’язано із проектуванням та використанням мікроконтролерів, а також при виконанні курсових проектів, бакалаврських робіт та магістерських дисертацій, в яких використовуються мікроконтролери.

ЛЕКЦІЯ 1. Мікроконтролери. Характеристики

Мета. Огляд основних поширених родин мікроконтролерів, їх характеристик і функціональних блоків.

Вступ. Однокристальні мікроконтролери знаходять широке застосування як у побутових пристроях (фотоапарати, телефони, друкарки, пральні машини), так і в складних технічних системах (системи збору та контролю інформації, телекомунікація, авіоніка, мехатроніка, автоніка, робототехніка). Мікроконтролери на базі стандартних індустріальних архітектур випускаються багатьма фірмами. На даний час найбільш поширені мікроконтролери марок PIC, AVR (Microchip), MSP (Texas Instruments), STM (STMicroelectronics).

План.

1. Основні відмінності між МП і МК
2. Типи мікроконтролерів
3. Організація доступу до пам'яті
4. Системи команд – CISC та RISC
5. Родина МК ARM
   5.1. Родина STM32
   5.2. Аналогова периферія МК STM32G4
   5.3. Засоби розроблення та налагодження
6. Родина МК MSP
   6.1. Ключові характеристики МК MSP
   6.2. Область застосування МК MSP
   6.3. Серії МК MSP
   6.4. Програмні інструменти розроблення
   6.5. Апаратні інструменти розроблення
7. Родина МК AVR
   7.1. Особливості МК AVR XMEGA
   7.2. Особливості МК MEGA AVR
   7.3. Особливості МК TINY AVR
   7.4. Нова серія AVR
   7.5. Програмні засоби розроблення

1. Основні відмінності між МП і МК

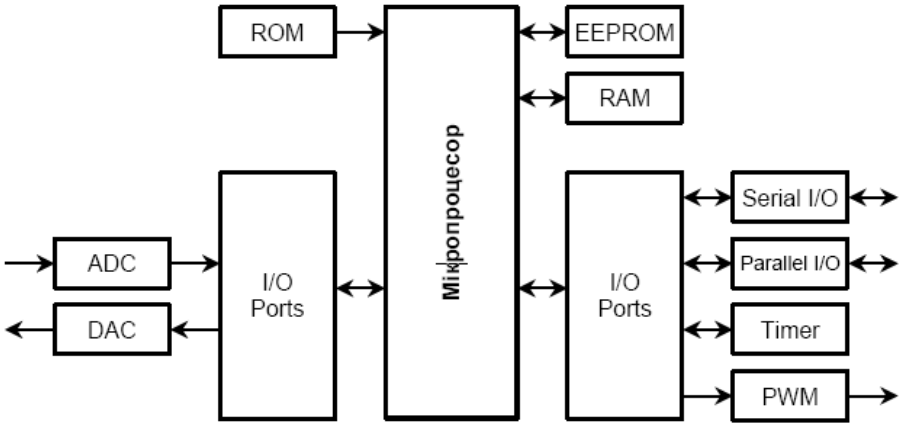
Мікропроцесор (МП) – програмований пристрій, що приймає двійкові дані від пристрою вводу, обробляє ці дані відповідно до інструкцій, збережених у пам'яті, і видає результати на вихід. Іншими словами, МП виконує програму, збережену в пам'яті, та передає дані від та до зовнішнього світу через порти введення/виведення. Загалом, МП називають серцем будь-якої мікропроцесорної системи, яке виконує усі операції, а також контролює решту системи.

Мікроконтролер (МК) – можна розглядати як спеціалізований комп'ютер-на-кристалі або одно-кристальний комп'ютер. Слово «мікро» натякає, що пристрій є маленький, а слово «контролер» – що пристрій може використовуватися для контролю однієї чи більше функцій об'єктів, процесів чи подій. МК також називають вбудованими (embedded) контролерами, оскільки МК часто вбудовані у пристрої та системи, які вони контролюють.

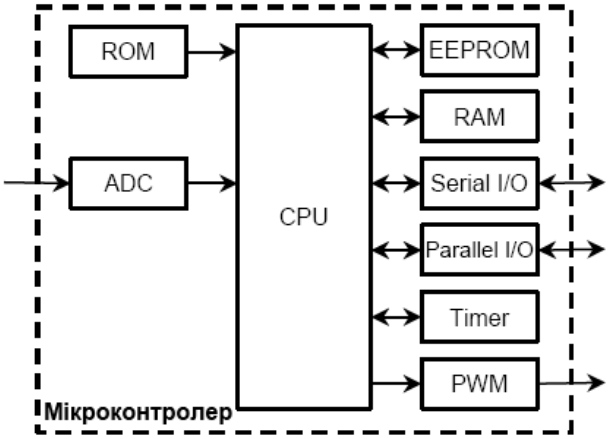
МК містить спрощений процесор, пам'ять (RAM та ROM), порти введення/виведення, периферійні пристрої – лічильники/таймери, аналого-цифрові перетворювачі і т. п., і все це інтегровано в один кристал. Ця особливість і відрізняє МК від мікропроцесорних систем. Натомість, МП має лише процесор із регістрами загального призначення, а периферійні пристрої розміщені зовні. Якщо мікропроцесорна система є системою загального призначення,

що може бути запрограмованою для виконання великого числа різних функцій, то МК призначені для однієї задачі і виконують лише одну конкретну програму. Ця програма збережена в ROM і, як правило, не змінюється.

На рис. 1.1 показана основна відмінність між мікропроцесорними системами та МК: мікропроцесорні системи для своєї функціональності потребують додаткових мікросхем, в той час як у МК функції усіх цих додаткових мікросхем є інтегровані у тому самому кристалі МК.



а)



б)

Рисунок 1.1 – Структура: а) МП; б) МК

При описанні МК в англомовній літературі використовуються наступні абревіатури і скорочення:

CPU (Central Processing Unit) – центральний процесор.

RAM (Random Access Memory) – пам’ять з довільним доступом, чи оперативний запам’ятовуючий пристрій (ОЗП). RAM призначена для зберігання проміжних результатів та інших тимчасових даних протягом виконання програми.

ROM (Read Only Memory) – пам’ять тільки для читання, чи постійний запам’ятовуючий пристрій (ПЗП). ROM зберігає програмні інструкції (програму) та таблиці з довідковими даними. У сучасних МК вона реалізована як flesh-пам’ять.

EEPROM (Electrically Erasable Programmable ROM) – енергонезалежна постійна пам’ять або запам’ятовувальний пристрій, який програмується і очищається за допомогою електрики. Одним із різновидів EEPROM є flash-пам’ять. Основна властивість EEPROM пам’яті – це

можливість вибіркового програмування окремих байтів, на відміну від поблокового програмування flash-пам'яті програм.

I/O Ports (input/output) – паралельні порти введення/виведення надають інтерфейс між МК та периферійними пристроями введення/виведення, такими як: клавіатура, дисплей тощо. Виводи портів можуть бути як двонаправленими, так і однонаправленими, або лише на вхід, або на вихід.

Serial I/O – послідовні порти для обміну даними, що реалізують асинхронні (напр., UART) або синхронні (напр., SPI) інтерфейси обміну даними. Асинхронний інтерфейс використовує протокол зі стартовим та стоповим бітами для передачі та прийому. Стартовий та стоповий біти вбудовані у кожен байт даних. Синхронні інтерфейси використовують синхронізуючі імпульси для кожного біта.

Timer/Counter (таймер/лічильник) – використовують для відліку часу або/та меж часових інтервалів між подіями, підрахунку кількості подій та генерації швидкості передачі даних (у бодах) для послідовних портів. Таймери також можуть керувати певними I/O-виводами у МК, наприклад, здійснювати підрахунок кількості імпульсів, що поступають на його вхід, чи навпаки, виводити певні послідовності імпульсів.

PWM (Pulse Width Modulation, широтно-імпульсна модуляція ШІМ) – це спосіб кодування аналогового сигналу шляхом зміни ширини (тривалості) прямокутних імпульсів несучої частоти. Найбільш часто PWM використовують для керування моторами різних типів та активним навантаженням, наприклад, лампою розжарювання.

ADC (Analog to Digital Converter) – аналого-цифровий перетворювач забезпечує інтерфейс для роботи з аналоговими пристроями, наприклад, з давачами, що видають аналогові електричні еквіваленти для фактичних фізичних параметрів, які ми хочемо контролювати.

DAC (Digital to Analog Converter) – цифро-аналоговий перетворювач забезпечує інтерфейс для роботи з виконавчими пристроями.

2. Типи мікроконтролерів

Існує багато ознак класифікації МК. Найбільш поширеною ознакою є розрядність МК:

• Вбудовані *8-розрядні* МК.

Мають просту систему команд та велику номенклатуру вбудованих пристроїв. Причиною життєздатності 8-розрядних МК є використання їх для керування реальними об'єктами, де застосовуються, в основному, алгоритми з переважанням логічних операцій, швидкість обробки яких практично не залежить від розрядності процесора.

Приклади: 8051 чи MCS-51 (Intel); ATtiny/ATmega/ATXmega, PICI2/l6/l8, ZilogZ86 (Microchip+ATmel); 68HC05, 68HC08, 68HC11 (Motorola); ST6, ST7, ST9, STM8L (STMicroelectronics); COP8 (National Semiconductor); TMS370 (Texas Instruments) та інші.

• 16-розрядні МК.

Структури та системи команд зорієнтовані на прискорену реакцію на зовнішні події, і, відповідно, на таких МК можуть будуватися системи реального часу середньої продуктивності. Приклади: MCS-96 (Intel); PIC24 (Microchip); MSP430 (Texas Instruments); 68HC16 (Motorola); MB90 (Fujitsu); M16C (Mitsubishi) та ін.

• 32-розрядні МК.

Велика частина з них побудовані на ядрі ARM та призначені для систем телефонії, оброблення та передачі інформації, телебачення. Приклади: STM32 (STMicroelectronics); Sitara

ARM (Texas Instruments); HT32F125x (Holtek); LPC3000, LPC2900 (NXP); PIC32Mx, SAM (Arm Cortex) (Microchip) та ін.;

• Цифрові сигнальні процесори (DSP, Digital Signal Processor).

Використовуються для складного математичного оброблення аналогових сигналів у режимі реального часу. Застосовуються у телефонії та зв'язку.

Приклади: C5000, C6000 (Texas Instruments); ADSP-21xx, SigmaDSP (Analog Devices), MSC81xx (Freescale).

3. Організація доступу до пам'яті

Існує дві фундаментальні архітектури, що використовуються процесорами для доступу до пам'яті: архітектура Фон-Неймана (Von Neumann architecture), відома ще як Принстонська, та Гарвардська архітектура (Harvard architecture).

Архітектура Фон-Неймана використовує одну пам'ять для зберігання як програмних інструкцій (команд), так і даних. В наявності є лише одна загальна шина для адрес та даних між процесором та пам'яттю (рис. 1.2). Команди та дані витягуються у послідовному порядку, таким чином обмежуючи швидкість передачі даних чи пропускну здатність.

Рисунок 1.2 – Архітектура Фон-Неймана

Гарвардська архітектура використовує фізично розділену пам'ять для програм та даних. Це, відповідно, вимагає окремих шин для програми та окремо для даних (рис. 1.3).

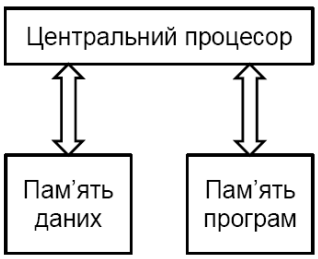


Рисунок 1.3 – Гарвардська архітектура

У такій архітектурі команди та операнди можуть бути видобуті одночасно з пам'яті, що робить МК з такою архітектурою більш швидшими у порівнянні з МК, що використовують архітектуру Фон-Неймана. Також шина даних та шина програм можуть мати різну розрядність, що дає можливості для кращої оптимізації пам'яті даних та пам'яті програм у відповідності до архітектурних вимог.

Переваги архітектури Фон-Неймана полягають у спрощеній схемотехніці процесора та у гнучкості розподілу ресурсів між областями пам’яті, що дуже важливо в операційних системах реального часу. Архітектура Фон-Неймана по сьогодні є основною для ЕОМ загального призначення.

Гарвардська архітектура широко застосовується в спеціалізованих обчислювачах, зокрема в МК та цифрових сигнальних процесорах, де необхідний високоінтенсивний обмін даними. Також за гарвардською архітектурою зазвичай організується кеш-пам'ять в ЕОМ загального призначення, яка розділяється окремо на кеш-пам'ять команд та кеш-пам'ять даних (але, точніше, це стосується внутрішньої організації процесора, а не архітектури ЕОМ).

4. Системи команд – CISC та RISC

CISC (Complex Instruction Set Computer) – процесор з повним набором команд. Такі процесори мають у своєму складі велику кількість різноманітних команд, які, в свою чергу, можуть відрізнятися форматом та довжиною. Прості команди можуть бути виражені за допомогою короткого командного коду розміром в один байт, який виконується дуже швидко. Складні команди можуть складатися з декількох байт коду та займати багато часу на виконання. При цьому виконання певної як завгодно складної команди із системи команд процесора реалізовується апаратно усередині самого процесора. У систему команд CISC-процесора може входити, наприклад, обчислення квадратного кореня, що потребує багато десятків тактів. Додавання кожної нової команди призводить до збільшення загального числа транзисторів у процесорі. CISC-процесори також характеризуються великою кількістю методів адресації пам’яті та порівняно невеликою кількістю робочих регістрів. На практиці рідко використовується увесь перелік команд CISC-процесора, як правило не більше 20%.

Типовими прикладами CISC-архітектур були системи VAX, PDP-11, IBM System/360, родини МК Motorola 68000 та Intel x86. Однак з появою високорівневих мов та оптимізувальних компіляторів, розвитком електроніки, який спричинив здешевлення комп’ютерної пам’яті, виявилось, що використання високорівневих машинних команд суттєво обмежує можливості до оптимізації програми, підвищення її швидкодії.

RISC (Reduced Instruction Set Computer) – процесор зі скороченим набором команд. У процесорах з такою архітектурою використовується обмежений набір швидких команд з фіксованою шириною. Кожна команда повинна виконуватися, в ідеалі, за один машинний цикл, тому в системі команд може бути відсутнє ділення. У таких процесорах міститься менше число транзисторів, що знижує їхню вартість та енергоспоживання. При цьому, як правило, підвищується їхня продуктивність. Але, там де CISC-процесор виконував одну команду, для RISC-процесора необхідно писати невелику програму. RISC-процесори характеризується збільшеною кількістю регістрів загального призначення та простими способами адресації пам’яті.

Майже усі сучасні МК є побудовані за RISC-архітектурою. До МК із CISC-архітектурою відносяться МК фірм ARC, Alfa, ARM, Microchip, i860, i960, MIPS, RISC-V, SPARC, Intel з ядром MCS-51, які на сьогодні за ліцензією ще випускаються цілим рядом виробників. Істинними CISC-процесорами вважалися процесори ПК з архітектурою x86, але починаючи з Intel 486DX, вони є CISC-процесорами з RISC-ядром. Вони безпосередньо перед виконанням перетворюють CISC-інструкції x86-процесорів у більш простий набір внутрішніх інструкцій RISC.

5. Родина МК ARM

Процесори ARM є ключовим компонентом для великої кількості 32-бітових вбудовуваних систем. Процесори ARM широко використовуються в мобільних телефонах, планшетах та інших портативних пристроях. ARM основані на RISC-архітектурі, що дозволяє зменшити споживання енергії процесором і таким чином використовувати у вбудовуваних системах. Хоча ARM основані на RISC-архітектурі, вони не повністю повторюють принципи побудови таких систем. Для того, щоб ARM були більш пристосовані для використання у вбудовуваних системах, прийняті наступні відхилення від принципів RISC:

1. Змінна кількість циклів виконання для простих інструкцій. Прості інструкції ARM можуть виконуватися більш ніж за один цикл. Наприклад, виконання інструкцій Load і Save залежить від кількості переданих їм регістрів.

2. Можливість об'єднувати команди простого і циклічного зсуву з командами оброблення інформації.

3. Умовне виконання – інструкція виконується тільки у тому випадку, якщо виконується конкретна умова. Це збільшує продуктивність і дозволяє позбутися від операторів галуження.

4. Покращені інструкції – процесори ARM підтримують покращені DSP-інструкції для операцій з цифровими сигналами.

Програміст може розглядати ядро ARM як набір функціональних блоків – ALU, MMU та ін., – з'єднаних шиною даних. Дані поступають в процесор через шину даних. Декодер інструкцій обробляє інструкції перед їх виконанням. ARM можуть працювати тільки з даними, які записані в регістрах. ALU зчитує дані з регістрів, виконує необхідні операції і записує результат назад в регістр, звідки його можна записати у зовнішню пам'ять. Процесори ARM містять до 18 регістрів: 16 регістрів даних і 2 регістри процесів. Всі регістри 32-розрядні і іменуються від R0 до R15. Регістри R13, R14, R15 використовуються для виконання визначених специфічних задач: R13 – вказівник стеку, R14 – зв'язуючий регістр, R15 – лічильник.

В залежності від контексту ці регістри можуть використовуватися як регістри загального призначення. Також є два програмних регістра CPSR (Current Program Status Register) і SPSR (Saved Program Status Register), які використовуються для збереження стану процесора і програми.

Одними із серії процесорів для вбудовуваних систем, є процесори, основані на архітектурі ARM Cortex-M4. Вони мають засоби цифрової обробки сигналів (Digital Signal Processing, DSP). Мікроконтролери, основані на базі ARM Cortex-M4 мають наступні внутрішні модулі (рис. 1.4). Процесор ARM Cortex-M4 використовує МК STM32F407VG. Як видно з рис. 1.4, МК STM32F407VG має достатньо периферії і може використовуватися для розв'язування багатьох практичних задач різної направленості.

Рисунок 1.4 – Модулі МК STM32F407VG

5.1. Родина МК STM32

Родина МК STM32 побудована з використанням 32-розрядного ядра Cortex різних версій (в МК на платі STM32F407 Discovery використовується ядро Cortex-M4). Деякі основні характеристики серії МК STM32 показані в табл. 1.1.

Таблиця 1.1 – Основні характеристики ядра МК серії МК STM32

Характеристика	**Значення**
Ширина слів для даних	32
Архітектура	Гарвардська
Набір інструкцій	RISC
Організація пам’яті програм	32
Буфер попередньої вибірки	2x64
Середній розмір інструкції, байт	2
Тип переривань	векторизований
Затримка реагування на переривання	12 циклів
Режими енергозбереження	сон, сон із виходом, глибокий сон
Інтерфейс налагодження	ST-Link, JTAG

МК STM32 побудовані за гарвардською архітектурою і мають 3-ступеневий конвеєр, який мінімізує час виконання команд. З їх використанням можна будувати системи з декількома режимами керування енергоспоживанням. В них використовуються внутрішні інтерфейси пам'яті шириною більше, ніж середня довжина інструкції. Це мінімізує число доступів до шини пам'яті і відповідно споживання електроенергії, яке пов'язане з шинними операціями і читанням енергонезалежної пам'яті. Технологія безперервного оброблення переривань з виключенням внутрішніх операцій із стеком скорочує час реакції на переривання і виключає зайві операції із стеком.

На рис. 1.5 показано спрощене подання цифрового периферійного пристрою.

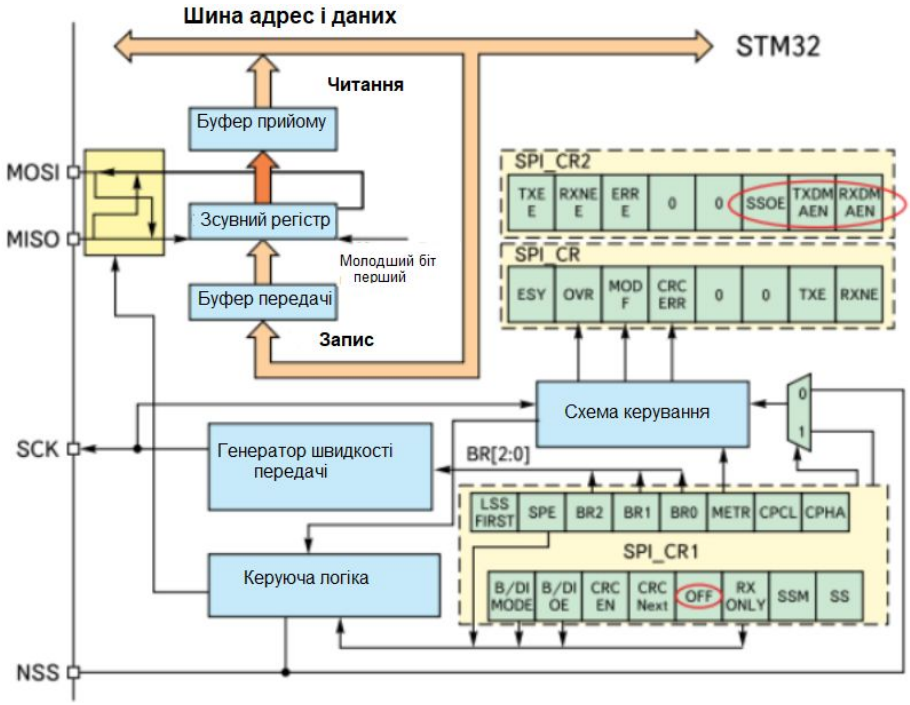


Рисунок 1.5 – Цифровий периферійний пристрій

Периферійний вузол може бути розділений на два головних блоки. Перший блок – це ядро, яке містить скінчені автомати, лічильники та іншу комбінаторну або послідовнісну логіку. Ядро призначене для виконання задач без участі процесора, таких як прості задачі передачі даних, керування аналоговими входами або виконання функцій, прив'язаних до синхросигналів. Ядро периферійного вузла зв'язується із зовнішнім середовищем через порти введення/виведення МК. Зовнішні з'єднання можуть складатися з декількох сигналів або складних шин. Другий блок – налаштування і керування периферією, які здійснює застосунок через регістри, з'єднані із внутрішньою шиною, яка спільно використовується з іншими ресурсами МК.

Серія МК STM32G4

Одним із відомих виробників ARM мікроконтролерів є фірма STMicroelectronics. STM випустила перший мікроконтролер STM32F3 із змішаними сигналами на базі процесорного ядра Cortex-M4. Ці мікроконтролери містять у своєму складі великий набір аналогової периферії: 12-бітові АЦП, 12-бітові ЦАП, операційні підсилювачі з програмованими коефіцієнтами підсилення, компаратори. Крім того, STM32F3 здатні ефективно виконувати цифрову обробку сигналів завдяки підтримці DSP-команд та операцій чисел з плаваючою

крапкою. На даний час випускається новіший мікроконтролер STM32G4 із змішаними сигналами у трьох серіях:

- STM32G4x1 Access line – початкова серія з базовим набором периферії;
- STM32G4x3 Performance line – серія з розширеним набором аналогової і цифрової периферії;
- STM32G4x4 Hi-resolution line – серія з розширеним набором периферії і таймером підвищеної роздільної здатності HRTIM.

Серія STM32G4x1 Access line

Всі мікроконтролери STM32G4 побудовані на базі процесорного ядра Cortex-M4 з максимальною робочою частотою 170 МГц і продуктивністю до 213 DMIPS і 550 CoreMark. Це більш ніж в два рази вище, ніж у STM32F3 (90 DMIPS і 245 CoreMark відповідно). З одного боку, приріст продуктивності досягається за рахунок збільшення тактової частоти, що в свою чергу стало можливим завдяки переходу на новий технологічний процес 90 нм. З іншого боку, пропорційне збільшення продуктивності при виконанні коду з Flash пам’яті забезпечується вбудованим апаратним прискорювачем ART.

Як відомо, Flash-пам'ять є відносно повільної. Для того щоб узгодити швидкодіючий процесор і повільну пам’ять на високих частотах, доводиться використовувати цикли очікування, через що фактична продуктивність мікроконтролера не може збільшуватися пропорційно частоті. Саме цю проблему вирішує апаратний прискорювач ART. Прискорювач заздалегідь зчитує дані і команди з Flash пам’яті, використовуючи власну кеш-пам'ять. Якщо в кеш-пам’яті прискорювача є необхідні дані, то процесору не потрібно виконувати цикли очікування і втрачати час даремно.

Варто зазначити, що існує й альтернативний спосіб підтримки продуктивності на підвищених частотах за рахунок виконання програми з ОЗП (SRAM). У випадку з STM32G4 таке рішення забезпечить стабільну продуктивність близько 2,4 CoreMark / МГц.. При виконанні коду з SRAM пікова продуктивність 3,2 CoreMark / МГц принципово не може бути досягнута, так як процесорному ядру доводиться паралельно зчитувати дані і команди з одного джерела (SRAM). Для застосунків реального часу вирішити проблему можна за допомогою спеціального ОЗП – CCM-SRAM (Core-Coupled Memory). Зазвичай код програми розміщують в CCM-SRAM, а дані розміщують у звичайній SRAM, що дозволяє ядру не втрачати часу і паралельно використовувати інтерфейс даних (D-bus) і інтерфейс команд (I-bus). У STM32G4 є вбудована CCM-SRAM. При виконанні коду з CCM-SRAM продуктивність виявляється максимальною. Також варто відзначити, що висока ефективність обчислень STM32G4 забезпечується за рахунок апаратних математичних співпроцесорів CORDIC і FMAC.

У випадку з STM32G4x1 Access line обсяг Flash досягає 128 кбайт (1 банк), обсяг CCM-SRAM складає 10 кбайт, а сумарний обсяг SRAM (звичайна SRAM + CCM-SRAM) – 32 кбайт. У найближчому майбутньому STMicroelectronics планує почати випуск моделей з 512 кбайт Flash і 112 кбайт SRAM.

Склад системної периферії мікроконтролерів STM32G4 можна вважати стандартним для сучасних родин STM32: вбудований супервізор живлення з функціями POR/PD/RBOR/PVD, вбудовані RC-резонатори LSI 32 кГц (точність 5%) і HSI 16 МГц (точність 1%), ФАПЧ (PLL), система скидання і тактування RCC з можливістю індивідуального налаштування тактування периферії, годинник реального часу RTC, системний 24-бітовий декрементний таймер SysTick,

пара сторожових таймерів (віконний WWDG і незалежний IWDG), блок розрахунку контрольної суми CRC. Крім того, варто відзначити можливість очікувального живлення мікроконтролера від батарейки (VBAT).

Мікроконтролери STM32G4x1 Access line отримали багатий набір таймерів, в тому числі відразу два таймери з комплементарними виходами для управління трифазними електродвигунами або трифазними імпульсними перетворювачами. Крім того, у STM32G4x1 є набір звичних 32-бітових і 16-бітових таймерів загального призначення і базових таймерів, а також 16-бітовий таймер для режимів зниженого споживання LPTIM.

Інтерфейсна частина периферії STM32G4x1 Access line вибиралася, виходячи з вимог сучасного ринку. Крім обов'язкових послідовних інтерфейсів SPI/I2C/USART/SAI, мікроконтролери мають на борту FDCAN, USB FS і USB Type-C™/UCPD. Головною перевагою серії STM32G4x1 Access line є потужний набір аналогової периферії:

• два 12-бітові АЦП послідовного наближення з числом каналів, що досягає 23, швидкістю перетворення до 4 MSPS і можливістю збільшення роздільної здатності до 16 біт за рахунок апаратної передискретизації;

• чотири 12-бітові ЦАП: два з зовнішніми виходами (і вихідним буфером) з швидкодією до 1 MSPS і два без зовнішніх виходів з швидкодією 15 MSPS;

• чотири швидкодіючі аналогові rail-to-rail-компаратори;

• три операційні підсилювачі з програмованим коефіцієнтом підсилення;

• джерело опорної напруги VREFBUF з трьома вихідними напругами 2,048/2,5/2,95 В.

Серія STM32G4x3 Performance line

Серія STM32G4x3 відрізняється від базової серії родини STM32G4x1 обсягом вбудованої пам’яті, складом і кількістю периферійних блоків.

Серед важливих відмінностей варто відзначити появу інтерфейсів для підключення зовнішньої пам'яті – FSMC і QuadSPI. Це корисно для застосунків, в яких не вистачає вбудованої пам’яті мікроконтролера. При цьому обсяги вбудованої пам'яті також зросли до 512 кбайт FLash і до 128 кбайт SRAM (в тому числі 32 кбайт CCM-SRAM). Кількісні зміни торкнулися числа комунікаційних інтерфейсів, таймерів і аналогових блоків. У складі STM32G4x3 Performance line присутні до трьох модулів FDCAN, до трьох таймерів з трифазними комплементарними виходами, до 5 АЦП, до 7 ЦАП, до 7 компараторів, до 6 ОП.

Серія STM32G4x4 Hi-resolution line

Дана серія відрізняється від мікроконтролерів STM32G4x3 наявністю таймера з підвищеною роздільною здатністю HRTIM.

Ключові відмінності між серіями родини STM32G4 показані в табл. 1.2.

Таблиця 1.2 – Серії родини STM32G4

Параметр	STM32G474 Hi-Resolution line	STM32G473 Performance line	STM32G431 Access line
Процесорне ядро, робоча частота, МГц	Arm Cortex-M4, 170		
Flash макс., кбайт	512 (2×256 dual bank)		128 (single bank)

RAM макс., кбайт	96		22
CCM – SRAM (code-SRAM), кбайт	32		10
12-бітовий АЦП (SAR)	до 5 (4 MSPS)		до 2 (4 MSPS)
Компаратори	7		4
ОП з програмованим коефіцієнтом підсилення (точність 1%)	6		3
12-бітовий ЦАП	7		4
Таймер для керування двигуном (170 МГц)	3		2
CAN-FD	3		1
FSMC	1	1	–
12-канальний таймер підвищеної роздільної здатності (Hi-resolution Timer)	1	–	–
Живлення, В	1,72…3,6		

5.2. Аналогова периферія МК STM32G4

АЦП

Мікроконтролери STM32G4 можуть мати до п'яти аналого-цифрових перетворювачів послідовного наближення (SAR) з числом вхідних каналів до 42. Розрядність АЦП становить 12 біт, однак за допомогою вбудованого блоку передискретизації роздільну здатність можна збільшити до 16 біт. АЦП здатні працювати з зовнішніми і внутрішніми сигналами (давач температури, опорна напруга VREF, напруга живлення VDDA, виходи ОП). Вбудовані АЦП STM32G4 підтримують традиційні режими роботи: режим з одиночним перетворенням, режими з циклічним або одноразовим вимірюванням вибраних каналів, а також режим з вимірюванням інжектованих каналів. Швидкість перетворення АЦП залежить від частоти тактування, тривалості стадії вибірки і розрядності. При використанні розрядності 12 біт максимальна частота вимірювань становить 4 MSPS. У режимі Dual mode два АЦП по черзі вимірюють напругу на одному каналі, що дозволяє збільшити швидкість роботи до 7,5 MSPS. Варто відзначити, що АЦП дозволяє автоматично виконувати зсув результату вимірювання і коректування зміщення. Основні метрологічні характеристики вбудованих АЦП подані в таблиці 1.3.

Таблиця 1.3 – Основні характеристики вбудованих АЦП

Параметр	Значення
Кількість АЦП	до 5
Тип АЦП	послідовного наближення
Число каналів	до 42
Роздільна здатність	12/10/8/6
Роздільна здатність з апаратною передискретизацією	16

Максимальна швидкодія	4 MSPS
Програмний зсув	до 8 біт
Компенсація підсилення	є
Компенсація зміщення	є
Діапазон вимірюваних напруг	0…VDDA (3,6 В)
Максимальний вхідний опір джерела сигналу, кОм	50
Вбудований конденсатор схеми вибірки зберігання, типів, пФ	5
Час підготовки до вимірювань	1 цикл
Час калібрування, типів., мкс	1,93
Час перетворення 12-біт (з врахуванням вибірки), мкс	0,256…10,883
Споживання від VDDA, макс., мкА	730 мкА

ЦАП

Мікроконтролери STM32G4 можуть мати до 7 ЦАП. Існують два види вбудованих ЦАП:

• повноцінні 12-бітові ЦАП з вихідним буфером і частотою перетворення до 1 MSPS;

• 12-бітові ЦАП без зовнішніх виходів і з частотою перетворення до 15 MSPS.

Якщо перший тип ЦАП можна вважати класичним, то другий виконує службові функції і використовується, наприклад, для задання граничних напруг спрацьовування вбудованих аналогових компараторів. ЦАП здатні виступати як генератори шуму і трикутних імпульсів. При необхідності виходи ЦАП можна синхронізувати для роботи в режимі Dual mode. Основні метрологічні та інші характеристики вбудованих ЦАП подані в табл. 1.4.

Таблиця 1.4 – Основні характеристики ЦАП

Параметр	Значення
Кількість ЦАП	до 7
Роздільна здатність	12 або 8
Швидкодія вихідних каналів	1 MSPS
Швидкодія внутрішніх каналів	15 MSPS
Вихідна напруга з активним буфером	0…Vref — 0,2 В
Вихідна напруга без буфера	0…Vref
Вихідний імпеданс АЦП (без буферу), кОм	9,6…13,8
Типовий опір навантаження (при активному буфері), кОм	до 25
Коефіцієнт подавлення нестабільності по живленню PSRR, дБ	-80 (тип)
Споживання від VDDA, макс., мкА	670
DNL	±2 LSB
INL	±2 LSB
Гарантована монотонність, біт	10
Зміщення при коді 0x800	±12 LSB (3,6 В)

Зміщення при коді 0x800	±5 LSB (3,6 В)
SNR, дБ	71,2 (навантаження >5 кОм)
THD, дБ	-78 (навантаження >5 кОм)

ОП

У мікроконтролерів STM32G4 може бути до шести вбудованих rail-to-rail-операційних підсилювачів з програмованим коефіцієнтом підсилення 2, 4, 8, 16, -1, -3, -7, -15 (1%) 32, 64, -31, -63 (2%). ОП можуть використовуватися самостійно або в зв'язці з іншими аналоговими блоками (АЦП і ЦАП).

Основні метрологічні та інші характеристики вбудованих ОП подані в табл. 1.5.

Таблиця 1.5 – Характеристики вбудованих ОП

Параметр	Значення
Кількість ОП	До 6
Діапазон вхідних напруг	0…VDDA (3,6 В)
Програмне підсилення	2…64 (неінвертуючі), -1…63 (інвертуючі)
Максимальне вхідне початкове зміщення, мВ	±1,5
Максимальне вхідне зміщення, мВ	±3
Температурний дрейф вхідного зміщення, мкВ/°С	±10
Вихідний струм (без PGA), мкА	500
Вихідний струм (з PGA), мкА	270
Ємнісне навантаження, пФ	50
CMRR, дБ	60
PSRR, дБ	80
GBW, МГц	13
АО, дБ	95

Компаратори

У складі мікроконтролерів STM32G4 може бути до 7 вбудованих компараторів. Компаратори використовуються для порівняння пари зовнішніх сигналів або для порівняння зовнішнього сигналу з внутрішнім опорним сигналом VREF і його похідними (1/2 VREF, 1/4 VREF 3/4 VREF). Як опорний сигнал також можуть виступати сигнали з ЦАП. Гістерезис компараторів встановлюється програмно в діапазоні 0 ... 63 мВ. Характеристики вбудованих компараторів подані в табл. 1.6.

Таблиця 1.6 – Характеристики вбудованих компараторів

Параметр	Значення
Кількість компараторів	До 7
Діапазон вхідних напруг	0…VDDA (3,6 В)

Програмований гістерезис, мВ	0/9/18/27/36/45/54/63

Математичні прискорювачі

Процесорне ядро Cortex-M4F має вбудований блок операцій чисел з плаваючою крапкою (FPU) і підтримує DSP-інструкції. Це дозволяє мікроконтролеру виконувати цифрову обробку сигналів. Крім того, для досягнення максимальної продуктивності на високих частотах в мікроконтролерах STM32G4 використовується прискорювач ART і пам'ять CCM-SRAM.

Для підвищення ефективності цифрової обробки сигналів пропонуються два математичних прискорювачі: CORDIC і FMAC. Ці обчислювальні блоки здатні розвантажити основний процесор і забезпечити швидку обробку даних в реальному часі. CORDIC є апаратний обчислювальний блок, що дозволяє розраховувати найбільш використовувані в цифровій обробці сигналів (ЦОС) функції (табл. 1.7). Функції мають один або два аргументи. Обчислення ведуться в форматі чисел з фіксованою крапкою q1.31 або q1.15.

Таблиця 1.7 – Функції підтримувані в CORDIC

Функція	Перший аргумент (ARG1)	Другий аргумент (ARG2)	Основний результат (RES1)	Додатковий результат (RES2)
Косинус	кут θ	модуль m	$m \times \cos\theta$	$m \times \sin\theta$
Синус	кут θ	модуль m	$m \times \sin\theta$	$m \times \cos\theta$
Фаза	x	y	atan2(y, x)	$\sqrt{x^2+y^2}$
Модуль	x	y	$\sqrt{x^2+y^2}$	atan2(y,x)
Арктангенс	x	–	$\tan^{-1} x$	–
Гіперболічний косинус	x	–	cosh x	sinh x
Гіперболічний синус	x	–	sinh x	cosh x
Гіперболічний арктангенс	x	–	$\tanh^{-1} x$	–
Натуральний логарифм	x	–	ln x	–
Квадратний корінь	x	–	$\sqrt{x}$	–

Важливим аспектом при виконанні ЦОС є точність обчислень. У випадку з CORDIC точність залежить від числа виконуваних ітерацій і типу розраховуваної функції. За один такт CORDIC виконує 4 ітерації. Для отримання максимальної точності при добуванні квадратного кореня потрібно 3 такти, а для інших функцій – 6 тактів.

FMAC (Filter Math Accelerator) є математичним прискорювачем, що дозволяє виконувати математичні операції з масивами чисел. У складі FMAC є апаратний помножувач 16×16, 24-бітовий акумулятор (додавання і віднімання), власна пам'ять 256×16 біт. Крім того, в пам'яті можуть бути визначені буфери (два вхідних і один вихідний). За допомогою FMAC можна реалізувати фільтри з нескінченною і скінченною імпульсною характеристикою (HIX, CIX фільтри), а також виконувати операції згортки і кореляції.

Таким чином, у мікроконтролерів STM32G4 є всі необхідні інструменти не тільки для

збору аналогових даних, але і для їх подальшої обробки в режимі реального часу.

Вбудовані таймери

Представники родини STM32G4 мають великий набір таймерів (табл. 1.8). Це має велике значення для систем зі змішаними сигналами, так як саме за допомогою таймерів звичайно реалізується управління електродвигунами та інверторами.

Таблиця 1.8 – Вбудовані таймери мікроконтролерів STM32G4

Тип таймеру	Таймери	Роздільна здатність, біт	Рахунок	Дільник	DMA	Канал-ів	Компле-ментарні виходи
З підвищеною роздільною здатністю	HRTIM	16	вверх	1/2/4 (x2, x4, x8, x16, x32, з DLL)	+	12	+
З розширеними можливостями	TIM1, TIM8, TIM20	16	вверх, вниз, вверх/вниз	Любе ціле число з ряду 1…65536	+	4	4
Загального призначення	TIM2, TIM5	32	вверх, вниз, вверх/вниз	Любе ціле число з ряду 1…65536	+	4	–
Загального призначення	TIM3, TIM4	16	вверх, вниз, вверх/вниз	Любе ціле число з ряду 1…65536	+	4	–
Загального призначення	TIM15	16	вверх	Любе ціле число з ряду 1…65536	+	2	1
Загального призначення	TIM16, TIM17	16	вверх	Любе ціле число з ряду 1…65536	+	1	1
Базові	TIM6, TIM7	16	вверх	Любе ціле число з ряду 1…65536	+	0	–

З точки зору керування трифазними імпульсними схемами особливе значення мають таймери з комплементарними виходами і можливістю формування мертвого часу. У мікроконтролерів STM32G4 може бути до трьох таймерів з розширеними можливостями: TIM1, TIM8, TIM20.

Для керування транзисторними півмостами також можуть бути використані таймери TIM15, TIM16, TIM17. Вони мають по одній парі комплементарних виходів.

У складі STM32G4 є 32-бітові та 16-бітові таймери загального призначення з можливістю рахунку вверх і вниз, а також вверх/вниз. Цей функціонал необхідний для реалізації інкрементальних давачів положення і швидкості обертання (енкодерів).

Для реалізації внутрішніх алгоритмів можуть використовуватися базові таймери, у яких немає зовнішніх виводів.

Окремо варто відзначити таймер з підвищеною роздільною здатністю HRTIM. HRTIM реалізований тільки в мікроконтролерах старшої серії STM32G4x4 Hi-resolution line. Він

дозволяє генерувати цифрові сигнали з прецизійними часовими інтервалами, що має велике значення для формування надточних ШІМ-сигналів і сигналів з прецизійним фазовим зсувом.

У складі HRTIM присутні відразу 7 таймерів: 1 ведучий і 6 ведених. HRTIM має 12 вихідних сигналів, які можуть працювати попарно з генерацією мертвого часу. Також він має 6 входів для контролю аварійних сигналів і 10 входів для контролю зовнішніх подій, наприклад, для виявлення точки перетину нуля в перетворювачах з ZVS.

Для тактування HRTIM використовується частота ядра 170 МГц. Період вхідного тактового сигналу додатково ділиться на 32 підперіоди, що забезпечує внутрішній опорний сигнал 5,44 ГГц. Тактовий сигнал проходить ланцюги компенсації, які забезпечують роздільну здатність на рівні 184 пс незалежно від температури, напруги живлення і технологічного розкиду при виробництві мікроконтролера. Ведені таймери HRTIM здатні працювати незалежно або узгоджено, що дозволяє керувати сучасними імпульсними схемами: резонансними LLC-перетворювачами, трифазними інверторами та іншим.

Телекомунікаційні інтерфейси

Мікроконтролери STM32G4 мають великий набір комунікаційних інтерфейсів.

Стандартні інтерфейси SPI/I2C зазвичай використовуються для зв'язку з іншими мікросхемами, що знаходяться на тій же платі, що і керуючий МК.

Інтерфейс USART необхідний для роботи з шинами RS-232, RS-422 і RS-485.

МК STM32G4 мають інтерфейси FDCAN (до трьох модулів), USB FS і USB Type-C ™ / UCPD.

У складі STM32G4 на додаток до традиційного USB FS був доданий UCPD. UCPD підтримує специфікації USB Type-C Rev. 1.3, USB Power Delivery Rev. 2.0 і 3.0. USB Type-C є перспективним і універсальним роз’ємом, призначеним для заміни попередніх версій роз’ємів USB. USB Power Delivery дозволяє забезпечувати живлення пристроїв, що підключаються з потужністю споживання до 100 Вт.

Блоки FDCAN значно відрізняються за своїми можливостями від блоків bxCAN, які реалізовані в інших родинах STM32 загального призначення. Головною перевагою FDCAN є максимальна швидкість передачі до 8 Мбіт/с, збільшена за рахунок розширеного і налаштовуваного поля даних. Наявність FDCAN має величезне значення для застосунків автоматики, так як останнім часом CAN набуває все більшого поширення завдяки своїй простоті, надійності і помірній вартості.

Додатковою пропозицією для аудіо застосувань є інтерфейс SAI.

Функції безпеки.

У STM32G4 реалізовані різні блоки, так чи інакше відповідають за функції безпеки:

• блок захисту пам'яті MPU;

• блок розрахунку контрольної суми CRC;

• блок шифрування AES (у моделей STM32G48x);

• генератор випадкових чисел RNG;

• блок захисту від злому TAMP.

Мікроконтролери STM32G4 пропонують широкий вибір механізмів для захисту вбудованої пам'яті:

• RDP (Readout Protection) – захист Flash, OTP, SRAM від читання в процесі виконання програми. Захист від читання поширюється на всю пам'ять;

• PCROP (Proprietary Code Protection) – захист вибраних ділянок Вибрані ділянки визначаються як код «тільки для виконання». PCROP дозволяє захистити не всю Flash, а тільки її частину;

• WRP (Write Protection) – захист пам'яті від запису і стирання.

Також варто відзначити функцію виявлення помилок ECC (Error Code Correction), що дозволяє виявляти і виправляти поодинокі бітові помилки в Flash. CCM-SRAM також має механізм контролю парності.

Для забезпечення безпеки при прошивці МК є наступні рішення:

• SFI (Secure internal firmware install) – технологія, що дозволяє виконувати безпечне програмування у неперевіреного виробника за рахунок кодування всієї прошивки за допомогою ключа AES-GCM [5].

• SFU (Secure Firmware Upgrade) – технологія, яка дозволяє виконати безпечне оновлення ПЗ в польових умовах.

Споживання.

В МК STM32G4 досягнуто низьке енергоспоживання завдяки ряду технологічних і структурних рішень, таких як перехід на техпроцес 90 нм, реалізація широкого вибору режимів зниженого споживання, реалізація гнучкої системи тактування, добавлення малоспоживаючої периферії (LPTIM), можливість очікувального живлення від батарейки.

Для живлення цифрової та аналогової периферії у всіх режимах, за винятком Standby і VBAT, мікроконтролери STM32G4 використовують два вбудованих регулятора: основний MR (Main Regulator) і малоспоживаючий LPR (Low-Power Regulator). Основний регулятор має два режими роботи: Range 1 і Range 2. У високопродуктивному режимі Range 1 вихідна напруга регулятора складає 1,2 В (Main regulator range 1 normal mode) або 1,28 В (Main regulator range 1 boost mode), а тактова частота ядра може досягати 150 і 170 МГц відповідно. У енергозберігаючому режимі Range 2 вихідна напруга регулятора складає 1,0 В, тактова частота обмежена 26 МГц, заборонені запис і стирання Flash.

У сплячих режимах для додаткової економії енергії живлення проводиться від LPR. У режимі VBAT живлення Backup-домену здійснюється від зовнішньої батареї 1,55 ... 3,6 В.

У табл. 1.9 показано особливості режимів зниженого споживання STM32G4.

Таблиця 1.9 – Режими зниженого споживання STM32G4

Режим	Регуля-тор	Ядро	Flash	SRAM	Тактування	Периферія
Run	R1	Так	ВКЛ*	ВКЛ	Любе	Вся
	R2					Вся крім USB, RNG
LPRun	LPR	Так	ВКЛ*	ВКЛ	Любе, крім PLL	Вся крім USB, RNG
Sleep	R1	–	ВКЛ*	ВКЛ**	Любе	Вся
	R2					Вся крім USB, RNG
LPSleep	LPR	–	ВКЛ*	ВКЛ**	Любе, крім PLL	Вся крім USB, RNG
Stop 0	MR	–	–	ВКЛ	LSE/LSI	Backup, CCS, BOR,PVD,PVM,RTC,LPUART ,USART,

						I2C,DAC,VREFBUF,OPAMP,COMP,LPTIM1,IWDG,GPIO
Stop 1	LPR	–	–	ВКЛ	LSE/LSI	Backup, CCS, BOR,PVD,PVM,RTC,LPUART,USART, I2C,DAC,VREFBUF,OPAMP,COMP,LPTIM1,IWDG,GPIO
Standby	LPR	–	–	SRAM2 ВКЛ	LSE/LSI	Backup, BOR,RTC,IWDG,GPIO (5 виводів)
	OFF			–		
Shutdown	OFF	–	–	–	LSE	Backup,RTC,GPIO (5 виводів)
VBAT	OFF	–	–	–	LSE	Backup, RTC
* – Можна відключити живлення і виключити тактування.						
** – SRAM1, SRAM2 и CCM SRAM можуть бути відключені окремо.						

5.3. Засоби розроблення та налагодження

Екосистема всіх мікроконтролерів STM32 об'єднує самі мікроконтролери, апаратні і програмні засоби розробки і налагодження, а також документацію. Для створення програм для STM32G4 можна використовувати безкоштовне середовище STM32CubeIDE, а також численні бібліотеки проміжного рівня. Крім того, для роботи з STM32G4 пропонуються різні налагоджувальні плати:

- NUCLEO-G431KB – налагоджувальна плата родини Nucleo-32 на базі мікроконтролера STM32G431KB. Плата має мінімальний набір компонентів, але може використовуватися для побудови прототипів спільно з платами розширення Arduino Uno V3. NUCLEO-G431KB має традиційний налагоджувач ST-LINK V2-1, доступний через роз'єм USB Micro-B.
- NUCLEO-G431RB – налагоджувальна плата родини Nucleo-64 на базі мікроконтролера STM32G431RB. Плата може працювати з платами розширення Arduino Uno V3 і ST morpho. NUCLEO-G431RB має традиційний налагоджувач ST-LINK V2-1, доступний через роз'єм USB Micro-B.
- NUCLEO-G474RE – налагоджувальна плата родини Nucleo-64 на базі мікроконтролера STM32G474RE. Плата може працювати з платами розширення Arduino Uno V3 і ST morpho. NUCLEO-G474RE має традиційний налагоджувач ST-LINK V2-1, доступний через роз'єм USB Micro-B.
- B-G474E-DPOW1 Discovery kit – налагоджувальний набір, який, на відміну від плат Nucleo, має на борту деякий набір функціональних блоків, в тому числі підвищуючий/понижуючий DC/DC-перетворювач, підсилювач потужності класу D, світлодіоди, джойстик, роз'єми USB Type-C™ і USB Micro-B, штифти для підключення плат розширення (2×32), вбудований програматор ST-LINK V3E.
- B-G431B-ESC1 Discovery kit – налагоджувальний набір, який являє собою ESC-контролер (Electronic Speed Controller), призначений для керування трифазним безколекторним двигуном за допомогою FOC-алгоритму або традиційного 6-ступеневого алгоритму. Набір складається з двох плат, з'єднаних перемичкою. На першій розміщується мікроконтролер STM32G431CB, драйвер L6387, силові транзистори STL180N6F7, а на другій – програматор і потенціометр. При необхідності плата

налагоджувача може бути від'єднана. Набір є готовим рішенням для дронів з живленням від 6S LiPo-батарей і струмом двигуна до 40 А.

- STM32G474E-EVAL – налагоджувальна і демонстраційна плата, побудована на базі мікроконтролера STM32G474QET6U. За допомогою цієї плати можна оцінити можливості практично всіх вбудованих периферійних блоків: АЦП, компараторів, ОП, ЦАП. На платі розміщені трансивери і роз'єми для послідовних інтерфейсів RS-232, RS-485, USB Type C, CAN 2.0 A/B. Плата забезпечена призначеними для користувача кнопками, світлодіодами, потенціометрами, а також кольоровим TFT-дисплеєм з роздільною здатністю 240x320 і SPI-інтерфейсом. STM32G484E-EVAL – повністю відповідає за своїми можливостями платі STM32G474E-EVAL і, крім того, підтримує функції криптографії завдяки використанню мікроконтролера STM32G484QET6U.

Галузі застосування

Мікроконтролери STM32G4 виробляються за сучасним 90 нм техпроцесом, який дозволяє збільшити ступінь інтеграції, розширити перелік вбудованої периферії, збільшити швидкодію і знизити споживання. У порівнянні з іншими родинами загального призначення, нове родина STM32G4 володіє найбагатшим набором вбудованої аналогової периферії.

Висока продуктивність і розвинена аналогова периферія роблять мікроконтролери STM32G4 надзвичайно привабливими для застосунків, пов'язаних з обробкою аналогових сигналів в реальному часі. Як цільові програми для STM32G4 можна виділити побутову та промислову автоматику, автоматику будівель, електроприводи, джерела живлення, багатофазні імпульсні інвертори, коректори коефіцієнта потужності, вимірювальні прилади, системи освітлення і так далі.

6. Родина МК MSP

6.1. Ключові характеристики МК MSP

Розрядність 16 біт

Для сучасних вимірювальних пристроїв і систем можливостей 8-розрядних платформ дуже часто виявляється недостатньо, а побудова їх на основі 32-розрядних рішень призводить до появи непотрібної надмірності і необгрунтованого підвищення енергоспоживання. У цьому випадку 16-розрядні мікроконтролери MSP430 є найкращим вибором, оскільки вони, не маючи обмежень, властивих 8-розрядних систем, володіють сумірною з ними вартістю і економічністю.

Ортогональна система команд

Всі мікроконтролери MSP430 мають однаковий набір з 27 продуманих ортогональних інструкцій, що дозволяють домогтися високої щільності коду. Це означає, що при одній і тій же функціональності прошивка мікроконтролера MSP430 потребують меншої кількості пам'яті програм, ніж для 8-розрядних платформ.

Ультра мале енергоспоживання

Можливість тактування кожного периферійного модуля асинхронно від ядра дозволило створити безліч варіантів режимів і довести струм споживання в самому економічному з них до 15 нА. Очевидно, що це властивість найкращим чином підходить для пристроїв з батарейним

живленням, багато з яких є основою для широкого кола інтелектуальних систем.

Незалежна сегнетоелектрична пам'ять FRAM.

FMAN – одна з ключових переваг родини МК MSP. Сегнетоелектрична пам'ять FRAM з довільним доступом по продуктивності не поступається традиційним статичним ОЗУ, але при цьому, на відміну від них, зберігає дані при відключенні живлення. Це дозволяє позиціонувати родину МК MSP430 не тільки як заміну традиційних МК з Flash-пам'яттю, а й для застосунків, що вимагають справжнього енергонезалежного ОЗП, наприклад, в системах збору та накопичення даних, де необхідно часто оновлювати або перезаписувати інформацію «на льоту».

Розвинена периферія

Крім традиційних портів введення-виведення загального призначення (GPIO) МК MSP430 можуть містити вузли, необхідні для роботи з аналоговими сигналами (АЦП, ЦАП, компаратори, операційні та трансімпедансні підсилювачі) і цифровими сигналами (таймери, ШІМ-модулятори, сигнальні процесори, модулі розрахунку CRC). До складу кожного, навіть самого простого МК зазвичай входить як мінімум один модуль, що дозволяє йому обмінюватися даними за допомогою популярних інтерфейсів: USART, SPI, USB і інших. Крім цього, в родині MSP430 існують спеціалізовані МК, призначені для роботи з рідкокристалічними індикаторами, сенсорними клавіатурами, ультразвуковими давачами і іншою специфічною апаратурою.

Всебічна технічна підтримка.

Компанія Texas Instruments в рамках концепції «One Platform, One Ecosystem, Endless Possibilities» («одна платформа, одна екосистема, безмежні можливості») зобов'язується підтримувати розробника на кожному етапі роботи над проектом. З офіційного сайту компанії можна завантажити детальну технічну документацію на всі мікроконтролери родини, безкоштовні інструменти для розробки програмного забезпечення, приклади готових проектів з сирцевим кодом, навчальні інформаційні текстові та відеоматеріали, а також замовити налагоджувальні та демонстраційні плати. Крім цього, родина MSP430 підтримується багатьма сторонніми виробниками, що пропонують свої інструменти для роботи з даними приладами, наприклад, середовище розробки програмного забезпечення IAR Embedded Workbench від шведської компанії IAR Systems.

6.2. Області застосування родини МК MSP

Пристрої на батарейках

Ультра мале споживання в поєднанні з підвищеною розрядністю і можливістю організації незалежної оперативної пам'яті вже давно стало «візитною карткою» цієї родини. Тому одне з ключових напрямків використання мікроконтролерів MSP430 - автономні пристрої на батарейках, наприклад, пристрої Інтернету речей (IoT-пристрої).

Заміна спеціалізованих мікросхем

Цей напрямок сформувався порівняно недавно і полягає в заміні традиційних вузькоспеціалізованих мікросхем, таких як незалежна пам'ять, перетворювачі інтерфейсів, контролери скидання, монітори живлення, годинник реального часу і багато інших, більш

функціональними аналогами на основі мікроконтролерів MSP430. Такий підхід дозволяє не тільки знизити вартість пристрою, оскільки в більшості випадків рішення на основі MSP430 коштуватимуть дешевше, але і підвищити функціональність цих вузлів за рахунок можливості гнучкого налаштування і програмування, в тому числі і в реальному часі. Для підтримки цього напрямку була запущена спеціальна програма «25 функцій за 25 центів», в рамках якої можна завантажити приклади реалізації 25 найбільш популярних вузлів, використовуваних в сучасній радіоапаратурі.

Традиційні автоматизовані пристрої та системи

Наявність специфічних особливостей не означає, що мікроконтролери MSP430 не можна використовувати в традиційних пристроях і системах. Навпаки, за рахунок підвищеної розрядності реалізувати систему на основі MSP430 можна простіше і швидше, ніж на традиційних 8-розрядних мікроконтролерах. А мале енергоспоживання і приваблива ціна будуть додатковими стимулами для вибору цієї платформи.

6.3. Серія МК MSP430

На даний момент родина МК MSP430 представлена трьома серіями:

- Value Line (MSP430FR2xxx);
- CapTIvate (MSP430FR25x/26x);
- Ultrasonic Sensing (MSP430FR504x/604x).

МК серії Value Line мають високу продуктивність і щільністю коду, характерними для 16-розрядних застосунків, в поєднанні з традиційними для 8-розрядних рішень економічністю і невисокою вартістю. Особливістю цієї серії є можливість роботи в широкому діапазоні напруги живлення 1,8...3,6 В і ультра малий струм споживання, максимальне значення якого в активному режимі не перевищує 2 мА, в режимі очікування – менше 1 мкА, а в вимкненому стані – всього 15 нА. Все це робить їх ідеальними для малогабаритних і портативних пристроїв з батарейним живленням.

Мікроконтролери серії Value Line мають 0,5...4 кбайт вбудованої FRAM-пам'яті і позиціонуються як альтернатива традиційним 8-розрядним рішенням.

Мікроконтролери серії CapTIvate орієнтовані на використання в пристроях з сенсорним керуванням. Всі мікроконтролери цієї серії містять вбудований спеціалізований периферійний модуль CapTIvate, що дозволяє контролювати кнопки, повзунки (slider) та обертові регулятори (spinner), що не містять механічних контактів. Використовувана в модулі унікальна технологія вимірювання ємності в широкому діапазоні (10 фФ ... 300 пФ) відрізняється високою чутливістю і забезпечує чітку роботу органів керування як при використанні різних металевих, скляних, пластикових і інших накладок, так і в найскладніших умовах – при наявності вологи, бруду, жиру та інших забруднень. Ще однією особливістю модуля CapTIvate є можливість одночасної роботи з декількома давачами, які можна включати як послідовно, так і паралельно.

МК серії Ultrasonic Sensing, завдяки наявності спеціалізованого модуля USS (Ultrasonic Sensing Solution), орієнтовані на використання в застосунках, що вимагають визначення витрати рідини або газу. Особливістю модуля USS є мінімальна кількість зовнішніх компонентів, що позитивно позначається на вартості кінцевого пристрою. Крім цього, до складу периферійних модулів мікроконтролерів даної серії входять всі необхідні допоміжні вузли, що

використовуються в подібних програмах: годинник реального часу, АЦП, компаратори, драйвери РКІ та інші. МК Ultrasonic Sensing також містять мало споживаючий математичний співпроцесор (Low-EnergyAccelerator, LEA), що дозволяє виконувати множення матриць, швидкі перетворення Фур'є, цифрову фільтрацію і інші обчислення без витрат процесорного часу. Все це дозволяє реалізувати досить складні в функціональному плані застосунки фактично в однокристальному виконанні. Завдяки малому енергоспоживанню мікроконтролери серії Ultrasonic Sensing ідеально підходять для використання в портативних і стаціонарних вимірювальних приладах з батарейним живленням, в тому числі і в лічильниках води, газу і теплової енергії.

6.4. Програмні інструменти розроблення

Компанія Texas Instruments, підтримуючи концепцію «One Platform, One Ecosystem, Endless Possibilities» («одна платформа, одна екосистема, безмежні можливості»), надає розробнику повний комплекс безкоштовних інструментів для швидкого створення застосунків на основі мікроконтролерів MSP430. Основним інструментом для роботи з родиною MSP430 є безкоштовний програмний пакет Code Composer Studio, створений на основі Eclipse. Він дозволяє створювати програмне забезпечення для мікроконтролерів MSP430, C2000, F24x/C24x, Sitara і багатьох інших. До складу пакету включені редактор сирцевого коду, компілятор C/C++ з можливістю оптимізації, засоби управління проектами і безліч інших інструментів. Спеціалізоване програмне забезпечення CapTIvate Design Center призначене для розробки застосунків з сенсорними пристроями на основі мікроконтролерів серії CapTIvate. За допомогою даного інструменту можна в графічному режимі конфігурувати модуль CapTIvate і в найкоротші терміни починати роботу над будь-яким проектом, що містить сенсорні елементи керування. Аналогічне програмне забезпечення Ultrasonic Sensing Design Center розроблено і для МК серії Ultrasonic Sensing. При цьому крім утиліти конфігурації модуля в графічному режимі даний програмний пакет містить безліч готових бібліотек з прикладами їх застосування. Крім цього, на сайті Texas Instruments присутня докладна технічна документація на всі програмні інструменти, в тому числі і відеоуроки по їх швидкому освоєнню.

6.5. Апаратні інструменти розроблення

Щоб швидко оцінити можливості того чи іншого МК, компанія Texas Instruments пропонує широкий вибір налагоджувальних плат для всіх лінійок родини MSP430. Більшість з них містить вбудований програматор-налагоджувач eZ-FET, що дозволяє швидко почати роботу з платами. Для самостійних проектів використовується програматор MSP-FET, за допомогою якого можна виконувати налагодження застосунків по інтерфейсах JTAG або SWD.

Серія Value Line

MSP-EXP430FR2355 – налагоджувальна плата на основі MSP430FR2355. Крім мікроконтролера і вбудованого програматора налагоджувача, на платі встановлено дві спеціальні кнопки, два індикаторних світлодіода і давач освітленості. Додаткові давачі, приводи і іншу периферію можна підключити за допомогою стандартного роз'єму Grove. Крім цього, плата містить 40-контактний роз'єм Launch Pad, що дозволяє інтегрувати її з іншими платами розширення екосистеми Booster Pack.

MSP-EXP430FR2311 – налагоджувальна плата на основі MSP430FR2311. Містить

вбудований програматор налагоджувач eZ-FET, зворотний канал якого можна використовувати для передачі інформації на комп'ютер по інтерфейсу UART. Крім цього, на платі встановлені одна призначена для користувача кнопка, два індикаторних світлодіоди і фотодіод. Додаткові плати розширення екосистеми Booster Pack можна підключити до 20-контактного роз'єму Launch Pad.

MSP-EXP430FR2433 – налагоджувальна плата на основі MSP430FR2433. Містить вбудований програматор налагоджувач eZ-FET і інструменти для визначення ультра малих величин споживаного струму Energy Trace. Крім цього, на платі встановлено дві призначені для користувача кнопки і два світлодіоди. Плата містить 20-контактний роз'єм Launch Pad, що дозволяє інтегрувати її з іншими платами розширення екосистеми Booster Pack.

MSP-EXP430FR4133 – налагоджувальна плата на основі MSP430FR4133 з вбудованим програматором налагоджувачем eZ-FET, вимірювачами ультра малих струмів Energy Trace і цифро-буквовим рідкокристалічним індикатором. Плата містить 20-контактний роз'єм Launch Pad, що дозволяє інтегрувати її з іншими платами розширення екосистеми Booster Pack, а також дві спеціальні кнопки і два світлодіоди.

Серія CapTIvate

Щоб швидко вивчити технологією CapTivate і освоїти можливості центру CapTIvate Design Center використовують плату EVM430-CAPMINI на основі МК MSP430FR2512, що містить 4-кнопкову сенсорну клавіатуру, чотири світлодіоди і звуковий індикатор. Плата має можливість подвійного живлення: як через інтерфейс USB, так і автономно – від батареї CR1632.

Якщо чотирьох кнопок для розроблення недостатньо, то можна придбати, яка входить в частину екосистеми Booster Pack, плату розширення BOOSTXL-CAPKEYPAD, що містить 12 сенсорних кнопок зі світлодіодним підсвічуванням. Ця плата розрахована на роботу зі спеціалізованим програматором CAPTIVATE-PGMR, орієнтованим на використання з налагоджувальними платами серії CapTIvate в середовищі CapTIvate Design Center. До цієї платі також можна підключити додатково до трьох додаткових ємнісних давачів в обхід основної клавіатури.

Плати CAPTIVATE-FR2676 і CAPTIVATE-FR2633 на основі мікроконтролерів MSP430FR2676 і MSP430FR2633, відповідно, дуже схожі між собою по функціональності і призначені для освоєння різноманітних сенсорних пристроїв. Обидві плати містять 20-контактний роз'єм для налагодження, 48-контактний роз'єм для підключення зовнішніх давачів, а також роз’єм для подачі зовнішнього живлення на плату. Ці плати також можна інтегрувати в екосистему Booster Pack.

Серія Ultrasonic

Для демонстрації можливостей серії МК Ultrasonic пропонуються плати EVM430-FR6047 і EVM430-FR6043 на основі МК MSP430FR6047 і MSP430FR6043, відповідно. Кожна з плат містить вбудований програматор налагоджувач eZ-FET, інтегрований рідкокристалічний індикатор і може входити в екосистему Booster Pack. Живлення здійснюється як від зовнішнього джерела, так і від інтерфейсу USB.

7. Родина МК AVR

В залежності від функціональних характеристик МК AVR розділяють на наступні родини: XMEGA, Mega, Tiny.

7.1. Особливості МК AVR XMEGA

• **Високоточні аналогові функції** – 12-розрядні аналого-цифрові перетворювачі(АЦП) з підсилювальним каскадом і загальною продуктивністю 4 млн. вибірок в секунду, швидкодіючі 12-розрядні цифро-аналогові перетворювачі (ЦАП) високої потужності, а також інші функціональні можливості, що знижують потребу в зовнішніх компонентах.

• **Продуктивність в реальному часі** – Система обробки подій спрощує взаємодію між периферійними пристроями, забезпечуючи 100-процентну прогнозованість часу відгуку. Усі периферійні пристрої можуть використати прямий доступ до пам'яті (DMA) для передачі даних, що дозволяє розвантажити ЦП.

• **Технологія Atmel picoPower®** – Повноцінне функціонування забезпечується при напрузі 1,6 В. Лічильник реального часу працює при струмі 100 нА, а дані повністю зберігаються в пам'ять SRAM для максимально швидкого пробудження.

• **Високий рівень інтеграції** – Пристрої AVR XMEGA об'єднують в собі криптоблоки AES (Advanced Encryption Standard) і DES (Data Encryption Standard), до 32 виходів широтно-імпульсної модуляції (ШІМ), 8 інтерфейсів UART, 4 інтерфейси TWI (I2C) і 4 канали послідовного периферійного інтерфейсу (SPI), модуль генератора для циклічного контролю надлишковості (CRC) і багато що інше.

• **Бібліотека програмного забезпечення AVR** – Повна бібліотека драйверів пристроїв і комунікаційних стеків дозволяє заощадити час і зусилля, зосередившись на важливіших завданнях проектування.

• **Технологія сенсорного введення Atmel QTouch®** – Підтримка бібліотеки QTouch дозволяє з легкістю реалізувати повноцінний ємнісний сенсорний інтерфейс для кнопок, повзунків і коліс прокрутки.

• **Підключення пристроїв USB** – Швидкісне функціонування без необхідності у зовнішніх кристалах, 31 вивід для підключення кінцевих пристроїв, а також спеціальна функція багатопакетної передачі, що дозволяє збільшити швидкість обміну даними, понизивши навантаження на ЦП.

• **Модуль XMEGA Custom Logic(XCL)** – Пристрої AVR XMEGA серії E оснащуються інноваційним модулем XMEGA Custom Logic(XCL), що складається з двох незалежних 8-розрядних таймерів/лічильників і двох таблиць для визначення зв’язуючої логіки. Цей модуль розроблений з метою зниження загальної вартості комплектуючих і економії місця на друкованій платі. Він дозволяє замінити такі зовнішні схеми, як елементи затримки, RS- і D-тригери, логічні схеми вибору кристала на D-тригері, логічні елементи AND, NAND, OR, NOR, XOR, XNOR, NOT, MUX AND/OR/XOR. У поєднанні з приймачем USART цей модуль може забезпечити підтримку спеціалізованих комунікаційних протоколів.

Розвиток цієї серії зупинено, оскільки нішу продуктивних контролерів зайняли 32-бітові МК на базі ядер групи ARM Cortex-M.

7.2. Особливості МК MEGA AVR

• **Широка номенклатура** – родина megaAVR є найбільш різноманітною з точки зору характеристик МК, наприклад об'ємів пам'яті, кількості виведень, набору периферійних пристроїв, можливості повторного використання коду в різних проектах.

• **Технологія picoPower** – деякі пристрої megaAVR відрізняються наднизьким споживанням потужності і підтримують індивідуальне налаштування режимів очікування з малим енергоспоживанням, що робить їх ідеальними для систем з живленням від акумуляторів.

• **Високий рівень інтеграції** – МК megaAVR оснащені вбудованою FLASH-пам’яттю, пам'яттю SRAM, внутрішньою пам'яттю EEPROM, інтерфейсами SPI, TWI(I2C), USB, CAN, LIN, приймачем/передавачем USART, сторожовим таймером, зовнішнім або внутрішнім прецизійним генератором, контактами для введення/виведення даних загального призначення, що спрощує проектування і скорочує перелік використовуваних в системі компонентів.

• **Аналогові функції** – аналогові функції реалізуються з допомогою АЦП, ЦАП, вбудованого температурного датчика, внутрішнього джерела опорної напруги, детектора пониження напруги, високошвидкісного аналогового компаратора і аналогового підсилювача з програмованим коефіцієнтом. Висока міра інтеграції дозволяє створювати системи з меншою кількістю зовнішніх аналогових компонентів.

• **Швидка розробка** – Мікроконтролери megaAVR прискорюють розробку систем за допомогою потужної функції внутрішньосхемного програмування і налагодження (ISP). Крім того, внутрішньосхемне налагодження спрощує програмування пристроїв під час виробництва та їх оновлення в процесі експлуатації.

• **Підтримка Інтернету речей** – IoP (Інтернет речей) може застосовуватися майже у будь-якому застосуванні — від звичайної автоматики будівель і побутової автоматики до медичних систем і систем охорони здоров'я. Щоб виконати завдання обчислення вбудованими системами і передачі даних в Інтернет, рішенням IoP зазвичай потрібно обчислювальну потужність. Все частіше пристрої Інтернету речей працюють від акумуляторів, тому енергоспоживання стає ключовим чинником при виборі продукту. З цієї точки зору мікроконтролери megaAVR займають передові позиції поряд з кращими у світі МК.

7.3. Особливості МК TINY AVR

• **Компактність** – МК tinyAVR спеціально призначені для систем, чутливих до розмірів і вартості. Наймініатюрніші мікросхеми tinyAVR мають габарити 1,5×1,4 мм. Їх можна використати як одинокристальні рішення для компактних систем. У більших системах вони дозволяють реалізувати зв’язуючу і розподілену логіку.

• **Ємнісне сенсорне введення** – бібліотека Atmel QTouch® допомагає розробникам вбудовувати сенсорні кнопки, повзунки і колеса прокрутки в застосунки на базі МК Atmel AVR загального призначення. Безкоштовна бібліотека QTouch містить по декілька файлів для кожного пристрою, які дозволяють реалізувати підтримку різного числа каналів відстеження дотиків, що забезпечує гнучкість і ефективність сенсорних застосувань.

• **Швидкість і ефективність програмування** – процесор AVR забезпечує МК tinyAVR той же рівень продуктивності, що і у більших пристроїв AVR, а також обчислювальну потужність, що у декілька разів перевищує потужність МК тих же розмірів виробництва конкуруючих фірм. Ці пристрої відрізняються гнучкістю і універсальністю, мають високу ефективність програмування і можуть використовуватися в широкому спектрі застосувань.

• **Високий рівень інтеграції** – кожний контакт мікросхеми підтримує різні функції, наприклад введення/виведення даних, АЦП або ШІМ. Навіть контакт скидання можна

використати для введення/виведення даних. МК tinyAVR оснащені універсальним послідовним інтерфейсом (USI), який можна використати як інтерфейси SPI, UART або TWI.

• **Робоча напруга 0,7 В** – більшість МК працюють при напрузі живлення 1,8 В і вище. МК tinyAVR оснащені імпульсним підвищуючим стабілізатором, завдяки якому напруга живлення від однієї батареї типу AA або AAA перетворюється в стабільну напругу 3 В для живлення усієї системи.

7.4. Нові серія МК AVR

На початку 2020 року Microchip анонсував три серії мікроконтролерів, що належать до нової родини МК AVR:

• AVR-DA;

• AVR-DB;

• AVR-DD.

Разом з назвою серії змінилося і позначення пристроїв. Маркування тепер має вигляд «AVRXXYYZZ», де:

• XX – обсяг пам'яті в кілобайтах;

• YY – родина;

• ZZ – кількість виводів корпусу.

Серія AVR-DA

Серія AVR-DA складається з 11 пристроїв з варіантами вибору обсягу пам'яті від 32 до 128 кбайт в корпусах з 28 ... 64 виводами. Позначення AVR-xxDAyy, де xx=32,64,128 – обсяг Flash пам’яті (кб), yy=28, 32, 48, 64 – кількість виводів.

Зміни торкнулися ядра і його системи живлення: ядро може функціонувати на збільшеній максимальній частоті 24 МГц у всьому діапазоні напруги живлення 1,8...5,5 В.

Вперше в пристроях AVR з'явився модуль Zero Cross Detector – детектор перетину змінним струмом нульового рівня. Раніше це була периферія, властива тільки для PIC-контролерів.

АЦП було оновлено: нова версія забезпечує оцифровування аналогової напруги з частотою до 130 Гц і роздільною здатністю 12-біт з можливістю включення диференціального режиму роботи. Акумулятор був збільшений до 128 вибірок. Як і в попередній версії, підтримуються наступні режими роботи:

• одиничне перетворення;

• режим безперервного перетворення;

• режим накопичення;

• режим порівняння з порогом;

• режим запуску за подією;

• режим вимірювання температури (від вбудованого давача температури).

У пристроях нової серії з'явився модуль ЦАП. Нагадаємо, що контролери Mega такого не мали. Перетворювач працює на швидкості 140 ksps і має роздільну здатність 10 біт.

У порівнянні з серією ATmega, було збільшено кількість таких модулів периферії:

• кількість модулів USART збільшено до шести;

• кількість аналогових компараторів збільшено до трьох.

Звернемо увагу на наявність специфічної периферії – Peripheral Touch Controller, сенсорного контролера, що дозволяє реалізувати ємнісні сенсорні елементи управління – кнопки, повзунки, обертові регулятори і 2D-поверхні. Завдяки бібліотеці QTouch Library

налаштування цього модуля зводиться до декількох клацань миші.

Для оцінки можливостей нової серії і швидкого прототипування пристроїв на її базі компанія Microchip випустила налагоджувальну плату AVR128DA48 Curiosity Nano Evaluation kit, яка зображена на рис. 1.6.

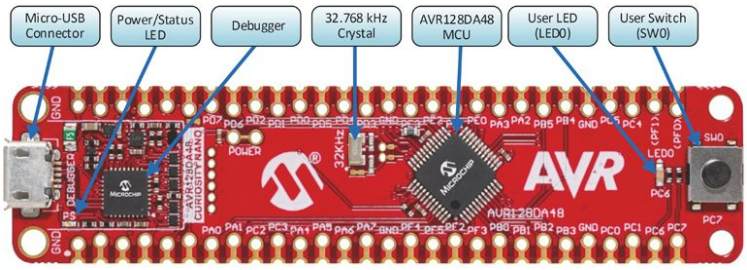


Рисунок 1.6 – Налагоджувальна плата AVR128DA48 Curiosity Nano Evaluation kit

Серія налагоджувальних плат Curiosity Nano – це найпростіші налагоджувальні плати виробництва Microchip. Плати серії Curiosity Nano містять стандартний набір компонентів:

• одну призначену для користувача кнопку;

• один призначений для користувача світлодіод;

• вбудований програматор/зневадник з USB-портом.

Дана плата, на додаток до стандартного набору компонентів, має розпаяний годинниковий кварц.

Плати Curiosity Nano можуть підключатися як процесорний модуль в базову плату Curiosity Nano Base, яка містить три порти розширення microBUS, використовувані для підключення модулів розширення Click Boards виробництва MikroElektronika, і один порт розширення Xplained Pro для підключення однойменних модулів розширення Microchip. Базова плата зображена на рис. 1.7.

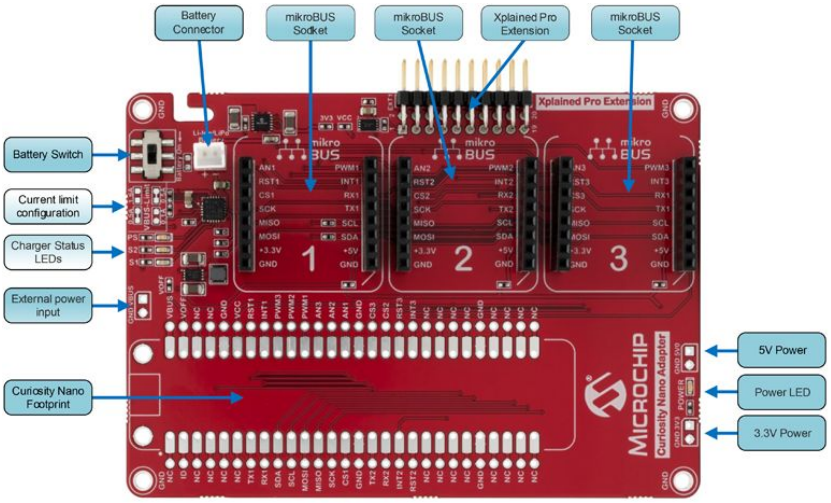


Рисунок 1.7 – Плата Curiosity Nano Base

Серія AVR-DB

Серія AVR-DB складається з 11 пристроїв з обсягом пам'яті 32 ... 128 кбайт в корпусах, що мають 28...64 виводів. Позначення AVR-xxDByy, де xx=32,64,128 – обсяг Flash пам’яті (КБайт), yy=28, 32, 48, 64 – кількість виводів.

Серія DB дуже схожа на вже розглянуту серію: ті ж обсяги пам'яті і корпусу, частота ядра і напруга живлення, подібний набір периферії. Однак є і відмінності. У наборі периферії відбулася заміна – Peripheral Touch Controller серії DA замінили на операційні підсилювачі. Кожен операційний підсилювач має в петлі зворотного зв'язку резистивний дільник з налаштованим співвідношенням опорів, що дозволяє налаштувати коефіцієнт підсилення без використання зовнішніх елементів. Для підвищення коефіцієнта підсилення операційні підсилювачі можуть з'єднуватися каскадом.

Наступна відмінність від серії DA – підтримка інноваційної технології MVIO, суть якої полягає в тому, що Port C отримав незалежне живлення VDDIO2, що дозволяє послідовним інтерфейсам, виведеним на цей порт, комунікувати з зовнішніми пристроями, які живляться від напруги, відмінної від живлення мікроконтролера.

Модифікації піддався Clock Controller, що підтримує не тільки зовнішній годинниковий кварц, але і високочастотні кварцові резонатори з частотою до 32 МГц.

Для серії ABR-DB доступна налагоджувальна плата AVR128DB48 Curiosity Nano Evaluation kit. На плату додали кварц 16 МГц і не розпаяний роз’єм для підключення окремого живлення для Port C.

Серія AVR-DD

Дана серія включає в себе 12 пристроїв з обсягами пам’яті 16...64 кБ в корпусах з 14...32 виводами.

Серія AVR-DD займає нішу більш компактних пристроїв зі зменшеним обсягом пам’яті. У ній набір периферії більше не містить таких специфічних модулів, як Peripheral Touch Controller або операційні підсилювачі. В іншому якісний склад периферії не змінився, але кількість модулів було зменшено:

- один модуль Zero Cross Detector;
- один компаратор;
- шість каналів системи подій;
- два модуля USART, один SPI і один I2C.

Серію DD з серією DB об'єднує підтримка технології MVIO на Port C і підтримка зовнішнього кварцу високої частоти.

Порівняння серій DA, DB і DD

Розглянуті серії підтримують максимальну частоту ядра 24 МГц у всьому діапазоні напруги живлення 1,8...5,5 В. В табл. 1.10 показано характеристики розглянутих серій.

Таблиця 1.10 – Порівняння серій DA, DB і DD

Назва	**AVR-DA**	**AVR-DB**	**AVR-DD**
Максимальна частота ядра, МГц	24	24	24
Flash-пам’ять, кбайт	32… 128	32… 128	16…64
Пам’ять SRAM, кбайт	4…16	4…16	2…8
Пам’ять EEPROM, байт	512	512	256
Виводи	28…64	28…64	14…32
Виводи I/O	22…54	22…54	11…27
12 біт АЦП (канали)	1 (10…22)	1 (9…22)	1 (7…23)
10 біт ЦАП (виходи)	1 (1)	1 (1)	1 (1)

Компаратори	3	3	1
Сенсорний контролер (PTC)	1	–	–
Операційні підсилювачі	–	2…3	–
Виводи MVIO	–	–	8
Детектор пересічення нуля (ZCD)	1…3	1…3	1
Система подій, канали	8…10	8…10	6
Віконний сторожовий таймер (WWDT)	1	1	1
Конфігурована логіка (CCL), LUT	1(4-6)	1(4-6)	1(4)
USART/SPI/I2C	(3/5/6)/2/(1/2)	(3/5/6)/2/(1/2)	2/1/1
Таймер 16 біт	4/6/7	4/6/7	3/4
Таймер 12 біт	1	1	1
Діапазон робочих температур, °C	I = 85, E = 125	I = 85, E = 125	I = 85, E = 125

AVR-DA і DB займають нішу продуктивних 8-бітових контролерів з великим набором периферії. Основна відмінність в тому, що серія DA має Peripheral Touch Controller, а серія DB – операційні підсилювачі.

Серія DD займає нішу більш компактних, але менш продуктивних пристроїв з урізаним набором периферії. Серії DB і DD схожі в тому, що мають підтримку технології MVIO і зовнішнього кварцу високої частоти. У нових серіях застосовані і інші перевірені технології Microchip, що підвищують надійність, гнучкість системи і зменшують енергоспоживання:

• Core Independent Peripherals – незалежна від ядра периферія, здатна продовжити роботу навіть при переході контролера в енергозберігаючий режим і відключення ядра;

• Cyclic Redundancy Check Memory Scan – модуль, що дозволяє виявити пошкодження коду програми, що зберігається в Flash-пам'яті;

• Configurable Custom Logic – модуль налаштування користувацької логіки, що дає можливість реалізувати нескладні цифрові пристрої, що функціонують без залучення процесора;

• Event System – система подій, що дозволяє модулям периферії взаємодіяти один з одним без участі процесора, в тому числі і в сплячому режимі.

7.5. Програмні засоби розроблення

Підтримка нових серій включена в інтегровані середовища розробки від Microchip:

• Microchip Studio (заміна Atmel Studio) – середовище розроблення для МК PIC і AVR. Підтримка нових пристроїв доступна після встановлення пакету підтримки пристроїв (Device Family Pack) AVR-Dx_DFP.

• MPLAB X IDE – середовище розроблення для МК PIC і AVR, включаючи останні серії. Плагін MPLAB Code Configurator дозволяє графічну конфігурацію пристрою і генерацію оптимізованого коду.

Галузі застосування

Розглянуті серії відносяться до контролерів широкого спектру застосувань і можуть використовуватися в різних галузях, які потребують автоматичного керування в реальному часі: в побутовій електроніці, медицині, промисловій електроніці і пристроях інтернету речей, як основний обчислювач або допоміжний пристрій.

Нові серії відмічені знаком Functional Safety Ready, що означає, що вони можуть використовуватися в застосунках, критичних до відмов: автомобільній і промисловій електроніці. За запитом замовника надається звіт зі статистикою відмови контролера і керівництво по забезпеченню вимог стандартів безпеки.

Велика екосистема, що включає в себе засоби розроблення, плати налагодження, технічну документацію та приклади проектів дозволяє скоротити час, необхідний на проектування та виведення на ринок нового пристрою.

Питання.

1. Які основні відмінності між МП і МК?
2. За якими ознаками класифікуються МК.?
3. Порівняння системи команд МК CISC і RISC.
4. Серії МК STM32G4
5. Аналогова периферія МК STM32G4
6. Родина МК MSP і її основні характеристики
7. Програмні та апаратні інструменти розроблення МК MSP
8. Родина МК PIC і AVR. Особливості МК AVR XMEGA, MEGA, TINy
9. Нова серія МК AVR-DA, AVR-DB, AVR-DD і їх можливості.
10. Програмні та апаратні інструменти розроблення МК.

ЛЕКЦІЯ 2. Архітектура мікроконтролерів

Мета. Вивчення архітектури мікроконтролерів і характеристик структурних елементів.

Вступ. Мікроконтролери AVR мають єдину базову структуру, але відрізняються кількістю та характеристиками структурних елементів. В мікроконтролерах AVR DA, AVR DB, AVR DD добавлені додаткові функціональні блоки.

План.

1. Базова структура МК AVR
1.2. Організація пам’яті МК ATmegs8/ATmega48/ATmega8515/ATmega8535
1.3. Регістр стану SREG
1.4. Регістр керування МК
1.5. Регістр стеку SP, SPH, SPL
1.6. Регістри інших структурних елементів
1.7. Паралельні порти введення/виведення
1.8. Таймери/лічильники
1.9. Сторожовий таймер
1.10. USART
1.11. Послідовний інтерфейс SPI
1.12. Інтегрований аналоговий компаратор
2. Базова структура МК AVR DB

1. Базова структура МК AVR

Всі мікроконтролери (МК) родини AVR мають Гарвардську архітектуру, в якій розділяється пам’ять програм, пам'ять даних і пам'ять введення/виведення, як показано на рис. 2.1. При такій архітектурі засоби адресації дозволяють створювати ефективні програми з високою швидкодією.

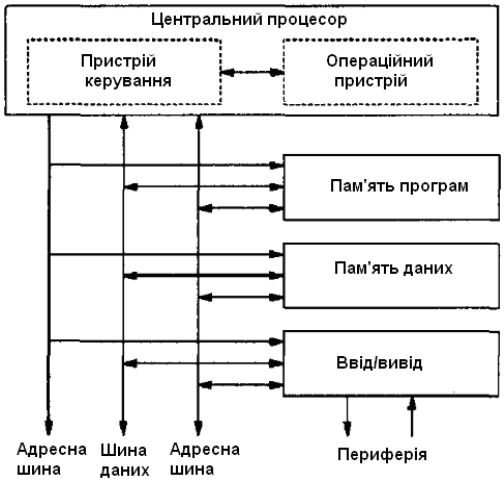


Рисунок 2.1 – Архітектура МК

МК AVR мають також єдину базову структуру. До складу МК входять:

• **GCK (**Generator Clock)− генератор тактового сигналу;

• **CPU**, Central Processing Unit –центральний процесор;

• **RAM (**Random Access Memory), **SRAM** (Static Random Access Memory) − пам'ять з довільним доступом,чи оперативний запам'ятовуючий пристрій статичного типу. Пам'ять призначена для зберігання проміжних результатів та інших тимчасових даних протягом виконання програми.

• **ROM** (Read Only Memory) − пам'ять тільки для читання, чи постійно запам'ятовуючий пристрій. Пам'ять ROM зберігає програмні інструкції (програму виконання) та таблиці з довідковими даними. У сучасних МК вона реалізована у вигляді пам'яті виконаної за технологією Flash (FlashROM);

• **EEPROM** (Electrically Erasable and Programmable ROM) – енергонезалежна пам'ять даних. У сучасних МК вона представляє собою flash-пам'ять. Основна відмінність від flash-пам'яті програм (ROM) – це можливість вибіркового програмування окремих байтів, на відміну від поблокового програмування flash-пам'яті програм.

• **I/O Ports** (input/output) – паралельні порти введення/виведення надають інтерфейс між МК та периферійними пристроями введення/виведення, такими як: клавіатура, дисплей тощо. Виводи портів можуть бути як двонаправленими, так і однонаправленими, або лише на вхід, або на вихід.

• **Serial I/O** – послідовні порти для обміну даними,що реалізовують асинхронні (напр., UART) або синхронні (напр., SPI) інтерфейси обміну даними. Асинхронний інтерфейс використовує протокол зі стартовим та стоповим бітами для передачі та прийому. Стартовий та стоповий біти вбудовані у кожен байт даних. Синхронні інтерфейси використовують синхронізуючі імпульси для кожного біта.

• **Timer/Counter** (таймер/лічильник) – використовують для відліку часу або/та меж часових інтервалів між подіями, підрахунку кількості подій та генерації швидкості передачі даних (у бодах) для послідовних портів. Таймери також можуть керувати певними I/O-виводами у МК, наприклад, здійснювати підрахунок кількості імпульсів, що поступають на його вхід, чи навпаки, виводити певні послідовності імпульсів.

• **PWM** (Pulse Width Modulation, широтно-імпульсна модуляція, ШІМ) – це спосіб кодування аналогового сигналу шляхом зміни ширини (тривалості) прямокутних імпульсів несучої частоти. Найбільш часто PWM використовують для керування моторами різних типів та активним навантаженням, наприклад, лампою розжарювання.

• **ADC** (Analog to Digital Converter) – аналогово-цифровий перетворювач забезпечує інтерфейс для роботи з аналоговими пристроями, наприклад, з давачами, що видають аналогові електричні еквіваленти для фактичних фізичних параметрів, які потрібно контролювати.

• **DAC** (Digital to Analog Converter) – цифро-аналоговий перетворювач забезпечує інтерфейс для роботи з виконавчими пристроями.

До складу центрального процесора **CPU** входять:

− програмний лічильник команд (**PC**, Program Counter);

− арифметико-логічний пристрій (**ALU,** Arithmetic Logic Unit);

− блок регістрів загального призначення (**GPR**, General Purpose Registers) та інші елементи.

При старті МК значення *програмного лічильника* дорівнює $000, що є адресою першої команди в пам'яті програм (FLASH). МК завантажує з пам'яті програм два байти (код команди та її операнд) та передає на виконання в *декодер команд*. Подальші дії вже залежать від самої команди. Якщо це проста команда (арифметична, логічна тощо), то ця команда буде виконана, а

на наступному такті значення програмного лічильника буде збільшене, і з наступної комірки пам’яті будуть завантажені чергові два байти команди для виконання. Якщо ж зустрінеться команда переходу, тоді у програмний лічильник завантажиться адреса, що вказана у команді (абсолютний перехід), або лічильник збільшиться не на 1, а на необхідну величину (відносний перехід), і на наступному такті МК завантажить команду вже з нової адреси.

ALU безпосередньо з'єднаний тільки з регістрами загального призначення (R0-R31). Операції можуть здійснюватися над одним або двома регістрами загального призначення, або регістром та константою у другому операнді. Якщо команди працюють з константами, тоді як перший операнд можуть використовуватися лише регістри із другої половини (R16-R31) регістрів загального призначення. Команди 16-розрядного додавання та віднімання працюють тільки з чотирма останніми парами регістрів. Регістри загального призначення розміщені в адресному просторі SRAM (пам’яті даних), але для швидкої роботи з ними винесені фізично за межі SRAM.

Операції ALU підрозділяються на три основні категорії: арифметичні, логічні й операції над бітами.

Схема ядра CPU МК AVR показана на рис. 2.2.

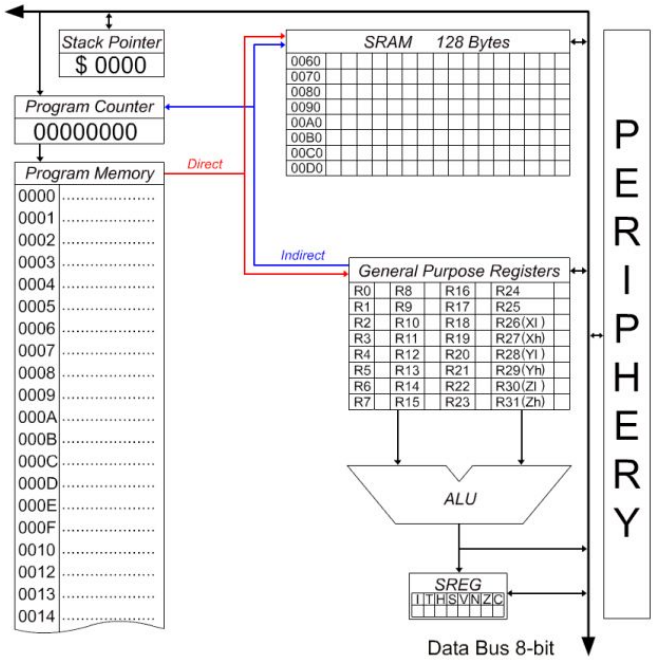


Рисунок 2.2 – Схема ядра ЦП МК AVR

Процесор формує адресу наступної команди, вибирає команду з пам'яті й організовує її виконання. Код команди має формат "слово" (16 біт) або "два слова".

До складу процесора крім програмного лічильника PC, арифметико-логічного пристрою ALU і блоку регістрів загального призначення GPR входять:

• регістр стану МК **SREG**;

• регістр-вказівник стеку **SP** або (старший SPH і SPL молодший байт);

Регістри загального призначення можна поділити на 3 групи:

• *Молодші R0…R15.* Ці регістри не можуть працювати з командами, що оперують з константами, наприклад, не можна в R0 записати число (константу), але можна скопіювати в нього число з будь-якого іншого регістра.

• *Старші R16…R31.* Повноцінні регістри, що працюють майже зі всіма командами.

• *Індексні R26…R31.* Можуть використовуватися як звичайні регістри загального призначення, але крім цього можуть утворювати регістрові 16-розрядні пари X(R26:R27), Y(R28:R29), Z(R30:R31). Такі регістрові пари використовуються як вказівники для непрямої адресації пам'яті.

Крім регістрів загального призначення в МК є регістри спеціальних функцій, які у родині AVR називаються регістрами введення/виведення (**IOR**, I/O Registers). За участю цих регістрів здійснюється:

− керування роботою МК і окремих його пристроїв;

− визначення стану МК і окремих його пристроїв;

− введення даних у МК й окремі його пристрої, виведення даних і виконуються інші функції.

Для нумерації регістрів введення/виведення використовуються номери від 0 до 63 (від $00 до $3F, де $ – показник шістнадцятирічного коду).

Кожному регістру присвоєне ім'я, пов'язане з функцією, яку виконує цей регістр. МК різних типів мають різний склад регістрів введення/виведення, при цьому регістри з однаковими номерами можуть мати різні імена.

МК AVR є пристроями синхронного типу. Дії, що виконуються у МК, прив'язані до імпульсів тактового сигналу.

Як генератор тактового сигналу GCK використовується:

− внутрішній генератор із зовнішнім кварцовим чи керамічним резонатором (XTAL);

− внутрішній RC-генератор (IRC);

− внутрішній генератор із зовнішнім RC-колом (ERC);

− зовнішній генератор (EXT).

У МК, які мають внутрішній генератор із зовнішнім резонатором (XTAL), резонатор підключається до виводів XTAL1 н XTAL2,які через конденсатори малої ємності (20-30 пФ) з'єднуються із шиною GND.

Тактова частота визначається робочою частотою резонатора. XTAL1 і XTAL2 є входом і виходом, відповідно, який інвертує підсилювач, що з використанням кварцового чи керамічного резонатора працює як вбудований генератор, як показано на рис. 2.3. При використанні зовнішнього джерела тактової частоти вивід XTAL2 повинний залишитися вільним, сигнал подається на вивід XTAL1.

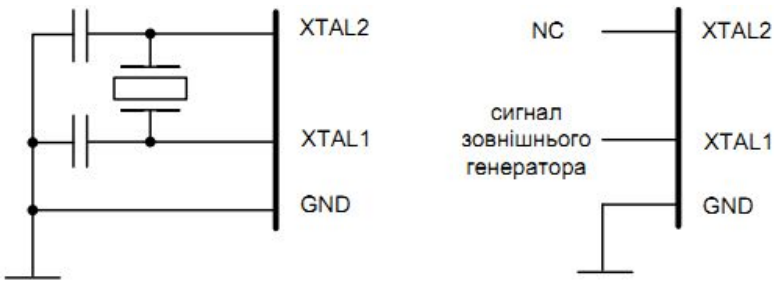


Рисунок 2.3 – Приєднання тактового генератора та зовнішнього джерела тактового сигналу

Для програмування МК AVR використовується SPI-інтерфейс: MOSI, MISO, SLK Reset, +5 В, Gnd (рис. 2.4).

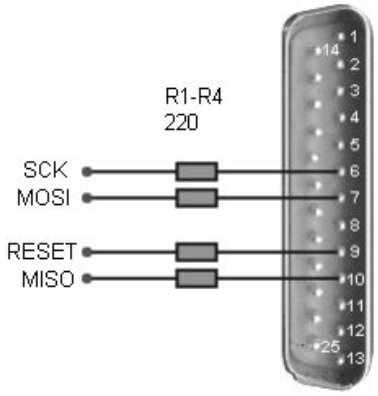
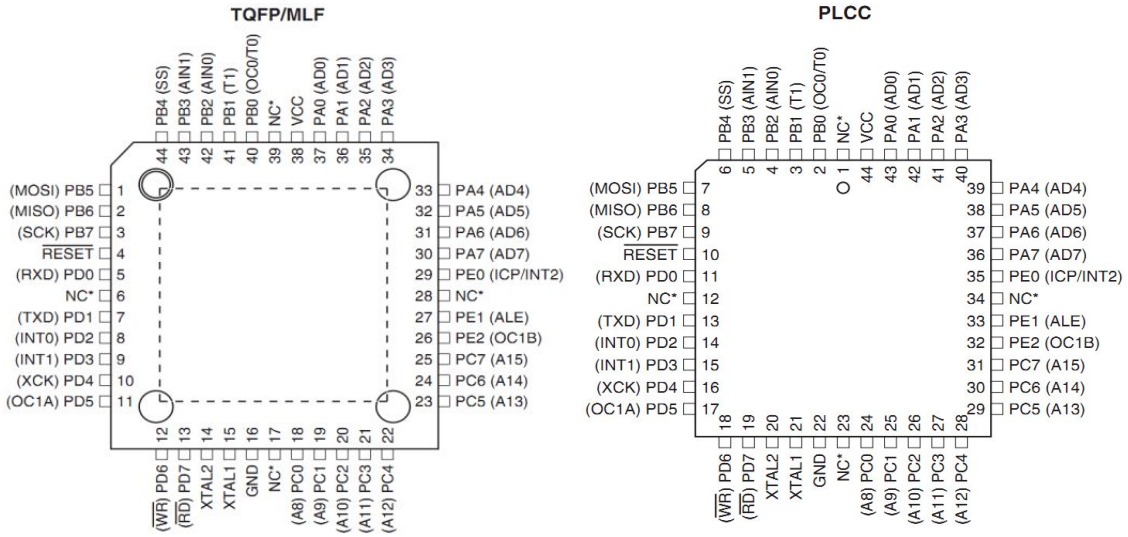


Рисунок 2.4 – Підключення МК до LPT-порту ПК для програмування через SPI-інтерфейс

Одними з бюджетних моделей родини МК ATMEGA є ATmega8/ATmega48/ATmega8515/ATmega8535/ATmega32. Зовнішній вигляд МК ATmega8515 показано на рис. 2.5, а структурна схема – на рис. 2.6.



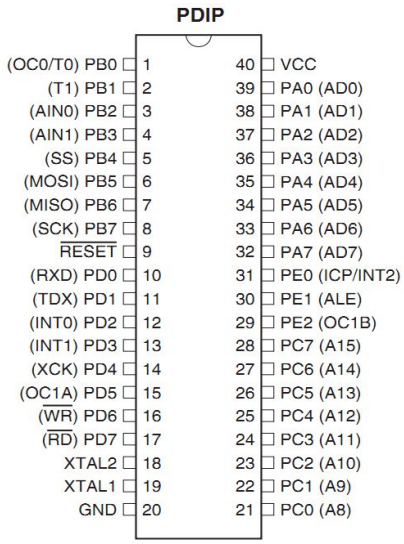


Рисунок 2.5 – Конструктивне виконання корпусів МК ATmega8515:

TQFP/MLF (Thin Quad Flat Package)– 44 виводів, розмір 11мм ×11 мм × 1мм;
PLCC Plastic J-Leaded Chip Carrier – 44 виводи, 16 мм ×16 мм × 3мм;
PDIP (Plastic Dual Inline Package) – 40 виводів, розмір 15мм×50 мм ×3 мм;

PB0-PB7 порт *B*;
PD0-PD7 порт *D*;
OC0/T0 – вихід лічильника 0/вхід таймера 0;
T1 – вхід таймера 1;
AIN0 –"+" аналоговий компаратор, *AIN1* – "-" аналоговий компаратор;
{*SS, MOSI, MISO, SCK*} – інтерфейс SPI;
RESET – вивід для подачі сигналу скидання;
{RXD, TXD} – прийом/передача даних UART;
INT0, INT1 – зовнішнє переривання 0/1;
XCK – вихід системного тактового сигналу;
{*OC1A*, *OC1B*} – вихід A/B таймера лічильника 1;
{WR, RD} – прапор запису/читання зовнішньої пам'яті;
{XTAL1, XTAL2} – вхід/вихід для зовнішнього резонатора;
GND – земля;
PA0-PA7 порт *A*;
PC0-PC7 порт *C*;
VCC – живлення;
AD0-AD7 – адреси і дані зовнішньої пам'яті;
ICP – вивід функції "Захоплення" входу таймера/лічильника;
ALE – прапор доступу до адресації зовнішньої пам'яті через порт A;
A8-A15 – адреси зовнішньої пам'яті.

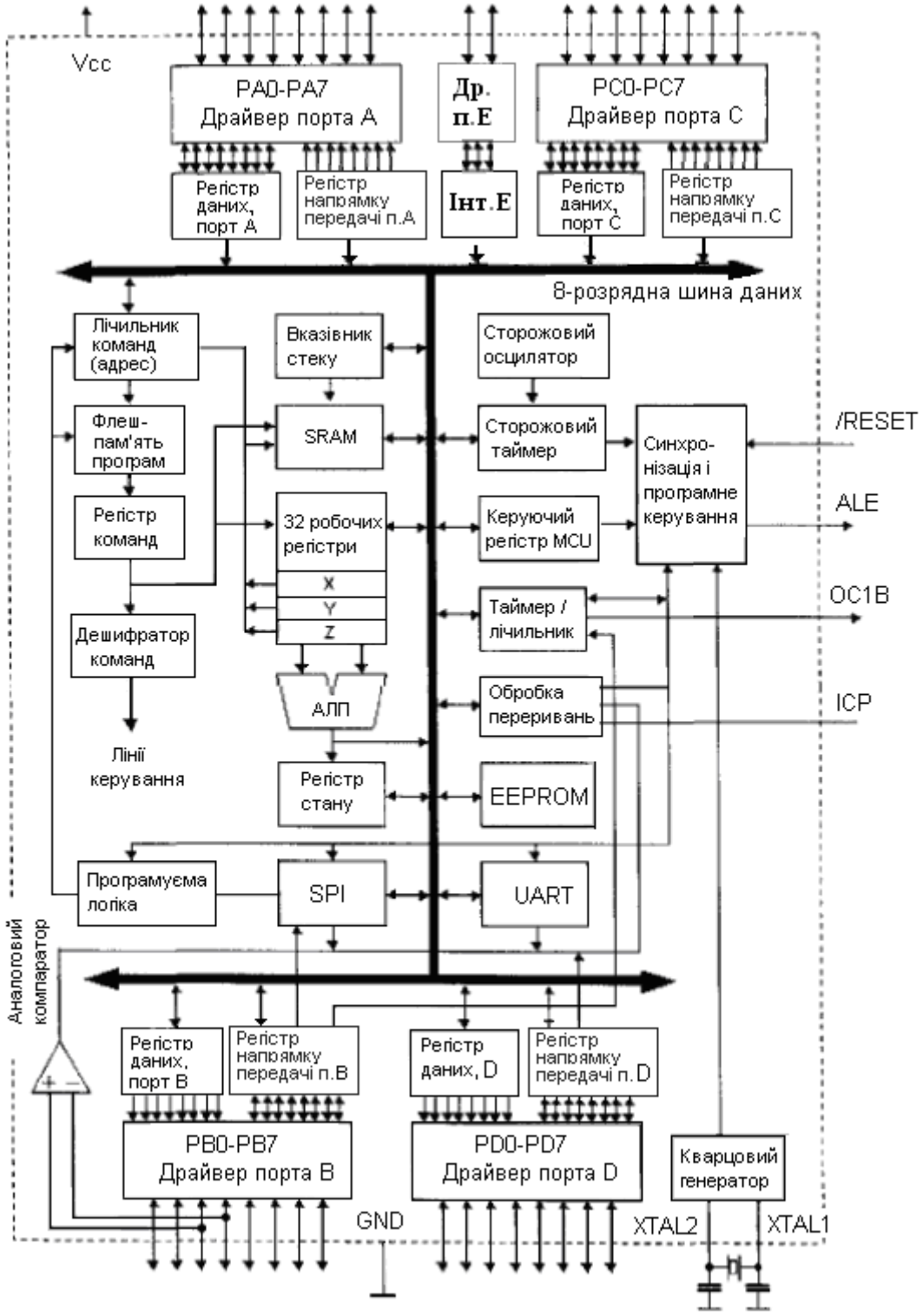


Рисунок 2.6 – Структура схема МК ATmega8515

До ядра МК ATmegs8/ATmega48/ATmega8515/ATmega8535 входять блок процесора, який об’єднує арифметико-логічний пристрій (АЛП) з регістром ознак (SREG) і пристрій керування, пам’ять програм (Flash), регістри загального призначення, пам’ять даних статичного типу (SRAM).

Пристрій керування містить схему синхронізації, регістр керування МК (MCUCR), генератор, а також регістр команд з дешифратором, програмний лічильник і вказівник стеку.

Структурні елементи МК ATmegs8/ATmega48/ATmega8515/ATmega8535мають наступні характеристики:

• кількість регістрів загального призначення:

- 32 робочі регістри (8-бітові) безпосередньо з'єднані з арифметико-логічним пристроєм.

• енергонезалежна пам'ять даних і програм:

- 8/32 кбайт 16-бітової енергонезалежної вбудованої FLESH-пам'яті програм (програмованої в системі in-system-programmable (ISP));
- 512/1024 байтів EEPROM-пам'яті даних (програмованої, енергонезалежної);
- 512/2048 байтів оперативної SRAM-пам'яті даних (енергозалежної);
- можливість програмування захисту FLESH і EEPROM пам'яті.
- інтерфейс до зовнішньої пам'яті;

• периферія:

- 8-розрядні порти введення/виведення PA, PB, PC, PD (32 програмовані лінії);
- 3-розрядний порт PE (призначений для виконання спеціальних функцій).
- послідовний асинхронний приймач-передавач USART (Universal Serial Asynchronous Receiver-Transmitter);
- послідовний синхронний інтерфейс SPI (Serial Peripheral Interface);
- TWI байтовий послідовний інтерфейс Master/Slave SPI;
- 1x8/2x8-розрядний таймер/лічильник T0;
- 1x16-розрядний таймер/лічильник T1 з можливістю порівняння;
- широтно-імпульсний модулятор (ШІМ, PWM);
- -/8-канальний 10-бітовий ADC;
- блок оброблення внутрішніх і зовнішніх переривань;
- інтегрований сторожовий таймер;
- аналоговий компаратор.

Для пам'яті програм і енергонезалежної пам'яті даних в складі МК ATmega8515 є засоби внутрішньо-системного програмування з використанням інтерфейсу SPI.

Тактова частота МК 8 МГц.

На рис. 2.7 зображена програмна модель МК ATMEL, що являє собою діаграму програмно доступних ресурсів. Центральним блоком на цій діаграмі є реєстровий файл на 32 оперативних регістра (R0-R31), безпосередньо доступних ALU. Старші регістри об'єднані парами й утворять три 16-розрядних регістри, призначених для непрямої адресації комірок пам'яті .

Всі арифметичні і логічні операції, а також частина операцій роботи з бітами виконуються в ALU тільки над вмістом оперативних регістрів. Варто звернути увагу, що команди (SUBI, SBCI, ANDI, ORI, SBR, CBR) як другий операнд використовують константу, а як перший операнд використовуються тільки регістри з другої половини реєстрового файлу (R16-R31). Команди 16-розрядного додавання з константою ADI і віднімання константи SBI як перший операнд використовують тільки регістри R24, R26, R28, R30. Під час виконання арифметичних і логічних операцій чи операцій роботи з бітами ALU формує ті чи інші ознаки результату операції, тобто встановлює або скидає біти в регістрі стану SREG.

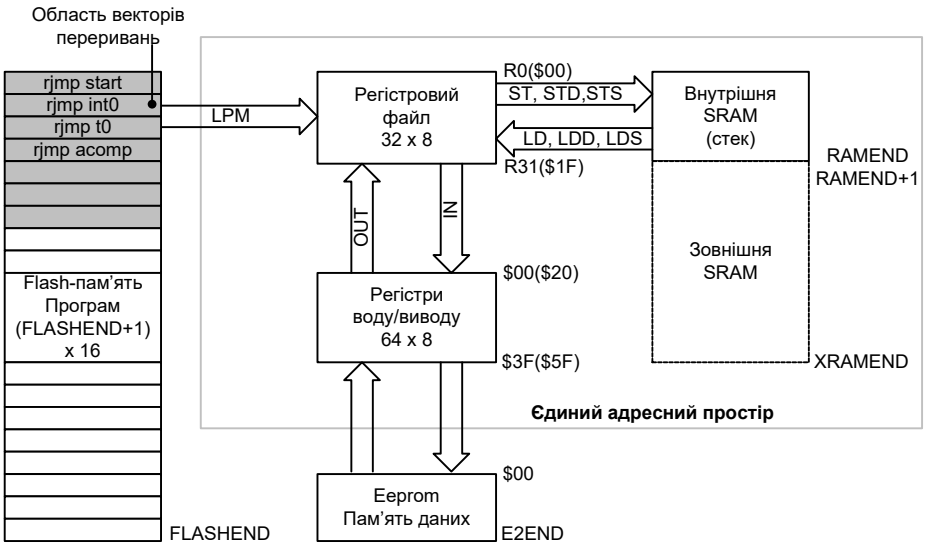


Рисунок 2.7 – Програмна модель МК ATMEL

1.2. Організація пам'яті МК ATmegs8/ATmega48/ATmega8515/ATmega8535

Пам'ять даних ділиться на: регістрову, оперативну статичну і постійну енергонезалежну FLASH пам'ять.

Пам'ять даних адресується по байтах та повністю лінійна, без будь-яких поділів на сторінки, сегменти чи банки. Регістри загального призначення, регістри керування та периферії, внутрішня оперативна пам'ять знаходяться в одному адресному просторі SRAM (рис. 2.8).

Адресний простір SRAM поділений на декілька областей:

• $0000-$001F – виділено під 32 регістри загального призначення. В асемблері до них звертаються за їхніми безпосередніми назвами R0, R1,…, R31, однак можна зчитувати та записувати у них дані за їхніми повними адресами у SRAM від 0x0000 до 0x001F за допомогою групи команд `load/store`.

• $0020-$005F – виділено під 64 регістри введення/виведення (порти, таймери, АЦП, регістри керування і т.п.). Ці регістри також мають свою внутрішню адресацію від $00 до $3F, яка використовується разом зі швидкими командами IN/OUT. Але також можна працювати з ними за допомогою команди `load/store`, використовуючи їхні повні адреси у SRAM.

• $0060-RAMEND – виділено під внутрішню статичну оперативну пам'ять даних. Значення RAMEND залежить від моделі МК. Розміщувати дані в пам'яті можна тільки починаючи з адреси $0060.

Оперативна статична пам'ять SRAM розміром 512 байт призначена для зберігання даних під час виконанні програми. Розширення адресного простору від RamEnd+1 аж до верхньої межі $FFFF можливе за рахунок підключення зовнішнього пристрою пам'яті SRAM. При виключенні напруги живлення дані в пам'яті SRAM втрачаються.

Слід зазначити, що регістри загального призначення та регістри введення/виведення не віднімають простір у пам'яті даних, а лише відсувають початок її адресації.

Стек розміщується в пам'яті SRAM, звичайно під нього виділяється область адрес, починаючи з$0060+$025F. Стек росте в сторону зменшення адрес і його розмір контролює програміст.

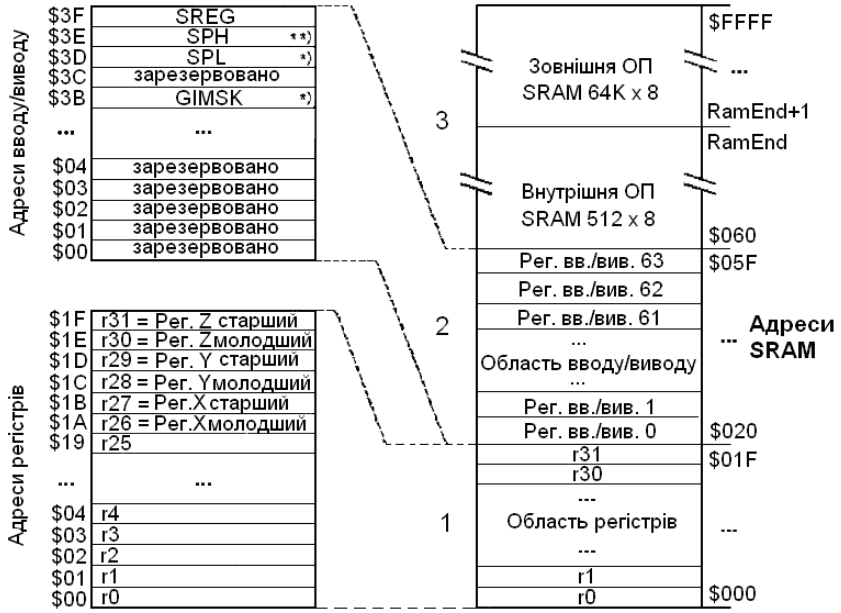


Рисунок 2.8 – Організація оперативної статичної пам’яті SRAM МК ATmega8515:
RamEnd = $001F + $005F + $200 = $0260

Для довготривалого зберігання даних, які можуть змінюватися в процесі роботи, використовується пам’ять EEPROM розміром 512 байт. Дані в EEPROM можуть бути записані при програмуванні МК. При виключенні живлення дані зберігаються.

В області введення/виведення розміщені всі регістри для програмування, керування і сигналізації про всі периферійні функції МК Atmega8515. В табл. 2.1 вказані регістри введення/виведення базової серії МК Atmel.

Таблиця 2.1 – Регістри введення/виведення

Назва	Функція
SREG	Регістр стану
SPH	Вказівник стеку, старший байт
SPL	Вказівник стеку, молодший байт
GIMSK	Загальний регістр маски переривань
GIFR	Загальний регістр прапорів переривань
TIMSK	Регістр маски переривань таймера/лічильника
TIFR	Регістр прапорів переривань таймера/лічильника
MCUCR	Регістр загального керування МК
TCCR0	Регістр керування таймером/лічильником T/C0
TCNT0	Регістр лічильник T/C0 (8 біт)
TCCR1A	Регістр керування A таймера/лічильника T/C1
TCCR1B	Регістр керування B таймера/лічильника T/C1
TCNT1H	Регістр лічильник T/C1, старший байт
RCNT1L	Регістр лічильник T/C1, молодший байт
OCR1AH	Регістр порівняння A таймера/лічильника T/C1, старший байт
OCR1AL	Регістр порівняння A таймера/лічильника T/C1, молодший байт
OCR1BH	Регістр порівняння B таймера/лічильника T/C1, старший байт
OCR1BL	Регістр порівняння B таймера/лічильника T/C1, молодший байт
ICR1H	Регістр захоплення таймера/лічильника T/C1, старший байт
ICR1L	Регістр захоплення таймера/лічильника T/C1, молодший байт

WDTCR	Регістр керування сторожовим таймером
EEARH	Регістр адреса EEPROM, старший байт
EEARL	Регістр адреса EEPROM, молодший байт
EEDR	Регістр даних EEPROM
EECR	Регістр керування EEPROM
PORTA	Регістр даних порту A
DDRA	Регістр напрямку передачі даних порту A
PINA	Виводи порту A
PORTB	Регістр даних порту B
DDRB	Регістр напрямку передачі даних порту B
PINB	Виводи порту B
PORTC	Регістр даних порту B
DDRC	Регістр напрямку передачі даних порту C
PINC	Виводи порту C
PORTD	Регістр даних порту D
DDRD	Регістр напрямку передачі даних порту D
PIND	Виводи порту D
SPDR	Регістр введення/виведення даних SPI
SPSR	Регістр стану SPI
SPCR	Регістр керування SPI
UDR	Регістр даних UART
USR	Регістр стану UART
UCR	Регістр керування UART
UBRR	Регістр швидкості передачі даних UART
ACSR	Регістр керування і стану аналогового компаратора

Для доступу до регістрів введення/виведення краще використовувати команди `in` і `out`. Команда `out` виводить один байт з одного з 32-х регістрів загального призначення у регістр введення/виведення. Команда `in` зчитує байт з одного регістра введення/виведення і виводить у один з 32-х регістрів загального призначення.

Область введення/виведення має свою адресацію з 0x0000 по 0x003F. При такій адресації області введення/виведення необхідно використовувати спеціальні команди `in, out, sbi, cbi, sbis, sbic`. У регістрів області введення/виведення з (0x0000) – (0x001F) (0-31) можна змінити окремі розряди командами `sbi, cbi`, або опитати командами `sbis, sbic`. Область введення/виведення може також адресуватися як загальна область пам'яті SRAM (при цьому необхідно збільшити адресу на 0x20). При такій адресації використовуються команди `sts, lds`.

Вводити чи зчитувати значення з регістрів введення/виведення (периферії) можна лише через регістри загального призначення `r0-r31`. Тобто, для того щоб записати значення у якийсь з периферійних регістрів, спочатку присвоюється це значення якомусь з регістрів загального призначення, а потім з цього регістра пересилається у периферійний регістр, рис. 2.9.

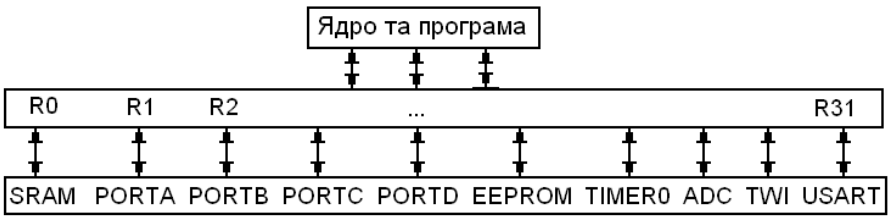


Рисунок 2.9 – Регістри загального призначення як посередники

Назви регістрів області введення/виведення та їх адреси показані в табл. 2.2.

Таблиця 2.2 – Назви регістрів області введення/виведення та їх адреси

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
$3F ($5F)	SREG	I	T	H	S	V	N	Z	C	9
$3E ($5E)	SPH	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	11
$3D ($5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	11
$3C ($5C)	Reserved	-								
$3B ($5B)	GICR	INT1	INT0	INT2	-	-	-	IVSEL	IVCE	56, 77
$3A ($5A)	GIFR	INTF1	INTF0	INTF2	-	-	-	-	-	78
$39 ($59)	TIMSK	TOIE1	OCIE1A	OCIE1B	-	TICIE1	-	TOIE0	OCIE0	92, 123
$38 ($58)	TIFR	TOV1	OCF1A	OCF1B	-	ICF1	-	TOV0	OCF0	92, 124
$37 ($57)	SPMCR	SPMIE	RWWSB	-	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	168
$36 ($56)	EMCUCR	SM0	SRL2	SRL1	SRL0	SRW01	SRW00	SRW11	ISC2	28,41,77
$35 ($55)	MCUCR	SRE	SRW10	SE	SM1	ISC11	ISC10	ISC01	ISC00	28,40,76
$34 ($54)	MCUCSR	-	-	SM2	-	WDRF	BORF	EXTRF	PORF	40,48
$33 ($53)	TCCR0	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	90
$32 ($52)	TCNT0	Timer/Counter0 (8 Bits)								92
$31 ($51)	OCR0	Timer/Counter0 Output Compare Register								92
$30 ($50)	SFIOR	-	XMBK	XMM2	XMM1	XMM0	PUD	-	PSR10	30,65,95
$2F ($4F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	118
$2E ($4E)	TCCR1B	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	121
$2D ($4D)	TCNT1H	Timer/Counter1 - Counter Register High Byte								122
$2C ($4C)	TCNT1L	Timer/Counter1 - Counter Register Low Byte								122
$2B ($4B)	OCR1AH	Timer/Counter1 - Output Compare Register A High Byte								122
$2A ($4A)	OCR1AL	Timer/Counter1 - Output Compare Register A Low Byte								122
$29 ($49)	OCR1BH	Timer/Counter1 - Output Compare Register B High Byte								122
$28 ($48)	OCR1BL	Timer/Counter1 - Output Compare Register B Low Byte								122
$27 ($47)	Reserved	-								-
$26 ($46)	Reserved	-								-
$25 ($45)	ICR1H	Timer/Counter1 - Input Capture Register High Byte								123
$24 ($44)	ICR1L	Timer/Counter1 - Input Capture Register Low Byte								123
$23 ($43)	Reserved	-								-
$22 ($42)	Reserved	-								-
$21 ($41)	WDTCR	-	-	-	WDCE	WDE	WDP2	WDP1	WDP0	50
$20[1] ($40)[1]	UBRRH	URSEL	-	-	-	UBRR[11:8]				157
	UCSRC	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	155
$1F ($3F)	EEARH	-	-	-	-	-	-	-	EEAR8	18
$1E ($3E)	EEARL	EEPROM Address Register Low Byte								18
$1D ($3D)	EEDR	EEPROM Data Register								19
$1C ($3C)	EECR	-	-	-	-	EERIE	EEMWE	EEWE	EERE	19
$1B ($3B)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	74
$1A ($3A)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	74
$19 ($39)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	74
$18 ($38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	74
$17 ($37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	74
$16 ($36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	74
$15 ($35)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	74
$14 ($34)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	74
$13 ($33)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	75
$12 ($32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	75
$11 ($31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	75
$10 ($30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	75
$0F ($2F)	SPDR	SPI Data Register								131
$0E ($2E)	SPSR	SPIF	WCOL	-	-	-	-	-	SPI2X	131
$0D ($2D)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	129
$0C ($2C)	UDR	USART I/O Data Register								153
$0B ($2B)	UCSRA	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	153
$0A ($2A)	UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	154
$09 ($29)	UBRRL	USART Baud Rate Register Low Byte								157
$08 ($28)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	162
$07 ($27)	PORTE	-	-	-	-	-	PORTE2	PORTE1	PORTE0	75
$06 ($26)	DDRE	-	-	-	-	-	DDE2	DDE1	DDE0	75
$05 ($25)	PINE	-	-	-	-	-	PINE2	PINE1	PINE0	75
$04 ($24)	OSCCAL	Oscillator Calibration Register								38

$04-$3F – адресація області введення/виведення (периферії). ($24-$05F) – адресація всієї області SRAM

1.3. Регістр стану SREG

Під час виконання арифметичних, логічних та порозрядних операцій ALU формує певні інформативні ознаки результату виконання як біти регістру стану **SREG** (Status Register), що потім можуть бути використані в програмі для подальших арифметично-логічних операцій чи команд умовних переходів. Такі інформативні біти часто називають прапорцями. Їх всього вісім, і кожен з них може розпізнаватися особливими командами, що визначають подальший хід виконання програми (наприклад, `brcs, brhc, brtc, brie` і так далі).

Регістр стану містить біти (прапори) умов і доступний для запису та читання. Після скидання він ініціалізується нулями.

	7	6	5	4	3	2	1	0	
$3F($5F)	I	T	H	S	V	N	Z	C	SREG

Значення окремих бітів:

Біт 0 - C (прапор перенесення, Carry) – вказує на перенесення із старшого розряду в результаті виконання якої-небудь арифметичної чи логічної операції. Прапор перенесення встановлюється різними командами і може опитуватися напряму. До операцій, які змінюють цей прапор, відносяться додавання, віднімання, циклічні зсуви і деякі логічні операції. Перенесення із старшого розряду виникає при додаванні двох однобайтних чисел. Прапор C встановлюється і при відніманні, якщо у старшому розряді виникає позика.

Прапор встановлюється/скидається командами `sec/clc`.

Біт 1 - Z (прапор нуля, Zero) – завжди встановлюється, якщо результат якої-небудь арифметичної або логічної операції дорівнює нулю. У іншому випадку прапор скидається.

Прапор встановлюється/скидається командами `sez/clz`.

Біт 2 - N (прапор негативного результату, Negative) – вказує на негативний результат виконання арифметичної або логічної операції (тобто в старшому розряді присутня “1”). Цей прапор виставляється, якщо отриманий результат попадає в діапазон значень [-128, -1] або [128, 255]. Так числа -21 і 255 для асемблера є ідентичними, оскільки їх двійкове подання однакове 0b1110_1011.

Прапор встановлюється/скидається командами `sen/cln`.

Біт 3 - V (прапор переповнення при обчисленнях в доповняльних кодах, Two’s Complement Overflow) – встановлюється при виході за межі [-128, 127] у двох випадках:

• обидва операнди позитивні (біт 7 = 0), а при додавання виникає перенесення з розряду 6 у розряд 7;

• обидва операнди негативні (біт 7 = 1), а в результаті розряд 7 позитивний.

Прапор встановлюється/скидається командами `sev/clv`.

Біт 4 - S (прапор знаку, Sign) – є результатом суми за модулем 2 між прапорами N і V (логічна порозрядна операція `N xor V`). Прапор S використовується для визначення фактичного знаку результату виконання арифметичної операції. Якщо не виникає переповнення в доповняльних кодах (V=0), то як знак приймається значення N. Якщо V=1, то прапор знаку приймає інвертоване значення N.

Прапор встановлюється/скидається командами `ses/cls`.

Біт 5 - H (прапор перенесення з молодшої половини байту, Half Carry) – вказує на перенесення чи позику з молодшої половини байту (розряди 0-3). В іншому випадку, він скидається.

Прапор встановлюється/скидається командами `seh/clh`.

Біт 6 - T (прапор копіювання, Bit Copy Storage) – використовується при роботі з бітами регістрів загального призначення в командах `bld (bit load)` і `bls (bit store)` Команда `bld` встановлює біт T у заданий біт регістра загального призначення, а команда `bls` читає заданий біт регістра загального призначення і записує його в біт T.

Прапор встановлюється/скидається командами `set/clt`.

```
set         ;встановити біт T
bld r16,5   ;встановити 5-й біт у r16

clt         ;очистити біт T
bld r16,5   ;очистити 5-й біт у r16
bls r16,6   ;копіювати 6-й біт r16 у прапор T
```

Біт 7 - I (загальний дозвіл переривань, Global Interrupt Enable) – якщо встановлений в “0” то призупиняються будь-які переривання, якщо в “1” – то дозволені переривання активуються. Дозвіл окремих переривань реалізується за допомогою додаткових регістрів GIMSK і TIMSK.

Прапор встановлюється/скидається командами `sei/cli`.

1.4. Регістр керування МК (MCUCR)

Регістр керування МК MCUCR (Microcontroller Unit) містить розряди керування загальними функціями МК. Регістр доступний для читання і запису.

	7	6	5	4	3	2	1	0	
$35($55)	SRE	SRW	SE	SM	ISC11	ISC10	ISC01	ISC00	MCUCR

Якщо розряд SRE (External **SR**AM **E**nable) встановлений в лог. 1, то дозволений доступ до зовнішньої пам’яті SRAM, а порти A, C (а також виводи PD6, PD7 порту D) будуть переключені в їх альтернативні функції AD0,…,AD7, A8,…,A15, /WR, /RD.

Якщо в розряд SRW (External **SR**AM **W**ait State) записана лог. 1, то в послідовність звернення до зовнішньої пам’яті SRAM буде добавлено стан очікування в один системний такт. Якщо в розряд SRW записано лог. 0, то доступ до зовнішньої пам’яті SRAM виконується за два такти.

Розряд SE (**S**leep **E**nable) необхідно встановити в лог.1, щоб дозволити команду sleep.

Розряд SM (**S**leep **M**ode) задає вибір між двома станами пониженого енергоспоживання ”Idle”, “Power Down”. Розряди ISC11-ISC00 визначають спосіб активації зовнішнього переривання.

1.5. Регістр стеку SP, SPH, SPL

Стек МК AVR служить для зберігання адрес повернення при виклику підпрограм, а також для передачі параметрів в підпрограми. Крім цього, в стеку, як у буфері, тимчасово зберігаються змінні і проміжні результати.

Регістр стеку SP реалізований з двох 8-бітових регістрів SPH, SPL. В регістрах зберігається адреса, яка вказує на верхівку стеку. В області стеку зберігаються тимчасові дані, локальні змінні, адреса повернення після виклику підпрограм і переривань. При збільшенні стеку адреси комірок зменшуються.

Простір стеку необхідно визначити в програмі до виклику підпрограм або переривань. Вказівник стеку завжди має вказувати на адресу більшу від $60. Вказівник стеку зменшується на 1 при виконанні команди PUSH і збільшується на 1 при виконанні команди POP. Вказівник стеку зменшується на 2 при виклику підпрограм або переривань і збільшується на 2 при поверненні з підпрограми командою RET та при поверненні з переривань командою RETI.

1.6. Регістри інших структурних елементів

GICR (**G**eneral **I**nterrupt **C**ontrol **R**egister) – регістр контролю загальним перериванням. Регістр контролює розміщення таблиці вектора переривань.

GIFR (**G**eneral **I**nterrupt **F**lag **R**egister) – регістр прапорів загального переривання.

TIMSK (**T**imer/Counter **i**nterrupt **ma**s**k** **r**egister) – регістр маски переривань Таймера/Лічильника.

TIFR (**T**imer/Counter **i**nterrupt **f**lag **r**egister)– регістр прапорів переривань Таймера/Лічильника.

SPMCR (**S**tore **p**rogram **m**emory **c**ontrol **r**egister) – регістр містить біти для керування операціями завантажувача (boot loader).

EMCUCR (**E**xtended **MCUc**ontrol **r**egister) – регістр розширеного контролю МК.

MCUCR (**MCUc**ontrol **r**egister) – регістр контролю МК.

MCUCSR (**MCUc**ontrol and **s**tatus **r**egister) – регістр контролю і статусу МК.

TCCR0 (**T**imer/**C**ounter **c**ontrol **r**egister) – регістр контролю таймера/лічильника 0 (8-біт).

TCNT0 (**T**imer/**Cou**n**t**er register) – регістр таймера/лічильника 0.

OCR0 (**O**utput **c**ompare **r**egister) – регістр порівняння виходу таймера/лічильника 0.

SFIOR (**S**pecial **f**unctio **IOr**egister) – регістр спеціальних функцій введення/виведення.

TCCR1A (**T**imer/**C**ounter1 **c**ontrol **r**egister **A**) – регістр А контролю таймера/лічильника 1.

TCCR1B (**T**imer/**C**ounter1 **c**ontrol **r**egister **B**) – регістр В контролю таймера/лічильника 1.

TCNT1H (**T**imer/**Cou**n**t**er1 [15:8]) – старший регістр таймера/лічильника 1.

TCNT1L (**T**imer/**Cou**n**t**er1 [7:0]) – молодший регістр таймера/лічильника 1.

OCR1AH (**O**utput **c**ompare **r**egister **1 A**) – старший регістр 1 А порівняння виходу.

OCR1AL (**O**utput **c**ompare **r**egister **1 A**) – молодший регістр 1 А порівняння виходу.

OCR1BH (**O**utput **c**ompare **r**egister **1 B**) – старший регістр 1 В порівняння виходу.

OCR1BL (**O**utput **c**ompare **r**egister **1 B**) – молодший регістр 1 В порівняння виходу.

ICR1H (**I**nput **c**apture **r**egister **1** [15:8]) – старший регістр 1 захоплення входу.

ICR1L (**I**nput **c**apture **r**egister **1** [7:0]) – молодший регістр 1 захоплення входу.

WDTCR (**w**atch**d**og **t**imer **c**ontrol **r**egister) – сторожовий таймер.

UBRRH (**U**SART **b**aund **r**ate **r**egister) – старший регістр швидкості передачі даних.

UCSRC (**U**SART **c**ontrol and **s**tatus **r**egister) – регістр контролю і статусу UART.

EEARH (**EE**PROM **a**ddress **r**egister) – старший регістр адресації EEPROM.

EEARL (**EE**PROM **a**ddress **r**egister) – молодший регістр адресації EEPROM.

EEDR (**EE**PROM **d**ata **r**egister) – регістр даних EEPROM.

EECR (**EE**PROM **c**ontrol **r**egister) – регістр контролю EEPROM.

PORTX (**Port X** data register) – регістр даних порту X (де X=A,B,C,D,E).

DDRX (Port X **d**ata **d**irection **r**egister) – регістр напрямку передачі даних порту X (де X=A,B,C,D, E).

PINX (Port X input **pin** address) – регістр адресу вхідних виводів порту X (де X=A,B,C,D,E).

SPDR (**SP**I **d**ata **r**egister) – регістр даних SPI.

SPSR (**SP**I **s**tatus **r**egister) – регістр статусу SPI.

UDR (**U**SART I/O **d**ata **r**egister) – регістр вводу/виводу USART.

UCSRA (**U**SART **c**ontrol and **s**tatus **r**egister A) – регістр А контролю і статусу USART.

UCSRB (**U**SART **c**ontrol and **s**tatus **r**egister B) – регістр В контролю і статусу USART.

UBRRL – (**U**SART **b**aund **r**ate **r**egister) – молодший регістр швидкості передачі даних

ACSR (**A**nalog comparator **c**ontrol and **s**tatus **r**egister) – регістр контролю і статусу аналогового компаратора.

OSCCAL (**osc**illator **c**alibration **r**egister) – регістр калібрування осцилятора.

1.7. Паралельні порти введення/виведення

Всі паралельні порти введення/виведення A, B, C, D є 8-розрядними і двонаправленими. Кожен із виводів може працювати як на введення, так і на виведення інформації.

На рис. 2.10 зображено еквівалентну схему вхідного ланцюга одного розряду порту введення/виведення. Всі виводи порту мають індивідуальні резистори навантаження, вхідні схеми кожної лінії мають по два захисних діоди.

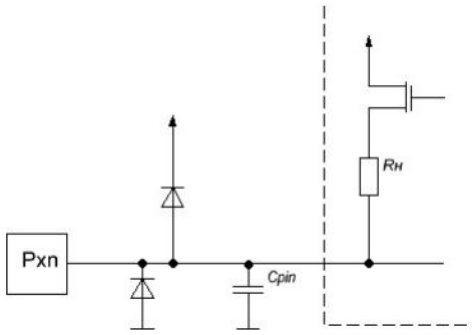


Рисунок 2.10 – Еквівалентна схема вхідних кіл розряду порту введення/виведення

Для кожного порту введення-виведення в МК існує три спеціальних регістри:

− PORTX – регістр даних;

− DDRX – регістр керування;

− PINX – регістр безпосереднього читання стану лінії порту, де «X» – назва порту (A, B, C, D).

Кожен розряд порту зв’язаний з трьома розрядами спеціальних регістрів:

− DDXn – розряд номеру n регістра DDRX;

− PORTXn – розряд номеру n регістра PORTX;

При зверненні до портів використовується регістр напрямку передачі даних DDXn, регістр порту PORTXn, вивід PinXn. Порти альтернативно використовуються для виконання інших особливих функцій, наприклад, для мультиплексованої шини адрес і даних зовнішньої пам’яті SRAM, входів таймерів, переривань, аналогових компараторів, послідовних інтерфейсів SPI, інтерфейсів USART, виходів таймерів.

1.8. Таймери/лічильники

8-розрядний таймер/лічильник T/C0 (TCNT0) з регістрами TCCR0, TIMSK, TIFR використовується для відліку часу або як лічильник для підрахунку числа імпульсів на виводі входу/виходу.

16-розрядний таймер/лічильник T/C1 (TCNT1) з регістрами TCCR1A, TCCR1B, TIMSK, TIFR, ICR1, OCR1A, OCR1B може розв'язувати більш складні задачі. Крім регістра захоплення він має два вихідних регістри порівняння. Вміст вихідного регістра постійно апаратно порівнюється з вмістом лічильника, і при співпадінні активується наперед визначена дія для відповідного виводу. За допомогою регістра порівняння можна виробляти два вихідних сигнали, модульованих по ширині імпульсу. При цьому може бути запрограмована роздільна здатність 8, 9 або 10 розрядів.

За допомогою функції захоплення на вході шляхом запуску зовнішнього сигналу на вході ICP поточний стан лічильника можна зберегти в регістрі ICR1. Альтернативно на вхід ICP, як джерело запуску для регістра захоплення, може бути приєднаний аналоговий компаратор.

Таймер/лічильник T/C1 в нормальному режимі працює як підсумовуючий лічильник. При роботі в режимі широтно-імпульсного модулятора він працює як зворотний і підсумовуючий лічильник.

1.9. Сторожовий таймер

Для запобігання переходу МК в режим нескінченого циклу використовується сторожовий таймер. Якщо за заданий час затримки програма користувача не виконає команду скидання системи, то це зробить сторожовий таймер. Після скидання сторожового таймера відлік часу затримки відновлюється. Якщо потрібно контролювати хід виконання програми, то програміст повинен активувати сторожовий таймер і через регулярні відрізки часу включати в програмі команду скидання, яка забезпечує своєчасне скидання перед початком нового відліку часу.

Якщо в простій програмі контроль за допомогою сторожового таймера не потрібен, то його можна відключити.

1.10. USART

МК ATmega8515 має один апаратний USART (Universal Synchronous and Asynchronous serial receiver and transmitter) – універсальний синхронний і асинхронний послідовний приймач-передавач. USART передає блоки (фрейми) даних, які містять стартовий біт, біти даних (5,6,7,8 або 9), біт парності, стоп-біт. Передача блоків даних здійснюється між передавачем і приймачем із вибраною швидкістю (максимум 250 kbps).

Основною областю застосування асинхронної передачі даних, як правило, є не обмін даними в складі схеми, а комунікація між блоками, розділеними просторово і з признаками власного інтелекту. Наприклад, зв'язок між персональним комп'ютером і друкаркою, модемом, або реєстратором.

1.11. Послідовний інтерфейс SPI

Послідовний інтерфейс периферійних пристроїв SPI (Serial peripheral Interface) використовується для комунікації МК з периферійними блоками, такими як зсувові регістри, символо-цифрові модулі індикації, системи реєстрації даних, інші мікропроцесорні системи.

Через інтерфейс SPI можна швидко і просто обмінюватися даними між керуючим МК (master) і підпорядкованим МК (slave).

1.12. Інтегрований аналоговий компаратор

Інтегрований аналоговий компаратор порівнює вхідну напругу на своєму неінвертуючому вході AIN0 з вхідною напругою на інвертуючому вході AIN1. Як тільки напруга на неінвертуючому вході AIN0 стане більшою, ніж на інвертуючому AIN1, на виході ACO встановлюється лог. 1.

Вихідний сигнал компаратора може бути використаний для спрацювання функції захоплення на вході T/C1. Крім того, через аналоговий компаратор можна викликати переривання.

2. Базова структура МК AVR DB

Структурна схем МК AVR32/64/128DBx із родини МК AVR DB показана на рис. 2.11.

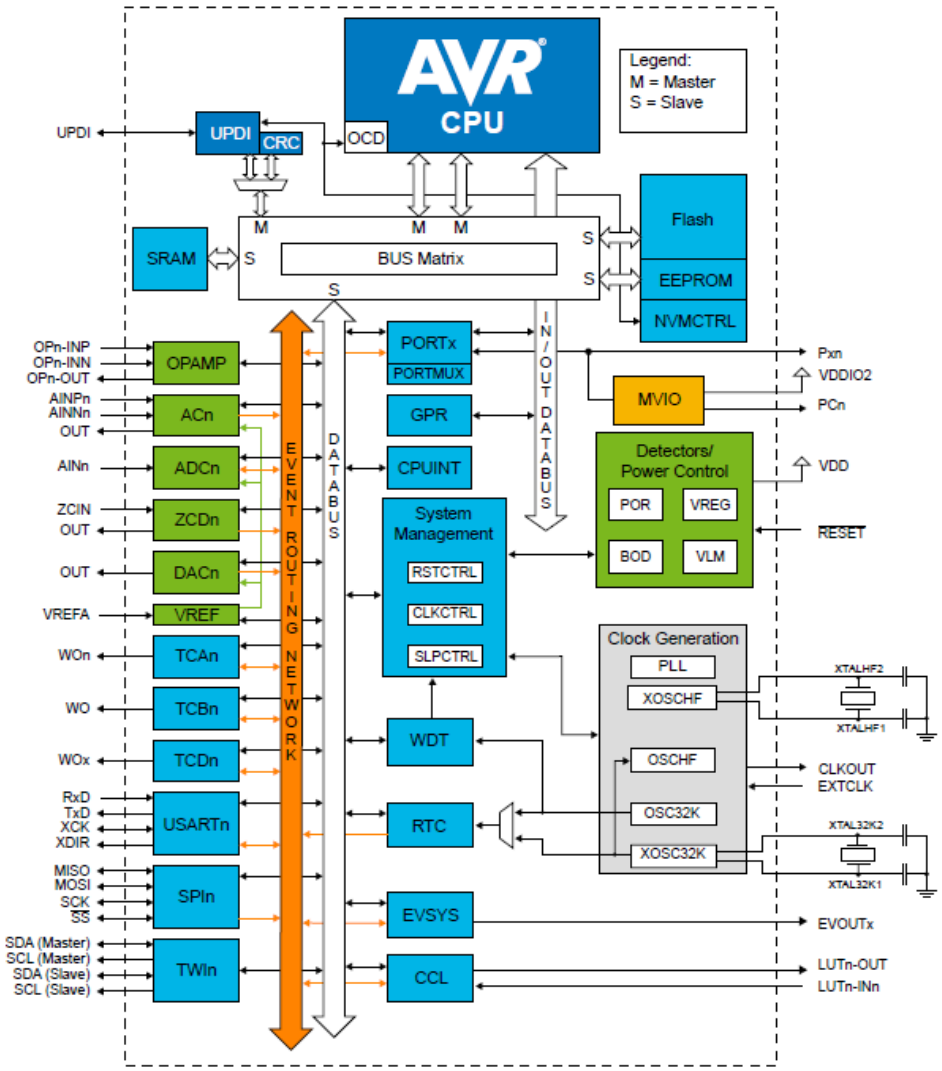


Рисунок 2.11 – Структурна схема МК AVR128DBx

Склад периферії МК AVR-DB показаний в табл. 2.3.

Таблиця 2.3 – Периферія МК AVR-DB

Функціональні блоки	AVR32/64/ 128DB28	AVR32/64/ 128DB32	AVR32/64/ 128DB48	AVR64/128 DB64
Макс. частота, МГц	24	24	24	24
16-біт Timer/Counter типу A (TCA)	1	1	2	2
16-біт Timer/Counter типу B (TCA)	3	3	4	5
16-біт Timer/Counter типу D (TCA)	1	1	1	1
USART	1	1	1	1
SPI	2	2	2	2
12-бітовий диференційний ADC (каналів)	1 (9)	1 (13)	1 (18)	1 (22)
10-бітовий DAC (виходів)	1 (1)	1 (1)	1 (1)	1 (1)
Аналоговий компаратор (C)	3	3	3	3
Детектор перетину нуля (ZCD)	1	1	2	2
Контролер периферії з сенсорами дотику (PTC)	-	-	-	-
Операційний підсилювач (OP)	2	2	3	3
Конфігурована замовником логіка на LUT-таблицях (CCL LUT)	4	4	6	6
Сторожовий таймер (WD)	1	1	1	1
Канали системи подій (EVSYS)	8	8	10	10
Контакти введення/виведення	22	26	41	55
Порти	PA[7:0], PC[3:0], PD[7:1], PF[6,1,0]	PA[7:0], PC[3:0], PD[7:1], PF[6:0]	PA[7:0], PB[5:0], PC[7:0], PD[7:0], PE[3:0], PF[6:0]	PA[7:0], PB[7:0], PC[7:0], PD[7:0], PE[7:0], PF[6:0], PG[7:0]
Зовнішні переривання	22	26	41	55
Контрольні суми CRCSCAN	1	1	1	1
Уніфікований інтерфейс налаштування і налагодження (UPDI)	1	1	1	1

Джерела живлення.

Родина AVR DB має декілька джерел живлення:

- VDD: живлення ліній I/O, XOSCHF і внутрішнього регулятора напруги;

- AVDD: живлення ліній I/O, XOSCHF (зовнішній 32768 кГц осцилятор) і аналогова периферія;

- VDDIO2: живлення ліній I/O, опційно як диференційна напруга від V_{DD}.

Центральний процесорний пристрій (CPU).

Особливості:

• 8-розрядний високопродуктивний процесор AVR RISC:
  – 135 інструкцій;
  – апаратний перемножувач;
• 32 8-розрядних регістри, безпосередньо підключені до АЛП;
• стек в оперативній пам'яті;
• покажчик стека доступний у просторі пам'яті вводу-виводу;
• пряма адресація до 64 КБ уніфікованої пам'яті;
• ефективна підтримка 8-, 16- та 32-бітової арифметики;
• захист змін конфігурації для системно-важливих функцій;
• підтримка вбудованого налагодження (OCD):
  – дві апаратні точки зупинки;
  – зміна потоку, переривання та точки зупинки програмного забезпечення;
  – зчитування під час виконання регістра покажчика стеку (SP), програмного лічильника (PC) і регістра стану (SREG);
  – реєстрація файлу для читання та запису в режимі зупинки.

Пам’ять.

Основна пам'ять МК AVR DB – оперативна пам'ять даних SRAM, пам'ять даних EEPROM та флеш-пам'ять програм. Крім того, периферійні регістри розташовані в просторі пам'яті введення-виведення I/O. Карта пам’яті показана на рис. 2.12.

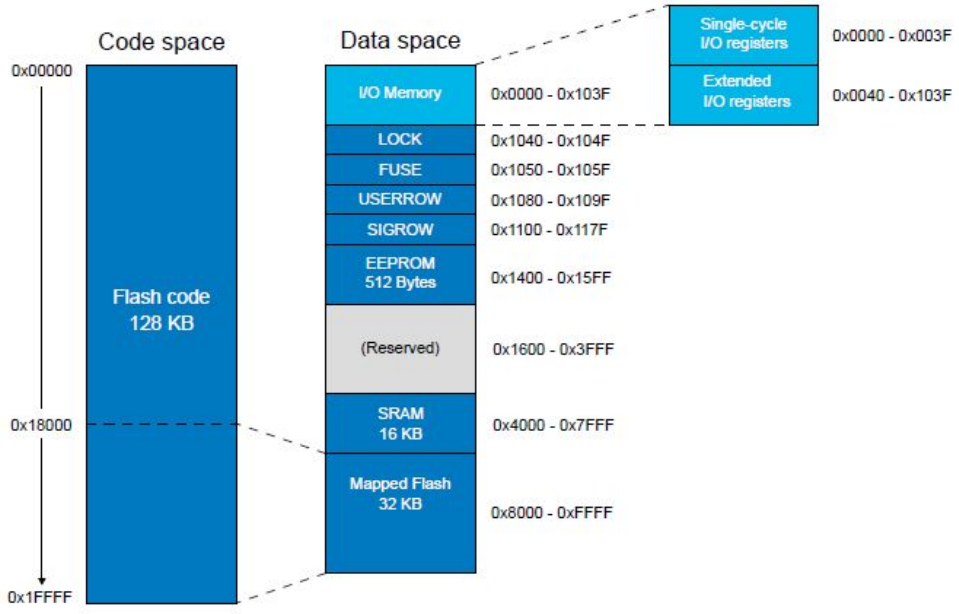


Рисунок 2.12 – Карта пам’яті

Регістри загального призначення (GPR).

МК AVR DB має чотири регістри загального призначення . Ці регістри можна використовувати для зберігання будь-якої інформацію, і вони особливо корисні для зберігання

глобальних змінних і прапорів переривання. Інтерпретація бітових значень повністю визначається програмним забезпеченням. Регістри загального призначення, які знаходяться в діапазоні адрес 0x001C - 0x001F, є побітово доступними за допомогою інструкцій SBI, CBI, SBIS та SBIC.

Нові функціональні блоки, які добавлені в МК AVR DB.

Контролер енергонезалежної пам’яті (NVMCTRL).

Контролер енергонезалежної пам’яті є інтерфейсом між центральним процесором і енергонезалежною пам’яттю (Flash-пам’ять, EEPROM, рядок підпису, рядок користувача та запобіжники). Це перепрограмовані блоки пам'яті, які зберігають свої значення, навіть якщо вони не живляться. Flash-пам’ять в основному використовується для зберігання програм, але також може використовуватися для зберігання даних. The EEPROM пам’ять використовується для зберігання даних і може бути запрограмована, поки центральний процесор виконує програму з Flash-пам’яті.

Контролер синхросигналів (CLKCTRL).

Контролер синхронізації контролює, розподіляє та попередньо масштабує сигнали синхронізації від доступних осциляторів. CLKCTRL підтримує внутрішні та зовнішні джерела синхронізації. CLKCTRL базується на системі автоматичного запиту джерела синхронізації, яка реалізована в усіх периферійних пристроях. Периферійні пристрої автоматично запитуватимуть необхідні джерела синхронізації. Запит направляється до коректного джерела синхронізації, якщо доступні декілька джерел синхронізації.

Основний тактовий сигнал (CLK_MAIN) використовується центральним процесором, оперативною пам'яттю та всіма периферійними пристроями, підключеними до шини введення/виведення. Головне джерело сигналів можна вибрати та попередньо масштабувати. Деякі периферійні пристрої можуть використовувати те саме джерело синхронізації, що й основний годинник, або працювати асинхронно до основного домену синхронізації.

Контролер сну (SLPCTRL).

Контролер сну налаштовує режим енергоспоживання і має три режими сну: бездіяльність (idle), очікування (stendby), вимкнення живлення (power-down).

Режими сну використовуються для вимкнення периферійних пристроїв і доменів синхронізації в пристрої з метою економії енергії. Контролер сну (SLPCTRL) контролює та обробляє переходи між активним і сплячим режимами. Доступно чотири режими: один активний режим, у якому виконується програмне забезпечення, і три режими сну.

Усі режими сну доступні, і в них можна перейти з активного режиму. В активному режимі ЦП виконується код програми. Коли пристрій переходить в сплячий режим, виконання програми припиняється. Код програми вирішує, в який режим сну перейти і коли. Переривання використовуються для виведення пристрою зі сну. Доступні джерела пробудження переривань залежать від налаштованого режим сну. Коли виникає переривання, пристрій вийде з режиму сну та виконає процедуру обслуговування переривань перед продовженням нормального виконання програми з першої інструкції після інструкції SLEEP. Будь-який RESET виведе пристрій з режиму сну.

Вміст файлу реєстру, SRAM і регістрів, зберігається під час сну. Якщо RESET відбувається під час сну, пристрій скинутий і запуститься на виконання з вектора Reset.

Контролер скидання RESET (RSTCTRL).

Особливості:

• повертає пристрій до початкового стану після скидання;

• ідентифікує джерело попереднього скидання;

• джерела скидання джерел живлення:

– скидання при включенні живлення (POR);

– скидання при зниженні напруги живлення менше допустимого (BOD);

• джерела скидання користувача:

– зовнішнє скидання (RESET);

– скидання сторожового таймера (WDT);

– скидання програмного забезпечення (SWRST);

– скидання уніфікованого інтерфейсу програм і налагодження (UPDI).

Контролер скидання (RSTCTRL) керує скиданням пристрою. Отримавши запит на скидання, він встановлює пристрій у початковий стан і дозволяє програмному забезпеченню ідентифікувати джерело скидання. Контролер Reset також може використовувати для виконання програмного скидання (SWRST).

Контролер переривань центрального процесора (CPUINT).

Особливості:

• короткий і передбачуваний час відповіді на переривання;

• окрема конфігурація переривання та векторна адреса для кожного переривання;

• переривання з пріоритетом за рівнем і векторною адресою;

• немасковані переривання (NMI) для критичних функцій;

• два рівня пріоритету переривання: 0 (звичайний) і 1 (високий):

– один із запитів на переривання може бути додатково призначений як переривання рівня пріоритету 1;

– додаткова циклічна схема пріоритету для переривань рівня пріоритету 0;

• вектори переривань, які необов’язково розміщуються в розділі програми або розділі завантажувача

• вибір компактної векторної таблиці (CVT).

Периферійні пристрої можуть мати одне або декілька переривань. Усі переривання вмикаються та налаштовуються окремо. Якщо переривання ввімкнено та налаштовано, воно генеруватиме запит на переривання, коли виникає умова переривання.

Контролер переривань центрального процесора (CPUINT) обробляє запити на переривання та визначає пріоритети. Коли включено переривання і виникає умова переривання, CPUINT отримає запит на переривання. Залежно від рівня пріоритету переривання та рівня пріоритету будь-якого поточного переривання запит на переривання або підтверджується, або залишається в стані очікування, доки він не матиме пріоритет. Після повернення з обробника переривань виконання програми продовжується з того місця, де воно було до переривання, і будь-які незавершені переривання обслуговуються після виконання однієї інструкції.

CPUINT пропонує NMI для критичних функцій, одне переривання з високим пріоритетом, яке можна вибрати, і додаткову схему циклічного планування для переривань із нормальним пріоритетом. Кругове планування гарантує, що всі переривання обслуговуються протягом певного часу.

Система подій (EVSYS).

Особливості:

• система прямої передачі сигналів від периферії до периферії;

• периферійні пристрої можуть безпосередньо виробляти, використовувати та реагувати на периферійні події;

• короткий і передбачуваний час відгуку;

• доступно до 10 паралельних каналів подій;

• кожен канал керується одним генератором подій і може мати декілька користувачів подій;

• більшість периферійних пристроїв і програмне забезпечення можуть надсилати та/або отримувати події;

• система подій працює в активному режимі, режимі очікування та режимі сну.

Система подій забезпечує пряму передачу сигналів від периферії до периферії. Це дозволяє змінювати один периферійний пристрій (генератор подій), щоб ініціювати дії в інших периферійних пристроях (користувачах подій) через канали подій, без використання ЦП. EVSYS розроблений для забезпечення короткого та передбачуваного часу відгуку між периферійними пристроями, враховуючи автономне периферійне керування та взаємодію, а також для синхронізації дій у декількох периферійних модулях. Таким чином, це потужний інструмент для зменшення складності, розміру та часу виконання програмного забезпечення.

Зміна стану генератора подій називається подією і звичайно відповідає одній з умов переривання периферійного пристрою. Події можна пересилати безпосередньо на інші периферійні пристрої за допомогою спеціальної мережі маршрутизації подій. Маршрутизація кожного каналу налаштовується програмним забезпеченням, включаючи генерацію та використання подій. Тільки один сигнал події може бути направлений на кожен канал. Кілька периферійних пристроїв можуть використовувати події з одного каналу.

EVSYS може підключати периферійні пристрої, такі як АЦП, аналогові компаратори, контакти портів I/O, лічильник реального часу, таймер/лічильники та налаштовувана периферійна логіка. Події також можна генерувати за допомогою програмного забезпечення.

Мультиплексор портів (PORTMUX).

Мультиплексор портів може ввімкнути або вимкнути функціональність контактів або змінити положення контактів за замовчуванням на альтернативне.

Контакти I/O з різновольтовим живленням (MVIO).

Особливості:

• піднабір контактів введення/виведення пристрою може живитися від V_{DDIO2};

• джерело живлення V_{DDIO2} може наростати та зменшуватися незалежно від живлення VDD;

• конфігурація з одним або подвійним джерелом живлення визначається запобіжником;

• доступ до порту та перевизначення периферійних пристроїв незалежно від конфігурації живлення;

• біт стану живлення V_{DDIO2};

• переривання та подія для зміни стану живлення V_{DDIO2};

• канал АЦП для вимірювання напруги живлення V_{DDIO2}.

Функція MVIO дозволяє живити підмножину контактів вводу/виводу від іншої області напруги вводу/виводу, ніж решта контактів вводу/виводу. Це усуває потребу мати зовнішні перемикачі рівня для зв’язку або керування зовнішніми компонентами, що працюють на іншому рівні напруги. Плати вводу/виводу з підтримкою MVIO живляться від напруги, що подається на

контакт(и) живлення V_{DDIO2}, тоді як звичайні контакти вводу/виводу живляться від напруги, що подається на контакт(и) V_{DD}. MVIO можна налаштувати в одному з двох режимів живлення:

• Режим єдиного джерела живлення, коли контакти вводу-виводу з підтримкою MVIO живляться на тому ж рівні напруги, що й контакти, що не підтримують MVIO, тобто V_{DD}. Користувач повинен підключити контакт(и) V_{DDIO2} до контакту(ів) V_{DD}.

• Режим подвійного живлення, у якому контакти вводу/виводу з підтримкою MVIO живляться від напруги V_{DDIO2}, яка може відрізнятися від напруги, що подається на контакт(и) V_{DD}. Конфігураційний запобіжник визначає режим живлення MVIO. Втрата або збільшення потужності на V_{DDIO2} сигналізується бітом статусного регістра. Цей біт стану має відповідну функцію переривання та події. Виводи MVIO мають таку ж цифрову поведінку, як і звичайні виводи вводу-виводу, наприклад, GPIO, послідовний зв'язок (USART, SPI, I2C) або підключений до периферійних пристроїв ШІМ. Вхідні рівні запуску тригера Шмітта масштабуються відповідно до напруги V_{DDIO2}. Розділена напруга V_{DDIO2} доступна як вхідний сигнал для АЦП.

Джерела опорної напруги (VREF).

Особливості:

• програмовані джерела опорної напруги:

– одне опорне джерело для аналого-цифрового перетворювача 0 (ADC0);

– одне опорне джерело для цифри-аналогового перетворювача 0 (DAC0);

– одне опорне джерело, спільне для всіх аналогових компараторів (AC);

• кожне опорне джерело підтримує такі напруги: – 1,024 В; 2,048 В; 4,096 В; 2,500В; VDD; VREFA.

Периферійний пристрій опорної напруги (VREF) забезпечує регістри керування для джерел опорної напруги, які використовуються декількома периферійними пристроями. Користувач може вибрати опорні напруги для ADC0, DAC0 і ACs шляхом запису у відповідні регістри периферійного пристрою VREF.

Джерело опорної напруги вмикається автоматично за запитом периферійного пристрою. Користувач може ввімкнути джерела опорної напруги і, таким чином, перекрити автоматичне відключення невикористаних джерел, записавши відповідний біт ALWAYSON у VREF.ADC0REF, VREF.DAC0REF і VREF.ACREF. Це зменшить час запуску за рахунок збільшення енергоспоживання.

Лічильник реального часу (RTC).

Особливості:

• 16-бітова роздільна здатність;

• вибір джерела синхросигналу;

• програмоване 15-бітове попереднє масштабування джерела синхросигналу;

• один регістр порівняння;

• регістр одного періоду;

• чистка таймера у разі переповнення періоду;

• додаткове переривання/подія при переповненні та порівнянні;

• періодичні переривання та події;

• виправлення помилок кристалу.

Периферійний пристрій RTC пропонує дві функції синхронізації: лічильник реального часу (RTC) і таймер періодичних переривань (PIT).

Функцію PIT можна ввімкнути незалежно від функції RTC.

RTC.

RTC підраховує (попередньо масштабовані) такти в регістрі лічильника та порівнює вміст регістра лічильника з регістром періоду та регістром порівняння.

RTC може генерувати як переривання, так і події при порівняльному збігу або переповненні. Це створить переривання порівняння та/або подію при першій лічбі після того, як лічильник дорівнювати значенню регістру порівняння, і переривання переповнення та/або подія при першій лічбі після значення лічильника, яке дорівнює значенню регістра Period. Переповнення скине лічильник значення до нуля.

Периферійний пристрій RTC звичайно працює безперервно, включно з режимами сну з низьким енергоспоживанням, щоб відстежувати час. Це може виводити пристрій із сплячих режимів та/або періодично вимикати роботу пристрою.

Опорним тактовим сигналом звичайно є вихідна частота 32,768 кГц із зовнішнього кристалу. RTC також можна синхронізувати із зовнішнім тактовим сигналом, внутрішнім генератор 32,768 кГц (OSC32K) або OSC32K, поділеним на 32.

Периферійний пристрій RTC містить 15-бітовий програмований попередній масштабувач, який може зменшити масштаб опорного годинника перед як попаде до лічильника. Для RTC можна налаштувати широкий діапазон роздільної здатності та періодів очікування. З джерела тактового сигналу 32,768 кГц, максимальна роздільна здатність становить 30,5 мкс, а періоди очікування можуть становити до двох секунд. З роздільною здатність 1 с, максимальний період очікування більше 18 годин (65536 секунд).

RTC також підтримує корекцію помилок кварцового кристала, коли він працює за допомогою зовнішнього кварцу. Зовнішнє калібрувальне значення може буде використано для корекції. RTC можна регулювати за допомогою програмного забезпечення з точністю ±1 PPM, і максимальне регулювання становить ±127 PPM. Операція корекції RTC буде або прискорювати (через пропуск лічби), або сповільнювати (шляхом додавання додаткової лічби) попередній дільник для врахування помилки кристалу.

PIT.

Таймер періодичних переривань PIT використовує те саме джерело синхронізації (CLK_RTC), що й функція RTC, і може генерувати запит на переривання або подію рівня на кожному *n*-му періоді такту. Значення *n* можна вибрати з {4, 8, 16,... 32768} для переривань і з {64, 128, 256,... 8192} для подій.

Конфігурована замовником логіка (CCL).

Особливості:

• зв’язуюча логіка для проектування друкованих плат загального призначення;

• 6 програмованих таблиць пошуку (LUT);

• комбінаторні логічні функції: будь-який логічний вираз, який є функцією до трьох входів;

• функції послідовнісної логіки (D тригер D із затвором; JK тригер; D-засувка; RS-засувка);

• гнучкий вибір входу LUT: введення/виведення; події; наступний вихід LUT; Внутрішні периферійні пристрої (аналоговий компаратор; таймери/лічильники; USART; SPI);

• тактується системним годинником або іншими периферійними пристроями;

• вихід можна підключити до контактів введення/виведення або системи подій;

• додатковий синхронізатор, фільтр або детектор межі, доступний на кожному виході LUT;

• додаткова генерація переривання з кожного виходу LUT (позитивний фрони; негативний фронт; обидва фронти).

Конфігурована користувачем логіка CLL – це програмований логічний периферійний пристрій, який можна підключити до контактів пристрою, подій або інших внутрішніх периферійних пристроїв. CCL може служити «сполучною логікою» між периферійними пристроями та зовнішніми пристроями. CCL може усунути потребу у зовнішніх логічних компонентах, а також може допомогти розробнику подолати обмеження в режимі реального часу шляхом об'єднання незалежних від ядра периферійних пристроїв (CIP) для обробки частини найбільш критичних у часі завдань програми, незалежно від ЦП.

Периферійний пристрій CCL забезпечує ряд таблиць пошуку (LUT). Кожна LUT складається з трьох входів, таблиці істинності, синхронізатора/фільтра та детектора країв. Кожний LUT може генерувати вихідні дані як програмований користувачем логічний вираз із трьома входами. Вихідні дані генеруються з вхідних даних за допомогою комбінаторної логіки та можуть бути відфільтровані для усунення імпульсних викидів (spikes). CCL може бути налаштований для створення запиту на переривання при зміні виходів LUT. Сусідні LUT можна комбінувати для виконання певних операцій. Послідовнісні пристрої можна використовувати для генерації складних сигналів.

Формування аналогових сигналів (OPAMP).

Особливості:

• схема внутрішнього резисторного сходинкового ланцюга полегшує формування аналогового сигналу з нульовими зовнішніми компонентами;

• можливі конфігурації (автономний операційний підсилювач загального призначення; буфер одиничного підсилення; неінвертуючий/інвертуючий підсилювач з програмованим підсиленням (PGA); каскадні PGA; інструментальний підсилювач);

• вибір входу (контакти введення/виведення; ЦАП; земля; опорна напруга VDD/2; вихід з іншого операційного підсилювача; внутрішня резисторна сходинка);

• вибір виходу (на контактах введення/виведення; як вхід для АЦП; як вхід для змінного струму; як вхід для іншого операційного підсилювача);

• внутрішній таймер генерує подію READY після завершення встановлення;

• підтримка низького енергоспоживання (додаткова операція, викликана подією);

• режим дампу, керований системою подій, для підтримки інтеграції сигналу;

• калібрування зсуву та підсилення за допомогою АЦП.

Периферійний пристрій обробки аналогового сигналу (OPAMP) має три операційні підсилювачі (ОП), позначені OP*n*, де *n*=0,1,2. Ці операційні підсилювачі реалізовані за гнучкою схемою з'єднання за допомогою аналогових мультиплексорів і ланцюга сходинок резисторів. Це дозволяє формувати велику кількість конфігурацій аналогового сигналу, багато з яких не потребують зовнішніх компонентів. Мультиплексор на неінвертуючому (+) вході кожного операційного підсилювача дозволяє підключатися до зовнішнього контакту, потрібної позиції ланцюга сходинок резисторів, виходу ЦАП, землі, VDD/2 або виходу іншого операційного підсилювача. Другий мультиплексор на інвертуючому (-) вході кожного операційного підсилювача дозволяє підключатися до зовнішнього контакту, потрібної позиції ланцюга сходинок резисторів, виходу операційного підсилювача або виходу ЦАП. Ще три мультиплексори, підключені до кожного ланцюга резисторів, забезпечують додаткову гнучкість конфігурації. Два з цих мультиплексорів вибирають верхнє і нижнє підключення ланцюга резисторів, а третій контролює положення змінного підключення.

Детектор переходів нуля (ZCD).

Особливості:

• виявлення переходів нуля на змінних сигналах високої напруги;

• потрібен тільки один зовнішній резистор;

• вихід детектора доступний на контакті;

• полярність вихідного сигналу детектора можна інвертувати;

• генерація переривання за наростаючим, спадаючим або за обома краями фронту сигналу;

• генерація подій: вихід детектора.

Детектор виявляє, коли змінна напруга перетинає порогову напругу поблизу потенціалу землі. Пороговим значенням є опорна напруга перехресного нуля, ZCPINV. Підключення від вхідного контакту ZCIN до змінної напруги повинно здійснюватися через послідовне обмеження струму резистором R_{SERIES}. Детектор подає або джерело або споживач струму до вхідного контакту, щоб підтримувати постійну напругу на контакті, тим самим запобігаючи значення напруги на контакті, яке може прямо змістити діоди захисту від електростатичного розряду в пристрої. Коли прикладена напруга перевищує опорну напругу, ZCD споживає струм. При прикладеній напрузі меншій від опорної напруги, детектор стає джерелом струму.

Уніфікований інтерфейс програм і налаштування (UPDI).

Особливості:

• однопровідний інтерфейс UPDI для зовнішнього програмування та налагодження на кристалі (OCD);

– використовує спеціальний контакт пристрою для програмування;

– під час операції жодні контакти GPIO не зайняті;

– асинхронний напівдуплексний протокол UART для програміста;

• програмування:

– вбудоване виявлення помилок і створення сигнатур помилок;

– перевизначення генерації відповіді для швидшого програмування;

• налагодження:

– доступ із відображенням пам’яті до адресного простору пристрою (NVM, RAM, I/O);

– немає обмежень на тактову частоту пристрою;

– необмежена кількість точок зупинки програми користувача;

– дві апаратні точки зупинки;

– підтримка розширених функцій OCD;

• зчитування під час виконання програмного лічильника процесора (PC), покажчика стеку (SP) і регістру стану (SREG) для профілювання коду;

• виявлення та сигналізація стану Break/Stop у CPU;

• контроль потоку програм для вказівок із налагодження запуску, зупинки та скидання;

– ненав’язливий моніторинг мікросхеми під час виконання без доступу до системних регістрів;

– інтерфейс для читання результату перевірки CRC Flash на заблокованому пристрої.

Уніфікований інтерфейс програми та налагодження (UPDI) – це власний інтерфейс для зовнішнього програмування та OCD пристрій.

UPDI підтримує програмування простору енергонезалежної пам’яті (NVM), Flash, EEPROM, запобіжників, бітів блокування та рядка користувача. Деякі регістри, відображені в пам’яті, доступні лише з увімкненим правильним привілеєм доступу (ключ, біти блокування) і

лише в режимі зупинки OCD або певних режимах програмування. Ці режими розблоковуються, надіславши правильний ключ до UPDI.

UPDI розділений на три окремі рівні протоколу: фізичний рівень UPDI (PHY), рівень зв’язку даних UPDI (DL) і рівень доступу UPDI (ACC). Рівень PHY за замовчуванням обробляє двонаправлений зв’язок UART через лінія контакту UPDI до підключеного програматора/налагоджувача та забезпечує відновлення даних і відновлення годинника на вхідний кадр даних у режимі One-Wire Communication. Отримані вказівки та відповідні дані обробляються рівнем DL, який встановлює зв'язок з рівнем ACC на основі декодованої інструкції. Доступ до системної шини та регістрів, відображених у пам’яті, надається через рівень ACC.

Питання.

1. Яку архітектуру використовують МК AVR?
2. Які структурні елементи має МК ATmega8515?
3. Які види пам’яті має МК ATmega8515?
4. Які регістри введення/виведення має МК ATmega8515?
5. Який зв’язок є між назвами і адресацією регістрів?
6. Які прапори умов входять в регістр стану SREG?
7. Регістр керування МК MCU.
8. Призначення і регістри стеку в МК AVR.
9. Призначення і регістри паралельних портів введення/виведення МК AVR.
10. Призначення і регістри таймерів/лічильників МК AVR.
11. Призначення сторожового таймера в МК AVR.
12. Призначення UART в МК AVR.
13. Призначення послідовного інтерфейсу SPI в МК.
14. Призначення аналогового компаратора в МК.
15. Які нові функціональні блоки добавлені в МК AVR DA, AVR DB, AVR DD?
16. Використання регістрів загального призначення.
17. Призначення контролера енергонезалежної пам’яті.
18. Призначення контролера джерел синхросигналів.
19. Призначення контролера сну.
20. Призначення контролера скидання.
21 Призначення контролер переривань центрального процесора.
22. Призначення системи подій.
23. Призначення мультиплексора портів.
24. Використання контактів з різновольтним живленням.
25. Використання джерел опорної напруги.
26. Призначення лічильника реального часу.
27. Можливості конфігурованої замовником логіки.
28. Можливості формування аналогових сигналів.
29. Призначення детектора переходів нуля.
30. Можливості уніфікованого інтерфейсу програм і налаштування.

ЛЕКЦІЯ 3. Асемблер і інструкції мікроконтролера

Мета. Вивчення синтаксису, операторів, функцій, препроцесора, директив, режимів адресування, набору інструкцій мікроконтролера

Вступ. МК AVR мають понад 100 машинних інструкцій, які підрозділяються на групи арифметичні і логічні, переходів, пересилання даних, роботи з бітами і керування МК. В інструкціях можуть використовуватися наступні способи адресації даних: пряма, непряма, відносна, безпосередня, бітова.

План.

1. Синтаксис
2. Вирази асемблера
2.1. Операнди
2.2. Оператори
2.3. Функції асемблера
3. Препроцесор
3.1. Вбудовані макроси
4. Режими адресації програм і даних
5. Прийняті позначення
6. Набір інструкцій МК AVR

1. Синтаксис

Асемблер МК AVR не розпізнає регістр букв.

Кожна команда асемблера переважно записується в один рядок довжиною до 120 знаків. Рядки можуть містити наступні елементи:

Позначки (точки для переходів) – алфавітно-цифрові знаки, які мають починатися з букви і закінчуватися двокрапкою.

Мнемоніки команд – символічне подання машинних команд.

Директиви (вказівки для асемблера) – визначена послідовність символів, яка починається з крапки і не транслюється в машинні команди, а тільки використовується в процесі асемблювання.

Коментарі – починаються з символу ”;” і додаються в програму для кращого її розуміння.

Командний рядок може бути записаний в одній з чотирьох форм:

```
[позначка:] директива [операнди] [коментарі]
[позначка:] команда [операнди] [коментарі]
[позначка:] коментарі
порожній рядок

Приклад:
label: .EQU var1=100 ; Присвоїти змінній var1 100 (директива)
       .EQU var2=200 ; Присвоїти змінній var1 200
test:   rjmp test    ; Нескінченний цикл (інструкція)
```

Коментар має наступні форми:

```
; класичний коментар асемблера
// коментар в стилі C++
/* багаторядковий коментар в стилі C,
   який не може бути вкладений */
```

Ключові слова асемблера є зарезервованими і не можуть бути перевизначеними. Ключові слова включають мнемоніки команд, назви регістрів `R0-R31, X,Y,Z` і всі функції. Ключові слова розпізнаються незалежно від регістру букв.

Довгий командний рядок можна продовжити на наступні, використовуючи символ (\).

1.1. Двійкові числа з символом _

Для покращення візуального сприйняття двійкових чисел символ (_) може використовуватися всередині двійкових чисел:

```
0b1100_1010
0b_11_00_10_10_
```

1.2. Стрічкові і символьні константи

Символьні константи записуються в апострофах (‘) і можуть використовуватися скрізь, де використовуються цілі числа:

```
.db 'H', 'e', 'l', 'l', 'o', '\\', 'n'
.db '\0', '\177', '\xff'
```

Стрічкові константи записуються у лапках (“) і можуть використовуватися тільки у директивах `.db, .message, .warning, .error`.

Розпізнаються також екранові символи в стилі Сі:

Таблиця 1.1 – Екрановані послідовності

Екранована послідовність	Значення
\n	Символ нового рядка
\r	Символ повернення каретки
\a	Звуковий сигнал тривоги
\b	Повернення на один символ назад
\f	Подача аркуша на один рядок
\t	Горизонтальна табуляція
\v	Вертикальна табуляція
\\	Вліво нахилена похила
\0	Нульовий символ (Null)

Приклади:

```
.db "Hello\n"  // є еквівалентним до:
.db 'H', 'e', 'l', 'l', 'o', '\\', 'n'
.db '\0', '\177', '\xff'
.db "Hello, world", '\n', 0
```

2. Вирази асемблера

В асемблері використовуються константні вирази. Вирази можуть складатися з операндів, операцій і функцій. Вирази можуть записуватися у круглих дужках і тоді вони завжди спочатку оцінюються, а потім об’єднуються з виразами поза дужками.

2.1. Операнди

До операндів відносяться:
- визначені користувачем позначки, які задають значення за адресою позначки;
- визначені користувачем змінні за допомогою директиви SET;
- визначені користувачем константи за допомогою директиви EQU;
- цілочисельні константи у різних форматах:
  - двійковий: 0b00001010, 0b11111111, 0b1110_0001
  - вісімковий (ведучий нуль): 010, 077
  - десятковий (за замовчуванням): 10, 255
  - шістнадцятковий (дві нотації C та Pascal): 0x0a, 0xff, $0a, $ff
- поточне значення вказівника команд PC;
- константи з плаваючою крапкою (тільки в AVRASM2).

2.2. Оператори

Асемблер підтримує ряд операцій, які описані і розміщені у порядку спадання пріоритету у таблиці 1. Асоціативність двійкових операцій вказує порядок оцінювання ланцюжків операцій. Ліва асоціативність означає, що операції оцінюються зліва направо, тобто, 2 - 3 - 4 розглядається як (2 - 3) - 4, а права асоціативність для операцій 2-3-4 – розглядається як 2 - (3 - 4). Деякі операції не асоціативні, тобто не використовуються у ланцюжках операціях.

Таблиця 2.1 – Оператори асемблера

Опера-ція	Описання	Приклад
!	Логічне НІ (Logical Not)	`ldi r16,!0xf0 ; Load r16 with 0x00`
~	Побітове НІ (Bitwise Not)	`ldi r16,~0xf0 ; Load r16 with 0x0f`
-	Унарний мінус (Unary Minus)	`ldi r16,-2 ; Load -2(0xfe) in r16`
*	Множення (Multiplication)	`ldi r30,label*2 ; Load r30 with label*2`
/	Ділення (Division)	`ldi r30,label/2 ; Load r30 with label/2`
%	Ділення за модулем (Modulo, AVRASM2)	`ldi r30,label%2 ; Load r30 with label%2`
+	Додавання (Addition)	`ldi r30,c1+c2 ; Load r30 with c1+c2`
-	Віднімання (Subtraction)	`ldi r17,c1-c2 ;Load r17 with c1-c2`
<<	Лівий зсув (Shift left)	`ldi r17,1<<bitmask ;Load r17 with 1 shifted left bitmask times`
>>	Правий зсув (Shift right)	`ldi r17,c1>>c2 ;Load r17 with c1 shifted right c2 times`
<	Менше ніж (Less than)	`ori r18,bitmask*(c1<c2)+1 ;Or r18 with an expression`
<=	Менше рівне Less than or equal)	`ori r18,bitmask*(c1<=c2)+1 ;Or r18 with an expression`
>	Більше (Greater than)	`ori r18,bitmask*(c1>c2)+1 ;Or r18 with an expression`
>=	Більше рівне (Greater than or equal)	`ori r18,bitmask*(c1>=c2)+1 ;Or r18 with an expression`
==	Рівне (Equal)	`andi r19,bitmask*(c1==c2)+1 ;And r19 with an expression`

!=	Не рівне (Not equal)	`.SET flag=(c1!=c2)  ;Set flag to 1 or 0`
&	Побітове І (Bitwise And)	`ldi r18,High(c1&c2) ;Load r18 with an expression`
^	Побітове виключальне АБО Bitwise Xor)	`ldi r18,Low(c1^c2) ;Load r18 with an expression`
|	Побітове АБО (Bitwise Or)	`ldi r18,Low(c1|c2) ;Load r18 with an expression`
&&	Логічне І (Logical And)	`ldi r18,Low(c1&&c2)  ;Load r18 with an expression`
||	Логічне АБО (Logical Or)	`ldi r18,Low(c1||c2) ;Load r18 with an expression`
?	Операція умови (Conditional operator, AVRASM2)	`ldi r18, a > b? a : b  ; Load r18 with the maximum numeric value of a and b.`

2.3. Функції асемблера

В асемблері визначені наступні функції:

- LOW(вираз) повертає молодший байт виразу;
- HIGH(вираз) повертає старший байт виразу;
- BYTE2(вираз) повертає другий байт виразу (аналогічно до HIGH);
- BYTE3(вираз) повертає третій байт виразу ;
- BYTE4(вираз) повертає четвертий байт виразу;
- LWRD(вираз) повертає 0-15 біти виразу;
- HWRD(вираз) повертає 16-31 біти виразу;
- PAGE(вираз) повертає 16-21 біти виразу;
- EXP2(вираз) повертає вираз у степені 2;
- LOG2(вираз) повертає цілу частину log2(вираз);
- INT (вираз) повертає цілу частину виразу з плаваючою крапкою;
- FRAC(вираз) повертає дробову частину виразу з плаваючою крапкою;
- Q7(вираз) перетворює дробову частину виразу з плаваючою крапкою у форму придатну для команд FMUL/FMULS/FMULSU (Знак + 7 бітів дробової частини)
- Q15(вираз) перетворює дробову частину виразу з плаваючою крапкою у форму придатну для команд FMUL/FMULS/FMULSU (Знак + 15 бітів дробової частини)
- ABS(вираз) повертає абсолютне значення константного виразу
- DEFINED(символ) повертає true, якщо символ визначений директивами .equ, .set, .def. Звичайно використовується з директивою `.if`, наприклад `.if defined(foo)`.
- STRLEN(вираз) повертає довжину стрічкової константи в байтах.

```
.equ  var  =0x1234
.equ  var1 =0xABCDEF89

  .CSEG
  ldi  r16, Low(var)    ; r16 = 0x34
  ldi  r17, High(var)   ; r17 = 0x12
  ldi  r18, Low(var1)   ; r18 = 0x89
  ldi  r19, Byte2(var1  ; r19 = 0xEF
  ldi  r20, Byte3(var1) ; r20 = 0xCD
  ldi  r21, Byte4(var1) ; r21 = 0xAB
```

```
  ldi  r22, Exp2(3)     ; r22 = 8
  ldi  r23, Log2(17)    ; r23 = 4
table:  .dw  Lwrd(var1), Hwrd(var1)  ; 0xEF89, 0xABCD
```

3. Препроцесор

Таблиця 3.1 – Директиви препроцесора

Директиви	Описання	
#define	Визначення макросу препроцесорної змінної або функції	`#define EIGHT (1 << 3)` `#define SQR(X) ((X)*(X))`
#undef	Відміна попереднього визначення макросу	`#undef SQR`

#ifdef	Коротка форма для `#if defined (name)`	`#ifdef FOO`
		`// do something`
		`#endif`

#ifndef	Коротка форма для `#if !defined (name).`	`#ifndef name`

#if, #elif	Асемблювання за умовою	`#if 0`
		`// code here is never included`
		`#endif`
		`#if defined(__ATmega48__) ||`
		`defined(__ATmega88__)`
		`// code specific for these devices`
		`#elif defined (__ATmega169__)`
		`// code specific for ATmega169`
		`#endif // device specific code`

#else	Асемблювання за умовою	`#if defined(__ATmega48__) ||`
		`defined(__ATmega88__)`
		`// code specific for these parts`
		`#elif defined (__ATmega169__)`
		`// code specific for ATmega169`
		`#else`
		`#error "Unsupported part:"`
		`__PART_NAME__`
		`#endif // part specific code`

#endif	Завершення блоку умов	`#endif`
#error, #warning, #message	Виведення повідомлення про помилку і збільшення лічильника error, warning відповідно. Message на лічильники не впливає.	`#error "Unsupported part:"` `__PART_NAME__`
#include	Вставлення файлів	`#include "file"` `#include <file>`

#pragma (general) (загального призначення)	Асемблер оцінює внутрішньо константні цілочислові вирази як 64-бітові знакові. Коли такі вирази використовуються в інструкціях з безпосередніми операндами (LDI, CPI, ORI, ANDI, SUBI, SBCI) то вони обрізаються. На інтерпретації операндів впливають опції:
	• optional=integer – якщо значення операнду виходить за межі [-128,255] виводиться warning;
	• optional=overflow (default) – операнд оцінюється як беззнаковий
	• optional=none – відміна виведення повідомлень
		`#pragma warning range byte option`
		`#pragma overlap option`
		`#pragma error instruction`
		`#pragma warning instruction`

#pragma (AVR)	Задають специфічні властивості окремих моделей МК	1.#pragma AVRPART ADMIN PART_NAME string 2. #pragma AVRPART CORE CORE_VERSION version-string 3. #pragma AVRPART CORE INSTRUCTIONS_NOT_SUPPORTED mnemonic[operand[,operand]][:...] 4. #pragma AVRPART CORE NEW_INSTRUCTIONS mnemonic[operand[,operand]][:...] 5. #pragma AVRPART MEMORY PROG_FLASH size 6. #pragma AVRPART MEMORY EEPROM size 7. #pragma AVRPART MEMORY INT_SRAM SIZE size 8. #pragma AVRPART MEMORY INT_SRAM START_ADDR address 9. #pragma partinclude num
#	Порожня директива	#define MY_IDENT(X) .db #X, '\n', 0 Так виклик MY_IDENT(FooFirmwareRev1) буде розширено до .db "FooFirmwareRev1", '\n', 0
##	Зчеплення	#define FOOBAR subi 1. #define IMMED(X) X##i 2. #define SUBI(X,Y) X ## Y При виклику макросів 1. IMMED(ld) r16,1 2. SUBI(FOO,BAR) r16,1 вони будуть розширені до 1. ldi r16,0x1 2. subi r16,0x1
Predefined macros	Вбудовані макроси (множина директиви #pragma general)	

3.1. Вбудовані макроси

Вбудовані макроси (pre-defined macros) є множиною директиви #pragma (general).

Таблиця 3.2 — Вбудовані макроси

Назва	Тип	Визначення	Опис
__AVRASM_VERSION__	Integer	built-in	Assembler version, encoded as (1000*major + minor)
__CORE_VERSION__	String	built-in	AVR core version
__DATE__	String	built-in	Build date. Format: "Jun 28 2004"
__TIME__	String	built-in	Build time. Format: "HH:MM:SS".
__CENTURY__	Integer	built-in	Build time century (typically 20)
__YEAR__	Integer	built-in	Build time year, less century (0-99)
__MONTH__	Integer	built-in	Build time month (1-12)
__DAY__	Integer	built-in	Build time day (1-31)
__HOUR__	Integer	built-in	Build time hour (0-23)
__MINUTE__	Integer	built-in	Build time minute (0-59)
_SECOND__	Integer	built-in	Build time second (0-59)
__FILE__	String	built-in	Source file name
__LINE__	Integer	built-in	Current line number in source file

__PART_NAME__	String	#pragma, General Purpose	AVR part name
__partname__	Integer	#pragma, General Purpose	partname corresponds to the value of __PART_NAME__. Example: #ifdef __ATmega8__
__CORE_coreversion__	Integer	#pragma, General Purpose	coreversion corresponds to the value of __CORE_VERSION__. Example: #ifdef __CORE_V2__

4. Режими адресації програм і даних

Режими *адресації пам'яті даних*:

- безпосередня;
- пряма:
    - одного регістра;
    - двох регістрів;
    - області пам'яті введення-виведення;
    - пам'яті даних SRAM;
- непряма:
    - з використанням регістрів X,Y,Z;
    - з використанням регістрів X,Y,Z і їх попереднім декрементом;
    - з використанням регістрів X,Y,Z і їх наступним інкрементом;
- непряма із зміщенням (відносна):
    - з використанням регістра Z.

Режими *адресації пам'яті програм*:

- непряма з використанням регістра Z;
- пряма з використанням команд JAMP, CALL;
- відносна з використання команди переходу RJMP.

Бітова адресація регістрів загального призначення і регістру SREG.

Регістри з R26 по R31 реєстрового файлу X, Y і Z є регістрами вказівниками непрямої адресації.

Безпосередня адресація. Вказується один з операндів (константа *к*) безпосередньо в команді. Безпосередня адресація використовується командою LDI пересилання константи в регістр, а також деякими командами арифметичних і логічних операцій.

```
ldi r1, 0x3a  ; завантаження в r1 шістнадцяткової константи
ldi r2, 40    ; завантаження в r2 десяткової константи
```

Пряма адресація одного регістра Rd. Регістр Rd з номером d, містить операнди до яких звертається команда (рис. 3.1, а). Прикладом такої адресації є команди для роботи зі стеком (PUSH, POP), обміну тетрадами в регістрі (SWAP), ряд команд арифметичних і логічних операцій:

```
com Rd     ; доповнення до 1
inc Rd; збільшення на 1 вмісту Rd
```

Пряма адресація двох регістрів Rd і Rr. Регістри Rd і Rr з номерами d і r містять операнди до яких звертається команда (рис. 3.1, б). Цей вид адресації використовується командами

пересилання даних з регістра в регістр, більшістю арифметичних команд та рядом команд логічних операцій. Результат команди записується в регістр Rd:

```
add Rd,Rr  ; сума <Rd>+<Rr> записується в Rd
cp Rd,Rr   ; порівняння <Rd> і <Rr>
```

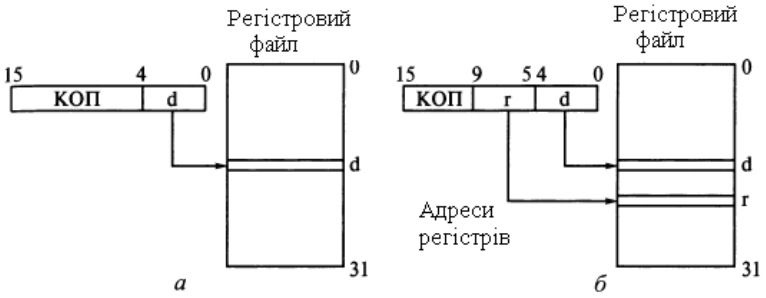


Рисунок 3.1 – Пряма адресація одного (а) і двох (б) регістрів загального призначення

Пряма адресація області введення/виведення. Даний вид адресації використовують для обміну між адресою вибраного порту P в області введення/виведення, і одним з регістрів загального призначення Rn, n=r/d в командах IN і OUT (рис. 3.2):

```
in Rd,P; записати значення з порту P в регістр Rd
out P,Rr; записати значення з регістра Rr в порт P
```

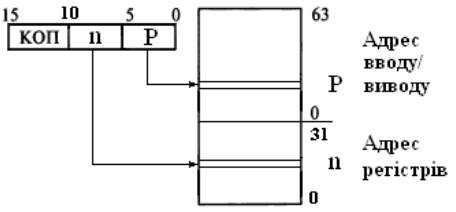


Рисунок 3.2 – Пряма адресація області введення/виведення

Пряма адресація пам'яті даних SRAM. Команди прямої адресації пам'яті даних LDS і STS складаються з двох командних слів, і для їх виконання потрібно три такти (рис. 3.4). Даний спосіб адресації застосовується при зверненні до любої комірки SRAM, як внутрішньої так і зовнішньої. Старше командне слово містить код операції і адреса регістра приймача або передавача даних. Молодше командне слово містить адресу області пам'яті, до якої необхідно звернутися.

```
lds r16,0xCCCC  ; завантаження в r16 комірки зовнішньої пам'яті
sts 0x60, r15   ; запис r15 в першу комірку внутрішньої пам'яті
```

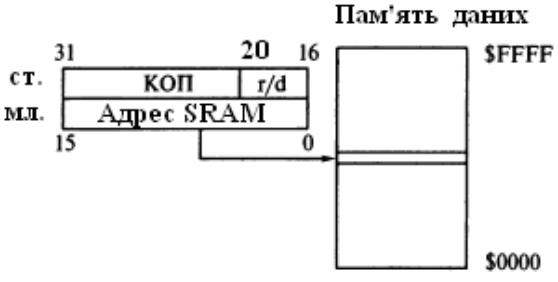


Рисунок 3.4 – Пряма адресація пам'яті даних

Непряма адресація пам'яті даних з використання регістрів X, Y, Z. Дані передаються між регістром, вказаним в командному слові 5-розрядною адресою n, і коміркою пам'яті, яка адресується одним з регістрів вказівників X (R26, R27), Y (R28, R29) або Z(R30, R31), (рис. 3.5):

```
ld r16,y   ; завантаження в r16 вмісту комірки з адресою в y
st x,r15   ; запис r15 в комірку з адресою в x
```

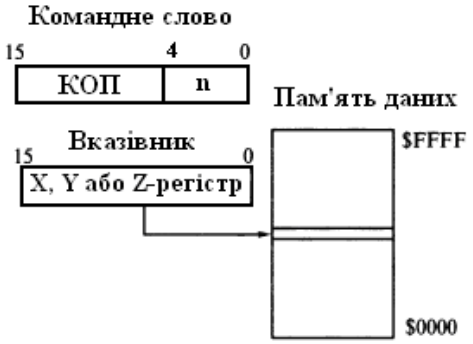


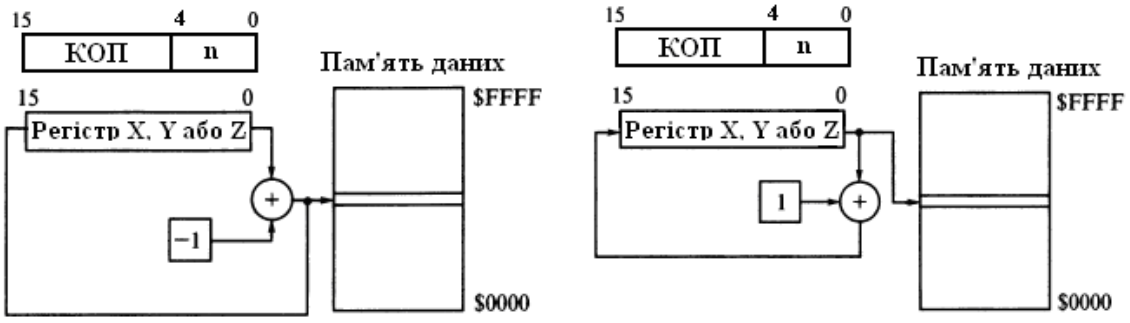
Рисунок 3.5 – Непряма адресація пам'яті даних з використанням регістрів X, Y, Z

Непряма адресація пам'яті даних з попереднім декрементом. При цьому способі адресації вміст регістра вказівника X, Y або Z спочатку зменшується на 1, а потім відбувається звернення до пам'яті за отриманою адресою (рис. 3.6, *а*). Це спосіб адресації команд LD (пересилання байта даних з пам'яті в регістр Rd) і ST (пересилання байта даних з регістра Rr в пам'ять), всього шість команд - по дві для кожного регістра.

```
st -y,r15 ; попередньо зменшити значення в y на 1, після чого записати в r15
ld r16,-z ; попередньо зменшити значення в z на 1, після чого записати в r16
```

Непряма адресація пам'яті даних з наступним інкрементом. При цьому способі адресації вміст індексного регістра X, Y або Z використовується як адреса звернення до пам'яті даних, а потім збільшується на 1 (рис. 3.6, *б*). Це спосіб адресації команд LD (пересилання байту даних з пам'яті в регістр Rd) і ST (пересилання байту даних з регістра Rr в пам'ять), всього шість команд – по дві для кожного регістра x, y, z.

```
st y+,r15  ; записати дані з r15 за адресою y, після чого збільшити y на 1
ld r16,z+  ; завантажити дані в r16 з z, після чого збільшити z на 1
```



а *б*

Рисунок 3.6 – Непряма адресація пам'яті даних із: а) перед декрементом; б) після інкрементом

Непряма адресація пам'яті даних із зміщенням (відносна адресація). При цьому способі адреса комірки пам'яті визначається додаванням вмісту регістра вказівника Y або Z з 6-розрядним зміщенням, яке задається в команді (рис. 3.7). Саме зміщення вказівника після виконання цієї команди залишається незмінним. Так адресуються команди LDD (пересилання байта з комірки пам'яті SRAM в регістр Rd) і STD (пересилання байту з регістра Rr в комірку SRAM). Такі команди використовуються при видобуванні даних з таблиць. Вони складаються з одного командного слова і виконуються за один такт.

```
ldd r16,z+0x3F   ; завантажити вміст комірки <z>+0x3f в регістр r16
std z+7, r15     ; записати  <r15> в комірку з адресою <z>+7
```

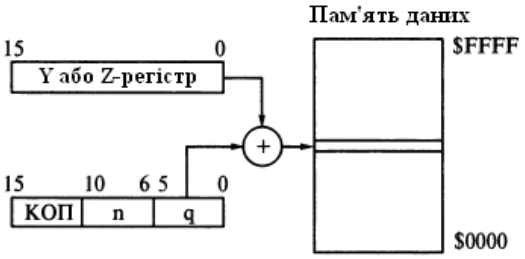


Рисунок 3.7 – Непряма адресація пам'яті даних із зміщенням

Непряма константна адресація пам'яті програм FLASH з використанням регістра Z. МК AVR дозволяють звертатися до комірок пам'яті програм для зчитування констант (визначених директивою асемблера .db), а в моделях родини Mega – і для записування даних у Flash-пам'ять програм, використовуючи механізм непрямої адресації через регістр Z. При цьому старші 15 розрядів регістра визначають адресу слова, а молодший нульовий розряд 0 (LSB) визначає, який байт має бути завантажений в регістр – молодший (біт 0 = 0) або старший (біт 0 = 1). Даний вид адресації використовує команда зчитування з комірок пам'яті в регістр R0/Rd (LPM, ELPM) і запис в пам'ять з регістрів R1:R0 (SPM), рис. 3.8.

```
lpm   ; копіювати в r0 вміст байту з адресою в z
lpm Rd, z   ; копіювати в Rd вмісту байту з адресою в z
spm; записати за адресою z слово з регістрів r0/r1
```

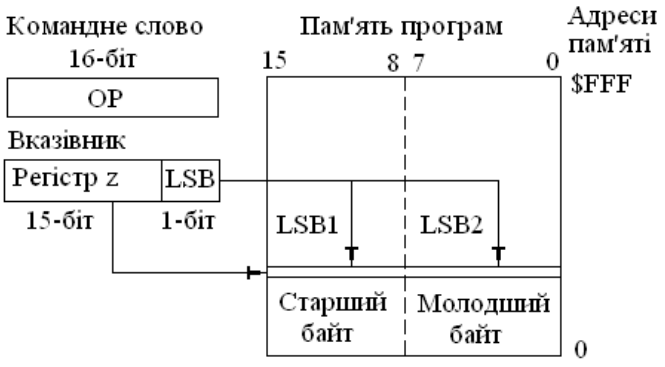


Рисунок 3.8 – Непряма константна адресація пам'яті програм

Непряма адресація пам'яті програм FLASH з після інкрементом регістра Z. МК AVR дозволяють зчитувати комірки пам'яті програм регістр Rd (LPM Z+, ELPM Z+) і після інкремент регістра Z, рис. 3.9.

```
lpm Rd, z+  ; копіювати в Rd вмісту байту з адресою в z з після декретом z
```

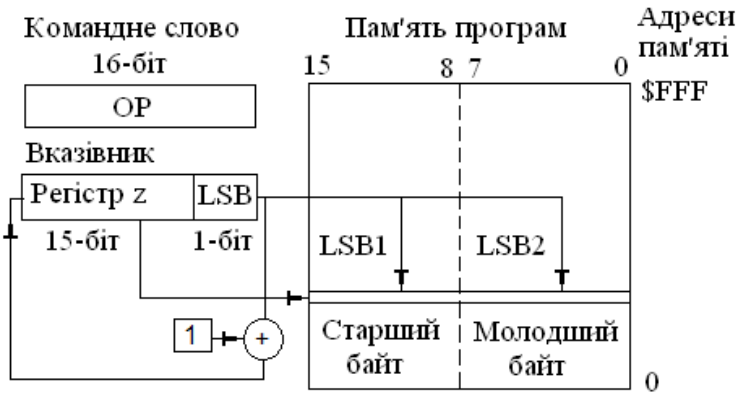


Рисунок 3.9 – Непряма адресація пам’яті програм з після інкрементом Z

Пряма адресація пам’яті програм FLASH з використанням команд JMP, CALL. Команди JMP, CALL виконують прямий перехід (Long Call) і виклик підпрограм (Long Call) за вказаною адресою. Команди JMP і CALL складаються з двох командних слів і для їх виконання потрібно три (JMP) і чотири (CALL) тактових цикли. Тобто в лічильник команд завантажується безпосередньо вказана адреса, а виконання програм продовжується з адреси, що міститься в обох командних словах, рис. 3.10.

```
label1: jmp label2
label2: jmp label1
```

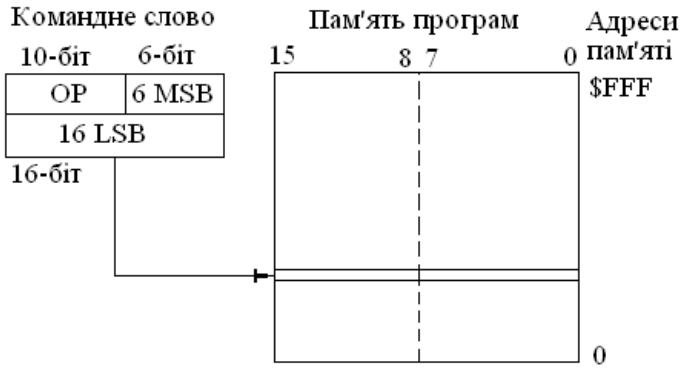


Рисунок 3.10 – Пряма адресація в пам’яті програм

Непряма адресація пам’яті програм FLASH з використанням команд IJMP, ICALL. Команди IJMP, ICALL здійснюють непрямий перехід за адресою заданою в регістрі Z.

```
label1: ldi r31,05      ; старший байт в z
        ldi r30,0x0a    ; молодший байт в z
ijmp; непрямий перехід за адресою заданою в z
```

Відносна адресація пам’яті програм FLASH з використанням команд RJMP, RCALL. Адреса обчислюється шляхом додавання вмісту програмного лічильника PC і константи k, яка задається в команді (рис. 3.11). Відносну адресацію використовують команди відносного переходу (RJMP) і відносного виклику підпрограми (RCALL), велика група команд умовних переходів.

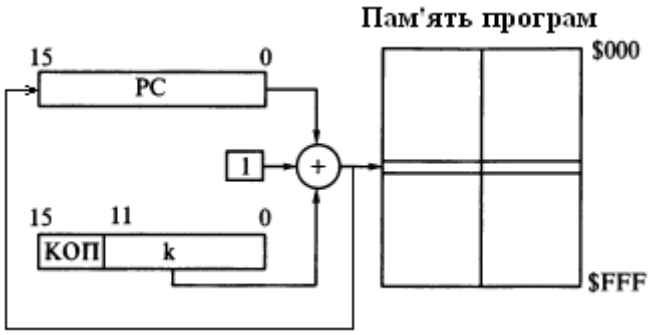


Рисунок 3.11 – Відносна адресація пам’яті програм

Бітова адресація. Цей вид адресації дозволяє вказати один з восьми бітів любого із 32 регістрів загального призначення або першої половини регістрів введення/виведення з номерами 0-31, а також регістра SREG. Для цього потрібно вказати ім’я регістра загального призначення Ri (i = 0...31) або ім’я регістра введення/виведення Pi (i = 0.. .31), або ім’я SREG і номер біта b (b = 0.. .7). Команди SBI і CBI встановлюють в 1 і скидають в 0 вказаний біт регістра введення/виведення, команди BLD, BST – обмінюють значення біта T з регістра SREG і адресованого біта з регістра загального призначення.

Крім цього є група команд бітових операцій, яка забезпечує встановлення і скидання бітів регістра стану SREG. При запису мнемоніки команд допускається використання символьних (штатних) імен регістрів введення/виведення і імен бітів.

Існує група команд умовного переходу, де як умови використовується або значення біта в регістрі загального призначення (`SBRC, SBRS`), або значення біта в регістрі введення/виведення (`SBIC, SBIS`).

```
sbi PortD, PD5
sbrc r1,1
```

5. Прийняті позначення

При описанні набору інструкцій МК AVR використовуються наступні позначення і скорочення:

- SREG – регістр стану в області введення/виведення;
- C – прапор перенесення (розряд 0 регістра SREG);
- Z – нульовий прапор (розряд 1 регістра SREG);
- N – прапор негативного результату (розряд 2 регістра SREG);
- V – прапор переповнення при обчисленнях в додаткових кодах (розряд 3 регістра SREG);
- S – прапор знаку (розряд 4 регістра SREG);
- H – прапор половинного перенесення (розряд 4 регістра SREG);
- T – прапор копіювання (розряд 6 регістра SREG);
- I – прапор загального дозволу переривань (розряд 6 регістра SREG);
- Rd – регістр отримувач;
- Rr – регістр відправник;
- R – результат виконання деякої команди;
- Kn – n-бітова константа (наприклад, K8=1);
- k – адресна константа для роботи з лічильником команд;

• b – ідентифікатор деякого розряду у робочому регістрі або в регістрі введення/виведення (3 біти);

• s – вказує на деякий розряд в регістрі стану SREG;

• X,Y,Z – вказівник при непрямій адресації (X=R27:R26, Y=R29:R28, Z=R31:R30);

• Rdl – молодший байт регістрової пари Rdh:Rdl;

• Rdh – старший байт регістрової пари Rdh:Rdl;

• P – адреса деякого порту введення/виведення;

• q – зміщення при непрямій адресації (6 біт);

• Стек – область пам'яті для зберігання адреси повернення або регістр проміжного зберігання даних;

• SP – вказівник стеку при адресації стеку;

• X – зарезервовані розряди в коді операції, які асемблером встановлюються в 0.

6. Набір інструкцій МК AVR

Кількість інструкцій (команд) МК залежить від базової моделі родини AVR. Порівняно із звичайними RISC контролерами, МК AVR мають розширену систему команд, що ще більш зменшило розмір коду і підвищило швидкодію. Були реалізовані CISC подібні команди, але без втрат в RISC продуктивності і споживаної потужності.

МК ATmega8515 має 128 машинних команд, в той час як інші RISC МК такого ж рівня мають 50÷60 машинних команд. Завдяки малому часу виконання команд із свого розширеного набору, МК AVR розв'язують задачі в 4-16 рази швидше порівняно з МК інших виробників.

Завдяки тому, що більшість МК AVR мають 32 робочих регістри, кожний з яких напряму зв'язаний з АЛП, багато операцій виконуються за один період такту сигналу синхронізації.

Інструкції асемблера МК AVR поділяються на наступні групи:

- арифметичні і логічні інструкції;
- переходів (передачі управління);
- пересилання і завантаження даних;
- для роботи з бітами;
- керування роботою МК.

Таблиця 3.1 – Арифметичні і логічні інструкції

Мнемо-ніка	Операнди	Описання	Функція	Прапори	Цикли
ADD	Rd,Rr	Додати	Rd = Rd + Rr	Z,C,N,V,H,S	1

Add without Carry		Додати два регістри Rd, Rr (0<=d<=31, 0<=r<=31) і результат записати в регістр Rd.
		add r1,r2 ; додати r2 до r1 (r1=r1+r2)
		add r28, r28 ; додати r28 до самого себе (r28=r28+r28)			

ADC	Rd,Rr	Додати з врахуванням перенесення	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
Add with Carry		Додати два регістри Rd, Rr (0<=d<=31, 0<=r<=31) і прапор C та результат записати в Rd. ; додати r1:r0 і r3:r2 add r2,r0 adc r3,r1			
ADIW	Rd, K6	Додати безпосереднє значення до регістрової пари	Rd+1:Rd = Rd+1:Rd + K	Z,C,N,V,S	2
Add Immediate To Word		Додати безпосереднє значення K (0-63) до регістрової пари Rd+1:Rd d={24, 26, 28, 30}. adiw r25:24,1 ; додати 1 до r25:r24			

		adiw ZH:ZL,63 ; додати 63 до Z вказівника(r31:r30)			
SUB	Rd,Rr	Відняти	Rd = Rd - Rr	Z,C,N,V,H,S	1
Subtract without Carry		Відняти два регістри Rd, Rr (0<=d<=31, 0<=r<=31) і результат записати в Rd. sub r13,r12 ; відняти r12 від r13 brne noteq ; перехід, якщо r12<>r13 ... noteq: nop			
SUBI	Rd,K8	Відняти від регістра константу K	Rd = Rd - K8	Z,C,N,V,H,S	1
Subtract Immediate		Відняти від регістра Rd (16<=d<=31) константу K (0-255) і результат записати у регістр Rd. subi r22,$11 ; відняти $11 (0x11, 17) від r22 brne noteq ; перехід, якщо r22<>$11 ... noteq: nop			
SBC	Rd,Rr	Відняти два регістри з врахуванням позики	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
Subtract with Carry		Відняти від регістра Rd (16<=d<=63, 0<=r<=63) і прапор C та результат записати в Rd ; відняти r1:r0 від r3:r2 sub r2,r0 ; відняти молодший байт sbc r3,r1 ; відняти старший байт і прапор C			
SBCI	Rd,K8	Відняти від регістра константу K з врахуванням позики	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
Subtract with Carry Immedtiate		Відняти від регістра Rd регістр Rr (0<=d<=31) константу K (0-255) і прапор C та результат записати в Rd. ; відняти ;4f23 від r17:r16 subi r16,$23 ; відняти молодший байт sbci r17,$4f ; відняти старший байт і прапор C			
SBIW	Rd,k	Відняти безпосередньо від слова константу K	Rdh:Rgl=Rdh:Rdl - K	Z,C,N,V,S	2
Subtract Immedtiate from Word		Відняти від регістрів Rdh:Rdl константу K (0-255) sbiw XH:XL,1			
AND	Rd,Rr	Логічне І (AND)	Rd = Rd * Rr	Z,N,V,S	1
Logical AND		Логічне AND двох регістрів Rd, Rr (0<=d<=31, 0<=r<=31) і результат записати в Rd. and r2,r3 ; побітове and r2 і r3 ldi r16,1 ; задати бітову маску 0000 0001 в r16 and r2, r16 ; виділити біт 0 в r2			
ANDI	Rd,K8	Логічне І (AND) регістра з константою K	Rd = Rd * K8	Z,N,V,S	1
Logical AND with Immediate		Логічне AND регістра Rd (16<=d<=31) і константи K (0-256) та результат записати в Rd andi r17,$0F ; очистити старший півбайт r17 andi r18,$10 ; виділити біт 4 в r18 andi r19,$AA ; очистити непарні біти в r19			
OR	Rd,Rr	Логічне АБО (OR)	Rd = Rd V Rr	Z,N,V,S	1
Logical OR		Логічне OR двох регістрів Rd, Rr (0<=d<=31, 0<=r<=31) і результат записати в Rd. or r15,r16 ; побітове or r15 і r16 bst r15,6 ; записати біт 6 з r15 у прапор T brts ok ; перейти, якщо прапор T встановлений ... ok: nop			
ORI	Rd,K8	Логічне АБО (OR) регістра з константою K	Rd = Rd V K8	Z,N,V,S	1
Logical OR with Immediate		Логічне OR регістра Rd (16<=d<=31) і константи K (0-256) та результат записати в Rd. ori r16,$F0 ; очистити старший півбайт r17 ori r17,1 ; виділити біт 4 в r18			
EOR	Rd,Rr	Логічне виключальне АБО (XOR)	Rd = Rd EOR Rr	Z,N,V,S	1
Logical Exclusive OR		Логічне XOR регістрів Rd, Rr (0<=d<=31, 0<=r<=31) і результат записати в Rd.			

		eor r4,r4 ; очистити r4 eor r0,r22 ; побітове виключне АБО			
COM	Rd	Доповнення до 1	Rd = $FF - Rd	Z,C,N,V,S	1
One's Complement		Доповнення до 1 регістра Rd (0<=d<=31). com r4 ; доповнення до 1 r4 (інверсний код) breq zero ; перехід, якщо 0 ... zero: nop			
NEG	Rd	Доповнення до 2	Rd = $00 - Rd	Z,C,N,V,H,S	1
Two's Complement		Доповнення до 2 регістра Rd (0<=d<=31) sub r11,r0 ; відняти r0 від r11 brpl positive ; перехід, якщо результат позитивний neg r11 ; доповнення до 2 r11 (доповняльний код) positive: nop значення 128($80, 1000_0000(залишається без зміни)			
SBR	Rd,K8	Встановити біт(и) у регістрі	Rd = Rd V K8	Z,C,N,V,S	1
Set Bit(s) in Register		Встановити задані біт(и) у регістрі Rd (16<=d<=31). sbr r16,3 ; встановити біти 0, 1 в r16 sbr r17,$f0 ; встановити 4-ри старші біти в r17			
CBR	Rd,K8	Очистити біт(и) в регістрі	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1
Clear Bit(s) in Register		Очистити задані біт(и) у регістрі Rd (16<=d<=31). cbr r16,1 ; очистити біт 0 в r16 cbr r17,$f0 ; очистити 4-ри старші біти в r17			
INC	Rd	Інкремент регістра	Rd = Rd + 1	Z,N,V,S	1
Increment Register		Додати 1 до регістра Rd (0<=d<=31). clr r22 ; очистити r22 loop: inc r22 ; збільшити значення r22 на 1 ... cpi r22,$4F ; порівняти r22 і $4f brne loop ; перехід, якщо не рівні nop			
DEC	Rd	Декремент регістра	Rd = Rd -1	Z,N,V,S	1
Decrement Register		Відняти 1 від регістра Rd (0<=d<=31). ldi r17,$10 loop: add r1,r2 dec r17 ; зменшити значення r17 на 1 cpi r22,$4f ; порівняти r22 і $4f brne loop ; перехід, якщо r17<>0 nop			
TST	Rd	Тест на нуль або негативне	Rd = Rd · Rd	Z,C,N,V,S	1
Test for Zero or Negative		Перевірити чи регістр Rd (0<=d<=31) нульовий або негативний. tst r0 ; перевірити r0 breq zero ; перехід, якщо r0=0 ... zero: nop			
CLR	Rd	Очистити регістр	Rd = 0	Z,C,N,V,S	1
Clear Register		Очистити всі біти регістра Rd (0<=d<=31). clr r18 ; очистити r18 loop: inc r18 ; збільшити r18 ... cpi r18,$50 ; порівняти r18 і $50 brne loop			
SER	Rd	Встановити регістр	Rd = $FF		1
Set Register		Встановити всі біти регістра Rd (0<=d<=31). clr r16 ; очистити r16 ser r17 ; встановити всі біти r17 out $18,r16 ; записати всі нулі в Port B			

		nop ; затримка out $18,r17 ; записати всі одиниці в Port B			
SBIW	Rdl,K6	Відняти безпосереднє значення (0-63) від слова	Rdh:Rdl = Rdh:Rdl - K6	Z,C,N,V,S	2
Subtract Immediate from Word		Відняти безпосереднє значення K (0-63) від регістрової пари Rd+1:Rd d={24, 26, 28, 30}. sbiw r25:r24,1 ; відняти 1 від r25:r24 sbiw YH:YL,63 ; відняти 63 від вказівника Y (r29:r28)			
MUL	Rd,Rr	Множення беззнакове	R1:R0 = Rd * Rr	Z,C	2

Multiply Unsigned		Помножити беззнакове значення регістра Rd на беззнакове значення регістра Rr (0<=d<=31, 0<=r<=31) і результат записати в регістри R1:R0.
		mul r5,r4 ; помножити r4 на r5 (r1:r0=r5*r4)
		movw r4,r0 ; скопіювати результат в r5:r4			

MULS	Rd,Rr	Помножити знакові значення	R1:R0 = Rd * Rr	Z,C	2

Multiply Signed		Помножити знакове значення регістра Rd на знакове значення регістра Rr (16<=d<=31, 16<=r<=31) і результат записати в регістри R1:R0.
		muls r21,r20 ; помножити r20 на r21 (r1:r0=r21*r20)
		movw r20,r0 ; скопіювати результат в r21:r20			

MULSU	Rd,Rr	Помножити знакове і беззнакове значення	R1:R0 = Rd * Rr	Z,C	2

Multiply Signed with Unsigned		Помножити знакове значення регістра Rd на беззнакове значення регістра Rr (16<=d<=23, 16<=r<=23) і результат записати в регістри R1:R0.
		mulsu r21,r20 ; помножити r20 на r21 (r1:r0=r21*r20)
		movw r20,r0 ; скопіювати результат в r21:r20			

FMUL	Rd,Rr	Помножити беззнакові числа формату 1.7 із зсувом результату на 1 розряд вліво	R1:R0 = (Rd * Rr) << 1	Z,C	2
Fractional Multiply Unsigned		Помножити беззнакове значення у форматі 1.7 регістра Rd на беззнакове значення у форматі 1.7 регістра Rr (16<=d<=23, 16<=r<=23), результат зсунути на один розряд вліво і записати в регістри R1:R0 у форматі 1.15. fmul r23:r22 ; помножити r23 на r22 у форматі 1.7 і результат у форматі 1.15 (r1:r0=r21*r20) movw r23:r22.r1:r0 ; скопіювати результат в r21:r20			
FMULS	Rd,Rr	Помножити знакові числа формату 1.7 із зсувом результату на 1 розряд вліво	R1:R0 = (Rd *Rr) << 1	Z,C	2
Fractional Multiply Signed		Помножити знакове значення у форматі 1.7 регістра Rd на знакове значення у форматі 1.7 регістра Rr (16<=d<=23, 16<=r<=23), результат зсунути на один розряд вліво і записати в регістри R1:R0 у форматі 1.15. fmuls r23:r22,r1:r0 ; помножити r23 на r23 у форматі 1.7 і результат у форматі 1.15 (r1:r0=r23*r22) movw r23:r22,r1:r0 ; скопіювати результат в r23:r22			
FMULSU	Rd,Rr	Помножити знакове число формату 1.7 на беззнакове число формату 1.7 із зсувом результату на 1 розряд вліво	R1:R0 = (Rd * Rr) << 1	Z,C	2
Fractional Multiply Signed with Unsigned		Помножити знакове значення у форматі 1.7 регістра Rd на беззнакове значення у форматі 1.7 регістра Rr (16<=d<=23, 16<=r<=23), результат зсунути на один розряд вліво і записати в регістри R1:R0 у форматі 1.15. fmulsu r23:r22,r1:r0 ; помножити r23 на r23 у форматі 1.7 і результат у форматі 1.15 (r1:r0=r23*r22) movw r23:r22,r1:r0 ; скопіювати результат в r23:r22			

Таблиця 3.2 – Інструкції переходів (передачі керування)

Мнемо-ніка	Опе-ранд	Описання	Функція	Прапори	Цикли
Безумовні переходи					
RJMP	k	Безумовний відносний перехід	PC = PC + k +1		2

Relative Jump		Безумовний відносний перехід за адресою позначки k, (-2 Kwords<=k<=2 Kwords). Дозволяє адресувати 4 Kwords = 8 Кбайт програмної пам'яті. cpi r16,$42 ; порівняти r16 і $42 brne error ; перехід, якщо r16 <> $42 rjmp ok ; безумовний відносний перехід error: add r16,r17 ; add r17 до r16 inc r16 ; інкремент r16 ok: ; позначка для rjmp			
IJMP		Безумовний непрямий перехід за адресою регістра (Z)	PC = Z		2
Indirect Jump to (Z)		Безумовний непрямий перехід за адресою регістра Z. 16-бітовий регістр Z дозволяє адресувати 64 kwords = 128 кбайт програмної пам'яті. mov r30,r0 ; задати зміщення в Z для ijump ijmp ; перейти на підпрограму за адресою r31:r30			
JMP	k	Безумовний перехід за адресою k	PC = k		3
Jump		Безумовний непрямий перехід за адресою позначки k у межах всієї програмної пам'яті (0<=k<=4 Mwords). 4 Mwords = 8 Мбайт. Інструкція доступна тільки в ATmega168. ... jmp farp ; безумовний перехід на позначку farp ... farp:nop			
RCALL	k	Відносний виклик підпрограми	STACK = PC+1, PC = PC + k + 1		3/4*
Relative Call Subroutine		Відносний виклик підпрограми за адресою позначки k, (-2 kwords<=k<=2 kwords). Дозволяє адресувати 4 kwords = 8 кбайт програмної пам'яті. Адреса повернення записується у стек. rcall routine ; виклик підпрограми ... routine: push r14 ; записати r14 в стек ... pop r14 ; відновити r14 ret ; повернення з підпрограми			
ICALL		Непрямий виклик підпрограми за адресою (Z)	STACK = PC+1, PC = Z		3/4*
Indirect Call to (Z)		Непрямий виклик підпрограми за адресою регістра Z. 16-бітний регістр Z дозволяє адресувати 64 kwords = 128 кбайт програмної пам'яті. mov r30,r0 ; задання адреси підпрограми в Z icall ; виклик підпрограми за адресою в r31:r30			
CALL	k	Виклик підпрограми	STACK = PC+2, PC = k		4/5*
Call Subroutine		Виклик підпрограми за адресою позначки k у межах всієї програмної пам'яті (0<=k<=4 Mwords). 4 Mwords = 8 Мбайт. Інструкція доступна тільки в ATmega168. mov r16,r0 ; копіювати r0 в r16 call check ; виклик підпрограми nop ... check: cpi r16, $42 ; порівняння r16 і $42 breq error ; перехід, якщо r16 == $42 ret ; повернення з підпрограми ... error: rjmp error ; нескінченний цикл			
RET		Повернення з підпрограми	PC = STACK		4/5*
Subroutine Return		Повернення з підпрограми. Адреса повернення завантажується із стеку. call routine ; виклик підпрограми ... routine: push r14 ; записати r14 у стек ... pop r14 ; відновити r14 ret ; повернення з підпрограми			

RETI		Повернення з переривання	PC = STACK	I	4/5*
Interrupt Return		Повернення з переривання. Адреса повернення завантажується із стеку і встановлюється глобальний прапор переривань. Збереження регістра стану перед викликом підпрограми переривання і його відновлення після повернення з підпрограми переривання має зробити прикладна програма. extint: push r0 ; записати r0 у стек ... pop r0 ; відновити r0 reti ; повернення і встановлення дозволу на переривання			
Переходи за умовами					
CPSE	Rd,Rr	Порівняти Rd і Rr, якщо вони однакові то пропустити наступну інструкцію	if (Rd ==Rr) PC = PC 2 or 3		1/2/3
Compare, Skip if equal		Порівняти Rd і Rr, якщо Rd == Rr то пропустити наступну інструкцію (0<=d<=31, 0<=r<=31). inc r4 ; збільшити на 1 r4 cpse r4,r0 ; порівняти r4 і r0 neg r4 ; виконати, якщо r4<>r0 nop			
CP	Rd,Rr	Порівняти регістри Rd, Rr	Rd -Rr	Z,C,N,V,H,S	1
Compare		Порівняти Rd і Rr, за результатом порівняння (Rd-Rr) встановити прапори регістра стану SREG (0<=d<=31, 0<=r<=31). cp r4,r19 ; порівняти r4 і r19 brne noteq ; перейти на noteq, якщо r4 <> r19 ... noteq: nop			
CPC	Rd,Rr	Порівняти регістри Rd, Rr з врахуванням перенесення	Rd - Rr - C	Z,C,N,V,H,S	1
Compare with Carry		Порівняти Rd і Rr з врахуванням попереднього значення прапора перенесення/позики C, за результатом порівняння (Rd-Rr) встановити прапори регістра стану SREG (0<=d<=31, 0<=r<=31). ; порівняти r3:r2 з r1:r0 cp r2,r0 ; порівняти молодші байти cpc r3,r1 ; порівняти старші байти brne noteq ; перейти, якщо не рівні ... noteq:nop			
CPI	Rd,K8	Порівняти регістр з константою	Rd - K	Z,C,N,V,H,S	1
Compare with Immediate		Порівняти Rd і константу K, за результатом порівняння (Rd-K) встановити прапори регістра стану SREG (0<=d<=31, 0<=K<=255). cpi r19,3 ; порівняти r19 і 3 brne error ; перехід, якщо r19<>3 ... error: nop			
SBRC	Rr,b	Якщо біт b в рег. Rr очищений, то пропустити наступну інструкцію	if(Rr(b)==0) PC = PC + 2 or 3		1/2/3

Skip if bit in register cleared		Якщо заданий біт у регістрі Rr скинутий, то пропустити наступну інструкцію (0<=r<=31, 0<=b<=7).
		sub r0,r1 ; відняти r1 від r0
		sbrc r0,7 ; пропустити, якщо біт 7 в r0 скинутий
		sub r0,r1 ; виконується тільки при умові, якщо біт7
		; в r0 не скинутий
		nop			

SBRS	Rr,b	Якщо біт b в рег. Rr встановлений, то пропустити наступну інструкцію	if(Rr(b)==1) PC = PC + 2 or 3		1/2/3
Skip if bit in register set		Якщо заданий біт у регістрі Rr встановлений, то пропустити наступну інструкцію (0<=r<=31, 0<=b<=7). sub r0,r1 ; відняти r1 від r0 sbrs r0,7 ; пропустити, якщо біт 7 в r0 встановлений neg r0 ; виконується тільки при умові, якщо біт7 ; в r0 скинутий nop			

SBIC	P,b	Якщо біт b порту P очищений, то пропустити наступну інструкцію	if(I/O(P,b)==0) PC = PC + 2 or 3		1/2/3
Skip if bit in I/O register cleared		Якщо заданий біт у регістрі введення/виведення I/O скинутий, то пропустити наступну інструкцію (0<=P<=31, 0<=b<=7). e2wait: sbic $1C,1 ; пропустити наступну інструкцію якщо ; біт EEWE скинутий rjmp e2wait ; запис EEPROM не завершено nop			
SBIS	P,b	Якщо біт b порту P встановлений, то пропустити наступну інструкцію	if(I/O(P,b)==1) PC = PC + 2 or 3		1/2/3
Skip if bit in I/O register set		Якщо заданий біт у регістрі введення/виведення I/O встановлений, то пропустити наступну інструкцію (0<=P<=31, 0<=b<=7). waitset: sbis $10,0 ; пропустити наступну інструкцію якщо ; біт 0 Port D встановлений rjmp waitset ; біт не встановлений nop			
BRBC	s,k	Відносний перехід на k, якщо біт k в рег. SREG очищений	if(SREG(s)==0) PC = PC + k + 1		1/2
Branch if Status flag cleared		Якщо заданий біт s у регістрі стану SREG скинутий, то перейти за адресою позначки k (0<=s<=7, -64<=k<=63). cpi r20,5 ; Порівняти r20 і 5 brbc 1,noteq ; перейти, якщо прапор нуля Z скинутий ... noteq:nop			
BRBS	s,k	Відносний перехід на k, якщо біт k в рег. SREG встановлений	if(SREG(s)==1) PC = PC + k + 1		1/2
Branch if Status flag set		Якщо заданий біт s у регістрі стану SREG встановлений, то перейти за адресою позначки k (0<=s<=7, -64<=k<=63). bst r0,3 ; завантажити T біт з третього біта r0 brbs 6, bitset ; перейти, якщо біт T встановлений ... bitset: nop			
BREQ	k	Відносний перехід на k за рівністю (якщо біт Zero в рег. SREG встановлений)	if(Z==1) PC = PC + k + 1		1/2
Branch if equal		Якщо заданий біт Z у регістрі стану SREG встановлений, то перейти за адресою позначки k (-64<=k<=63). cp r1, r0 ; Порівняти регістри r1 і r0 breq equal ; перейти, якщо значення регістрів однакові ... equal: nop			
BRNE	k	Відносний перехід на k за нерівністю (якщо біт Zero в рег. SREG не встановлений)	if(Z==0) PC = PC + k + 1		1/2
Branch if not equal		Якщо заданий біт Z у регістрі стану SREG не встановлений, то перейти за адресою позначки k (-64<=k<=63). eor r27,r27 ; чистка r27 loop:inc r27 ; збільшення r27 ... cpi r27,5 ; порівняння r27 і 5 brne loop ; перехід, якщо r27<>5 nop			
BRCS	k	Відносний перехід на k, якщо біт Carry в рег. SREG встановлений	if(C==1) PC = PC + k + 1		1/2
Branch if carry set		Якщо заданий біт перенесення C у регістрі стану SREG встановлений, то перейти за адресою позначки k (-64<=k<=63). cpi r26, $56 ; порівняти r26 і $56 brcs carry ; перейти, якщо прапор C == 1 ...			

		carry: nop			
BRCC	k	Відносний перехід на k, якщо біт Carry в рег. SREG очищений	if(C==0) PC = PC + k + 1		1/2
Branch if carry cleared		Якщо заданий біт перенесення C у регістрі стану SREG встановлений, то перейти за адресою позначки k (-64<=k<=63). add r22,r23 ; додати r23 до r22 brcc nocarry ; перейти, якщо прапор C == 0 ... nocarry: nop			
BRSH	k	Відносний перехід на k, якщо прапор C==0 і Rd>=Rs (беззнакове)	if(C==0) PC = PC + k + 1		1/2
Branch if same or higher		Відносний перехід на k, якщо прапор перенесення C==0 і значення регістрів Rd>=Rs (0<=s,d<=31, -64<=k<=63). subi r19,4 ;відняти 4 від r19 brsh highsm ; перехід, якщо r19 >= 4 (беззнакове) ... highsm:nop			
BRLO	k	Відносний перехід на k, якщо прапор C==1 і Rd<Rs (беззнакове)	if(C==1) PC = PC + k + 1		1/2
Branch if lower		Відносний перехід на k, якщо прапор перенесення C==1 і значення регістрів Rd<Rs (0<=s,d<=31, -64<=k<=63). eor r19,r19 ; очистити r19 loop:incr19 ; збільшити на 1 r19 ... cpi r19,$10 ; порівняти r19 і $10 brlo loop ; перейти, якщо r19 < $10 (беззнакове) nop			
BRMI	k	Відносний перехід на k, якщо прапор Negative в рег. SREG встановлений	if(N==1) PC = PC + k + 1		1/2
Branch if minus		Відносний перехід на k, якщо прапор Negative в регістрі стану SREG встановлений (-64<=k<=63). subi r18,4 ; відняти 4 від r18 brmi negative ; перейти, якщо r18 негативне ... negative: nop			
BRPL	k	Відносний перехід на k, якщо прапор Negative в рег. SREG очищений	if(N==0) PC = PC + k + 1		1/2
Branch if plus		Відносний перехід на k, якщо прапор Negative в регістрі стану SREG скинутий (-64<=k<=63). subi r26,$50 ; відняти $50 від r26 brpl positive ; перейти, якщо r26 позитивне ... positive:nop			
BRGE	k	Відносний перехід на k, якщо прапор S==0 і Rd>=Rs (знакове)	if(S==0) PC = PC + k + 1		1/2
Branch if greater than or equal (signed)		Відносний перехід на k, якщо прапор знаку S==0 і значення регістрів Rd>=Rs (знакове) (0<=s,d<=31, -64<=k<=63). cp r11,r12 ; порівняти r11 і r12 brge greateq ; перейти, якщо r11 ≥ r12 (знакове) ... greateq: nop			
BRLT	k	Відносний перехід на k, якщо прапор S==0 і Rd<Rs	if(S==1) PC = PC + k + 1		1/2
Branch if less than (signed)		Відносний перехід на k, якщо прапор знаку S==0 і значення регістрів Rd<Rs (знакове) (0<=s,d<=31, -64<=k<=63). cp r16,r1 ; порівняти r16 і r1 brlt less ; перейти, якщо r16 < r1 (знакове)			

		... less:nop			
BRHS	k	Відносний перехід на k, якщо прапор Half carry встановлений	if(H==1) PC = PC + k + 1		1/2
Branch if half carry flag set		Відносний перехід на k, якщо прапор перенесення з півбайту H==1 (-64<=k<=63). brhs hset ; перейти, якщо прапор H == 1 ... hset: nop			
BRHC	k	Відносний перехід на k, якщо прапор Half carry очищений	if(H==0) PC = PC + k + 1		1/2
Branch if half carry flag cleared		Відносний перехід на k, якщо прапор перенесення з півбайту H==0 (-64<=k<=63). brhc hset ; перейти, якщо прапор H == 0 ... Hclear: nop			
BRTS	k	Відносний перехід на k, якщо прапор T встановлений	if(T==1) PC = PC + k + 1		1/2
Branch if T flag set		Відносний перехід на k, якщо біт T регістра SREG встановлений (-64<=k<=63). bst r3,5 ; записати біт 5 з r3 у прапор T brts tset ; перейти, якщо T == 1 ... tset:nop			
BRTC	k	Відносний перехід на k, якщо прапор T очищений	if(T==0) PC = PC + k + 1		1/2
Branch if T flag cleared		Відносний перехід на k, якщо біт T регістра SREG скинутий (-64<=k<=63). bst r3,5 ; записати біт 5 з r3 у прапор T brtc tset ; перейти, якщо T == 1 ... tclear:nop			
BRVS	k	Відносний перехід на k, якщо прапор переповнення Overflow встановлений	if(V==1) PC = PC + k + 1		1/2
Branch if overflow flag set		Відносний перехід на k, якщо прапор переповнення V встановлений (-64<=k<=63). add r3,r4 ; додати r4 до r3 brvs overfl ; перейти, при переповненні V == 1 ... overfl:nop			
BRVC	k	Відносний перехід на k, якщо прапор переповнення Overflow очищений	if(V==0) PC = PC + k + 1		1/2
Branch if overflow flag cleared		Відносний перехід на k, якщо прапор переповнення V очищений (-64<=k<=63). add r3,r4 ; додати r4 до r3 brvc noover ; перейти, при переповненні V == 1 ... noover:nop			
BRIE	k	Відносний перехід на k, якщо прапор глобального переривання Interrupt встановлено	if(I==1) PC = PC + k + 1		1/2
Branch if interrupt enabled		Відносний перехід на k, якщо прапор глобального переривання I встановлено (-64<=k<=63). brie inten ; перейти, якщо I == 1 ... inten:nop			
BRID	k	Відносний перехід на k, якщо прапор глобального переривання Interrupt скинуто	if(I==0) PC = PC + k + 1		1/2
Branch if interrupt		Відносний перехід на k, якщо прапор глобального переривання I скинуто (-64<=k<=63).			

disabled	brie intdis ; перейти, якщо I == 0 ... intdis:nop

Таблиця 6.3 – Інструкції пересилання і завантаження даних

Мнемо-ніка	Операнди	Описання	Функція	Прапори	Цикли
MOV	Rd,Rr	Скопіювати регістр	Rd = Rr		1
Copy register		Скопіювати вміст регістра Rr у регістр Rd (0<=d<=31, 0<=r<=31). mov r16,r0 ; копіювати r0 в r16 call check ; виклик підпрограми ... check: cpi r16,$11 ; порівняння r16 і $11 ... ret ; повернення з підпрограми			
MOVW	Rd,Rr	Скопіювати регістрову пару	Rd+1:Rd = Rr+1:Rr, r,d парне		1
Copy register pair		Скопіювати вміст однієї регістрової пари у іншу Rd+1:Rd=Rr+1:Rr, d,r {0,2,…,30}. movw r17:16,r1:r0 ; копіювати r1:r0 в r17:r16 call check ; виклик підпрограми ... check: cpi r16,$11 ; порівняння r16 і $11 ... cpi r17,$32 ; порівняння r17 і $32 ... ret ; повернення з підпрограми			
LDI	Rd,K8	Завантажити у регістр константу	Rd = K		1
Load Immediate		Завантажити константу K у регістр Rd (16<=d<=31, 0<=K<=255) clr r31 ; чистка старшого байта Z. ldi r30, $F0 ; завантажити у молодший байт Z $F0 lpm ; завантажити з пам'яті програм константу яка ; адресується вказівником Z			
LDS	Rd,k	Завантажити у регістр байт за адресою пам'яті	Rd = (k)		2*

Load Direct		Завантажити у регістр Rd байт з 16-розрядною адресою з простору даних (регістровий файл, дані I/O, внутрішня і зовнішня SRAM пам'ять) (0<=d<=31, 0<=K<=65535).
		lds r2,$FF00 ; завантажити в r2 байт з адресою $FF00
		add r2,r1 ; додати r1 до r2
		sts $FF00,r2 ; записати байт назад			

LD	Rd,X	Завантажити у регістр дані з пам'яті, адреса якої міститься у регістрі вказівнику X	Rd = (X)		2*

Load Indirect		Завантажити у регістр Rd байт з 16-розрядною адресою з простору даних яка знаходиться у регістрі вказівнику X (регістровий файл, дані I/O, внутрішня і зовнішня SRAM пам'ять) (0<=d<=31).
		clr r27 ; очистити старший байт X
		ldi r26,$60 ; завантажити у молодший байт X $60
		ld r0,X+ ; завантажити в r0 байт за адресою X
		;(збільшити на 1 значення X)
		ld r1,X ; завантажити в r1 байт за адресою X
		ldi r26,$63 ; завантажити в молодший байт X $63
		ld r2,X ; завантажити в r2 байт за адресою X
		ld r3,-X ; зменшити на 1 значення X, завантажити в r3
		; за адресою X			

LD	Rd,X+	Завантажити у регістр дані з пам'яті, адреса якої задана в X та інкрементувати X	Rd = (X), X=X+1		2*
Load Indirect and Post-		Завантажити у регістр Rd байт з 16-розрядною адресою з простору даних яка знаходиться у			

Increment		регістрі вказівнику X (регістровий файл, дані I/O, внутрішня і зовнішня SRAM пам'ять) (0<=d<=31) і збільшити на 1 значення X.
		clr r27 ; очистити старший байт X
		ldi r26,$60 ; завантажити в молодший байт X $60
		ld r0,X+ ; завантажити в r0 байт за адресою в X
		; (збільшити на 1 значення регістра X)
		ld r1,X ; завантажити в r1 байт за адресою в X
		ldi r26,$63 ; завантажити в молодший байт X $63
		ld r2,X ; завантажити в r2 байт за адресою в X
		ld r3,-X ; зменшити на 1 значення X і завантажити в r3
		; байт за адресою в X			

LD	Rd,-X	Декрементувати X та завантажити у регістр дані з пам'яті, адреса якої міститься регістрі X	X=X-1, Rd = (X)		2*

Load Indirect and Pre-Decrement		Зменшити на 1 значення X, завантажити у регістр Rd байт з 16-розрядною адресою з простору даних яка знаходиться у регістрі вказівнику X (регістровий файл, дані I/O, внутрішня і зовнішня SRAM пам'ять) (0<=d<=31).
		clr r27 ; очистити старший байт X
		ldi r26,$60 ; завантажити в молодший байт X $60
		ld r0,X+ ; завантажити в r0 байт за адресою в X
		; (збільшити на 1 значення регістра X)
		ld r1,X ; завантажити в r1 байт за адресою в X
		ldi r26,$63 ; завантажити в молодший байт X $63
		ld r2,X ; завантажити в r2 байт за адресою в X
		ld r3,-X ; зменшити на 1 значення X і завантажити в r3
		; байт за адресою в X			

LD	Rd,Y	Завантажити у регістр дані з пам'яті, адреса якої міститься в регістрі вказівнику Y	Rd = (Y)		2*
Load Indirect		Аналогічно до ld Rd,X			
LD	Rd,Y+	Завантажити у регістр дані з пам'яті, адреса якої задана у регістрі Y та інкрементувати Y	Rd = (Y), Y=Y+1		2*
Load Indirect and Post-Increment		Аналогічно до ld Rd,X+			
LD	Rd,-Y	Декрементувати Y та завантажити у регістр дані з пам'яті, адреса якої міститься регістрі Y	Y=Y-1, Rd = (Y)		2*
Load Indirect and Pre-Decrement		Аналогічно до ld Rd,-X			
LDD	Rd,Y+q	Завантажити у регістр дані з пам'яті, адреса якої задано в регістрі Y із зміщенням q	Rd = (Y+q)		2*

Load Indirect with displacement		Завантажити один байт з або без зміщення q з простору даних (регістровий файл, дані I/O, внутрішня і зовнішня SRAM пам'ять) (0<=d<=31, 0<=q<=63), який адресується регістром Y.
		clr r29 ; очистити старший байт Y
		ldi r28,$60 ; завантажити в молодший байт Y $60
		ld r0,Y+ ; завантажити в r0 байт за адресою $60
		; збільшити на 1 значення Y
		ld r1,Y ; завантажити в r1 байт за адресою $61
		ldi r28,$63 ; завантажити у молодший байт Y $63
		ld r2,Y ; завантажити у r2 байт за адресою $63
		ld r3,-Y ; зменшити на 1 значення Y, завантажити байт
		; за адресою Y у регістр r3
		ldd r4,Y+2 ; завантажити у r4 байт за адресою $64			

LD	Rd,Z	Завантажити у регістр дані з пам'яті, адреса якої міститься в регістрі вказівнику Z	Rd = (Z)		2*
Load Indirect		Аналогічно до ld Rd,X			
LD	Rd,Z+	Завантажити у регістр дані з пам'яті, адреса якої задана в регістрі Z та інкрементувати Z	Rd = (Z), Z=Z+1		2*
Load Indirect and Post-Increment		Аналогічно до ld Rd,X+			
LD	Rd,-Z	Декрементувати Z та завантажити у регістр дані	Z=Z-1, Rd = (Z)		2*

		з пам'яті, адреса якої міститься регістрі Z			
Load Indirect and Pre-Decrement		Аналогічно до ld Rd,-X			
LDD	Rd,Z+q	Завантажити у регістр дані з пам'яті, адреса якої задано в регістрі Z із зміщенням q	Rd = (Z+q)		2*
Load Indirect with displacement		Аналогічно до ldd Rd,Y+q			
STS	k,Rr	Прямий запис у пам'ять з регістра	(k) = Rr		2*

Store Direct		Записати байт з регістра у простір пам'яті (регістровий файл, дані I/O, внутрішня і зовнішня SRAM пам'ять) за адресою k (0<=r<=31, 0<=k<65535).
		lds r2,$FF00 ; завантажити в r2 байт за адресою $FF00
		add r2,r1 ; додати r1 до r2
		sts $FF00,r2 ; записати r2 за адресою $FF00			

ST	X,Rr	Непрямий запис у пам'ять з регістра	(X) = Rr		2*

Store Indirect		Записати байт з регістра Rr у простір пам'яті, яка адресується регістром вказівником X (0<=r<=31).
		clr r27 ; очистити старший байт X
		ldi r26,$60 ; завантажити у молодший байт X $60
		st X+,r0 ; записати r0 у простір пам'яті, яка адресується
		; регістром X, збільшити на 1 значення X
		st X,r1 ; записати r1 у простір пам'яті з адресою $61
		ldi r26,$63 ; завантажити у молодший байт X $63
		st X,r2 ; записати r2 у простір пам'яті з адресою $63
		st -X,r3 ; зменшити на 1 значення X, записати r3 у
		; простір пам'яті з адресою $62			

ST	X+,Rr	Непрямий запис у пам'ять з після інкрементом	(X) = Rr, X=X+1		2*

Store Indirect and Post-Increment		Записати байт з регістра Rr у простір пам'яті, яка адресується регістром вказівником X (0<=r<=31), збільшити на 1 значення X.
		clr r27 ; очистити старший байт X
		ldi r26,$60 ; завантажити у молодший байт X $60
		st X+,r0 ; записати r0 у простір пам'яті, яка адресується
		; регістром X, збільшити на 1 значення X
		st X,r1 ; записати r1 у простір пам'яті з адресою $61
		ldi r26,$63 ; завантажити у молодший байт X $63
		st X,r2 ; записати r2 у простір пам'яті з адресою $63
		st -X,r3 ; зменшити на 1 значення X, записати r3 у
		; простір пам'яті з адресою $62			

ST	-X,Rr	Непрямий запис у пам'ять з перед інкрементом	X=X-1, (X)=Rr		2*

Store Indirect and Pre-Decrement		Зменшити на 1 значення X, записати байт з регістра Rr у простір пам'яті, яка адресується регістром вказівником X (0<=r<=31).
		clr r27 ; очистити старший байт X
		ldi r26,$60 ; завантажити у молодший байт X $60
		st X+,r0 ; записати r0 у простір пам'яті, яка адресується
		; регістром X, збільшити на 1 значення X
		st X,r1 ; записати r1 у простір пам'яті з адресою $61
		ldi r26,$63 ; завантажити у молодший байт X $63
		st X,r2 ; записати r2 у простір пам'яті з адресою $63
		st -X,r3 ; зменшити на 1 значення X, записати r3 у
		; простір пам'яті з адресою $62			

ST	Y,Rr	Непрямий запис у пам'ять з регістра	(Y) = Rr		2*
Store Indirect		Аналогічно до st X,Rr			
ST	Y+,Rr	Непрямий запис у пам'ять з після інкрементом	(Y) = Rr, Y=Y+1		2
Store Indirect and Post-Increment		Аналогічно до st X+,Rr			
ST	-Y,Rr	Непрямий запис у пам'ять з перед інкрементом	Y=Y-1, (Y) = Rr		2
Store Indirect and Pre-Decrement		Аналогічно до st -X,Rr			

STD	Y+q,Rr	Непрямий запис у пам'ять із зміщенням	(Y+q) = Rr		2

Store Indirect with displacement		Записати один байт з або без зміщення q з регістра Rr у простір даних (регістровий файл, дані I/O, внутрішня і зовнішня SRAM пам'ять) (0<=d<=31, 0<=q<=63), який адресується регістром Y.
		clr r29 ; очистити старший байт регістра Y
		ldi r28,$60 ; завантажити у молодший байт Y $60
		st Y+,r0 ; записати r0 у простір пам'яті за адресою в Y,
		; збільшити на 1 значення Y
		st Y,r1 ; записати r1 у простір пам'яті за адресою $61
		ldi r28,$63 ; завантажити у молодший байт Y $63
		st Y,r2 ; записати r2 у простір пам'яті за адресою $63
		st -Y,r3 ; зменшити на 1 значення Y, записати r3 у простір пам'яті за адресою $62
		std Y+2,r4 ; записати r4 у простір пам'яті за адресою $64			

ST	Z,Rr	Непрямий запис у пам'ять із зміщенням	(Z) = Rr		2
Store Indirect		Аналогічно до st X,Rr			
ST	Z+,Rr	Непрямий запис у пам'ять з після інкрементом	(Z) = Rr, Z=Z+1		2
Store Indirect and Post-Increment		Аналогічно до st X+,Rr			
ST	-Z,Rr	Непрямий запис у пам'ять з перед інкрементом	Z=Z-1, (Z) = Rr		2
Store Indirect and Pre-Decrement		Аналогічно до st -X,Rr			
STD	Z+q,Rr	Непрямий запис у пам'ять із зміщенням	(Z+q) = Rr		2
Store Indirect with displacement		Аналогічно до std Y+q,Rr			
LPM	None	Завантажити з пам'яті програм у регістр R0	R0 = (Z)		3
Load Program Memory		Завантажити один байт у регістр r0 з пам'яті програм, яка адресується регістром Z. clr r31 ; очистити старший байт регістра Y ldi r30,$60 ; завантажити у молодший байт Y $60 clr r0 lpm			
LPM	Rd,Z	Завантажити з пам'яті програм у регістр	Rd = (Z)		3

Load Program Memory		Завантажити один байт у регістр Rd (0<=d<=31) з пам'яті програм (16-бітові слова), яка адресується регістром Z.
		ldi ZH, high(Table_1<<1) ; ініціалізації регістра Z
		ldi ZL, low(Table_1<<1)
		lpm r16,Z ; завантажити у r16 константу з програмної
		; пам'яті, адресованої регістром Z (r31:r30)
		...
		Table_1:
		.dw 0x5876 ; 0x76 є адреса для ZLSB = 0
		; 0x58 є адреса для ZLSB = 1			

LPM	Rd,Z+	Завантажити з пам'яті програм у регістр з після інкрементом	Rd = (Z), Z=Z+1		3
Load Program Memory and Post-Increment		Завантажити один байт у регістр Rd (0<=d<=31) з пам'яті програм (16-бітові слова), яка адресується регістром Z, збільшити на 1 значення Z.			
SPM	None	Записати у пам'ять програм	(Z) = R1:R0		-
Store Program Memory		Використовується для чистки сторінки програмної пам'яті, для запису сторінки у програмну пам'ять, для встановлення бітів блокування завантажувача. В деяких пристроях пам'ять може записуватися словами, а у деяких сторінками. Для адресації слів і сторінок використовуються регістри RAMPZ:Z, а для даних – регістри r1:r0.			
IN	Rd,P	Завантажити з порту у регістр	Rd = P		1
In Port		Завантажити дані з простору введення-виведення (порти, таймери, конфігураційні регістри) у регістр Rd (0<=d<=31, 0<=P<=63). in r25,$16 ; прочитати Port B у r25 cpi r25,4 ; порівняти r25 і константу 4 breq exit ; перехід, якщо r25=4			

		... exit: nop			
OUT	P,Rr	Записати з регістра у порт	P = Rr		1
Out Port		Записати дані з регістра Rr у простір введення-виведення (порти, таймери, конфігураційні регістри) (0<=r<=31, 0<=P<=63). clr r16 ; очистити r16 ser r17 ; записати $ff у r17 out $18,r16 ; записати нулі в Port B nop out $18,r17 ; записати одиниці в Port B			
PUSH	Rr	Записати вміст регістра в стек	STACK = Rr		2
Push register on Stack		Записати дані з регістра Rr у стек (0<=r<=31). call routine ; виклик підпрограми ... routine: push r14 ; записати r14 у стек push r13 ; записати r13 у stack ... pop r13 ; прочитати із стеку в r13 pop r14 ; прочитати із стеку в r14 ret ; повернення з підпрограми			
POP	Rd	Завантажити регістр із стеку	Rd = STACK		2
Pop register from Stack		Завантажити дані із стеку в регістр Rd у стек (0<=d<=31). call routine ; виклик підпрограми ... routine: push r14 ; записати r14 у стек push r13 ; записати r13 у стек ... pop r13 ; прочитати із стеку в r13 pop r14 ; прочитати із стеку в r14 ret ; повернення з підпрограми			

Таблиця 3.4 – Порозрядні інструкції

Мне-моніка	Опе-ранд	Описання	Функція	Прапори	Цик ли
LSL	Rd	Логічний зсув вліво із встановленням перенесення	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H, S	1
Logical shift left		Зсунути всі біти регістра Rd на один розряд вліво. Біт 0 очистити, а біт 7 записати у прапор C регістра стану SREG (0<=d<=31). Ця операція ефективно множить на 2 беззнакові і знакові числа. add r0, r4 ; додати r4 до r0 lsl r0 ; помножити r0 на 2			
LSR	Rd	Логічний зсув вправо із встановленням перенесення	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
Logical shift right		Зсунути всі біти регістра Rd на один розряд вправо. Біт 7 очистити, а біт 0 записати у прапор C регістра стану SREG (0<=d<=31). Ця операція ефективно ділить на 2 беззнакові числа. add r0, r4 ; додати r4 до r0 lsr r0 ; поділити r0 на 2			
ROL	Rd	Логічний зсув вліво через біт перенесення	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H, S	1
Rotate left through carry		Зсунути всі біти регістра Rd на один розряд вліво. Прапор C записати у біт 0, а біт 7 записати у прапор C регістра стану SREG (0<=d<=31). Ця операція разом із LSL ефективно множить на 2 багатобайтові беззнакові і знакові числа. lsl r18 ; помножити r19:r18 на 2 rol r19 ; знакове або беззнакове число brcs oneenc			

		... oneenc: nop			
ROR	Rd	Логічний зсув вправо через біт перенесення	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
Rotate right through carry		Зсунути всі біти регістра Rd на один розряд вправо. Прапор C записати у біт 7, а біт 0 записати у прапор C регістра стану SREG (0<=d<=31). Ця операція разом з ASR ефективно ділить на 2 багатобайтові знакові числа, а разом з LSR ефективно ділить на 2 багатобайтові ,беззнакові числа. lsr r19 ; поділити r19:r18 на два ror r18 ; r19:r18 беззнакове двобайтове ціле brcc zeroenc1 ; перейти, якщо прапор C==0 asr r17 ; поділити r17:r16 на два ror r16 ; r17:r16 знакове двобайтове ціле brcc zeroenc2 ; перейти, якщо C==0 ... zeroenc1:nop ... zeroenc2:nop			
ASR	Rd	Арифметичний зсув вправо	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1

Arithmetic shift right		Зсунути всі біти регістра Rd на один розряд вправо. Біт 7 зберігає своє значення, а біт 0 записати у прапор C регістра стану SREG (0<=d<=31). Ця операція разом ефективно ділить на 2 знакове числа із збереженням його знаку.
		ldi r16,$10 ; завантажити $10 у r16
		asr r16 ; r16=r16 / 2
		ldi r17,$FC ; завантажити -4 у r17
		asr r17 ; r17=r17/2			

SWAP	Rd	Переставлення тетрад	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)		1
Swap nibbles		Переставити тетради регістра Rd (0<=d<=31). inc r1 ; збільшити на 1 r1 swap r1 ; переставити тетради r1 inc r1 ; збільшити на 1 старшу тетраду r1 swap r1 ; переставити старшу тетраду на своє місце			
BSET	s	Встановити біт s в SREG	SREG(s) = 1	SREG(s)	1
Set flag		Встановити біт s у регістрі стану SREG (0<=s<=7). bset 6 ; встановити прапор T bset 7 ; дозволити переривання			
BCLR	s	Очистити біт s в SREG	SREG(s) = 0	SREG(s)	1
Clear flag		Очистити біт s у регістрі стану SREG (0<=s<=7). bclr 0 ; очистити прапор перенесення C bset 7 ; заборонити переривання			
SBI	P,b	Встановити біт b в регістрі I/O	I/O(P,b) = 1		2
Set bit in I/O register		Встановити біт b у регістрі введення/виведення I/O (0<=P<=31, 0<=b<=7). out $1E,r0 ; записати адресу EEPROM sbi $1C,0 ; встановити біт читання в EECR in r1,$1D ; читати дані EEPROM			
CBI	P,b	Очистити біт b в регістрі I/O	I/O(P,b) = 0		2
Clear bit in I/O register		Очистити біт b у регістрі введення/виведення I/O (0<=P<=31, 0<=b<=7). cbi $12,7 ; очистити біт 7 Port D			
BST	Rr,b	Записати біт b з регістра Rr у біт T регістрі SREG	T = Rr(b)	T	1
Bit store from register to T		Записати біт b з регістра Rr у біт T регістрі SREG (0<=r<=31, 0<=b<=7). bst r1,2 ; записати біт 2 регістра r1 у прапор T bld r0,4 ; завантажити біт T у біт 4 регістра r0			
BLD	Rd,b	Скопіювати біт T у біт b рег. Rd	Rd(b) = T		1
Bit load from register to T		Скопіювати прапор T регістра SREG у біт b регістра Rd (0<=d<=31, 0<=b<=7). bst r1,2 ; записати біт 2 регістра r1 у прапор T			

		bld r0,4 ; завантажити біт T у біт 4 регістра r0			
SEC		Встановити біт Carry	C =1	C	1
Set carry flag		Встановити прапор перенесення C регістра стану SREG. sec ; встановити прапор C == 1 adc r0,r1 ; r0=r0+r1+1			
CLC		Очистити біт Carry	C = 0	C	1
Clear carry flag		Очистити прапор перенесення C регістра стану SREG. add r0, r0 ; додати r0 до r0 clc ; очистити прапор перенесення			
SEN		Встановити біт Negative	N = 1	N	1
Set negative flag		Встановити прапор негативного значення N регістра стану SREG. add r2,r19 ; додати r19 до r2 sen ; встановити прапор N			
CLN		Очистити біт Negative	N = 0	N	1
Clear negative flag		Очистити прапор негативного значення N регістра стану SREG. add r2,r3 ; додати r3 до r2 cln ; очистити прапор N			
SEZ		Встановити біт Zero	Z = 1	Z	1
Set zero flag		Встановити прапор нульового значення N регістра стану SREG. add r2,r19 ; додати r19 to r2 sez ; встановити прапор Z			
CLZ		Очистити біт Zero	Z = 0	Z	1
Clear zero flag		Очистити прапор нульового значення N регістра стану SREG. add r2,r19 ; додати r19 to r2 clz ; очистити прапор Z			
SEI		Встановити біт Interrupt	I = 1	I	1
Set interrupt flag		Встановити прапор глобального переривання I регістра стану SREG. sei ; дозволити глобальні переривання sleep ; затримка, очікування на переривання			
CLI		Очистити біт Interrupt	I = 0	I	1
Clear interrupt flag		Очистити прапор глобального переривання I регістра стану SREG. Переривання будуть негайно заборонені. in temp, SREG ; записати значення SREG у регістр temp cli ; заборонити переривання на часову послідовність sbi EECR, EEMWE ; почати запис EEPROM sbi EECR, EEWE out SREG, temp ; відновити значення SREG (I прапор)			
SES		Встановити біт Signed	S = 1	S	1
Set signed flag		Встановити прапор знаку S регістра стану SREG. add r2,r19 ; додати r19 до r2 ses ; встановити прапор S			
CLS		Очистити біт Signed	S = 0	S	1
Clear signed flag		Очистити прапор знаку S регістра стану SREG. add r2,r19 ; додати r19 до r2 cls ; очистити прапор S			
SEV		Встановити біт Overflow	V = 1	V	1
Set overflow flag		Встановити прапор переповнення V регістра стану SREG. add r2,r19 ; додати r19 до r2 sev ; очистити прапор V			
CLV		Очистити біт Overflow	V = 0	V	1
Clear overflowflag		Очистити прапор переповнення V регістра стану SREG. add r2,r19 ; додати r19 до r2 clv ; очистити прапор V			

SET		Встановити біт T	T = 1	T	1
Set T-flag		Встановити прапор T регістра стану SREG. set ; встановити прапор T			
CLT		Очистити біт T	T = 0	T	1
Clear T-flag		Очистити прапор T регістра стану SREG. clt ; очистити прапор T			
SEH		Встановити біт Half carry	H = 1	H	1
Set half carry flag		Встановити прапор перенесення з тетради регістра стану SREG. seh ; встановити прапор H			
CLH		Очистити біт Half carry	H = 0	H	1
Clear half carry flag		Очистити прапор перенесення з тетради регістра стану SREG. clh ; очистити прапор H			
Інструкції керування ЦП					
NOP		Немає операції			1
No operation		Інструкція триває один машинний такт без виконання операцій. clr r16 ; очистити r16 ser r17 ; встановити r17 out $18,r16 ; записати нулі у Port B nop ; затримка на один такт out$18,r17 ; записати одиниці у Port B			
SLEEP		Перемкнути МК в “режим сну			1

Sleep		Інструкція перемикає схеми в режим ”сну”, визначений керуючим регістром MCU.
		mov r0,r11 ; копіює r11 у r0
		ldi r16,(1<<SE) ; дозвіл режиму сну
		out MCUCR, r16
		sleep ; Перемкнути MCU в режим сну			

WDR		Перевстановити сторожовий таймер			1
Watchdog Reset		Перевстановлення сторожового таймера. Інструкція має бути виконана в за обмежений час, який задається в подільнику WD.			

Висновки.

В МК AVR для позначення результату виконання інструкції використовують прапори, які входять в склад регістра стану SREG. При вході в підпрограму оброблення переривань вміст регістра стану рекомендується зберігати, а після виходу з цієї підпрограми – відновлювати, щоб програма продовжила роботу з гарантовано коректними розрядами умов.

Команди AVR асемблера поділяються на наступні групи: арифметичні і логічні інструкції; пересилання і завантаження даних; переходів (передачі управління); для роботи з розрядами; керування роботою ЦП.

В асемблері використовуються константні вирази. Вирази можуть складатися з операндів, операцій і функцій. Вирази можуть записуватися у круглих дужках і тоді вони завжди спочатку оцінюються, а потім об’єднуються з виразами поза дужками.

Для адресації пам’яті даних використовуються режими адресації: безпосередня, пряма, непряма з використанням регістрів X,Y,Z та їх перед декрементом і після декрементом, із зміщенням (відносна). Для адресації пам’яті програм FLASH використовується непряма, пряма і відносна адресація. Бітова адресація застосовується до регістрів. Регістри з R26 по R31 реєстрового файлу працюють як X, Y і Z регістри вказівники непрямої адресації.

Питання.

1. Особливості синтаксису асемблера МК AVR.
2. Операнди і оператори асемблера МК AVR.
3. Функції асемблера МК AVR.
4. Препроцесор і вбудовані макроси МК AVR.
5. Режими адресації програм і даних в МК AVR.
6. Арифметичні інструкції додавання і віднімання.
7. Логічні інструкції.
8. Інструкції безумовних і умовних переходів.
9. Інструкції копіювання і завантаження даних.
10. Інструкції записування даних.
11. Інструкції для роботи з портами і стеком.
12. Інструкції логічного, циклічного і арифметичного зсувів.
13. Інструкції встановлення і скидання бітів в регістрах.

ЛЕКЦІЯ 4. Директиви і макроси мови асемблера для мікроконтролера

Мета. Вивчення директив і макросів мови асемблера для мікроконтролера

Вступ. Програми на асемблері містять директиви, які не транслюються в машинні команди, а тільки використовується в процесі асемблювання. Директиви дозволяють структурувати програму асемблера. В програмах на асемблері використовуються макроси, які дозволяють зменшити розмір основної програми.

План.

1. Директиви AVR асемблера
    1.1. Директива BYTE
    1.2. Директива CSEG
    1.3. Директива CSEGSIZE
    1.4. Директиви DB, DW, DD, DQ
    1.5. Директива DEF
    1.6. Директива DEVICE
    1.7. Директива DSEG
    1.8. Директиви ELIF, ELSE
    1.9. Директива ENDIF
    1.10. Директиви ENDM, ENDMACRO
    1.11. Директива EQU
    1.12. Директива ESEG
    1.13. Директива EXIT
    1.14. Директиви IF, IFDEF, IFNDEF
    1.15. Директива INCLUDE
    1.16. Директива LIST
    1.17. Директива LISTMAC
    1.18. Директива MACRO
    1.19. Директива MESSAGE
    1.20. Директива NOLIST
    1.21. Директива ORG
    1.22. Директива OVERLAP/NOOVERLAP
    1.23. Директива SET
    1.24. Директива UNDEF
    1.25. Директива WARNING
2. Директиви і структура асемблерної програми
3. Завантаження даних у регістри
4. Створення і використання макросів

1. Директиви

При написанні програм на мові асемблера використовують директиви. Директиви не транслюються безпосередньо в інструкції, а вказують компілятору положення програми в пам’яті, визначають макроси, ініціалізують пам’ять і т. і. Всі директиви починаються з крапки. Перелік директив показано в табл. 4.1.

Таблиця 4.1 – Директиви

Директиви	Описання	
.BYTE	Резервування N байтів в пам’яті SRAM або EEPROM	`.equ N 32` `table:  .byte N`
.CSEG	Сегмент коду	
.CSEGSIZE	Розмір пам’яті програм FLASH	`.CSEGSIZE = 10 | 12 | 14 | 16`

.DB	Резервування байту (байтів) в пам'яті програм FLASH або EEPROM	`.CSEG`
		`consts: .DB 0, 255, 0b01010101, -128, 0xaa`
		`.ESEG`
		`const2: .DB 1,2,3`

.DD	Резервування _слова (слів) в пам'яті програм FLASH або EEPROM	`.CSEG` `varlist: .DD 0, 0xfadebabe, -2147483648, 1 << 30`
.DEF	Присвоєння регістру символьного імені	`.def   temp = r16`
.DD	Резервування четвірного слова (слів) в пам'яті програм FLASH або EEPROM	`.ESEG` `eevarlst: .DQ 0,0xfadebabedeadbeef, 1 << 62`
.DEVICE	Задання асемблеру моделі МК	`.device atmega8`

.DSEG	Сегмент даних в SRAM	`.DSEG ; Start data segment`
		`var1: .BYTE 1 ; reserve 1 byte to var1`
		`table: .BYTE tab_size ; reserve tab_size bytes`

.DW	Резервування_подвійного слова (2-х слів) в пам'яті програм FLASH або EEPROM	`.CSEG`
		`varlist: .DW 0, 0xffff, 0b1001110001010101, -32768, 65535`
		`.ESEG`
		`eevarlst: .DW 0,0xffff,10`

.ELIF, .ELSE	Асемблювання за умовою	`.IFDEF DEBUG`
		`.MESSAGE "Debugging.."`
		`.ELSE`
		`.MESSAGE "Release.."`
		`.ENDIF`

.ENDIF	Асемблювання за умовою	`.MESSAGE "Release.."`
		`.ELSE`
		`.MESSAGE "Debugging.."`
		`.ENDIF`

.ENDM, .ENDMACRO	Кінець макровизначення	`.MACRO SUBI16 ; Start macro definition`
		`subi r16,low(@0) ; Subtract low byte`
		`sbci r17,high(@0) ; Subtract high byte`
		`.ENDMACRO`

.EQU	Присвоїти значення символьній позначці. Така позначка не може бути змінена або перевизначена	`.EQU io_offset = 0x23`
		`.EQU porta = io_offset + 2.CSEG ; Start code segment`
		`clr r2 ; Clear register 2`
		`out porta,r2 ; Write to Port A`

.ERROR	Виводить повідомлення про помилку	`.IFDEF TOBEDONE`
		`.ERROR "Error.."`
		`.ENDIF`

.ESEG	Визначає початок сегменту констант EEPROM	`.ESEG` `eevar1: .DW 0xffff ; initialize 1 word in EEPROM`
.EXIT	Вихід з файлу	`.EXIT ; Exit this file`

.IF. .IFDEF, .IFNDEF	Асемблювання за умовою	`.MACRO SET_BAT`
		`.IF @0>0x3F`
		`.MESSAGE "Address larger than 0x3f"`
		`lds @2, @0`
		`sbr @2, (1<<@1)`
		`sts @0, @2`
		`.ELSE`

		.MESSAGE "Address less or equal 0x3f" .ENDIF .ENDMACRO
.INCLUDE	Прочитати asm код із вказаного файлу	; iodefs.asm: .EQU sreg = 0x3f ; Status register .EQU sphigh = 0x3e ; Stack pointer high .EQU splow = 0x3d ; Stack pointer low ; incdemo.asm .INCLUDE iodefs.asm ; Include I/O definitions in r0,sreg ; Read status register
.LIST	Включити генерацію роздруку програми	.NOLIST ; Disable listfile generation .INCLUDE "macro.inc" ; The included files will not .INCLUDE "const.def" ; be shown in the listfile .LIST ; Reenable listfile generation
.LISTMAC	Включити вставлення макророзширень у файл роздруку	.MACRO MACX ; Define an example macro add r0,@0 ; Do something eor r1,@1 ; Do something .ENDMACRO ; End macro definition .LISTMAC ; Enable macro expansion MACX r2,r1 ; Call macro, show expansion
.MACRO	Початок макророзширення	.MACRO SUBI16 ; Start macro definition subi @1,low(@0) ; Subtract low byte sbci @2,high(@0) ; Subtract high byte .ENDMACRO ; End macro definition .CSEG ; Start code segment SUBI16 0x1234,r16,r17 ; Sub.0x1234 from r17:r16
.MESSAGE	Вивести повідомлення	.IFDEF DEBUG .MESSAGE "Debug mode" .ENDIF
.NOLIST	Відключити генерацію роздруку програми	.NOLIST ; Disable listfile generation .INCLUDE "macro.inc" ; The included files will not .INCLUDE "const.def" ; be shown in the listfile .LIST ; Reenable listfile generation
.ORG	Встановити адрес початку сегменту (програми, даних, констант)	.DSEG ; Start data segment .ORG 0x120; Set SRAM address to hex 120 variable: .BYTE 1 ; Reserve a byte at SRAM adr. 0x120 .CSEG .ORG 0x10 ; Set Program Counter to hex 10 mov r0,r1 ; Do something
.OVERLAP, .NOOVERLAP	Задати перекриття секції	.overlap .org 0 ; section #1 rjmp default .nooverlap .org 0 ; section #2 rjmp RESET ; No error given here .org 0 ; section #3 rjmp RESET ; Error here because overlap with #2
.SET	Присвоїти значення символьній позначці. Така позначка може бути змінена або перевизначена	.set a=0x1a .set a=a+1
.UNDEF	Відмінити символьне ім’я присвоєне регістру	.DEF var1 = R16 ldi var1, 0x20 ... ; do something more with var1 .UNDEF var1 .DEF var2 = R16 ; R16 can now be reused without warning.
.WARNING	Вивести стрічку повідомлення	.IFDEF EXPERIMENTAL_FEATURE .WARNING "This is not properly

		tested, use at own risk." .ENDIF

1.1. Директива BYTE

Директива BYTE резервує байти в пам’яті даних SRAM або енергонезалежній пам’яті EEPROM. Для того, щоб можна було зіслатися на зарезервовану пам'ять, необхідно задати позначку перед директивою BYTE. Директива має один параметр, який задає число зарезервованих байтів. Директива не може бути використана в сегменті коду CSEG. Виділений байт не ініціалізується, а за замовчуванням має значення 0xFF.

Синтаксис:

```
LABEL: .BYTE expression
```

Приклад**:**

```
.DSEG
var1:    .BYTE 1            ; зарезервувати 1 байт для var1
table:   .BYTE tab_size     ; зарезервувати tab_size байтів
.CSEG
         ldi r30,low(var1)  ; завантажити у регістр Z, low
         ldi r31,high(var1) ; завантажити у регістр Z, high
         ld r1,Z            ; завантажити var1 у регістр r1
```

1.2. Директива CSEG

Директива CSEG визначає початок сегмента коду. Файл асемблера може складатися з декількох сегментів коду, які об’єднуються в один при асемблюванні. За замовчуванням тип сегменту Code. Сегмент коду має свій власний лічильник адрес розміром слово (2 байти). Директива ORG може бути використана для розміщення коду і констант у заданій адресі пам'яті програм. Директива не має параметрів.

Синтаксис:

```
.CSEG
```

Приклад**:**

```
.DSEG              ; початок сегменту даних
vartab: .BYTE 4    ; резервування 4 байт у SRAM
.CSEG              ; початок сегменту коду
const:  .DW 2      ; запис 0x0002 у програмну пам’ять
        mov r1,r0  ; якась робота
```

1.3. Директива CSEGSIZE

Пристрої ПЛІС AT94K підтримують можливість конфігурації користувачем розмірів пам’яті програм і даних. Програма і дані в SRAM пам’яті діляться на три блоки: 10K×16 для програм SRAM, фіксований 4K×8 для даних і 6K×16 або 12K×8 для конфігурації, який можна розділити між областями пам'яті і програми сегментами розміром 2K×16 або 4K×8.

Директива CSEGSIZE вказує розмір блоку пам'яті програм.

Синтаксис:

```
.CSEGSIZE = 10 | 12 | 14 | 16
```

Приклад**:**

```
.CSEGSIZE = 12     ; Задання розміру пам’яті програм 12K×16
```

1.4. Директиви DB, DW, DD, DQ

Директива DB резервує константи байти в пам'яті програм FLASH або пам'яті EEPROM. Для того, щоб послатися на виділені ресурси перед директивою потрібно задати позначку. Директива DB сприймає один або декілька виразів. Директива DB має бути розміщена в сегменті коду або сегменті EEPROM.

Кожний вираз має оцінюватися як число між -128 і 255. Якщо вираз оцінюється як негативне число, то воно записується у пам'ять як число з доповненням до 2.

Якщо директива DB задана в сегменті коду і містить більш як один вираз, то вирази пакуються по два байти у кожне слово пам’яті програм. Невикористані половини слова пам’яті програм заповнюються нулями.

Синтаксис:

```
LABEL:  .DB список виразів
```

Приклад:

```
.CSEG
consts: .DB 0, 255, 0b01010101, -128, 0xaa
.ESEG
const2: .DB 1,2,3
```

Директиви DW, DD, DQ подібні до директиви DB, але резервують під константи відповідно слово, подвійне слово і почотирне слово в пам'яті програм або пам'яті EEPROM.

Синтаксис:

```
LABEL:  .DW список виразів
LABEL:  .DD список виразів
LABEL:  .DQ список виразів
```

Приклад:

```
.CSEG
varlist:  .DW 0, 0xffff, 0b1001110001010101, -32768, 65535
.ESEG
eevarlst: .DW 0,0xffff,10
```

```
.CSEG
varlist:  .DD 0, 0xfadebabe, -2147483648,  1 << 30
.ESEG
eevarlst: .DQ 0,0xfadebabedeadbeef, 1 << 62
```

1.5. Директива DEF

Директива DEF дозволяє посилатися на регістри через символьні імена. Регістру можна присвоїти декілька символьних імен. Символьні імена можуть бути перевизначені пізніше в програмі.

Синтаксис:

```
.DEF Symbol=Register
```

Приклад:

```
.DEF temp=R16
.DEF ior=R0
.CSEG
```

```
ldi temp,0xf0  ; завантажити 0xf0 у регістр temp
in ior,0x3f    ; прочитати SREG у регістр ior
eor temp,ior   ; Виключне АБО над регістрами temp і ior
```

1.6. Директива DEVICE

Директива DEF визначає пристрій на якому буде виконуватися код.

Синтаксис:

```
.DEVICE <device code>
```

Директива застаріла і замінена новою директивою `#pragma AVRPART PART_NAME`.

1.7. Директива DSEG

Директива DSEG визначає початок сегменту даних. Файл асемблерної програми може містити декілька сегментів даних, які об'єднуються в один сегмент при асемблюванні. Звичайно сегмент даних складається тільки з директив BYTE з позначками. Сегменти даних мають свій власний лічильник адрес, який має розмір байт. Директива ORG може бути використана, щоб розмістити змінні в специфічних областях пам'яті SRAM. Директива DSEG не має параметрів.

Синтаксис:

```
.DSEG
```

Приклад:

```
.DSEG                       ; початок сегменту даних
var1:   .BYTE 1             ; резервування 1 байту для var1
table:  .BYTE tab_size      ; резервування tab_size байт
.CSEG
        ldi r30,low(var1)   ; завантаження регістра Z (low)
        ldi r31,high(var1)  ; завантаження регістра Z (high)
        ld r1,Z             ; завантаження var1 у регістр 1
```

1.8. Директиви ELIF, ELSE

ELIF включатиме код до відповідного ENDIF, якщо вираз має значення істина. ELSE включатиме код до відповідного ENDIF, якщо початковий вираз IF вираз має значення фальш.

Синтаксис:

```
.ELIF<expression>
.ELSE
.IFDEF <symbol> |.IFNDEF <symbol>
...
.ELSE | .ELIF<expression>
...
.ENDIF
```

Приклад:

```
.IFDEF DEBUG
.MESSAGE "Debugging.."
.ELSE
.MESSAGE "Release.."
.ENDIF
```

1.9. Директива ENDIF

Директива ENDIF визначає кінець директив IF, IFDEF або IFNDEF.

Синтаксис:

```
.ENDIF
.IFDEF <symbol> |.IFNDEF <symbol>
...
.ELSE | .ELIF<expression>
...
.ENDIF
```

Приклад:

```
.IFNDEF DEBUG
.MESSAGE "Release.."
.ELSE
.MESSAGE "Debugging.."
.ENDIF
```

1.10. Директиви ENDM, ENDMACRO

Директива ENDMACRO визначає кінець макровизначення. Директива не має параметрів. ENDM – альтернативна форма, повністю еквівалентна з ENDMACRO.

Синтаксис:

```
.ENDMACRO
.ENDM
```

Приклад:

```
.MACRO SUBI16              ; початок macro визначення
       subi r16,low(@0)    ; відняти молодший байт
       sbci r17,high(@0)   ; відняти старший байт
.ENDMACRO
```

1.11. Директива EQU

Директива EQU присвоює позначці значення. Ця позначка може потім бути використана в наступних виразах. Позначка із присвоєним значенням директивою EQU, є константою і не може бути зміненою або перевизначеною.

Синтаксис:

```
.EQU label = expression
```

Приклад:

```
.EQU io_offset = 0x23
.EQU porta     = io_offset + 2
.CSEG                  ; початок сегменту коду
        clr r2         ; чистка регістра 2
        out porta,r2   ; запису у порт A
```

1.12. Директива ESEG

Директива ESEG визначає початок сегменту EEPROM. Файл з асемблерною програмою може містити декілька сегментів EEPROM, які об'єднується в один сегмент при асемблюванні. Сегмент EEPROM звичайно складається тільки з директив DB і DW (з позначками). Сегменти EEPROM мають свій власний лічильник адрес, який має розмір байт. Директива ORG може бути використана, щоб розмістити змінні у специфічній області EEPROM. Директива ESEG не має параметрів.

Синтаксис:

```
.ESEG
```

Приклад:

```
.DSEG                    ; початок сегменту даних
var1:   .BYTE 1          ; резервування 1 байта для var1
table:  .BYTE tab_size   ; резервування tab_size байтів
.ESEG
eevar1: .DW 0xffff       ; ініціалізація 1 слова у EEPROM
```

1.13. Директива EXIT

Директива EXIT вказує асемблеру зупинити асемблювання файлу, звичайно, асемблер виконує виконується до символу кінця файлу (EOF). Якщо директива EXIT з'являється у підключеному файлі, то асемблер продовжить роботу з наступного рядка після директиви INCLUDE.

Синтаксис:

```
.EXIT
```

Приклад:

```
.EXIT  ; вихід з файлу
```

1.14. Директиви IF, IFDEF, IFNDEF

Директива IFDEF із визначеним символом буде включати код до відповідної директиви ELSE. Символ має бути визначений директивою EQU або SET. Директива IF включатиме код, якщо вираз оцінено як не 0. Код включається до відповідної директиви ELSE або ENDIF.
Можливе вкладення до 5 рівнів.

Синтаксис:

```
.IFDEF <symbol>
.IFNDEF <symbol>
.IF <expression>
.IFDEF <symbol> |.IFNDEF <symbol>
...
.ELSE | .ELIF<expression>
...
.ENDIF
```

Приклад:

```
.MACRO SET_BAT
.IF @0>0x3F
.MESSAGE "Address larger than 0x3f"
lds @2, @0
```

```
sbr @2, (1<<@1)
sts @0, @2
.ELSE
.MESSAGE "Address less or equal 0x3f"
.ENDIF
.ENDMACRO
```

1.15. Директива INCLUDE

Директива INCLUDE вказує асемблеру почати читання з вказаного файлу. Асемблер потім обробляє вказаний файл до кінці файлу (EOF) або до директиви EXIT. Файл, що підключається, також може безпосередньо містити директиви INCLUDE.

Синтаксис:

```
.INCLUDE "filename"
```

Приклад:

```
; iodefs.asm:
.EQU sreg   = 0x3f     ; регістр SREG
.EQU sphigh = 0x3e     ; вказівник стеку (high)
.EQU splow  = 0x3d     ; вказівник стеку (low)
; incdemo.asm
.INCLUDE iodefs.asm    ; підключення визначень I/O
        in r0,sreg     ; читання регістру SREG
```

1.16. Директива LIST

Директива LIST вказує асемблеру включити друк файлу. Асемблер друкує файл, який містить початковий код, адреси і коди операцій. Генерація такого файлу включена за замовчуванням. Директива може також бути використана разом з директивою NOLIST для того, щоб генерувати для друку вибрані частини асемблерного коду.

Синтаксис:

.LIST

Приклад:

```
.NOLIST                ; заборона друку файлу
.INCLUDE "macro.inc"   ; файл, що підключається
.INCLUDE "const.def"   ; не буде роздруковуватися
.LIST                  ; відновлення друку
```

1.17. Директива LISTMAC

Директива LISTMAC вказує асемблеру, що, коли викликається макровизначення, то його розширення має виводитися у файл друку. За замовчуванням виводиться у файл друку тільки виклик макровизначення з параметрами.

Синтаксис**:**

```
.LISTMAC
```

Приклад:

```
.MACRO MACX         ; визначення макро
        add  r0,@0  ; деяке виконання
```

```
        eor  r1,@1  ; деяке виконання
.ENDMACRO           ; кінець макровизначення
.LISTMAC            ; дозвіл макророзширення
        MACX r2,r1  ; виклик макро, виведення розширення
```

1.18. Директива MACRO

Директива MACRO вказує асемблеру на початок макровизначення. Директива MACRO сприймає ім’я_макро як параметр. Коли ім'я макровизначення буде записане в програмі, макровизначення розширюється за місцем використання. Директива MACRO може сприймати до 10 параметрів. Ці параметри називаються позначаються як `@0-@9` в межах макровизначення. При виклику директиви MACRO параметри вказуються через кому. Макровизначення закінчується директивою ENDMACRO.

Для виведення макровизначення у файл друку використовується директива LISTMAC. Макровизначення у файлі друку позначається символом +.

Синтаксис:

```
.MACRO macroname
```

Приклад:

```
.MACRO SUBI16                   ; початок макровизначення
        subi @1,low(@0)         ; відняти байт (low)
        sbci @2,high(@0)        ; відняти байт (high)
.ENDMACRO                       ; кінець макровизначення
.CSEG                           ; початок сегменту коду
        SUBI16 0x1234,r16,r17   ; Sub.0x1234 from r17:r16
```

1.19. Директива MESSAGE

Директива MESSAGE виводить рядок. Використовується при асемблюванні з умовами.

Синтаксис:

```
.MESSAGE "<string>"
```

Приклад:

```
.IFDEF DEBUG
.MESSAGE "Debug mode"
.ENDIF
```

1.20. Директива NOLIST

Директива NOLIST вказує асемблеру відключити виведення файлу друку.

Синтаксис:

```
.NOLIST
```

Приклад:

```
.NOLIST                 ; Заборонити генерацію файлу друку
.INCLUDE "macro.inc"    ; Підключені файли
.INCLUDE "const.def"    ; не будуть виводитися у файл друку
.LIST                   ; дозвіл на генерацію файлу друку
```

1.21. Директива ORG

Директива ORG встановлює значення лічильника адрес в абсолютне значення. Значення для встановлення задається як параметр. Якщо директива ORG знаходиться в межах сегменту даних, то вона задає значення SRAM лічильника адрес SRAM, якщо знаходиться в межах сегменту коду, то задає значення лічильника адрес в програмній пам'яті, якщо знаходиться в межах сегменту EEPROM, то задає значення лічильника адрес EEPROM.

Синтаксис:

```
.ORG expression
```

Приклад:

```
.DSEG                ; початок сегменту даних
.ORG 0x120           ; встановити SRAM адресу в hex 120
variable: .BYTE 1    ; зарезервувати byte в SRAM з адресою 0x120
.CSEG
.ORG 0x10            ; встановити лічильник програми у hex 10
          mov r0,r1  ; якась дія
```

1.22. Директиви OVERLAP/NOOVERLAP

Ці директиви впливають тільки на поточний активний сегмент (`.cseg/.dseg/.eseg`). Директиви `overlap/nooverlap` позначають секцію, якій дозволяється перекрити сегмент код/дані сегментом код/дані, визначеними у іншому місці, без виведення повідомлень про помилки або попередження. Таке перекриття не залежить від значень директиви `#pragma overlap`. Атрибути `overlap/nooverlap` залишаються в дії для всієї директиви ORG, але не поширюються на всі сегменти `.cseg/.dseg/.eseg` разом (для кожного сегменту директиви потрібно задати окремо).

Синтаксис:

```
.OVERLAP
.NOOVERLAP
```

Приклад:

```
.overlap
.org 0                  ; секція #1
    rjmp  default
.nooverlap
.org 0                  ; секція #2
    rjmp  reset         ; немає помилки
.org 0                  ; секція #3
    rjmp  reset         ; помилка, так як є перекриття з #2
```

Типове використання цих директив, це встановлення за замовчуванням деякої форми коду або даних, які можуть або не можуть бути пізніше модифіковані шляхом перекриття коду або даних, без необхідності блокувати виявлення перекриття асемблера в цілому.

1.23. Директива SET

Директива SET назначає значення позначці. Ця позначка може бути потім використана в наступних виразах. На відміну від директиви EQU, позначці може бути назначено нове значення в програмі.

Синтаксис:

```
.SET label = expression
```

Приклад:

```
.SET FOO = 0x114     ; назначити FOO адресу комірки SRAM
        lds r0, FOO  ; завантажити адресу в r0
.SET FOO = FOO + 1   ; назначити нове значення (перевизначити) FOO
        lds r1, FOO  ; завантажити нову адресу в r1
```

1.24. Директива UNDEF

Директива UNDEF використовується для відміни значень символів, що були присвоєні регістрам директивою DEF. Це дозволяє створювати прості області видимості регістрів.

Синтаксис:

```
.UNDEF symbol
```

Приклад:

```
.DEF var1 = R16
ldi var1, 0x20
... ; якась дія
.UNDEF var1
.DEF var2 = R16 ;R16 тепер може повторно використовуватися без повідомлень
```

1.25. Директива WARNING

Директива .WARNING виводить рядок повідомлень, але на відміну від директиви ERROR не зупиняє асемблювання. Директива використовується при умовному асемблюванні.

Синтаксис:

```
.WARNING"<string>"
```

Приклад:

```
.IFDEF EXPERIMENTAL_FEATURE
.WARNING "Ця гілка коду недостатньо протестована."
.ENDIF
```

2. Директиви і структура асемблерної програми

Асемблерна програма повинна мати чітку структурну організацію, щоб не вносити затримки в основний програмний цикл і забезпечувати швидкодію, а також бути зручною для читання і розширення коду.

МК Atmel мають три види пам’яті: оперативну пам’ять (SRAM), пам’ять програм (FLASH) та енергонезалежну пам'ять даних (EEPROM), і відповідно, в програмі передбачена можливість розмітки цих областей за допомогою директив та при необхідності – ініціалізації їх значеннями.

```
; ==== SRAM – сегмент даних ===============
.DSEG
; ===== FLASH – сегмент програмного коду
.CSEG
; ===== EEPROM – сегмент для енергонезалежних даних
.ESEG
```

.DSEG (Data segment) – визначає початок сегменту даних. У цьому сегменті розмічається область оперативної пам’яті SRAM за допомогою директиви `.byte` (та позначок). Тобто, резервується місце для даних в оперативній пам’яті з використанням позначок, адреси яких встановлює компілятор.

.CSEG (Code segment) – визначає початок сегменту коду, де розміщується основна виконавча програма.

.ESEG (EEPROM segment) – визначає початок сегменту енергонезалежних даних. У цій області можна розмітити пам’ять як для розміщення конкретних даних, так і резервування простору для програмного збереження даних.

Оскільки кожна модель МК AVR має специфічний для неї набір периферійних пристроїв, то відповідно адреси регістрів, що їм відповідають, відрізняються у кожній моделі. Для того, щоб не працювати з адресами регістрів, а тільки з їхніми іменами (визначеними у середовищі Atmel Studio), усі відповідності між іменами та адресами регістрів винесені в окремі бібліотечні файли. Тому для програми на мові асемблер середовище Microchip Studio само підключає за допомогою директиви `.include` необхідний бібліотечний файл для вибраної в проекті моделі МК, наприклад "`m32Adef.inc`".

Приклад можливої структурної організації програми:

```
     .include "m32Adef.inc"    ; підключення бібліотечного файлу МК
     ; ===== макроси ======
     .macro Clear
        clr @0
     .enfmacro
     ;===== директиви ====
     def _temp = r16
     equ Fig_0 = -64
     ; ==== SRAM – сегмент даних (оперативна пам'ять ) ===============
     .DSEG
     .org $000 ;встановлює адресу SRAM у значення 0x000
     value:  .byte 1  ;резервує 1 байт для позначки value на початку SRAM
     vector: .byte 3  ;резервує 3 байти для позначки vector за адресою SRAM+1
     table:  .byte 10 ;резервує 3 байти для позначки table за адресою SRAM+4
     ; ===== FLASH – сегмент програмного коду ====
     .CSEG
     ; ====== вектор переривань ====
     .org  $000  ;встановлює програмний лічильник у значення 0x000ж
     ;…
     jmp reset
     ; ====== підпрограми переривань ====
     .org $028
     reti
reset:
     ; ======= ініціалізація пам’яті  ====
     v1: .db 0, 255, 0b01010101, -125, 0xb3
     v2: .db "Hello my dear friend"
     v3: .dw  -30125, 0xa0b5, 0b1010101010101010, 56, 65535
     ; ===== ініціалізація стеку
     ldi r16, Low(RamEnd)  ; налаштування стеку
     out SPL,r16
     ldi r16, High(RamEnd)
     out SPH,r16
     ; ===== ініціалізація внутрішньої периферії
```

```
      ldi r16, 0x00 ; ініціалізація регістрів загального призначення
      ldi r17, 0xFF
      out PortA,r16 ; порт A – на вихід
      out DDRA, r17
      out PortB,r17 ; порт B – на вхід з підтягуючим резистором
      out DDRB, r16
      ; ===== ініціалізація зовнішньої периферії
      ;…
      ; ===== ініціалізація дозволів на переривання
      ;…
main:
      ; ==== основний програмний код (в циклі) ====
      rjmap main
      ; === підпрограми =====
      ; === табличні дані
      Table: .db 15,25,35,45
      ; …
      ; ===== EEPROM – сегмент для енергонезалежних даних ======
      .ESEG
EEinit: .db 1,2,3,4
```

При старті програми МК, а особливо при її рестарті, регістри загального призначення та оперативна пам'ять не завжди обнулені. Часто там залишаються значення з попереднього запуску. Тому у програмі необхідно всі комірки пам'яті як оперативної, так і регістри загального призначення або встановити в необхідне значення, або обнулити. Ну хоча б проініціалізувати ті комірки пам'яті, з якими планується працювати.

Директива `.byte` – резервує вказану кількість байтів пам'яті в областях SRAM та EEPROM.

```
      .DSEG
      value:   .byte  1    ;резервує 1 байт для позначки value
      vector:  .byte  3    ;резервує 3 байти для позначки vector
      table:   .byte 10    ;резервує 10 байтів для позначки table
```

За замовчуванням при старті МК значення стеку має мати значення кінця SRAM. Але все одно краще проініціалізувати його на початку програми, і бути впевненим, що стек розміщений у визначеному місці.

Ініціалізація *внутрішньої периферії* передбачає налаштування роботи різних таймерів, інтерфейсів, портів введення/виведення і т. п.

Ініціалізація *зовнішньої периферії* передбачає налаштування роботи дисплеїв, зовнішньої пам'яті, ініціалізацію різних пристроїв, давачів, усього, що підключено ззовні до МК.

Коли основна ініціалізація виконана, тоді даються дозволи на переривання від периферії МК та загальний дозвіл (прапорець I регістру стану SREG). Після цього керування передається основному циклу програми.

Структурно підпрограми переривань розміщуються одразу після вектора переривань, а звичайні підпрограми після блоку основного циклу.

В кінці сегменту FLASH розміщуються таблиці з константами.

У наведеному прикладі програмний код (після директиви .CSEG) починається записом вектора визначених переривань. Перше переривання, скидання МК, здійснює стрибок на програмну позначку `reset`, з якої починається ініціалізація наявної периферії та закінчується основним програмним циклом – програмною петлею на позначку `main`.

Навіщо зациклювати основну програму? Якщо не буде в кінці коду програми переходу на початок позначки `main`, тоді програма буде виконуватися далі, до кінця FLASH (а вона в деяких моделей може бути достатньо великою, наприклад ATmega32A має 32 Кб). Пусті області програмної пам' яті заповнені значеннями FF, і відповідно МК пройдеться по них до кінця обсягу FLASH, ніби виконуючи пустий оператор, а потім продовжить виконання з початку програми, тобто стрибок на позначку `reset`, ініціалізація периферії і т.п.

Якщо програма є зациклена, то одразу після команди переходу на позначку `main` можна розміщувати необхідні підпрограми, до яких буде звернення з основного програмного циклу.

Читання/записування даних з регістрів периферії (SRAM, PORTA, PORTB, PORTC, PORD, EEPROM, TIMER0, TIMER1, ADC, TWI, USART) *можна здійснювати лише через посередництво регістрів загального призначення R0-R31*. Для того щоб записати значення у якийсь з периферійних регістрів, спочатку присвоюється це значення якомусь з регістрів загального призначення, а потім з цього регістра пересилається в периферійний регістр.

За замовчуванням пам'ять даних ініціалізована значенням FF. Для збереження табличних константних значень, тобто таких, які не змінюються в процесі виконання програми, наприклад таблиця значень синусів та косинусів, може використовувати як пам'ять програм FLASH, так і пам'ять EEPROM. Розміщення масивів даних в цих сегментах здійснюється за допомогою таких директив:

`.db` (define constant byte(s)) – вказує на розміщення масиву байтів у пам'яті програм FLASH або EEPROM. Кожне значення (вираз) масиву може приймати значення від -128 до 255. Якщо значення задане від'ємним числом, тоді воно буде розміщене як 8-бітне число у доповняльному двійковому коді. Масив повинен містити щонайменше одне значення, два і більше значень відокремлюються між собою комами. Якщо директива `.db` вказана у сегменті .CSEG та масив містить більше ніж одне значення, тоді значення пакуються по два байти у кожне слово пам'яті програм. Якщо масив містить непарне число значень, тоді останнє значення буде розміщене у своєму власному слові програмної пам'яті, а невикористана половина програмного слова буде встановлена у нуль.

`.dw` (define constant word(s)) – вказує на розміщення масиву слів (2 байтових значень) у пам'яті програм FLASH або EEPROM. Кожний елемент масиву може приймати значення від -32768 до 65535. Якщо значення виражене від'ємним числом, тоді воно буде розміщене як 16-бітове число у доповняльному двійковому коді.

`.dd` (define constant doubleword(s)) та `.dq` (define constant quadword(s)) – ці директиви подібні до попередніх та використовуються для резервування масивів 32- та 64-бітових значень.

`.org` (set program origin) – ця директива встановлює абсолютне значення адреси для комірки пам'яті. Може використовуватися у всіх трьох сегментах пам'яті: даних, програми та EEPROM. Є певні особливості щодо її використання. Якщо директива розміщується у сегментах SRAM та EEPROM, то адресація даних побайтова, а якщо розміщується у сегменті програми – то адресація послівна (по 2 байти). Ще один момент стосується пам'яті даних SRAM. Якщо вказати директиву з параметром 0, то буде адресуватися область, в якій розміщені регістри загального призначення. Тому область даних потрібно адресувати із значення 0x60 чи навіть зі 0x100 (для МК з розширеною областю пам'яті для регістрів введення/виведення). У програмах ці директиви в секції коду задають область яку займає вектор переривань. Вони там необхідні, оскільки кожне переривання здійснює перехід на чітко встановлену адресу в межах цього вектора, що розміщений на початку області програмного коду.

```
.DSEG
```

```
     .org $150 ;встановлює адресу SRAM у значення 0x150
var: .byte 1   ;резервує 1 байт у SRAM за адресою 0x150
     .CSEG
     .org $40  ;встановлює програмний лічильник у значення 0x40
      inc r0   ;виконується якась робота
```

.def – директива дозволяє встановити символьні псевдоніми для регістрів загального призначення. В процесі написання програми для роботи із змінними вибираються певні регістри загального призначення. Наприклад, r0 – секунди годинника, r1 – хвилини, r2 – години, r16 – тимчасові проміжні значення і т. п. Тому на початку програми потрібно визначати таблицю відповідних символьних псевдонімів для регістрів загального призначення, які будуть використовуватися у програмі.

```
.def _second = r0
.def _minute = r1
.def _hour   = r2
.def _temp   = r16
.def _temp2  = r17
.CSEG
ldi _temp,25       ;встановлює r16 у значення 25
mov _minute,_temp  ;пересилає значення з r16 у r1
```

.equ – директива закріплює за символьними позначками константні числові значення або вирази, які в процесі компіляції будуть підмінені у програмі замість позначок. Закріплені за позначками значення потім поміняти не можна.

```
.equ XTAL= 8000000
.equ BaudRate = 9600
.equ BaudDiv = XTAL / (16 * BaudRate) - 1
```

.set – директива, як і .equ, закріплює за позначкою числове значення, але дозволяє переназначати значення для цієї позначки по ходу виконання програми.

```
     .set Size = 5
Loop:
     ldi r16, Size  ; r16 = Size = 5
     .set Size = 10
     ldi r16, Size ; r16 = Size = 10
     .set Size = Size + 10
     ldi r16, Size ; r16 = Size = 20
     rjmp Loop
```

У наведеному прикладі директива .set розбиває програму на три підсегменти, і в кожному з них позначка Size має своє визначене значення. Навіть у циклі, де Size приймає нове значення, при переході на початок циклу там буде діяти інше значення для Size, яке визначене ще перед початком циклу. Тому такі маніпуляції з директивою .set слід застосовувати у програмі уважно.

3. Завантаження даних у регістри

Після розбивки програми на сегменти і виділення і ініціалізації даних, виникає необхідність завантажувати дані у регістри загального призначення. Дані у регістри загального призначення можна завантажувати наступними способами:

• пряме завантаження:

```
  LDS R2, $DF     ;запис в R2 вмісту комірки $00DF (регістри r0-r31)
```

• завантаження константи безпосереднє (immediate):

```
  LDI R16, $23    ;запис в R6 числа $23 (регістри r16-r31)
```

• завантаження з програмної пам’яті (регістри X, Y, Z):

```
  CLR R31
  LDI R30, $F0   ;записати в регістрову пару Z (R31:R30) адресу $00F0
  LPM            ;записати вміст комірки пам’яті з адресою $F0 в R0
```

• непряме завантаження з після інкрементом та перед декрементом:

```
  CLR R27
  LDI R26, $20 ;записати в регістрову пару X (R27:R26) адресу $0020
  LD R0, X+ ;завантажити в R0 вміст SRAM за адресою $20 (X+1 після інкремент)
  LD R1, X ;завантажити в R1 вміст за адресою $21
  LDI R26, $23 ;записати в регістрову пару X адресу $0023
  LD  R3,  -X  ;завантажити  в  R3  вміст  SRAM  за  адресою  $22  (X-1  ;перед
декремент)
```

4. Створення і використання макросів

В асемблерних програмах крім підпрограм використовуються макроси. Для створення макросів передбачені директиви початку `.macro` та кінця макросу `.endmacro` або `.endm`. На відміну від виклику підпрограм, де здійснюється перехід за позначкою та повернення у вихідну точку, виклик макросу означає, що компілятор виконає вставку коду макросу у точку виклику. Скільки разів макрос буде викликаний, на стільки ж і збільшиться обсяг вихідного програмного коду. Приклад макросу, який додає значення двох регістрів і повертає результат у третьому регістрі:

```
.macro  Addition
  clr  @0
  add  @0, @1
  add  @0, @2
.endmacro

  .CSEG
  ldi    r18, 5  ; r18 = 5
  ldi    r19, 8  ; r19 = 8
  Addition  r10, r18, r19  ; r10 = r18+r19 = 13
```

Макрос може приймати до 10 параметрів. Посилання на ці параметри позначаються в середині макросу як `@0-@9`. Порядок слідування визначається при виклику макросу. У наведеному прикладі `@0` це `r10`, `@1 – r18`, а `@2 – r19`. Під час компіляції при підставленні коду макросу у точки виклику замість параметризованих змінних підставляються аргументи, вказані через кому після імені викликаного макросу.

Висновки.

Директиви асемблера не транслюються у машинні коди, а використовуються для налаштування розміщення програми у пам’яті, визначення макросів, ініціалізації пам’яті і т. п. Макроси заміняють невеликі і часто повторювані фрагменти коду, що дозволяє зменшити розмір асемблерної програми. Під час компіляції код макросу підставляється у точку виклику.

Питання.

1. Директиви асемблера BYTE, DB, DW, DD, DQ.
2. Директиви асемблера CSEG, DSEG, ESEG, CSEGSIZE.
3. Директиви асемблера DEF, UNDEF, EQU, SET.
4. Директиви асемблера IF, IFDEF, IFNDEF, ELIF, ELSE, ENDIF.
5. Директиви асемблера LIST, NOLIST, LISTMAC.
6. Директиви асемблера MACRO, ENDM, ENDMACRO.
7. Директиви асемблера WARNING, MESSAGE, EXIT.
7. Використання директив для структурування асемблерної програми.
8. Способи завантаження даних у регістри.
9. Макроси і їх використання.

ЛЕКЦІЯ 5. Паралельні порти введення/виведення мікроконтролера

Мета. Вивчення структури та техніки програмування паралельних портів введення/виведення мікроконтролера.

Вступ. Всі порти введення/виведення базової серії МК AVR є 8-розрядними і двонаправленими. Кожний вивід порту може бути індивідуально конфігурований як вхід або вихід. При функціонуванні виводу як вхід до нього може бути підключений підтягуючий резистор. Для керування виводами портів призначені три регістри: регістр даних, регістр напрямку передачі даних і регістр виводів.

План.

1. З’єднання виводів портів введення/виведення
2. Керування роботою портів введення/виведення
3. Підключення периферійних пристроїв до портів введення/виведення
4. Електричні характеристики системи введення/виведення
5. Варіанти підключення світлодіодів та кнопок до МК

1. З’єднання виводів портів введення/виведення

Паралельні порти введення/виведення є в усіх моделях МК AVR. Виводи портів, окрім основного призначення – побайтового чи побітового введення/виведення, можуть також використовуватися для роботи з деякою внутрішньою периферією МК (у залежності від того, які альтернативні функції закріплені за ними).

Модель ATmega8515 має чотири двонаправлені 8-розрядні порти введення/виведення A, B, C та D з внутрішніми підтягуючими резисторами. Налаштувати та керувати можна відразу як усіма 8-розрядами, так і кожним виводом порту окремо. Значення на виводах портів можна встановлювати командами SBI, CBI. Блок схема внутрішнього з’єднання виводів порту показана на рис. 5.1.

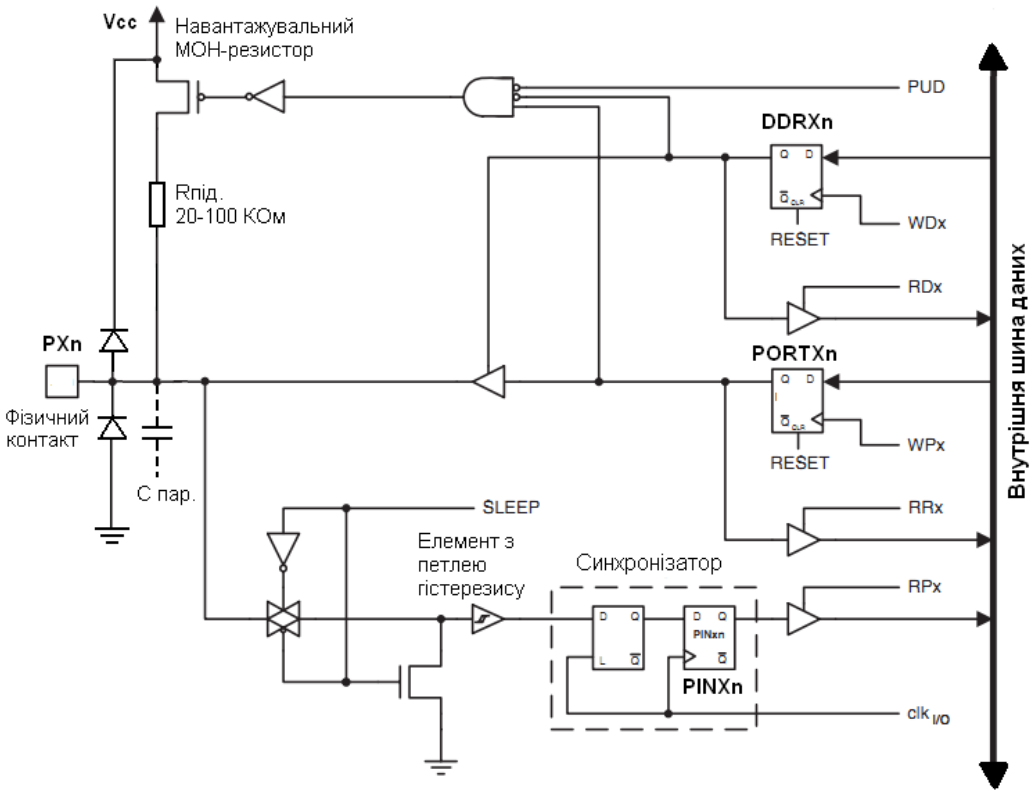


Рисунок 5.1 –Блок-схема внутрішнього з'єднання виводів порту PORTXn:
WDx, RDx – запис/читання регістру DDXn, WPx, RPx – запис/читання регістру PORTXn,
PUD – заборона підтягуючого резистора, RPx – читання регістр PINXn,
SLEEP – перехід у режим SLEEP за сигналом від МК

2. Керування роботою портів введення/виведення

Для керування виводами портів PXn призначені три регістри: регістр даних порту PORTX, регістр напрямку передачі даних DDX (DDRX), регістр PINX (літера X відповідає назві регістра A, B, C чи D, n – відповідає номеру виводу 0,…,7). Звернення до портів здійснюється за різними адресами в області пам'яті введення/виведення.

Усі виводи портів мають внутрішній діодний захист від стрибків підвищеної напруги (верхній діод відкривається при напрузі вищій за напругу живлення МК і перепад напруги обробляють фільтри блоку живлення) та від'ємної напруги (яка нейтралізується на землю нижнім діодом). Діодний захист працює лише від імпульсних завад, а при перевищенні постійної напруги на вході порт ймовірно вийде з ладу. Кожний вивід має також незначну внутрішню паразитну ємність.

Регістр напряму передачі DDRX – визначає призначення фізичного виводу порту - вхід або вихід:

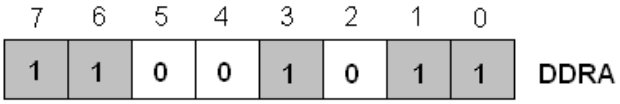
• якщо біт n в регістрі DDRXn містить лог. 0, то відповідний фізичний вивід PINXn сконфігурований як вхід (стан за замовчуванням, який встановлюється після скидування або включення живлення). У цьому випадку регістр PORTXn від'єднаний від фізичного виводу PINXn. Якщо на виході регістра PORTXn встановлена лог. 1, то до виводу PINXn підключається підтягуючий резистор.

• якщо біт n в регістрі DDRXn містить лог. 1, то відповідний фізичний вивід PINXn сконфігурований як вихід. У цьому випадку регістр PORTXn під'єднаний до виводу PINX.

Підтягуючий резистор відключений від виводу PINXn, незалежно від значення на виході PORTXn.

Регістр порту PORTX – містить значення для виведення, якщо він сконфігурований на вихід. Якшо він сконфігурований на вхід, то до фізичного вхідного виводу може бути підключений підтягуючий резистор.

Регістр DDRA:



Значення бітів PORTXn і бітів DDRXn визначають налаштування виводів PINXn. Можливі конфігурації фізичних виводів порту PINXn показані в табл. 5.1.

Таблиця 5.1 – Конфігурації виводів PINXn

DDRXn	PORTXn	PUD (SFIOR)	I/O	Підтягу-вання	PINXn
0	0	X	IN, вхід		Hi-Z
0	1	0	IN, вхід	так	Vcc
0	1	1	IN, вхід		Hi-Z
1	0	X	OUT, вихід		Низький рівень
1	1	X	OUT, вихід		Високий рівень

Якщо DDRXn=”0” і PORTXn=”0”, то вивід PINXn знаходиться у високоімпедансному стані Hi-Z (без підтягуючого регістра). У цьому стані повний опір входу PXn є дуже високим і не впливає на зовнішнє підключення. У стані Hi-Z входи виводів МК використовуються для підключення до сторонніх шин даних, не заважаючи при цьому їхній роботі. Виводи PXn будуть у високоімпедансному стані при умовах RESET, навіть без тактових сигналів.

Якщо DDRXn=”0” і в PORTXn записується “1”, тоді підключається внутрішній резистор Rпід. (input pull-up), який підтягує вивід PINXn до напруги живлення МК, рис. 1. Якщо підтягнути зовнішніми резисторами PINXn до “0”, то PINXn буде працювати як джерело струму.

Якщо заборонити підключення підтягуючих резисторів, встановивши біт PUD=”1” в регістрі спеціальних функцій SFIOR, то вивід PINXn залишиться у високоімпедансному стані.

Якщо DDRXn=”1” і PORTXn=”0”, тобто вивід PINXn працює на вихід і на ньому встановлюється низький рівень напруги (цифрова земля) та відключаються підтягуючі резистори.

Якщо DDRXn=”1” і PORTXn=”1”, тобто вивід PINXn працює на вихід і на ньому встановлюється високий рівень напруги (Vcc, напруга живлення МК) та відключаються підтягуючі резистори.

При перемиканні між станами ({DDRXn, PORTXn}=0b00) і ({DDRXn, PORTXn}=0b11) потрібно використати один з проміжних станів ({DDRXn, PORTXn}=0b01) або ({DDRXn, PORTXn}=0b10).

Аналогічно при перемиканні між станами ({DDRXn, PORTXn}=0b01) і ({DDRXn, PORTXn}=0b10) потрібно використати один з проміжних станів ({DDRXn, PORTXn}=0b00) або ({DDRXn, PORTXn}=0b11).

Регістр PINXn. Незалежно від налаштування бітів DDRXn, вивід PINXn можна прочитати через біти регістра PINXn. Для стабільності зчитування стану вивода PINXn використовується синхронізуюча ланка, що складається із тригера защіпки та розряду PINxn (рис. 1). Значення сигналу на виводі PINXn, фіксується синхронізатором при низькому рівні сигналу і потім перезаписується у розряд PINXn за наростаючим фронтом тактового сигналу. Відповідно, між операціями запису у порт та зчитування його стану є затримка.

Синхронізація сигналів, при зовнішньо прикладеному значенні PINXn показана на рис. 5.2, а при програмно прикладеному значенню – на рис. 5.3.

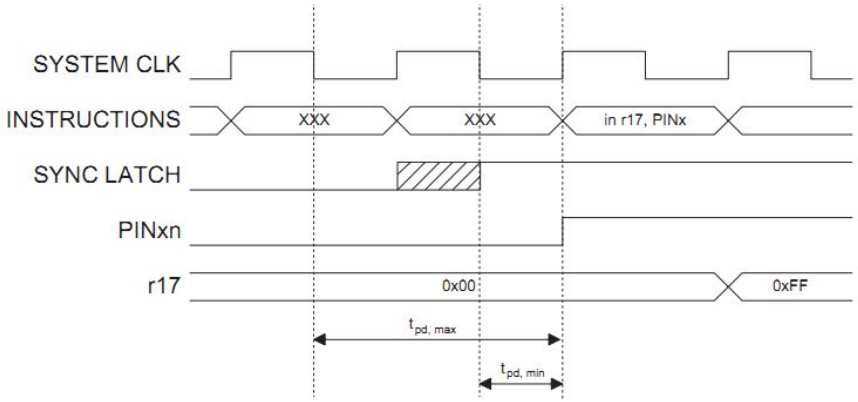


Рисунок 5.2 – Синхронізація сигналів при зовнішньо прикладеному значенню до PINXn

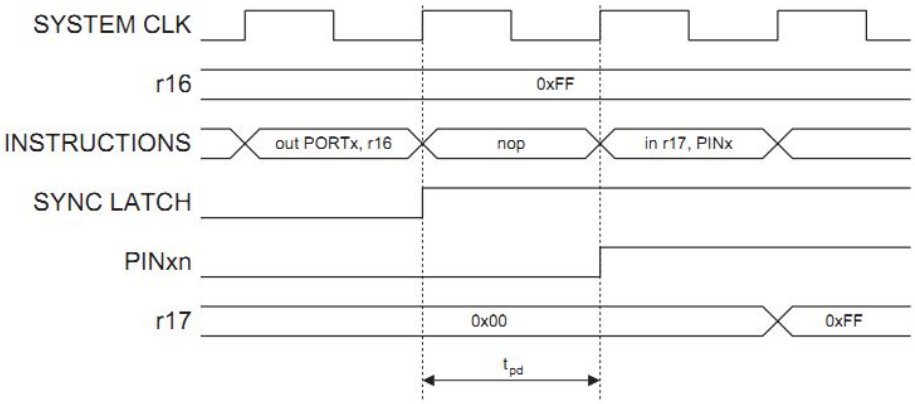


Рисунок 5.3 – Синхронізація сигналів при програмно прикладеному значенню до PINXn

При читанні програмно назначеного значення у програмі між командами `out` та `in` необхідно вставляти порожню команду `nop`.

Значення бітів регістра PINXn відповідають реальним рівням сигналів на виводах PINXn: “1” – при високому рівні, “0” – при низькому рівні. Якщо виводи налаштовані як високоімпедансні входи, та немає підключення зовнішніх сигналів, тоді значення відповідних бітів PINXn будуть рівними “0”. Однак, від найменших завад, наприклад дотик пальця до корпусу МК, на виводі може з’явитися “1”. З цієї причини не підключені входи до шин сигналів необхідно підтягувати через резистор до напруги живлення або землі. Найпростіше підтягнути вхід до напруги живлення з використанням підтягуючого резистора.

В МК AVR визначені наступні рівні сигналів:

• низький рівень -0.5 В ÷ 0.3·Vcc В;

• високий рівень 0.7·Vcc В ÷ Vcc+0.5 В;

Нижчі чи вищі рівні напруги можуть вивести з ладу виводи МК.

Перехід від низького рівня до високого відбувається при одній напрузі, а зворотній перехід при дещо нижчій напрузі. Така гістерезисна петля є своєрідним механізмом захисту від перемикання під дією завад. Розмах величини гістерезисної петлі становить 0.05·Vcc і залежить від моделі МК.

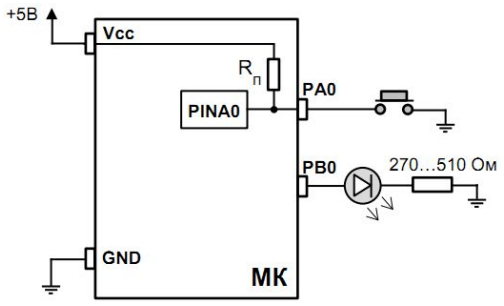
Приклад коду, в якому біти 0, 1, 6 і 7 PORTB встановлюються в “1”, а виводи DDRB0-DDRB3 встановлюються як вихідні, а DDRB4-DDRB7 – як вхідні (за замовчуванням). До виводів PINX6, PINX7 будуть підключені підтягуючі резистори.

```
; код асемблера
; біти для PORTB у стан ”1”
ldi r16,(1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0)
; напрямок - на вихід
ldi r17,(1<<DDRB3)|(1<<DDRB2)|(1<<DDRB1)|(1<<DDRB0)
out PORTB,r16   ; виведення значень бітів у регістр PORTB
out DDRB,r17    ; виведення значень бітів у регістр DDRB
; порожня інструкція для синхронізації
nop
; читання виводів PINB у регістр r16
in r16,PINB
...

; C код
unsigned char i;
...
/* біти 0,1,7,7 PORTB у стан ”1” - на вихід */
PORTB = (1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0);
/* напрямок - на вихід */
DDRB = (1<<DDRB3)|(1<<DDRB2)|(1<<DDRB1)|(1<<DDRB0);
/* порожня інструкція для синхронізації */
_NOP();
/* читання виводів PINB */
i = PINB;
...
```

3. Підключення периферійних пристроїв до портів введення/виведення

При підключенні різноманітних кнопок та клавіатур до МК для уникнення дії завад використовуються підтягуючі резистори до напруги живлення чи землі, рис. 5.4. Найпростіше налаштувати вхід на використання внутрішнього підтягуючого резистора до напруги живлення.



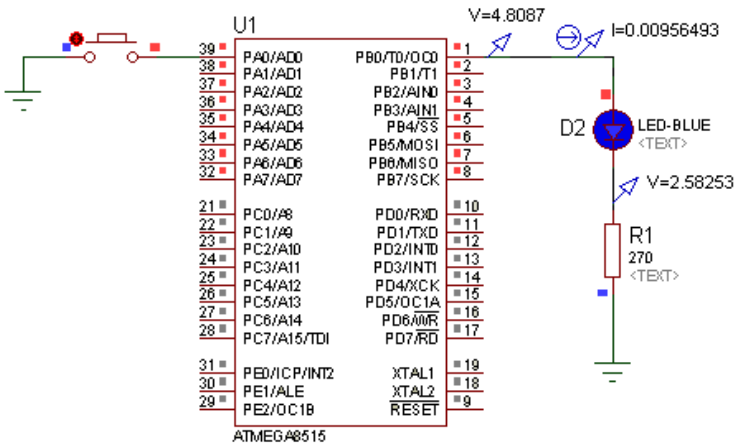


Рисунок 5.4 – Підключення кнопки та світлодіода до МК

Приклад коду з підключення кнопки та світлодіода до МК:

```
    include "m8515def.inc"
    .CSEG
    ldi r16,0x00
    ldi r17,0xFF
    ;Порт A на вхід, з підтягуючим резистором
    out DDRA,r16
    out PORTA,r17
    ;Порт B на вихід, з початковим рівнем ”0”
    out  DDRB, r17
    out  PORTB, r16

main:
    in  r16, PINA
    out  PORTB, r16
    rjmp  main
```

На початку програми налаштовуються порти, до яких будуть підключені зовнішні пристрої. Кнопка під’єднана до 0-виводу порту А, світлодіод до 0-виводу порту В. Увесь порт А налаштований на вхід і підключені внутрішні підтягуючі резистори. Порт В налаштований на вихід. В основному програмному циклі зчитується стан входів усього порта А і заноситься байт стану у регістр r16. На наступному такті програми значення регістра r16 заноситься на виходи порта В. Коли кнопка не натиснута то високий рівень із виводу PORTA0 переноситься на вивід PORTB0. Через світлодіод протікатиме струм 9.5 мА, що спричинить його світіння і спад напруги 4.8-2.6=2.2 В. Натисканням кнопки шунтується підтягуючий резистор і на вивід порта PORTA0 подається низький рівень, який передається на PORTB0 і світлодіод гасне.

4. Електричні характеристики портів введення/виведення

Максимальне абсолютне значення допустимого струму для одного виводу портів введення/виведення складає 40 мА.

Рекомендовані значення на виходах портів A, B, C, D:

- низький рівень “0”:

• 0.7 В – струм 20 мА при Vcc=5 В;

• 0.5 В – струм 10 мА при Vcc=3 В;

- високий рівень “1”:

• 4.2 В – струм -20 мА при Vcc=5 В;

• 2.2 В – струм -10 мА при Vcc=3 В;

Однак загальний струм, що протікає через Vcc і Vgrd не має перевищувати 200 мА.

Отже при підключенні пристроїв до портів введення/виведення необхідно чітко прораховувати кількість струму, що буде протікати через виводи МК та через які саме.

Оскільки виводи МК забезпечують високу навантажувальну здатність при любому рівні сигналу, то до них можна безпосередньо підключати світлодіодну індикацію.

Вольт-амперна характеристика світлодіода така ж як у звичайних діодів (рисунок 5.5).

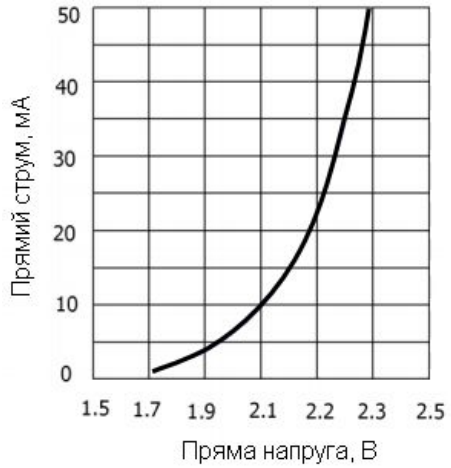


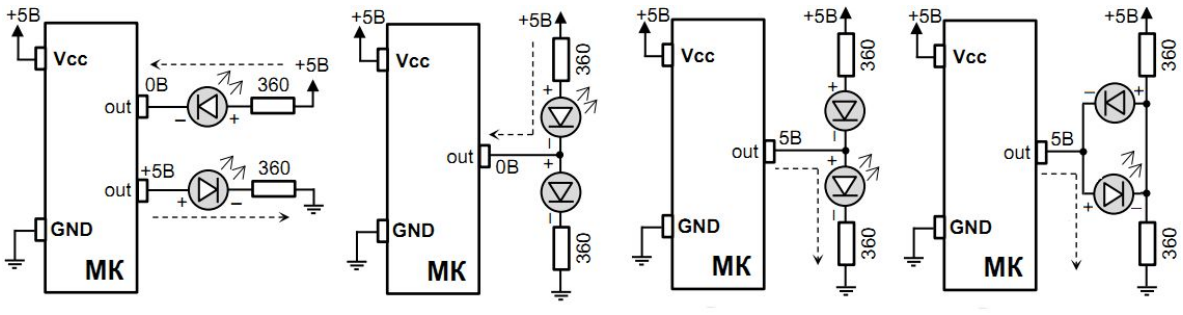
Рисунок 5.5 – Вольт-амперна характеристика світлодіода FYL-3014LURC1A (червоний)

Для уникнення виходу з ладу виводу МК та світлодіоду необхідно підключати світлодіоди через обмежувальні резистори. Опір такого резистора визначають виходячи із струму, який має споживати світлодіод. Наприклад, для вказаного світлодіоду для робочого струму I_{diod}=9.5 мА на вольт-амперній характеристиці відповідає напруга 2,1 В. Тоді опір R_{diod}=2.1/0.0095=221 Ом. З рівняння $I_{diod}\cdot(R_{diod} + R_1)$=4.8 В, визначається R_1=4.8/I_{diod} – R_{diod} = 505 – 221 = 284 Ом.

Величина резистора R1 забезпечує робочий струм світлодіоду і допустимий струм через один вивід порта I/O (до 20 мА).

5. Варіанти підключення світлодіодів та кнопок до МК

При підключенні світлодіодів необхідно чітко дотримуватися полярності: високий потенціал (+) до анода діода, низький потенціал (–) до катода. Відповідно до цього, можна по різному підключати світлодіоди: або до зовнішньої позитивної напруги та засвічувати світлодіод низьким рівнем (“0”) з виходу МК, або до зовнішньої землі та засвічувати високим рівнем (“1”) з виходу МК (рисунок 5.6, а). Вибір одного з цих двох способів підключення найчастіше зумовлюється зручністю підведення землі чи позитивної напруги до світлодіоду.



а) б) в) г)

Рисунок 5.6 – Способи підключення світлодіодів: а) керування одним світлодіодом одним виходом МК; б), в), г) керування двома світлодіодами одним виходом МК;

Для деяких задач необхідно почергово засвічувати два світлодіоди, наприклад, червоний світлодіод – канал передає дані, зелений – канал відключений. Тоді їх можна підключити разом до одного виводу МК, як на рис. 6. При подачі “0” на вихід – засвітиться верхній світлодіод (б), при подачі “1” – засвітиться нижній (в, г). Якщо цей вивід МК налаштувати як високоімпедансний вхід, тоді струм потече від додатної напруги до землі через обидва світлодіоди, і вони будуть обидва світитися, але дещо тьмяніше.

У схемі з’єднання показаній на рис. 5.7 відповідними сигналами МК можна засвітити будь-який з шести світлодіодів, а також їх комбінації.

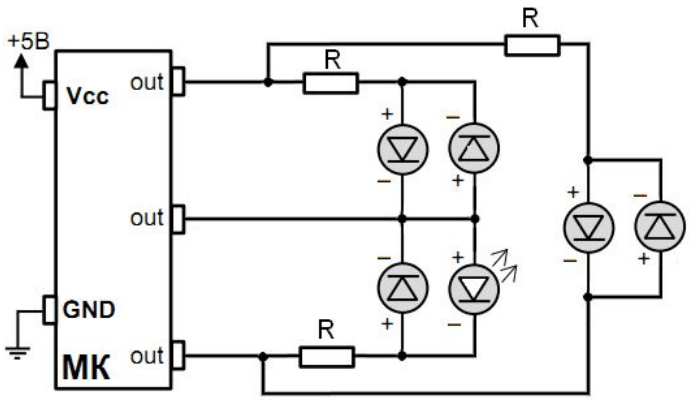


Рисунок 5.7 – Керування шістьма світлодіодами трьома виходами МК

Робота з кнопками.

Для забезпечення завадостійкості при підключенні кнопок до виводів МК необхідно використовувати зовнішній підтягуючий резистор, оскільки у деяких випадках внутрішній резистор може не справлятися із наявними завадами. При цьому вивід можна підтягувати як до напруги живлення, так і до землі. Відповідно, кнопки мають бути підключеними до землі та напруги живлення (рисунок 5.8, а). У другому випадку (з підтягуючим резистором до землі) логіка входу є прямою: при відпущеній кнопці – “0”, при натиснутій – “1”.

Для запобігання аварій через невірне налаштування виводів портів (на вихід, замість того, щоб на вхід), до яких підключені кнопки, необхідно підключати послідовно з кнопками захисні резистори (рисунок 5.8, б).

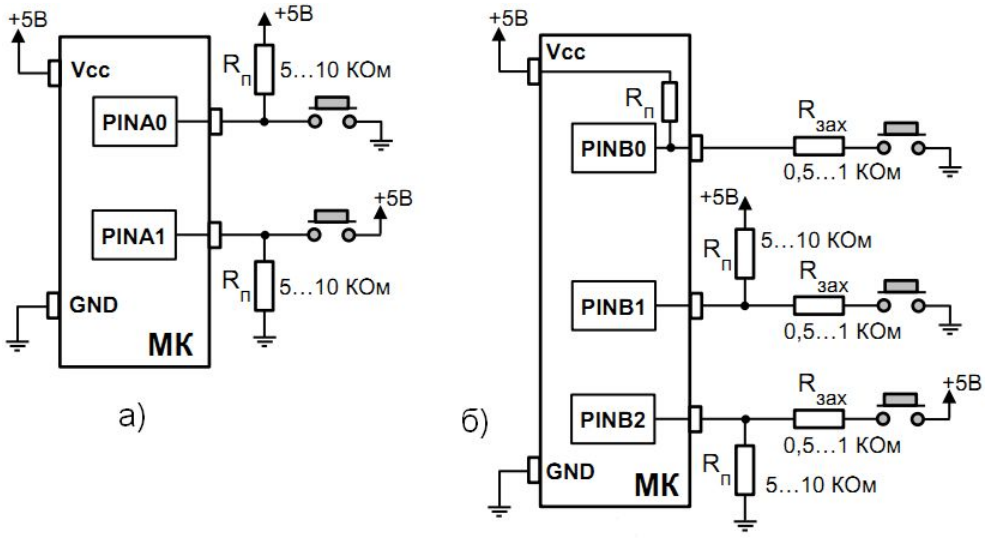


Рисунок 5.8 – Підключення кнопок

Висновки.

Всі порти введення/виведення базової серії МК AVR є 8-розрядними і двонаправленими. Кожний вивід порту може бути індивідуально сконфігурований на вхід або на вихід.

Виводи портів, окрім основного призначення – побайтового чи побітового введення/виведення, можуть також використовуватися для роботи з деякою внутрішньою периферією МК (у залежності від того, які альтернативні функції закріплені за ними).

Для керування виводами портів PXn призначені три регістри: регістр даних порту PORTX, регістр напрямку передачі даних DDRX, регістр виводів PINX (літера X відповідає назві регістра A, B, C чи D, n – відповідає номеру вивода 0,…,7).

Усі виводи портів мають внутрішній діодний захист від стрибків підвищеної напруги та від'ємної напруги.

При підключенні різноманітних кнопок та клавіатур до МК для уникнення дії завад необхідно використовувати підтягуючі резистори до напруги живлення чи землі.

До виводів портів можуть підключатися світлодіоди, кнопки, клавіатури.

Питання.

1. Скільки портів введення/виведення мають МК AVR?
2. Які регістри призначені для керування роботою портів введення/виведення?
3. Як конфігуруються порти введення/виведення в МК AVR?
4. Схеми та варіанти підключення кнопок до портів введення/виведення в МК AVR.
5. Схеми та варіанти підключення світлодіодів до портів введення/виведення в МК AVR.
6. Електричні характеристики системи введення/виведення МК AVR.

ЛЕКЦІЯ 6. Індикатори, кнопки, клавіатури та оптичні давачі

Мета. Вивчення організації динамічної індикації для відображення інформації, підключення кнопок та клавіатур до МК AVR, типів та особливостей роботи оптичних давачів.

Вступ. МК AVR використовуються не тільки для керування пристроями, але і для організації взаємодії з користувачем. При збільшенні числа елементів відображення інформації, та підключенні кнопок та клавіатур потрібні додаткові апаратні та програмні засоби.

План.

1. Варіанти організації динамічної індикації
2. Відображення інформації на LCD індикаторі
3. Кнопки та давачі
4. Оптичні давачі
4.1. Щілинний оптрон
4.2. Відбивальний оптрон
4.3. Швидкість спрацьовування оптронів
4.4. Коефіцієнт підсилення за струмом
4.5. Особливості роботи в інфрачервоному діапазоні
4.6. Дискретні оптичні датчики

1. Варіанти організації динамічної індикації

Для організації динамічної індикації використовується матриця, що складається з ліній рядків і ліній стовпчиків (рис. 6.1). На перетині рядка і стовпчика матриці розміщений індикаторний елемент – світлодіод.

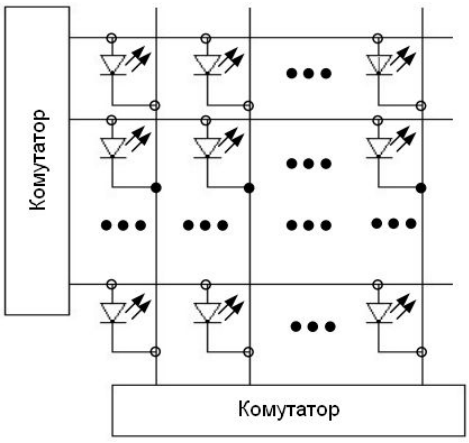


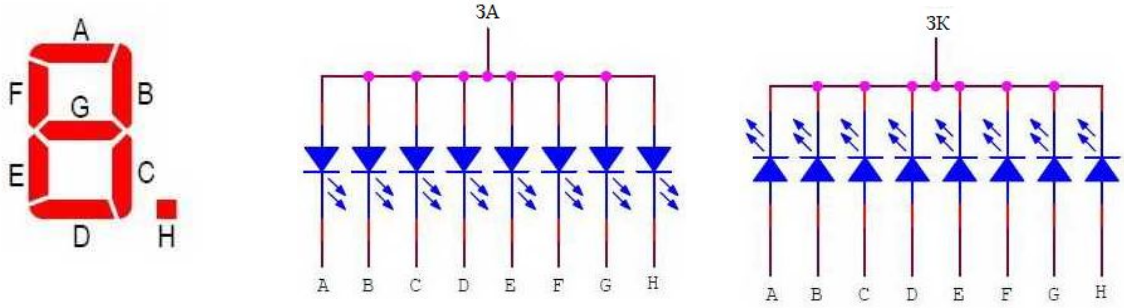
Рисунок 6.1 – Організація динамічної індикації

Для того, щоб засвітити той або інший елемент, необхідно подати на матрицю не один, як у звичайних індикаторах, а два сигнали: логічна 1 на відповідному рядку і логічний 0 на відповідному стовпчику матриці. Через односторонню провідність світлодіода кожна комбінація сигналів на входах рядків і стовпчиків однозначно включає тільки один індикаторний елемент.

Головна перевага динамічної індикації – невелике число керуючих ліній: для матриці світлодіодів розміром `N×N` елементів потрібно всього `2·N` керуючих сигналів. Основним

недоліком динамічної індикації є те, що при почерговому виведенні інформації на кожен світлодіод матриці його яскравість світіння, буде в N^2 разів нижче, ніж при безпосередньому виведенні інформації на один окремий світлодіод. Тому в пристроях з динамічною індикацією, інформацію виводять не на кожний світлодіод окремо, а на один рядок або на один стовпчик – у цьому випадку яскравість світіння світлодіодів падає тільки в N разів.

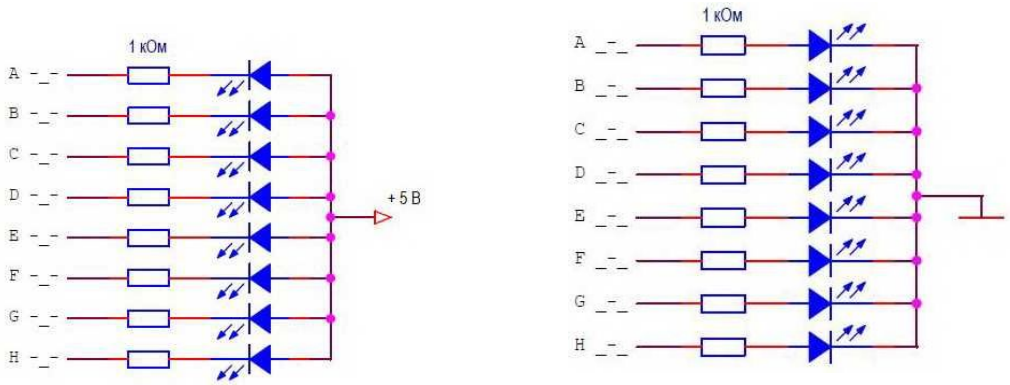
Як більш інформативні індикаторні елементи використовуються семисегментні індикатори. Семисегментний світлодіодний індикатор відображає символ за допомогою семи світлодіодів – сегментів цифри. Восьмий світлодіод засвічує десяткову крапку. Так що в семисегментному індикаторі 8 сегментів. Сегменти позначаються латинськими літерами від "A" до "H". Аноди або катоди кожного світлодіода об'єднуються в індикаторі і утворюють загальний провід. Тому існують індикатори із загальним анодом і загальним катодом, рис. 6.2.



а) б) в)

Рисунок 6.2 – 7-сегментний індикатор: а) нумерація сегментів; б) індикатор із загальним анодом; в) індикатор із загальним катодом

Підключати світлодіодні індикатори до мікроконтролера необхідно через резистори, що обмежують струм, рис. 6.3.



а) б)

Рисунок 6.3 – Підключення сегментів: а) спільний анод; б) спільний катод

Світлодіодним індикатором можна керувати статично або динамічно. Для підключення кожного семисегментного індикатора до МК потрібно вісім виводів. Якщо індикаторів (розрядів) 3 - 8, то в МК не вистачить стільки виводів. В цьому випадку індикатори можна підключити в мультиплексованому режимі, в режимі динамічної індикації. Виводи однойменних

сегментів кожного індикатора об'єднуються. Виходить матриця світлодіодів, підключених між виводами сегментів і загальними виводами індикаторів. Для підключення трьох індикаторів потрібно 11 виводів, а не 24, як при статичному режимі керування. Схеми мультиплексованого керування 3-розрядним індикатором із загальним анодом та загальним катодом показані на рис. 6.4.

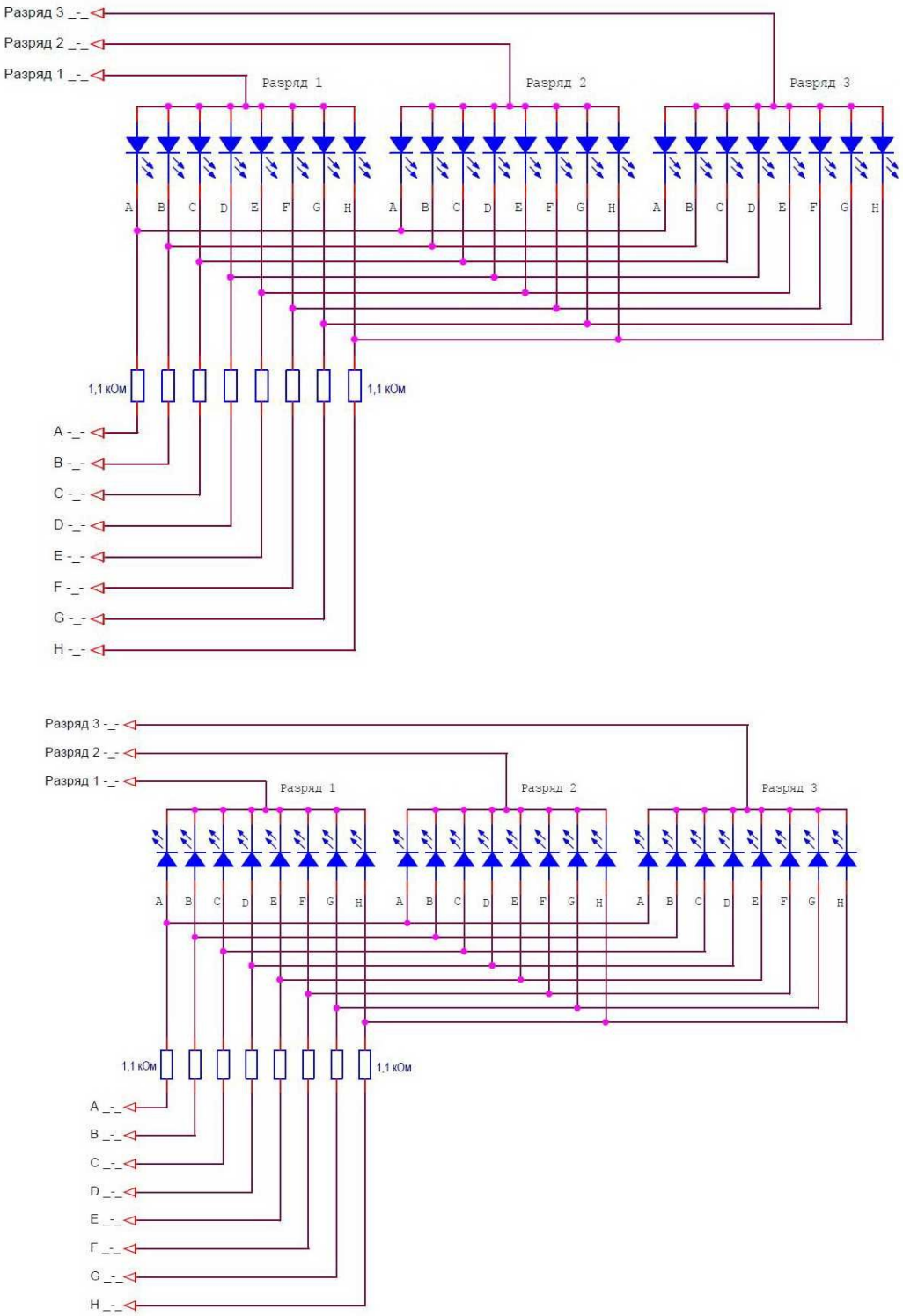


Рисунок 6.4 – Схеми мультиплексованого керування 3-розрядним індикатором

При динамічній індикації в кожен момент часу горить тільки одна цифра. На загальний вивід одного з розрядів подається сигнал високого рівня (5 В), а на виводи сегментів надходять

сигнали низького рівня для тих сегментів, які повинні світитися в цьому розряді. Через певний час засвічується наступний розряд. На його загальний вивід подається високий рівень, а на виводи сегментів сигнали стану для цього розряду. І так для всіх розрядів в нескінченному циклі. Час циклу називається часом регенерації індикаторів. Якщо час регенерації досить малий, то людське око не помітить перемикання розрядів. Буде здаватися, що всі розряди світяться постійно. Для запобігання мерехтіння індикаторів потрібна частота циклу регенерації бути не менше 70 Гц.

Якщо індикатори споживають більший струм, то необхідно використовувати додаткові ключі, особливо для сигналів вибору розрядів. Загальний струм розряду в 8 разів більше струму одного сегмента. Схеми підключення світлодіодного індикатора із загальним анодом в мультиплексованому режимі з транзисторними ключами вибору розрядів показана на рис. 6.5. Для вибору розряду в цій схемі необхідно сформувати сигнал низького рівня. Відповідний ключ відкриється і подасть живлення на розряд індикатора.

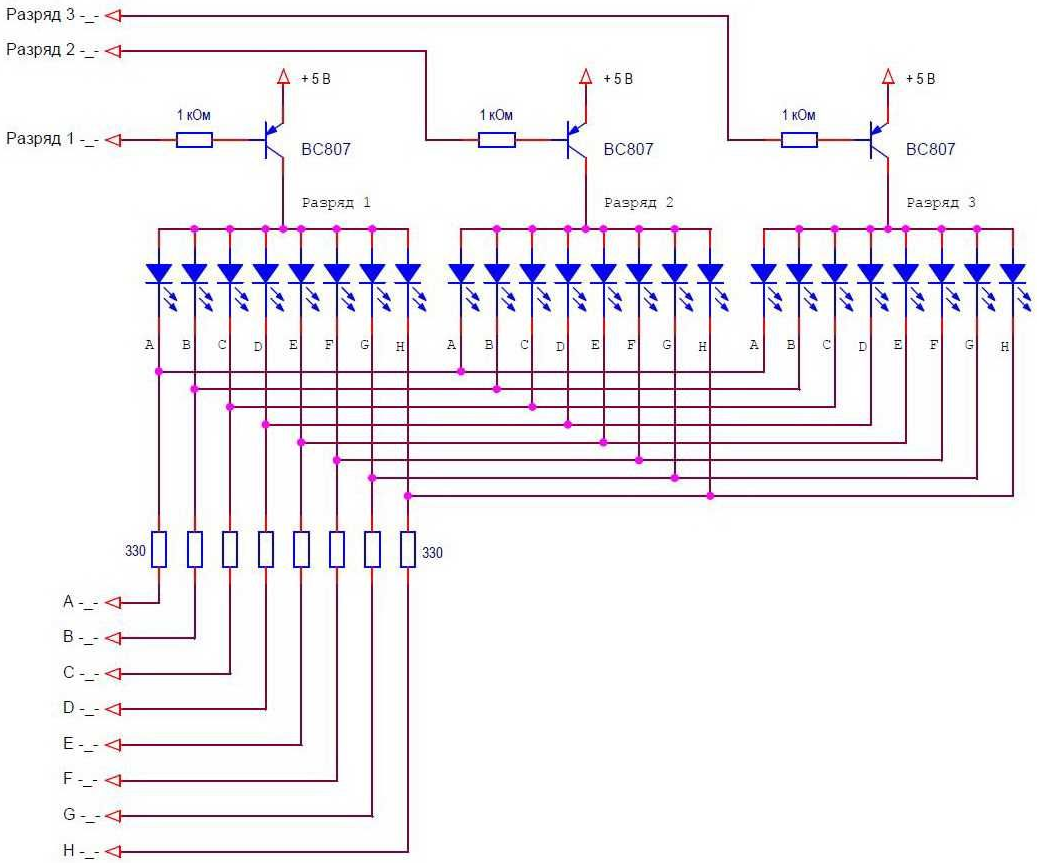


Рисунок 6.5 – Схема світлодіодного індикатора із загальним анодом в мультиплексованому режимі з транзисторними ключами вибору розрядів

Схеми підключення світлодіодного індикатора із загальним катодом в мультиплексованому режимі з транзисторними ключами вибору розрядів показана на рис. 6.6. Для вибору розряду в цій схемі необхідно сформувати сигнал високого рівня. Відповідний ключ відкриється і замкне загальний вивід розряду на землю.

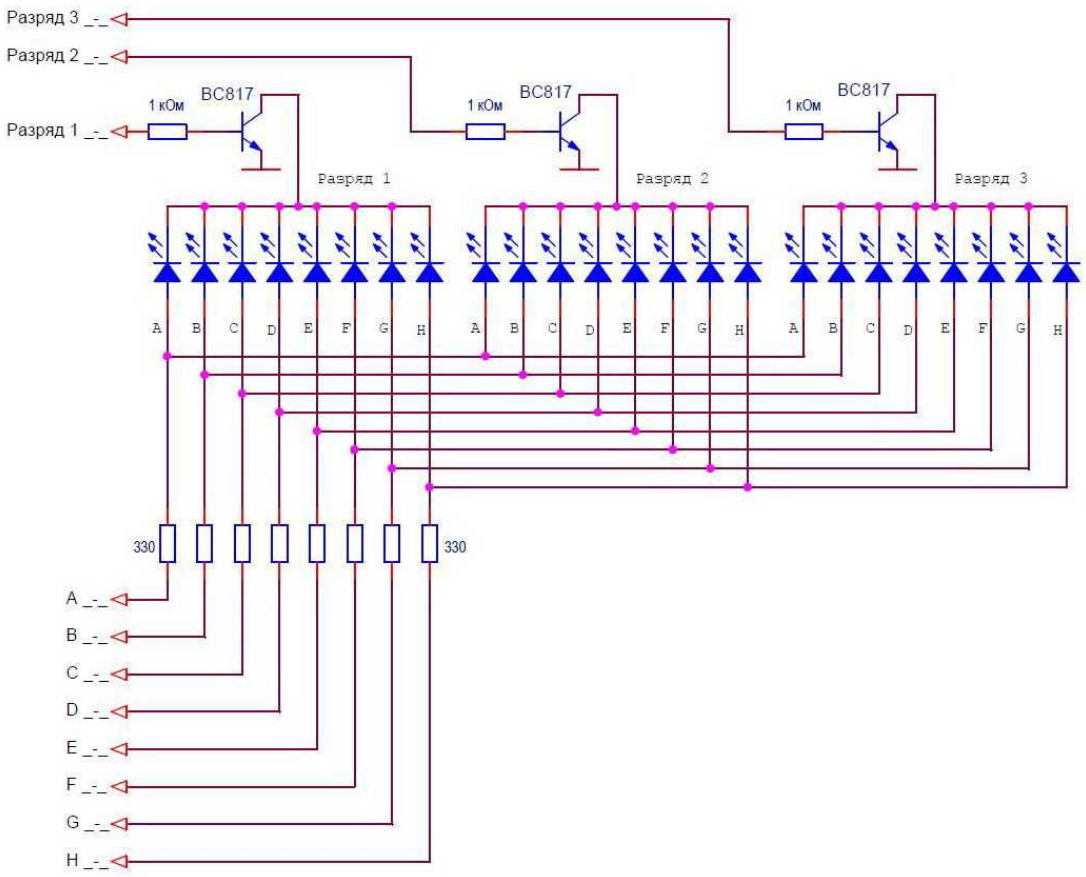


Рисунок 6.6 – Схема світлодіодного індикатора із загальним катодом в мультиплексованому режимі з транзисторними ключами вибору розрядів

Бувають індикатори великих розмірів, в яких кожен сегмент складається з декількох світлодіодів, з'єднаних послідовно. Для живлення таких індикаторів потрібно джерело з напругою більшою, ніж 5 В. Ключі повинні забезпечувати комутацію підвищеної напруги з керуванням від сигналів рівнів мікроконтролера (звичайно 5 В).

Схема ключів, замикаючих сигнали індикатора на землю, залишається незмінною. А ключі живлення будуються за іншою схемою, наприклад, як на рис. 6.7.

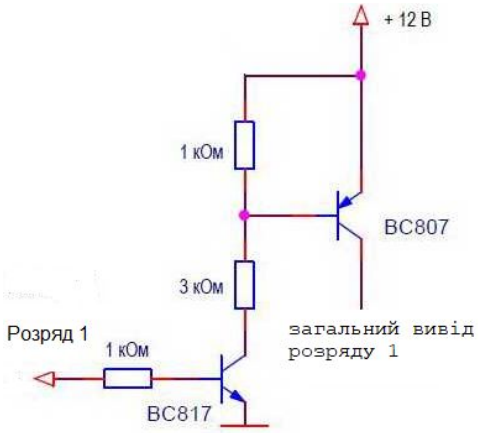


Рисунок 6.7 – Схема ключів живлення

Електрична схема підключення динамічного індикатора до МК AT90S2313 без додаткових елементів показана показана на рис. 6.8. До порту B (`PortB`) МК підключені катоди всіх світлодіодів матриці, а до порту D – аноди кожного індикатора матриці. На лініях порту D організовується “біжуча” одиниця. На лінії порту B при кожному положенні біжучої одиниці виводиться семисегментний код того символу, який повинен горіти в даному знакомісці. Для індикаторів із загальним анодом замість біжучої одиниці, використовується біжучий нуль. Символ, що буде світитись на індикаторі визначається сигналом логічної 1 на виводі 1, 2, 3 або 4 індикатора. На шині даних (A, B, …, G, DP) активним рівнем є рівень логічного 0.

Перевага такого способу індикації – у відсутності яких-небудь додаткових компонентів (крім самих світлодіодних індикаторів), головний недолік – значна перевитрата ліній портів.

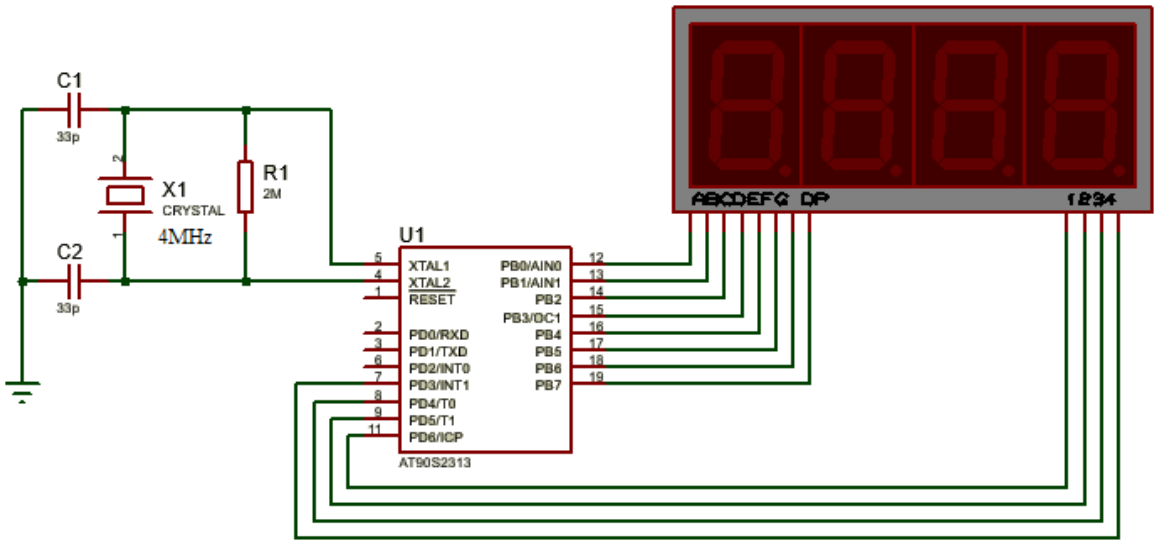


Рисунок 6.8 – Підключення динамічного індикатора до МК AT90S2313

Схема реалізації динамічної індикації з додатковим елементом (буферним регістром) показана на рис. 6.9.

До порту B (`PortB`) МК підключені катоди всіх світлодіодів матриці, а також – входи регістра защіпки 74HC373 (або регістра затримки 74HC377). До виходів регістра підключені аноди кожного з семисегментних індикаторів. На порт B спочатку виводиться біжуча одиниця яка з одного поточного виводу записується в регістр сигналом LE. Потім на лінії порту B видається семисегментний код символу, який повинен горіти в знакомісці, що визначає сигнал на виході буферного регістра. В даному випадку лінії порту B використовуються в режимі часового мультиплексування, тобто по них по черзі передається і код символу, і номер знакомісця. Перевага такого способу індикації – менша витрата ліній портів МК (окрім порту B – додаткові дві лінії) і можливість роботи до 8 індикаторів одночасно.

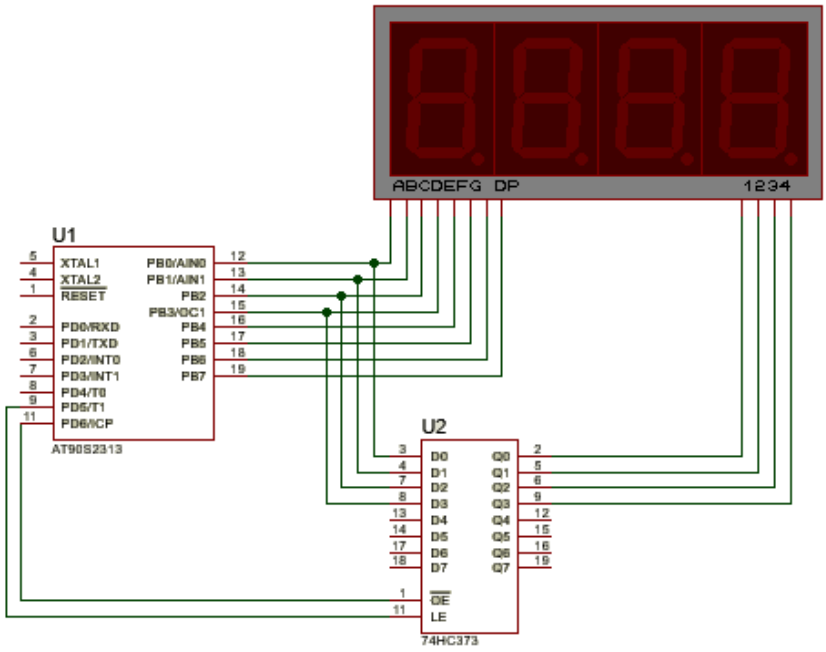


Рисунок 6.9 – Схема динамічної індикації з регістром защіпкою

Ще один варіант індикації з одним додатковим елементом (регістром зсуву 74164) показаний на рис. 6.10. У цьому варіанті біжуча одиниця реалізується за допомогою регістра зсуву. Порт B використовується в режимі часового мультиплексування – як для видачі символу, так і для занесення чергового біта до регістра зсуву. Схема використовує в альтернативному варіанті дві лінії порту B.

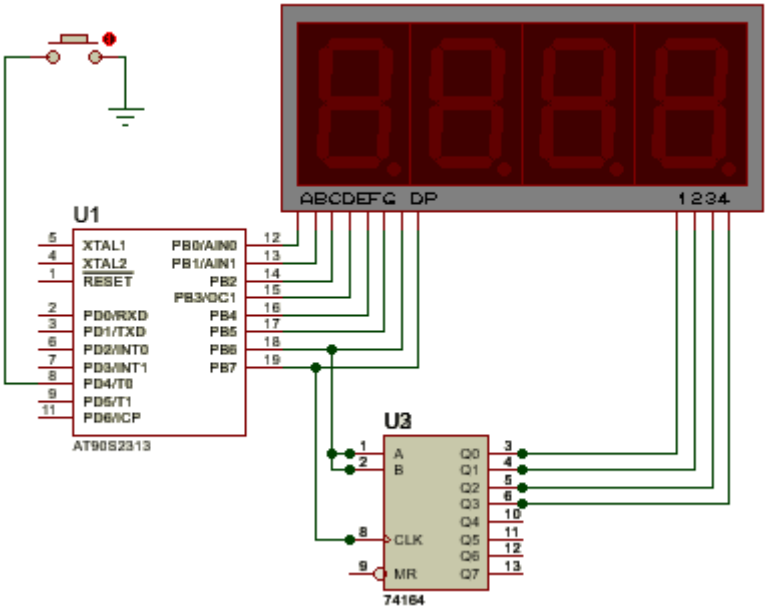


Рисунок 6.10 – Схема динамічної індикації з регістром зсуву

Інший варіант реалізації схеми з одним додатковим елементом (дешифратором) показаний на рис. 11. Така схема підходить тільки для індикаторів із загальним катодом, оскільки для організації біжучого нуля використовується дешифратор з виходом логічного 0. Схема використовує три додаткові лінії (крім порту D) і у багатьох випадках може бути корисною.

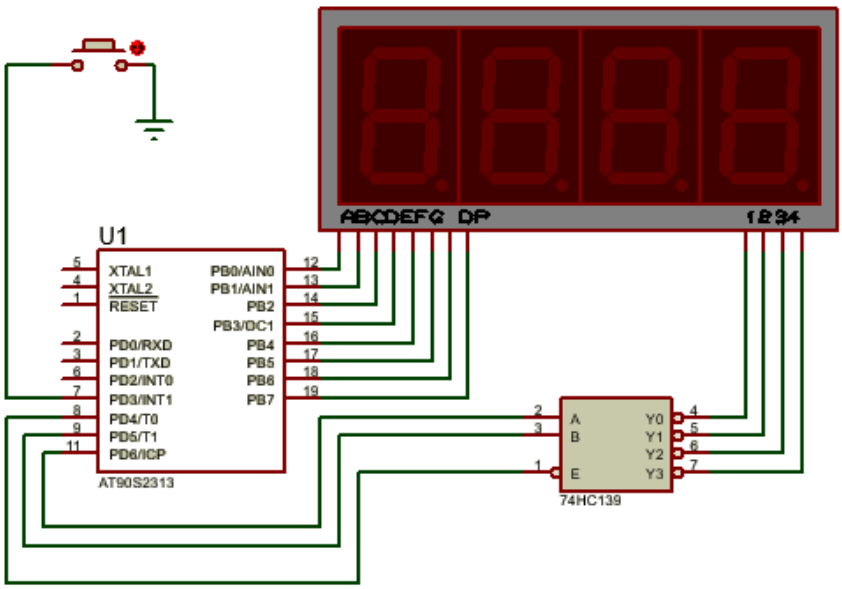


Рисунок 6.11 – Схема динамічної індикації з дешифратором

Схема реалізації динамічної індикації з двома додатковими елементами показана на рис. 6.12. У схемі використовуються два послідовні регістри зсуву: один – для розгортки зображення по стовпчиках (замінює порт B), а інший – для розгортки зображення по рядках (замінює порт D). Перевага такої схеми – всього три лінії порту МК. Крім того, при такому варіанті реалізації блок динамічної індикації легко оформити і у вигляді окремої плати. Недолік такої схеми – два додаткових елемента.

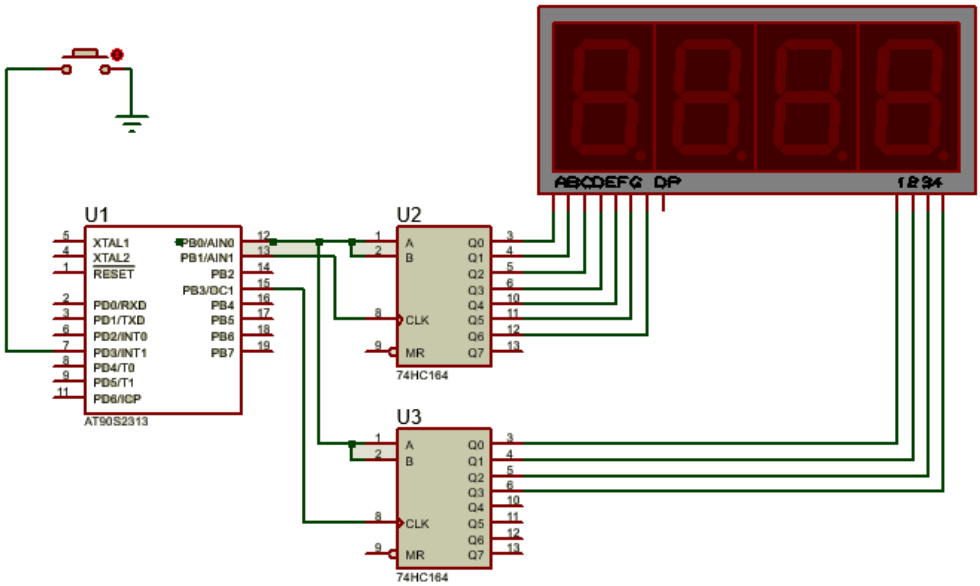


Рисунок 6.12 – Схема динамічної індикації з двома незалежними послідовними регістрами зсуву

Ще одна схема динамічної індикації з двома додатковими елементами показана на рис. 6.13. Перевага такої схеми – використання всього двох ліній порту. Недолік – складніша програма управління і велика тривалість формування вихідних сигналів, що викликає деяке паразитне підсвічування індикаторів (яке помітне тільки в темряві).

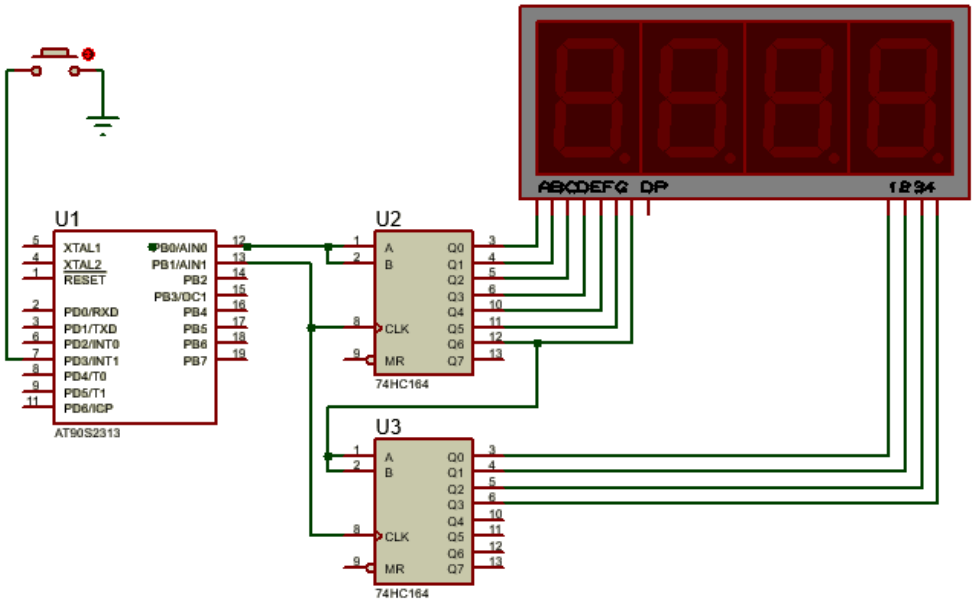


Рисунок 6.13 – Схема динамічної індикації з двома залежними послідовними регістрами зсуву

2. Відображення інформації на LCD індикаторі

Для відображення поточних значень температури (температури жала та температури, що задана блоком керування) використовують LCD індикатор з контролером типу HD44780 фірми Hitachi [4], який є промисловим стандартом де-факто на ринку цифро-буквові індикаторів.

Є два способи підключення LCD індикатора – на 8-біт і 4-біти. У 8-бітовому режимі байти швидко передаються без зсуву, а в 4-бітовому режимі байти передаються із зсувом, але через чотири виводи контролера.

У 8-бітовому режимі LCD на базі HD44780 підключається безпосередньо до портів МК (рис. 6.14).

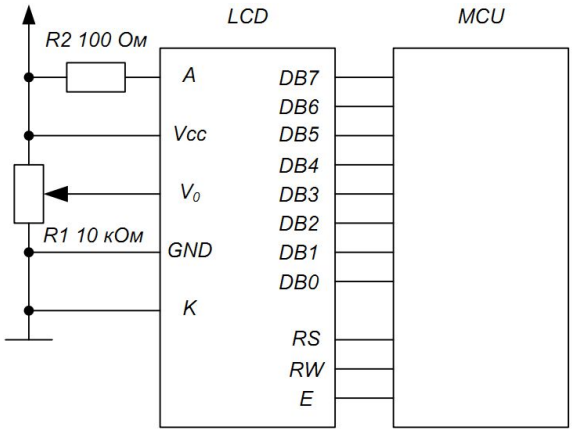


Рисунок 6.14 – 8-бітове підключення LCD індикатора до МК

У 4-бітовому режимі підключення LCD індикатора до МК використовується чотири виводи порту D, рис. 6.15.

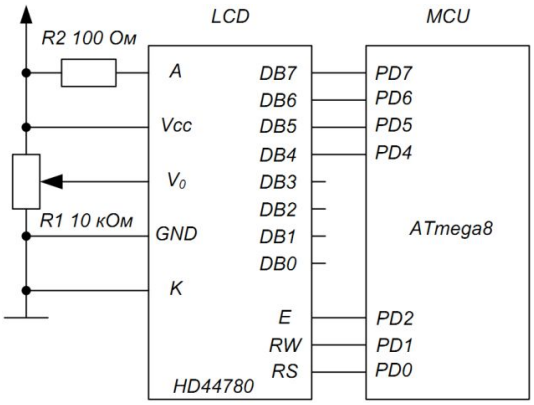


Рисунок 6.15 – 4-бітове підключення LCD індикатора до МК

Позначення на рисунку:

`DB7...DB0` – виводи даних/адрес;

`E` – вхід синхронізуючого сигналу передачі по шині даних.

`R/W` – сигнал керування задає тип звернення до LCD індикатора: 1 – читання, 0 – запис.

`RS` – сигнал керування визначає тип даних, що передаються: 0 – команда, 1 – дані. Дані будуть записані за поточною адресою, а команда виконана контролером LCD;

`GND` – мінус живлення, він же загальний;

`Vcc` – плюс живлення, 5В;

`Vo` – вхід контрастності;

`A,K` – анод, катод.

Структура адресації контролера HD44780 показана на рис. 6.16.

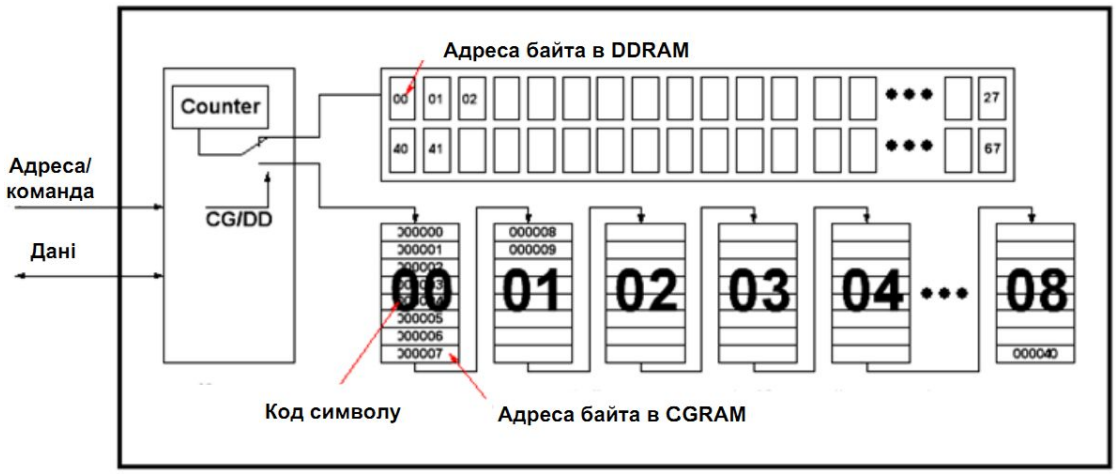


Рисунок 6.16 – Структура адресації контролера HD44780

Контролер HD44780 має свій блок керування, що обробляє команди й пам'ять. Пам’ять ділиться на три типи: DDRAM, CGROM, CGRAM.

DDRAM – пам'ять дисплея. Все що запишеться в DDRAM буде виведено на екран. Наприклад, якщо записується код 0x31 – на екрані відображається символ «1», так як 0x31 це

ASCII код цифри 1. Але є одна особливість – DDRAM пам'ять (буфер даних) набагато більша ніж видима область екрану. Як правило, DDRAM містить 80 комірок – 40 у першому рядку й 40 у другому. Рядок дисплею має 16 позицій, тому для відображення всіх елементів буфера даних вікно дисплею переміщується вздовж масиву (рис. 6.17).

Всі елементи, які попадають у вікно, відображаються на екрані дисплею, інші залишаються поза зоною видимості.

Для переміщення вікна дисплею є спеціальна команда. Також є поняття курсору – це місце в яке буде записаний наступний символ, тобто поточне значення лічильника адреси. Курсор може бути на екрані, поза екраном або бути зовсім відключеним.

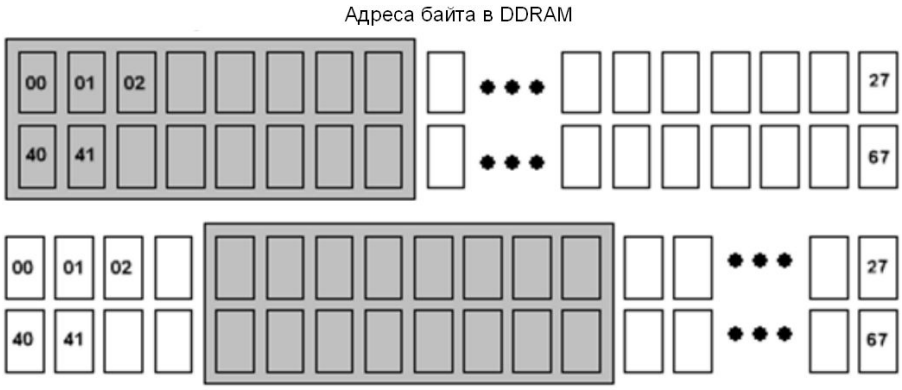


Рисунок 6.17 – Видима (сіра) й невидима (біла) область екранної пам'яті

CGROM – таблиця символів. Коли записується в комірку DDRAM байт, то з таблиці береться символ і відображається на екрані. CGROM не можна змінити, тому важливо, щоб таблиця мала кириличні букви.

CGRAM – теж таблиця символів, але її можна змінювати, створюючи свої символи. Адресується вона лінійно, тобто спочатку розміщуються 8 байт одного символу, а за ним наступні символи, як показано на рис. 18. Знакомісце займає 5x8 бітів. Один біт дорівнює одній крапці на екрані. Усього в CGRAM може бути 8 символів, відповідно CGRAM має 64 байти пам'яті. Ці програмовувані символи мають коди від 0x00 до 0x07. Так, записавши, наприклад, у перші 8 байт CGRAM (перший символ з кодом 00) будь який символ, і записавши в DDRAM нуль (код першого символу в CGRAM) побачимо на екрані цей символ.

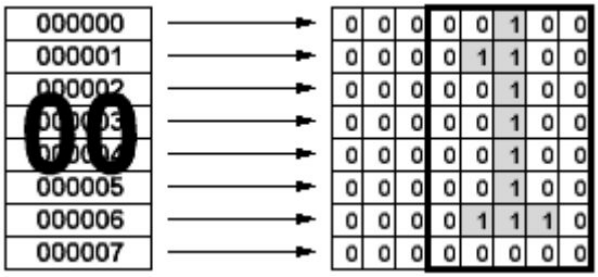


Рисунок 6.18 – Формування символу в комірці CGRAM

Система команд. Командою вибирається в яку саме пам'ять і починаючи з якої адреси будуть записуватись дані, а потім пересилаються байти. Система команд контролера дуже проста. Про те, що передається команда контролеру дисплея повідомить сигнал RS=0. Сама команда складається зі старшого біта, який визначає за що відповідає дана команда й бітів параметрів, що вказують контролеру HD44780 що далі робити.

Таблиця 6.1 – Система команд контролера HD44780

№	DB7	DB6	DB5	DB4	DN3	DB2	DB1	DB0	Описання команди
1	0	0	0	0	0	0	0	1	Очищення екрану. Курсор за адресою 0 в DDRAM
2	0	0	0	0	0	0	1	-	Курсор в 0. Дисплей відносно буфера DDRAM в початковій позиції
3	0	0	0	0	0	1	I/D	S	Курсор зсовується вправо (I/D=1) або вліво (I/D=0). Зсув дисплею разом з курсором (S=1)
4	0	0	0	0	1	D	C	B	Включення (D=1) або виключення (D=0) дисплею. Включення (C=1) або виключення (C=0) курсору, блимання курсору (B=1)
5	0	0	0	1	S/C	R/L	-	-	Зсув курсору (S/C=0) або дисплею (S/C=1), вправо (R/L=1) або вліво (R/L=0)
6	0	0	1	DL	N	F	-	-	Розрядність шини даних: 4-біт (DL=0), 8-біт (DL=1); кількість рядків дисплею: один (N=0), два (N=1); шрифт: 5x7 (F=0), 5x10 (F=1) крапок
7	0	1	AG	AG	AG	AG	AG	AG	Перемкнути адресацію на CGRAM і задати адресу
8	1	AD	AD	AD	AD	AD	AD	AD	Перемкнути адресацію на DDRAM і задати адресу
9		BF			AC				Читання стану busy-прапора і лічильника адрес

Примітка. 1-8 – команди, які записуються при значеннях сигналів R/S=R/W=0;
9 – читання при RS=0, R/W=1;
10 – запис даних при RS=1, R/W=0 в буфер даних DDRAM або в пам’ять знакогенератора CGRAM;
11 – читання даних при RS=R/W=1 з пам’яті DDRAM або CGRAM.

`I/D` – інкремент або декремент лічильника адреси. За замовчуванням встановлений в 0 (декремент). Інкремент 1.

`S` – зсув екрана, якщо поставити 1, то з кожним новим символом буде пересуватись вікно екрана, поки не досягне кінця DDRAM.

`D` – включити дисплей. Якщо поставити 0 то зображення зникне, а в цей час можемо у відеопам'ять записати нову інформацію. Для того, щоб вона з'явилася в цю позицію треба записати 1.

`C` – включити курсор у вигляді прочерку (C=1).

`B` – зробити курсор у вигляді блимаючого чорного квадрату.

S/C – зсув курсору або екрана. Якщо встановлено 0, то зсувається курсор, якщо 1, то – екран. По одному разу за команду.

R/L – визначає напрямок зсуву курсору й екрану (0 – ліворуч, 1 – праворуч).

D/L – біт, що визначає ширину шини даних (1 – 8 біт, 0 – 4 біти).

N – число рядків (0 – один рядок, 1 – два рядки).

F – розмір символу (0 – 5x8, 1 – 5x10) крапок.

AG – адреса в пам'яті CGRAM.

AD – адреса в пам'яті DDRAM.

3. Кнопки та давачі

За допомогою кнопок та давачів як периферійних елементів в МК поступає різна інформація, яка використовується для зміни алгоритму роботи програми.

Схема підключення контактного давача до МК наведена на рис. 6.19. Давач підключений до лінії PD0 порту D МК. Через цей вхід МК зчитує стан давача. Давач можна підключити і до будь-якої іншої лінії будь-якого з портів МК.

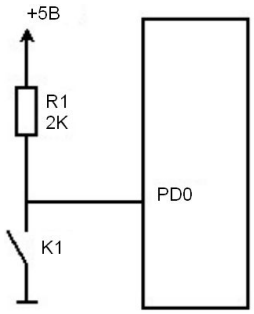


Рисунок 6.19 – Підключення контактного давача до МК

У початковому стані контакти давача розімкнені. На вхід МК через резистор R1 прикладається напруга від джерела живлення +5 В. МК сприймає цю напругу як сигнал логічної одиниці. При спрацьовуванні давача контакти замикаються і з’єднують вивід МК із загальною шиною. Тепер МК сприймає вхідний рівень сигналу як логічний нуль. Резистор R1 при цьому служить струмообмежувальним елементом, запобігаючи короткому замиканню між шиною живлення і загальною шиною. Деякі МК мають свої внутрішні резистори навантаження, які можуть замінити зовнішній резистор. Схема підключення декількох давачів або кнопок до МК показана на рис. 6.20.

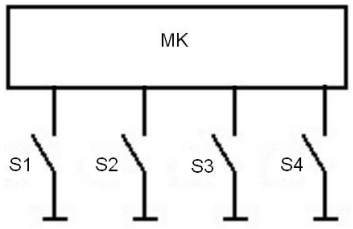


Рисунок 6.20 – Підключення кнопок або простих давачів до МК

У схемі, що зображена рис. 6.21, при натисненні однієї з кнопок змінюється постійна напруга на відповідному вході МК, яка розпізнається і дешифрується в певну команду. Ця напруга максимальна (приблизно 5 В), коли кнопки не натиснуті, і мінімальна (0 В) при натиснутій клавіші S1.

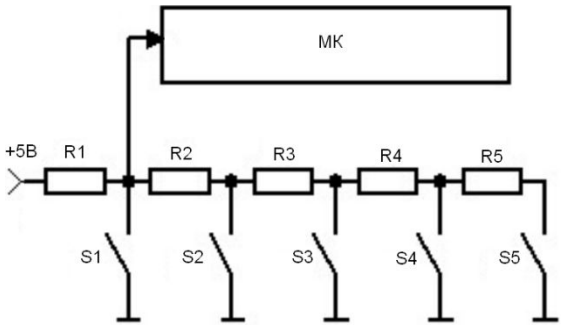


Рисунок 6.21 – Зміна напруги на аналоговому вході МК при включені кнопок

Крім кнопок до МК може підключатися і клавіатура. Існує два види клавіатур: із скануванням та з кодуванням.

Блок-схема 12-клавішної клавіатури з скануванням показана на рис. 6.22. Клавіші розташовані у вузлах матриці з чотирма лініями рядків і трьома лініями стовпців. На лінії стовпчиків по черзі подається логічний “0”. У цей момент перевіряється стан чотирьох ліній рядків. Якщо натиснутих клавіш немає, всі лінії рядків мають високий рівень (вони підключені до напруги +5V через резистори). Якщо є натиснута клавіша, то відповідна лінія рядка матиме стан логічного “0”. Знаючи номери стовпця і рядка, можна отримати позицію натиснутої клавіші.

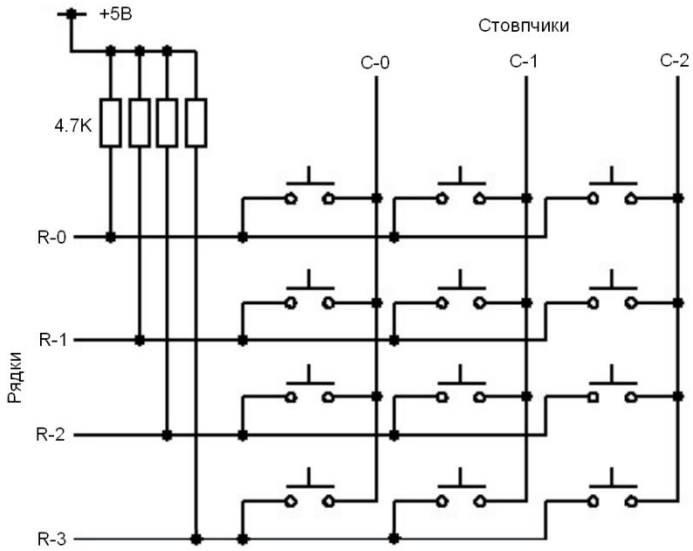


Рисунок 6.22 – Матрична клавіатура 4x3 у схемі з скануванням

У клавіатурі з кодуванням застосовують спеціалізовані мікросхеми, які виявляють натиснення клавіші і передають її код. Прикладом такого пристрою є інтегральна мікросхема (IMC) MM74C922 фірми National Semiconductors.

IMC 74922 має генератор синхроімпульсів, лічильник і дешифратор, які сканують клавіатуру, визначають натиснуті клавіші й перетворюють цю подію у двійковий код. При натисканні клавіші мікросхема переводить вивід DAV у високий стан, вказуючи, що оброблено натискання і результат виставлено на виводи D0, D1, D2, D3 (рис. 6.23).

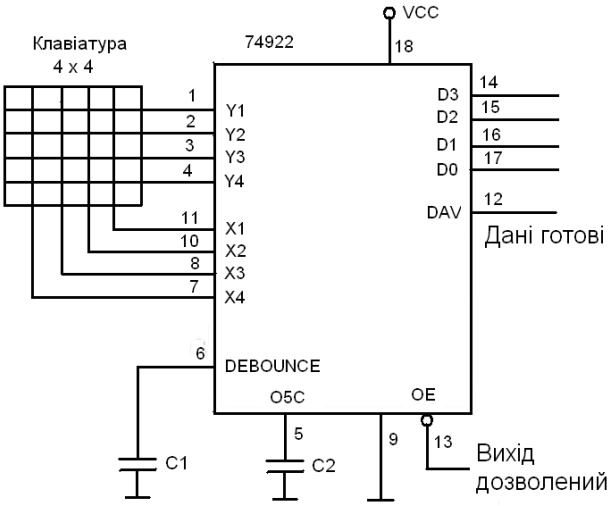


Рисунок 6.23 – Клавіатура 4x4 на IMC 74922

Структура матричної клавіатури 5x3 показана на рис. 6.24.

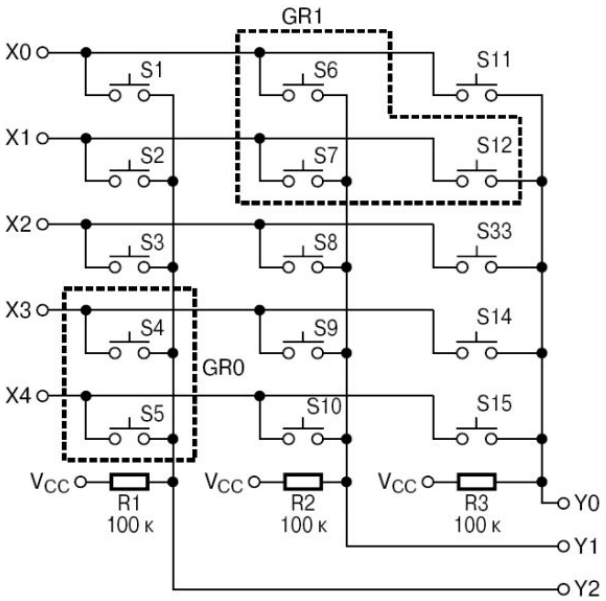


Рисунок 6.24 – Матрична клавіатура 5x3

Горизонтальні лінії X0...X4 (рядки) є вихідними, а вертикальні – Y0...Y2 (стовпці) є вхідними. У неактивному стані на виходах X0....X4 підтримується рівень логічної “1”, а входи Y0...Y2 звичайно підтягнуті до рівня живлення Vcc резисторами. Для цієї мети, як правило,

використовуються вбудовані резистори (pullup), які є у деяких портах МК. При скануванні на горизонтальні лінії матриці подається сигнал “біжучого” нуля. Для кожного стану біжучого нуля, зчитується і аналізується код Y0...Y2. Для натиснутої клавіші матриці, відповідний вивід стовпця буде переведений сигналом біжучого нуля в стан логічного “0”. Код натиснутої клавіші можна визначити програмно.

Крім визначення положення натиснутої клавіші потрібно програмно захиститися від "брязкоту" контактів, тобто від впливу перехідних процесів, а також з одночасним натисненням декількох клавіш.

Для запобігання протіканню небезпечних струмів при одночасному натисненні декількох клавіш у одному стовпчику, в лініях X0...X4 звичайно встановлюють послідовно розв'язуючі резистори або діоди. З цією ж метою можна використовувати і інший метод сканування, при якому всі неактивні горизонтальні лінії, крім лінії біжучого нуля, програмно призначаються входами. Для запобігання впливу перехідних процесів можна повторно зчитувати стан входів матриці скануванням через певну часову затримку.

У невеликих клавіатурних матрицях комбінації декількох натиснутих клавіш не використовуються. Але якщо це все-таки буде потрібно, то для сканування краще використовувати метод з призначенням входам неактивних рядків. У цьому випадку можна упевнено визначати дві одночасно натиснуті клавіші в одному рядку і навіть визначати позиції трьох одночасно натиснутих клавіш. Але це справедливо тільки якщо топологія трьох натиснутих кнопок не утворює прямого кута, як показано на рис. 6.25.

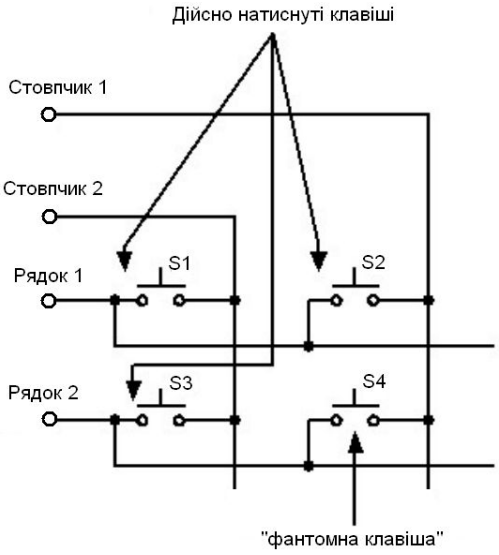


Рисунок 6.25 – Помилкова клавіша при скануванні

Для раціонального використання ресурсу портів їх виводи, які використовуються для сканування клавіатури, можна використовувати і для інших функцій, наприклад, для підтримки динамічної світлодіодної індикації. На рис. 6.26 показаний приклад такого мультиплексування портів МК.

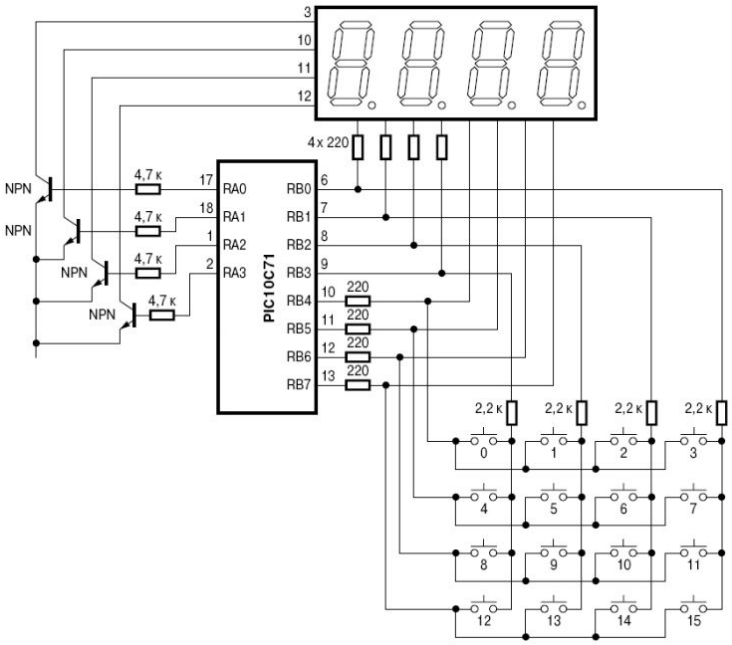


Рисунок 6.26 – Мультиплексорне керування матричною клавіатурою і 4-розрядним семисегментним світлодіодним індикатором

Схема матричної клавіатури на 32 клавіші показана на рис. 6.27.

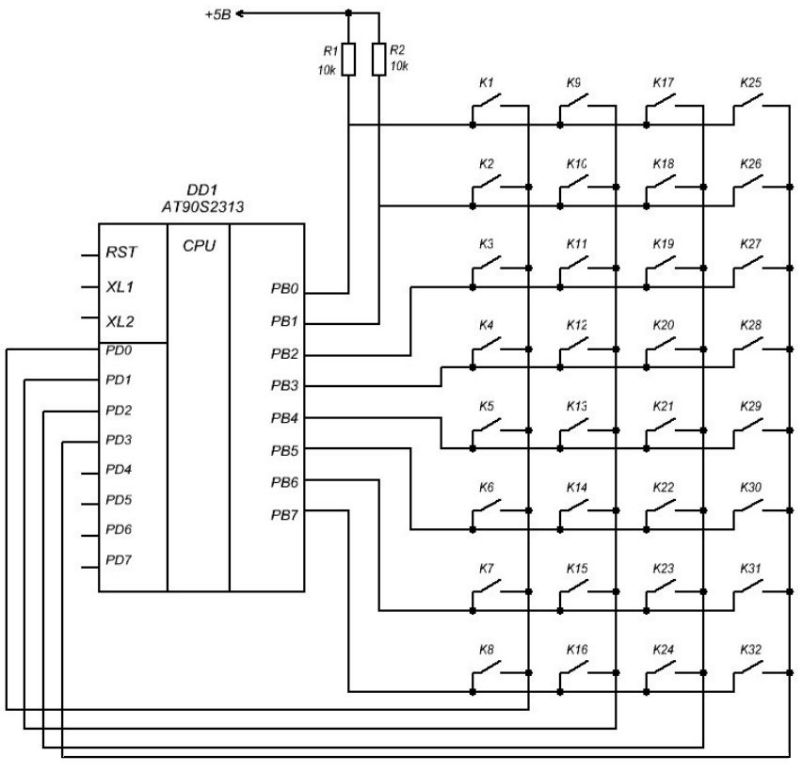


Рисунок 6.27 – Схема підключення клавіатури як матриці клавіш

Додавши один дешифратор, можна зекономити дві лінії порту PD. Схема клавіатури з дешифратором наведена на рис. 6.28. У цій схемі для вибору одного з чотирьох стовпчиків

клавіатури використовується дешифратор DD2 типу К555ИД4. У такій схемі для сканування стовпчиків МК повинен подавати на виходи PD0 і PD1 двійковий код номера стовпчика. Код поступає на входи A0 і A1 дешифратора. В результаті один з його виходів (той, номер якого відповідає коду, що поступив) прийме нульове значення. На решті виходів буде одиниця. Так, при коді 00B на вході дешифратора вихід Q0 (вивід 9) приймає нульове значення. Для коду 01B - нуль буде на виході Q1 і так далі. Таким чином, МК може перебирати всі чотири стовпчики, використовуючи всього два розряди.

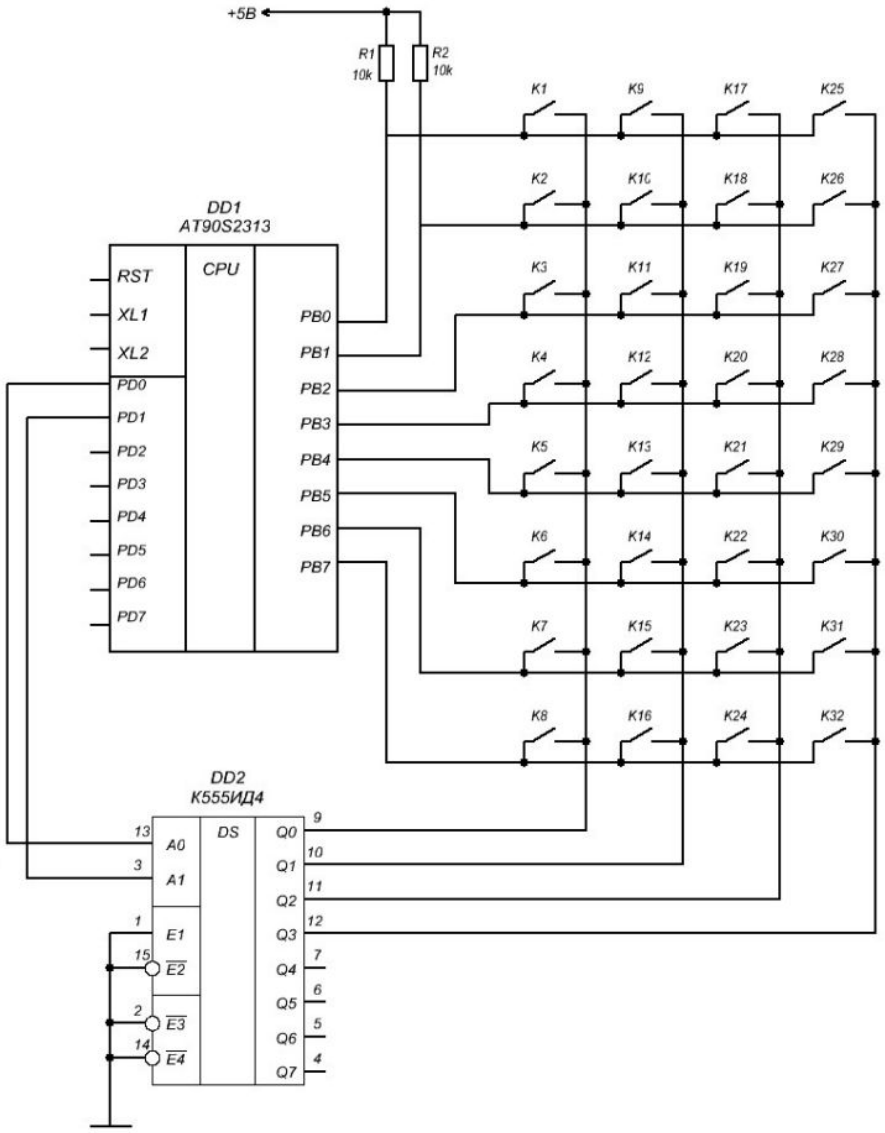


Рисунок 6.28 – Підключення клавіатури з використанням дешифратора

Існують і інші способи зменшення числа виводів МК, призначених для сканування клавіатури. Реалізація одного з таких способів наведена на рис. 29. У звичайних матричних клавіатурах при замиканні кожного контакту утворюється електричне з'єднання між вихідним і вхідним ланками схеми сканування. Але якщо змінити топологію розташування клавіш так, щоб при замиканні контакту забезпечувалася зміна потенціалів групи шин, то число кодованих клавіш можна значно збільшити. Така конструкція клавіатури забезпечує кодування більшого числа клавіш, ніж звичайна матрична, при однаковому числі шин, що використовуються для

кодування. Збільшення досягнуте за рахунок того, що додаткові кодові комбінації утворюються парами шин кодування.

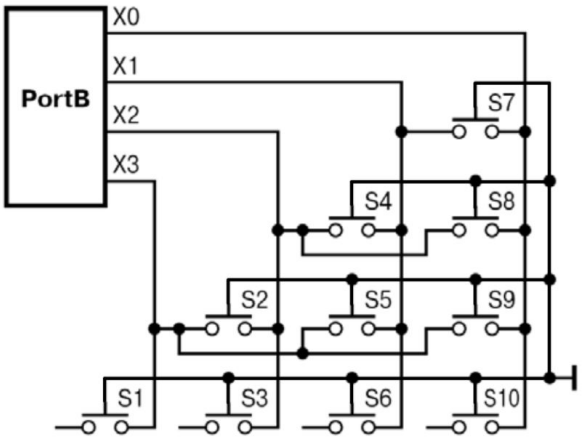


Рисунок 6.29 – Топологія клавіатури з одним контактором

Для подібного варіанту топології сканування не потрібне. Всі лінії є входами з підтягуванням до напруги живлення резисторами. Контактор з’єднаний із загальною шиною. При замиканні контакту потенціал контактора передається на одну або дві кодові шини, на перетині яких він знаходиться. Якщо ввести другий замикаючий контактор з потенціалом живлення, то можна удвічі збільшити число сканованих клавіш. Для цього варіанту необхідно проводити сканування, наприклад, з періодом 20-30 мс. При скануванні використовується ефект пам'яті на паразитній ємності ліній. У першій фазі сканування всі шини визначаються виходами, і на них подається потенціал логічного “0”. Потім виводи порту перевизначаться як входи, і проводиться зчитування їх стану. Факт замикання фіксується по зміні потенціалу, після чого проводиться друга фаза сканування. Цього разу на шини подається потенціал живлення, проводиться аналіз можливих станів ліній і визначаються позиції замкнутих контактів. Наприклад, при використанні чотирьох шин із звичайною матричною топологією можна сканувати всього 4 кнопки, а при використанні вищеописаного методу вже 20 (рис. 6.30).

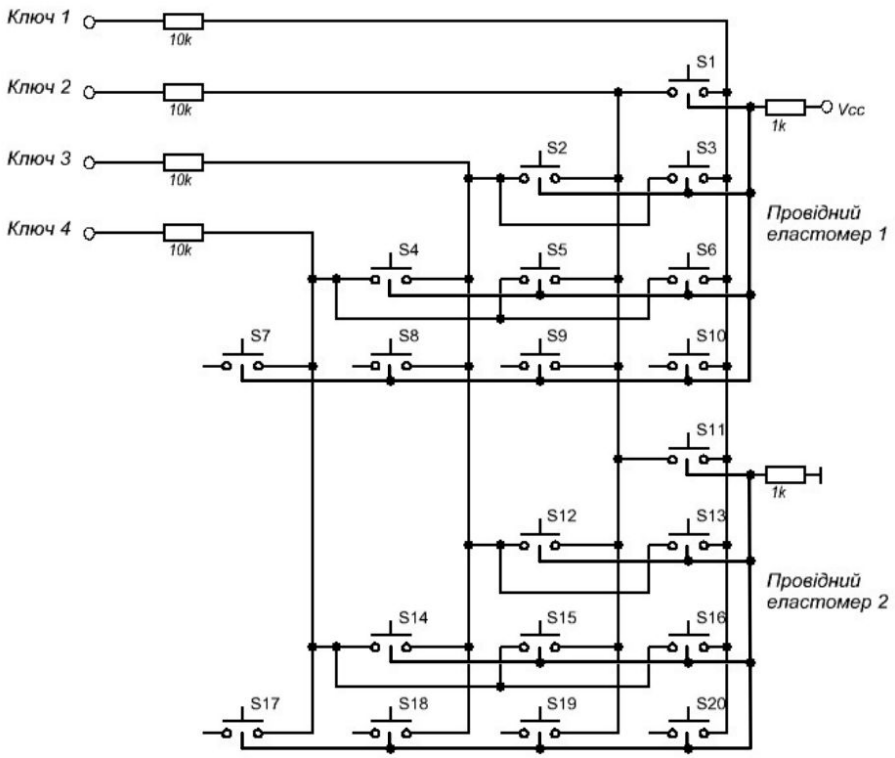


Рисунок 6.30 – Топологія клавіатури на 20 клавіш з 4 лініями сканування

Часто виникає необхідність використання великої кількості кнопок для різний цілей. Існують різні варіанти реалізації цього завдання. Коли необхідно підключити клавіатуру до МК з обмеженою кількістю вільних виводів, тоді використовують підключення клавіатури по трьох сигнальних дротах. Додаткові елементи: регістр зсуву SN74198N і декілька резисторів (рис. 6.31).

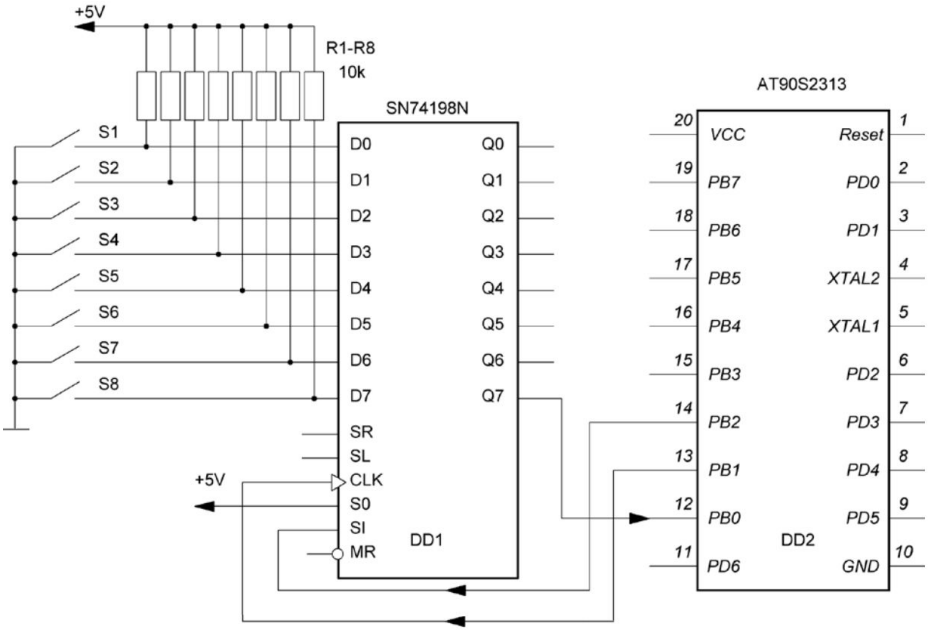


Рисунок 6.31 – Підключення клавіатури з використанням регістра зсуву

Кнопки підключені одним виводом до корпусу, другим до регістра зсуву. Резистори R1-R8 створюють високий логічний рівень на вході регістра зсуву при розімкнутих кнопках. Як тільки кнопка замикається, на вході регістра утворюється логічний нуль, оскільки він виявляється безпосередньо підключений до загального виводу. Значення резистора в 10 кОм не дає протікати занадто великому струму, поки кнопка замкнута, і підтримує високий логічний рівень, поки кнопка розімкнена.

Регістр зсуву має вхідний порт (D0 – D7), вихідний порт (Q0 – Q7) і сигнали керування (SR, SL, CLK, S0, S1, MR).

Вхідний і вихідний порт можна використовувати як в паралельному, так і в послідовному режимі.

D0 - D7 – вхід для 8 сигналів з кнопок.

Q0 - Q7 – паралельний вихід. Використовується лише Q7.

CLK – тактовий вхід. Усе в регістрі робиться тільки по тактах на цьому виводі (за наростаючим фронтом).

S0 і S1 – задають тип виходу регістра (Q0-Q7) – послідовний або паралельний. При S0=1 і S1=1 на вихід передаються дані з входів (D0-D7) за сигналом CLK.

S1=0 – зсув бітів на виході на 1-розряд у сторону старшого біта за сигналом CLK, За 8 тактів можна прочитати стан 8 кнопок у послідовному коді з виводу Q7.

Можна підключити 16 кнопок з використанням додаткового регістра зсуву. На вхід SR регістра DD2 подається сигнал з Q7 регістра DD1. Сигнал на МК буде подаватись з Q7 регістра DD2 (рис. 6.32).

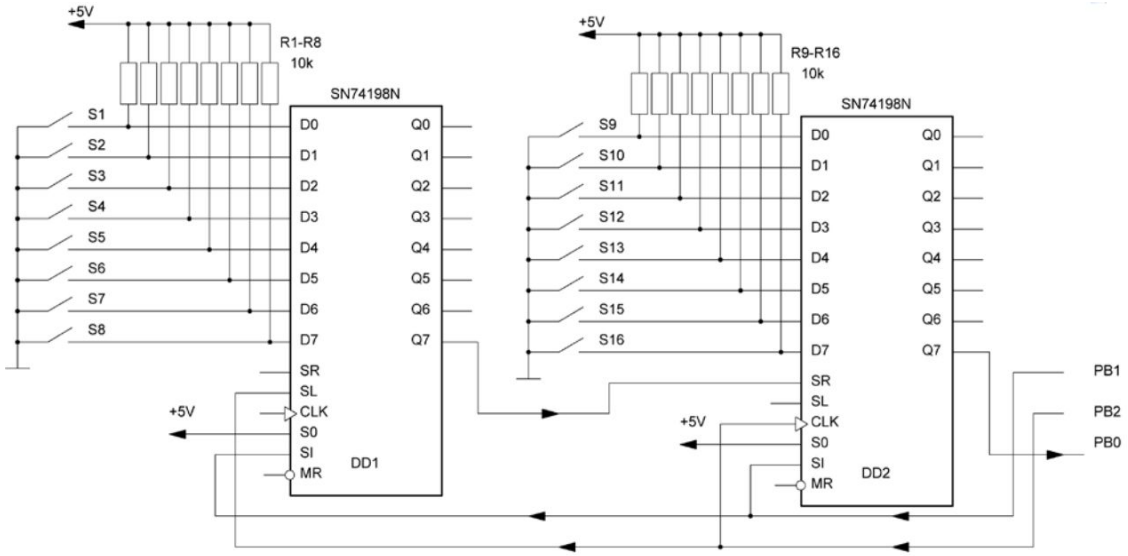


Рисунок 6.32 – Підключення 16 кнопок з використанням регістра зсуву

4. Оптичні давачі

Як оптичний давач найчастіше виступає світлодіод та фотоприймач, який називається оптроном. Випускаються оптрони з закритим (optoisolator) оптичним каналом (у монолітному виконанні) і відкритим оптичним каналом (щілинні і відбивальні оптрони).

4.1. Щілинний оптрон

На рис. 6.33 показаний оптичний давач – щілинний оптрон (slotted optical switch). Фототранзистор і направлений на нього світлодіод закріплені на пластиковій підставці і розділені проміжком так, що коли якийсь предмет рухається у щілині, він перекриває світло між світлодіодом і давачем.

Рисунок 6.33 – Щілинний оптрон

Щілинні оптрони часто використовуються для вимірювання швидкості двигуна за допомогою диска з прорізами, розміщеного на осі двигуна. Коли вісь обертається, диск перекриває світловий промінь. Інше застосування щілинного оптрона – це індикація того, відкриті чи закриті двері або, наприклад, кожух охоронного приладу. Пластина на дверях, потрапляючи в щілину, блокує світло при закриванні дверей. Механічна комп'ютерна миша також використовує щілинні оптрони.

4.2. Відбивальний оптрон

На рис. 6.34 показаний інший тип оптичного датчика – відбивальний оптрон (reflective sensor). Принцип роботи цього давача такий же, як і щілинного, з тією різницею, що фототранзистор приймає відбите, а не пряме світло. Більшість давачів відбиття характеризується фокусною відстанню – оптимальною відстанню, на якій має бути розміщений відбивальний об'єкт від давача. Ця відстань дорівнює 0,254... 1,270 см. Типове застосування відбивальних оптронів – це реєстрація обертання двигуна за нанесеними на його вісь темними позначками.

Коли вісь обертається, давач реєструє зміну темних і відбивальних ділянок. Обидва типи оптронів мають схожі характеристики, які необхідно враховувати при проектуванні систем.

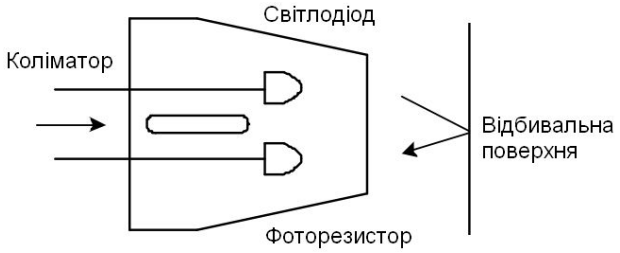


Рисунок 6.34 – Відбивальний оптрон

4.3. Швидкість спрацьовування оптронів

Оскільки фототранзистор є оптичним приладом, що діє досить повільно, то це обмежує максимальну швидкість реєстрації. Типовий час включення фототранзистора 8 мкс, а вимкнення 50 мкс. Ці часові параметри визначаються швидкістю носіїв під дією світла в переході база-емітер транзистора.

4.4. Коефіцієнт підсилення за струмом

Оптопара світлодіод-фототранзистор має обмежений коефіцієнт підсилення, звичайно, менше одиниці. Відношення струму, що протікає в колекторі фототранзистора, до струму через світлодіод називається коефіцієнтом передачі за струмом (КПС) (Current Transfer Ratio, CTR). Типове значення КПС для щілинних оптронів складає 0,1. Це означає, що при струмі 10 мА, що протікає через світлодіод, струм колектора фототранзистора складе 1 мА. КПС інколи задається як коефіцієнт, інколи наводиться у вигляді таблиці, що показує різні значення струмів колектора для різних величин струму світлодіода. КПС залежить від характеристик світлодіода і фототранзистора. КПС слід брати до уваги при створенні інтерфейсу між оптроном і МК системою. По-перше, якщо оптрон приєднується до цифрового входу (рис. 6. 35), вихід транзистора необхідно узгоджувати за логічними рівнями з входом цифрового пристрою.

Для швидкого насичення фототранзистора величину резистора навантаження Rнав. слід обмежити. Наприклад, якщо через світлодіод протікає струм 10 мА, а мінімальна величина КПС дорівнює 0,1; то величину резистора навантаження слід вибрати 5 кОм. Менша величина резистора забезпечить кращу стійкість до шумів (через менший імпеданс) і, можливо, більшу швидкість перемикання, але не гарантує сумісності зі всіма пристроями, оскільки транзистор не зміг би пропускати достатній струм для забезпечення низького логічного рівня. Щоб знизити величину навантажувального резистора, можна використовувати оптопару з більш високим КПС або живити світлодіод великим струмом.

Випускаються оптрони зі складеним транзистором Дарлінгтона на виході, що забезпечує КПС > 1, але швидкість перемикання такого оптрона нижча і складає 20% від швидкості одиночних транзисторів. До того ж напруга насичення транзистора Дарлінгтона більша, ніж одиночного транзистора.

Відбивальні давачі також характеризуються КПС. Оскільки давач розрахований на прийом відбитого світла, то КПС залежатиме від властивостей відбивальної поверхні і відстані до неї. КПС відбивального давача задається відносно деякої стандартної відбивальної поверхні, розміщеної на характерній фокальній відстані.

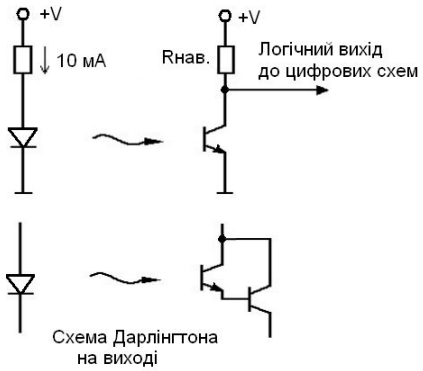


Рисунок 6.35 – Оптрони з цифровим виходом

4.5. Особливості роботи оптронів в інфрачервоному діапазоні

У більшості щілинних і відбивальних давачів використовуються інфрачервоні світлодіоди і фототранзистори. Це означає, що реакція давача може відрізнятися від реакції у видимому діапазоні спектра. Об'єкти, що добре відбивають у видимому світлі, можуть відбивати менш ефективно в інфрачервоних (ІЧ) променях. Слід також враховувати завади, що вносяться природним і штучним освітленням.

На рис. 6.36 показано, що оптрон, який працює з прямокутними імпульсами, може бути обладнаний фільтром для усунення подібних завад.

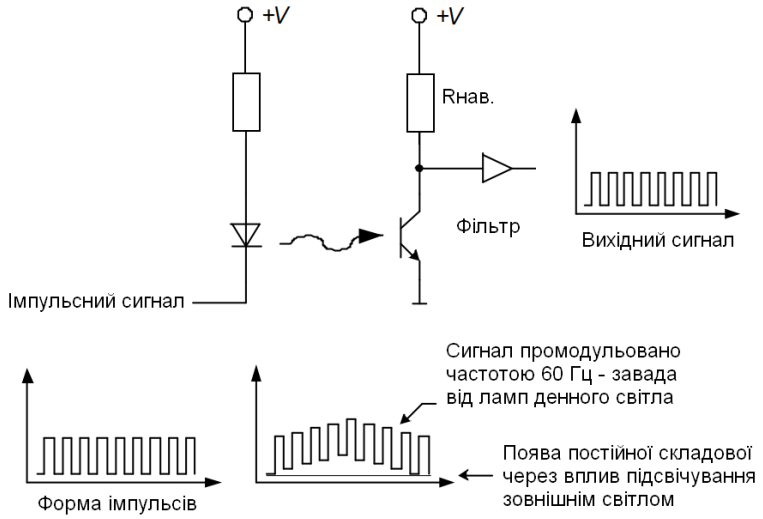


Рисунок 6.36 – Фільтрація сигналу оптичного давача

У даному прикладі освітлювальний прилад додає до вихідного сигналу постійний зсув і коливання з частотою мережі 50 Гц. Після проходження сигналу через фільтр, налаштований на частоту паразитної модуляції мережі, ці завади вирізаються фільтром, і відфільтрований від завад сигнал може бути перетворений в цифровий. Фільтр може бути реалізований як за допомогою електроніки, так і програмно. ІЧ-промені широко застосовуються в побутовій техніці. У пульті дистанційного керування телевізора на ІЧ-променях використовується модуляція сигналу з частотою 40 кГц. Для забезпечення роботи з такою частотою в давачі використовується високошвидкісний фотодіод.

Подібна фільтрація має і недоліки. Один з недоліків – обмеження швидкості. Через значний час вмикання і вимикання фототранзистора частота модуляції, з якою пристрій ще працюватиме, обмежена величиною порядку 10 кГц.

Недолік також в тому, що в результаті фільтрації з'являється деяка затримка в часі, що досягає декількох циклів роботи механічної частини. Так, якщо навіть давач розрахований на роботу з частотою 10 кГц, в цілому система зможе підтримувати не вище 1 кГц, бо розраховується за найповільнішою частотою. Крім того, якщо зовнішнє освітлення має сильний вплив, то фототранзистор насичується і фільтрація не в змозі відновити вихідний сигнал. У даному випадку необхідно захистити фототранзистор від зовнішнього засвічування.

4.6. Дискретні оптичні давачі

У деяких пристроях іноді застосовують дискретний оптичний давач: світлодіод і фототранзистор. Вони розраховані на роботу в інфрачервоному діапазоні, як і ті, що застосовуються в щілинних і відбивальних оптронах. Дискретні оптопари використовуються в системах, де великий об'єкт заступає шлях світловому променю між світлодіодом і фототранзистором, або там, де відстань дуже велика для використання оптрона.

Дискретні оптопари мають аналогічні схеми включення і використовуються так само, як розглянуті раніше оптрони. Проте існує декілька додаткових зауважень. Оскільки відстань між світлодіодом і фототранзистором у такій системі більша, ніж в стандартній ІС, КПС, відповідно, нижчий. Часто з'являється необхідність підлаштувати струм світлодіода або поріг чутливості для надійної роботи схеми. У деяких випадках може знадобитися лінза (об'єктив) на один з елементів оптопари для фокусування світла.

Необхідність фокусування – досить поширена проблема, що виникає при використанні дискретних елементів. Ця проблема буде особливо відчутна, якщо світлодіод і фототранзистор розташовані на різних механічних підставках.

У оптроні світлодіод і фототранзистор узгоджені за довжиною світлової хвилі. Хоча велика частина фототранзисторів і світлодіодів ІЧ-діапазону можуть працювати спільно, але узгодження максимуму їх спектральної чутливості все ж може відрізнятися. Отже, при використанні дискретних елементів вибирають світлодіод і фототранзистор, що працюють в одній і тій же області ІЧ-спектра.

На рис. 6.37 показано три основні способи підключення оптичного давача до МК. У всіх випадках світлодіод підключений через обмежувальний резистор, емітер фототранзистора з’єднується з землею, а колектор підключений до живлення через резистор навантаження і з’єднаний також з МК чи іншим пристроєм – компаратором або АЦП. Всі ці способи також ефективні і при інших підключеннях пристроїв керування, як, наприклад, з використанням МК для вмикання і вимикання світлодіода. Оптрони з закритим оптичним каналом працюватимуть і при іншій схемі підключення фототранзистора, наприклад, не за наведеною вище схемою із спільним колектором, а за іншою – із спільним емітером (живлення позитивної полярності подається на колектор, і сигнал знімається через резистор, що з’єднує емітер з землею).

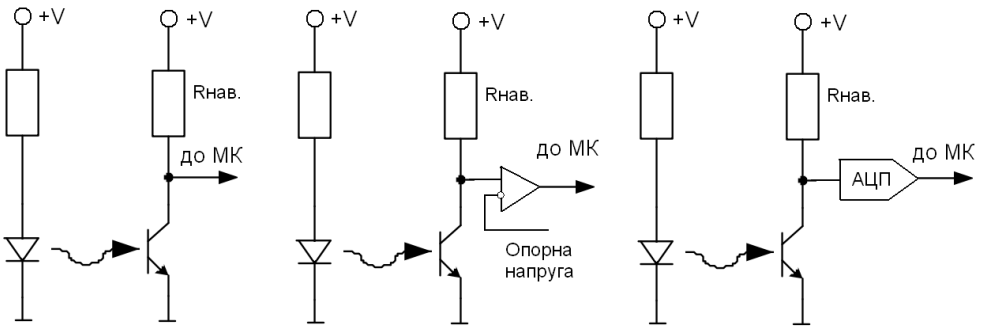


Рисунок 6.37 – Схеми підключення оптичних давачів до МК

На рис. 6.38 показано, як оптрони з закритими оптичними каналами можуть бути використані для розв'язки двонаправленого сигналу між двома системами. Вивід пристрою, що використовує інтерфейс SPI, ізольовується за постійним струмом від МК. Для забезпечення необхідної розв'язки використовуються два оптрони з закритими оптичними каналами. Вихід

SPI-пристрою забезпечений буфером (для забезпечення необхідного струму світлодіода), і вихід оптрона U1 переходить до стану низького рівня, коли SPI-пристрій встановлює на виводі низький логічний рівень. Вихід U1 може бути підключений до порту МК.

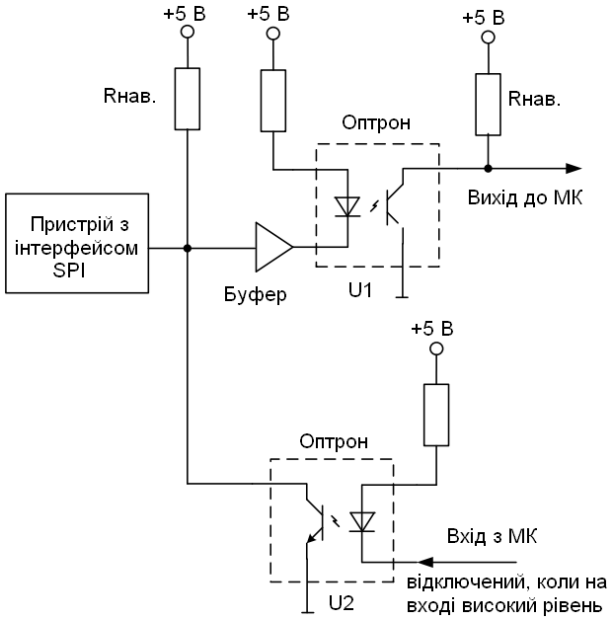


Рисунок 6.38 – Двонаправлена розв’язка з використанням оптронів із закритими оптичними каналами

Другий оптрон (U2) встановлює вивід порту в нуль, коли МК формує на світлодіоді сигнал низького рівня. Якщо МК не передає дані на вивід I/O, він повинен залишити світлодіод у вимкненому стані (OFF state), так, щоб SPI-пристрій міг подавати сигнали на цей вивід. Коли МК встановлює на виводі I/O низький логічний рівень, вихід оптрона U2 також встановиться в низький рівень, щоб МК ігнорував передачі на виході U2 в той час, коли він керує SPI-пристроєм. З іншого боку, може бути використаний у відповідь сигнал (return) для перевірки коректного проходження даних через SPI-пристрій.

На рис. 388не показане коло, доповнене ще однією оптопарою і ще одним портом МК, які знадобляться для формування тактового сигналу інтерфейсу SPI. Для керування двонаправленим виводом потрібно щоб МК використовував два виводи порту (один вхід і один вихід), що дозволяє забезпечити ізоляцію за постійним струмом периферійного блока або системи. У більшості випадків необхідні високошвидкісні оптрони з закритими оптичними каналами. Цілком можливо, що підійдуть діодно-транзисторні оптрони чи оптрони з логічним виходом, але слід пам'ятати, що оптрон U2 (рис. 38), безпосередньо з’єднаний з двонаправленим виводом, повинен мати вихід з відкритим колектором. Для забезпечення достатньої швидкості перемикання слід ретельно підібрати резистор навантаження, щоб уникнути надмірного часу наростання, без перевищення навантажувальної здатності виводу. Якщо як U2 застосовується оптрон з діодно-транзисторним виходом, то він повинен забезпечувати достатній струм світлодіода для гарантованого встановлення виводу катода світлодіода в низький логічний рівень.

Висновки.

Для організації динамічної індикації використовується матриця, що складається з ліній рядків і ліній стовпчиків світлодіодів.

Як більш інформативні індикаторні елементи використовуються семисегментні індикатори. Для динамічної індикації семи сегментних індикаторів використовують схеми без додаткових елементів, з одним додатковим елементом, з двома додатковими елементами.

Для відображення поточних значень вимірюваних величин використовують LCD індикатор з контролером типу HD44780.

За допомогою кнопок та давачів як периферійних елементів в МК поступає різна інформація, яка використовується для зміни алгоритму роботи програми.

Як оптичний давач найчастіше виступає світлодіод та фотоприймач, який називається оптроном. Використовуються як щілинні, так і відбивальні оптрони. У деяких пристроях іноді застосовують дискретний оптичний давач: світлодіода і фототранзистора. Вони розраховані на роботу в інфрачервоному діапазоні. Дискретні оптопари мають аналогічні схеми включення і використовуються так само, як оптрони.

Питання.

1. Переваги і недоліки динамічної індикації?
2. Схема динамічної індикації без додаткових елементів.
3. Схема динамічної індикації з одним додатковим елементом.
4. Схема динамічної індикації з регістром зсуву.
5. Схема динамічної індикації з дешифратором.
6. Схема динамічної індикації з двома додатковими елементами.
7. Схема динамічної індикації з двома лініями керування.
8. Підключення LCD індикатора HD44780 до AVR мікроконтролерів.
9. Структура адресації контролера HD44780.
10. Схеми підключення матричної клавіатури.
11. Мультиплексорне керування матричною клавіатурою.
12. Підключення клавіатури з використанням дешифратора.
13. Клавіатура з одним контактором.
14. Підключення клавіатури з використанням регістра зсуву.
15. Щілинний оптрон.
16. Відбивальний оптрон.
17. Особливості роботи оптронів в інфрачервоному діапазоні.
18. Схеми підключення оптичних давачів до МК.
19. Двонаправлена розв’язка з використанням оптронів із закритими оптичними каналами.

ЛЕКЦІЯ 7. Стек. Підпрограми. Макроси. Переривання і їх оброблення

Мета. Вивчення організації стеку, викликів підпрограм, макросів, внутрішніх і зовнішніх переривань в МК AVR та їх оброблення, організація затримок.

Вступ. Стек неявно використовується підпрограмами та перериваннями для збереження адреси повернення в основну програму. Замість невеликих підпрограм можуть використовуватися макроси. За принципом виклику підпрограм працюють переривання. В МК AVR всі переривання апаратні і діляться на внутрішні і зовнішні. Внутрішні переривання можуть виникнути від любого внутрішнього функціонального блоку: таймерів, аналогового компаратора, послідовного порту. Внутрішнє переривання – це подія, яка виникає в системі і перериває виконання основної програми.

План.
1. Стек
2. Підпрограми
3. Макроси
4. Переривання і їх оброблення
5. Внутрішні і зовнішні переривання
6. Оброблення подій у нескінченому циклі та затримки
7. Приклад зовнішніх переривань

1. Стек

Стек розміщується в оперативній пам’яті SRAM. Він реалізований як LIFO-буфер (last input first output, останній увійшов – перший вийшов). Початок стеку розміщується у кінцевій адресі SRAM і по мірі його заповнення він збільшується (адреси зменшуються) в напрямку початку SRAM. Керується стек спеціальним 2-байтовим регістром SP, що складається з регістрів SPH (старший байт) і SPL (молодший байт). Деякі молодші моделі мають тільки регістр SPL. Значення стеку за замовчуванням рівне 0x0000, тому стек необхідно проініціалізувати вручну. Розміщення початку стеку може бути довільним, однак, якщо його розмістити доволі близько до оперативних даних, тобто ближче до початку SRAM, тоді, по мірі збільшення стеку, можуть перекритися дані у SRAM, що призведе до їх спотворення та не коректної роботи програми. Тому стек розміщують у кінцевій адресі SRAM:

```
ldi r16,Low(RAMEND)
out SPL,r16
ldi r16,High(RAMEND)
out SPH,r16
```

Значення константи `RAMEND` залежить від моделі МК та зберігається у конфігураційному файлі цієї моделі.

Стек неявно використовується підпрограмами при виклику команд `call, rcall, icall, ret, reti` та перериваннями для збереження адреси повернення в основну програму. Також стек використовується для тимчасового зберігання значень регістрів для звільнення їх для іншого використання.

Для роботи зі стеком призначені дві команди: `push` (push register on stack, виконується за 2 такти) – заносить значення регістра загального призначення у стек; `pop` (pop register from stack,

виконується за 2 такти) – видобуває значення зі стеку у регістр загального призначення; Ці команди працюють лише через регістри загального призначення, і якщо необхідно зберегти у стеку значення регістра введення/виведення, то його перед тим необхідно скопіювати у регістр загального призначення.

При використанні стеку необхідно контролювати логіку роботи з ним, тобто порядок і кількість завантажених байтів у стек та порядок і кількість видобутих байтів зі стеку. При порушенні такої логіки відбувається зміщення лічильника стеку, і можна переписати збережені адреси повернення підпрограми, а якщо це відбувається циклічно, то перезаписати усі дані в SRAM.

На рис. 7.1 показано приклад явного використання стеку для обміну значень двох регістрів без використання третього проміжного регістра.

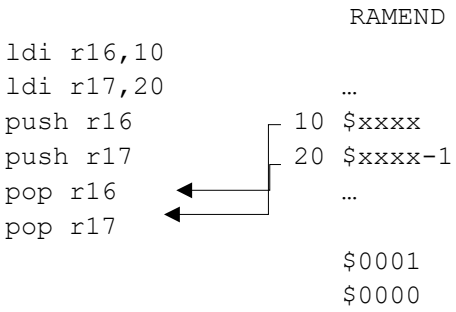


Рисунок 7.1 – Обмін значень двох регістрів через стек

2. Підпрограми

Підпрограми це функціонально закінчені частини програмного коду, які часто використовуються. Підпрограми викликаються однією з наступних команд:

`call` – викликає підпрограми в межах всієї програмної пам'яті (до 8 Мбайт). Команда має розмір 4 байти (32 біти), 22 біти з яких відведені під адресу, однак, для переважної більшості моделей програмний лічильник займає 16 біт, тому команда може адресувати для цих моделей лише до 128 Кбайт програмної пам'яті FLASH. Команда виконується за 4 такти.

`rcall` – відносний виклик підпрограми, здійснює перехід у програмній пам'яті на вказане число (-4095… +4096) відносно значення поточної адреси (програмного лічильника. Команда виконується за 3 такти.

`icall` – непрямий виклик підпрограми за адресою регістра Z. 16-бітовий регістр Z дозволяє адресувати 128 Кбайт програмної пам'яті. Команда виконується за 3 такти.

При виклику підпрограми за допомогою однієї з команд `call` у стек автоматично заноситься адреса (2 байти) наступної команди після команди виклику. Виконання підпрограми завершується командою `ret` (4 такти), яка присвоює програмному лічильнику значення адреси повернення, що видобувається із стеку.

Для команд виклику адреси підпрограм вказують за допомогою позначок, а асемблер вже сам обчислює потрібні значення.

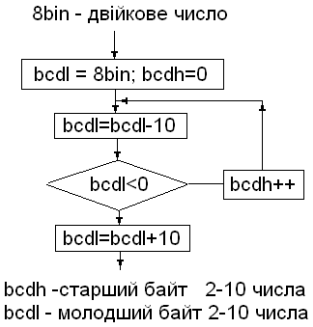
При використанні підпрограм необхідно обов'язково проініціалізувати стек (!).

```
.include "m8515def.inc"
.def  _8bin    =r18
.def  _bcdlow  =r18
.def  _bcdhigh =r19
```

```
        .CSEG
        ;Ініціалізація стеку
        ldi  r16, Low(RAMEND)
        out  SPL, r16
        ldi  r16, High(RAMEND)
        out  SPH, r16

        .org  $100
; $100
        ldi  _8bin,35 ;r18=35
; $101
        rcall  BCD    ;$102 -> STACK; PC=$150
; $102
        nop           ;r19=3; r18=5
; $103
        ldi _8bin,71  ;r18=71
; $104
        rcall BCD     ;$105 -> STACK; PC=$150
; $105
        nop           ;r19=7; r18=1
; програма двійково-десяткового кодування до 99 (незапаковане)
```

```
         .org  $150
  BCD:   clr  _bcdhigh
  BCD1:  subi _bcdlow, 10
         brcs  BCD2
         inc  _bcdhigh
         rjmp  BCD1
  BCD2:   subi  _bcdlow, -10
         ret  ; PC < STACK
```



3. Макроси

Макроси, на відміну від підпрограм, не мають окремого адресного розміщення в програмі. В макроси виносяться невеликі, часто повторювані частини програмного коду. Програмний код макросу вставляється під час асемблювання у точку виклику. Кожний виклик макросу збільшує обсяг програмної пам’яті, в той же час як виклики підпрограм – не збільшують.

Макроси можуть зберігатися в окремих файлах. Приклад використання макросу для швидкого множення беззнакового однобайтного числа на 10 (0b0000_1010):

```
.macro   R16mult10    ;r20 = r20*10 = (r20<<1)+ (r20<<3)
  lsl  r20
  mov  _temp, r20
  lsl  r20
  lsl  r20
  add  r20, _temp
.endmacro

  0010 (2)
  0100
 10000
+--------
```

```
10100 (20)

.def  _temp = r16
  .CSEG
  …
  ldi  r20, 14    ;r20 = 14
  R16mult10       ;r20 = r20*10 = 140
  subi  r20, 121  ;r20 = r20 – 121 = 19
  R16mult10       ;r20 = r20*10 = 190
  subi  r20, 180  ;r20 = r20 – 180 = 10
  R16mult10       ;r20 = r20*10 = 100
  …
```

Макроси можуть мати вхідні параметри. Як параметри, у макрос можна передавати константи або назви регістрів. Макрос може приймати до 10 параметрів. Посилання на ці параметри позначаються в середині макросу як `@0-@9`. Порядок слідування визначається при виклику макросу. При асемблюванні, код макросу вставляється у точці виклику і замість параметризованих змінних підставляються фактичні значення, вказані через кому після імені викликуваного макросу.

Приклад з трьома макросами, що спрощує процес програмування і покращує візуальне сприйняття самого коду.

```
.include "m8515def.inc"
.def  _temp = r16
.macro  outi
    ldi  _temp, @1
  .if @0 < 0x40
    out  @0, _temp
  .else
    sts  @0, _temp
  .endif
.endm

.macro  addi
  ldi  _temp, @1
  add  @0, _temp
.endm

.macro  ldi2
  ldi  _temp, @1
  mov  @0, _temp
.endm

.CSEG
outi  DDRA, 0xFF        ; DDRA  = 0b1111_1111
outi  PORTA, 0x00       ; PORTA = 0b0000_0000

outi  DDRB, 0x00        ; DDRB  = 0b0000_0000
outi  PORTB, 0b01010101 ; PORTB = 0b0101_0101

ldi2  r0, 50    ;r0 = 50
addi  r0, 25    ;r0 = r0 + 25

ldi2  r1, 50    ;r1 = 50
```

```
addi  r1, -25   ;r1 = r1 + -25 = 25
```

Перший макрос `outi` використовує умовну компіляцію, що дозволяє виводити константу у будь-який регістр введення/виведення. У деяких моделях МК, з великою кількістю периферії, частина регістрів введення/виведення розміщується у пам'яті даних після адреси 0x005F, для якої не працюють команди `out` та `in`. При роботі з такими регістрами необхідно напряму звертатися до SRAM через відповідні команди доступу до пам'яті. Тому макрос `outi` спрощує задачу запису константи у регістр введення/виведення так, як не потрібно слідкувати в якій області пам'яті знаходиться цей регістр.

Другий макрос `addi` реалізує команду додавання до будь-якого регістра загального призначення константи.

Третій макрос `ldi2` присвоює константу регістру загального призначення.

4. Переривання і їх оброблення

Основним завданням МК є очікування і оброблення деяких подій. Найбільш *універсальний спосіб* оброблення подій – організація основної програми у виді нескінченного циклу. Всередині цього циклу деяким способом відслідковується поява подій. При виникненні події встановлюється значення одного біту (прапора) у відведеному регістрі, комірці пам'яті або навіть безпосередньо стану виводу МК. Основна програма в циклі перевіряє значення потрібних прапорів, і при зміні одного з них переходить до відповідної процедури обробки події.

Як альтернатива універсальному способу відслідкування подій в сучасних МК використовуються *апаратні переривання* (interrups). При виникненні деякої події тут також встановлюються деякі біти прапори, але перехід до їх оброблення здійснюється апаратно. Програмісту потрібно лише написати підпрограму оброблення переривань. У деяких випадках основна програма може містити один рядок нескінченого циклу – `loop: rjmp loop`, все інше буде виконуватися через переривання.

У МК AVR переривання реалізують механізм оброблення подій від вбудованих периферійний пристроїв. Оскільки моделі AVR відрізняються одна від одної кількістю та різноманітністю вбудованих пристроїв, то відповідно, і кількість переривань у них є різною. Основною відправною точкою механізму переривань є таблиця векторів переривань. Ця таблиця є послідовним списком адрес у програмній пам'яті (табл. 7.1). Як правило, ця таблиця розміщується на початку адресного простору пам'яті програм. Інформацію про наявні переривання та адреси їхніх векторів переривань можна отримати в інструкції виробника на конкретну модель.

Таблиця 7.1 – Таблиця векторів переривань для ATmega8515

Номер вектора	Адреса програми	Джерело переривання	Описання переривання
1	$000	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog Reset
2	$001	INT0	External Interrupt Request (IRQ0)
3	$002	INT1	External Interrupt Request (IRQ1)
4	$003	TIMER1 CAPT	Timer/Counter1 Capture Event
5	$004	TIMER1 COMPA	Timer/Counter1 Compare Match A

6	$005	TIMER1 COMPB	Timer/Counter1 Compare Match B
7	$006	TIMER1 OVF	Timer/Counter1 Overflow
8	$007	TIMER0 OVF	Timer/Counter0 Overflow
9	$008	SPI, STC	Serial Transfer Complete
10	$009	USART, RXC	USART, Rx Complete
11	$00A	USART, UDRE	USART Data Register Empty
12	$00B	USART, TXC	USART, Tx Complete
13	$00C	ANA_COMP	Analog Comparator
14	$00D	INT2	External Interrupt Request (IRQ2)
)15	$00E	TIMER0 COMP	Timer/Counter0 Compare Match
16	$00F	EE_RDY	EEPROM Ready
17	$010	SPM_RDY	Store Program memory Ready

Для того, щоб певна подія для вбудованого пристрою МК могла бути згенерованою, необхідно перше за все активізувати цей периферійний пристрій та дати дозвіл на переривання для цієї події, а також дати загальний дозвіл на переривання у регістрі стану SREG.

Прапори переривань утворюють ієрархію. Зверху ієрархії знаходиться біт I регістра SREG, який дозволяє (логічна 1, команда `sei`) або забороняє (логічний 0, команда `cli`) любі апаратні переривання. За замовчуванням прапор I очищений і при запуску МК переривання заборонені. Для кожного конкретного переривання є свій дозволяючий/забороняючий біт у відповідному регістрі (наприклад, для таймерів це регістри TIMSK, для зовнішніх пристроїв – GIMSK). При використанні переривань ці біти потрібно встановлювати в стан логічної 1.

Загальна схема обробки переривань наступна. При виникненні любого переривання апаратно очищається прапор I регістра SREG. Він автоматично встановлюється знову, коли закінчиться оброблення переривань (за командою `reti`). Цей прапор можна встановлювати програмно в програмі-обробнику (безпосередньо або командою `sei`), дозволивши вкладені переривання. Після скидання прапора I МК визначає, запит на оброблення якого саме переривання відбувся, – це встановлюється за прапором конкретного переривання (для таймерів – в регістрі TIFR, для зовнішніх переривань – в регістрі GIFR). Після визначення типу переривання МК автоматично обчислює адресу відповідного вектора переривань, які звичайно розміщуються в початкових адресах пам’яті програм. За цією адресою має знаходитися команда `rjmp` (або `jmp` для МК з розміром пам’яті 16 Кбайт і більше), яка містить адресу підпрограми обробника.

Перед тим як перейти за вектором переривання, МК скидає прапори переривання і автоматично зберігає вміст лічильника команд у стеку. Тому для успішного виклику переривань в програму ініціалізації МК необхідно добавити наступні рядки (їх звичайно ставлять після позначки, на яку здійснюється перехід за вектором скидання RESET)

```
RESET:
ldi temp,low(RAMEND)    ; завантаження вказівника стеку
out SPL, temp
ldi temp,high(RAMEND)   ; завантаження вказівника стеку
out SPL, temp
. . .    ; інші команди
sei      ; дозвіл переривання
```

Підпрограма обробник повинна закінчуватися командою виходу з переривання `reti`. За цією командою відновлюється лічильник команд і знову встановлюється загальний прапор дозволу переривань.

У випадку, якщо подія трапилася, виставився для неї відповідний прапор переривання, але загальний дозвіл ще не наданий (в регістрі стану SREG), тоді ця подія буде оброблена при наданні відповідного загального дозволу переривання.

Слід зазначити, що для деяких подій немає відповідних прапорів переривань. Для них переривання генеруються протягом усього часу, поки присутня відповідна умова, що необхідна для генерації переривання. Відповідно, якщо умови, що викликають переривання, зникнуть до надання дозволів на переривання, то генерація переривання не відбудеться.

Якщо переривання генерується безперестанно, тобто є постійна умова для безпрапорних переривань або в черзі стоять переривання з виставленими прапорами, то між генеруваннями переривань виконується одна команда з основного коду.

5. Внутрішні і зовнішні переривання

МК ATmega8515 може обробляти 3 зовнішні і 14 внутрішніх переривань.

Зовнішні переривання це реакція МК на зміну рівня сигналу на його виводах. Такі переривання бувають індивідуальними (INT), тобто для конкретного одного виводу МК, та груповими (PCINT), тобто одне переривання для групи виводів. МК серії tiny мають як мінімум хоча б одне INT0, МК серії mega мають як мінімум два INT0 та INT1. Групові переривання (PCINT) вже залежать від конкретної моделі, і тому можуть бути або не бути.

Індивідуальні зовнішні переривання (INT) можуть генеруватися при наявності зростаючого чи спадаючого фронту сигналу, або постійного низького рівня на виводі МК (рис. 7.2). При налаштуванні на низький рівень прапори переривання не виставляються та переривання генеруються поки присутня умова низького рівня.

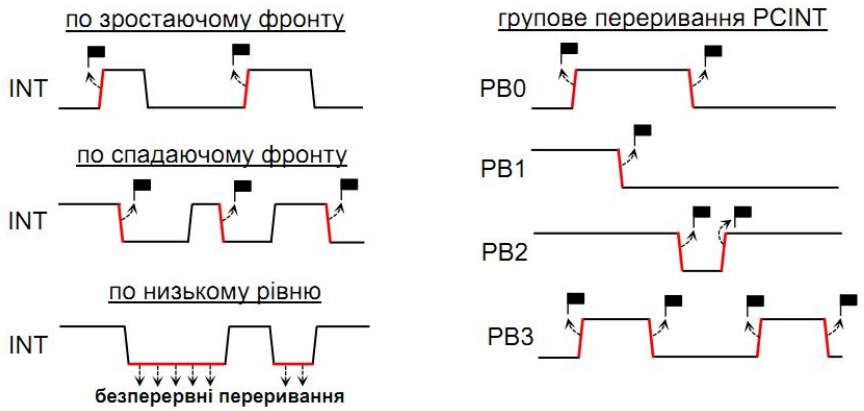


Рисунок 7.2 – Діаграми генерування зовнішніх переривань

Групове зовнішнє переривання (PCINT) охоплює, як правило, виводи цілого порту. При цьому можна індивідуально вибирати, які виводи порту будуть генерувати переривання, а які ні. Генерація переривання буде при будь-якій зміні рівня сигналу (при зростанні та спаданні фронтів) на будь-якому виводі групи PCINT.

МК ATmega8515 має 3 виводи для зовнішніх переривань INT0, INT1, INT2. INT0 та INT1 можуть бути налаштовані на переривання зростаючого/спадаючого фронту сигналу і низького рівня сигналу, а INT2 – лише на реагування зростаючого чи спадаючого фронту сигналу.
Для роботи із зовнішніми перериваннями в ATmega8515 використовуються 4 регістри:

GICR (задаються дозволи на переривання INT0, INT1 та INT2);

GIFR (виставляються прапори при виникненні переривань);

MCUCR (налаштовуються умови для переривань INT0 та INT1);

MCUCSR (налаштовується умова для переривання INT2).

Дозволи на зовнішні переривання задаються в загальному регістрі прапорів переривань GICR:

7	6	5	4	3	2	1	0
INT1	INT0	INT2	-	-	-	IVSEL	IVCE

Як входи зовнішніх переривань використовуються входи портів з альтернативною функцією: PD2, PD3 – для переривань INT0, INT1. Запити зовнішніх переривань INT0, INT1 можуть бути подані низьким рівнем сигналу переривання (L), переходом від високого рівня сигналу до низького (HL – за спадаючим фронтом), переходом від низького рівня сигналу до високого (LH – за наростаючим фронтом), запит INT2 тільки переходами (LH) і (HL).

В регістрі керування MCUCR задаються біти для умов переривань:

7	6	5	4	3	2	1	0
SRE	SRW10	SE	SM1	ISC11	ISC10	ISC01	ISC00

В залежності від типу запиту в регістрі управління МК MCUCR необхідно встановити біти ISC11, ISC10l і ISC01, ISC00 для кожного з переривань INT1, INT0 (табл. 1, x=0,1) та визначити біт ISC2 в регістрі EMCUCR для переривання INT2 (при ISC2 = 0 переривання здійснюється за спадаючим фронтом, при ISC2 = 1 – за наростаючим фронтом).

Таблиця 7.2 – Рівні і фронти сигналів переривань

ISCx1	ISCx0	Тип
0	0	L
0	1	-
1	0	HL
1	1	LH

Режими переривання за рівнем (L) в INT0/INT1 і фронтами (HL, LH) в INT2 детектуються асинхронно. Режими переривання за фронтом (HL, LH) в INT0/INT1 детектуються тільки синхронно – в момент перепаду рівнів тактового сигналу МК. Тому асинхронне зовнішнє переривання використовується для виведення в робочий режим МК, який перебуває в стані енергозбереження.

Усі моделі МК AVR мають два базових режими енергозбереження: Idle mode і Power Down mode. В режимі Idle mode (режим очікування) зупиняється основний процесорний модуль, а всі периферійні пристрої (таймери, АЦП, порти) продовжують функціонувати. У цьому режимі споживання енергії зменшується тільки на 30-50%.

У режимі Power Down mode (режим “глибокого” енергозбереження) зупиняються всі вузли МК, за винятком сторожового таймера (якщо він включений), системи оброблення асинхронних зовнішніх переривань і модуля TWI. Вихід з цього режиму можливий за скиданням МК або сторожового таймера, або переривання від TWI. Споживання в цьому режимі може становити до декількох десятків мікроампер.

6. Оброблення подій у нескінченому циклі та затримки

Оброблення подій у нескінченому циклі може використовуватися для підрахунку натискання кнопки і висвітлення загальної кількості натискань у двійковому коді на світлодіодах. У програмі враховується брязкіт кнопок. При натисканні або відпусканні кнопки генерується множина імпульсів, з яких потрібно вибрати один. Звичайно реагують не на натискання кнопки, а на відпускання. Послідовність виявлення відпускання кнопки наступна: спочатку виявляється натискання кнопки (низький рівень на виводі PINx), робиться перша пауза для пропуску брязкоту (0,2 сек), і потім виявляється відпускання (перший високий рівень на виводі PINx). Після виявлення відпускання, наприклад збільшується лічильник натискань і робиться друга пауза проти брязкоту (0,5 сек) і цикл повторюється спочатку. Величини 0,2 і 0,5 уточнюються в залежності від реальної конструкції кнопок.

Для відліку тривалості пауз необхідний імітатор таймера. Для формування часового інтервалу без таймера можна використати тривалість машинного такту. В МК AVR більшість команд виконується за один машинний такт. Тому для формування інтервалу в 1 сек при тактовій частоті 4 МГц потрібно виконати послідовність з чотирьох мільйонів однотактних команд. Для формування часових інтервалів використовуються програмні затримки з використанням команд декременту. Наприклад в три регістри записується одне число Byte2, Byte1, Byte0. Потім послідовно зменшується молодший байт Byte0, коли його величина досягне нуля, зменшується на одиницю наступний байт Byte1 і здійснюється перехід на зменшення молодшого байту із значення 0 в 255 (воно отримується автоматично при відніманні 1 від 0). Такі ітерації продовжуються до тих пір, поки значення всіх байтів стануть нульовими. Послідовність команд, яка реалізує цей алгоритм показана в табл. 2, варіант 1. Недолік такої реалізації в тому, що кількість тактів є змінною, так як команда *brne* виконується за один такт, якщо умова не виконується і за два такти – якщо виконується. Реалізація з постійною кількістю тактів показана у варіанті 2, для якого число N, яке відповідає потрібному інтервалу часу T (сек), при тактовій частоті F (Гц), можна отримати за такою формулою $N=T\cdot F/5$. Так, при T=1 сек, F=4 МГц, N=1·4000000/5=800000=0x0c3500. Отже, для варіанту 2 число 0x0c3500 забезпечить затримку в 1 сек (`Byte0=00, Byte1=0x35, Byte2=0x0c`).

Таблиця 7.3 – Програмні затримки

Варіант 1	Тактів	Варіант 2	Тактів
`Delay:`		`Delay:`	
`dec Byte0`	1	`subi Byte0,1`	1
`brne Delay`	1,2	`sbci Byte1,0`	1
`dec Byte1`	1	`sbci Byte2,0`	1
`brne Delay`	1,2	`brcc Delay`	2
`dec Byte2`	1		
`brne Delay`	1,2		
К-ть тактів	6÷9		5

Приклади підпрограм, які реалізують затримки кратні мілі- і мікросекундам.

```
; ======================== Time Delay Subroutines ====================
; Name:     delayYx1mS
; Purpose:  provide a delay of (YH:YL) x 1 mS
; Entry:    (YH:YL) = delay data
; Exit:     no parameters
; Notes:    16-бітовий регістр забезпечує затримку до 65.535 сек
;           потрібно delay1mS

delayYx1mS:
    call  delay1mS    ; затримка на 1 мсек
    sbiw  YH:YL, 1    ; корегування лічильника затримки
    brne  delayYx1mS  ; лічильник не нульовий

; попадає сюди, коли лічильник затримки нульовий
    ret

; ---------------------------------------------------------------------------
; Name:     delayTx1mS
; Purpose:  provide a delay of (temp) x 1 mS
; Entry:    (temp) = delay data
; Exit:     no parameters
; Notes:    8-бітовий регістр забезпечує затримку до 255 мсек
;           потрібно delay1mS

delayTx1mS:
    call  delay1mS     ; затримка на 1 мсек
    dec   temp         ; корегування лічильника затримки
    brne  delayTx1mS   ; лічильник не нульовий
; попадає сюди, коли лічильник затримки нульовий
    ret

; ---------------------------------------------------------------------------
; Name:     delay1mS
; Purpose:  provide a delay of 1 mS
; Entry:    no parameters
; Exit:     no parameters
; Notes:    розбирає fclk/1000 цикли годинника (включно з 'call')

delay1mS:
    push  YL                          ; [2] зберігає регістри
    push  YH                          ; [2]
    ldi   YL, low (((fclk/1000)-18)/4) ; [1] лічильник затримки
    ldi   YH, high(((fclk/1000)-18)/4) ; [1]

delay1mS_01:
    sbiw  YH:YL, 1                    ; [2] корегує лічильник затримки
    brne  delay1mS_01                 ; [2] лічильник затримки не нульовий
                                      ; iter clk = 4
; попадає сюди, коли лічильник затримки нульовий
    pop  YH                           ; [2] відновлення регістрів
    pop  YL                           ; [2]
    ret                               ; [4]
```

```
                              ; serial clk = 14 + 4 (call) = 18
    ; ---------------------------------------------------------------------------
    ; Name:     delayTx1uS
    ; Purpose:  provide a delay of (temp) x 1 uS with a 16 MHz clock frequency
    ; Entry:    (temp) = delay data
    ; Exit:     no parameters
    ; Notes:    8-бітовий регістр забезпечує затримку до 255 мк сек
    ;           потрібно delay1uS

    delayTx1uS:
        call delay1uS                       ; delay for 1 uS
        dec  temp                           ; decrement the delay counter
        brne delayTx1uS                     ; counter is not zero

    ; попадає сюди, коли лічильник затримки нульовий
        ret

    ; ---------------------------------------------------------------------------
    ; Name:     delay1uS
    ; Purpose:  забезпечує затримку 1 мк сек з частотою годинника 16 МГц
    ; Entry:    no parameters
    ; Exit:     no parameters
    ; Notes:    додати інші push/pop для годинника з частотою 20 МГц

    delay1uS:
         push temp ; [2] ці команди не роблять нічого, а тільки займають такти
годинника
        pop  temp ; [2]
        push temp ; [2]
        pop  temp ; [2]
        ret       ; [4]
                  ; [4] call
        ; разом clk 16
    ; ============================ End of Time Delay Subroutines ==============
```

7. Приклад зовнішніх переривань

Для імітації зовнішніх переривань можуть бути використані кнопки, рис. 7.3.

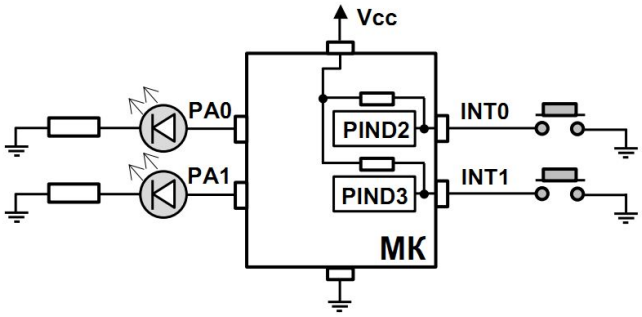


Рисунок 7.3 – Використання переривань для індикації світлодіодів

Підключимо до виводів INT0 та INT1 кнопки, а самі виводи налаштуємо на вхід з підключеними внутрішніми підтягуючими резисторами. INT0 налаштуємо на переривання за

спадаючим фронтом сигналу на виводі, а INT1 – на переривання за зростаючим фронтом сигналу. Переривання повинні інвертувати сигнали на виходах, відповідно, PA0 та PA1 порту A (світлодіоди будуть засвічуватися/гаснути).

```
.include "m8515def.inc"
    .CSEG
    .org  $000
    rjmp   RESET     ;Reset обробник
    .org  $001
    rjmp   EXT_INT0  ;IRQ0 обробник
    .org  $002
    rjmp   EXT_INT1  ;IRQ1 обробник

    .org  $010
    reti            ;Зберігає обробник готовності читання пам'яті
;-------- Підпрограми переривань --------
EXT_INT0:
    sbic  PORTA, 0
    rjmp  elsePA0
    sbi  PORTA, 0
    reti
elsePA0:
    cbi  PORTA, 0
    reti

EXT_INT1:
    sbic  PORTA, 1
    rjmp  elsePA1
    sbi  PORTA, 1
    reti
elsePA1:
    cbi  PORTA, 1
    reti
;-----------------------------------------------------
    ;ініціалізація стеку
RESET:
    ldi  r16, Low(RAMEND)
    out  SPL, r16
    ldi  r16, High(RAMEND)
    out  SPH, r16

    ldi  r16, 0x00
    ldi  r17, 0xFF

    out  DDRA, r17     ;Порт A на вихід
    out  PORTA, r16

    out  DDRD, r16     ;Порт D на вхід
    out  PORTD, r17    ;Підтягуючі резистори

    ;Зовнішні переривання INT0, INT1
    ldi  r16, (1<<INT0)|(1<<INT1)  ;вибір переривань
    out  GICR, r16
```

```
    ldi  r16, (1<<ISC01) | (0<<ISC00)  |  (1<<ISC11) | (1<<ISC10)
    out  MCUCR, r16  ;INT0-за спад. фронтом; INT1- за зрост. фронтом

    ;скидування прапорів зовнішнього переривання
    ldi  r16, (0<<INTF0) | (0<<INTF1)
    out  GIFR, r16

    ;загальний дозвіл на переривання
    sei

main:
    rjmp  main
```

У програмі не використовуються усі наявні для вказаної моделі МК переривання, тому на початку коду у таблиці переривань записані лише вектори рестарту МК та тих переривань, які будуть використовуватися. Однак правилом хорошого тону є записувати також останній у таблиці вектор переривання, що позначає кінець таблиці. Для останнього вектора немає підпрограми, а лише команда виходу з переривання. Це гарантує, що не буде записуватися на місці таблиці переривань прикладний програмний код.

У таблиці переривань можна використати не абсолютну команду переходу `jmp`, а відносну `rjmp`, якщо є впевненість, що підпрограми переривань знаходяться на віддалі 2 кбайт. Це зекономить по 1 такту.

При використанні переривань, потрібно обов'язково проініціалізувати стек, оскільки він використовується для збереження адреси повернення.

Ще один момент стосується регістру стану SREG. Якщо програма переривання використовує команди, що виставляють прапори в регістрі стану, тоді перед початком роботи в перериванні необхідно зберегти SREG у стек, а при виході із переривання, відновити його значення зі стеку. Це необхідно зробити, оскільки переривання вклинюються в роботу основної програми, і може трапитися так, що командою порівняння операндів `cp` виставляються прапори, і за їх значенням потрібно перейти за допомогою команд `branch`. Якщо програма перейде на переривання, в якому її підпрограма переривань змінить прапори у регістрі стану SREG, і тому при поверненні в основну програму буде спотворена логіка для умовного переходу. Тому при перериваннях необхідно зберігати SREG у стеку.

```
Interrupt_subroutine:
  in r16,SREG   ;рег.заг.призн. ← SREG
  push r16      ;рег.заг.призн. → стек
  ...
  pop r16       ;рег.заг.призн. ← стек
  out SREG,r16  ;рег.заг.призн. → SREG
  reti          ;вихід з переривання
```

Висновки.

Стек неявно використовується підпрограмами та перериваннями для збереження адреси повернення в основну програму. Підпрограми це функціонально виділені частини програмного коду, які часто використовуються. Замість невеликих за розміром підпрограм можуть використовуватися макроси.

Основним завданням МК є очікування і оброблення деяких подій. Для відслідкування подій у сучасних МК використовуються *апаратні переривання* (interrups). У МК AVR переривання реалізують механізм оброблення подій від вбудованих периферійний пристроїв. Основною відправною точкою механізму переривань є таблиця векторів переривань, яка розміщується на початку адресного простору пам'яті програм.

Для того, щоб певна подія для вбудованого пристрою МК могла бути згенерованою, необхідно перше за все активізувати цей периферійний пристрій та дати дозвіл на переривання для цієї події, а також дати загальний дозвіл на переривання у регістрі стану SREG.

Прапори переривань утворюють ієрархію. Зверху ієрархії знаходиться біт I регістра SREG, який дозволяє (логічна 1, команда `sei`)) або забороняє (логічний 0, команда `cli`) любі апаратні переривання.

Для роботи зі зовнішніми перериваннями в МК AVR використовуються регістри GICR, GIFR, MCUCR, MCUCSR.

Питання.

1. Стек. Розміщення у пам'яті і команди роботи із стеком.
2. Команди для виклику підпрограм.
3. Створення і виклик макросів, передача їм параметрів.
4. Яка різниця між універсальним і апаратним оброблення подій?
5. Для чого призначена і де розміщується таблиця вектора переривань?
6. Як задаються дозволи на переривання?
7. При яких змінах сигналу можуть відбуватися переривання?
8. Які виводи призначені для оброблення зовнішніх переривань?
9. Яке призначення регістрів GICR, GIFR?
10. Яке призначення регістрів MCUCR, MCUCSR?
11. Як організовуються програмні затримки?

ЛЕКЦІЯ 8. Пам’ять SRAM, FLASH, EEPROM

Мета. Вивчення особливостей програмування пам’яті SRAM, FLASH, EEPROM. Режими програмування МК

Вступ. МК AVR мають три типи пам’яті – даних SRAM, програм FLASH та EEPROM. Змінні розміщуються у пам’яті даних SRAM, константи у пам’яті FLASH і EEPROM, а програми – в пам’яті FLASH. МК мають засоби для обмеження можливостей програмування і зберігання даних. Для налаштування режимів програмування МК використовуються fuse-біти.

План.

1. Робота з внутрішньою пам’яттю даних SRAM
2. Робота з пам’яттю програм FLASH
3. Робота з пам’яттю EEPROM
4. Обмеження можливостей програмування і зберігання даних
4.1. Розряди запобігання RCEN, FSTRT, SPIEN
4.2. Байти сигнатури
4.3. Процес програмування
4.4. Паралельний режим програмування
4.5. Послідовний режим програмування
5. Налаштування режимів програмування МК
6. Налаштування завантажувача

1. Робота з внутрішньою пам’яттю даних SRAM

Змінні у програмах на асемблері розміщується у регістрах загального призначення та стеку. Так як кількість регістрів загального призначення обмежена, то потрібно змінні з деяких регістрів зберігати у стеку, звільняючи місце для інших змінних. Однак при великій кількості змінних і почерговому використанні регістрів потрібна складна логіка для контролю за переміщенням змінних в стек і зі стеку. Тому при нестачі регістрів змінні можна розміщувати в пам’яті даних SRAM. Звичайно, для роботи із змінними в SRAM, буде необхідно завантажувати їх в регістри загального призначення, модифікувати і записувати назад в пам’ять даних. Для цього слід передбачити декілька регістрів, що будуть використовуватися як тимчасові змінні.

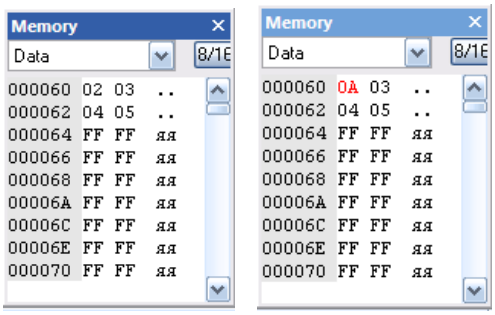
Внутрішня пам'ять даних SRAM є енергозалежною, організована в одному адресному просторі з 32 регістрами загального призначення і 64 регістрами введення/виведення та має розмір 512 байт. При необхідності може використовуватися зовнішня пам'ять розміром 64 кБайт даних на окремій ІС. Доступ зо зовнішньої пам’яті даних SRAM здійснюється за допомогою тих самих інструкцій, що і для внутрішньої пам’яті даних. Читання або запис у зовнішню пам’ять SRAM даних задається виводами PD6 (WR’) і PD7(RD’). При роботі з внутрішньою пам’яттю SRAM ці виводи не активні. Дозвіл на роботу із зовнішньою пам’яттю SRAM задається бітом SRE регістра MCUCR.

Для розміщення змінних в SRAM потрібно спочатку зарезервувати під них пам'ять за допомогою директиви `.byte`:

```
.DSEG
var1:  .BYTE 1      ; резервування 1 байту для змінної var1
table: .BYTE 3      ; резервування 3 байт під таблицю table
```

Для роботи із змінними у пам’яті даних SRAM використовуються команди `lds, ldd, st, sts`. Більшість з них є однотипними та відрізняються лише використанням різних індексних регістрових пар X, Y, Z. Найпростішими є команди `sts`, яка записує у вказану адресу комірки пам’яті значення з регістра загального призначення, і `lds`, яка завантажує з вказаної адреси комірки пам’яті значення у регістр загального призначення.

```
.DSEG
var1:  .BYTE 1
table: .BYTE 3
.CSEG
ldi r16, 2
sts var1, r16     ; $060<- 2
ldi r16, 3
sts table, r16    ; $061<- 3
ldi r16, 4
sts table+1, r16 ; $062<- 4
ldi r16, 5
sts table+2, r16 ; $063<- 5
lds r16, table+2
lsl r16  ;  r16=5*2=10=$0A
sts var1, r16
```



У командах `st` та `ld` використовують індексні регістрові пари X, Y, Z в які спочатку заноситься адреса комірки пам’яті в SRAM, а потім у залежності від алгоритму, використовується одна з трьох нотацій для регістрової пари (звичайна, з плюсом, з мінусом).

Команди із звичайною нотацією, наприклад `st X, Rd`, заносять значення з регістра загального призначення Rd у комірку SRAM за адресою, що вказана в регістровій парі X.

Команди з плюсом після регістрової пари у нотації, наприклад `ld R,X+, st X+,Rd`, зчитують/записують дані з/у SRAM, а потім збільшують значення адреси в регістровій парі на одиницю.

Команди з мінусом перед регістровою парою у нотації, наприклад `ld R,-X, st -X,Rd`, спочатку зменшують значення адреси в регістровій парі на одиницю, а потім зчитують/записують дані з/у SRAM.

Приклад використання команд для роботи з пам’яттю даних SRAM. Три вибірки з порту A зчитуються у пам'ять даних SRAM, а потім у зворотному порядку записуються у порт B.

```
.include "m8515def.inc"
  .DSEG
data:  .byte 3    ; резервування 3 байт для даних

  .CSEG
  ldi  r16, 0x00
  ldi  r17, 0xFF

  out  DDRA, r16  ; порт A на вхід
  out  PORTA, r16

  out  DDRB, r17  ; порт B на вихід
  out  PORTB, r16

  ldi  ZL, low(data)   ; Z <- адреси позначки data
  ldi  ZH, high(data)

input:  in  r16, PINA  ; r16  <- порт A
  st  Z+, r16          ; SRAM <- r16;   (Z++)
```

```
  inc  r16             ; r16++
  cpi  r16, 3
  brne  input          ; if(r16!=3) перехід на позначку input

  clr  r16             ; r16=0
output:  ld  r16, -Z   ; (Z--);    r16 <- SRAM
  out  PORTB, r16      ; порт B <- r16
  inc  r16    ; r16++
  cpi  r16, 3
  brne  output         ; if(r16!=3) перехід на позначку output
```

Деякі моделі МК мають додаткові команди для роботи із SRAM: `std Y+k, Rr` та `ldd Rr, Y+k`. Ці команди працюють лише із регістровими парами Y та Z, та виконують непрямий відносний запис/зчитування у/з SRAM. Адреса комірки пам'яті, до якої звертаються, отримується в результаті додавання значення регістрової пари та константи, при цьому значення регістрової пари не змінюється.

```
ldi ZL, low(data)
ldi ZH, high(data)  ; Z=$060 <- адреса позначки data

ldi r16, 5      ; r16=5
std Z+1, r16    ; $061<-5 (Z=$060)
std Z+2, r16    ; $062<-5 (Z=$060)
ldd r16, Z+3    ; r16<-$063 (Z=$060)
```

Команди для роботи з пам'яттю даних SRAM виконується МК за 2 такти.

2. Робота з пам'яттю програм FLASH

МК ATMega8515 має програмовану Flash пам'ять програм. Програми і константи записані у пам'ять програм зберігаються при відключенні живлення. Так як всі інструкції МК AVR є 16- або 32-бітові, то Flash пам'ять організована із сторінок розміром 4 кбайт × 16-біт. Кількість сторінок залежить від загального обсягу Flash пам'яті. 4 кбайт Flash пам'ять має 128 сторінок. Запис даних у Flash пам'ять складається з наступних кроків:

1. Чистка сторінки пам'яті.
2. Підготовка даних у тимчасовому буфері.
3. Власне запис даних.

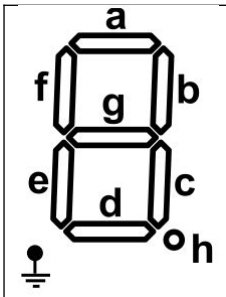
Для забезпечення безпеки програм Flash пам'ять розділяється на дві секції – завантажувальну і прикладну. Flash пам'ять розрахована на 10_000 циклів запису/стирання. Для роботи з пам'яттю програм призначені команди `spm, lpm`.

У пам'яті програм можна резервувати місце під таблиці констант, наприклад, таблиці синусів, косинусів, вектори значень для двійково-десяткового кодування тощо. Такі таблиці краще зберігати у пам'яті програм після коду програми, щоб вони не заважали виконанню коду. Таблиці можуть розміщуватися і в довільному місці коду, але у цьому випадку потрібні додаткові інструкції, щоб їх обійти.

Роботу з пам'яттю програм можна продемонструвати на прикладі циклічного виведення чисел від 1 до 99 із затримкою 0,5 сек на 7-сегментний індикатор. 7-сегментний графічний індикатор складається із світлодіодних сегментів. Засвічуючи певні сегменти, можна вивести необхідне число. 7-сегментні індикатори є із спільним анодом (+) та спільним катодом (–). Це означає, що у такому індикаторі в усіх сегментах один з виводів об'єднаний з рештою подібних виводів інших сегментів і тоді на спільний вивід подається високий (+), або низький (–) рівень

напруги в залежності від конструкції вибраного індикатора. На решту виводів подають протилежні за рівнем напруги для засвічування сегментів.

Для індикатора із спільним катодом на спільний вивід подається низький рівень, а для засвічування сегментів необхідно з МК подавати високі рівні на інші виводи. На рис. 8.1 показано індикатор з прийнятою літерною нумерацією сегментів, та наведена таблиця з відповідними кодами, які необхідно подати на індикатор, щоб засвітилася певна цифра. Наприклад, для висвічування цифри «5» необхідно засвітити сегменти `a, f, g, c, d`, тобто подати на них високий рівень.



Зобр.		h	g	f	e	d	c	b	a	10-знач.	16-знач.
0	0 b	0	0	1	1	1	1	1	1	63	3F
1	0 b	0	0	0	0	0	1	1	0	6	6
2	0 b	0	1	0	1	1	0	1	1	91	5B
3	0 b	0	1	0	0	1	1	1	1	79	4F
4	0 b	0	1	1	0	0	1	1	0	102	66
5	0 b	0	1	1	0	1	1	0	1	109	6D
6	0 b	0	1	1	1	1	1	0	1	125	7D
7	0 b	0	0	0	0	0	1	1	1	7	7
8	0 b	0	1	1	1	1	1	1	1	127	7F
9	0 b	0	1	1	0	1	1	1	1	111	6F

Рисунок 8.1 – Кодування 7-сегментного індикатора із спільним катодом

Схема підключення 7-сегментного індикатора до портів МК показана на рис. 8.2.

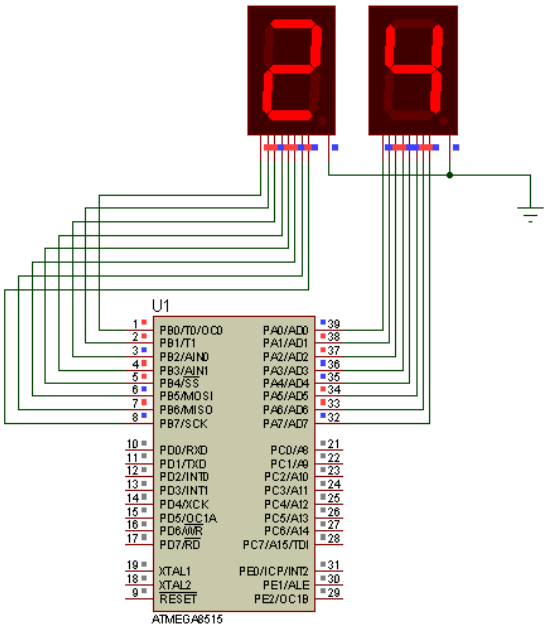


Рисунок 8.2 – Схема підключення індикаторів для виведення 2-розрядного числа

Для резервування простору у пам’яті програм FLASH використовуються директиви: `.db` (байт), `.dw` (слово або 2 байти), `.dd` (2 слова або 4 байти) та `.dq` (4 слова або 8 байт). Для заданої задачі необхідно створити таблицю з 10-ти однобайтних значень, які б відповідали кодам виводу від 0 до 9 для 7-сегментного індикатора. Значення розміщуються у таблиці у порядку зростання відображуваного числа. Тобто, якщо необхідно вивести число «5», тоді до адреси позначки таблиці додається значення зсуву 5 і отримується необхідний код.

Зчитування значення з пам’яті FLASH виконується командою `lpm` (3 такти) за адресою, що вказана у регістровій парі Z. Тобто, спочатку необхідно завантажити в Z адресу позначки таблиці, а потім до Z додати необхідне зміщення у таблиці та виконати команду зчитування.

```
.include "m8515def.inc"
.def  _8bin  = r18
.def  _bcdL  = r18
.def  _bcdH  = r19
; Константи
.equ  Fig_0 = 63   ; 0
.equ  Fig_1 = 6    ; 1
.equ  Fig_2 = 91   ; 2
.equ  Fig_3 = 79   ; 3
.equ  Fig_4 = 102  ; 4
.equ  Fig_5 = 109  ; 5
.equ  Fig_6 = 125  ; 6
.equ  Fig_7 = 7    ; 7
.equ  Fig_8 = 127  ; 8
.equ  Fig_9 = 111  ; 9

  .CSEG
  ldi  r16, Low(RAMEND) ; ініціалізація стеку
  out  SPL, r16
  ldi  r16, High(RAMEND)
  out  SPH, r16

  ldi  r16, 0x00
  ldi  r17, 0xFF
  out  DDRA, r17  ; порт A на вихід
  out  PORTA, r16
  out  DDRB, r17  ; порт B на вихід
  out  PORTB, r16

  clr  r20    ; r20=0  (число для виводу на індикатори)


main:  mov  _8bin, r20  ; r18 <- r20
  ;---------- Виклик підпрограми 2-10 кодування --------------------
  rcall  BCD  ; вхід: _8bin;    вихід: _bcdL(мол.),  _bcdH(старш.)

  ;---------- Виведення молодшого розряду десяткового числа -------------
  ldi  ZL, low(SegTable*2)
  ldi  ZH, high(SegTable*2)  ;Z <- адреса позначки SegTable у байтах
  clr  r16
  add  ZL, _bcdL
  adc  ZH, r16      ; Z = Z + _bcdL (зміщення)
  lpm  r16, Z       ; r16 <- FLASH(Z)
  out  PORTA, r16   ; порт A < r16

  ;---------- Виведення старшого розряду десяткового числа --------------
  ldi  ZL, low(SegTable*2)
  ldi  ZH, high(SegTable*2)  ;Z <- адреса позначки SegTable у байтах
  clr  r16
```

```
  add  ZL, _bcdH
  adc  ZH, r16      ; Z = Z + _bcdH (зміщення)
  lpm  r16, Z       ; r16 <- FLASH(Z)
  out  PORTB, r16   ; порт B <- r16

  ;---------- Виклик підпрограми затримки ---------------------------
  rcall  P05sec
  ;---------- Інкремент числа r20 до 99 -----------------------------
  inc  r20      ; r20++
  cpi  r20, 100
  brne  main    ; if (r20 != 100)  goto main
  clr  r20      ; else  r20=0;  goto main
  rjmp  main

  ;Підпрограма Двійково-Десяткового Кодування (до 99, незапаковане)
BCD:   clr  _bcdH
BCD1:  subi  _bcdL, 10    ;  :79: => :7:9:
  brcs  BCD2
  inc  _bcdH
  rjmp  BCD1
BCD2:  subi  _bcdL, -10
  ret

;Підпрограма паузи 0.5 сек.
P05sec: ldi  r16, 0x00
  ldi  r17, 0x05
  ldi  r18, 0x05
delay:  subi  r16, 1
  sbci  r17, 0
  sbci  r18, 0
  brne  delay
  ret

;Вектор даних
SegTable:  .db   Fig_0,Fig_1,Fig_2,Fig_3,Fig_4,Fig_5,Fig_6,Fig_7,Fig_8,Fig_9
       ;коди цифр    0     1     2     3     4     5     6     7     8     9
```

3. Робота з пам’яттю EEPROM

МК AVR мають внутрішню енергонезалежну пам’ять EEPROM (**E**lectricaly **E**rasable **PROM**) для зберігання даних і констант розміром від 64 до 512 байт і поділена на сторінки. Так МК ATMega8515 має пам’ять EEPROM розміром 512 байт, розмір сторінки 4 байти і кількість сторінок 128.

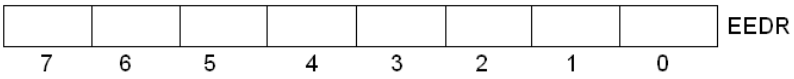
Пам’ять EEPROM організована в окремому адресному просторі, в якому може бути прочитаний/записаний окремий байт під час виконання програми користувача. Пам'ять EEPROM розрахована на 100_000 циклів запису/стирання.

Для роботи з EEPROM призначені 3 регістри введення/виведення:

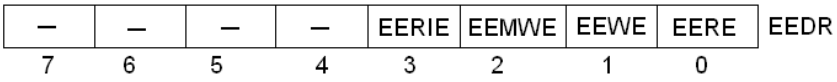
• EEAR – 16-розрядний регістр адреси EEPROM-пам’яті, який фізично розташовується у двох регістрах EEARH:EEARL. З цих двох регістрів використовується тільки 9-розрядів для адресації 512 байт. У цей регістр завантажується адреса комірка, до якої будуть звертатися як для запису, так і для читання.

15	14	13	12	11	10	9	8	
-	-	-	-	-	-	-	EEAR8	EEAR
EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	
7	6	5	4	3	2	1	0	

• EEDR – регістр даних. У цей регістр завантажуються дані для записування в EEPROM, а також розміщуються дані зчитані з EEPROM-пам’яті.



• EECR – регістр керування доступом до EEPROM-пам’яті. У ньому використовуються 4 біти для визначення поведінки МК при роботі з EEPROM.



EERIE – лог. 1 дозволяє переривання, а лог. 0 забороняє переривання EEPROM.

EEMWE – загальний дозвіл запису у пам’ять, При лог. 1 і якщо за час 4-тактів EEWE також буде лог. 1, то дані будуть записані. При значенні лог. 0 дані не записуються.

EEWE – керування процесом запису, лог. 1 дає дозвіл на запис, лог. 0 – не дає.

EERE – керування процесом читання, лог. 1 дає дозвіл на читання, лог. 0 – не дає.

Для програмування пам’яті EEPROM необхідно виконати наступні дії:

1. Дочекатися закінчення процесу програмування пам'яті EEPROM (якщо він активний), тобто, поки розряд EEWE повернеться в стан лог. 0.

2. Записати нову адресу в регістр EEAR.

3. Записати необхідний байт даних в регістр EEDR.

4. Встановити в лог. 1 розряд EEMWE регістра EECR.

5. Упродовж наступних чотирьох тактів після встановлення розряду EEMWE в розряд EEWE записати лог. 1. Тим самим буде запущений процес програмування.

По закінченню циклу програмування, розряд EEWE апаратно автоматично скидається у лог. 0. Програма користувача повинна безперервно опитувати цей розряд, чекаючи появи лог. 0, перше ніж приступити до програмування наступного байту.

При запису одного байта в пам'ять EEPROM має бути також встановлений у лог. 1 розряд EEMWE. Після того як розряд EEMWE встановлений, рівень лог. 1 в ньому зберігається впродовж чотирьох періодів такту системної синхронізації, а потім буде виконано апаратне скидання у лог. 0. Програма користувача може програмувати байт за допомогою запису лог. 1 в розряд EEWE тільки упродовж цих чотирьох тактів системної синхронізації. Якщо буде встановлений розряд EEWE, але без встановлення також і розряду EEMWE, то процес програмування не почнеться.

Тривалість запису в EEPROM є достатньо великою та складає приблизно 2-9 мсек при тактовій частоті 1 МГц.

Приклади підпрограм асемблера для запису/читання EEPROM.

```
; Запис EEPROM
EEPROM_write:
; Поки завершиться попередній write
sbic EECR,EEWE
rjmp EEPROM_write
; Задання адреси з (r18:r17)в EEAR
out  EEARH, r18
out  EEARL, r17
; Запис даних з (r16) в EEDR
out  EEDR,r16
; Запис лог.1 у біт EEMWE
sbi  EECR,EEMWE
```

```
; Читання EEPROM
EEPROM_read:
; Поки завершиться попередній write
sbic EECR,EEWE
rjmp EEPROM_read
; Задання адреси з (r18:r17) у EEAR
out  EEARH, r18
out  EEARL, r17
; Початок читання eeprom записом лог.1
у EERE
sbi  EECR,EERE
```

; Початок запису eeprom записом лог.1 у біт EEWE sbi EECR,EEWE ret	; Читання даних з EEDR in r16,EEDR ret

Інший приклад програми у якій число записується в EEPROM, а потім зчитується назад і виводиться на лінійку світлодіодів, які підключені до порту A (рис. 8.3).

```
.include "m8515def.inc"
  .CSEG
  ldi  r16, 0x00
  ldi  r17, 0xFF
  out  DDRA, r17     ; порт A на вихід
  out  PORTA, r16
  ldi  r18, 159      ; r18=0b10011111
;---------- запис в E_Pass -----------------------------
E_Write:  sbic  EECR, EEWE    ; очікуємо готовності EEPROM
  rjmp  E_Write
  ldi  r17, high(E_Pass)      ; отримуємо адресу позначки E_Pass
  ldi  r16, low(E_Pass)
  out  EEARH, r17      ; завантажуємо адресу в рег. EEAR
  out  EEARL, r16
  out  EEDR, r18       ; заносимо байт даних для запису
  cli
  sbi  EECR, EEMWE     ; вст. попередній дозвіл на запис
  sbi  EECR, EEWE      ; вст. остаточний дозвіл на запис
  sei
;---------- читання з E_Pass ----------------------------
E_Read:  sbic  EECR, EEWE     ; очікуємо готовності EEPROM
  rjmp  E_Read
  ldi  r17, high(E_Pass)     ; отримуємо адресу мітки E_Pass
  ldi  r16, low(E_Pass)
  out  EEARH, r17     ; завантажуємо адресу в рег. EEAR
  out  EEARL, r16
  sbi  EECR, EERE     ; вст. дозвіл на читання
  in  r20, EEDR       ; зчитуємо байт з рег. EEDR
  out  PORTA, r20     ; виводимо байт на світлодіоди
main:  rjmp  main
;розмітка пам'яті EEPROM
  .ESEG
E_Pass: .db  145  ;0b10010001
```

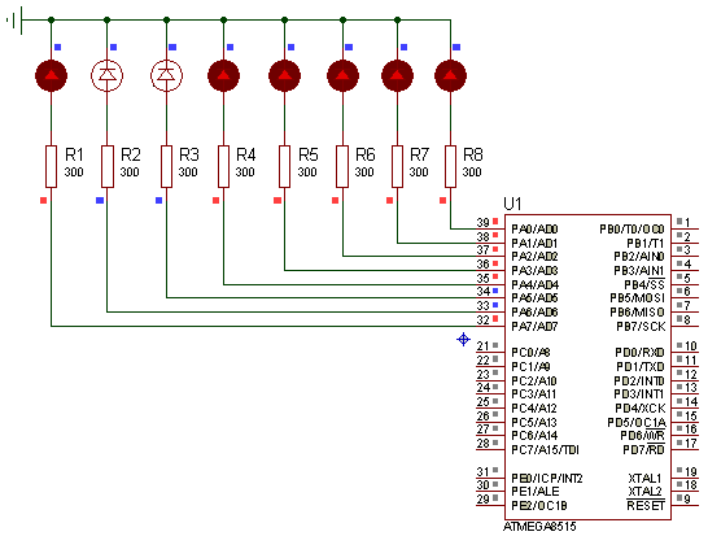


Рисунок 8.3 – Результат запису у пам'ять EEPROM, числа 159 (0b1001_1111)

4. Обмеження можливостей програмування і зберігання даних

Для активації механізмів блокування призначений байт бітів блокування (табл. 8.1). Він містить розряди блокування для прикладної секції LB0, LB1 і для для завантажувальної секції (boot loader) BLB01, BLB02, BLB11, BLB12. Режими блокування пам'яті показані в табл. 8.2.

Таблиця 8.1 – Байт бітів блокування

Блокувальні біти	Номер біта	Описання	Значення за замовчуванням
	7	-	1 (незапрограмований)
	6	-	1 (незапрограмований)
BLB12	5	Boot lock bit	1 (незапрограмований)
BLB11	4	Boot lock bit	1 (незапрограмований)
BLB02	3	Boot lock bit	1 (незапрограмований)
BLB01	2	Boot lock bit	1 (незапрограмований)
LB2	1	Lock bit	1 (незапрограмований)
LB1	0	Lock bit	1 (незапрограмований)

Таблиця 8.2 – Режими блокування пам'яті

LB Mode	LB1	LB2	Тип захисну
1	1	1	Ніяких обмежень із сторони захисту (незапрограмований стан)
2	0	1	Програмування FLASH і EEPROM заборонене в послідовному і паралельному режимі. Біти запобіжники (Fuse bits) заблоковані у режимах паралельного і послідовного програмування. Тому біти запобіжники мають програмуватися перед бітами блокування.
3	0	0	Програмування і верифікація (зчитування) FLASH і EEPROM заборонене в послідовному і паралельному режимі. Біти запобіжники (Fuse bits)В заблоковані у режимах паралельного і послідовного програмування.
BLB0 Mode	BLB02	BLB01	
1	1	1	Немає обмежень для SPM і LPM доступу у прикладній секції
2	1	0	SPM недозволений для запису у прикладній секції
3	0	0	SPM недозволений для запису у прикладній секції, LPM виконання із завантажувальної секції недозволене для читання прикладної секції. Якщо вектор переривань розміщений у завантажувальній секції, переривання будуть недозволені при виконання з прикладної секції.
4	0	1	LPM виконання з завантажувальної секції недозволені для читання прикладної секції. Якщо вектор переривань розміщений у завантажувальній секції, переривання будуть недозволені при виконання з прикладної секції.

Примітка. SPM (Store program memory), LPM (load program memory).

Стерти розряди блокування можна при стиранні всього кристалу, при цьому очищається увесь блок.

4.1. Байти сигнатури

Для ідентифікації мікросхеми використовується три байти сигнатури, табл. 8.3. Їх можна прочитати в послідовному і паралельному режимах програмування. Ці байти розміщені в окремому адресному просторі.

Таблиця 8.3 – Значення байтів сигнатури

Адреса	Байт даних	Значення
0x000	0x1E	Компанія виробник
0x001	0x90	512x16-розрядів Flash-пам’ять
	0x91	1024x16-розрядів Flash-пам’ять
	0x92	20148x16-розрядів Flash-пам’ять
	0x93	4096x16-розрядів Flash-пам’ять
0x002	0x07	Atmega пристрій

Деякі МК зберігають тут калібраційні значення для RC генераторів. Так ATmega 8 використовує адреси 0x0001, 0x0002, 0x0003 для частот 1 МГц, 2 МГц, 4 МГц і 8 МГц. При натисканні Reset за замовчування завантажуються калібраційні значення для 1 МГц у OSCCALL регістр. Для інших частот калібраційні значення вводяться вручну.

4.2. Процес програмування

У заводській поставці FLASH і EEPROM пам’ять стерта, тобто записана значенням $FF. Запрограмувати пам'ять можна в паралельному або послідовному режимі. Паралельний режим надає всі можливості програмування. Послідовний режим не потребує особливої напруги програмування і тому мікросхема може бути запрограмована в складі системи.

4.3. Паралельний режим програмування

Для паралельного режиму програмування потрібна напруга +12 В. Схема з’єднання МК при паралельному режимі програмування показана на рис. 8.4.

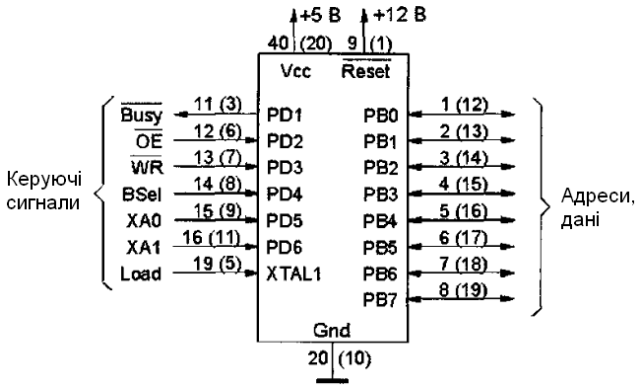


Рисунок 8.4 – з’єднання МК при паралельному програмуванні
(для DIP корпуса з 40-виводами)

• /Busy:

• лог. 0 – виконується програмування;

• лог. 1 – мікросхема готова до отримання нового командного слова;

• /OE (Output Enable) – активує каскад вихідного підсилювача-формувача при зчитуванні даних;

• /WR – сигнал запису, запускає процес програмування;

• /BSel (Byte Select):

• лог. 0 – вказує на молодший байт адреси або слова даних;

• лог. 1 – вказує на старший байт адреси або слова даних;

• XA0, XA1 – визначають дію при появленні на вході XTAL1 позитивного імпульсу завантаження (Load). Функції сигналів показані в табл. 8.4:

Таблиця 8.4 – Функції розрядів XA0, XA1

XA1	XA0	Функція
0	0	Завантажити старший (BSel=1) або молодший (BSel=0) байт адреси пам’яті FLASH або EEPROM.
0	1	Завантажити старший (BSel=1) або молодший (BSel=0) байт даних
1	0	Завантажити командне слово
1	1	Відсутні дії

Перед появленням імпульсу низького рівня /WR або /OE має бути завантажене одне з командних слів (табл. 8.5), яке визначає операцію.

Таблиця 8.5 – Командне слово для виконання дій програмування і перевірки

Командне слово	Функція
1000_0000	Стерти всю інформацію на кристалі
0100_0000	Програмування розрядів запобіжників
0010_0000	Програмування розрядів блокування
0001_0000	Програмування FLASH пам’яті
0001_0001	Програмування пам’яті EEPROM
0000_1000	Зчитування байтів сигнатури
0000_0100	Зчитування розрядів блокування і запобігання
0000_0010	Читання Flash пам’яті
0000_0011	Читання пам’яті EEPROM

4.4. Послідовний режим програмування

В послідовному режимі програмування використовується інтерфейс SPI з використанням методу “пропалювання”. Програмуючим пристроєм є інтерфейс SPI, який працює як ведучий пристрій і програмований пристрій МК є веденим. Послідовний інтерфейс складається з ліній SCK, MOSI (вхід), MISO (вихід). За наростаючим фронтом імпульсів в лінії SCK біти даних записуються в МК, а за спадаючим фронтом - зчитуються.

Схема підключення виводів МК при послідовному програмуванні показна на рис. 8.5.

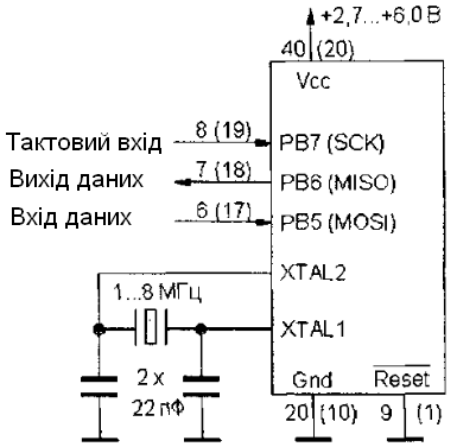


Рисунок 8.5 – Підключення виводів МК при послідовному програмуванні

5. Налаштування режимів програмування МК

При налаштуванні режимів програмування МК використовують програмні середовища і програматори, де застосовується таке поняття як “плавкі перемички” – fuse bits. Fuse bits називають область з 4-байтів, які відповідають за початкову (глобальну) конфігурацію. Ці біти визначають вибір задаючого генератора (зовнішнього чи внутрішнього), вибір режиму роботи подільника частоти тактових імпульсів, використання виводу скидання (reset) за своїм призначенням чи як додаткового порту введення-виведення, кількість пам’яті яка виділяється для завантажувача та інше.

Потрібно знати, що у ф’юзів незвичайна функціональність – вибір позначки у програмному середовищі напроти функції ф’юза означає, що ця функція невикористана.

Чому саме так? Зараз конфігураційні байти записуються у флеш пам’ять і поміняти їх можна скільки разів завгодно. Раніше, коли FLASH пам’яті ще не було, для конфігурації використовували спеціальні перемички (fuse), які одноразово фізично перепалювалися. Тому зараз, з цієї причини, якщо перемичка ціла «1» значить що функція не використана і навпаки – перемичку перепалили «0» значить функція використана.

Кожний МК має свій набір плавких запобіжників (fuses), які наведені в технічній документації. Але є плавкі запобіжники загальні для багатьох МК. Нижче наведено назви і призначення основних з fuse-бітів:

RESERVED – зарезервований біт.

OCDEN (On Chip Debug ENable) – дозволяє роботу схеми внутрішнього налагоджувача. *Не рекомендується* залишати встановленим цей біт в комерційних виробах, так як програму можна буде зчитати з пам’яті МК.

JTAGEN – дозволяє роботу інтерфейсу програмування/налагодження JTAG. В порівнянні з інтерфейсом SPI, JTAG має розширені можливості. Не рекомендується залишати цей біт встановленим, так як при цьому зростає струм споживання МК.

SELFPRGEN – дозвіл програмі МК на запис в пам’ять програм, тобто на самопрограмування.

DWEN – дозвіл на роботу одно-провідного інтерфейсу налагодження DebugWire. *Не рекомендується* залишати його встановленим в комерційних виробах.

EESAVE – біт, після встановлення якого при стиранні пам'яті МК вміст EEPROM даних зберігається.

SPIEN – біт дозволу роботи інтерфейсу внутрішньо схемного програмування МК по SPI. Цей біт можна легко перевстановити за допомогою паралельного програматора (або JTAG, якщо такий дозволений і є в МК). Всі МК випускаються із встановленим бітом SPIEN, який зняти по інтерфейсу SPI *неможливо*.

WDTON – біт, після встановлення якого сторожовий таймер WDT включається зразу після подачі живлення і не може бути відключений програмно. Якщо біт не встановлений, то включенням і виключенням WDT можна керувати програмно.

Група бітів BODLEVEL яка визначає поріг спрацювання схеми BOD – детектора рівня напруги живлення. При пониженні напруги живлення нижче цього рівня скидається МК.

BODEN – біт, який включає схему апаратного детектора недопустимого рівня напруги живлення BOD.

RSTDISBL – біт, який відключає від виводу МК сигнал зовнішнього скидання і підключає до виводу схему порту введення-виведення. Цей біт є тільки в тих МК, у яких вивід апаратного скидання RESET суміщений з одним із портів введення-виведення. Помилкове встановлення біту може відключити RESET і тоді стане недоступним програмування по інтерфейсу ISP.

CKDIV8 – біт попереднього ділення частоти кварцового (або іншого наявного) тактового генератора на 8. Так, при встановленні біту і наявності кварцового резонатора на 8 МГц реальна тактова частота МК буде 1 МГц.

CKOUT – біт дозволу виведення тактової частоти на один з виводів МК (для тактування інших пристроїв).

SUT1 і SUT0 – біти керування режимом запуску тактових генераторів МК.

CKOPT – біт режиму роботи вбудованого генератора тактової частоти для роботи з кварцовим резонатором.

Група бітів CKSEL0…CKSEL3 – комбінація яких визначає тип і частоту працюючого тактового генератора. При помилковій комбінації бітів МК може не працювати.

PLLCK – біт дозволу на використання вбудованого синтезатора частоти для тактування ядра МК.

BOOTRST – біт, який визначає адресу, з якої почнеться виконання програми після скидання. Якщо біт встановлений, то програма починається не з адреси 0000h (як звичайно), а з адреси області завантажувача (Boot Loader).

Два біти BOOTSZ – визначають розмір області пам'яті програм, яка виділяється для завантажувача (Boot Loader).

У табл. 8.6 показано Значення fuse-бітів для різних моделей МК.

Таблиця 8.6 – Значення fuse-бітів для різних моделей МК

	Tiny				ATmega								
	2313	25/ 45/ 85	13	26	261/ 461/ 861	8	16	48/ 88/ 168	128	169	329	8515	8535
RESERVED									M103C	+		S8515C	S8535C
OCDEN							+		+	+	+		
JTAGEN							+		+	+	+		

SELFPRGEN	+	+	+		+			+					
DWEN	+	+	+		+		+	+					
EESAVE	+	+	+	+	+	+	+	+	+	+	+	+	+
SPIEN	+	+	+	+	+	+	+	+	+	+	+	+	+
WDTON	+	+	+		+	+		+	+	+	+	+	+
BODLEVEL2	+	+			+			+		+			
BODLEVEL1	+	+	+		+			+		+			
BODLEVEL0	+	+	+	**BODLEVEL**	+	**BODLEVEL**	**BODLEVEL**	+	**BODLEVEL**	+	+	**BODLEVEL**	**BODLEVEL**
BODEN				+		+	+		+			+	+
RSTDISBL	+	+	+	+	+	+		+			+		
CKDIV8	+	+	+		+			+		+	+		
CKOUT	+	+			+			+		+	+		
SUT1	+	+	+	+	+	+	+	+	+	+	+	+	+
SUT0	+	+	+	+	+	+	+	+	+	+	+	+	+
CKOPT				+		+	+		+			+	+
CKSEL3	+	+		+	+	+	+	+	+	+	+	+	+
CKSEL2	+	+		+	+	+	+	+	+	+	+	+	+
CKSEL1	+	+	+	+	+	+	+	+	+	+	+	+	+
CKSEL0	+	+	+	+	+	+	+	+	+	+	+	+	+
PLLCK				+									
BOOTRST						+	+	+	+	+	+	+	+
BOOTSZ1						+	+	+	+	+	+	+	+
BOOTSZ0						+	+	+	+	+	+	+	+

Примітка. Кольором позначено біти встановлення яких потребує підвищеної уваги.

В табл. 8.7, 8.8 показано fuse-біти з технічної документації для МК ATmega 8515.

Таблиця 8.7 – Старший байт Fuse бітів

Fuse біти	№ біту	Описання	Значення за замовчуванням
S8515C	7	AT90S4414/8515 режим сумісності	1 (не програмований)
WDTON	6	Watchdog always on	1 (не програмований)
SPIEN	5	Enable serial program and data downloading	0 (програмований, доступний SPI)
CKOPT	4	Oscillator options	1 (не програмований)
EESAVE	3	EEPROM memory is preserved through chip erase	1 (не програмований, EEPRON not preserved)
BOOTSZ1	2	Select boot size	0 (програмований)
BOOTSZ0	1	Select boot size	0 (програмований)
BOOTRST	0	Select reset vector	1 (не програмований)

Таблиця 8.8 – Молодший байт Fuse бітів

Fuse біти	№ біту	Описання	Значення за замовчуванням
BODLEVEL	7	Brown-out Detector trigger level	1 (не програмований)
BODON	6	Brown-out Detector enable	1 (не програмований, BOD disabled)
SUT1	5	Select start-up time	1 (не програмований)
SUT0	4	Select start-up time	0 (програмований)
CKSEL3	3	Select clock source	0 (програмований)
CKSEL2	2	Select clock source	0 (програмований)
CKSEL1	1	Select clock source	0 (програмований)
CKSEL0	0	Select clock source	1 (не програмований)

6. Налаштування завантажувача

Мікроконтролери AVR мають можливість самопрограмування, тобто можуть самостійно змінювати вміст своєї flash пам'яті. У практичному плані це означає, що, написавши для МК спеціальну програму-завантажувач (так званий bootloader), можна оновлювати його прошивку, не використовуючи програматор. Причому інтерфейс, за яким в мікроконтролер буде передаватися код програми, може бути абсолютно довільним. Звичайно для цих цілей використовується один з апаратно підтримуваних інтерфейсів, наприклад, SPI, I2C або RS-232. Однак існують і завантажувачі, засновані на програмній реалізації таких інтерфейсів як USB і 1-Wire.

Пам'ять програм мікроконтролерів AVR розділена на дві секції – секцію прикладної програми і секцію завантажувача. Завантажувач – це просто програма, яка зберігається в однойменній секції і може здійснювати запис/читання обох секцій пам'яті мікроконтролера. Ця особливість дозволяє завантажувачу модифікувати і навіть видаляти самого себе! Для того щоб використати завантажувач, його потрібно за допомогою програматора записати в Flash пам'ять МК. Область пам'яті, в яку буде записуватися завантажувач, вказується в конфігураційному файлі компонувача (linker) при компіляції початкового коду завантажувача. Розмір секції прикладної програми і секції завантажувача задається за допомогою конфігураційних бітів BOOTSZ1, BOOTSZ0. Для МК ATmega8515 розміри секцій пам'яті в залежності від значень конфігураційних бітів і показані в табл. 8.9.

Таблиця 8.9 – Розміри секцій пам’яті

BOOTSZ1	BOOTSZ0	Boot size, words	Pages	Application Flash Section	Boot Loader Flash Drction
1	1	128	4	0x000-0xF7F	0xF80-0xFFF

1	0	256	8	0x000- 0xEFF	0xF00- 0xFFF
0	1	512	16	0x000- 0xDFF	0xE00- 0xFFF
0	0	1024	32	0x000- 0xBFF	0xC00- 0xFFF

Якщо завантажувач не використовується, то прикладна програма може використовувати увесь доступний обсяг пам’яті МК.

Після запису завантажувача у МК його потрібно активувати (запустити). Зробити це можна двома способами:

– перемістити вектор скидання в початок завантажувальної секції;

– здійснити перехід на початок завантажувального секції.

Реалізація першого способу вимагає встановлення конфігураційного біту BOOTRST в байті запобіжника. Відомо, що після подачі живлення або події скидання, МК починає виконувати свою програму з нульової адреси пам'яті програм, тобто з вектора скидання. За цією адресою, як правило, розміщується асемблерна команда безумовного переходу на прикладну програму. Встановлення біту BOOTRST переносить вектор скидання МК на початок його завантажувальної секції (наприклад, на адресу 0xC00, якщо обидва біти BOOTSZ1, BOOTSZ0 дорівнюють нулю), відповідно при старті він відразу починає виконувати код завантажувача.

BOOTRSR=1 – Reset Vector = Application reset (0x000)

BOOTRSR=0 – Reset Vector = Boot Loader Reset.

Реалізація другого способу має на увазі, що МК містить завантажувач і прикладну програму, яка буде переходити в секцію завантажувача при настанні якоїсь події. Для реалізації такого переходу потрібно: створити покажчик на функцію, привласнити йому адресу початку секції завантажувача та викликати функцію за допомогою цього покажчика. Наприклад, так:

```
//адреса початку завантажувальної секції МК
#define BOOT_SECTION  0xC00
//оголошення покажчика на функцію
void(*pBootloader)(void) ;
...
//якщо натиснута кнопка, запустити завантажувач
if (Button()){
//ініціалізація покажчика
   pBootloader = (void(*)())BOOT_SECTION;
   //перехід на початок секції завантажувача
   pBootloader ();
}
де (void(*)()) – це вираз для перетворення типу даних
```

Перед переведення МК у секцію завантажувача потрібно завершити активні дії прикладної програми.

Зауваження. Якщо прошити прикладу програму у МК за допомогою програматора, то завантажувач буде вилучений.

Висновки.

Внутрішня пам'ять даних SRAM використовується для розміщення невеликої кількості змінних так як її розмір всього 512 байт. У зовнішній пам’яті даних SRAM можна розмісти до 64 кбайт даних.

Програми зберігаються в пам’яті програм FLASH. Розмір пам’яті програм залежить від моделі МК (1 кбайт – 128 кбайт). Для забезпечення безпеки програм Flash пам’ять розділяється на дві секції – завантажувальну і прикладну. У пам’яті програм можна резервувати місце під таблиці констант, звичайно після коду програми.

МК AVR мають внутрішню енергонезалежну пам’ять EEPROM для зберігання даних і констант розміром від 64 до 512 байт. Для роботи з EEPROM призначені 3 регістри введення/виведення EEAR, EEDR, EECR.

Питання.

1. Дати характеристику внутрішній пам’яті SRAM.
2. Які команди використовуються для роботи з пам’яттю SRAM.
3. Дати характеристику пам’яті програм FLASH.
4. Які команди використовуються для роботи з пам’яттю програм FLASH.
5. Дати характеристику пам’яті EEPROM.
6. Які команди використовуються для роботи з пам’яттю EEPROM.
7. Які регістри використовуються для роботи з пам’яттю EEPROM.
8. Яка послідовність дій при програмуванні пам’яті EEPROM.
9. Засоби обмеження можливостей програмування і зберігання даних.
10. Паралельний і послідовний режим програмування МК.
11. Налаштування режимів програмування МК.

ЛЕКЦІЯ 9. Таймери/лічильники

Мета. Вивчення роботи таймерів/лічильників

Вступ. МК AVR мають різні реалізації таймерів. Перше за все це 8- та 16-розрядні таймери/лічильники. Таймери використовуються для відліку і вимірювання часових інтервалів, а лічильники для підрахунку числа зовнішніх подій та формування часових інтервалів. Присутні в моделях МК AVR сторожові таймери (Watchdog Timer) призначені для контролю за непередбачуваним зациклюванням програм.

План.

1. Призначення таймерів/лічильників
2. Таймер/лічильник T0 МК ATmega8515
3. Таймер/лічильник T1
4. Режим таймера
4.1. Функція захоплення (capture)
4.2. Функція порівняння (compare)
4.3. Режим ШІМа (PWM)
5. Програмування таймера T0
5.1. Режим лічильника
5.2. Режим таймера
6. Програмування функцій порівняння, захоплення і ШІМ таймера T1
6.1. Функція порівняння
6.2. Функція захоплення
6.3. Режим ШІМа
7. Сторожовий таймер

1. Призначення таймерів/лічильників

Мікроконтролери (МК) AVRl в залежності від класу (Tiny, Classic, Mega, ATXMega, AVR DA, AVR DB, AVR DD) і типу моделі мають в своєму складі від одного до трьох таймерів/лічильників загального призначення – T0, T1 і T2.

Перший таймер (8-розрядний T0), який є у всіх моделях, може використовуватися для відліку і вимірювання часових інтервалів або як лічильник зовнішніх подій, а в моделі ATmega8515 ще і для порівняння із заданим значенням. При переповненні лічильного регістра таймера генерується запит на переривання. Два інших таймери (16-розрядний T1 і 8-розрядний T2) крім названих мають ще додаткові функції. Обидва таймери можуть генерувати запит на переривання не тільки при переповненні лічильного регістра, але і від ряду інших подій. Вони можуть також використовуватися як широтно-імпульсні модулятори. Крім того, таймер T2 може працювати в асинхронному (відносно тактового сигналу МК) режимі. Кожний таймер/лічильник використовує один або більше виводів МК. Ці виводи можуть бути або лініями портів введення/виведення з альтернативною функцією, або виділеними виводами МК. Всі виводи МК ATmega8515, які відносяться до таймерів/лічильників, і їх функції показані в табл. 9.1.

Таблиця 9.1 – Виводи таймерів-лічильників

Назва	ATmega8515	Описання
T0	PB0	вхід зовнішнього сигналу таймера T0
OC0	PB0	вихід схеми порівняння таймера T0
T1	PB1	вхід зовнішнього сигналу таймера T1
ICP	ICP/PE0	вхід захоплення таймера T1
OC1A	PD5	вихід схеми порівняння таймера T1
OC1B	OC1B/PE2	вихід схеми порівняння таймера T1

При використанні ліній портів введення/виведення необхідно сконфігурувати виводи у відповідності з їх функціональним призначенням (вхід або вихід). У всіх МК ATmel є також сторожовий таймер, який використовується для запобігання зацикленню програми.

2. Таймер/лічильник T0 МК ATmega8515

Основне призначення таймера/лічильника T0 (8-розрядний):

- одноканальний лічильник;
- лічильник зовнішніх подій;
- скидання значень лічильника при порівнянні на співпадіння із заданим значенням;
- джерело переривань при переповненні або порівнянні на співпадіння із заданим значенням;
- генератор частоти;

Структурна схема таймера/лічильника T0 МК ATmega8515 показана на рис. 9.1.

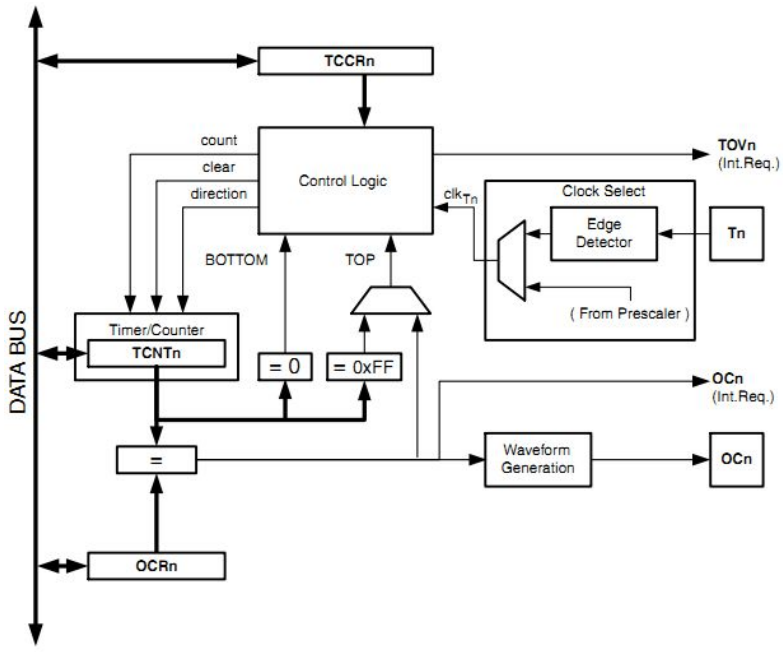


Рисунок 9.1 – Структурна схема таймера/лічильника T0

Таймер має 8-розрядний базовий лічильник TCNT0 і регістр порівняння виходу OCR0, регістр керування TCCR0, схему керування тактовими імпульсів. Крім того, він зв'язаний з розрядом регістра прапорів переривань TIFR і розрядом маски переривань TIMSK.

Після подачі напруги живлення лічильник TCNT0 встановлюється в нульовий стан. При переході таймера/лічильника TCNT0 із стану MAX ($FF) або TOP (задається у OCR0) в стан BOTTOM ($00) встановлюється в 1 прапор TOV0 в регістрі TIFR і генерується запит на переривання. Дозвіл переривання здійснюється встановленням в 1 розряду TOIE0 регістра маски TIMSK. Прапор загального дозволу переривання I регістра SREG МК також має бути встановлений в 1.

Таймер/лічильник T0 може працювати в двох режимах:

1) таймера, коли на вхід поступають тактові імпульси (безпосередньо або через попередній подільник схеми керування);

2) лічильника подій, коли вміст лічильника інкрементується за активним фронтом сигналу на вході T0 МК (лінія порту PB0).

Таймер лічильника може програмуватися на роботу в режимі збільшення і зменшення значень. Схема керування режимом роботи лічильника TCCR0 показана на рис. 9.2.

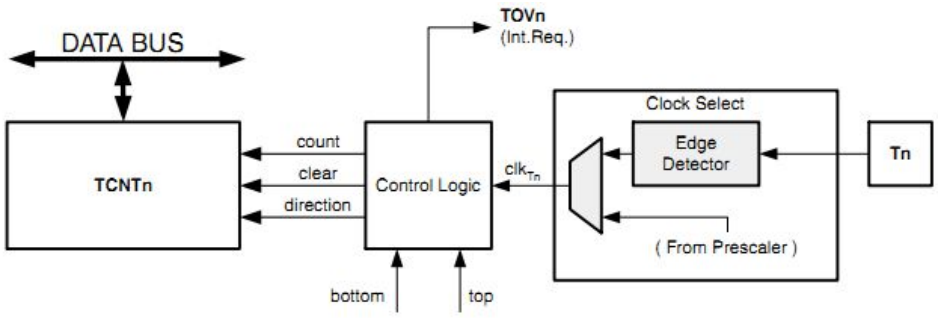


Рисунок 9.2 – Схема керування режимом роботи лічильника TCCR0:
`count` – інкремент/декремент на 1; `direction` – напрямок роботи лічильника;
`clear` – чистка лічильника; `clk` – тактовий сигнал лічильника;
`top/bottom` – максимальне/мінімальне значення лічильника.

Лічильник може тактуватися сигналом `clk` внутрішньо з використанням дільника або зовнішніми тактовими сигналами, які подаються на вивід T0.

Вибір тактового сигналу, а також запуск і зупинка таймера/лічильника задається розрядами CS02-CS00 регістра керування таймером TCCR0 (табл. 9.2). Відповідність між станами цих розрядів і джерелом тактового сигналу показана в табл. 9.3.

Таблиця 9.2 – Формат регістра TCCR0

Розряд	7	6	5	4	3	2	1	0
Ім'я	FOC0	WGM00	COM01	CPM00	WGM01	CS02	CS01	CS00

Таблиця 9.3 – Вибір джерела тактового сигналу для таймера/лічильника T0

CS02	CS01	CS00	Джерело тактового сигналу
0	0	0	Таймер/лічильник зупинений
0	0	1	СК (тактовий сигнал МК)

0	1	0	CK/8
0	1	1	CK/64
1	0	0	CK/256
1	0	1	CK/1024
1	1	0	Зовнішнє джерело імпульсів на вивід T0. Інкремент лічильника за *спадаючим* фронтом імпульсу
1	1	1	Зовнішнє джерело імпульсів на вивід T0. Інкремент лічильника за *наростаючим* фронтом імпульсу

8-розрядний компаратор постійно порівнює поточне значення лічильника TCNT0 і регістра порівняння виходу OCR0, рис. 9.3.

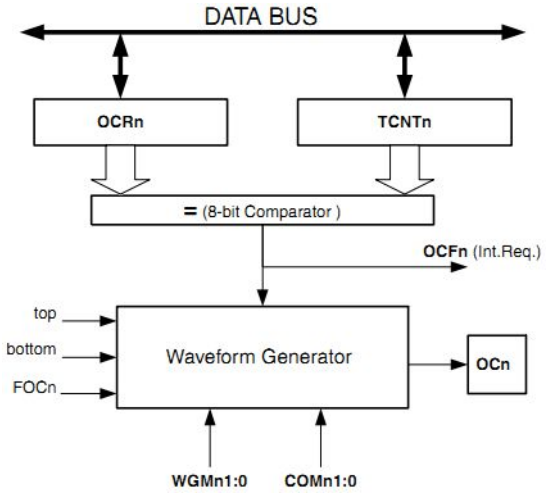


Рисунок 9.3 – Схема порівняння значень лічильника і регістра порівняння виходу

Компаратор сигналізує про співпадіння і при цьому встановлює прапор OCF0 (Output Compare Flag). Якщо встановлені прапори OCIE0=1 і I=1 (SREG), то OCF0 генерує переривання. Результат порівняння може використовуватися генератором, для генерування ШІМ сигналу(згідно встановлених бітів WGM01:2 і COM01:0) або змінної частоти на виводі OC0 (Output Compare Pin).

При роботі таймера/лічильника в режимі підрахунку зовнішніх подій необхідно врахувати, що сигнал, присутній на виводі T0, синхронізується частотою тактового генератора МК (стан виводу T0 зчитується за наростаючим фронтом внутрішнього тактового сигналу). В зв'язку з цим, для забезпечення коректної роботи таймера від зовнішнього сигналу, проміжок часу між сусідніми імпульсами повинен бути більшим періоду тактового сигналу МК. Вміст таймера/лічильника при підрахунку зовнішніх подій інкрементується навіть тоді, коли вивід T0 сконфігурований на вихід. Ця особливість дає користувачу можливість програмно керувати процесом підрахунку.

3. Таймер/лічильник T1

Таймер/лічильник TCNT1 (16-розрядний) має значно більше функцій, ніж таймер/лічильник T0. Основне призначення таймера/лічильника T1:

- два незалежних блоки порівняння виходу;
- два регістри порівняння виходу OCR1A, OCR1B;
- один блок захоплення входу;
- фазо-коректний широтно-імпульсний модулятор (ШІМ);
- ШІМ із змінним періодом;
- лічильник зовнішніх подій;
- чотири незалежних джерела переривань (TOV1, OCF1A, OCF1B і ICF1).

Структурна схема таймера/лічильника TCNT1 показана на рис. 9.4.

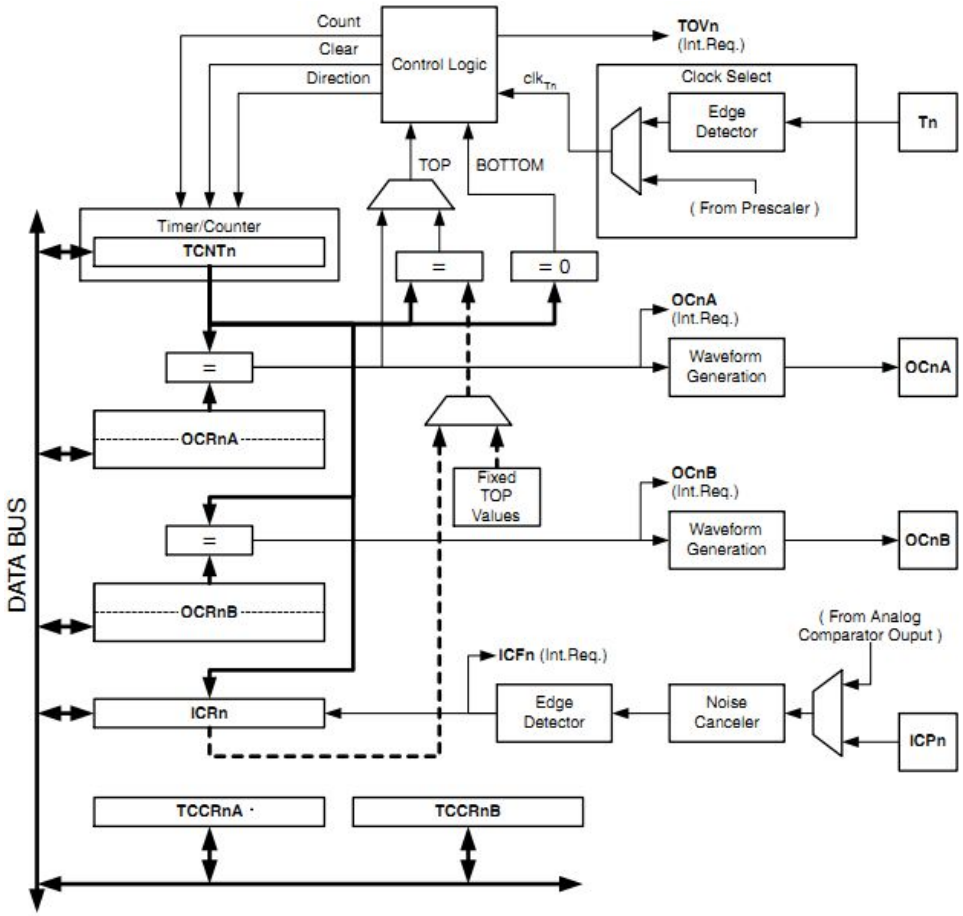


Рисунок 9.4 – Структурна схема таймера-лічильника TCNT1

Таймер/лічильник TCNT1 керується двома 8-розрядними регістрами керування: TCCR1A і TCCR1B. Формат регістрів показаний в табл. 9.4, 9.6. Значення окремих розрядів цих регістрів буде описано далі.

Таблиця 9.4 – Формат керуючого регістра TCCR1A

Регістр/біт	7	6	5	4	3	2	1	0
TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10

Bit 1:0 – WGM11:0: Режим генерації форми сигналу.

Bit 2 – FOC1B: Встановити порівняння виходу для каналу B.

Bit 3 – FOC1A: Встановити порівняння виходу для каналу A.

Bit 5:4 – COM1B1:0: Режим порівняння для каналу B

Bit 7:6 – COM1A1:0: Режим порівняння для каналу A

Якщо один або обидва біти COM1A1:0 або COM1B1:0 встановлені, то вихід OC1A або OC1B змінює функціональне призначення приєднаного виводу. При цьому DDR біт також має бути встановлений.

Коли OCR1A або OCR1B приєднані до виводів, тоді функція COM1x1:0 залежить від бітів WGM13:0 і визначає режим таймера/лічильника:

- нормальний;
- швидкий ШІМ (PWM);
- фазо-коректний ШІМ (PWM) ;
- фазо- і частотно-коректний ШІМ (PWM).
- частотно-імпульсний (CTX).

Описання бітів керування режимом роботи генерала форми сигналу показані в табл. 5

Таблиця 9.5 – Біти керування режимом роботи генерала форми сигналу

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	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	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	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	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Формат керуючого регістра TCCR1B показано в табл. 9.6.

Таблиця 9.6 – Формат керуючого регістра TCCR1B

Регістр/біт	7	6	5	4	3	2	1	0
TCCR1B	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10

Bit 2:0 – CS12:0: вибір тактового сигналу.

Bit 4:3 – WGM13:2: режим генератора форми сигналу.

Bit 6 – ICES1: вибір захоплення фронту вхідного сигналу.

Bit 7 – ICNC1: схема подавлення завад на вході.

По відношенню до тактового сигналу таймер/лічильник T1 може працювати в двох режимах, аналогічно таймеру T0. Вибір джерела тактового сигналу, а також запуск і зупинка таймера/лічильника здійснюється за допомогою розрядів CS12...CS10 регістра керування таймером TCCR1B. Відповідність між станом цих розрядів і режимом роботи таймера/лічильника показана в табл. 9.7.

Таблиця 9.7 – Вибір джерела тактового сигналу для таймера/лічильника T1

CS12	CS11	CS10	Джерело тактового сигналу
0	0	0	Таймер/лічильник зупинено
0	0	1	СК (тактовий сигнал МК)
0	1	0	СК/8
0	1	1	СК/64
1	0	0	СК/256
1	0	1	СК/1024
1	1	0	Виведення T1, інкремент лічильника за спадаючим фронтом імпульсу
1	1	1	Виведення T1, інкремент лічильника за наростаючим

			фронтом імпульсу

Таймер/Лічильник (TCNT1), регістр порівняння виходу (OCR1A/B) і регістр захоплення входу (ICR1) є 16-розрядними, тому для доступу до них використовується спеціальна процедура. Керує роботою таймера/лічильника 8-розрядний регістр керування Timer/Counter Control Registers (TCCR1A/B). Запити на переривання виставляються в регістрі прапорів TIFR, а дозволи на переривання встановлюються в регістрі TIMSK.

Таймер лічильник може тактуватися від внутрішнього дільника або зовнішнього джерела імпульсів, що подається на вивід T1. Джерело тактових імпульсів задається бітами CS12:0 регістра TCCR1B (Timer/Counter Control Register B). Схема дільника для таймера/лічильник TCNT0, TCNT1 показана на рис. 9.5.

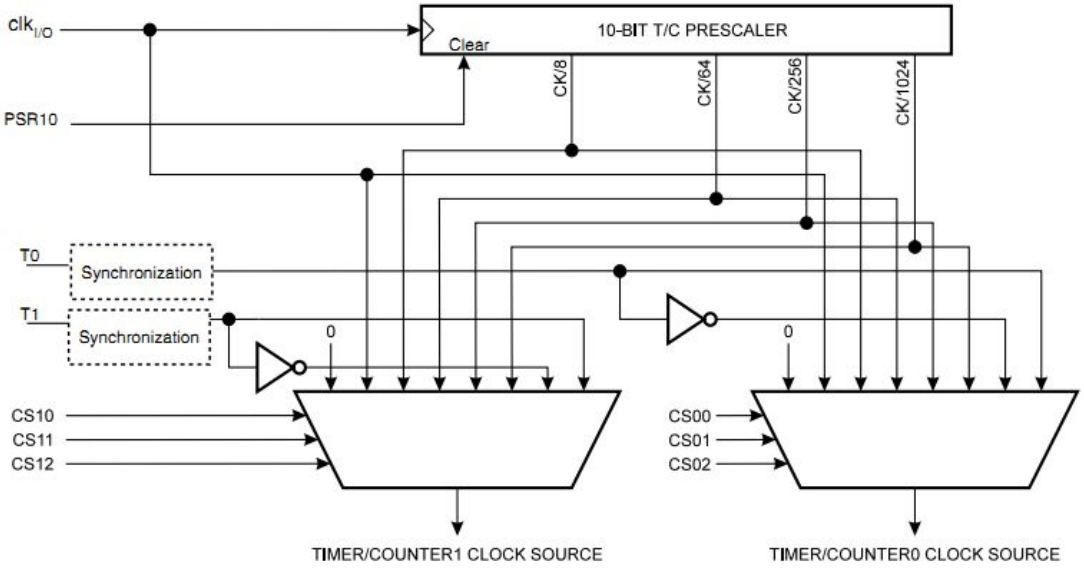


Рисунок 9.5 – Схема дільника таймерів/лічильників TCNT0/TCNT1

Таймер/лічильник TCNT1 є програмованим і двонаправленим, рис. 9.6.

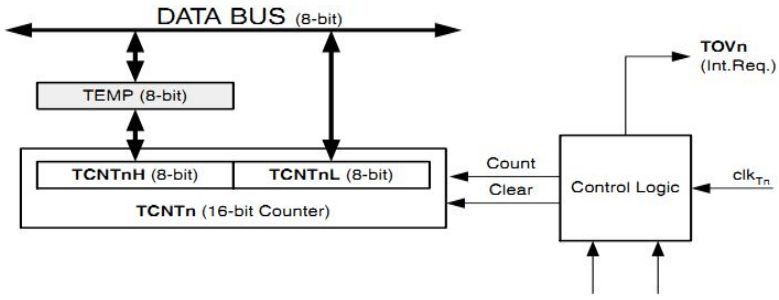


Рисунок 9.6 – Блок-схема керування таймера/лічильника TCNT1:
`count` – інкремент/декремент на 1;`direction` – зменшення/збільшення;
`clear` – очищення лічильника; `top` – сигнал про досягнення значення Top;
`botton` – сигнал про досягнення значення Botton.

16-розрядний лічильник TCNT1 відображується на два 8-розрядні регістри TCNT1H, TCNT1L. Центральний процесор має непрямий доступ тільки до регістра TCNT1H через тимчасовий регістр TEMP. В тимчасовий регістр записується значенням TCNT1H при читанні регістра TCNT1L і в TCNT1H записується значення регістру TEMP при записуванні в регістр TCNT1L. Це дозволяє центральному процесору читати або записувати 16-розрядний лічильник за один такт з використанням 8-розрядної шини даних.

3.1. Блок захоплення вхідних подій

Таймер/лічильник TCNT1 має блок захоплення вхідних подій і збереження їх часу появлення. Вхідні сигнали зовнішніх подій подаються на вхід ICP1 або через аналоговий компаратор. Блок-схема захоплення входу показана на рис. 9.7.

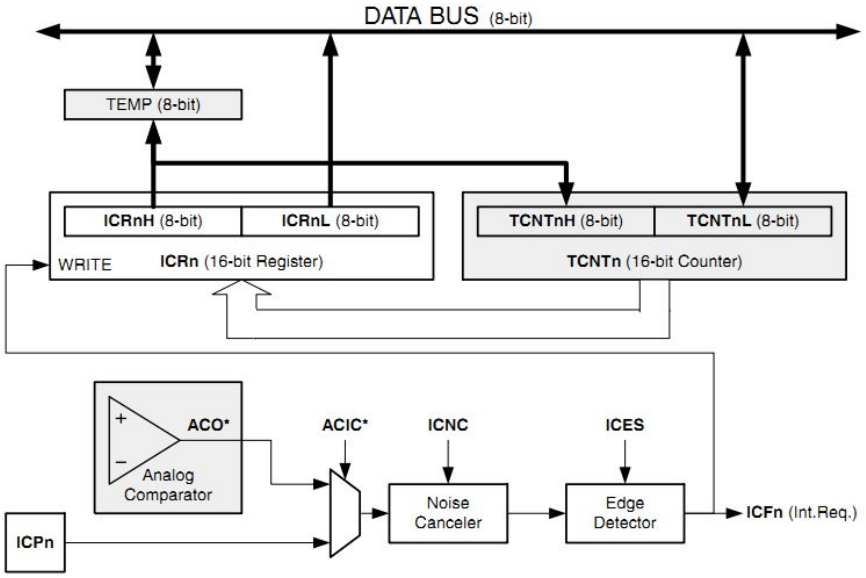


Рисунок 9.7 – Блок-схема захоплення входу таймера/лічильника TCNT1

При зміні логічного рівня на виводі ICPn (Input Capture pin) або на виході аналогового компаратора ACO (Analog Comparator output) встановлюється механізм захоплення при якому 16-розрядне значення TCNT1 записується у регістр ICR1 (Input Capture Register). Якщо біт TICIE1 = 1 прапор ICF1 генерує переривання захоплення входу. Прапор ICF1 автоматично очищається після оброблення переривання. 16-розрядне значення в регістрі ICR1 читається послідовно, спочатку байт ICR1L, а потім байт ICR1H. При читанні молодшого байту, старший байт копіюється в регістр TEMP. Коли центральний процесор читає старший байт в області I/O, він має доступ до регістра TEMP.

Коли джерелом події є аналоговий компаратор, то встановлюється біт ACIC (Analog Comparator Input Capture) у регістрі ACSR (Analog Comparator Control and Status Register).

3.2. Блок порівняння виходу

16-розрядний компаратор постійно порівнює значення таймера/лічильника TCNT1 з регістром OCR1x (Output Compare Register). При співпадінні їх значень компаратор видає

сигнал, який встановлює прапор OCF1x (Output Compare Flag). Якщо дозволено переривання OCIE1x=1, то прапор OCF1x генерує переривання порівняння виходу. Прапор автоматично чиститься після оброблення переривань. Сигнал співпадіння порівняння виходу використовує генератор форми сигналів (waveform generator). Схема блоку порівняння виходу показана на рис. 9.8.

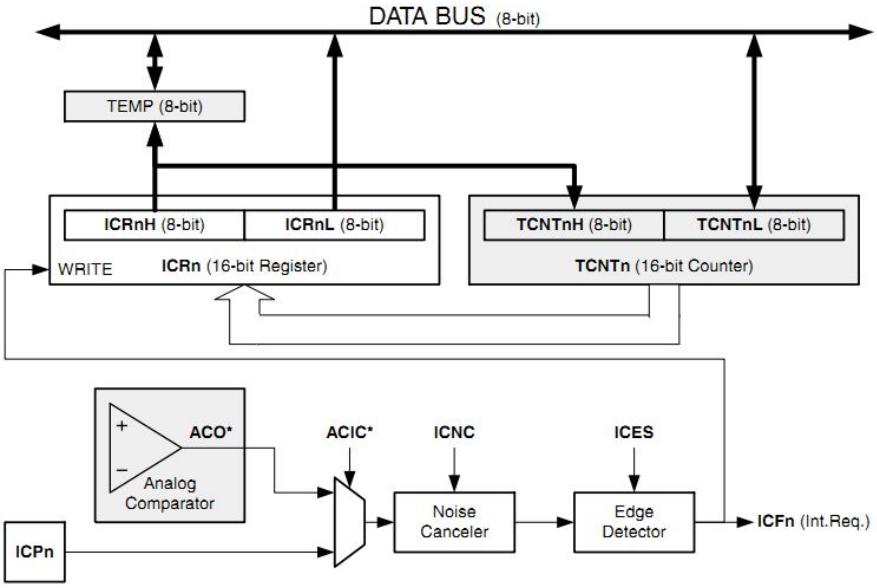


Рисунок 9.8 – Блок-схема порівняння виходу

3.3. Блок захоплення порівняння виходу

Біти режиму порівняння виходу COM1x1:0 мають дві функції. Перша – генератор форми сигналу використовує біти COM1x1:0 для визначення стану OC1x (Output Compare) при наступному порівнянні співпадіння. Друга – біти COM1x1:0 контролюють джерело виходу виводу OC1x. На рис. 9.9 показана спрощена блок схема логіки, на яку впливає значення біта COM1x1:0.

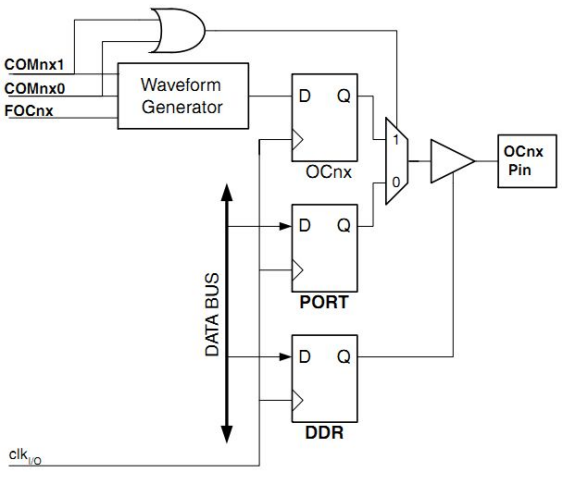


Рисунок 9.9 – Блок-схема логіки порівняння захоплення

Функція порту I/O загального призначення перевизначена порівнянням виходу регістра OC1x з генератора форми сигналу, якщо біт COM1x1:0 встановлений. В той час як вивід напряму OC1x все ще контролюється регістром DDR (Data Direction Register). Біт напряму для виводу DDR.OCx1 повинен бути встановлений як вихідний, тому що значення OCx1 є доступним на виводі.

Режими функціонування таймера/лічильника TCNT1 і виводів порівняння виходу визначаються комбінацією режимів WGM13:0 (Waveform Generation mode) і бітів COM1x1:0. Біти COM1x1:0 визначають інвертований і неінвертований режим роботи ШІМ (PWM). Генератор форми сигналів по різному використовує біти COM1x1:0 в нормальному, PWM і CTC режимах.

Normal режим

У найпростішому *нормальному (*Normal) режимі WGM13:0 = 0, лічильник тільки інкрементується і не очищається. Лічильник просто при досягненні свого максимального 16-розрядного значення (MAX=0xFFFF) скидається у значення BOTTOM (0x0000).

Режим Fast PWM (Pulse Width Modulation).

У цьому режимі значення лічильника змінюється від BOTTOM до TOP, рис. 9.10. Якщо налаштувати генератор так, що коли значення у лічильному регістрі більше ніж у регістрі порівняння, то на виході буде 1, а коли менше – 0. Якщо збільшити значення у регістрі порівняння, то ширина імпульсів стане меншою.

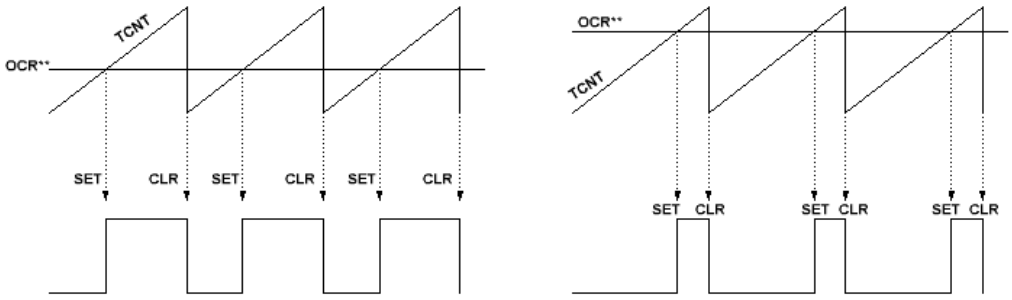


Рисунок 9.10 – Швидкий ШІМ

Недолік режиму Fast PWM у зсуві фаз між вихідними сигналами при зміні значення в регістрі OCR. Цей недолік усунуто у фазо-коректному ШІМ (phase correct PWM) де значення лічильника змінюється від BOTTOM до TOP і від TOP до BOTTOM, рис. 9.11. Коректні ШІМ використовуються для створення трифазних синусоїд

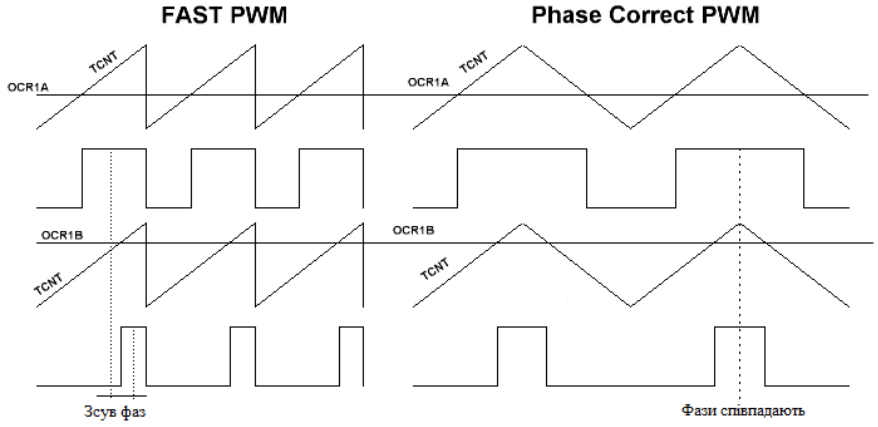


Рисунок 9.11 – Швидкий і фазокоректний ШІМ

У режимі швидкої широтно імпульсної модуляції (швидкий ШІМ) (WGM13:0 = 5, 6, 7, 14, або 15) використовується тільки інкремент лічильника від BOTTOM до TOP. За рахунок цього досягається частота сигналів майже в два рази більша, ніж у режимах з інкрементом та декрементом лічильника. Тому швидкий ШІМ використовується для регулювання потужності, у випрямлячах, ЦАП. Висока частота дозволяє використовувати малорозмірні зовнішні компоненти (котушки, конденсатори).

Роздільна здатність швидкого ШІМ може бути фіксованою 8-, 9- 10-біт або задаватися ICR1 або OCR1A. Мінімальна роздільна здатність 3-біти, а максимальна – 16-бітів.

Прапор Timer/Counter Overflow Flag (TOV1) встановлюється кожний раз при досягненні лічильником значення TOP. В той самий такт встановлюються прапори OC1A або ICF1 (в залежності від того, які регістри OCR1A або ICR використовувалися при заданні значення TOP). Якщо одне з переривань дозволене, то підпрограма оброблення переривань може використовуватися для встановлення нового значення TOP.

Часова діаграма режиму швидкий ШІМ показана на рис. 9.12.

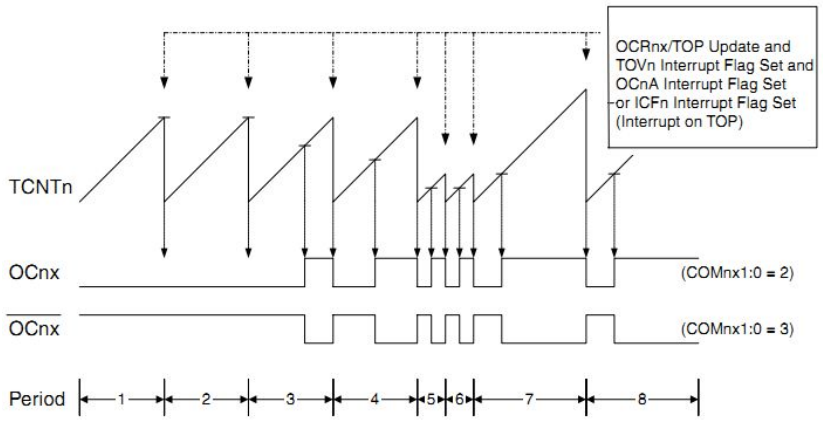


Рисунок 9.12 – Часова діаграма режиму швидкий ШІМ

Режим Phase Correct PWM.

Режим фазо-коректного ШІМ (WGM13:0 = 1, 2, 3, 10 або 11) використовує інкрементно-декрементну роботу лічильника від BOTTOM до TOP і від TOP до BOTTOM. У неінвертованому режимі порівняння OC1x очищується при співпадінні порівняння TCNT1 і OCR1x при інкременті лічильника і встановлюється при декременті лічильника. Такі інкрементно-декрементні режими мають меншу операційну частоту, але завдяки симетричній формі вони використовуються для керування 3-фазними моторами.

У режимі фазо-коректного ШІМ лічильник може досягати значення 0x00FF, 0x01FF, 0x03FF (WGM13:0 = 1, 2, or 3) або значення в ICR1 (WGM13:0 = 10) або значення OCR1A (WGM13:0 = 11). Лічильник при досягненні значення TOP міняє свій напрямок.

Часова діаграма режиму фазо-коректного ШІМ показана на рис. 9.13. На діаграмі показано неінвертований і інвертований вихід ШІМ. Невеликі горизонтальні лінії на TCNT1 позначають співпадіння порівнянь між OCR1x та TCNT1. Прапор OC1x встановлюється при співпадінні порівнянь.

Прапор Timer/Counter Overflow Flag (TOV1) встановлюється кожний раз при досягненні лічильником значення BOTTOM. В той самий такт, коли OCR1x присвоюється значення TOP, встановлюються прапори OC1A або ICF1 (в залежності від того, які регістри OCR1A або ICR використовувалися при заданні значення TOP).

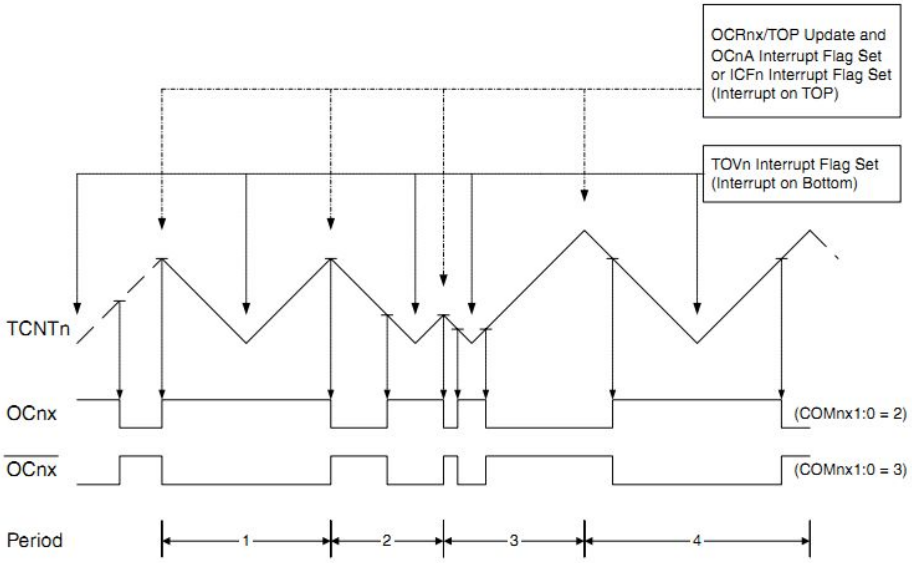


Рисунок 9.13 – Часова діаграма режиму фазо-коректний ШІМ

Режим CTC.

У режимі *очищення часу при порівнянні* (clear timer on compare, CTC) WGM13:0 = 4 або 12, використовуються регістри OCR1A або ICR1 для маніпуляції із роздільною здатністю лічильника. В режимі CTC лічильник очищається коли значення TCNT1 співпадає з OCR1A (WGM13:0 = 4) або з ICR1 (WGM13:0 = 12). Регістри OCR1A і ICR1 визначають TOP значення лічильника і його роздільну здатність.

Часова діаграма CTC режиму показана на рис. 9.14. Значення лічильника TCNT1 збільшується аж до співпадіння при порівнянні з OCR1A або ICR1, після чого він очищається.

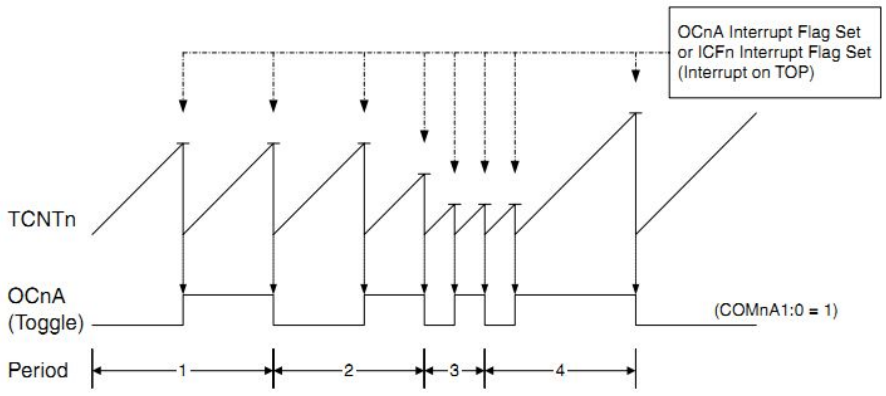


Рисунок 9.14 – Часова діаграма режиму CTC

Переривання генеруються кожний раз, коли лічильник досягне значення TOP з використанням прапора OCF1A або ICF1, відповідно до того, який регістр задав значення TOP. Якщо дозволені переривання, то підпрограма обробки переривань може бути використана для зміни значення TOP.

4. Режим таймера

Принцип роботи таймера/лічильника T1 в режимі таймера такий же, як і таймера/лічильника T0. За кожним імпульсом, поступаючим на тактовий вхід таймера/лічильника, інкрементується вміст лічильника TCNT1. При переході таймера/лічильника із стану SFFFF в стан $0000 встановлюється прапор TOV1 регістра TIFR і генерується запит на переривання. Дозвіл переривання здійснюється встановленням в 1 розряду TOIE1 (біт 7) регістра маски TIMSK (вважається, що прапор загального дозволу переривань I регістра SREG також повинен бути встановлений в 1).

Крім основних функцій – підрахунку числа імпульсів тактового сигналу МК і числа зовнішніх подій – в режимі таймера T1 доступні і додаткові.

4.1. Функція захоплення (capture)

Дана функція полягає у збереженні у визначений момент часу стану лічильника/таймера TCNT1 в регістрі захоплення ICR1. Ця дія може виконуватися або по активному фронту сигналу на виводі ICP МК (лінія порту PE0), або по сигналу від аналогового компаратора, які визначають сигнал захоплення на вході блоку керування таймера. При цьому встановлюється прапор ICF1 регістра TIFR і генерується запит на переривання. Дозвіл переривання здійснюється встановленням в 1 розряду TICIE1 (біт 3) регістра TIMSK.

Для керування схемою захоплення використовують два розряди регістра TCCR1B: ICNC1 і ICES1. Розряд ICNC1 керує схемою подавлення завад. Якщо цей розряд скинутий в 0, схема подавлення завад виключена і захоплення здійснюється по першому ж активному фронту на виводі ICP МК. Якщо цей розряд встановлений в 1, то при появленні активного фронту на виводі ICP схема керування виконує чотири вибірки з частотою, рівною тактовій частоті МК. Захоплення буде виконано тільки в тому випадку, якщо всі вибірки мають рівень, який відповідає активному фронту сигналу (рівень 1 – для наростаючого фронту і рівень 0 – для спадаючого).

Активний фронт сигналу, по якому буде збережено вміст лічильника TCNT1 в регістрі захоплення, 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 наростаючий фронт.

Фізично регістр захоплення ICR1 розміщений в двох регістрах ICR1H:ICR1L, доступних тільки для читання. Оскільки регістр захоплення є 16-розрядним, при його читанні, використовується спеціальний регістр TEMP. При читанні регістра ICR1L (молодший байт) його вміст посилається в процесорний пристрій, а вміст регістра ICR1H (старший байт) зберігається в регістрі TEMP. При читанні регістра ICR1H повертається значення, збережене в регістрі TEMP. Отже, при читанні регістра ICR1 першим має бути прочитаний регістр ICR1L. Переривання на час звернення до регістрі ICR1 повинні бути заборонені.

4.2. Функція порівняння (compare)

Дана функція полягає в неперервному (кожний машинний цикл) порівнянні вмісту лічильника TCNT1 і регістра порівняння. При співпадінні їх вмісту встановлюється прапор відповідного переривання. В МК ATMega8515 є два регістри порівняння (OCR1A і OCR1B), причому операція порівняння виконується для кожного регістра окремо. Якщо значення лічильника стає рівним числу, яке знаходиться в регістрі порівняння, то в наступному машинному циклі встановлюється відповідний цьому регістру прапор переривання в регістрі TIFR (для регістра OCR1A – прапор OCF1A, для регістра OCR1B – прапор OCF1B) і генерується запит на переривання. Дозвіл переривань здійснюється встановленням в 1 відповідних розрядів регістра TIMSK (OCIE1A – біт 6 для запиту OCF1A і OCIE1B – біт 5 для запиту OCF1B). Поряд з встановлення прапора в регістрі TIFR при рівності лічильника і регістра порівняння можуть виконуватися і другі дії: скидання таймера/лічильника (тільки для регістра OCR1A) і зміна стану визначеного виводу МК (для обох регістрів). Поведінка МК, тобто виконання або невиконання вказаних дій, визначається декількома розрядами регістрів керування TCCR1A і TCCR1B. Стан розрядів COM1x1, COM1x0 (x = A,B) визначає поведінку виводу OC1x при співпадінні вмісту лічильника TCNT1 і регістра порівняння OCRlx згідно табл. 9.8. При зміні стану цих розрядів відповідні переривання від схеми компаратора рекомендується заборонити (для запобігання фальшивої генерації переривання). Щоб таймер/лічильник міг керувати виводом, останній має бути сконфігурований як вихід порту.

Якщо розряд CTC1 регістра керування TCCR1B встановлений в 1, то при співпадінні вмісту лічильника TCNT1 і регістра порівняння OCR1A відбувається скидання таймера/лічильника в нульовий стан. Кожний регістр порівняння фізично розміщується у двох регістрах вводу/виводу: OCR1A – OCRlAH:OCRlAL; OCR1B – OCR1BH:OCR1BL.

Оскільки регістри порівняння є 16-розрядними, при їх читанні і запису використовується спеціальний регістр TEMP. Процес запису і читання 16-розрядних регістрів виконується так само, як і для регістра TCNT1. Переривання на час звернення до регістрів порівняння OCR1A і OCR1B повинні бути заборонені.

Таблиця 9.8 – Керування виводом OC1x (x=A, B)

COM1x1	COM1x0	Описання
0	0	Таймер/лічильник відключений від виводу OC1x
0	1	Стан виводу міняється на протилежний
1	0	Вивід скидається в 0
1	1	Вивід встановлюється в 1

4.3. Режим ШІМа (PWM)

Широтно-імпульсна модуляція є одним з видів неперервної імпульсної модуляції, при якій ширина імпульсу пропорційна значенню модульованого сигналу. Відповідно в даному випадку широтно-імпульсна модуляція полягає в генеруванні сигналу з програмованою частотою і шпаруватістю. Для переведення таймера/лічильника Т1 в режим ШІМа і задання частоти ШІМ-сигналу використовують розряди PWM11:PWM10 регістра керування таймером TCCR1A. Відповідність між станом цих розрядів і режимом роботи таймера/лічильника Т1 показана в табл. 9.9.

Таблиця 9.9 – Керування режимом ШІМ таймера/лічильника Т1

PWM11	PWM10	Описання
0	0	Режим ШІМ таймера/лічильника включений
0	1	8-розрядний ШІМ
1	0	9-розрядний ШІМ
1	1	10-розрядний ШІМ

Для генерації ШІМ-сигналу використовується схема порівняння таймера/лічильника, тому в МК ATx8515 модулятор є здвоєним (два регістри порівняння). Назви регістрів порівняння і правила звернення до них були описані раніше. Сигнал знімається з виходу схеми порівняння таймера/лічильника.

В розглядуваному режимі лічильник TCNT1 функціонує як реверсивний, модуль рахунку якого (TOP) залежить від режиму роботи модулятора. Частота ШІМ-сигналу залежить від частоти тактового сигналу f_{TCK1} таймера/лічильника Т1 і модуля рахунку ШІМа. Значення модуля рахунку і частота ШІМ-сигналу для кожного режиму роботи модулятора показані в табл. 9.10.

Таблиця 9.10 – Режими ШІМ

Режим модулятора	Модуль рахунку (TOP)	Частота ШІМ-сигналу
8-розрядний	255	f_{TCK1}/510
9-розрядний	511	f_{TCK1}/1022
10-розрядний	1023	f_{TCK1}/2046

При роботі таймера/лічильника Т1 в режимі ШІМ стан лічильника змінюється від 0 до значення TOP, а потім знову до 0, після чого цикл повторюється. При рівності стану лічильника і вмісту регістра порівняння стан відповідного цьому регістру вивода МК змінюється згідно табл. 9.11 (x позначає A або B). Таким чином, тривалість ШІМ-сигналу дорівнює $2n/ f_{TCK1}$, де n – вміст регістру порівняння.

Таблиця 9.11 – Поведінка виводів схеми порівняння в режимі ШІМ

COM1x1	COM1x0	Поведінка вивода OC1x
0	0	Таймер лічильник Т1 відключений від вивода
0	1	OC1A в режимі порівняння, OC1B відключений від вивода
1	0	Скидається в 0 при прямому рахунку і встановлюється в 1 при зворотному (неінвертований ШІМ сигнал)
1	1	Встановлюється в 1 при прямому рахунку і скидається в 0 при зворотному (неінвертований ШІМ сигнал)

Відповідно, якщо в регістр порівняння записати значення 0 або TOP, то при наступному співпадінні стану лічильника і вмісту регістра порівняння вихід схеми порівняння переключиться в стійкий стан згідно табл. 9.12 (x = A або B).

Таблиця 9.12 – Стійкі стани виходу системи порівняння

COM1x1	COM1x0	Регістр OCR1x	Стан виводу OCR1x
1	0	0	0
1	0	TOP	1
1	1	0	1
1	1	TOP	0

Особливістю роботи таймера/лічильника T1 в режимі ШІМ є те, що при запису в регістр порівняння, молодші 10 розрядів записуваного числа в дійсності зберігаються в спеціальному тимчасовому регістрі. Зміна вмісту регістра порівняння відбувається тільки в момент досягнення лічильником максимального значення TOP. Завдяки такому рішенню запобігається появлення в ШІМ-сигналі імпульсу із випадковою тривалістю. Відповідно при читанні регістра порівняння в проміжку між записом в нього і дійсною зміною, повертається вміст тимчасового регістра, тобто завжди повертається значення записане останнім.

При роботі таймера/лічильника T1 в режимі ШІМ може генеруватися переривання по переповненню лічильника, а також переривання від схеми порівняння. Прапори переривань встановлюються в 1 при зміні лічильником напрямку рахунку: прапор TOV1 – в точці 0, а прапори OCF1A (для регістра OCR1A) і OCF1B (для регістра OCR1B) – в точці TOP. Дозволи і обробка відповідних переривань виконуються як звичайно.

5. Сторожовий таймер

Основна функція сторожового таймера (Watchdog Timer) – захист МК пристроїв від збоїв. Використовуючи сторожовий таймер можна перервати виконання зацикленої програми або вийти з інших непередбачуваних ситуацій. Структура сторожового таймера показана на рис. 9.14.

Для керування сторожовим таймером призначений регістр WDTCR. Формат регістра показаний в табл. 9.13.

Таблиця 9.13 – Формат регістра WDTCR

Розряд	7	6	5	4	3	2	1	0
Ім’я	–	–	–	WDCE	WDE	WDP2	WDP1	WDP0

Для включення/виключення сторожового таймера використовують два розряди регістра WDTCR – WDE і WDCE. Якщо розряд WDE встановлений в 1, сторожовий таймер включений, якщо скинутий в 0 – виключений. Безпосередньо перед включенням таймера рекомендується також скинути його командою RESET.

Для запобігання випадкового виключення таймера призначений розряд WDCE. Справа в тому, що виключення сторожового таймера (скидання розряду WDE) можна здійснити тільки при встановленому розряді WDCE, який через чотири машинних такти після встановлення в 1 апаратно скидається. Завдяки цьому можливість випадкового виключення сторожового таймера практично виключається.

Виходячи із цього, для виключення сторожового таймера рекомендується наступний порядок дій:

– однією командою записати 1 в розряди WDE і WDCE;

– за наступні чотири машинні тактів записати 0 в розряд WDE.

Період виключення сторожового таймера задається за допомогою розрядів WDP2-WDP0 регістра WDTCR згідно табл. 9.14.

Щоб запобігти випадковому скиданню МК при зміні періоду сторожового таймера, необхідно перед записом розрядів WDP2-WDP0 заборонити роботу таймера або скинути його командою RESET.

Таблиця 9.14 – Задання періоду сторожового таймера для ATmega8515

WDP2	WDP1	WDP0	Число тактів генератора	Період виключення	
				V_{CC}=3,0 В	V_{CC}=5,0 В
0	0	0	16·K (1024)	17,1 мс	16,3 мс
0	0	1	32·K	34,3 мс	32,5 мс
0	1	0	64·K	68,5 мс	65 мс
0	1	1	128·K	0,14 с	0,13 с
1	0	0	256·K	0,27 с	0,26 с
1	0	1	512·K	0,55 с	0,52 с
1	1	0	1024·K	1,1 с	1,0 с
1	1	1	2048·K	2,2 с	2,1 с

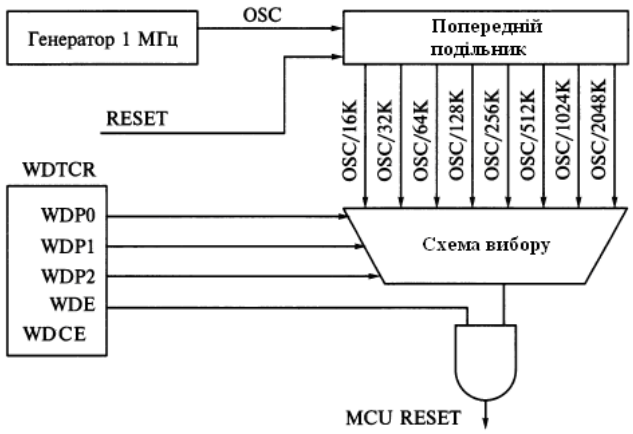


Рисунок 9.14 – Структурна схема сторожового таймера

Висновки.

Мікроконтролери (МК) AVR в залежності від класу (Tiny, Classic, Mega) і типу моделі мають в своєму складі від одного до трьох таймерів/лічильників загального призначення – T0, T1 і T2.

Перший таймер (8-розрядний T0), який є у всіх моделях, може використовуватися для відліку і вимірювання часових інтервалів або як лічильник зовнішніх подій, а в моделі ATmega8515 ще і для порівняння із заданим значенням.

Два інших таймери (16-розрядний T1 і 8-розрядний T2) крім названих мають ще додаткові функції. Обидва таймери можуть генерувати запит на переривання не тільки при переповненні лічильного регістра, але і від ряду інших подій. Вони можуть також використовуватися як широтно-імпульсні модулятори. Крім того, таймер T2 може працювати в асинхронному (відносно тактового сигналу МК) режимі.

Питання.

1. Які виводи використовують таймери/лічильники МК ATmega8515?
2. Розказати про структурну схему таймера/лічильника T0?
3. Які функції може виконувати таймер/лічильник T0?
4. Розказати про структурну схему таймера/лічильника T1?
5. Які функції може виконувати таймер/лічильник T1?
6. Пояснити роботу таймера/лічильника T1 в режимі таймера?
7. Пояснити роботу таймера/лічильника T1 в режимі захоплення?
8. Пояснити роботу таймера/лічильника T1 в режимі порівняння?
9. Пояснити роботу таймера/лічильника T1 в режимі ШІМ?
10. Який максимальний час захоплення можна зареєструвати при роботі програми прикладу 4?
11. Що таке шпаруватість сигналів? Як змінюється часова діаграма вихідного сигналу при зміні значення, яке завантажується в регістр порівняння в режимі ШІМ?
12. Які значення необхідно занести в регістри порівняння в програмі прикладу 5 для формування ШІМ-сигналів шпаруватістю 4 (відношення періоду до тривалості сигналу) при тому ж періоді ШІМ-сигналів?
13. Який максимальний час виключення сторожового таймера ATmega8515?

ЛЕКЦІЯ 10. Послідовний обмін даними по каналу UASRT

Мета. Вивчення передачі і прийому інформації по послідовному каналу USART (Universal Asynchronous Reciever-Trasmitter) та програмування введення/виведення

Вступ. МК AVR мають у своєму складі модуль повно дуплексного універсального асинхронного/синхронного приймач/передавача UASRT. Через нього можна передавати і приймати інформацію заданому послідовним кодом. За допомогою цього модуля мікроконтролер (МК) може обмінюватися даними з різними периферійними пристроями.

План.

1. Організація передачі даних через UART/USART
2. Регістри USART
3. Формат передачі кадру даних USART
4. Підключення USART
5. Швидкість прийому/передачі
6. Ініціалізація USART. Передача та приймання даних

1. Організація передачі даних через UART/USART

Універсальний асинхронний/синхронний послідовний приймач/передавач (USART) забезпечує обмін даними МК AVR із зовнішніми пристроями по послідовному каналу в повнодуплексному режимі. Основні можливості USART:

- повнодуплексна взаємодія;
- асинхронний і синхронний режим;
- синхронна (за тактовими сигналами) робота master-slave;
- налаштовування швидкості передачі даних;
- підтримка фреймів даних розміром 5, 6, 7, 8 і 9 біт, 1 або 2 Stop біти;
- генерація бітів парності/непарності і апаратна їх перевірка;
- виявлення перекриття даних;
- виявлення помилки фреймів;
- фільтрування шумів;
- три окремих переривання TX Complete, TX Data Register Empty і RX Complete;
- режим багатопроцесорної взаємодії;
- асинхронний режим з подвоєною швидкодією.

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

При *асинхронній* передачі даних для синхронізації використовують стартовий та стоповий біти, які визначають початок та кінець передачі слова даних. Асинхронна передача даних використовується для комунікації пристроїв та блоків, розділених у просторі та які мають певну автономність один від одного. Наприклад, між ПК та принтером, між пристроєм на базі МК та комп’ютером.

Модуль USART підтримує як синхронний, так і асинхронний режими роботи. Однак на практиці його найчастіше використовують саме в асинхронному режимі, а синхронний режим реалізують за допомогою модуля SPI. Тому буде розглядатися тільки асинхронний режим UART.

При взаємодії з програмою в модулі USART передбачені переривання при виникненні наступних подій:

```
$009    rjmp USART_RXC      ; прийом завершено
$00a    rjmp USART_UDRE     ; регістр даних передавача порожній
$00b    rjmp USART_TXC      ; передача завершена
```

Модуль USART використовує виводи МК – для входу даних RxD (PD0), для виходу TxD (PD1). 8-розрядів, які передаються і приймаються зберігаються у регістрі UDR.

Блок-схема USART МК ATmega8515 показана на рис. 10.1.

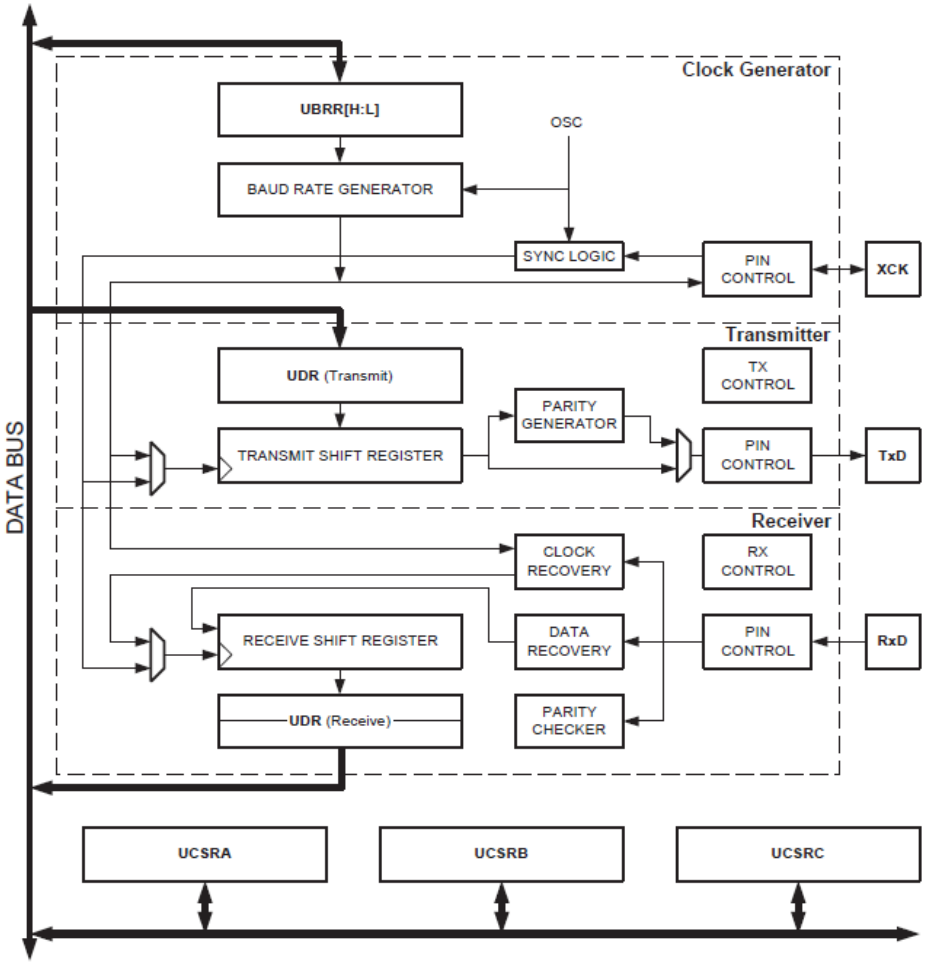


Рисунок 10.1 – Блок схема модуля USART

Модуль USART складається з 3-х частин: тактового генератора швидкості передачі, блоків передавача та приймача.

Блок передавача містить однорівневий буфер (UDR), зсувний регістр (transmit shift register), схему формування біта парності (parity generator) та схему керування виводами (pin control, tx control).

Блок приймача містить дворівневий буфер (UDR), зсувний регістр (receive shift register) схеми відновлення тактового сигналу (clock recovery) та даних (data recovery), схему контролю

парності (parity checker) та схему керування (pin, rx).

Буферні регістри приймача та передавача мають одну адресу в просторі введення/виведення та спільне позначення як регістр даних UDR. У цьому регістрі зберігаються молодші 8-розрядів даних, що приймаються чи передаються. При читанні регістра UDR відбувається звернення до буферного регістра приймача, а при записуванні – до буферного регістра передавача.

У модулях USART буфер приймача дворівневий (FIFO-буфер). При будь-якому зверненні до регістра UDR цей буфер змінює свій стан. Тому необхідно спочатку зчитати з нього дані, а потім вже виконувати інші операції.

Регістр тактового генератора швидкості передачі UBRR задає необхідний коефіцієнт ділення для системного тактового сигналу, після чого цей сигнал ще подається на додаткові дільники, які вибираються за значенням біту U2X регістра UCSRA.

Схема відновлення тактового сигналу у приймачі призначена для синхронізації внутрішнього тактового сигналу, що формується тактовим генератором швидкості передачі, з пакетами даних, які поступають на вивід RxD. Схема відновлення даних зчитує та фільтрує кожний розряд отримуваного пакету.

2. Регістри USART

Регістр даних введення/виведення USART- UDR:

RXB[7-0]	UDR (Read)
TXB[7-0]	UDR (Write)

При записуванні дані поміщаються в буферний регістр передавача TXB. Записування даних можливе, коли встановлений прапор UDRE у керуючому регістрі UCSRA. Записані дані передаються у зсувний регістр передавача (якщо він порожній) і послідовно передаються на вивід TxD.

Буферний регістр отримувача складається з дворівневого FIFO. FIFO змінює свій стан при любому доступі до буфера отримувача. У зв’язку з цим не можна використовувати модифікуючі інструкції запису (SBI або CBI) у цьому місці. Також треба бути уважним при використання інструкцій тестування SBIC або SBIS, так як вони також будуть змінювати стан FIFO.

Для керування роботою і визначення поточного стану USART використовуються регістри UCSRA, UCSRB, UCSRC:

RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA

RXC: USART Receive Complete – прапор встановлений коли у буфері отримуча є непрочитані дані і скинутий в 0 – якщо буфер порожній.

TXC: USART Transmit Complete – прапор встановлений коли увесь фрейм покинув зсувний регістр передача і відсутні нові дані в буфері передавачі UDR.

UDRE: USART Data Register Empty – прапор встановлений в 1 то буфер порожній і готовий до запису даних.

FE: Frame Error – прапор встановлений в 1 коли стоп біт дорівнює 1, і встановлюється в 0 при помилці стоп біта.

DOR: Data OverRun – прапор встановлюється в 1 коли буфер отримувача повний (містить два символи) і виявляється новий стартовий біт.

PE: Parity Error – прапор встановлюється в 1 коли наступний символ у буфері отримувача має помилку парності.

U2X: Double the USART Transmission Speed – прапор встановлюється в 0 при асинхронних операціях.

MPCM: Multi-processor Communication Mode прапор встановлюється в 1 при багатопроцесорних операціях.

RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB

RXCIE: RX Complete Interrupt Enable – прапор 1 дозволяє переривання за прапором RXC.

TXCIE: TX Complete Interrupt Enable – прапор 1 дозволяє переривання за прапором TXC.

UDRIE: USART Data Register Empty Interrupt Enable – прапор 1 дозволяє переривання за прапором UDRE.

RXEN: Receiver Enable – прапор (1) дає доступ до отримувача.

TXEN: Transmitter Enable – прапор (1) дає доступ до передавача.

UCSZ2: Character Size – біт UCSZ2 разом з бітом UCSZ1:0 в UCSRC задає розмір символу (число бітів у фреймі) в передавачі або отримувачі.

RXB8: Receive Data Bit 8 – 9-й біт даних, має бути прочитаний перед читанням молодших бітів (для фреймів з 9-бітів).

TXB8: Transmit Data Bit 8 – 9-й біт даних, має бути записаний перед записуванням молодших бітів (для фреймів з 9-бітів).

URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC

Регістр UCSRC розміщений в одному адресному просторі з регістром UBRRH.

URSEL: Register Select – біт вибору між UCSRC і UBRRH. Він читається як 1 при читанні UCSRC.

UMSEL: USART Mode Select – біт вибору між асинхронним (0) і синхронним (1) режимами.

UPM1:0: Parity Mode – встановлення дозволу і вибір типу генерації і перевірки контролю парності.

UPM1	UPM0	Режим
0	0	Заборонено
0	1	Резерв
1	0	Контроль парності
1	1	Контроль непарності

7 бітів даних	Число одиничних бітів	8 бітів з бітом парності	
		парність	непарність
0000000	0	0000000**0**	0000000**1**
1010001	3	1010001**1**	1010001**0**

1101001	4	1101001**0**	1101001**1**
1111111	7	1111111**1**	1111111**0**

USBS: Stop Bit Select – задання числа Stop бітів, які будуть вставлені передавачем. USBS=0 (1-біт), USBS=1 (2-біти).

UCSZ1:0: Character Size – разом з UCSZ2 в UCSRB задає розмір символу (число бітів у фреймі) для отримувача і передавача.

UCSZ2	UCSZ1	UCSZ0	Розмір символу
0	0	0	5-біт
0	0	1	6-біт
0	1	0	7-біт
0	1	1	8-біт
1	0	0	резерв
1	0	1	резерв
1	1	0	резерв
1	1	1	9-біт

UCPOL: Clock Polarity – використовується тільки в синхронному режимі. Встановлює залежність між змінами даних і сигналами синхронізації.

UCPOL	Зміна даних при передачі (Виведення на TxD вивід)	Вибір отриманих даних (Введення в RxD вивід)
0	Наростаючий фронт XCK	Спадаючий фронт XCK
1	Спадаючий фронт XCK	Наростаючий фронт XCK

Для керування швидкістю передачі даних використовується регістр UBRRH/UBRRL:

URSEL	-	-	-	UBRR[11:8]	UBRRH
UBRR[7:0]					UBRRL

Bit 15 – URSEL: Register Select – біт вибирає між доступом до UBRRH або UCSRC. Читається як 0 при читанні з UBRRH.

UBRR11:0: USART Baud Rate Register – 12 бітовий регістр, який містить швидкість передачі даних.

2. Формат передачі кадру даних USART

Формат передачі кадру USART за своєю структурою (рис. 10.2) ідентичний інтерфейсу RS-232, з тією лише відмінністю, що в інтерфейсі RS-232 логічні рівні формуються напругами від ±3 до ±12В, а в модулі UART логічні рівні відповідають TTL-рівням (0 та 5 В).

Початок кадру даних завжди задається низьким рівнем стартового біту. Після цього передається байт даних (5-9 бітів) починаючи з молодших розрядів. Якщо використовується перевірка парності бітів, то наступним передається біт парності, що доповнює байт даних “1” або “0” так, щоб кількість одиниць в байті даних була парною (при опції “Парність”) або непарною (при опції «Непарність»). Це дозволяє виявляти непарну кількість спотворених бітів.

Останніми передаються стоп біти (1 або 2), що задаються високим рівнем. Якщо на лінії передача даних відсутня, тоді на ній завжди присутній високий рівень.

Швидкість передачі вимірюється у бодах (baud) – кількістю бітів переданих за секунду (bps). Формат кадру задається відповідними бітами регістрів керування UCSRB та UCSRC.

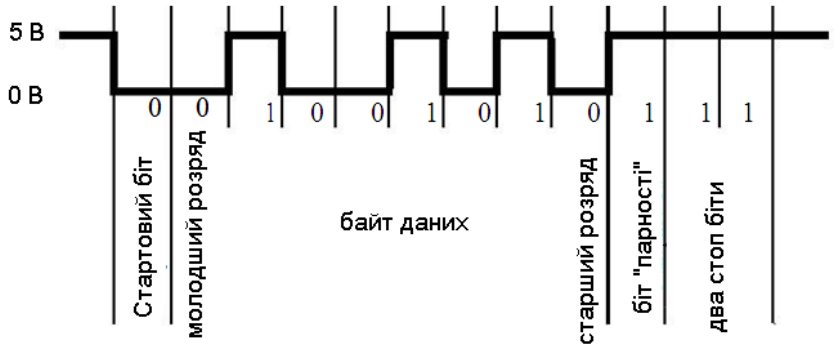


Рисунок 10.2 – Формат передачі кадру даних USART

Таблиця 10.1 – Задання розміру байта даних

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

4. Підключення USART

У МК AVR протокол передачі даних UART реалізований апаратно. Приймач даних під'єднаний до виводу RxD, а передавач до виводу TxD. При з'єднанні між собою двох модулів UART для передачі даних, необхідно з'єднати навхрест між собою виводи модулів передачі та приймання, як показано на рис. 10.3.

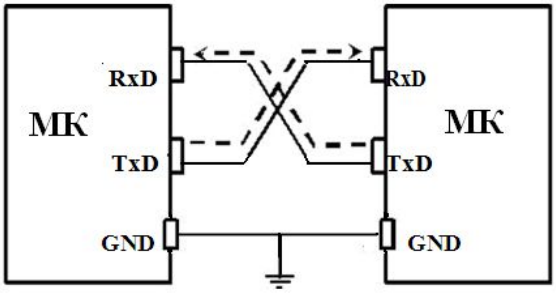


Рисунок 10.3 – Схема з'єднання двох модулів USART

5. Швидкість прийому/передачі

Блок тактового генератора має контролер швидкості передачі (baud rate generator), який керує швидкістю передачі. Контролер є звичайним подільником частоти системного тактового сигналу з програмованим коефіцієнтом поділу, значення якого знаходиться у регістрі UBRR. Регістр UBRR є 12-розрядний та фізично розміщується у двох регістрах UBRRH та UBRRL.

В асинхронному режимі швидкість обміну визначається не лише значенням регістра UBRR, але і станом розряду U2X у регістрі керування UCSRA. Якщо цей біт встановлений в “1”, то коефіцієнт поділу подільника зменшується у 2 рази, а швидкість, відповідно, подвоюється.

Швидкість обміну в асинхронному режимі визначається за наступними формулами:

при U2X=0 BAUD = f_{CLK} / (16·(UBRR+1), UBRR = f_{CLK}/(16·BAUD) – 1
при U2X=1 BAUD = f_{CLK} / (8·(UBRR+1), UBRR = f_{CLK}/(8·BAUD) - 1

Швидкість обміну в синхронному (Master) режимі визначається за наступною формулою:

BAUD = f_{CLK} / (2·(UBRR+1), UBRR = f_{CLK}/(2·BAUD) – 1

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

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

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

Значення UBRR, які дозволяє отримати ці швидкості передачі при використанні різних резонаторів та отримувані при цьому похибки відносно теоретичних значень показні в табл. 10.2-10.5.

Варто звернути увагу, що на частотах тактового генератора рівних 3.6864, 7.3728, 11.0592 та 14.7456 МГц похибка передачі є рівною нулю для усієї лінійки стандартних швидкостей.

Таблиця 10.2 – Значення UBRR для різних f_{CLK}

Baud rate, bps	Fosc = 1.0 МГц				Fosc = 1.8432 МГц				Fosc = 2.0 МГц			
	U2X=0		U2X=1		U2X=0		U2X=1		U2X=0		U2X=1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	25	0.2%	51	0.2%	47	0%	95	0%	51	0.2%	103	0.2%
4800	12	0.2%	25	0.2%	23	0%	47	0%	25	0.2%	51	0.2%
9600	6	-7.0%	12	0.2%	11	0%	23	0%	12	0.2%	25	0.2%
14.4K	3	8.5%	8	-3.5%	7	0%	15	0%	8	-3.5%	16	2.1%
19.2K	2	8.5%	6	-7.0%	5	0%	11	0%	6	-7.0%	12	0.2%
28.8K	1	8.5%	3	8.5%	3	0%	7	0%	3	8.5%	8	-3.5%
38.4K	1	-18.6%	2	8.5%	2	0%	5	0%	2	8.5%	6	-7.0%
57.6K	0	8.5%	1	8.5%	1	0%	3	0%	1	8.5%	3	8.5%
76.8K	-	-	1	-18.6%	1	-25%	2	0%	1	-18.6%	2	8.5%
115.2K	-	-	0	8.5%	0	0%	1	0%	0	8.5%	1	8.5%
230.4K	-	-	-	-	-	-	0	0%	-	-	-	-

250K	-	-	-	-	-	-	-	-	-	-	0	0%
Max[1]	62.5 kbsps		125 kbsps		115.2 kbsps		230.4 kbsps		125 kbsps		250 kbsps	

Таблиця 10.3 – Значення UBRR для різних f_{CLK}

Baud rate, bps	Fosc = 3.6864 МГц				Fosc = 4.0 МГц				Fosc = 7.3728 МГц			
	U2X=0		U2X=1		U2X=0		U2X=1		U2X=0		U2X=1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	95	0.0%	191	0.0%	103	0.2%	207	0.2%	191	0.0%	383	0.0%
4800	47	0.0%	95	0.0%	51	0.2%	103	0.2%	95	0.0%	191	0.0%
9600	23	0.0%	47	0.0%	25	0.2%	51	0.2%	47	0.0%	95	0.0%
14.4K	15	0.0%	31	0.0%	16	2.1%	34	-0.8%	31	0.0%	63	0.0%
19.2K	11	0.0%	23	0.0%	12	0.2%	25	0.2%	23	0.0%	47	0.0%
28.8K	7	0.0%	15	0.0%	8	-3.5%	16	2.1%	15	0.0%	31	0.0%
38.4K	5	0.0%	11	0.0%	6	-7.0%	12	0.2%	11	0.0%	23	0.0%
57.6K	3	0.0%	7	0.0%	3	8.5%	8	-3.5%	7	0.0%	15	0.0%
76.8K	2	0.0%	5	0.0%	2	8.5%	6	-7.0%	5	0.0%	11	0.0%
115.2K	1	0.0%	3	0.0%	1	8.5%	3	8.5%	3	0.0%	7	0.0%
230.4K	0	0.0%	1	0.0%	0	8.5%	1	8.5%	1	0.0%	3	0.0%
250K	0	-7.8%	1	-7.8%	0	0	1	0.0%	1	-7.8%	3	-7.8%
0.5M	-	-	0	-7.8%	-	-	0	0.0%	0	-7.8%	1	-7.8%
1M	-	-	-	-	-	-	-	-	-	-	0	-7.8%
Max[1]	230.4 kbsps		460.8 kbsps		250 kbsps		0.5 Mbsps		460.8 kbsps		921.6 kbsps	

Таблиця 10.4 – Значення UBRR для різних f_{CLK}

Baud rate, bps	Fosc = 8.0 МГц				Fosc = 11.0592 МГц				Fosc = 17.7456 МГц			
	U2X=0		U2X=1		U2X=0		U2X=1		U2X=0		U2X=1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	207	0.2%	416	-0.1%	287	0.0%	575	0.0%	383	0.0%	767	0.0%
4800	103	0.2%	207	0.2%	143	0.0%	287	0.0%	191	0.0%	383	0.0%
9600	51	0.2%	103	0.2%	71	0.0%	143	0.0%	95	0.0%	191	0.0%
14.4K	34	-0.8%	68	0.6%	47	0.0%	95	0.0%	63	0.0%	127	0.0%
19.2K	25	0.2%	51	0.2%	35	0.0%	71	0.0%	47	0.0%	95	0.0%
28.8K	16	2.1%	34	-0.8%	23	0.0%	47	0.0%	31	0.0%	63	0.0%
38.4K	12	0.2%	25	0.2%	17	0.0%	35	0.0%	23	0.0%	47	0.0%
57.6K	8	-3.5%	16	2.1%	11	0.0%	23	0.0%	15	0.0%	31	0.0%
76.8K	6	-7.0%	12	0.2%	8	0.0%	17	0.0%	11	0.0%	23	0.0%
115.2K	3	8.5%	8	-3.5%	5	0.0%	11	0.0%	7	0.0%	15	0.0%
230.4K	1	8.5%	3	8.5%	2	0.0%	5	0.0%	3	0.0%	7	0.0%
250K	1	0.0%	3	0.0%	2	-7.8%	5	0.0%	3	-7.8%	6	5.3%
0.5M	0	0.0%	1	0.0%	-	-	2	0.0%	1	-7.8%	3	-7.8%
1M	-	-	0	0.0%	-	-	-	-	0	-7.8%	1	-7.8%
Max[1]	0.5 Mbsps		1 Mbsps		691.2 kbsps		1.3824 Mbsps		921.6 kbsps		1.8432 Mbsps	

Таблиця 10.5 – Значення UBRR для різних f_{CLK}

Baud rate, bps	Fosc = 16.0 МГц				Fosc = 18.4320 МГц				Fosc = 20.0 МГц			
	U2X=0		U2X=1		U2X=0		U2X=1		U2X=0		U2X=1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	416	-0.1%	832	0.0%	479	0.0%	959	0.0%	520	0.0%	1041	0.0%
4800	207	0.2%	416	-0.1%	239	0.0%	479	0.0%	259	0.2%	520	0.0%
9600	103	0.2%	207	0.2%	119	0.0%	239	0.0%	129	0.2%	259	0.2%
14.4K	68	0.6%	138	-0.1%	79	0.0%	159	0.0%	86	-0.2%	173	-0.2%
19.2K	51	0.2%	103	0.2%	59	0.0%	119	0.0%	64	0.2%	129	0.2%
28.8K	34	-0.8%	68	0.6%	39	0.0%	79	0.0%	42	0.9%	86	-0.2%
38.4K	25	0.2%	51	0.2%	29	0.0%	59	0.0%	32	-1.4%	64	0.2%
57.6K	16	2.1%	34	-0.8%	19	0.0%	39	0.0%	21	-1.4%	42	0.9%
76.8K	12	0.2%	25	0.2%	14	0.0%	29	0.0%	15	1.7%	32	-1.4%
115.2K	8	-3.5%	16	2.1%	9	0.0%	19	0.0%	10	-1.4%	21	-1.4%
230.4K	3	8.5%	8	-3.5%	4	0.0%	9	0.0%	4	8.5%	10	-1.4%
250K	3	0.0%	7	0.0%	4	-7.8%	8	2.4%	4	0.0%	9	0.0%
0.5M	1	0.0%	3	0.0%	-	-	4	-7.8%	-	-	4	0.0%
1.0M	0	0.0%	1	0.0%	-	-	-	-	-	-	-	-
Max(1)	1 Mbsps		2 Mbsps		1.152 Mbsps		2.304 Mbsps		1.25 Mbsps		2.50 Mbsps	

1. UBRR=0, Error=0.0% (похибка визначає відхилення швидкості передачі).

Примітка. Похибка визначає відхилення від швидкості передачі. Максимальна тактова частота, яка встановлюється платою STK500, дорівнює 3,69 МГц.

6. Ініціалізація USART. Передача та приймання даних

Ініціалізацію USART необхідно виконати до передачі та приймання даних. Для ініціалізації USART необхідно налаштувати контролер швидкості передачі, режим функціонування і формат фрейму. Також необхідно перевірити прапори TXC, RXC, чи передача завершена і буфери порожні.

Для *активації* приймання/передачі модуля UART необхідно надати дозволи на роботу передавача та приймача, встановивши відповідні біти TXEN (Transmit Enable) та RXEN (Receive Enable) у регістрі керування UCSRB. Тоді відповідні виводи МК TxD та RxD, підключаються до модуля UART та працюють на передачу та приймання, незалежно від налаштувань регістрів керування портом, до якого вони належать.

Приклади простої функції ініціалізації USART на асемблері:

```
USART_Init:
   ; встановити швидкість передачі
   out UBRRH, r17
   out UBRRL, r16
   ; задати дозвіл на роботу приймача та передавача
   ldi r16, (1<<RXEN)|(1<<TXEN)
   out UCSRB,r16
   ; задати формат фрейму : 8-біт даних, 2 стоп біти
   ldi r16, (1<<URSEL)|(1<<USBS)|(3<<UCSZ0)
   out UCSRC,r16
ret
```

Для відправлення байту даних необхідно записати його значення у регістр даних UDR.

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

Приклади простої функції передачі даних з розміром фрейму 5÷8 біт на асемблері:

```
USART_Transmit:
   ; очікування, поки буфер передачі стане порожнім
   sbis UCSRA,UDRE
   rjmp USART_Transmit
   ; вивести дані (r16) у буфер, послати дані
   out UDR,r16
ret
```

Якщо передається фрейм з 9-бітів (UCSZ=7), то дев'ятий біт необхідно записати у біт TXB8 регістра UCSRB перед записом молодшого байту в регістр UDR.

В наступному прикладі показана функція передачі даних з розміром фрейму 9 біт на асемблері:

```
USART_Transmit:
   ; очікування, поки буфер передачі стане порожнім
   sbis UCSRA,UDRE
   rjmp USART_Transmit
   ; скопіювати 9-біту з r17 у TXB8
   cbi UCSRB,TXB8
   sbrc r17,0
   sbi UCSRB,TXB8
   ; вивести молодший байт даних (r16) у буфер, послати дані
   out UDR,r16
ret
```

USART працює в режимі прийняття даних при встановленому біті RXEN в регістрі UCSRB. *Приймання* даних починається з моменту виявлення приймачем коректного старт-біту. Кожний наступний біт кадру зчитується зі швидкістю, заданою для модуля UART, та розміщується у зсувному регістрі, аж поки не буде виявлений перший стоп-біт. Наступний стоп-біт ігнорується. Після цього вміст зсувного регістра пересилається у буфер приймача UDR, звідки прийняте значення може бути зчитаним.

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

Це правило відноситься також і до прапорів помилок FE, DOR і PE (регістр UCSRA), які можуть бути перевіреними ще перед читанням регістру даних UDR.

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

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

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

Приклади простої функції приймання даних із розміром фрейму 5÷8 біт на асемблері:

```
USART_Receive:
   ; очікування на отримання даних
   sbis UCSRA, RXC
   rjmp USART_Receive
```

```
  ; зчитування і повернення отриманих даних з буфера
  in r16, UDR
ret
```

В наступному прикладі показана функція приймання даних із розміром фрейму 9 біт на асемблері:

```
USART_Receive:
  ; очікування на отримання даних
  sbis UCSRA, RXC
  rjmp USART_Receive
  ; читання бітів помилок і 9-го біта, а потім даних з буфера
  in r18, UCSRA
  in r17, UCSRB
  in r16, UDR
  ; якщо помилка, повернути -1
  andi r18,(1<<FE)|(1<<DOR)|(1<<PE)
  breq USART_ReceiveNoError
  ldi r17, HIGH(-1)
  ldi r16, LOW(-1)
USART_ReceiveNoError:
  ; відфільтрувати 9-й біт, тоді вийти з функції
  lsr r17
  andi r17, 0x01
ret
```

Інтерфейс RS-232

Інтерфейс RS-232 забезпечує зв’язок між термінальним обладнанням та апаратурою передачі даних, використовуючи послідовний обмін двійковими даними. У стандарті передбачені асинхронний та синхронний режими обміну. Варто зазначити, що інтерфейс не забезпечує гальванічної розв’язки пристроїв. Рівні сигналів інтерфейсу RS-232 при прийманні і передаванні показані на рис. 10.4.

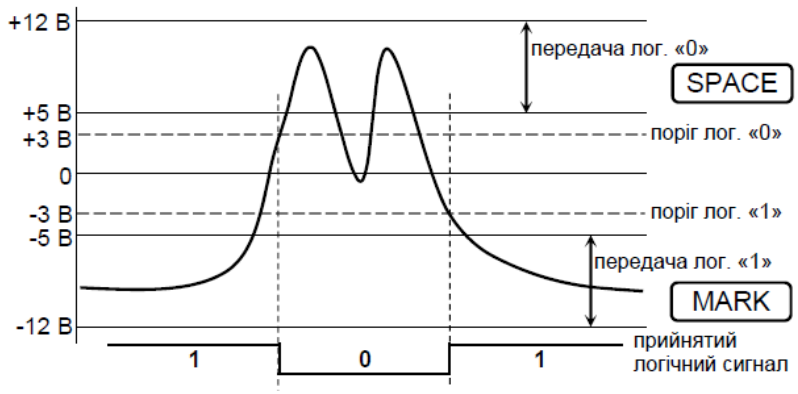


Рис. 10.4 – Рівні сигналів інтерфейсу RS-232

Логічній «1» відповідає напруга на вході приймача в діапазоні від -12 до -3 В. Для ліній послідовних даних цей стан називається «MARK» (лог. «1»). Логічному «0» відповідає діапазон від +3 до +12 В. Для послідовних даних цей стан називається SPACE. Діапазон від -3 до +3 В – зона нечутливості, обумовлена гістерезисом приймача: стан ліній буде рахуватися зміненим лише після перетинання порогу чутливості. Рівні сигналів на виходах передавачів повинні бути в діапазонах від -12 до -5 В та від +5 до +12 В для подання, відповідно, «1» та «0». Різниця потенціалів між схемними землями з’єднувальних пристроїв повинна бути меншою 2 В, інакше можливе неправильне сприйняття сигналів.

Формат кадру інтерфейсу RS-232 співпадає з форматом кадру модуля UART, рис. 10.5.

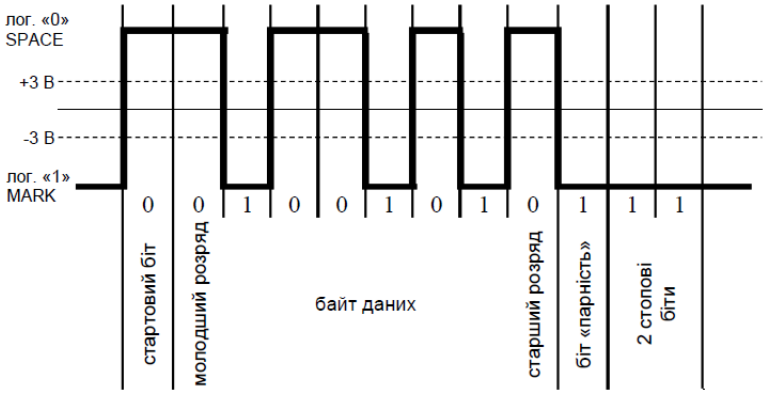


Рис. 10.5 – Формат кадру 8-N-1 інтерфейсу RS-232 для числа 82 (01010010)

При передачі використовуються різні формати кадру стеку. Стандартом “де-факто” був формат 8-N-1. Цифра 8 позначає кількість бітів інформації в пакеті, буква N вказує на відсутність службового біта перевірки на парність/непарність, цифра 1 позначає кількість стоп-бітів в кінці пакету.

Класифікація параметрів формату:

- по першій цифрі: від 5 до 9 інформаційних бітів в пакеті;

- по другій букві: N (no parity) – без біта перевірки, E (even parity) – з бітом перевірки на парність, O (odd parity) – з бітом перевірки на непарність;

- по третій цифрі: 1 або 2 стоп біти.

Варто зазначити, що приймачем другий стоп-біт ігнорується, і відповідно, помилки кадрування виявляються лише для першого стоп-біта.

Для підключення МК через модуль UART до COM-порту ПК необхідно виконати перетворення TTL-рівнів у рівні інтерфейсу RS-232 та навпаки за допомогою спеціалізованої мікросхеми, наприклад, MAX232. Якщо на ПК відсутній фізичний COM-порт, тоді необхідно або використовувати адаптери USB/RS-232, або спеціалізовані мікросхеми на зразок FT232RL (після встановлення драйвера з’явиться віртуальний COM-порт).

Питання.

1. Основні можливості UASRT.
2. Регістри UASRT.
3. Формат передачі кадра даних UASRT.
4. Визначення швидкості передачі/прийому даних.
5. Ініціалізація UASRT. Прийом/передача даних.

ЛЕКЦІЯ 11. Послідовний інтерфейс SPI

Мета. Вивчення послідовного інтерфейсу SPI

Вступ. МК AVR мають у своєму складі послідовний інтерфейс SPI. Через нього можна передавати і приймати інформацію заданому послідовним кодом. За допомогою цих інтерфейсів може обмінюватися даними з різними периферійними пристроями або іншими МК.

План.

1. Послідовний периферійний інтерфейс SPI
2. Протокол передачі даних SPI
3. Схеми з'єднань передачі даних SPI
4. Приклади ініціалізації SPI як ведучого і веденого
5. Процес прийому і передачі даних

1. Послідовний периферійний інтерфейс SPI

Інтерфейсі SPI (Serial peripheral interface, послідовний периферійний інтерфейс) підтримує високошвидкісну синхронну передачу даних між МК та периферійними пристроями або між декількома МК. Інтерфейс SPI в Atmega8515 має наступні особливості:

- повнодуплексна, 3-дротова синхронна передача даних;
- режим робота Ведучий (Master) або підпорядкований (Slave);
- передача даних починаючи з молодшого або старшого біту;
- прапор переривання кінець передачі;
- прапор захисту від колізій при запису даних;
- вихід з режиму очікування Idle;
- режим роботи ведучого SPI з подвоєною швидкістю.

У інтерфейсі SPI дані передаються по одній лінії побітово з визначеними інтервалами часу, а синхронізуючі імпульси – по окремій виділеній лінії. Це спрощує задачу синхронізації – не потрібно спеціально задавати швидкості обміну, але при цьому збільшується число сигнальних ліній (не менше трьох). Для адресації, при підключенні більше ніж двох МК до одного інтерфейсу, потрібний додатковий четвертий вивід /SS (Slave select). Він позначається з інверсією, так як вибір здійснюється подачею низького рівня на цей вхід.

В склад модуля SPI входять, рис. 11.1:

– 8-розрядний зсувний регістр SPDR (приймає байт з шини даних МК, зсовує його вправо або вліво з видачею послідовного коду на вивід МК, одночасно з виводом приймає послідовний код із входу МК і через буферний регістр передає його на шину даних МК);

– 8-розрядні регістри керування SPCR і стану SPSR;

– попередній подільник частоти;

– схеми керування.

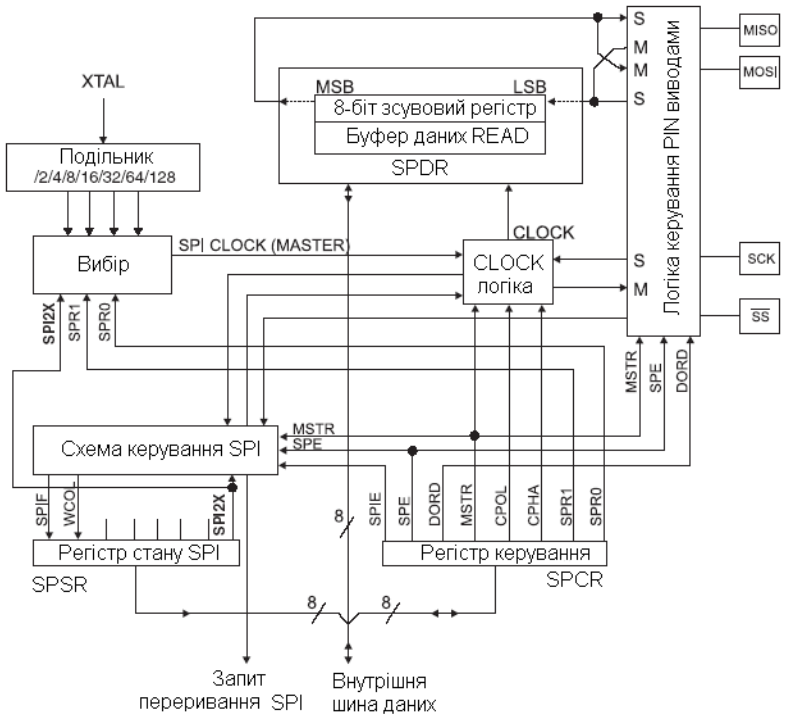


Рис. 11.1. Структурна схема SPI

В МК ATmega8515 для сигналів інтерфейсу SPI призначені наступні виводи порту PB: PB5 – MOSI, PB6 – MISO, PB7 – SCK, PB4 – ‘SS.

Для керування роботою SPI використовують регістри SPCR, SPSR, SPDR.

Регістр керування SPCR задає режими роботи SPI:

7	6	5	4	3	2	1	0
SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0

• SPIE (SP interrupt enable) – дозвіл переривання SPI (якщо встановлені біти SPSR.SPIF і SREG.I);

• SPE (SP enable) – дозвіл доступу до SPI (якщо біт встановлений);

• DORD (Data order) – порядок даних (1 – першим передається LSB біт, 0 – першим передається MSB біт);

• MSTR (Master/Slave select) – режим роботи порту SPI (1 – ведучий, 0 – ведений);

• CPOL (Clock Polarity) – полярність тактового сигналу (в режимі очікування тактовий сигнал SCK має низький логічний рівень – 0 або високий логічний рівень – 1);

Функціонування CPOL:

CPOL	Ведучий фронт	Завершуючий фронт
0	Наростання	Спадання
1	Спадання	Наростання

• CPHA (Clock Phase) – вибір фази тактового сигналу (читання біта по ведучому фронту сигналу або завершуючому фронту тактового сигналу:

CPHA	Ведучий фронт	Завершуючий фронт
0	Вибірка	Встановлення
1	Встановлення	Вибірка

• SPR1, SPR0 (SP Clock Rate Select 1 and 0) – вибір частоти тактового сигналу ведучого МК.

SPI2X	SPR1	SPR0	Тактова частота
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

Регістр стану SPI (SPI Status Registr) – **SPSR**

7	6	5	4	3	2	1	0
SPIF	WCOL	-	-	-	-	-	SPI2X

• SPIF (SPI Interrupt flag) – прапор виставляється при закінченні передачі.

• WCOL (Write COLlision Flag) – прапор встановлюється, якщо під час передачі є спроба запису даних.

• SPI2X (Double SPI Speed Bit) – коли прапор встановлюється в 1 тактова частота подвоюється для для ведучого МК.

Регістр даних SPI (SPI Data Registr) – **SPDR**:

7	6	5	4	3	2	1	0
MSB							LSB

В регістр SPDR дані записуються і дані з нього зчитуються при передачі даних між регістровим файлом і зсувним регістром SPI (SPI shift register). Запис даних у регістр ініціює передачу. Читання регістра спричиняє читання приймаючого буфера зсувного регістру.

2. Протокол передачі даних SPI

Протокол обміну по SPI аналогічний логіці роботи зсувного регістру і полягає в послідовному побітовому виведенні/введенні даних за певними напрямами тактового сигналу. Установка даних і вибірка здійснюється по протилежних фронтах тактового сигналу.

Специфікація SPI передбачає 4 режими передачі даних, які відрізняються між собою співвідношенням фази і полярності тактового сигналу SCK і переданих даних. Ці режими описуються двома бітами: полярність CPOL (clock polarity) і фаза CPHA. Полярність визначає

вихідний рівень сигналу синхронізації SCK. Фаза CPHA визначає послідовність встановлення і вибірки даних, рис. 11.3, 11.4.

В залежності від комбінації бітів CPHA і CPOL є чотири формати передачі даних:

	Ведучий фронт	Завершуючий фронт	SPI режим
CPOL=0,CPHA=0	Вибірка (Наростання)	Встановлення (Спадання)	0
CPOL=0,CPHA=1	Встановлення (Наростання)	Вибірка (Спадання)	1
CPOL=1,CPHA=0	Вибірка (Спадання)	Встановлення (Наростання)	2
CPOL=1,CPHA=1	Встановлення (Спадання)	Вибірка (Наростання)	3

При MSTR=1 порт SPI працює як ведучий. При цьому вивід MOSI є виходом даних, вивід MISO – входом даних, вивід SCK – виходом для імпульсів, які використовуються як зсувні при прийманні даних ведучим МК. Функція виводу ‘SS залежить від стану розряду DDRB.4. При DDRB.4=1 вивід SS порту SPI не підключений до виводу порту PB4, при DDRB.4=0 значення сигналу на вході впливає на роботу порту SPI. Якщо PB4 (SS)=1, порт працює як ведучий, а при значенні сигналу 0 перемикається в режим веденого. Для переведення порту SPI в робочий стан встановлюється біт SPE регістра SPCR. При MSTR=0 порт SPI працює в режимі веденого. При цьому вивід MOSI є входом даних, вивід MISO – виходом даних, вивід SCK – входом для імпульсів зсуву, вивід ‘SS – входом. Виводи SPI підключаються до виводів порту PB також при встановленні біта SPE регістра SPCR. Порт переходить в робочий стан за сигналом логічного 0 на вході SS. Схема з’єднання двох МК для обміну даними по каналу SPI зображена на рис. 11.2.

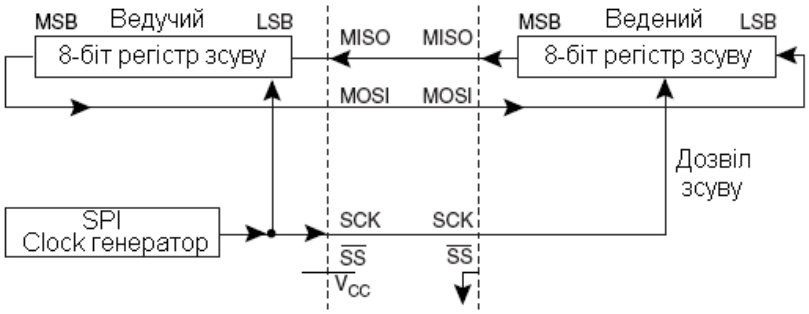


Рис. 11.2. Схема з’єднання двох МК

Передача даних починається після запису даних у регістр SPDR ведучого МК. Порядок видачі визначається станом біта DORD. Після видачі останнього розряду встановлюється прапор SPIF (біт 7 регістра SPSR) і одночасно виробляється запит переривання SPI_STC з адресою вектора $008 (при встановленому прапорі SPIE регістра SPCR). Одночасно з передачею приймаються дані від веденого МК. По закінченню прийому даних у веденому МК також встановлюється в 1 прапор SPIF і виробляється запит на переривання. При спробі запису у регістр даних SPDR під час передачі чергового байту встановлюється в 1 прапор WCOL регістра SPSR. Швидкість передачі ведучого МК задається бітами SPR1, SPR0 регістра SPCR.

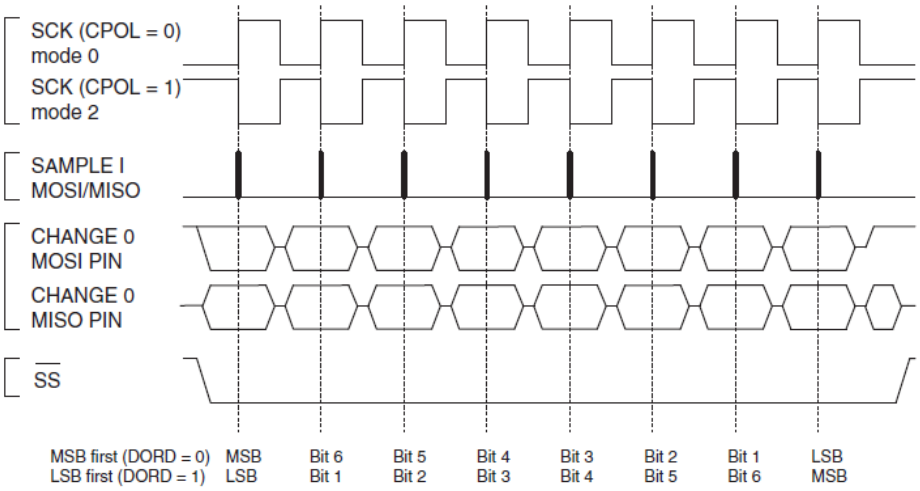


Рис. 11.3. Протокол передачі даних при CPHA=0

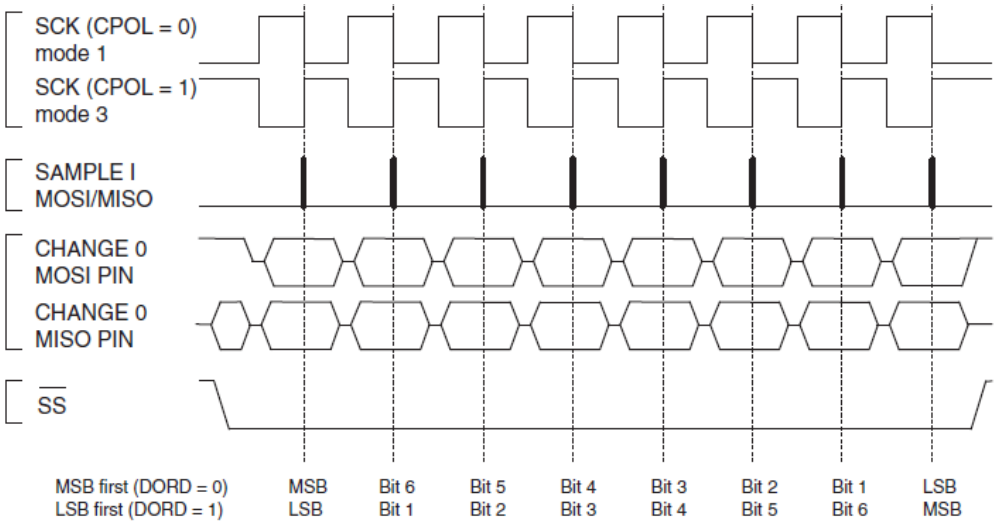
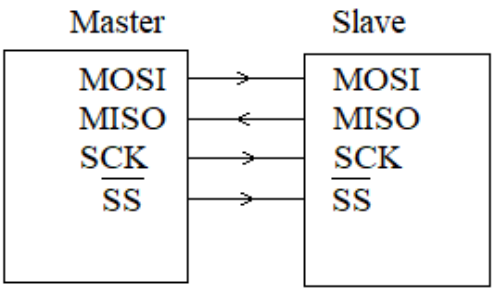


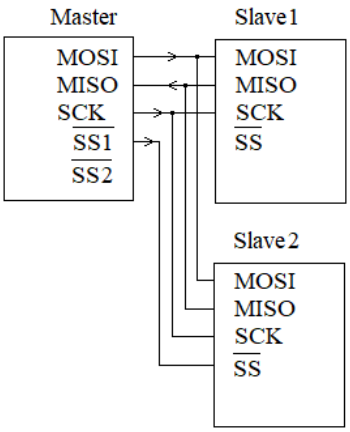
Рис. 11.4. Формати передачі даних при CPHA=1

3. Схеми з’єднань SPI

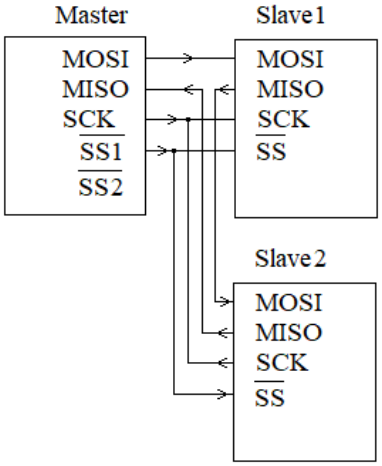
Типова схема з’єднання двох пристроїв по SPI:



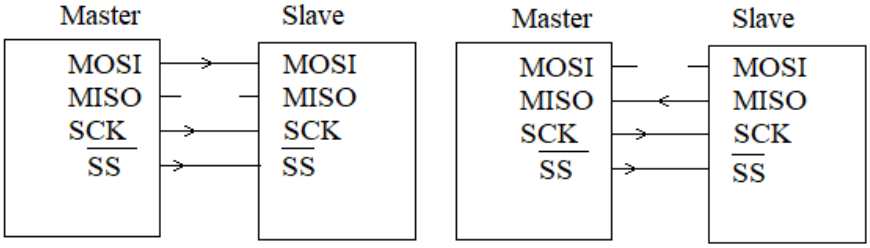
Також можливе підключення до ведучого пристрою декількох ведених пристроїв. Однак в будь-який момент часу обмін може відбуватися тільки з одним з них, інші повинні знаходитися в неактивному стані.



Виняток становить каскадна схема з'єднань по SPI. При такому підключенні зсувні регістри пристроїв утворюють один великий регістр, і кількість ліній SPI залишається рівною чотирьом. Правда, таке підключення підтримують далеко не всі мікросхеми



Також можливий скорочений варіант схеми підключення, коли лінія MOSI або MISO не використовується. Тобто передача даних здійснюється тільки в одну сторону. Такі схема, наприклад, використовуються при підключенні до мікроконтролеру зовнішніх мікросхем ЦАП і АЦП.



4. Приклади ініціалізації SPI як ведучого і веденого

Приклади коду для ініціалізації SPI як Master і як виконати просту передачу. DDR_SPI у прикладах потрібно замінити на фактичний регістр напрямку, який контролює вивід SPI. DD_MOSI, DD_MISO та DD_SCK потрібно замінити фактичними бітами напрямків передачі даних для цих виводів. Наприклад, якщо MOSI розміщено на виводі PB5, то замінити DD_MOSI на DDB5, а DDR_SPI на DDRB.

```
SPI_MasterInit:
; встановити MOSI і SCK на output, всі інші на input
ldi r17,(1<<DD_MOSI)|(1<<DD_SCK)
out DDR_SPI,r17
; дозвіл SPI, Master, встановлення порядку частоти clock fck/16
ldi r17,(1<<SPE)|(1<<MSTR)|(1<<SPR0)
out SPCR,r17
ret

SPI_MasterTransmit:
; початок передачі даних (r16)
out SPDR,r16
Wait_Transmit:
; очікування на закінчення передачі
sbis SPSR,SPIF
rjmp Wait_Transmit
ret
```

Наступний код показує, як ініціалізувати SPI як підпорядкований і як виконати просте приймання даних.

```
SPI_SlaveInit:
; встановити MISO на output, всі інші на input
ldi r17,(1<<DD_MISO)
out DDR_SPI,r17
; дозвіл SPI
ldi r17,(1<<SPE)
out SPCR,r17
ret

SPI_SlaveReceive:
; очікування завершення прийому
sbis SPSR,SPIF
rjmp SPI_SlaveReceive
; читання отриманих даних і повернення
in r16,SPDR
ret
```

Мінімальний програмний код на мові Сі для роботи з SPI модулем складається з двох функцій: функції ініціалізації та функції передачі/прийому байта даних.

Ініціалізація включає в себе конфігурацію виводів SPI модуля і керуючого регістра

```
#define SPI_PORTX PORTB
#define SPI_DDRX DDRB
#define SPI_MISO 6
#define SPI_MOSI 5
#define SPI_SCK 7
#define SPI_SS 4

/ * Ініціалізація SPI модуля в режимі master * /
```

```
void SPI_Init ( void )
{
/ * Налаштування портів введення-виведення
    всі виводи, крім MISO * /
SPI_DDRX | = (1 << SPI_MOSI) | (1 << SPI_SCK) | (1 << SPI_SS) | (0 << SPI_MISO);
SPI_PORTX | = (1 << SPI_MOSI) | (1 << SPI_SCK) | (1 << SPI_SS) | (1 << SPI_MISO);
/ * Дозвіл spi, старший біт вперед, майстер, режим 0 * /
SPCR = (1 << SPE) | (0 << DORD) | (1 << MSTR) | (0 << CPOL) | (0 << CPHA) |
       (1 << SPR1)
SPSR = (0 << SPI2X);
}
```

5. Передача і прийом даних

Процес передачі/прийому даних за допомогою SPI модуля, що працює в режимі Master, складається з наступної послідовності дій:

1. Встановлення низького логічного рівня на лінії ‘SS.
2. Завантаження даних в регістр SPDR.
3. Очікування закінчення передачі (перевірка прапора SPIF)
4. Збереження прийнятих даних (читання SPDR), якщо потрібно.
5. Повернення на крок 2, якщо передані не всі дані.
6. Встановлення високого логічного рівня на лінії ‘SS.

Нижче показано декілька варіантів функції передачі/прийому даних.

Передача і прийом одного байту даних по SPI.

```
void SPI_WriteByte (uint8_t data)
{
  SPI_PORTX & = ~ (1 << SPI_SS);
  SPDR = data;
  while (! (SPSR & (1 << SPIF)));
  SPI_PORTX |= (1 << SPI_SS)
}

uint8_t SPI_ReadByte (uint8_t data)
{
  uint8_t report;
  SPI_PORTX & = ~ (1 << SPI_SS);
  SPDR = data;
  while (! (SPSR & (1 << SPIF)));
  report = SPDR;
  SPI_PORTX | = (1 << SPI_SS);
retun report;
}
```

Передача декількох байтів даних по SPI.

(`*data` – покажчик на масив даних, що передаються, а `num` – розмірність масиву)

```
void SPI_WriteArray (uint8_t num, uint8_t * data)
{
SPI_PORTX & = ~ (1 << SPI_SS);
while (num -) {
SPDR = * data ++;
while (! (SPSR & (1 << SPIF)));
```

```
}
SPI_PORTX | = (1 << SPI_SS);
}
...
// Приклад використання:
uint8_t buf [3] = {12, 43, 98};
...
SPI_WriteArray (3, buf);
```

Передача і прийом декількох байтів даних по SPI.

(`*data` – покажчик на масив даних, що передаються, а `num` – розмірність масиву. Прийняті дані будуть зберігатися в тому ж масиві.)

```
void SPI_ReadArray (uint8_t num, uint8_t * data)
{
  SPI_PORTX & = ~ (1 << SPI_SS);
  while (num -) {
  SPDR = * data;
  while (! (SPSR & (1 << SPIF)));
     * Data ++ = SPDR;
  }
  SPI_PORTX | = (1 << SPI_SS);
}
```

Питання.

1. Призначення і можливості інтерфейсу SPI.
2. Регістри для роботи і керування SPI.
3. Регістр SPCR.
4. Регістр SPSR.
5. Протокол передачі даних.
6. Схеми з’єднань МК в SPI.
7. Ініціалізація SPI, як ведучого і веденого.
8. Послідовність дій при прийомі і передачі даних.

ЛЕКЦІЯ 12. Послідовний інтерфейс TWI/I2C

Мета. Вивчення двопровідного послідовного інтерфейсу TWI/I2C.

Вступ. TWI/I2C – послідовна шина даних для зв’язку інтегральних схем, розроблена як проста шина внутрішнього зв'язку для створення керуючої електроніки. Використовується для з'єднання низькошвидкісних периферійних компонентів з материнською платою, вбудовуваними системами та мобільними телефонами.

План

1. Інтерфейс TWI/I2C
1.1. Електричне з’єднання
1.2. Відкритий стік для двонаправленої передачі
2. Передача даних і формат пакету
2.1. Передача біту
2.2. Умова початку (START) і зупинки (STOP) передачі даних
3. Структура модуля TWI
4. Регістри TWI
5. Алгоритми основних транзакцій TWI
6. Програмна реалізація TWI
7. Приймання і передача даних з використанням апаратного TWI

1. Інтерфейс TWI/I2C

В деяких моделях МК є двопровідний послідовний байт орієнтований інтерфейс TWI (Two-wire Serial Interface). Двопровідний послідовний інтерфейс TWI сумісний з протоколом I2C компанії Philips.

Інтерфейс TWI має наступні особливості:

• для передачі даних використовується тільки два провідники;
• підтримуються операції Ведучий (Master) – Ведений (Slave);
• пристрій може функціонувати як передавач або приймач;
• 7-бітовий адресний простір дозволяє підключення до 127 ведених (slave) пристроїв;
• підтримується арбітраж багатьох ведучих (master) пристроїв;
• швидкість передачі даних до 400 kHz;
• драйвери з обмеженою вихідною швидкістю;
• схема подавлення завад відкидає імпульсні сплески сигналів на лініях шин;
• повністю програмовані адреси підпорядкованих пристроїв із підтримкою загального виклику.
• розпізнавання адрес викликає пробудження МК із режиму сну.

Двопровідний послідовний інтерфейс TWI забезпечує взаємодію між одним або декількома ведучими МК та з декількома веденим пристроями (енергонезалежною пам’яттю, контролерами паралельних портів, LCD-дисплеями, давачі температури, АЦП). Інтерфейс TWI дозволяє об’єднати до 128 пристроїв, використовуючи дві двонаправлені лінії, одну SDA для даних, другу SCL для тактових сигналів.

Всі пристрої під’єднані до інтерфейсу TWI мають індивідуальні адреси. Пристрій, який починає і закінчує передачу називається ведучим (Master). Ведучий пристрій також генерує

тактові сигнали. Ведений пристрій адресується ведучим пристроєм. Статус пристрою задається програмно.

1.1. Електричне з’єднання

Обидві лінії шини через зовнішні підтягуючі резистори R1, R2 підключені до джерела живлення VC, рис. 12.1. Драйвери всіх пристроїв виконані по схемі з відкритим колектором або відкритим стоком, що дозволяє реалізувати “монтажне І” для вихідних сигналів. Низький рівень на виході одного або декількох пристроїв встановлює в низький рівень лінію шини. Високий рівень встановлюється на лінії шини, коли виходи всіх пристроїв знаходяться у високоімпедансному (третьому) стані, дозволяючи підтягуючим резисторам підтягнути лінію до високого рівня. Пристрої підключені до TWI повинні мати живлення, для виконання любих операцій з шиною.

Число підключених до шини пристроїв обмежується ємністю шини в 400 пФ і 7-бітовим адресним простором.

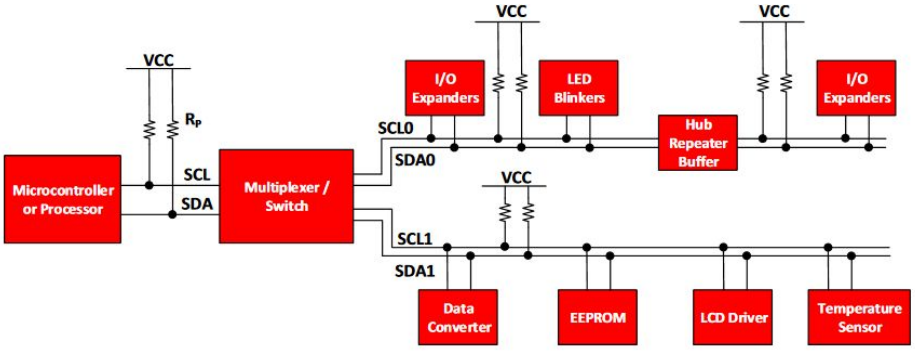


Рисунок 12.1 – Схема з’єднання пристроїв в інтерфейсі TWI/I2C

Протокол TWI/I2C використовує відкритий-стік/відкритий колектор з вхідним буфером на одній лінії, що дозволяє використовувати одну лінії для двонаправленої передачі даних

1.2. Відкритий стік для двонаправленої передачі

Відкритий стік відноситься до типу виходу, який може або підтягнути шину до напруги (заземлення, у більшості випадків), або «відпустить» шину і тоді її підтягне підтягуючий резистор. Якщо шина звільнена ведучим або підпорядкованим, підтягуючий резистор (R_{PU}) на лінії відповідає за підтягування шини до напруги живлення. Оскільки жоден пристрій не може встановити високий рівень на лінії, це означає, що на шині ніколи не буде стан, коли один пристрій намагається передати високий стан, а інший передає низький, що спричинить замикання шини живлення на землю. TWI/I2C вимагає, щоб ведучий пристрій, у середовищі з декількома ведучими, тримав високий стан, але бачачи, що лінія іншим пристроєм переведена в низький стан, має зупинити зв'язок, тому що інший пристрій використовує шину. Інтерфейси на основі інверторів не надають такого типу свободи, тому в цьому є перевага TWI/I2C.

На рис. 12.2 показана спрощена внутрішня структура ведучого або веденого пристрою на SDA/SCL лініях, яка складаються з буфера для зчитування вхідних даних і перемикального FET транзистора для передачі даних. Тільки пристрій здатний перевести шину у низький стан (забезпечуючи замкнення на землю) або розблокувати шину (високий опір відносно землі) і дозволити підтягуючому резистору підвищити напругу. Це важлива концепція, яку слід розуміти, при роботі з TWI/I2C пристроями, оскільки жоден пристрій не може утримувати шину у високому стані. Ця властивість забезпечує двонаправлений зв'язок.

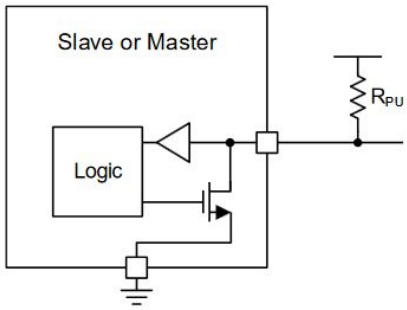


Рисунок 12.2 – Внутрішня структура SDA/SDL лінії

Схеми з відкритим витоком (колектором) можуть тільки перевести лінію у низький стан або звільнити її, дозволяючи підтягуючому резистору перевести її у високий стан. Логічна схема, яка хоче передати низький стан буде активувати перемикальний FET транзистор, який забезпечить заземлення і переведе лінію у низький стан, рис. 12.3.

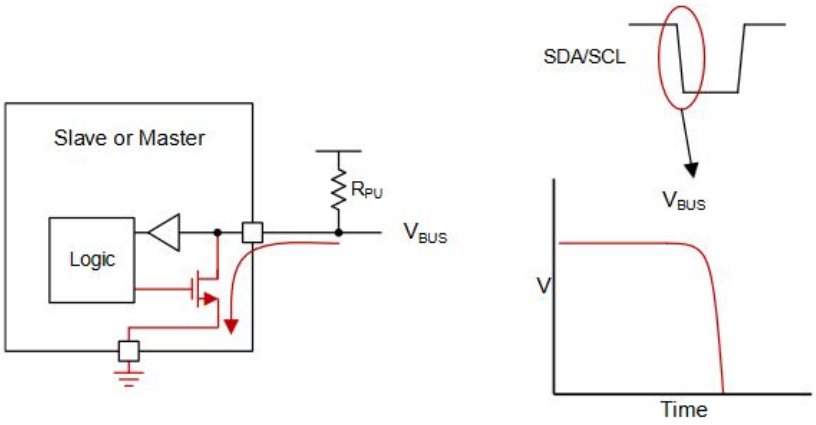


Рисунок 12.3 – Переведення шини у низький стан

2. Передача даних і формат пакету

2.1. Передача біту

Кожний біт, який передається по лінії даних SDA супроводжується імпульсом на тактовій лінії SCL, рис. 12.4.

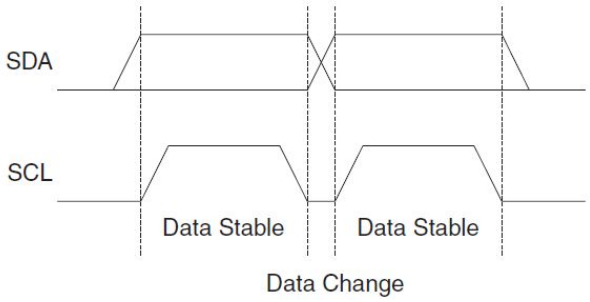


Рисунок 12.4 – Передача біту даних

2.2. Умови початку (START) і зупинки (STOP) передачі даних

Інтерфейс TWI є послідовним: всі дані і адреси передаються по лінії SDA побітово. Ведучий пристрій починає передачу станом START, а закінчує – станом STOP. Стартовий біт формується шляхом зміни рівня сигналу на лінії SDA 1=>0 при SCL лог.1, а стоповий – при зміні рівня сигналу на лінії SDA 0=>1 при SCL лог.1, рис. 12.5. Кожний біт, який передається супроводжується тактовим сигналом на лінії SCL, який формує ведучий пристрій. За час дії сигналу SCL (лог.1) стан лінії SDA має залишатися незмінним. Між станами START і STOP шина вважається зайнятою і ніякий інший ведучий пристрій не може отримати контроль над шиною. Особливий випадок трапляється, коли між умовами СТАРТ та СТОП видається нова умова START. Це називається умовою REPEATED START і використовується, коли ведучий бажає ініціювати нову передачу без відмови від управління шиною. Після REPEATED START шина вважається зайнятою до наступного стану СТОП. Це ідентично до поведінки START, і тому START використовується для опису як START, так і REPEATED START.

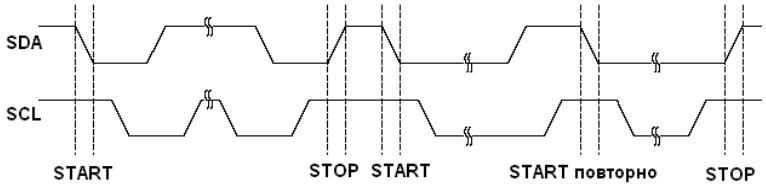


Рисунок 12.5 – Умови START, REPEATED START, STOP

2.3. Формат адресних пакетів і пакетів даних

По шині можуть передаватися два типи пакетів: адресні і з даними.

Адресні пакети. Всі адресні пакети є 9-бітовими і містять: 7-бітову адресу, яка передається веденим пристроям (починаючи із старшого), потім контрольний біт читання/запису R/W(1 – читати R, 0 – писати W) та біт підтвердження ACK, рис. 12.6.

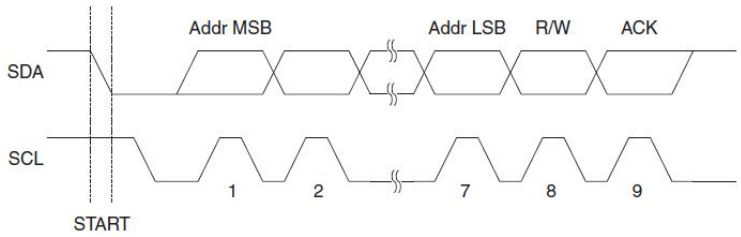


Рисунок 12.6 – Формат адресного пакету

Адреса 0000_000 використовується для загального виклику. Адреси 1111_xxx зарезервовані. У відповідь на загальний виклик всі підпорядковані пристрої переводять лінію SDA у лог.0 у ACK-циклі підтвердження (9-й такт SCL). Кадр даних, який після цього передається, буде отриманий всіма підпорядкованими пристроями, які встановили підтвердження. Для виклику одного підпорядкованого пристрою задається його адреса у кадрі адресу. Кожний із пристроїв порівнює отриману адресу із власною. При розпізнанні підпорядкованим пристроєм своєї адреси, він повертає у лінію SDA сигнал підтвердження ACK (лог.0 SDA на 9-такті SCL). Після отримання підтвердження ведучий пристрій починає передачу кадра даних.

Пакет даних. Пакет даних містить 8-бітів даних і один біт підтвердження ACK, який формує приймач, рис. 12.7.

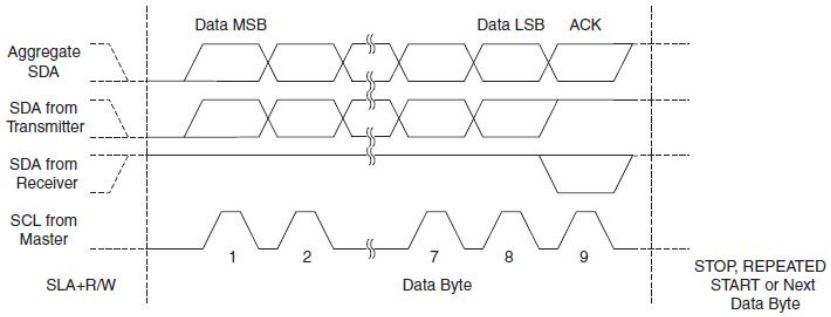


Рисунок 12.7 – Формат пакету даних

Дані передаються послідовно, починаючи із старшого біта. Після прийому кожного байта приймач виробляє сигнал підтвердження ACK. Не отримавши підтвердження від приймача, передавач може зупинити передачу даних, сформувавши стан STOP.

Приймач може збільшити тривалість періоду SCL у лог.0. Така зміна не впливає на тривалість SCL у лог.1, яку формує передавач. Збільшення тривалості періоду SCL у лог. 0 дозволяє вирівняти швидкодію передавача і приймача.

Після закінчення передачі байту даних, можуть передаватися наступні байти без зміни приймача, вибиратися новий приймач або закінчуватися обмін із звільненням шини.

3. Структура модуля TWI

Структура модуля TWI показана на рис. 12.8. Модуль містить блок шинного інтерфейсу (SDA, SCL) з регістром даних TWDR і контролером станів Start/Stop, блок контролю адреса з регістром TWAR і схемою порівняння, блок керування з регістрами TWCR і TWSR, контролер

швидкості передачі з попереднім дільником і регістром швидкості TWBR. Виводи SCL, SDA з'єднують TWI з іншими блоками МК. Вихідний драйвер обмежує швидкість, а вхідний – фільтрує імпульси з тривалістю менше 50 нс. Блок контролю швидкості задає частоту імпульсів на лінії SCL в режим роботи ведучого. Частота імпульсів задається значеннями бітів в регістрах TWBR і TWSR.

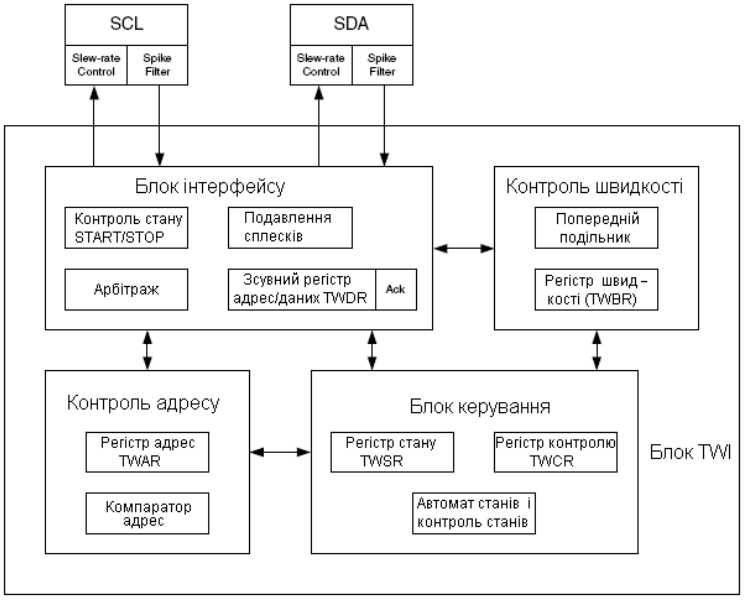


Рисунок 12.8 – Структура модуля TWI

Частота тактових сигналів МК, працюючого в режимі веденого, має бути в 16 більшою від частоти SCL. SCL частота визначається наступним рівнянням:

$$\mathrm{SCL} = \frac{\mathrm{CPU}_{\mathrm{CLOCK}}}{16 + 2(TWBR)4^{TWPS}},$$

де *TWBR* значення TWBR регістра;

TWPS значення бітів попереднього дільника в регістрі TWSR.

Блок інтерфейсу містить зсувний регістр даних/адрес TWDR, START/STOP контролер і апаратний арбітраж. В регістрі TWDR зберігаються дані/адреси, які передаються або приймаються і біт підтвердження прийому/передачі. START/STOP контролер генерує і визначає умови START, повторний START і STOP.

Блок контролю адреси перевіряє чи отримана адреса співпадає з 7-розрядною адресою в регістрі TWAR. Блок контролю порівнює адреси навіть якщо МК знаходиться в режимі sleep.

Блок керування аналізує зміни стану шини згідно із станом регістру TWCR і генерує відповідь, яка змінює стан регістру TWSR або генерує TWI переривання TWINT.

Модуль TWI може працювати в наступних режимах: ведучий з передачею байтів, ведучий з прийомом байтів, ведений з прийомом байтів, ведений з передачею байтів.

4. Регістри TWI

TWBR (TWI Bit Rate Register) – регістр темпу передачі бітів дозволяє вибрати коефіцієнти ділення для генератора темпу передачі бітів. Генератор темпу передачі бітів є частотним дільником, який задає частоту тактових сигналів SCL у режимі ведучого МК.

7	6	5	4	3	2	1	0
TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0

TWCR (TWI Control Register) – регістр дозволяє контролювати операції TWI.

7	6	5	4	3	2	1	0
TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE

TWINT (TWI Interrup Flag) – прапор встановлюється апаратно, коли TWI завершив роботи і очікує відповіді програми.

TWEA (TWI Enable Acknowledge Bit) – контролює генерацію імпульсу підтвердження (TWEA=1). При TWEA=0 пристрій тимчасово від’єднується від шини.

TWSTA (TWI START Condition Bit) – пристрій записує в TWSTA=1, хоче стати ведучим. Апаратура перевіряє чи шина доступна і вільна та генерує умову START.

TWSTO (TWI STOP Condition Bit) – при запису TWSTO=1 у режимі Ведучий, генерується умова STOP, якщо шина вільна.

TWWC (TWI Write Collision Flag) – біт встановлюється в лог. 1 при спробі запису у регістр TWDR коли TWINT=0. Прапор очищається при запису в TWDR, коли TWINT=1.

TWEN (TWI Enable Bit) – при TWEN=0, TWI отримує контроль над виводами МК, які під’єднані до SCL і SDA, що дозволяє низькошвидкісну передачу і використання фільтра імпульсних завад.

TWIN (TWI Interrupt Enable) – при TWIN=1 і SREG.I=1 активується запит на TWI переривання, якщо прапор TWINT=1.

TWSR (TWI Status Register) – регістр відображає стан TWI логіки і шини (TWS7-TWS3) та дозволяє задати коефіцієнти подільник генератора темпу бітів.

7	6	5	4	3	2	1	0
TWS7	TWS6	TWS5	TWS4	TWS3	-	TWPS1	TWPS0

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

TWDR (TWI Data Register) – регістр містить наступний байт для передачі, а в режимі приймання – останній прийнятий байт.

7	6	5	4	3	2	1	0
TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0

TWAR (TWI Address Register) – регістр містить в старших 7-бітах адресу підпорядкованого пристрою в режимах Slave Transmitter або Receiver. В режимі Master не використовується. Біт TWGCE використовується для розпізнання адреси загального виклику (0x00).

7	6	5	4	3	2	1	0
TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE

5. Алгоритми основних транзакцій TWI

В МК Atmel протокол TWI можна реалізувати програмно або програмно-апаратно у тих моделях, які мають вбудований апаратний протокол TWI. Для програмної реалізації інтерфейсу TWI потрібний набір програм, які емулюють роботу ведучого і веденого пристрою. Набір програм зменшується, якщо ведений пристрій має вбудований порт із інтерфейсом TWI. Алгоритми основних транзакцій шини, запису і читання, показані на рис. 12.9, 12.10.

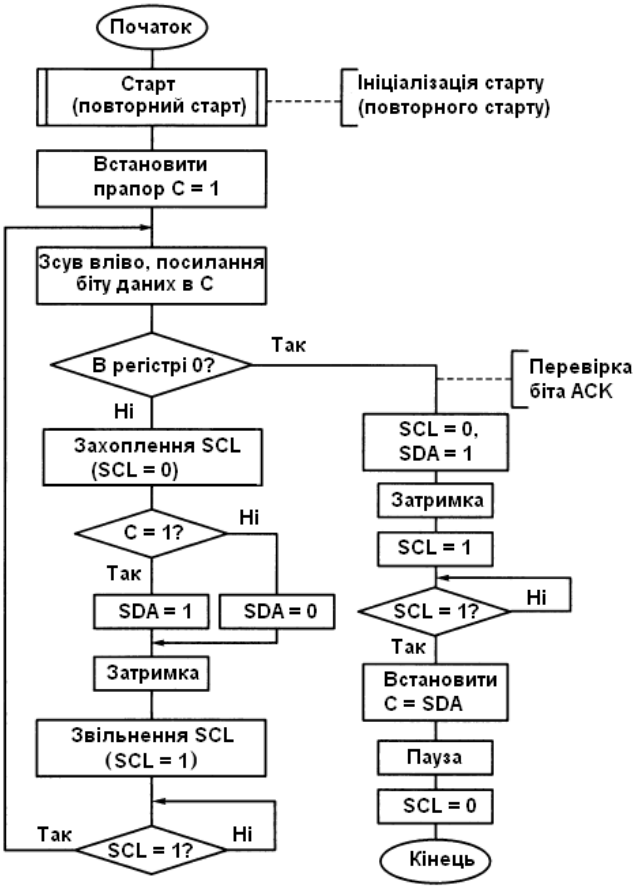


Рисунок 12.9 – Схема алгоритму запису байту даних

Процедура транзакції запису (передачі адресного байту і запис байту даних) починається із захоплення лінії SDA (SDA = 0) – формування стартового біту і встановлення прапора C = 1, який використовується для виявлення признаку закінчення циклу передачі. Шляхом циклічного зсуву вліво байта даних перший біт, який передається, витісняє 1 з прапора C в молодший розряд регістра даних. Це запобігає можливості передчасного виходу з циклу, коли в регістрі

даних під час передачі байта залишаються нульові біти. Лінія SCL переводиться в лог. 0. Значення біту С використовується для керування станом лінії SDA. Якщо передаваний біт, встановлений в С, дорівнює 1, лінія SDA приймає значення SDA = 1, у іншому випадку SDA = 1. Далі через час затримки встановлюється лінія SCL в лог. 1 і після перевірки, якщо ведений пристрій не гальмує роботу на лінії SCL, виконується циклічний перехід для виведення наступного біту даних. На наступних ітераціях циклу виконується логічний зсув. Після виявлення признаку кінця передачі, коли всі біти регістра даних дорівнюють 0, виконується перехід до процедури перевірки біта підтвердження.

Отримання біта підтвердження АСК починається із захоплення лінії SCL (SCL = 0) і звільнення ведучим лінії SDA (SDA = 1). Після часової паузи лінія SCL перемикається в стан 1 і, якщо вона вільна від впливу ведених пристроїв, значення SDA зчитується як сигнал підтвердження АСК (встановлення або скидання біту С). Після паузи на лінії SCL встановлюється 0.

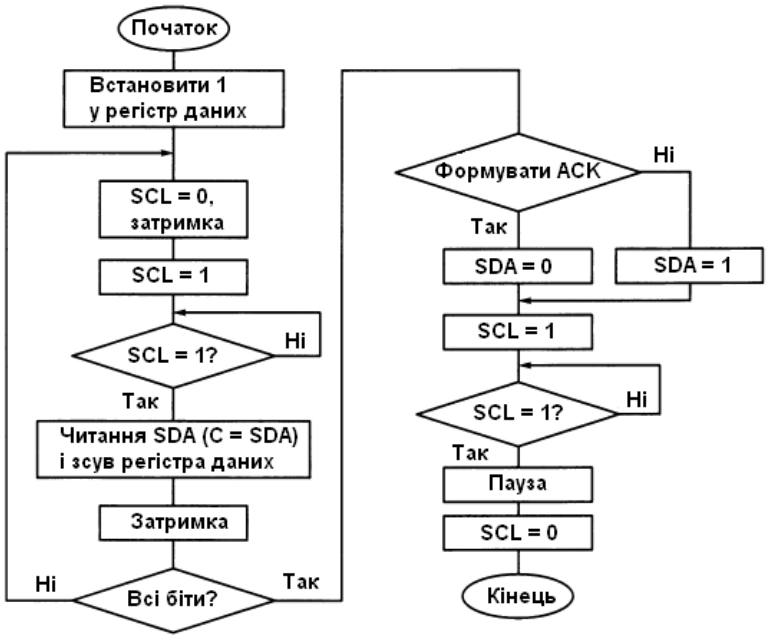


Рисунок 12.10 – Схема алгоритму читання байту даних

Процедура транзакції читання (прийом даних від веденого) починається з встановлення признаку кінця прийому (1) у молодший розряд регістра даних. Цикл читання бітів даних починається із захоплення лінії SCL=0, встановлення на лінії SCL=1, підтвердження високого рівня сигналу SCL=1 і наступного введення біту даних в регістр даних шляхом опитування лінії SDA і зсуву вмісту регістра даних. Після чергової паузи виконується перевірка признаку кінця прийому байта даних за значенням прапора С. Якщо не всі біти отримані (С=0), прийом продовжується. Якщо прийняті всі біти (С=1), ведучий пристрій переходить до формування біта підтвердження прийому АСК для веденого пристрою. При необхідності формування біта АСК лінія SDA встановлюється в 0, лінія SCL переводиться в стан 1. Після підтвердження SCL=1 і паузи лінія SCL знов повертається в стан 0. На цьому читання байту даних закінчується. Процедури формування стартового, стопового і повторного старту бітів зводяться до встановлення початкових станів сигналів SDA=1, SCL=1 і наступних змін згідно вказаних вище часових діаграм. Використовувані затримки часу (паузи) необхідні для забезпечення надійної

передачі. Їх встановлюють, згідно рекомендацій, на період і тривалість сигналу SCL; часу утримання неактивного стану ліній інтерфейсу; на час перед повторним стартом.

6. Програмна реалізація інтерфейсу TWI

Для програмної реалізації інтерфейсу TWI використовується МК ATmega8535 і програмований паралельний порт (ППП) PCA9554 фірми Philips, рис. 12.11.

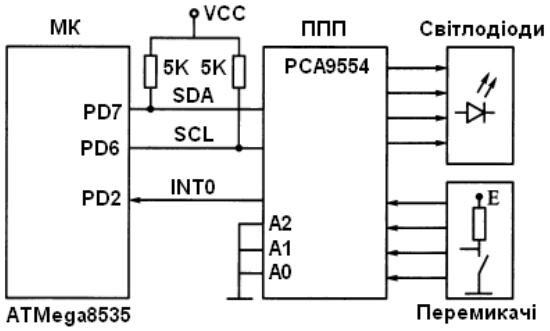
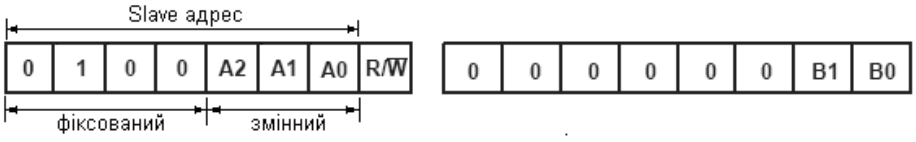


Рисунок 12.11 – Схема зв'язку МК з ППП по інтерфейсу TWI

ППП є мікросхемою, яка має канал послідовного зв'язку TWI, з однієї сторони, і 8-розрядний паралельний канал введення/виведення, з іншої сторони. Для звернення до ППП в мікроконтролерній системі використовується одна з восьми адрес в діапазоні $20-$27. При цьому три молодших розряди визначають шляхом встановлення сигналів лог. 0 і лог. 1 на входах A2-A0, рис.12.12, а. Останній розряд R/W визначає операцію: 1 – читання, 0 – записування.



а *б*

Рисунок 12.12 – Регістр керування (конфігурації) ППП: адресний (а) і командний (б) байти

Після успішного розпізнання адреси посилається командний байт, який записується в керуючий (конфігураційний) регістр ППП, рис. 12.12, б. Розряди B1, B0 встановлюють операцію (читання/запис) і внутрішній регістр (вхідний, вихідний, зміни полярності або конфігураційний): 0x00 – читання байту з вхідного регістру, 0x01 – читання/запис байту у вихідний регістр, 0x02 – зміна полярності сигналів при читанні/запису байту, 0x03 – конфігураційний регістр (встановлює напрямок передачі розрядів портів I/O, лог. 1 – на вхід, лог. 0 – на вихід). Звернення до порту містить три посилання: адрес, команди, дані. Обмін з портом виконується по запиту переривання, яке формується мікросхемою ППП при зміні сигналів на входах порту. Для цього вихід INT ППП підключений до входу INT0 МК (для ATmega8535 лінія порту PD2).

В алгоритмі основної програми передбачена наступна послідовність дій:

- ініціалізація порту МК з лініями інтерфейсу;
- налаштування системи переривань МК;

- часова затримка;
- налаштування конфігурації ППП;
- виклик процедури обміну з ППП;
- переведення МК в режим пониженого енергоспоживання і очікування переривань від ППП.

При надходженні запиту виконуються наступні дії:
- ініціалізація обміну з ППП;
- формування стану START і посилання адресу;
- запис у ППП команди введення ($00);
- зміна напрямку обміну;
- повторний старт;
- читання даних з ППП;
- обмін тетрадами;
- зберігання даних;
- формування стану STOP;
- формування стану START і посилання адресу;
- запис у ППП команди виведення ($01);
- зворотне пересилання даних у ППП;
- формування стану STOP і вихід з переривання.

7. Приймання/передача даних з використанням апаратного TWI

Процедура передачі одного байту даних від ведучого до веденого пристрою:

1. Перший крок в передачі даних по TWI – встановлення умови START. Для цього в регістрі TWCR встановлюються в лог. 1 біти – переривання TWINT (очищується записом лог.1, TWINT=1), TWSTA (TWSTA=1) і активізації модуля TWI (TWEN=1). Після формування стану START прапор TWINT встановлюється в лог. 1. Для переходу до наступної дії виконується перевірка коду стану $08 в регістрі TWSR ($08 – SLA+W has been transmitted; ACK has been received, SLA+W has been transmitted; NOT ACK has been received). Очікування встановлення прапора TWINT можна замінити обробкою запиту переривання. Функція встановлення умови старту реалізована в процедурі SendStart.

2. Вміст пакету з адресою і нульовим значенням біту напрямку записується в регістр TWDR. Передача ініціалізується скиданням прапора TWINT. Після закінчення передачі адресу і отримання біту підтвердження прапора TWINT встановлюється в 1, а в регістре TWSR встановлюється код статусу $18, ($18 – SLA+W has been transmitted; ACK has been received. $20 – SLA+W has been transmitted; NOT ACK has been received). Функції запису реалізована в процедурі SendAdr.

3. Після перевірки кода статусу в регістр даних TWDR завантажується байт даних для передачі і скидається прапор TWINT. По закінченню передачі даних і отримання біта підтвердження прапор TWINT встановлюється в лог. 1, а в регістрі TWSR встановлюється код статусу $28 ($28 – Data byte has been transmitted; ACK has been received. $30 – Data byte has been transmitted; NOT ACK has been received). Ці дії забезпечують передачу від ведучого пристрою до веденого як команд, так і даних. Програмна реалізація функції виводу даних (а також команд) реалізована в процедурі SendComData.

4. Після успішного завершення передачі виконуються команди для формування стану Stop. Функція завершення реалізована в процедурі Stop.

Процедура прийому одного байту даних *ведучим пристроєм*:

1. В регістр TWCR виводиться команда для формування стану Start; після формування стану Start прапор TWINT встановлюється в лог. 1. Для переходу до наступної дії перевіряється код стану ($08) в регістрі TWSR. (Очікування встановлення прапора TWINT також можна замінити обробкою запиту переривання.);

2. Вміст пакету з адресою і одиничним значенням біту напрямку записується в регістр TWDR і ініціалізується передача. Після закінчення передачі адресу і отримання біта підтвердження прапор TWINT встановлюється в 1, а в регістрі TWSR встановлюється код статусу ($40);

3. Після скидання прапора TWINT (дозвіл прийому) і отримання байту даних від веденого дані з регістра TWDR переписуються в один з регістрів загального призначення. При успішному прийомі код статусу в регістрі TWSR приймає значення $50. При необхідності формується біт підтвердження прийому;

4. Після закінчення прийому виконується команда для формування стану Stop.

Процедура прийому одного байту даних *веденим пристроєм*:

1. В регістр TWCR виводиться команда, скидається прапор TWINT;

2. Після прийому першого байта з адресою веденого пристрою формується запит переривання, перевіряється код статусу в регістрі TWSR; якщо він дорівнює $60, прийом власного адреса виконано успішно;

3. Передається біт підтвердження шляхом встановлення TWEN=1 і скидається прапор TWINT;

4. Перевіряється код статусу в регістрі TWSR (якщо він дорівнює $80, значить в регістр TWDR прийнято байт даних); передається біт підтвердження (TWEN=1) і скидається прапор TWINT;

Дії 4-го кроку повторюються, поки не будуть прийняті всі повідомлення. При виникненні стану Stop в регістрі TWSR формується код статусу $A0 (кінець пакету).

Аналогічно можна подати дії веденого пристрою при передачі даних. Детальний опис Всі варіанти обміну і значення статусних кодів детально описані в технічній документації МК.

Питання.

1. Призначення і можливості інтерфейсу TWI/I2c.
2. Передача даних і формат пакету.
3. Структура модуля TWI і основні функціональні блоки.
4. Регістри інтерфейсу TWI
5. Алгоритми основних транзакцій TWI
6. Програмна реалізація TWI
7. Приймання і передача даних з використанням апаратного TWI.

ЛЕКЦІЯ 13. Аналого-цифрові перетворювачі

Мета. Вивчення аналого-цифрових перетворювачів.

Зміст.

1. Аналого-цифрові перетворювачі
2. Регістри стану і керування

1. Аналого-цифрові перетворювачі

Аналого-цифрові перетворювачі (АЦП, Analog to Digit Converter, ADC) є пристроями, які приймають вхідні аналогові сигнали та генерують відповідні їм цифрові сигнали, які можуть обробляти МК та інші цифрові пристрої. АЦП дає еквівалентне подання аналогового сигналу у цифровому (двійковому) коді, рис. 13.1.

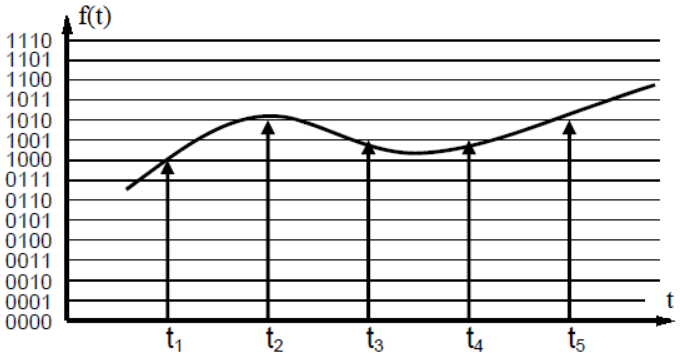


Рис. 13.1 – Квантування та дискретизація аналогового сигналу

Багатоканальний АЦП входить в більшість сучасних моделей МК AVR. Звичайно число каналів дорівнює 8, але в різних моделях воно може варіювати від 4 каналів в молодших моделях родини Tiny, 6 каналів в ATmega8 і до 16 каналів в ATmega2560. Багатоканальність означає, що на вході єдиного модуля АЦП встановлений аналоговий мультиплексор, який може підключати цей вхід до різних виводів МК для здійснення вимірювань декількох незалежних аналогових величин з рознесенням по часу. Входи мультиплексора можуть працювати окремо (в несиметричному режимі для виміру напруги відносно "землі") або (в деяких моделях) об'єднуватися в пари для вимірювання диференціальних сигналів. Іноді АЦП додатково забезпечується підсилювачем напруги з фіксованими значеннями коефіцієнта підсилення 10 і 200.

Сам АЦП є перетворювачем послідовного наближення з пристроєм вибірки-зберігання і фіксованим числом тактів перетворення, рівним 13 (або 14 для диференційного входу). Перше перетворення після ввімкнення займає 25 тактів для ініціалізації АЦП. Тактова частота формується аналогічно тому, як це робиться для таймерів – за допомогою спеціального дільника тактової частоти МК, з коефіцієнтом ділення від 1 до 128. Але на відміну від таймерів, вибір тактової частоти АЦП не зовсім довільний, так як швидкодія аналогових компонентів обмежена. Тому коефіцієнт ділення потрібно вибирати таким, щоб при заданій частоті роботи МК тактова частота АЦП вкладалася в рекомендований діапазон 50-200 кГц (тобто максимум близько 15 тис. вимірювань в секунду). Збільшення частоти вибірки допустимо, якщо не потрібно

досягнення високої точності перетворення. Роздільна здатність АЦП в МК AVR – 10 двійкових розрядів, чого для більшості типових застосувань досить. Абсолютна похибка перетворення залежить від ряду факторів і в ідеальному випадку не перевищує ±2 молодших розряди, що відповідає загальній точності вимірювання приблизно 8 двійкових розрядів. Для досягнення цього результату необхідно приймати спеціальні заходи: не тільки правильно підбирати тактову частоту в рекомендований діапазон, але і знижувати по максимуму інтенсивність цифрових шумів. Для цього рекомендується, як мінімум, не використовувати невикористані виводи того ж порту, до якого підключений АЦП, робити правильну розводку друкованої плати і додатково включати спеціальний режим пониження шуму (ADC Noise Reduction). АЦП може працювати у двох режимах: окремого перетворення або вільного запуску. Другий режим доцільний лише при максимальній частоті вибірок. В інших випадках його слід уникати, оскільки обійти в цьому випадку необхідність паралельної обробки цифрових сигналів, як правило, неможливо, а це означає зниження точності перетворення.

Блок схема АЦП ATMega8535 показана на рис. 13.2.

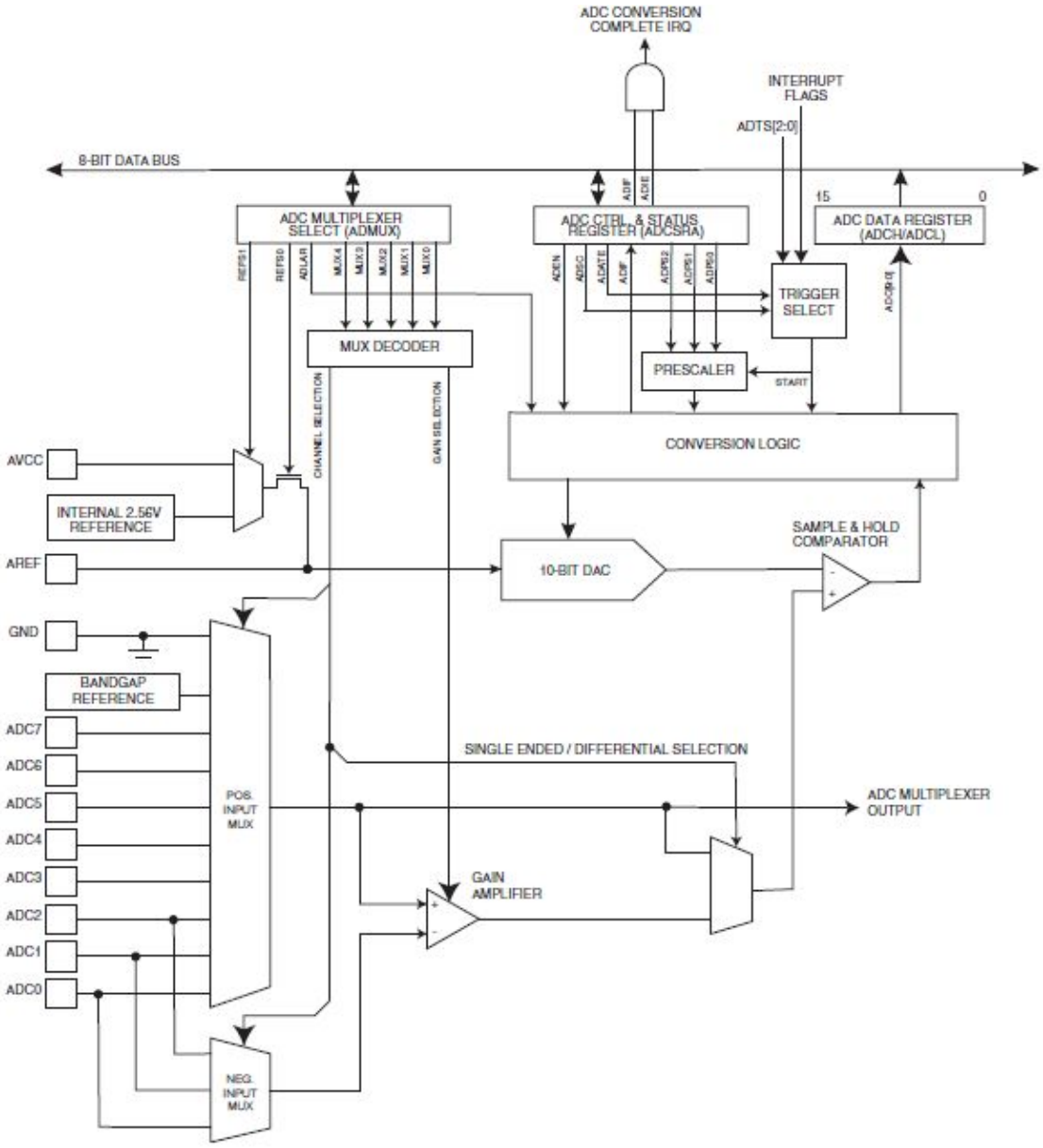


Рисунок 13.2 – Блок схема АЦП МК ATMega8535

АЦП ATMega 8535 має наступні основні параметри:

• Розрядність АЦП 10-біт (тому результат перетворення зберігається у двох регістрах ADCH:ADCL).

• Інтегральна нелінійність 0,5 LSB (least significant bit, молодший біт).

• Абсолютна похибка 2 LSB.

• Час перетворення 65-260 мкс.

• Швидкодія до 15 тис. вибірок за секунду.

• 8-мультиплексованих входів.

• 7-диференційних входів.

• 2-диференціальні входи з коефіцієнтом підсилення 10X або 200X.

• АЦП може функціювати коли процесор знаходиться в режимі сну (sleep mode).

• АЦП може генерувати запит на переривання по завершенню перетворення.

• АЦП має внутрішнє джерело опорної напруги 2.26 В.

• Неперервний і одиночний режим перетворення.

АЦП має окремий контакт для зовнішнього джерело живлення $AV_{CC,}$ яке не має відрізнятися від V_{CC}. Є внутрішнє опорне джерело живлення 2.56 В.

АЦП перетворює аналогову вхідну напругу у цифрове 10-бітове значення методом послідовного наближення. Мінімальне значення відповідає GND, а максимальне значення відповідає значенню на контакті AREF мінус один наймолодший біт. До штифта AREF можна підключити AV_{CC} або 2.56 В. Внутрішня опорна напруга може бути розв’язана зовнішнім конденсатором на виводі AREF для покращення завадостійкість.

Аналоговий вхідний канал вибирається бітами MUX в ADMUX. Будь-які вхідні контакти АЦП, а також GND і опорна напруга з фіксованою забороненою зоною можуть бути обрані як одиночні входи в АЦП. Як вхідні контакти АЦП можуть бути обрані як позитивні та негативні входи для підсилювача диференціального посилення.

Якщо вибрано диференціальні канали, каскад диференціального підсилювач підсилює різницю напруг між вибраною парою вхідних каналів на вибраний коефіцієнт посилення. Потім це підсилене значення стає аналоговим входом АЦП. Якщо використовуються односторонні канали, підсилювач підсилення повністю пропускається.

АЦП вмикається встановленням біта ADEN в ADCSRA. Напруга опорна та вхідних каналів не стане чинною, доки не буде встановлено біт ADEN. АЦП не споживає електроенергію, коли ADEN очищений, тому рекомендується вимкнути АЦП перед переходом в режим енергозбереження сплячого режиму.

Функціонування АЦП.

АЦП генерує 10-бітовий результат, який знаходиться у регістрі даних ADC (ADCH і ADCL). За замовчуванням результат заданий із зсунути праворуч, але його можна за бажанням зсунути ліворуч, встановивши біт ADLAR в ADMUX.

Якщо результат зсунутий ліворуч і потрібна точність не більше ніж 8 біт, достатньо прочитати регістр ADCH. В іншому випадку спочатку потрібно прочитати ADCL, а потім ADCH, щоб переконатися, що вміст реєстрів даних належить до того самого перетворення. Після зчитування ADCL доступ АЦП до регістрів даних блокується. Це означає, що якщо ADCL було прочитано, і перетворення завершується до зчитування ADCH, жоден регістр не оновлюється, а результат перетворення втрачено. Коли ADCH зчитується, доступ АЦП до регістрів ADCH і ADCL знову активується.

АЦП має власне переривання, яке може викликатися після завершення перетворення. Коли доступ АЦП до регістрів даних заборонений між зчитуванням ADCH і ADCL, переривання спрацює, навіть якщо результат буде втрачено.

Окреме перетворення починається записом лог. 1 в ADSC. Цей біт залишається в лог. 1, поки триває перетворення, і буде апаратно очищеним, коли перетворення завершиться. Якщо під час перетворення вибрано інший канал даних, АЦП завершить поточне перетворення перед виконанням зміни каналу.

Початок перетворення.

У режимі вільного запуску АЦП постійно відбирає та оновлює регістр даних ADC. Режим вільного запуску вибирається шляхом запису лог. 1 в біт ADFR регістра ADCSRA. Перше перетворення необхідно розпочати записом лог. 1 біт ADSC регістра ADCSRA. У цьому режимі АЦП виконуватиме послідовні перетворення незалежно від того, очищено прапор переривання ADIF чи ні.

Крім того, перетворення може бути ініційовано автоматично різними джерелами, рис. 13.3. Автоматичний запуск вмикається встановленням біта ввімкнення автоматичного запуску АЦП ADATE в ADCSRA. Джерело запуску вибирається встановленням бітів вибору тригера АЦП ADTS у SFIOR. Коли на вибраному тригерному сигналі виникає позитивний фронт, попередній дільник АЦП скидається і починається перетворення. Це забезпечує спосіб запуску перетворень через фіксовані проміжки часу. Якщо після завершення перетворення тригерний сигнал все ще встановлено, нове перетворення не розпочнеться. Якщо під час перетворення на тригерному сигналі виникає інший позитивний фронт, цей фронт ігноруватиметься. Зверніть увагу, що прапор переривання буде встановлено, навіть якщо конкретне переривання вимкнено або біт дозволу глобального переривання в SREG очищено. Таким чином можна запустити перетворення без переривання. Однак прапор переривання має бути очищено, щоб запустити нове перетворення під час наступної події переривання.

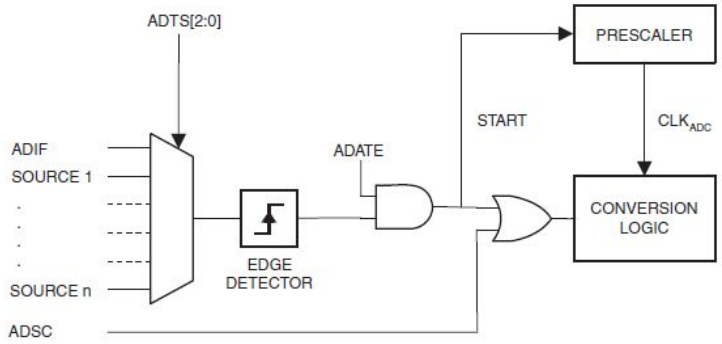


Рисунок 13.3 – Логіка автоматичного запуску перетворень АЦП

Використання прапора переривання АЦП як джерела запуску змушує АЦП почати нове перетворення, щойно поточне перетворення завершиться. Потім АЦП працює у вільному режимі, постійно відбираючи та оновлюючи регістр даних АЦП. Перше перетворення має бути розпочато записом логічної одиниці в біт ADSC в ADCSRA. У цьому режимі АЦП виконуватиме послідовні перетворення незалежно від того, чи скинуто прапор переривання АЦП, ADIF чи ні.

Якщо ввімкнуто автоматичний запуск, окремі перетворення можна розпочати, записавши ‘1’ в ADSC в ADCSRA. ADSC також можна використовувати, щоб визначити, чи триває

перетворення. Біт ADSC читатиметься як один під час перетворення, незалежно від того, як було розпочато перетворення.

Попереднє масштабування і перетворення синхросигналів.

Схема попереднього дільника сигналів показана на рис. 13.4.

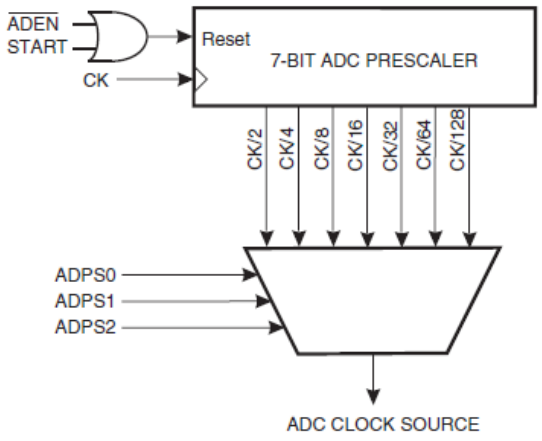


Рисунок 13.4 – Схема попереднього дільника

За замовчуванням схема послідовного наближення вимагає вхідної тактової частоти від 50 до 200 кГц для отримання максимальної роздільної здатності. Якщо необхідна роздільна здатність нижче 10 біт, вхідна тактова частота АЦП може бути вищою за 200 кГц, щоб отримати вищу частоту дискретизації.

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

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

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

Коли використовується автоматичний запуск, попередній дільник скидається, коли відбувається подія запуску. Це забезпечує фіксовану затримку від тригерної події до початку перетворення. У цьому режимі вибірка та утримання відбувається через два такти АЦП після наростаючого фронту на тригері сигнал джерела. Для логіки синхронізації використовуються три додаткові такти ЦП.

У режимі вільного виконання нове перетворення розпочнеться відразу після завершення перетворення, тоді як ADSC залишається у стані ‘1’.

Канали диференційного підсилення.

При використанні каналів диференційного підсилення необхідно враховувати певні аспекти перетворення.

Диференційні перетворення синхронізуються з внутрішнім тактовим сигналом CK_{ADC2}, рівним половині тактового сигналу АЦП. Ця синхронізація виконується автоматично інтерфейсом АЦП таким чином, що вибірка та утримання відбувається на певній фазі CK_{ADC2}. Розпочате користувачем перетворення (тобто, усі одиничні перетворення та перше вільне перетворення), коли CK_{ADC2} в ‘0’, займе стільки ж часу, скільки і однократне перетворення (13 тактів АЦП із наступного попередньо масштабованого такту). Для перетворення, ініційовананого користувачем, коли CK_{ADC2} в ‘1’, через механізм синхронізації знадобиться 14 тактів АЦП. У вільному режимі нове перетворення розпочинається відразу після завершення попереднього перетворення, і оскільки CK_{ADC2} у цей час має високий рівень, усе запускається автоматично (тобто всі, крім першого) вільні перетворення займуть 14 тактових циклів АЦП.

Ступінь підсилення оптимізовано для смуги пропускання 4 кГц при всіх налаштуваннях підсилення. Вищі частоти можуть мати нелінійне підсилення. Зовнішній фільтр низьких частот слід використовувати, якщо вхідний сигнал містить більш високі частотні компоненти, ніж смуга пропускання каскаду підсилення. Зауважте, що тактова частота АЦП не залежить від обмеження пропускної здатності каскаду підсилення. Наприклад, тактовий період АЦП може становити 6 мкс, що дозволяє здійснювати дискретизацію каналу зі швидкістю 12 kSPS, незалежно від пропускної здатності цього каналу.

Якщо використовувані канали диференційного підсилення і перетворення запускаються за допомогою автоматичного запуску, АЦП має бути вимкнуто між перетвореннями. Коли використовується автоматичний запуск, попередній дільник АЦП скидається перед початком перетворення. Оскільки ступінь підсилення залежить від стабільного тактового генератора АЦП до перетворення, це перетворення не буде дійсним. При відключенні та повторному ввімкненні АЦП між кожним перетворенням (запис ADEN в ADCSRA в ‘0’, а потім в ‘1’), виконуються лише розширені перетворення. Результат розширених перетворень буде дійсним.

Зміна каналів або опорних напруг.

Біти MUXn і REFS1:0 у регістрі ADMUX буферизуються окремо через тимчасовий регістр, до якого ЦП має довільний доступ. Це гарантує, що вибір каналів і опорних напруг відбуватиметься лише в безпечній точці під час перетворення. The вибір каналів і опорних напруг постійно оновлюється, доки не почнеться перетворення. Після початку перетворення вибір каналу та опорної напруги блокується, щоб забезпечити достатній час дискретизації для АЦП. Безперервне оновлення відновлюється в останньому ADC перед завершенням перетворення (встановлення в ‘1’ ADIF в ADCSRA). Зверніть увагу, що перетворення починається на наступному наростаючому фронті синхронізації АЦП після запису ADSC. Таким чином, користувачеві рекомендується не записувати нові значення вибору каналу або опорної напруги в ADMUX до тих пір, поки не буде записаний один такт АЦП після ADSC.

Якщо використовується автоматичний запуск, точний час події запуску може бути невизначеним. Необхідно бути особливо обережним під час оновлення реєстру ADMUX, щоб контролювати, на яке перетворення вплинуть нові налаштування.

Якщо в ADATE і ADEN записано ‘1’, подія переривання може відбутися в будь-який час. Якщо ргістр ADMUX змінюється протягом цього періоду, користувач не може визначити, чи відбудеться наступне перетворення базуючись на старих або нових налаштуваннях. ADMUX можна безпечно оновити наступним чином способи:

1. Коли ADATE або ADEN очищено.

2. Під час перетворення потрібен мінімум один такт АЦП після події тригера.

3. Після перетворення, перед використанням прапора переривання як джерело запуску, він очищується.

Під час оновлення ADMUX в одній із цих умов нові налаштування вплинуть на наступні перетворення АЦП. Слід бути особливо обережним при зміні диференційних каналів. Як тільки диференційний канал вибрано, для підсилення може знадобитися до 125 мкс для стабілізації нового значення. Таким чином, перетворення не слід починати протягом перших 125 мкс після вибору нового диференційного каналу. Як альтернатива, результати перетворення, отримані в цьому період слід вилучити. Такий самий час встановлення слід дотримуватися для першого диференційного перетворення після зміни опорної напруги АЦП (шляхом зміни бітів REFS1:0 в ADMUX).

Вхідні канали АЦП.

При зміні вибору каналу, потрібно дотримуватися наступних вказівок, щоб переконатися, що вибрано правильний канал:

• У режимі одиночного перетворення завжди вибирати канал перед початком перетворення. Вибір каналу можна змінити за один такт АЦП після запису в ADSC. Однак найпростіший спосіб – дочекатися завершення перетворення перед зміною вибору каналу.

• У режимі вільного виконання завжди вибирається канал перед початком першого перетворення. Вибір каналу можна змінити за один такт АЦП після запису в ADSC. Однак найпростіший спосіб – дочекатися завершення першого перетворення, а потім змінити вибір каналу. Оскільки наступне перетворення вже почалося автоматично, наступний результат відображатиме попередній вибір каналу. Подальші перетворення відображатимуть новий вибір каналу.

Під час перемикання на канал диференційного підсилення перший результат перетворення може мати низьку точність через необхідний час встановлення для схеми автоматичного усунення зсуву. Тому бажано не враховувати перший результат перетворення.

Опорна напруга АЦП.

Опорна напруга для АЦП (V_{REF}) вказує діапазон перетворення для АЦП. Односторонні канали, які перевищують V_{REF}, призведуть до кодів, близьких до 0x3FF. V_{REF} можна вибрати як AVCC, внутрішній опорний сигнал 2,56 В або зовнішній контакт AREF.

AVCC підключається до АЦП через пасивний комутатор. Внутрішній опорний сигнал 2,56 В генерується з внутрішнього опорного джерела забороненої зони (VBG) через внутрішній підсилювач. У будь-якому випадку зовнішній контакт AREF підключається безпосередньо до АЦП і опорний сигнал напругу можна зробити більш стійким до шуму, підключивши конденсатор між контактом AREF і землею. VREF також можна виміряти на контакті AREF вольтметром з високим імпедансом. Зауважимо, що VREF є джерелом високого імпедансу, тому до системи слід підключати лише ємнісне навантаження. Якщо користувач має фіксоване джерело напруги, підключене до контакту AREF, користувач не може використовувати інші варіанти опорної напруги в програмі, оскільки вони будуть закорочені на зовнішню напругу. Якщо на контакт AREF не подається зовнішня напруга, користувач може перемикатися між AVCC і 2,56 В. Перший результат перетворення АЦП після перемикання джерела опорної напруги може бути неточним, тому рекомендується вилучити цей результат.

Визначення точності АЦП.

Однотактний АЦП з n-розрядами лінійно перетворює напругу між GND і VREF за $2n$ кроків (LSB). Найнижчий код читається як 0, а найвищий код читається як $2n$-1.

Наступні параметри описують відхилення від ідеальної поведінки:

• *Зсув*: відхилення першого переходу (від 0x000 до 0x001) порівняно з ідеальним переходом (при 0,5 LSB), рис. 13.5а. Ідеальне значення: 0 LSB.

• *Помилка підсилення*: після коригування зсуву помилка підсилення визначається як відхилення останнього переходу (0x3FE до 0x3FF) порівняно з ідеальним переходом (на 1,5 LSB нижче максимального), рис. 13.5б. Ідеальне значення: 0 LSB.

• Інтегральна нелінійність (INL): після коригування зсуву та похибки підсилення INL є максимальним відхиленням фактичного переходу порівняно з ідеальним переходом для будь-якого коду, рис. 13.5в. Ідеальне значення: 0 LSB.

• Диференційна нелінійність (DNL): максимальне відхилення фактичної ширини коду (інтервал між двома сусідніми переходами) від ідеальної ширини коду (1 LSB), рис. 13.5г. Ідеальне значення: 0 LSB.

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

• Абсолютна точність: максимальне відхилення фактичного (нескоригованого) переходу порівняно з ідеальним переходом для будь-якого коду. Це комплексний ефект зсуву, похибки підсилення, диференційної похибки, нелінійності та похибки квантування. Ідеальне значення: ±0,5 LSB

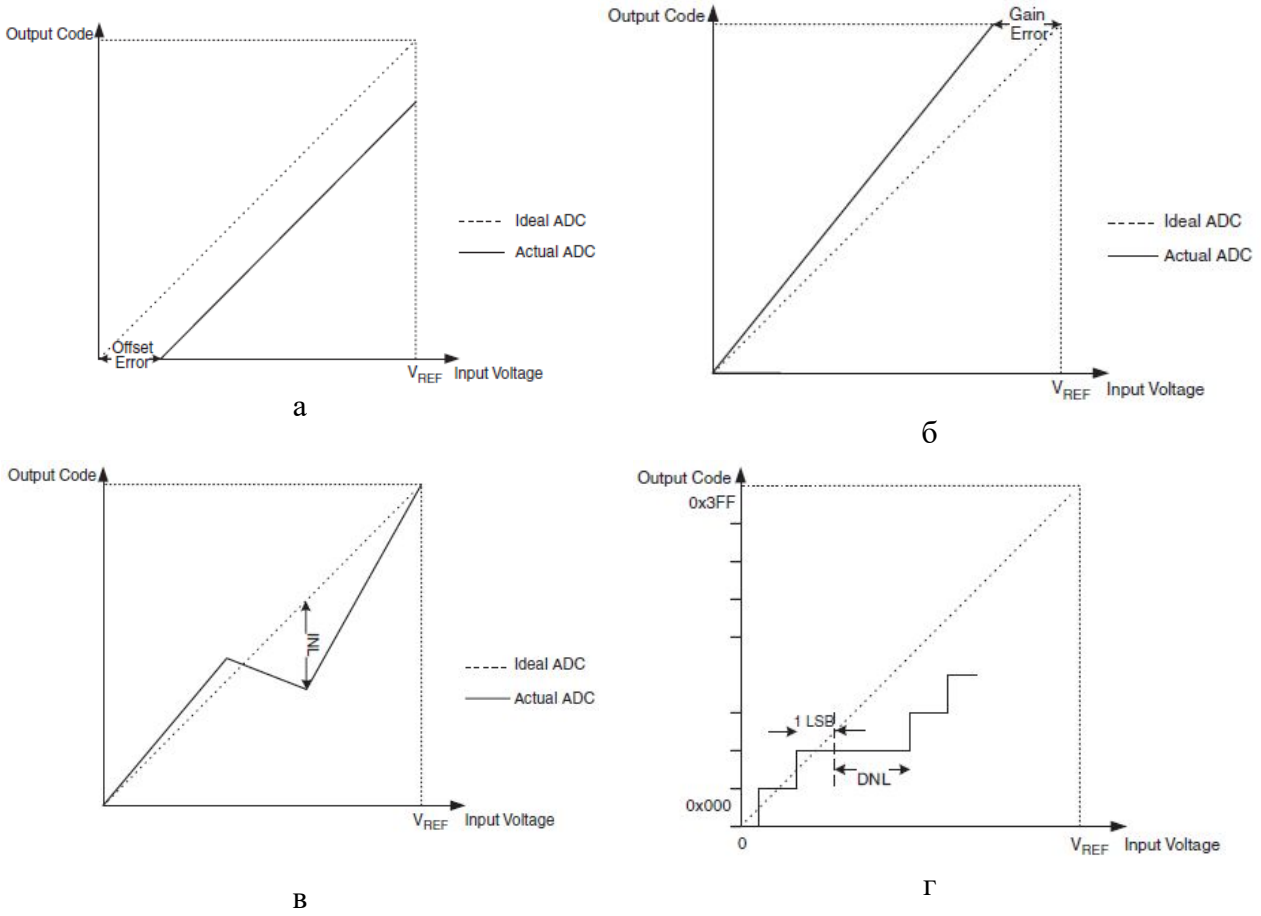


Рисунок 13.5 – Відхилення результатів перетворення АЦП від ідеального

Результати перетворення АЦП.

Після завершення перетворення (‘1’ в ADIF) результат перетворення знаходяться в регістрі ADC (ADCL, ADCH).

Для одиночного перетворення результатом є

$$ADC = \frac{V_{IN} \cdot 1024}{V_{REF}}$$

де V_{IN} – напруга на вибраному вхідному контакті, а VREF – вибрана опорна напруга. 0x000 задає аналогову землю, а 0x3FF задає вибрану опорну напругу мінус один LSB.

Якщо використовуються диференційні канали, результатом є

$$ADC = \frac{(V_{POS} - V_{NEG}) \cdot GAIN \cdot 512}{V_{REF}}$$

де V_{POS} – напруга на позитивному вхідному виводі, V_{NEG} – напруга на негативному вхідному виводі, GAIN – вибраний коефіцієнт підсилення, а VREF – вибрана опорна напруга. Результат поданий у формі доповнення до двох, від 0x200 (-512d) до 0x1FF (+511d). Зауважимо, що якщо бажано виконати швидку перевірку полярності результатів, достатньо прочитати MSB результату (ADC9 у ADCH). Якщо біт дорівнює одиниці, результат є негативним, а якщо біт дорівнює нулю, результат є позитивним. На рис. 13.6 показано значення диференційного вхідного діапазону.

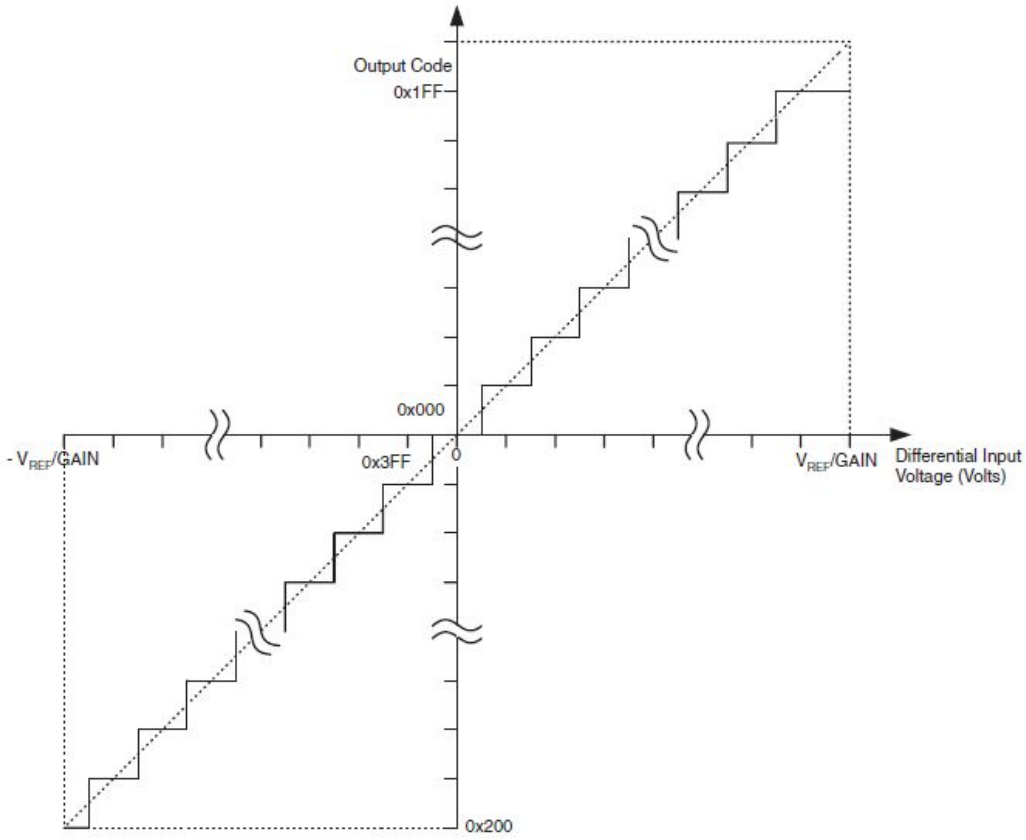


Рисунок 13.6 – Значення диференційного вхідного діапазону.

В табл. 13.1 показано отримані вихідні коди, якщо пару диференційних вхідних каналів (ADCn - ADCm) вибрано з підсиленням GAIN і опорною напругою VREF.

Таблиця 13.1 – Зв’язок між вхідною напругою і вихідними кодами

V_{ADCn}	Прочитаний код	Відповідне десяткове значення
$V_{ADCn} + V_{REF}/GAIN$	0x1FF	511
$V_{ADCn} + (511/512)V_{REF}/GAIN$	0x1FF	511
$V_{ADCn} + (510/512)V_{REF}/GAIN$	0x1FE	510
...		
$V_{ADCn} + (1/512)V_{REF}/GAIN$	0x001	1
V_{ADCn}	0x000	0
$V_{ADCn} - (1/512)V_{REF}/GAIN$	0x3FF	-1
...		
$V_{ADCn} -(511/512) V_{REF}/GAIN$	0x201	-511
$V_{ADCn} -V_{REF}/GAIN$	0x200F	-512

Приклад:

ADMUX = 0xED (ADC3 - ADC2, 10x підсилення, 2.56V опорна напруга, результат налаштований зліва)

Напруга на ADC3 300 мВ, напруга на ADC2 500 мВ.

ADCR = 512 * 10 * (300 - 500)/2560 = -400 = 0x270

В ADCL буде 0x00 і в ADCH буде 0x9C. Запис 0 в ADLAR налаштує результат справа: ADCL = 0x70, ADCH = 0x02.

2. Регістри АЦП

Регістр ADMUX (ADC Multiplexer Selection Register) – регістр керування мультиплексором

7	6	5	4	3	2	1	0
REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

• REFS1, FERSS0 (Reference Selection Bits) – біти вибору опорного джерела для АЦП.

Якщо ці біти змінити під час перетворення — зміна напруги відбудеться тільки при наступному перетворенні.

REFS1	REFS0	Вибір джерела опорної напруги
0	0	AREF, внутрішній Vref відключений
1	0	AVCC із зовнішнім конденсатором на виводі AREF
1	0	Резерв
1	1	Внутрішнє джерело опорної напруги 2,56 В на виводі AREF

• ADLAR (ADC Left Adjust Result) – відповідає за подання результату АЦП в регістрі даних. Встановлення в лог. 1 приводить до негайного зсуву результату перетворень вліво (для поточного і наступних перетворень), встановлення в лог. 0 приводить до зсуву вправо.

• MUX4,…,MUX0 (Analog Channel and Gain Selection Bits) – вибір аналогових каналів і коефіцієнтів підсилення.

Одиничні канали: 00000 – ADC0, 00001 – ADC1, 00010 – ADC2, 00011 – ADC3, 00100 – ADC4, 00101 – ADC5, 00110 – ADC6, 00111 – ADC7.

Диференціальні канали:

	Диф.вхід+	Диф. вхід-	Підсилення		Диф.вхід+	Диф. вхід-	Підсилення
01000	ADC0	ADC0	10X		...		
01001	ADC1	ADC0	10X	10100	ADC4	ADC1	1X
01010	ADC0	ADC0	200X	10101	ADC5	ADC1	1X
01011	ADC1	ADC0	200X	10110	ADC6	ADC1	1X
01100	ADC2	ADC2	10X	10111	ADC7	ADC1	1X
01101	ADC3	ADC2	10X	11000	ADC0	ADC2	1X
01110	ADC2	ADC2	200X	11101	ADC5	ADC2	1X
01111	ADC3	ADC2	200X	11110	1,22V (BG)	ADC2	1X
10000	ADC0	ADC1	1X	11111	0V(GND)		

Після завершення перетворень (ADIF лог. 1) результат знаходиться у регістрі ADC. Для одиничного перетворення результат наступний:

$$ADC = V_{IN} \cdot 1024 / V_{REF},$$

де V_{IN} – напруга на вибраному вхідному контакті;

V_{REF} – вибрана опорна напруга. 0x000 відповідає землі, 0x3FF відповідає вибраній опорній напрузі мінус мінус один найменш значущий біт.

Регістр ADCSRA – контроль та стан АЦП:

ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0

ADEN – лог. 1 включення, лог. 0 виключення поточного перетворення АЦП.

ADSC – лог. 1 в режимі окреме перетворення запускає кожне перетворення. У режимі вільного виконання запускає перше перетворення. По завершенню перетворення скидається в лог. 0.

ADFR – лог. 1 запуск, лог. 0 – зупинка режиму вільного виконання. У режимі вільного виконання АЦП робить вибірки і оновлює регістр даних неперервно.

ADIF – прапор переривання. Встановлюється в лог. 1 коли перетворення закінчено і регістр даних оновлено.

ADIE – дозвіл переривання АЦП. При лог. 1 і встановленні прапора I в SREG, то дозволені переривання завершення перетворень в АЦП.

ADPS2, ADPS1, ADPS0 – задають коефіцієнти дільника:

ADPS2	ADPS1	ADPS0	Дільник
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Регістр даних ADC (ADCL, ADCH) – містить результати перетворень.

ADLR=0

15	14	13	12	11	10	9	8	
-	-	-	-	-	-	ADC9	ADC8	ADCH
ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL

ADLR=1

15	14	13	12	11	10	9	8	
ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
ADC1	ADC0	-	-	-	-	-	-	ADCL

Регістр спеціальних функцій I/O SFIOR

7	6	5	4	3	2	1	0
ADTS2	ADTS1	ADTS0	-	ACME	PUD	PSR2	PSR10

Біти ADTS2:0 визначають джерело ініціалізації перетворення АЦП.

Якщо в ADATE в ADCSRA записано ‘1’, значення цих бітів вибирає, яке джерело ініціюватиме перетворення АЦП. Якщо в ADATE записано ‘0’, параметри ADTS2:0 не матимуть ефекту. Перетворення буде ініційовано наростаючим фронтом вибраного прапора переривання. Зауважте, що перемикання з очищеного джерела запуску на встановлене джерело запуску створить позитивний фронт сигналу запуску. Якщо ADEN в ADCSRA встановлено, це розпочне перетворення. Перехід у режим вільного запуску (ADTS[2:0]=0) не спричинить тригерну подію, навіть якщо встановлено прапор переривання ADC.

Питання.

1. Призначення і принцип роботи АЦП.
2. Основні параметри АЦП.
3. Режими перетворення АЦП.
4. Опорні напруги АЦП.
5. Режим одиничного перетворення.
6. Режим неперервного перетворення.
7. Відхилення результатів перетворення АЦП від ідеального

8. Результати перетворення АЦП.
9. Регістри АЦП.

ЛЕКЦІЯ 14. Компаратор. ЦАП

Мета. Вивчення компаратора і цифро-аналогового перетворення сигналів.

Зміст.

1. Компаратор

Модуль аналогового компаратора входить практично в усі сучасні МК. Фізично компаратор є швидкодіючим операційним підсилювачем з великим коефіцієнтом підсилення, частотною корекцією і виходом на цифровий логічний елемент. Зворотний зв'язок через зовнішній резистор з виходу на вхід не передбачається. Вихідний сигнал компаратора має низький/високий логічний рівень, який запам'ятовується в програмно-доступному регістрі.

Компаратор порівнює між собою по амплітуді дві напруги, які присутні на його позитивному і негативному входах. Результат порівняння читається з внутрішнього регістра МК і може служити джерелом переривання, рис. 14.1.

В цілому МК компаратор дуже схожий на звичайний компаратор на ІС, але тільки вихідний сигнал out, як правило, захований усередині. Ще одна відмінність – лінії аналогового компаратора можуть налаштовуватися як цифрові виходи.

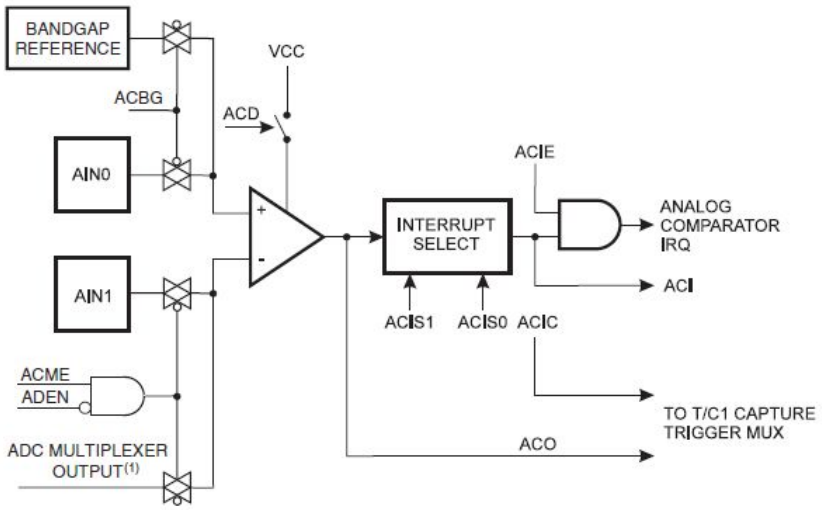


Рисунок 14.1 – Структурна схема аналогового компаратора МК ATtiny8

Аналоговий компаратор МК має два входи – AIN0 і AIN1, на які надходять аналогові сигнали, що порівнюються за величиною напруги. Коли напруга на позитивному виводі AIN0

більша ніж на негативному AIN1, вихід аналогового компаратора (analog comparator output, ACO) встановлюється у високий рівень. Вихід компаратора можна налаштувати на запуск функції запису входу таймера/лічильника 1. Крім того, компаратор може викликати окреме переривання, виключно для аналогового компаратора. Користувач може вибрати запуск переривання на компараторі за зростанням, спаданням або перемикання рівня вихідного сигналу.

Для керування роботою аналогового компаратора використовуються регістри FSIOR, ACSR.

Структура регістра FSIOR (special function IO register):

-	-	-	-	ACME	PUD	PSR2	PSR10

Коли ACME (Analog Comparator Multiplexer Enable) встановлений в 1і АЦП вимкнено (ADEN в ADCSRA дорівнює нулю), мультиплексор АЦП вибирає негативний вхід для аналогового компаратора. Коли цей біт встановлено в 0, AIN1 подається на негативний вхід аналогового компаратора.

Структура регістра ACSR (Analog Comparator Control and Status Register):

ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0

ACD (Analog Comparator Disable) . Коли цей біт встановлюється в 1, живлення аналогового компаратора вимикається. Цей біт можна встановити в будь-який час, щоб вимкнути аналоговий компаратор. Це зменшить споживання електроенергії в режимах активного та неактивного режиму. При зміні біта ACD переривання аналогового компаратора необхідно вимкнути, очистивши біт ACIE в ACSR. Інакше при зміні біта може виникнути переривання.

ACBG (Analog Comparator Bandgap Select). Коли цей біт встановлено в 1, опорна напруга з фіксованою забороненою зоною замінює позитивний вхід аналогового компаратора. Коли цей біт очищено, AIN0 подається до позитивного входу аналогового компаратора.

ACO (Analog Comparator outup). Вихід аналогового компаратора синхронізується, а потім безпосередньо підключається до ACO. Синхронізація вводить затримку на 1-2 такти.

ACI (Analog Comparator Interrupt Flag). Цей біт встановлюється апаратним забезпеченням, коли вихідна подія компаратора запускає режим переривання, визначений ACIS1 і ACIS0. Програма переривання аналогового компаратора виконується, якщо встановлені біти ACIE і біт I в SREG. ACI очищається апаратним шляхом під час виконання відповідного вектора обробки переривань. Крім того, ACI очищається шляхом запису логічної 1.

ACIE (Analog Comparator Interrupt Enable). Коли в біти ACIE і I в SREG записується 1, то активується переривання аналогового компаратора. Коли записується 0, переривання вимкнено.

ACIC (Analog Comparator Input Capture Enable). Коли записується 1, то це дозволяє функції захоплення входу в Таймер/лічильник 1 запускатися аналоговим компаратором. Вихід компаратора в цьому випадку безпосередньо підключений до інтерфейсної логіки Input Capture, завдяки чому компаратор використовує подавлювач завад і функції вибору фронту переривання входу захоплення таймера/лічильника 1. Коли записується 0, зв’язок між аналоговим компаратором і функцією вхідного розривається. Щоб компаратор ініціював переривання вхідного захоплення таймера/лічильника 1, необхідно встановити біт TICIE1 в регістрі маски переривань таймера (TIMSK).

ACIS1, ACIS0 (Analog Comparator Interrupt Mode Selected). Ці біти визначають, які події компаратора викликають переривання аналогового компаратора, табл. 14.1.

Таблиця 14.1 — Переривання аналогового компаратора

ACIS1	ACIS0	Режим переривання
0	0	Переривання компаратора при зміні рівня виходу
0	1	Резерв
1	0	Переривання компаратора за наростаючим фронтом виходу
1	1	Переривання компаратора за спадаючим фронтом виходу

```
;Початкове налаштування аналогового компаратора (в припущенні ACIE = 0)
sbi ACSR,ACI ; запис "1" в ACI прапор для його очистки
sei ; дозвіл глобальних переривань
sbi ACSR,ACIS0 ; дозволити переривання
sbi ACSR,ACIS1 ; за наростаючим фронтом сигналу
sbi ACSR,ACIE  ; дозволити переривання аналогового компаратора
```

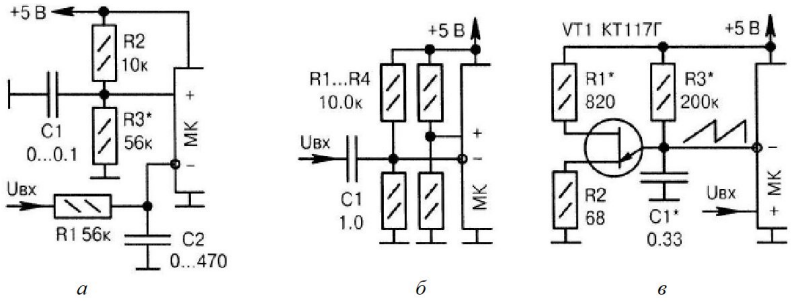
Застосування аналогового компаратора виправдане в таких випадках:

– при малій амплітуді вхідних сигналів 30...300 мВ;

– при необхідності порівняння рівнів двох сигналів (“більше/менше”);

– при підвищених вимогах до швидкодії, коли швидкодії внутрішнього АЦП недостатньо.

На рис. 14.2, *а...н* показані схеми подання сигналів на компаратор МК.

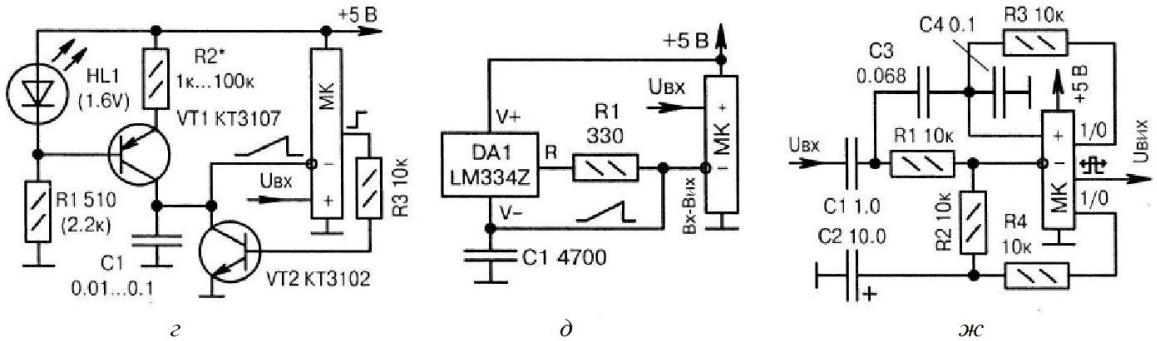


а *б* *в*

а) дільником *R2, R3* встановлюється поріг спрацьовування компаратора. Необхідність застосування фільтрувальних конденсаторів С1, С2, визначається експериментально за відсутності хибних перемикань;

б) прийом змінної напруги малої амплітуди 50... 100 мВ. Якщо форма вхідного сигналу "синусоїда", то відбувається її програмне перетворення в "прямокутник";

в) пилкоподібна напруга формується одноперехідним транзистором VT1. Усередині МК програмно підраховується час досягнення рівності напруги на обох входах;

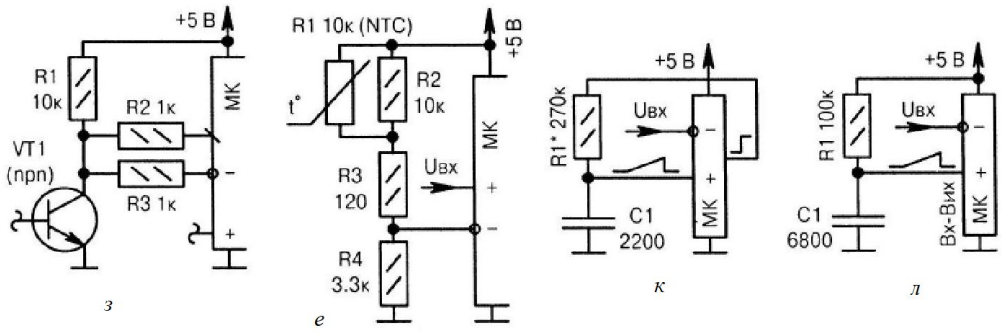


г *д* *ж*

г) пилкоподібну напругу формує сам МК через ключ VT2 (скидання) і генератор стабільного струму на елементах VT1, R1, R2, HL1. Світлодіод також служить індикатором живлення;

д) аналогічно, але з генератором стабільного струму на мікросхемі DA1. Пилкоподібна напруга може формуватися з певним періодом (режим автоматичного виміру) або одноразово (режим поодинокого виміру). У останньому випадку для економії енергії в паузах на негативному виводі компаратора має бути високий рівень;

ж) "дельта-модулятор" на основі компаратора МК. Вхідний аналоговий сигнал програмно перетворюється у вихідний цифровий сигнал Uвих, що модулюється методом ШІМ;



з) вхідний сигнал може одночасно поступати на компаратор і на звичайну лінію порта МК, наприклад, для повторної перевірки показів або для організації сервісних функцій;

е) поріг спрацьовування аналогового компаратора МК залежить від температури довкілля (терморезистор R1). Резистор R2 лінеаризує температурну характеристику;

к) пилкоподібна (експоненціальна) напруга на конденсаторі С1 формується за допомогою подання високого/низького рівня на верхньому за схемою виводі резистора R1;

л) аналогічно, але з використанням поєднаної лінії порту, через яку періодично розряджається конденсатор С1 низьким рівнем.

Рисунок 14.2 – Схеми подання сигналів на компаратор МК

2. Цифро-аналогові перетворювачі

2.1. Загальні відомості

Цифро-аналоговий перетворювач (ЦАП) призначений для перетворення числа, визначеного, як правило, у вигляді двійкових кодів, у напругу або струм пропорційно значенню цифрового коду.

Дуже часто ЦАП входить до складу мікропроцесорних систем. У цьому випадку, якщо не потрібна висока швидкодія, цифро-аналогове перетворення може бути дуже просто здійснено за допомогою ШІМ. Схема ЦАП з ШІМ показана на рис. 14.3.

Найпростіше організовується цифро-аналогове перетворення в тому випадку, якщо МК має вбудовану функцію ШІМ. Вихід ШІМ керує ключем *Тім*. У залежності від заданої розрядності перетворення (можливі режими 8, 9 і 10 розрядів) контролер за допомогою власного таймера/лічильника формує послідовність імпульсів, відносна тривалість яких $\gamma = Тім/T$ визначається співвідношенням

$$\gamma = D/2^N \qquad (1.1)$$

де N – розрядність перетворення, а D – код перетворення.

Фільтр нижніх частот згладжує імпульси, виділяє середнє значення напруги. У результаті вихідна напруга перетворювача (формула 1.2):

$$Vout = \gamma\ Vref = Dvref / 2^N \qquad (1.2)$$

Розглянута схема забезпечує ідеальну лінійність перетворення, не містить прецизійних елементів (за винятком джерела опорної напруги). Основний її недолік – низька швидкодія.

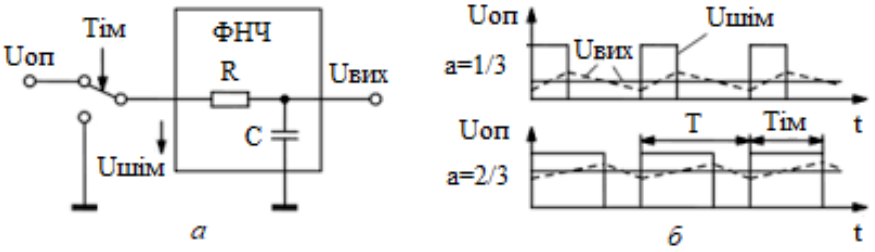


Рисунок 14.3 – ЦАП з ШІМ модуляцією: а) структурна схема; б) часова діаграма

Паралельні ЦАП мають більшу швидкодію і тому вони можуть застосовуватися для більш широкого кола задач. Більшість схем паралельних ЦАП реалізована на додаванні струмів, що пропорційні вазі цифрових двійкових розрядів, причому повинні додаватись тільки струми тих розрядів, значення яких дорівнює 1.

ЦАП випускаються як окремі інтегральні схеми (ІС). У загальному випадку ІС ЦАП показана на рис. 14.4.

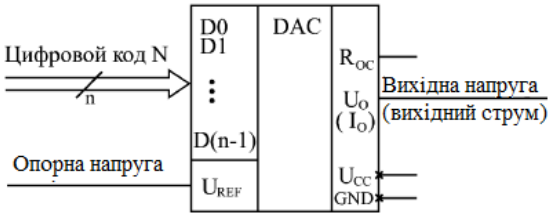


Рисунок 14.4 – ІС ЦАП

На цифрові входи ЦАП подається n-розрядний код N, на аналоговий вхід – опорна напруга *Uоп* (U_{REF}). Вихідним сигналом є напруга *Uвих* (U_O) або струм *Iвих* (I_O). При цьому вихідний струм або вихідна напруга пропорційні вхідному коду та опорній напрузі. Для деяких мікросхем опорна напруга повинна мати строго заданий рівень, для інших допускається змінювати її значення в широких межах, у тому числі змінювати її полярність (позитивну на негативну і навпаки). ЦАП з великим діапазоном зміни опорної напруги називається помножуючим ЦАП, так як його можна легко використовувати для множення вхідного коду будь-яку опорну напругу.

Крім інформаційних сигналів, мікросхеми ЦАП потребують також підключення одного або двох джерел живлення та загального дроту. Зазвичай цифрові входи ЦАП забезпечують сумісність із стандартними виходами мікросхем ТТЛ.

Найчастіше у випадку, якщо ЦАП має струмовий вихід, його вихідний струм перетворюється на вихідну напругу за допомогою зовнішнього операційного підсилювача та вбудованого в ЦАП резистора $R_{ЗЗ}$, один із виводів якого виведений на зовнішні виводи мікросхеми (рис. 14.5). Тому, якщо не обговорено інше, надалі вважатимемо, що вихідний сигнал ЦАП – напруга *Uo*

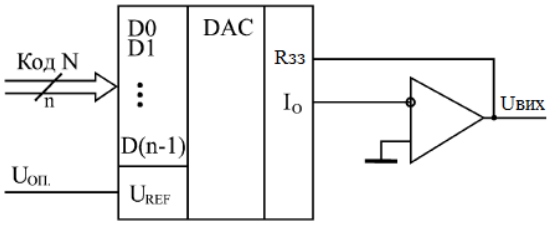
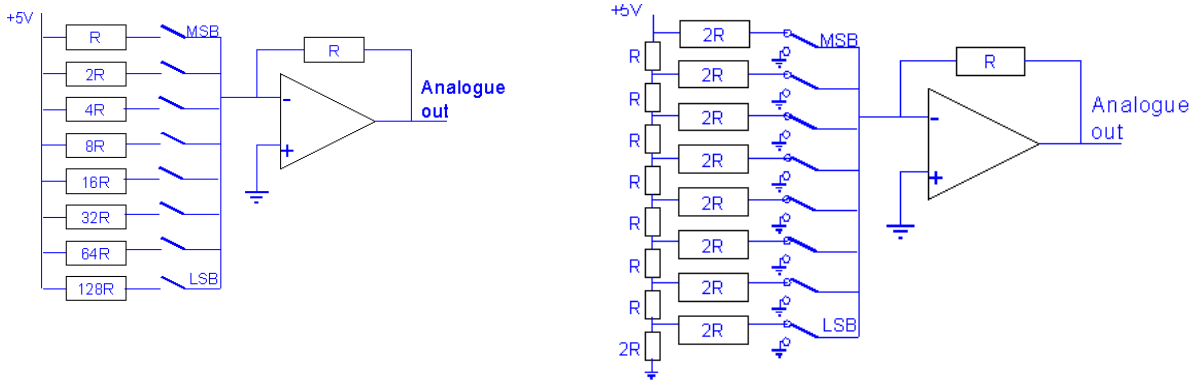


Рисунок 14.5 – Підключення виводів ЦАП

Суть перетворення вхідного цифрового коду у вихідний аналоговий сигнал досить проста. Вона полягає у підсумовуванні кількох струмів (за кількістю розрядів вхідного коду), кожен наступний з яких удвічі більший за попередній. Для отримання цих струмів використовуються транзисторні джерела струму, або резистивні матриці, комутовані транзисторними ключами. Як приклад на рис. 14.6 показано цифро-аналогове перетворення на основі резистивної матриці R-2R-4R-8R-…-128R та R-2R та ключів (насправді використовуються ключі на основі транзисторів).



а *б*

Рисунок 14.6 – ЦАП з резистивною матицею:
а) струм ділиться дільником у степені 2^n (1, 2, 4 …);
б) використання у дільнику резисторів двох номіналів

Відповідно до концепції віртуального короткого замкнення, напруга на інвертуючому вхідному терміналі операційного підсилювача така ж, як і напруга на його неінвертуючому вхідному терміналі. Таким чином, напруга на вузлі інвертуючої вхідної клеми буде нуль вольт. Тоді рівняння вузлових струмів на інвертуючому вході (рис. 14.6), а буде:

$$\frac{0+V_R b_2}{2^0 R}+\frac{0+V_R b_1}{2^1 R}+\frac{0+V_R b_0}{2^2 R}+\frac{0-V_0}{R_f}=0$$

$$=>\frac{V_0}{R_f}=\frac{V_R b_2}{2^0 R}+\frac{V_R b_1}{2^1 R}+\frac{V_R b_0}{2^2 R}$$

$$=>V_0=\frac{V_R R_f}{R}\left\{\frac{b_2}{2^0}+\frac{b_1}{2^1}+\frac{b_0}{2^2}\right\}$$

Підставляючи R=2Rf, отримаємо:

$$=>V_0=\frac{V_R R_f}{2R_f}\left\{\frac{b_2}{2^0}+\frac{b_1}{2^1}+\frac{b_0}{2^2}\right\}$$

$$=>V_0=\frac{V_R}{2}\left\{\frac{b_2}{2^0}+\frac{b_1}{2^1}+\frac{b_0}{2^2}\right\}$$

Отримане рівняння є рівнянням вихідної напруги 3-бітового ЦАП із двійково зваженими резисторами. Оскільки кількість бітів на двійковому (цифровому) вході становить три, можна отримати сім можливих значень вихідної напруги, змінюючи двійковий вхід від 000 до 111 для фіксованої опорної напруги V_R.

Можна написати узагальнене рівняння вихідної напруги N-розрядного ЦАП з двійково зваженими резисторами, на основі рівняння вихідної напруги 3-розрядного ЦАП з двійково зваженими резисторами:

$$=>V_0=\frac{V_R}{2}\left\{\frac{b_{N-1}}{2^0}+\frac{b_{N-2}}{2^1}+\ldots.+\frac{b_0}{2^{N-1}}\right\}$$

Недоліки ЦАП з двійково зваженими резисторами:

– різниця між значеннями опору, що відповідають LSB і MSB, буде збільшуватися зі збільшенням кількості бітів, присутніх на цифровому вході;

– важко розробити більш точні резистори, оскільки кількість бітів, присутніх на цифровому вході, збільшується.

У зв’язку із складністю виготовлення резисторів різних номіналів з високою точністю, більшого поширення набули резистивні матриці з двох номіналів резисторів R, 2R. Замкнутому положенню ключа відповідає одиниця у цьому розряді вхідного коду N (розряди D0…D3). Операційний підсилювач може бути як вбудованим (у разі ЦАП з виходом по напрузі), так і зовнішнім (у разі ЦАП з виходом по струму). Деякі ЦАП можуть бути під’єднані до паралельних портів МК.

Першим ключем комутується струм величиною U_{REF}/2R, другим ключем – струм U_{REF}/4R, третім – струм U_{REF}/8R, четвертим – струм U_{REF}/16R. Тобто струми, що комутуються сусідніми ключами, різняться вдвічі, як і ваги розрядів двійкового коду. Струми, комутовані всіма ключами, підсумовуються і перетворюються на вихідну напругу за допомогою операційного підсилювача з опором $R_{ЗЗ}$=R ланцюга негативного зворотного зв'язку. При замкнутому положенні кожного ключа (одиниця у відповідному розряді вхідного коду ЦАП) струм, що комутується цим ключем, надходить на підсумовування. При розімкнутому положенні ключа (нуль у відповідному розряді вхідного коду ЦАП) струм, що комутується цим ключем, на підсумовування не надходить. Сумарний струм I_O від усіх ключів створює на виході

операційного підсилювача напругу $U_O=I_O \cdot R_{ЗЗ}=I_O R$. Тобто внесок першого ключа (старшого розряду коду) у вихідну напругу становить $U_{REF}/2$, другого – $U_{REF}/4$, третього – $U_{REF}/8$, четвертого – $U_{REF}/16$. Таким чином, при вхідному коді N = 0000 вихідна напруга схеми буде нульовою, а при вхідному коді N=1111 вона дорівнює – $15U_{REF}/16$.

У загальному випадку вихідна напруга ЦАП при $R_{ЗЗ}$=R буде пов'язана з вхідним кодом N і опорною напругою U_{REF} простою формулою $U_{ВИХ} = -N \cdot U_{REF} \cdot 2^{-n}$ де n – кількість розрядів вхідного коду. Знак мінус виходить через інверсію сигналу операційним підсилювачем. Цей зв'язок показано в табл. 14.2.

Таблиця 14.2 – Перетворення ЦАП в однополярному режимі

Вхідний код	Вихідна напруга
000...000	0
000...001	-2^{-n} UREF
...	
100…000	-2^{-1} UREF
...	
111…111	$(1-2^{-n})$ UREF

Деякі мікросхеми ЦАП передбачають можливість роботи у біполярному режимі, у якому вихідна напруга змінюється не від нуля до U_{REF}, а від $-U_{REF}$ до $+U_{REF}$. При цьому вихідний сигнал ЦАП $U_{ВИХ}$ множиться на 2 і зсувається на величину U_{REF}. Зв'язок між вхідним кодом N і вихідною напругою $U_{ВИХ}$ буде наступною: $U_{ВИХ} = U_{REF} (1-N \cdot 2^{1-n})$, що показано в табл. 14.3. Таке біполярне перетворення при можливості зміни знака опорної напруги називається також чотириквадрантним множенням (тобто і опорна, і вихідна напруга можуть бути в даному випадку як позитивними, так і негативними).

Таблиця 14.3 – Перетворення ЦАП в біполярному режимі

Вхідний код	Вихідна напруга
000...000	U_{REF}
...	
011...111	-2^{-n} UREF
100…000	0
...	
111…111	$-(1-2^{-n})$ UREF

Наявні на ринку мікросхеми ЦАП різняться кількістю розрядів (від 8 до 24), величиною затримки перетворення (від одиниць наносекунд до одиниць мікросекунд), припустимою величиною опорної напруги (зазвичай – одиниці вольт), величинами похибок перетворення та

іншими параметрами. Розрізняються вони також технологією виготовлення та особливостями внутрішньої структури, що нерідко накладає обмеження їх використання. Тому вибирати мікросхему ЦАП для конкретного застосування необхідно з використанням докладної інформації, що надається фірмами-виробниками.

Розглянемо загальні принципи включення ЦАП до цифрових схем без урахування їх окремих особливостей. Іноді буває необхідно зменшити кількість розрядів ЦАП. Для цього необхідно подати сигнали логічного нуля на необхідне число молодших розрядів ЦАП (але не старших розрядів). На рис. 14.7 показано, як з 10-розрядного ЦАП можна зробити 8-розрядний, подавши нулі на два молодші розряди. Збільшення кількості розрядів ЦАП є набагато складнішим завданням, що вимагає побудови складних аналогових схем, тому воно зустрічається досить рідко. Значно простіше підібрати мікросхему з потрібною чи більшою, ніж потрібно, кількістю розрядів.

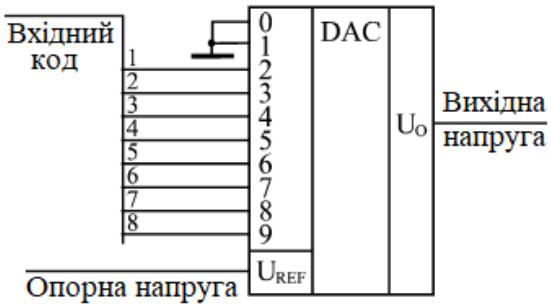


Рисунок 14.7 – Підключення до ЦАП меншої кількості входів

2. Перетворення послідовності кодів у вихідну напругу

Основне застосування мікросхем ЦАП полягає у отриманні аналогового сигналу із послідовності цифрових кодів (рис. 14.8).

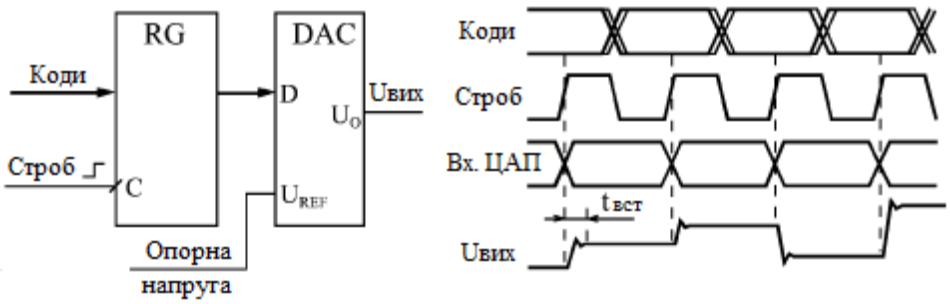


Рисунок 14.8 – Перетворення кодів в аналоговий сигнал

Як правило, коди подаються на входи ЦАП через паралельний регістр, що дозволяє забезпечити одночасність зміни всіх розрядів коду ЦАП. При неодночасній зміні розрядів вхідного коду на виході ЦАП з'являються короткі імпульси напруги, рівні яких не відповідають жодному з кодів. Проте, навіть за одночасної зміні всіх розрядів вхідного коду ЦАП, рівень напруги, відповідний поданому коду, встановлюється не відразу, а за час встановлення ЦАП $t_{вст}$, що зв'язано з неідеальністю внутрішніх елементів ЦАП. Вихідний струм ЦАП, як правило, встановлюється значно швидше за вихідну напругу, так як він не залежить від інерційності операційного підсилювача. Зрозуміло, що умова правильної роботи ЦАП полягає в тому, щоб

тривалість збереження вхідного коду була більшою, ніж час встановлення ЦАП $t_{вст}$, інакше вихідний сигнал не встигне прийняти значення, що відповідає вхідному коду.

Якщо подавати коди на вхід ЦАП рідко, наведена на рис. 8 схема може використовуватися, наприклад, в керованому джерелі живлення, вихідна напруга якого визначається вхідним кодом. Правда, при цьому необхідно забезпечити великий вихідний струм джерела живлення, застосувавши зовнішній підсилювач струму. Якщо ж подавати коди на вхід ЦАП з високою частотою, то можна отримати генератор (він синтезатор) аналогових сигналів довільної форми. У цьому випадку коди, що надходять на ЦАП, називають кодами вибірок (тобто миттєвих значень) генерованого аналогового сигналу.

Генератор пилкоподібних сигналів. У найпростішому випадку джерелом вхідних кодів ЦАП можна використовувати звичайний двійковий лічильник (рис. 14.9).

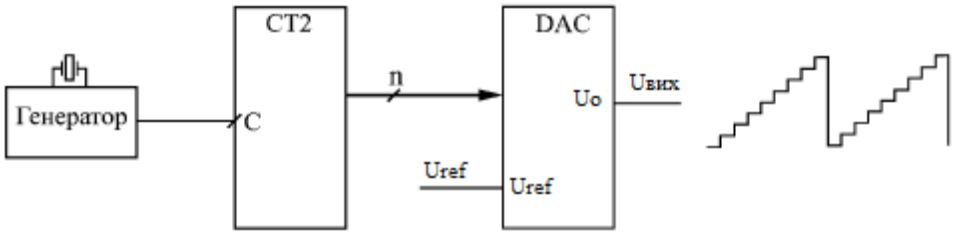


Рисунок 14.9 – Генератор пилкоподібного аналогового сигналу

Вихідна напруга ЦАП наростатиме при цьому на величину $2^{-n}U_{REF}$ з кожним тактовим імпульсом, формуючи пилкоподібні вихідні сигнали амплітудою U_{REF}. Тривалість кожної сходинки дорівнює періоду тактового генератора T, а період вихідного сигналу дорівнює 2^nT. Кількість сходинок у періоді вихідного сигналу дорівнює 2^n. Якщо в даній схемі використовувати синхронні лічильники із синхронним перенесенням, то вхідний регістр ЦАП не потрібен, тому що всі розряди лічильника перемикаються одночасно. Якщо використовуються асинхронні лічильники або синхронні лічильники з асинхронним перенесенням, то вхідний регістр ЦАП необхідний.

Генератор сигналів довільної форми. У випадку, коли потрібно формувати аналогові сигнали довільної форми (синусоїдальні, дзвоноподібні, шумові, трикутні, імпульсні і т.д.), як джерело кодів, що надходять на ЦАП, необхідно використовувати пам'ять, що працює в режимі читання (рис. 14.10).

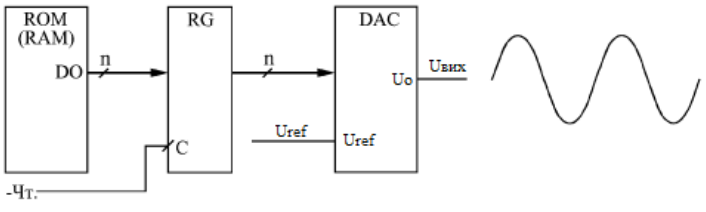


Рисунок 14.10 – Генератор сигналів довільної форми

Якщо пам’ять постійна, то набір форм сигналів, що генеруються, задається раз і назавжди. Якщо ж пам’ять оперативна, то будується односпрямований інформаційний буфер з періодичним режимом роботи, що дозволить записувати в пам'ять коди для генерації різних сигналів. В обох випадках необхідний вхідний регістр ЦАП, інформація записується стробом читання з пам'яті. Як і в попередньому випадку, вихідний сигнал ЦАП складатиметься зі

сходинок, висота яких кратна $2^{-n}U_{REF}$. Амплітуда вихідного сигналу не перевищує U_{REF}. Якщо адреси пам'яті перебираються лічильником, період вихідного аналогового сигналу дорівнює 2^mT, де T – період тактового сигналу читання з пам'яті "-Чт.", а m – кількість адресних розрядів пам'яті.

3. Обчислення кодів вибірок сигналів

Якщо треба обчислити коди вибірок для генерації якогось періодичного сигналу, то його період розділити на 2^m частин і обчислити відповідні 2^m значень цього сигналу Ui. Потім потрібно перерахувати значення сигналу в коди за формулою $Ni = 2^nUi/A$ де A – амплітуда сигналу, і взяти найближче ціле значення коду. Нульове значення сигналу дасть при цьому нульовий код 000...000, максимальне значення сигналу (рівне амплітуді A) дасть максимальний код 111...111. В результаті подачі цих кодів на ЦАП з періодом T генеруватиметься аналоговий сигнал необхідної форми з амплітудою, що дорівнює U_{REF} і з періодом $T_{ВИХ}=2^mT$. Приклад такого обчислення показано на рис. 14.11.

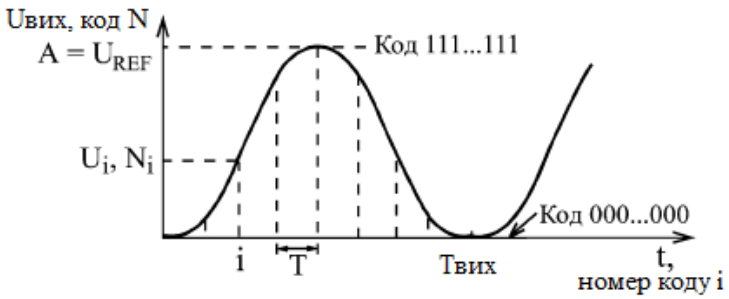


Рисунок 14.11 – Обчислення кодів вибірок

Перетворення цифрових кодів в аналоговий сигнал – це єдине застосування мікросхем ЦАП. Вони можуть також використовуватися для керованої обробки аналогових сигналів, наприклад, для посилення та ослаблення аналогових сигналів у задане число разів. Для цього найкраще підходять помножуючі ЦАП, які допускають зміну рівня опорної напруги в широких межах, у тому числі зі зміною його знаку. Таких мікросхем ЦАП випускається зараз досить багато, з різною швидкодією та різною кількістю розрядів вхідного коду.

Найпростіша схема – це *цифровий атенюатор аналогового сигналу* (рис. 14.12), що застосовується часто для регулювання амплітуди вихідного сигналу генератора на основі ЦАП.

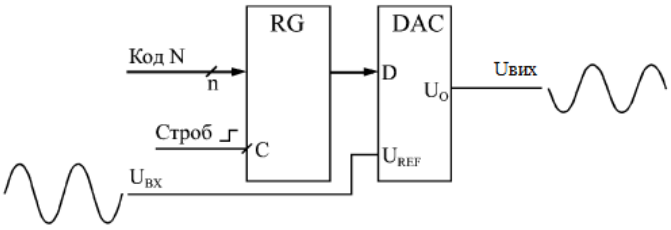


Рисунок 14.12 – Цифровий атенюатор аналогового сигналу

Схема майже нічим не відрізняється від схеми на рис. 8. Але дві важливі відмінності все ж таки є: замість постійної опорної напруги подається змінний аналоговий сигнал, а ЦАП повинен бути обов'язково помножуючим. Вихідний сигнал пов'язаний із вхідним за простою формулою

$U_{ВИХ}$=-$U_{ВХ}$·$N2^{-n}$, тобто вихідний сигнал пропорційний вхідному (з інверсією), а коефіцієнт пропорційності визначається вхідним цифровим кодом N. Коефіцієнт пропорційності змінюється в даному випадку від нуля та майже до одиниці з кроком, рівним 2^{-n}. Вхідний регістр ЦАП у цьому випадку також необхідний, оскільки при неодночасному перемиканні розрядів вхідного коду на вихідний сигнал ЦАП можуть накладатися короткі імпульси значної амплітуди. Вимоги до швидкодії ЦАП (до величини його часу встановлення) у цьому включенні не дуже високі, оскільки амплітуду вихідного сигналу зазвичай потрібно змінювати нечасто. А частота вхідного аналогового сигналу може бути досить великою, вона не пов’язана з часом встановлення ЦАП.

Керований підсилювач вхідного сигналу. Існує також схема включення ЦАП, яку можна використовувати як керований підсилювач аналогового сигналу з коефіцієнтом підсилення, що задається вхідним кодом N (рис. 14.13).

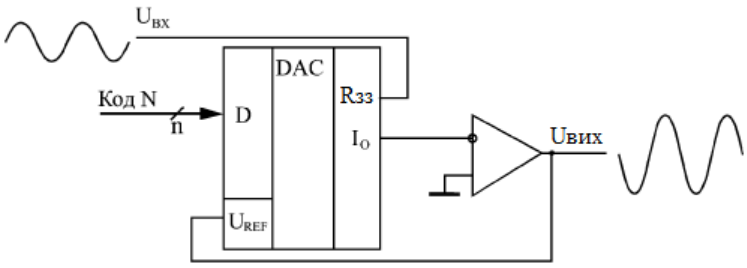


Рисунок 14.13 – Керований підсилювач вхідного сигналу

У цьому випадку вихідний струм ЦАП дорівнює величині $U_{ВХ}/R_{ОС}$, а як опорна напруга використовується вихідна напруга, то виходить, що вихідна напруга пов'язана з вхідною за формулою $U_{ВИХ}$=-$U_{ВХ}$·$N2^{-n}/N$ Тобто коефіцієнт пропорційності між вихідною і вхідною напругою обернено пропорційний коду N. Код N може змінюватися в цьому випадку від 1 до (2^{n-1}), що відповідає коефіцієнту підсилення від приблизно одиниці до 2^n. Наприклад, при 10-розрядному ЦАП коефіцієнт підсилення схеми може досягати 1024. Як і в попередньому випадку, швидкість перемикання ЦАП не дуже важлива, оскільки коефіцієнт підсилення зазвичай не потрібно перемикати занадто часто. На схемі для простоти не показаний вхідний регістр ЦАП, який знову ж таки необхідний, щоб забезпечити одночасність перемикання всіх розрядів вхідного коду.

Послідовне вмикання атенюатора і підсилювача. Використовуючи послідовне вмикання схем рис. 14.12 та рис. 14.13 можна забезпечити приведення до стандартного рівня вхідної напруги, що змінюється в дуже широких межах (рис. 14.14).

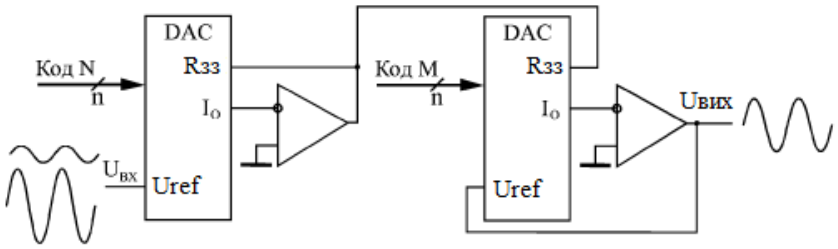


Рисунок 14.14 – Послідовне вмикання атенюатора і підсилювача

Таке завдання часто зустрічається в аналого-цифрових системах. Коефіцієнт передачі всієї схеми буде дорівнювати відношенню вхідних кодів N/M обох ЦАП і може бути встановлений з високою точністю як в діапазоні від 0 до 1 (атенюатор), так і в діапазоні від 1 до $2n$ (підсилювач). На схемі не показані вхідні регістри обох ЦАП, але вони також потрібні.

Схема зсуву аналогового сигналу на величину, що задається вхідним цифровим кодом. Зсув є, власне, додавання аналогового сигналу з постійною напругою. Таке завдання часто зустрічається в аналого-цифрових системах. Схема зсуву (рис. 14.15) включає перетворювач цифрового коду у вихідну напругу і аналоговий суматор на операційному підсилювачі. Величина напруги зсуву вхідного сигналу дорівнюватиме $U_{REF} \cdot 2^{-n} N$. Оскільки застосовуються два інвертуючі операційні підсилювачі, інверсії вхідного сигналу на виході в даному випадку не буде. Якщо потрібен як позитивний, і негативний зсув, необхідно застосовувати ЦАП з біполярним вихідним сигналом.

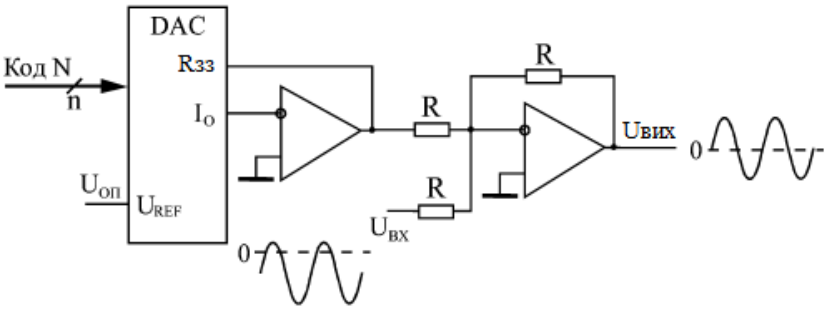


Рисунок 14.15 – Схема зсуву аналогового сигналу

5. ЦАП в мікроконтролерах

Деякі МК AVR (табл. 14.4) мають модуль ЦАП. Нижче наведено основні характеристики цього модуля:

– 10-/12-розрядна роздільна здатність;

– частота перетворення до 1 МГц;

– гнучкий діапазон перетворення;

– кілька джерел запуску;

– як вихід ЦАП може бути один неперервний вихід в одноканальному режимі роботи або два окремих виходи зі схемою вибірки-зберігання в двоканальному режимі роботи.

– вбудоване калібрування зміщення і коефіцієнта передачі;

– внутрішнє або зовнішнє джерело опорної напруги;

– можливість використання як вхід для аналогового компаратора та АЦП;

– наявність енергозберігаючого режиму роботи;

Таблиця 14.4 — Характеристики ЦАП

МК	Каналів	Розрядів, біт	МК	Каналів	Розрядів, біт
ATxmega64A1	4	12	ATxmega16A4	2	12
ATxmega128A1	4	12	ATxmega32A4	2	12
ATxmega192A1	4	12	ATxmega64A4	2	12
ATxmega256A1	4	12	ATmega16M1	1	10

ATxmega384A1	4	12	ATmega32M1	1	10
ATxmega64A3	2	12	ATmega64M1	1	10
ATxmega128A3	2	12	AVR32/64/128DA	1	10
ATxmega192A3	2	12	AVR32/64/128DB	1	10
ATxmega256A3	2	12	AVR32/64/128DD	1	10

Структурна схем МК ATmega 16M1/32M1/64M1 показана на рис. 14.16.

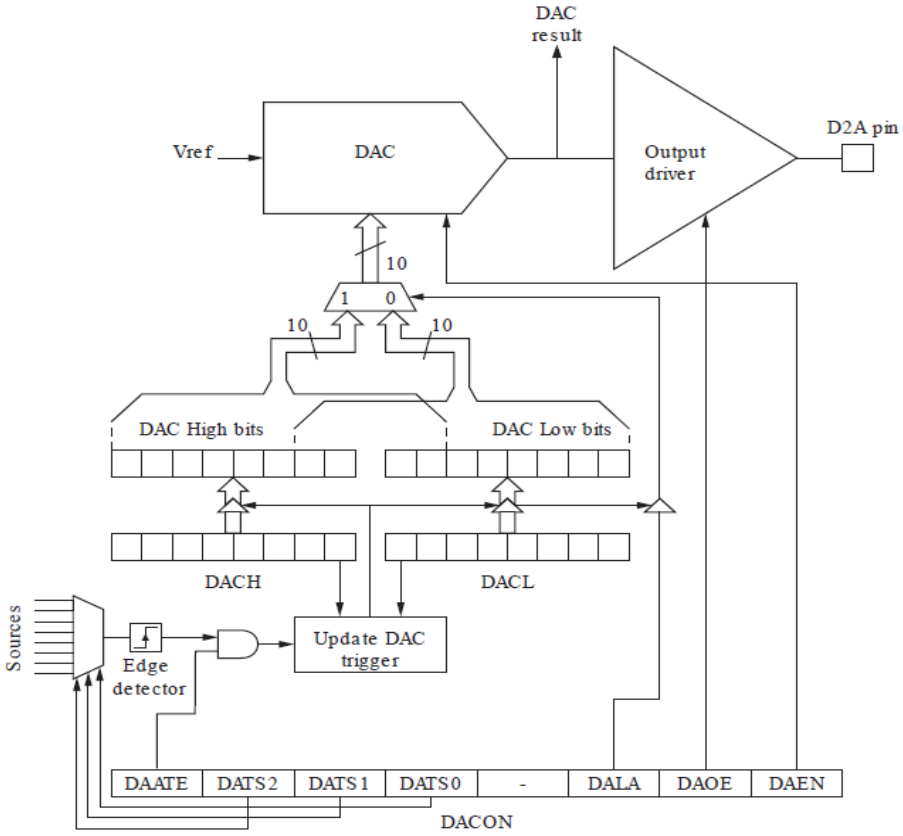


Рисунок 14.16 – Структурна схема МК ATmega 16M1/32M1/64M1

ЦАП генерує аналоговий сигнал пропорційний до значення на вхідних регістрах. Для точного контролю частоти дискретизації є можливість оновити значення на вхідних регістрах через різні тригерні події.

5.1. Початок перетворень

Для налаштування ЦАП використовується регістр DACON. Як тільки біт DAEN у регістрі DACON встановлено, DAC перетворює значення, наявне в регістрах DACH і DACL, відповідно до налаштування регістра DACON.

Крім того, перетворення може бути ініційовано автоматично різними джерелами. Автоматичний запуск вмикається встановленням біта ввімкнення автоматичного запуску DAC, DAATE у DACON. Джерело тригера вибирається установкою біта вибору тригера ЦАП, DATS у DACON. Коли на вибраному тригерному сигналі виникає позитивний фронт, DAC перетворює значення, наявне в регістрах DACH і DACL, відповідно до налаштування регістра DACON. Це

забезпечує спосіб запуску перетворень через фіксовані проміжки часу. Якщо після завершення перетворення тригерний сигнал все ще встановлений, нове перетворення не розпочнеться. Якщо під час перетворення на тригерному сигналі виникає інший позитивний фронт, цей фронт ігноруватиметься. Зверніть увагу, що прапор переривання буде встановлено, навіть якщо певне переривання вимкнено або біт глобального дозволу переривання в SREG очищено. Таким чином, перетворення може бути запущено без переривання. Однак прапор переривання має бути очищено, щоб запустити нове перетворення під час наступної події переривання.

5.2. Джерела опорної напруги

Опорна напруга АЦП (VREF) вказує діапазон перетворення для ЦАП. VREF можна вибрати як AV_{CC}, внутрішній опорний сигнал 2,56 В або як зовнішній контакт AREF.

AV_{CC} підключається до ЦАП через пасивний комутатор. Внутрішній опорний сигнал 2,56 В генерується з внутрішнього опорного джерела забороненої зони (V_{BG}) через внутрішній підсилювач. У будь-якому випадку зовнішній контакт AREF підключається безпосередньо до ЦАП, і опорну напругу можна зробити більш стійкою до шуму, підключивши конденсатор між контактом AREF і землею. VREF також можна виміряти на контакті AREF за допомогою вольтметра з високим опором. Зауважте, що VREF є джерелом з високим опором, тому до системи слід підключати лише ємнісне навантаження.

Якщо користувач має фіксоване джерело напруги, підключене до контакту AREF, користувач не може використовувати інші варіанти опорної напруги в програмі, оскільки вони будуть закорочені на зовнішню напругу. Якщо до контакту AREF не подається зовнішня напруга, користувач може перемикатися між AV_{CC} і 2,56 В як до опорного вибору. Перший результат перетворення ЦАП після перемикання джерела опорної напруги може бути неточним, тому користувачеві рекомендується відкинути цей результат.

5.3. Регістри ЦАП

DCON – регістр контролю перетворення цифрового значення в аналогове (Digital to Analog Conversion Control Register)

Регістр DCON:

DAATE	DATS[2:0]			-	DALA	DAOE	DAEN
7	6	5	4	3	2	1	0

DAATE (DAC Auto Trigger Enable): дозвіл ЦАП на автоматичний запуск.

Цей біт встановлюється для оновлення вхідного значення ЦАП за позитивним фронтом тригерного сигналу, вибраного за допомогою біта DACTS2-0 у регістрі DACON. Цей біт скидується, щоб автоматично оновлювати вхід ЦАП, коли значення записується в регістр DACH.

DATS[2:0] (DAC Trigger Selection): вибір тригера ЦАП

Ці біти необхідні лише у випадку, якщо ЦАП працює в режимі автоматичного запуску. Це означає, що встановлено біт DAATE. Ці три біти вибирають подію переривання, яка генеруватиме оновлення вхідних значень ЦАП. Оновлення буде створено за наростаючим фронтом вибраного прапора переривання, незалежно від того, увімкнено переривання чи ні.

Значення	Описання
000	Аналоговий компаратор 0
001	Аналоговий компаратор 1
010	Запит зовнішнього переривання 0
011	Timer/Counter 0 порівняння співпадіння
100	Timer/Counter 0 переповнення
101	Timer/Counter 1 порівняння співпадіння B
110	Timer/Counter 1 переповнення
111	Timer/Counter захоплення події

DALA (Digital to Analog Left Adjust): цифро-аналогове перетворення з вирівнюванням вліво.

Встановлюється цей біт для налаштування ліворуч вхідних даних ЦАП. Очищується цей біт для налаштування праворуч вхідних даних ЦАП. Біт DALA впливає на конфігурацію регістрів даних ЦАП. Зміна цього біта впливає на вихід ЦАП на наступному записуванні DACH.

DAOE (Digital to Analog Output Enable): дозвіл виведення цифро-аналогового перетворення.

Цей біт встановлюється для виведення результату перетворення на D2A. Біт очищується для внутрішнього використання ЦАП.

DAEN (Digital to Analog Enable): дозвіл цифро-аналогового перетворення.

Встановлення цього біту вмикає ЦАП. Чистка його біту вимикає DAC.

DAC (Digital to Analog Converter input Register) – вхідний регістр цифро-аналогового перетворювача.

Регістри DACH і DACL містять значення, яке потрібно перетворити в аналогову напругу. Запис у регістр DACL забороняє оновлення вхідного значення, доки не буде записано DACH. Тож звичайний спосіб запису 10-бітового значення в регістр DAC – це спочатку запис DACL у DACH. Щоб легше працювати лише з вісьмома бітами, є можливість налаштувати вхідне значення ліворуч. Таким чином, достатньо написати DACH, щоб оновити значення DAC.

Для роботи з 10-розрядним ЦАП необхідно оновити два регістри. Щоб уникнути проміжного значення, вхідні значення ЦАП, які дійсно перетворюються в аналоговий сигнал, буферизуються в недоступних регістрах. У звичайному режимі оновлення тіньового регістру виконується, коли записується регістр DACH.

Якщо встановлено біт DAATE, вхідні значення ЦАП оновлюватимуться за подією запуску, вибраною за допомогою бітів DATS.

Щоб уникнути неправильних вхідних значень ЦАП, оновлення можна виконати лише після запису регістрів DACL і DACH відповідно. Можна працювати з 8-бітною конфігурацією, лише записавши значення DACH. У цьому випадку оновлення виконується для кожної події тригера.

Якщо біт DAATE очищено, ЦАП перебуває в режимі автоматичного оновлення. Запис у регістр DACH автоматично оновлює вхідні значення ЦАП значеннями регістрів DACH і DACL.

Це означає, що якою б не була конфігурація біта DAATE, зміна регістра DACL не впливає на вихід ЦАП, доки регістр DACH також не буде оновлено. Отже, щоб працювати з 10 бітами,

DACL має бути записаний спочатку перед DACH. Щоб працювати з 8-розрядною конфігурацією, запис DACH дозволяє оновлювати ЦАП.

Регістр DAC.

						DAC0	DAC9
14	14	13	12	11	10	9	8
DAC7	DAC6	DAC5	DAC4	DAC3	DAC2	DAC1	DAC0
7	6	5	4	3	2	1	0

Питання.

1. Призначення і структурна схема компаратора.
2. Регістри компаратора МК.
3. Схеми подання сигналів на входи компаратора.
4. Призначення і принцип роботи ЦАП.
5. Перетворення ЦАП в однополярному і біполярному режимах.
6. Перетворення послідовних кодів у вихідну напругу в ЦАП.
7. Генератор сигналів довільної форми на ЦАП
8. Обчислення кодів вибірок сигналів в ЦАП.
9. Атенюатор на ЦАП.
10. Керований підсилювач вхідного сигналу на ЦАП.
11. Схеми зсуву аналогового сигналу на ЦАП.
12. Регістри ЦАП.

ЛЕКЦІЯ 15. Електродвигуни. Крокові двигуни

Мета. Вивчення принципів роботи електродвигунів постійного і змінного струму, сервоприводів і крокових двигунів.

Вступ. Електродвигуни постійного і змінного струму, сервоприводи і крокові двигуни є основою сучасної техніки так як забезпечують перетворення електричної енергії у механічну.

Для керування роботою сучасних електродвигунів широко використовуються як аналогові так і цифрові засоби на основі мікроконтролерів.

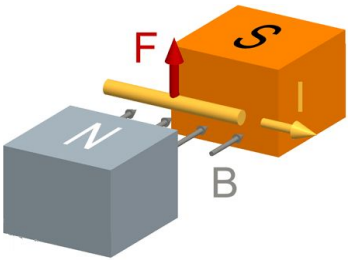
План.

1. Принцип роботи електродвигунів.
1.1. Типи двигунів.
2. Електродвигуни постійного струму.
2.1. Керування безколекторним двигуном.
3. Двигуни змінного струму.
4. Сервопривід і принцип роботи.
4.1. Схема керування сервоприводом.
4.2. Характеристики сервоприводів.
5. Крокові двигуни.

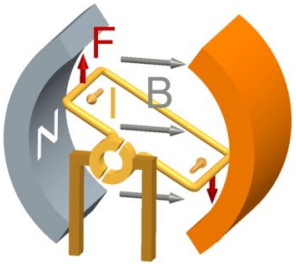
1. Принцип роботи електродвигунів

Робота електродвигунів основана на наступних принципах:

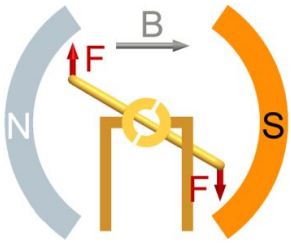
1. Відповідно до закону Ампера на провідник зі струмом *I* в магнітному полі *B* діє сила *F*.



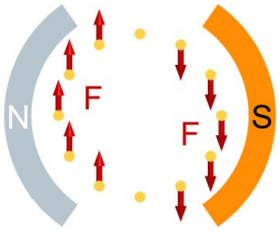
2. Якщо провідник зі струмом *I* зігнути в рамку і помістити в магнітне поле, то дві сторони рамки, що знаходяться під прямим кутом до магнітного поля *B*, будуть відчувати протилежно спрямовані сили *F*.



3. Сили, що діють на рамку, створюють крутний момент або момент сили, що обертає її.



4. Електродвигуни мають декілька витків на якорі, щоб забезпечити більший постійний крутний момент.



5. Магнітне поле може створюватися як постійними магнітами, так і електромагнітами. Електромагніт звичайно є дротом намотаним на сердечник. Таким чином, згідно із законом електромагнітної індукції, струм який протікає у рамці якора буде індукувати струм в обмотці електромагніту індуктора, який в свою чергу буде створювати магнітне поле.

1.1. Типи електродвигунів

Всі електродвигуни можна поділити на два види:

- електродвигуни постійного струму;
- електродвигуни змінного струму (асинхронні та синхронні).

В основу роботи будь-яких електродвигунів покладено принцип електромагнітної індукції.

Електродвигун постійного струму складається з нерухомої частини – індуктора і рухомої частини – якоря. Як індуктор у малопотужних двигунах постійного струму нерідко використовуються постійні магніти.

Електродвигун змінного струму складається з нерухомої частини - статора і рухомої частини – ротора.

2. Електродвигуни постійного струму

Електродвигун постійного струму іноді називають синхронним двигуном із самосинхронізацією. Електродвигун постійного струму складається з постійного магніту на індукторі (статорі), електромагніту з декількома обмотками, щітко-колекторного вузла з двома пластинами (ламелями) і двома щітками. Простий двигун має два положення ротора (2 "мертві точки"), з яких неможливий самозапуск, і нерівномірний крутний момент. У першому наближенні магнітне поле полюсів статора рівномірне (однорідне). Схема електродвигуна постійного струму показана на рис. 15.1.

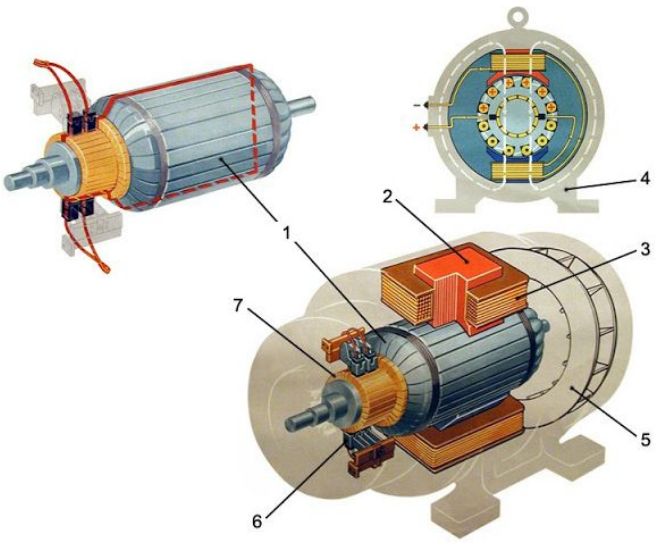


Рисунок 15.1 – Електродвигун постійного струму: 1 – якір, 2 – сердечник полюса, 3 – обмотка полюса, 4 – статор, 5 – вентилятор, 6 – щітки, 7 - колектор

Електродвигуни постійного струму з щітково-колекторним вузлом можуть бути:

- колекторним, в якому давачем положення ротора і перемикачем струму в обмотках є один і той ж пристрій – щітково-колекторний вузол, рис. 15.2;

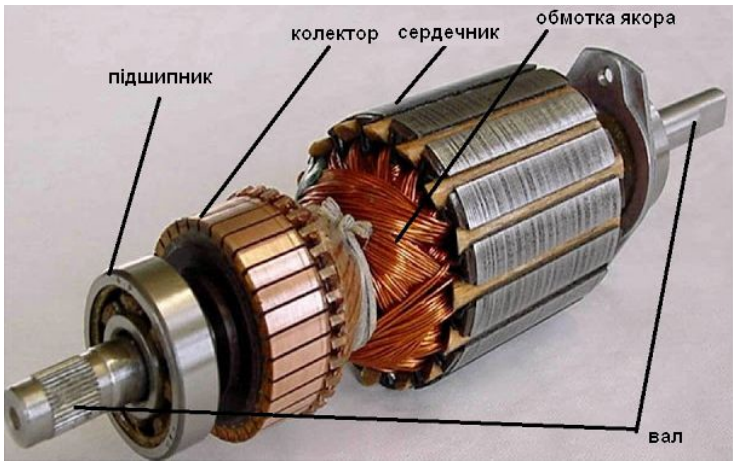


Рисунок 15.2 – Колекторний двигун постійного струму

- безколекторним, із замкнутою електромеханічною системою, яка складається із статора з котушками для створення магнітного поля, якора з постійними магнітами розміщеними у певному порядку, давача положення ротора, перетворювача координат і підсилювача потужності, рис. 15.3.

Рисунок 15.3 – Безколекторний електродвигун постійного струму з внутрішнім якорем

Електромагнітний механізм може бути із зовнішнім якорем, рис. 15.4.

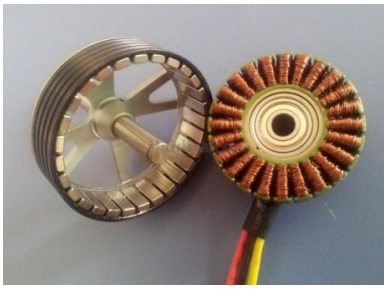


Рисунок 15.4 – Безколекторний електродвигун постійного струму із зовнішнім якорем

Електродвигуни з внутрішнім якорем мають високу швидкість обертання тому застосовуються у безпілотних літальних апаратах, а з зовнішнім якорем – мають точне позиціювання і використовуються у станках з числовим програмним керуванням.

Для роботи безколекторного двигуна потрібний спеціальний контролер, який включає обмотки таким чином, що б вектори магнітних полів статора і якора були ортогональні один до одного. По суті, драйвер регулює крутний момент, який діє на якір, рис. 15.5.

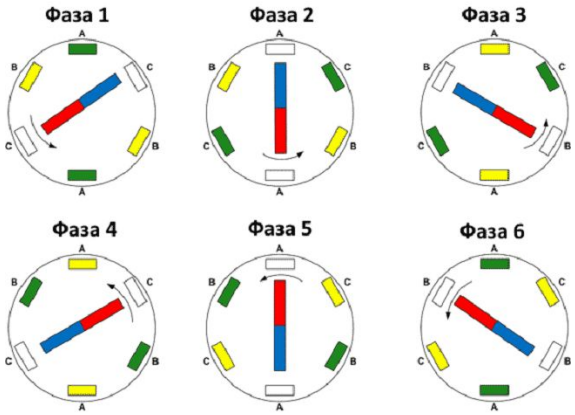


Рисунок 15.5 – Фази роботи безколекторного електродвигуна

Без контролера керування безколекторний електродвигун не буде працювати. З цієї причини такий електродвигун не може бути генератором.

2.1. Керування безколекторним двигуном

Комутацію обмоток безколекторного електродвигуна виконує електроніка. Драйвер відслідковує положення якора за допомогою давачів Хола або використовується зворотна електрорушійна сила (ЕРС), яка виникає у непідключених котушках статора. Контролер є апаратно-програмним комплексом, який відслідковує ці зміни і задає порядок комутації.

Більшість безколекторних двигунів виконана у трифазному виконанні. Для керування таким приводом у контролері є перетворювач постійної напруги у трифазну імпульсну, рис. 15.6.

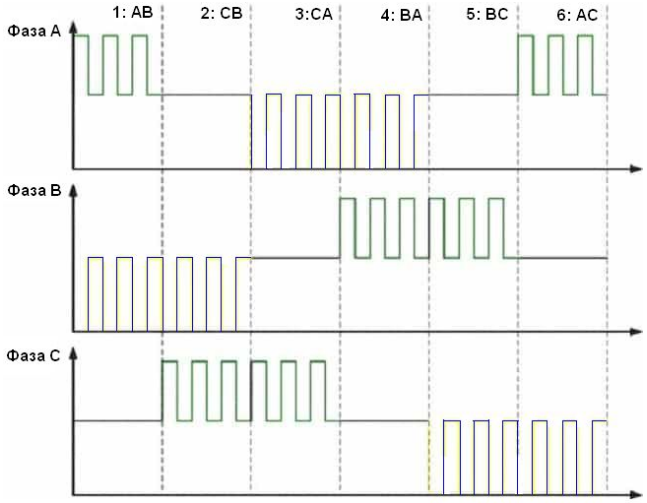


Рисунок 15.6 – Фази і етапи роботи приводу безколекторного електродвигуна: зелений колір – позитивні імпульси, синій колір – негативні імпульси

Для пояснення роботи приводу використовуються рис. 15.4 і рис. 15.6.

1 – на котушку А подається позитивний імпульс, а на В – негативний, в результаті якір зсунеться.

2 – котушка А відключається і позитивний імпульс подається на С (В залишається без змін), далі подається сигнал для наступного набору імпульсів.

3 – на С подається позитивний імпульс, а на А – негативний.

4 – працює пара В і А, на які подається позитивний і негативний сигнал.

5 – позитивний імпульс подається на В, а на С – негативний.

6 – включається котушка А (подається позитивний імпульс) і повторяється негативний імпульс на С. Далі цикл повторюється.

У цій ніби то простій схемі, дійсно є багато складностей. Потрібно не тільки відслідковувати положення якора, щоб сформувати наступну серію імпульсів, але і керувати частотою обертання, регулюючи струм у котушках. Крім цього потрібно вибрати оптимальні параметри для розгону і гальмування. Контролер також має мати блок керування його роботою. Тому пристрій є достатньо складним, рис. 15.7.

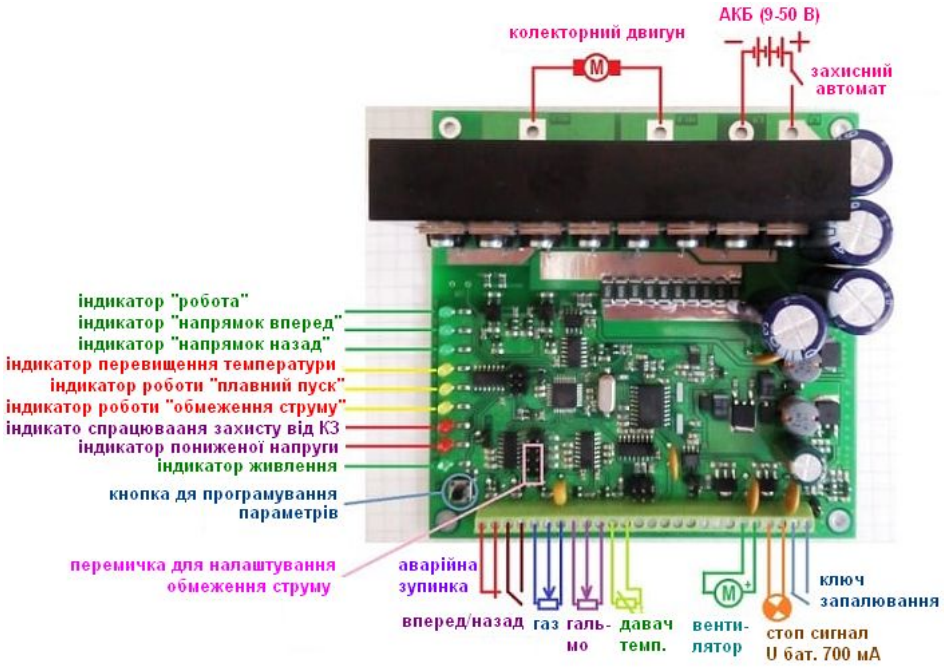


Рисунок 15.7 – Багатофункціональний контролер керування безколекторним електромотором

3. Двигуни змінного струму

За типом роботи дані двигуни бувають синхронні і асинхронні. Принципова відмінність полягає в тому, що в синхронних двигунах перша гармоніка магніторушійної сили статора переміщається зі швидкістю обертання ротора (з цієї причини ротор також крутиться зі швидкістю обертання магнітного поля в статорі), а у асинхронних двигунів – зберігається постійна різниця між швидкістю обертання ротора і швидкістю обертання магнітного поля в статорі (поле крутиться швидше ротора).

Синхронний двигун змінного струму – ротор крутиться синхронно з магнітним полем напруги живлення. Такі двигуни традиційно застосовуються при величезних потужностях (від сотень кіловат і вище), рис. 15.8.

Рисунок 15.8 – потужний синхронний двигун

Є синхронні двигуни з дискретним кутовим рухом ротора – крокові двигуни. У крокових двигунах положення ротора фіксується подачею живлення на відповідні обмотки. Перехід в інше положення досягається шляхом зняття напруги живлення з одних обмоток і подачі його на інші обмотки. Ще один вид синхронних двигунів – вентильний реактивний електродвигун, живлення обмоток якого забезпечується напівпровідниковими елементами.

Асинхронний двигун змінного струму – частота обертання ротора відрізняється від частоти крутного магнітного поля. Інша назва асинхронних двигунів – індукційні, зумовлена тим, що струм в обмотці ротора індукується обертовим полем статора. Асинхронні двигуни становлять зараз більшу частину електродвигунів. В основному вони використовуються як електродвигуни для перетворення електричної енергії в механічну. Причому, в основному використовуються асинхронні двигуни з короткозамкненим ротором. За кількістю фаз асинхронні двигуни бувають одно-, дво- і трифазні.

Найбільше поширення асинхронні двигуни отримали у виробництві і побуті.

Однофазний асинхронний двигун з короткозамкненим ротором має на статорі тільки одну робочу обмотку, на яку під час роботи мотора подається змінний струм, рис. 15.9. Для запуску такого електромотора на його статорі є допоміжна пускова обмотка, яка короткочасно підключається до мережі через конденсатор або індуктивність, або замикається на коротко пусковими контактами рубильника. Це потрібно для створення початкового зсуву фаз, щоб ротор почав крутитися, інакше пульсуюче магнітне поле статора не зсуне ротор з місця.

Рисунок 15.9 – Однофазний асинхронний двигун з короткозамкненим ротором

Ротор такого мотора, як і будь-якого іншого асинхронного мотора з короткозамкненим ротором, являє собою циліндричний сердечник з залитими алюмінієм пазами і відразу відлитими вентиляційними лопатями. Такий ротор називається короткозамкненим. Однофазовий двигуни використовуються в малопотужних пристроях, в таких як кімнатні вентилятори або невеликі насоси.

Двофазний асинхронний двигун з короткозамкненим ротором більш ефективний при роботі від однофазної мережі змінного струму. Він містить на статорі дві робочі обмотки, які розмішуються перпендикулярно, при цьому одна з обмоток підключається до мережі змінного струму безпосередньо, а друга – через фазо-зсувний конденсатор, тому виходить, що крутиться магнітне поле, рис. 15.10. Такий електромотор без конденсатора не запуститься. Також електродвигун має короткозамкнений ротор. Двофазні електродвигуни, які живляться від однофазних мереж називають конденсаторними двигунами, тому що фазозсувний конденсатор є обов'язковою їх частиною.

Двофазні електродвигуни набагато ширше використовуються, ніж однофазні, наприклад, пральні машини і різні верстати.

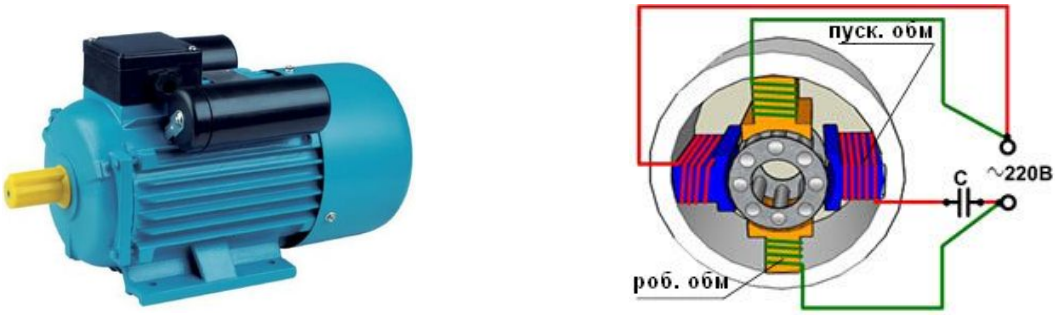


Рисунок 15.10 – Двофазний асинхронний двигун з короткозамкненим ротором

Трифазний асинхронний двигун з короткозамкненим ротором має на статорі три робочі обмотки, які зсунуті одна відносно одної так, що при підключенні в трифазну мережу, їх магнітні поля виходять зміщеними в просторі одне відносно одного на 120 °, рис. 15.11. При включенні трифазного мотора до трифазної мережі змінного струму, з'являється крутне магнітне поле, що приводить в рух короткозамкнений ротор.

Обмотки статора трифазного електромотора можна з’єднати за схемою «зірка» або «трикутник». При живленні мотору за схемою «зірка» потрібна вища напруга, ніж для схеми «трикутник», тому на двигуні вказуються дві напруги, наприклад: 127/220 або 220/380. Трифазні двигуни використовуються у промисловості для приведення в дію різних верстатів, лебідок, циркулярних пил, підйомних кранів і т.п.

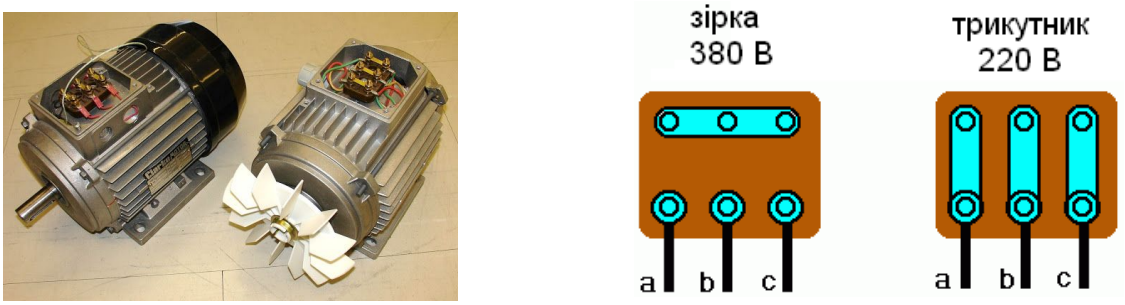


Рисунок 15.11 – Трифазний асинхронний двигун з короткозамкненим ротором

Трифазний асинхронний двигун з фазним ротором має статор подібний описаним вище типам двигунів, шихтований магнітопровід з трьома закладеними в його пази обмотками, але в фазний ротор не залиті алюмінієві стержні, а вкладена справжня трифазна обмотка, з'єднана як «зірка», рис. 15.12. Кінці зірок обмотки фазного ротора виведені на три контактних кільця, насаджених на вал ротора, і електрично відокремлених від нього.

За допомогою щіток на кільця подається трифазна змінна напруга, а включення може бути безпосереднє або через реостати. Двигуни з фазним ротором коштують дорожче, але їх пусковий момент під навантаженням значно вищий, ніж у інших типів електродвигунів з короткозамкненим ротором. Саме в наслідок підвищеної сили і величезного пускового моменту, даний вид двигунів використовується у приводах ліфтів і підйомних кранів, тобто там де двигун запускається під навантаженням а не в холосту, як у двигунів з короткозамкненим ротором.

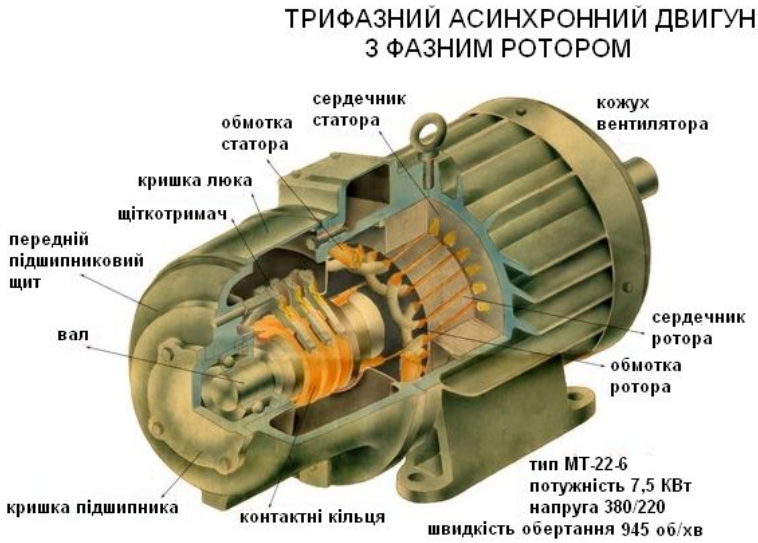


Рисунок 15.12 – Трифазний асинхронний двигун з фазним ротором

4. Сервопривід і принцип роботи

Сервопривід (або слідкуючий привід) – це привід з керуванням через негативний зворотний зв’язок, що дозволяє точно керувати параметрами руху. Сучасні сервоприводи складаються із таких частин: двигун постійного струму, редуктор, вихідний вал, потенціометр і електронний блок керування, рис. 15.13.

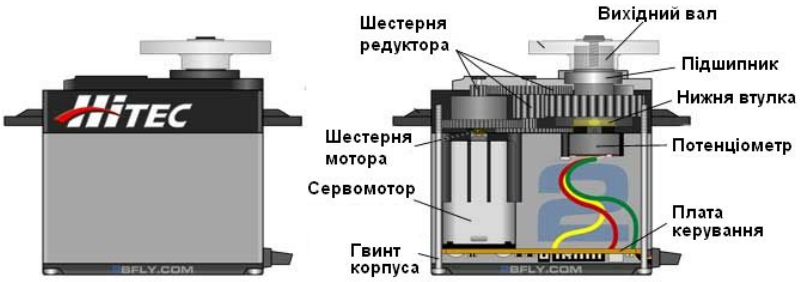


Рисунок 15.13 – Сервопривід

Принцип роботи сервоприводу:

1. Сервопривід отримує імпульсний сигнал, керуюче значення якого визначає кут повороту вихідного валу.

2. Блок керування (контролер) порівнює значення поступившого сигналу із значеннями давача поворотів вихідного валу.

3. В залежності від результату порівняння блок керування повертає сигнал, який визначає дії, які має виконати електродвигун – повернутися, прискоритися, сповільнитися.

Більшість сервоприводів використовують три дроти для роботи. Дріт для живлення з напругою 4,8 В або 6 В, загальний дріт (земля) і сигнальний дріт. Керуючий сигнал передає інформацію про необхідне положення вихідного валу. Вал з'єднаний з потенціометром, який є давачем його положення. Блок керування за опором потенціометра і значенням керуючого сигналу визначає, в який бік потрібно обертати мотор, щоб отримати потрібне положення

вихідного валу. Чим вища напруга живлення сервоприводу, тим швидше він працює і більший момент розвиває.

Керуючий сигнал є імпульсом змінної ширини. Імпульси повторюються з постійною частотою (як правило, з частотою 50 Гц). Положення сервоприводу визначається шириною імпульсу. Для типового сервоприводу, який використовується в радіокерованих моделях, тривалість імпульсу в 1500 мкс означає, що сервопривід повинен зайняти середнє положення. Збільшення або зменшення довжини імпульсу змусить сервопривід повернутися за годинниковою або проти годинникової стрілки, відповідно.

Таким чином, для керування сервоприводом можна використовувати ШІМ сигнал з частотою 50 Гц. При цьому для положення "0" тривалість імпульсу повинна становити 1000 мкс, для середнього положення – 1500 мкс, а для положення "максимум" – 2000 мкс мікросекунд, рис. 15.14.

Рисунок 15.14 – Керуючі сигнали сервоприводу

4.1. Схема керування сервоприводом

Схема керування сервоприводом показана на рис. 15. Сервопривід отримує імпульсні сигнали частотою 50 Гц (T=20 мс) і тривалістю 0,9; 1,5; 2,0 мс. Імпульсні сигнали поступають на компаратор і одночасно активують генератор опорного імпульсу. Тривалість опорного імпульсу відображає положення потенціометра, який фізично зв’язаний з вихідним валом редуктора. Керуючий сигнал і опорний імпульс порівнюються компаратором, де аналізується різниця їх тривалостей. Отримане значення зберігається в пристрої вибірки-зберігання і використовується для регулювання режиму роботи електродвигуна.

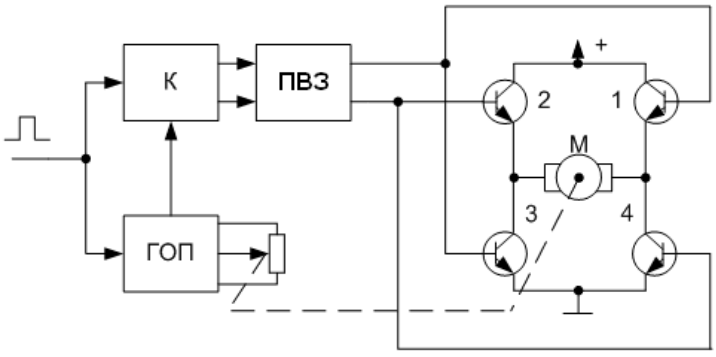


Рисунок 15.15 – схема керування сервоприводом: ГОП – генератор опорного імпульсу, К – компаратор, ПВЗ – пристрій вибірки-зберігання, М – електричний мотор, включений у діагональ силового моста

4.2. Характеристики сервоприводів

Розмір і вага. Розміри бувають: “мікро”, “міні”, “стандартний” і “гігант”. У межах кожного класу розміри можуть трохи змінюватися. Середні розміри сервоприводів:

Мікро: 24мм х 12мм х 24мм, вага 5-10 г.

Міни: 30мм х 15мм х 35мм, вага 25 г.

Стандарт: 40мм х 20мм х 37мм, вага 50-60 г.

Гігант: 49х25х40 мм, вага 50-90 г.

Рисунок 15.16 – Зовнішній вигляд сервоприводу

Швидкість. Швидкість сервоприводів вимірюється часом повороту важеля на валу сервоприводу на кут 60 градусів при напрузі живлення 4,8 В і 6 В. Наприклад, сервопривід з параметром 0,22с / 60 ° при 4,8 В повертає вал на 60 градусів за 0,22 с при напрузі живлення 4,8 В. Найбільш швидкі сервоприводи мають час повертання від 0,06 до 0,09 с.

Кут повороту. Це максимальний кут, на який може повернутися вихідний вал. Сервоприводи мають кут повороту 180° або 360°.

Момент на валу. Момент сервоприводу вимірюється вагою вантажу в кг, який сервопривід може утримувати нерухомо на важелі з плечем 1 см. Вказують дві цифри, для напруги живлення 4,8 В і 6 В. Наприклад, якщо вказано, що сервопривід розвиває 10 кг/см, то це значить, що на важелі довжиною 1см сервопривід може розвинути зусилля 10 кг, перше ніж зупиниться. Для важеля довжиною 2 см такий сервопривід зможе розвинути зусилля 5кг, а для 5 мм – 20кг.

Цифрові і аналогові сервоприводи.

Цифрові і аналогові сервоприводи механічно не відрізняються один від одного. У них ті ж корпуси, мотори, шестерні і потенціометри. Керуючий сигнал для аналогових і цифрових сервоприводів однаковий, але вони відрізняються електронним блоком керування мотором.

А аналогових сервопроводах використовуються керуючі імпульси з частотою 50 Гц. В стані спокою на мотор не подається напруга і у випадку невеликого відхилення від рівноваги на мотор падається короткий сигнал малої потужності. Тому утворюються “мертві зони” по часу і віддалі.

Цю проблему розв’язують цифрові сервопроводи в яких використовується мікроконтролер, який приймає керуючі імпульси, обробляє їх і посилає сигнали на мотор з частотою 200 Гц, рис. 15.17. Тому цифровий сервопривід швидше реагує на зовнішні впливи і швидше досягає необхідну швидкість і крутний момент.

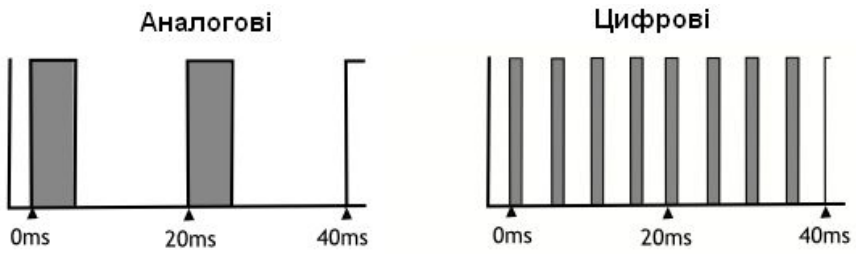


Рисунок 15.17 – Частота сигналів керування електромотором

Тип редуктора. Редуктори бувають з металу, карбону або пластмаси. Пластмасові (нейлонові) шестерні слабо витримують навантаження і удари, але зносостійкі. Карбонові міцніші пластмасових, але набагато дорожчі. Металічні витримають великі навантаження, удари, падіння, але мають найменшу зносостійкість, рис. 15.18

Рисунок 15.18 – Типи редукторів

5. Крокові двигуни

Кроковий двигун — це електродвигун, який живиться імпульсним струмом, що дозволяє повертати ротор на заданий кут.

Кроковий двигун складається із статора і ротора. В склад статора входять обмотки (котушки), а на роторі встановлені постійні магніти. Обмотки, наприклад 4, розміщуються на статорі під кутом 90° одна до одної. Струм на обмотки подається по колу – на одну за одною. Напрямок обертання валу визначається порядком подачі струму на обмотки. Струм через обмотки протікає з інтервалом в 1 секунду. Вал двигуна повертається на 90 °, кожний раз, коли через обмотку протікає струм.

Можливі різні способи подачі струму на обмотки і в результаті різні режими керування кроковим двигуном:

- повнокрокове керування однією обмоткою;
- повнокрокове керування двома обмотками;
- півкроковий режим однообмоточний;
- півкроковий режим двообмоточний;
- режим мікрокроку.

Є різні типи крокових двигунів:

- із постійним магнітом;

- із змінним магнітним опором;
- гібридний.

Повнокрокове керування однією обмоткою. При такому керуванні струм протікає тільки через одну обмотку. Такий метод дозволяє отримати менше половини крутного моменту і мотор матиме 4 кроки, рис. 15.19.

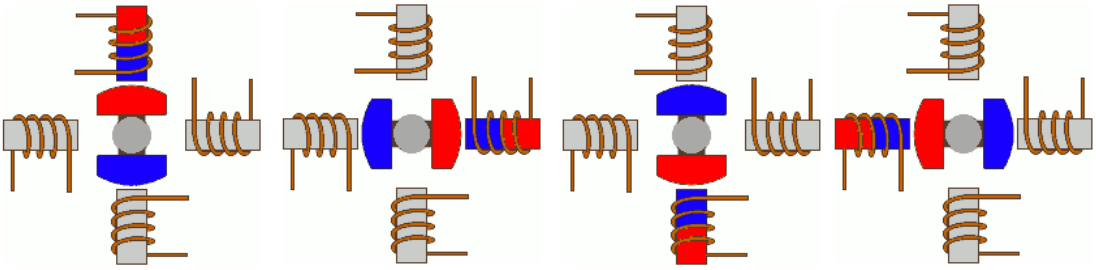


Рисунок 15.19 – Повнокрокове керування однією обмоткою

Повнокрокове керування двома обмотками. При такому керуванні струм на обмотки подається попарно, рис. 15.20. В залежності від підключення обмоток (послідовне чи паралельне), мотору потрібна подвійна напруга або подвійний струм, порівняно із збудженням однієї обмотки. У цьому випадку мотор буде видавати 100% номінального крутного моменту. Такий мотор має 4 кроки на повний оберт.

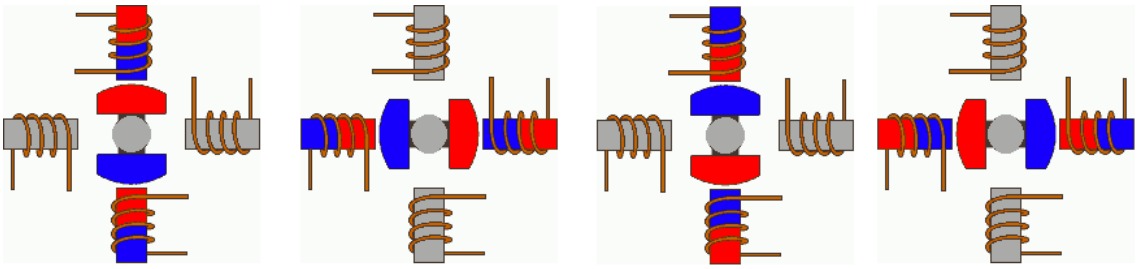
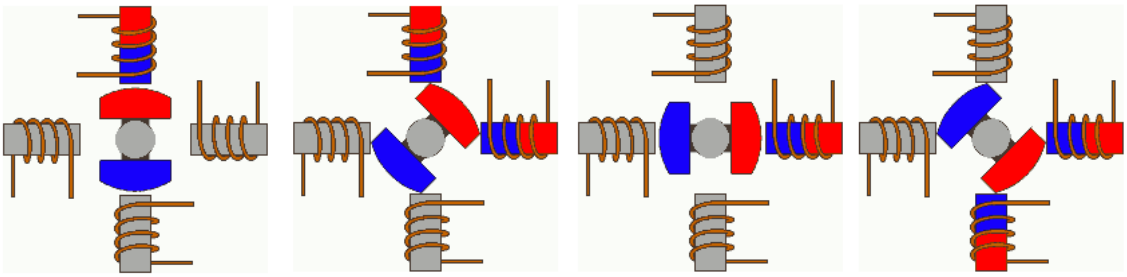
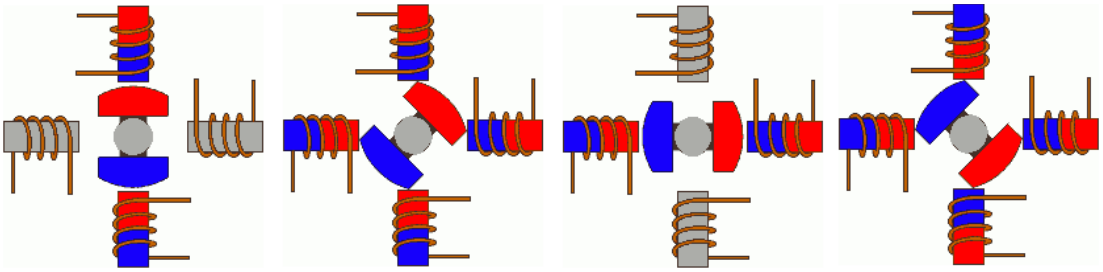


Рисунок 15.20 – Повнокрокове керування двома обмотками

Півкроковий режим керування. Режим дозволяє отримати подвоєну точність системи позиціювання. Режим може бути реалізований з використанням одної або двох обмоток.



а) однообмоточний режим



б) двообмоточний режим

Рисунок 15.21 – Півкроковий режим керування: а – однообмоточний; б – двообмоточний

Режим мікрокроку. Такий режим є найбільш поширеним способом керування крковими двигунами. При цьому на обмотки подаються не імпульси, а сигнали, які по формі подібні до синусоїди. Також можуть використовуватися форми цифрових сигналів, рис. 15.22. В такому режимі крковий двигун може працювати плавно як і двигун постійного струму.

Рисунок 15.22 – Форми сигналу обмоток при мікрокрковому керуванні

Метод мікрокроку в дійсності є способом живлення мотору, а не керування обмотками, тому він може застосовуватися у повнокрокових режимах керування. Схема роботи режиму мікрокроку показана на рис. 15.23.

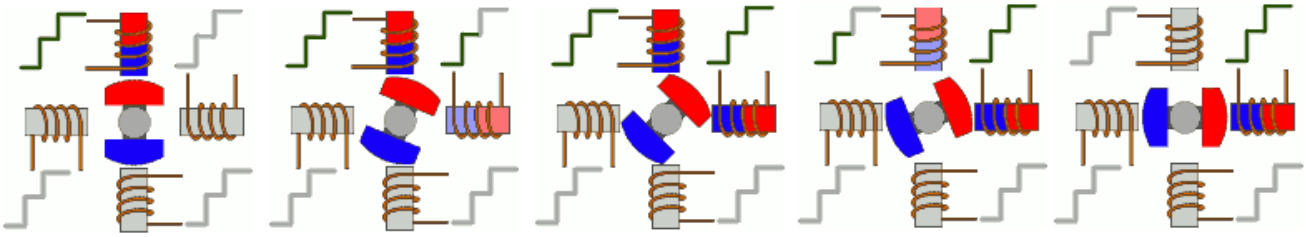


Рисунок 15.23 – Режим мікрокроку

4.1. Типи крокових двигунів

Крковий двигун з постійними магнітами. Ротор такого мотора має постійний магніт у формі диску з двома або більшою кількістю полюсів, рис. 15.24. Обмотки статора будуть притягувати або відштовхувати постійний магніт на роторі, створюючи крутний момент. Звичайно, величина кроку таких двигунів знаходиться в діапазоні 45-90 °.

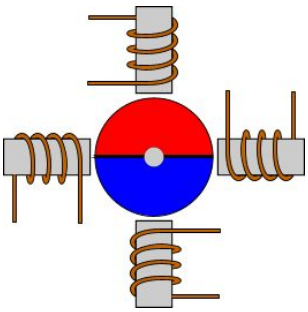


Рисунок 15.24 – Крокoвий двигун з постійними магнітами

Крокoвий двигун із змінним магнітним опором. Такі двигуни на роторі не мають постійного магніту. Замість цього ротор виготовляється з магнітом'якого металу у виді зубчастого диску. Статор має більше 4-х обмоток, які живляться в протилежних парах і притягують ротор, рис. 15.25. Відсутність постійного магніту значно знижує крутний момент. Зате в цих двигунах відсутній гальмівний момент. Гальмівний момент створюється постійними магнітами ротора, які притягуються до арматури статора при відсутності струму в обмотках. Крокові двигуни із змінним магнітним опором мають крок в межах 5-15 °.

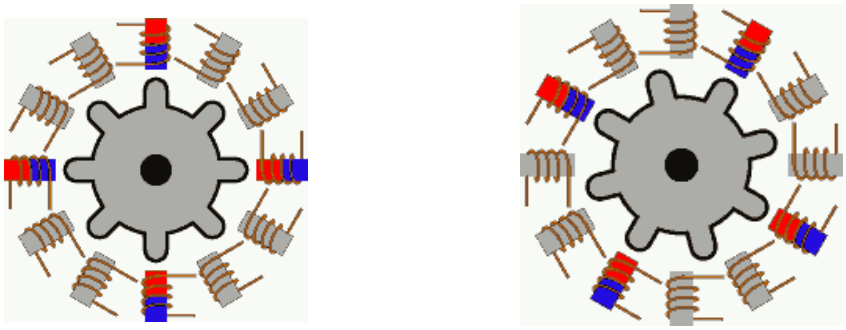


Рисунок 15.25 – Крокoвий двигун із змінним магнітним опором

Гібридний крокoвий двигун. Об'єднує в собі характеристики крокових двигунів з постійними магнітами і із змінним магнітним опором. Він має відмінні утримуючі і динамічні крутні моменти, а також малу величину кроку в межах 0,9-5°. Цей тип двигунів використовується у високоточних станках з числовим програмним керування і в роботах. Головний їх недолік – висока вартість.

Звичайний мотор з 200 кроками на один оберт має 50 позитивних і 50 негативних полюсів з 8-ма обмотками. У зв'язку з тим, що неможливо виготовити такий зубчастий магніт, два 50-зубі диски були приварені з обох сторін циліндричного постійного магніту. Таким чином один диск має позитивний полюс на своїх зубах, а інший – негативний полюс. Диски зміщені таким чином, що виступи зубів одного диску співпадають з впадинами зубів іншого диску. Таким чином ніби створюється один 100-зубий диск, рис. 15.26.

Рисунок 15.26 – Ротор з двома 50-зубими дисками

Приклад гібридного крокового двигуну з 75 кроками на один оберт. Шість обмоток спарені. Друга пара обмоток зміщена відносно першої (вертикальна пара) на кут 60+5 °, а третя відносно другої — також на 60+5 °. Кутова різниця і є причиною обертання ротора.

Послідовність зміни кроків гібридного крокового двигуна показана на рис. 15.27.

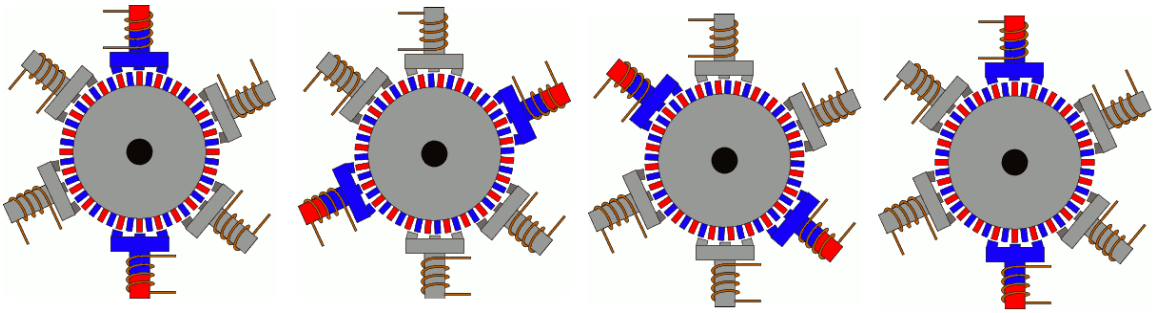


Рисунок 15.27 – Робота гібридного крокового двигуна

Питання.

1. Принципи роботи електродвигунів.
2. Електродвигун постійного струму.
3. Безколекторний двигун постійного струму і керування його роботою.
4. Двигуни змінного струму.
5. Синхронний і асинхронний двигун змінного струму.
6. Однофазний асинхронний двигун з короткозамкненим ротором.
7. Двофазний асинхронний двигун з короткозамкненим ротором.
8. Двофазний асинхронний двигун з короткозамкненим ротором.
9. Сервопривід, принцип роботи і схема керування.
10. Крокові двигуни. Повнокрокове керування однією і двома обмотками.
11. Крокові двигуни. Півкроковий і мікрокроковий режим керування.
12. Кроковий двигун із змінним магнітним опором. Гібридний кроковий двигун.

СПИСОК ЛІТЕРАТУРИ

1. Новацький А. О. Мікропроцесорні та мікроконтролерні системи. Частина 2. Проектування мікропроцесорних систем. Підручник. / Новацький Анатолій Олександрович. – НТУУ «Київський політехнічний інститут імені Ігоря Сікорського» – К.: Видавництво «Політехніка», 2021. – 462 с.

2. Смірнов В. В. Програмування мікроконтролерних систем : навч. посіб. / В. В. Смірнов, Н. В. Смірнова, Ю. М. Пархоменко. – Кропивницький : Центральноукр. НТУ, 2021. – 262 с.

3. Грищук Ю. С. Мікроконтролери: Архітектура, програмування та застосування в електромеханіці : навч. посіб. / Ю. С. Грищук. – Харків : НТУ «ХПІ», 2019. – 384 с.

4. Програмування мікроконтролерів AVR : [навчальний посібник] / С. М. Цирульник, О. Д. Азаров, Л. В. Крупельницький, Т. І. Трояновська. – Вінниця : ВНТУ, 2018. – 111 с.

5. Програмування мікроконтролерів систем автоматики : конспект лекцій для студентів базового напряму 050201 “Системна інженерія” / Укл.: А. Г. Павельчак, В. В. Самотий, Ю.В. Яцук – Львів : Львівська політехніка. – 2012. – 143 с.

6. Трамперт В. Измерение и регулирование с помощью AVR-микроконтроллеров.: Пер. с нем. – К.: “МК-Пресс”, 2007. – 208 с.

7. Трамперт Вольфганг. AVR-RISC микроконтроллеры. – Пер. с нем. – К.: “МК-Пресс”, 2006. – 464 с.

8. Richard Barnett, Larry O’Cull, Sarah Cox. Embedded C programming and the Atmel AVR. – NY: Delmar, Cengage Learning, 2006. – 532 p.

9. Сайт фірми Microchip: [Електронний ресурс] / Microcontrollers & Microprocessors. – Режим доступу: https://www.microchip.com, вільний. – Загол. з екрану. – Мова анг.