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

МІКРОПРОЦЕСОРИ ТА МІКРОКОНТРОЛЕРИ

КУРС ЛЕКЦІЙ

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для студентів,
які навчаються за спеціальності 153 «Мікро- та наносистемна техніка»,
освітньою програмою «Мікро- та наноелектроніка»

Київ
КПІ ім. Ігоря Сікорського
2020

Мікропроцесори та мікроконтролери: Курс лекцій [Електронний ресурс] : навч. посіб. для студ. спеціальності 153 «Мікро- та наносистемна техніка», освітньої програми «Мікро- та наноелектроніка» / КПІ ім. Ігоря Сікорського ; уклад.: Д. Д. Татарчук, Ю. В. Діденко. – Електронні текстові дані (1 файл: 19,5 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2020. – 238 с.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 3 від 05.11.2020 р.) за поданням Вченої ради факультету електроніки (протокол № 10/2020 від 19.10.2020 р.)

Електронне мережне навчальне видання

МІКРОПРОЦЕСОРИ ТА МІКРОКОНТРОЛЕРИ
КУРС ЛЕКЦІЙ

Укладачі: *Татарчук Дмитро Дмитрович*, канд. техн. наук, доц.
Діденко Юрій Вікторович, канд. техн. наук

Відповідальний редактор *Волхова Т. Л.*, доцент, канд. техн. наук, доц.

Рецензент: *Мельник І. В.*, проф., д-р техн. наук, проф.

Розглянуто базові принципи створення обчислювальних систем на основі мікропроцесорів та мікроконтролерів, класифікацію мікроконтролерів, описано сучасні дротові та бездротові інтерфейси. Детально розглянуто сімейство мікроконтролерів STM32: їх будову та принципи роботи з ними.

Посібник призначений для здобувачів ступеня бакалавра за освітньою програмою «Мікро- та наноелектроніка» спеціальності 153 «Мікро- та наносистемна техніка». Також може бути корисним студентам, аспірантам, викладачам для ознайомлення із сучасними тенденціями розвитку мікропроцесорної техніки.

ЗМІСТ

СПИСОК СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

ACK –	acknowledgment
ADC –	Analog-to-digital converter
ALU –	Arithmetic and logic unit
ARM –	Advanced RISC Machine
CAN –	Controller Area Network
CISC –	Complex Instruction Set Computing
CR –	control register
CRC –	Cyclic redundancy check
CTS –	Request to Send
DFS –	Dynamic Frequency Selection
DRAM –	Dynamic RAM
DSSS –	Direct Sequence Spread Spectrum
EEPROM –	Electrically Erasable Programmable Read-Only Memory
EPROM –	Erasable Programmable Read Only Memory
ETM –	external trace macrocell
FHSS –	Frequency Hopping Spread Spectrum
FSK –	Frequency Shift Keying
ISO –	International Organization for Standardization
MAC –	media access control (medium access control)
NOP –	No operation (немає операції)
NVIC –	Nested vector interrupt controller
OSI –	Open System Interconnection
PC –	program counter
PIC –	Programmable Interrupt Controller
PPI –	Parallel Peripheral Interface
PROM –	Programmable Read-Only Memory
PSK –	Phase Shift Keying

PSR –	Program Status Register
PVD –	Power Voltage Detector
PWM	pulse-width modulation
RAM –	Random Access Memory
RISC –	Reduced Instruction Set Computing
ROM –	Read Only Memory
RS-232 –	Recommended Standard 232
RTS –	Request to Send
SPI –	Serial Peripheral Interface
SR –	state register
SRAM –	Static Random Access Memory
TCP –	Transmit Power Control
UART -	Universal Asynchronous Receiver-Transmitter
USART –	Universal Synchronous/Asynchronous Receiver/Transmitter
Wi-Fi –	Wireless Fidelity
WLAN –	Wireless Local Area Network
WMAN –	Wireless Metropolitan Area Network
WPAN –	Wireless Personal Area Network
WWAN –	Wireless Wide Area Network
АЛП –	арифметично-логічний пристрій
АЦП –	аналогово-цифровий перетворювач
ВІС –	велика інтегральна схема
ВКМ –	внутрішня контрольна магістраль
ЗЕ –	запам’ятовуючий елемент
ЗП –	запам’ятовуючий пристрій
КМОН –	комплементарна МОН структура
КП –	керуючий пристрій
КР –	керуючий регістр
КВВП –	контролер векторизованих вкладених переривань

МК –	мікроконтролер
МОН –	метал-оксид-напівпровідник
МП –	мікропроцесор
МПС –	мікропроцесорна система
ОЗП –	оперативний запам'ятовуючий пристрій
ОП –	операційний пристрій
ОСРЧ	операційна система реального часу
ПВВ –	пристрої для введення-виведення інформації
ПЗП –	постійний запам'ятовуючий пристрій
ПЛ –	програмний лічильник
ППІ –	паралельний периферійний інтерфейс
РДП –	регістр дозволу переривань
РЗП –	регістри загального призначення
РП –	регістровий пристрій
РПП –	регістр пріоритету переривань
РСП –	регістр стану процесора
РФЗ –	регістр фіксації запитів переривань
ТГ –	тактовий генератор
ЦАП –	цифрово-аналоговий перетворювач
ЦОС –	цифрова обробка сигналів
ЦП –	центральний процесор
ША –	шина адреси
ШД –	шина даних
ШІМ –	широтно-імпульсна модуляція
ШК –	шина керування

ВСТУП

В результаті науково-технічного прогресу в техніці все ширше використовуються системи на основі мікропроцесорів. Такі системи характеризуються функціональною завершеністю, високою технологічністю виробництва і великим ступенем адаптивності. Часто для адаптування системи до нових задач достатньо лише змінити програму мікропроцесора. Високий ступінь адаптивності дозволяє використовувати такі системи для виконання різноманітних задач по обробці інформації.

Широке використання таких систем потребує підготовки великої кількості спеціалістів здатних до їх розробки та експлуатації. Тому метою даного посібника є викладення основ мікропроцесорної техніки необхідних для розробки та експлуатації мікропроцесорних систем.

В посібнику розглядаються мікропроцесори та мікроконтролери. Наводиться класифікація мікропроцесорів, розглядаються відмінності між різними класами мікропроцесорів та особливості роботи з ними. Даний матеріал призначений для підготовки бакалаврів за спеціальністю 153 – «Мікро- та наносистемна техніка».

Для розуміння викладеного в посібнику матеріалу необхідне знання основ курсів «Інформатика» та «Алгоритми і структури даних».

1. БАЗОВІ ПОНЯТТЯ І КЛАСИФІКАЦІЯ

1.1. Визначення і базові поняття

В найпростішому випадку, згідно принципів фон Неймана, будь-яка система обробки інформації повинна мати наступні пристрої (рис. 1.1):

- арифметично-логічний пристрій (АЛП), який виконує арифметичні і логічні операції;

- керуючий пристрій (КП), який організує процес виконання програм;

- запам’ятовуючий пристрій (ЗП), або пам'ять для зберігання програм і даних;

- зовнішні пристрої для введення-виведення інформації (ПВВ).

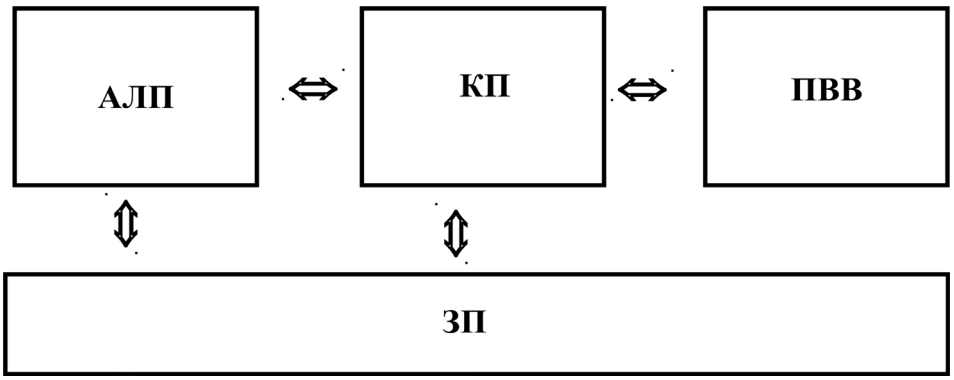


Рис. 1.1. Структура системи обробки інформації згідно принципів фон Неймана

Дані принципи в тій чи іншій мірі лежать в основі будь-яких сучасних систем обробки інформації. Що ж представляє собою сучасна мікропроцесорна система?

Мікропроцесорна система (МПС) – це функціонально і конструктивно закінчений виріб, до складу якого входить мікропроцесор. МПС призначена для отримання, аналізу, передавання та перетворення інформації.

Мікропроцесор (МП) – програмно-керований пристрій, як правило, виконаний на одній великій інтегральній схемі (ВІС), який здійснює керування процесом обробки інформації.

В сучасній техніці мікропроцесори часто об’єднують в одну систему з різноманітними додатковими пристроями: оперативними і постійними запам'ятовуючими пристроями (ОЗП і ПЗП), інтерфейсами, службовими пристроями (тактовими генераторами (ТГ), таймерами, регістрів) тощо. Таке об’єднання називають ***мікроконтролером*** (МК). Мікропроцесори та мікроконтролери є основними елементами МПС.

Інтерфейс (англ. interface – засіб спряження, сполучення) є сукупністю уніфікованих технічних і програмних засобів, необхідних для підключення зовнішніх пристроїв. Він забезпечує перетворення сигналів МП у сигнали, які сприймаються зовнішніми пристроями, і навпаки, підсилення сигналів та являє собою апаратні засоби і набір програм передачі даних (уніфікований протокол обміну інформацією).

В даний час існує величезна кількість типів мікроконтролерів, які відрізняються архітектурою процесорного модуля, розміром і типом вбудованої пам'яті, набором периферійних пристроїв, типом корпусу тощо.

Мікропроцесори і мікроконтролери є основою мікропроцесорних систем.

У мікропроцесорній техніці часто використовують термін ***контролер*** (від англ. controller – керуючий пристрій). Він має різне значення залежно від характеру застосування. В одному з варіантів його використовують як назву пристроїв управління периферійними пристроями мікропроцесорних систем (контролер клавіатури, контролер переривань тощо). Саме в цьому сенсі цей термін буде використано в даному посібнику.

Архітектурою процесора називається комплекс його апаратних і програмних засобів, що надаються користувачеві, а саме: набір доступних програмних регістрів і виконавчих (операційних) пристроїв, система команд і способів адресації, об’єм і структура пам'яті, види і способи обробки переривань.

Регістр процесора – блок комірок пам'яті, який утворює надшвидку оперативну пам'ять всередині процесора; використовується самим

процесором і здебільшого недоступний програмісту: наприклад, при виборці з пам'яті чергової команди вона поміщається в регістр команд, до якого програміст звернутися не може.

При описі архітектури і функціонування мікропроцесора зазвичай використовується його подання у вигляді сукупності програмно-доступних регістрів, що утворюють регістрову або програмну модель. У цих регістрах містяться оброблювані дані (операнди) і керуюча інформація. Відповідно, у регістрову модель входить група регістрів загального призначення (РЗП), призначених для зберігання операндів, і група службових регістрів, які забезпечують керування ходом виконання програми, режимом роботи процесора та організацію роботи з пам'яттю.

Регістри загального призначення утворюють регістровий пристрій (РП), який представляє собою внутрішню регістрову пам'ять процесора. Склад службових регістрів визначається конкретною архітектурою мікропроцесора, зазвичай до нього входять:

- програмний лічильник – ПЛ (program counter – PC);
- регістр стану процесора – РСП (state register – SR);
- керуючий регістр (керує режимом роботи процесора) – КР (control register – CR).

Регістрів РСП та КР може бути кілька. Це залежить від складності процесора.

Як працює мікропроцесорна система?

У загальних рисах роботу мікропроцесорної системи можна описати так. Спочатку за допомогою будь-якого зовнішнього пристрою в пам'ять системи вводиться програма. Пристрій управління зчитує вміст комірки пам'яті, де знаходиться перша інструкція (команда) програми (адреса цієї команди міститься у ПЛ), і організовує її виконання. Ця команда може задавати виконання арифметичних або логічних операцій, читання з пам'яті даних для виконання цих операцій або записування їх результатів у пам'ять,

введення даних із зовнішнього пристрою в пам'ять або виведення даних з пам'яті у зовнішній пристрій.

Як правило після виконання однієї команди пристрій управління починає виконувати команду з комірки пам'яті, яка знаходиться безпосередньо за тільки що виконаною командою. Однак цей порядок може бути змінений за допомогою спеціальних команд передачі управління (переходу), які вказують, з якої комірки пам'яті слід продовжити виконання програми. Такий перехід може бути організований певним чином, що дозволяє створювати складні програми.

Результат роботи процесора можна представити як зміну стану регістрів, яка відбувається внаслідок виконання операторів програми або оброблення сигналів від зовнішніх пристроїв. Зміна стану регістрів приводить до зміни стану всієї системи. Зміна стану регістрів веде до зміни електричного потенціалу на пов’язаних із ними зовнішніх виводах мікроконтролера. Це дає змогу керувати зовнішніми пристроями (вмикати, вимикати, змінювати режими їх роботи).

Даний опис дещо спрощений. Реальні системи на основі мікропроцесорів та мікроконтролерів мають більш складну архітектуру і, як наслідок, дещо інший алгоритм роботи. Приклад структурної схеми сучасного мікроконтролера наведено на рис. 1.2.

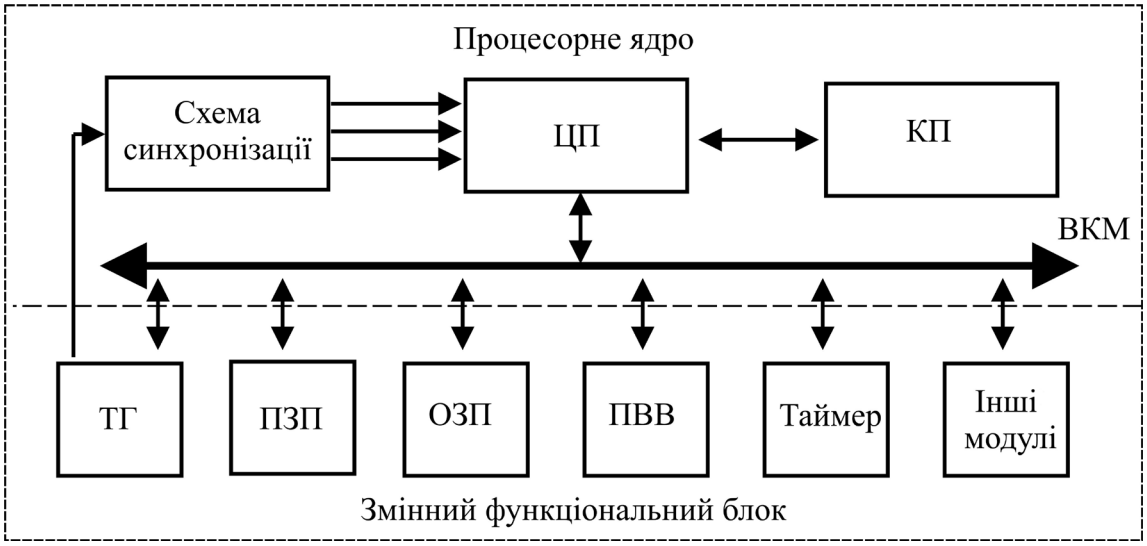


Рис. 1.2. Структурна схема сучасного мікроконтролера

Навіть мікроконтролери з однаковою архітектурою можуть відрізнятися набором функціональних вузлів і характером зв’язку між ними. Однак будь-який мікроконтролер містить такі функціональні блоки:

- центральний процесор або процесорне ядро, яке складається з керуючого пристрою (КП) і операційних пристроїв (ОП);

- внутрішня пам'ять (РП, кеш-пам'ять, блоки ОЗП і ПЗП);

- інтерфейсний блок, який забезпечує вихід на системну шину та обмін даними із зовнішніми пристроями через паралельні або послідовні порти введення / виведення;

- периферійні пристрої (таймерні модулі, аналого-цифрові перетворювачі, спеціалізовані контролери);

- різні допоміжні схеми (генератори тактових імпульсів, схеми налагодження і тестування, сторожовий таймер тощо).

Зв'язок між складовими частинами мікропроцесорної системи здійснюється за допомогою внутрішньої контрольної магістралі (ВКМ), яка складається із шини даних (ШД), шини адреси (ША) та шини керування (ШК). Синхронізація роботи складових частин мікропроцесорної системи здійснюється за допомогою тактового генератора (ТГ) та спеціальної схеми синхронізації.

1.2. Класифікація мікроконтролерів

Існуючі мікроконтролери можна класифікувати за різними ознаками. Так за розрядністю шини даних мікроконтролери можна поділити на наступні класи:

- 1-бітні (MC14500 фірми Motorola);

- 4-бітні (Atmel MARC4, Winbond W742, NEC uPD75 тощо);

- 8-бітні (Intel MCS-48, Intel MCS-51, Atmel ATtiny/ATmega/ATXmega, Microchip PIC12/16/18, Zilog Z86 тощо);

- 16-бітні (Intel MCS-96, Texas Instruments MSP430, Motorola 68HC16, Fujitsu MB90, Infineon C16, Mitsubishi M16C, Microchip PIC24 тощо);

- 32-бітні (Atmel ARM, Fujitsu MB91, NEC V850, NXP LPC2xxx тощо).

Можливо з часом з’являться мікроконтролери з більшою розрядністю шини даних, адже існують 64-бітні мікропроцесори (Athlon 64, Opteron, Sempron, Turion 64, Phenom, Athlon II, Phenom II, Pentium D, Xeon, ARM Cortex-A53, Cortex-A57 тощо).

За функціональним призначенням мікроконтролери можна поділити на мікроконтролери загального призначення і спеціалізовані. Спеціалізовані, в свою чергу, можна поділити на мікроконтролери цифрової обробки сигналів (з плаваючою крапкою та з фіксованою крапкою), мікроконтролери для керування двигунами та інші (рис. 1.3).

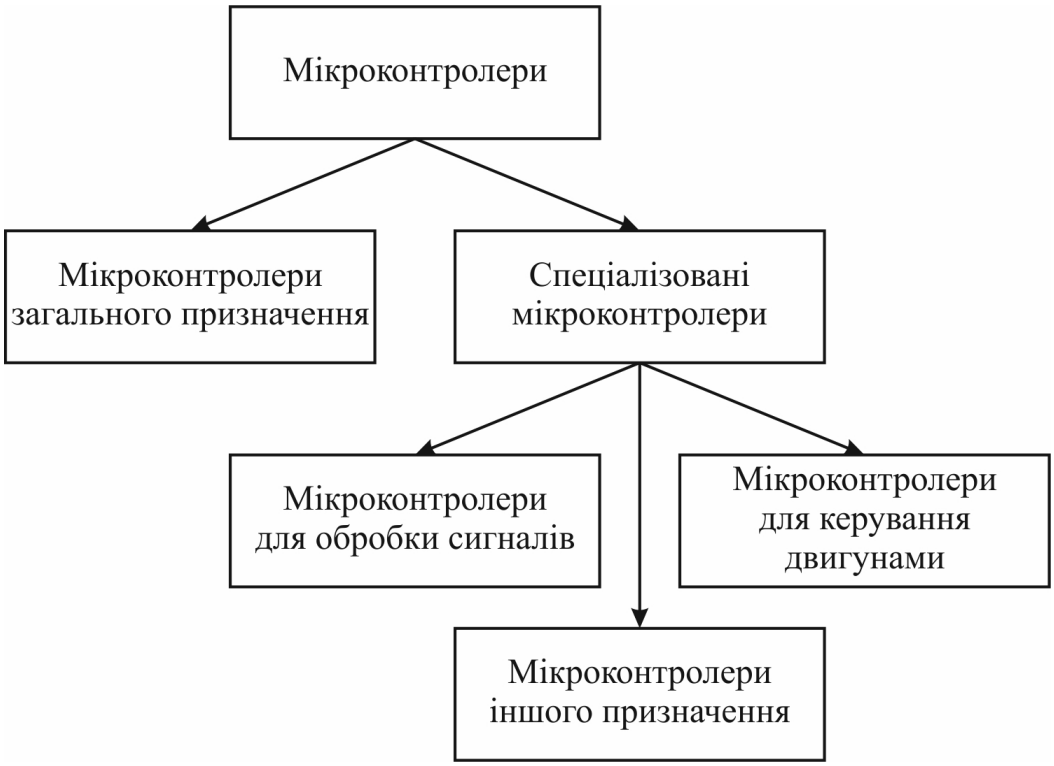


Рис. 1.3. Класифікація мікроконтролерів за призначенням

За типом архітектури мікроконтролери можна поділити на CISC (Complex Instruction Set Computing) та RISC (Reduced Instruction Set Computing) архітектури.

Особливості CISC-архітектури полягають в тому, що команди виконуються по черзі (одна за одною) і мають різну довжину і структуру.

Вибірка команди з пам'яті здійснюється побайтово, а сама команда виконується за кілька тактів. CISC-архітектуру мають: МК із сімейства Motorola HC05 / HC08, МК з ядром MCS-51, МК із сімейства Infineon C500 і низка інших.

Основна ідея RISC архітектури полягає в заміні складних команд однотипними простими і виконанні їх єдиним потоком на паралельному конвеєрі. Всі команди мають фіксовану довжину і в ідеалі мають виконуватися за один, а не за кілька тактів, чим досягається підвищення швидкодії. Саме ця архітектура вважається більш перспективною і натепер є більш поширеною.

Одним з перших МК з архітектурою RISC став PIC-контролер 16C54 фірми Microchip. Завдяки високій продуктивності й трьом десяткам простих команд PIC-контролери швидко завоювали популярність у всьому світі. Невдовзі їх приклад наслідували розробники з фірм Atmel, Scenix та інші.

Іноді в класифікації за типом архітектури окремо виділяють мікроконтролери з ARM архітектурою. Архітектура ARM (від англ. Advanced RISC Machine – вдосконалена RISC-машина) – це RISC архітектура на основі ліцензованих 32-бітних і 64-бітних мікропроцесорних ядер розробки компанії ARM Limited.

За характером роботи з пам’яттю мікроконтролери можна поділити на два типи: принстонський тип та гарвардський тип.

Принстонський тип був розроблений Джоном фон Нейманом і незалежно від нього академіком С. О. Лебедєвим. У ньому використовується спільна пам'ять для зберігання програм і даних (рис. 1.4). Основна перевага полягає у спрощенні схемотехніки центрального процесора (ЦП) та в гнучкості розподілу ресурсів між областями пам'яті.

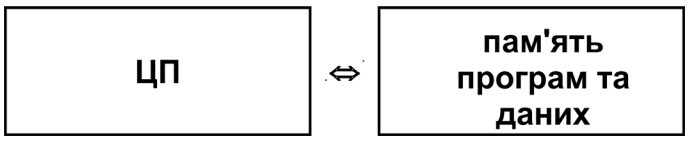


Рис. 1.4. Принстонський тип роботи з пам’яттю

Особливістю гарвардського типу є наявність розділених адресних просторів для зберігання команд і даних (рис. 1.5). Цей тип майже не використовувався до кінця 1970-х років, поки розробники МК нарешті не зрозуміли, що саме він дає їм певні переваги. Зокрема, аналіз реальних програм показує, що об’єм пам'яті даних МК, використовуваний для зберігання проміжних результатів, приблизно на порядок менший за необхідний об’єм пам'яті програм. Значить, можна скоротити розрядність шини даних, зменшити число транзисторів в мікросхемі, а заодно і прискорити доступ до інформації. Як наслідок, зараз більшість сучасних МК використовують RISC-архітектуру гарвардського типу.

Рис. 1.5. Гарвардський тип роботи з пам’яттю

Мікроконтролери об'єднуються в сімейства. До одного сімейства відносять вироби, які мають однакове ядро – сукупність таких понять, як система команд, циклограма роботи ЦП, організація пам'яті програм і пам'яті даних, система переривань і базовий набір периферійних пристроїв. Відмінності між різними представниками одного сімейства полягає, в основному, в складі периферійних пристроїв і об’ємі пам'яті програм або даних. Найбільш важлива особливість сімейства – програмна сумісність на рівні двійкового коду всіх МК даного сімейства. Зазвичай фірма-виробник дає ім'я сімейства свого МК. Наприклад, PIC16 – сімейство 16-бітних мікроконтролерів фірми Microchip, AVR – сімейство мікроконтролерів фірми Atmel.

Контрольні запитання

1. Наведіть структурну схему системи обробки інформації.
2. Дайте визначення поняття «мікропроцесорна система».
3. Дайте визначення поняття «мікропроцесор».
4. Чим мікроконтролер відрізняється від мікропроцесора?
5. Дайте визначення поняття «інтерфейс».
6. Дайте визначення поняття «архітектура процесора».
7. Дайте визначення поняття «регістр процесора».
8. Наведіть складові регістрової моделі.
9. Які функції виконують регістри загального призначення?
10. Які функції виконують службові регістри?
11. Що являє собою регістровий пристрій?
12. Наведіть типовий склад службових регістрів.
13. Опишіть загальний алгоритм роботи мікропроцесорної системи.
14. Наведіть узагальнену структурну схему сучасного мікроконтролера.
15. Перелічіть ознаки, за якими можна класифікувати мікроконтролери.
16. Наведіть класифікацію мікроконтролерів за розрядністю шини даних.
17. Наведіть класифікацію мікроконтролерів за функціональним призначенням.
18. Наведіть класифікацію мікроконтролерів за типом архітектури.
19. У чому полягає основна ідея CISC архітектури?
20. У чому полягає основна ідея RISC архітектури?
21. Що представляє собою ARM архітектура?
22. Наведіть класифікацію мікроконтролерів за типом роботи з пам’яттю.
23. Опишіть суть принстонського типу роботи з пам’яттю.
24. Опишіть суть гарвардського типу роботи з пам’яттю.
25. У чому перевага гарвардського типу роботи з пам’яттю, порівняно із принстонським?
26. Назвіть ознаки, за якими мікроконтролери відносять до різних сімейств.

2. БУДОВА МІКРОПРОЦЕСОРНИХ СИСТЕМ

2.1. Загальна структура мікропроцесорної системи

Будова будь-якої МПС базується на двох основних принципах:

- модульність побудови;
- магістральний принцип обміну інформацією.

Модульний принцип означає, що систему будують на основі обмеженої кількості типів конструктивно і функціонально завершених модулів. Виходячи з цього, будь-яка МПС повинна складатися, як мінімум, з модуля процесора, модуля пам'яті та модуля управління введенням / виведенням. Основою модуля процесора є МП.

Магістральний принцип обміну інформацією визначає характер зв'язків між модулями МПС. Існує два принципи взаємодії елементів модулів і самих модулів в системі: принцип довільних зв'язків, що реалізують правило «кожен з кожним», і принцип упорядкованих зв'язків – магістральний або шинний.

При використанні принципу довільних зв'язків всі сигнали між пристроями передаються по окремих лініях зв'язку. Кожен пристрій, що входить в систему, передає свої сигнали і коди незалежно від інших пристроїв. При цьому в системі виходить багато ліній зв'язку та різних правил (протоколів) обміну інформацією.

При магістральному принципі всі сигнали між пристроями передаються по одних лініях зв'язку, але в різний час (це називається часовим мультиплексуванням). Обмін інформацією по окремих лініях зв'язку може здійснюватися в одному напрямку (односпрямована передача) або в двох напрямках (двоспрямована передача). Група ліній зв'язку, по яких передаються сигнали і коди, називається магістраллю або шиною (англомовний термін – bus).

Застосування принципу магістрального обміну дозволяє мінімізувати кількість зв'язків між пристроями системи, скоротити кількість виводів ВІС.

Значна перевага магістрального принципу зв'язку полягає в тому, що всі пристрої повинні приймати і передавати інформацію по одним і тим же правилам (протоколам обміну). Це забезпечує стандартизацію інтерфейсів всіх пристроїв, що входять в систему.

При використанні магістрального принципу обміну інформацією всі модулі МПС з'єднуються з єдиною магістраллю, яку часто називають ***системною шиною*** (рис. 2.1). Шина являє собою набір електричних провідників, об'єднаних функціонально і часто фізично, наприклад, на одній друкованій платі.

Рис. 2.1. Магістрально-модульна структура МПС

Весь інформаційний потік, який циркулює в МПС, зазвичай розділяють на три групи: адреси, дані та керуючі сигнали. Відповідно до цього у системній шині (магістралі) виділяють три шини нижнього рівня:

- шина адреси – ША;
- шина даних – ШД;
- шина керування – ШК.

ША призначена для однозначного визначення адреси елемента МПС (наприклад, комірки пам'яті або пристрою введення / виведення). Зазвичай адресу задає процесор, тому шина адреси найчастіше односпрямована.

ШД служить для обміну даними між елементами МПС. Шина даних завжди двоспрямована.

ШК призначена для керування роботою елементів МПС. По ній передаються керуючі сигнали. Окремі лінії шини керування можуть бути односпрямованими або двоспрямованими.

При описі архітектури і функціонування мікропроцесор зазвичай розглядають у вигляді набору програмно-доступних елементів – регістрів, які утворюють так звану програмну модель МП. У цих регістрах містяться оброблювані дані (операнди) і керуюча інформація. Відповідно, в програмну модель входить група регістрів загального призначення (РЗП), які призначені для зберігання операндів, і група службових регістрів, які забезпечують керування режимами роботи МП. Регістри загального призначення утворюють регістровий пристрій (РП).

Як було показано вище існує дві основних архітектури CISK та RISK.

CISC-архітектура характеризується великим набором різноформатних команд з використанням численних способів адресації. Це класична архітектура процесорів, яка почала свій розвиток в 1940-х роках з появою перших комп'ютерів. Основною метою її було скорочення розміру програм, що зменшувало вимоги до об’єму оперативної пам'яті. Розширення спектру операцій, що реалізуються системою команд, дозволяло зменшити розмір програм, а також трудомісткість їх написання і налагодження. Для CISC-процесорів характерні:

- велика кількість команд (іноді багатофункціональних);
- велика кількість форматів команд різної розрядності;
- велика кількість методів адресації;
- широке використання команд обробки типу «регістр–пам'ять»;
- порівняно невелика кількість РЗП (8 – 16).

Характерними особливостями RISC-мікропроцесорів є:

- розширений обсяг реєстрової пам'яті: від 32 до кількох сотень РЗП, які входять до складу МП;
- використання в командах обробки даних тільки реєстрової адресації (звернення до пам'яті використовується в командах завантаження і збереження вмісту регістрів, а також в командах передачі керування);
- відмова від апаратної реалізації складних способів адресації;
- фіксований формат команд (зазвичай 4 байта);

- виключення з набору команд, які рідко використовуються, а також команд, які не вписуються в прийнятий формат.

Переважне використання реєстрової адресації значно підвищує продуктивність МП. Фіксований формат команд, відмова від складних і рідко використовуваних команд і способів адресації суттєво спрощує пристрій керування, що дає змогу зменшити розмір кристалу RISC-процесорів, знизити їх вартість і підвищити тактову частоту. Використання фіксованого формату команд забезпечує також більш ефективну роботу конвеєра МП, зменшує число тактів простою і очікування, що дає додаткове зростання продуктивності. Тому RISC-процесори в 2–4 рази більш продуктивні за CISC-процесори з тією ж тактовою частотою, незважаючи на дещо більший (приблизно на 30%) об’єм програм.

Важливою архітектурною особливістю мікропроцесорів є використовуваний варіант реалізації пам'яті та організація вибірки команд і даних. Як було показано вище, за цими ознаками розрізняють процесори Принстонского і Гарвардського типу.

Принстонська архітектура, яка часто називається архітектурою Фон-Неймана (за ім’ям керівника розробки), характеризується загальною оперативною пам'яттю для зберігання програм і даних. Для звернення до цієї пам'яті використовується загальна системна шина, по якій в процесор надходять і команди, і дані (рис. 2.2).

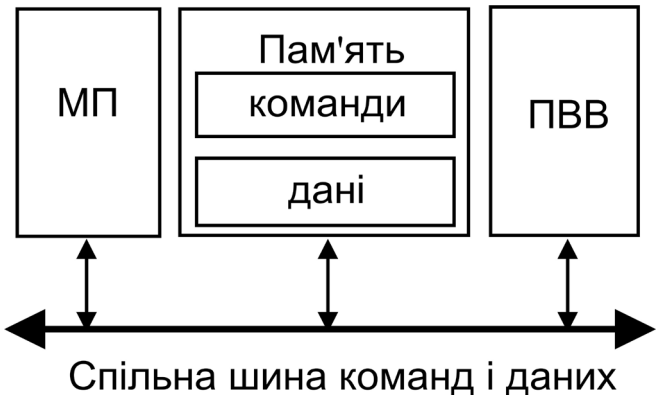


Рис. 2.2. Структура МПС з принстонським типом доступу до пам’яті

Така архітектура має ряд переваг. Наявність загальної пам'яті дає змогу оперативно перерозподіляти її об’єм для зберігання окремих масивів команд і даних залежно від розв'язуваних завдань. Забезпечується можливість більш ефективного використання пам'яті МП. Перерозподіл пам'яті не викликає ніяких проблем, головне – щоб програма і дані разом містилися в пам'яті. Як правило, в системах з такою архітектурою пам'ять буває достатньо великого об’єму (до десятків і сотень мегабайт). Це дає змогу вирішувати найскладніші завдання. Використання загальної шини для передачі команд і даних значно спрощує налагодження, тестування і поточний контроль функціонування системи, підвищує її надійність. Однак їй притаманні й суттєві недоліки. Справа в тому, що при єдиній шині команд і даних процесор змушений по одній цій шині приймати дані (з пам'яті або ПВВ) і передавати дані (в пам'ять або ПВВ), а також читати команди з пам'яті. Одночасно ці пересилання кодів по шині відбуватися не можуть, вони мають проводитися по черзі, що різко обмежує продуктивність системи.

Саме тому зростаючі вимоги до продуктивності мікропроцесорних систем спричинили в останні роки перехід до архітектури з гарвардським типом доступу до пам'яті (з двома системними шинами).

Кожна пам'ять з'єднується з процесором окремої шиною (рис. 2.3), що дозволяє одночасно з читанням / записуванням даних при виконанні поточної команди робити вибірку наступної команди. Завдяки такому розподілу потоків команд і даних та поєднанню операцій їх вибірки і виконання реалізується більш висока продуктивність, ніж при використанні архітектури принстонського типу.

Недоліки архітектури гарвардського типу пов'язані з необхідністю використання більшої кількості шин та ускладненням структури процесора. Крім того, істотний недолік – це фіксований об’єм пам'яті, виділеної для команд і даних, призначення якої не може оперативно перерозподілятися відповідно до вимог розв'язуваної задачі. Тому доводиться застосовувати пам'ять більшого об’єму, і коефіцієнт використання пам'яті, як правило,

виявляється більш низьким, ніж в системах з архітектурою принстонського типу.

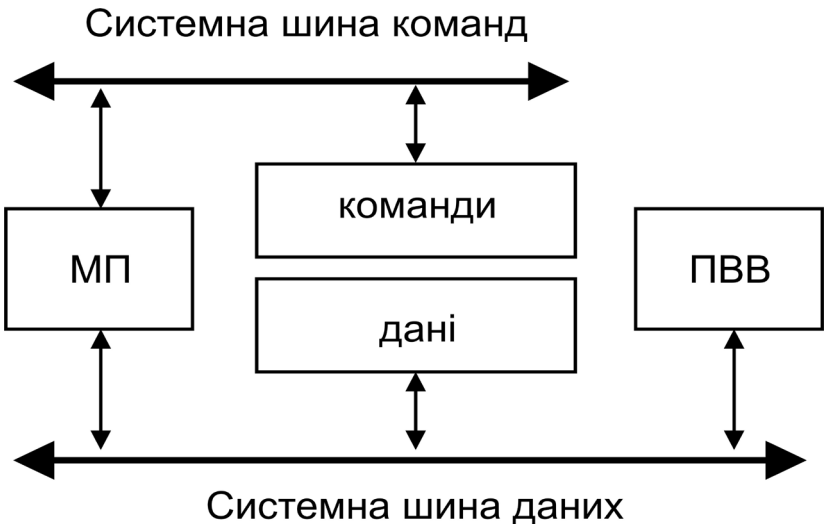


Рис. 2.3. Структура МПС з гарвардським типом доступу до пам’яті

2.2. Запам’ятовуючі пристрої

Запам’ятовуючі пристрої (ЗП) призначені для зберігання програми, вихідних даних і результатів обробки, обміну інформацією між окремими частинами мікропроцесорної системи.

Максимально можливий об’єм інформації, який можна записати у ЗП, визначається його інформаційною ємністю. Одиницею вимірювання інформаційної ємності є біт – мінімальний об’єм цифрової інформації, який може приймати значення 0 або 1, або слово, яке складається з декількох біт, наприклад, з чотирьох, восьми або шістнадцяти. Біт зберігається у запам’ятовуючому елементі (ЗЕ), а слово – у запам’ятовуючій комірці, яка є групою ЗЕ, до яких можливе лише одночасне звертання.

Для вимірювання значних обсягів інформаційної ємності використовують кратні одиниці вимірювання, наприклад, кілобіт (1 кбіт = 2^{10} = 1024 біт), кілобайт (1 кбайт = 2^{10} = 1024 байт), мегабіт (1 Мбіт = 2^{20} = 1048576 біт), мегабайт (1 Мбайт = 2^{20} = 1048576 байт) тощо.

Організація ЗП визначає, скільки слів і якої розрядності він зберігає. Так, наприклад, при ємності 512 біт може зберігатися 64 восьмирозрядних (восьмибітних) слова або 128 чотирирозрядних (чотирибітних).

Для зберігання оперативної інформації використовують ОЗП (RAM – Random Access Memory). Такі ЗП поділяються на статичні та динамічні.

У статичних ЗП – SRAM (Static RAM) – як ЗЕ використовуються тригери з колами установки і встановлення нуля. За використання КМОН-інверторів тригер складається з шести транзисторів (рис. 2.4). Такі ЗП достатньо дорогі та займають багато місця на кристалі, але мають значну швидкодію.

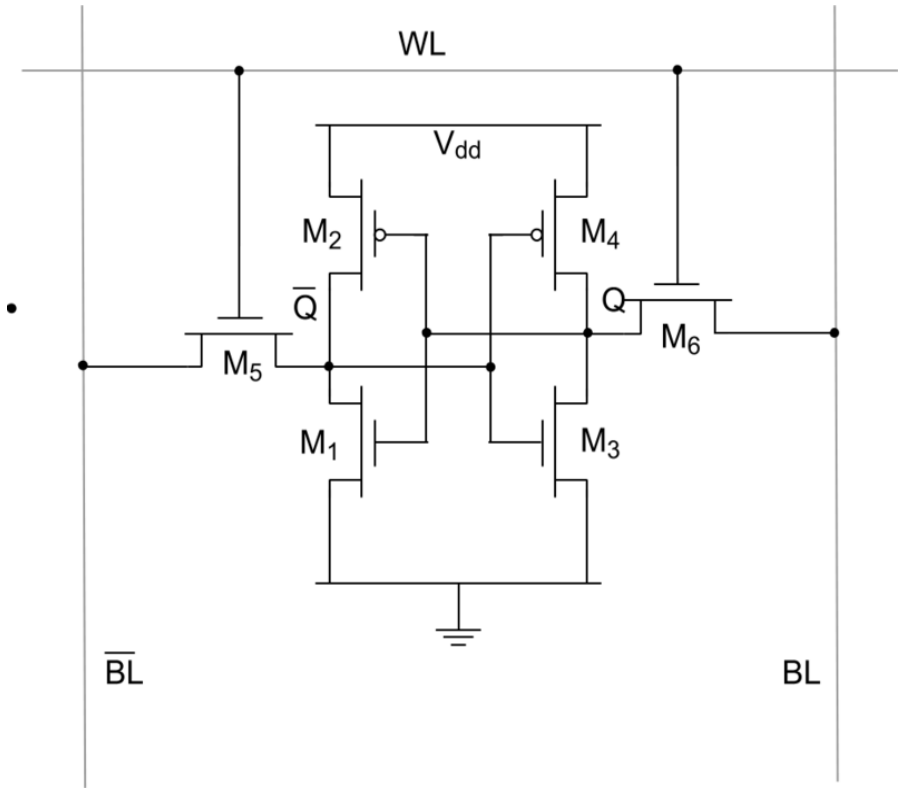


Рис. 2.4. ЗЕ SRAM, виконаний за КМОН технологією

Принцип роботи такого ЗЕ наступний. Лінія *WL* (Word Line) керує двома транзисторами доступу (М5 і М6). Лінії $\overline{BL}$ і *BL* (Bit Line) - бітові лінії, використовуються і для записування, і для читання даних.

Запис. При подачі «0» на лінію $\overline{BL}$ або *BL* паралельно включені транзисторні пари (М5 і М1) і (М6 і М3) утворюють логічні схеми 2–АБО,

наступне подання «1» на лінію WL приводить до відповідного перемикання тригера.

Читання. При подачі «1» на лінію WL відкриваються транзистори M5 і M6. Рівні потенціалів, записані в тригері, виставляються на лінії $\overline{BL}$ і BL і потрапляють на схеми читання.

У мікросхемі SRAM ЗЕ об'єднані в матрицю, керування якою здійснюють за допомогою спеціальної схеми (рис. 2.5).

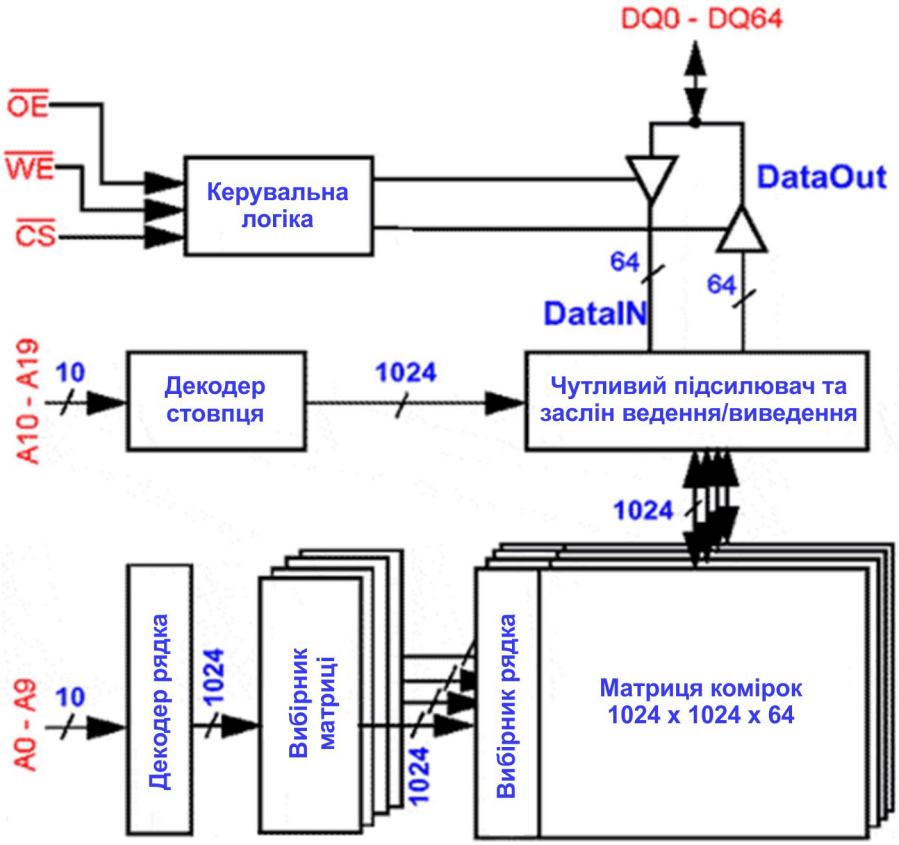


Рис. 2.5. Типова структурна схема мікросхеми SRAM

Алгоритм роботи такої схеми наступний:

Цикл читання

Цикл читання починається зі скидання сигналу $\overline{CS}$ (Chip Select – вибір чіпа) в низький стан. Це вказує мікросхемі, що чіп "обраний", і зараз з ним будуть працювати.

До того моменту, коли сигнал стабілізується, на адресних лініях має бути встановлена адреса (тобто номер рядка (row) і номер стовпця (column)),

а сигнал $\overline{WE}$ має бути переведений у високий стан (відповідає операції читання комірки). Рівень сигналу $\overline{OE}$ (Output Enable – дозвіл виведення) не має ніякого значення, тому що на виході поки нічого не міститься, точніше вихідні лінії знаходяться у так званому високоімпедансному стані.

Через деякий час, який визначається швидкодією керуючої логіки і швидкоплинністю перехідних процесів в інверторах, на лініях виходу з'являються очікувані дані, які аж до закінчення робочого циклу можуть бути безпосередньо зчитані. Зазвичай час доступу до ЗЕ статичної пам'яті не перевищує 1...2 нс.

Цикл записування

Цикл записування відбувається у зворотньому порядку. Спочатку встановлюють на шину адресу запису і, одночасно з цим, скидають сигнал $\overline{WE}$ в низький стан. Потім, дочекавшись, коли адресу буде декодовано і надіслано на відповідні бітові лінії, скидають $\overline{CS}$ в низький рівень, наказуючи мікросхемі подати сигнал високого рівня на необхідну лінію row. Засувка, що утримує тригер, відкриється і, в залежності від стану bit-лінії, тригер перемкнеться в той чи інший стан.

У динамічних ЗП – DRAM (Dynamic RAM) – дані зберігаються у вигляді зарядів ємностей МОН-структур. Основою ЗЕ тут є просто конденсатор невеликої ємності. Такий ЗЕ значно простіший тригерного, що дає змогу розмістити на кристалі значно більше ЗЕ. Оскільки конденсатор з часом втрачає свій заряд, то зберігання даних вимагає їхньої періодичної регенерації (через кожні декілька мікросекунд) за допомогою спеціальних підсилювачів-регенераторів. Динамічні ЗП у 4...5 разів дешевші статичних і у стільки ж разів мають більшу інформаційну ємність. Обидва ці типи пам’яті енергозалежні – при вимкненні джерела живлення ІМС інформація втрачається. Структура мікросхеми DRAM аналогічна структурі мікросхеми SRAM.

Для зберігання сталої інформації використовують ЗП ROM-типу (Read Only Memory – ПЗП). У якості ЗЕ в них використовують перемички, діоди,

біполярні і МОН-транзистори. Вони бувають кількох видів – маскові ЗП, програмовані ЗП (PROM, EPROM та EEPROM).

У маскові ЗП типу ROM інформація записується при виготовленні за допомогою спеціального шаблона – маски. Матриця діодного ЗП такого типу являє собою координатну сітку з горизонтальних ліній вибірки слів і вертикальних ліній зчитування (рис. 2.6). Код слова визначається наявністю діода (що відповідає одиниці) або його відсутністю (відповідає нулю) у вузлах координатної сітки. Також в якості ЗЕ можуть бути використані напівпровідникові біполярні транзистори, МОН-транзистори тощо.

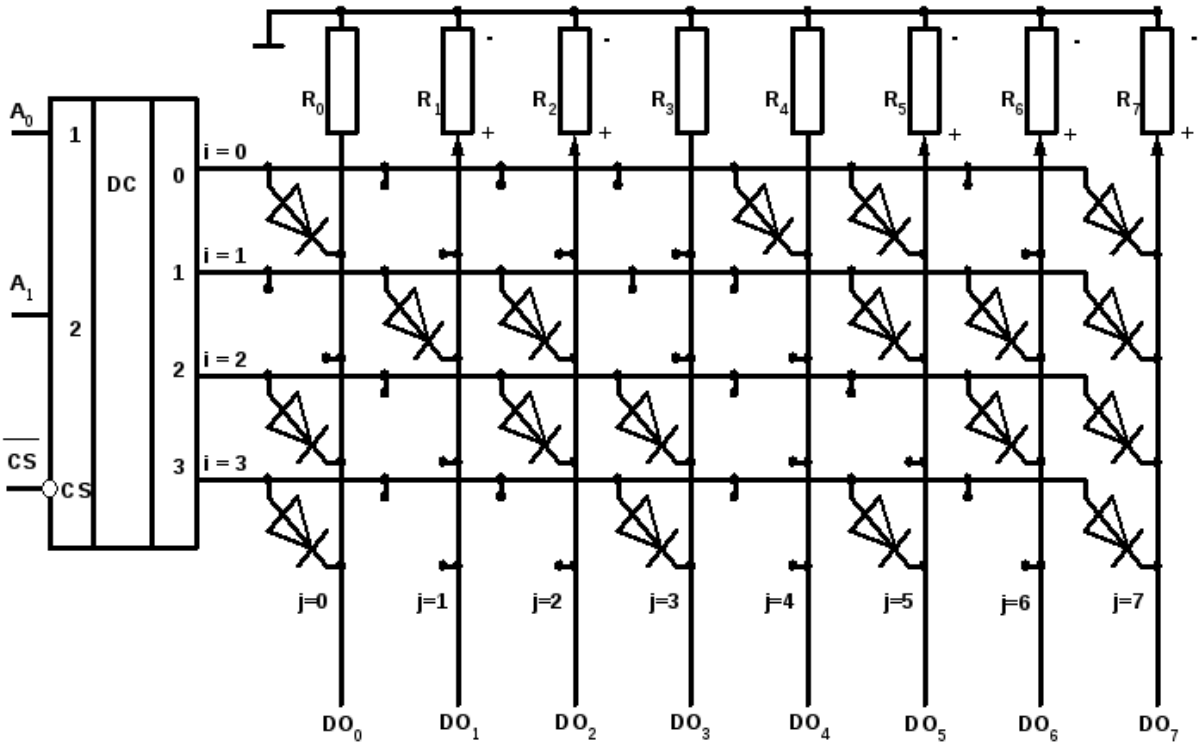


Рис. 2.6. Масковий ЗП на основі діодної матриці

Такі ЗП компактні та дешеві. Їх застосовують для зберігання інформації масового призначення – кодів літер і цифр, таблиць типових функцій, стандартного програмного забезпечення тощо. Користувач не може змінити інформацію, яка зберігається.

Мікросхеми ЗП типу PROM (Programmable ROM – програмовані ПЗП) програмують одноразово видаленням або створенням перемичок у вузлах

координатної сітки. У вихідній заготовці наявні (або відсутні) всі перемички. При програмуванні користувачем за допомогою спеціального програмуючого пристрою залишаються (або видаляються) тільки необхідні. Перемички можуть являти собою у найпростішому випадку плавкі елементи ввімкнені послідовно з діодом, або два зустрічно увімкнених діоди, один з яких пробивається при програмуванні. Такий вид ЗП зараз майже не використовується.

ЗП типу EPROM (Erasable Programmable Read Only Memory) дають змогу не тільки записувати в них інформацію, а й стирати її та замінювати на нову – вони є репрограмовуваними. У якості ЗЕ в них використано транзистори ЛІЗМОН-типу (МОН-транзистори з лавинною інжекцією заряду, рис. 2.7). Такі транзистори мають так званий плаваючий заслін – обмежену з усіх боків діелектриком провідну зону. Введений у неї під дією імпульсу напруги у 20...25 В заряд зберігається дуже тривалий час. Цей заряд забезпечує закритий стан транзистора.

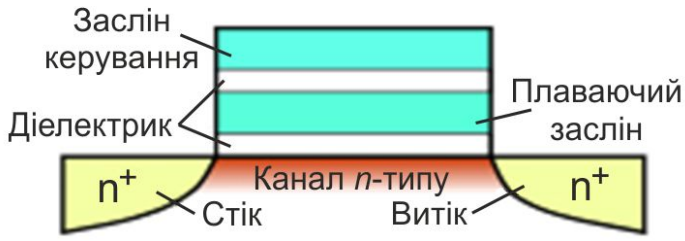


Рис. 2.7. Транзистор з плаваючим заслоном

Стирання інформації відбувається під дією ультрафіолетового випромінювання, для чого корпус ІМС має спеціальне прозоре віконце (рис. 2.8). Виникнення фото- і теплових струмів дозволяє заряду покинути плаваючий заслін. Стирання триває декілька хвилин, одразу стирається вся інформація. Опромінення веде до змін властивостей матеріалів транзисторів, тому кількість циклів перепрограмування є обмеженою (10...100 разів).

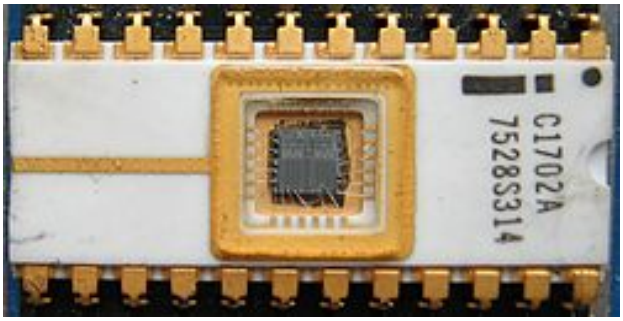


Рис. 2.8. Мікросхема ЗП типу EPROM

ЗП типу EEPROM (Electrically Erasable Programmable Read-Only Memory) також є репрограмовуваними. Їхньою основою є МНОН-транзистори, які, на відміну від звичайних МОН-транзисторів, мають двошаровий підзаслонний діелектрик – окрім тонкого шару SiO_2 розміщується також більш товстий шар нітриду кремнію Si_3N_4. Під дією електричного поля достатньо високої напруженості носії заряду проходять через тонкий шар і накопичуються на границі поділу шарів. Після зняття поля заряд залишається у приграничному шарі нітриду кремнію, що і забезпечує зберігання інформації протягом десятків років. Для стирання інформації необхідно видалити носії заряду з приграничного шару, для чого подається напруга, яка створює електричне поле протилежного напряму. При цьому інформацію можна стирати не зі всього кристалу, а вибірково. Тривалість процесу достатньо коротка. Кількість циклів перепрограмування складає від десятків тисяч до мільйона разів. Однією з різновидів EEPROM є флеш-пам'ять (англ. Flash Memory). Її ЗЕ подібні елементам пам’яті типу EEPROM, але в схемах флеш-пам’яті не передбачене стирання окремих слів. Інформація може стиратися або вся одразу, або достатньо великими блоками за єдиним сигналом, миттєво (flash – спалах). За своєю дією флеш-пам’ять подібна до пам’яті ОЗП, але є енергонезалежною. Пристрої флеш-пам’яті поступово витісняють такі накопичувачі, як жорсткі магнітні диски, оптичні диски та інші носії. Вони у сотні разів скорочують споживану потужність, мають достатньо велику надійність, невеликі розміри і вагу та значно більшу швидкодію. Крім того, однією з найважливіших переваг таких ЗП є

відсутність рухомих механічних частин і повна технологічна сумісність з ІМС. Флеш-пам’ять може бути частиною багатьох типів ІМС мікроконтролерів.

Також натепер ведеться активна робота по розробці ЗП на основі мемристорів. Мемристор – це пасивний елемент, що змінює опір залежно від значення заряду, який протікає через нього. Після зняття напруги мемристор зберігає свій стан.

2.3. Арифметико-логічний пристрій

Арифметико-логічний пристрій (АЛП) (англ. Arithmetic and logic unit, ALU) – блок процесора, який під керуванням керуючого пристрою (КП) служить для виконання арифметичних і логічних перетворень (починаючи від елементарних) над даними, званими в цьому випадку операндами. Розрядність операндів зазвичай називають розміром або довжиною машинного слова.

Арифметико-логічний пристрій (рис. 2.9) умовно можна розділити на групу регістрів, мікропрограмний пристрій (керуючий пристрій), який задає послідовність мікрокоманд (команд), та операційний пристрій, в якому реалізується задана послідовність мікрокоманд (команд).

Рис. 2.9. Структура арифметико-логічного пристрою

Регістри діляться на акумулятори (регістри, в яких зберігається результат виконання операції), регістри операндів (регістри, в яких

зберігаються дані, необхідні для виконання поточної операції), регістри адреси (призначені для зберігання адреси операндів результату), індексні регістри (призначені для зберігання адреси наступного елемента при обробці масивів даних) та допоміжні регістри загального призначення.

Вміст деяких операційних регістрів може бути використаний для розміщення операндів команди. Такі регістри називають програмно-доступними. До них належать: акумулятори, індексні регістри і деякі допоміжні регістри. Решта регістрів є програмно-недоступними.

Розрізняють два види мікрокоманд: зовнішні, які надходять в АЛП від зовнішніх джерел і викликають в ньому перетворення інформації, і внутрішні, які генеруються в АЛП і впливають на керуючий пристрій (операційний пристрій), змінюючи певним чином нормальний порядок проходження команд.

Така складна логічна структура АЛП зумовлена кількістю різних мікрооперацій, необхідних для виконання всього комплексу завдань, поставлених перед арифметико-логічним пристроєм. На вході кожного регістру зібрані відповідні логічні схеми, які дають змогу реалізувати задані мікрооперації. Порядок виконання мікрокоманд визначається алгоритмом виконання операцій.

Всі виконувані в АЛП операції є логічними операціями (функціями), які можна поділити на наступні групи:

- операції двійковій арифметики для чисел з фіксованою крапкою;
- операції двійкової (або шістнадцяткової) арифметики для чисел з плаваючою крапкою;
- операції десяткової арифметики;
- операції індексної арифметики (при модифікації адрес команд);
- операції спеціальної арифметики;
- операції над логічними кодами (логічні операції);
- операції над алфавітно-цифровими полями.

Сучасні ЕОМ загального призначення зазвичай реалізують операції всіх наведених вище груп, а малі та мікроЕОМ, мікропроцесори і спеціалізовані ЕОМ часто не мають апаратури арифметики чисел з плаваючою крапкою, десяткової арифметики і операцій над алфавітно-цифровими полями. У такому випадку ці операції виконують спеціальні підпрограми.

До арифметичних операцій відносять додавання і віднімання («короткі операції»), а також множення і ділення («довгі операції»). Групу логічних операцій складають операції диз'юнкція (логічне АБО) і кон'юнкція (логічне І) над багаторозрядними двійковими словами, порівняння кодів на рівність. Спеціальні арифметичні операції включають в себе нормалізацію, арифметичний зсув (зсуваються тільки цифрові розряди, знаковий розряд залишається на місці), логічний зсув (знаковий розряд зсувається разом з цифровими розрядами). Також існує група операцій редагування алфавітно-цифрової інформації. Кожна операція в АЛП є логічною функцією або послідовністю логічних функцій описуваних, як правило, двійковою логікою.

За способом дії над операндами АЛП поділяють на послідовні та паралельні. У послідовних АЛП операнди представляються в послідовному коді, а операції проводяться послідовно в часі над їх окремими розрядами. У паралельних АЛП операнди представляються паралельним кодом і операції здійснюються паралельно в часі над усіма розрядами операндів.

За способом представлення чисел розрізняють АЛП:

- для чисел з фіксованою крапкою;
- для чисел з плаваючою точкою;
- для десяткових чисел.

За характером використання елементів і вузлів АЛП поділяють на блокові та багатофункціональні. У блоковому АЛП операції над числами з фіксованою і плаваючою крапкою, десятковими числами і алфавітно-цифровими полями виконуються в окремих блоках, при цьому підвищується швидкість роботи, оскільки блоки можуть паралельно виконувати відповідні

операції, але значно зростає складність обладнання. У багатофункціональних АЛП операції для всіх форм представлення чисел виконуються одними і тими ж схемами, які комутуються потрібним чином залежно від необхідного режиму роботи.

2.4. Інтерфейси

В даний час використовується безліч різних типів МПС, які відрізняються операційними системами, центральними процесорами, мережевими інтерфейсами і багатьма іншими параметрами. Через ці відмінності проблема взаємодії між різними системами привела до необхідності створення єдиного стандарту. У 1977 р. Міжнародна організація по стандартизації (International Organization for Standardization – ISO) створила комітет з розробки стандартів обміну даними. Результатом роботи цього комітету стала модель взаємодії відкритих систем (Open System Interconnection – OSI).

Щоб спростити проблему, комітет ISO пішов шляхом поділу складного процесу обміну інформацією на кілька простіших задач (рівнів). Процес обміну інформацією розділили на сім рівнів:

– (7) прикладний рівень – забезпечує виконання прикладних процесів користувачів і визначає смисловий зміст інформації, якою обмінюються відкриті системи в процесі їх взаємодії. З цією метою даний рівень, крім протоколів взаємодії прикладних процесів, підтримує протоколи передачі файлів, віртуального терміналу, електронної пошти та їм подібні;

– (6) представницький рівень – визначає єдиний для всіх відкритих систем синтаксис переданої інформації;

– (5) сеансовий рівень – відповідає за організацію сеансів зв'язку між прикладними процесами, розташованими в різних абонентських системах. На даному рівні створюються порти для прийому і передачі повідомлень та організовуються з'єднання – логічні канали між процесами;

– (4) транспортний рівень – служить для організації передавання даних між двома взаємодіючими відкритими системами і організації процедури сполучення абонентів мережі та системи передачі даних;

– (3) мережевий рівень – призначений для організації процесів маршрутизації інформації та управління мережею передачі даних;

– (2) канальний рівень – забезпечує функціональні та процедурні засоби для встановлення, підтримання і розривання з'єднань на рівні каналів передачі даних, а також забезпечує, якщо можливо, корекцію помилок, які виникають на фізичному рівні;

– (1) фізичний рівень – забезпечує механічні, електричні, функціональні та процедурні засоби організації фізичних з'єднань при передавання бітів даних між фізичними об'єктами.

Реалізація другого і першого рівнів моделі OSI реалізується в МПС за допомогою різноманітних інтерфейсів.

Як уже відзначалося у першому розділі, інтерфейс – це сукупність уніфікованих технічних і програмних засобів, необхідних для підключення зовнішніх пристроїв. Він забезпечує перетворення сигналів МП у сигнали, які сприймаються зовнішніми пристроями, і навпаки, а також підсилення сигналів і являє собою набір апаратних і програмних засобів для забезпечення передавання даних згідно встановленого протоколу обміну інформацією.

У мікропроцесорних системах використовують різноманітні інтерфейси, що зумовлено великою різноманітністю задач, які виконують МПС.

За способом передачі даних інтерфейси можна поділити на синхронні та асинхронні. При синхронному способі передавання даних робота передавального і приймального пристроїв синхронізується спеціальним синхронізуючим сигналом.

Крім того інтерфейси можна поділити на послідовні та паралельні. У послідовних інтерфейсах передавання (приймання) інформації здійснюється

послідовно біт за бітом. При паралельному передаванні даних передається одночасно ціла група бітів. Як правило – це байт або слово.

2.4.1. Послідовний периферійний інтерфейс

Послідовний периферійний інтерфейс (Serial Peripheral Interface – SPI) – це послідовний синхронний стандарт передачі даних в режимі повного дуплексу, призначений для забезпечення простого і недорогого високошвидкісного сполучення мікроконтролерів і периферії. SPI також іноді називають чьотирьохпровідним (англ. Four-wire) інтерфейсом.

В SPI використовують чотири цифрові сигнали:

– MOSI – вихід ведучого, вхід веденого (англ. Master Out Slave In), який служить для передавання даних від ведучого пристрою до веденого;

– MISO – вхід ведучого, вихід веденого (англ. Master In Slave Out), який служить для передавання даних від веденого пристрою ведучому;

– SCLK або SCK – послідовний тактовий сигнал (англ. Serial Clock), який служить для передавання тактового сигналу для ведених пристроїв;

– $\overline{CS}$ або $\overline{SS}$ – вибір мікросхеми, вибір веденого (англ. Chip Select – CS, Slave Select – SS).

Конкретні імена портів інтерфейсу SPI можуть відрізнятися залежно від виробника апаратних засобів, при цьому можливі наступні варіанти:

MOSI: SIMO, SDI (на пристрої), DO, DON, SI, MRSR;

MISO: SOMI, SDO (на пристрої), DI, DIN, SO, MTST;

SCLK: SCK, CLK;

$\overline{SS}$: $\overline{nCS}$, $\overline{CS}$, $\overline{CSB}$, $\overline{CSN}$, $\overline{nSS}$, $\overline{STE}$, $\overline{SYNC}$.

SPI є синхронним інтерфейсом, в якому будь-яка передача синхронізована із загальним тактовим сигналом, який генерується ведучим пристроєм (процесором). Приймаюча (ведена) периферія синхронізує отримання бітової послідовності з тактовим сигналом (за один такт синхронізуючого сигналу передається в обох напрямах один біт даних). До

одного послідовного периферійного інтерфейсу провідного пристрою мікросхеми може приєднуватися одна або кілька мікросхем (рис. 2.10, 2.11). Ведучий пристрій обирає ведений, активуючи сигнал «вибір кристалу» (англ. Chip select) на веденій мікросхемі. Периферія, яка не обрана процесором, не приймає участі в процесі обміну даними по SPI.

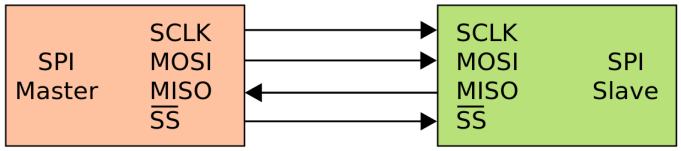


Рис. 2.10. Підключення ведучого та веденого пристроїв при використанні послідовного периферійного інтерфейсу за схемою один до одного

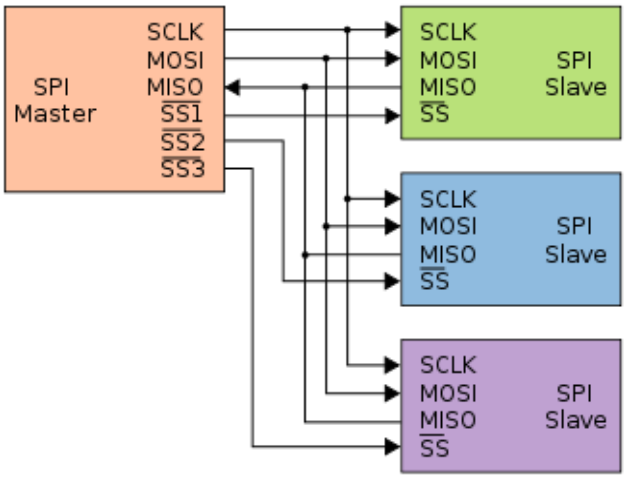


Рис. 2.11. Підключення ведучого та ведених пристроїв при використанні послідовного периферійного інтерфейсу за схемою один до багатьох

Швидкість передавання даних визначається синхросигналом SCK (Serial Clock), який генерує ведучий пристрій. Ведені пристрої використовують синхросигнал для визначення моментів зміни бітів на лінії даних, при цьому ведені пристрої ніяк не можуть впливати на частоту слідування бітових інтервалів. Як у ведучому пристрої, так і в веденому пристрої, є лічильник імпульсів синхронізації (бітів). Лічильник у веденому

пристрої дозволяє останньому визначити момент закінчення передавання пакета. Лічильник скидається при виключенні підсистеми SPI, така можливість завжди є у ведучому пристрої. У веденому пристрої лічильник зазвичай скидається деактивацією інтерфейсного сигналу $\overline{SS}$.

Оскільки дії ведучого і веденого пристроїв тактуються одним і тим же сигналом, то до стабільності цього сигналу не пред'являється ніяких вимог, за винятком обмеження на тривалість напівперіодів, яка визначається максимальною робочою частотою найповільнішого пристрою. Це дозволяє використовувати SPI в системах з нестабільною тактовою частотою, а також полегшує програмну емуляцію ведучого пристрою.

За внутрішньою структурою SPI інтерфейс являє собою тактований регістр зсуву. Процес обміну даними між ведучим і веденим пристроями показано на рис. 2.12.

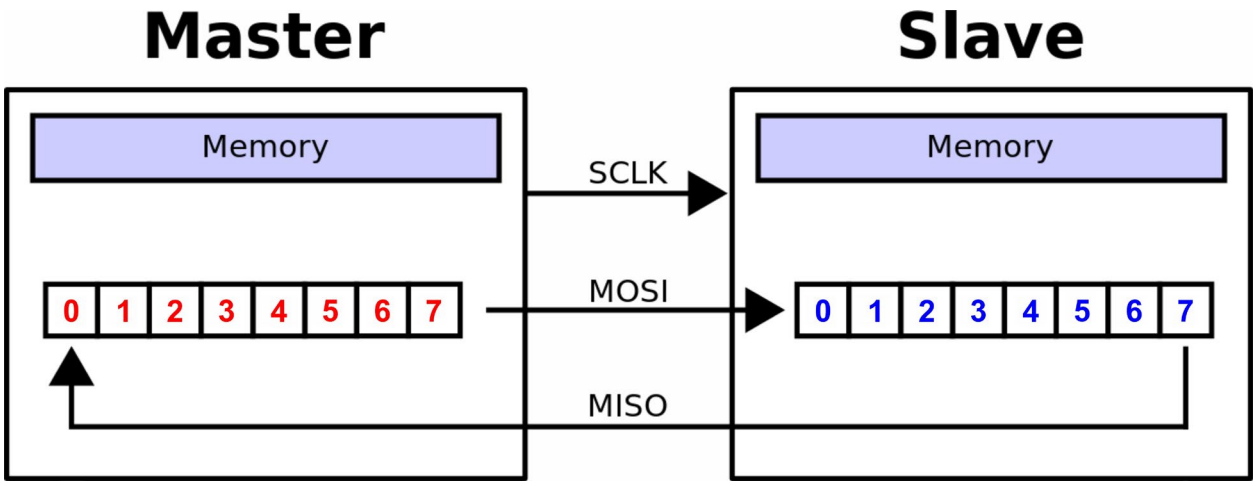


Рис. 2.12. Напрям обміну даними між ведучим і веденим пристроями при використанні послідовного периферійного інтерфейсу

Передача здійснюється пакетами. Довжина пакету, як правило, становить 1 байт (8 біт) або слово (16 біт). Однак відомі реалізації SPI з іншою довжиною пакета. Ведучий пристрій ініціює цикл зв'язку встановленням сигналу низького рівня на виводі вибору веденого пристрою ($\overline{SS}$) того пристрою, з яким необхідно встановити з'єднання. При низькому

рівні сигналу $\overline{SS}$ схемотехніка веденого пристрою знаходиться в активному стані; тактовий сигнал SCLK від ведучого пристрою сприймається веденим і викликає зміщення бітів. Процес зміщення бітів показано на рис. 2.13.

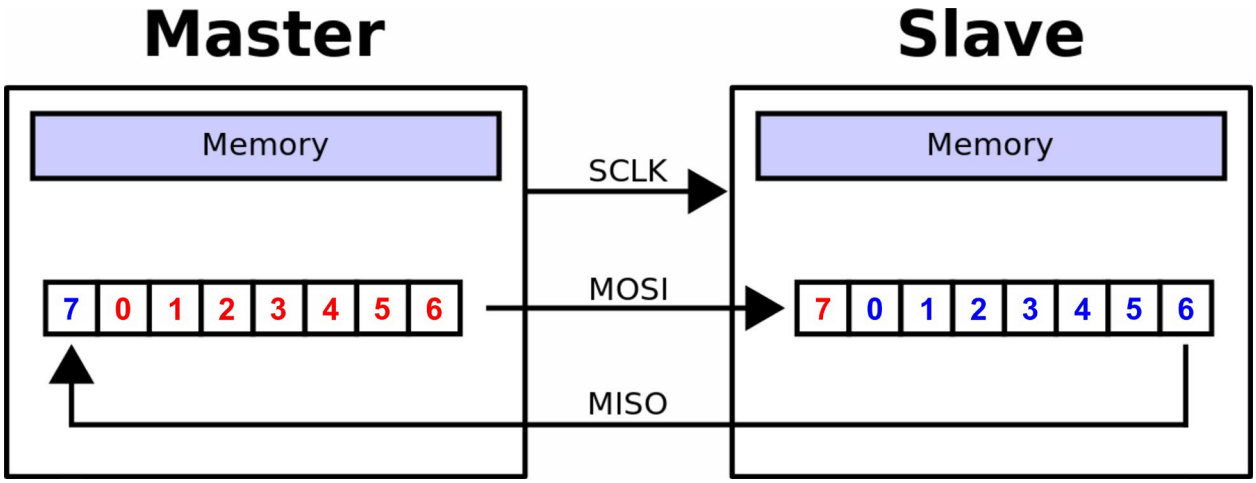


Рис. 2.13. Зміщення бітів після першого такту SCLK

Для успішного обміну даними ведучий і ведений пристрої повинні мати однакові налаштування, а саме значення фази (CPHA) і полярності (CPOL) сигналу SCLK по відношенню до сигналів даних. Всього можливі чотири комбінації CPHA і CPOL сигналу SCLK по відношенню до сигналів даних. Режими роботи визначаються комбінацією бітів CPHA і CPOL:

– CPOL = 0 – сигнал синхронізації починається з низького рівня;

– CPOL = 1 – сигнал синхронізації починається з високого рівня;

– CPHA = 0 – вибірка даних проводиться по передньому фронту сигналу синхронізації;

– CPHA = 1 – вибірка даних проводиться по задньому фронту сигналу синхронізації.

Для позначення режимів роботи інтерфейсу SPI прийняті наступні позначення:

– режим 0 (CPOL = 0, CPHA = 0);

– режим 1 (CPOL = 0, CPHA = 1);

– режим 2 (CPOL = 1, CPHA = 0);

– режим 3 (CPOL = 1, CPHA = 1).

Часові діаграми роботи інтерфейсів для різних режимів роботи зображені на рис. 2.14.

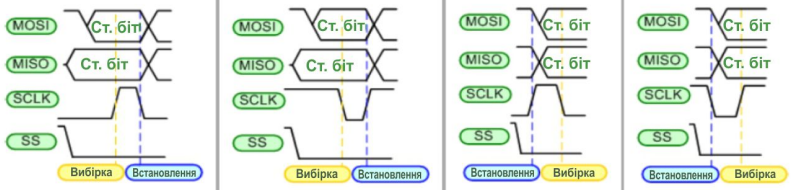
Режим SPI	0	1	2	3
CPOL	0	1	0	1
CPHA	0	0	1	1
Часова діаграма першого цикла синхронізації				

Рис. 2.14. Часові діаграми роботи інтерфейсу SPI

Переваги послідовного периферійного інтерфейсу:

– простота протоколу;

– високий рівень надійності;

– висока швидкість передачі даних.

Недоліки послідовного периферійного інтерфейсу:

– складність топології при кількох ведених пристроях;

– неможливість перевірки успішності процесу обміну даними без значного ускладнення процесу передавання.

2.4.2. Паралельний периферійний інтерфейс

Паралельний периферійний інтерфейс (Parallel Peripheral Interface – PPI) – це синхронний стандарт передавання даних, який підтримує приймання/передавання даних у паралельному форматі (від 8 до 16 біт за один такт синхронізуючого сигналу). Кожен біт групи передається по окремій сигнальній лінії (рис. 2.15). Передавання і приймання даних відбувається по черзі (напрям передачі обирається сигналом на лінії вибору напряму передачі даних). У якості ліній приймання/передавання даних використовуються лінії портів введення/виведення загального призначення.

Для реалізації передавання за схемою «один до одного» необхідно використати на дві лінії більше, ніж кількість бітів, які передають за один такт синхронізуючого сигналу (10 ліній на 8 бітів або 18 ліній на 16 бітів).

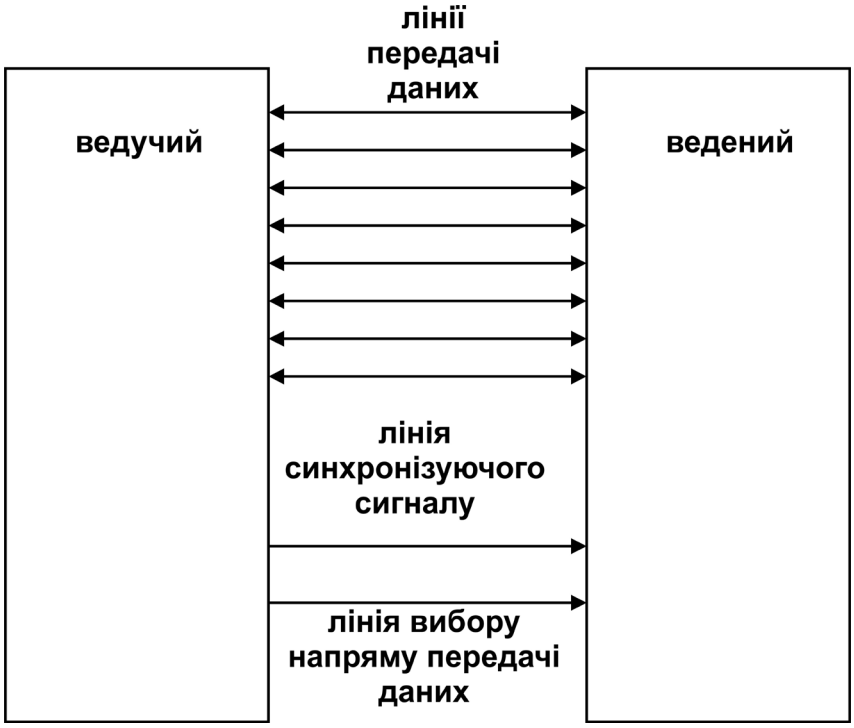


Рис. 2.15. Підключення ведучого та веденого пристроїв при використанні паралельного периферійного інтерфейсу за схемою «один до одного»

Через складність топології реалізують, як правило, системи з одним ведучим і одним веденим пристроєм, хоча принципово можливо реалізувати і систему з кількома веденими пристроями. Тоді необхідно додатково використовувати кілька ліній для вибору веденого (по одній лінії на кожного веденого).

Настроювання параметрів передавання/приймання, а також контроль поточного стану здійснюється на програмному рівні шляхом встановлення керуючих бітів відповідних регістрів.

Швидкість передачі даних можна розрахувати як добуток частоти синхронізуючого сигналу на кількість бітів переданих за один такт.

Переваги паралельного периферійного інтерфейсу:

– простота протоколу;

– високий рівень надійності;

– висока швидкість передачі даних.

Недоліком паралельного периферійного інтерфейсу є складність топології.

2.4.3. Послідовна асиметрична шина

Послідовна асиметрична шина (IIC, англ. Inter-Integrated Circuit, також відома як I2C або I^2C) використовує для передачі інформації дві лінії (рис. 2.16): лінію даних (SDA) і лінію синхронізації (SCL). Кожен пристрій має унікальну адресу і може працювати як передавач або приймач, залежно від призначення пристрою. Адреса може бути 7 або 10 біт в залежності від реалізації. Пристрої з 7-бітовим і 10-бітовим адресами можуть одночасно використовуватися на одній шині, незалежно від швидкості передачі. Швидкість передачі може приймати одне з двох значень: 100 кбіт/с (стандартний режим) або 400 кбіт/с (швидкісний режим). Кількість підключених до шини пристроїв обмежується двома факторами – максимальною кількістю доступних адрес (залежить від розміру адреси), а також максимальною сумарною вхідною ємністю підключених до шини пристроїв – 400 пФ.

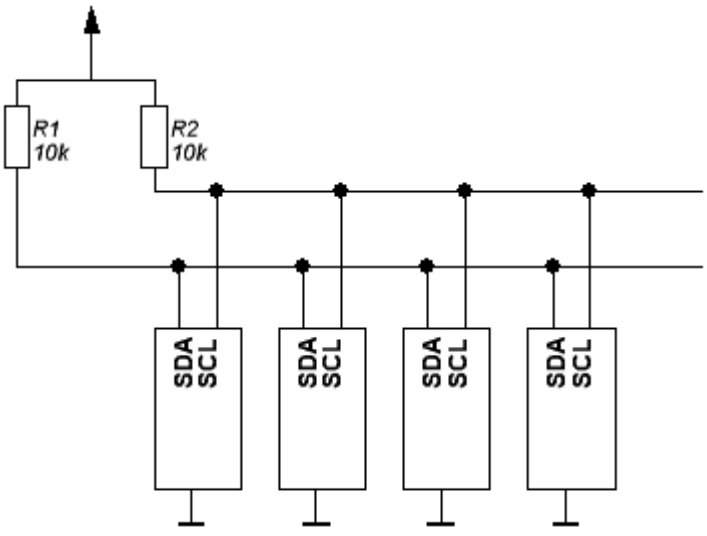


Рис. 2.16. Підключення пристроїв до послідовної асиметричної шини

Всі підключені пристрої можуть бути класифіковані як ведучі та ведені в процесі передавання даних. Ведучий ініціює передавання даних і виробляє сигнали синхронізації. При цьому будь-який пристрій, який адресується, вважається веденим по відношенню до ведучого.

Сигнал даних оновлюється по задньому фронту тактового сигналу, а його вибірка відбувається по передньому фронту (рис. 2.17).

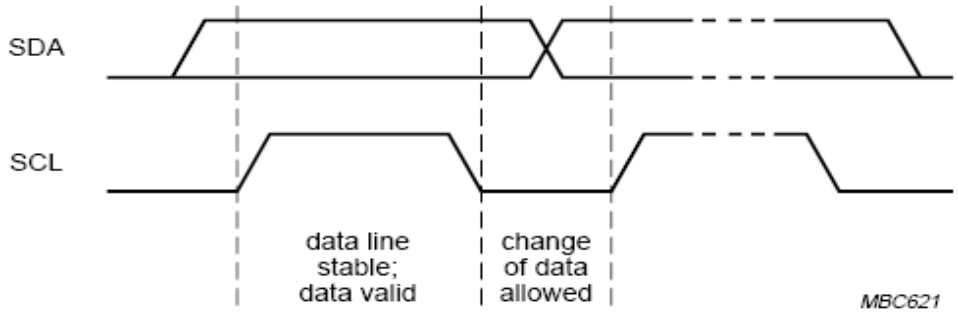


Рис. 2.17. Часова діаграма роботи послідовної асиметричної шини

Дані на лінії SDA мають бути стабільними протягом "ВИСОКОГО" ("HIGH") періоду синхроімпульсу. Стан лінії даних має змінюватися, тільки якщо лінія синхронізації в стані "НИЗЬКИЙ" ("LOW"). Перехід лінії SDA з "ВИСОКОГО" стану в "НИЗЬКИЙ", в той час як SCL знаходиться в "ВИСОКОМУ" стані означає START. Перехід лінії SDA з "НИЗЬКОГО" стану в "ВИСОКИЙ" при SCL в "ВИСОКОМУ" стані означає STOP (рис. 2.18).

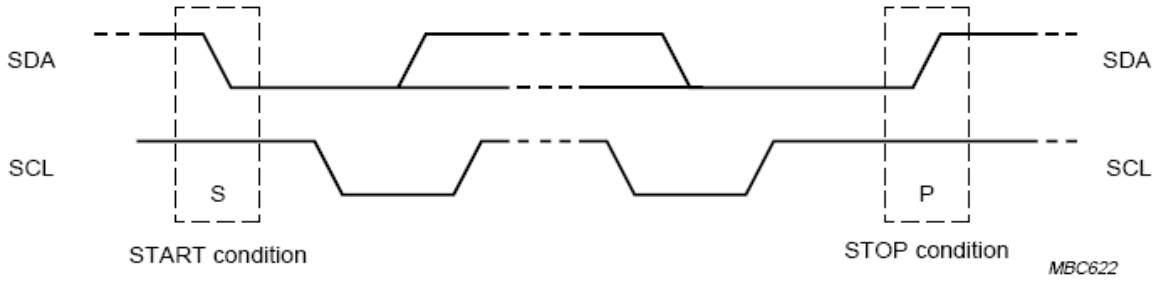


Рис. 2.18. СТАРТ і СТОП умови при передаванні даних за допомогою послідовної асиметричної шини

Сигнали START і STOP завжди виробляються ведучим пристроєм. Вважається, що шина зайнята після сигналу START. Шина вважається вільною через певний час після сигналу STOP.

Дані передають пакетами, кожен пакет складається з дев'яти біт. 8 біт даних і 1 біт підтвердження (ACK) / непідтвердження (NACK) прийому. Перший пакет надсилає ведучий пристрій до веденого. Цей пакет містить фізичну адресу веденого пристрою і біт напряму передачі. Після сигналу підтвердження (веденим) відбувається передавання даних від ведучого до веденого (біт напряму дорівнює нулю) або навпаки (біт напряму дорівнює 1). Після кожного байту приймаючий пристрій виставляє сигнал підтвердження / непідтвердження прийому. Після передачі останнього байту передаючий пристрій виставляє сигнал NACK, після чого ведучий пристрій виставляє сигнал STOP, що означає кінець сеансу зв’язку (рис. 2.19).

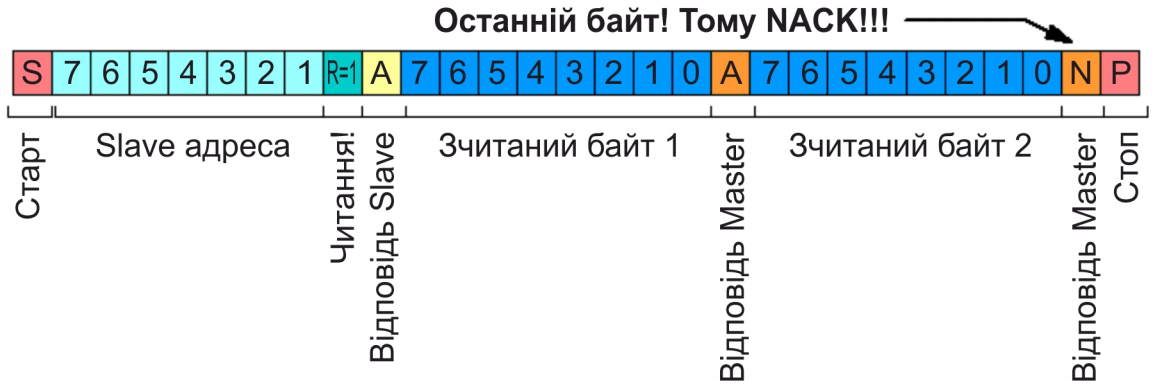


Рис. 2.19. Структура пакетів послідовної асиметричної шини при зчитуванні інформації ведучим від веденого

Часові діаграми сигналів ліній послідовної асиметричної шини показані на рис. 2.20.

Визначення сигналів START і STOP пристроями, підключеними до шини досить легко реалізувати пристроями, у яких апаратно реалізовані необхідні кола. Однак мікроконтролери без таких кіл мають здійснювати зчитування значення лінії SDA як мінімум двічі за період синхронізації для того, щоб визначити зміну стану.

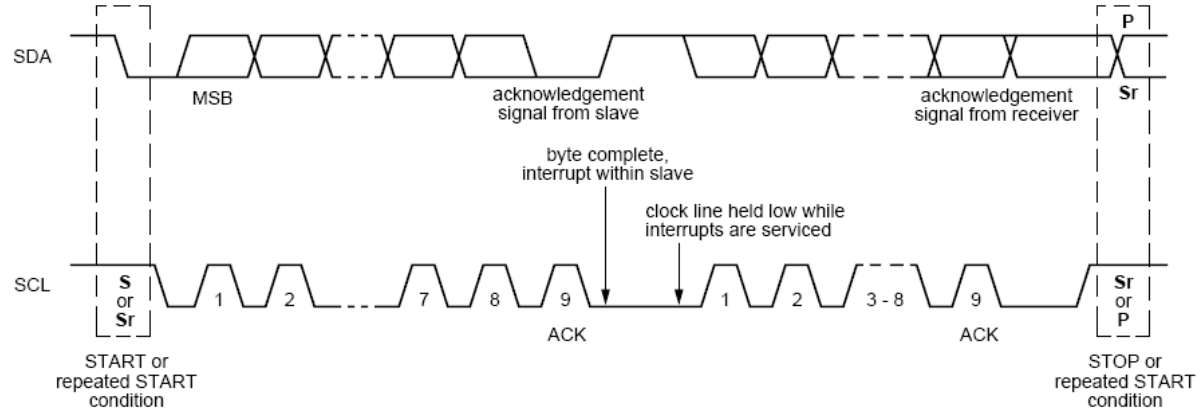


Рис. 2.20. Часові діаграми роботи послідовної асиметричної шини

Внаслідок різних технологій мікросхем (МОН, КМОН, біполярна), які можуть бути підключені до шини, рівні логічного нуля ("НИЗЬКИЙ") і логічної одиниці ("ВИСОКИЙ") не фіксовані і залежать від відповідного рівня напруги живлення пристрою. За один синхроімпульс передається один біт. Кожен байт, що передається по лінії SDA, має складатися з 8 біт. Кількість байт, переданих за один сеанс зв'язку необмежена. Дані передаються, починаючи з найбільш значущого біта. Кожен байт має закінчуватися бітом підтвердження (рис. 2.21). Підтвердження при передаванні даних є обов'язковим. Тактовий сигнал, необхідний для підтвердження, генерується ведучим пристроєм. Передавач звільняє лінію SDA ("ВИСОКИЙ") під час тактового сигналу при підтвердженні. Приймач для підтвердження має встановити на лінії SDA стан "НИЗЬКИЙ", коли такт в стані "ВИСОКИЙ".

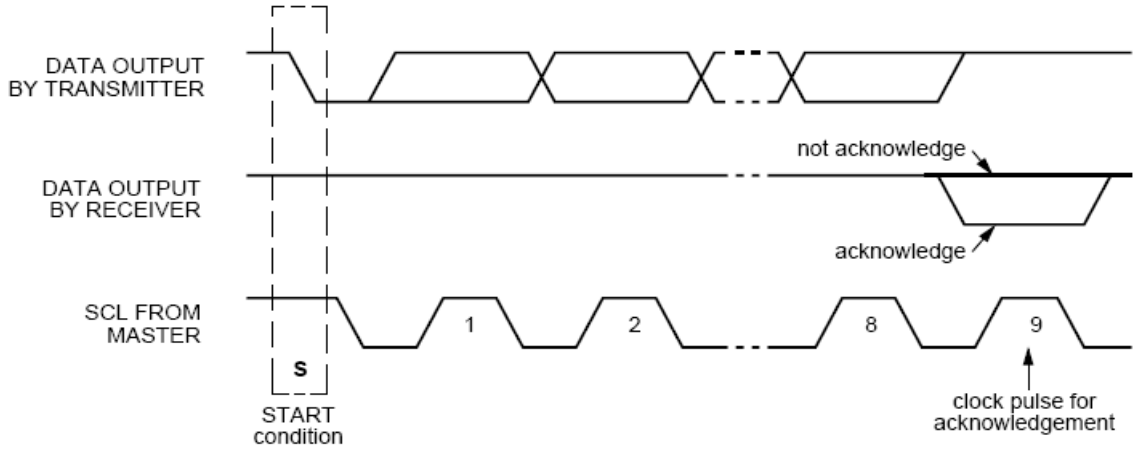


Рис. 2.21. Сигнал підтвердження при роботі послідовної асиметричної шини

В загальному випадку протокол роботи послідовної асиметричної шини достатньо складний і описує поведінку системи в різних штатних і нештатних ситуаціях. Ознайомитись з повною специфікацією можна на сайті фірми Philips Semiconductors.

Переваги шини I2C:

– проста топологія системи, що потребує лише дві сигнальні лінії навіть за великої кількості пристроїв на шині;

– відносно адаптується до потреб різних ведених пристроїв;

– підтримує кілька ведучих пристроїв на одній шині;

– включає в себе систему підтверджуючих сигналів (ACK / NACK) для покращення обробки помилок.

Недоліки шини I2C:

– має достатньо складний протокол передавання даних, що збільшує складність програмного та/або апаратного забезпечення;

– нав'язує накладні витрати протоколу (службову інформацію: адреса, різноманітні підтвердження тощо), що знижує пропускну здатність;

– потребує використання підтягуючих резисторів, які обмежують тактову частоту, займають корисне місце на друкованій платі в системах, обмежених за розміром, збільшують розсіювану потужність.

2.4.4. Асинхронний послідовний інтерфейс RS-232

RS-232 (англ. Recommended Standard 232, інша назва EIA232) – стандарт фізичного рівня для асинхронного інтерфейсу UART (універсальний асинхронний приймач-передавач УАПП, англ. – Universal Asynchronous Receiver-Transmitter). Широко відомий як послідовний порт персональних комп'ютерів. В даний час використовується для підключення до комп'ютерів широкого спектру обладнання, невибагливого до швидкості обміну, особливо при значній відстані його від комп'ютера і відхиленні умов застосування від стандартних.

RS-232 – інтерфейс передавання інформації між двома пристроями на відстані до 20 м. Інформація передається по проводах з рівнями сигналів, що відрізняються від стандартних 5 В, для забезпечення більшої стійкості до перешкод. Асинхронна передача даних здійснюється із заданою швидкістю при синхронізації рівнем сигналу стартового імпульсу.

Інтерфейс RS-232 працює в дуплексному режимі, що дає змогу передавати і приймати інформацію одночасно, тому що використовуються різні лінії для приймання і передавання. Швидкість роботи RS-232 залежить від відстані між пристроями (таблиця 2.1). На мінімальній відстані швидкість зазвичай дорівнює 115,2 кбіт/с, але є обладнання, яке підтримує швидкість до 921,6 кбіт/с. Зазвичай швидкість передавання у стандарті RS-232 вимірюється в Бодах. Один Бод дорівнює один біт/с.

Таблиця 2.1.

Залежність швидкості передачі даних від відстані в RS-232

Швидкість, Бод	**Відстань (довжина кабелю), м**
19 200	15
9 600	150
4 800	300
2 400	900

Протокол інтерфейсу передбачає два режими передавання даних: синхронний і асинхронний, а також два методи управління обміном даних: апаратний і програмний. Кожен режим може працювати з будь-яким методом управління. У протоколі також передбачається варіант управління передаванням даних за спеціальними сигналами, що встановлюються ведучим пристроєм (DSR – сигнал стану готовності, DTR – сигнал готовності передачі даних).

Для передавання даних по інтерфейсу RS-232 використовується код NRZ, який не передбачає самосинхронізації, тому для синхронізації

використовується стартовий і стоповий біти, які дають змогу виділити бітову послідовність і синхронізувати приймач з передавачем.

Пристрої для зв'язку по послідовному каналу з'єднуються кабелями з 9-ти або 25-ти контактними рознімачами типу D. Зазвичай вони позначаються DB-9, DB-25, CANNON 9, CANNON 25 тощо. Рознімачі складаються з розетки і вилки. Кожен контакт позначений і пронумерований. Нумерацію контактів розеток рознімачів DB-9 та DB-25 показано на рис 2.22 і 2.23. Опис сигналів представлений у таблиці 2.2.

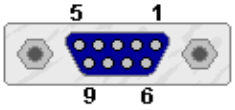


Рис. 2.22. Нумерація контактів розетки рознімача DB-9

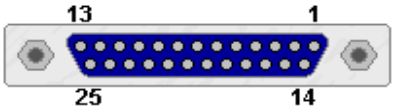


Рис. 2.23. Нумерація контактів розетки рознімача DB-25

Таблиця 2.2.

Призначення сигналів

Сигнал	**Контакт**		**Опис сигналу**
	DB-9	DB-25	
TXD	3	2	Transmit Data – лінія передавання даних
RTS	7	4	Request to Send – запит на передавання даних
DTR	4	20	Data Terminal Ready – готовність обладнання
RXD	2	3	Receive Data – лінія приймання даних
CTS	8	5	Clear to Send – готовність до передавання
DSR	6	6	Data Set Ready – готовність даних
GND	5	7	System Ground – заземлюючий контакт
DCD	1	8	Carrier Detect – виявлення несучої
RI	9	22	Ring Indicator – індикатор виклику

В RS-232 використовують два рівні сигналів: логічні 1 і 0. Логічну 1 іноді позначають MARK, логічний 0 – SPACE. Логічній 1 відповідають від'ємні рівні напруги, а логічному 0 – додатні. Відповідні значення напруг представлені в таблиці 2.3.

Таблиця 2.3.

Логічні рівні, використовувані в RS-232

Рівень	**Передавач**	**Приймач**
Логічний 0	+5…+15	+3…+25
Логічна 1	–5…–15	–3…–25
Не визначений	–3…+3	

Для забезпечення зв'язку між пристроями можуть бути використані кілька схем:

– мінімальна трьохпровідна (рис. 2.24);

– повна семипровідна (рис. 2.25);

– п'ятипровідна з керуванням потоком (рис. 2.26).

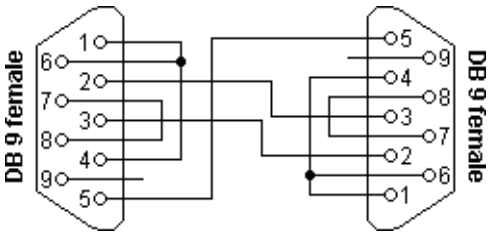


Рис. 2.24. Мінімальна схема з'єднання двох пристроїв при використанні інтерфейсу RS-232

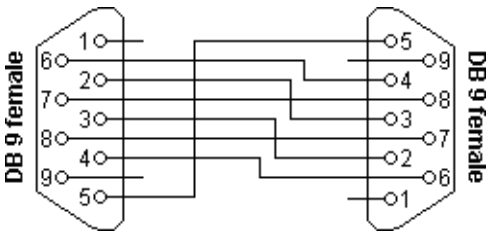


Рис. 2.25. Повна схема з'єднання двох пристроїв при використанні інтерфейсу RS-232

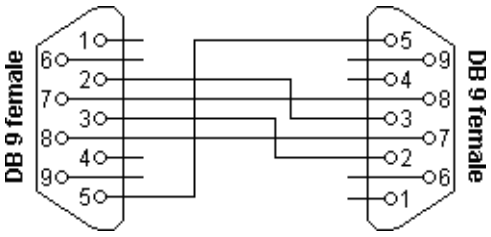


Рис. 2.26. П’ятипровідна схема з керуванням потоком

Зрозуміло, що рознімачі використовують при підключенні двох самостійних пристроїв, а у випадку, коли необхідно здійснити зв'язок між двома мікросхемами на одній платі, просто реалізують електричне з’єднання між відповідними виводами мікросхем (найчастіше за мінімальною схемою для економії місця на платі).

Згідно стандарту RS-232 повідомлення складається зі стартового біта, кількох біт даних, біта парності та стопового біта (рис. 2.27).

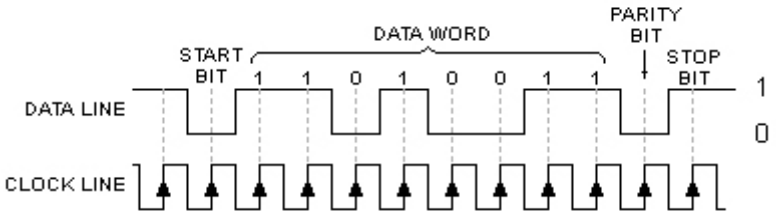


Рис. 2.27. Структура повідомлення згідно стандарту RS-232

Стартовий біт (start bit) означає початок передавання, зазвичай дорівнює 0. Дані (data bits) – 5, 6, 7 або 8 біт даних. Першим бітом є менш значущий біт. Стоповий біт (stop bit) означає завершення передавання повідомлення.

Біт парності (parity bit) – біт призначений для перевірки парності. Служить для виявлення помилок при передаванні даних. При використанні контролю парності підраховується кількість одиниць в групі бітів даних. Залежно від результату встановлюється біт парності. Приймальний пристрій також підраховує кількість одиниць і потім звіряє біт парності. Залежно від налаштування каналу може бути здійснений контроль парності (even) або

непарності (odd) кількості одиниць. При контролі на парність біти даних і біт парності завжди мають містити парну кількість одиниць. При контролі на непарність біти даних і біт парності завжди мають містити непарну кількість одиниць.

Доступні також ще три режими використання біта парності: Mark (означає, що пристрій завжди встановлює біт парності в 1), Space (означає, що пристрій завжди встановлює біт парності в 0) та NONE (біт парності не використовується).

Перевірка на парність – це найпростіший спосіб виявлення помилок. Він може визначити виникнення помилок в одному біті, але за наявності помилок в двох бітах вже не помітить помилок. Також такий контроль не відповідає на питання, який саме біт помилковий.

2.4.5. Асинхронні послідовні інтерфейси RS-422 та RS-485

Інтерфейс RS-422 дає змогу одночасно відправляти і приймати повідомлення по окремих лініях (повний дуплекс). Дуплекс забезпечується за рахунок того, що використовується одночасно два канали, один з яких працює на приймання, інший – на передавання (рис. 2.28).

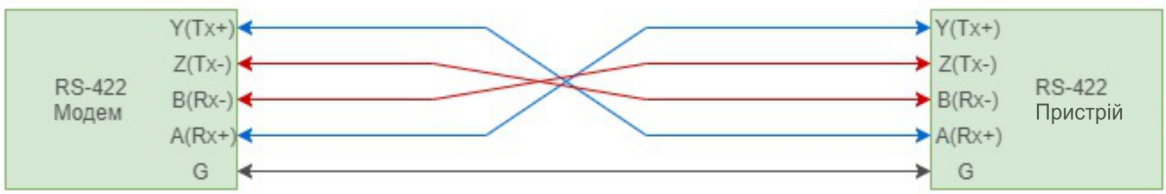


Рис. 2.28. Схема з’єднання двох пристроїв RS-422

Для передавання використовують диференційний сигнал, тобто різницю потенціалів між провідниками A і B, які представляють собою виту пару (два скручених проводи). Використовується принцип диференціального передавання одного сигналу. По проводу A йде вихідний сигнал, по проводу

В протифазний. Коли на одному проводі логічна 1, на іншому логічний 0 і навпаки (рис. 2.29). Цим досягається висока стійкість до синфазної перешкоди, що діє на обидва проводи однаково. Електромагнітна перешкода, проходячи через ділянку лінії зв'язку, наводить в кожному проводі однаковий потенціал, при цьому інформативна різниця потенціалів залишається без змін.

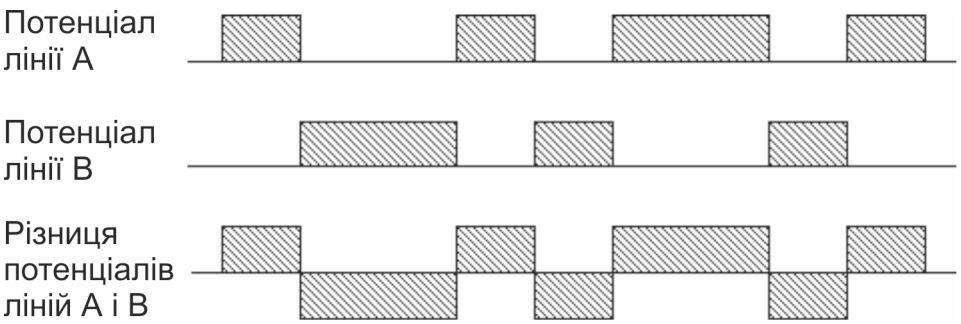


Рис. 2.29. Диференційний сигнал

Передавач має забезпечувати рівень сигналу 1,5 В при максимальному навантаженні (32 стандартних входи і 2 термінальних резистори) і не більше 6 В без навантаження. На стороні приймача мінімальний рівень сигналу має бути не менше 200 мВ.

Швидкість передачі даних в RS-422 залежить від відстані (табл. 2.4) і може змінюватися в межах від 9600 біт/с (1200 метрів) до 10 Мбіт/с (10 метрів).

Таблиця 2.4.

Залежність швидкості передачі даних від відстані в RS-422

Швидкість	**Відстань (довжина кабелю), м**
9600 біт/с	1200
375 кбіт/с	300
2,4 Мбіт/с	100
10 Мбіт/с	10

У мережі RS-422 може бути тільки один ведучий пристрій і до 10 ведених пристроїв. Кількість пристроїв, що підключаються до однієї лінії інтерфейсу, залежить від вхідного опору приймачів. Вхідний опір приймача за стандартом має бути більше або дорівнювати 12 кОм.

Для ослаблення паразитного сигналу, відбитого від кінця лінії, на віддаленому кінці лінії між провідниками витої пари включають резистор з номіналом, рівним хвильовому опору лінії (рис. 2.30).

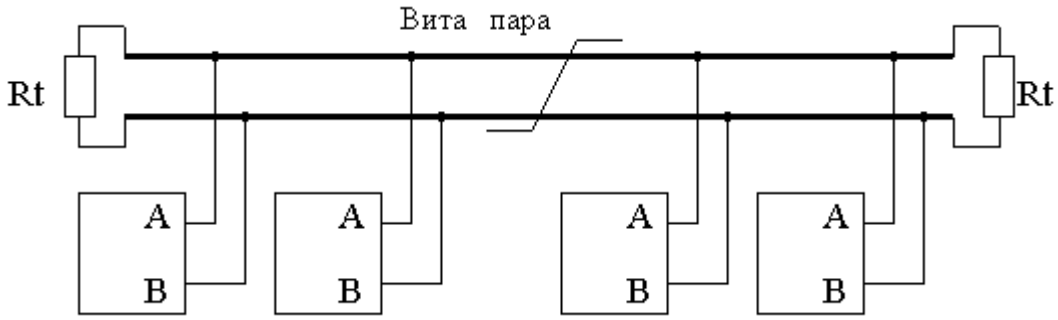


Рис. 2.30. Підключення кількох пристроїв RS-422 у спільну мережу

Лінія RS-422 являє собою 4 проводи для приймання-передавання даних (2 скручених проводи для передавання і 2 скручених проводи для приймання) і один загальний провід заземлення GND.

Скручування проводів між собою (вита пара) дозволяє зменшити вплив від наведених сигналів і завад, тому що наведення діє однаково на обидва проводи, а інформацію визначає різниця потенціалів між провідниками A і B однієї лінії.

Напруга на лініях передачі даних може знаходиться в діапазоні від –6 В до +6 В. Логічному нулю відповідає різниця між A і B більше +0,2 В. Логічній одиниці відповідає різниця між A і B менше –0,2 В.

Стандарт RS-422 не визначає конкретний тип рознімача, зазвичай це може бути клемна колодка або рознімач DB9. Розлініювання RS-422 залежить від виробника пристрою і вказується в документації на нього.

Інтерфейс RS-485 може існувати у двох варіантах: двопровідному і чотирипровідному. Двопровідний варіант використовується для

напівдуплексної передачі, коли інформація може передаватися в обох напрямках, але в різний час. Для дуплексної передачі використовують чотири лінії зв'язку: за двома інформація передається в одному напрямку, за двома іншими – в зворотному.

Недоліком чотирипровідної схеми є необхідність жорсткого визначення ведучого і ведених пристроїв на стадії проектування системи, в той час як у двопровідній схемі будь-який пристрій може бути як в ролі ведучого, так і веденого. Перевагою чотирипровідної схеми є можливість одночасного передавання і приймання даних, що буває необхідно при реалізації деяких складних протоколів обміну.

За електричними характеристиками та принципами передавання даних RS-422 повністю сумісний з RS-485 (табл.2.5).

Таблиця 2.5.

Технічні характеристики стандартів RS-422 та RS-485

Параметр	RS-422	RS-485
Кількість передавачів (Tx) і приймачів (Rx)	1 Tx, 10 Rx	32 Tx, 32 Rx
Максимальна довжина кабелю, м	1200	1200
Максимальна швидкість передачі даних, Мбіт/с	10	10
Діапазон напруг "1" передавача, В	+2...+10	+1,5...+6
Діапазон напруг "0" передавача, В	–2...–10	–1,5...–6
Максимальний струм КЗ передавача, мА	150	250
Допустимий опір навантаження передавача, Ом	100	54
Чутливість по входу приймача, мВ	± 200	± 200
Вхідний опір приймача, кОм	4	12
Діапазон напруг вхідного сигналу приймача, В	± 7	+7...+12
Рівень логічної одиниці приймача, мВ	> 200	> 200
Рівень логічного нуля приймача, мВ	< 200	< 200

Однак треба мати на увазі, що при використанні стандартних рознімачів електрична розлініювання контактів може відрізнятися, оскільки для RS-422 воно не стандартизоване і залежить від виробника.

Для з'єднання пристроїв RS-485 часто використовується 9-контактний рознімач DB-9. Розлініювання контактів для цього випадку вказане на рис. 2.31.

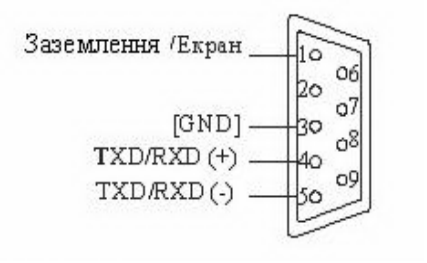


Рис. 2.31. Розлініювання контактів RS-485 для розетки рознімача DB-9

Більш детальну інформацію про інтерфейси RS-422 та RS-485 можна отримати за посиланням.

У таблиці 2.6 наведені найбільш суттєві відмінності між інтерфейсами RS-232, RS-422 та RS-485.

Таблиця 2.6.

Основні відмінності інтерфейсів RS-232, RS-422 та RS-485

Назва	**RS-232**	**RS-422**	**RS-485**
Тип передачі	повний дуплекс	повний дуплекс	напівдуплекс (2 проводи), повний дуплекс (4 проводи)
Максимальна дистанція	150 метрів при 9600 біт/с	1200 метрів при 9600 біт/с	1200 метрів при 9600 біт/с
Задіяні контакти	TXD, RXD, RTS, CTS, DTR, DSR, DCD, GND	TxA, TxB, RxA, RxB, GND	DataA, DataB, GND
Максимальна кількість ведених пристроїв	1	10	32 (з використанням повторювачів до 256)

2.4.6. Синхронний інтерфейс CAN

Інтерфейс CAN (Controller Area Network) реалізує протокол послідовного передавання даних, який ефективно підтримує розподілене керування в режимі реального часу з дуже високим рівнем безпеки. Область застосувань інтерфейсу CAN орієнтована на високошвидкісні мережі та спеціалізовані системи передавання даних.

У протоколі CAN відсутнє точне визначення фізичного рівня, тому для транспортування повідомлень CAN можуть використовуватися різні фізичні середовища. Проте, найчастіше використовують виту пару.

Решта рівнів моделі ISO в протоколі CAN взагалі не визначені, тому прикладна програма працює безпосередньо з регістрами модуля CAN. Взагалі кажучи, модуль CAN можна використовувати так само, як і UART, не реалізуючи складний стек протоколів, але оскільки протокол CAN використовується і в області автоматизації промислового виробництва, існує кілька специфікацій програмного забезпечення, що визначають, яким чином повідомлення CAN можуть бути використані для передавання даних між обладнанням різних виробників. Найбільш поширеними стандартами прикладного рівня є CANopen і DeviceNet. Єдине призначення цих стандартів – забезпечення взаємодії між обладнанням різних виробників. Якщо ви розробляєте власну закриту систему, у вас немає необхідності використовувати ці протоколи прикладного рівня і ви вільно можете реалізувати свій внутрішній протокол, як це і робить більшість розробників.

Протокол CAN має лінійну архітектуру і описаний в стандарті ISO 11898-1 та може бути коротко охарактеризований наступним чином:

– фізичний рівень використовує диференціальне передавання даних (найчастіше по витій парі);

– для управління доступом до шини використовується неруйнівне вирішення конфліктів (bit-wise);

– повідомлення мають малі розміри (здебільшого 8 байт даних) і захищені контрольної сумою;

– в повідомленнях відсутні явні адреси, натомість у кожному повідомленні міститься числове значення (пріоритет), яке визначає його черговість на шині, а також може служити ідентифікатором вмісту повідомлення;

– продумана схема оброблення помилок, яка забезпечує повторне передавання повідомлень, якщо вони не були отримані належним чином;

– є ефективні засоби для ізоляції збоїв і видалення збійних вузлів з шини.

Сам по собі протокол CAN визначає лише, як малі пакети даних можна безпечно перемістити з точки А в точку В за допомогою комунікаційного середовища. Всі інші характеристики визначаються протоколом більш високого рівня.

Максимальна протяжність кабелю складає 40 метрів за максимальної швидкості передавання в 1 Мбіт/с. Однак при знижені швидкості передавання даних до 50 кбіт/с мережа може мати довжину до 1000 метрів (табл. 2.7).

Таблиця 2.7.

Залежність швидкості передавання даних від відстані для CAN

Швидкість	**Відстань (довжина кабелю), м**
1 Мбіт/с	40
500 кбіт/с	100
125 кбіт/с	500
10 кбіт/с	5000

Принцип підключення користувачів до шинної лінії за допомогою різниці напруг є аналогічним вже розглянутим вище принципам функціонування інтерфейсу RS-485.

Для рознімачів CAN стандартів не існує. Зазвичай, кожен протокол більш високого рівня описує один або кілька бажаних типів рознімачів. Основні типи такі:

– 9-ти контактний DSUB (він же DB-9, рис. 2.32);

– 5-ти контактний Mini-C та/або Micro-C (рис. 2.33);

– 6-ти контактний рознімач Deutsch.

а *б*

Рис. 2.32. Рознімач DB-9 штекер (а) та розліниювання контактів для CAN-інтерфейсу (б)

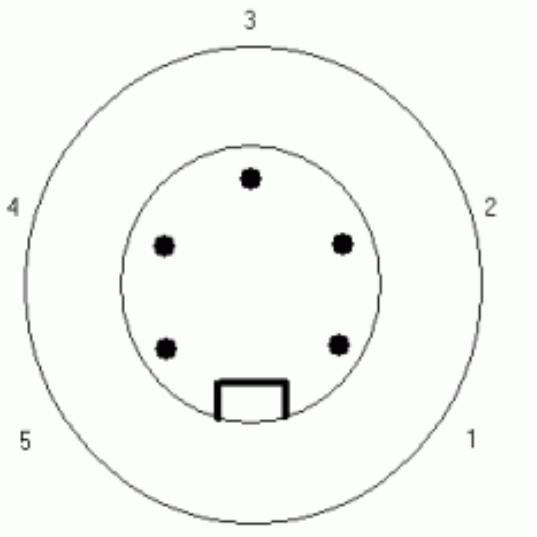


Рис. 2.33. Рознімач 5-ти контактний Mini-C

Рекомендоване розліниювання контактів наведено в таблиці 2.8. Рознімачі з більшою кількістю контактів можуть використовувати додаткові лінії (див. рис. 2.32)

Таблиця 2.8.

Рекомендоване розліниювання контактів рознімачів у CAN

№ контакту	Назва контакту	Призначення контакту
1.	----------	Резерв
2.	CAN_L	Лінія шини CAN_L (домінантна низька)
3.	CAN_GND	Лінія заземлення шини
4.	----------	Резерв
5.	CAN_SHLD	Опціонально екран

Структура повідомлення пристроїв CAN показана на рис. 2.34.

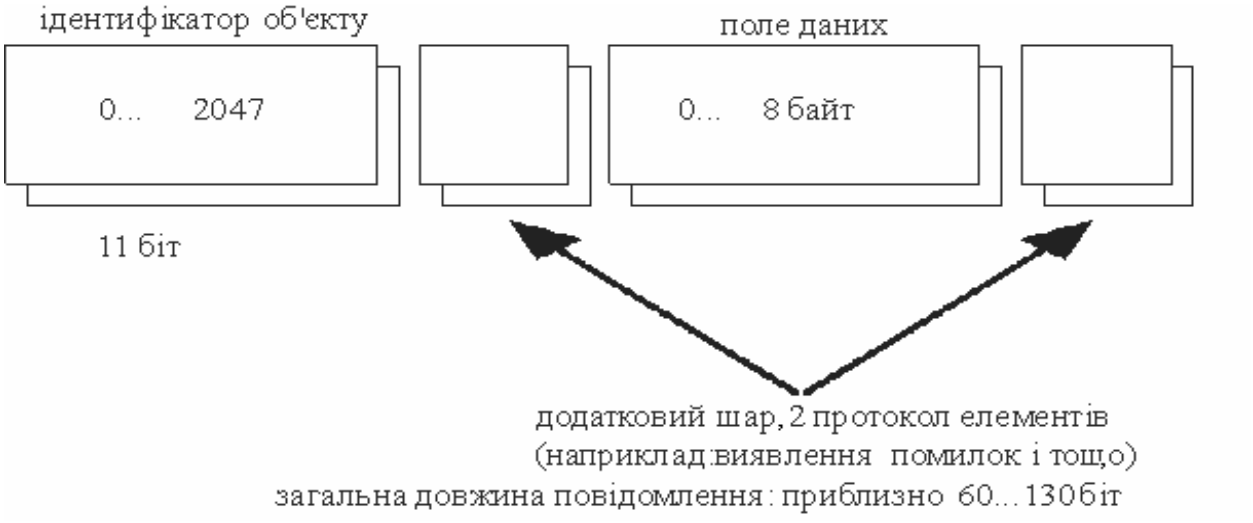


Рис. 2.34. Структура повідомлення протоколу CAN

Поле даних може мати різну довжину від 0 до 64 бітів і мати інформацію, призначену для передавання. Поле «Ідентифікатор об’єкта» (11 біт для стандарту CAN 2.0A або 29 біт для стандарту CAN 2.0B) має декілька функцій. Воно служить як мітка для повідомлення. Через це воно і використовується для регулювання доступу до шини, тобто для ліквідації колізій. Дане поле надає сигналу певний пріоритет. Останній, в свою чергу, тим вище, чим менше сигналів логічної 1 має поле. Таким чином, найвищий пріоритет у сигналі з ідентифікатором об’єкта 0.

У CAN існують чотири типи повідомлень:

– кадр даних (data frame) – передає дані;

– кадр віддаленого запиту (remote frame) – служить для запиту на передавання кадру даних з тим же ідентифікатором;

– кадр перевантаження (overload frame) – забезпечує проміжок між кадрами даних або запиту;

– кадр помилки (error frame) – передається вузлом, який виявив в мережі помилку.

Для абстрагування від середовища передавання специфікація CAN уникає описувати двійкові значення як «0» і «1». Замість цього застосовуються терміни «рецесивний» (двійкове значення «1») і «домінантний» (двійкове значення «0»), при цьому мається на увазі, що при передаванні одним вузлом мережі рецесивного біта, а іншим домінантного, прийнятий буде домінантний біт. Наприклад, при реалізації фізичного рівня на радіоканалі відсутність сигналу означає рецесивний біт, а наявність – домінантний; тоді як в типовій реалізації провідної мережі рецесивному біту відповідає наявність сигналу, а домінантному, відповідно, відсутність. Стандарт мережі вимагає від «фізичного рівня», фактично, єдиної умови: щоб домінантний біт міг придушити рецесивний, але не навпаки. Наприклад, в оптичному волокні домінантному біту відповідає «світло», а рецесивному – «темрява». В електричному проводі може бути так: рецесивний стан – висока напруга на лінії (від джерела з великим внутрішнім опором), домінантний – низька напруга (домінантний вузол мережі «підтягує» лінію на землю). Якщо лінія знаходиться в рецесивному стані, перевести її в домінантне може будь-який вузол мережі (включивши світло в оптичному волокні або закоротивши високу напругу).

CAN – це мережа з кількома ведучими пристроями, в якій сигнали різного пріоритету передаються широкомовно (тобто безадресово). Її ключовий концепт – це спосіб розподілу доступу до шини, заснований на

попередженні конфліктів. Механізм попередження конфліктів аналогічний механізму, який використовується при роботі інтерфейсу I^2C.

Якщо шина вільна, будь-яка станція може розпочинати передавання сигналу, в якому вона надсилає заголовок повідомлення. Він містить ідентифікатор, спеціально призначений для даного сигналу.

У разі одночасного передавання кадрів двома і більше вузлами проходить арбітраж доступу: під час передавання ідентифікатора вузол одночасно перевіряє стан шини. Якщо при передаванні рецесивного біта приймається домінантний, то вважається, що інший вузол передає повідомлення з великим пріоритетом, і передавання відкладається до звільнення шини. Таким чином, на відміну, наприклад, від Ethernet в CAN не відбувається непродуктивної втрати пропускної здатності каналу при колізіях. Ціна цього рішення – можливість того, що повідомлення з низьким пріоритетом ніколи не будуть передані.

CAN використовує кілька механізмів контролю та запобігання виникненню помилок, а саме:

– контроль передавання: в процесі передавання бітові рівні в мережі порівнюються з переданими бітами;

– додавання доповнюючих бітів (bit stuffing): після передавання п'яти однакових бітів підряд автоматично передається біт протилежного значення. Таким чином кодуються всі поля кадрів даних або запиту, крім розмежувача контрольної суми, проміжку підтвердження і EOF.

– контрольна сума: передавач обчислює її і додає в переданий кадр, приймач розраховує контрольну суму прийнятого кадру в реальному часі (одночасно з передавачем), порівнює з сумою в самому кадрі і, в разі збігу, передає домінантний біт в проміжку підтвердження;

– контроль значень полів при прийманні.

Розробники оцінюють ймовірність невиявлення помилки передавання як $4{,}7 \times 10^{-11}$.

Базовій специфікації CAN бракує багатьох можливостей, необхідних в реальних системах: передавання даних довше 8 байт, автоматичного розподілу ідентифікаторів між вузлами, однакового управління пристроями різних типів і виробників. Тому незабаром після появи CAN на ринку почали розроблятися протоколи високого рівня для нього. У низку поширених на даний момент протоколів входять: CANopen, DeviceNet, CAN Kingdom, J1939, SDS, NMEA-2000 (морський транспорт), ARINC-825 (авіація) (нім.), UAVCAN (робототехніка і літальні апарати). Більш детальну інформацію можна знайти в описі протоколів CAN.

2.4.7. Інтерфейс USB

USB (Universal Serial Bus) – універсальна послідовна шина. Даний інтерфейс призначений для підключення різноманітних периферійних пристроїв до мікропроцесорних систем.

Стандарт USB було розроблено за участі таких компаній як: «Compaq», «Digital Equipment», IBM, Intel, «Microsoft», NEC і «Northern Telecom». У 1995 році було анонсовано кілька версій протоколу, а вже у 1996 році було розпочато випуск перших комп'ютерів з портами USB та периферійних пристроїв для них.

Натепер існує кілька різних специфікацій USB. Найчастіше використовують USB 2.0 та USB 3.0. Стандарт USB 1.0, який отримав широке поширення, був представлений у січні 1996 року. Версія 1.1 тепер майже не використовується через надто низьку швидкість передавання даних (12 Мбіт/сек), тому застосовується тільки для сумісності.

Стандарт USB 2.0, був представлений у квітні 2000 року. Як і в разі специфікацій USB 1.0 і USB 1.1, в специфікації USB 2.0 для підключення периферійних пристроїв використовується кабель, що складається з двох пар проводів: одна кручена пара проводів для приймання і передавання даних, а інша – для живлення периферійного пристрою (рис. 2.35).

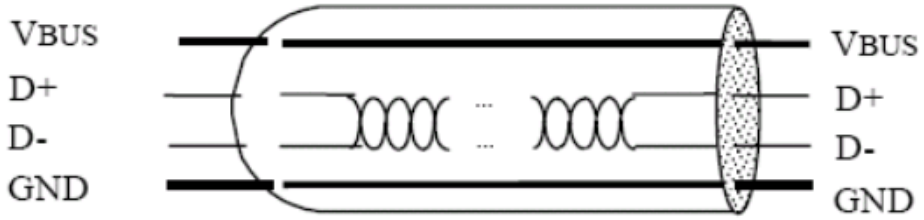


Рис. 2.35. USB кабель: Vbus – напруга живлення шини; D+ – лінія даних умовно названа додатньою; D- – лінія даних умовно названа від'ємною; GND – заземлюючий провідник

Принципова схема з'єднання USB-пристроїв показана на рис. 2.36.

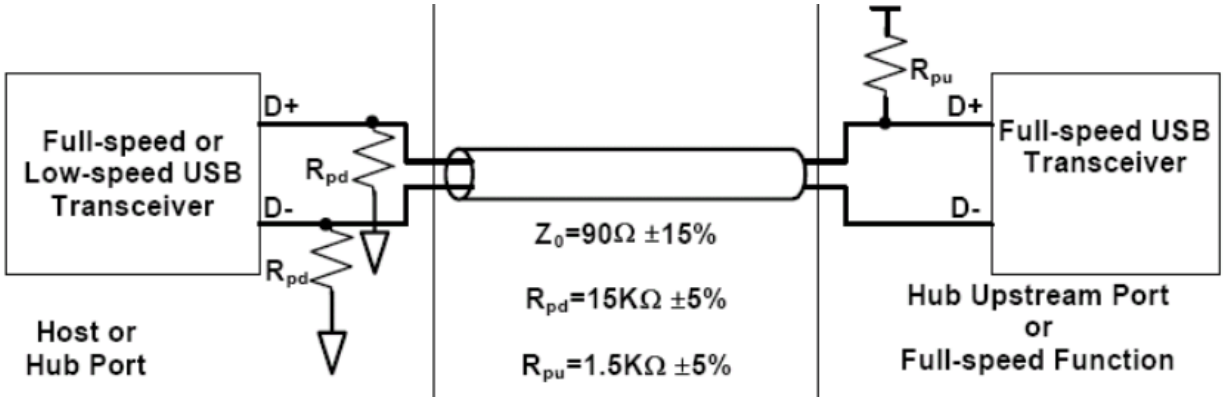


Рис. 2.36. Принципова схема з'єднання USB-пристроїв

Якщо на лініях D+ і D- часто змінюються рівні сигналів, значить, відбувається передавання даних. Частота зміни дорівнює швидкості передавання. Якщо напруга на лінії D+ більша за напругу на лінії D-, то передається логічна одиниця. Якщо напруга на лінії D+ менша за напругу на лінії D-, то передається логічний нуль. Якщо на лініях певний час відсутні зміни, значить, канал зв'язку знаходиться в стані спокою (Idle), скидання (Reset) тощо.

Напруга живлення по шині USB дорівнює 5 В при силі струму до 500 мА. Цього, недостатньо для периферійних пристроїв із високим енергоспоживанням таких, наприклад, як принтери. Тому вони комплектуються власними блоками живлення.

Пристрої USB 2.0 підтримують три режими роботи:

– Low-speed, 10...1500 кбіт/с (клавіатури, миші, джойстики, геймпади);

– Full-speed, 0,5...12 Мбіт/с (аудіо-, відеотехніка);

– High-speed, 25...480 Мбіт/с (відеопристрої, пристрої зберігання інформації).

Специфікація USB 3.0 з'явилася у 2008 році. У специфікації USB 3.0 рознімачі і кабелі сумісні з USB 2.0. Специфікація USB 3.0 підвищує максимальну швидкість передавання інформації до 5 Гбіт/с, що більше швидкості передавання даних пристроїв USB 2.0 (максимально 480 Мбіт/с.)

31 липня 2013 року USB 3.0 Promoter Group оголосила про прийняття специфікації наступного інтерфейсу – USB 3.1, швидкість передавання якого може сягати 10 Гбіт/с.

22 вересня 2017 некомерційна організація USB Implementers Forum (USB-IF) опублікувала специфікацію стандарту USB 3.2, яка є заключною ревізією для USB 3.x. Нова специфікація передбачає подвоєння максимально можливої швидкості передавання даних порівняно з USB 3.1 Gen 2 – з 10 до 20 Гбіт/с за рахунок використання двох ліній на 5 Гбіт/с або 10 Гбіт/с (тільки для рознімача USB Type-C за рахунок його двосторонніх контактів і використання дублюючих виводів). Були внесені поправки в роботу хост адаптерів для плавного переходу від двоканального режиму дублюючих виводів до одноканального режиму. Сучасні кабелі USB Type-C, що є в наявності, вже підтримують такий «дволінійний» режим.

Специфікації USB 3.2 замінюють стандарти USB 3.0 і USB 3.1; пристрої, які задовольняють дані специфікації, підтримують наступні швидкості:

– SuperSpeed USB (USB 3.2 Gen 1) швидкість до 5 Гбіт/с;

– SuperSpeed+ USB 10Gbps (USB 3.2 Gen 2) швидкість до 10 Гбіт/с;

– SuperSpeed+ USB 20Gbps (USB 3.2 Gen 2x2) зі швидкістю до 20 Гбіт/с.

У специфікаціях також прописаний варіант з двома лініями, кожна з яких працює по протоколу USB 3.0: SuperSpeed+ USB 10Gbps (USB 3.2 Gen

1x2) зі швидкістю до 10 Гбіт/с по двох лініях, кожна з яких відповідає USB 3.1 Gen 1.

Специфікація USB4 передбачає використання нового базового протоколу. Максимальна швидкість для даної специфікації сягає 40 Гбіт/с. При цьому зберігається обернена сумісність із USB 3.2, USB 2.0.

Кабелі USB орієнтовані, тобто мають фізично різні наконечники «до пристрою» (Тип B) і «до хосту» (Тип A) (рис. 2.37). Розліниювання контактів рознімачів стандартів USB 1.0-2.0 наведено у таблицях 2.9, 2.10.

Звичайний	**Mini**	**Micro**
Тип A 4×12 мм 	3×7 мм 	2×7 мм 
Тип B 7×8 мм 	3×7 мм 	2×7 мм 

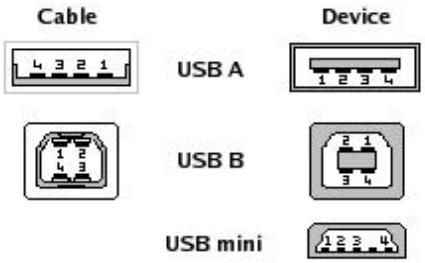


Рис. *2.37.* Типи рознімачів стандартів USB 1.0-2.0

Таблиця 2.9.

Розліниювання звичайних рознімачів стандартів USB 1.0-2.0

Контакт	**Сигнал**	**Колір провідника**
1	Vbus (VCC)	червоний
2	D-	білий
3	D+	зелений
4	GND	чорний

Таблиця 2.10.

Розліниювання рознімачів Mini/Micro-A,B (USB 1.0-2.0)

Контакт	Сигнал	Колір провідника
1	Vbus = +5B (VCC)	червоний
2	Лінія даних (D−)	білий
3	Лінія даних (D+)	зелений
4	On-The-Go ID визначає кінець кабелю	не має проводу
5	GND	чорний

Рознімач типу USB 3.0 Micro-B зображено на рис. 2.38. Розліниювання контактів даного рознімача наведено у таблиці 2.11.

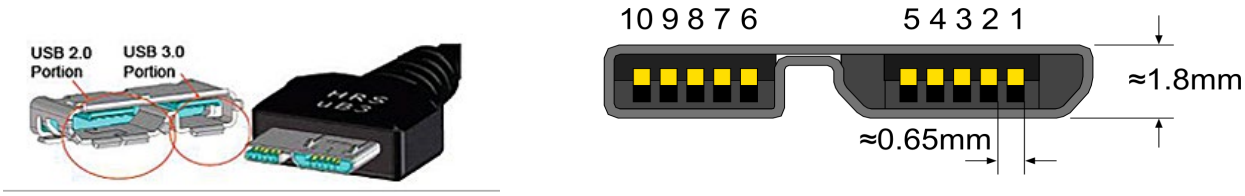


Рис. 2.38. Рознімач типу USB 3.0 Micro-B

Таблиця 2.11.

Розліниювання контактів рознімача USB 3.0 Micro-B

Контакт	Сигнал	Колір провідника	Контакт	Сигнал	Колір провідника
1	Vbus = +5 B (VCC)	Червоний	6	USB 3.0 лінія передачі сигналу (-)	Синій
2	USB 2.0 диференціальна (вита) пара (D-)	Білий	7	USB 3.0 лінія передачі сигналу (+)	Жовтий
3	USB 2.0 диференціальна (вита) пара (D+)	Зелений	8	GND_DRAIN	Білий
4	USB OTG ID (ідентифікація лінії)	Білий	9	USB 3.0 лінія приймання сигналу (-)	Фіолетовий
5	GND	Чорний	10	USB 3.0 лінія приймання сигналу (+)	Оранжевий

Рознімач типу USB 3.1 Type-C зображено на рис. 2.39. Розлініювання контактів рознімача типу USB 3.1 Type-C наведено у таблиці 2.12.

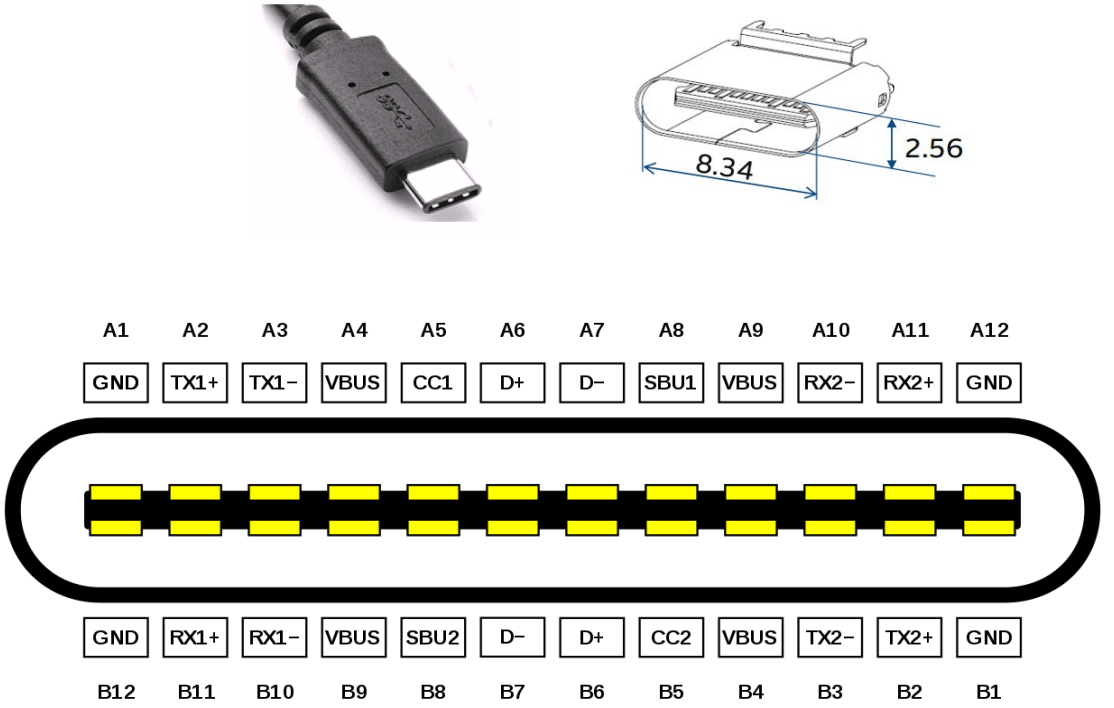


Рис. 2.39. Рознімач типу USB 3.1 Type-C

По обидва боки кабелю розташовуються ідентичні 24-контактні компактні рознімачі, близькі за розмірами до рознімачів мікро-B стандарту USB 2.0. Розміри рознімача – 8,4 мм на 2,6 мм. Рознімач надає 4 пари контактів для живлення і заземлення, дві диференціальні пари D+/D- для передавання даних на швидкостях менше SuperSpeed (в кабелях Type-C підключена тільки одна з пар), чотири диференціальні пари для передачі високошвидкісних сигналів SuperSpeed, два допоміжних контакти (sideband), два контакти конфігурації для визначення орієнтації кабелю, виділений канал конфігураційних даних і контакт живлення +5 В для активних кабелів. Підключення раніше випущених пристроїв до комп'ютерів, оснащених рознімачем USB Type-C, потребує кабелю або адаптера, що мають рознімач типу A або типу B на одному кінці та штекер USB Type-C на іншому кінці. Стандартом не допускаються адаптери з рознімачем USB Type-C, оскільки їх

використання могло б створити «безліч неправильних і потенційно небезпечних» комбінацій кабелів.

Таблиця 2.12.

Розліниювання контактів рознімача USB 3.1 Type-C

Контакт	Назва	Опис	Контакт	Назва	Опис
A1	GND	заземлення	B12	GND	заземлення
A2	SSTXp1	диф. пара №1 SuperSpeed, передача (+)	B11	SSRXp1	диф. пара №2 SuperSpeed, прийом (+)
A3	SSTXn1	диф. пара №1 SuperSpeed, передача (-)	B10	SSRXn1	диф. пара №2 SuperSpeed, прийом (-)
A4	V_{BUS}	живлення	B9	V_{BUS}	живлення
A5	CC1	канал конфігурації	B8	SBU2	Sideband № 2 (SBU)
A6	Dp1	диф. пара не SuperSpeed, положення 1 (+)	B7	Dn2	диф. пара не SuperSpeed, положення 2 (-)
A7	Dn1	диф. пара не SuperSpeed, положення 1 (-)	B6	Dp2	диф. пара не SuperSpeed, положення 2 (+)
A8	SBU1	Sideband № 1 (SBU)	B5	CC2	канал конфігурації
A9	V_{BUS}	живлення	B4	V_{BUS}	живлення
A10	SSRXn2	диф. пара №4 SuperSpeed, прийом (-)	B3	B3	диф. пара №3 SuperSpeed, передача (-)
A11	SSRXp2	диф. пара №4 SuperSpeed, прийом (+)	B2	SSTXp2	диф. пара №3 SuperSpeed, передача (+)
A12	GND	заземлення	B1	GND	заземлення

Кабелі USB 3.1 з двома штекерами Type-C на кінцях мають повністю відповідати специфікації – містити всі необхідні провідники, мають бути активними, включати в себе чіп електронної ідентифікації, який перераховує ідентифікатори функцій залежно від конфігурації каналу і повідомлення, що визначаються вендором (VDM) із специфікації USB Power Delivery 2.0. Пристрої з рознімачами USB Type-C можуть підтримувати шини живлення із струмом в 1,5 або 3 ампери при напрузі 5 вольт на додачу до основного живлення. Джерела живлення мають повідомляти про можливість надання збільшених струмів через конфігураційний канал або повністю підтримувати специфікацію USB Power Delivery через конфігураційний контакт (кодування

BMC) або більш старі сигнали, кодовані як BFSK через контакт V_{BUS}. Кабелі USB 2.0, які не підтримують шину SuperSpeed, можуть не містити чіп електронної ідентифікації, якщо тільки вони не можуть передавати струм 5 ампер.

Швидкість передавання даних залежить від довжини кабелю – чим довше, тим повільніше:

– швидкість 10 Гбіт/с (Gen 2) при довжині кабелю до 1 метру;

– швидкість до 5 Гбіт/с (Gen 1) при довжині кабелю до 2 м.

Усі USB-пристрої в системі з'єднуються за топологією, яка називається багатоярусна зірка (рис. 2.40).

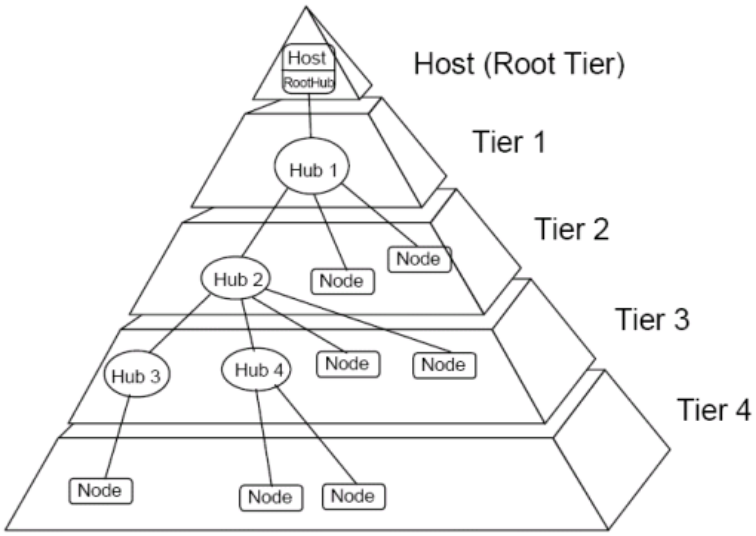


Рис. 2.40. Топологія підключення USB-пристроїв

Вся інформація передається кадрами, які відправляються через рівні проміжки часу. У свою чергу кожен кадр складається з транзакцій (рис. 2.41).

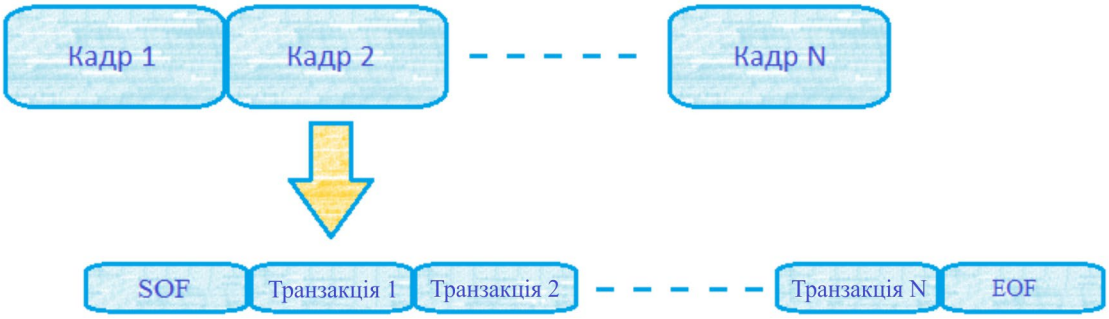


Рис. 2.41. Структура кадру USB

Кожен кадр включає в себе пакет SOF (Start Of Frame), потім слідують транзакції для різних кінцевих точок, завершується все пакетом EOF (End Of Frame). Пакет EOF – це не зовсім пакет в звичному розумінні цього слова – це інтервал часу, протягом якого обмін даними заборонений. Кожна транзакція складається з пакетів і має структуру зображену на рис. 2.42.

Рис. 2.42. Структура транзакції USB

Перший пакет (Token) містить в собі інформацію про адресу пристрою USB, а також про номер кінцевої точки, якій призначена ця транзакція, а також інформацію про тип транзакції. Пакет Data відповідно містить дані, які необхідно передати, а пакет Status призначений для перевірки успішності передавання даних.

Пакет Token має структуру зображену на рис. 2.43.

Рис. 2.43. Структура пакету Token

Пакети Token в залежності від значення поля PID бувають чотирьох типів:

- In (PID = 1001);
- Out (PID = 0001);
- Setup (PID = 1001);
- Start of Frame (PID = 0101).

Пакет In сповіщає кінцевий USB-пристрій про те, що host-пристрій готовий прийняти від нього інформацію. Пакет Out, навпаки, сигналізує про готовність host-пристрою передати інформацією. Пакет Setup містить інформацію необхідну для керування процесом передавання, а пакет Start Of

Frame використовується для того, щоб ініціювати початок кадру. Поле PID включає в себе 4 біта, але при передаванні вони доповнюються ще 4-ма бітами, які отримують шляхом інвертування перших 4-ох біт.

Поле Sync призначено для синхронізації приймача і передавача. Поля Address і Endpoint містять адресу USB пристрою і номер кінцевої точки, якій призначена транзакція. Поле CRC містить контрольну суму. Поле EOP ідентифікує кінець пакету.

Пакет Data має схожу структуру, але замість полів Address і Endpoint містить поле Data, що містить інформацію, яку необхідно передати (рис. 2.44).

Рис. 2.44. Структура пакету Data

Пакет Status складається всього з 3-ох полів (рис. 2.45).

Рис. 2.45. Структура пакету Status

Поле PID в даному пакеті може приймати одне з двох значень:

– інформація прийнята коректно – PID = 0010;

– виникла помилка при прийманні інформації – PID = 1010.

Пакет Start of Frame, який ініціює початок кадру, зображено на рис. 2.46. Він відрізняється від пакету Data лише одним полем. Замість поля даних Data він включає в себе поле Frame, яке містить номер переданого кадру.

Рис. 2.46. Структура пакету Start of Frame

Розглянемо для прикладу процес запису даних в USB-пристрій. Тобто розглянемо приклад структури кадру запису. Кадр складається з транзакцій і має такий структуру зображену на рис. 2.47.

Рис. 2.47. Структура кадру запису інформації

Першою пересилається транзакція Setup даного кадру, яка містить інформацію відповідно до структури зображеної на рис. 2.48.

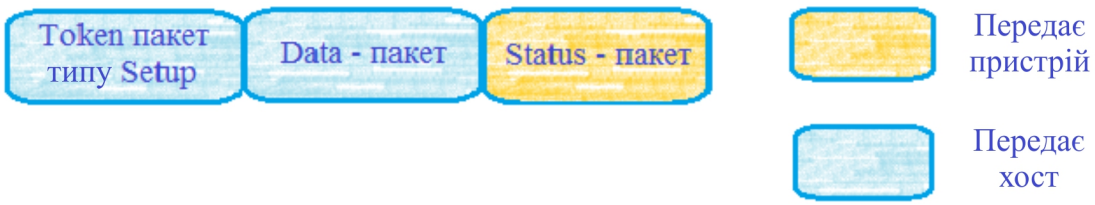


Рис. 2.48. Структура транзакції Setup кадру запису інформації

За нею слідують транзакції OUT (рис. 2.49), які містять пакет даних від host-пристрою і пакет Status від отримувача інформації.

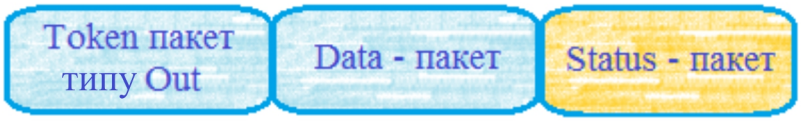


Рис. 2.49. Структура транзакції OUT кадру запису інформації

Відповідно при зчитуванні даних з host-пристрою кадр буде мати структуру зображену на рис. 2.50.

Рис. 2.50. Структура кадру зчитування інформації

При цьому транзакція SETUP залишається такою ж, як і у кадрі передавання інформації. Зміниться лише значення поля PID у пакеті Token. Також замість транзакцій OUT буде передано транзакції IN (рис. 2.51).

Рис. 2.51. Структура транзакції IN кадру запису інформації

Загалом протокол передавання даних по шині USB достатньо складний і виходить за рамки даного курсу, тому на цьому завершимо розгляд шини USB. Більш детальну інформацію можна подивитися в специфікаціях шини USB.

2.5. Бездротові інтерфейси

Останнім часом все більшого поширення і значення набули бездротові сенсорні мережі. Це привело до розробки великої кількості бездротових стандартів. Відповідно з'явилася низка мікроконтролерів з підтримкою бездротових інтерфейсів, які стали невід'ємною частиною мікропроцесорних систем. Грамотне використання таких мікроконтролерів потребує досконалого знання бездротових інтерфейсів. Найбільш поширені з них представлені у таблиця 2.13.

Таблиця 2.13.

Бездротові стандарти

Технологія	Стандарт	Використання	Швидкість	Радіус дії	Частоти
Wi-Fi	802.11 a,c	WLAN	до 1 Гбіт/с	до 300 м	5 ГГц
Wi-Fi	802.11 b	WLAN	до 11 Гбіт/с	до 300 м	2,4 ГГц
Wi-Fi	802.11 g	WLAN	до 54 Гбіт/с	до 300 м	2,4 ГГц
Wi-Fi	802.11 n	WLAN	до 300 Мбіт/с (перспектива до 600 Мбіт/с)	до 300 м	2,4-2,5 ГГц 5,0 ГГц
WiMax	802.16 d	WMAN	до 75 Мбіт/с	25-80 км	1,5-11 ГГц
WiMax	802.16 e	Mobile WMAN	до 40 Мбіт/с	1-5 км	2,3-13,6 ГГц
WiMax 2	802.16 m	WMAN, Mobile WMAN	1 Гбіт/с-WMAN, 100 Мбіт/с-(Mobile WMAN)	120-150 км	до 11 ГГц
WiMax-3	802.16n	WMAN, Mobile WMAN	10 Гбіт/с-WMAN, 1 Гбіт/с- (Mobile WMAN)	стандарт в розробці	стандарт в розробці
Bluetooth v. 1.1	802.15.1	WPAN	до 1 Мбіт/с	до 10 м	2,4 ГГц
Bluetooth v. 2.0	802.15.3	WPAN	до 2,1 Мбіт/с	до 100 м	2,4 ГГц
Bluetooth v. 3.0	802.11	WPAN	до 24 Мбіт/с	до 100 м	2,4 ГГц
Bluetooth v. 4.0	802.11	WPAN	1 Мбіт/с	до 10 м	2,4 ГГц
Bluetooth v. 4.1	802.11 n	WPAN	1 Мбіт/с	до 20 м	2,45 ГГц 2,6 ГГц
Bluetooth v. 4.2	802.11	WPAN	1 Мбіт/с	до 20 м	2,4 ГГц
Bluetooth v. 5.0	802.11	WPAN	2 Мбіт/с	до 40 м	2,4 ГГц
UWB	802.15.3a	WPAN	110-480 Мбіт/с	до 10 м	7,5 ГГц
ZigBee	802.15.4	WPAN	20-250 кбіт/с	до 100 м	2,4 ГГц-16 каналів 915 МГц-10 каналів 868 МГц-1 канал
Інфра-червона лінія зв’язку	IrDa	WPAN	до 15 Мбіт/с	5-5- см Односторонній зв'язок до 10 м	інфра-червоні промені

З таблиці 2.13 видно, що кількість різноманітних бездротових стандартів достатньо велика, що викликано великим різноманіттям умов їх використання.

Бездротові мережі, також як і провідні, прийнято класифікувати за територіальною ознакою. Зазвичай виділяють чотири типи:

– WWAN (Wireless Wide Area Network);

– WMAN (Wireless Metropolitan Area Network);

– WLAN (Wireless Local Area Network);

– WPAN (Wireless Personal Area Network).

WWAN (бездротова глобальна (регіональна) мережа) – це територіально розподілена мережа, яка може покривати значний географічний простір (регіон, країну, низку країн).

WMAN (бездротова міська мережа) – високошвидкісна комунікаційна мережа, яка охоплює територію діаметром до 50 км, і за масштабом є проміжною між WLAN і глобальною (WWAN) мережею.

WLAN (бездротова локальна мережа) – мережа, призначена для обслуговування невеликих територій (будівель, підприємств), де можна обійтися відносно короткими лініями зв'язку (до 500 м).

WPAN (бездротова персональна мережа) – використовується всередині особистого робочого простору (Personal Operating Space, POS). Під особистим робочим простором мають на увазі простір, який оточує користувача (радіусом до 10 метрів). Одною з характерних ознак є використання передавачів з дуже малим енергоспоживанням які можуть передавати інформацію від портативних комп'ютерів, стільникових телефонів, відеокамер і різних побутових пристроїв.

Технологія Wi-Fi (Wireless Fidelity) призначена для побудови бездротових локальних мереж, організації точок публічного доступу до Інтернету (Hot Spots). Технологія базується на стандартах IEEE 802.11. Це сімейство є базовим стандартом WLAN і підтримує передавання даних зі швидкостями від 1 до 2 Мбіт/с і працює на фізичному і канальному рівні

моделі OSI. На фізичному рівні визначені два широкосмугових радіочастотних методи передавання і один – в інфрачервоному діапазоні. Радіочастотні методи працюють в ISM-діапазоні 2,4 ГГц і зазвичай використовують смугу 83 МГц в діапазоні 2,400...2,483 ГГц. Стандарт 802.11 використовує технологію розширення спектру сигналу прямої послідовності (Direct Sequence Spread Spectrum, DSSS) і технологію розширення спектру сигналу стрибкоподібною перебудовою частоти (Frequency Hopping Spread Spectrum, FHSS). Для модуляції сигналу FHSS використовує технологію Frequency Shift Keying (FSK). При роботі на швидкості 1 Мбіт/с використовується FSK-модуляція по Гауссу другого рівня, а при роботі на швидкості 2 Мбіт/с – четвертого рівня. Метод DSSS використовує технологію модуляції Phase Shift Keying (PSK). При цьому на швидкості 1 Мбіт/с використовується диференціальна двійкова PSK, а на швидкості 2 Мбіт/с – диференціальна квадратична PSK-модуляція. Заголовки фізичного рівня завжди передаються на швидкості 1 Мбіт/с , в той час як дані можуть передаватися зі швидкостями 1 і 2 Мбіт/с.

Розглянемо группу стандартів Wi-Fi IEEE 802.11.

IEEE 802.11a

Високошвидкісний стандарт WLAN для частоти 5 ГГц. Підтримує швидкості передачі даних 6, 9, 12, 18, 24, 36, 48, 54 Мбіт/с. Використовувана радіочастотна технологія: OFDM. Кодування: Convolution Coding. Модуляції: BPSK, QPSK, 16-QAM, 64-QAM. До недоліків 802.11a відносять достатньо високу споживану потужність радіопередавачів для частот 5 ГГц.

IEEE 802.11b

Стандарт WLAN для частоти 2,4 ГГц був прийнятий у 1999 р. Підтримує швидкості передачі даних 1, 2, 5,5, 11 Мбіт/с. В якості базової радіотехнології в ньому використовується метод DSSS з 8-розрядними

послідовностями Уолша. Кодування: Barker 11 і CCK. Модуляції: DBPSK і DQPSK.

Оскільки обладнання, що працює на максимальній швидкості 11 Мбіт/с, має менший радіус дії, ніж на більш низьких швидкостях, то стандартом 802.11b передбачено автоматичне зниження швидкості при погіршенні якості сигналу.

IEEE 802.11d

Стандарт Wi-Fi IEEE 802.11d дає змогу регулювати смуги частот в пристроях різних виробників за допомогою спеціальних опцій, введених в протоколи управління доступом до середовища передачі. Введений для адаптації протоколу 802.11 до законодавства конкретних країн щодо умов розподілу частотного діапазону.

IEEE 802.11e

Дозволяє розширити функціональні можливості стандартів IEEE 802.11a, IEEE 802.11b за рахунок застосування методів забезпечення якості обслуговування (QoS). Визначаєє класи якості для передачі різних типів медіафайлів.

IEEE 802.11f

Описує порядок зв'язку між рівнозначними точками доступу, що необхідно для побудови розподілених бездротових мереж передавання даних. Спрямований на уніфікацію параметрів точок доступу стандарту Wi-Fi різних виробників. Стандарт дає змогу користувачеві працювати з різними мережами при переміщенні між зонами дії окремих мереж.

IEEE 802.11g

Встановлює додаткову техніку модуляції для частоти 2,4 ГГц. Призначений для забезпечення швидкостей передачі даних до 54 Мбіт/с.

Описує швидкості передачі даних еквівалентні 802.11a. Протокол сумісний з 802.11b. Використовувані радіочастотні технології: DSSS і OFDM. Кодування: Barker 11 і CCK. Модуляції: DBPSK і DQPSK. Підтримує швидкості передачі даних 1, 2, 5,5, 11 Мбіт/с при використанні технології DSSS і 6, 9, 12, 18, 24, 36, 48, 54 Мбіт/с при використанні технології OFDM.

IEEE 802.11h

В даному стандарті рівні MAC і PHY доповнюються алгоритмами оптимального вибору частот. Передбачається, що їх рішення базується на протоколах DFS (Dynamic Frequency Selection) і TCP (Transmit Power Control). Процедура динамічного регулювання потужності для 802.11h передбачає її зміну залежно від рівня перешкод з наступним переходом на інший радіоканал в тому випадку, якщо підвищенням потужності не вдається забезпечити необхідне відношення сигнал/шум.

Використовується для запобігання створенню проблем метеорологічним і військовим радарам. Регламентує динамічне зниження випромінюваної потужності Wi-Fi обладнання та/або динамічний перехід на інший частотний канал при виявленні тригерного сигналу (в більшості європейських країн наземні станції стеження за метеорологічними супутниками і супутниками зв'язку, а також радари військового призначення працюють в діапазонах, близьких до 5 МГц). Цей стандарт є необхідною вимогою ETSI, що пред'являється до обладнання, допущеного для експлуатації на території країн Європейського Союзу.

IEEE 802.11i

Виправляє проблеми безпеки в областях аутентифікації і протоколів шифрування. У перших варіантах стандартів Wi-Fi 802.11 для забезпечення безпеки мереж Wi-Fi використовувався алгоритм WEP. Передбачалося, що цей метод може забезпечити конфіденційність і захист даних, які передаються, авторизованих користувачів бездротової мережі від

прослуховування. Тепер цей захист можна зламати всього за кілька хвилин. Тому в стандарті 802.11i були розроблені нові методи захисту мереж Wi-Fi, реалізовані як на фізичному, так і програмному рівнях. В даний час для організації системи безпеки в мережах Wi-Fi 802.11 рекомендується використовувати алгоритми Wi-Fi Protected Access (WPA). Вони також забезпечують сумісність між бездротовими пристроями різних стандартів і різних модифікацій. Протоколи WPA використовують вдосконалену схему шифрування RC4 і метод обов'язкової аутентифікації з використанням EAP. Стійкість і безпеку сучасних мереж Wi-Fi визначається протоколами перевірки конфіденційності та шифрування даних (RSNA, TKIP, CCMP, AES). Найбільш рекомендованим підходом є використання WPA2 з шифруванням AES.

IEEE 802.11k

Цей стандарт спрямований на реалізацію балансування навантаження в мережі Wi-Fi. Зазвичай через бездротову локальну мережу абонентський пристрій з'єднується з тією точкою доступу, яка забезпечує найбільш сильний сигнал. Нерідко це призводить до перевантаження мережі в одній точці, коли до однієї точки доступу підключається одразу багато користувачів. Для контролю подібних ситуацій в стандарті 802.11k запропонований механізм, який обмежує кількість абонентів, що підключаються до однієї точки доступу, і дає можливість створення умов, за яких нові користувачі будуть приєднуватися до іншої точки доступу навіть не зважаючи на більш слабкий сигнал від неї. В цьому випадку пропускна здатність мережі збільшується завдяки більш ефективному використанню ресурсів.

IEEE 802.11m

У стандарті 802.11m об'єднуються підсумовуються поправки і виправлення для всієї групи стандартів 802.11.

IEEE 802.11n

У стандарті 802.11n використовуються частотні канали в частотних діапазонах Wi-Fi 2.4GHz і 5GHz. Сумісний з 11b / 11a / 11g. Підтримує частотні канали Wi-Fi шириною 20MHz і 40MHz (2x20MHz). Використовувана радіочастотна технологія: OFDM, а саме OFDM MIMO (Multiple Input Multiple Output) аж до рівня 4x4 (4 передавача і 4 приймача). При цьому мінімум 2 передавача на точку доступу та 1 передавач на пристрій користувача.

IEEE 802.11p

Визначає взаємодію Wi-Fi обладнання, яке рухається зі швидкістю до 200 км/год повз нерухому точку доступу Wi-Fi, віддалену на відстань до 1 км. Частина стандарту Wireless Access in Vehicular Environment (WAVE). Стандарти WAVE визначають архітектуру і додатковий набір службових функцій та інтерфейсів, які забезпечують безпечний механізм радіозв'язку між рухомими транспортними засобами. Ці стандарти розроблені для таких додатків, як, наприклад, організація дорожнього руху, контроль безпеки руху, автоматизований збір платежів, навігація і маршрутизація транспортних засобів тощо.

IEEE 802.11r

Визначає швидкий автоматичний роумінг Wi-Fi пристроїв при переході із зони покриття однієї точки доступу Wi-Fi до зони покриття іншої точки доступу Wi-Fi. Цей стандарт орієнтований на реалізацію мобільності та призначений для мобільних пристроїв з Wi-Fi модулями, наприклад, смартфонів, планшетних комп'ютерів, Wi-Fi IP-телефонів тощо. До появи цього стандарту при русі користувач часто втрачав зв'язок з однією точкою доступу, був змушений шукати іншу і заново виконувати процедуру підключення. Це займало багато часу. Пристрої з підтримкою 802.11r можуть виконувати процес переключення між точками доступу в автоматичному

режимі. Таким чином значно зменшується час, коли абонент не доступний в мережах Wi-Fi.

IEEE 802.11s

Стандарт для реалізації повнозв'язних мереж (Wireless Mesh), де будь-який пристрій може служити як маршрутизатором, так і точкою доступу. Якщо найближча точка доступу перевантажена, дані перенаправляються до найближчого незавантаженого вузла. При цьому пакет даних передається від одного вузла до іншого, поки не досягне кінцевого місця призначення. В даному стандарті введені нові протоколи на рівнях MAC і PHY, які підтримують трансляцію і багатоадресну передачу, а також одноадресну доставку по системі точок доступу Wi-Fi. З цією метою в стандарті введено четирьохадресний формат кадру.

IEEE 802.11t

Стандарт створений для інституалізації процесу тестування рішень стандарту IEEE 802.11. Описуються методики тестування, способи вимірів і обробки результатів (treatment), вимоги до випробувального устаткування.

IEEE 802.11u

Визначає процедури взаємодії мереж стандарту Wi-Fi з зовнішніми мережами. Стандарт визначає протоколи доступу, протоколи пріоритету і заборони на роботу із зовнішніми мережами.

IEEE 802.11v

У стандарті розроблені поправки, спрямовані на вдосконалення систем управління мережами стандарту IEEE 802.11. Модернізація на MAC і PHY рівнях дозволяє централізувати та впорядкувати конфігурацію клієнтських пристроїв, з'єднаних з мережею.

IEEE 802.11y

Додатковий стандарт зв'язку для діапазону частот 3,65...3,70 ГГц. Призначений для пристроїв останнього покоління, які працюють із зовнішніми антенами на швидкостях до 54 Мбіт/с на відстані до 5 км на відкритому просторі. Стандарт повністю не завершений.

IEEE 802.11w

Визначає методи і процедури поліпшення захисту і безпеки рівня управління доступом до середовища передавання даних (MAC). Протоколи стандарту структурують систему контролю цілісності даних, автентичності їх джерела, заборони несанкціонованого відтворення або копіювання, конфіденційності даних та інших засобів захисту. У стандарті введено захист фрейму управління (MFP: Management Frame Protection), а додаткові заходи безпеки дозволяють нейтралізувати зовнішні атаки. Крім того, ці заходи забезпечують безпеку для найбільш вразливої мережевої інформації, яка передається по мережах з підтримкою IEEE 802.11r, k, y.

IEEE 802.11ac.

Новий стандарт Wi-Fi, який працює тільки в частотній смузі 5 ГГц і забезпечує значно вищі швидкості як на індивідуального клієнта Wi-Fi, так і на точку доступу Wi-Fi. 802.11ac є розвитком широко поширеної технології 802.11n.

Рішення на базі Wi-Fi стандарту 802.11ac досягає високих швидкостей передавання даних (таблиця 2.14) за допомогою тривимірної функціональної матриці:

– більша кількість частотних каналів в сумі до: 80 МГц або навіть 160 МГц (в порівнянні з максимумом в 40 МГц для 802.11n);

– модуляція: до QAM256 (в 802.11n максимум QAM64);

– більший рівень MIMO: до 8 просторових потоків (в 802.11n до 4 потоків).

Технологія 802.11ac працює тільки на частотах Wi-Fi 5 ГГц. Тому двосмугові точки доступу найчастіше продовжують використовувати 801.11n на частотах 2,4 ГГц. Але Wi-Fi клієнти 802.11ac працюють в менш завантаженому спектрі частот 5 ГГц.

Таблиця 2.14.

Розподіл швидкостей у 802.11ac для різних конфігурацій

Channel bandwidth	Transmit - Receive antennas	Modulation and coding etc	Typical client scenario	Throughput (individual link rate)	Throughput (aggregate link rate)
80 MHz	1x1	256-QAM 5/6, short guard interval	Smartphone	433 Mbps	433 Mbps
80 MHz	2x2	256-QAM 5/6, short guard interval	Tablet, PC	867 Mbps	867 Mbps
160 MHz	1x1	256-QAM 5/6, short guard interval	Smartphone	867 Mbps	867 Mbps
160 MHz	2x2	256-QAM 5/6, short guard interval	Tablet, PC	1.73 Gbps	1.73 Gbps
160 MHz	4x Tx AP, 4 clients of 1x Rx	256-QAM 5/6, short guard interval	Multiple smartphones	867 Mbps per client	3.47 Gbps
160 MHz	8x Tx AP, 4 clients with total of 8x Rx	256-QAM 5/6 , short guard interval	Digital TV, set-top box, tablet, PC, smartphone	867 Mbps to two 1x clients 1.73 Gbps to one 2x client 3.47 Gbps to one 4x client	6.93 Gbps
160 MHz	8x Tx AP, 4 clients of 2x Rx	256-QAM 5/6, short guard interval	Multiple set-top boxes, PCs	1.73 Gbps to each client	6.93 Gbps

Wi-Fi Direct

Wi-Fi Direct дозволяє комп'ютерам і портативним ґаджетам зв'язуватися один з одним безпосередньо за існуючим протоколом Wi-Fi без використання маршрутизаторів і точок доступу. Тобто з'єднання встановлюється так само просто, як через Bluetooth. Важливим моментом є те, що для організації прямого з'єднання достатньо, щоб тільки один з пристроїв відповідав стандарту Wi-Fi Direct. Іншими словами, до сертифікованої апаратури може бути підключено будь-яке сучасне обладнання з підтримкою Wi-Fi. Максимальна відстань передавання даних сягає 100 метрів.

Формат кадру у сімействі стандартів 802.11

Стандарт 802.11 використовує три класи кадрів, що транспортуються через канал: інформаційні, службові та керуючі. Формат інформаційного кадру представлений на рис. 2.52.

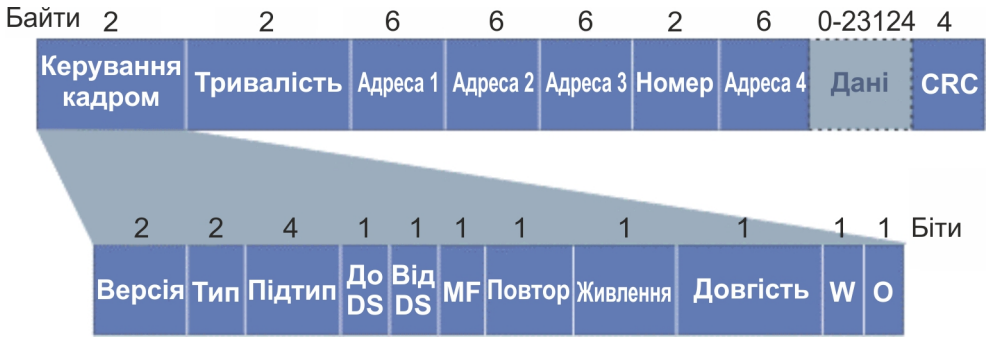


Рис. 2.52. Формат інформаційного кадру у сімействі стандартів 802.11

Поле керування кадром має 11 субполів. Субполе «Версія протоколу» дозволяє двом протоколам працювати в межах одного осередку. Поля «Тип» та «Підтип» задають різновид кадру (інформаційний, службовий, керуючий тощо). Біти «До DS» і «Від DS» вказують на напрям транспортування кадру: до міжстільникової системи (наприклад, Ethernet) або від неї. Біт «MF» вказує на те, що далі йде ще один фрагмент. Біт «Повтор» вказує на те, вперше чи повторно відсилається фрагмент. Біт «Керування живленням» використовується базовою станцією для перемикання в режим зниженого енергоспоживання або для виходу з цього режиму. Біт «Довгість» вказує чи це останній кадр, чи у відправника є ще кадри для пересилки. Біт «W» є покажчиком використання шифрування в тілі кадру згідно з алгоритмом WEP (Wired Equivalent Proto col). Однобітове поле «O» повідомляє приймачу, що кадри з цим бітом (= 1) мають оброблятися строго по порядку. Поле «Тривалість» задає час передавання кадру і його підтвердження. Це поле може бути присутнім в службових кадрах. Кадр містить чотири адреси: MAC-адресу призначення (адреса 1 - 48 двійкових розряди), MAC-адресу джерела (адреса 2 - 48 двійкових розряди), MAC-адресу бездротового

пристрою, що є безпосереднім одержувачем кадру (адреса 3) та MAC-адресу бездротового пристрою, який передав кадр (адреса 4).

Поле «Номер» служить для нумерації фрагментів. З 16 біт номера 12 ідентифікують кадр, а 4 – фрагмент. Поле «Дані» містить безпосередньо дані, які передаються, а поле «CRC» – контрольну суму. Керуючі кадри мають подібний формат, тільки там відсутні поля базових станцій, так як ці кадри не виходять за межі стільникового осередку. У службових кадрах відсутні поля «Дані» та «Номер», ключовим тут є вміст поля «Підтип» (RTS, CTS або ACK).

WiMAX

WiMAX (Worldwide Interoperability for Microwave Access) – це нова технологія, що базується на стандарті IEEE 802.16, який також називають Wireless MAN (Wireless Metropolitan Area Networks – бездротові мережі масштабу міста). Ця технологія надає ефективні засоби для побудови бездротових мереж передавання даних міського масштабу. WiMAX дає змогу бездротовим мережам охоплювати відстані до 50 км і передавати дані, голос і відео на високих швидкостях до 75 Мбіт/с. Короткі відомості про стандарти сімейства 802.16 наведені в таблиці 2.15.

Таблиця 2.15.

Відомості про стандарти сімейства 802.16

Стандарт	Полоса, ГГц	Швидкість, Мбіт/с	Модуляція	Радіус дії, км
802.16	10...66	32...134	QPSK	2...5
802.16a RevD	2...11	до 75	OFDM, QPSK, 16QAM, 64QAM	7...50
802.16e	2...6	до 15	OFDM, QPSK, 16QAM, 64QAM	2...5

Весь потік даних у мережах IEEE 802.16 – це потік пакетів. На основному підрівні MAC формуються пакети даних, які потім передаються

на фізичний рівень, інкапсулюються у пакети фізичного рівня і транслюються через канал зв'язку. Пакет даних підрівня MAC складається із загального MAC заголовку, поля даних (воно може бути відсутнє) і контрольної суми (рис. 2.53).

Рис. 2.53. Пакет MAC рівня у сімействі стандартів 802.16

Для керування процесом передавання (приймання) інформації використовують керуючі пакети, які включаються в поле даних пакету MAC рівня. Структура керуючого пакету представлена на рисунку 2.54.

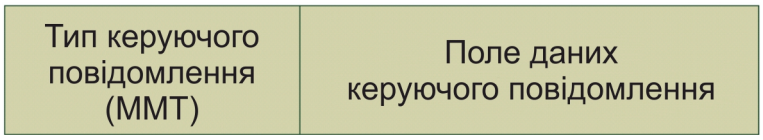


Рис. 2.54. Структура керуючого пакету у сімействі стандартів 802.16

Більш докладний розгляд даного питання виходить за рамки даного курсу. Бажаючі отримати більш докладну інформацію можуть знайти її в специфікаціях стандартів 802.16.

IEEE 802.15.1 (Bluetooth)

Технологія Bluetooth була розроблена компанією Ericsson. Згодом для просування на ринок був утворений консорціум у складі Ericsson, IBM, Intel, Nokia і Toshiba. Сьогодні до складу основних членів входять 3Com, Agere Systems, Ericsson, IBM, Intel, Microsoft, Motorola, Nokia, Toshiba. Мета творців Bluetooth – забезпечити можливість безпровідного зв'язку між різноманітними електронними приладами побутової техніки на відстані до 100 м.

У стандарті Bluetooth використовують радіочастоти 2400... 2483,5 МГц. Цей діапазон використовується в багатьох країнах для безліцензійного доступу і називається ISM (Industrial, Scientific, Medicine – промисловий, науковий і медичний). В технології Bluetooth весь діапазон розбитий на 78 каналів шириною 1 МГц кожен. У верхній і нижній частинах діапазону передбачені захисні невикористовувані смуги шириною 3,5 і 2 МГц відповідно.

За вихідною потужністю всі пристрої ділять на три класи:

– перший клас – до 100 мВт;

– другий клас – до 2,5 мВт;

– третій клас – до 1 мВт.

Для передавання даних використовується гаусова частотна модуляція, яка передбачає зміну частоти несної у часі відповідно до кривої Гаусса, що дає змогу обмежити спектр випромінюваного сигналу. Обмін даними здійснюється в середині часових інтервалів (слотів) довжиною 625 мкс.

Після передачі кожного слота відбувається перехід на інший частотний канал. Частина слотів зарезервована для синхронних каналів (передавання голосу), а всього протоколом передбачено до трьох синхронних каналів зі швидкістю 64 кбіт/с. Паралельно з синхронними даними можуть передаватися і асинхронні.

Для організації дуплексного зв'язку використовується метод часового мультиплексування, тобто в одному часовому слоті передає один пристрій, а в наступному – інший. При симетричній організації обміну асинхронними даними максимальна швидкість становить 433,9 кбіт/с в кожну сторону. Максимальна швидкість обміну досягається при асиметричному обміні та становить 723,2 кбіт/с в одну сторону і 57,6 кбіт/с – в іншу. Реальна швидкість передавання даних завжди менша за пропускну здатність каналу, оскільки частина трафіку використовується для передавання службової інформації, передбаченої протоколом передавання.

Bluetooth служить для організації каналів зв'язку типу «один до одного», однак можлива також і організація типу «один до багатьох». У будь-якому випадку один з пристроїв є ведучим (master), а всі інші – веденими (slave). Утворена таким чином структура називається пікомережею (piconet). В одній такій мережі можуть брати участь один ведучий пристрій і до семи ведених. Додатково в пікомережі можуть бути присутні й інші пристрої, які називаються блокованими (parked) і не беруть участь в обміні даними, але знаходяться в синхронізації з ведучим пристроєм.

IEEE 802.15.3

Стандарт IEEE 802.15.3 є прямим спадкоємцем Bluetooth. Протокол IEEE 802.15.3 забезпечує швидкість передавання даних до 55 Мбіт/с на відстані до 100 м; одночасно працювати в такій мережі можуть до 245 користувачів. При виникненні перешкод з боку інших побутових пристроїв чи інших мереж, мережі на основі IEEE 802.15.3 можуть автоматично перемикати канали. Також підтримуються швидкості передавання даних – 11, 22, 33 і 44 Мбіт/с. Крім того в мережах IEEE 802.15.3 за необхідності може здійснюватися шифрування даних за стандартом AES 128.

Інформація в мережах стандартів IEEE 802.15.1 та IEEE 802.15.3 передається кадрами. Існує декілька форматів кадрів, найважливіший з яких показаний на рис. 2.55. На початку кадру вказується код доступу, який зазвичай служить ідентифікатором головного вузла. Це дозволяє двом головним вузлам, які розташовані достатньо близько, щоб «чути» один одного, розрізняти, кому з них призначаються дані. Потім слідує 54-бітний заголовок, в якому містяться поля, характерні для кадру підрівня MAC. Далі розташовано поле даних, розмір якого обмежений 2744 бітами (для передачі за п'ять тактів). Якщо кадр має довжину, яка відповідає одному тактовому інтервалу, то формат залишається таким же, з тією різницею, що поле даних в цьому випадку становить 240 біт.

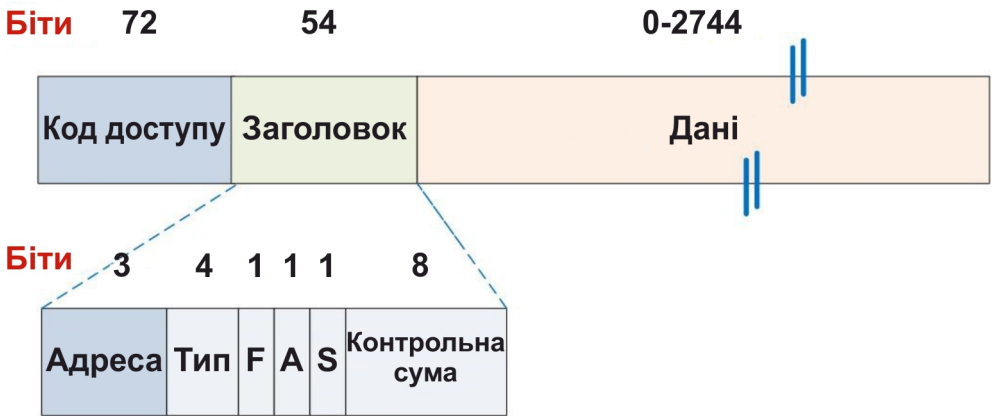


Рис. 2.55. Структура кадру у стандартах IEEE 802.15.1 та IEEE 802.15.3

Поле «Адреса» ідентифікує однин з восьми пристроїв, якому призначена інформація. Поле «Тип» визначає тип переданого кадру (ACL, SCO, опитування або порожній кадр), метод корекції помилок і кількість тимчасових інтервалів, з яких складається кадр. Біт «F» (Flow – потік) виставляється підлеглим вузлом і повідомляє про те, що його буфер заповнений. Це така примітивна форма управління потоком. Біт «A» (Acknowledgement – підтвердження) являє собою підтвердження (ACK), надіслане одночасно з кадром. Біт «S» (Sequence – послідовність) використовується для визначення повторної передачі кадру. Далі слідує 8-бітна контрольна сума заголовку. Весь 18-бітний заголовок кадру повторюється тричі, що в підсумку становить 54 біта, як показано на рис. 2.55. На приймаючій стороні нескладна схема аналізує всі три копії кожного біта. Якщо вони збігаються, кожен біт приймається таким, яким він є. В іншому випадку все вирішує більшість. Отже на передачу 10 біт витрачається в даному випадку 54 біта. Причина дуже проста: за все треба платити. За забезпечення передавання даних за допомогою дешевих малопотужних пристроїв (2,5 мВт) з невисокими обчислювальними можливостями доводиться платити великою надмірністю інформації, яка передається.

У кадрах, залежно від типу, може бути три різних формати поля даних. У найпростішому випадку довжина поля даних завжди дорівнює 240 біт. Можливі три варіанти: 80, 160 або 240 біт корисної інформації. При цьому біти поля даних, які залишились, використовуються для виправлення

помилок. Найнадійніша версія (80 біт корисної інформації) має дуже просту структуру: один і той же вміст повторюється три рази (що і складає 240 біт). Тобто метод тут застосовується той же, що і в заголовку кадру.

Ultra Wideband

UWB (Ultra Wideband) – це технологія надширокосмугового радіозв'язку. Вона розроблена корпорацією Intel для швидкостей передачі даних до 500 Мбіт/с на відстань до кількох метрів. Для передачі даних використовуються дуже короткі радіоімпульси (менше 1 нс) в широкому діапазоні частот 3,1...10,6 ГГц. За допомогою UWB-технології можна створювати спеціальні мережі, в яких кілька пристроїв зможуть підтримувати зв'язок між будь-якими вузлами. Короткі сигнали UWB порівняно стійкі до багатопроменевого загасання, яке виникає при відбитті хвилі від стін, стелі, будівель, транспортних засобів тощо. Високошвидкісні UWB-пристрої добре підходять для роботи з потоками відео і додатками, які потребують швидкого пересилання даних. Низькошвидкісне UWB-обладнання може застосовуватися для відстежування місцеположення на місцевості власників бездротових пристроїв і різних об'єктів. Для мобільних пристроїв важливим є той факт, що в широкому спектрі потрібно набагато менше витрат енергії, ніж для передавання вузькосмугового сигналу, в силу різного рівня сигналу: в широкому спектрі можна використовувати шумоподібні сигнали з малим відношенням сигнал/шум. Тому чіпи UWB економніші, ніж, наприклад, чіпи Bluetooth і мають при цьому набагато більшу пропускну здатність.

IEEE 802.15.4 (ZigBee)

Стандарт IEEE 802.15.4 (ZigBee) орієнтований головним чином на використання в якості засобу зв'язку між автономними приладами та обладнанням. У корпоративному секторі це можуть бути, наприклад, складські системи, системи автоматизації виробництва, різні датчики, сенсори, сервоприводи, електронні мітки, а в домашніх умовах – ПК, ігрові

приставки, системи безпеки, освітлення, кондиціонування, радіофіковані іграшки і навіть пульти дистанційного керування.

Стандарт IEEE 802.15.4 визначає параметри фізичного рівня (PHY) і протокол управління доступом (MAC), пропонуючи підтримку мереж різних топологій. Ключові функції PHY-рівня включають в себе контроль за енергією і якістю ланок, а також оцінку каналів для більш успішного співіснування з мережами інших бездротових операторів. MAC-рівень визначає автоматичне підтвердження отримання пакетів, забезпечує можливість передавання даних в певні часові інтервали і підтримує 128-бітні функції безпеки AES. Якщо в межах досяжності ZigBee-пристроїв виявиться обладнання Wi-Fi або Bluetooth, їх канали можуть бути використані як тунель для трафіку ZigBee.

Стандарт IEEE 802.15.4 передбачає невелику дальність дії (близько 10 метрів) і пропускну здатність каналу (до 250 кбіт/с). Передавання на цій швидкості ведеться в діапазоні 2,4 ГГц. Доступні також діапазони 858 МГц (20 кбіт/с) і 902 ... 928 МГц (40 кбіт/с).

Передавання даних у протоколах сімейства IEEE 802.15.4 здійснюється фреймами. Стандарт передбачає чотири типи фреймів: фрейм маячка, фрейм даних, фрейм повідомлення про отримання та фрейм команд MAC-підрівня.

Фрейм даних (рис. 2.56) починається з преамбули, яка спільно з полем "Старт" служить для синхронізації приймача, поле "Довжина" містить довжину поля MAC-підрівня в 8-бітових байтах (октетах). Поле "Управління" містить службову інформацію про управління кадрами, поле "Номер" містить порядковий номер даних, поле "Адреса" містить адресну інформацію, в тому числі 16-бітну коротку або 64-бітну розширену адресу. Завершується фрейм полем контрольної суми.

Рис. 2.56. Структура фрейму даних у сімействі стандартів IEEE 802.15.4

2.6. Інтерфейс Ethernet

Залежно від швидкості передавання даних і передавального середовища існує декілька варіантів технології. Незалежно від способу передавання, стек мережевого протоколу і програми працюють однаково у всіх варіантах.

Більшість Ethernet-карт та інших пристроїв має підтримку декількох швидкостей передавання даних, використовуючи автовизначення (autonegotiation) швидкості та дуплексності для досягнення найкращого з'єднання між двома пристроями. Якщо автовизначення не спрацьовує, швидкість підстроюється під партнера і включається режим напівдуплексної передачі. Наприклад, наявність в пристрої порту Ethernet 10/100 говорить про те, що через нього можна працювати за технологіями 10BASE-T і 100BASE-TX, а порт Ethernet 10/100/1000 - підтримує стандарти 10BASE-T, 100BASE-TX і 1000BASE-T.

10 Мбіт/с Ethernet

10BASE5, IEEE 802.3 (званий також «Товстий Ethernet» діаметр кабелю приблизно 12 мм) – перша розробка технології зі швидкістю передачі даних 10 Мбіт/с. Згідно стандарту, IEEE використовує коаксіальний кабель з хвильовим опором 50 Ом (RG-8) (рис. 2.57), з максимальною довжиною сегмента 500 метрів.

а *б*

Рис. 2.57. Кабель (*а*) і рознімач (*б*) стандарту 10BASE5, IEEE 802

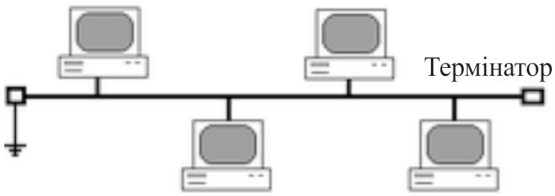
10BASE2, IEEE 802.3a (званий «Тонкий Ethernet» – діаметр кабелю приблизно 6 мм) використовує кабель RG-58, з максимальною довжиною сегмента 185 метрів. Для підключення кабелю до мережевої карти потрібен Т-конектор (рис. 2.58, *а*), а на кабелі має бути BNC-конектор. Потрібна наявність термінаторів (рис. 2.58, *б*) на кожному кінці . Багато років цей стандарт був основним для технології Ethernet.

а

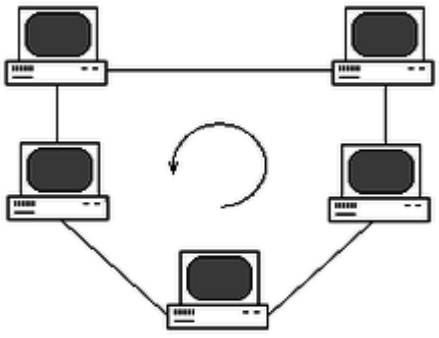
б

Рис. 2.58. Т-конектор (*а*) і термінатор (*б*) 10BASE5, IEEE 802

Мережі на основі коаксіальних кабелів будуються на основі шинної топології або на основі кільцевої топології (рис. 2.59).



а



б

Рис. 2.59. Шинна (*а*) і кільцева (*б*) топології

StarLAN 10 – перша розробка, яка використовує виту (кручену) пару для передачі даних на швидкості 10 Мбіт/с (рис. 2.60). Надалі еволюціонував в стандарт 10BASE-T.

а

б

Рис. 2.60. Кабель стандарту 10BASE-T (на 4 витих пари) із рознімачем типу 8P8C (*а*) та рознімач 8P8C (*б*)

Існує дві схеми з'єднання 8P8C з витою парою (таблиця 2.16):

– пристрій–пристрій: в цьому випадку використовується стандарт EIA/TIA-568B з одного боку та EIA/TIA-568A з іншого;

– пристрій–мережевий комутатор: в цьому випадку використовується стандарт EIA/TIA-568A з обох боків кабелю.

Таблиця 2.16.

Розлініювання контактів рознімача 8P8C

№	EIA/TIA-568B	EIA/TIA-568A
1.	Біло-Помаранчевий	Біло-Зелений
2.	Помаранчевий	Зелений
3.	Біло-Зелений	Біло-Помаранчевий
4.	Синій	Синій
5.	Біло-Синій	Біло-Синій
6.	Зелений	Помаранчевий
7.	Біло-Коричневий	Біло-Коричневий
8.	Коричневий	Коричневий

Можна використовувати різні типи кабелів витих пар (таблиця 2.17).

Таблиця 2.17.

Позначення для поширених типів кабелів з кручених пар

Загальноприйнята назва	Позначення згідно ISO/IEC 11801	Загальний екран	Екран для пар
UTP	U/UTP	ні	ні
STP, ScTP, PiMF	U/FTP	ні	фольга
FTP, STP, ScTP	F/UTP	фольга	ні
STP, ScTP	S/UTP	обплетення	ні
SFTP, S-FTP, STP	SF/UTP	обплетення, фольга	ні
SFSFTP, SF-SFTP, SF²TP	SF/SFTP	обплетення, фольга	обплетення, фольга
FFTP	F/FTP	фольга	фольга
SSTP, SFTP, STP PiMF	S/FTP	обплетення	фольга
SSTP, SFTP	SF/FTP	обплетення, фольга	фольга

Літерний код перед зворотною рискою позначає тип загального екрану для всього кабелю, код після риски позначає тип індивідуального захисту, для кожної кручений пари:

U – unshielded, без екрану
F – foil, фольга
S – braided screening, обплетення з дроту (тільки зовнішній екран)
TP – twisted pair, вита пара
TQ – індивідуальний екран для двох витих пар (на 4 проводи)

Існує декілька категорій кабелю вита пара, які нумеруються від CAT 1 до CAT 8.2. Кабель вищої категорії зазвичай містить більше пар дротів і кожна пара має більше витків на одиницю довжини:

CAT 1 – телефонний кабель, одна пара. Використовується тільки для передачі голосу або даних за допомогою модему. Смуга частот 0,1...0,4 МГц.

CAT 2 – старий тип кабелю з 2-х пар провідників, підтримував передачу даних на швидкостях до 4 Мбіт/с, використовувався в мережах token ring і ARCNet. Зараз зустрічається в телефонних мережах. Смуга частот 1...4 МГц.

CAT 3 – 2-парний кабель, використовувався для побудові локальних мереж 10BASE-T і token ring, підтримує швидкість передачі даних тільки до 10 Мбіт/с. На відміну від попередніх двох, відповідає вимогам стандарту IEEE 802.3. Також зустрічається в телефонних мережах. Смуга частот 16 МГц.

CAT 4 – кабель складається з 4-х скручених пар, використовувався в мережах token ring, 10BASE-T, 10BASE-T4, швидкість передачі даних не перевищує 16 Мбіт/с, зараз не використовується. Смуга частот 20 МГц.

CAT 5 – 4-парний кабель, це і є те, що зазвичай називають кабель «вита пара». Використовується при побудові локальних мереж 10BASE-T, 100BASE-TX і 1000BASE-T і для прокладки телефонних ліній, підтримує швидкість передачі даних до 100 Мбіт/с при використанні 2 пар і до 1000 Мбіт/с при використанні 4 пар. Смуга частот 100 МГц.

CAT 5e – 4-парний кабель, вдосконалена категорія 5. Швидкість передачі даних до 100 Мбіт/с при використанні 2 пар і до 1000 Мбіт/с при використанні 4 пар. Кабель категорії 5e є найпоширенішим і використовується для побудови комп'ютерних мереж. Іноді зустрічається двохпарний кабель категорії 5e. Переваги даного кабелю в нижчій собівартості та меншій товщині. Смуга частот 100 МГц.

CAT 6 – Застосовується в мережах Fast Ethernet і Gigabit Ethernet, складається з 4 пар провідників і здатний передавати дані на швидкості до 10000 Мбіт/с. Пропускає сигнали частотою до 200 МГц. Існує категорія CAT6e, в якій збільшена частота сигналу, який пропускається, до 500 МГц. Смуга частот 250 МГц.

CAT 6A – складається з 4 пар провідників і здатний передавати дані на швидкості до 10 Гбіт/с на відстань до 100 метрів. Доданий в стандарт ISO/IEC 11801:2002 поправка 2 в лютому 2008 року. Кабель цієї категорії має або загальний екран (F/UTP), або екрани навколо кожної пари (U/FTP). Смуга частот 500 МГц.

CAT 7 – специфікація на даний тип кабелю затверджена тільки міжнародним стандартом ISO 11801, але не ANSI/TIA-568-C. Швидкість передачі даних – до 10000 Мбіт/с, частота сигналу, що пропускається, до 600...700 Мгц. Кабель цієї категорії екранований.

CAT 7A – міжнародний стандарт ISO 11801, швидкість передачі даних до 10 Гбіт/с. Загальний екран і екрани навколо кожної пари (F/FTP або S/FTP). Смуга частот 1000 МГц.

CAT 8/CAT 8.1 – у розробці, технічна рекомендація ISO/IEC TR 11801-99-1 і міжнародний стандарт ISO 11801 редакція 3 (для Cat.8.1), американський стандарт ANSI/TIA-568-C.2-1 (для Cat. 8). Повністю сумісний з кабелем категорії 6A. Швидкість передачі даних до 40 Гбіт/с при використанні стандартних конекторів 8P8C. Кабель цієї категорії має або загальний екран, або екрани навколо кожної пари (F/UTP або U/FTP). Смуга частот 1600...2000 МГц.

CAT 8.2 – у розробці, міжнародний стандарт ISO 11801 редакція 3. Повністю сумісний з кабелем категорії 7A. Швидкість передачі даних до 40 Гбіт/с при використанні стандартних конекторів 8P8C або GG45/ARJ45 і TERA. Кабель цієї категорії має загальний екран і екрани навколо кожної пари (F/FTP або S/FTP). Смуга частот 1600...2000 МГц.

Мережі на витій парі використовують топологію «зірка» (рис. 2.61). Термінатори для роботи по кручений парі вбудовані в кожен пристрій, і застосовувати додаткові зовнішні термінатори в лінії не потрібно.

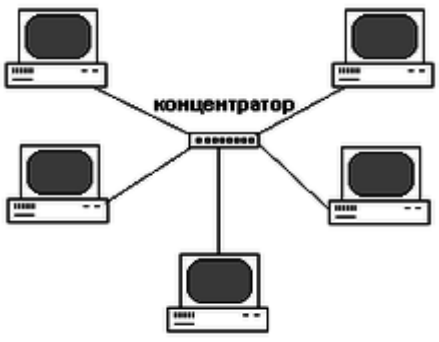


Рис. 2.61. Топологія типу «зірка»

10BASE-T, IEEE 802.3i – для передачі даних використовується 4 проводи кабелю витої пари (дві виті пари) категорії 3 або категорії 5. Максимальна довжина сегмента – 100 метрів.

FOIRL – (Fiber-optic inter-repeater link). Базовий стандарт для технології Ethernet, яка використовує для передачі даних оптичний кабель. Максимальна відстань передачі даних без повторювача – 1 км.

10BASE-F, IEEE 802.3j – основний термін для позначення сімейства 10 Мбіт/с Ethernet-стандартів, які використовують оптичний кабель на відстані до 2 кілометрів: 10BASE-FL, 10BASE-FB і 10BASE-FP. З перерахованого тільки 10BASE-FL набув широкого поширення.

10BASE-FL (Fiber Link) – покращена версія стандарту FOIRL. Поліпшення торкнулося збільшення довжини сегмента до 2 км.

10BASE-FB (Fiber Backbone) – зараз не використовують. Даний стандарт, призначався для об'єднання повторювачів в магістраль.

Швидкий Ethernet (Fast Ethernet, 100 Мбіт/с)

100BASE-T – загальний термін для позначення стандартів, які використовують в якості середовища передачі даних виту пару. Довжина сегмента – до 100 метрів. Включає в себе стандарти 100BASE-TX, 100BASE-T4 і 100BASE-T2.

100BASE-TX, IEEE 802.3u – розвиток стандарту 10BASE-T для використання в мережах топології «зірка». Задіяна вита пара категорії 5, фактично використовуються тільки дві неекрановані пари провідників, підтримується дуплексна передача даних, відстань до 100 м.

100BASE-T4 – стандарт, який використовує виту пару категорії 3. Задіяні всі чотири пари провідників, передавання даних йде в напівдуплексному режимі. Практично не використовується.

100BASE-T2 – стандарт, який використовує виту пару категорії 3. Задіяні тільки дві пари провідників. Підтримується повний дуплекс, коли

сигнали поширюються в протилежних напрямках по кожній парі. Швидкість передачі в одному напрямку – 50 Мбіт/с. Практично не використовується.

100BASE-FX – стандарт, який використовує багатомодове волокно. Максимальна довжина сегмента – 400 метрів у напівдуплексному режимі (для гарантованого виявлення колізій) або 2 кілометри в повному дуплексі.

100BASE-SX – стандарт, який використовує багатомодове волокно. Максимальна довжина обмежена тільки величиною затухання в оптичному кабелі та потужністю передавачів і складає, за різними матеріалами, від 2 до 10 кілометрів.

100BASE-FX WDM – стандарт, який використовує одномодове волокно. Максимальна довжина обмежена тільки величиною затухання в волоконно-оптичному кабелі та потужністю передавачів. Інтерфейси бувають двох видів, відрізняються довжиною хвилі передавача і маркуються або цифрами (довжина хвилі), або однією латинською літерою A (1310) або B (1550). У парі можуть працювати тільки парні інтерфейси: з одного боку передавач з довжиною хвилі 1310 нм, а з іншого – 1550 нм.

Гігабітний Ethernet (Gigabit Ethernet, 1 Гбіт/с)

1000BASE-T, IEEE 802.3ab – основний гігабітний стандарт. Використовує виту пару категорії 5e. У передаванні даних беруть участь 4 пари, кожна пара використовується одночасно для передавання в обох напрямках зі швидкістю 250 Мбіт/с. Відстань – до 100 метрів.

1000BASE-TX був створений Асоціацією Телекомунікаційної Промисловості (англ. Telecommunications Industry Association, TIA). Він був опублікований в березні 2001 року як «Специфікація фізичного рівня дуплексного Ethernet 1000 Мб/с (1000BASE-TX) симетричних кабельних систем категорії 6 (ANSI / TIA / EIA -854-2001)». Поширення не отримав через високу вартість кабелів, фактично застарів. Стандарт розділяє прийняті та відіслані сигнали по парах (дві пари передають дані та дві пари приймають, кожна на 500 Мбіт/с), що спрощує конструкцію приймальних і

передавальних пристроїв. Ще однією істотною відмінністю 1000BASE-TX була відсутність схеми цифрової компенсації наведень і зворотних перешкод, в результаті чого складність, рівень енергоспоживання і ціна реалізацій мають ставати нижче, ніж у стандарту 1000BASE-T. Для роботи технології потрібна кабельна система 6-ї категорії. На основі даного стандарту створено велику кількість продуктів для промислових мереж.

1000BASE-X – загальний термін для позначення стандартів зі змінними пристроями передавання приймання.

1000BASE-SX, IEEE 802.3z – стандарт, який використовує багатомодове волокно в першому вікні прозорості з довжиною хвилі, що дорівнює 850 нм. Дальність проходження сигналу становить до 550 метрів.

1000BASE-LX, IEEE 802.3z – стандарт, який використовує одномодове або багатомодове оптичне волокно в другому вікні прозорості з довжиною хвилі, що дорівнює 1310 нм. Дальність проходження сигналу залежить тільки від типу використовуваних приймачів і, як правило, становить для одномодового оптичного волокна до 5 км і для багатомодового оптичного волокна до 550 метрів.

1000BASE-CX – стандарт для коротких відстаней (до 25 метрів), який використовує 2-парний екранований кабель. Сигнал передається по одній парі, приймається по іншій парі проводів.

1000BASE-LH (Long Haul) – стандарт, який використовує одномодове волокно. Дальність проходження сигналу без повторювача – до 100 кілометрів.

2,5- і 5-гігабітні варіанти (NBASE-T, MGBASE-T)

У 2014 з'явилися приватні ініціативи NBASE-T (Cisco) і MGBASE-T (Broadcom) по створенню стандартів Ethernet зі швидкістю, проміжною між 1 і 10 Гбіт/с. Даний стандарт повинен використовувати існуючу кабельну інфраструктуру категорії 5e на відстанях до 100 метрів, надаючи швидкості в 2,5 або 5 Гбіт/с. Серед причин появи ініціатив – поширення Wi-Fi

маршрутизаторів, які підтримують швидкості більше 1 Гбіт/с і неможливість використання 10 Гбіт/с стандартів Ethernet по довгих кабелях 5e- і 6-й категорій. Восени 2016 року даний стандарт був затверджений як IEEE 802.3bz.

10-гігабітний Ethernet (10G Ethernet, 10 Гбіт / с)

Стандарт 10-гігабітного Ethernet включає в себе сім стандартів фізичного середовища для LAN, MAN і WAN. В даний час він описується поправкою IEEE 802.3ae і має увійти в наступну ревізію стандарту IEEE 802.3.

10GBASE-CX4 – технологія 10-гігабітного Ethernet для коротких відстаней (до 15 метрів), передбачає використання мідного кабелю CX4 і конекторів InfiniBand.

10GBASE-SR – технологія 10-гігабітного Ethernet для коротких відстаней (до 26 або 82 метрів) за умови використання багатомодового волокна. При використанні новітнього багатомодового волокна підтримує відстані до 300 метрів.

10GBASE-LX4 – використовує ущільнення по довжині хвилі для підтримки відстаней від 240 до 300 метрів за умови використання багатомодового волокна. Також підтримує відстані до 10 кілометрів при використанні одномодового волокна.

10GBASE-LR і 10GBASE-ER – ці стандарти підтримують відстані до 10 і 40 кілометрів, відповідно.

10GBASE-SW, 10GBASE-LW і 10GBASE-EW – ці стандарти використовують фізичний інтерфейс, сумісний за швидкістю і форматом даних з інтерфейсом OC-192 / STM-64 SONET / SDH. Вони подібні до стандартів 10GBASE-SR, 10GBASE-LR і 10GBASE-ER відповідно, оскільки використовують ті ж самі типи кабелів і відстані передачі.

10GBASE-T, IEEE 802.3an-2006 – прийнятий в червні 2006 року після 4 років розробки. Використовує виту пару категорії 6 (максимальна відстань 55 метрів) і 6а (максимальна відстань 100 метрів).

10GBASE-KR – технологія 10-гігабітного Ethernet для кросплатформенних модульних комутаторів, маршрутизаторів і серверів.

40-гігабітний і 100 Gigabit Ethernet

802.3ba затверджує стандарти Ethernet - 40 Gigabit Ethernet (або 40GbE) і 100 Gigabit Ethernet (або 100GbE).

Формат кадрів канального рівня практично однаковий для всіх Ethernet сумісних технологій. Технологія Ethernet передбачає кадри чотирьох форматів, які незначно відрізняються один від одного. Один з форматів кадру (802.3) підрівні MAC наведено на рис. 2.62.

Преамбула	SFD	DA	SA	L/T	Data	FCS
7 байт (10101010)	10101011	6 байт	6 байт	2 байта	46 - 1500 байт	4 байта

Рис. 2.62. Формат кадру Ethernet

Роздільник кадрів, що дозволяє визначити початок кадру і забезпечити синхронізацію між передавачем і приймачем, представлений преамбулою і початковим обмежувачем кадру (Start of Frame Delimiter – SFD). Преамбула кадру складається з семи байт 10101010, необхідних для входження приймача в режим синхронізації. Початковий обмежувач – 10101011 визначає початок кадру. У деяких форматах усі 8 байт, які перераховані, називаються преамбулою. Також формат кадру включає поля фізичних адрес вузла призначення (DA – Destination Address) і вузла джерела (SA – Source Address). У технологіях Ethernet фізичні адреси отримали назву MAC-адрес. MAC-адреси містять 48 двійкових розрядів. Поле «L» визначає довжину поля даних «Data», яке може бути від 46 до 1500 байт. Якщо поле даних менше 46

байт, то воно доповнюється до 46 байт. В даний час часто використовується формат кадру стандарту Ethernet-II, в якому замість поля «L» задається поле типу «T», де вказано протокол мережевого рівня. Решта полів кадру Ethernet-II ідентичні полям кадру стандарту 802.3. Поле контрольної суми (FCS – Frame Check Sequence) довжиною в 4 байта дає змогу визначити наявність помилок в отриманому кадрі, за рахунок використання алгоритму перевірки на основі циклічного коду CRC.

2.7. Тактові генератори

Для роботи мікроконтролеру необхідне тактування. Більшість мікроконтролерів здатні працювати в дуже широкому діапазоні частот від нуля до десятків мегагерц. Це можливо завдяки використанню повністю статичної логіки. Деякі розробники використовують тактову частоту в 1 Гц і менше при налагодженні програмного забезпечення. Широкий діапазон можливих робочих частот дозволяє розробнику краще налагодити мікроконтролер на виконання конкретних заданих функцій.

Існує три способи завдання тактової частоти мікроконтролера, кожен з яких має свої переваги і недоліки. Перший спосіб – використання кварцового резонатора, підключеного згідно зі схемою на рис 2.63.

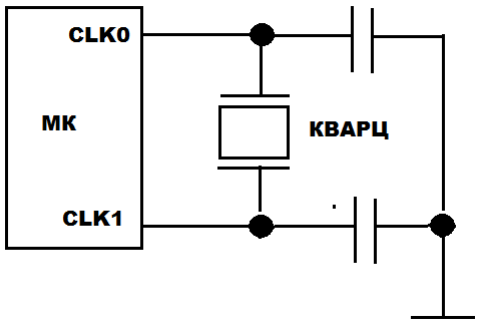


Рис. 2.63. Схема підключення кварцового резонатора

Цей спосіб дозволяє дуже точно задати тактову частоту мікроконтролера (розкид частоти зазвичай становить не більше 0,01%).

Номінали ємності конденсаторів в даній схемі визначаються виробником мікроконтролера для конкретної резонансної частоти кварцу. Іноді потрібно включити резистор великого номіналу. При використанні конденсаторів меншої ємності форма тактових імпульсів буде більш правильною (рис. 2.64).

а *б*

Рис. 2.64. Правильна (*а*) та неправильна (*б*) форма імпульсів

Недоліками цього способу є необхідність підключення додаткових елементів і крихкість кристалів кварцу. Ці недоліки можна усунути використовуючи замість кварця керамічний резонатор, який є більш стійким до механічного навантаження. Деякі керамічні резонатори мають вбудовані ємності, що спрощує схему підключення. При використанні керамічних резонаторів розкид частот сягає 0,5 %.

Другий спосіб тактування – використання RC-генератора (рис. 2.65). В цьому випадку тактова частота задається номіналами резистора та конденсатора.

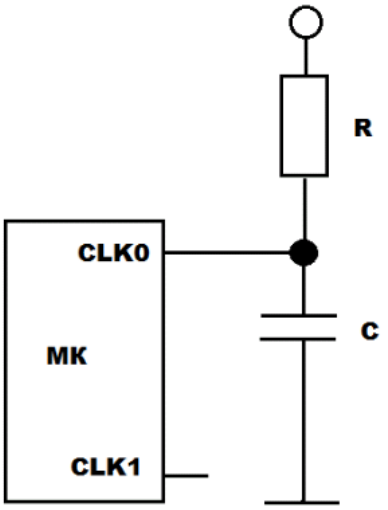


Рис. 2.65. Схема підключення RC-генератора

Номінали резистора та конденсатора, необхідні для тактування із заданою частотою, можна знайти в керівництві по мікроконтролеру. Перевагою даного способу є його дешевизна, а недоліком – низька точність і низька стабільність тактової частоти. Тому RC-генератори використовують, як правило, лише у випадках, коли немає потреби у високій стабільності тактової частоти. Підвищити точність можна використовуючи керовані резистори та конденсатори, але це здорожчує схему і збільшує масогабаритні характеристики. Деякі типи мікроконтролерів випускаються із вбудованими RC-генераторами, що спрощує схему і здешевлює процес виготовлення пристроїв на їх основі.

Третім способом тактування мікроконтролерів є використання зовнішніх генераторів.

2.8 Коло скидання до початкового стану

Запуск контролера має відбуватись тоді, коли вже встановилась потрібна напруга живлення. Для того, щоб бути впевненим, що напруга досягла потрібного стану, використовують схему, наведену на рис. 2.66. Дана схема підключається до вивода мікроконтролера, який відповідає за запуск мікроконтролера. Дане коло затримує запуск мікроконтролера на час, який потрібен для встановлення необхідного значення напруги живлення. Тривалість затримки формується підбором номіналів резисторів та конденсатора.

Кнопка скидання дає змогу за необхідності перезапустити мікроконтролер в ручному режимі. Така необхідність може виникнути, наприклад, при налагодженні програми або при «зависанні» системи. Резистор 100 Ом додатково слугує для обмеження струму при ручному перезапуску мікроконтролера. В деяких мікроконтролерах коло скидання до початкового стану може бути спрощене за рахунок того, що деякі елементи даного кола входять до складу мікроконтролера. Також для підвищення надійності роботи МПС можуть бути використані спеціальні пристрої –

монітори напруги живлення. Такі пристрої монтуються у корпус з трьома виводами і за зовнішнім виглядом нагадують транзистор.

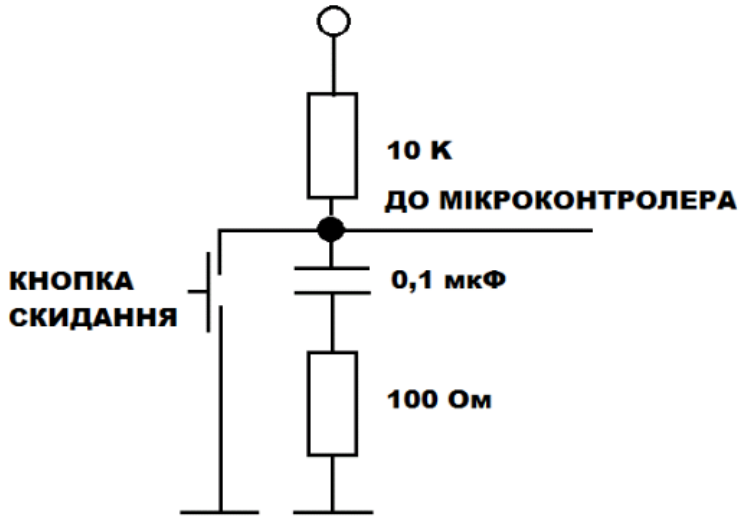


Рис. 2.66. Коло скидання до початкового стану

2.9 Таймери

Таймер (англ. timer, від англ. time – час) – один пристроїв або засіб, що відраховує інтервал(и) часу. Таймер можна також вважати одним з видів годинників. Таймер може бути використаний для контролю послідовності подій чи процесів.

Таймер є одним з базових блоків мікроконтролера (рис 2.67). Зазвичай в сучасних мікроконтролерах є кілька таймерів.

Здебільшого під таймерами мають на увазі пристрої, що відмірюють заданий інтервал часу з моменту запуску, але таймер може бути використаний у декілька способів:

– як пристрій для визначення часових інтервалів;

– як лічильник зовнішніх імпульсів;

– як дільник частоти зовнішнього сигналу;

– як частина більш складного пристрою, наприклад, пристрою для формування сигналів з широтно-імпульсною модуляцією (ШІМ-сигналів);

– як сторожовий таймер.

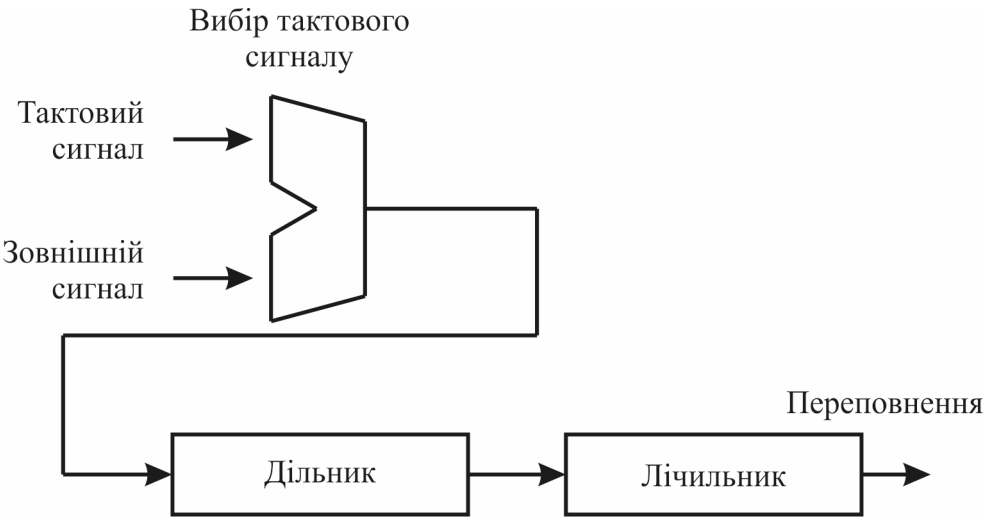


Рис. 2.67. Структура таймера у мікроконтролері

Перші три пункти зрозумілі. Розглянемо два останніх.

Широтно-імпульсна модуляція (ШІМ – англ. pulse-width modulation, PWM), або модуляція за тривалістю імпульсів (англ. pulse-duration modulation, PDM) – процес керування шириною (тривалістю) високочастотних імпульсів за законом, який задає низькочастотний сигнал (рис. 2.68). ШІМ-сигнали можуть бути використані, наприклад, для керування кроковим двигуном або іншим пристроєм. Також вони можуть бути використані для формування мікроконтролером пилоподібного сигналу або синусоїдального сигналу тощо.

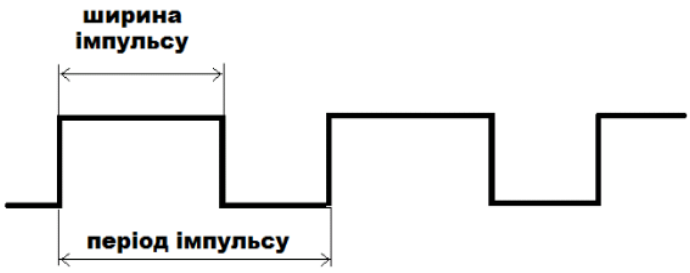


Рис. 2.68. ШІМ-сигнал

Структурна схема пристрою ШІМ показана на рис. 2.69. Вона працює наступним чином. При вмиканні ШІМ-пристрою на виході встановлюється високий рівень сигналу, який підтримується поки задане значення довжини імпульсу більше, ніж значення лічильника таймеру. Коли значення

зрівняються, то встановлюється низький рівень, який утримується доти, доки значення лічильника не досягне значення періоду імпульсу. Потім значення лічильника таймера обнуляється, встановлюється високий рівень і процес повторюється знову, поки ШІМ-пристрій не буде вимкнений.

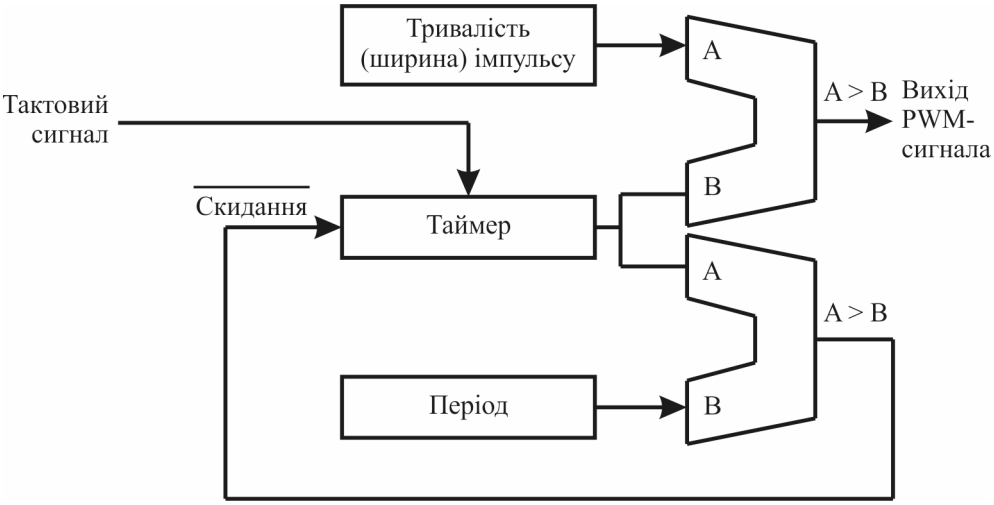


Рис. 2.69. Структура пристрою ШІМ

Сторожовий таймер (англ. Watchdog timer, WDT) – апаратно реалізована схема контролю за зависанням системи. Являє собою таймер, який періодично перезапускається контрольованою системою. Якщо скидання не відбулося протягом деякого інтервалу часу, відбувається примусовий перезапуск системи. Структурну схему сторожового таймеру наведено на рис. 2.70.

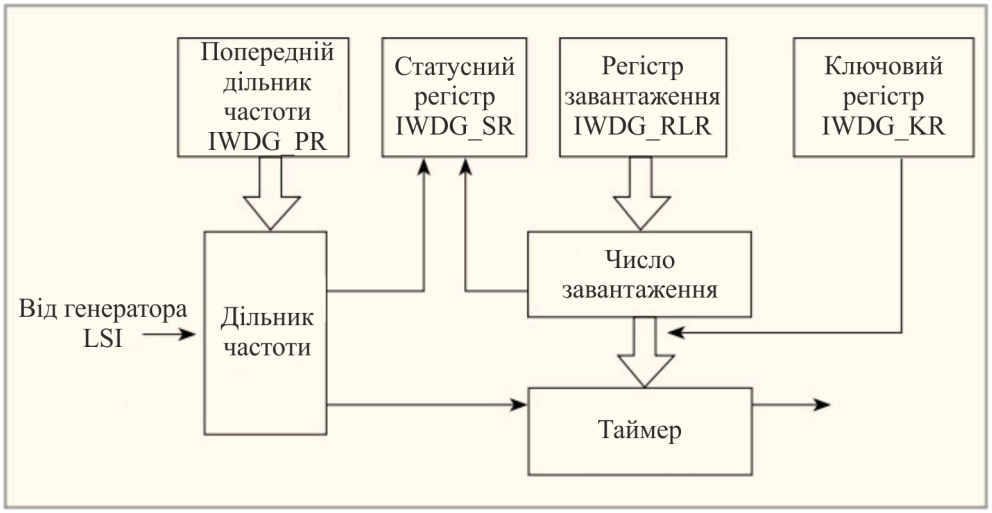


Рис. 2.70. Структурна схема сторожового таймеру

Ідея рішення полягає в тому, що сторожовий таймер ініціалізується і запускається самим процесором на початку роботи програми. Після чого даний таймер самостійно відраховує заданий час, і якщо протягом цього часу процесор жодного разу не перезавантажить його, то таймер обнулиться і перезапустить процесор. Таким чином, при нормальній роботі процесора таймер буде регулярно перезапускатися, а в разі «зависання» процесора таймер через заданий час сформує сигнал скидання для процесора і перезапустить його. Таймер тактується незалежним зовнішнім тактовим сигналом через дільник частоти. Отже тактова частота визначається як результат ділення зовнішньої частоти на дільник. Період перезавантаження таймеру визначається числом завантаження, яке завантажується з регістру завантаження. Стан таймера визначається значенням статусного регістру. Частина бітів регістру визначає режим роботи таймеру і може бути змінена програмістом. Інша частина бітів інформаційна, за їх значенням можна визначити поточний стан таймеру. Ця частина бітів призначена лише для зчитування.

2.10. Контролер переривань

Для організації взаємодії між зовнішніми і внутрішніми процесами у МПС використовується система переривань. Переривання – це тимчасове припинення основного процесу обчислень для виконання деяких запланованих або незапланованих дій, які викликані зовнішніми чинниками, роботою апаратури або роботою програми. Переривання ділять на зовнішні та внутрішні. Зовнішні переривання – це переривання спричинені зовнішніми чинниками. Внутрішні переривання – це переривання спричинені внутрішніми чинниками МПС. В свою чергу внутрішні переривання ділять на апаратні та програмні. Апаратні переривання генеруються безпосередньо апаратними вузлами МПС, а програмні – програмою мікроконтролера.

Внутрішні переривання обробляються безпосередньо процесором, а для попередньої обробки зовнішніх запитів застосовують спеціалізовані пристрої – контролери переривань.

Контролер переривань (англ. Programmable Interrupt Controller, PIC) – мікросхема чи інтегрований блок процесора, що відповідає за послідовну обробку запитів на переривання від різних пристроїв. Використовується для забезпечення коректності процедури переривання в мікропроцесорних системах. До його функцій відносяться ідентифікація переривання, визначення його пріоритетності та черговості виконання, а також запуск відповідної процедури обробки. Структура контролера переривань складається з чотирьох основних блоків: блоку фіксації запитів, блоку дозволу запитів, блоку аналізу пріоритету запитів і керуючого блоку (рис. 2.71). Призначення блоків очевидне із їх назви.

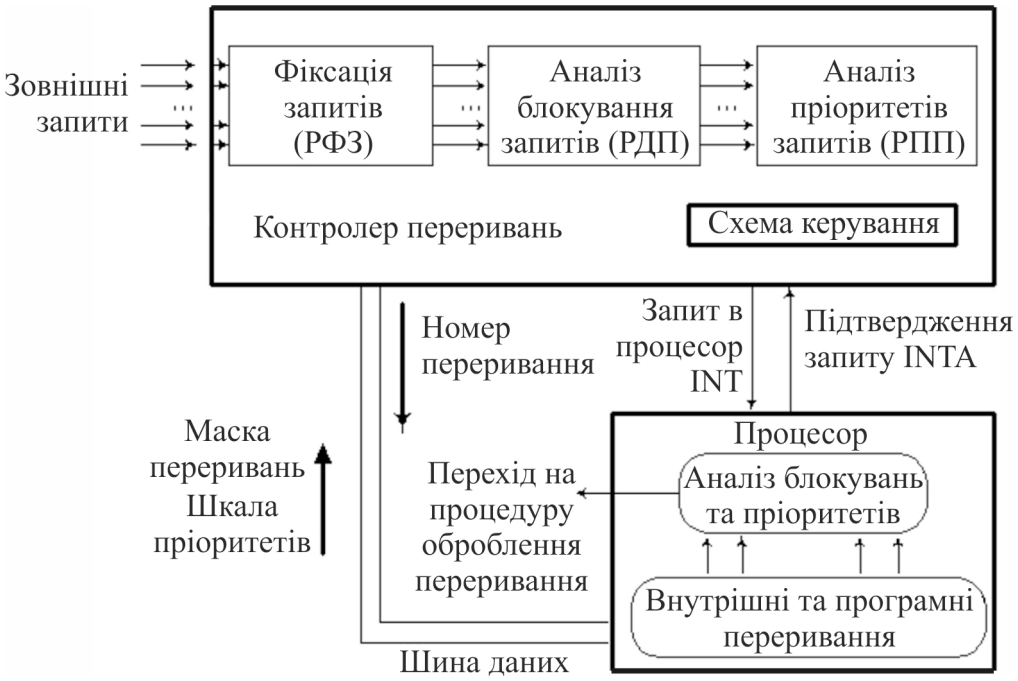


Рис. 2.71. Структурна схема контролера переривань

Блок фіксації запитів фіксує факт появи запиту на переривання. Структура блоку наведена на рис. 2.72. Основою блоку фіксації запитів є регістр фіксації запитів переривань (РФЗ), який і служить для фіксації зовнішніх запитів на переривання. В даному регістрі кожен біт відповідає за

певне переривання. Як правило, коли мав місце якийсь запит на переривання, то відповідний біт РФЗ встановлюється в активний стан (“0” або “1” в залежності від реалізації системи). Коли запит на переривання обслужений, то відповідний біт РФЗ встановлюється у пасивний стан.

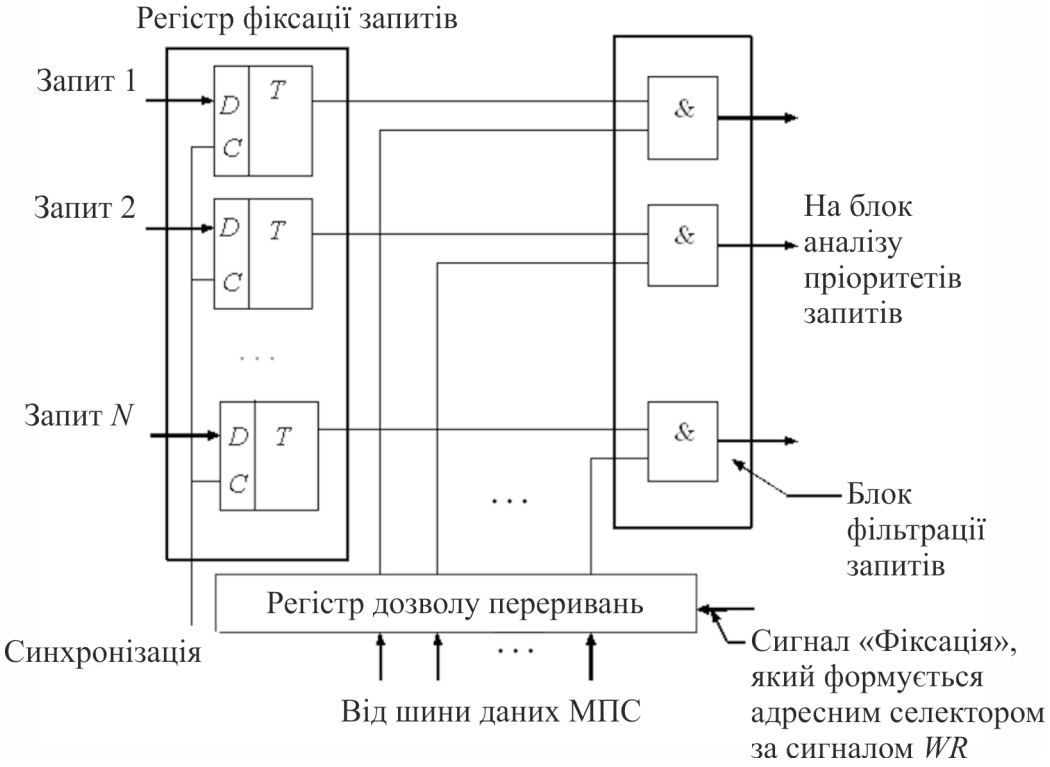


Рис. 2.72. Структурна схема блоку фіксації запитів

Основою блоку дозволу запитів становить регістр дозволу переривань (РДП). В цьому регістрі у вигляді двійкового коду розміщена інформація про те, які переривання дозволені в даній МПС, а які – ні. Кожен біт регістру відповідає за певне переривання. Код РДП прийнято називати маскою переривань. Як правило Код РДП встановлюється програмою МПС при її ініціалізації.

Інформація про пріоритетність запитів переривань міститься у регістрі пріоритету переривань (РПП). Шкала пріоритетів задається програмно при ініціалізації МПС. Кожному перериванню відповідає код пріоритетності. Зазвичай, чим менше числове значення коду, тим вищий пріоритет.

Схема керування координує роботу контролера переривань на основі інформації регістрів контролера переривань і формує запити до мікропроцесора на виконання процедур обробки переривань. Отримавши запит, мікропроцесор зберігає у системному стеку адресу поточного оператора програми, а також іншу інформацію необхідну для поновлення виконання програми після завершення обробки переривання (вміст регістрів контексту) і запускає відповідну процедуру обробки переривання. Після закінчення процедури обробки переривання МП відновлює вміст регістрів контексту і продовжує виконання перерваної програми з того місця на якому вона була перервана.

Адреси процедур обробки переривань розміщуються у таблиці переривань. Один рядок таблиці описує одне переривання і складається з номеру переривання і адреси за якою розміщений перший оператор процедури обробки відповідного переривання.

2.11. Порти введення-виведення

Зазвичай взаємодія між мікроконтролером і зовнішніми пристроями реалізується через порти введення-виведення. У багатьох мікроконтролерах виводи портів введення-виведення служать також для виконання інших функцій. При знайомстві з мікроконтролером дуже важливо зрозуміти, як здійснюється введення-виведення даних. У більшості мікроконтролерів окремі виводи портів введення-виведення можуть бути запрограмовані на введення або виведення даних Типова схема підключення зовнішнього виводу показана на рис 2.73.

Працюючи з портами введення-виведення потрібно звернути увагу на те, що при введенні даних зчитується значення сигналу, який надходить на зовнішній вивід, а не вміст тригера даних. Якщо до зовнішнього виводу підключені виходи інших пристроїв, то вони можуть встановити свій рівень вихідного сигналу, який буде зчитаний замість очікуваного значення даних,

записаних в тригер. У деяких мікроконтролерах існує можливість вибору між читанням даних, встановлених на виході тригера або на зовнішньому виводі.

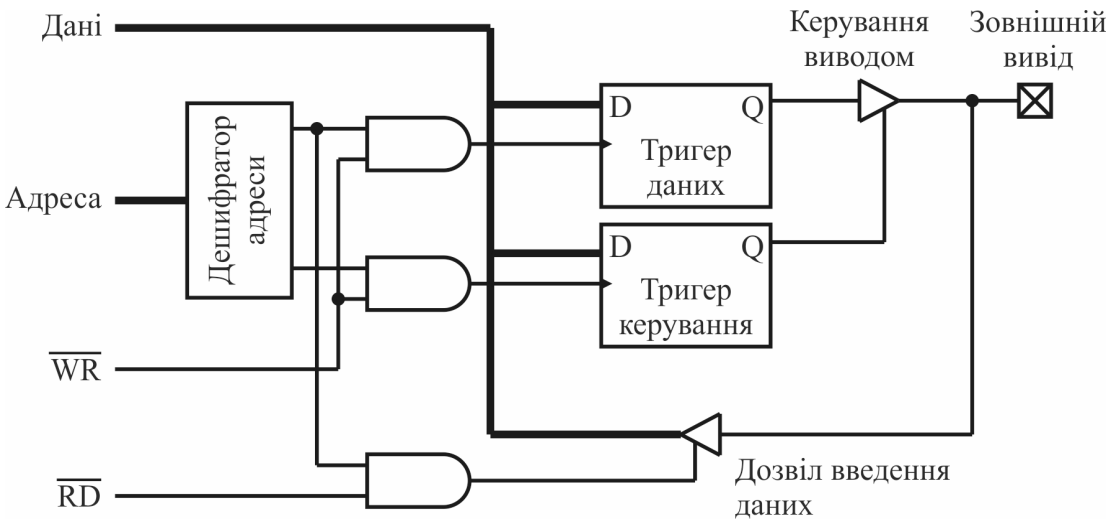


Рис. 2.73. Типова схема підключення виводу порту введення-виведення

Ще один поширений варіант підключення зовнішнього виводу – схема з «відкритим стоком», яка зображена на рис 2.74. Стан даного виводу відповідає стану підключеної до нього шини доти, доки він не буде переключений на роботу в режимі виходу і встановлений у стан “0”. У цьому випадку вивід з'єднаний з «землею» через МДП транзистор, керований логічною схемою «І», підключеною до виходів тригерів управління і даних.

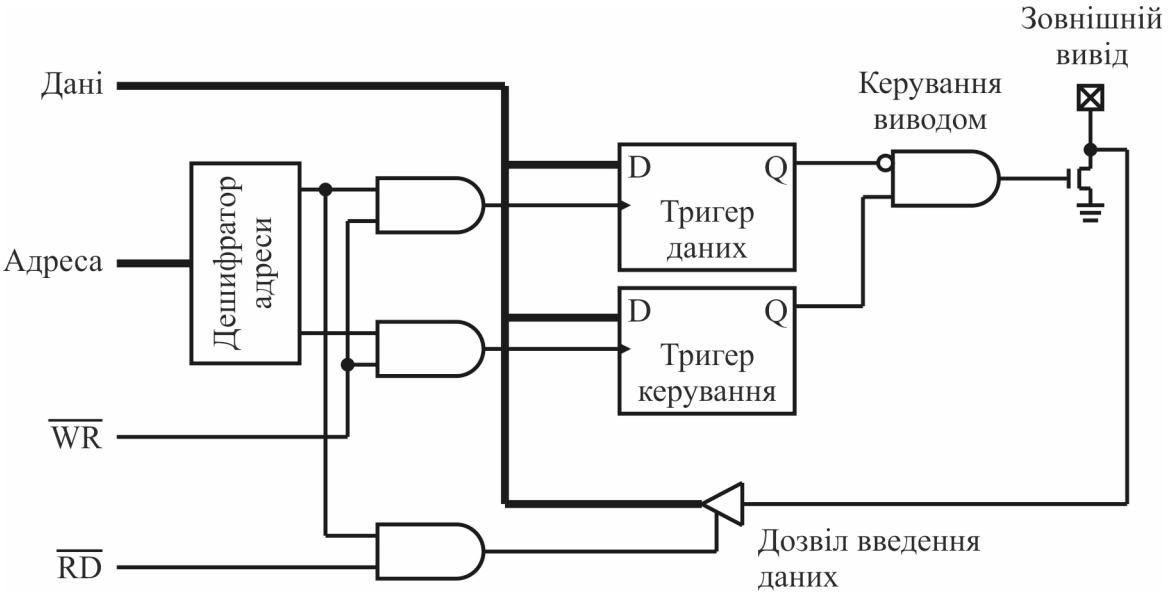


Рис. 2.74. Схема підключення виводу з «відкритим стоком»

Таке підключення виводу дозволяє створювати шини з об'єднанням виходів пристроїв за схемою «монтажне І». У цій схемі шина підключається до напруги живлення (логічна “1”) через резистор, а до потенціалу «землі» (логічний “0”) – через кілька ключів або транзисторів.

2.12. Аналогове введення-виведення

Часто мікроконтролеру доводиться взаємодіяти з аналоговими пристроями. Для реалізації цієї взаємодії в мікроконтролерах передбачено наявність таких вбудованих пристроїв як компаратори, аналогово-цифрові перетворювачі (АЦП), цифрово-аналогові перетворювачі (ЦАП) тощо.

Перетворити аналоговий сигнал в цифровий код, придатний для обробки мікропроцесором, можна шляхом визначення часу заряджання RC-кола за допомогою таймера, або шляхом використання аналогового компаратора чи АЦП, а перетворити цифровий код в аналоговий сигнал можна за допомогою ЦАП.

В першому способі використовується залежність часу заряду конденсатора у RC-колі (за відомих значень опору та ємності) від величини напруги, поданої на RC-коло. Час заряду фіксується за допомогою таймера включеного у режимі лічильника. Кількість імпульсів відрахована лічильником за час заряджання і є кодом, який характеризує величину сигналу. Описаний спосіб має ряд суттєвих недоліків і натепер майже не використовується.

Другий спосіб базується на використанні компаратора. Компаратор – це пристрій, який має два входи і один вихід (рис. 2.75). На схемі компаратор позначається так само, як операційний підсилювач. На один вхід подається стала в часі опорна напруга, а на інший – напруга, яка перевіряється. Перед початком вимірювання на виході компаратора встановлюється “0”. Коли рівень напруги, яка перевіряється, досягає рівня опорної напруги, то на виході компаратора встановлюється “1”.

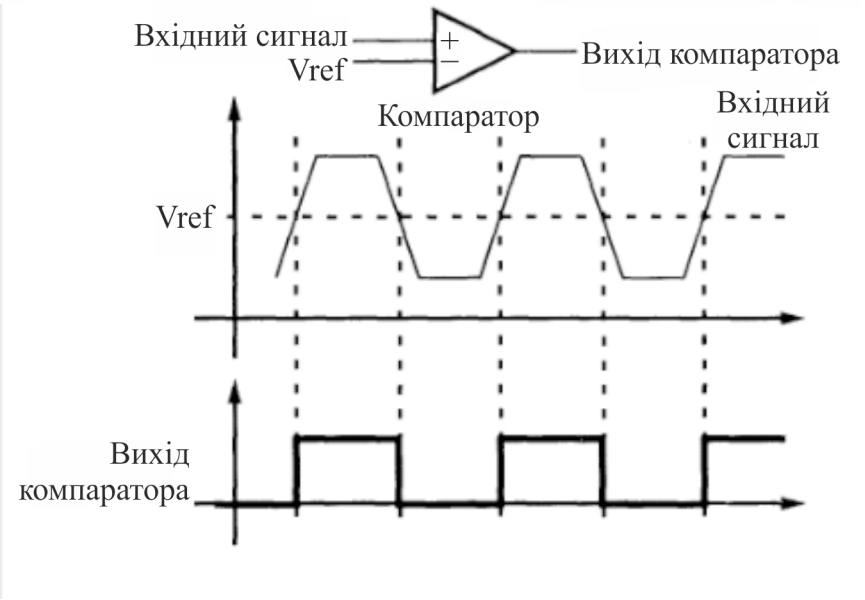


Рис. 2.75. Принцип дії компаратора

Такий спосіб використовується в тому випадку, коли достатньо контролювати момент досягнення сигналом заданого рівня напруги.

Коли необхідно розрізняти рівні напруги сигналу, то використовують АЦП. Аналогово-цифровий перетворювач, АЦП (англ. Analog-to-digital converter, ADC) – пристрій, що перетворює вхідний аналоговий сигнал в дискретний код (цифровий сигнал), який кількісно характеризує амплітуду вхідного сигналу.

Існує кілька основних типів архітектури АЦП, хоча в межах кожного типу існує також безліч варіацій. Різні типи вимірювального обладнання використовують різні типи АЦП. Наприклад, в цифровому осцилографі використовується висока частота дискретизації, але не потрібна висока роздільна здатність. В цифрових мультиметрах потрібна більша роздільна здатність, але можна пожертвувати швидкістю вимірювання. Системи збору даних загального призначення за швидкістю дискретизації і роздільною здатністю зазвичай займають проміжне місце між осцилографами і цифровими мультиметрами. В обладнанні такого типу використовуються АЦП послідовного наближення або сигма-дельта АЦП. Існують також паралельні АЦП для пристроїв, які потребують швидкісної обробки аналогових сигналів з високою роздільною здатністю і завадостійкістю.

Основними характеристиками АЦП є розрядність і частота дискретизації, які характеризують точність і швидкодію АЦП. Ці характеристики залежать від типу АЦП (рис. 2.76). Розрядність АЦП характеризує кількість дискретних значень, які перетворювач може видати на виході. Вимірюється в бітах. Наприклад, АЦП, здатний видати 256 дискретних значень (0..255), має розрядність 8 бітів, оскільки $2^8 = 256$. Отже роздільна здатність складає 1/256 (≈ 0,39 %) від максимальної напруги, яка може бути оцифрована в поточній системі (як правило залежить від конструкції АЦП і від опорної напруги). Для максимальної напруги 10 В роздільна здатність 8-ми бітного АЦП становитиме 10/256 В ≈ 0,039 В.

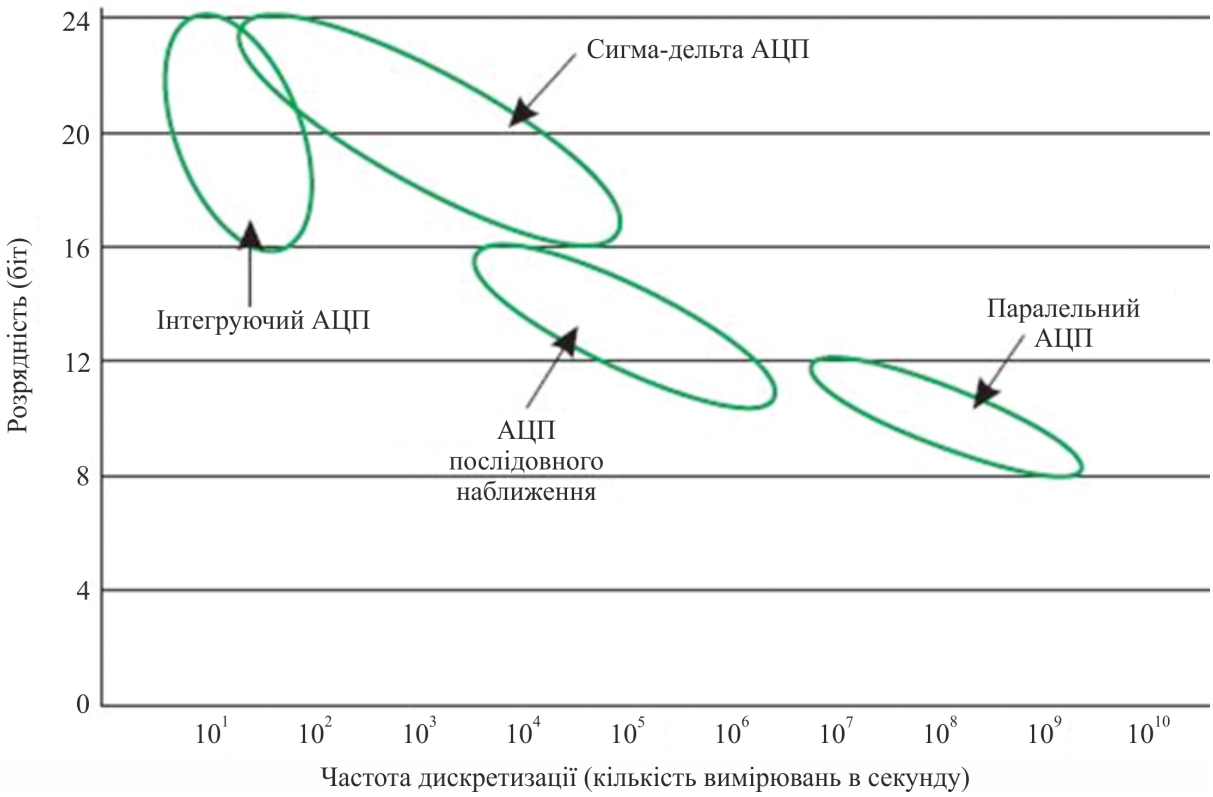


Рис. 2.76. Приблизні параметри для різних типів АЦП

За необхідності максимальної швидкодії використовують паралельні АЦП (рис. 2.77). Вони мають найвищу швидкість перетворення, але порівняно невелику розрядність – від 8-ми до 10 розрядів.

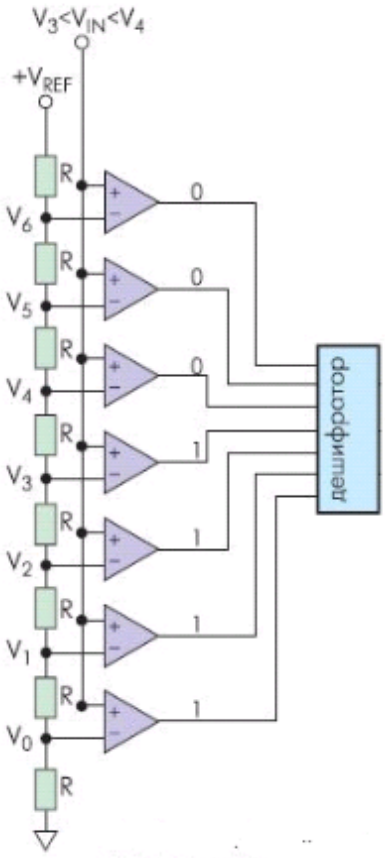


Рис. 2.77. Структура паралельного АЦП

Принцип дії заснований на тому, що зі збільшенням вхідної напруги компаратори послідовно встановлюють свої виходи в логічну одиницю замість логічного нуля, починаючи з компаратора, який відповідає за молодший значущий розряд.

Основними недоліками паралельних АЦП є їх складність, висока вартість, велике енергоспоживання та відносно невисока точність.

Коли необхідна розрядність 12, 14 або 16 розрядів і не потрібна висока швидкість перетворення, а визначальними факторами є невисока ціна і низьке енергоспоживання, то зазвичай застосовують АЦП послідовного наближення (рис. 2.78). На даний момент АЦП послідовного наближення дають змогу вимірювати напругу з точністю до $\approx 0{,}0015\ \%$ з частотою дискретизації від 10^5 до 10^6 відліків/сек.

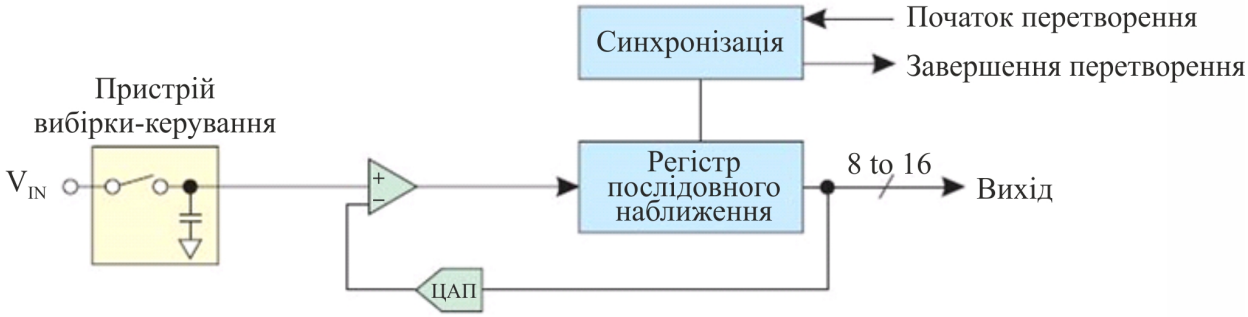


Рис. 2.78. Структура АЦП послідовного наближення

В основі АЦП даного типу лежить спеціальний регістр послідовного наближення. На початку циклу перетворення всі виходи цього регістра встановлюються в логічний “0”, за винятком першого (старшого) розряду. Це формує на виході внутрішнього цифро-аналогового перетворювача (ЦАП) сигнал, значення якого дорівнює половині вхідного діапазону АЦП. А вихід компаратора перемикається в стан, що визначає різницю між сигналом на виході ЦАП і вимірюваною вхідною напругою. Старший біт встановлюється рівним “1”, а всі інші в “0”. Якщо напруга сигналу більша за половину вхідного діапазону, то старший біт залишається рівним “1”, нижчий сусідній з ним розряд установлюється в “1” , а сигнал виходу ЦАП збільшується на чверть (на половину від різниці між поточним сигналом ЦАП і максимальним). Якщо ж менший, то він стає рівним “0”, в “1” установлюється нижчий сусідній з ним розряд, а вихідний сигнал ЦАП зменшується вдвічі. І так відбувається уточнення результату поки не відбудеться співпадання з похибкою зумовленою бітністю АЦП. Інтервал невизначеності на кожному кроці зменшується в два рази. На рис. 2.79 наведено процес послідовного наближення для 8-ми бітного АЦП.

Зі збільшенням бітності АЦП час оцифровування збільшується нелінійно. Хоча теоретично напруга на виході ЦАП для кожного з N внутрішніх тактів перетворення має встановлюватися за однаковий проміжок часу, насправді (внаслідок перехідних процесів) цей проміжок в перших тактах значно більший, ніж в останніх. Тому час перетворення 16-розрядного

АЦП послідовного наближення більше, ніж в два рази перевищує час перетворення 8-розрядного АЦП даного типу.

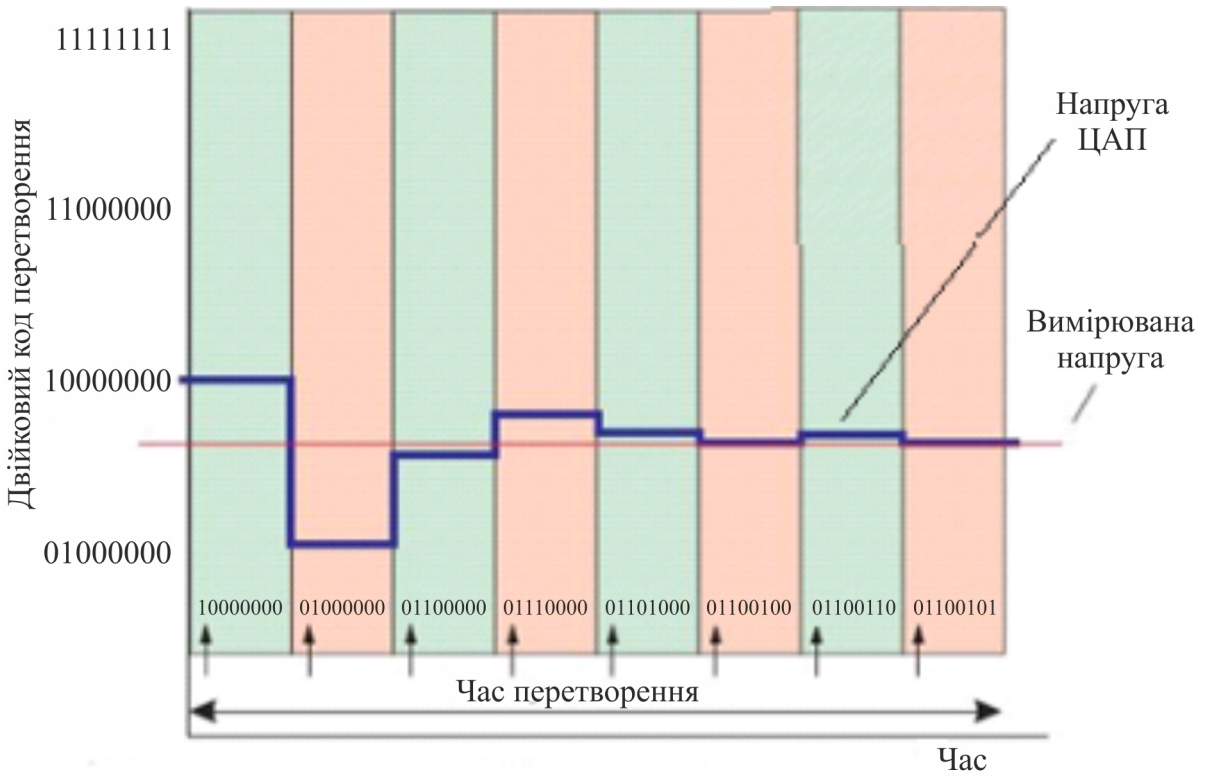


Рис. 2.79. Процес послідовного наближення

З наведеного опису слідує, що для реалізації АЦП послідовного наближення використовують ЦАП. Цифро-аналоговий перетворювач – електронний пристрій для перетворення цифрового (як правило двійкового) сигналу на аналоговий.

У найпростішому випадку ЦАП може бути реалізований за схемою суматора струмів, наприклад, на основі диференціального підсилювача у інвертуючому ввімкненні, на інверсному вході якого відбувається додавання струмів (рис. 2.80). При такій схемі підключення вихідна напруга буде пропорційною сумі вхідних струмів.

Опори вхідних резисторів R_1–R_n (див. рис. 2.80) мають співвідноситись як основа системи числення, для двійкової системи кожен наступний – в 2 рази більший.

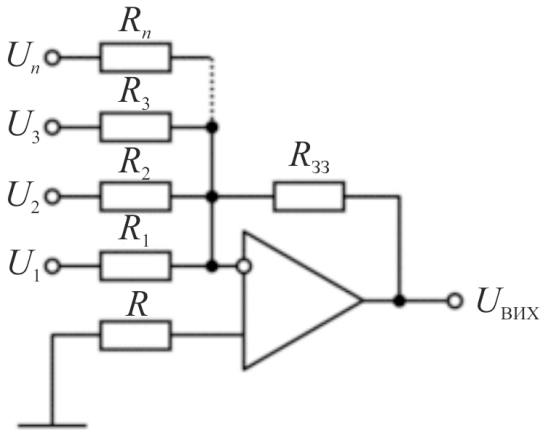


Рис. 2.80. Схема найпростішого ЦАП

Для проведення більшості вимірювань часто не потрібна така висока швидкість перетворення, яку дає АЦП послідовного наближення, зате необхідна велика роздільна здатність. Сигма-дельта АЦП можуть забезпечувати роздільну здатність до 24 розрядів, але при цьому поступаються в швидкості перетворення. Так, в сигма-дельта АЦП при 16 розрядах можна отримати частоту дискретизації до 10^5 відліків/с, а при 24 розрядах ця частота зменшується до 10^3 відліків/с і менше, залежно від пристрою.

Зазвичай сигма-дельта АЦП застосовуються в різноманітних системах збору даних і у вимірювальному обладнанні (вимірювання тиску, температури, ваги тощо), коли не потрібна висока частота дискретизації і необхідна роздільна здатність більше, ніж та, яку забезпечують 16 розрядів.

Принцип дії сигма-дельта АЦП складніший. Ця архітектура відноситься до класу інтегруючих АЦП. Але основна особливість сигма-дельта АЦП полягає в тому, що частота проходження вибірок, при яких власне і відбувається аналіз рівня напруги вимірюваного сигналу, істотно перевищує частоту появи відліків на виході АЦП (частоту дискретизації). Ця частота проходження вибірок називається частотою передискретизації. Так, сигма-дельта АЦП зі швидкістю перетворення 10^5 відліків/с, в якому використовується частота передискретизації в 128 разів більше, буде робити вибірку значень вхідного аналогового сигналу з частотою $12{,}8 \times 10^6$ відліків/с.

Блок-схема сигма-дельта АЦП першого порядку наведена на рис. 2.81. Аналоговий сигнал подається на інтегратор, виходи якого приєднані до компаратора, який в свою чергу приєднаний до 1-розрядного ЦАП у петлі зворотного зв'язку. Шляхом серії послідовних ітерацій інтегратор, компаратор, ЦАП і суматор дають потік послідовних бітів, в якому міститься інформація про величину вхідної напруги.

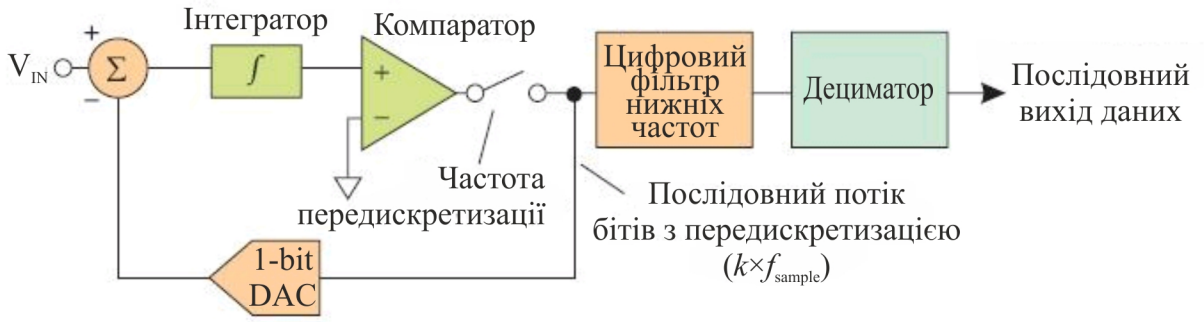


Рис. 2.81. Структура сигма-дельта АЦП

Результуюча цифрова послідовність потім подається на фільтр нижніх частот (ФНЧ) для придушення компонентів з частотами вище частоти Котельникова (вона становить половину частоти дискретизації АЦП). Після видалення високочастотних складових наступний вузол – дециматор – проріджує дані. У розглянутому нами АЦП дециматор буде залишати 1 біт з кожних отриманих 128 у вихідний цифровий послідовності.

Оскільки внутрішній ФНЧ у сигма-дельта АЦП являє собою невід'ємну частину для здійснення процесу перетворення, то час встановлення ФНЧ стає фактором, який обмежує швидкодію АЦП. Тому необхідно враховувати перехідні процеси при стрибкоподібній зміні вхідного сигналу.

Важливою перевагою сигма-дельта АЦП є те, що всі його внутрішні вузли можуть бути виконані інтегральним способом на одному кремнієвому кристалі, що значно знижує вартість кінцевих пристроїв і підвищує стабільність характеристик АЦП.

Ми розглянули лише кілька найпоширеніших видів АЦП. Насправді кількість видів АЦП значно більша. Крім розглянутих видів також існують

АЦП диференціального кодування, АЦП порівняння з зубчастим сигналом, АЦП Уілкінсона, інтегруючі АЦП, АЦП з урівноваженням заряду, двошвидкісні АЦП, конвеєрні АЦП тощо.

Контрольні запитання

1. Назвіть основні принципи побудови мікропроцесорних систем.
2. Що означає модульний принцип?
3. Що означає магістральний принцип?
4. Дайте визначення поняття «часове мультиплексування».
5. Дайте визначення поняття «системна шина».
6. На які групи розділяється інформаційний потік?
7. Перелічіть шини нижнього рівня, з яких складається системна шина.
8. Назвіть характерні особливості CISC-мікропроцесорів.
9. Назвіть характерні особливості RISC-мікропроцесорів.
10. Наведіть структурну схему мікропроцесорної системи з принстонським типом доступу до пам’яті.
11. Перелічіть основні недоліки архітектури принстонського типу.
12. Наведіть структурну схему мікропроцесорної системи з гарвардським типом доступу до пам’яті.
13. Перелічіть основні недоліки архітектури гарвардського типу.
14. Дайте визначення поняття «запам’ятовуючий пристрій».
15. Дайте визначення поняття «інформаційна ємність запам’ятовуючого пристрою».
16. Поясніть термін «організація запам’ятовуючого пристрою».
17. Опишіть принцип роботи статичного запам’ятовуючого пристрою.
18. Опишіть принцип роботи динамічного запам’ятовуючого пристрою.
19. Що представляє собою масковий запам’ятовуючий пристрій?
20. Опишіть принцип дії PROM пам’яті.
21. Опишіть принцип дії EPROM пам’яті.

22. Опишіть принцип дії EEPROM пам’яті.
23. Що таке Flash пам'ять?
24. Назвіть функції, які виконує арифметико-логічний пристрій.
25. Наведіть структурну схему арифметико-логічного пристрою.
26. Назвіть групи, на які поділяють операції виконувані арифметико-логічним пристроєм.
27. Наведіть класифікацію арифметико-логічних пристроїв за способом дії над операндами.
28. Назвіть групи, на які поділяють арифметико-логічні пристрої за способом представлення чисел.
29. Наведіть класифікацію арифметико-логічних пристроїв за характером використання елементів і вузлів.
30. Перелічіть рівні, на які поділяють процес обміну інформацією.
31. Назвіть функції прикладного рівня.
32. Назвіть функції представницького рівня.
33. Назвіть функції сеансового рівня.
34. Назвіть функції транспортного рівня.
35. Назвіть функції мережевого рівня.
36. Назвіть функції канального рівня.
37. Назвіть функції фізичного рівня.
38. Наведіть класифікацію інтерфейсів за способом передачі данних.
39. Чим відрізняються послідовні інтерфейси від паралельних?
40. Чим відрізняються синхронні інтерфейси від асинхронних?
41. Дайте визначення послідовного периферійного інтерфейсу.
42. Перелічіть цифрові сигнали, які використовують в SPI.
43. Опишіть схеми підключення послідовних периферійних інтерфейсів.
44. Намалюйте схему обміну даними при використанні послідовних периферійних інтерфейсів.
45. Опишіть режими роботи послідовних периферійних інтерфейсів. Поясніть відмінності між ними.

46. Назвіть переваги та недоліки послідовних периферійних інтерфейсів.
47. Дайте визначення паралельного периферійного інтерфейсу.
48. Назвіть переваги і недоліки паралельного периферійного інтерфейсу.
49. Намалюйте схему підключення паралельних периферійних інтерфейсів.
50. Що являє собою послідовна асиметрична шина?
51. Намалюйте схему підключення до послідовної асиметричної шини.
52. Назвіть керуючі сигнали, які використовують при передаванні даних за допомогою послідовної асиметричної шини.
53. Наведіть структуру пакету при передаванні даних за допомогою послідовної асиметричної шини?
54. Назвіть переваги та недоліки послідовної асиметричної шини.
55. Що являє собою асинхронний послідовний інтерфейс RS-232?
56. Перелічіть режими роботи, які підтримує асинхронний послідовний інтерфейс RS-232.
57. Опишіть структуру повідомлення асинхронного послідовного інтерфейсу RS-232.
58. Що являє собою асинхронний послідовний інтерфейс RS-422?
59. Що являє собою асинхронний послідовний інтерфейс RS-485?
60. Намалюйте схему з’єднання двох пристроїв RS-422.
61. Намалюйте схему з’єднання кількох пристроїв RS-422 у спільну мережу.
62. Що являє собою лінія RS-422?
63. Перелічіть варіанти використання інтерфейсу RS-485.
64. Назвіть недоліки і переваги чотирипровідної схеми підключення інтерфейсу RS-485.
65. Назвіть недоліки і переваги двопровідної схеми підключення інтерфейсу RS-485.
66. Що являє собою синхронний інтерфейс CAN?
67. Опишіть структуру повідомлень при використанні синхронного інтерфейсу CAN.

68. Назвіть типи повідомлень, які використовують при передаванні інформації за допомогою синхронного інтерфейсу CAN.
69. Що являє собою CAN-мережа?
70. Що являє собою інтерфейс USB?
71. Перелічіть сигнали, які використовує інтерфейс USB.
72. Намалюйте принципову схему з'єднання USB-пристроїв.
73. Наведіть класифікацію бездротових мереж за територіальною ознакою.
74. Наведіть характеристики стандарту 802.11a.
75. Наведіть характеристики стандарту 802.11b.
76. Наведіть характеристики стандарту 802.11g.
77. Наведіть характеристики стандарту 802.11i.
78. Наведіть характеристики стандарту 802.11n.
79. Наведіть характеристики стандарту 802.11r.
80. Опишіть формат кадру у сімействі стандартів 802.11.
81. Наведіть характеристики стандарту 802.16.
82. Наведіть характеристики стандарту 802.15.1
83. Наведіть характеристики стандарту 802.15.3.
84. Наведіть характеристики технології Ultra Wideband.
85. Наведіть характеристики стандарту 802.15.4.
86. Намалюйте схему підключення кварцового резонатора у якості тактового генератора.
87. Намалюйте схему підключення RC-генератора у якості тактового генератора.
88. Намалюйте схему підключення кола скидання до початкового стану.
89. Намалюйте структурну схему таймера мікропроцесора.
90. У які способи може бути використаний таймер мікропроцесора?
91. Що таке широтно-імпульсна модуляція?
92. Намалюйте структурну схему пристрою широтно-імпульсної модуляції.
93. Намалюйте структурну схему сторожового таймеру.
94. Що таке контролер переривань?

95. Перелічіть функції контролеру переривань.
96. Що таке компаратор? Опишіть принцип його дії.
97. Що таке АЦП?
98. Перелічіть типи АЦП.
99. Назвіть основні характеристики АЦП.
100. Що таке ЦАП?
101. Наведіть схему простого ЦАП.

3. СІМЕЙСТВО МІКРОКОНТРОЛЕРІВ STM32

3.1. Загальні відомості

Сімейство ARM Cortex – це покоління процесорів із стандартизованою архітектурою ARMV7 M, яке призначено для вирішення широкого кола технічних задач. Сімейство Cortex має три основних профілі: профіль A для високопродуктивних додатків, R – для додатків реального часу, M – для бюджетних програм. Після літери вказується число, яке позначає рівень продуктивності: від 1 для мінімального рівня, до 8 для максимального. У STM32 використовується профіль Cortex-M3, розроблений спеціально для систем, які поєднують в собі високу продуктивність і низьке енергоспоживання.

Ядро Cortex-M3 включає систему переривань, системний таймер, систему налагодження і карту пам'яті. Адресний простір Cortex-M3 розділений на строго визначені області для коду програми, SRAM, периферійних і системних пристроїв. Ядро Cortex-M3 виконано за Гарвардською архітектурою і, відповідно, має кілька шин, що дає змогу виконувати операції паралельно. Сімейство Cortex може оперувати фрагментованими даними (unaligned data), що відрізняє його від попередніх архітектур ARM, що гарантує максимальну ефективність використання внутрішнього статичного ОЗП. Сімейство Cortex також має можливості встановлення і скидання бітів в межах двох областей пам'яті розміром 1 Мбайт за методом bit banding (бітові смуги). Цей метод надає ефективний доступ до регістрів і прапорів ПВВ, розташованих в області статичного ОЗП, і виключає необхідність інтеграції повнофункціонального бітового процесора. Узагальнена функціональна структура мікроконтролера сімейства STM32 наведена на рисунку 3.1.

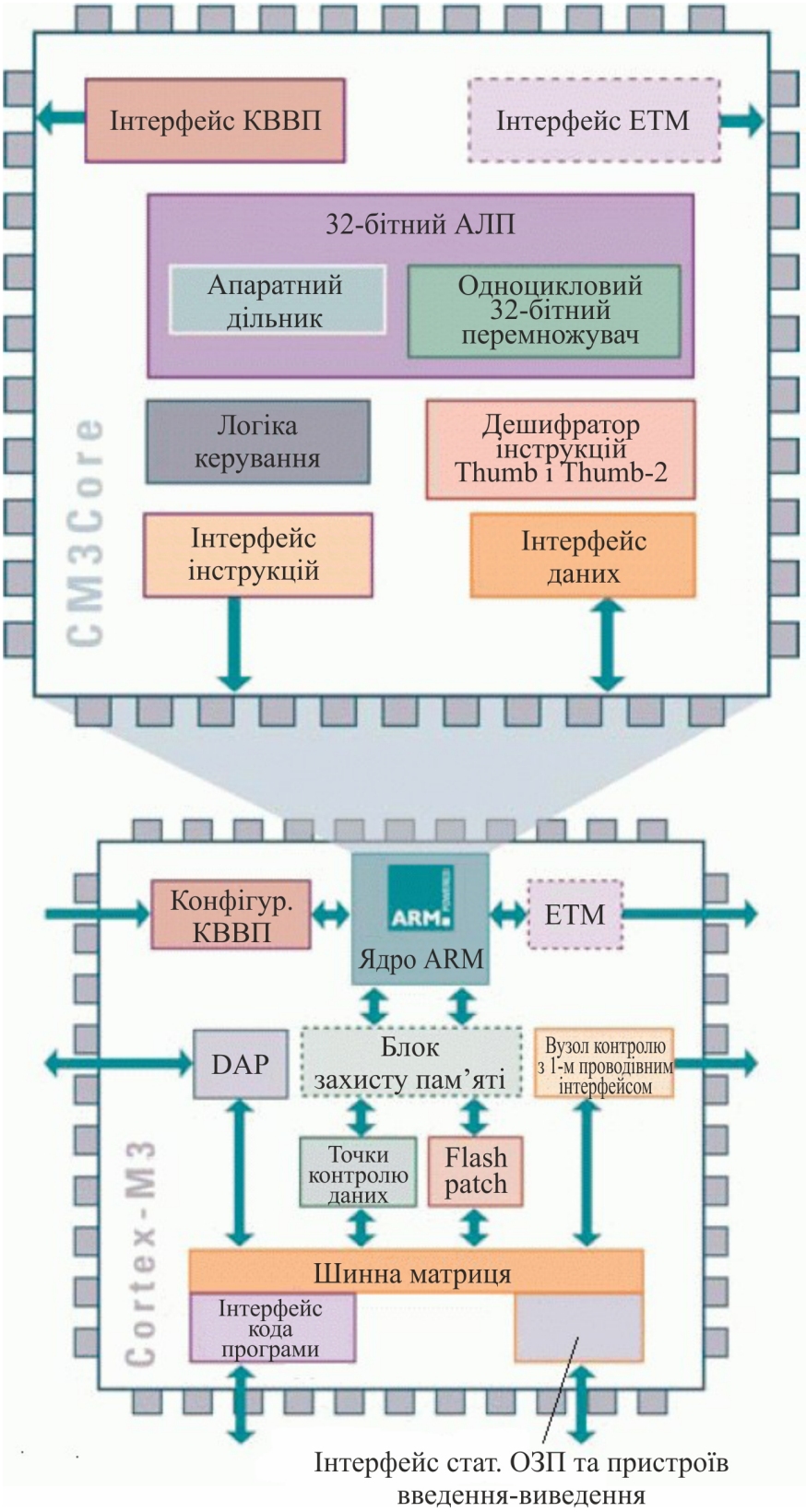


Рис. 3.1. Функціональна структура мікроконтролера сімейства STM32: КВВП – контролер векторизованих вкладених переривань (NVIC – Nested vector interrupt controller); ETM – external trace macrocell (макрокомірка зовнішнього трасування – інтерфейс для внутрішньосхемного налагодження програми)

Ядро Cortex-M3 32-бітове і підтримує два режими роботи: потоковий режим (Thread) і режим обробника (Handler), для кожного з яких можна налаштувати свої власні стеки. Завдяки цьому є можливість розробки більш інтелектуального програмного забезпечення і підтримки операційних систем реального часу (ОСРЧ). В ядро Cortex також входить 24-бітний автоматично перезавантажуваний таймер, призначений для генерації періодичних переривань і використовуваний ядром ОСРЧ. ЦПП сімейства Cortex підтримує набір інструкцій ARM Thumb-2. Він являє собою суміш 16- і 32-бітних інструкцій, що дає змогу досягти продуктивності 32-бітного набору інструкцій ARM і щільності коду, властивої 16-бітного набору інструкцій. Набір інструкцій Thumb-2 орієнтований на компілятори мов C/C++, що дозволяє писати програми для Cortex-мікроконтролерів повністю на мові C.

Одним з ключових компонентів ядра Cortex-M3 є контролер векторизованих вкладених переривань (КВВП). КВВП визначає вектори переривань для 240 джерел, для кожного з яких може бути встановлений програмними засобами пріоритет. Вектор переривання – закріплений за пристроєм номер, який ідентифікує відповідний обробник переривань, тобто функцію, яка реагує на подію і обслуговує її, після чого повертає управління в перерваний код. Вектори переривань об'єднуються в таблицю векторів переривань, яка містить адреси обробників переривань. При розробці КВВП особливу увагу було приділено збільшенню швидкості обробки переривань. З моменту отримання запиту на переривання до виконання першої команди процедури обробки переривання проходить всього лише 12 циклів. Частково це досягнуто за рахунок автоматичних операцій зі стеком, які виконує спеціальний мікрокод всередині ЦПП. У випадку одночасного виникнення кількох переривань КВВП використовує спосіб впорядкованої обробки переривань з затримкою перед викликом наступної процедури обробки переривання всього лише 6 циклів. У цьому разі переривання з більш високим пріоритетом може витіснити переривання з нижчим пріоритетом, не витрачаючи при цьому додаткових циклів ЦПП. Структура переривань також

тісно пов'язана з підтримуваними ядром Cortex-M3 економічними режимами роботи. Передбачена можливість програмування ЦПП на автоматичний перехід в економічний режим роботи після закінчення обробки переривання.

Сімейство STM32 складається з двох груп: Performance Line та Access Line, сумісних по розташуванню виводів і програмному забезпеченню. Група Performance Line працює на тактових частотах до 72 МГц і оснащена повним набором ПВВ, а група Access Line працює на частотах до 36 МГц та інтегрує обмежений набір ПВВ (рис. 3.2), функціональна структура мікроконтролерів різних груп представлена на рисунках 3.3, 3.4.

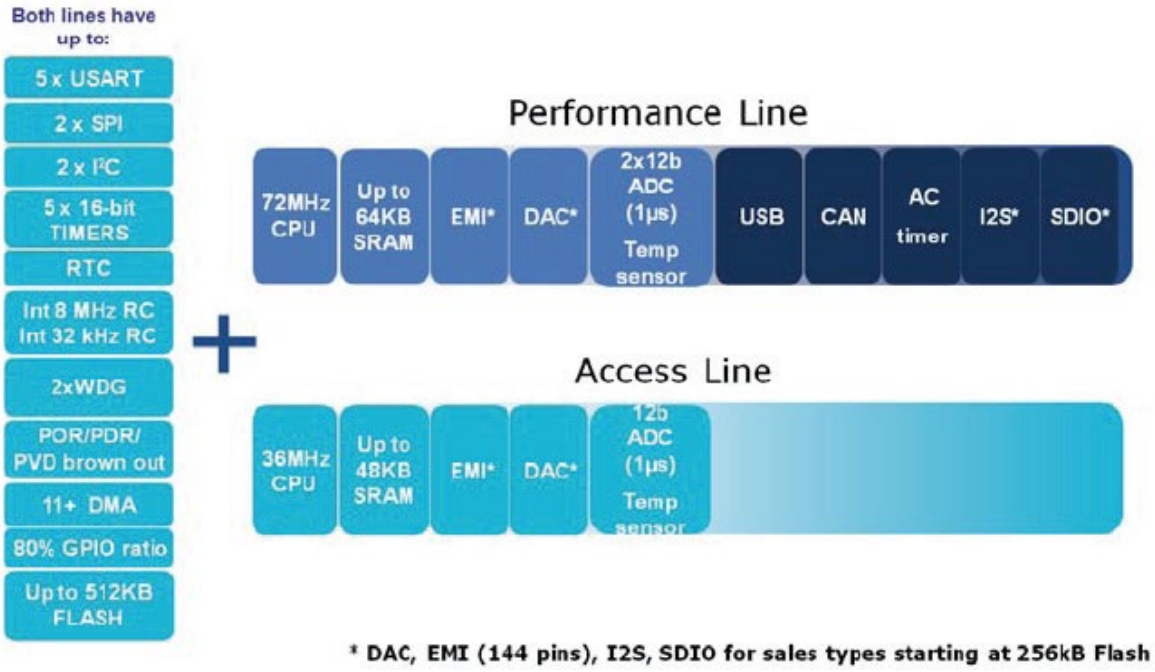


Рис. 3.2. Набір пристроїв у групах мікроконтролерів сімейства STM32

Сучасні мікроконтролери, крім високої продуктивності та багатофункціональності, мають, забезпечувати високий рівень безпеки. Дана вимога була врахована у МК STM32. У них інтегрований ряд апаратних блоків, які відповідають за безпеку роботи мікроконтролера, таких як: малопотужний супервізор джерела живлення, система захисту синхронізації та два окремих незалежних сторожових таймера. Інтегрована система захисту синхронізації здатна виявляти перебої в роботі основного зовнішнього генератора і, у разі необхідності, безпечно перемикатися на роботу від внутрішнього RC-генератора з частотою 8 МГц.

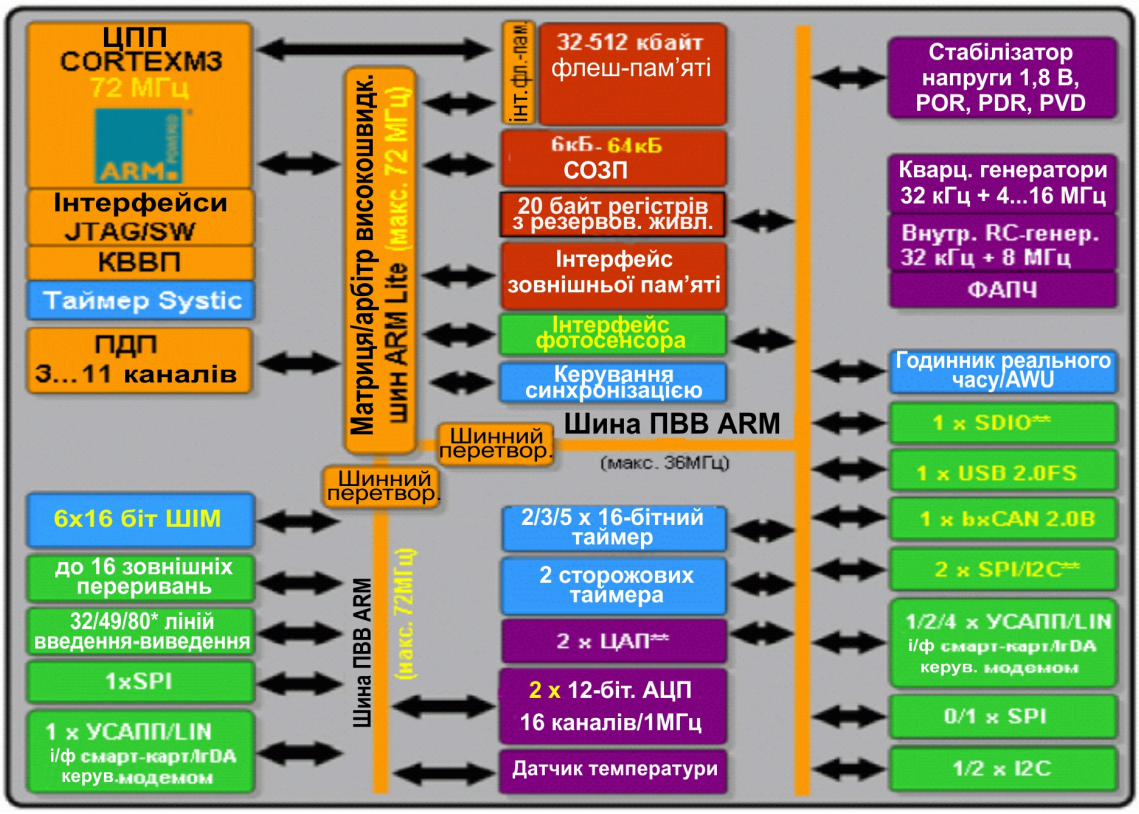


Рис. 3.3. Функціональна структура мікроконтролерів сімейства STM32 групи Performance Line

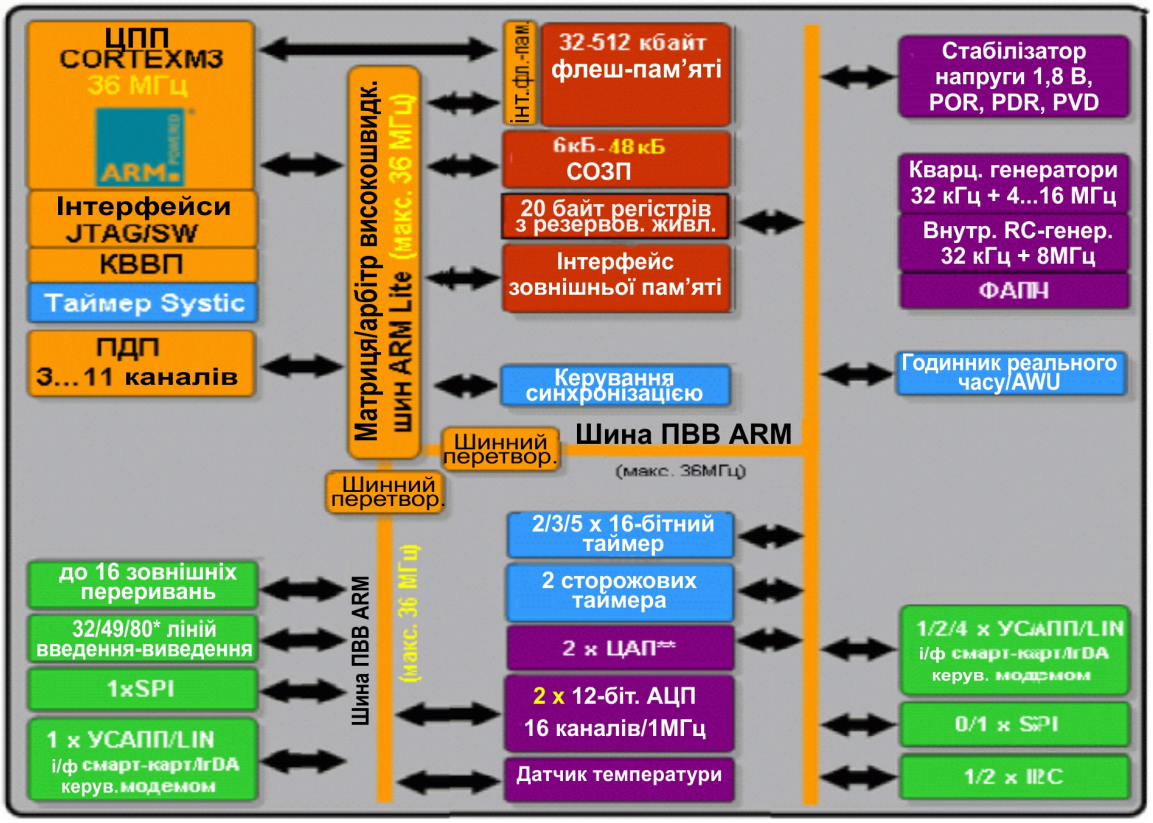


Рис. 3.4. Функціональна структура мікроконтролерів сімейства STM32 групи Access Line

Для захисту від зовнішнього втручання Flash пам'ять МК STM32 оснащена програмованим блокуванням читання через порт внутрішньосхемного налагодження. Після активізації блокування неможливо записати що-небудь у Flash пам'ять, що виключає можливість внесення змін в таблицю векторів переривань і в програму. Також для захисту програм від несанкціонованого копіювання в МК інтегровані годинник реального часу і область енергонезалежного статичного ОЗП, які живляться від окремого резервного батарейного джерела. У цій області є вхід реагування на зовнішнє втручання. При зміні стану на даному вході генерується переривання і обнуляється вміст енергонезалежного статичного ОЗП.

Для підвищення ефективності розробки програмного забезпечення МК Cortex-M3 оснащені системою налагодження програмного коду CoreSight. Доступ до системи CoreSight організований через спеціальний порт Debug Access Port (DAP порт), зв'язок з якими здійснюється, або за стандартним інтерфейсом JTAG або по послідовному 2-провідного інтерфейсу. Система CoreSight, дає змогу проводити внутрішньосхемне трасування коду, використання, встановлення контрольних точок та читання і змінювання стану контрольних регістрів.

3.2. Засоби розробки для мікроконтролерів сімейства STM32

Для МК сімейства STM32 існує достатньо широкий вибір програмних засобів розробки. Наведемо лише основні та найпопулярніші програмні пакети (табл. 3.1).

Найбільш популярними (але і найдорожчими) серед розробників є інструментарії від компаній Keil і IAR Systems. Це обумовлено найбільш розвиненими С-інструментами з точки зору оптимізації та компактності коду. Також, крім провідних позицій в С-інструментаріях, дані компанії надають широкі набори додаткового програмного забезпечення: операційні системи реального часу, USB-стеки, TCP/IP-стеки і багато іншого, але за

додаткову плату. До того ж ці дві компанії забезпечують хорошу технічну підтримку.

Таблиця 3.1.

Засоби розробки програмного забезпечення для STM32

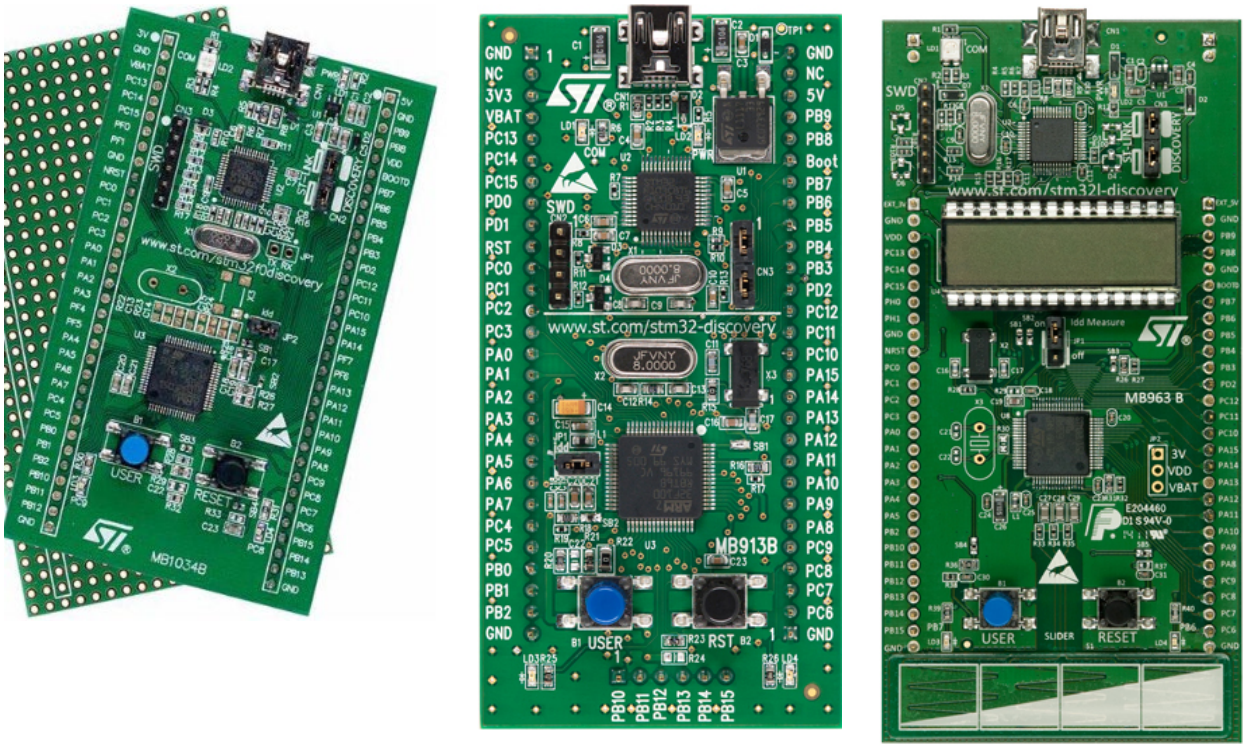
Інструмент	**Середовище розробки**	**C/C++ компілятор**	**Обмеження**	**Програматор**	**Посилання**
IAR Systems	Embedded Workbench	IAR C/C++	32 Кб або повна з обмеженням на 30 днів	J-Link ST-Link	www.iar.com
Keil	uVision (MDK-ARM)	Keil C/C++	32 Кб	ULink ST-Link	www.keil.com
Raisonance	Ride7 + RKIT-ARM	GCC C/C++	без обмежень по відладці	RLink	www.raisonance.com
Atollic	TrueStudio	GCC C/C++	без обмежень по функціоналу	ST-Lin STICE	www.atollic.com
Open source	Eclipse	GCC C/C++	без обмежень	ARM-Link	www.eclipse.org

Також популярними є засоби на основі компілятора GCC. Існують як платні їх варіанти, так і безкоштовні. GCC є лідером за кількістю підтримуваних процесорів і операційних систем. Як приклад варіанту платних засобів у зведеній таблиці наведено інструментарії від компаній Raisonance і Atollic. У порівнянні з двома вище описаними варіантами користувач отримує за меншу плату повноцінний С-інструментарій з середовищем розробки і технічною підтримкою. Також існують варіанти повністю безкоштовного інструментарію, наприклад, середовище розробки Eclipse і компілятор GCC.

Виробник кожного середовища розробки, надає власний варіант програматора, але підтримуються і пристрої інших виробників. Більшість середовищ розробки підтримують ST-Link – найдешевший варіант на сьогоднішній день. Потрібно визнати, що «рідні» програматори надають

більше можливостей з налагодження програмного забезпечення, хоча різниця в ціні також відчутна. Додамо, що при виборі програматора існує кілька варіантів від одного виробника: як простіші з підтримкою основних налагоджувальних функцій, так і професійні версії з підтримкою повного спектру функцій налагодження і трасування. Наприклад, програматори для IAR Embedded Workbench – J-Link і J-Trace, для Keil uVision – ULink і ULink-Pro.

Крім того для полегшення процесу вивчення мікроконтролерів сімейства STM32 існує ряд налагоджувальних плат від ST Microelectronics, найбільш доступними з яких є STM32F0DISCOVERY (рис. 3.5, *а*) STM32VLDISCOVERY (рис. 3.5, *б*) та STM32LDISCOVERY (рис. 3.5, *в*).



а *б* *в*

Рис. 3.5. Налагоджувальні плати STM32F0DISCOVERY (*а*) STM32VLDISCOVERY (*б*) та STM32LDISCOVERY (*в*)

Технічні характеристики налагоджувальних плат наведено у таблиці 3.2.

Таблиця 3.2.

Характеристики налагоджувальних плат на основі STM32

STM32F0DISCOVERY	STM32VLDISCOVERY	STM32LDISCOVERY
Характеристики: - МК: STM32F051R8T6 (Cortex M0, 48МГц, flash 64Кб, RAM 8Кб); - вбудований ST-link / V2, який можна використовувати окремо від плати; - живлення від USB або від зовнішнього джерела 3/5В; - 4 світлодіоди і 2 кнопки; - інтерфейси: USART, SPI, I2C, HDMI; - таймери 16 і 32 біт.	**Характеристики:** - МК: STM32F100RBT6B (Cortex M3, 24МГц, flash 128Кб, RAM 8Кб); - вбудований ST-link / V2, який можна використовувати окремо від плати; - живлення від USB або від зовнішнього джерела 3/5В; - 4 світлодіода і 2 кнопки; - інтерфейси: USART, SPI, I2C, HDMI; - таймери 16 і 32 біт;	**Характеристики:** - МК STM32L152RBT6 (Cortex M3, 32МГц, flash 128Кб, RAM 8Кб, EEPROM 4Кб) - інтерфейси: USB, USART, SPI, I2C; - 8 таймерів; - 12-бітний АЦП 24-канали; -12-бітний ЦАП; - годинник реального часу; - контролер LCD 8x40 - вбудований ST-link / V2. - LCD дисплей 24x8; - 4 світлодіода; - 2 кнопки; - сенсорна клавіатура.

3.3. ЦПП Cortex

Щоб підкреслити відмінності між вбудованим ядром Cortex і внутрішнім RISC ЦПП, далі будуть використовуватися терміни процесор Cortex і ЦПП Cortex, відповідно. У даному розділі будуть розглянуті ключові особливості ЦПП Cortex.

3.3.1. Архітектура ЦПП Cortex

Основою процесорів Cortex є 32-бітове RISC ЦПП. Цей ЦПП має спрощену версію програмної моделі ARM7 / 9, але розширену систему команд з хорошою підтримкою цілочислової арифметики, поліпшеною системою бітової маніпуляції та підвищеною продуктивністю.

ЦПП Cortex здатний виконувати більшу частину команд за один цикл. Як у ARM7 і ARM9 це досягається за допомогою триступінчастого конвеєра (рис. 3.6).

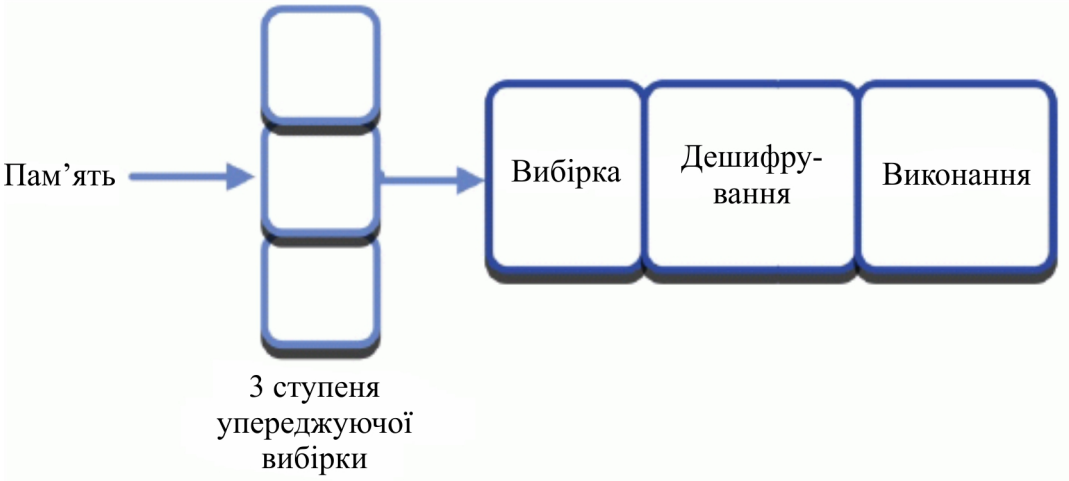


Рис. 3.6. Триступінчастий конвеєр

Під час виконання однієї інструкції, наступна дешифрується, а третя зчитується з пам'яті. Цей механізм відмінно працює з лінійним кодом, але, якщо потрібно виконати перехід, то, перш ніж продовжити виконання коду програми, буде потрібно очищення і перезавантаження конвеєра. У ЦПП ARM7 і ARM9 переходи істотно обмежують продуктивність виконання коду програми. Щоб уникнути цього, триступінчастий конвеєр ЦПП Cortex оснащений логікою передбачення переходів. Це означає, що при досягненні інструкції умовного переходу виконується упереджуюча вибірка і, в результаті, інструкції умовного переходу будуть доступні для виконання і зниження продуктивності не відбудеться.

Найгірший випадок – непрямий перехід, при якому упереджуюча вибірка неможлива. В цьому випадку єдиним виходом із ситуації є очищення і перезавантаження конвеєра, що знижує продуктивність ядра.

ЦПП Cortex – це RISC-процесор, виконаний за схемою читання / записування. Для виконання операцій обробки даних спочатку необхідно помістити операнди з пам'яті в центральний регістровий файл, потім виконати задану операцію над даними і перезаписати результат у пам'ять (рис. 3.7).

Отже, вся активність програми фокусується навколо файлу регістрів ЦПП. Даний файл містить шістнадцять 32-бітних регістрів (рис. 3.8, *а*). Регістри R0–R12 – регістри загального призначення, які можуть

використовуватися для зберігання програмних змінних. У регістрів R13–R15 є особливі функції в рамках ЦПП Cortex.

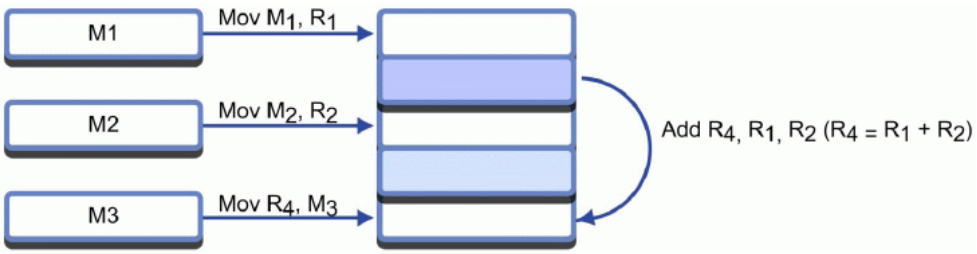
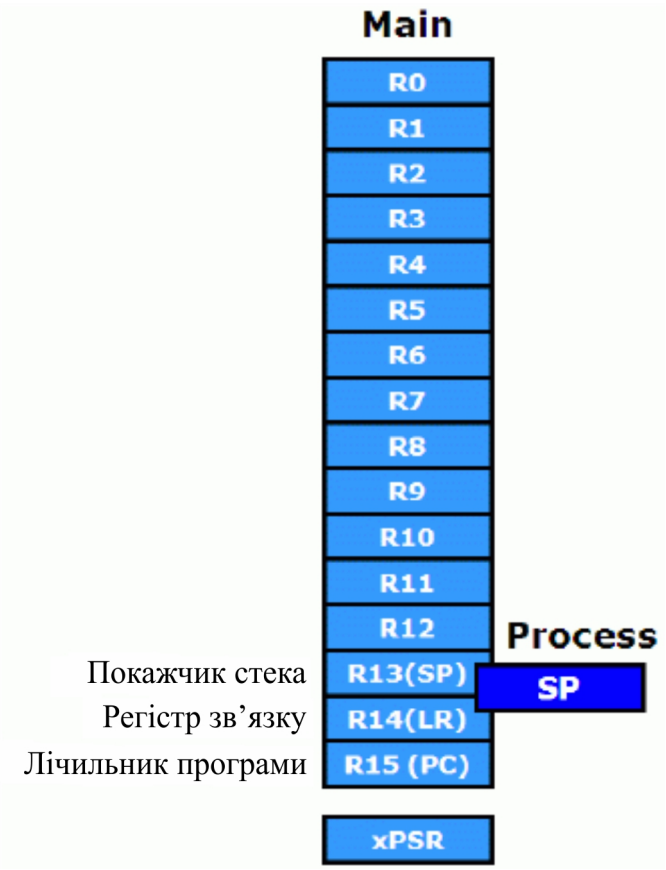
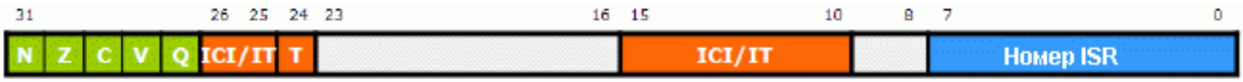


Рис. 3.7. Схема роботи ЦПП Cortex



а



ISR - процедура оброблення переривання
ICI - інструкція, яка відновлюється перериванням

б

Рис. 3.8. Файл регістрів ЦПП Cortex (*а*);
регістр статусу програми ЦПП Cortex (*б*)

Регістр R13 виступає в ролі покажчика стека. Даний регістр є банковим, що робить можливою роботу ЦПП Cortex в двох режимах, в кожному з яких використовується свій власний простір стека. Дана можливість зазвичай використовується операційними системами реального часу (ОСРЧ), які можуть виконувати свій "системний" код в захищеному режимі. У двох стеків ЦПП Cortex є власні назви: основний стек і стек процесу.

Регістр R14 – це регістр зв'язку. Він використовується для зберігання адреси повернення з підпрограми. Завдяки йому, ЦПП Cortex швидко переходить до підпрограми і виходить з неї. Якщо ж в програмі використовується кілька рівнів вкладень підпрограм, то компілятор буде автоматично зберігати вміст регістру R14 у стек.

Останній регістр R15 – лічильник програми. Оскільки він є частиною центрального файлу регістрів, процедура його читання і обробки така сама, як і для інших регістрів.

Крім файлу регістрів, є окремий регістр (xPSR), який називається регістром статусу програми (рис. 3.8, *а*, *б*). Він не входить до основного файлу регістрів, а доступ до нього здійснюють за допомогою двох спеціальних інструкцій. Регістр статусу програми містить поля статусу, від яких залежить виконання інструкцій. Даний регістр розділений на три частини: регістр статусу прикладної програми, регістр виконання програми і регістр переривань.

Біти регістра xPSR розділені на три групи, до кожної з яких можливий доступ за їх назвою (рис. 3.8, *б*). Верхні п'ять біт (прапори коду умови) іменуються регістром статусу прикладної програми. Перші чотири прапори коду умови N, Z, C, V (індикація негативного (N) або нульового (Z) результату, перенесення (C) і переповнення (V)) встановлюються і скидаються автоматично за результатами виконання інструкції обробки даних.

П'ятий біт Q використовується при виконанні математичних інструкцій з насиченням алгоритмів цифрової обробки сигналів (ЦОС) для індикації досягнення змінною свого максимального або мінімального значення. Так само як і 32-бітові інструкції ARM, деякі інструкції Thumb-2 виконуються тільки за умови збігу коду умови інструкції та стану прапорів регістру статусу прикладної програми. Якщо коди умови інструкції не збігаються, то інструкція проходить по конвеєру як NOP (Немає операції). Цим гарантується рівномірність проходження інструкцій по конвеєру і мінімізується число перезавантажень конвеєра. У Cortex для цього використано регістр статусу виконання програми, який пов'язаний з бітами 26 ... 8 регістру xPSR.

3.3.2. Режими роботи ЦПП Cortex

Процесор Cortex-M3 може використовуватися в звичайному режимі («flat»), а також підтримує операційні системи реального часу. ЦПП може працювати в двох режимах (рис. 3.9): режим Thread (або потоковий режим) і режим Handler (або режим обробника). Режим Thread використовують для виконання коду програми. Режим Handler використовують для обробки виняткових ситуацій (переривань тощо). Також цей режим використовують операційні системи реального часу (ОСРЧ). Після закінчення обробки виняткових ситуацій процесор повертається в режим Thread. Основний стек (R13) доступний в обох режимах. В режимі Handler може бути використаний також стек процесу.

Код програми ЦПП Cortex може бути в привілейованим або непривілейованим. Привілейований код має доступ до всієї системи команд і всіх ресурсів. Непривілейованому коду недоступні деякі команди (наприклад команди доступу до xPSR та і його бітових груп), заборонений доступ до більшості системних регістрів керування Cortex, а також обмежено доступ до пам'яті та периферії.

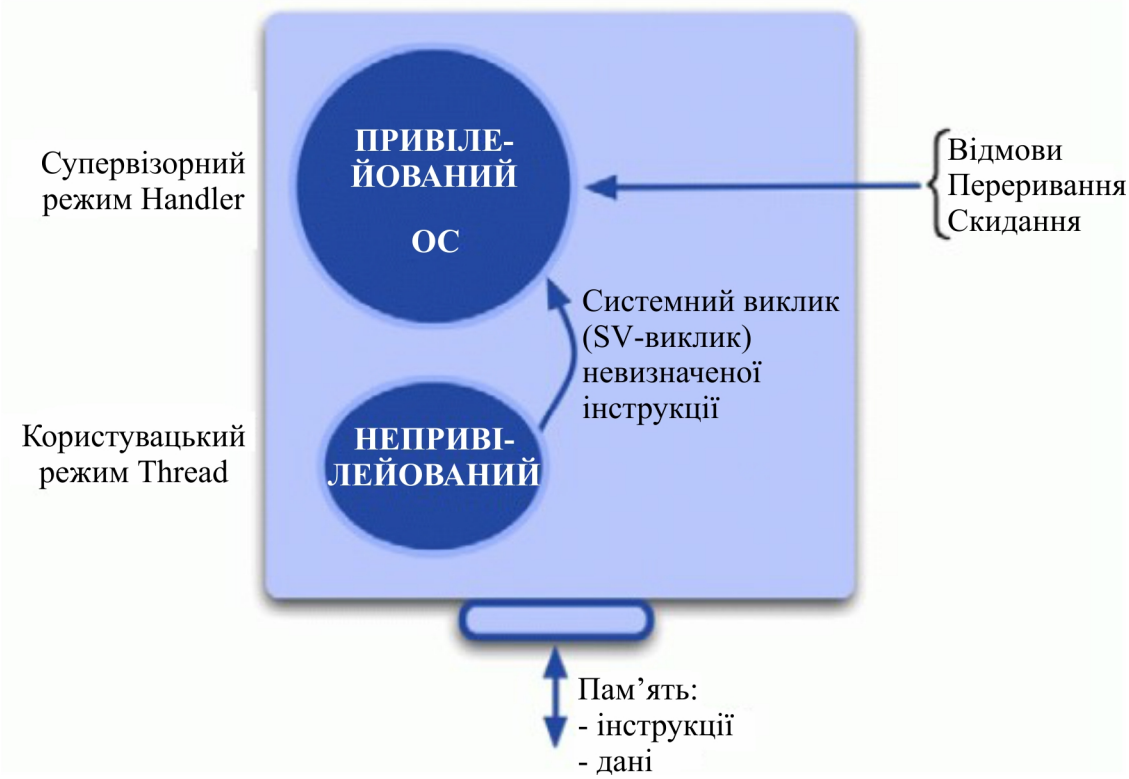


Рис. 3.9. Режими роботи ЦПП Cortex

У режимі Thread привілейованість коду визначає стан регістру CONTROL. У режимі Handler програмний код завжди привілейований.

У режимі Thread тільки привілейований код може програмно змінювати значення записане в регістр CONTROL і таким чином змінювати рівень привілейованості програмного коду. Непривілейований програмний код може лише використовувати інструкцію виклику супервізора, для передачі керування привілейованому коду.

Зазвичай, щоб почати виконання інструкцій, достатньо вказати процесору вектор скидання і стартову адресу стеку, після чого можна виконувати С-код програми. Однак, якщо використовується ОСРЧ або виконується розробка критичної до захищеності програми, процесор може бути використаний у більш складній конфігурації, за якої весь код програми поділяється на системний та прикладний, і тому помилки у прикладному коді не викликають збоїв ОСРЧ.

3.3.3. Карта пам’яті

Процесор Cortex-M3 є стандартизованим мікроконтролерним ядром і, тому, його карта пам'яті чітко розписана (рис. 3.10). Незважаючи на використання декількох внутрішніх шин, адресний простір є лінійним і має розмір 4 ГБ.

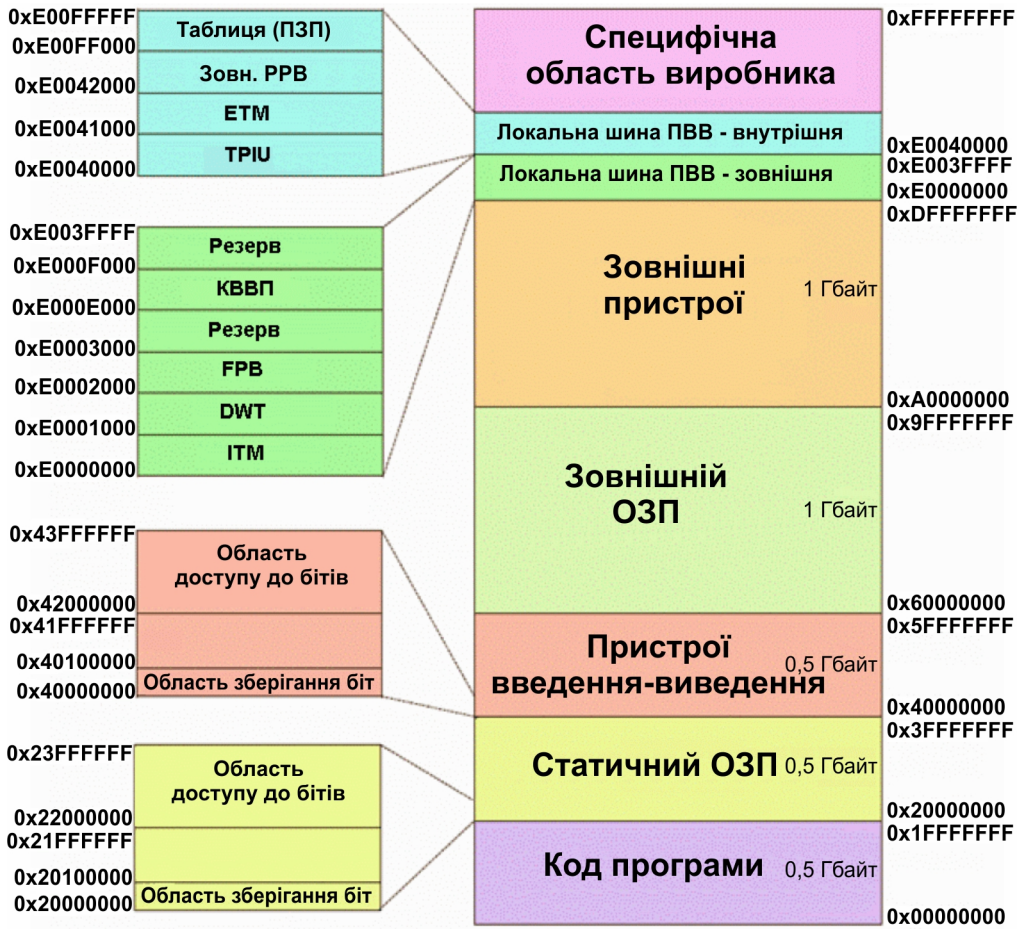


Рис. 3.10. Розподіл пам’яті процесора Cortex-M3

Перший 1 ГБ пам'яті розділений рівномірно між областю коду програми і областю статичного ОЗП. Простір коду програми оптимізовано для роботи з шиною команд (I-CODE). Аналогічно, простір статичного ОЗП оптимізовано для роботи з шиною даних (D-CODE). Шина I-CODE використовується для отримання інструкцій, а шина D-CODE – для доступу до даних розміщених в області коду.

Наступні 0,5 ГБ пам'яті – область вбудованих ПВВ. В цій області знаходяться всі надані користувачеві виробником мікроконтролера ПВВ.

Перший 1 МБ в областях статичного ОЗП і ПВВ може бути адресований побітно. Для цього використовується метод bit banding. Таким чином, всі дані, які зберігаються в цих областях, можуть оброблятися як пословно, так і побітно. Наступний 1 ГБ адресного простору виділений для зовнішнього статичного ОЗП. За ним слідує 1 ГБ адресного простору виділений для зовнішніх ПВВ. Останні 0,5 ГБ зарезервовані для системних ресурсів процесора Cortex і майбутніх розширень процесора Cortex. Всі регістри процесора Cortex розташовані за фіксованими адресами в усіх Cortex-мікроконтролерах. Завдяки цьому, полегшується перенесення програм між різними МК STM32 і Cortex-мікроконтролерами інших виробників.

3.3.4. Сегментація пам’яті

Системи команд ARM7 і ARM9 здатні оперувати знаковими і беззнаковими змінними у форматі байта (8-біт), півслова (16-біт) і слова (32-біта). Це дозволяє ЦПП вільно підтримувати цілочислові змінні без потреби в додаткових бібліотеках, як це зазвичай потрібно для 8- і 16-бітових мікроконтролерів.

ЦПП Cortex має режими адресації для слів, півслів і байт, але він підтримує і бітову сегментацію, що дає змогу упаковувати програмні прапори в змінні формату півслова і слова і не використовувати окремий байт для кожного прапора.

Зазвичай для маніпуляції з бітами необхідно зчитати байт, в якому знаходиться потрібний біт, встановити потрібне значення даного біту за допомогою бітових операцій, а потім записати результат у пам’ять. Впровадження технології бітової сегментації в областях пам'яті SRAM і периферійних пристроїв дає змогу здійснювати бітову маніпуляцію без додаткових команд зчитування/записування. Область з побітовою адресацією

карти пам'яті Cortex складається з області бітових сегментів (розміром до 1 МБ реальної пам'яті і регістрів периферійних пристроїв), а також з області псевдоімен бітових сегментів, яка займає до 32 МБ карти пам'яті (рис. 3.11). При бітовій сегментації кожному біту в області бітових сегментів відповідає адреса слова з області псевдоімен. Таким чином, встановлюючи або скидаючи значення за адресами слів псевдоімен, можна встановлювати і скидати біти в реальній пам'яті.

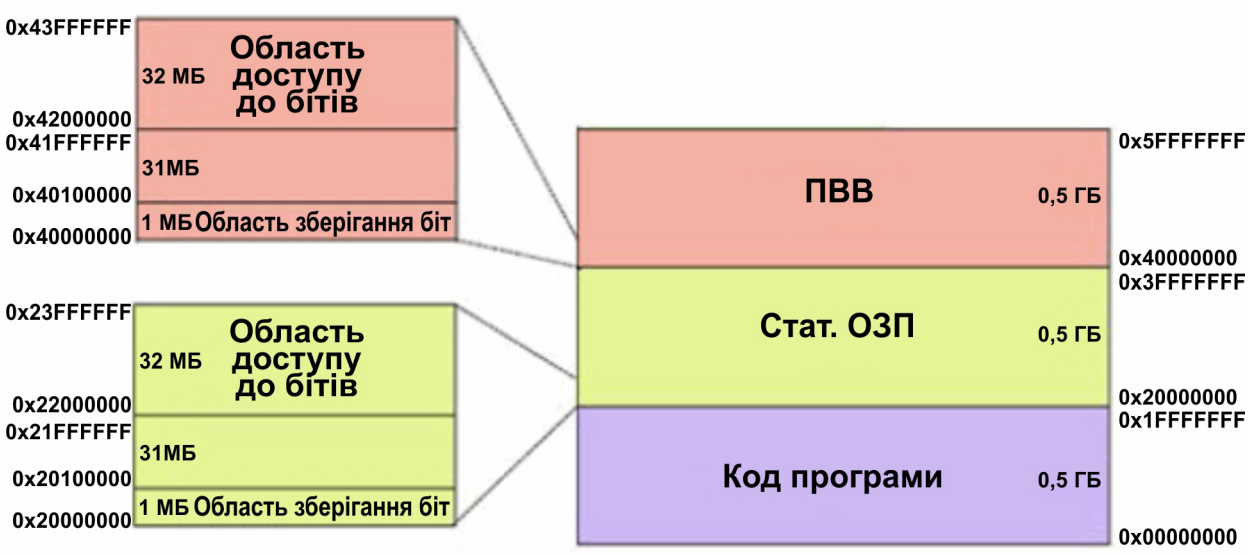


Рис. 3.11. Області бітової сегментації пам’яті процесора Cortex-M3

На практиці, для доступу до конкретного біта необхідно обчислити адресу псевдоімені бітового сегменту для заданої комірки пам'яті з області SRAM або периферійних пристроїв. Формула для обчислення адреси псевдоімені приведена нижче:

адреса в області псевдоімен = початкова адреса області + зміщення слова;

зміщення слова = зміщення (байт) від початку бітових сегментів *0x20 + номер біта * 4.

Для прикладу розглянемо, як здійснюється запис у вихідний регістр даних портів введення/виведення загального призначення (GPIO) для установлення і скидання окремих ліній введення/виведення. Фізична адреса вихідного регістра порту B – 0x40010C0C. В цьому прикладі будемо

встановлювати і скидати восьмий біт в слові, використовуючи вищенаведену формулу.

Адреса слова = 0x40010C0C;
Початок бітових сегментів периферійних пристроїв = 0x40000000;
Початок псевдоімен бітових сегментів периферійних пристроїв 0x42000000;
Байтове зміщення від початку бітових сегментів = 0x40010C0C–0x40000000 = 0x10C0C;
Зсув слова = (0x10C0C *0x20) + (8*4) = 0x2181A0;
Адреса псевдоімені = 0x42000000 + 0x2181A0 = 0x422181A0.

Тепер можна створити покажчик на цю адресу, використовуючи наступний рядок мови програмування С:

#define PortBbit8 (* ((volatile unsigned long *) 0x422181A0))

Цей покажчик потім може використовуватися для установлення і скидання біту порту введення / виведення:

PortBbit8 = 1; // встановити високий рівень сигналу на 8-ій лінії порту B
PortBbit8 = 0; // встановити низький рівень сигналу на 8-ій лінії порту B

Кожна з операцій (установлення або скидання) на STM32, що працює з частотою 72 МГц, займе 80 нс часу. При традиційному підході кожна з операцій (установлення або скидання) займе 180 нс часу при тій же тактовій частоті.

3.3.5. Шини ЦПП Cortex

Процесор Cortex-M3 побудований за Гарвардською архітектурою пам'яті з роздільними шинами для коду та даних. Ці шини носять назви I-code і D-code. Обидві шини мають доступ до програмного коду і даних в

діапазоні адрес 0x00000000 – 0x1FFFFFFF. Додаткова системна шина використовується для доступу до системної керуючої області Cortex в діапазоні адрес 0x20000000 – 0xDFFFFFFF і 0xE0100000 – 0xFFFFFFFF. Вбудована система налагодження Cortex має додаткову структуру шин, яка має назву Private Peripheral Bus.

Системна шина і шина даних підключаються до зовнішнього мікроконтролеру через сукупність високошвидкісних шин, організованих у вигляді матриці. Така організація шин дає змогу реалізовувати паралельні канали зв'язку між шинами Cortex і зовнішніми пристроями (наприклад, між DMA і SRAM). Якщо два пристрої (наприклад, ЦПП Cortex і канал DMA) намагаються отримати доступ до одного і того ж периферійного пристрою, вбудований арбітр вирішить конфлікт і надасть доступ пристрою з вищим пріоритетом.

3.3.6. Системний таймер

До складу ядра Cortex входить 24-бітний лічильник з функціями автоматичного перезавантаження і генерації переривання. Він називається таймером SysTick і призначений для використання в якості стандартного таймера в усіх Cortex-мікроконтролерах. Таймер SysTick може використовуватися для формування системних інтервалів часу в ОСРЧ або для генерації періодичних переривань для обробки запланованих завдань. За допомогою регістра керування і статусу таймера SysTick, який розташований в області системних ресурсів процесора Cortex-M3, користувач може вибрати джерело синхронізації таймера. Якщо встановити біт CLKSOURCE, то таймер SysTick буде працювати на тактовій частоті ЦПП. Якщо ж його скинути, то таймер буде працювати на частоті, рівній 1/8 тактової частоти ЦПП.

Таймер SysTick призначений для формування шкали часу в операційних системах реального часу.

Таймер SysTick пов'язаний з трьома регістрами. Для встановлення періодичності відліку необхідно ініціалізувати регістр поточного значення і регістр значення перезавантаження. У регістрі керування і статусу є біт ENABLE, який дозволяє активізувати роботу таймера, і біт TICKINT, який керує активністю лінії переривання таймера. Далі ми будемо розглядати структуру переривань Cortex і використання таймера SysTick для генерації першої виняткової ситуації в мікроконтролері STM32.

3.4. Оброблення переривань

Контролер вкладених векторизованих переривань (КВВП, NVIC) є стандартним блоком ядра Cortex. Це означає, що у будь-якого Cortex-мікроконтролера буде одна і та ж структура переривань, незалежно від його виробника, що забезпечує переносимість коду.

КВВП підтримує вкладені переривання і, зокрема, в МК STM32 використовується 16 рівнів пріоритетів.

КВВП містить одне немасковане переривання і до 240 ліній зовнішніх переривань, які можуть бути з'єднані з периферійними пристроями. Також є додаткові 15 джерел ядра Cortex, які обслуговують внутрішні виняткові ситуації самого ядра Cortex.

Структура переривань КВВП розроблена з урахуванням програмування повністю на С і виключає необхідність використання будь-яких макросів на асемблері або спеціальних, несумісних з ANSI, директив.

При виникненні переривання від периферійного пристрою, КВВП починає переведення ЦПП Cortex у режим обробки переривання. Як тільки ЦПП Cortex входить до режиму обробки переривання, воно автоматично зберігає значення набору регістрів в стек (без додаткових команд в коді програми). У той час, як в стеці зберігаються дані, визначається початкова адреса процедури обробки переривань. З моменту виникнення переривання

до початку виконання першої команди процедури обробки переривання проходить 12 циклів внутрішнього тактування.

КВВП підтримує кілька методів, які дозволяють обробляти кілька запитів переривань з мінімальними затримками між перериваннями і з гарантуванням дотримання пріоритетності їх обробки.

Якщо підчас обробки переривання виникає переривання з більш високим пріоритетом, обробка активного переривання припиняється, протягом наступних 12 циклів виконується збереження в стек нового набору даних і запускається обробка переривання з більш високим пріоритетом. По завершенні його обробки, із стеку автоматично відновлюються збережені дані та продовжується обробка переривання з нижчим пріоритетом.

Якщо під час обробки переривання з великим пріоритетом виникає переривання з меншим пріоритетом, КВВП ставить його в чергу на обробку відповідно до його пріоритету.

Щоб включити в роботу КВВП необхідно виконати три дії. Спочатку конфігурувати таблицю векторів використовуваних переривань. Потім налаштувати регістри КВВП з метою активізації та установки рівнів пріоритету переривань КВВП. І, нарешті, налаштувати ПВВ і дозволити підтримку ними переривань.

Таблиця векторів Cortex починається з адреси 0x00000004 (табл. 3.3), тому що перші чотири байти використовуються для зберігання початкової адреси покажчика стеку.

Кожен з векторів переривань займає 4 байти і вказує на початкову адресу кожної конкретної процедури обробки переривання. Перші 15 векторів – адреси обробки виняткових ситуацій, що виникають в ядрі Cortex. До них відносяться вектор скидання, немасковане переривання, управління аваріями і помилками, виняткові ситуації системи налагодження і переривання таймеру SysTick. Також переривання можуть генерувати деякі команди ядра. Переривання від призначених для користувача периферійних пристроїв починаються з 16-го вектора.

Таблиця 3.3.

Таблиця векторів виняткових ситуацій

Номер	Тип виняткової ситуації	Пріоритет	Тип пріоритету	Опис
1	Reset	–3 (вищий)	фіксований	перезавантаження
2	NMI	–2	фіксований	немасковане переривання
3	Hard Fault	–1	фіксований	обробник аварійних станів за замовчуванням
4	MemManageFault	0	програмований	помилка в роботі з пам’яттю
5	BusFault	1	програмований	помилка роботи з шиною
6	UsageFault	2	програмований	помилка в програмі
7-10	зарезервовано			зарезервовано
11	SVCall	3	програмований	виклик системних служб
12	DebugMonitor	4		точки зупинки в режимі налагодження
13	зарезервовано			зарезервовано
14	PendSV	5	програмований	запит до системного пристрою
15	SYSTICK	6	програмований	спрацювання системного таймеру
16	переривання 0	7	програмований	зовнішнє переривання 0
	…	…	…	
256	переривання 240	7	програмований	зовнішнє переривання 240

У програмі таблиця векторів зазвичай наводиться в окремому файлі та містить адреси процедур обробки переривань:

```
                AREA    RESET, DATA, READONLY
                EXPORT          __Vectors
__Vectors       DCD     __initial_sp            ; Вершина стека
                DCD     Reset_Handler           ; Обробник Скидання
                DCD     NMI_Handler             ; Обробник NMI
                DCD     HardFault_Handler       ; Обробник апаратної помилки
                DCD     MemManage_Handler       ; Обробник помилки БПП
                DCD     BusFault_Handler        ; Обробник помилки шини
                DCD     UsageFault_Handler      ; Обробник помилки використання
                DCD     0                       ; Зарезервовано
                DCD     0                       ; Зарезервовано
                DCD     0                       ; Зарезервовано
                DCD     0                       ; Зарезервовано
                DCD     SVC_Handler             ; Обробник SVCall
                DCD     DebugMon_Handler        ; Обробник монітора налагодження
                DCD     0                       ; Зарезервовано
                DCD     PendSV_Handler          ; Обробник PendSV
                DCD     SysTick_Handler         ; Обробник SysTick
```

Наприклад, якщо використовується переривання таймера SysTick, то оголошення на С процедури обробки переривання виконується наступним чином:

```
void SysTick_Handler (void)
{
....
}
```

Після налаштування таблиці векторів і оголошення процедури обробки переривань, можна налаштувати КВВП на обробку переривання таймера SysTick. Зазвичай, для цього виконують дві операції: задають пріоритет переривання, а потім дозволяють джерело переривання.

Регістри КВВП знаходяться в області системних ресурсів Cortex-M3 і доступ до них можливий при роботі ЦПП тільки в привілейованому режимі. Налаштування внутрішніх виняткових ситуацій процесора Cortex виконується з допомогою регістрів системного управління та системних пріоритетів, а призначених для користувача ПВВ – за допомогою відповідних

регістрів IRQ. Переривання SysTick є внутрішньої винятковою ситуацією процесора Cortex і, тому, керування ним здійснюється через системні регістри.

Деякі внутрішні виняткові ситуації постійно дозволені. До них відносяться переривання по скиданню, немасковане переривання, а також переривання таймера SysTick, тому, ніяких дій з КВВП для дозволу цього переривання робити не потрібно. Для налаштування переривання SysTick нам необхідно лише активізувати сам таймер і його переривання за допомогою відповідного регістра керування:

```
SysTickCurrent = 0x9000; // початкове значення системного таймеру
SysTickReload = 0x9000; // значення перезавантаження
SysTickControl = 0x07; // запуск таймеру і дозвіл переривання
```

Пріоритет кожної з внутрішніх виняткових ситуацій Cortex може бути встановлений в системному регістрі переривань. Пріоритети виняткових ситуацій Reset, NMI і hard fault фіксовані для того, щоб ядро безпомилково знаходило потрібний обробник виняткових ситуацій. Кожна з решти виняткових ситуацій має восьмибітне поле в одному з трьох системних регістрів переривань. У STM32 використовують 16 рівнів пріоритетів переривань, так що активні тільки чотири старших біта восьмибітного поля.

Кожен з призначених для користувача периферійних пристроїв може бути керованим через блок регістрів запитів переривань IRQ. Кожен призначений для користувача периферійний пристрій має біт дозволу переривання Interrupt Enable bit (рис. 3.12). Ці біти розташовані в двох 32-бітних регістрах IRQ Set Enable. Відповідно існують регістри IRQ Clear Enable, які використовують для заборони переривань. КВВП містить також регістри, які відображають стан активних переривань, за якими користувач може визначити поточний стан і джерело переривання.

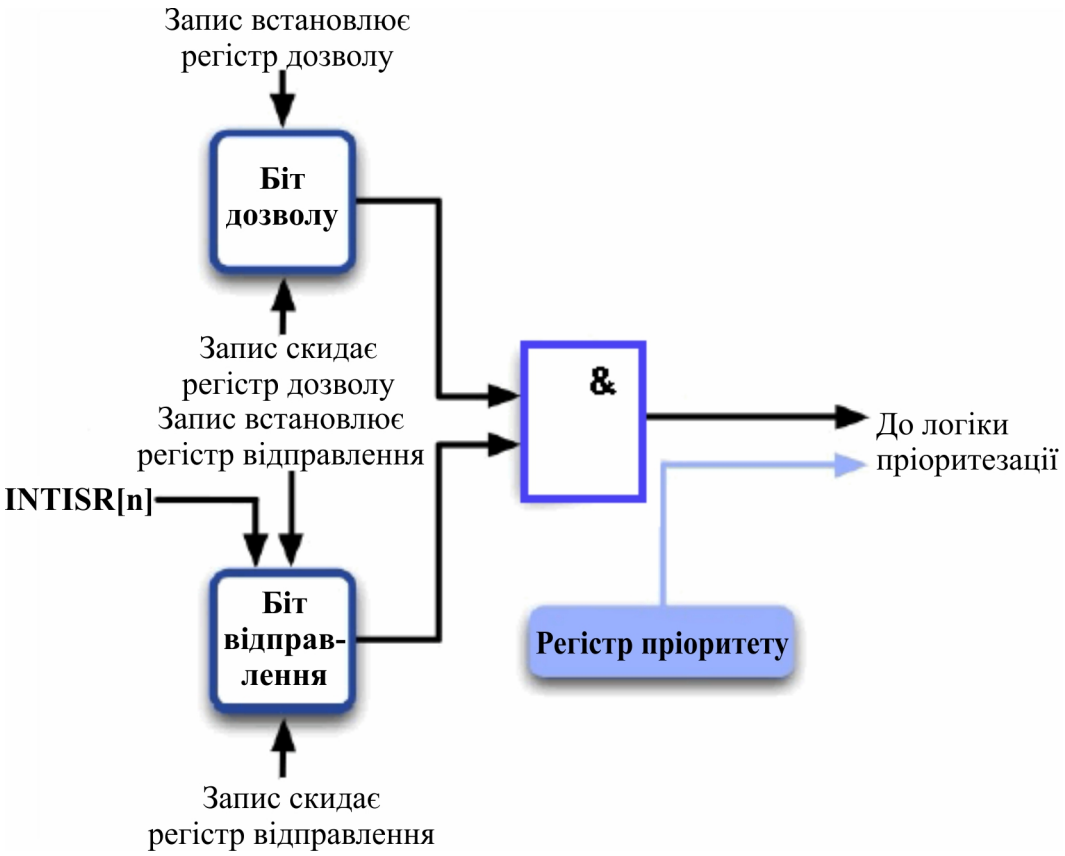


Рис. 3.12. Керування перериваннями процесора Cortex-M3

За замовчуванням поле пріоритету визначає 16 рівнів пріоритету, причому рівень 0 – найвищий пріоритет, а 15 – найнижчий.

Рівні пріоритетів можна також розділити на групи і підгрупи. Це не забезпечить додаткових рівнів пріоритетів, але полегшить управління, якщо в системі використовується велика кількість переривань. Для цього потрібно запрограмувати поле PRIGROUP в регістрі Application Interrupt and Reset Control Register (рис. 3.13).

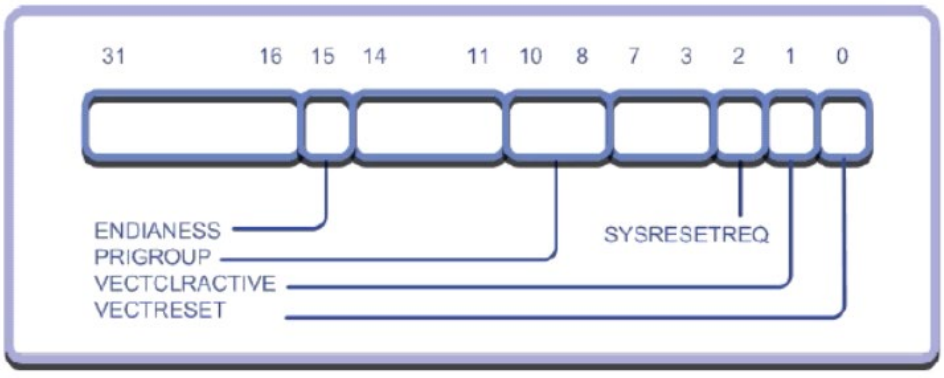


Рис. 3.13. Регістр керування перериваннями та скиданням

Трибітне поле PRIGROUP керує розподілом 4-бітних полів пріоритету на групи і підгрупи (табл. 3.4). Залежно від того, яка група встановлена, буде доступна різна кількість пріоритетів і субпріорітетів.

Таблиця 3.4.

Розподілом полів пріоритету на групи і підгрупи

PRIGROUP (3-біта)	Група пріоритету		Підгрупа пріоритету	
	Кількість біт	Кількість рівнів	Кількість біт	Кількість рівнів
011	4	16	0	0
100	3	8	1	2
101	2	4	2	4
110	1	2	3	8
111	0	0	4	16

При програмуванні МК сімейства STM32 мовою С (компілятор gcc) для встановлення групи пріоритетів використовують функцію NVIC_PriorityGroupConfig. За допомогою цієї функції можна встановити одну з 5 груп. Можливі комбінації пріоритетів і субпріоритетів (NVIC_IRQChannelPreemptionPriority і NVIC_IRQChannelSubPriority) для можливих значень PriorityGroup вказані в таблиці 3.5.

Таблиця 3.5.

Комбінації пріоритетів і субпріоритетів при використанні функції NVIC_PriorityGroupConfig

NVIC_PriorityGroup	NVIC_IRQChannel PreemptionPriority	NVIC_IRQChannel SubPriority
NVIC_PriorityGroup_0	0	0-15
NVIC_PriorityGroup_1	0-1	0-7
NVIC_PriorityGroup_2	0-3	0-3
NVIC_PriorityGroup_3	0-7	0-1
NVIC_PriorityGroup_4	0-15	0

Як вже було сказано, залежно від того, яка група встановлена, буде доступна різна кількість пріоритетів і субпріорітетів. Пріоритети дають змогу реалізувати механізм вкладених переривань, тоді як субпріорітети визначають послідовність обробки переривань з однаковим пріоритетом при одночасному спрацьовуванні.

Таким чином, якщо встановлено NVIC_PriorityGroup_0, то згідно з таблицею можна встановити перериванням тільки один пріоритет NVIC_IRQChannelPreemptionPriority = 0 і призначити різним перериванням субпріорітети NVIC_IRQChannelSubPriority від 0 до 15. Це означає, що обробка переривання не буде перериватися іншим перериванням, оскільки у всіх однаковий пріоритет. А субпріорітет NVIC_IRQChannelSubPriority визначає тільки послідовність обробки при одночасному спрацьовуванні переривань.

Якщо ж встановлено NVIC_PriorityGroup_1, то можна одним перериванням (більш пріоритетним) встановити NVIC_IRQChannelPreemptionPriority = 0, а менш пріоритетним NVIC_IRQChannelPreemptionPriority = 1. До того ж усім перериванням можна встановити субпріорітет від 0 до 7, який визначає порядок обробки при одночасному надходженні переривань з однаковим пріоритетом. В цьому випадку переривання з пріоритетом 0 (високим) зможуть переривати обробку переривання з пріоритетом 1. А якщо одночасно виникнуть переривання з однаковим пріоритетом, то першим буде оброблено те, у якого вищий субпріорітет. Наведемо кілька прикладів конфігурування КВВП. В прикладах будемо використовувати стандартну бібліотеку функцій SPL (standard peripheral library). Вибір бібліотеки здійснюється на етапі створення проекту. Можна використовувати також бібліотеку HAL (hardware abstraction layer) та CMSIS (Cortex® Microcontroller Software Interface Standard). На даний час всі три бібліотеки підтримуються виробниками мікроконтролерів на базі ядра Cortex®, а також виробниками програмного забезпечення. Бібліотеки SPL і HAL, це бібліотеки для високорівневого програмування на рівні прикладних

програм (коли доступ до ресурсів МК здійснюється опосередковано через виклик необхідної функції), включаючи використання операційних систем реального часу. Бібліотека CMSIS призначена для прямого доступу до ресурсів МК. Бібліотеки SPL і HAL можна використовувати спільно. Розробник рекомендує для високорівневого програмування використовувати бібліотеку HAL, оскільки вона більш нова і в майбутньому буде здійснено повний перехід на HAL (SPL не буде підтримуватись). Однак SPL більш інтуїтивно зрозуміла для користувача-початківця і дає змогу краще зрозуміти роботу МК. А перехід на HAL можна здійснити самостійно розглянувши документацію на дану бібліотеку.

Приклад 3.1. Налаштування переривання USART1_IRQn для випадку коли інші призначені для користувача переривання не використовують:

```
NVIC_InitTypeDef NVIC_InitStructure;
/* Enable the USARTx Interrupt */
NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

У наведеному прикладі використано лише одне переривання (NVIC_IRQChannelPreemptionPriority = 0;) і тому не має значення, що саме вказано в подальших командах (NVIC_IRQChannelSubPriority = 0;). Однак у ситуації, коли відбувається запит на виконання дуже важливого переривання, треба призупинити обробку менш важливого переривання, і терміново обслужити запит, який прийшов.

Розглянемо приклад налаштування двох переривань з однаковими пріоритетами і різними субпріорітетами.

Приклад 3.2:

```
NVIC_InitTypeDef NVIC_InitStructure;
/* Enable the USARTx Interrupt */
NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);

/* Enable the EXTI0_IRQn Interrupt */
NVIC_InitStruct.NVIC_IRQChannel = EXTI0_IRQn;
NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStruct.NVIC_IRQChannelSubPriority = 1;
NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStruct);
```

У цьому прикладі встановлений субпріоритет (NVIC_IRQChannelSubPriority = 1;) для EXTI0_IRQn. А це означає, що коли EXTI0_IRQn і USART1_IRQn відбудуться одночасно, першим буде оброблено USART1_IRQn. Але жодне з цих переривань не зможе перервати обробку іншого, тому що у них однаковий пріоритет (NVIC_IRQChannelPreemptionPriority = 0;)

Тепер розглянемо приклад, коли EXTI0_IRQn і EXTI1_IRQn мають більший пріоритет і мають право перервати обробку USART1_IRQn.

Приклад 3.3:

```
/* Set NVIC Priority Group */
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
```

```
 NVIC_InitTypeDef NVIC_InitStructure;
/* Enable the USARTx Interrupt */
NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);

/* Enable the EXTI0_IRQn Interrupt */
NVIC_InitStruct.NVIC_IRQChannel = EXTI0_IRQn;
NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStruct);

/* Enable the EXTI0_IRQn Interrupt */
NVIC_InitStruct.NVIC_IRQChannel = EXTI1_IRQn;
NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStruct.NVIC_IRQChannelSubPriority = 1;
NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStruct);
```

В даному прикладі встановлено NVIC_PriorityGroup_1. Згідно таблиці 3.5 в цьому випадку можна встановити два пріоритети. В даному разі різні субпріорітети логічно встановлювати тільки для переривань з однаковим пріоритетом. Що і зроблено для EXTI0_IRQn і EXTI1_IRQn. Тепер EXTI0_IRQn або EXTI1_IRQn можуть перервати обробку USART1_IRQn. Але не можуть перервати обробку один одного тому що у них однаковий пріоритет. А якщо EXTI0_IRQn і EXTI1_IRQn виникнуть одночасно, тоді першим буде оброблений EXTI0_IRQn, бо у нього вище субпріорітет.

3.5. Тактування контролерів STM32

Тактування – це основа роботи всіх складових частин мікроконтролера. Тактова частота задає продуктивність як мікроконтролера в цілому, так і окремих його частин. Чим вища частота, тим швидше працює мікроконтролер, але і більше споживає енергії. І навпаки, чим менша частота тактування, тим менше споживання енергії і менше швидкість роботи контролера. Те ж стосується і периферійних пристроїв. Тактування слід налаштовувати залежно від поточних завдань. Для коректної роботи периферії необхідно правильно налаштувати і сам мікроконтролер, і певні його модулі для роботи на конкретній частоті.

STM32 має 2 джерела тактування, а саме: HSI – внутрішній RC-генератор частотою 8 МГц (для STM32F103) та HSE – зовнішній кварцовий резонатор від 4 до 16 МГц.

HSI не потребує ніяких додаткових елементів, але не може забезпечити високої стабільності частоти. HSE забезпечує високу стабільність частоти, але потребує підключення кварцового резонатора і кількох конденсаторів.

Мікроконтролер містить модуль PLL (рис. 3.14), який може множити частоту. Тобто, обравши джерело тактування (HSI або HSE), а потім, налаштувавши модуль PLL, на виході отримують потрібну системну частоту (SYSCLK).

Мікроконтролер завжди стартує від внутрішнього RC-генератора (8 МГц), а всі зміни тактування виконуються програмно. STM32 може повернутися до тактування з HSI у разі, якщо обраний раніше HSE з якихось причин не працює. Також можна програмно змінювати режими тактування по ходу виконання програми. Наприклад, для переходу в режим енергозбереження тощо. Розглянемо приклад налаштування тактування на 72 МГц.

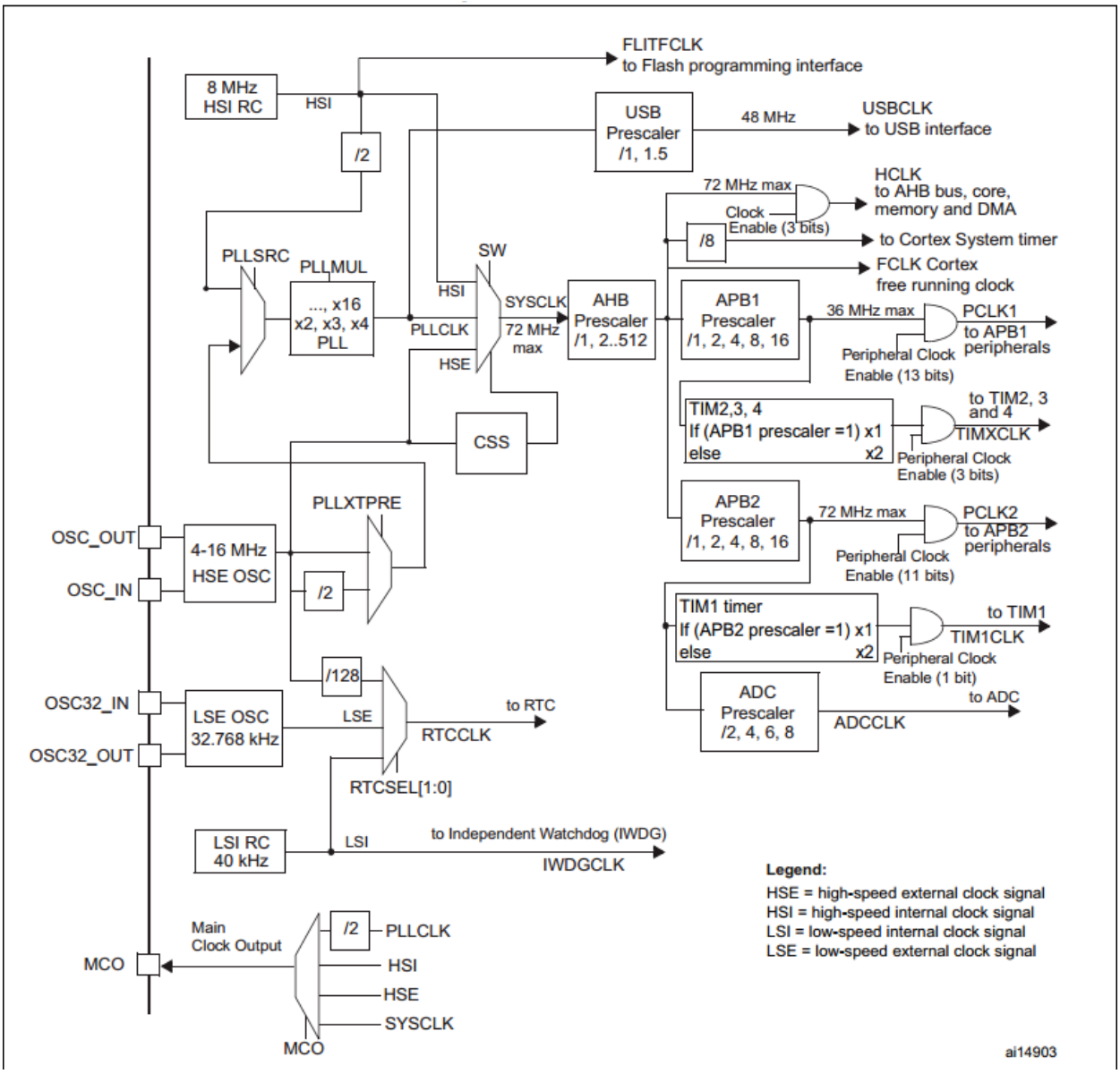


Рис. 3.14. Схема тактування мікроконтролерів STM32F

Приклад 3.4:

```
#include "stm32f10x.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_flash.h"
void SetSysClockTo72 (void)
{
  ErrorStatus HSEStartUpStatus;
```

```
/* SYSCLK, HCLK, PCLK2 and PCLK1 configuration ---------- */
/*Системний RESET RCC (не обов'язково, але корисно для  налагодження) */
   RCC_DeInit ();
/* Включаємо HSE (зовнішній кварц) */
   RCC_HSEConfig (RCC_HSE_ON);
/* Чекаємо поки HSE буде готовий */
   HSEStartUpStatus = RCC_WaitForHSEStartUp();
/* Якщо з HSE все в порядку */
   if (HSEStartUpStatus == SUCCESS)
   {
/* Наступні дві команди стосуються виключно роботи з FLASH.
Якщо ви не збираєтеся використовувати в своїй програмі функцій
роботи з Flash,  FLASH_PrefetchBufferCmd() та FLASH_SetLatency()
можна закоментувати */
/* Включаємо Prefetch Buffer */
      FLASH_PrefetchBufferCmd (FLASH_PrefetchBuffer_Enable);
/* FLASH Latency. Рекомендовано встановлювати:
FLASH_Latency_0 - 0 <SYSCLK≤ 24 MHz
FLASH_Latency_1 - 24 MHz <SYSCLK ≤ 48 MHz
FLASH_Latency_2 - 48 MHz <SYSCLK ≤ 72 MHz */

      FLASH_SetLatency (FLASH_Latency_2);
/* HCLK = SYSCLK */
/* Дивись на схемі (рис. 3.14) AHB Prescaler. Частота не ділиться
(RCC_SYSCLK_Div1) */
      RCC_HCLKConfig (RCC_SYSCLK_Div1);
/* PCLK2 = HCLK */
/* Дивись на схемі (рис. 3.14) APB2 Prescaler. Частота не ділиться
(RCC_HCLK_Div1) */
      RCC_PCLK2Config (RCC_HCLK_Div1);
```

```
/* PCLK1 = HCLK / 2 */
/* Дивись на схемі (рис. 3.14) APB1 Prescaler. Частота ділиться на 2 (RCC_HCLK_Div2) тому що на виході APB1 має бути не більше 36МГц */
    RCC_PCLK1Config (RCC_HCLK_Div2);
/* PLLCLK = 8MHz * 9 = 72 MHz */
/* Вказуємо PLL звідки брати частоту (RCC_PLLSource_HSE_Div1) і на скільки її множити (RCC_PLLMul_9) */
/* PLL може брати частоту з кварцу як є (RCC_PLLSource_HSE_Div1) або поділену на 2 (RCC_PLLSource_HSE_Div2). Дивись схему (рис. 3.14) */
    RCC_PLLConfig (RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);
/* Включаємо PLL */
    RCC_PLLCmd (ENABLE);
/* Чекаємо поки PLL буде готовий */
    while (RCC_GetFlagStatus (RCC_FLAG_PLLRDY) == RESET)
    {
    }
/* Перемикаємо системне тактування на PLL */
    RCC_SYSCLKConfig (RCC_SYSCLKSource_PLLCLK);
/* Чекаємо поки переключитися */
    while (RCC_GetSYSCLKSource ()! = 0x08)
    {  }
  }
  else
  {
/* Проблеми з HSE. Тут можна написати свій код, якщо треба щось робити, коли мікроконтролер не зміг перейти на роботу із зовнішнім кварцом */
/* Поки тут заглушка - нескінченний цикл */
```

```
        while (1)
        {
        }
    }
}
```

На схемі (рис. 3.14) також є LSI (low-speed internal clock) і LSE (low-speed external clock). Це внутрішній низькочастотний RC-генератор і зовнішній низькочастотний кварцовий резонатор. Як видно, LSI використовується для тактування таймера Watchdog (WDG) і може використовуватися для тактування вбудованого годинника реального часу (RTC). А LSE тільки для тактування годинника реального часу (RTC). До речі, тактування RTC може бути здійснено і від HSE. Але зручніше використовувати часовий кварц 32768 Гц (LSE).

Також слід відзначити, що мікроконтролер має вихід тактової частоти MCO (Main Clock Output), який є необхідним для налаштування тактизації зовнішніх пристроїв. Це функція RCC_MCOConfig. Параметри можуть бути такі:

RCC_MCO_NoClock – вмикаємо MCO
RCC_MCO_SYSCLK – подати SYSCLK на MCO
RCC_MCO_HSI – подати HSI на MCO
RCC_MCO_HSE – подати SYSCLK на MCO
RCC_MCO_PLLCLK_Div2 – подати поділену на 2 частоту PLLCLK

3.6. Порти введення-виведення мікроконтролерів STM32

У МК STM32 передбачено до 80 двонапрямлених ліній введення-виведення. Всі лінії введення-виведення розділені на 5 портів по 16 ліній введення-виведення в кожній.

Кожна цифрова лінія введення-виведення може виконувати функцію лінії введення-виведення загального призначення або альтернативну функцію. Кожен з виводів може виконувати додаткову функцію одного з 16 входів зовнішніх переривань. Порти називаються A, B, C, D, E і сумісні з напругою 5 В. Багато із зовнішніх виводів можуть виконувати альтернативну функцію лінії введення-виведення вбудованого ПВВ, наприклад, модуля I2C. Крім того, 16 вхідних ліній вбудованого блоку зовнішніх переривань можуть бути з'єднані з будь-якими з ліній портів введення-виведення.

Кожен порт підтримує можливість роздільної конфігурації ліній на введення або на виведення. Крім того, передбачені регістри як для записування цілих слів, так і для роботи з окремими бітами. Після завершення конфігурації доступ до неї можна заблокувати.

Кожен порт пов'язаний з двома 32-бітними регістрами конфігурації. Спільно вони утворюють 64-бітний конфігураційний регістр. Ці 64 біта розділені на 4-бітові поля, які дозволяють налаштувати відповідну їм лінію введення-виведення. У свою чергу, 4-бітне поле конфігурації складається з 2-бітного поля режиму і 2-бітного поля конфігурації. Поле режиму дозволяє вказати, в якому напрямку працює лінія: на введення або на виведення, а поле конфігурації дозволяє налаштувати характеристики керування:

– якщо лінія налаштована як вхід, то до неї можна підключити підтягуючий резистор (до напруги живлення або до земляної шини);

– лінія, налаштована на виведення, може бути або двотактною, або з відкритим стоком, і для неї можна вказати її максимальну швидкодію: 2, 10 або 50 МГц.

Розглянемо приклад. Пін 13 порту C (PC13) налаштуємо як вихід (будемо вважати, що він керує вмиканням-вимиканням діоду). Пін 0 порту B (PB0) налаштуємо як вхід. Будемо вважати, що до нього підключено кнопку. В нажатому положенні кнопка має замикати пін на земляну шину.

Приклад 3.5:

```
//повідомляємо компілятору, який мікроконтролер будемо використовувати
#include "stm32f10x.h"
//підключаємо функції роботи з портами для даної серії МК
#include "stm32f10x_gpio.h"
//підключаємо функції роботи з тактовим генератором для даної серії МК
#include "stm32f10x_rcc.h"

int main(void)
{
  int i;
  /* створюємо структуру конфігурації порту з ім’ям GPIO_InitStructure */
GPIO_InitTypeDef  GPIO_InitStructure;

  /* ініціалізуємо PC13 */
  // дозволяємо тактування порту C - PORTC Clock
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
  /* конфігуруємо PC13. Для цього заповнюємо поля структури конфігурації
порту і визиваємо бібліотечну функцію ініціалізації*/
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(GPIOC, &GPIO_InitStructure);

//встановлюємо “0” (низький рівень) на PC13
  GPIO_ResetBits(GPIOC, GPIO_Pin_13); // Set C13 to Low level ("0")

  /* ініціалізуємо PB0 */
  // дозволяємо тактування порту B PORTB Clock
```

```
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
  /* конфігуруємо PB0 */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
  GPIO_Init(GPIOB, &GPIO_InitStructure);
//запускаємо нескінченний цикл
  while (1)
  {
   //аналізуємо стан кнопки підключеної до PB0
        if (GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_0) != 0) {
        /*Якщо на PB0 високий рівень, то інвертуємо стан PC13, використовуючи побітовий оператор «виключне АБО» */
        GPIOC->ODR ^= GPIO_Pin_13;

        /* затримка*/
        for(i=0;i<0x100000;i++);

        /* після затримки знову інвертуємо стан PC13, використовуючи побітовий оператор «виключне АБО» */
        GPIOC->ODR ^= GPIO_Pin_13;

        /* затримка */
        for(i=0;i<0x100000;i++);
   }
   Else
   {
        /*Якщо на PB0 низький рівень, то встановлюємо на PC13 високий рівень, використовуючи функцію GPIO_SetBits */
        GPIO_SetBits(GPIOC, GPIO_Pin_13);
```

```
    }
  }
}
```

Наведена в прикладі програма опитує стан кнопки і блимає світлодіодом тільки коли кнопка відпущена, тобто коли на вході B0 високий рівень сигналу.

Структура GPIO_InitTypeDef для налаштування виводів порту має наступні параметри:

GPIO_Pin – номери пінів, які конфігуруються. Приклад для декількох пінів:

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2;

GPIO_Speed – задає швидкість для обраних пінів. Може приймати наступні значення: GPIO_Speed_10MHz, GPIO_Speed_2MHz, GPIO_Speed_50MHz

GPIO_Mode – задає режим роботи пінів. Може приймати наступні значення:

GPIO_Mode_AIN – аналоговий вхід;

GPIO_Mode_IN_FLOATING – вхід без підтяжки (Float)

GPIO_Mode_IPD – вхід з підтяжкою до землі (Pull-down)

GPIO_Mode_IPU – вхід з підтяжкою до напруги живлення (Pull-up)

GPIO_Mode_Out_OD – вихід з відкритим стоком (Open Drain)

GPIO_Mode_Out_PP – вихід з двома станами (Push-Pull)

GPIO_Mode_AF_OD – вихід з відкритим стоком для альтернативних функцій (Alternate Function). Використовується, коли виводи керуються периферією, яка може бути задіяна на цьому виводі. Наприклад USART, I2C тощо.

GPIO_Mode_AF_PP – те ж саме, що і перед цим, але з двома станами.

У наведеному прикладі ми двічі викликаємо функцію RCC_APB2PeriphClockCmd для різних портів:

RCC_APB2PeriphClockCmd (RCC_APB2Periph_GPIOC, ENABLE);

RCC_APB2PeriphClockCmd (RCC_APB2Periph_GPIOB, ENABLE);

Це можна зробити і за один виклик, використавши операцію АБО (|):

RCC_APB2PeriphClockCmd (RCC_APB2Periph_GPIOC | RCC_APB2Periph_GPIOB, ENABLE);

3.7. Зовнішні переривання мікроконтролерів STM32

Попередній приклад (приклад 3.5) достатньо простий, але не достатньо ефективний. Контролер весь час завантажений опитуванням піна PB0 і не може виконувати ніяких інших дій. Доцільніше було б використати для цього систему зовнішніх переривань. Зовнішні переривання визиваються при зміні логічного рівня сигналу на виводі мікроконтролера. Їх використовують у випадку, коли необхідно оперативно реагувати на зовнішні по відношенню до мікроконтролера події. Для цього необхідно налаштувати на потрібний режим роботи відповідний пін, налаштувати зовнішнє переривання і написати відповідний обробник для даного переривання. Будь-які лінії портів введення-виведення STM32 можуть працювати із зовнішніми перериваннями. У STM32F103 всього 20 ліній зовнішніх переривань:

EXTI0 ... EXTI15 – для роботи з пінами портів (рис. 3.15);

EXTI16 – підключений до PVD (Power Voltage Detector – детектор напруги живлення);

EXTI17 – RTC Alert event (будильник);

EXTI18 – USB Wakeup event;

EXTI19 – Ethernet Wakeup event.

У різних серіях STM32 кількість EXTI може бути різною. У деяких мікроконтролерах через EXTI заведені переривання від периферії. І не тільки типу wakeup. Наприклад, на лінії EXTI можуть бути заведені виходи аналогових компараторів.

Із рис. 3.15 видно, що переривання EXTI0 можна задіяти для роботи з нульовими лініями всіх портів. Переривання EXTI1 можна задіяти для роботи з першими лініями всіх портів. І так далі.

Якщо необхідно одночасно налаштувати переривання на лініях PA0 і PB0, при змінах на будь-якому з цих двох пінів буде викликатися один обробник – EXTI0. В такому випадку необхідна додаткова перевірка стану входів.

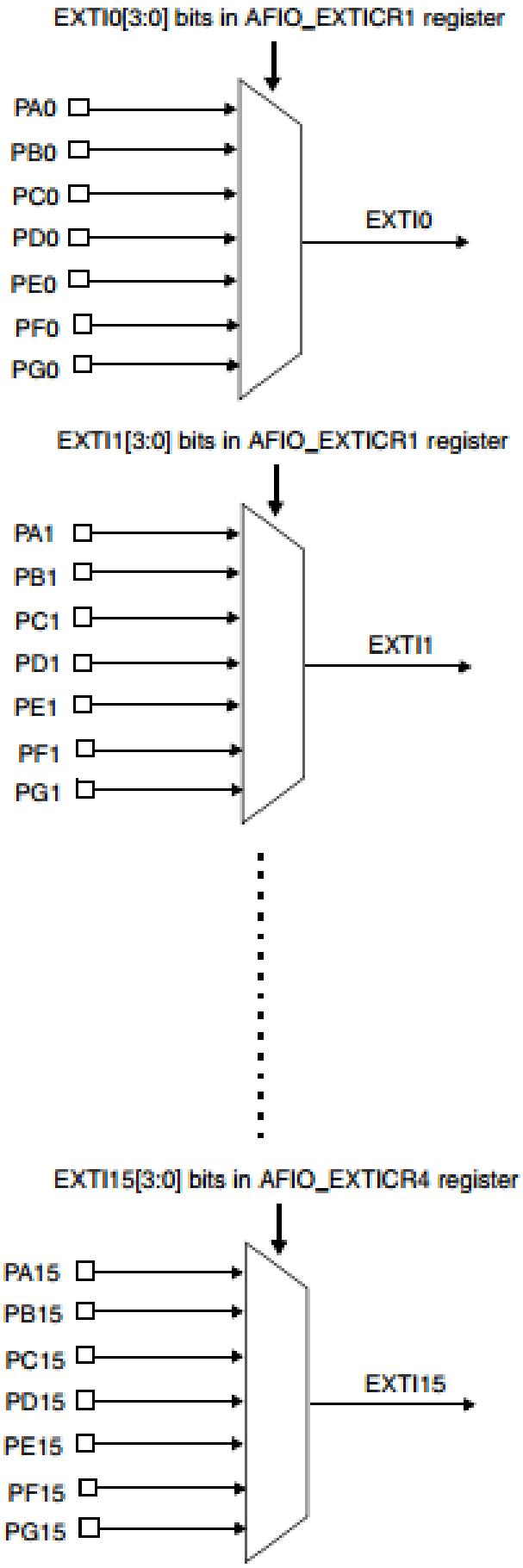


Рис. 3.15. Розподіл зовнішніх переривань EXTI0...EXTI15 STM32F103

Ініціалізація зовнішнього переривання виконується наступним чином:

– включають тактування порту GPIO і AFIO;

– налаштовують пін порту на вхід;

– налаштовують EXTI в NVIC;

– вказують порт і пін, який буде використовуватися як джерело для EXTI;

– налаштовують параметри EXTI.

Приклад 3.6:

```
//повідомляємо компілятору, який мікроконтролер будемо використовувати
#include "stm32f10x.h"
//підключаємо функції роботи з портами для даної серії МК
#include "stm32f10x_gpio.h"
//підключаємо функції роботи з тактовим генератором для даної серії МК
#include "stm32f10x_rcc.h"
//підключаємо функції роботи з EXTI для даної серії МК
#include "stm32f10x_exti.h"

int main(void)
{
  int i;
  /* створюємо структуру конфігурації порту з ім’ям GPIO_InitStructure */
GPIO_InitTypeDef  GPIO_InitStructure;
  /* створюємо структуру конфігурації EXTI з ім’ям EXTI_InitStructure */
    EXTI_InitTypeDef EXTI_InitStructure;
  /* створюємо структуру конфігурації NVIC з ім’ям NVIC_InitStructure */
    NVIC_InitTypeDef NVIC_InitStructure;

  /* ініціалізуємо PC13 */
  // дозволяємо тактування порту C - PORTC Clock
```

```
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
  /* конфігуруємо PC13. Для цього заповнюємо поля структури конфігурації порту і визиваємо бібліотечну функцію ініціалізації*/
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(GPIOC, &GPIO_InitStructure);

//встановлюємо “0” (низький рівень) на PC13
  GPIO_ResetBits(GPIOC, GPIO_Pin_13); // Set C13 to Low level ("0")
//конфігуруємо NVIC
//включаємо тактування NVIC
RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
//дозволяємо зовнішнє переривання EXTI0
EXTI_InitStructure.EXTI_Line = EXTI_Line0;
EXTI_InitStructure.EXTI_LineCmd = ENABLE;
//налаштовуємо спрацювання EXTI0 по наростаючому фронту
EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
//ініціалізуємо конфігурацію
EXTI_Init(&exti_init);
//зв’язуємо EXTI0 з PB0
SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource0);
NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
//визначаємо пріоритет і субпріоритет
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
NVIC_Init(&NVIC_InitStructure);
```

```
/* нескінченний цикл */
while(1) ;
}

//обробник переривання EXTI0
void EXTI0_IRQHandler(void)
{
	EXTI_ClearITPendingBit(EXTI_Line0);
	GPIO_ToggleBits(GPIOC, GPIO_Pin_13);
}
```

3.8. АЦП мікроконтролерів STM32

STM32, залежно від версії, може містити кілька незалежних аналогово-цифрових перетворювачів. АЦП мають окреме джерело живлення з напругою в діапазоні від 2,4 до 3,6 В, залежно від типу корпусу.

АЦП мають роздільну здатність 12-біт і швидкість перетворення 1 МГц. АЦП може містити до 18 каналів, 16 з яких виводяться назовні. Решту два використовують для підключення вбудованого температурного датчика і внутрішнього опорного сигналу (рис 3.16).

Розглянемо схему включення АЦП. Аналого-цифровий перетворювач має окремий вхід живлення (пін V_{DDA}). Для чіткого перетворення треба, щоб живлення АЦП було стабільним, тому краще подавати живлення на V_{SSA} та V_{DDA} окремо (рис. 3.17).

Бажано використовувати два паралельно включених фільтруючих конденсатора: електролітичний ємністю 1 мкФ, і керамічний – 10 нФ. На схемі вони позначені одним як «1 μF // 10 nF».

У випадку, коли немає потреби у високій стабільності АЦП, можна використовувати спрощену схему підключення представлену на рис. 3.18.

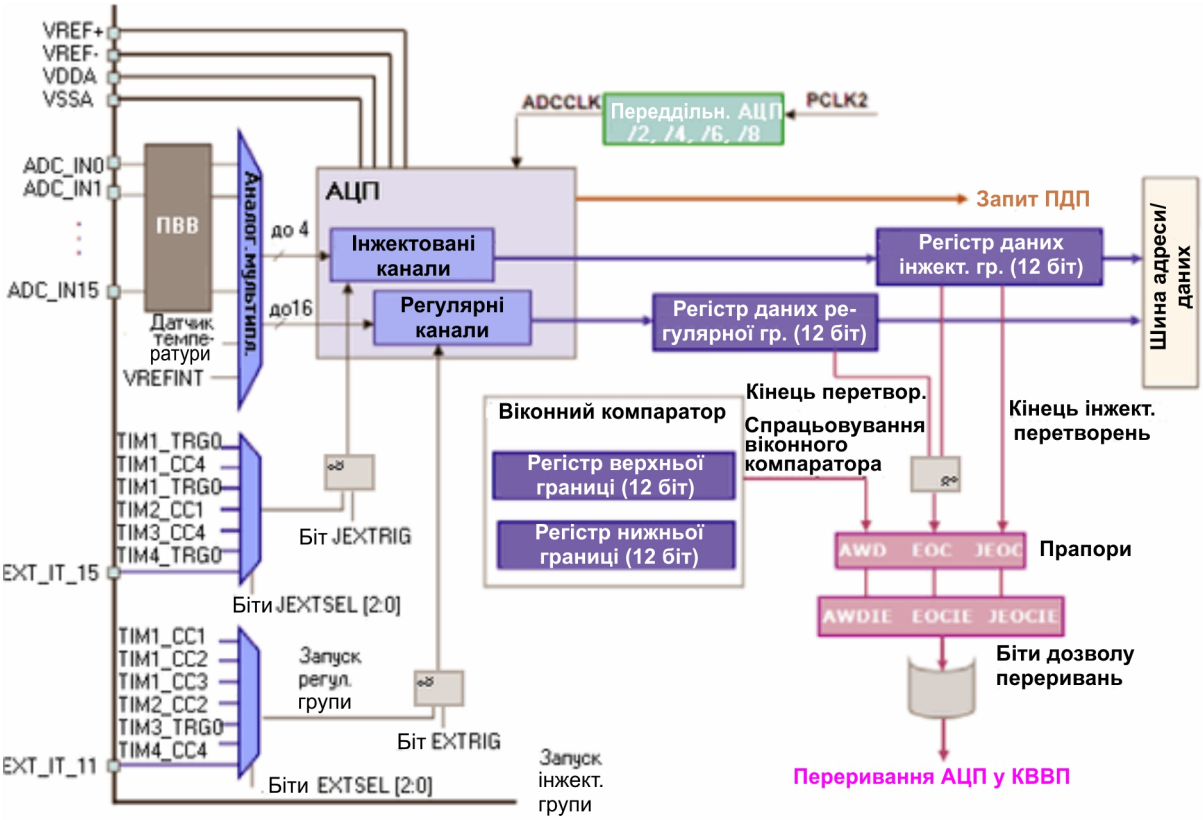


Рис. 3.16. Система АЦП у структурі мікроконтролерів STM32F

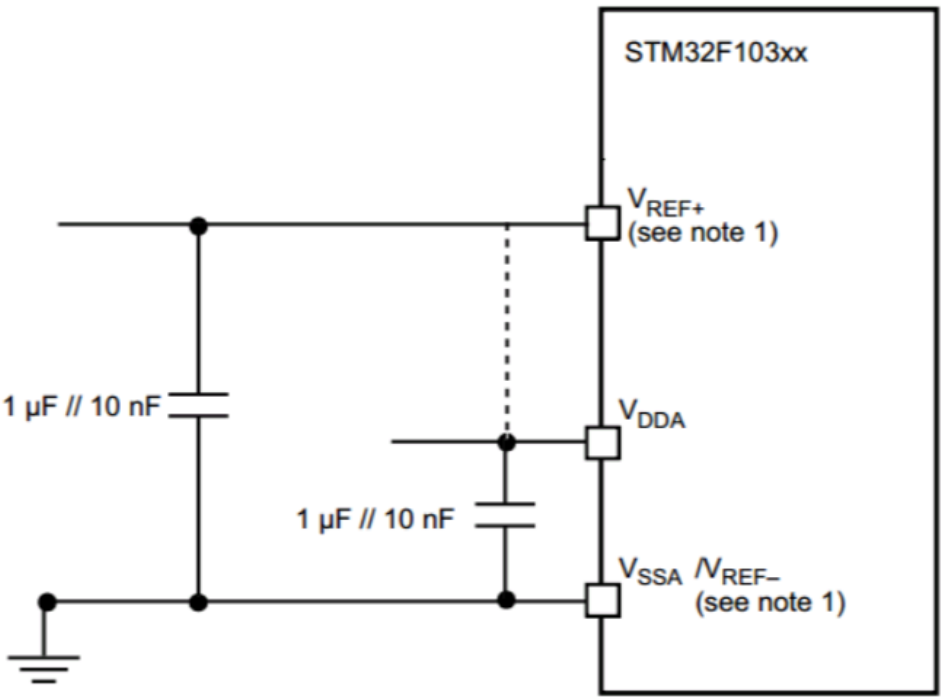


Рис. 3.17. Рекомендована схема підключення опорної напруги і напруги живлення внутрішніх АЦП мікроконтролерів STM32F

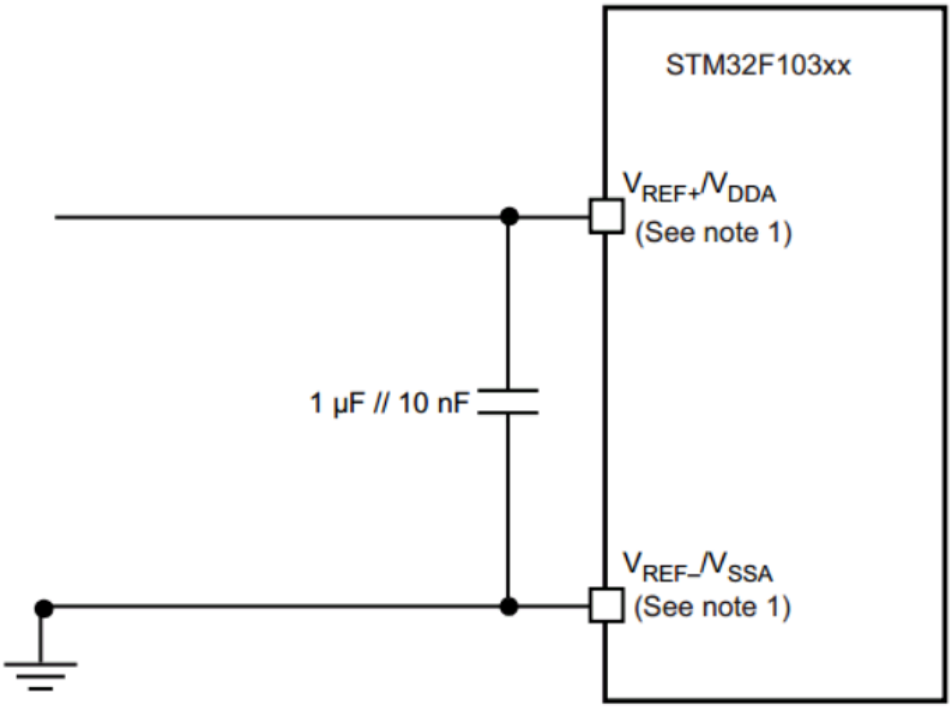


Рис. 3.18. Спрощена схема підключення опорної напруги і напруги живлення внутрішніх АЦП мікроконтролерів STM32F

Для вирішення особливих завдань на $V_{REF–}$ та V_{REF+} може подаватися опорна напруга, відмінна від напруги живлення АЦП. Але вона не може бути вищою за напругу живлення АЦП (V_{DDA}). А V_{DDA}, в свою чергу, не може бути вищою за 3,6 В.

Внутрішні АЦП мікроконтролерів STM32F можуть працювати в кількох різних режимах перетворення:

– одноразове перетворення;

– неперервне перетворення;

– перетворення по тригеру;

– перетворення за таймером.

Крім того АЦП мікроконтролерів STM32F забезпечують наступні можливості:

– вирівнювання бітів результату (вправо або вліво);

– генерування переривань і сигналів для DMA;

– швидкість оцифровування – до 1 MSPS (до одного мільйона вибірок в секунду);

– автоматичне калібрування;

– режим сканування входів за списком;

– реалізація аналогового сторожа (watchdog, коли АЦП відстежує рівень сигналу і, коли той виходить за вказані межі, генерується переривання).

Канали АЦП мікроконтролерів STM32 діляться на дві групи: регулярні канали (regular) та інжектовані (injected). Результати вимірювань регулярних каналів зберігаються в одному регістрі і вимагають збереження результатів в пам'яті мікроконтролера, а інжектовані канали мають власні регістри для зберігання результату. АЦП можуть працювати як в одиночному, так і в парному режимі. Парна робота використовується для підвищення швидкості перетворень. При цьому можливі наступні варіанти:

1. Regular / Injected simultaneous (одночасний для Regular і Injected каналів). Перший АЦП сканує канали починаючи з 0 до 15 для регулярних каналів, або з 0 до 3 для інжектованих каналів. Другий – в зворотному напрямку, з 15 до 0 для регулярних або 3 до 0 для інжектованих каналів. Таким чином, кожен канал за той же період часу буде оброблений двічі.

2. Fast interleaved (швидкий почерговий). Доступний тільки для одного обраного регулярного каналу. При запуску перетворення АЦП2 стартує перший, а АЦП1 – із затримкою в сім тактів. Якщо встановлений режим «continuous», то така послідовність буде повторюватися далі, що дає можливість подвоїти частоту вимірювання вхідного сигналу.

3. Slow interleaved (повільний почерговий). Все як в попередньому режимі, тільки АЦП1 стартує з затримкою в 14 тактів, а після закінчення першого перетворення АЦП2 теж витримує затримку 14 тактів перед повторним стартом.

4. Alternate trigger (альтернативний по тригеру). Доступний тільки для інжектованих каналів. АЦП1 починає послідовно конвертувати всі інжектовані канали. Коли відбувається наступна подія запускає конвертацію, АЦП2 починає робити те ж саме. Якщо встановлений режим «discontinuous», то буде перетворений тільки один канал на кожну подію.

5. Combined regular / injected simultaneous mode (комбінований одночасний режим для регулярних / інжектованих каналів). Відбувається перетворення в обох групах каналів. Але група інжектованих каналів може перервати обробку групи регулярних каналів. Тобто, якщо відбудеться подія, що запускає конвертування інжектованих каналів, а в цей час АЦП займається обробкою регулярних каналів, то АЦП переключиться на обробку інжектованих каналів.
6. Combined regular simultaneous + alternate trigger mode. Регулярна група каналів може бути перервана почерговим запуском інжектованих каналів. При завершенні перетворення групи зупиняються всі інші перетворення обох груп, і результати обробки зберігаються в регістрах даних кожного АЦП.

Розглянемо приклад:

Режим Single continuous. АЦП постійно вимірює напругу на одному з каналів, а зчитування даних відбувається довільним чином. У цьому прикладі відбувається опитування каналу 1 (ADC12_IN1) (пін A1), а передача результату перетворення відбувається по USART1.

Приклад 3.7:

```
#include "stm32f10x.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_usart.h"
#include "stm32f10x_adc.h"
#include "misc.h"
volatile char buffer[50] = {'\0'};

void usart_init(void)
```

```
{
/* дозвіл тактування USART1 та GPIOA */
RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1                    |
                       RCC_APB2Periph_GPIOA, ENABLE);

    /*конфігурування  NVIC */
    NVIC_InitTypeDef NVIC_InitStructure;
    /* дозвіл переривання USARTx */
    NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    /* конфігурування GPIOs */
    //визначення структури для конфігурації пінів портів;
    GPIO_InitTypeDef GPIO_InitStructure;

    /* конфігурування піну (PA.09) як Tx для USART1 */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    /*конфігурування піну (PA.10) як Rx для USART1*/
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    /*конфігурування USART1 */
```

```
// визначення структури для конфігурації
USART_InitTypeDef USART_InitStructure;

/* параметри конфігурації  USART1 */
/*          - BaudRate = 115200 baud
        - Word Length = 8 Bits
        - One Stop Bit
        - No parity
        - Hardware flow control disabled (RTS and CTS signals)
        - Receive and transmit enabled
        - USART Clock disabled
        - USART CPOL: Clock is active low
        - USART CPHA: Data is captured on the middle
        - USART LastBit: The clock pulse of the last data bit is not output to
                         the SCLK pin
 */
USART_InitStructure.USART_BaudRate = 115200;
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
USART_InitStructure.USART_StopBits = USART_StopBits_1;
USART_InitStructure.USART_Parity = USART_Parity_No;
USART_InitStructure.USART_HardwareFlowControl =
                                    USART_HardwareFlowControl_None;
USART_InitStructure.USART_Mode=USART_Mode_Rx|USART_Mode_Tx;
USART_Init(USART1, &USART_InitStructure);

/* дозвіл USART1 */
USART_Cmd(USART1, ENABLE);
```

```
    /* дозвіл переривання USART1 при надходженні даних (Receive interrupt: дане переривання генерується модулем USART1, коли в регістр приймання даних поступили нові дані */
    //USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);

}

void USARTSend(const unsigned char *pucBuffer, unsigned long ulCount)
{
  //
  // цикл, поки не відіслані всі дані.
  //
  while(ulCount--)
  {
    USART_SendData(USART1, *pucBuffer++);// Last Version USART_SendData(USART1,(uint16_t) *pucBuffer++);
    /* Loop until the end of transmission */
    while(USART_GetFlagStatus(USART1, USART_FLAG_TC) == RESET)
    {
    }
  }
}

void SetSysClockTo72(void)
{
     ErrorStatus HSEStartUpStatus;
  /*конфігурування  SYSCLK, HCLK, PCLK2 and PCLK1 ----------------------*/
  /* RCC деініціалізація системи тактування (лише для режиму відладки) */
  RCC_DeInit();
```

```
/* дозвіл HSE */
RCC_HSEConfig( RCC_HSE_ON);

/* затримка, поки HSE не буде готовий до роботи */
HSEStartUpStatus = RCC_WaitForHSEStartUp();

if (HSEStartUpStatus == SUCCESS)
{
   /* увімкнути буфер*/
     //FLASH_PrefetchBufferCmd( FLASH_PrefetchBuffer_Enable);

   /* перехід Flash 2 у стан очікування */
   //FLASH_SetLatency( FLASH_Latency_2);

   /* HCLK = SYSCLK */
   RCC_HCLKConfig( RCC_SYSCLK_Div1);

   /* PCLK2 = HCLK */
   RCC_PCLK2Config( RCC_HCLK_Div1);

   /* PCLK1 = HCLK/2 */
   RCC_PCLK1Config( RCC_HCLK_Div2);

   /* PLLCLK = 8MHz * 9 = 72 MHz */
   RCC_PLLConfig(0x00010000, RCC_PLLMul_9);

   /* Enable PLL */
   RCC_PLLCmd( ENABLE);

   /* Wait till PLL is ready */
```

```
      while (RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET)
      {
      }

      /* Select PLL as system clock source */
      RCC_SYSCLKConfig( RCC_SYSCLKSource_PLLCLK);

      /* очікування поки включиться в роботу PLL як джерело тактування */
      while (RCC_GetSYSCLKSource() != 0x08)
      {
      }
   }
   else
   { /* If HSE fails to start-up, the application will have wrong clock configuration.
    User can add here some code to deal with this error */

      /* Go to infinite loop */
      while (1)
      {
      }
   }
}

int main(void)
{
        const unsigned char mytext[] = " Hello World!\r\n";
        int adc_value;

        SetSysClockTo72();
```

```
//USART1
usart_init();
USART_SendData(USART1, '\r');
USARTSend(mytext, sizeof(mytext));

//ADC
ADC_InitTypeDef ADC_InitStructure;
GPIO_InitTypeDef  GPIO_InitStructure;
/* input of ADC (it doesn't seem to be needed, as default GPIO state is floating input) */
GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AIN;
GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_1 ;            // that's ADC1 (PA1 on STM32)
GPIO_Init(GPIOA, &GPIO_InitStructure);

//clock for ADC (max 14MHz --> 72/6=12MHz)
RCC_ADCCLKConfig (RCC_PCLK2_Div6);
// enable ADC system clock
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

// define ADC config
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
ADC_InitStructure.ADC_ScanConvMode = DISABLE;
ADC_InitStructure.ADC_ContinuousConvMode = ENABLE; // we work in continuous sampling mode
ADC_InitStructure.ADC_ExternalTrigConv                    = ADC_ExternalTrigConv_None;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfChannel = 1;
```

```
	ADC_RegularChannelConfig(ADC1,ADC_Channel_1,
1,ADC_SampleTime_28Cycles5); // define regular conversion config
	ADC_Init ( ADC1, &ADC_InitStructure);   //set config of ADC1

	// дозвіл ADC
	ADC_Cmd (ADC1,ENABLE);//enable ADC1

	//калібрування ADC (не обов'язково, рекомендовано при включенні живлення)
	ADC_ResetCalibration(ADC1);//скидання попереднього калібрування
	while(ADC_GetResetCalibrationStatus(ADC1));
	ADC_StartCalibration(ADC1);//калібрування (ADC повинен бути виключений)
	while(ADC_GetCalibrationStatus(ADC1));

	// запуск перетворення
	ADC_Cmd (ADC1,ENABLE);//enable ADC1
	// старт перетворення (нескінченне перетворення бо «continuous mode»)
	ADC_SoftwareStartConvCmd(ADC1, ENABLE);

	while (1)
	{
		adc_value = ADC_GetConversionValue(ADC1);
		sprintf(buffer, "%d\r\n", adc_value);
		USARTSend(buffer, sizeof(buffer));
	}
}
```

3.9. Контролер DMA мікроконтролерів STM32

Контролер DMA використовують для прямого доступу до пам'яті МК (DMA – direct memory access). Його основне призначення – передавання

даних на апаратному рівні між пам'яттю і периферійними пристроями без безпосередньої участі процесора. У попередньому розділі в прикладі було розглянуто використання DMA при роботі з АЦП. Тепер розглянемо роботу DMA докладніше на прикладі ще однієї типової задачі: відправлення даних через USART.

Розглянемо таблицю 3.5 і схему контролера (рис. 3.19) з документації, щоб зрозуміти, як налаштувати DMA і які у нього є можливості. У DMA контролера 7 каналів. У мікроконтролерів STM32 може бути кілька DMA контролерів, які можуть працювати паралельно.

Таблиця 3.5.

Розподіл каналів контролера DMA1 мікроконтролерів STM32

Peripherals	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
ADC1	ADC1	-	-	-	-	-	-
SPI/I^2S	-	SPI1_RX	SPI1_TX	SPI2/I2S2_RX	SPI2/I2S2_TX	-	-
USART	-	USART3_TX	USART3_RX	USART1_TX	USART1_RX	USART2_RX	USART2_TX
I^2C	-	-	-	I2C2_TX	I2C2_RX	I2C1_TX	I2C1_RX
TIM1		TIM1_CH1	-	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	
TIM2	TIM2_CH3	TIM2_UP	-	-	TIM2_CH1	-	TIM2_CH2 TIM2_CH4
TIM3	-	TIM3_CH3	TIM3_CH4 TIM3_UP	-	-	TIM3_CH1 TIM3_TRIG	-
TIM4	TIM4_CH1	-	-	TIM4_CH2	TIM4_CH3	-	TIM4_UP

Зі схеми і таблиці видно, що певна периферія закріплена за певними каналами. Наприклад, ADC1 може використовувати тільки канал 1. При цьому цей же канал призначений для роботи таймерами TIM2_CH3 та TIM4_CH1. Одночасна робота каналу DMA з кількома пристроями неможлива. І немає можливості перенести ADC1 на інший канал. Для роботи з USART1 можуть бути задіяні канали 4 і 5. Зверніть увагу, що, скажімо, ADC2 взагалі тут не фігурує. Тобто, не будь-яку периферію можна використовувати з DMA. Можливих комбінацій, як бачимо, не так багато і треба все чітко планувати, щоб канали DMA були зайняті виключно у справі.

Також слід розуміти, що у DMA є обмеження по швидкості.

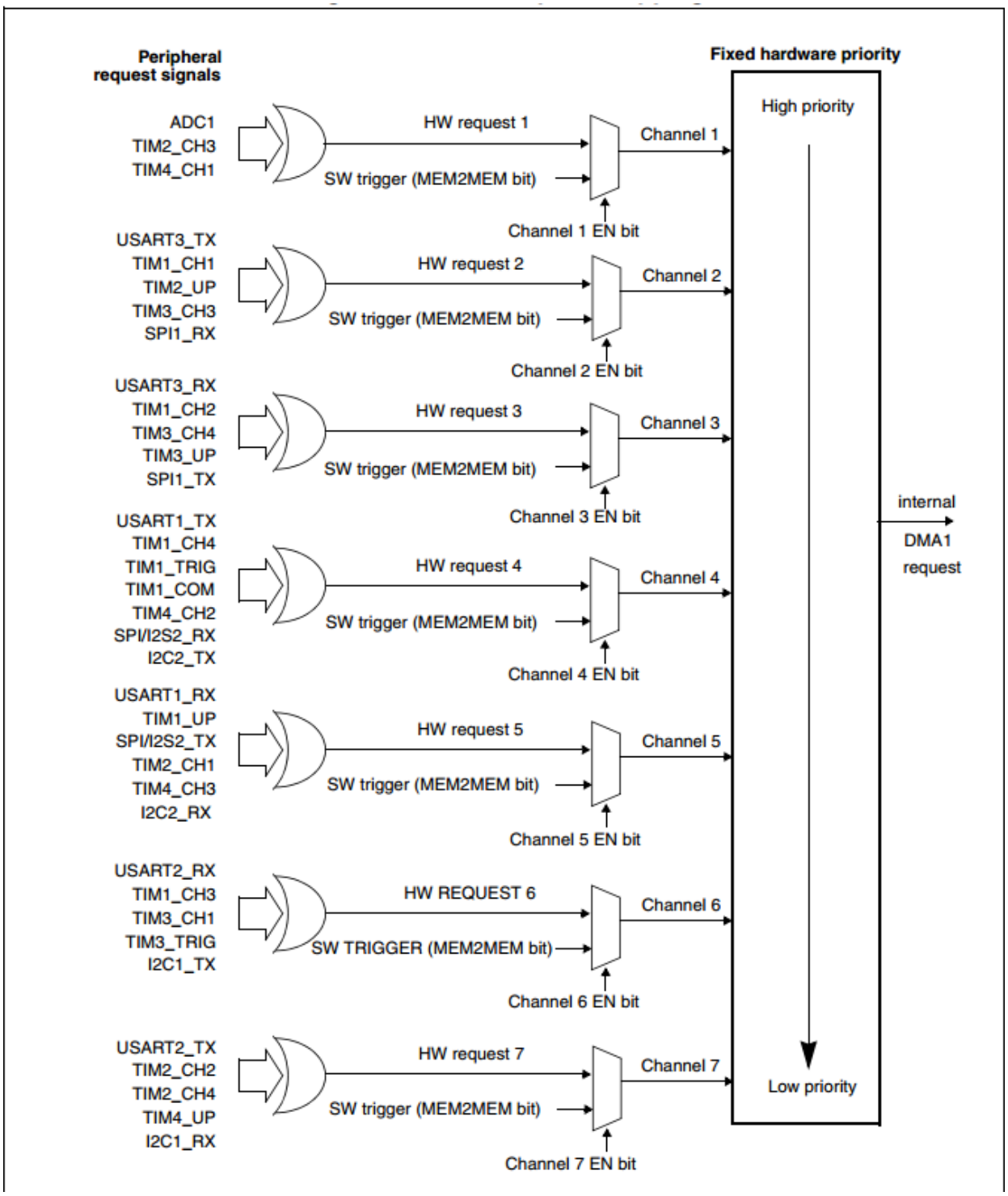


Рис. 3.19. Спрощена схема контролера DMA1 мікроконтролерів STM32

Налаштування DMA виконується через структуру InitTypeDef, опис якої наведено нижче:

```
typedef struct
{
    uint32_t DMA_PeripheralBaseAddr;
    uint32_t DMA_MemoryBaseAddr;
    uint32_t DMA_DIR;
    uint32_t DMA_BufferSize;
    uint32_t DMA_PeripheralInc;
    uint32_t DMA_MemoryInc;
    uint32_t DMA_PeripheralDataSize
    uint32_t DMA_MemoryDataSize;
    uint32_t DMA_Mode;
    uint32_t DMA_Priority;
    uint32_t DMA_M2M;
} DMA_InitTypeDef;
```

DMA_PeripheralBaseAddr – адреса периферійного пристрою.
DMA_MemoryBaseAddr – адреса пам'яті.
DMA_DIR – напрям передачі. Дані можуть передаватися з периферії в пам'ять і навпаки (DMA_DIR_PeripheralDST, DMA_DIR_PeripheralSRC).
DMA_BufferSize – розмір буфера даних.
DMA_PeripheralInc – вказує чи треба інкрементувати адреси даних в периферії (DMA_PeripheralInc_Enable, DMA_PeripheralInc_Disable).
DMA_MemoryInc – вказує чи треба інкрементувати адреси даних в пам'яті (DMA_MemoryInc_Enable, DMA_MemoryInc_Disable).

Видно, що в налаштуваннях DMA

DMA_MemoryInc = Enable
DMA_PeripheralInc = Disable

Це тому, що дані АЦП розкладаються в масив і потрібно включити інкрементацію адрес в пам'яті. Щоб дані з різних каналів записувалися у різні ділянки пам’яті. А вихідний регістр в АЦП один, і слід вимкнути інкрементацію на периферії.

DMA_PeripheralDataSize – розмір одиниці даних для периферії.
DMA_MemoryDataSize – розмір одиниці даних для пам'яті.

Ці поля можуть набувати наступних значень:
- DMA_PeripheralDataSize_Byte;
- DMA_PeripheralDataSize_HalfWord;
- DMA_PeripheralDataSize_Word;
- DMA_MemoryDataSize_Byte;
- DMA_MemoryDataSize_HalfWord;
- DMA_MemoryDataSize_Word.

DMA_Mode – режим роботи каналу DMA (DMA_Mode_Circular, DMA_Mode_Normal)
DMA_Priority – пріоритет каналу DMA (DMA_Priority_VeryHigh, DMA_Priority_High, DMA_Priority_Medium, DMA_Priority_Low)
DMA_M2M – передача пам'ять → пам'ять (DMA_M2M_Enable, DMA_M2M_Disable).

Розглянемо приклад. У цьому прикладі слід звернути увагу на роботу функції USARTSendDMA. Вона тільки записує в буфер інформацію для відправлення і запускає DMA канал на відправку. Вона не чекає поки буде відправлений буфер, тому, якщо викликати функцію USARTSendDMA до закінчення відправки буфера, вона перезапише буфер, що треба мати на увазі.

Приклад 3.8:

```
#include "stm32f10x.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_usart.h"
#include "stm32f10x_dma.h"
#include "misc.h"
#include <string.h>

#define RX_BUF_SIZE 80
volatile char RX_FLAG_END_LINE = 0;
volatile char RXi;
volatile char RXc;
volatile char RX_BUF[RX_BUF_SIZE] = {'\0'};
volatile char buffer[80] = {'\0'};

void clear_RXBuffer(void) {
   for (RXi=0; RXi<RX_BUF_SIZE; RXi++)
      RX_BUF[RXi] = '\0';
   RXi = 0;
}

void usart_dma_init(void)
{
   /* Enable USART1 and GPIOA clock */
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1                    |
RCC_APB2Periph_GPIOA, ENABLE);
   RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
```

```
/* DMA */
DMA_InitTypeDef DMA_InitStruct;
DMA_InitStruct.DMA_PeripheralBaseAddr = (uint32_t)&(USART1->DR);
DMA_InitStruct.DMA_MemoryBaseAddr = (uint32_t)&buffer[0];
DMA_InitStruct.DMA_DIR = DMA_DIR_PeripheralDST;
DMA_InitStruct.DMA_BufferSize = sizeof(buffer);
DMA_InitStruct.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
DMA_InitStruct.DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_InitStruct.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte;
DMA_InitStruct.DMA_MemoryDataSize = DMA_MemoryDataSize_Byte;
DMA_InitStruct.DMA_Mode = DMA_Mode_Normal;
DMA_InitStruct.DMA_Priority = DMA_Priority_Low;
DMA_InitStruct.DMA_M2M = DMA_M2M_Disable;
DMA_Init(DMA1_Channel4, &DMA_InitStruct);

/* NVIC Configuration */
NVIC_InitTypeDef NVIC_InitStructure;
/* Enable the USARTx Interrupt */
NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);

/* Configure the GPIOs */
GPIO_InitTypeDef GPIO_InitStructure;

/* Configure USART1 Tx (PA.09) as alternate function push-pull */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
```

```
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOA, &GPIO_InitStructure);

/* Configure USART1 Rx (PA.10) as input floating */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
GPIO_Init(GPIOA, &GPIO_InitStructure);

/* Configure the USART1 */
USART_InitTypeDef USART_InitStructure;

/* USART1 configuration ------------------------------------------------------*/
/* USART1 configured as follow:
   - BaudRate = 115200 baud
   - Word Length = 8 Bits
   - One Stop Bit
   - No parity
   - Hardware flow control disabled (RTS and CTS signals)
   - Receive and transmit enabled
   - USART Clock disabled
   - USART CPOL: Clock is active low
   - USART CPHA: Data is captured on the middle
   - USART LastBit: The clock pulse of the last data bit is not output to
     the SCLK pin
 */
USART_InitStructure.USART_BaudRate = 115200;
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
USART_InitStructure.USART_StopBits = USART_StopBits_1;
USART_InitStructure.USART_Parity = USART_Parity_No;
USART_InitStructure.USART_HardwareFlowControl =
```

```
                                                    USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode =
                                                    USART_Mode_Rx | USART_Mode_Tx;

    USART_Init(USART1, &USART_InitStructure);

    /* Enable USART1 */
    USART_Cmd(USART1, ENABLE);

    USART_DMACmd(USART1, USART_DMAReq_Tx, ENABLE);
    //DMA_Cmd(DMA1_Channel4, ENABLE);

    DMA_ITConfig(DMA1_Channel4, DMA_IT_TC, ENABLE);
    NVIC_EnableIRQ(DMA1_Channel4_IRQn);

     /* Enable the USART1 Receive interrupt: this interrupt is generated when the
    USART1 receive data register is not empty */
    USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
}

void USART1_IRQHandler(void)
{
    if ((USART1->SR & USART_FLAG_RXNE) != (u16)RESET)
    {
          RXc = USART_ReceiveData(USART1);
          RX_BUF[RXi] = RXc;
          RXi++;

          if (RXc != 13) {
             if (RXi > RX_BUF_SIZE-1) {
```

```
            clear_RXBuffer();
          }
        }
        else {
          RX_FLAG_END_LINE = 1;
        }

        //Echo
        USART_SendData(USART1, RXc);
    }
}

void USARTSendDMA(const unsigned char *pucBuffer)
{
    strcpy(buffer, pucBuffer);

    /* Restart DMA Channel*/
    DMA_Cmd(DMA1_Channel4, DISABLE);
    DMA1_Channel4->CNDTR = strlen(pucBuffer);
    DMA_Cmd(DMA1_Channel4, ENABLE);
}

void DMA1_Channel4_IRQHandler(void)
{
    DMA_ClearITPendingBit(DMA1_IT_TC4);
    DMA_Cmd(DMA1_Channel4, DISABLE);
}

void SetSysClockTo72(void)
{
```

```
  ErrorStatus HSEStartUpStatus;
  /* SYSCLK, HCLK, PCLK2 and PCLK1 configuration ----------------------------
*/
  /* RCC system reset(for debug purpose) */
  RCC_DeInit();

  /* Enable HSE */
  RCC_HSEConfig( RCC_HSE_ON);

  /* Wait till HSE is ready */
  HSEStartUpStatus = RCC_WaitForHSEStartUp();

  if (HSEStartUpStatus == SUCCESS)
  {
    /* Enable Prefetch Buffer */
    //FLASH_PrefetchBufferCmd( FLASH_PrefetchBuffer_Enable);

    /* Flash 2 wait state */
    //FLASH_SetLatency( FLASH_Latency_2);

    /* HCLK = SYSCLK */
    RCC_HCLKConfig( RCC_SYSCLK_Div1);

    /* PCLK2 = HCLK */
    RCC_PCLK2Config( RCC_HCLK_Div1);

    /* PCLK1 = HCLK/2 */
    RCC_PCLK1Config( RCC_HCLK_Div2);

    /* PLLCLK = 8MHz * 9 = 72 MHz */
```

```
    RCC_PLLConfig(0x00010000, RCC_PLLMul_9);

    /* Enable PLL */
    RCC_PLLCmd( ENABLE);

    /* Wait till PLL is ready */
    while (RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET)
    {
    }

    /* Select PLL as system clock source */
    RCC_SYSCLKConfig( RCC_SYSCLKSource_PLLCLK);

    /* Wait till PLL is used as system clock source */
    while (RCC_GetSYSCLKSource() != 0x08)
    {
    }
  }
  else
  { /* If HSE fails to start-up, the application will have wrong clock configuration.
   User can add here some code to deal with this error */

    /* Go to infinite loop */
    while (1)
    {
    }
  }
}

int main(void)
```

```
{
  // Set System clock
  SetSysClockTo72();

  /* Initialize LED which connected to PC13 */
  GPIO_InitTypeDef  GPIO_InitStructure;
  // Enable PORTC Clock
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
  /* Configure the GPIO_LED pin */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(GPIOC, &GPIO_InitStructure);

  GPIO_ResetBits(GPIOC, GPIO_Pin_13); // Set C13 to Low level ("0")

  // Initialize USART
  usart_dma_init();
  USARTSendDMA("Hello.\r\nUSART1 is ready.\r\n");

  while (1)
  {
    if (RX_FLAG_END_LINE == 1) {
      // Reset END_LINE Flag
      RX_FLAG_END_LINE = 0;

      /* !!! This lines is not have effect. Just a last command
USARTSendDMA(":\r\n"); !!!! */
      USARTSendDMA("\r\nI has received a line:\r\n"); // no effect
      USARTSendDMA(RX_BUF); // no effect
```

```
USARTSendDMA(":\r\n"); // This command does not wait for the finish of the sending of buffer. It just write to buffer new information and restart sending via DMA.

        if (strncmp(strupr(RX_BUF), "ON\r", 3) == 0) {
          USARTSendDMA("THIS IS A COMMAND \"ON\"!!!\r\n");
          GPIO_ResetBits(GPIOC, GPIO_Pin_13);
        }

        if (strncmp(strupr(RX_BUF), "OFF\r", 4) == 0) {
          USARTSendDMA("THIS IS A COMMAND \"OFF\"!!!\r\n");
          GPIO_SetBits(GPIOC, GPIO_Pin_13);
        }

        clear_RXBuffer();
      }
    }
}
```

3.10. Таймери мікроконтролерів STM32

Таймери в мікроконтролерах STM32 діляться за функціоналом на три типи:

– basic timers (базові таймери);

– general-purpose timers (загального призначення: TIM2, TIM3, TIM4);

– advanced-control timers (таймери з розширеними можливостями: TIM1).

У різних мікроконтролерах кількість таймерів різна. Згідно з документацією до контролера STM32F103C8 він містить 3 таймери general-purpose, і один advanced-control (табл. 3.6).

Таблиця 3.6.

Таймери мікроконтролерів STM32103xx

Peripheral		STM32F103Tx		STM32F103Cx		STM32F103Rx		STM32F103Vx	
Flash - Kbytes		64	128	64	128	64	128	64	128
SRAM - Kbytes		20		20		20		20	
Timers	General-purpose	3		3		3		3	
	Advanced-control	1		1		1		1	

Таймери можуть не тільки генерувати переривання через певний час і вимірювати час між подіями. Вони також можуть генерувати PWM (ШІМ) сигнал, і поодинокі імпульси, працювати з периферією, наприклад, запускати перетворення АЦП, виконувати захоплення сигналу тощо.

Спочатку розглянемо таймери загального призначення.

Таймери підраховують імпульси, якими вони тактуються. У STM32 мікроконтролерах є дільник частоти для кожного таймера. Значення дільника можуть бути від 1 до 65535. Крім того, можна визначити число лічильника, досягнувши якого таймер буде генерувати переривання за переповненням, обнуляти значення лічильника і рахувати спочатку. Комбінуючи ці два параметри, можна домогтися потрібної частоти, з якою таймер буде генерувати переривання. Розглянемо приклад, у якому таймер буде налаштований таким чином, щоб він рахував до заданого числа та генерував переривання за переповненням. У цьому прикладі контролер буде тактуватися від зовнішнього кварцевого резонатора із частотою 8 МГц. Таймер також буде працювати на цій частоті, так як значення дільника APB1 = 1. Буде встановлено дільник 8000. Таймер буде рахувати з частотою 8000000/8000 = 1000 разів в секунду. Встановимо період = 500. Тобто, дорахувавши до 500 таймер буде генерувати переривання (це 2 рази в секунду). У обробнику переривання пропишемо команду для зміни стану піна (вважатимемо, що пін керує вмиканням світлодіоду).

Приклад 3.9:

```
#include "stm32f10x.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_tim.h"
#include "misc.h"

void SetSysClockToHSE(void)
{
   ErrorStatus HSEStartUpStatus;
   /*конфігурування  SYSCLK, HCLK, PCLK2 and PCLK1----------------------*/
   /* RCC system reset() //для режиму налагодження*/
   RCC_DeInit();

   /* дозвіл на тактування від зовнішнього високочастотного генератора HSE
*/
   RCC_HSEConfig( RCC_HSE_ON);

   /* чекаємо поки HSE буде запущений */
   HSEStartUpStatus = RCC_WaitForHSEStartUp();

   if (HSEStartUpStatus == SUCCESS)
   {
      /* HCLK = SYSCLK */
      RCC_HCLKConfig( RCC_SYSCLK_Div1);

      /* PCLK2 = HCLK */
      RCC_PCLK2Config( RCC_HCLK_Div1);
```

```
        /* PCLK1 = HCLK */
        RCC_PCLK1Config(RCC_HCLK_Div1);

        /* обираємо HSE як джерело тактування системи */
        RCC_SYSCLKConfig( RCC_SYSCLKSource_HSE);

        /* чекаємо, поки включиться PLL*/
        while (RCC_GetSYSCLKSource() != 0x04)
        {
                // PLL успішно включено, ніякі дії не потрібні
        }
    }
    else
    {
/* тут можна прописати дії на випадок невдачі ініціалізації PLL */
    }
}

//обробник переривання від TIMER4
void TIM4_IRQHandler(void)
{
        if (TIM_GetITStatus(TIM4, TIM_IT_Update) != RESET)
        {
           // обов'язково очищаємо біт запиту на переривання
           TIM_ClearITPendingBit(TIM4, TIM_IT_Update);
//інвертуємо стан PC13 (змінюємо рівень сигналу піну на протилежний)
           GPIOC->ODR ^= GPIO_Pin_13;
        }
}
```

```
int main(void)
{
/*встановлюємо тактування від високочастотного генератора HSE*/
   SetSysClockToHSE();

   /*ініціалізуємо пін PC13 як вихід для керування світлодіодом*/
   GPIO_InitTypeDef  GPIO_InitStructure;
   // дозволяємо тактування порту C
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
   /* конфігуруємо пін PC13 */
   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_Init(GPIOC, &GPIO_InitStructure);
// встановлюємо низький рівень (логічний "0") на піні Set PC13
   GPIO_ResetBits(GPIOC, GPIO_Pin_13);
   //починаємо конфігурувати TIMER4
// створюємо структуру опису параметрів конфігурації TIMER4
   TIM_TimeBaseInitTypeDef TIMER_InitStructure;
// створюємо структуру опису параметрів конфігурації NVIC
   NVIC_InitTypeDef NVIC_InitStructure;
   // дозволяємо тактування порту TIMER4
   RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);
//заповнюємо структуру опису параметрів конфігурації TIMER4
   TIM_TimeBaseStructInit(&TIMER_InitStructure);
   TIMER_InitStructure.TIM_CounterMode = TIM_CounterMode_Up;
   TIMER_InitStructure.TIM_Prescaler = 8000;
   TIMER_InitStructure.TIM_Period = 500;
   //ініціалізуємо (запускаємо) створену конфігурацію TIMER4
   TIM_TimeBaseInit(TIM4, &TIMER_InitStructure);
```

```
    TIM_ITConfig(TIM4, TIM_IT_Update, ENABLE);
    TIM_Cmd(TIM4, ENABLE);

    /*конфігуруємо NVIC */
    /*конфігуруємо параметри переривання TIM4_IRQn*/
    NVIC_InitStructure.NVIC_IRQChannel = TIM4_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    /*даємо дозвіл на переривання TIM4_IRQn*/
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    //ініціалізуємо (запускаємо) створену конфігурацію NVIC
    NVIC_Init(&NVIC_InitStructure);

    while(1)
    {
       // Тут прописуємо все, що нам потрібно.
    }
}
```

Часто виникає необхідність вимірювати довжину імпульсу або час між двома подіями. Для вирішення цього завдання теж можна застосовувати таймер. Розглянемо приклад, в якому таймер запускають натисканням кнопки, підключеної до PB0, а зупиняють натисканням кнопки, підключеної до PB1. Після зупинки таймера проводиться розрахунок часу між цими двома подіями і результат відправляється у послідовний порт USART.

Приклад 3.10:

```
#include "stm32f10x.h"
#include "stm32f10x_gpio.h"
```

```
#include "stm32f10x_rcc.h"
#include "stm32f10x_usart.h"
#include "stm32f10x_tim.h"
#include "misc.h"

volatile int TimeResult;
volatile int TimeSec;
volatile uint8_t TimeState = 0;

void SetSysClockTo72(void)
{
  ErrorStatus HSEStartUpStatus;
  /*конфігурування SYSCLK, HCLK, PCLK2 та PCLK1 ------------------------*/
  /*RCC system reset() //скидання системного тактування для режиму налагодження*/
  RCC_DeInit();

  /* дозвіл на тактування від HSE */
  RCC_HSEConfig( RCC_HSE_ON);

  /* чекаємо поки стартує HSE*/
  HSEStartUpStatus = RCC_WaitForHSEStartUp();

  if (HSEStartUpStatus == SUCCESS)
  {
    /* дозволяємо використання буфера */
    //FLASH_PrefetchBufferCmd( FLASH_PrefetchBuffer_Enable);

    /* Flash 2 wait state */
    //FLASH_SetLatency( FLASH_Latency_2);
```

```
        /* HCLK = SYSCLK */
        RCC_HCLKConfig( RCC_SYSCLK_Div1);
        /* PCLK2 = HCLK */
        RCC_PCLK2Config( RCC_HCLK_Div1);
        /* PCLK1 = HCLK/2 */
        RCC_PCLK1Config( RCC_HCLK_Div2);
        /* PLLCLK = 8MHz * 9 = 72 MHz */
        RCC_PLLConfig(0x00010000, RCC_PLLMul_9);
        /*дозволяємо використання PLL */
        RCC_PLLCmd( ENABLE);
        /*чекаємо поки PLL запусститься*/
        while (RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET)
        {
        }
        /* задаємо PLL як джерело системного тактування */
        RCC_SYSCLKConfig( RCC_SYSCLKSource_PLLCLK);
        /*чекаємо поки PLL запуститься */
        while (RCC_GetSYSCLKSource() != 0x08)  {    }
    }
    else
    { /* тут можна описати дії на випадок, якщо HSE не запуститься */
    }
}

//обробник запиту на переривання від TIMER4
void TIM4_IRQHandler(void)
{       if (TIM_GetITStatus(TIM4, TIM_IT_Update) != RESET)
        {//обов'язково очищаємо біт запиту на переривання від TIMER4
            TIM_ClearITPendingBit(TIM4, TIM_IT_Update);
            TimeSec++;
```

```
    }
}
//функція ініціалізації USART1
void usart_init(void)
{

    /* Enable USART1 and GPIOA clock */
RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1 |  RCC_APB2Periph_
                                        GPIOA, ENABLE);

    /*конфігурування портів*/
    /*створення структури для конфігурування портів*/
    GPIO_InitTypeDef GPIO_InitStructure;
    /* конфігурування PA.09 як USART1 Tx */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    //ініціалізація створеної конфігурації
    GPIO_Init(GPIOA, &GPIO_InitStructure);
    /* конфігурування PA.10 як USART1 Rx */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    /* конфігурування USART1 */
    /*створення структури для конфігурування USART1*/
    USART_InitTypeDef USART_InitStructure;
   /* опис конфігурації USART1 ------------------------------------------------*/
    /* Для USART1 встановлюємо:
        - BaudRate = 115200 baud
```

```
            - Word Length = 8 Bits
            - One Stop Bit
            - No parity
            - Hardware flow control disabled (RTS and CTS signals)
            - Receive and transmit enabled
            - USART Clock disabled
            - USART CPOL: Clock is active low
            - USART CPHA: Data is captured on the middle
            - USART LastBit: The clock pulse of the last data bit is not output to
                        the SCLK pin        */
      USART_InitStructure.USART_BaudRate = 115200;
      USART_InitStructure.USART_WordLength = USART_WordLength_8b;
      USART_InitStructure.USART_StopBits = USART_StopBits_1;
      USART_InitStructure.USART_Parity = USART_Parity_No;
      USART_InitStructure.USART_HardwareFlowControl =    USART_Hardware
                                          FlowControl_None;
     USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
      //ініціалізація створеної конфігурації
      USART_Init(USART1, &USART_InitStructure);
      USART_Cmd(USART1, ENABLE); /*запускаємо USART1 */
}
//функція передавання даних
void USARTSend(const unsigned char *pucBuffer)
{
   while (*pucBuffer)
   {
      USART_SendData(USART1, *pucBuffer++);
      while(USART_GetFlagStatus(USART1, USART_FLAG_TC) == RESET)
      {
      }
```

```
    }
}

int main(void)
{
    char buffer[80] = {'\0'};
    SetSysClockTo72();

    /*ініціалізуємо PC13 */
//створюємо структуру параметрів конфігурації порту
    GPIO_InitTypeDef  GPIO_InitStructure;
    // включаємо тактування порту C
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
    /*задаємо параметри конфігурації */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
//ініціалізуємо (запускаємо) конфігурацію
    GPIO_Init(GPIOC, &GPIO_InitStructure);
    GPIO_SetBits(GPIOC, GPIO_Pin_13); // встановлюємо C13 у стан логічного "0"

    /* ініціалізуємо PB0 PB1 */
    //дозволяємо тактування порту B
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
    /*записуємо параметри конфігурації порту C у структуру
GPIO_InitStructure, створену раніше при ініціалізації порту C*/
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
//ініціалізуємо (запускаємо) конфігурацію
```

```
    GPIO_Init(GPIOB, &GPIO_InitStructure);

    //конфігуруємо TIMER4
//створюємо структуру для запису параметрів конфігурації TIMER4
    TIM_TimeBaseInitTypeDef TIMER_InitStructure;
//створюємо структуру для запису параметрів конфігурації NVIC
    NVIC_InitTypeDef NVIC_InitStructure;
//включаємо тактування TIMER4
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);
//заповнюємо структуру параметрів конфігурації TIMER4
    TIM_TimeBaseStructInit(&TIMER_InitStructure);
    TIMER_InitStructure.TIM_CounterMode = TIM_CounterMode_Up;
    TIMER_InitStructure.TIM_Prescaler = 7200;
    TIMER_InitStructure.TIM_Period = 10000;
//ініціалізуємо (запускаємо) конфігурацію
    TIM_TimeBaseInit(TIM4, &TIMER_InitStructure);
    TIM_ITConfig(TIM4, TIM_IT_Update, ENABLE);
    TIM_Cmd(TIM4, ENABLE);

    /*конфігуруємо NVIC*/
    /*заповнюємо раніше створену структуру конфігурації NVIC і дозволяємо переривання TIM4_IRQn*/
    NVIC_InitStructure.NVIC_IRQChannel = TIM4_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
//ініціалізуємо (запускаємо) конфігурацію
    NVIC_Init(&NVIC_InitStructure);
//запускаємо функцію ініціалізації usart
    usart_init();
```

```
    while(1)
    {
      if (TimeState == 0) {
        //якщо натиснуто кнопку запуску таймера то:
        if (GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_0) == 0) {
          TIM_Cmd(TIM4, ENABLE);
          TIM_SetCounter(TIM4, 0);
          TimeSec = 0;
          // Set Status "ON"
          TimeState = 1;
          // встановлюємо CP13 у стан логічного “0”
          GPIO_ResetBits(GPIOC, GPIO_Pin_13);
          USARTSend("Started...");
        }
      }
      if (TimeState == 1) {
        // якщо натиснуто кнопку зупинки таймера то:
        if (GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_1) == 0) {
          TimeResult = TIM_GetCounter(TIM4)/10 + TimeSec * 1000; //час у мс
          TIM_Cmd(TIM4, DISABLE);
          TimeState = 0;
          GPIO_SetBits(GPIOC, GPIO_Pin_13); //встановлюємо на CP13 “1”
//записуємо інформацію в буфер
          sprintf(buffer, "Time: %d ms\r\n", TimeResult);
//передаємо записану у буфер інформацію
          USARTSend(buffer);
        }
      }
    }
}
```

Однією з типових задач для мікроконтролера є обробка вхідних сигналів. Використовуючи STM32 цю задачу можна вирішити за допомогою таймера загального призначення.

Ідея захоплення сигналу полягає в тому, що при зміні стану вхідного сигналу таймер зберігає в спеціальний регістр поточне значення лічильника і генерує переривання. Також є можливість налаштувати дільник таким чином, щоб таймер реагував тільки на кожен n-ний імпульс. Додатково можна налаштувати фільтр, який використовується коли сигнал зашумлений. Фільтр працює як зворотний лічильник. Тобто, коли на вході сигнал змінив стан, таймер віднімає від числа, зазначеного у фільтрі, одиницю і чекає наступної вибірки. Перевірка сигналу повторюється, поки лічильник фільтру не досягне нуля, і, якщо після цього сигнал залишився стабільним, надсилається запит на переривання.

Приклад 3.11:

```
#include "stm32f10x.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_tim.h"

void SetSysClockTo72(void)
{
  ErrorStatus HSEStartUpStatus;
  /*конфігурування  SYSCLK, HCLK, PCLK2 and PCLK1 ----------------------*/
  /* RCC system reset() скидання системного тактування для налагодження*/
  RCC_DeInit();
  /* дозвіл на використання HSE */
  RCC_HSEConfig( RCC_HSE_ON);
  /*чекаємо готовності HSE*/
```

```
    HSEStartUpStatus = RCC_WaitForHSEStartUp();
//перевіряємо статус HSE
    if (HSEStartUpStatus == SUCCESS)
    {//якщо все гаразд то:
       /* Enable Prefetch Buffer */
       //FLASH_PrefetchBufferCmd( FLASH_PrefetchBuffer_Enable);
       /* Flash 2 wait state */
       //FLASH_SetLatency( FLASH_Latency_2);
       /* HCLK = SYSCLK */
       RCC_HCLKConfig( RCC_SYSCLK_Div1);
       /* PCLK2 = HCLK */
       RCC_PCLK2Config( RCC_HCLK_Div1);
       /* PCLK1 = HCLK/2 */
       RCC_PCLK1Config( RCC_HCLK_Div2);
       /* PLLCLK = 8MHz * 9 = 72 MHz */
       RCC_PLLConfig(0x00010000, RCC_PLLMul_9);
       /*дозволяємо використання PLL */
       RCC_PLLCmd( ENABLE);
       /*чекаємо готовності PLL*/
       while (RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET)
       {
       }
       /*задаємо PLL як джерело системного тактування */
       RCC_SYSCLKConfig( RCC_SYSCLKSource_PLLCLK);
       /*чекаємо виконання попередньої команди*/
       while (RCC_GetSYSCLKSource() != 0x08)
       {     }
    }
    else
    { /*тут можна запрограмувати дії на випадок якщо з HSE не все гаразд*/
```

```
    }
}

volatile uint16_t PPMBuffer[] = {0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000};
volatile uint8_t PPMi = 0;
volatile uint16_t PPMValue_Prev, PPMValue;
//функція ініціалізації ppm (ppm англ. Parts per million)
void ppm_init() {
/*створення структури для параметрів ініціалізації пінів пов'язаних з таймером TIMER2*/
    GPIO_InitTypeDef gpio_cfg;
    GPIO_StructInit(&gpio_cfg);
    /*дозвіл тактування порту A і конфігурування таймеру TIMER2, канал 1 */
      RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
      //gpio_cfg.GPIO_Mode = GPIO_Mode_IN_FLOATING;
      gpio_cfg.GPIO_Mode = GPIO_Mode_IPU;
      gpio_cfg.GPIO_Pin = GPIO_Pin_1;
//ініціалізація створеної конфігурації
      GPIO_Init(GPIOA, &gpio_cfg);
      /*дозвіл тактування таймеру TIMER2 enable clock */
      RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
      /*створення конфігурації таймеру TIMER2 */
// створення структури для параметрів ініціалізації таймеру TIMER2
      TIM_TimeBaseInitTypeDef timer_base;
      TIM_TimeBaseStructInit(&timer_base);
//заповнення структури параметрами ініціалізації таймеру TIMER2
      timer_base.TIM_Prescaler = 72;
//ініціалізація створеної конфігурації
      TIM_TimeBaseInit(TIM2, &timer_base);
```

```
    /* встановлення параметрів захоплення сигналу:
     - Channel: 1
     - Count: Up
     - Source: Input
     - Divider: Disable
     - Filter: Disable */
// створення структури для параметрів захоплення сигналу
    TIM_ICInitTypeDef timer_ic;
//заповнення створеної структури значеннями параметрів захоплення сигналу
    timer_ic.TIM_Channel = TIM_Channel_2;
//timer_ic.TIM_ICPolarity = TIM_ICPolarity_BothEdge;
//!!! BothEdge not supported
    timer_ic.TIM_ICPolarity = TIM_ICPolarity_Rising;
    timer_ic.TIM_ICSelection = TIM_ICSelection_DirectTI;
    timer_ic.TIM_ICPrescaler = TIM_ICPSC_DIV1;
    timer_ic.TIM_ICFilter = 0;
//ініціалізація створеної конфігурації
    TIM_ICInit(TIM2, &timer_ic);

    /*дозвіл переривання за переповненням лічильника*/
    TIM_ITConfig(TIM2, TIM_IT_CC2, ENABLE);
    /* дозвіл таймеру TIMER2 */
    TIM_Cmd(TIM2, ENABLE);
    /* дозвіл переривання від TIMER2 */
    NVIC_EnableIRQ(TIM2_IRQn);
}
```

```
//обробник переривання від TIMER2
void TIM2_IRQHandler(void){
   volatile uint16_t PPM;
   if (TIM_GetITStatus(TIM2, TIM_IT_CC2) != RESET)
     {
      /*обнуління прапору запиту на переривання*/
      TIM_ClearITPendingBit(TIM2, TIM_IT_CC2);

      PPMValue_Prev = PPMValue;
      PPMValue = TIM_GetCapture2(TIM2);
      PPM = (PPMValue >= PPMValue_Prev) ? (PPMValue - PPMValue_Prev) :
(UINT16_MAX - PPMValue_Prev + PPMValue);
      if (PPM < 3000) { // Pause
         PPMBuffer[PPMi] = PPM;
         PPMi++;
         if (PPMi > 7) {
            PPMi = 0;
         }
      }
      else {
         PPMi = 0;
      }
      /* over-capture */
      if (TIM_GetFlagStatus(TIM2, TIM_FLAG_CC2OF) != RESET)
      {
       TIM_ClearFlag(TIM2, TIM_FLAG_CC2OF);
       // ...
      }
     }
}
```

```
int main(void)
{
  SetSysClockTo72();
  ppm_init();

  while(1)
  {
    // Тут можна прочитати дані PPM з масиву PPMBuffer
    // Наприклад: PPMBuffer[n]; n – номер каналу 0..7
  }
}
```

Крім розглянутих вище функцій таймери мікроконтролера STM32 також можуть бути використані для формування вихідних сигналів, наприклад ШІМ сигналів. Ініціалізація ШІМ (PWM) виконується наступним чином: налаштовується вихід порту відповідного каналу таймера, який буде задіяний для формування PWM сигналу, виконується базове налаштування таймера, виконується налаштування каналу таймера (налаштування параметрів PWM), вмикається таймер. Налаштування параметрів PWM виконується через структуру TIM_OCInitTypeDef:

```
typedef struct
{
 uint16_t TIM_OCMode;
 uint16_t TIM_OutputState;
 uint16_t TIM_OutputNState;
 uint16_t TIM_Pulse;
 uint16_t TIM_OCPolarity;
 uint16_t TIM_OCNPolarity;
 uint16_t TIM_OCIdleState;
 uint16_t TIM_OCNIdleState;
} TIM_OCInitTypeDef;
```

Важливими для поставленої задачі є такі параметри:

– TIM_OCMode – режим виходу (TIM_OCMode_Timing | TIM_OCMode_Active | TIM_OCMode_Inactive | TIM_OCMode_Toggle | TIM_OCMode_PWM1 | TIM_OCMode_PWM2). Потрібні TIM_OCMode_PWM1 або TIM_OCMode_PWM2, це два режими PWM (TIM_OCMode_PWM2 – з вирівнюванням по центру);

– TIM_OutputState – стан виходу (TIM_OutputState_Disable | TIM_OutputState_Enable);

– TIM_Pulse – скважність (відношення періоду слідування імпульсів до тривалості імпульсу) ШІМ сигналу. У МК STM32 лежить в межах від 0x0000 до 0xFFFF;

– TIM_OCPolarity – (TIM_OCPolarity_High | TIM_OCPolarity_Low) – TIM_OCPolarity_High – прямий ШІМ, TIM_OCPolarity_Low – інвертований.

Інші параметри призначені для advanced таймерів. З їх призначенням можна ознайомитись в документації на SPL.

Розглянемо простий приклад, в якому скважність ШІМ сигналу змінюється кнопками (наприклад, для зміни інтенсивності світла від світлодіоду). При натисканні кнопки пов’язаної з PA0 довжина імпульсу буде збільшуватись, при натисканні кнопки пов’язаної з PA1 довжина імпульсу буде зменшуватись. ШІМ сигнал буде формуватись на піні PB6 таймером TIMER4. Частота ШІМ розраховується, як частота тактування таймеру, поділена на дільник таймеру. Наприклад, якщо таймер тактується частотою 8 МГц, а дільник TIM_Prescaler = 800, частота ШІМ буде 8000000/800 = 10 кГц.

Приклад 3.12:

```
#include "stm32f10x.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_gpio.h"
```

```
#include "stm32f10x_tim.h"

#define PERIOD 1000

int main(void)
{
   int TIM_Pulse = 0;
   int i;
/*створення структури для параметрів ініціалізації пінів
   GPIO_InitTypeDef port;
/*створення структури для параметрів ініціалізації таймеру
   TIM_TimeBaseInitTypeDef timer;
/*створення структури для параметрів ШІМ сигналу
   TIM_OCInitTypeDef timerPWM;
    /*дозвіл тактування порту A */
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
   /*дозвіл тактування порту B */
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
   /*дозвіл тактування таймеру TIMER4, канал 1 */
   RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);
 //заповнення створеної структури для ініціалізації порту A
   GPIO_StructInit(&port);
   port.GPIO_Mode = GPIO_Mode_IPU;
   port.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1;
   port.GPIO_Speed = GPIO_Speed_2MHz;
//ініціалізація створеної конфігурації
   GPIO_Init(GPIOA, &port);
//заповнення створеної структури для ініціалізації порту B
   GPIO_StructInit(&port);
   port.GPIO_Mode = GPIO_Mode_AF_PP;
```

```
    port.GPIO_Pin = GPIO_Pin_6;
    port.GPIO_Speed = GPIO_Speed_2MHz;
//ініціалізація створеної конфігурації
    GPIO_Init(GPIOB, &port);
/*заповнення параметрами структури ініціалізації таймеру TIMER4
    TIM_TimeBaseStructInit(&timer);
    timer.TIM_Prescaler = 720;
    timer.TIM_Period = PERIOD;
    timer.TIM_ClockDivision = 0;
    timer.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInit(TIM4, &timer);
/*заповнення структури параметрів ШІМ сигналу
    TIM_OCStructInit(&timerPWM);
    timerPWM.TIM_OCMode = TIM_OCMode_PWM1;
    timerPWM.TIM_OutputState = TIM_OutputState_Enable;
    timerPWM.TIM_Pulse = 10;
    timerPWM.TIM_OCPolarity = TIM_OCPolarity_High;
//ініціалізація створеної конфігурації
    TIM_OC1Init(TIM4, &timerPWM);
//запуск таймера
    TIM_Cmd(TIM4, ENABLE);
    while(1)
    {
/*Аналіз стану кнопок. Якщо натиснуто кнопку пов’язану з PA0 збільшувати
довжину імпульсу, якщо PA1 зменшувати довжину імпульсу */
        if (GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0) == 0) {
            if (TIM_Pulse < PERIOD) TIM_Pulse++; TIM4->CCR1 = TIM_Pulse;
        }
        if (GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_1) == 0) {
            if (TIM_Pulse > 0)
```

```
            TIM_Pulse--;
          TIM4->CCR1 = TIM_Pulse;
      }
      /*затримка */
      for(i=0;i<0x10000;i++);
   }
}
```

3.11. Годинник реального часу мікроконтролерів STM32

До складу STM32 входить вбудований годинник реального часу. Він може працювати незалежно від основного живлення мікроконтролера. Для роботи вбудованого годинника на спеціальний вивід треба подати живлення напругою 3 В. Наприклад, підключити батарейку. Годинник споживає дуже мало енергії, тому батарейки вистачає на тривалий час. Такий годинник може працювати, наприклад, як будильник – виводити мікроконтролер з режиму енергозбереження.

Основою годинника реального часу у STM32 є так званий домен резервної батареї – Battery backup domain (рис. 3.20).

Це частина мікроконтролера, яка може живитися від батарейки. До неї відносяться сам годинник з регістрами налаштування, LSE генератор, тобто схема обслуговування зовнішнього низькочастотного (32768 Гц) кварцу, виводи PC13...PC15 (у МК STM32F103C8). Крім того, ця зона в STM32F103C8 містить 42 16-бітових регістра, які зберігають свої значення, поки на вхід, до якого підключається батарейка, подається напруга. Їх можна використовувати для своїх цілей.

Якщо треба запустити годинник реального часу (RTC), то робити це потрібно тільки за умови, що він ще не ініціалізований. Проводити налаштування, коли годинник вже запущений не можна. Після того, як RTC налаштований, його параметри конфігурації зберігаються в регістрах до тих

пір, поки підключена батарейка. Розглянемо приклад програми, яка ініціалізує RTC і встановлює початковий момент тільки під час першої ініціалізації. Дані про дату та час виводяться в порт USART.

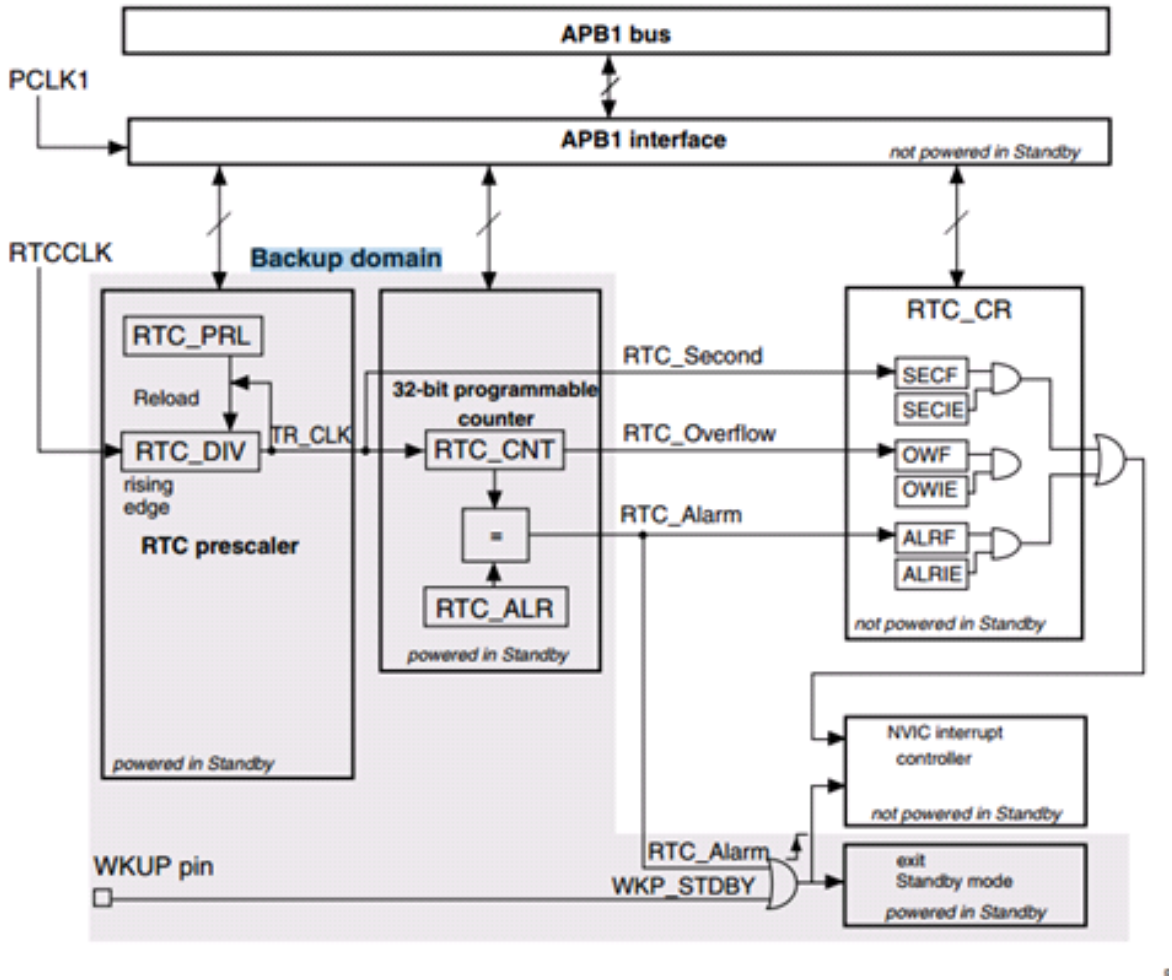


Рис. 3.20. Спрощена схема домену резервної батареї МК STM32F103C8

Приклад 3.13:

```
#include "stm32f10x.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rtc.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_pwr.h"
#include "stm32f10x_usart.h"
#include "stdio.h"
```

```
#include "misc.h"

#include "tim2_delay.h"

void SetSysClockToHSE(void)
{
  ErrorStatus HSEStartUpStatus;

  /* SYSCLK, HCLK, PCLK2 and PCLK1 configuration ----------------------------
*/
  /* RCC system reset() // використовують лише для налагодження*/
  RCC_DeInit();
  /* дозвіл на тактування від HSE */
  RCC_HSEConfig( RCC_HSE_ON);
  /*чекаємо готовності HSE*/
  HSEStartUpStatus = RCC_WaitForHSEStartUp();
  if (HSEStartUpStatus == SUCCESS)
  {
    /*Дозволяємо буфер*/
    //FLASH_PrefetchBufferCmd( FLASH_PrefetchBuffer_Enable);

    /* Flash 0 wait state */
    //FLASH_SetLatency( FLASH_Latency_0);
    /* HCLK = SYSCLK */
    RCC_HCLKConfig( RCC_SYSCLK_Div1);
    /* PCLK2 = HCLK */
    RCC_PCLK2Config( RCC_HCLK_Div1);
    /* PCLK1 = HCLK */
    RCC_PCLK1Config(RCC_HCLK_Div1);
    /* встановлюємо HSE як джерело системного тактування */
```

```
        RCC_SYSCLKConfig( RCC_SYSCLKSource_HSE);

        /* чекаємо на підключення PLL*/
        while (RCC_GetSYSCLKSource() != 0x04)
        {
        }
    }
    else
    { /* Тут можна прописати команди на випадок, коли сталася помилка підключення HSE*/
    }
}

//ініціалізація usart
void usart_init(void)
{
        /* дозвіл тактування USART1 та порту A*/
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1 |
                               RCC_APB2Periph_GPIOA, ENABLE);
        /*конфігурування порту A */
//створення структури для параметрів конфігурації порту
        GPIO_InitTypeDef GPIO_InitStructure;
        /*конфігурування PA.09 як USART1 Tx */
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
//ініціалізація (запуск) створеної конфігурації
        GPIO_Init(GPIOA, &GPIO_InitStructure);
        /* конфігурування PA.10 як USART1 Rx*/
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
```

```
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
//ініціалізація (запуск) створеної конфігурації
    GPIO_Init(GPIOA, &GPIO_InitStructure);
     /*конфігурування USART1 */
//створення структури для параметрів конфігурації USART
    USART_InitTypeDef USART_InitStructure;

   /*конфігурація  USART1 ------------------------------------------*/
    /* USART1 configured as follow:
        - BaudRate = 115200 baud
        - Word Length = 8 Bits
        - One Stop Bit
        - No parity
        - Hardware flow control disabled (RTS and CTS signals)
        - Receive and transmit enabled
        - USART Clock disabled
        - USART CPOL: Clock is active low
        - USART CPHA: Data is captured on the middle
        - USART LastBit: The clock pulse of the last data bit is not output to
                the SCLK pin
    */
    USART_InitStructure.USART_BaudRate = 115200;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl =
                                        USART_HardwareFlowControl_None;
USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
//ініціалізація (запуск) створеної конфігурації
    USART_Init(USART1, &USART_InitStructure);
```

```
    /*дозвіл USART1 */
    USART_Cmd(USART1, ENABLE);
}
//функція відправлення даних через USART1
void USARTSend(char *pucBuffer)
{
  while (*pucBuffer)
  {
    USART_SendData(USART1, *pucBuffer++);
    while(USART_GetFlagStatus(USART1, USART_FLAG_TC) == RESET)
    {
    }
  }
}

//функція ініціалізації RTC потрібними параметрами
unsigned char RTC_Init(void)
{
  /*увімкнути тактування модулів керування живленням і керуванням резервною областю*/
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_PWR | RCC_APB1Periph_BKP, ENABLE);
  // Дозволити доступ до області резервних даних
  PWR_BackupAccessCmd(ENABLE);
  // якщо RTC вимкнений, ініціалізувати
  if ((RCC->BDCR & RCC_BDCR_RTCEN) != RCC_BDCR_RTCEN)
  {
    // скидання даних в резервній області
    RCC_BackupResetCmd(ENABLE);
    RCC_BackupResetCmd(DISABLE);
```

```
        // встановити як джерело тактування кварц 32768
        RCC_LSEConfig(RCC_LSE_ON);
//чекаємо виконання попередньої команди
        while ((RCC->BDCR & RCC_BDCR_LSERDY) != RCC_BDCR_LSERDY)
{}
        RCC_RTCCLKConfig(RCC_RTCCLKSource_LSE);
//      встановлюємо дільник, щоб годинник відраховував секунди
RTC_SetPrescaler(0x7FFF);
        // вмикаємо RTC
        RCC_RTCCLKCmd(ENABLE);
        //чекаємо на синхронізацію
        RTC_WaitForSynchro();
        return 1;
    }
   return 0;
}

/*
Як конвертувати дату можно прочитати тут [58].
*/

// (UnixTime = 00:00:00 01.01.1970 = JD0 = 2440588)
#define JULIAN_DATE_BASE    2440588

typedef struct
{
   uint8_t RTC_Hours;
   uint8_t RTC_Minutes;
   uint8_t RTC_Seconds;
```

```
    uint8_t RTC_Date;
    uint8_t RTC_Wday;
    uint8_t RTC_Month;
    uint16_t RTC_Year;
} RTC_DateTimeTypeDef;

// Get current date
void RTC_GetDateTime(uint32_t RTC_Counter, RTC_DateTimeTypeDef*
RTC_DateTimeStruct) {
    unsigned long time;
    unsigned long t1, a, b, c, d, e, m;
    int year = 0;
    int mon = 0;
    int wday = 0;
    int mday = 0;
    int hour = 0;
    int min = 0;
    int sec = 0;
    uint64_t jd = 0;;
    uint64_t jdn = 0;

    jd = ((RTC_Counter+43200)/(86400>>1)) + (2440587<<1) + 1;
    jdn = jd>>1;

    time = RTC_Counter;
    t1 = time/60;
    sec = time - t1*60;

    time = t1;
    t1 = time/60;
```

```
    min = time - t1*60;

    time = t1;
    t1 = time/24;
    hour = time - t1*24;

    wday = jdn%7;

    a = jdn + 32044;
    b = (4*a+3)/146097;
    c = a - (146097*b)/4;
    d = (4*c+3)/1461;
    e = c - (1461*d)/4;
    m = (5*e+2)/153;
    mday = e - (153*m+2)/5 + 1;
    mon = m + 3 - 12*(m/10);
    year = 100*b + d - 4800 + (m/10);

    RTC_DateTimeStruct->RTC_Year = year;
    RTC_DateTimeStruct->RTC_Month = mon;
    RTC_DateTimeStruct->RTC_Date = mday;
    RTC_DateTimeStruct->RTC_Hours = hour;
    RTC_DateTimeStruct->RTC_Minutes = min;
    RTC_DateTimeStruct->RTC_Seconds = sec;
    RTC_DateTimeStruct->RTC_Wday = wday;
}

// Convert Date to Counter
uint32_t RTC_GetRTC_Counter(RTC_DateTimeTypeDef* RTC_DateTimeStruct)
{
```

```
    uint8_t a;
    uint16_t y;
    uint8_t m;
    uint32_t JDN;

    a=(14-RTC_DateTimeStruct->RTC_Month)/12;
    y=RTC_DateTimeStruct->RTC_Year+4800-a;
    m=RTC_DateTimeStruct->RTC_Month+(12*a)-3;

    JDN=RTC_DateTimeStruct->RTC_Date;
    JDN+=(153*m+2)/5;
    JDN+=365*y;
    JDN+=y/4;
    JDN+=-y/100;
    JDN+=y/400;
    JDN = JDN -32045;
    JDN = JDN - JULIAN_DATE_BASE;
    JDN*=86400;
    JDN+=(RTC_DateTimeStruct->RTC_Hours*3600);
    JDN+=(RTC_DateTimeStruct->RTC_Minutes*60);
    JDN+=(RTC_DateTimeStruct->RTC_Seconds);
    return JDN;
}

void RTC_GetMyFormat(RTC_DateTimeTypeDef* RTC_DateTimeStruct, char *
buffer) {
    const char WDAY0[] = "Monday";
    const char WDAY1[] = "Tuesday";
    const char WDAY2[] = "Wednesday";
    const char WDAY3[] = "Thursday";
```

```
   const char WDAY4[] = "Friday";
   const char WDAY5[] = "Saturday";
   const char WDAY6[] = "Sunday";
   const char * WDAY[7]={WDAY0, WDAY1, WDAY2, WDAY3, WDAY4,
WDAY5, WDAY6};

   const char MONTH1[] = "January";
   const char MONTH2[] = "February";
   const char MONTH3[] = "March";
   const char MONTH4[] = "April";
   const char MONTH5[] = "May";
   const char MONTH6[] = "June";
   const char MONTH7[] = "July";
   const char MONTH8[] = "August";
   const char MONTH9[] = "September";
   const char MONTH10[] = "October";
   const char MONTH11[] = "November";
   const char MONTH12[] = "December";
   const char * MONTH[12]={MONTH1, MONTH2, MONTH3, MONTH4,
MONTH5, MONTH6, MONTH7, MONTH8, MONTH9, MONTH10,
MONTH11, MONTH12};

   sprintf(buffer, "%s %d %s %04d",
         WDAY[RTC_DateTimeStruct->RTC_Wday],
         RTC_DateTimeStruct->RTC_Date,
         MONTH[RTC_DateTimeStruct->RTC_Month -1],
         RTC_DateTimeStruct->RTC_Year);
}
//головна функція
int main(void)
```

```
{
   char buffer[80] = {'\0'};
   uint32_t RTC_Counter = 0;
   RTC_DateTimeTypeDef RTC_DateTime;
    SetSysClockToHSE();
   TIM2_init();
    usart_init();
    if (RTC_Init() == 1) {
      /*якщо перша ініціалізація RTC встановлюємо початкову дату, наприклад  22.09.2016 14:30:00*/
      RTC_DateTime.RTC_Date = 22;
      RTC_DateTime.RTC_Month = 9;
      RTC_DateTime.RTC_Year = 2016;

      RTC_DateTime.RTC_Hours = 14;
      RTC_DateTime.RTC_Minutes = 30;
      RTC_DateTime.RTC_Seconds = 00;

      /*після ініціалізування необхідна затримка. Без неї час не встановлюється*/
      delay_ms(500);
      RTC_SetCounter(RTC_GetRTC_Counter(&RTC_DateTime));
   }

   while(1)
   {
      RTC_Counter = RTC_GetCounter();
      sprintf(buffer, "\r\n\r\nCOUNTER: %d\r\n", (int)RTC_Counter);
      USARTSend(buffer);
```

```
        RTC_GetDateTime(RTC_Counter, &RTC_DateTime);
        sprintf(buffer, "%02d.%02d.%04d  %02d:%02d:%02d\r\n",
                RTC_DateTime.RTC_Date, RTC_DateTime.RTC_Month,
RTC_DateTime.RTC_Year,
                RTC_DateTime.RTC_Hours, RTC_DateTime.RTC_Minutes,
RTC_DateTime.RTC_Seconds);
        USARTSend(buffer);
        // функція генерує у буфері дату власного формату
        RTC_GetMyFormat(&RTC_DateTime, buffer);
        USARTSend(buffer);
        /*затримка*/
        while (RTC_Counter == RTC_GetCounter()) {
        }
    }
}
```

Після першого вмикання мікроконтролер запустить функцію ініціалізації та запустить RTC. Під час наступних запусків, функція буде вмикати тільки тактування потрібних модулів і надавати доступ до області резервної копії PWR_BackupAccessCmd (ENABLE). Це необхідно, щоб отримати доступ до лічильника RTC. Перша ініціалізація RTC – достатньо тривалий процес, який може тривати близько секунди. Надалі мікроконтролер стартує без затримок. Якщо вимкнути живлення мікроконтролера і живлення від батарейки, конфігурація RTC буде скинута.

3.12. Резервні регістри мікроконтролерів STM32

У попередньому розділі згадувався домен резервного копіювання (Backup Domain) – частина мікроконтролера, яка живиться від додаткової батареї. Домен резервного копіювання містить резервні регістри (Backup

registers). Це 16-бітові регістри пам'яті, які зберігають своє значення після відключення основного живлення мікроконтролера. Якщо вимкнути живлення мікроконтролера і резервне живлення (батарейку), дані регістрів будуть втрачені. Тому ці регістри не можна вважати енергонезалежною пам'яттю. Проте їх можна використовувати для зберігання даних. У наступному прикладі при кожному старті програми (або при перезавантаженні за сигналом Reset) зчитується регістр BKP_DR1, збільшується на одиницю, записується, а його поточне значення виводиться в порт USART. Таким чином реалізований лічильник кількості запусків програми мікроконтролера. Щоб скинути лічильник, достатньо відключити батарейку.

Приклад 3.13:

```
#include "stm32f10x.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_pwr.h"
#include "stm32f10x_usart.h"
#include "stm32f10x_bkp.h"
#include "stdio.h"
#include "misc.h"
//функція ініціалізації usart
void usart_init (void)
{
  / * дозвіл тактування USART1 та порту A * /
  RCC_APB2PeriphClockCmd (RCC_APB2Periph_USART1 | RCC_APB2Perip
                                    h_GPIOA, ENABLE);
  / * створення структури для конфігурування порту A* /
  GPIO_InitTypeDef GPIO_InitStructure;
```

```
 / *конфігуруємо пін PA.09 як USART1 Tx*/
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
//ініціалізуємо створену конфігурацію
GPIO_Init (GPIOA, & GPIO_InitStructure);
/ * конфігуруємо пін PA.10 як USART1 Rx * /
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
//ініціалізуємо створену конфігурацію
GPIO_Init (GPIOA, & GPIO_InitStructure);
/ * конфігуруємо USART1 * /
/ * створення структури для конфігурування USART1* /
USART_InitTypeDef USART_InitStructure;
/ *конфігурування  USART1 ------------------------------------- -------- * /
/ * USART1 configured as follow:
        - BaudRate = 115200 baud
        - Word Length = 8 Bits
        - One Stop Bit
        - No parity
        - Hardware flow control disabled (RTS and CTS signals)
        - Receive and transmit enabled
        - USART Clock disabled
        - USART CPOL: Clock is active low
        - USART CPHA: Data is captured on the middle
        - USART LastBit: The clock pulse of the last data bit is not output to
                the SCLK pin
 * /
USART_InitStructure.USART_BaudRate = 115200;
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
```

```
   USART_InitStructure.USART_StopBits = USART_StopBits_1;
   USART_InitStructure.USART_Parity = USART_Parity_No;
   USART_InitStructure.USART_HardwareFlowControl =  USART_HardwareFlo
                                                wControl_None;
   USART_InitStructure.USART_Mode = USART_Mode_Rx |  USART_Mode_T
                                                        x;
//ініціалізуємо створену конфігурацію
   USART_Init (USART1, & USART_InitStructure);
   / *дозволяємо використання USART1 * /
   USART_Cmd (USART1, ENABLE);
}
//функція відправки даних через USART1
void USARTSend (char * pucBuffer)
{
   while (* pucBuffer)
   {
      USART_SendData (USART1, * pucBuffer ++);
      while (USART_GetFlagStatus (USART1, USART_FLAG_TC) == RESET)
      {
      }
   }
}
int main (void)
{
   uint16_t reload_counter = 0;
   char buffer [80] = { '\ 0'};
//ініціалізація USART1
   usart_init ();
   /*Включити тактування модулів керування живленням і керуванням
резервною областю*/
```

```
  RCC_APB1PeriphClockCmd (RCC_APB1Periph_PWR |
RCC_APB1Periph_BKP, ENABLE);
  // дозволити доступ до області резервної копії
  PWR_BackupAccessCmd (ENABLE);
  // Читаємо регістр BKP_DR1. Всього їх у STM32F103C8 42
  reload_counter = BKP_ReadBackupRegister (BKP_DR1);
  // збільшуємо на 1
  reload_counter ++;
  // Записуємо в той же регістр
  BKP_WriteBackupRegister (BKP_DR1, reload_counter);
  // Виводимо в USART поточне значення регістра
  sprintf (buffer, "BKP_DR1:% d \ r \ n", reload_counter);
  USARTSend (buffer);
  while (1)
  {  }
}
```

При використанні резервних регістрів необхідно пам’ятати, що це – 16-бітні регістри і, що в разі відключення основного і резервного живлення, їх вміст обнуляється.

3.13. Перерозподіл альтернативних функцій виводів у мікроконтролерах сімейства STM32

Із певними виводами мікроконтролера пов’язаний певний функціонал. Функцію конкретного виводу можна обирати тільки з переліку доступних для нього функцій. Наприклад, у мікроконтролері STM32F103 пін PA9 можна використовувати як лінію порту A9 або використовувати альтернативний функціонал, а саме як TX вихід послідовного порту USART1 (USART1_TX), або як другий канал першого таймера TIM1_CH2. На

випадок, коли необхідний USART1_TX, а A9 вже використовується, в мікроконтролері сімейства STM32 є можливість перерозподілу (remapping) альтернативного функціоналу за допомогою функції GPIO_PinRemapConfig.

Перерозподіляти можна не всі, а тільки деякі з альтернативних функцій. Інформацію про можливість перерозподілу альтернативних функцій можна знайти в документації на конкретну серію МК. У таблиці 3.7, для прикладу, наведено інформацію про основні та альтернативні функції виводів для мікроконтролерів STM32103x8 та STM32103xB.

Описані альтернативні функції для виводів мікроконтролера розбиті на дві групи: Default і Remap. Default – альтернативний функціонал виведення за замовчуванням. Remap – альтернативний функціонал виведення після виконання перерозподілу.

Виконати перерозподіл альтернативних функцій достатньо просто. Для цього необхідно включити тактуваняя AFIO (альтернативні функції введення-виведення) і визвати функцію GPIO_PinRemapConfig з потрібними опціями. Можливі варіанти опцій містяться у файлах опису мікроконтролера згенерованих за допомогою IDE. У нашому випадку це файл stm32f10x_gpio.c. Розглянемо приклад.

Приклад 3.14:

```
// Вмикаємо тактування AFIO (альтернативні функції введення-виведення)
RCC_APB2PeriphClockCmd (RCC_APB2Periph_AFIO, ENABLE);
// перерозподіляємо альтернативні функції для USART1
GPIO_PinRemapConfig (GPIO_Remap_USART1, ENABLE);
```

Після виконання даного коду для мікроконтролерів STM32103x8 та STM32103xB USART1_TX буде на виводі PB6, а USART1_RX на виводі PB7.

Таблиця 3.7.

Розподіл функцій у МК STM32103x8 та STM32103xB

Pins							Pin name	Type[1]	I / O Level[2]	Main function[3] (after reset)	Alternate functions[4]	
LFBGA100	UFBG100	LQFP48/UFQFPN48	TFBGA64	LQFP64	LQFP100	VFQFPN36					Default	Remap
H9	J12	-	-	-	57	-	PD10	I/O	FT	PD10	-	USART3_CK
G9	J11	-	-	-	58	-	PD11	I/O	FT	PD11	-	USART3_CTS
K10	J10	-	-	-	59	-	PD12	I/O	FT	PD12	-	TIM4_CH1 / USART3_RTS
J10	H12	-	-	-	60	-	PD13	I/O	FT	PD13	-	TIM4_CH2
H10	H11	-	-	-	61	-	PD14	I/O	FT	PD14	-	TIM4_CH3
G10	H10	-	-	-	62	-	PD15	I/O	FT	PD15	-	TIM4_CH4
F10	E12	-	F6	37	63	-	PC6	I/O	FT	PC6	-	TIM3_CH1
E10	E11		E7	38	64	-	PC7	I/O	FT	PC7	-	TIM3_CH2
F9	E10		E8	39	65	-	PC8	I/O	FT	PC8	-	TIM3_CH3
E9	D12	-	D8	40	66	-	PC9	I/O	FT	PC9	-	TIM3_CH4
D9	D11	29	D7	41	67	20	PA8	I/O	FT	PA8	USART1_CK/ TIM1_CH1[9]/ MCO	-
C9	D10	30	C7	42	68	21	PA9	I/O	FT	PA9	USART1_TX[9]/ TIM1_CH2[9]	-
D10	C12	31	C6	43	69	22	PA10	I/O	FT	PA10	USART1_RX[9]/ TIM1_CH3[9]	-

...

A6	A7	40	A4	56	90	31	PB4	I/O	FT	JNTRST	-	TIM3_CH1/ PB4/ SPI1_MISO
C5	C5	41	C4	57	91	32	PB5	I/O		PB5	I2C1_SMBAI	TIM3_CH2 / SPI1_MOSI
B5	B5	42	D3	58	92	33	PB6	I/O	FT	PB6	I2C1_SCL[9]/ TIM4_CH1[9]	USART1_TX
A5	B4	43	C3	59	93	34	PB7	I/O	FT	PB7	I2C1_SDA[9]/ TIM4_CH2[9]	USART1_RX
D5	A4	44	B4	60	94	35	BOOT0	I		BOOT0	-	-

Контрольні запитання

1. Перелічіть основні профілі сімейства ARM Cortex.
2. Опишіть склад ядра Cortex-M3.
3. Назвіть можливості ядра Cortex-M3.
4. Що таке контролер векторизованих вкладених переривань?
5. Дайте визначення поняття «вектор переривання».
6. Перелічіть блоки, які відповідають за безпеку роботи мікроконтролера STM32.
7. Опишіть групи мікроконтролерів сімейства STM32.
8. Назвіть основні засоби розробки програмного забезпечення для STM32.
9. Опишіть налагоджувальні плати на основі STM32.
10. Перелічіть режими роботи, які підтримує ЦПП Cortex.
11. Назвіть режими адресації, які підтримує ЦПП Cortex.
12. Опишіть архітектуру, за якою побудований процесор Cortex-M3.
13. Назвіть особливості контролера векторизованих вкладених переривань ядра Cortex-M3.
14. Опишіть алгоритм роботи контролера векторизованих вкладених переривань ядра Cortex-M3.
15. Опишіть методи обробки запитів на переривання, які підтримує контролер векторизованих вкладених переривань ядра Cortex-M3.
16. Перелічіть дії, які необхідно виконати, щоб включити в роботу контролер векторизованих вкладених переривань ядра Cortex-M3.
17. Опишіть розподіл полів пріоритету переривань ядра Cortex-M3на групи і підгрупи.
18. Порівняйте джерела тактування, які мають мікроконтролери сімейства STM32.
19. Назвіть функції модуля PLL у мікроконтролерах сімейства STM32.
20. Яким чином можна забезпечити тактування зовнішніх пристроїв при роботі з мікроконтролерами сімейства STM32?

21. Опишіть порти введення-виведення мікроконтролерів сімейства STM32.
22. Наведіть характеристики АЦП, які використовують у мікроконтролерах сімейства STM32.
23. Намалюйте рекомендовану схему підключення опорної напруги і напруги живлення внутрішніх АЦП мікроконтролерів сімейства STM32.
24. Намалюйте спрощену схему підключення опорної напруги і напруги живлення внутрішніх АЦП мікроконтролерів сімейства STM32.
25. Опишіть режими, у яких можуть працювати внутрішні АЦП мікроконтролерів сімейства STM32.
26. Наведіть відмінності регулярних та інжектованих каналів внутрішніх АЦП мікроконтролерів сімейства STM32.
27. Для чого використовують контролер DMA?
28. Назвіть переваги і недоліки використання контролера DMA.
29. Перелічіть типи таймерів, які підтримують мікроконтролери сімейства STM32. Чим вони відрізняються?
30. Як можна вимірювати довжину імпульсу за допомогою таймера?
31. Як за допомогою таймера реалізувати функцію захоплення вхідного сигналу?
32. Як за допомогою таймера сформувати ШІМ сигнал?
33. Назвіть функції годинника реального часу.
34. Опишіть реалізацію годинника реального часу у мікроконтролерах сімейства STM32.
35. Для чого використовують резервні регістри мікроконтролерів сімейства STM32?
36. Що таке альтернативний функціонал?
37. Для чого використовують альтернативний функціонал?

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Микропроцессорные средства и системы: курс лекций / Д. Н. Беклемишев, А. Н. Орлов, А. Л. Переверзев [и др.] ; под ред. Ю. В. Савченко. – М. : МИЭТ, 2013. – 288 с. ISBN 978-5-7256-0723-9
2. http://radiostorage.net/1022-klassifikaciya-mikrokontrollerov.html
3. http://csaa.ru/struktura-mikroprocessornoj-sistemy-i-osnovnye/
4. Колонтаєвський Ю. П. Конспект лекцій з дисципліни «Мікропроцесорна техніка» (для студентів, які навчаються за напрямом 6.050701 – Електротехніка та електротехнології всіх форм навчання) / Ю. П. Колонтаєвський ; Харків. нац. ун-т міськ. госп-ва ім. О. М. Бекетова. – Харків : ХНУМГ ім. О. М. Бекетова, 2016. – 78 с.
5. https://ru.wikipedia.org/wiki/SRAM_(%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C)
6. https://ru.wikipedia.org/wiki/EPROM
7. https://ru.wikipedia.org/wiki/Арифметико-логическое_устройство
8. https://ru.wikipedia.org/wiki/Serial_Peripheral_Interface
9. http://www.semiconductors.philips.com/i2c
10. The RS232 Standard. CAMI Research Inc.
11. Гук М. Интерфейсы ПК : справочник / М. Гук. – СПб : Питер Ком, 1999. – 416 с.
12. http://www.gaw.ru/html.cgi/txt/interface/rs232/start.htm
13. https://ipc2u.ru/articles/prostye-resheniya/otlichiya-interfeysov-rs-232-rs-422-rs-485/#rs-232-opisanie
14. RS-422 and RS-485 Application Note B&B Electronics Mfg. Co. Inc. P.O. Box 1040 -- Ottawa, IL 61350 PH (815) 433-5100 -- FAX (815) 434-7094 dresses: B&B Home Page: http://www.bb-elec.com
15. Мікропроцесорна техніка [Текст] : навч. посібник / В. В. Ткачов, Г. Грулер, Н. Нойбергер [та ін.]. – Д. : Національний гірничий університет, 2012. – 188 с. ISBN 978-966-350-359-2

16. https://uk.wikipedia.org/wiki/USB
17. http://njnmnp.narod.ru/note/USB_for_programmer.pdf
18. https://microtechnics.ru/osnovy-interfejsa-usb/
19. https://ru.wikipedia.org/wiki/USB
20. http://www.russianelectronics.ru/leader-r/news/company/2114/doc/26473/
21. https://ru.wikipedia.org/wiki/USB_Type-C
22. https://uk.wikipedia.org/wiki/USB-C
23. https://ru.wikipedia.org/wiki/WiMAX
24. http://wi-life.ru/texnologii/wi-fi/wi-fi-standarty
25. Лукин М. Стандарты беспроводной связи / Лукин М. // Современная электроника. – 2005. – № 1. – С. 10–12. (https://www.soel.ru/upload/clouds/1/iblock/15d/15d3b56c2b4ab2448ae964faf6421035/200501010.pdf)
26. https://ru.wikipedia.org/wiki/Ethernet
27. http://www.selteq.com/tablename/sq_product_item/id/131
28. https://uk.wikipedia.org/wiki/ВЧ_з%27єднувач
29. https://i.ebayimg.com/images/g/N~wAAOSwXEdaiJ2l/s-l300.jpg
30. https://prom.ua/p645169517-shteker-bnc-terminator.html
31. https://ru.wikipedia.org/wiki/Сетевая_топология
32. https://uk.wikipedia.org/wiki/Звита_пара
33. Предко М. Руководство по микроконтроллерам. Том 1. / М. Предко. – Москва : Постмаркет, 2001. – 416 с.
34. https://uk.wikipedia.org/wiki/Широтно-імпульсна_модуляція
35. https://uk.wikipedia.org/wiki/Сторожовий_таймер
36. http://blog.myelectronics.com.ua/wp-content/uploads/2016/06/WDT-STM32.jpg
37. https://novainfo.ru/article/10802
38. http://microchipinf.com/articles/41/61
39. https:// https://uk.wikipedia.org/wiki/Контролер_переривань
40. https://uk.wikipedia.org/wiki/Аналого-цифровий_перетворювач

41. http://mymcu.ru/articles/kak-rabotayut-analogo-tsifrovie-preobrazovateli-chto-mozhno-uznat-iz-spetsifikatsii-na-atsp.html
42. https://uk.wikipedia.org/wiki/ЦАП
43. Ознакомительное руководство по ARM-микроконтроллерам Cortex-M3
44. https://www.compel.ru/lib/ne/2011/2/4-mikrokontrolleryi-stm32-s-nulya
45. http://cxem.net/mc/mc194.php
46. http://www.avislab.com/blog/stm32-nvic_ru/
47. http://mypractic.ru/hal-gpio-generic-driver-funkcii-upravleniya-portami-vvoda-vyvoda#1
48. http://mypractic.ru/hal-tim-generic-driver-funkcii-upravleniya-tajmerami
49. http://www.avislab.com/blog/stm32-clock_ru/
50. http://www.avislab.com/blog/stm32-gpio_ru/
51. http://www.avislab.com/blog/stm32-exti_ru/
52. http://www.avislab.com/blog/stm32-adc_ru/
53. http://www.avislab.com/blog/stm32-tim_basic_ru/
54. http://www.avislab.com/blog/stm32-rtc_ru/
55. http://www.avislab.com/blog/stm32-bkp_ru/
56. http://www.avislab.com/blog/stm32-remap_ru/
57. Погорелый С. Д., Слободанюк Т. Ф. Глава 2. Язык ассемблера для шестнадцатиразрядного микропроцессора К1810ВМ86. Подпрограммы обработки прерываний // Программное обеспечение микропроцессорных систем : Справочник. – К. : Тэхника, 1989. – С. 56. – 301 с. ISBN 5-335-00169-0
58. https://ru.m.wikipedia.org/wiki/Юлианская_дата