В.О. Поджаренко, В.Ю. Кучерук, В.М. Севастьянов

ОСНОВИ МІКРОПРОЦЕСОРНОЇ ТЕХНІКИ

Міністерство освіти і науки України
Вінницький національний технічний університет

В.О. Поджаренко, В.Ю. Кучерук, В.М. Севастьянов

ОСНОВИ МІКРОПРОЦЕСОРНОЇ ТЕХНІКИ

Затверджено Вченою радою Вінницького національного технічного університету як навчальний посібник для студентів напряму підготовки "Метрологія та вимірювальна техніка". Протокол № 11 від "30" червня 2005 р.

Вінниця ВНТУ 2006

УДК 681.3
П 44

Р е ц е н з е н т и :

Л.І. Муравський, доктор технічних наук (ФМІ ім. Г.В. Карпенка)
О.М. Роїк, доктор технічних наук професор (ВНТУ)
М.М. Биков, кандидат технічних наук професор (ВНТУ)

Рекомендовано до видання Вченою радою Вінницького національного технічного університету Міністерства освіти і науки України

Поджаренко В.О., Кучерук В.Ю., Севастьянов В.М.
П44 **Основи мікропроцесорної техніки.** Навчальний посібник. - Вінниця: ВНТУ, 2006. - 226 с.

В посібнику розглянуті фундаментальні основи сучасної мікропроцесорної техніки, наведені приклади практичного застосування. Посібник розроблений у відповідності з планом кафедри та програмами дисциплін "Основи мікропроцесорної техніки", "Основи цифрової техніки".

УДК 681.3

ЗМІСТ

ВСТУП

Навряд чи сьогодні можна уявити діяльність людини без сучасних персональних комп'ютерів, великих ЕОМ і інших засобів обчислювальної техніки. Масовому впровадженню комп'ютерів у діяльність людини сприяли успіхи, досягнуті в напівпровідниковій електроніці. Важко повірити, що перша електронно-лічильна машина (1948р.) була створена до винаходу першої інтегральної мікросхеми. Без прогресу в технології напівпровідникової електроніки, на базі якої створюються інтегральні схеми, не з'явилася б світова інформаційна павутина — Інтернет і глобальні системи мобільного зв'язку, не були б досягнуті успіхи в освоєнні космічного простору.

XX століття можна сміло вважати століттям високих технологій. Найяскравіший приклад високих технологій XX століття — напівпровідникова електроніка, на базі якої й створюються інтегральні схеми. Дуже знаменно, що в останній рік минулого сторіччя Нобелівським лауреатом в області фізики став американський учений Дж. Кілбі — один із творців першої інтегральної мікросхеми (вересень 1958 р.). Необхідно відзначити, що транзистор був винайдений десятьма роками раніше (1947р.), а ідея інтегральної схеми була запропонована американським ученим Д. Даммером у 1954 р. Перша інтегральна схема складалася всього з одного германієвого транзистора, трьох резисторів і конденсатора. Проте це було революційним відкриттям в електроніці. Дж. Кілбі винайшов не просто інтегральну схему — він відкрив дорогу в майбутнє.

Повною мірою оцінити прогрес засобів обчислювальної техніки за останні 50 років можна простим порівнянням технічних характеристик перших ЕОМ із можливостями й характеристиками сучасних мікропроцесорів типу Pentium III чи Pentium IV.

Перша в СРСР електронно-обчислювальна машина була створена в 1951 р. Споживана нею потужність складала 25 кВт, а універсальний арифметико-логічний пристрій виконував усього 50 арифметичних чи логічних операцій у секунду.

Супер-ЕОМ Cray-1 була реалізована на мікросхемах у 1975 р. Тривалість машинного циклу Cray-1 складала 12.5 нс, що забезпечувало продуктивність порядку 100 мільйонів арифметичних операцій у секунду. Cray-1 містила приблизно 300 000 мікросхем, що були розміщені в обсязі 2.8 $м^3$. Концентрація великої кількості мікросхем у малому обсязі було обумовлено мінімізацією довжини з'єднувальних провідників і створювало серйозну проблему відведення тепла, яке виділялося при роботі. У Cray-1 теплова енергія відводилась по каналах охолодження стиснутим фреоном. Вартість Cray-1 складала від 10 до 15 млн. доларів у залежності від обсягу пам'яті й периферійного устаткування. Оскільки в розвитку засобів обчислювальної техніки першочергову роль відіграють досягнення в технології

напівпровідникової електроніки, становить інтерес розглянути особливості сучасної мікроелектронної бази.

Продуктивність процесорів, у першу чергу, залежить від тактової частоти й архітектури процесорів. Під архітектурою процесорів (мікропроцесорів (МП)) тут і далі розуміється структурна організація процесора, що включає процесорне (обчислювальне) ядро, пам'ять, функціональні пристрої, периферійні контролери й зв'язки між ними. Складність і функціональні можливості архітектури в основному визначаються кількістю логічних елементів, інтегрованих на кристалі. Для збільшення тактової частоти і реалізації складних процесорних архітектур необхідно зменшувати розміри окремих транзисторів. Для порівняння, МП 486SX — 1.6 млн., Pentium — 3.4 млн. транзисторів, а його наступні модифікації Pentium Pro — 5.5 млн. транзисторів, Pentium II — 7.5 млн., Pentium III — 8.5 млн., а новий Pentium IV має 42 млн. транзисторів. Сьогодні можна вважати освоєною технологію, що забезпечує на базі фотолітографічного процесу одержання транзистора з розмірами 0.13 мкм. Слід зазначити, що, на думку фахівців, прогнозований мінімальний розмір транзистора складе 0.1 мкм до 2007 р., 70 нм до 2008 р., 50 нм до 2011 р. і 20 нм до 2014 р. Сучасний рівень напівпровідникової технології яскраво ілюструють параметри нових процесорів фірми Intel і Texas Instruments.

У процесорі Pentium III, реалізованому на базі технології 0.18 мкм, досягнута тактова частота процесора 1 ГГц, а тактова частота системної шини складає 133 МГц. У новому процесорі Pentium IV на базі тієї ж технології тактова частота процесора складає 1.4 ГГц. Така висока тактова частота при 64-розрядній шині даних забезпечує швидкість обміну даними з пам'яттю, яка дорівнює 3.2 Гбайт/с. Крім того, на кристалі процесора інтегрована кеш-пам'ять першого рівня обсягом 8 кбайт і другого рівня обсягом 256 кбайт. У Pentium III і Pentium IV, реалізованих на базі технології 0.13 мкм, тактова частота буде складати відповідно 1.26 ГГц і 3 ГГц [4]. Збільшення реальної продуктивності процесора залежить не тільки від підвищення тактової частоти, але і розвитку архітектури.

За оцінкою фахівців фірми Texas Instruments, світового лідера в області сигнальних процесорів, у 2007 р. буде освоєна технологія 0.075 мкм, що дозволить довести кількість транзисторів, інтегрованих на кристалі одного сигнального процесора, до 100 мільйонів. У майбутньому освоєння нових технологій дозволить збільшити тактову частоту сигнальних процесорів до 1.1 ГГц, а кількість транзисторів у складі одного сигнального процесора до 500 мільйонів. До 2010 р. передбачається створення сигнального процесора із продуктивністю 3 трильйони інструкцій у секунду. У сигнальних процесорах другого покоління TMS320C64x, побудованих на базі вдосконаленої VLIW (Very Long Instruction Word) архітектури і технології 0.1 мкм, передбачається збільшити тактову частоту до 1.1 ГГц, що забезпечить продуктивність 8800 MIPS.

1 ЗАГАЛЬНІ ПОНЯТТЯ МІКРОПРОЦЕСОРНОЇ ТЕХНІКИ

1.1 Класифікація процесорних пристроїв

Прийнята така класифікація процесорних пристроїв, реалізованих на одному кристалі:

- мікропроцесори загального призначення для числової обробки (універсальні мікропроцесори);
- мікроконтролери для простих систем управління/контролю;
- сигнальні процесори для цифрової обробки сигналів;
- програмовані логічні інтегральні схеми.

Дана класифікація в міру вдосконалення архітектури процесорного ядра й впровадження нових технологій зазнала істотних змін. З урахуванням взаємного впливу архітектури мікропроцесорів різних типів надалі можливо будуть потрібні нові принципи класифікації мікропроцесорів.

Універсальні мікропроцесори призначені для використання в обчислювальних системах: персональних ЕОМ, робочих станціях, а в останній час і в масово-паралельних супер-ЕОМ. Основною їх характеристикою є наявність сучасних приладів для ефективної реалізації операцій із плаваючою точкою над 64-розрядними операндами.

Цифрові сигнальні процесори розраховані на обробку в реальному часі цифрових потоків, утворених шляхом оцифровування аналогових сигналів. Це зумовлює їх порівняно малу розрядність і переважно цілочисельну обробку. Але сучасні сигнальні процесори здатні проводити обчислення із плаваючою точкою над 32- і 40- розрядними операндами.

Цифрові сигнальні процесори (DSP) — відносно нова категорія процесорів. Призначення DSP полягає в тому, щоб одержувати поточні дані від аналогової системи і формувати відповідний відгук. DSP і їх ALU (Arithmetic Logic Unit — арифметико-логічний пристрій, що є апаратним засобом для виконання обчислень) працюють з дуже високою швидкістю, що дозволяє здійснювати обробку даних у реальному масштабі часу. DSP часто використовуються в активних шумо-заглушувальних мікрофонах, що встановлюються в літаках (другий мікрофон забезпечує сигнал навколишнього шуму, що віднімається із сигналу першого мікрофона, дозволяючи в такий спосіб заглушити шум і залишити тільки голос) чи для подавлення роздвоєння зображення в телевізійних сигналах.

Розробка DSP алгоритмів — це спеціальний розділ теорії управління. Викладання цієї теорії вимагає використання багатьох знань з математики і виходить за рамки даної книги (хоча пізніше ви познайомитеся з «нечіткою логікою», що являє собою нетрадиційний метод управління системами за допомогою комп'ютера).

DSP не призначені для автономного застосування. Звичайно вони входять до складу систем і використовуються як пристрої управління зов-

нішнім обладнанням, а також для обробки вхідних сигналів і формування відповідного відгуку.

Програмовні логічні інтегральні схеми – це матричні інтегральні схеми, що дозволяють програмно скомпонувати в одному корпусі електронну схему, еквівалентну схемі, що включає в себе від кількох десятків до кількох сотень інтегральних схем стандартної логіки. У порівнянні з іншими мікроелектронними технологіями, технологія програмовних логічних інтегральних схем забезпечує рекордно короткий проектно-технологічний цикл (від кількох годин до кількох днів), мінімальні затрати на проектування, максимальну гнучкість при необхідності модифікації апаратури.

Найбільшу спеціалізацію і різноманітність функцій мають **мікроконтролери**, які використовуються у вбудованих системах вимірювання та управління, у тому числі і в побутових приладах. Загальна кількість кристалів з різними системами команд перевищує 500, і всі вони, за рахунок існування виробів з їх використанням, мають свою постійну частку ринку.

Основне призначення вбудованих мікроконтролерів — забезпечити за допомогою недорогих засобів гнучке (програмовне) управління об’єктами й зв’язок із зовнішніми пристроями. Ці мікроконтролери не призначені для реалізації комплексу складних функцій, але вони здатні забезпечити ефективне управління в багатьох областях застосування. Недорогими будемо вважати мікроконтролери, вартість яких становить від 1,0 до 20,0 доларів за штуку (ціна залежить від технічних характеристик, кількості виводів корпусу, обсягу закупок).

Вбудовані мікроконтролери містять значну кількість допоміжних пристроїв, завдяки чому забезпечується їхнє включення в реалізовану систему з використанням мінімальної кількості додаткових компонентів. До складу цих мікроконтролерів звичайно входять:

- схема початкового запуску процесора (Reset);
- генератор тактових імпульсів;
- центральний процесор;
- пам’ять програм (E(E)P)ROM і програмний інтерфейс;
- пам’ять даних RAM;
- засоби введення-виведення даних;
- таймери, що фіксують число командних циклів.

Загальна структура мікроконтролера показана на рис. 1.1. Ця структура дає уявлення про те, як мікроконтролер зв’язується із зовнішнім світом.

Більш складні мікроконтролери, що вбудовуються, можуть додатково реалізувати такі можливості:

- вбудований монітор/налагоджувач програм;
- внутрішні засоби програмування пам’яті програм (ROM);
- обробка переривань від різних джерел;
- аналоговий введення-виведення;

• послідовний введення-виведення (синхронний і асинхронний);
• паралельний введення-виведення (включаючи інтерфейс з комп'ютером);
• увімкнення зовнішньої пам'яті (мікропроцесорний режим).

Рисунок 1.1 – Структура мікроконтролера

Усі ці можливості значно збільшують легкість застосування мікроконтролерів і роблять більш простим процес розробки систем на їхній основі. Слід відзначити, що для реалізації цих можливостей у більшості випадків потрібно розширення функцій зовнішніх виводів.

Дотепер десятки фірм, серед яких *Analog Devices*, *Atmel*, *Dallas Semiconductor*, *Oki*, *Philips*, *Infineon Technologies*, *Silicon Storage Technologies*, *Temic* і інші, продовжують випуск аналогів мікроконтролера 8051 (фірма Intel) — родоначальника всіх мікроконтролерів. У мікроконтролері 8051 реалізована CISC (Complex Instruction Set Computer) архітектура процесорного ядра, що оперує з повним набором інструкцій. У класичному мікроконтролері 8051 для виконання більшості інструкцій потрібно 1-2 машинних тактів. Фірма *Dallas Semiconductor* випускає аналог мікроконтролера 8051, у якому основні інструкції виконуються за чотири такти. В аналогах мікроконтролера 8051, що випускаються фірмами *Infineon* і *Philips Semiconductors*, для виконання основних інструкцій потрібно шість машинних тактів. Усього в мікроконтролері 8051 реалізовано 255 інструкцій. Фірмою *Intel* випускаються також мікроконтролери MCS151/251, цілком сумісні на рівні кодів інструкцій з мікроконтролером 8051. У 1994 р. фірма *Philips Semiconductors*, що випускає більш 60 модифікацій мікроконтролера 8051, створила на базі популярної 8-розрядної архітектури оригінальний 16-розрядний мікроконтролер 8051XA, сумісний на рівні кодів інструкцій з мікроконтролером 8051. Мікроконтролер 8051XA може працювати в двох режимах: розширеному й у режимі сумісності. У розширеному режимі використовуються нові можливості мікроконтролера, у тому числі й ефективні інструкції для мультизадачної обробки. У мікроконтролері 8051XA більшість інструкцій типу регістр-регістр виконується за три

машинних такти (100 нс) при тактовій частоті 30 МГц. Усього в мікроконтролері 8051XA реалізовано 479 інструкцій. У таблиці 1.1 наведено основні параметри найпоширеніших 8-розрядних мікроконтролерів.

Таблиця 1.1 – Основні параметри 8-розрядних мікроконтролерів

Фірма, тип	Тактова частота, МГц	Вбудована пам'ять		Напруга живлення, В	Таймери	АЦП, ЦАП	Послідовний інтерфейс
		OTP/ROM/ Flash/EEPROM	RAM				
1	2	3	4	5	6	7	8
Analog Devices AduC824	12…16	Flash (8 кбайт) EEPROM (640 байт)	256 байт	3/5	3 станд. 8051 (16-розр.)	24- і 16-розр., ЦАП	UART, I2C, SPI
AduC812						12-розр., ЦАП	
Atmel AT89	0…33	Flash (1..32 кбайт) EEPROM (128..512 байт)	128…512 байт	2.7…6	1...3 (16-розр.)		UART, SPI
AT90	0…12	Flash (1..8 кбайт) EEPROM (0..512 байт)	0…512 байт	2.7…6	1...4 (8- і 16-розр.)	10-розр.	UART, SPI
ATtiny	0…8	Flash (1..2 кбайт) EEPROM (0..128 байт)	0…128 байт	1.8…5.5	1...2 (8- і 16-розр.)	10-розр.	UART, SPI
ATmega	0…6	Flash (16..128 кбайт) EEPROM (512…4 кбайт)	1…4 кбайт	2.7…5.5	3...4 (8- і 16-розр.)	10-розр.	UART, SPI
Cybernetic Micro Systems P-51	1…60	-	8 кбайт програм і 4 кбайт даних	3.3/5	3 (16-розр.)	-	UART
Infineon SABC500	0…40	ROM (8…64 кбайт) OTP (8…64 кбайт)	256…3328 байт	4.5…5.5	3...5 (16-розр.)	10-розр.	1… 2 (UART, USART)
Philips 87C51Fx	33	OTP (8…32 кбайт)	256 байт	2.7…5.5	3 (16-розр.)	-	UART
Rx2	20	Flash (16..64 кбайт)	512…1024 байт	5	3 (16-розр.)	-	UART
Hitachi H8/3664	10	Flash (32 кбайт)	2 кбайт	2.5…5.5	3 (8- і 16-розр.)	10-розр.	UART, I2C, SPI
H8/3802	10	OTP (16 кбайт)	1 кбайт	2.5…5.5	5 (8- і 16-розр.)	8-розр.	UART, SPI

Продовження таблиці 1.1

1	2	3	4	5	6	7	8
Microchip PIC12Cxxx	10	OTP 768…3584 байт EEPROM 16 байт	25…128 байт	2.5…5.5	1	8-розр.	-
PIC14Cxxx	20	OTP (7 кбайт)	192 байт	2.7…6	2	8-розр.	I2C, SMB
PIC16C5xxx	20	OTP 576…3072 байт	24…73 байт	2…6.25	1	-	
PIC16C6xxx	20	OTP (896…14 кбайт) EEPROM (128 байт)	80…368 байт	2.5…6.25	1...3	-	UART, I2C, SPI
PIC16C7xxx	20	OTP (896…14 кбайт)	36…368 байт	2.5…6.25	1..2	8-, 12-розр.	USART, I2C, SPI,
PIC17Cxxx	33	OTP (4…32 кбайт)	232…902 байт	2.5…6	4	10-розр.	USART, I2C, SPI,
PIC18Cxxx	40	OTP (16…32 кбайт)	512…1536 байт	2.5…5.5	2	10-розр.	USART, I2C, SPI, MI2C
Motorola MC68HC908JK1	0…8	Flash (1.5 кбайт)	128 байт	3/5	16-розр. (2 канала ШІМ)	8-розр.	-
MC68HC908JK3		Flash (4 кбайт)	128 байт	3/5	16-розр. (2 канала ШІМ)	8-розр.	-
MC68HC908JL3		Flash (4 кбайт)	128 байт	3/5	16-розр. (2 канала ШІМ)	8-розр.	-
MC68HC908GP32		Flash (32 кбайт)	512 байт	3/5	16-розр. (2 канала ШІМ)	8-розр.	SCI, SPI
MC68HC908AS60		Flash (60 кбайт)	2 кбайт	5	16-розр. (8 каналів ШІМ)	8-розр.	SCI, SPI
NEC K0	10	Flash (4 кбайт)	256…3 кбайт	1.8…5.5	2...8 (8- і 16-розр.)	АЦП, ЦАП	I2C, IrDA, CAN
K0S	10	Flash (4 кбайт)	128…1 кбайт	1.8…5.5	2...6 (8- і 16-розр.)	АЦП	I2C, USB
Toshiba TLCS870/C	0.032…16	ROM (0…60 кбайт)	256…1024 байт	1.8…5.5	8-, 16- і 24-розр.	АЦП	1… 4 UART
TLCS870/x	0.032…16	ROM (0…60 кбайт)	512 байт	1.8…5.5	8- і 16-розр.	АЦП	UART, I2C
Zilog Z80	0…33	ROM (0…8 кбайт)	0…2 кбайт	3…5.5	4 (8- і 16-розр.)	10-розр.	UART
EZ80	0…80	ROM (0…8 кбайт)	0…8 кбайт	3/5	6 (16-розр.)	-	UART

Одна з проблем, що існують при використанні повного набору інструкцій (**CISC**-архітектура), полягає в складності реалізації компілятора. Застосування повного набору інструкцій, не формованих компілятором, немає сенсу. Переважним є використання скороченого набору інструкцій. У результаті пошуку найбільш оптимальної архітектури для числової обробки, з'явилася архітектура процесорного ядра зі скороченим набором інструкцій — **RISC** (Reduced Instruction Set Computer) архітектура. Ідея

RISC архітектури народилася в розроблювачів великих ЕОМ. Вони встановили, що різні види даних мають різну частоту виникнення. Найбільшу частоту виникнення (більш 50 %) мають локальні дані, а на виклик процедур приходиться не менш 30 % усього часу обробки. З іншого боку, однією з типових структур даних при числовій обробці є векторні дані. Поняття „вектор” часто застосовуване в обчислювальній техніці, відрізняється від поняття „вектор” в математиці й фізиці. Вектор — це впорядкований список даних. Число елементів списку — довжина вектора. Ідеальний випадок, якщо весь список можна зберігати в реєстровому файлі процесорного ядра (тобто кількість регістрів дорівнює довжині вектора). У такому випадку при звертанні до даних, що міститься в реєстровому файлі, можна досягти максимальної швидкодії. Реєстровий файл чи векторні регістри можна розглядати як буферну пам’ять, розташовану між операційним блоком процесорного ядра й основною пам’яттю. Крім того, вкрай вигідно мати можливість за допомогою однієї інструкції виконувати операції над безліччю даних, що можуть бути розташовані в пам’яті чи реєстровому файлі. Подібні інструкції дозволяють реалізувати ортогональну (симетричну) структуру, що дає можливість виконувати кожну операцію з будь-яким регістром, використовуючи при цьому будь-який набір команд. Скорочення набору інструкцій дозволяє зменшити площу, займану процесорним ядром на кристалі, а отже, припускає розширення площі для внутрішніх регістрів (реєстрового файлу). У мікропроцесорах з скороченим набором інструкцій підвищення ефективності досягається також за рахунок застосування компілятора для безпосередньої трансляції програм, написаних на мовах високого рівня. В наш час багато фірм випускають 8- і 16-розрядні мікроконтролери з RISC-подібною архітектурою. Це фірми *Atmel* (серія AVR), *Texas Instruments* (серія MSP430), *Infineon* (серія 166) і ін.

1.2 Функції, що виконуються мікропроцесорами у вимірювальних приладах

Загальні відомості. Як уже відзначалося, для сучасного етапу розвитку техніки характерно усе більш інтенсивне і глибоке проникнення в її різні галузі мікропроцесорів, радикально перетворюючі властивості багатьох пристроїв і відкриваючих нові можливості їхнього застосування. За широтою й ефективністю застосування мікропроцесорів одне з перших місць займає контрольно-вимірювальна техніка.

Природно виникає питання: «Що дає застосування мікропроцесорів у вимірювальних приладах як ефективні схемні рішення, в основі яких лежить мікропроцесорна система?»

У загальному плані відповіддю можуть служити заголовки ряду журнальних статей і інформаційних матеріалів: «Мікропроцесор робить революцію в електронному приладобудуванні». І це дійсно так. Але навіть якщо стримати патетичні інтонації, і перейти до звичайної технічної мови,

то варто сказати, що застосування мікропроцесорів у вимірювальній техніці дозволяє різко підвищити точність приладів, значно розширити їхні можливості, підвищити надійність, швидкодію, вирішити задачі, що раніше взагалі не вирішувалися.

Конкретний розгляд функцій мікропроцесорних систем у вимірювальних приладах показує, що за допомогою цих систем досягаються багатофункціональність приладів, спрощення управління процесом вимірювання, автоматизація регулювання, самокалібрування й автоматична перевірка, поліпшення метрологічних характеристик приладу, виконання обчислювальних процедур, статистична обробка результатів спостережень, визначення і переклад у лінійну форму функції вимірюваної фізичної величини, створення програмувальних, цілком автоматизованих приладів. З'явився новий клас «інтелектуальних» приладів, які також називають «думаючими» чи «розумними».

Радикально змінилася ідеологія побудови приладів. МП став основною частиною приладу, що привело до зміни конструкції і схемних рішень, компонування, управління, включенню обробки даних у вимірювальну процедуру (виконувану без участі експериментатора). Впровадження МП відкрило можливість побудови багатофункціональних приладів із гнучкими програмами роботи, зробило прилади більш економічними, полегшило вирішення задачі виходу на стандартну інтерфейсну шину (канал загального користування) і управління інтерфейсом. Усе це спростило експлуатацію приладів, різко підвищило продуктивність праці їхніх користувачів.

Розглянемо більш докладно основні можливості, особливості приладів, що містять мікропроцесорні системи, і з'ясуємо, в результаті чого досягаються ці можливості.

Багатофункціональність. Ідея побудови багатофункціональних вимірювальних приладів, призначених для вимірювання декількох параметрів сигналів чи характеристик об'єкта дослідження, не нова. Вона здійснюється вже більш трьох десятиліть. Але до застосування МП багатофункціональні прилади являли собою сукупність декількох функціональних вузлів, об'єднаних в одне конструктивне ціле. При експлуатації таких приладів перехід від однієї функції до іншої виробляється за допомогою комутуючих пристроїв. У результаті комутації сполучних ланцюгів користувач складає, з окремих вузлів визначений прилад для вимірювання конкретного параметра сигналу чи характеристики випробуваного об'єкта. Алгоритм роботи засобу вимірювання, закладений при його розробці, у процесі експлуатації зберігається незмінним. Інакше кажучи, *традиційні* багатофункціональні прилади виконані за схемою з *твердою логікою.* Для неї характерне протиріччя між багатофункціональністю, числом можливих функцій приладу, з одного боку, і економічністю, а також технічною ефективністю — з іншої. Проблема комутації і управління ніколи не втрачала гостроти

при конструюванні приладів, призначених для виконання ряду функцій, і далеко не завжди вирішувалась успішно.

Мікропроцесорна система, введена до складу багатофункціонального засобу вимірювання, радикально змінила його, перетворила пристрій із твердою логікою роботи в *програмно-керований пристрій.* Функціональні можливості такого пристрою визначаються виконуваною програмою, і можуть бути легко видозмінені шляхом переходу до іншої програми, збереженої в постійній пам'яті. Тому програмувальну логіку роботи подібних приладів іноді називають «збереженою». Вона створює гнучкість перебудови, дозволяє нарощувати функції при модернізації приладу без істотних змін у його схемі. Застосування програмувальної логіки, як правило, зменшує вартість приладу.

Підвищення точності приладів. Нагадаємо, що під точністю засобу вимірювань розуміють якість засобу вимірювань, що відбиває близькість до нуля його похибок. При цьому близькість до нуля систематичних похибок визначає правильність засобу вимірювань, а близькість до нуля випадкових похибок — збіжність показань засобу вимірювань.

Похибки засобу вимірювань відносяться до його метрологічних характеристик. Можна перерахувати такі шляхи підвищення точності вимірювального приладу, які досягаються в результаті введення мікропроцесорної системи до складу приладу:

- автоматична компенсація (виключення) систематичної похибки, зокрема, автоматичне встановлення нуля перед початком вимірювань;
- автоматичне виконання градуювальної операції (самокалібрування);
- виконання самоконтролю;
- зменшення впливу випадкових похибок шляхом проведення багаторазових спостережень (одиничних вимірювань) з наступним усередненням їхніх результатів;
- виявлення і виключення грубих похибок;
- виведення на дисплей інформації про числові значення похибок по ходу вимірювань.

Розширення вимірювальних можливостей приладів. Застосування МП дозволяє істотно розширити можливості вимірювань широкого переліку параметрів сигналів і характеристик пристроїв. Це зв'язано насамперед з використанням, здавалося б застарілих, видів вимірювань: непрямих і сукупних.

Через необхідність застосування декількох приладів, зняття ряду відліків і наступних обчислень непрямі вимірювання сприймаються як примітивні, несучасні. Навіть при використанні мікрокалькуляторів обчислення в деяких випадках можуть забирати значний час, і, головне, вони, вимагаючи постійної уваги і роботи, не дозволяють досягти високої продуктивності. Крім того, не дуже проста процедура оцінювання похибок непрямих вимірювань, а без цього ніяке вимірювання не може бути визнано достовірним.

Докорінно змінюється положення при включенні до складу приладу мікропроцесорної системи. За командою, одержуваною з клавіатури, вона автоматично відповідно до заданої програми вибирає режими вимірювань, запам'ятовує результати прямих вимірювань, проводить необхідні обчислення і видає знайдене значення вимірюваної фізичної величини на дисплей. Хоча вимірювання за своєю природою залишаються непрямими, експериментатор сприймає їх як прямі, оскільки, підімкнувши прилад об'єкта дослідження, безпосередньо одержує результат вимірювань.

Прикладом може служити вимірювання цифровим вольтметром потужності *P,* що розсіюється на навантажувальному резисторі. Вимірювання здійснюються згідно з формулою $P = U^2 / R$, де U — спад напруги на резисторі; R — опір резистора. Цифровому вольтметру задається програма, відповідно до якої спочатку вимірюється опір резистора і запам'ятовується отриманий результат, потім вимірюється напруга на резисторі, після чого обчислюється потужність.

Як інші приклади, можна привести вимірювання опорів резисторів на основі формули закону Ома $R = U / I$ та вимірювання коефіцієнта підсилення підсилювача відповідно до визначення $K = U_{вх} / U_{вих}$.

Наведені приклади відносяться до порівняно простих математичних співвідношень. Але на практиці нерідко виникає необхідність знаходження непрямим шляхом значень і таких фізичних величин, що залежать від великого числа безпосередньо вимірюваних інших фізичних величин. Застосування мікропроцесорних систем робить ці вимірювання простими і зручними для користувача, що одержує прямі показання приладу і не відчуває того, що фактично виконуються складні непрямі вимірювання.

Ще більш ефективні мікропроцесорні системи при сукупних вимірюваннях, тобто одночасних вимірюваннях декількох однойменних фізичних величин, при яких шукані значення величин знаходять розв'язуванням системи рівнянь, одержуваних при прямих вимірюваннях різних поєднань цих величин.

Спрощення і полегшення керування приладом. На перший погляд розширення функцій, які виконуються програмними приладами, повинно було б привести до збільшення числа органів управління. Але в дійсності це не так. Одним із критеріїв високого рівня програмного забезпечення вимірювального приладу є ступінь складності його передньої панелі.

Прийнято вважати, що «розумний» прилад повинен мати простий набір органів управління. Для сучасних приладів, що містять МП, характерна кнопкова система керування, конструктивно оформлювана у вигляді клавіатур (виносної чи на передній панелі приладу), що зовні нагадує клавіатуру калькулятора.

Так, наприклад, в одного з цифрових мультиметрів, що має багато функціональних можливостей, керування зміною функцій, діапазонів вимірювань і режимів роботи (всього 44 сполучення) здійснюється за допомогою клавіатури, що складається з 17 клавіш. Цього вдалося досягти вна-

слідок того, що кожна клавіша керує аналоговими схемами непрямим чином — через мікропроцесор, а останній селектує різні сполучення сигналів, що вводяться при натисканні клавіш.

Іншим прикладом спрощення керування, скорочення числа ручок і кнопок може служити малогабаритний 7-розрядний цифровий частотомір, що працює в діапазоні 10 Гц ... 1 ГГц. На передній панелі цього приладу є тільки два вхідних контакти (для сигналів частотою 10 ... 75 МГц, другий — для сигналів частотою 70 МГц ... 1 ГГц), кнопка вмикання приладу, ручка регулятора чутливості і двопозиційний кнопковий перемикач тривалості тимчасових воріт («часу вимірювання»): 1 с і 1 мс.

Радикально зменшує число органів керування автоматизація вибору меж вимірювань, інтервалу дискретизації напруги досліджуваного сигналу й інших режимів роботи приладу. У деяких приладах передбачена сигналізація про некоректні кроки експериментатора й видача на дисплей інструкцій, що вказують, що повинен робити експериментатор, яка правильна послідовність дій.

Можливість одержання математичних функцій вимірюваних значень. У залежності від розв'язуваної задачі експериментатора можуть цікавити не безпосередньо одержуване при вимірюванні значення фізичної величини, а його різні математичні функції. Багато приладів, що містять мікропроцесорні системи, дозволяють автоматично виконувати запрограмовані функціональні перетворення. Прикладами таких перетворень можуть служити.

1. Множення знайденого значення A на константу c. При цьому показання приладу $A_п = cA$. Константа вводиться за командою при натисканні клавіші.

2. Одержання відхилень результату вимірювання A від номінального значення $A_н$: абсолютного $A - A_н$ і відносного, вираженого у відсотках відносно номінального значення, тобто $100(A - A_н)/A_н$.

3. Зсув, що припускає вирахування константи з результату вимірювання.

4. Обчислення відношень: ділення на константу (наприклад, при визначенні значення постійного струму через резистор за виміряним вольтметром значенням спаду напруги на цьому резисторі), знаходження частки від ділення одного результату вимірювання на інший результат вимірювання (наприклад, при визначенні коефіцієнта підсилення за результатами вимірювань напруг на виході і вході підсилювача).

5. Подання результату вимірювання в логарифмічних одиницях. Наприклад, згасання чотириполюсника, вираженого в децибелах: $a = 20\lg(U_{вх}/U_{вих})$.

6. Лінеаризація залежностей. Така необхідність особливо часто зустрічається при електричних вимірюваннях неелектричних величин (наприклад, температури), коли напруга електричного сигналу на виході дат-

чика являє собою нелінійну функцію вимірюваної фізичної величини на його вході. У таких ситуаціях значення вихідної напруги датчика перетворяться за допомогою АЦП у числа, що обробляються мікропроцесорною системою за заданою програмою, і в підсумку одержується лінійний зв'язок між показаннями приладу і значеннями фізичної величини на вході датчика.

У деяких приладах передбачена можливість обчислення за бажанням користувача довільних (зрозуміло, у визначених межах) математичних співвідношень.

Одержання статистичних характеристик. Ряд вольтметрів, у складі яких є мікропроцесорна система, дозволяють формувати оцінки таких імовірнісних характеристик аналізованої випадкової змінної, як середнє значення, середня потужність, середньоквадратичне значення, дисперсія, середньоквадратичне відхилення, а також коефіцієнт кореляції двох випадкових змінних. Мікропроцесорні прилади, спеціально призначені для вимірювання статистичних характеристик сигналів, мають більш широкі можливості.

Мініатюризація й економічність апаратури. Різке зменшення числа компонентів у схемі приладу внаслідок виконання багатьох функцій мікропроцесорною системою, їх відносно невисока вартість, значне зниження споживаної потужності дозволяють створювати малогабаритні та економічні прилади.

Підвищення надійності приладів. Воно зумовлено зменшенням числа елементів схем, здійсненням автодіагностики, застосуванням вузлів з некаліброваними характеристиками (наприклад, підсилювача в каналі вертикального відхилення осцилографа), можливістю виконання корекції похибок, що поліпшує метрологічну надійність.

Скорочення тривалості розробки. Часто для одержання нових властивостей приладу, виконуваного на основі мікропроцесорної системи, не потрібно значних змін у схемі і тим більше в конструкції приладу. Основний зміст розробки полягає в створенні необхідного програмного забезпечення. З огляду на те, що для широко застосовуваних МП уже накопичена бібліотека досить розроблених типових прикладних програм та вимірювальних процедур, у багатьох випадках розроблення програмного забезпечення приладу в значній мірі зводиться до раціонального вибору наявних програм.

Організація вимірювальних систем. Прилад, що містить МП, звичайно оснащений інтерфейсами, що дозволяють під'єднувати його до стандартної інтерфейсної шини. Це дає можливість поєднувати визначену сукупність приладів у єдину вимірювальну систему (вимірювально-обчислювальний комплекс).

1.3 Поліпшення метрологічних характеристик приладів

Розглянемо основні можливості і способи зменшення похибок у приладах, що містять мікропроцесорні системи.

Виключення систематичної похибки. Найбільш часто систематичні похибки обумовлені зсувом нуля, невідповідністю реального значення коефіцієнта передачі тракту сигналу номінальному значенню, нерівномірністю амплітудно-частотної характеристики тракту передачі сигналу, впливом характеристики аналого-цифрового перетворювача (АЦП).

Наявність у приладі мікропроцесорної системи дозволяє скорегувати, виключити систематичні похибки. Коротко освітлимо шляхи розв'язування задачі. Для виключення зсуву нуля, наприклад у цифровому вольтметрі, його вхідні контакти замикаються накоротко і приєднуються до точки з нульовим потенціалом (заземлюються). При цьому число, одержуване на виході АЦП, характеризує зсув нуля. Воно запам'ятовується і віднімається з показань приладу. Надалі, коли вимірюється напруга, що підводиться до вхідних контактів приладу, автоматично вноситься виправлення, що усуває систематичну похибку, викликану зсувом нуля.

Принцип корекції систематичної похибки, пов'язаної з тим, що значення коефіцієнта передачі тракту сигналу (характеризуючи внесене ним посилення чи ослаблення) відрізняється від номінального, полягає в такому (рис. 1.2).

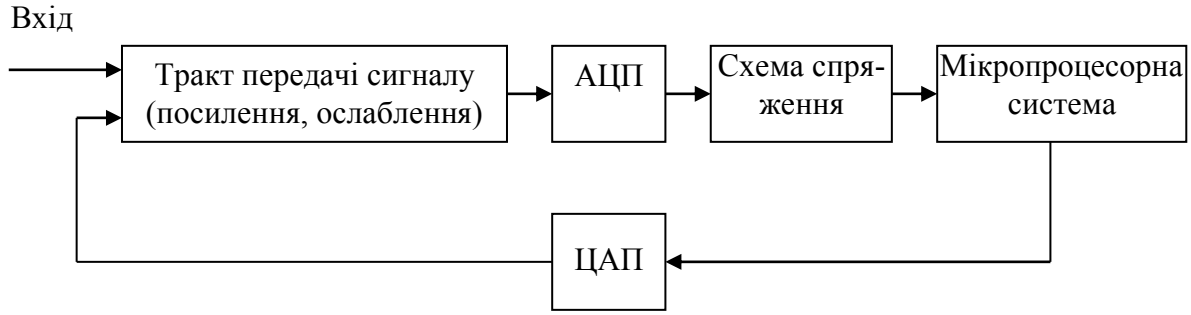


Рисунок 1.2 – До пояснення принципу корекції систематичної похибки

У пам'яті мікропроцесорної системи зберігається число B, що не руйнується при відімкненні живлення системи і відповідає строго визначеному значенню A_0 вхідної напруги, тобто число, що повинно бути отримане на виході АЦП, якщо на вхід вольтметра надходить напруга A_0 і коефіцієнт передачі тракту сигналу, а також коефіцієнт перетворення АЦП відповідають своїм номінальним значенням. Усередині приладу є цифро-аналоговий перетворювач (ЦАП), що містить зразкове джерело живлення. При підведенні числа B до входів ЦАП на його виході утвориться напруга, значення якої дорівнює A_0. Ця напруга подається на вхід приладу. У результаті аналого-цифрового перетворення виходить B', що відрізняється від числа B через наявність систематичної похибки. Її характеризує відношення чисел $\alpha = B/B'$. Значення коефіцієнта α обчислює мікропроцесор, і

воно фіксується в пам'яті. Таким чином, у пам'яті міститься поправочний множник.

Коли на вхід приладу надходить вимірювана напруга постійного струму, то на виході АЦП виходить число *C'*, що відповідає значенню цієї напруги. Введення поправочного множника, тобто виконуване мікропроцесором множення числа *C'* на коефіцієнт α, дає правильний результат вимірювання — число *C*.

Задача виключення систематичної похибки, яка обумовлена нерівномірністю АЧХ тракту передачі сигналу, особливо складна при використанні широкодіапазонних вольтметрів. Наявність мікропроцесорної системи в приладі істотно спрощує рішення цієї задачі.

На виносному вимірювальному пробнику (вимірювальній головці) закріплена табличка, що вказує масштабні коефіцієнти (поправочні множники) для сукупності частот, що входять у діапазон приладу 0...2 ГГц. Кожен масштабний коефіцієнт, що відповідає визначеній частоті f_i, може бути введений у прилад для виключення систематичної похибки, обумовленої відмінністю значення коефіцієнта передачі K_i від номінального значення K_0. У пам'яті каліброваних коефіцієнтів зберігається значення K_0=1000. Це значення можна висвічувати на дисплеї приладу і змінювати за командою з клавіатури (чи інтерфейсної шини). Якщо експериментатор вводить значення K_i, відмінне від K_0, то на дисплеї відображається фактичне (введене) значення коефіцієнта з метою залучення уваги експериментатора до факту зміни калібрування.

При вимірюванні напруги мікропроцесор виконує операцію множення на поправочний множник, тобто враховує введений масштабний коефіцієнт, враховуючи тим самим систематичну похибку.

Зменшення впливу випадкової похибки. Ця складова похибки вимірювання, як відомо, не може бути виключена. Її вплив можна зменшити раціональною обробкою результатів спостережень.

Для обліку випадкових похибок користуються імовірнісними характеристиками. З теорії ймовірностей відомо, що найбільш повно випадкові величини характеризуються законами розподілу ймовірностей. Але при рішенні багатьох вимірювальних задач цілком достатніми характеристиками випадкових похибок служать їхні найпростіші числові характеристики: середнє значення (математичне очікування) і середньоквадратичне відхилення. Оскільки число *N* спостережень завжди обмежено, то реально користуються статистичними числовими характеристиками, які називаються оцінками характеристик.

Оцінку середнього значення результатів спостережень обчислюють за формулою

$$A_{сер} = \frac{1}{N}\sum_{i=1}^{N} A_i, \qquad (1.1)$$

де A_i — результат i-го спостереження, не спотворений систематичною похибкою; N — число спостережень.

Оцінкою середньоквадратичного відхилення випадкової похибки результату спостережень служить вираз

$$\hat{\sigma}_\upsilon = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\upsilon_i^2}, \qquad (1.2)$$

де $\upsilon_i = A_i - A_{сер}$ — відхилення i-го результату спостереження від його середнього значення.

У теорії похибок доводиться, що середньоквадратичне відхилення результату вимірювання, що визначається як $\hat{\sigma}_{A\,сер}$, обчислене для N груп серії незалежних спостережень (в кожній групі по N спостережень), при великому числі N, набагато менше, ніж середньоквадратичне відхилення $\hat{\sigma}_\upsilon$. Розраховують $\hat{\sigma}_{A\,сер}$ за формулою

$$\hat{\sigma}_{A\,сер} = \frac{\hat{\sigma}_\upsilon}{\sqrt{N}} = \sqrt{\frac{\left(\sum_{i=1}^{N}\upsilon_i^2\right)}{N(N-1)}}. \qquad (1.3)$$

Формула (1.2) визначає абсолютну похибку. Для знаходження відносної середньоквадратичної випадкової похибки δ_υ значення $\hat{\sigma}_\upsilon$, обчислене за (1.2), відносять до $A_{сер}$:

З написаних виразів видно, що проведення багаторазових вимірювань з наступним усередненням – ефективний спосіб зменшення впливу випадкової похибки на результат вимірювання.

Компенсація внутрішніх шумів. Ця операція дозволяє підвищити чутливість вимірювального приладу, розширити діапазон вимірюваних значень напруги в сторону малих значень. Принцип компенсації, використаний у вимірювачі рівня високочастотних сигналів, зводиться до такого.

До складу приладу входить вимірювальний перетворювач, що здійснює перетворення високочастотної напруги змінного струму в напругу постійного струму, значення якої відповідає середньоквадратичному значенню напруги змінного струму. Ще до подачі досліджуваного сигналу *s(t)* протягом інтервалу часу, який витрачається на автоматичне регулювання нуля, вимірюється середній квадрат шумового сигналу *n{t}* на вході перетворювача. Результат вимірювання $\overline{n^2(t)}$ запам'ятовується. Після підведення до входу приладу корисного сигналу на вході перетворювача виходить сума сигналу і шуму. В перетворювачі сумарний сигнал *s(t)+n(t)* підноситься до квадрата, в результаті чого утворюється сигнал $s^2(t) + 2s(t)n(t) + n^2(t)$. Усереднення цього сигналу дає

$$\overline{s^2(t)} + 2\overline{s(t)n(t)} + \overline{n^2(t)} = \overline{s^2(t)} + \overline{n^2(t)}$$

(оскільки сигнали *s(t)* і *n(t)* незалежні, середнє значення їхнього добутку дорівнює нулю). З результату усереднення віднімається виміряний раніше середній квадрат $\overline{n^2(t)}$ шумового сигналу і отримана різниця дорівнює $\overline{s^2(t)}$. Витяг квадратного кореня дає середньоквадратичне значення "чистого" корисного сигналу *s(t),* оскільки шумова складова скомпенсована.

1.4 Процесорні похибки вимірювань

Розглядання процедури процесорних вимірювань як послідовності виконання аналогових, аналого-цифрових і цифрових вимірювальних перетворень, дозволяє уявити похибку у вигляді суми відповідних компонент. Відмінність результатів цифрових вимірювальних перетворень від потрібних визначають процесорні похибки вимірювань. Для вивчення цього виду похибок виділяють три причини їх появи:

- алгоритмічна похибка, зумовлена відмінністю прийнятого алгоритму вимірювання від адекватного (гіпотетичного);
- похибка округлення, викликана наявністю округлень проміжних цифрових перетворень в результаті вимірювання;
- динамічна процесорна похибка, джерелом появи якої є обмеженість швидкодії мікропроцесора (процесора).

Розвиваючи ідеї класифікації, засновані на виділенні факторів, що зумовлюють появу похибок, введемо в розгляд поняття **гіпотетичного алгоритму вимірювань**, що дозволяє одержати істинне значення вимірюваної величини. В тому разі, коли його можна сформувати, він відповідає значенню вимірюваної величини. Наприклад, застосовуючи закон Ома, можна інтерпретувати відповідні визначення сили струму *I*, напруги *U* і опору *R* $\left(I = U/R, U = IR, R = U/I\right)$ як гіпотетичні алгоритми вимірювань, виконання яких дозволяє визначити істинне значення *I*, *U* або *R*. Тоді виникнення алгоритмічної похибки буде зумовлено відмінністю прийнятого алгоритму вимірювання від гіпотетичного.

Застосування мікропроцесорів у вимірювальних приладах пов'язане з необхідністю оцінювати інструментальні похибки, які можуть виникнути при виконанні обчислювальних процедур, через обмеженість розрядної мережі процесора. Вони називаються похибками округлення.

При виборі алгоритмів обчислень, як правило, застосовується умова**,** щоб абсолютна похибка обчислень не перевищувала $\Delta_0 = 2^{-n}$, де *n* - розрядність МП.

Операції округлення в МП здійснюються, як правило, простим відсіканням невраховуваних розрядів або симетричним округленням з урахуванням значення старшого розряду, що відкидається. Якщо *l* - число нев-

рахованих при округленні розрядів при рівномірному законі розподілу ймовірностей округлення (що справедливо в більшості випадків при $n \geq 8$), то дисперсії похибки простого відсікання і симетричного округлення (при порівняно невеликих *l)*

$$D_y \cong 2^{-2n}/12 = \Delta_0^2/12. \quad (1.4)$$

Математичне сподівання інструментальної похибки округлення

$$M_y = \left[2^{-(n+l+1)}\right]. \quad (1.5)$$

Похибки відсікання мають від'ємний знак для будь-яких арифметичних операцій над числами, поданими у прямому і додатковому кодах. Тому при значній кількості послідовних арифметичних операцій похибки можуть нагромаджуватись і перевищити припустимий рівень. Особливо важливо проводити оцінювання цієї похибки при розрядній мережі мікропроцесорної системи, близької до розрядності аналого-цифрового перетворення.

У засобах вимірювання, що працюють під управлінням МП, суттєву роль відіграють часові затримки запуску аналого-цифрових перетворювачів порівняно з потрібною швидкістю вимірювань. Ці затримки значною мірою визначаються затратами процесорного часу на обробку переривань, управління, запам'ятовування, виконання інтерфейсних функцій. Тому при метрологічних випробуваннях мікропроцесорних засобів вимірювання, де похибка датування відліку залежить від стану вимірювальних модулів і програмних драйверів, необхідно передбачати експериментальне визначення характеристик цієї похибки.

Вплив похибки датування відліку на загальну похибку мікропроцесорного засобу вимірювання Δ залежить від швидкості зміни вхідного сигналу dx/dt:

$$\Delta = \Delta_{МПЗВ} + dx/dt\,\delta_t, \quad (1.6)$$

де $\Delta_{МПЗВ}$ - абсолютна похибка мікропроцесорного засобу вимірювання; δ_t - похибка датування відліків.

Похибка δ_t може визначатись як різниця астрономічного і реального часу виконання вимірювань або як міра відхилення інтервалу між послідовними опитуваннями АЦП і заданого інтервалу, котрий має підтримуватись постійним.

При організації опитування аналого-цифрових перетворювачів їх швидкість вибирають так, щоб у межах припустимих похибок можна було між двома послідовними вимірюваннями використати лінійну інтерполяцію $(dx/dt = const)$. Однак при дослідженні швидких процесів, як правило, не можна задовольнити цю умову, що призводить до появи динамічної похибки.

1.5 Архітектура процесорів

Мікропроцесорна система (МПС) – це зібрана в єдине ціле сукупність взаємодіючих інтегральних схем цифрової логіки та аналогових ланцюгів, організована у обчислювальну або в керуючу систему з мікропроцесором (мікроконтролером) як вузлом обробки інформації.

Узагальнена структура мікропроцесорної системи наведена на рис. 1.3. Коротко охарактеризуємо основні елементи, що входять до її складу.

Генератор тактових імпульсів – джерело послідовності прямокутних імпульсів, за допомогою яких здійснюється управління роботою МП у часі. Для сучасних МП не потрібний зовнішній генератор тактових імпульсів, він міститься безпосередньо в його схемі.

Основна пам’ять системи (зовнішня щодо МП) складається з **постійного** (ПЗП) і **оперативного** (ОЗП) запам’ятовувальних пристроїв.

ПЗП – це пристрій, в якому зберігається програма та сукупність констант. Вміст ПЗП не стирається при вимиканні живлення. ПЗП використовується як пам’ять програми.

ОЗП – це пам’ять програми і даних, що належать обробленню і результатам обчислень.

Пристрій введення-виведення (ПВВ) здійснює введення в систему даних, що належать обробленню. Пристрій виведення (ПВ) перетворює вихідні дані (результат оброблення інформації) у форму, зручну для сприйняття користувачем або зберігання. ПВВ служать гнучкі магнітні диски, клавіатура, дисплей, аналого-цифрові і цифроаналогові перетворювачі, графопобудовники, друкувальні пристрої тощо.

Далі розглянемо системи шин. **Шиною** називається група ліній передачі, що використовуються для виконання певної функції (по одній лінії на кожен передавальний біт). Особливістю структури МПС є магістральна організація зв’язків між модулями, що входять у її систему. Вона здійснюється за допомогою трьох шин. Ці шини з’єднують МП із запам’ятовувальним пристроєм (ПЗП, ОЗП) і інтерфейсами введення-виведення, внаслідок чого створюється можливість обміну даними між розглянутими модулями системи.

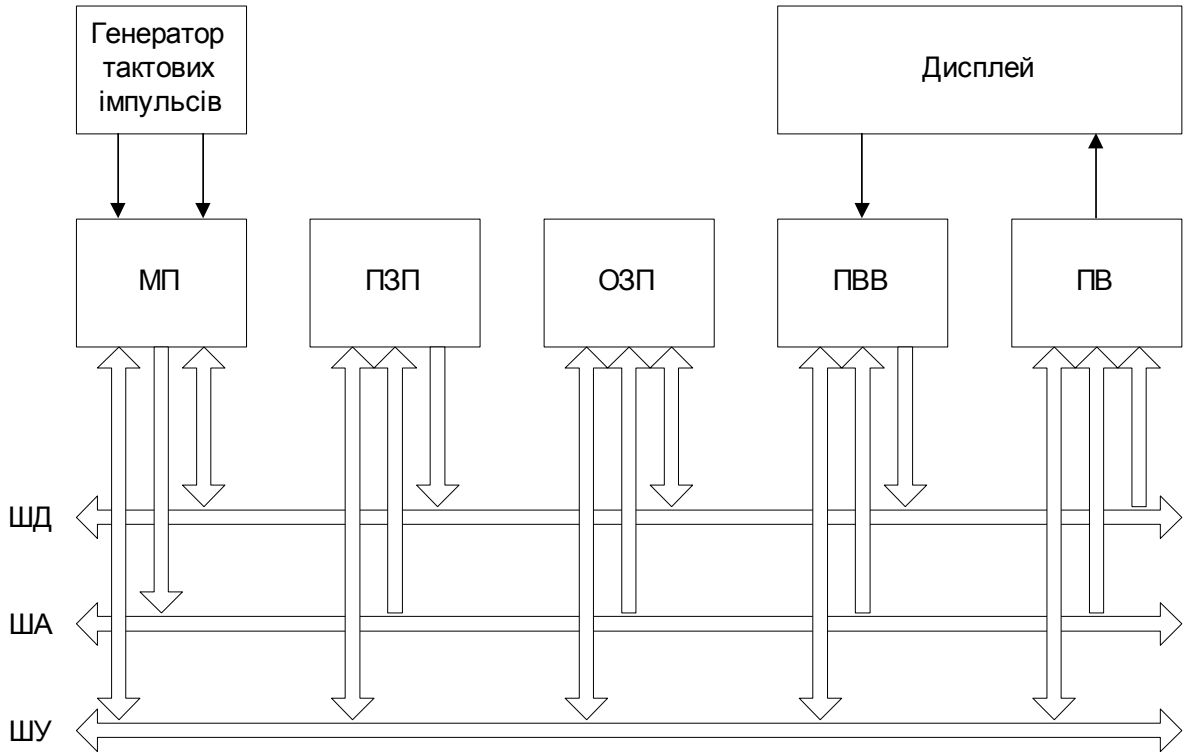


Рисунок 1.3 – Узагальнена структурна схема мікропроцесорної системи

Шина даних (ШД) – це двонаправлена шина: по ній дані можуть направлятися в МП або з нього. При цьому необхідно підкреслити, що одночасна передача даних в обох напрямках неможлива. Ці процедури рознесені в часі у результаті застосування часового мультиплексування.

По **шині адреси (ША)** інформація передається тільки в одному напрямі – від МП до модуля пам'яті або ПВВ.

Шина управління (ШУ) використовується для передавання сигналів, що обслуговують взаємодію, синхронізацію роботи всіх модулів системи і внутрішніх вузлів МП.

Перевагою шинної структури є можливість вмикання до МПС нових модулів, наприклад, кількох блоків ОЗП і ПЗП, для одержання потрібного обсягу пам'яті.

Порівняння архітектур CISC та RISC

В наш час існує багато RISC-процесорів, тому що склалася думка, що RISC-процесори більш швидкодійні, ніж CISC-процесори. Така думка не зовсім правильна. Існує багато процесорів, що називаються RISC, але насправді вони відносяться до CISC. Більш того, у деяких задачах CISC-процесори виконують програмний код швидше, ніж це роблять RISC-процесори, чи вирішують такі задачі, що RISC-процесори не можуть виконати.

Яка дійсна відмінність між RISC і CISC? CISC-процесори виконують великий набір команд з розвинутими можливостями адресації (безпосере-

дня, індексна і т.д.) і дають розробнику можливість вибрати найбільш придатну команду для виконання необхідної операції. У RISC-процесорах набір виконуваних команд скорочений до мінімуму. При цьому розробник повинний комбінувати команди, щоб реалізувати більш складні операції. Нижче буде показано як операції завантаження в стек («push») і вивантаження зі стеку («pop») реалізуються в RISC-процесорах за допомогою двох простих команд.

Можливість рівноправного використання всіх регістрів процесора називається «ортогональністю» чи «симетричністю» процесора. Це забезпечує додаткову гнучкість при виконанні деяких операцій. Розглянемо, наприклад, виконання умовних переходів у програмі. У CISC-процесорах умовний перехід, звичайно, реалізується відповідно до визначеного значення біта (прапорця) у регістрі стану. У RISC-процесорах умовний перехід може відбуватися при визначеному значенні біта, що знаходиться в будь-якому місці пам'яті. Це значно спрощує операції з прапорцями і виконання програм, що використовують ці прапорці.

Успіх при використанні RISC-процесорів забезпечується завдяки тому, що їх більш прості команди потребують для виконання значно менше число машинних циклів. Таким чином, досягається істотне підвищення продуктивності, що дозволяє RISC-процесорам ефективно вирішувати надзвичайно складні задачі.

Порівняння Гарвардської та Прінстонської архітектур

Багато років тому уряд Сполучених Штатів дав завдання Гарвардському і Прінстонському університетам розробити архітектуру комп'ютера для військово-морської артилерії. Прінстонський університет розробив комп'ютер, що мав загальну пам'ять для збереження програм і даних. Така архітектура комп'ютерів більше відома як архітектура Фон-Неймана за ім'ям наукового керівника цієї розробки (рис. 1.4).

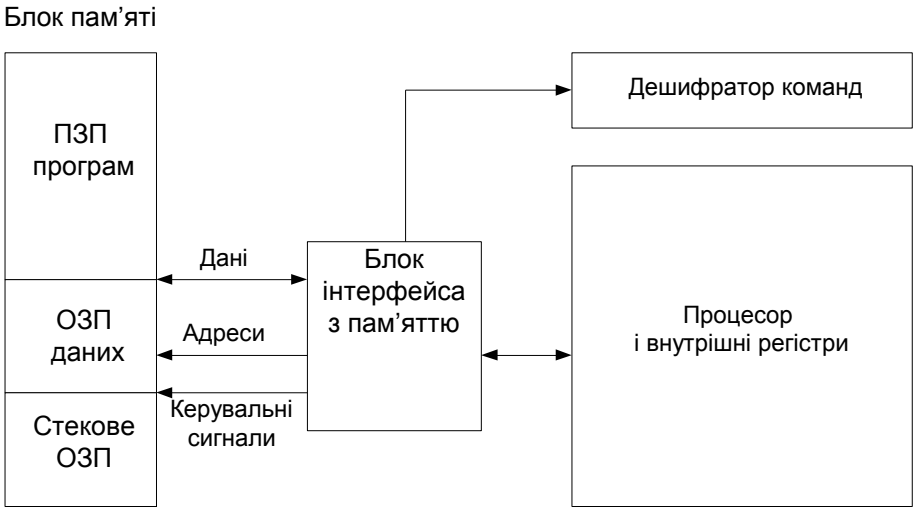


Рисунок 1.4 – Структура комп`ютера із Прінстонською архітектурою

У цій архітектурі блок інтерфейсу з пам’яттю виконує арбітраж запитів до пам’яті, забезпечуючи вибірку команд, читання і записування даних, розташовуваних у пам’яті чи внутрішніх регістрах. Може показатися, що блок інтерфейсу є найбільше вузьким місцем між процесором і пам’яттю, тому що одночасно з даними потрібно вибирати з пам’яті чергову команду. Однак у багатьох процесорах із Прінстонською архітектурою ця проблема вирішується шляхом вибірки наступної команди під час виконання попередньої. Така операція називається попередньою вибіркою («передвибіркою»), і вона реалізується в більшості процесорів з такою архітектурою.

Гарвардський університет подав розробку комп’ютера, у якому для збереження програм, даних і стеку використовувалися окремі банки пам’яті (рис. 1.5).

Прінстонська архітектура виграла змагання, тому що вона більше відповідала рівню технології того часу. Використання загальної пам’яті виявилося кращим через ненадійність лампової електроніки (це було до широкого поширення транзисторів) — при цьому виникало менше відмов.

Гарвардська архітектура майже не використовувалася до кінця 70-х років, доки виробники мікроконтролерів не зрозуміли, що ця архітектура дає переваги пристроям, які вони розробляли.

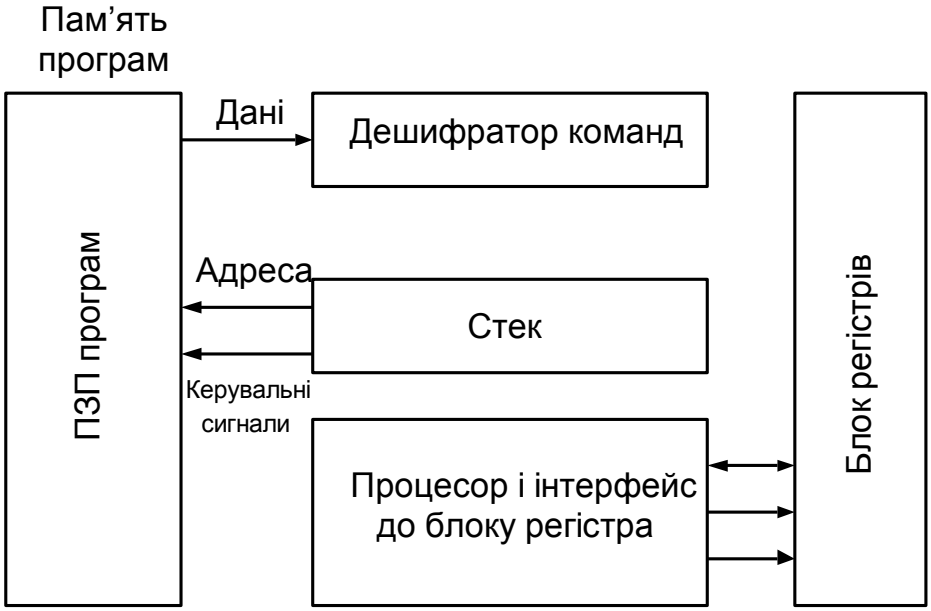


Рисунок 1.5 Структура комп`ютера із Гарвардською архітектурою

Наприклад, якщо процесору з Прінстонською архітектурою необхідно зчитати байт і помістити його в акумулятор, то він робить послідовність дій показану на рис. 1.6. У першому циклі з пам’яті вибирається команда, у наступному циклі дані, що повинні бути поміщені в акумулятор, зчитуються з пам’яті.

У Гарвардській архітектурі, що забезпечує більш високий ступінь паралелізму операцій, виконання поточної операції може поєднуватися з вибіркою наступної команди (рис 1.7). Команда також виконується за два цикли, але вибірка чергової команди відбувається одночасно з виконанням попередньої. Таким чином, команда виконується усього за один цикл (під час читання наступної команди).

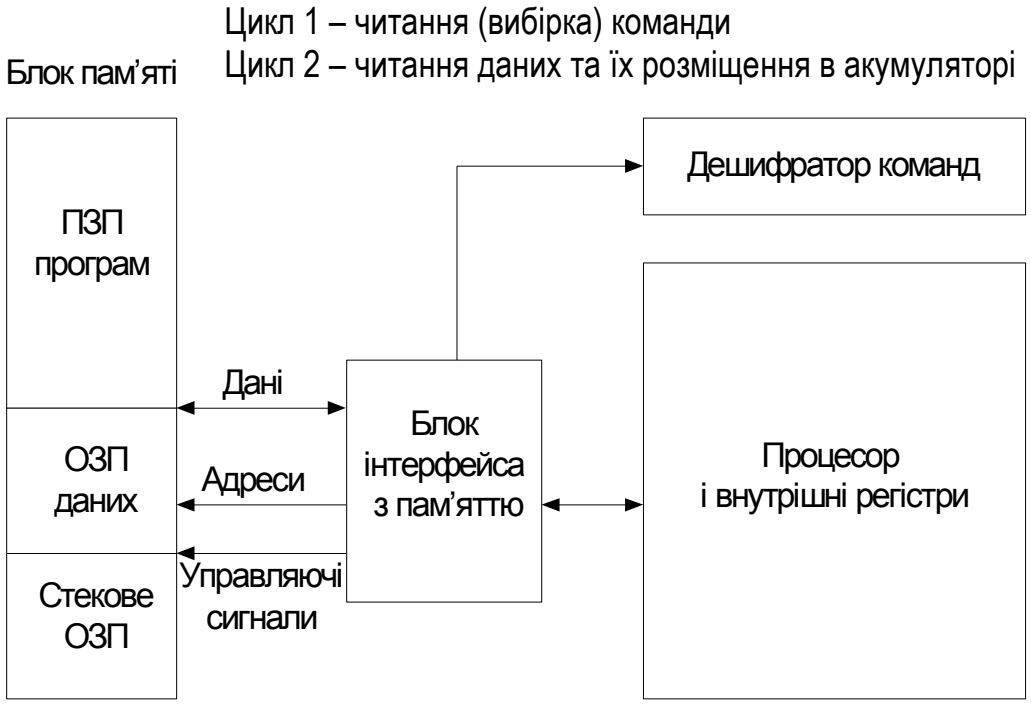
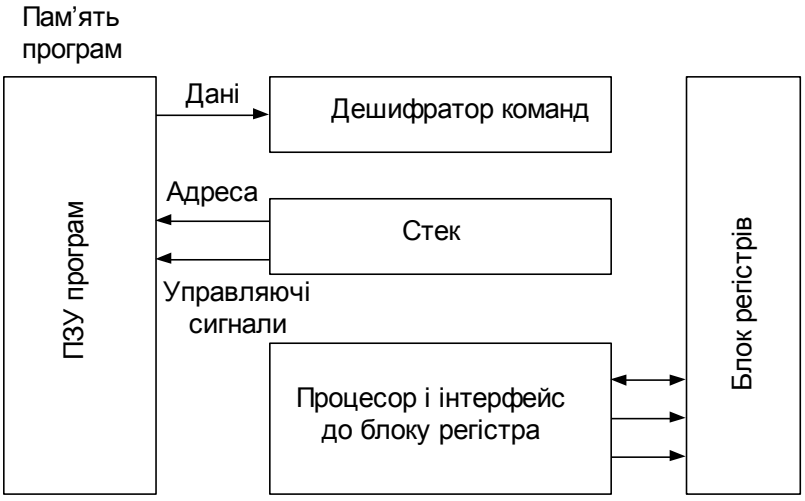


Рисунок 1.6 – Виконання команди *mov Acc, Reg* у Прінстонській архітектурі



Цикл 1 – завершення виконання попередньої команди – читання (вибірка) команди *mov Acc,R*
Цикл 2 – виконання команди *mov Acc,R* – читання (вибірка) наступної команди

Рисунок 1.7 – Виконання команди *mov Acc, R* в Гарвардській архітектурі

Цей метод реалізації операцій («паралелізм») дозволяє командам виконуватися за однакове число тактів, що дає можливість більш просто визначити час виконання циклів і критичних ділянок програми. Ця обставина є особливо важливою при виборі мікроконтролера для задач, де потрібно забезпечення строго заданого часу виконання.

Наприклад, мікроконтролер PIC фірми Microchip виконує будь-яку команду, крім тих, котрі модифікують вміст програмного лічильника, за чотири такти (один цикл). Це спрощує реалізацію критичних до часу процедур у порівнянні з мікроконтролером Intel 8051, де для виконання команд може знадобитися від 16 до 64 тактів. Через це часто не вдається підрахувати точний час виконання програми вручну і приходиться застосовувати симулятори чи апаратні емулятори.

Слід зазначити, що такі загальні способи порівняння продуктивності не слід використовувати для всіх процесорів і мікроконтролерів, у яких реалізуються ці дві архітектури. Порівняння краще проводити стосовно до конкретної програми. Різні архітектури і пристрої мають свої специфічні особливості, що дозволяють щонайкраще реалізувати ті чи інші програми. У деяких випадках конкретна програма може бути виконана тільки з використанням визначеної архітектури і специфічних особливостей мікроконтролера.

1.6 Типи пам’яті мікроконтролерів

Можна виділити три основних види пам’яті, що використовуються в мікроконтролерах. **Пам’ять програм** являє собою постійну пам’ять, призначену для збереження програмного коду і констант. Ця пам’ять не змінює свого вмісту в процесі виконання програми. **Пам’ять даних** призначена для збереження змінних у ході виконання програми. **Регістри мікроконтролера** - цей вид пам’яті включає внутрішні регістри процесора і регістри, що служать для управління периферійними пристроями.

Пам’ять програм

Для збереження програм звичайно служить один з видів постійної пам’яті: PROM (однократно-програмовне ПЗУ), EPROM (електрично-програмовне ПЗУ з ультрафіолетовим стиранням), EEPROM (ПЗУ з електричним записом і стиранням, до цього виду відносяться також сучасні мікросхеми Flash-пам’яті) чи ROM (багатократно-програмовне ПЗУ). Усі ці види пам’яті є енергонезалежними — це означає, що вміст пам’яті зберігається після вимикання живлення мікроконтролера. Така пам’ять необхідна, тому що мікроконтролер не містить яких-небудь пристроїв масової пам’яті (магнітних дисків), з яких завантажується програма в комп’ютерах. Програма постійно зберігається в мікроконтролері.

У процесі виконання програма зчитується з цієї пам’яті, а блок управління (дешифратор команд) забезпечує її декодування і виконання необхідних операцій. Вміст пам’яті програм не може змінюватися (перепрог-

рамовуватися) під час виконання програми. Тому функціональне призначення мікроконтролера не може змінитися, поки вміст його пам'яті програм не буде стерте (якщо це можливо) і перепрограмоване (заповнене новими командами).

Варто звернути увагу, що розрядність мікроконтролера (8, 16 чи 32 біт) указується відповідно до розрядності його шини даних. У Гарвардській архітектурі команди можуть мати більшу розрядність, ніж дані, щоб дати можливість зчитувати за один такт цілу команду. Наприклад, мікроконтролери PIC у залежності від моделі використовують команди з розрядністю 12, 14 чи 16 біт. У мікроконтролерах AVR команда завжди має розрядність 16 біт. Однак усі ці мікроконтролери мають шину даних розрядністю 8 біт.

У пристроях із Прінстонською архітектурою розрядність даних, звичайно, визначає розрядність (число ліній) використовуваної шини. У мікроконтролерах Motorola 68HC05 24-розрядна команда розміщується в трьох 8-розрядних комірках пам'яті програм. Для повної вибірки такої команди необхідно зробити три цикли зчитування цієї пам'яті.

Коли говориться, що пристрій є 8-розрядним, це означає розрядність даних, що здатний обробляти мікроконтролер.

Пам'ять ROM (ПЗУ) використовується тоді, коли програмний код заноситься в мікроконтролер на етапі його виробництва. Попередньо програма налагоджується і тестується, після чого передається фірмі-виробнику, де програма перетворюється на рисунок маски на скляному фотошаблоні. Отриманий фотошаблон з маскою використовується в процесі створення з'єднань між елементами, з яких складається пам'ять програм. Тому таку пам'ять часто називають багатократно-програмовна ROM.

ROM є самим дешевим типом постійної пам'яті для масового виробництва. Однак вона має ряд істотних недоліків, що привели до того, що в останні роки цей тип пам'яті майже не використовується. Основними недоліками є значні витрати засобів та часу на створення нового комплекту фотошаблонів і їхнє впровадження у виробництво. Звичайно, такий процес займає біля десяти тижнів і є економічно вигідним при випуску десятків тисяч приладів. Тільки при таких обсягах виробництва забезпечується перевага ROM у порівнянні з E(E)PROM. Існує також обмеження, пов'язане з можливістю використання таких мікроконтролерів тільки у визначеній сфері застосування, тому що його програма забезпечує виконання жорстко фіксованої послідовності операцій, і не може бути використана для вирішення яких-небудь інших задач.

Електрично-програмовна пам'ять EPROM складається з комірок, що програмуються електричними сигналами і витираються за допомогою ультрафіолетового світла. Пам'ять PROM може бути запрограмована тільки один раз. Ця пам'ять, звичайно, містить плавкі перемички, що перегорають під час програмування. В наш час така пам'ять використовується дуже рідко.

Комірка пам’яті EPROM являє собою MOS-транзистор із плаваючим затвором, що оточений діоксидом кремнію (SiO_2). Стек транзистора з’єднаний з «землею», а джерело підімкнене до напруги живлення за допомогою резистора. У стертому стані (до запису) плаваючий затвор не містить заряду, і MOS-транзистор закритий. У цьому випадку на джерелі підтримується високий потенціал, і при звертанні до комірки зчитується логічна одиниця. Програмування пам’яті зводиться до запису у відповідні комірки логічних нулів.

Програмування здійснюється шляхом подачі на керувальний затвор високої напруги (рис 1.8). Цієї напруги повинно бути досить, щоб забезпечити пробій між керувальним і плаваючим затворами, після чого заряд з керуючого затвора переноситься на плаваючий. MOS-транзистор перемикається у відкритий стан, закорочуючи джерело з землею. У цьому випадку при звертанні до комірки зчитується логічний нуль.

Щоб стерти вміст комірки, він висвітлюється ультрафіолетовим світлом, що дає заряду на плаваючому затворі достатню енергію, щоб він міг залишити затвор. Цей процес може займати від декількох секунд до декількох хвилин.

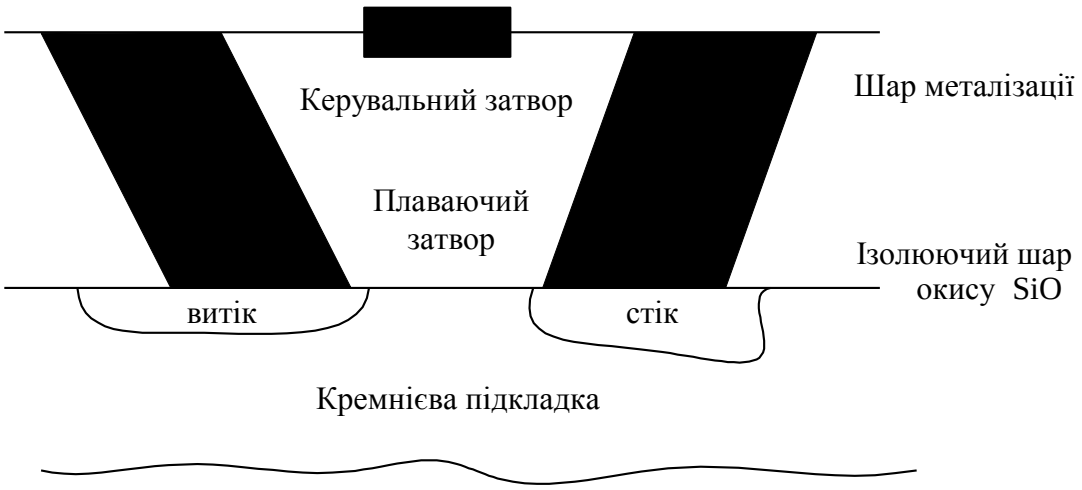


Рисунок 1.8 – Комірка пам`яті EPROM

Звичайно мікросхеми EPROM виготовляються в керамічному корпусі з кварцевим віконцем для доступу ультрафіолетового світла. Такий корпус досить дорогий, що значно збільшує вартість мікросхеми. Для зменшення ціни мікросхеми EPROM укладають у корпус без віконця (версія EPROM з однократним програмуванням). Зменшення вартості при використанні таких корпусів може бути настільки значним, що ці версії EPROM у даний час частіше використовуються замість масово-програмувального ROM.

Раніше мікроконтролери програмувалися тільки за допомогою паралельних протоколів, досить складних для реалізації. В наш час протоколи

програмування сучасної EPROM і EEPROM пам'яті істотно змінилися, що дозволило виконувати програмування мікроконтролера безпосередньо в складі системі, де він працює. Такий спосіб програмування одержав назву «in-system programming» чи «ISP». ISP-мікроконтролери можуть бути запрограмовані після того, як їх припаяли на плату. При цьому скорочуються витрати на програмування, тому що немає необхідності у використанні спеціального устаткування – програматорів.

Пам'ять **EEPROM** (Electricalle Erasable Programmable Memory – програмовна пам'ять, що стирається електрично) можна вважати новим поколінням EPROM пам'яті. У такій пам'яті комірка стирається не ультрафіолетовим світлом, а шляхом електричного з'єднання плаваючого затвора з «землею». Використання EEPROM дозволяє стирати і програмувати мікроконтролер, не знімаючи його з плати. Таким способом можна періодично обновляти його програмне забезпечення.

Пам'ять EEPROM більш дорога, ніж EPROM (у два рази дорожче за EPROM з однократним програмуванням), а також EEPROM працює набагато повільніше, ніж EPROM.

Основна перевага використання пам'яті EEPROM полягає в можливості її багаторазового перепрограмування без видалення з плати. Це дає величезний виграш на початкових етапах розробки систем на базі мікроконтролерів чи у процесі їхнього вивчення, коли багато часу іде на багаторазовий пошук причин нероботоздатності системи і виконання наступних циклів стирання-програмування пам'яті програм.

Функціонально Flash-пам'ять мало відрізняється від EEPROM. Основні відмінності полягають в способі стирання записаної інформації. У пам'яті EEPROM стирання відбувається окремо для кожної комірки, а в Flash-пам'яті стирання здійснюється цілими блоками. Якщо треба змінити вміст однієї комірки Flash-пам'яті, то потрібно перепрограмувати цілий блок (чи всю мікросхему). У мікроконтролерах з пам'яттю EEPROM можна змінювати окремі ділянки програми без необхідності перепрограмувати весь пристрій.

Часто вказується, що мікроконтролер має Flash-пам'ять, хоча насправді він містить EEPROM. В наш час між цими типами пам'яті є небагато відмінностей, тому деякі виробники використовують ці терміни як еквівалентні.

Пам'ять даних

При першому знайомстві з описом мікроконтролера багатьох здивує малий обсяг їхньої оперативної пам'яті даних RAM, що, звичайно, складає десятки чи сотні байтів. Якщо мікроконтролер використовує для збереження даних пам'ять EEPROM, то її обсяг також не перевищує декількох десятків байтів.

Якщо писати програми для персонального комп'ютера (PC), то, імовірно, виникає питання, що можна зробити з таким маленьким обсягом

пам’яті. Імовірно, ці програми для PC містять змінні, обсяг яких вимірюється в кілобайтах, не враховуючи використовувані масиви даних. При використанні масивів необхідний обсяг пам’яті може складати сотні кілобайтів. Так що ж можна зробити, маючи обсяг ОЗП порядку 25 байт?

Справа в тому, що програмування для мікроконтролера виконується за іншими правилами, ніж програмування PC. Застосовуючи деякі нескладні правила можна вирішувати багато задач з використанням невеликого обсягу пам’яті RAM. При програмуванні мікроконтролерів константи, якщо це можливо, не зберігаються як змінні. Максимально використовуються апаратні можливості мікроконтролерів (такі, як таймери, індексні регістри), щоб, якщо можна, обмежити розташування даних у RAM. Це означає, що при розробці прикладних програм необхідно попередньо подбати про розподіл ресурсів пам’яті. Прикладні програми повинні орієнтуватися на роботу без використання великих масивів даних.

Стек

У мікроконтролерах RAM використовується для організації виклику підпрограм і обробки переривань. При цих операціях вміст програмного лічильника й основних регістрів (акумулятор, регістр стану, індексні регістри і т.д.) зберігається і потім відновлюється при поверненні до основної програми.

Стек — це електронна структура даних, що функціонує аналогічно до своєї фізичної копії — стосу паперів. Коли що-небудь поміщається в стек, то воно залишається там доти, поки не буде вийняте назад. Уявіть різнобарвні аркуші паперу, що укладаються в стос один на одний. Коли аркуші віддаляються, то відбувається їхнє переміщення в зворотному порядку. З цієї причини, стек часто називають чергою типу LIFO (Last In, First Out) — «останнім прийшов – першим вийшов».

У Прінстонській архітектурі RAM використовується для реалізації безлічі апаратних функцій, включаючи функції стека. При цьому знижується продуктивність пристрою, тому що для доступу до різних видів пам’яті необхідні багаторазові звертання, що не можуть виконуватися одночасно. З цієї ж причини Прінстонська архітектура, зазвичай, вимагає більшої кількості тактів на виконання команди, ніж Гарвардська.

Процесори Гарвардської архітектури можуть мати три області пам’яті, що адресуються паралельно (в один і той же час): пам’ять програм, пам’ять даних, що включає простір введення-виведення, і стек.

У Гарвардській архітектурі стекові операції можуть реалізовуватись в пам’яті, спеціально виділеній для цієї мети. Це означає, що при виконанні команди виклику підпрограми «call» процесор з Гарвардською архітектурою виконує кілька дій одночасно. У Прінстонській архітектурі при виконанні команди «call» наступна команда вибирається після того, як у стек буде поміщений вміст програмного лічильника.

Необхідно пам’ятати, що мікроконтролери обох архітектур мають обмежений об’єм пам’яті для збереження даних. Перевищення цієї межі може викликати проблеми при виконанні програми.

Якщо в процесорі виділений окремий стек, і обсяг записаних у нього даних перевищує його ємність, то відбувається циклічна зміна вмісту покажчика стека, і покажчик стека починає посилатися на раніше заповнену комірку стека. Це означає, що після занадто великої кількості команд «call» у стеку з’явиться неправильна адреса повернення, що була записана замість правильної адреси. Якщо мікропроцесор використовує загальну область пам’яті для розташування даних і стека, то існує небезпека, що при переповненні стека відбудеться запис в область даних, або буде зроблена спроба записування даних, що завантажуються в стек, в область ROM.

Тепер розглянемо можливості збереження в стеку вмісту регістрів. У деяких архітектурах немає команд, що виконують завантаження вмісту регістрів у стек «push» і вивантаження зі стека «pop». З чотирьох сімейств мікроконтролерів, описаних у цій книзі, тільки два мають такі команди. Однак команди «push» і «pop» можуть бути легко реалізовані за допомогою індексного регістра, що вказує на область стека. При цьому замість кожної з команд «push» і «pop» використовуються дві команди, зазначені нижче:

`Push`	Завантаження даних у стек
`move [index], acc`	Зберегти вміст акумулятора в стеку
`decrement index`	Перейти до наступної комірки стека
`Pop`	Вивантажити дані зі стека
`increment index`	Перейти до попередньої комірки
`move acc, [index]`	Помістити вміст стека в акумулятор

Звичайно, таке рішення є менш ефективним, ніж використання спеціальних команд «push» і «pop», а використовуваний індексний регістр може знадобитися для інших цілей. Однак це рішення забезпечує імітацію стека при використанні процесорів, у яких такі команди відсутні.

Існує ще одна проблема з наведеними вище прикладами. Що станеться, якщо відбудеться переривання між першою і другою командою, що імітують операції «push» і «pop»? Якщо програма обробки переривання використовує стек, то записані в ньому дані будуть загублені. Для запобігання цього можна заборонити переривання перед виконанням цих команд чи переставити їх у такому порядку:

`Push`	Завантажити дані в стек
`decrement index`	Перейти до наступної комірки стека
`move [index], aci`	Зберегти вміст акумулятора в стеку
`Pop`	Вивантажити дані зі стеку
`move aci,[index]`	Помістити значення стека в акумулятор

```
increment index          Перейти до попередньої комірки стека
```

Якщо після першої команди програма буде перервана, то після виконання обробки переривання вміст стека не буде втрачено.

1.7 Регістри мікроконтролера. Простір введення-виведення

Подібно до всіх комп'ютерних систем, мікроконтролери мають безліч регістрів, що використовуються для управління різними пристроями, підімкненими до процесора. Це можуть бути регістри процесора (акумулятор, регістри стану, індексні регістри), регістри управління (регістри управління перериваннями, регістри управління таймером) чи регістри, що забезпечують введення-виведення даних (регістри даних і регістри управління паралельним, послідовним чи аналоговим введенням-виведенням). Звертання до цих регістрів може відбуватися різними способами.

Реалізовані мікроконтролером способи звертання до регістрів впливають на їхню продуктивність. Тому дуже важливо зрозуміти, як відбувається звертання до регістрів, щоб писати ефективні прикладні програми для мікроконтролерів. У процесорах з RISC-архітектурою всі регістри (часто й акумулятор) розташовуються за адресами, що явно задаються. Це забезпечує більш високу гнучкість при роботі процесора.

Розглянемо приклад процедури розгалуження програми за умови, що визначений біт у регістрі порту введення-виведення встановлений у 1. Для CISC-процесора відповідний псевдокод буде виглядати в такий спосіб:

```
Accumulator = IOPort;                    Завантажити вміст регістра
                                         IOPort в акумулятор
Accumulator = Accumulator&(1«Bit);       Маскувати всі біти акумулято-
                                         ра, крім Bit ;
if ZeroFlag != 0 goto Address;           Якщо Zero, то біт Bit=1
```

Ця процедура буде компілюватися в послідовність операцій:

```
if IQPort.Bit == 1 goto Address;
```

Асемблерний запис для мікроконтролерів Microchip PIC:

```
btfsc IOPort, Bit        Пропустити наступну команду,
                         якщо біт
Bit=0
goto Address
```

Більш ефективно ця процедура реалізується в мікроконтролері Intel 8051:

```
jb IOPort.Bit, Address   Перейти, якщо біт Bit=1
```

Використовуючи процесор, що може безпосередньо звертатися до будь-якого регістра, можна одержати переваги при розробці простих прикладних програм. Наприклад, у мікроконтролері PIC вміст акумулятора і

регістра стану не змінюється при передачі управління в залежності від значення біта в регістрі порту IOPort.

Одним з важливих питань є розміщення регістрів в адресному просторі. У деяких процесорах усі регістри і RAM розташовуються в одному адресному просторі. Це означає, що пам’ять з’єднана з регістрами. Такий підхід називається «відображенням пристроїв введення-виведення на пам’ять».

В інших процесорах адресний простір для пристроїв введення-виведення відділено від загального простору пам’яті. Основна перевага розміщення регістрів введення-виведення в окремому просторі адрес полягає в тому, що при цьому спрощується схема підімкнення пам’яті програм і даних до загальної шини. Пристрій введення-виведення, звичайно, займає маленький блок адрес, що робить незручним декодування їхньої адреси разом з великими блоками основної пам’яті. Окремий простір введення-виведення дає деякі переваги процесорам з Гарвардською архітектурою, забезпечуючи можливість зчитувати команду під час звертання до регістра введення-виведення.

1.8 Зовнішня пам’ять

Незважаючи на величезні переваги використання внутрішньої вбудованої пам’яті, у деяких випадках необхідне під’єднання до мікроконтролера додаткової зовнішньої пам’яті (як пам’яті програм, так і даних).

Існує два основних способи підключення зовнішньої пам’яті. Перший спосіб — під’єднання зовнішньої пам’яті до мікроконтролера, як до мікропроцесора. Багато мікроконтролерів містять спеціальні апаратні засоби для такого під’єднання. Другий спосіб полягає в тому, щоб під’єднати пам’ять до пристроїв введення-виведення і реалізувати звертання до пам’яті через ці пристрої програмними засобами. Такий спосіб дозволяє використовувати прості пристрої введення-виведення без реалізації складних шинних інтерфейсів. Вибір найкращого з цих способів залежить від конкретного використання мікроконтролера.

2 АПАРАТНІ ЗАСОБИ МІКРОКОНТРОЛЕРІВ

У попередньому розділі дано введення в мікроконтролери, коротко розглянуті їхні архітектури, реалізація звертання до програм, даних і пристроїв введення-виведення. У цьому розділі обговорюється внутрішня структура мікроконтролерів і їхній інтерфейс із зовнішніми пристроями.

При ознайомленні зі змістом розділу можна зрозуміти, що будуть розглянуті тільки деякі аспекти реалізації інтерфейсу. Багато пристроїв використовують інтерфейси і протоколи, що трохи відрізняються від описаних нижче, або інтерфейси, що тут взагалі не розглядаються, це зумовлено великою різноманітністю мікропроцесорної техніки.

2.1 Корпуси пристроїв

Коли розглядається «монтаж приладу в корпус», то буде описуватися матеріал (герметик), що використовується для захисту кристала, і технологія з'єднання кристала з провідниками на друкованій платі. Від вибору того чи іншого корпусу залежить ціна, розмір і якість кінцевого виробу. Для захисту кристалів використовуються два основних матеріали: пластмаса і кераміка.

Найбільш розповсюдженими є пластмасові корпуси. Кристал, поміщений у такий корпус, з'єднується з зовнішніми виводами за допомогою тонких алюмінієвих дротиків, що приварюються до кристала з використанням ультразвуку. Деякі кристали приєднуються до зовнішніх виводів за допомогою технології «С4», що буде описана пізніше. Для заливання пластикових корпусів використовується епоксидний герметик. Коли герметик затвердіє, кристал стає захищеним від світла, вологи і механічного впливу.

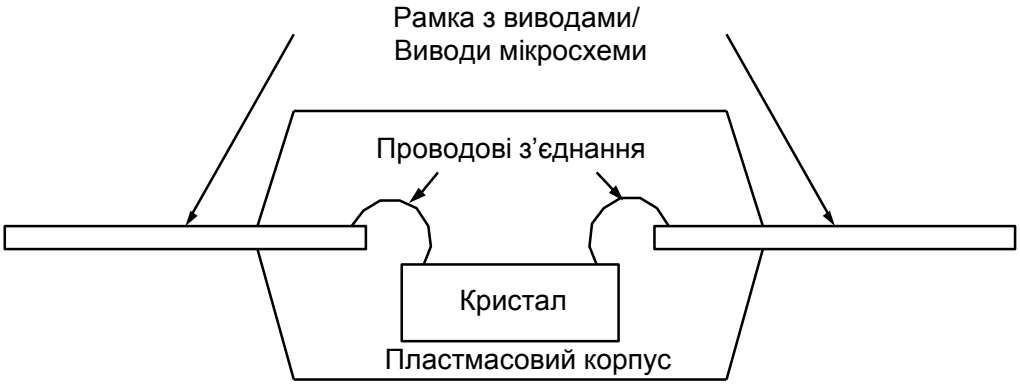


Рисунок 2.1 – Корпус для однократно програмовних пристроїв

Мікроконтролери з пам'яттю EPROM, поміщені в пластмасовий корпус, звичайно, називаються однократно програмовні (OTP - One-Time Programmable). Такий мікроконтролер може бути запрограмований тільки один раз, тому що пластиковий корпус не дозволяє робити стирання вмісту

пам’яті програм типу EPROM шляхом висвітлення кристала ультрафіолетом (рис. 2.1).

Одна з основних причин розміщення мікроконтролера в керамічному корпусі — це можливість створення в ньому кварцового віконця для стирання EPROM (рис. 2.2). Коли використовується керамічний корпус, кристал приклеюється до його нижньої половини і з’єднується провідниками з рамкою, на якій розташовані зовнішні виводи. Керамічні корпуси, звичайно, випускаються зі штировими виводами, що вставляються в наскрізні металізовані отвори на друкованій платі (монтаж за технологією PTH – Plated - Through Hole). Для пластмасових корпусів можливий більш широкий вибір варіантів монтажу на платі. Керамічний корпус може значно збільшити вартість окремої мікросхеми, оскільки він більш ніж у 10 разів дорожче пластмасового. Тому мікроконтролери в керамічних корпусах використовуються звичайно під час налагодження розроблювальних систем, коли додаткові витрати через застосування корпусів із кварцовим віконцем є виправданими.

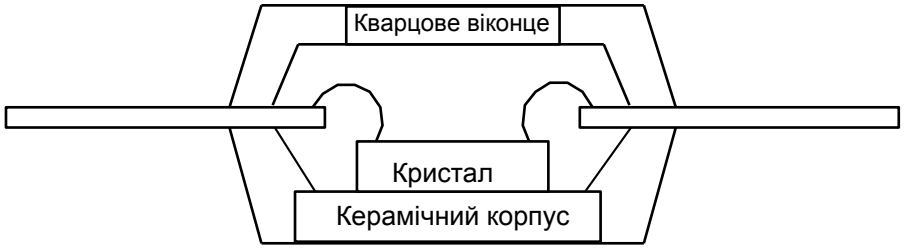


Рисунок 2.2 – Керамічний корпус із кварцевим віконцем

Технологія встановлення мікросхеми на друковану плату істотно змінилася за останні роки. У 80-х роках практично всі мікросхеми випускалися зі штировими виводами, що запаювалися в отвори на друкованій платі (рис. 2.3).

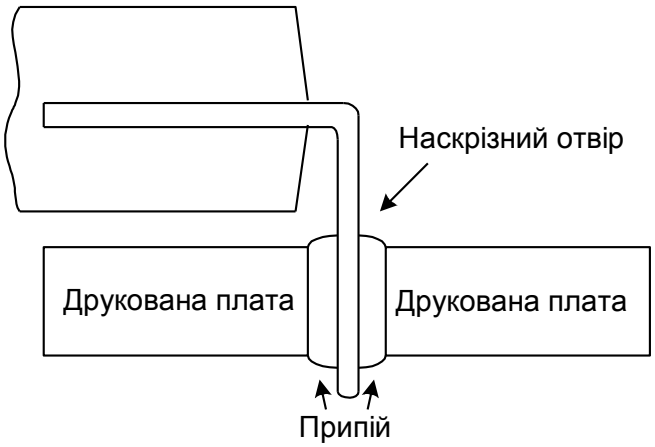


Рисунок 2.3 – Наскрізний отвір

Перевагою цієї технології монтажу (PTH-технологія) є її простота — для виробництва таких плат не потрібно складне устаткування і спеціальна підготовка. Недоліком є те, що отвір займає на платі значну площу, і відстань між сусідніми виводами мікросхеми повинна бути істотно більша, ніж при використанні технології поверхневого монтажу (SMT - Surface Mount Technology), коли виводи мікросхеми припаюються до поверхні плати (рис. 2.4).

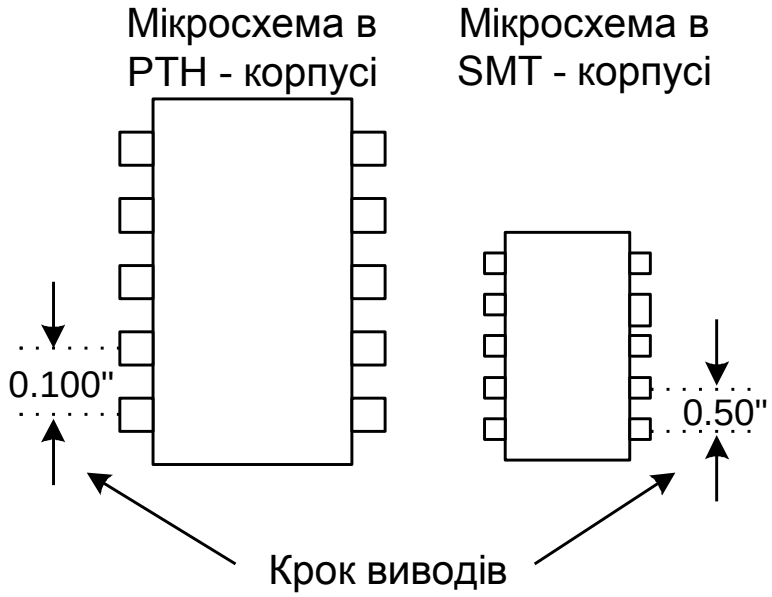


Рисунок 2.4 – Порівняльні розміри PTH- та SMT- корпусів

Для SMT-монтажу використовуються корпуси з двома основними типами виводів (рис. 2.5): типу «gull wing» і типу «J». Обидва типи виводів мають свої переваги. Корпуса з виводами типу «gull wing» дозволяють робити ручну пайку і забезпечують більш простий контроль паяних з'єднань. Застосування корпусів з виводами типу «J» зменшує площу друкованої плати. В наш час корпуси з виводами «gull wing» більш популярні, тому що їхнє використання дозволяє застосовувати більш просте виробниче устаткування і забезпечити перехід до дуже щільного монтажу, коли відстані між центрами виводів зменшуються до 0,41 мм.

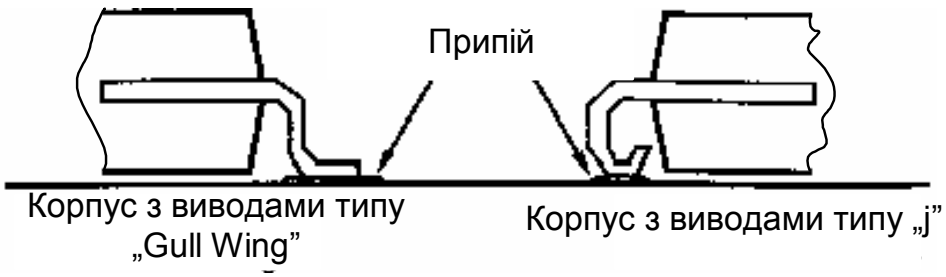


Рисунок 2.5 – Планарна технологія корпусів

Зменшення розміру корпусу і кроку розташування виводів дозволяє значно збільшити щільність упакування компонентів на платі, що визначається кількістю корпусів, розташовуваних на одиниці площі. Типове значення кроку розташування виводів для PTH-корпусів складає 2,54 мм, тоді як крок для SMT-корпусів від 1,27 мм до 0,41 мм. SMT-корпуси з малим кроком виводів відомі за назвою «fine pitch».

Щоб оцінити як вибір корпусу впливає на щільність упакування компонентів на платі, розглянемо приклад використання PTH-корпусу з кроком виводів 2,5 мм і SMT-корпусу з кроком 1,27 мм. SMT-корпус у два рази менше за усіма вимірами — це значить, що на місці одного PTH-корпусу можна розмістити чотири SMT-корпуси. Крім того, при відсутності отворів компоненти можуть розміщатися по обидва боки друкованої плати. У результаті одержуємо збільшення щільності упакування у вісім раз.

Щоб підвищити щільність упакування компонентів і збільшити число виводів мікросхеми, крок виводів для SMT-корпусів був зменшений до 0,5 мм. Але в процесі розробки корпусів з великим числом виводів були запропоновані нові технології, що дозволяють спростити монтаж друкованих плат.

Для виробника встановлення і зняття SMT-корпусів реалізується простіше, ніж PTH-корпусів. На посадкові місця наноситься спеціальна суміш припою і флюсу, що називається паяльною пастою. Потім плата поміщується в піч, де паста розплавляється і припаює мікросхему до плати. Щоб зняти компонент із плати, його виводи обдуваються гарячим повітрям чи азотом, що розплавляє припой, після чого компонент може бути вийнятий. У той час як виробнику, що має спеціальне устаткування, легше працювати з SMT-корпусами, для розробника чи радіоаматора це набагато складніше (особливо якщо мікросхему потрібно зняти з плати для перепрограмування).

Для мікросхем з великою кількістю виводів (більш 300) PTH-корпуси, звичайно, не використовуються через їхній великий розмір. З SMT-корпусами також виникають проблеми, тому що важко забезпечити паралельність усіх виводів, порушення якої викликає брак при монтажі мікросхем на плату. Для монтажу кристалів з дуже великою кількістю виводів використовуються технології кулькових виводів — BGA (Ball Grid Array), безпосереднього монтажу кристалів на плату - COB (Chip On Board) і автоматичного монтажу на стрічку — TAP (Tapi Automated Bonding).

У технології BGA для приєднання мікросхеми до плати використовується двовимірна матриця кульок припою, розташованих на нижній стороні корпусу (рис. 2.6). Для мікросхем з великим числом виводів BGA-корпуси дають значні переваги в порівнянні з традиційними SMT-корпусами. Розглянемо, як приклад, 304-вивідну мікросхему. SMT-корпус являє собою плоский прямокутний корпус QFP(Quad Flat Pack), по чоти-

рьох сторонах якого розташовані виводи з кроком 0,52 мм. BGA-корпус містить матрицю 16×19 кулькових виводів із кроком 1,27 мм. Мінімальний розмір кожної сторони SMT-корпусу складе 3,9 см (площа корпусу 15,4 см^2), тоді як розмір BGA-корпусу складе 2,16×2,54 см (5,5 см^2). У розглянутому прикладі BGA-корпус займає в три рази меншу площу, ніж QFP. Крім того BGA-корпус простіше встановлювати і паяти на плату, тому що він має більший крок виводів.

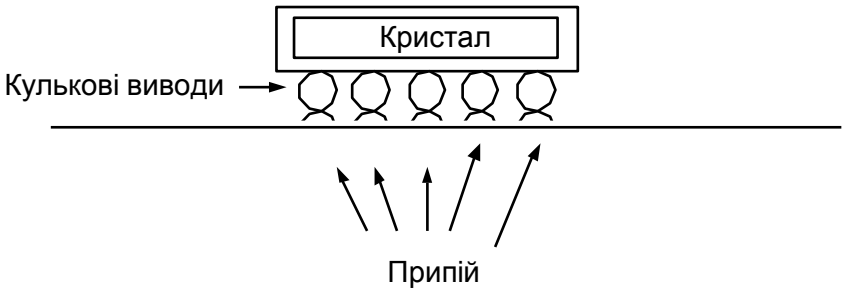


Рисунок 2.6 – Корпус типу BGA

Виводи QFP-корпусу дуже ніжні, тому що при кроці 0,52 мм їхній діаметр складає усього близько 0,3 мм. Кулькові виводи BGA-корпусу значно більш тверді. Через крихкість виводів встановлення і пайка SMT-корпусів, звичайно, здійснюється цілком автоматично, без участі людини. При монтажі BGA-корпусів немає необхідності дотримуватись такої обережності.

Встановлення BGA-корпусів на плату і їхній демонтаж вимагають того ж устаткування, що і SMT-корпуси. Однак для контролю якості монтажу плат з BGA-корпусами потрібна більш складна апаратура, включаючи рентгенівські установки. BGA-корпуси мають більш високу якість: відсоток браку для SMT-корпусів складає 0,002-0,005, а для BGA-корпусів - 0,0001-0,0002 і менше.

При використанні технології COB (Chip On Board) кристал безпосередньо монтується на плату. В наш час застосовуються два способи кріплення кристалів.

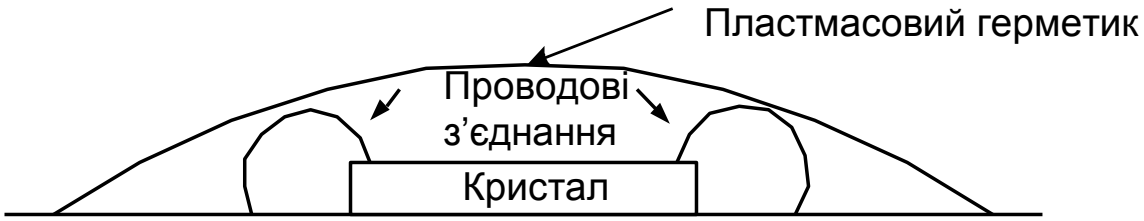


Рисунок 2.7 – Корпус типу COB

При першому способі кристал розміщується на платі, і контактні площадки на кристалі і платі з'єднуються в такий же спосіб, як усередині корпусу мікросхеми — за допомогою тонких алюмінієвих дротиків, що

приварюються ультразвуком (рис. 2.7). Сам кристал може бути приклеєний чи припаяний до плати. Припаювання кристала використовується тоді, коли плата служить у якості тепловідводу.

Другий спосіб відомий за назвою «технологія С4» і фактично дуже схожий на BGA-процес, описаний раніше. Кулькові виводи, що використовуються в цьому процесі, називаються «опуклостями» (bumps), тому що вони набагато менше BGA-кульок (рис. 2.8). Спочатку ця технологія була розроблена IBM для монтажу кристалів у керамічних корпусах без використання з'єднувальних дротиків.

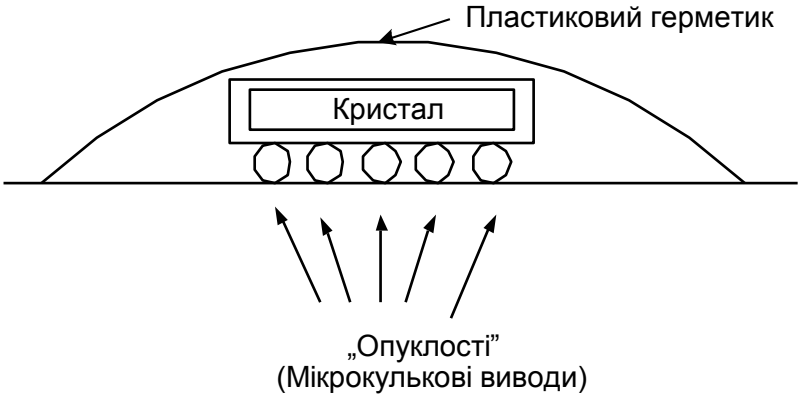


Рисунок 2.8 – Монтаж на плату із використанням технології С4

Технологія С4 вимагає істотних інвестицій у створення устаткування для монтажу і розробки спеціальних технологічних процесів. Через малу відстань між кристалом і платою там може залишитися вода, використовувана при промиванні плати, що приводить до зниження надійності виробів. В наш час технологія С4 знаходиться на експериментальній стадії. Причинами цього є труднощі в забезпеченні надійного встановлення кристала на плату і можливість відмов через спотворення кулькових виводів, що виникає внаслідок різниці коефіцієнтів температурного розширення кристала і плати.

Технологія COB є найкращою для створення апаратури з високою щільністю монтажу, коли є мало місця для розміщення мікросхем (наприклад, телефонна смарт-карта) чи потрібно спеціальний тепловідвід.

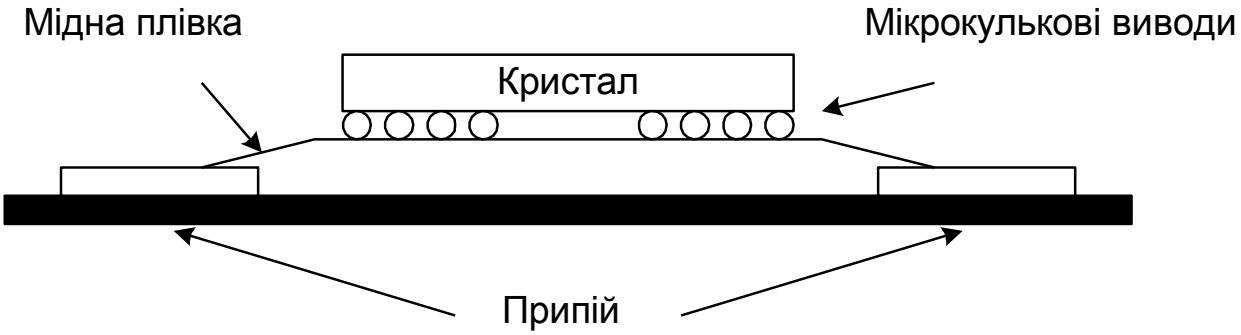


Рисунок 2.9 – Корпус типу TAP

Є ще одна технологія монтажу, що являє собою комбінацію SMT і COB. Це автоматичний монтаж на стрічку – TAB (Tape Automated Bonding). У даному випадку контактні площадки кристала приварюються до мідної стрічки з ізолювальним покриттям, на якій шляхом штампування створена рамка з виводами. Ці виводи потім припаюються до металевих провідників на друкованій платі (рис. 2.9.)

TAB-технологія вперше з'явилися в середині 1980 року як спосіб складання мікросхем з великим числом виводів. З удосконаленням SMT-корпусів і винаходом BGA технологія TAB значною мірою застаріла, хоча ще використовується деякими виробниками. Складність TAB-технології полягає в необхідності застосування спеціалізованого автоматичного устаткування для встановлення і припаювання TAB-компонентів, а також у труднощах контролю паяних з'єднань.

Від обраного способу монтажу залежить розмір виробу, його ціна, якість паяних з'єднань, а також вибір виробника, що зможе виконати складання продукту. У даному розділі питання монтажу електронних виробів розглянуті дуже коротко. Коли буде вибиратися спосіб монтажу для виробу, треба уважно проаналізувати всі можливості і проконсультуватися з інженерами підприємства, де планується організувати випуск. Вони є фахівцями в цій області і знають, яку технологію монтажу найкраще використовувати.

2.2 Технологія виготовлення кристалів

Як і інші електронні прилади, мікроконтролери стають усе меншими, швидкодійнішими, дешевшими і споживають менше потужності. Основною причиною поліпшення їхніх характеристик є удосконалення виробничих процесів і використовуваних технологій, а не використання нових комп'ютерних архітектур.

В наш час практично всі мікроконтролери випускаються за CMOS-технологіями (Complementary Metal Oxide Semiconductor – транзистори зі структурою метал-оксид-напівпровідник). У мікросхемах, що випускаються за цією технологією спільно використовуються p-канальні (PMOS) і n-канальні (NMOS) MOS-транзистори (рис. 2.10).

На рис. 2.10 дана схема CMOS-інвертора (логічного елемента «НЕ»). Коли на вхід схеми надходить низький потенціал, PMOS-транзистор відкритий, а NMOS – закритий. При цьому на виході установиться високий потенціал (логічна "1"), дорівнює напрузі живлення U_{cc}. Якщо на вхід поданий високий потенціал, то PMOS-транзистор буде закритий, а NMOS – відкритий, забезпечуючи на виході низький потенціал «землі» (логічний "0").

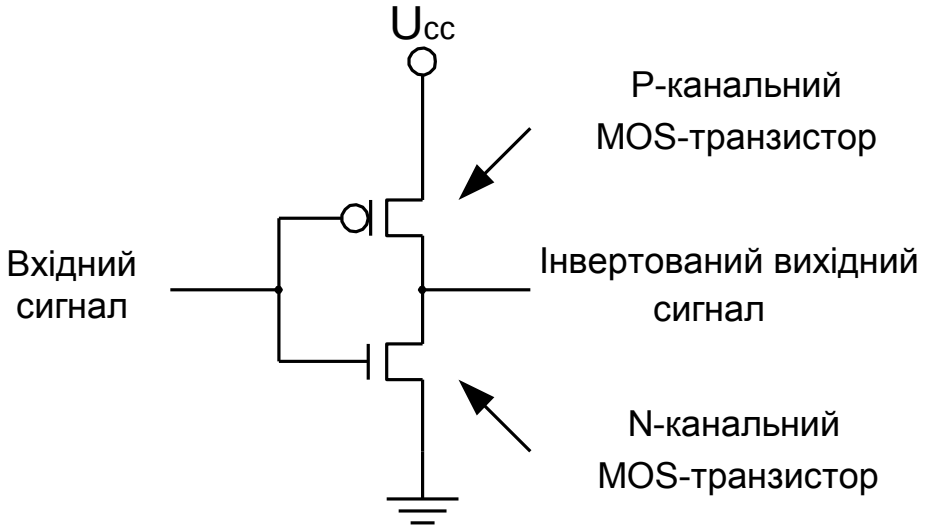


Рисунок 2.10 – Схема CMOS-інвертора

Коли елемент знаходиться в одному з цих станів, через транзистори протікає дуже маленький струм (струм витоку). Зі збільшенням частоти перемикання елемента зростає середня величина струму, що протікає, і споживана елементом потужність збільшується. Щоб скоротити енергоспоживання, доцільно експлуатувати CMOS-мікросхеми при мінімально можливій частоті перемикання. У «сплячому» режимі, коли частота перемикання знижується до нуля, споживання потужності значно скорочується, тому що перемикання елементів не відбувається, і струм у схемі не протікає.

Важливою характеристикою логічних елементів є поріг перемикання. Для CMOS-елементів значення порога перемикання, звичайно, складає від 1.4 В до $U_{cc}/2$. Різні логічні елементи можуть мати різні пороги перемикання. Перш ніж використовувати елемент у спроектованому пристрої необхідно переконатися, що його логічні рівні (“0” і “1”) і поріг перемикання відповідають рівням і порогам перемикання інших елементів пристрою.

CMOS-елементи можуть безпосередньо працювати з більшістю інших типів логічних елементів, що реалізують позитивну логіку. Варто тільки звернути увагу на можливість використання елементів зі зниженою напругою живлення, що мають більш низькі рівні логічної 1 (низьковольтна логіка). У цьому випадку необхідно, щоб рівень логічної 1 (високий рівень потенціалу) завжди був вище порога перемикання.

Існує багато особливостей, зв’язаних із застосуванням різних сімейств логічних елементів. Елементи цих сімейств виробляються за різними технологіями, тому вони мають специфічні характеристики, які необхідно враховувати при їхньому використанні.

Щоб показати як удосконалювалася технологія мікроконтролерів в останні роки, порівняємо дві мікросхеми. Перша – мікроконтролер 8748, який випускався в середині 80-х років (за функціями аналогічний контро-

леру клавіатури перших персональних комп'ютерів IBM PC), друга – сучасний мікроконтролер PIC 17C44. Обидві мікросхеми розміщаються в 40-вивідному корпусі з кроком 1,52 мм і мають приблизно однакову вартість.

Тип мікросхеми	8748H	PIC17C44
Технологія	HMOS	CMOS
Пам'ять EPROM	2 Кбайт	8 Кбайт
Пам'ять ROM	128 байт	454 байт
Максимальна частота	11 МГц	33 МГц
Струм живлення	100 мА	38 мА
Струм у «сплячому» режимі	не зазначений	1 мкА
Число ліній введення-виведення	27	33
Додаткові функції	8-бітний таймер синхронний/асинхронний послідовний порт	Три 16-бітних таймери синхронний/асинхронний послідовний порт Широтно-імпульсний модулятор

У багатьох задачах використовується батарейне живлення мікроконтролерів, а в деяких випадках застосовуються навіть конденсатори великої ємності, що забезпечують збереження роботоздатності при короткочасних вимкненнях живлення. Тому проблема зниження енергоспоживання дуже актуальна для мікроконтролерів.

Як зазначено вище, практично усі сучасні мікроконтролери виготовляються за CMOS-технологією, оскільки вони споживають значно менше потужності, ніж біполярні чи NMOS-мікроконтролери, які виготовлялися раніше.

Через велике різноманіття областей застосування і використовуваних при цьому джерел живлення ланцюг живлення мікроконтролерів проектується якомога надійнішим, щоб забезпечити їхнє нормальне функціонування в різних умовах.

Звернемо увагу на позначення, використовувані при описі живлення. Позначення «U_{cc}» і «U_{dd}» застосовуються для вказання напруги живлення (звичайно +5В, хоча можливі й інші значення). Аналогічно, позначення «V_{ss}» та «Gnd» використовуються для вказання «землі». У цій книзі використовується «U_{cc}» для позначення живлення (навіть коли виробник використовує «V_{dd}») і «Gnd» для позначення «землі».

2.3 Споживана потужність

При плануванні енергоспоживання для різних пристроїв, що використовують мікроконтролери, необхідно враховувати, що споживана ними потужність залежить від режиму функціонування. Є три значення потужності, споживаної мікроконтролером у різних робочих умовах. Перше — власна потужність, необхідна для нормальної роботи мікроконтролера. Друге – потужність, споживана пристроями введення-виведення, що потрібно враховувати, коли мікроконтролер робить обмін даними з зовнішніми пристроями. Третє – потужність, споживана в «сплячому» режимі, коли мікроконтролер очікує зовнішню подію, що перемикає його в робочий режим. В залежності від режиму роботи мікроконтролера, реалізований на його основі пристрій може при тому самому ресурсі енергії функціонувати протягом декількох годин чи декількох місяців.

Власна потужність – це потужність, що споживає мікроконтролер, коли до його виводів не підімкнені зовнішні пристрої. Значення цієї потужності залежить, головним чином, від струму споживаного при перемиканні CMOS-елементів, що є функцією швидкості роботи мікроконтролера.

Зі зменшенням тактової частоти власна потужність значно зменшується. Нижчезазначене споживання струму живлення для мікроконтролера PICMicro 16C73A на різних тактових частотах при напрузі живлення 5В.

Тактова частота	Струм живлення
1.0 МГц	550 мкА
2.0 МГц	750 мкА
3.0 МГц	1 мА
4.0 МГц	1.25 мА

Зі зниженням тактової частоти споживана потужність, що дорівнює добутку напруги живлення на струм, буде скорочуватися. Це означає, що програмне забезпечення повинно бути компактним, щоб збільшити термін роботи пристрою без зміни комплекту батарей, що може бути дуже важливим для деяких областей застосування.

Очевидно, що подальше зниження власної потужності можливо при зменшенні напруги живлення. Реалізація цієї можливості залежить від типу використовуваного мікроконтролера і підімкнених до нього пристроїв. Багато сучасних мікросхем можуть працювати при зниженні напруги живлення до 2 В.

Потужність, споживана у режимі введення-виведення, залежить від того, яку потужність мікроконтролер повинен затратити на управління роботою зовнішніх пристроїв. Значення цієї потужності визначається конкретним варіантом застосування мікроконтролера. У багатьох випадках мікроконтролер є єдиним активним пристроєм – наприклад, вводить вихідні дані з клавіатури і видає результати обробки на світлодіоди. Якщо мікроконтролер видає керувальні сигнали безупинно, навіть коли зовнішні при-

строї не вимагають обслуговування, то даний пристрій буде споживати більший струм, а значить і потужність, що необхідно для нормальної роботи.

Останнє, що залишилося обговорити – це споживання потужності в «сплячому» режимі. У цей режим мікроконтролер, звичайно, входить після виконання спеціальної команди. Тактовий генератор мікроконтролера при цьому зупиняється до настання деякої події, наприклад, надходження сигналу від сторожового таймера чи зміни стану визначеного входу.

Використання «сплячого» режиму може зменшити споживану потужність з рівня в декілька міліват до мікроват.

«Сплячий» режим — це віртуальний вимикач, реалізований усередині мікроконтролера. Його використання забезпечує ряд переваг. По-перше, такий віртуальний вимикач дешевий і надійний — при його вмиканні-вимиканні відмови виникають значно рідше, ніж при роботі електромеханічного вимикача. По-друге, у «сплячому» режимі зберігається вміст пам'яті даних RAM. «Сплячий» режим має один потенційний недолік, істотний для деяких програм: час, необхідний для виходу з «сплячого» режиму і запуску тактового генератора може досягати десяти мілісекунд. Дійсно, така затримка може виявитися занадто великою при взаємодії з іншою комп'ютерною системою. Однак, якщо основною задачею мікроконтролера є робота з людиною, то така затримка проблем не викликає.

При роботі мікроконтролера в «сплячому» режимі необхідно переконатися у відсутності струму, споживаного елементами навантаження, підімкнутими до його виводів. Вхідний струм мікроконтролера від світлодіоду, підімкненого до шини живлення, викликає збільшення споживання потужності в «сплячому» режимі.

2.4 Увімкнення живлення

Мікроконтролер часто застосовується в апаратурі, де його напруга живлення може виявитися нижче оптимальної. Як джерела живлення, можуть використовуватися батареї, випрямляч змінного струму з фільтрами, генератори постійного струму. Це означає, що мікроконтролерам приходиться працювати з різними значеннями вхідних сигналів і взаємодіяти з пристроями, що працюють з різними логічними рівнями. Тому мікроконтролери спеціально проектуються для використання в таких сурових умовах, що для багатьох інших електронних пристроїв є неприйнятними.

На щастя, більшість мікроконтролерів нормально працюють у широкому діапазоні зовнішніх умов. Єдине, на що варто звернути увагу при розробці проекту – це розв'язка напруги живлення. Як правило, для розв'язки використовується танталовий конденсатор ємністю 0,1 мкф, що вмикається якнайближче до виводів живлення. Цей конденсатор забезпечить підвищений вихідний струм при перехідних процесах, охороняючи апаратуру від хибних скидань і перекручувань даних. У такий спосіб просте увімкнення конденсатора звільнить від безлічі проблем.

2.5 Запуск (скидання в початковий стан)

При будь-якому застосуванні мікроконтролера важливо бути впевненим, що він працює в допустимих навколишніх умовах. Запуск мікроконтролера повинний мати місце тільки тоді, коли встановилася необхідна напруга живлення.

Як правило, пристрої, що використовують мікроконтролери, повинні починати роботу при увімкненні напруги живлення. Щоб бути впевненим, що запуск мікроконтролера відбудеться, коли напруга живлення досягне заданого стабільного значення, використовують схему, показану на рис. 2.11.

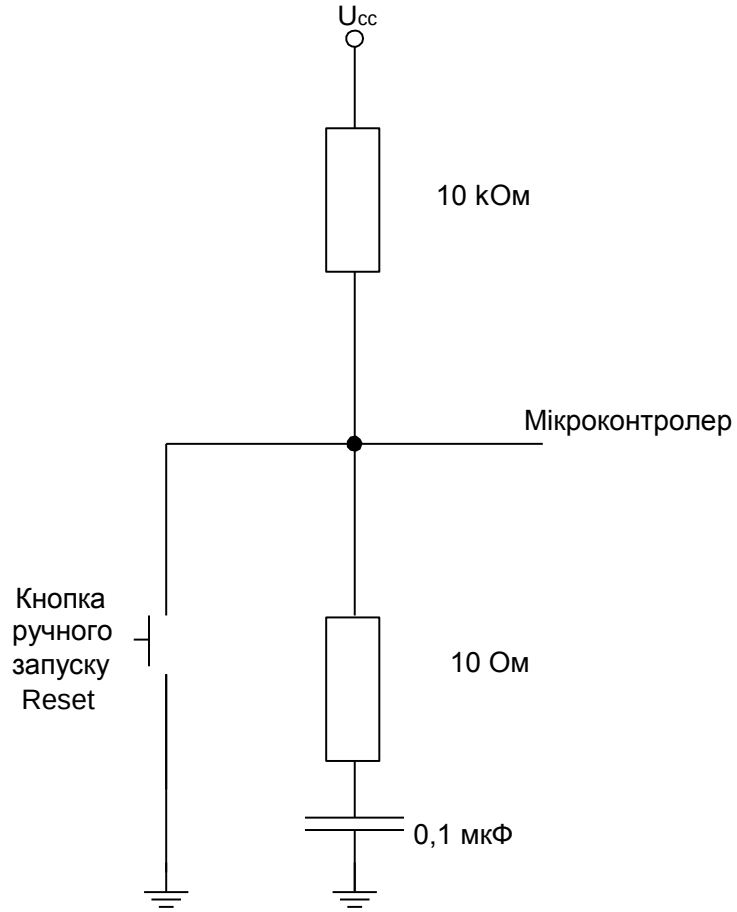


Рисунок 2.11 – Схема формування сигналу запуску RESET

У цій схемі сигнал RESET на вході мікроконтролера стає активним (приймає значення логічного “0”) приблизно через 22 мс (час затримки T_d =2,2 RC) після увімкнення живлення. Цього часу досить для стабілізації напруги живлення й установлення необхідної частоти тактового генератора перш, ніж почне працювати мікроконтролер.

Кнопка RESET використовується в процесі розробки пристрою для скидання мікроконтролера в початковий стан. При налагодженні пристрою дуже корисно мати можливість виконання скидання, щоб забезпечити по-

вторний запуск мікроконтролера. Резистор опором 100 Ом, що увімкнутий послідовно з конденсатором, служить для обмеження струму розряду конденсатора в момент скидання (заряджений конденсатор є джерелом великого струму, коли він закорочується на «землю»). Ця схема може бути використана для запуску мікроконтролерів, у яких сигнал RESET має високий активний рівень (наприклад, мікроконтролер 8051), шляхом інвертування напруги на конденсаторі (наприклад, за допомогою мікросхеми типу 7404).

Для деяких мікроконтролерів можна видалити RC-ланцюг у схемі запуску, тому що усередині них є схема, що забезпечує затримку увімкнення (пуск тактового генератора і початок виконання першої команди програми). У цьому випадку схема запуску може бути спрощена, як показано на рис. 2.12. Подивившись на цю схему можна подумати, що схему можна ще більш спростити, просто підімкнувши вивід RESET до шини живлення U_{cc}. Це правильно, але використовувати таке увімкнення слід тільки після того, як схема буде цілком налагоджена. Однак при цьому доцільно увімкнути струмообмежувальний резистор, щоб мати можливість повторного запуску шляхом закорочування виводу RESET на «землю».

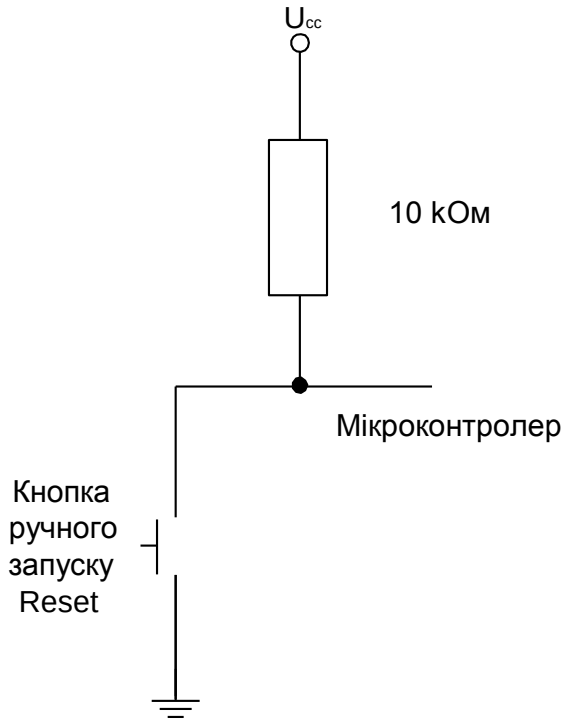


Рисунок 2.12 – Модифікована схема RESET

Ці схеми скидання найкраще використовувати у випадках, коли гарантована підтримка напруги живлення в робочому діапазоні. Для живлення багатьох пристроїв використовуються батареї, напруга яких спадає з часом. Зі спадом напруги можуть виникати порушення в роботі пристрою, тому що напруга вимкнення для одних приладів досягається раніше, ніж для інших.

Для вирішення цієї проблеми існують пристрої – монітори живлення, що стежать за рівнем напруги U_{cc}/U_{dd}. Якщо ця напруга спадає нижче визначеного рівня, (звичайно 4.5 В), то створюється сигнал RESET. Як правило, такі монітори живлення містять схему затримки і працюють аналогічно описаній вище RC-схемі запуску. Вони монтуються в такий же корпус, як трививідний транзистор.

2.6 Тактування системи

При проектуванні схеми тактових генераторів для мікропроцесорів, можна бути приємно здивованим, коли почати розробляти такі схеми для мікроконтролерів. Мікроконтролери побудовані так, що вимагають мінімального числа зовнішніх елементів для генерації тактових імпульсів.

Більшість мікроконтролерів здатні працювати в дуже широкому діапазоні частот: від нуля до десятків мегагерц. Це можливо завдяки використанню цілком статичної логіки. Деякі розробники використовують тактову частоту в 1 Гц і менше при налагодженні програмного забезпечен-ня. Широкий діапазон можливих робочих частот дозволяє розробнику краще настроїти мікроконтролер на виконання конкретних заданих функцій.

Існує три способи задання тактової частоти мікроконтролера, кожний з яких має свої переваги і недоліки. Перший спосіб – використання кварцового резонатора, увімкнутого відповідно до схеми на рис. 2.13. Цей спосіб дозволяє дуже точно задати тактову частоту мікроконтролера (розкид частот звичайно складає не більше 0,01%). Такий рівень точності потрібно для організації інтерфейсу мікроконтролера з іншими пристроями забезпечення точного ходу годин реального часу.

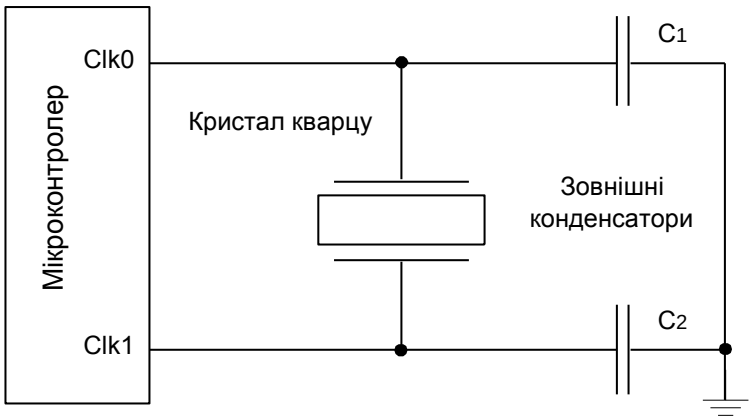


Рисунок 2.13 – Тактування з використанням кварцевого резонатора

Номінали ємностей конденсаторів у даній схемі визначаються виробником мікроконтролера для конкретної резонансної частоти кварцу. Іноді потрібно увімкнути резистор великого номіналу (порядку декількох МОм) між виводами Clk0 і Clk1, щоб генератор працював стабільно. Часто виробник радить використовувати змінний конденсатор, що вмикається до виводу Clk1, щоб забезпечити можливість точного підстроювання частоти.

Багато розробників не люблять використовувати у своїх схемах які-небудь змінні елементи. Якщо розкид частот не відповідає заданим вимогам, то можна використовувати інші схеми синхронізації.

При використанні малих ємностей тактові імпульси будуть мати кращу форму. Якщо увімкнути занадто великі ємності, то це приведе до погіршення форми імпульсів, і мікроконтролер не буде запускатися. Якщо немає впевненості в правильності обраних значень ємностей, то потрібно перевірити форму тактових імпульсів за допомогою осцилографа. Їхня форма повинна бути схожа на верхній рис. 2.14. Якщо схема перевантажена через увімкнення занадто великої ємності, то форма імпульсів буде схожа на нижній рис. 2.14, чи замість імпульсів взагалі буде постійна напруга на рівні приблизно $U_{dd}/2$.

Два основних недоліки цього способу синхронізації – необхідність увімкнення додаткових компонентів і крихкість кристалів кварцу. Обидва ці недоліки можуть бути усунуті, якщо використовувати керамічний резонатор. Керамічні резонатори істотно більш стійкі до ударного навантаження, і більшість з них мають вбудовані конденсатори, внаслідок чого кількість необхідних зовнішніх компонентів зменшується з трьох до одного. Керамічні резонатори, звичайно, мають розбіги частот порядку декількох десятих відсотка (близько 0,5%).

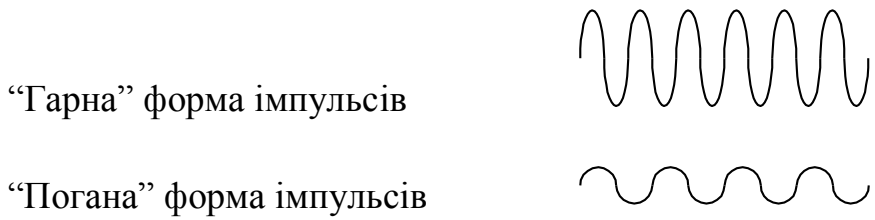


Рисунок 2.14 – Форми імпульсів для мікроконтролера PICMicro

Наступний спосіб синхронізації – використання RC-генератора. У цьому випадку необхідна частота тактових імпульсів задається шляхом відповідного вибору постійної часу RC-ланцюга (рис. 2.15). Це найдешевший спосіб задання частоти, але, на жаль, найменш точний. Якщо кварц забезпечує підтримку частоти з точністю в тисячні частки відсотка, керамічний резонатор – у десяті частки відсотка, то RC-ланцюг дає точність порядку десятків відсотків. Так, при експериментах з мікроконтролерами PIC було встановлено, що точність RC-генератора складає приблизно 20%. Очевидно, що це неприйнятно для багатьох проектів, де потрібен точний підрахунок часу. Однак є області застосування, де така точність цілком достатня.

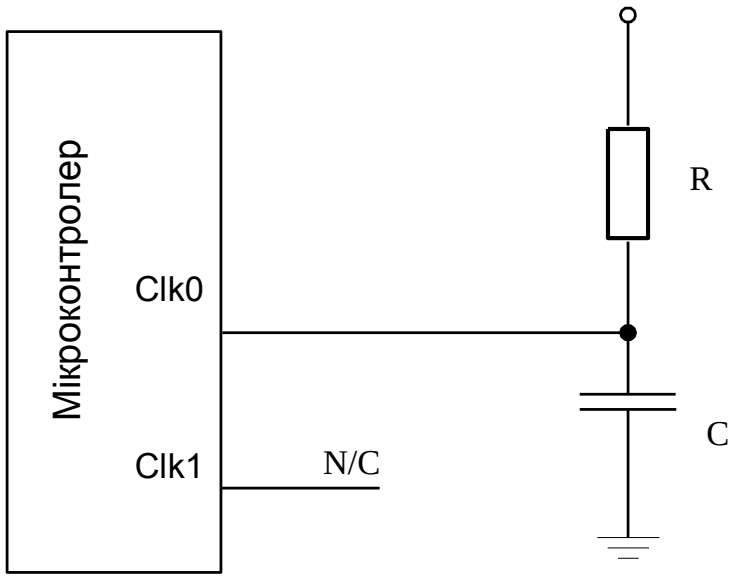


Рисунок 2.15 – Підімкнення RC-ланцюга для генерації тактових сигналів

Немає загальної формули розрахунку номіналів резистора і конденсатора для цього типу генератора. Причиною цього є нелінійні внутрішні ланцюги мікроконтролера. Значення R і C, що забезпечують необхідну частоту синхронізації, можна знайти в посібнику з застосування мікроконтролера.

Велика перевага цього способу синхронізації частоти — низька ціна. Зовнішні компоненти для схеми RC-генератора можуть коштувати менше одного цента! Точність задання тактової частоти можна підвищити, використовуючи змінні резистор чи конденсатор. Однак при цьому складність і ціна генератора збільшаться.

Інший спосіб синхронізації – це подача тактових імпульсів від зовнішнього генератора. Як зазначено вище, мікроконтролери працюють у широкому діапазоні частот. За допомогою зовнішнього тактового генератора можна задати будь-яку частоту синхронізації.

Деякі мікроконтролери містять вбудовані RC-кільцеві генератори, що дозволяють мікроконтролеру працювати без зовнішніх схем синхронізації. Робота внутрішнього генератора звичайно дозволяється шляхом відповідного програмування регістра конфігурації, що виробляється разом із програмуванням пам’яті програм мікроконтролера.

Командні цикли

Новачків в області застосування мікропроцесорів і мікроконтролерів, імовірно, здивує, що командні цикли не теж саме, що такти синхронізації. Командний цикл звичайно складається з декількох тактів, що необхідні процесору для виконання команди. На рис. 2.16 показаний командний цикл, що складається з чотирьох тактів. Протягом командного циклу мікропроцесор чи мікроконтролер виконує необхідні операції, використовуючи тактові сигнали для синхронізації цих операцій.

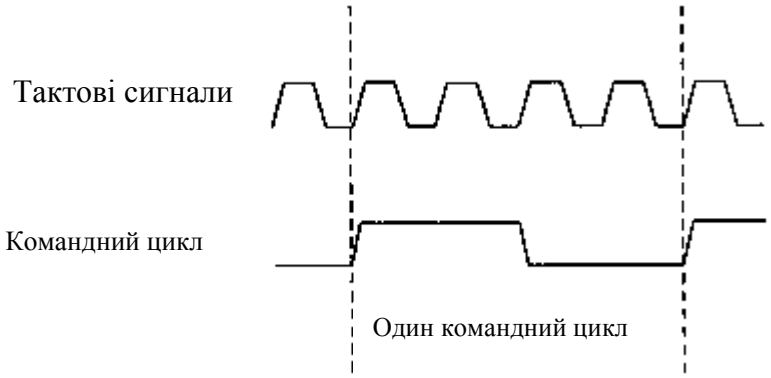


Рисунок 2.16 – Командний цикл і машинні такти

Деякі команди вимагають для виконання більше одного командного циклу. Це ускладнює розрахунок часу виконання програми. Часто для визначення цього часу приходиться звертатися до документації чи використовувати симулятор.

2.7 Програмний лічильник

Програмний лічильник PC (Program Counter) використовується для вказання наступної команди виконуваної програми. Реалізація цієї функції значно ускладнюється, коли необхідно зберегти вміст PC при виклику підпрограм і обробці запитів переривання чи забезпечити розгалуження програми. Рисунок 2.17 ілюструє виконання деяких функцій програмного лічильника, однак, на практиці вони виявляються істотно більш складними.

Програмний лічильник являє собою лічильник з паралельним введенням-виведенням. На рисунку 2.17 показане функціонування PC у процесорі з Прінстонською архітектурою. У процесорах цього типу вміст PC надходить по шині даних у схему керування пам’яттю, вказуючи адресу команди, що зчитується. Часто PC входить до складу схеми керування пам’яттю — це дозволяє уникнути передавання адреси по внутрішній шині даних. Важливі особливості функціонування програмного лічильника — паралельне завантаження нового вмісту, що надходить із шини даних, можливість скидання (повернення до адреси першої команди програми), реалізація інкременту (збільшення вмісту для адресації наступної команди). Сигнали, що забезпечують виконання цих операцій, формуються дешифратором команд, що керує послідовністю дій мікроконтролера.

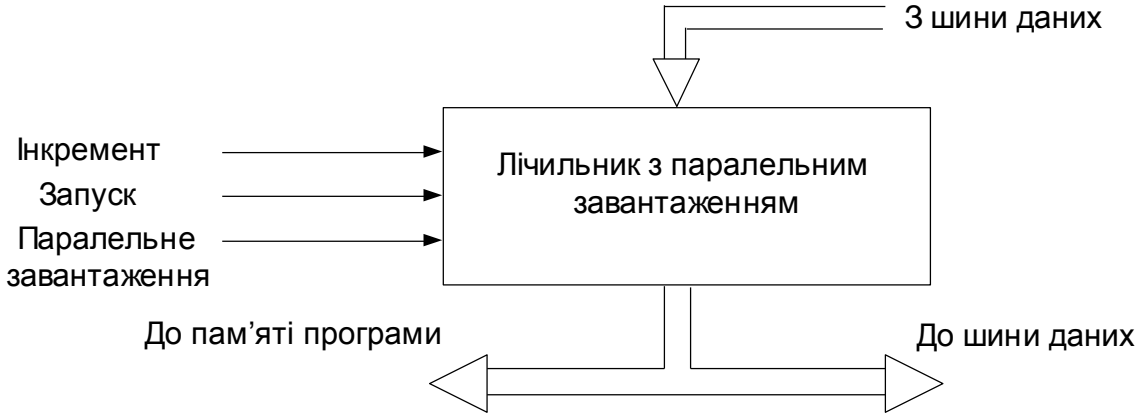


Рисунок 2.17 – Програмний лічильник

Паралельне завантаження використовується для запису в PC адреси переходу при виконанні команди «jump» (безумовний перехід) чи «call» (виклик підпрограми). У комп'ютерах із Прінстонскою архітектурою ця адреса надходить по шині даних. У 8-розрядних мікроконтролерах розрядність PC, звичайно, більше ніж 8 біт (тому що при такій розрядності адреси обсяг доступної пам'яті програм складає всього 256 байт). При завантаженні в PC нової адреси, він надходить по шині даних частинами по 8 біт, що вимагає виконання додаткових машинних циклів. Щоб скоротити час завантаження PC, деякі процесори мають команди розгалуження «branch», при яких завантажуються тільки 8 молодших розрядів адреси, а старші розряди залишаються без зміни. При виконанні такої команди досить передати по шині даних тільки один байт, тоді як для завантаження повної 16-розрядної адреси потрібно пересилання двох байтів.

Початковий вміст PC після запуску мікроконтролера може мати будь-яке значення. Хоча найбільш очевидним представляється використання як початкового значення 0x0000, деякі мікроконтролери починають виконання програми з інших адрес. Аналогічна ситуація має місце з адресами («векторами») переривань. При обслуговуванні переривань у PC звичайно завантажують вміст, що відрізняється від адреси, що завантажується при початковому запуску мікроконтролера, однак, для реалізації переривання і запуску можуть використовуватися ті самі апаратні засоби.

Після читання чергової команди вміст програмного лічильника збільшується (інкрементується), щоб забезпечити перехід до адреси наступної команди. Якщо виконується виклик підпрограми чи відбувається переривання, то адреса повернення (адреса наступної команди) може бути збережена у стеку без виконання додаткових тактів для інкременту вмісту програмного лічильника.

Необхідно зробити деякі важливі зауваження з приводу функціонування програмного лічильника. У роботі з мікроконтролерами, що мають Прінстонську архітектуру, необхідно стежити, щоб програмний лічильник не вийшов за межі пам'яті програм. Якщо це трапиться, то мікроконтролер

може почати вибірку даних замість команд і намагатися виконати обрані коди даних, що приведе до непередбаченого результату.

Інша проблема може виникнути, коли програмний лічильник досягне кінця адресної пам'яті. Деякі мікроконтролери в цьому випадку здійснюють циклічний перехід до команди, розміщеної за адресою 0x0000, в інших мікроконтролерах дешифратор команд продовжує інкрементувати програмний лічильник, що приводить до виконання невизначених команд.

2.8 Арифметико-логічний пристрій

Арифметико-логічний пристрій (АЛП) процесора використовується для виконання всіх математичних операцій у програмі. Ці операції включають додавання, віднімання, логічне "І", логічне "АБО", зсув вмісту регістрів і встановлення вмісту регістра стану відповідно до отриманих результатів. АЛП не використовується при читанні або записуванні даних чи команд, він слугує тільки для оброблення даних.

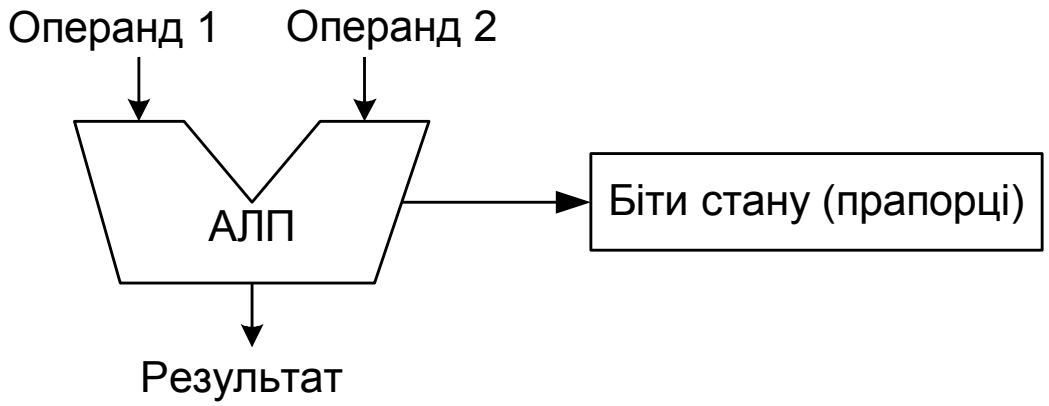


Рисунок 2.18 – Структура АЛП

АЛП можна представити як апаратний блок, що обробляє два слова даних (операнди) і зберігає отриманий результат (рис. 2.18). Як вводяться операнди в АЛП і куди надходить результат – залежить від конкретного типу мікроконтролера. У цьому полягає одне з основних розходжень між різними типами процесорів і їх систем команд. Деякі мікроконтролери вибирають один операнд із регістра-акумулятора і зберігають результат також в акумуляторі. Інші мікроконтролери дозволяють використовувати різні джерела операндів і місця розміщення результатів.

АЛП, звичайно, працюють тільки з додатними цілими числами. Однак при виконанні віднімання виходять від'ємні числа, якщо від'ємник більше зменшуваного. Для подання від'ємних чисел використовується додатковий код («доповнення до двох»). Це необхідно враховувати при знайомстві з роботою АЛП.

Розглянемо як виконується команда віднімання на прикладі мікроконтролера Microchip PIC. Замість віднімання одного числа від іншого, відбувається додавання від'ємного числа:

$$A - B = A + (-B),$$

де від'ємне число - B подається в додатковому коді. Щоб одержати додатковий код від'ємного двійкового числа, необхідно інвертувати значення кожного біта, а потім додати одиницю:

$$-B = (B \wedge 0x00FF) + 1.$$

Якщо є схема перетворення від'ємних чисел у додатковий код, то немає необхідності використовувати схему віднімання. Досить мати в складі АЛП суматор, що реалізує віднімання за допомогою такої заміни:

$$B=A+(B \wedge 0x00FF) + 1.$$

Цей метод виконання віднімання може викликати деякі ускладнення при аналізі отриманого результату, якщо враховувати прапорець перенесення, встановлюваний в результаті додавання і віднімання.

У «класичних» АЛП, що мають суматор і обчислювач, часто використовується загальний прапорець «переносу-займу». Цей прапорець встановлюється в "1", коли результат додавання більше ніж 0x00FF чи результат віднімання менше нуля. В обох випадках прапорець використовується для вказання того, що значення восьми старших бітів результату залежать від результату, отриманого при операції над вісьмома молодшими бітами.

Якщо АЛП не містить обчислювача (як у приведеному вище прикладі), то прапорець перенесення також встановлюється після додавання чи віднімання, але він має інше значення. Щоб зрозуміти це, розглянемо декілька прикладів. Перший приклад показує, що відбувається, коли одне число віднімається від іншого числа, що більше першого:

$$0x0077 - 0x0055 = 0x0077 + (-0x0055) =$$
$$= 0x0077 + (0x0055 \wedge 0x00FF) + 1 = 0x0077 + 0x00AA + 1 = 0x0122$$

Результат вийшов більший, ніж 0xFF, що приводить до установлення прапорця перенесення "1" у молодшому біті старшого байта (прапорець перенесення/займу в цьому випадку не дорівнює "1"). Молодші вісім бітів рівні 0x22 (що й очікувалося) – це значення буде записано як результат у приймач.

Наступний приклад ілюструє ситуацію, коли більше число віднімається від меншого числа:

$$0x0055 - 0x0077 = 0x0055 + (-0x0077) =$$
$$= 0x0055 + (0x0077 \wedge 0x00FF) + 1 = 0x0055 + 0x0088 + 1 = 0x00DE$$

У цьому прикладі молодші 8 розрядів подають число 22 у додатковому коді (що й очікувалося), а біт переносу в старший байт (прапорець перенесення) дорівнює "0", як очікувалося. У «класичному» АЛП для даного прикладу результат також буде мати значення 0xDE , але установиться прапорець займу-перенесення рівний "1".

Неважко помітити, що в АЛП, що не використовує обчислювач, прапорець перенесення встановлюється в "1", коли результат віднімання дода-

тний, і скидається в “0”, коли результат від’ємний. Тому формований біт перенесення в старший байт у цьому випадку можна було б назвати прапорцем «перенесення-знаку», тому що при відніманні він вказує знак результату.

Необхідно мати на увазі, що різні представники одного сімейства мікроконтролерів звичайно мають однакові АЛП. В деяких сімействах АЛП реалізують визначені операції, наприклад, множення, що не виконуються мікроконтролерами інших сімейств. Додаткові функції, що забезпечують різні мікроконтролери одного сімейства, реалізуються шляхом включення в їхню структуру додаткових апаратних засобів, аналогічно периферійним пристроям. При цьому структура і функції АЛП зберігаються, тому що додаткові пристрої, що вводяться в мікроконтролер, використовують свої регістри, що не зв’язані з регістром стану й акумуляторами. Наочним прикладом цього є сімейство 8-розрядних мікроконтролерів 68HC05, що випускаються фірмою Motorola.

Складність АЛП багато в чому визначає складність всього мікроконтролера в цілому. Часто над створенням АЛП працює група розробників, порівнянна за складом з тією, котра працює над іншою частиною мікропроцесора чи мікроконтролера (і навіть більша, коли розробляється процесор аналогічний за складністю персональному комп’ютеру). Від того, як працює АЛП, залежить функціонування процесора, що входить до складу мікроконтролера, а значить і функціонування всього мікроконтролера.

2.9 Сторожові таймери

Часто електричні перешкоди, вироблювані навколишнім устаткуванням, викликають звертання мікроконтролера за неправильною адресою, після чого його поводження стає непередбаченим (мікроконтролер «йде в рознос»). Щоб відслідковувати такі ситуації до складу мікроконтролера часто включають сторожові таймери.

Цей пристрій викликає скидання мікроконтролера, якщо його вміст не буде оновлено протягом визначеного проміжку часу (звичайно від десятків мілісекунд до декількох секунд). Якщо зміна вмісту програмного лічильника не відповідає заданій програмі, то команда модифікації сторожового таймера не буде виконана. У цьому випадку сторожовий таймер робить скидання мікроконтролера, встановлюючи його у вихідний стан.

Багато розробників не використовують сторожові таймери у своїх проектах, тому що не бачать необхідності їхнього застосування для боротьби з впливом електричних перешкод, наприклад, при розміщенні мікроконтролера в електронно-променевому дисплеї поблизу від трансформатора, що забезпечує гасіння зворотного ходу променя чи поруч з катушками запалювання в автомобілі. У сучасній електроніці імовірність виникнення електричних завад незначна, хоча вони іноді виникають у ситуаціях, схожих на перераховані вище.

Не рекомендується використовувати сторожовий таймер для маскування програмних проблем. Хоча цей таймер може зменшити імовірність програмних помилок, однак, навряд чи він забезпечить виключення всіх можливих причин їхнього виникнення. Замість того, щоб сподіватися на запобігання програмних збоїв апаратними засобами, краще більш ретельно протестувати програмне забезпечення в різних ситуаціях.

2.10 Підпрограми і функції

Звичайно, підпрограми і функції розглядаються в різних книгах і лекційних курсах як визначені прийоми програмування. Тут буде розглянута їхня апаратна реалізація, що забезпечує виклик підпрограм і передачу параметрів, необхідних для її виконання.

При виклику підпрограми потрібно зберегти вміст програмного лічильника для того, щоб команда повернення могла повернути керування вихідній програмі. Це може виконуватися автоматично (як частина команди CALL) шляхом збереження адреси повернення в стеку. При поверненні до вихідної програми адреса витягається зі стека і завантажується в програмний лічильник.

Однак деякі цифрові системи, наприклад, великі комп'ютери серії IBM 370, не мають стека для збереження вмісту програмного лічильника чи інших регістрів. У цій архітектурі при виконанні команди «goto» (перехід до підпрограми) адреса повернення зберігається в регістрі. При цьому відповідальність за збереження адреси повернення покладається на програміста. Найбільш простий спосіб імітувати команду «call» — це просто залишити адресу повернення в регістрі, якщо підпрограма не викликає інші підпрограми. При поверненні з підпрограми вміст цього регістра завантажується в програмний лічильник.

У деяких пристроях при виконанні команди «goto» адреса повернення взагалі ніде не зберігається. У такому випадку для імітації виклику підпрограми адресу повернення необхідно зберегти у вигляді змінної. Це може бути реалізоване в такий спосіб:

```
ReturnVar = AfterGoto;          Зберегти адресу повернення
goto Subroutine;                Викликати підпрограму Subroutine

Subroutine  :                   Початок підпрограми
ProgramCounter = ReturnVar;     Повернутися до команди наступної за «goto»
```

Функції – це складні підпрограми, при виконанні яких визначені параметри передаються з вихідної програми в підпрограму, а після завершення повертаються назад. Наприклад, функція мовою С може бути визначена в такий спосіб: int Func(int i, char far * Ptr);

У цьому прикладі «Func» – функція, що вимагає два вхідних параметри: ціле число і покажчик, і повертає значення цілого числа. Один з найбільш ефективних і які часто зустрічаються способів передавання параметрів у функцію – це помістити їх у стек перед викликом функції. У підпрограмі можна завантажити індексний регістр значенням покажчика стека й у такий спосіб одержати доступ до параметрів. Цей метод оброблення параметрів має істотну перевагу в порівнянні з іншими методами: для передавання параметрів використовується спеціально виділений фрагмент пам'яті. Змінні, визначені всередині функції (включаючи вхідні параметри), часто називаються «локальними» чи «автоматичними», тому що вони використовуються тільки даною функцією, і пам'ять для їхнього розміщення виділяється автоматично.

Аналогічний метод виклику функцій може бути реалізований у процесорах, що не мають стека, шляхом використання індексного регістра для емуляції стека. Якщо не можна безпосередньо завантажити в стек вміст програмного лічильника, то адресу повернення до вихідної програми можна зберегти в емульованому стеку.

Іншим способом передавання параметрів є їхнє збереження в регістрах процесора чи в пам'яті даних в якості спеціальних змінних. Передавання параметрів через регістри процесора, що використовується в деяких випадках, скорочує число регістрів, доступних при виконанні функції. Збереження параметрів у вигляді спеціальних змінних також можливо, але при цьому зменшується обсяг пам'яті, доступної для використання програмою. Дане обмеження може бути дуже істотним для мікроконтролерів.

Повернення параметра можна виконати кожним із трьох описаних вище способів. Але, звичайно, значення параметрів що повертаються, завантажуються в регістри процесора, тому що це найбільш швидкий і ефективний метод передавання даних.

2.11 Переривання

Багато користувачів вважають, що переривання – це та частина апаратного забезпечення, якій краще дати спокій, тому що їхнє використання вимагає чудового знання процесора для розроблення програми обробки переривання. У противному випадку при виникненні переривання система «засинає» чи «йде вразнос». Таке почуття звичайно з'являється в розробника після досвіду роботи з перериваннями для персонального комп'ютера, що має ряд особливостей, що ускладнюють створення обробника переривань. Багато з цих проблем не мають місця в устаткуванні, реалізованому на базі мікроконтролерів. Використання в даному устаткуванні переривань може істотно спростити його розробку і застосування.

У комп'ютерній системі переривання – це запуск спеціальної підпрограми (названої «оброблювачем переривання» чи «програмою обслуговування переривання»), що викликається сигналом апаратури. На час вико-

нання цієї підпрограми реалізація поточної програми зупиняється. Термін «запит на переривання» (interrupt request) використовується тому, що іноді програма відмовляється підтвердити переривання і виконати обробку переривання негайно (рис. 2.19).

Переривання в комп'ютерній системі аналогічні перериванням у повсякденному житті. Класичний приклад такого переривання — телефонний дзвінок під час перегляду телевізійної передачі. Коли дзвонить телефон, є три варіанти. Перший – проігнорувати дзвінок. Другий – відповісти на дзвінок, але сказати, що передзвоните пізніше. Третій –відповісти на дзвінок, відклавши всі поточні справи. У комп'ютерній системі також є три подібних відповіді, що можуть бути використані як реакція на зовнішній апаратний запит.

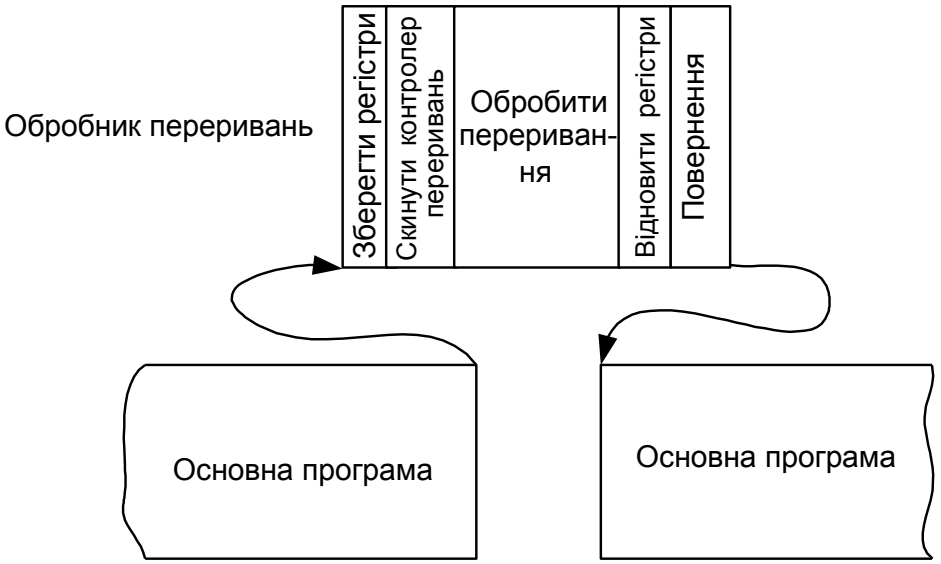


Рисунок 2.19 – Виконання переривання

Перша можлива відповідь – «не реагувати на переривання, поки не завершиться виконання поточної задачі» — реалізується шляхом заборони (маскування) обслуговування запиту переривання. Після завершення задачі можливий один із двох варіантів: скидання маски і дозвіл обслуговування, що приведе до виклику обробника переривання, чи аналіз значення бітів («пуллінг»), що вказують на надходження запитів переривання і безпосереднє виконання програми обслуговування без виклику обробника переривання. Такий метод оброблення переривань використовується, коли потрібно забезпечити заданий час виконання основної програми, тому що будь-яке переривання може порушити реалізацію необхідного інтерфейсу.

Небажане тривале маскування переривань, тому що протягом цього часу може відбутися накладення декількох подій, що викликають переривання, а розпізнаватися буде тільки одне. Допустима тривалість маскування залежить від конкретного застосування мікроконтролеру, типу і частоти проходження таких подій. Не рекомендується забороняти перериван-

ня на час більше, ніж половина мінімального очікуваного періоду проходження подій, що запитують переривання.

Обробник переривання завжди забезпечує таку послідовність дій:

1. Зберегти вміст регістрів контексту.

2. Скинути контролер переривань і устаткування, що викликало запит.

3. Обробити дані.

4. Відновити вміст регістрів контексту.

5. Повернутися до перерваної програми.

Регістри контексту – це регістри, що визначають поточний стан виконання основної програми. Зазвичай, до їхнього числа відносяться програмний лічильник, регістри стану й акумулятори. Інші регістри процесора, такі як індексні регістри, можуть бути використані в процесі обробки переривання, тому їхній вміст також необхідно зберегти. Всі інші регістри є специфічними для конкретного мікроконтролера і його застосування.

Після скидання у вихідний стан контролер переривань готовий сприймати наступний запит, а устаткування, що викликає переривання, готове надсилати запит, коли виникають відповідні причини. Якщо надійде новий запит переривання, то регістр маскування переривань процесора запобігає обробці переривання, але регістр стану переривань зафіксує цей запит, що буде очікувати свого обслуговування. Після завершення обробки поточного переривання маска переривань буде скинута, і запит який надійшов, надходить на обробку.

Вкладені переривання складні для реалізації деякими типами мікроконтролерів, що не мають стека. Ці переривання можуть також викликати проблеми, пов’язані з переповненням стека. Проблема переповнення актуальна для мікроконтролерів через обмежений обсяг їхньої пам’яті даних і стека: послідовність вкладених переривань може привести до того, що в стек буде поміщено більше даних, ніж це допустимо.

Нарешті, переривання оброблене. Другий приклад з телевізором показує, що можна швидко відреагувати на запит переривання, прийнявши необхідні дані, що будуть потім використані після вирішення поточної задачі. У мікроконтролерах це реалізується шляхом збереження даних, що надійшли, у масиві пам’яті і наступної їхньої обробки, коли виконання вихідної програми буде завершено. Такий спосіб обслуговування є гарним компромісом між негайною повною обробкою переривання, що потребує багато часу, і ігноруванням переривання, що може привести до втрати інформації про подію, що викликала переривання.

Відновлення регістрів контексту і виконання команди повернення з переривання переводить процесор в стан, у якому він знаходився до виникнення переривання.

Розглянемо, що відбувається з вмістом різних регістрів при обробленні переривання. Вміст регістра стану, звичайно, автоматично зберігається разом із вмістом програмного лічильника перед обробленням пере-

ривання. Це рятує від необхідності зберігати його в пам'яті програмними засобами за допомогою команд пересилання, а потім відновлювати при поверненні до вихідної програми. Однак таке автоматичне збереження реалізується не у всіх типах мікроконтролерів, тому організації обробки переривань варто приділити особливу увагу. Дане питання буде розглянуто для кожного з розглянутих у книзі мікроконтролерів у розділі, що описує їх застосування.

Якщо вміст регістра стану зберігається перед початком виконання обробника переривання, то за командою повернення виробляється його автоматичне відновлення.

Якщо вміст інших регістрів процесора змінюється при виконанні обслуговування переривання, то воно також повинно бути збережене в пам'яті до зміни і відновлено перед поверненням в основну програму. Звичайно, прийнято зберігати всі регістри процесора, щоб уникнути непередбачених помилок, що дуже важко локалізувати.

Адреса, що завантажується в програмний лічильник при переході до обробника переривання, називається «вектор переривання». Існує кілька типів векторів. Адреса, що завантажується в програмний лічильник при запуску мікроконтролера (reset) називається «вектор скидання». Для різних переривань можуть бути задані різні вектори, що рятують програму обслуговування від необхідності визначати причину переривання. Використання різними перериваннями одного вектора звичайно не викликає проблем при роботі мікроконтролерів, тому що найчастіше мікроконтролер виконує одну єдину програму. Цим мікроконтролер відрізняється від персонального комп'ютера, у процесі експлуатації якого можуть додаватися різні джерела переривань. (Якщо Ви коли-небудь, вмикали два пристрої до портів COM1 і COM3, то Ви уявляєте про що йде мова). У мікроконтролері, де апаратна частина добре відома, не повинно виникнути яких-небудь проблем при спільному використанні векторів переривань.

Останнє, що залишилося розглянути, – це програмні переривання. Існують процесорні команди, що можуть бути використані для імітації апаратних переривань. Найбільш очевидне використання цих команд — це виклик системних підпрограм, що розташовуються в довільному місці пам'яті, чи вимагають для звертання до них міжсегментних переходів. Ця можливість реалізована в мікропроцесорах сімейства Intel x86 і використовується в базовій системі введення-виведення BIOS (Basic Input/Output System) і операційній системі DOS персональних комп'ютерів для виклику системних підпрограм без необхідності фіксування точки входу. Замість цього використовуються різні вектори переривань, що вибирають команду, що повинна виконуватися, коли відбувається таке програмне переривання.

2.12 Таймери

Якщо Ви знайомі з мікропроцесорами, то, можливо, вважаєте, що таймери використовуються тільки для забезпечення заданої затримки. У мікроконтролерах таймери використовуються для вирішення набагато ширшого кола задач.

Звичайно, для перемикання таймера використовуються тактові імпульси процесора. Завантаживши в таймер початкове значення, можна відраховувати визначені інтервали часу, фіксуючи закінчення інтервалу за моментом переповнення таймера. Часто перед таймером вмикають попередній подільник тактової частоти, щоб мати можливість відраховувати більш довгі інтервали часу. Подільник забезпечує інкремент вмісту таймера після надходження визначеного числа тактових імпульсів.

Якщо потрібно реалізувати затримку в 10мс (0.01с) у системі з тактовою частотою 10 МГц, то можна використовувати схему, показану на рис. 2.20, у такий спосіб. Спочатку треба визначити необхідний коефіцієнт поділу. При тактовій частоті 10 МГц таймер повинний перемикатися до переповнення 1000 разів, що неможливо для 8-розрядного лічильника. Щоб забезпечити задану затримку, необхідно вибрати коефіцієнт поділу для попереднього подільника, що, звичайно, є ступенем двійки (тобто 1, 2, 4, 8,...256). Якщо вибрати коефіцієнт поділу 64, то таймер перемкнеться 156 разів при надходженні на вхід подільника 64 × 156 = 9984 тактових імпульсів, що досить близько до необхідної величини. Більш точне значення затримки може бути досягнуте шляхом додавання команд NOP чи інших команд, використовуваних для заповнення визначених проміжків часу. Для відліку заданого часу таймер можна очистити, а потім безупинно порівнювати його вміст зі значенням 156.

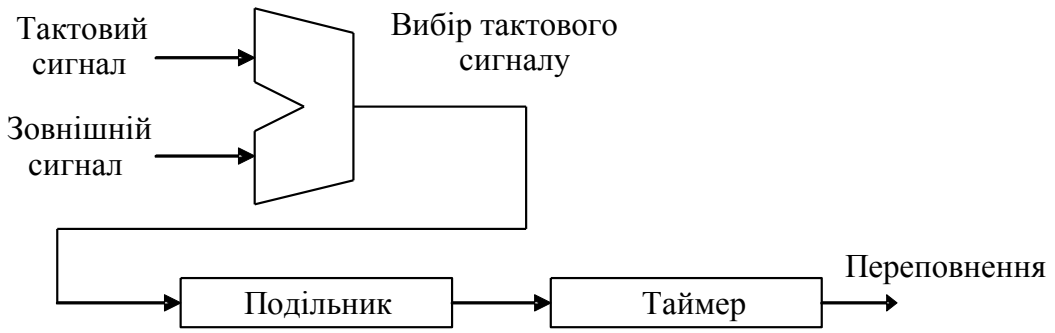


Рисунок 2.20 – Структура таймера в мікроконтролері

Більш ефективний спосіб відліку – завантажити в таймер число 100 = = 256 - 156 і чекати, коли прапорець переривання переповнення таймера установиться в “1”. Використовуючи даний метод, можна реалізувати мультизадачний режим виконання програм. Послідовність виконання буде здійснюватись, коли таймер відрахує 10 мс.

Не рекомендується використовувати цей метод для створення годинника реального часу. Оскільки при перезавантаженні таймера значення коефіцієнта поділу подільника невідомо, а його вміст скидається в "0" при записі в регістр таймера, то при відліку часу можуть виникати помилки.

Замість цього краще дати можливість таймеру рахувати безупинно (тоді подільник ніколи не скидається), а при його переповненні збільшувати вміст лічильника годинника реального часу. Коли програмі потрібно довідатися поточний час, він вибирає і перетворює вміст цього лічильника, одержуючи правильне значення часу. Такий метод використовується в IBM PC, де перемикання («тік») таймера відбувається 18,2 разів у секунду.

Дивлячись на рис. 2.20 можна помітити, що таймер може бути використаний для підрахунку зовнішніх подій, якщо використовувати для його перемикання зовнішній сигнал.

Якщо мікроконтролер містить два таймери, то можна легко реалізувати простий тахометр – лічильник числа подій, що відбуваються за одиницю часу (рис. 2.21). Програма реалізації тахометра спочатку скидає в "0" вміст лічильника TMR2, а потім встановлює визначений інтервал часу для спрацьовування таймера TMR1. Після закінчення цього інтервалу TMR1 спрацьовує (встановлюється в 1 біт переповнення TMR1), викликаючи зчитування поточного вмісту з TMR2.

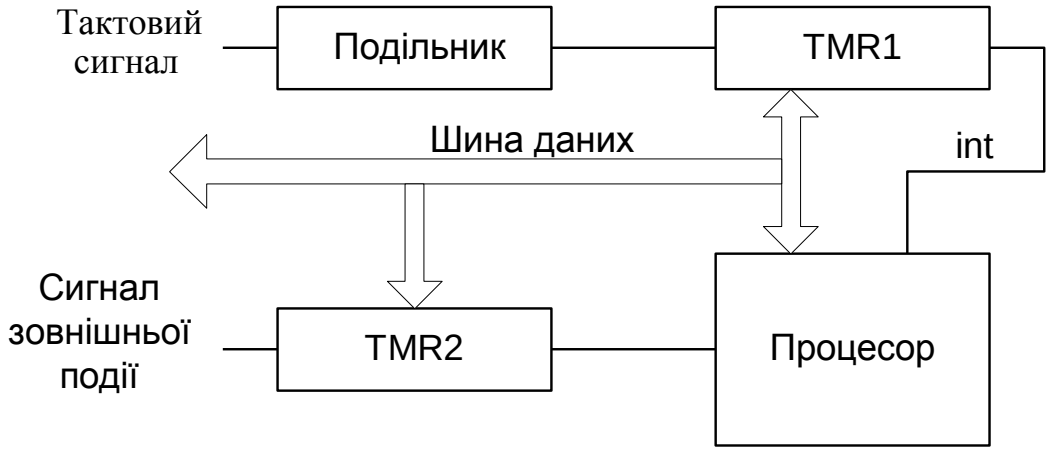
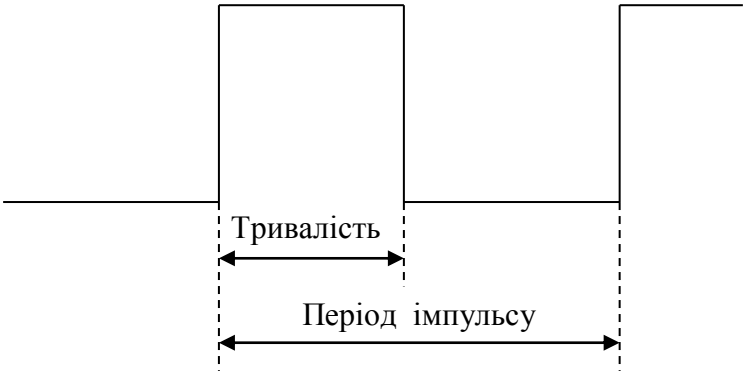


Рисунок 2.21 – Реалізація тахометра

Таймери в мікроконтролерах часто використовуються для введення-виведення сигналів із широтно-імпульсною модуляцією PWM (Pulse Width Modulated). PWM сигнал часто використовується для передачі значення аналогового сигналу в цифрову систему або з неї. Сигнал має повторювану форму, де тривалість імпульсу пропорційна значенню переданої аналогової величини (рис. 2.22).



$$\text{Шпаруватість} = \frac{\text{Тривалість}}{\text{Період}} \ 100\%$$

Рисунок 2.22 – Сигнал із широтно-імпульсною модуляцією PWM

PWM-сигнали часто використовується для керування електродвигунами, задаючи швидкість обертання чи положення вала в сервоприводі. На рис. 2.23 показана схема генерації PWM-сигналу на виході мікроконтролера. Поки задане значення «тривалості імпульсу» більше, ніж вміст таймера, на виході цієї схеми буде підтримуватися високий потенціал (вершина імпульсу). Коли вміст таймера стане рівним або більшим, ніж задане значення «періоду», то таймер скидається в 0, і процес повторюється. Такий метод одержання PWM-сигналу вимагає мінімальної участі процесора – треба тільки задати необхідні значення періоду і тривалості імпульсу. При цьому значення тривалості імпульсу може змінюватися процесором у будь-який час без зупинки процесу виведення.

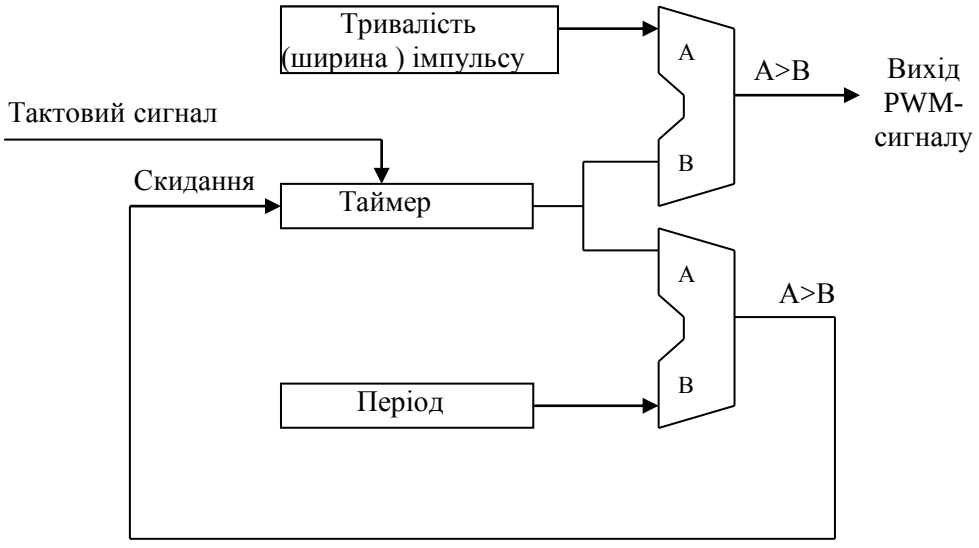


Рисунок 2.23 – Схема генератора PWM-сигналу

Якщо необхідно сформувати прямокутний імпульсний сигнал визначеної частоти, то можна використовувати ту ж схему, задавши період у два рази більше тривалості імпульсу.

Вимірювання тривалості імпульсу PWM-сигналу може бути зроблено за допомогою схеми, наведеної на рис. 2.24. У цій схемі на вхід RESET таймера подається сигнал скидання доти, поки вхідний сигнал має низький рівень. При надходженні високого рівня вхідного сигналу таймер запускається, і його вміст надходить у регістр тривалості імпульсу. Після закінчення імпульсу, коли на вході знову встановлюється низький рівень, вміст таймера зберігається в регістрі тривалості імпульсу, а таймер скидається в "0" до приходу наступного імпульсу. Для спрощення на схемі не показані деякі елементи затримки, які гарантують, що значення таймера буде записано в регістр тривалості до його скидання.

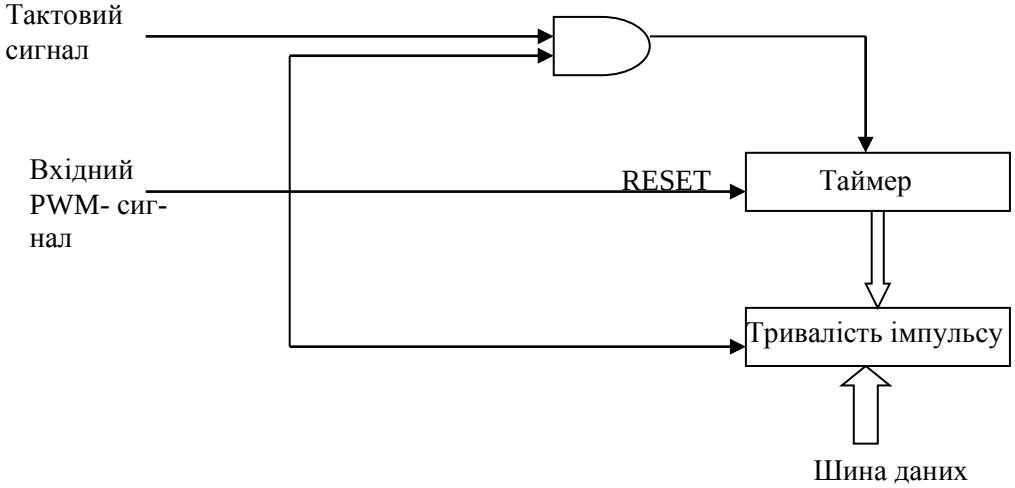


Рисунок 2.24 – Схема вимірювання тривалості імпульсу

2.13 Паралельне введення-виведення даних

Основний інтерфейс між мікроконтролером і зовнішніми пристроями реалізується через паралельні порти введення-виведення. У багатьох мікроконтролерах виводи цих портів служать також для виконання інших функцій, наприклад, послідовного чи аналогового введення-виведення. При першому знайомстві з мікроконтролером дуже важливо зрозуміти, як здійснюється паралельне введення-виведення даних.

У більшості мікроконтролерів окремі виводи портів можуть бути запрограмовані на введення чи виведення даних. Типова схема увімкнення зовнішнього виводу показана на рис. 2.25. Необхідно звернути особливу увагу на те, що при введенні даних зчитується значення сигналу, що надходить на зовнішній вивід, а не вміст тригера даних. Якщо до зовнішнього виводу увімкнуті виходи інших пристроїв, то вони можуть установити свій рівень вихідного сигналу, що буде використаний замість очікуваного значення даних, записаних у тригер. У деяких мікроконтролерах існує можливість вибору між читанням даних, установлених на виході тригера чи на зовнішньому виводі.

Можуть виникнути деякі проблеми в мікроконтролерах, що виконують читання даних і їх записування у регістр порту за допомогою однієї команди. При цьому введення неправильних даних приведе до їх записування в тригер у результаті виконання команди читання/записування.

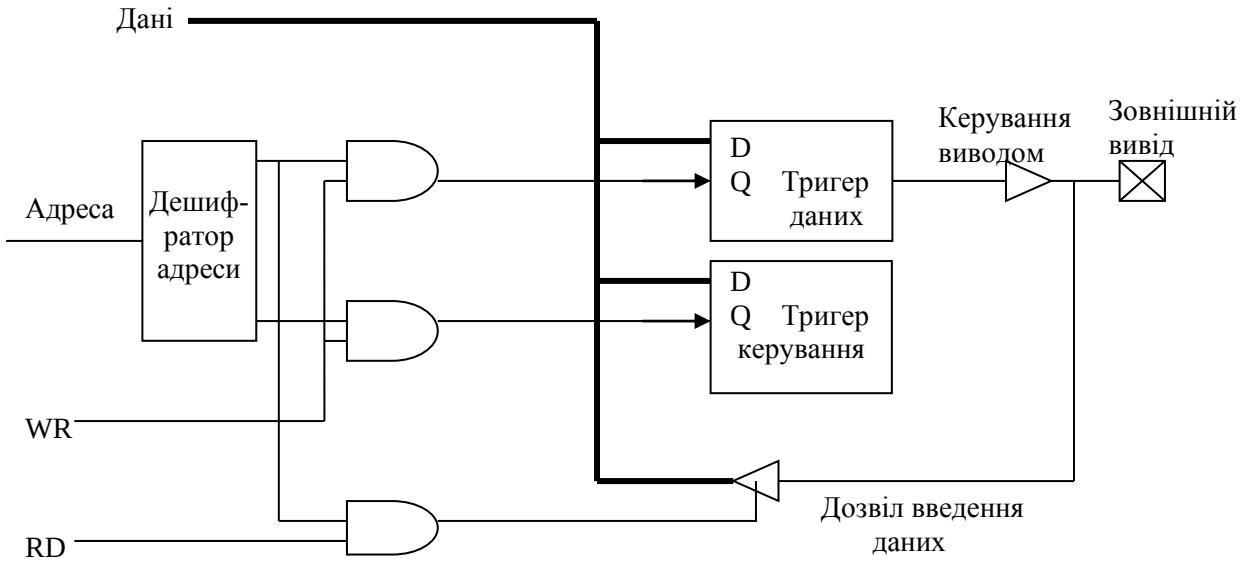


Рисунок 2.25 – Типова схема підімкнення зовнішнього виводу

Інший розповсюджений варіант увімкнення зовнішнього виводу – це схема з «відкритим колектором» (у дійсності з «відкритим стоком»), показана на рис. 2.26. Стан даного виводу відповідає стану увімкнутої до нього шини доти, поки він не буде перемкнутий на роботу в режимі виходу і встановлений у стан "0". У цьому випадку вивід з'єднаний з «землею» через MOS-транзистор, що керується логічною схемою «І», увімкнутої до виходів тригерів керування і даних.

Таке підімкнення виводу дозволяє створювати шини з об'єднанням виходів пристроїв за схемою «монтажне І». У цій схемі шина під'єднується до напруги живлення U_{cc} (логічна „1") через резистор, а до потенціалу «землі» (логічний „0") — через декілька ключів чи транзисторів. При їхньому вмиканні на шині встановлюється низький потенціал. Прикладом застосування даної схеми вмикання є двонаправлена шина з декількома джерелами інформації, кожний з яких може передавати дані, переводячи шину в стан з низьким потенціалом (логічний „0").

Звичайний вихід (рис. 2.25) функціонує як вихід з відкритим колектором, якщо він працює в режимі введення і переходить у режим виведення тільки для видачі «0». Коли на шину необхідно вивести «0», то спочатку «0» записується в тригер даних, а потім за допомогою тригера керування на виході встановлюється низький рівень потенціалу.

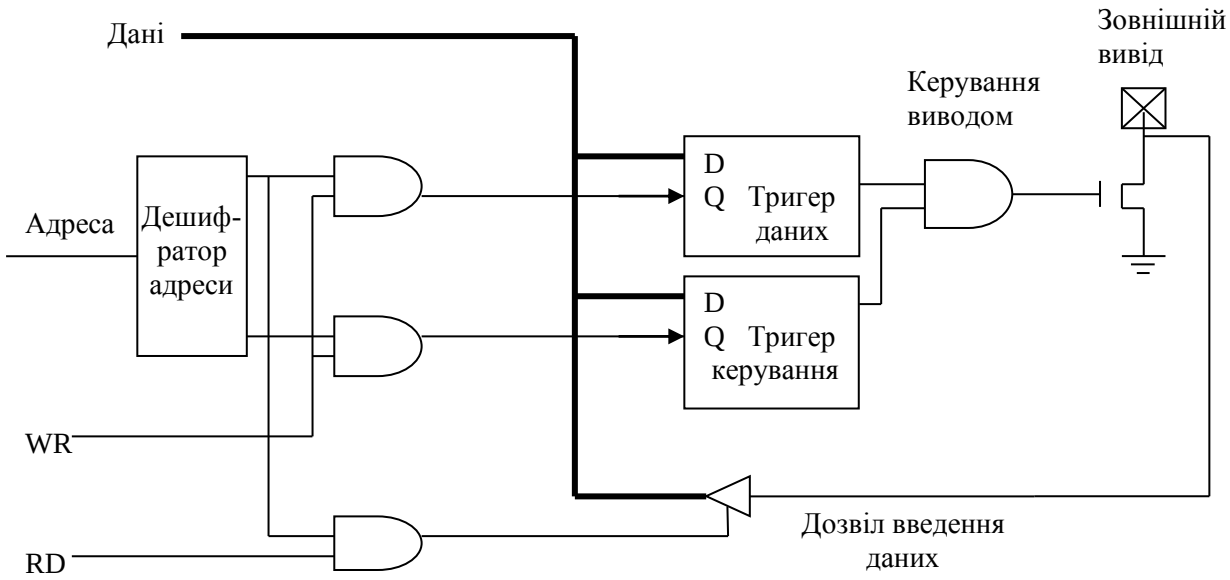


Рисунок 2.26 – Схема підімкнення виводу з відкритим колектором

Тригер керування дозволяє виведення даних на шину. Як показано на приведених схемах, у сучасних мікроконтролерах забезпечується індивідуальний доступ до тригерів даних і керування за допомогою адресної шини. У деяких більш старих типах мікроконтролерів виводи поєднуються в групи по 4 чи 8 розрядів, що програмуються на спільну роботу як входи або виходи. У цьому випадку при розробці пристроїв потрібно більш ретельно планувати роботу апаратного і програмного забезпечення, щоб забезпечити виконання зовнішніми виводами необхідних функцій.

Зовнішній вхід може бути також використаний для подачі запиту переривання. Це, звичайно, реалізується, коли вивід працює в режимі введення.

2.14 Перетворення логічних рівнів

При роботі з мікроконтролерами часто приходиться забезпечувати інтерфейс пристроїв, реалізованих на мікросхемах з різним типом логіки. При використанні серійних мікросхем з позитивною логікою, наприклад, ТТЛ (транзисторно-транзисторна логіка) чи КМОН (логіка на комплементарних МОН-транзисторах), реалізація інтерфейсу не викликає проблем, тому що можливе безпосереднє з’єднання цих мікросхем. Однак інтерфейс пристроїв, реалізованих на мікросхемах з негативною і позитивною логікою, наприклад, ЕЗЛ (емітерно-зв’язана логіка) і КМОН, може викликати деякі труднощі.

Хоча є спеціальні мікросхеми для виконання такого інтерфейсу, однак, звичайно вони забезпечують однобічне передавання даних, що ускладнює реалізацію двонаправлених шин. Крім того, застосування цих мікросхем може значно збільшити вартість проектованого пристрою.

Найбільш типовий метод перетворення логічних рівнів полягає в тому, щоб привести у відповідність пороги перемикання обох пристроїв. Як показано на рис. 2.27, нижній потенціал напруги живлення для КМОН-мікроконтролера зрушується нижче потенціалу «землі» таким чином, щоб стан перемикання став відповідати порогові перемикання мікросхем ЕЗЛ. Резистор з номіналом від 1 КОм до 10 КОм використовується для обмеження струму, що протікає через різне значення логічних рівнів мікросхем, що використовуються. При розгляданні схеми може скластися враження, що ціна зсуву рівня напруги живлення для мікроконтролера набагато вища ціни декількох інтерфейсних мікросхем. Насправді це нескладно, тому що сучасні КМОН-мікроконтролери споживають дуже малу потужність. У приведеній схемі необхідний зсув рівнів може бути забезпечений вмиканням у ланцюг живлення кремнієвих діодів, спад напруги на яких складає 0,7 В. Цей простий спосіб узгодження рівнів дозволяє з мінімальними додатковими витратами вмикати КМОН-мікроконтролери до мікросхем ЕЗЛ, забезпечуючи можливість двонаправленого передавання даних.

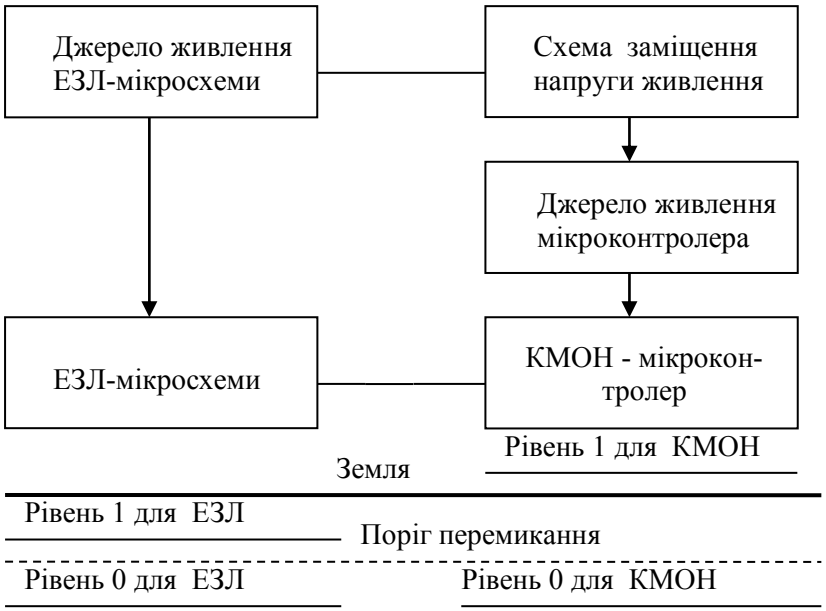


Рисунок 2.27 – Узгодження логічних рівнів ЕЗЛ- та КМОН- мікросхем

2.15 Обмін даними із зовнішніми пристроями

Послідовне введення-виведення даних

Найбільш розповсюджений вид зв'язку між різними системами (чи комп'ютерами) – це послідовний обмін. У цьому випадку байт даних передається по єдиному проводі, біт за бітом, із забезпеченням синхронізації між приймачем і джерелом даних. Очевидна перевага послідовного передавання даних полягає в тому, що вона вимагає невеликої кількості ліній зв'язку.

Існує безліч стандартних послідовних протоколів передачі даних, що застосовуються в мікроконтролерах. У деяких мікроконтролерах ці протоколи реалізуються внутрішніми схемами, розміщеними на кристалі, що дозволяє спростити розробку різних програм.

Асинхронний послідовний обмін

Найбільш розповсюджена форма послідовного зв'язку — асинхронний обмін, при якому байт даних посилається як пакет, що включає інформацію про початок і кінець передавання даних, а також інформацію для контролю помилок.

Першим передається не біт даних, а старт-біт, що вказує на початок передавання даних (початок пакета). Цей біт використовується приймачем для синхронізації процесу читання даних, що випливають за старт-бітом (молодший біт даних йде першим). Після бітів даних може випливати біт парності (контрольний біт), що використовується для перевірки правильності отриманих даних. Існує два типи перевірки на парність. Перевірка на непарність («odd») означає, що число одиниць у пакеті даних, включаючи біт парності, повинно бути непарним (наприклад, 0x55 буде мати біт парності рівним 1, щоб зробити число одиничних бітів рівним п'яти, тобто непарним). Перевірка на парність («even»), навпаки, означає що число одиничних бітів повинно бути парним (наприклад, при передаванні числа 0x55 біт парності дорівнює 0).

У деяких мікроконтролерах значення біта парності повинно визначатися програмно, а потім поміщатися в регістр. Простий алгоритм реалізації цієї процедури полягає у виконанні логічної операції « виключне АБО» (XOR) над усіма бітами переданого байта. Для мікроконтролера 8051 дана процедура виконується за допомогою такої програми:

```
    mov Count, 8        Обробити 8 біт
    mov A,0             Очистити регістр «А»
P_Loop:                 Адреса повернення після обробки кожного біта
   xrl A, Char          Операція XOR над молодшими бітами
    rrc A               Зрушити Char, щоб перейти до наступного біту
   djnz Count,P_Loop    Повторити 8 разів
```

Молодший біт регістра «А» буде містити біт парності ((«even») для операнда «Char». Кількість одиничних бітів разом з бітом парності буде парним числом. Щоб реалізувати перевірку на непарність, необхідно інвертувати молодший біт у регістрі «А».

За бітом парності випливає стоп-біт, що використовується приймачем для обробки кінця передавання пакета.

Асинхронний пакет даних показаний на рис. 2.28. Існує набір параметрів, що повинний бути відомий при реалізації обміну. Одним з таких параметрів є число переданих бітів даних, що визначається типом прийма-

льного і передавального пристроїв. Пакет на рис. 2.28 містить тільки 5 біт даних (таке число бітів використовувалося в телетайпах), але можливі пакети довжиною до 8 біт.

Поряд з бітами парності («odd») чи непарності («even») можливі інші варіанти контрольних бітів: «no», «mark» і «space». «no» означає відсутність біта парності в пакеті. «mark» чи «space» означає, що замість біта парності завжди посилається „1” («mark») чи „0” («space»), відповідно. Ці варіанти контрольних бітів використовуються досить рідко – у тих випадках, коли необхідно дати приймачу додатковий час на оброблення пакета.

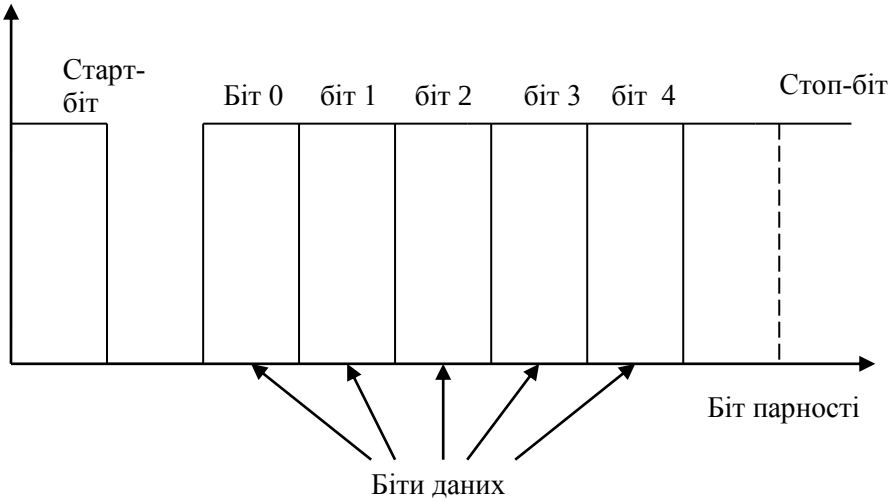


Рисунок 2.28 – Асинхронне послідовне передавання даних

Кількість стоп-бітів також може бути різна. Другий стоп-біт може вводитися для тієї ж мети, що і контрольні біти «mark» і «space» – щоб дати приймачу більше часу для обробки прийнятого пакета.

Практично всі сучасні пристрої використовують для асинхронного обміну формат даних «8-N-1», що означає передачу 8 біт даних, відсутність біта парності й один стоп-біт. Біт парності і додатковий стоп-біт, звичайно, не потребуються для послідовного зв’язку.

Найбільш популярний протокол асинхронного послідовного зв’язку називається „RS-232”, що у наш час є міжнародним стандартом. Це дуже старий стандарт, використовуваний для зв’язку комп’ютерів.

Асинхронний приймач чекає приходу старт-біту, коли на лінії встановлюється низький рівень. Через половину часу передачі одного біта (тривалість цього часу задається синхросигналом) лінія знову опитується, якщо на лінії усе ще встановлений низький рівень сигналу, то приймач чекає один період і зчитує дані (рис. 2.29). Якщо ж на лінії з’являється високий рівень сигналу, то приймач вважає, що відбулася помилка, і дані не приймаються. Цей метод використовується як при апаратній, так і при програмній реалізації асинхронного приймання даних. У програмно реалізованих приймачах використовуються програмні цикли для відліку затримок часу.

Інший розповсюджений метод асинхронної послідовної передачі даних — це використання коду типу «Манчестер» («manchester»). При цьому методі передача кожного біта даних синхронізується імпульсом, а значення біта („0” чи „1”) визначається проміжком часу до наступного імпульсу (рис. 2.30). Після передачі заданого числа бітів даних випливає стоп-імпульс, а потім приймання даних припиняється. Особливість манчестерського кодування полягає в тім, що біт якісно відрізняється від „1” чи „0”. Це дозволяє приймачу визначити чи є дані, що надходять, початком чи серединою посланого пакета (в останньому випадку дані не будуть прийматися до приходу старт-біта). Манчестерське кодування гарно підходить для використання у випадках, коли потік переданих даних може бути легко перерваний. Тому такий метод передавання даних є основним для зв’язку за допомогою інфрачервоного випромінювання, наприклад, у пульті дистанційного керування телевізором.

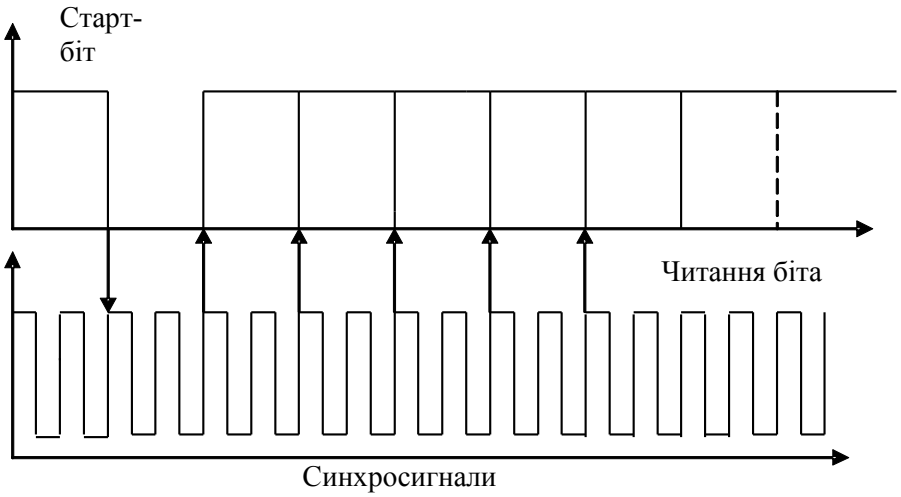


Рисунок 2.29 – Читання даних при асинхронному послідовному обміні

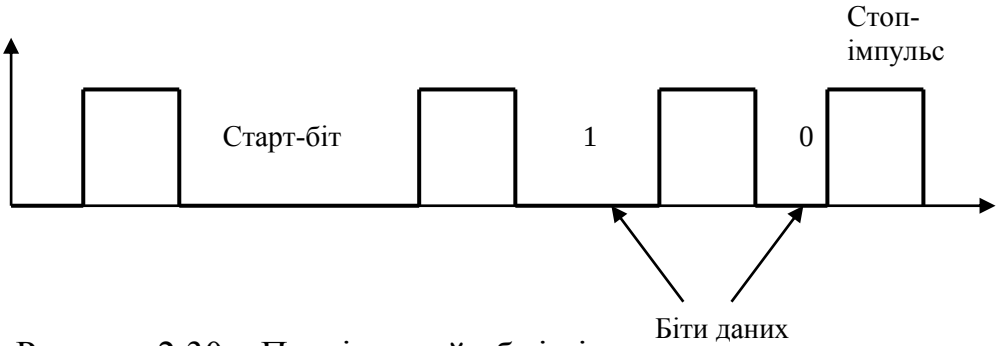


Рисунок 2.30 – Послідовний обмін із манчестерським кодуванням

Синхронний послідовний обмін

При реалізації синхронного обміну разом з даними посилається синхросигнал, що використовується приймачем для стробування даних (рис. 2.31).

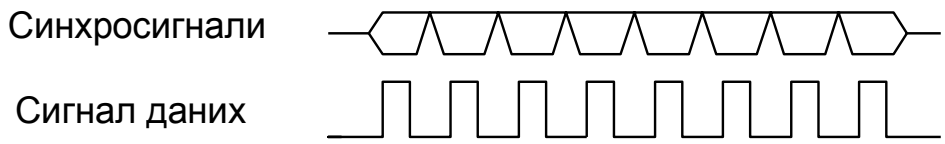


Рисунок 2.31 – Форма сигналів при синхронному передаванні даних

Типова схема для перетворення послідовних даних у паралельні показана на рис. 2.32. У цій схемі використовуються дві мікросхеми 8-розрядних регістрів типу 74LS374. Для більшості програм не потрібно вмикання другого регістра. Це перетворення може також бути виконане за допомогою спеціальної мікросхеми, але деякі воліють використовувати 8-розрядні регістри, тому що їх, звичайно, легше знайти, ніж інші типи мікросхем ТТЛ.

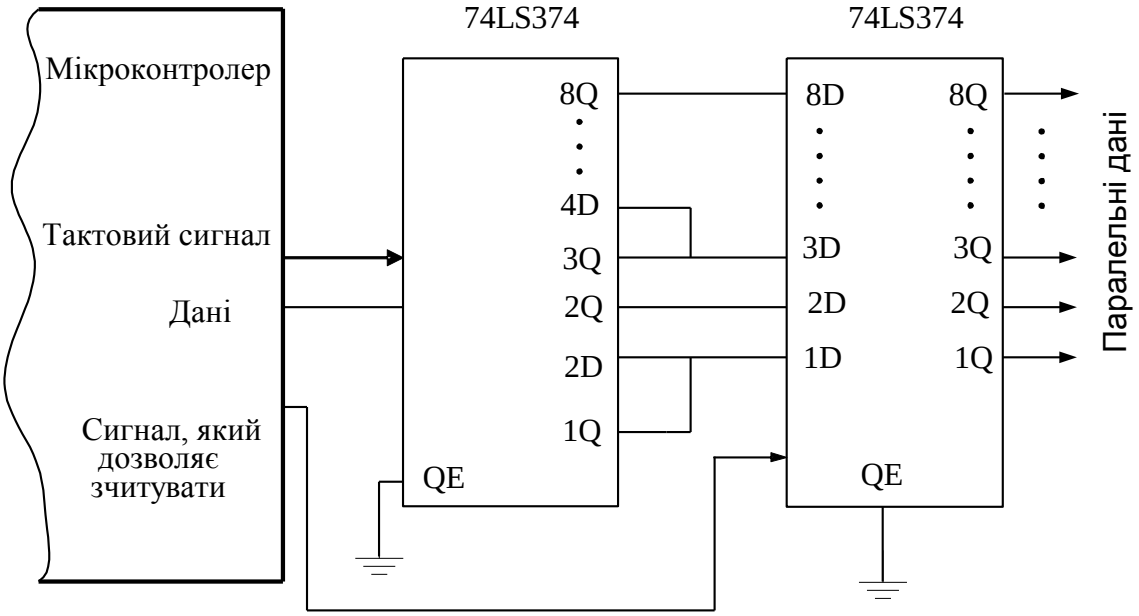


Рисунок 2.32 – Схема перетворення послідовних даних в паралельні

Існує два основних протоколи для синхронного зв’язку: Microwire і SPI. Ці методи застосовуються для взаємодії з різними мікросхемами (таких, як послідовний EEPROM у BASIC Stamps). Хоча стандарти Microwire і SPI дуже схожі, існують деякі розходження, про які необхідно згадати.

Дані протоколи частіше використовуються для синхронного послідовного передавання даних, ніж для об’єднання мікроконтролерів у єдину мережу. У цих протоколах кожен пристрій адресується індивідуально, хоча лінії передачі даних і синхронізації можуть бути загальними для багатьох пристроїв. Якщо сигнал дозволу вибірки (chip select) пристрою не активний, то цей пристрій ігнорує лінії даних і синхронізації. У кожен момент

часу тільки один з підімкнутих пристроїв може бути ведучим (master), тобто мати можливість задавати режим роботи шини (рис 2.33).

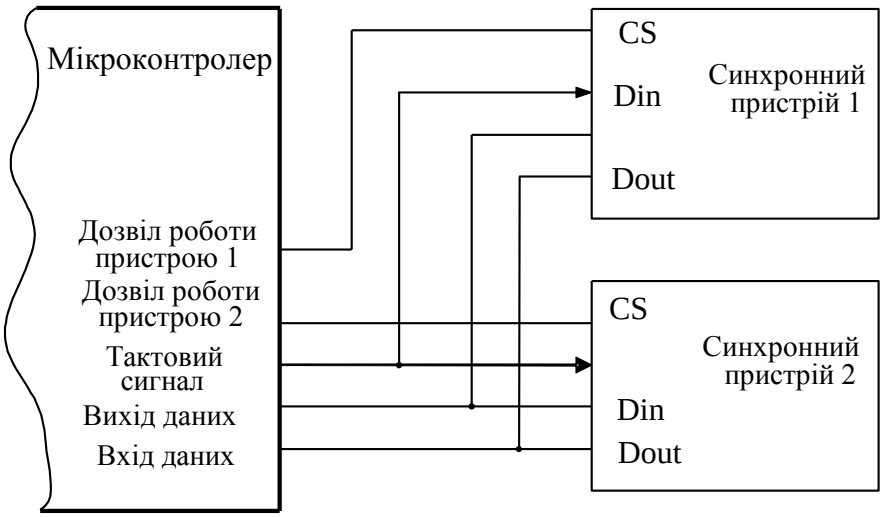


Рисунок 2.33 – Синхронна послідовна шина

Якщо синхронний послідовний порт вбудований у мікроконтролер, то передавальна схема має вигляд, показаний на рис 2.34.

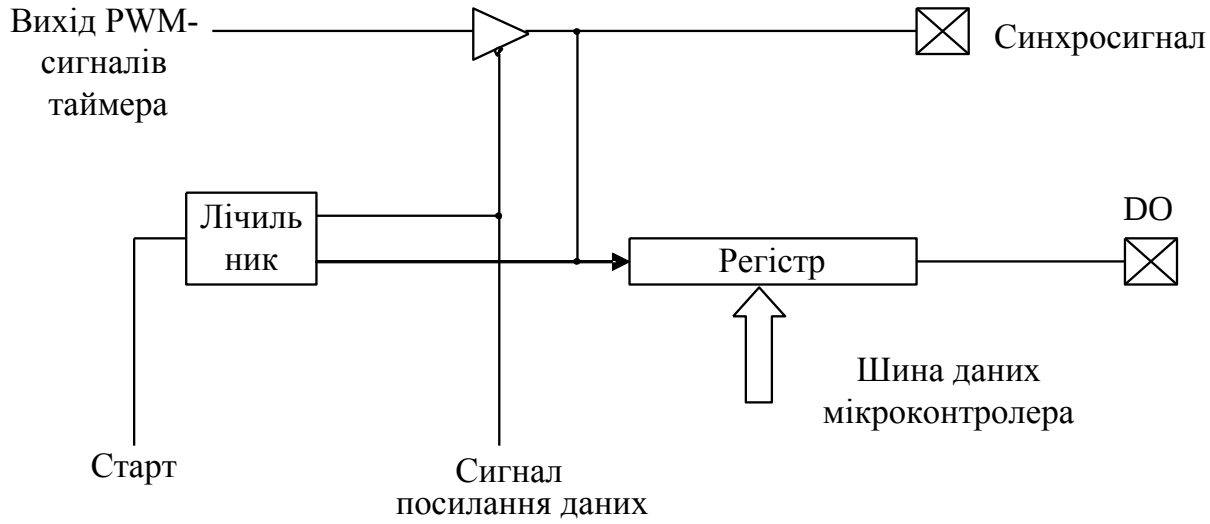


Рисунок 2.34 – Схема синхронного виведення даних

Ця схема виводить 8-розрядні дані. При реалізації протоколів, аналогічних стандарту Microwire, де спочатку видається старт-біт, цей біт посилається за допомогою команд читання і записування в порт введення-виведення. Схожа схема використовується для приймання даних, де дані, що надходять, спочатку послідовно вводяться в зсувовий регістр і потім зчитуються мікроконтролером.

Протокол MICROWIRE

Протокол Microwire забезпечує передавання даних зі швидкістю до 1 Мбіт у секунду. В одному пакеті передається шістнадцять бітів даних.

На рис. 2.35 показана форма сигналів при читанні 16 біт даних. Після вибору мікросхеми і посилки старт-біта передається 8-розрядна команда (позначена як «OP1», «OP2», «A5» — «A0» на рис. 2.35), потім випливають 16-розрядна адреса (її наявність не є обов'язковою) і 16 біт даних. При максимальній швидкості передачі 1 Мбіт у секунду тактовий сигнал змінюється кожні 500 нс. Передані біти повинні видаватися на лінію за 100нс до надходження переднього фронту тактового сигналу. Читання даних повинно відбуватися за 100 нс до надходження заднього фронту тактового сигналу. Хоча ці вимоги виконуються більшістю пристроїв, необхідно переконатися, що пристрій, з яким здійснюється зв'язок, відповідає даним умовам.

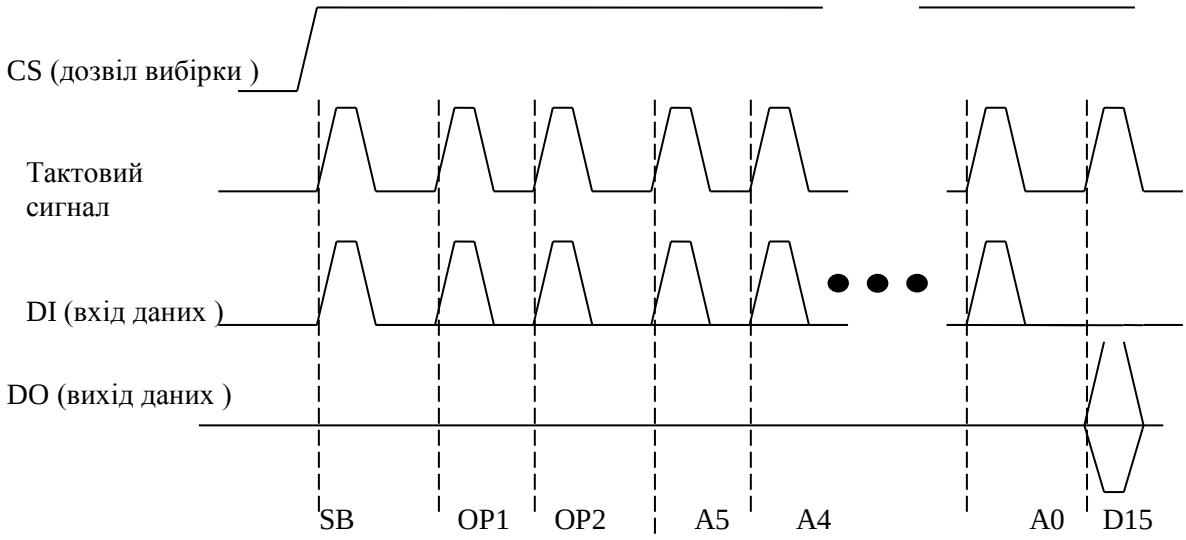


Рисунок 2.35 – Читання даних в протоколі Microwire

Протокол SPI

Протокол SPI схожий на протокол Microwire, але має кілька відмінностей:

- SPI здатний передавати дані зі швидкістю до 3 Мбіт у секунду;
- розрядність даних у SPI пакеті дорівнює 8 біт;
- передавач у SPI має можливість призупинити передавання даних;
- дані в SPI можуть передаватися у вигляді безлічі байтів, що називаються «блок» чи «сторінка».

Як і в протоколі Microwire, у SPI спочатку передається байт, що містить команду для приймального пристрою. Потім йде необов'язкова 16-розрядна адреса, після чого випливають 8-розрядні дані. Як було зазначено вище, протокол SPI дозволяє передавати кілька байтів (рис. 2.36). У протоколі SPI використовується симетричний тактовий сигнал, що має однакову тривалість високого і низького рівня. Вихідні дані повинні бути видані на

лінію принаймні за 30 нс до надходження переднього фронту тактового сигналу, а зчитування повинне відбуватися за 30 нс до заднього фронту.

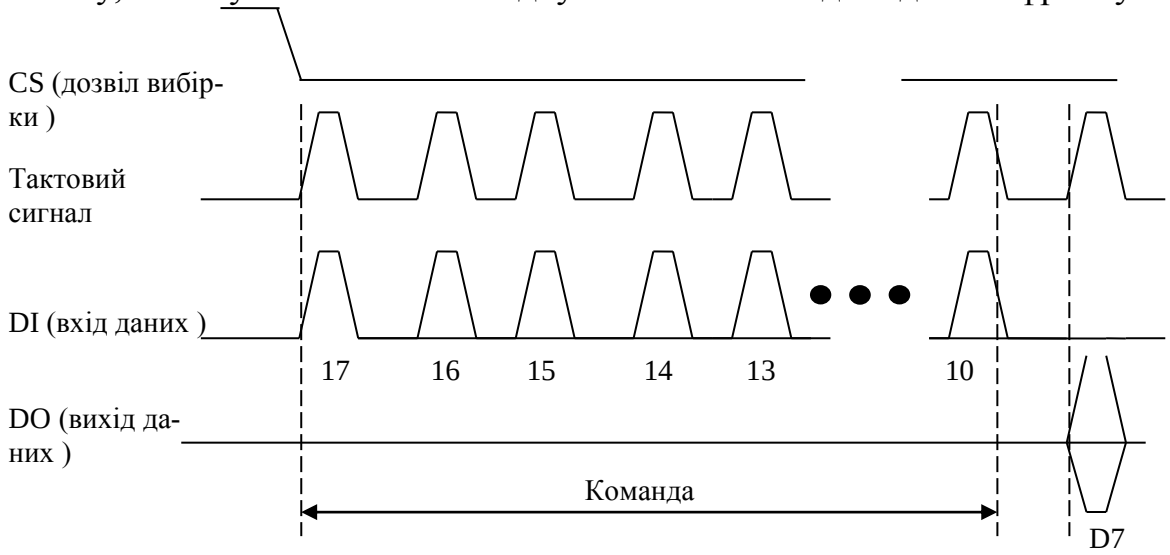


Рисунок 2.36 – Записування даних в протоколі SPI

При реалізації протоколів Microwire чи SPI можна спростити схему з'єднання, якщо підімкнути вхід Din і вихід Dout до однієї лінії (рис. 2.37). При такому способі з'єднання мікроконтролер повинен вимкнути свій вихідний драйвер, коли закінчить послідовне передавання даних. Після цього він може приймати дані, що надходять від іншого пристрою. Резистор між виводами даних слугує для обмеження струму в тих випадках, коли і мікроконтролер, і периферійний пристрій видають дані на лінію зв'язку.

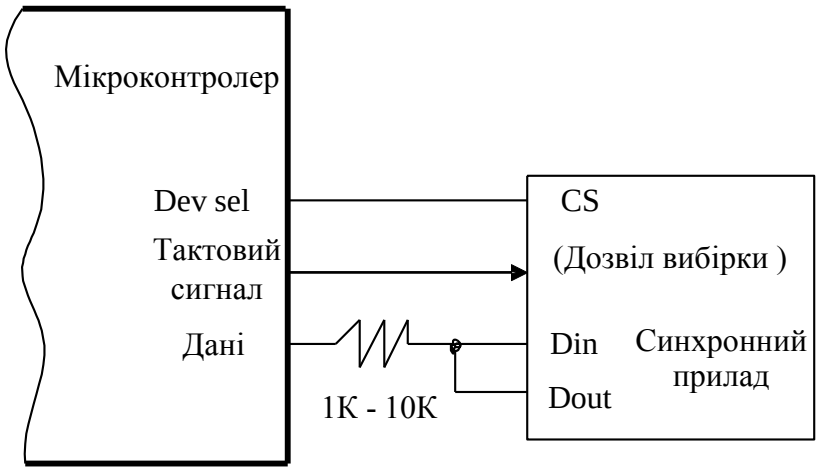


Рисунок 2.37 – Підімкнення входу "DI" та виходу "DO" до однієї лінії

Мережі

Коли термін «мережа» вживається стосовно до мікроконтролерів, то мають на увазі шини, що використовуються для підімкнення додаткових пристроїв і для забезпечення зв'язку між мікроконтролерами. У такий спо-

сіб мережа мікроконтролерів відрізняється від локальної мережі такої, як «Ethernet», що ймовірно приходить на думку при слові «мережа».

Існує безліч стандартів (у число яких входять описані вище Microwire і SPI), що можуть привести до здивування з приводу того, що є і що не є мережею. Далі під мікроконтролерною мережею будемо розуміти окрему лінію зв'язку (провід) і безліч підімкнутих до неї пристроїв, що можуть ініціювати передавання повідомлень і посилати відповідний відгук на отримане повідомлення.

У таких мережах є ведучий («master») – інтелектуальний пристрій, що може ініціювати передавання даних. Пристрої, що відповідають на запити, але не можуть їх ініціювати, називаються – ведені («slaves»). Мікроконтролерна мережа може мати декілька ведучих пристроїв, у цьому випадку мережний протокол вимагає увімкнення схеми арбітражу, що дозволить різним «ведучим» пристроям передавати дані не порушуючи інших повідомлень.

Звичайно, в мікроконтролерній мережі передається відносно мало даних, тому необхідна для них пропускна здатність дуже скромна в порівнянні з мережею «Ethernet». Часто мікроконтролерна мережа має пропускну здатність порядку декількох байтів у секунду, тоді як мережа персональних комп'ютерів може передавати декілька Мбайтів у секунду.

Далі будуть описані два найбільш популярні види мереж для мікроконтролерів, хоча подана в книзі інформація про ці мережі є досить повною, її недостатньо для розробки мережних додатків. Для одержання більш докладної інформації звертайтеся до посібників з використання мікроконтролерів чи до опису стандартів. Подана інформація дає розуміння основ описуваних протоколів, щоб можна було оцінити їхню придатність для конкретного застосування.

Протокол I2C

Найбільш популярний протокол для мережі мікроконтролерів – I2C, що призначений для зв'язку пристроїв у багатопроцесорних системах. Цей стандарт був розроблений компанією Philips наприкінці 70-х років як метод реалізації інтерфейсу між мікропроцесорами і периферійними пристроями, що не вимагає прокладання численних ліній для передавання між пристроями всіх розрядів адреси, даних і сигналів керування. Протокол I2C дозволяє розділяти мережні ресурси між декількома ведучими процесорами («multimastering»).

Шина I2C містить дві лінії: лінія SDA, що слугує для передавання даних, і лінія SCL, по якій передається синхросигнал, використовуваний для стробування даних. Обидві лінії підімкнуті через резистори до шини живлення («підтягнуті» до високого рівня потенціалу), що дозволяє декільком пристроям керувати їхнім станом шляхом з'єднання за схемою «монтажне І». Шина I2C для керування стереосистемою може мати вигляд, показаний на рис. 2.38.

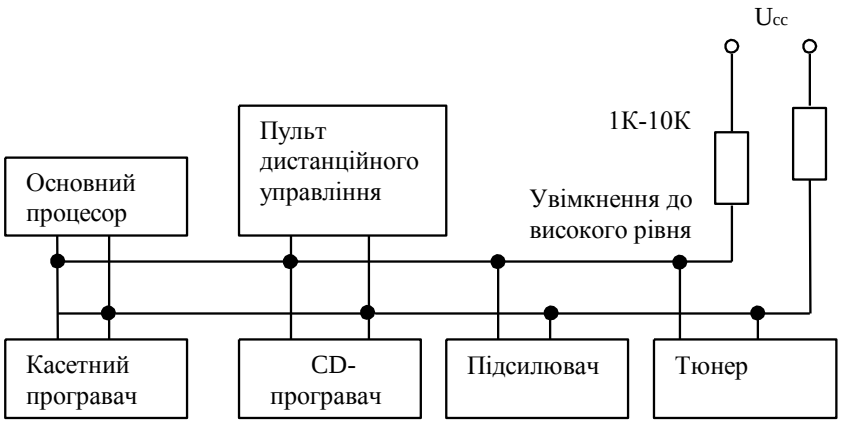


Рисунок 2.38 – Приклад I2C мережі

Двопровідна лінія використовується для визначення початку передавання даних, а також для передавання самих даних. Щоб почати передавання даних, шина переводиться в стартовий стан. При відсутності переданих даних шина знаходиться в стані (пасивному), що очікує («idle»). При цьому на обидві лінії сигнали не надходять, і на них встановлений високий рівень сигналу (потенціал U_{cc}). Щоб ініціювати передавання даних, ведучий пристрій, запитує керування шиною, установлює низький рівень спочатку на лінії SDA, а потім на лінії SCL (стартовий стан). У процесі пересилання даних такий стан шини є неробочим, тому що приймання переданих даних виконується тільки при високому (активному) рівні синхросигналу на лінії SCL. Щоб закінчити передавання даних виконуються зворотні дії: на лінії SCL встановлюється високий рівень сигналу, а потім у такий же стан переводиться лінія даних SDA (рис. 2.39).

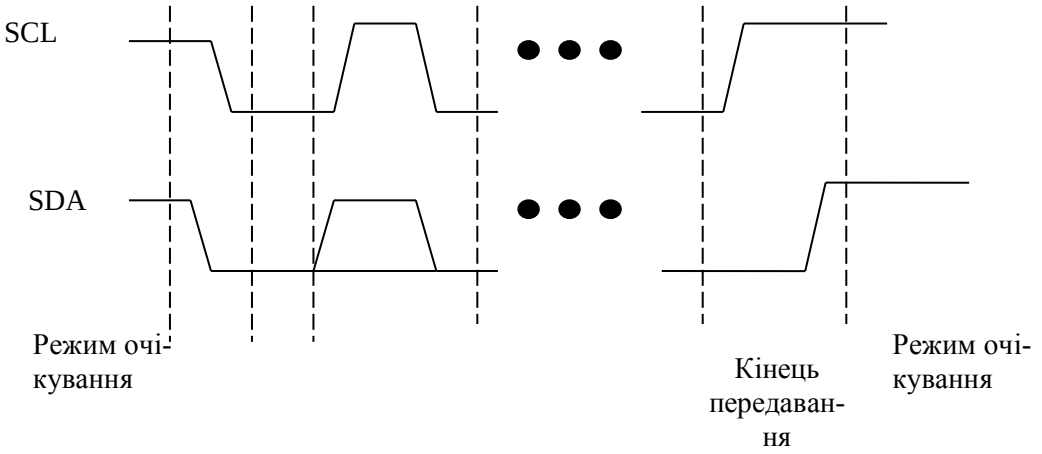


Рисунок 2.39 – Форма сигналів на шині I2C

Дані передаються синхронним способом, причому першим посилається старший біт (рис. 2.40). Після передачі 8 біт ведучий пристрій переводить лінію даних у "0" стан, що плаває, очікуючи підтвердження приймання даних від веденого пристрою. Таким підтвердженням є встановлення веденим пристроєм низького рівня сигналу на лінії SDA. Після біта під-

твердження на обох лініях встановлюється низький рівень. Потім виконується пересилання наступного байта або шина переводиться в стан кінця передачі. Це означає, що передача завершена, і приймач може готуватися до наступного запиту даних.

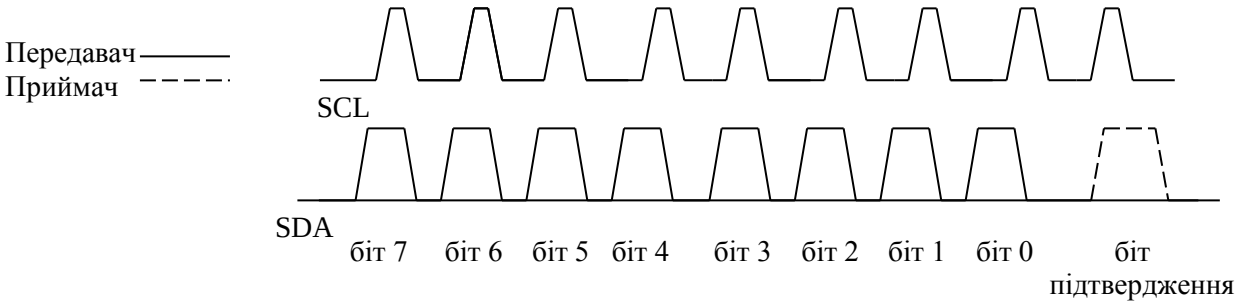


Рисунок 2.40 – Передавання байта по шині I2C

Існує дві максимальні швидкості передавання даних по шині I2C: «стандартний режим» ~ до 100 Кбіт/с і «швидкий режим» - до 400 Кбіт/с (рис. 2.41).

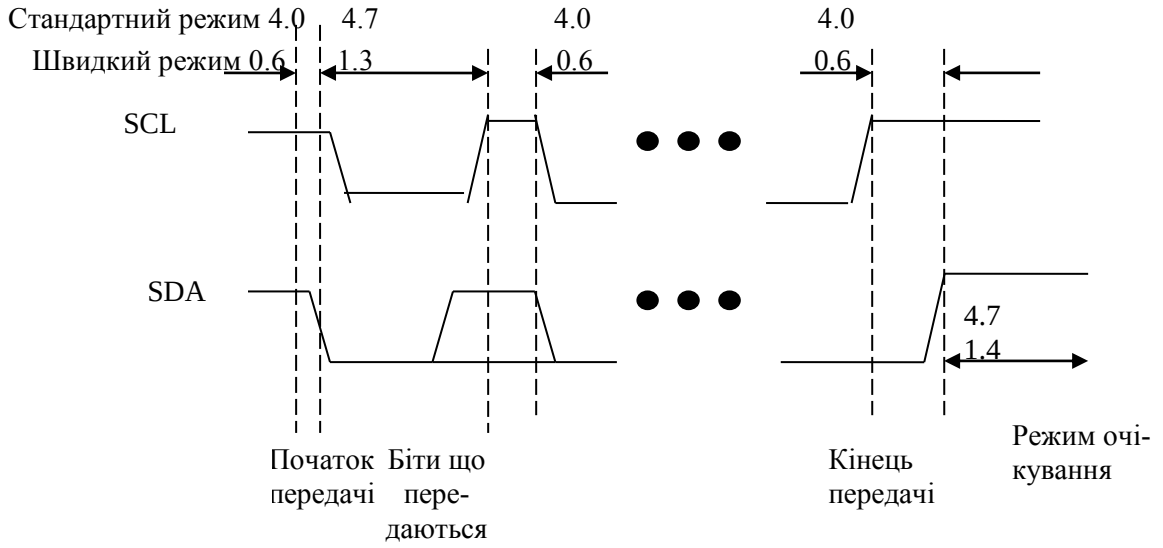


Рисунок 2.41 – Тимчасова діаграма сигналів на шині I2C

Формат команди, що надходить від ведучого пристрою до веденого, показаний на рис. 2.42. Адреса одержувача містить 7 біт. Існує незатверджений («вільний») стандарт, відповідно до якого чотири старших біти використовуються для зазначення типу пристрою, а наступні три біти використовуються для вибору одного з восьми пристроїв цього типу або служать для більш точного визначення типу пристрою. Оскільки цей стандарт не є обов'язковим, то деякі пристрої вимагають зазначення в якості трьох останніх адресних бітів визначених кодів, тоді як інші, наприклад, мікросхеми пам'яті EEPROM використовують ці біти для вибору адресата усередині пристрою. Існує також 10-розрядний стандарт для передавання адреси, у якому перші 4 біти містять 1, а біт, що випливає, має значення 0,

останні 2 біти є старшими бітами адреси, а завершальні 8 біт адреси передаються в наступному байті. Усе це означає, що дуже важливо розподілити адреси між пристроями, підімкнутими до шини.

Перші чотири біти адреси, звичайно, служать для визначення типу обираних пристроїв у відповідності з такими умовами:

0000 - Зарезервована адреса
0010 - Синтезатор голосу
0011 - Аудіо - інтерфейс
0100 - Звуковий генератор
0111 - Рідкокристалічний чи світлодіодний дисплей
1000 - Відеоінтерфейс
1001 - Аналогово-цифровий і цифроаналоговий інтерфейси
1010 - Послідовна пам’ять
1100 - Керування радіоприймачем
1101 - Годинник/календар
1111 - Зарезервовано для використання 10-розрядної адреси

Початок передавання

Кінець передавання

1 – Rw - тип обміну
2 – Ack - біт підтвердження
3 – додаткові дані чи початок наступного передавання

Рисунок 2.42 – Формат передавання даних по шині I2C

Перш, ніж закінчити обговорення протоколу I2C, варто звернути увагу на такі обставини. У деяких пристроях потрібне повторне посилання стартового біта, щоб скинути приймальний пристрій у вихідний стан для приймання наступної команди. Наприклад, при читанні з EEPROM-пам’яті з послідовною вибіркою перша команда посилає адресу осередку, з якого виконується зчитування, а друга команда виконує читання даних за цією адресою.

Варто також звернути увагу на можливість ініціювання процесу передавання даних декількома ведучими мікроконтролерами («multimastering»). Це може привести до виникнення колізій, коли два пристрої намагаються керувати шиною одночасно. Якщо один мікроконтролер взяв керування шиною, тобто встановив стартовий стан, до того, як інший спробує зробити те ж саме, то це не викликає проблем. Проблема виникає, коли кілька пристроїв ініціюють стартовий стан одночасно, і потрібно зробити арбітраж їхніх запитів.

На практиці здійснити арбітраж у цьому випадку досить просто. Під час передавання даних обидва передавачі точно синхронізують тактові ім-

пульси. Якщо при передаванні адреси біт, що повинний мати значення „1”, насправді приймає значення „0”, то це вказує на те, що шина зайнята іншим пристроєм. У цьому випадку ведучий пристрій вимикається від шини і чекає, коли наступить стан «кінець передачі», після якого повторює запит. Можливо, це важко зрозуміти за приведеним описом. У наступному розділі «Протокол CAN» буде показано як те ж саме відбувається з використанням асинхронної шини CAN, що має багато загального із шиною I2C.

Протокол I2C може бути легко реалізований програмним шляхом. Але при цьому швидкий режим не може бути реалізований через перевантаження процесора, навіть стандартний режим 100 Кбіт/с може виявитися занадто швидким для деяких мікроконтролерів. Програмна реалізація щонайкраще підходить тоді, коли в мережі є тільки один ведучий пристрій. У цьому випадку немає необхідності синхронізуватися з іншими пристроями чи приймати повідомлення від інших ведучих пристроїв, що працюють із занадто великою швидкістю, що не забезпечується при програмній реалізації.

Протокол CAN

Протокол CAN (Controller Area Network) був розроблений компанією Bosch кілька років назад як мережне рішення для зв’язку комп’ютерних систем, застосовуваних в автомобілях. У той час не існувало єдиного стандарту для зв’язку цифрових пристроїв в автомобілях. До появи протоколу CAN (чи протоколу J1850, що є аналогічним американським стандартом) автомобілі містили до трьох миль проводів вагою понад 90 кг, що зв’язували різні автомобільні електронні пристрої. Протокол CAN був розроблений, щоб задовольняти такі вимоги:

- висока швидкість обміну (до 1 Мбіт/с);
- нечутливість до електромагнітних перешкод;
- простота, невелика кількість рознімних контактів (для забезпечення механічної надійності);
- легкість підімкнення і видалення пристроїв.

Протокол CAN подібний протоколу J1850 і ґрунтується на тих же перших двох рівнях семирівневої моделі OSI, однак ці два стандарти електрично несумісні. Стандарт на протокол CAN з’явився раніше, тому даний стандарт реалізується практично у всіх моделях європейських і японських автомобілів, а в наш час активно використовується й американськими автомобільними компаніями.

Протокол CAN реалізується з використанням операції «монтажне І», що використовується також шиною I2C. Передавання даних на фізичному електричному рівні реалізується за допомогою драйверів, що відповідають стандарту RS-485, що забезпечують видачу на лінію зв’язку диференціальної напруги. Така мережа буде працювати навіть якщо один із двох провідників закорочений або обірваний, що особливо важливо для забезпе-

чення надійності автомобільного устаткування. Використання з'єднання, що реалізує «монтажне І», дозволяє виконувати арбітраж між різними ведучими пристроями: коли вихідні драйвери пристроїв активні, то шина CAN переводиться в низький стан, аналогічно I2C.

Приклад реалізації цього методу арбітражу показаний на рис. 2.43. Коли сигнал, видаваний драйвером, не збігається з рівнем, встановленим на лінії передавання даних (наприклад, при видачі „1" на шині виявляється „0"), то драйвер зупиняє передавання даних доти поки не завершиться пересилання поточного повідомлення. Цей простий і ефективний спосіб арбітражу виключає необхідність повторної посилки повідомлення при виникненні колізій на шині.

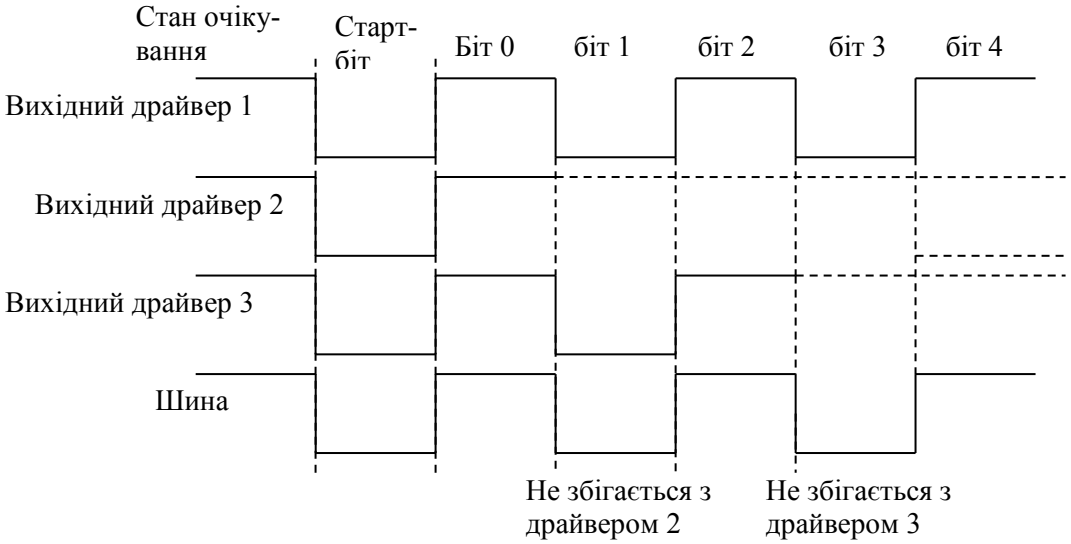


Рисунок 2.43 – Арбітраж при передаванні по шині CAN

Кожне повідомлення являє собою окремий кадр у потоці даних, що пересилається. Кадр передається як фрагмент цього асинхронного послідовного потоку, у якому пересилання даних не супроводжується посиланням синхросигнала. При цьому приймач і передавач повинні працювати на одній частоті: звичайно швидкість обміну встановлюється в межах від 200 Кбіт/с до 1 Мбіт/с. Формат кадру передавання даних показаний на рис. 2.44.

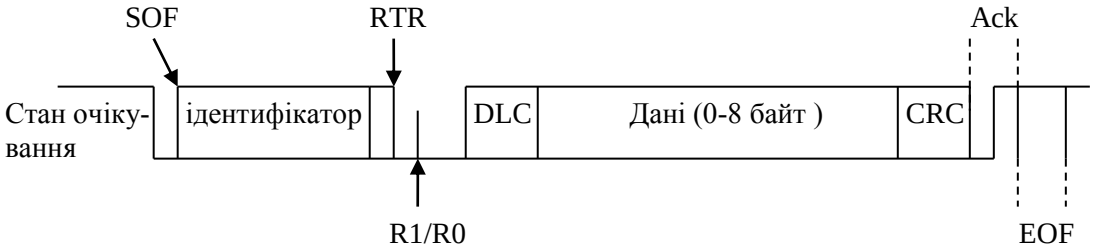


Рисунок 2.44 – Кадр передавання даних по шині CAN
із використанням 11-бітного ідентифікатора

При використанні протоколу CAN нульове значення біта називається «домінантним», а одиничне значення - «рецесивним» (за аналогією з генами в біології).

Різні поля кадру мають таке призначення:

SOF (Start Of Frame) – початок кадру, одиничний домінантний біт;

Ідентифікатор – 11-и чи 19-бітний ідентифікатор повідомлення;

RTR – біт, який вказує, що ведучий пристрій є передавачем (при RTR=1) або приймачем (при RTR=0) даних;

r1/r0 – зарезервовані біти, що повинні бути домінантними;

DLC – 4 біти, що вказують кількість переданих байтів;

Data – від 0 до 8 переданих байтів даних, де старший біт йде першим;

CRC – 15 бітний код контрольної суми, за яким випливає рецесивний біт;

Ack – 2-бітне поле підтвердження готовності (домінантний і рецесивний біти);

EOF (End Of Frame) – кінець кадру (поле, що містить не менше 7 рецесивних біти).

Останнє важливе зауваження, що стосується CAN, полягає в тому, що пристроям не даються специфічні імена чи адреси, тобто ідентифікуються повідомлення (використовуючи 11-и чи 19-и бітний ідентифікатор повідомлення). Цей метод адресації може забезпечити дуже гнучкий обмін повідомленнями.

Кадр у протоколі CAN має складний формат, що ускладнює його обробку в процесі передавання і приймання повідомлення. Така обробка може виконуватися як апаратно, так і програмно, але рекомендується використовувати тільки апаратну реалізацію CAN. Ряд виробників вбудовують CAN-інтерфейс у мікроконтролери, що випускаються ними. Також існує декілька пристроїв, що серійно випускаються, (особливо популярна мікросхема Intel 82527), що ефективно і з мінімальними додатковими витратами виконують функції CAN інтерфейсу.

2.16 Аналогове введення-виведення

Світ, що оточує мікроконтролер, складається не тільки з чорного і білого (одиниць і нулів), у дійсності там є багато відтінків сірого кольору (значень між нулем і одиницею). Часто мікроконтролер повинен взаємодіяти з аналоговими пристроями, які працюють з сигналами, що мають рівень між напругою живлення U_{cc} і «землею», вводити і виводити такі аналогові сигнали. Багато моделей мікроконтролерів у різних сімействах містять аналогово-цифрові (ADC -Analog-to-Digital Converter) і цифровоаналогові (DAC - Digital-to-Analog Convener) перетворювачі. Цей розділ містить вступну інформацію для реалізації введення-виведення аналогових сигналів. У ньому не будуть підніматися питання передавання аналогових

даних, що розглядалися раніше в підрозділі «Таймери», де була описана процедура читання і посилання даних з використанням широтно-імпульсної модуляції.

Існує три способи введення аналогового сигналу в мікроконтролер. Перший спосіб – використання датчика, за допомогою якого мікроконтролер визначає фізичне положення движка потенціометра. Другий спосіб – увімкнення аналогового компаратора, що визначає чи знаходиться значення напруги, що надходить, вище чи нижче заданого рівня (опорної напруги). Третій тип – використання мікроконтролера з інтегрованим на кристалі аналого-цифровим перетворювачем (АЦП), що забезпечує вимірювання значення напруги, що надходить на вхід. Кожний з цих джерел має визначені переваги для різних областей застосування.

У першому способі аналого-цифрове перетворення фактично не реалізується, а поточне значення опору потенціометра визначається за допомогою введення-виведення цифрових даних. Для визначення опору потенціометра до виводу мікроконтролера вмикається простий RC-ланцюг (рис 2.45). Опір визначається шляхом вимірювання часу, протягом якого потенціал на конденсаторі залишається більший порога перемикання. Чим більший опір, тим більше число "1" буде підраховано на вході за час вимірювання (рис. 2.46). Щоб виконати вимірювання, вивід паралельного порту переводиться в режим виходу, на якому встановлюється "1" (високий потенціал). Конденсатор розряджається через опір Rsub, що обмежує струм, запобігаючи коротке замикання на початку розряду. Коли конденсатор цілком розрядився, вихідний драйвер закривається, і конденсатор починає заряджатися через потенціометр. Вимірювання закінчується, коли напруга на виводі упаде нижче порога перемикання. Звичайно, для вимірювання часу використовується таймер.

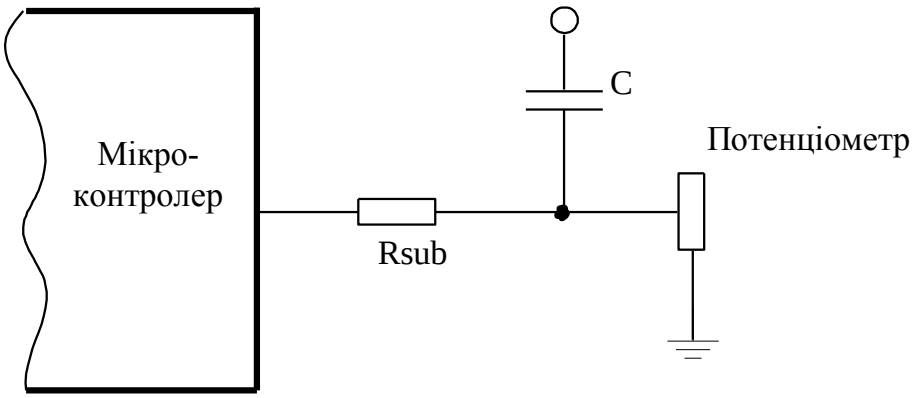


Рисунок 2.45 – Використання цифрового входу для визначення опору потенціометра

Якщо простежити за цією операцією за допомогою осцилографа, то можна спостерігати форму сигналу, показану на рис. 2.46. Для визначення значень R і C можна використовувати вираз: $t = 2.2 \times RC$, де t — час заря-

джання конденсатора. Опір Rsub, звичайно, вибирається в діапазоні від 100 Ом до 200 Ом. Ця схема вимірювання дає не зовсім точне значення опору R через розкид ємності конденсатора а також вплив нелінійного вихідного опору CMOS-входу мікроконтролера. Більш точні результати можна одержати, використовуючи прецизійний конденсатор, але його важко знайти і він занадто дорогий.

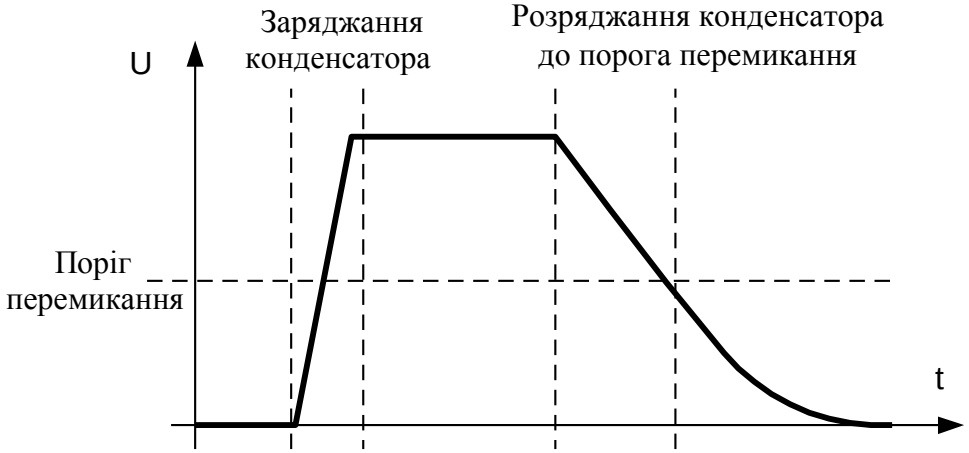


Рисунок 2.46 – Зміна вхідного потенціалу при зміні опору потенціометра

Можна задатися питанням де і чому використовується цей метод, якщо він не дуже точний. На практиці ця схема використовується в додатках, що не вимагають високої точності вимірювання. Дана схема може визначати відносне положення движка потенціометра. Наприклад, у комп'ютерах IBM PC така схема використовується для визначення поточного положення джойстика. Неточність вимірювання полягає в тому, що користувача просять перемістити джойстик у крайні положення, щоб зробити його калібрування. Звичайно, в такій схемі використовується танталовий конденсатор ємністю 0,1 мкф і потенціометр з опором 10 кОм, при цьому час заряду складає близько 22 мс. Цього значення досить, щоб мікроконтролер зміг виміряти його з необхідною точністю, у той же час така затримка занадто мала, щоб її міг помітити користувач. Використання танталових конденсаторів 0,1 мкф. найбільш зручно, тому що такі конденсатори використовуються для розв'язки живлення, тому вони завжди є в наявності. Проблема полягає в тім, що розкид ємностей для танталових конденсаторів може досягати 100% від їхнього номіналу. Це означає що необхідно робити калібрування перед використанням такого пристрою.

Наступний спосіб аналого-цифрового перетворення, звичайно, використовуваний у мікроконтролерах, – це аналоговий компаратор напруг. Компаратор являє собою просту схему, що порівнює дві напруги: вхідну й опорну (U_{ref}), і встановлює на виході 1, якщо вхідна напруга більша, ніж опорна (рис. 2.47). Цей спосіб найбільше зручно використовувати в таких пристроях, як термостати, де необхідно контролювати досягнення визна-

ченого рівня вимірюваної величини, що задається значенням вхідної напруги.

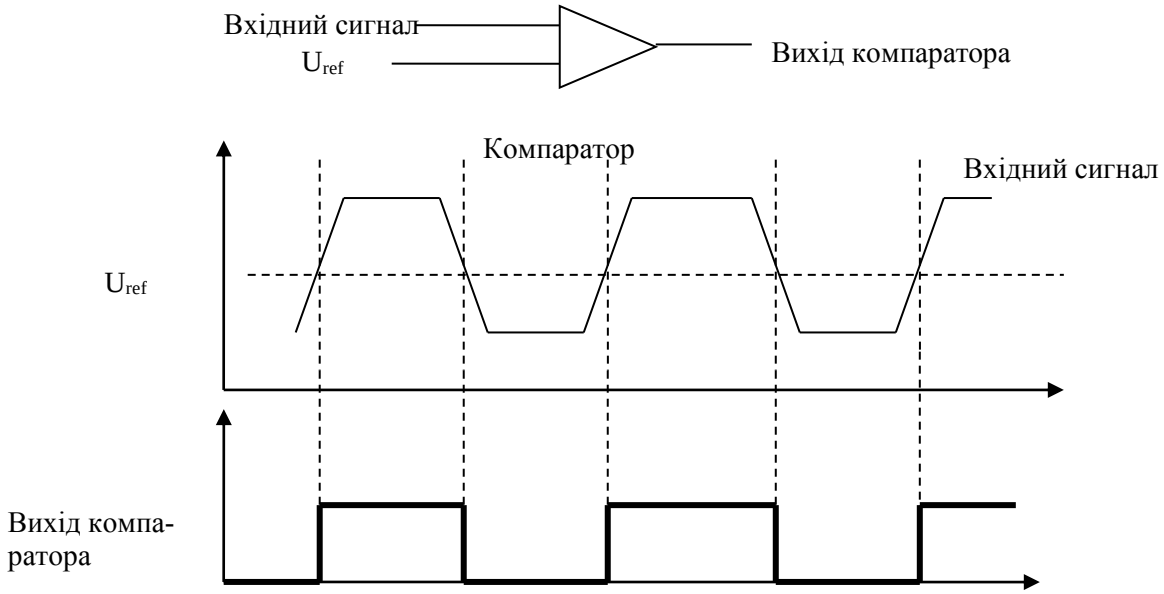


Рисунок 2.47 – Сигнал на вході та виході компаратора

Часто в мікроконтролерах, що використовують компаратори, опорну напругу формують всередині за допомогою резистивного подільника й аналогового мультиплексора, що робить вибір необхідної вихідної напруги (рис. 2.48). Така схема забезпечує одержання деякого набору опорних напруг. Використовується простий алгоритм перебирання набору різних опорних напруг до спрацьовування компаратора. Опорна напруга, при якій відбувається перемикання компаратора, відповідає значенню вхідної напруги, що надходить. Дана схема дає наближене значення вхідної напруги, тому що звичайна напруга U_{ref} задається з досить великим кроком. Наприклад, якщо схема забезпечує 8 рівнів опорної напруги, то при діапазоні напруг 5 В крок складе більш 700 мВ.

Інший спосіб аналого-цифрового перетворення – це використання паралельно увімкнутих компараторів (рис. 2.49). Цей метод є найбільш швидким у порівнянні з іншими методами перетворення ADC. Час перетворення визначається затримкою компараторів і пріоритетного дешифратора. Цей спосіб є відносно дорогим, тому що вимагає використання великого числа компараторів. Наприклад, щоб одержати 8-розрядну точність перетворення буде потрібно 256 компараторів.

В останньому способі аналого-цифрового перетворення використовується компаратор і аналогове джерело напруги (генератор пилкоподібної напруги), що лінійно збільшується, починаючи з 0 В и до U_{cc} (рис. 2.50). Ця схема називається інтегрувальним АЦП. На початку перетворення відбувається скидання таймер, і на виході генератора встановлюється 0 В. Потім відбувається запуск таймера і генератора. Коли напруга на виході генератора перевищить значення U_{in}, таймер зупиняється, і виробляється

сигнал «кінець перетворення» (ADC stop), що може викликати переривання мікроконтролера. При цьому вміст таймера буде пропорційний значенню напруги U_{in}.

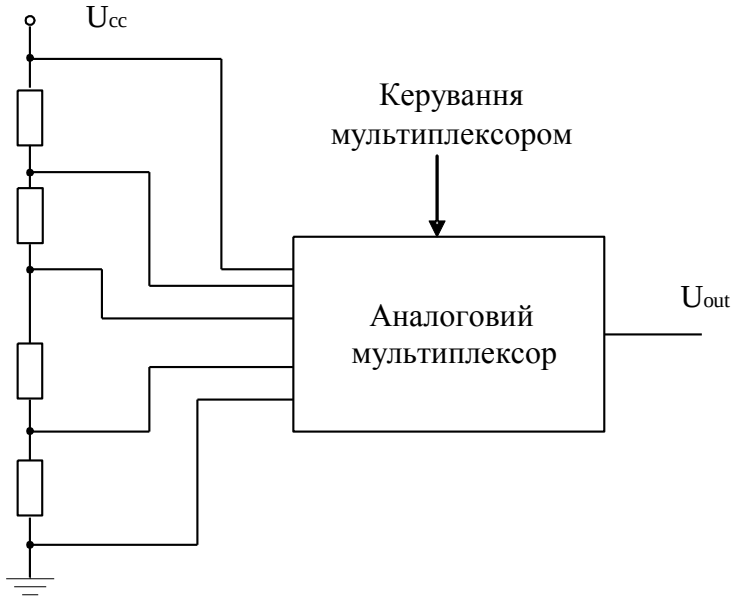


Рисунок 2.48 – Отримання опірних напруг за допомогою подільника

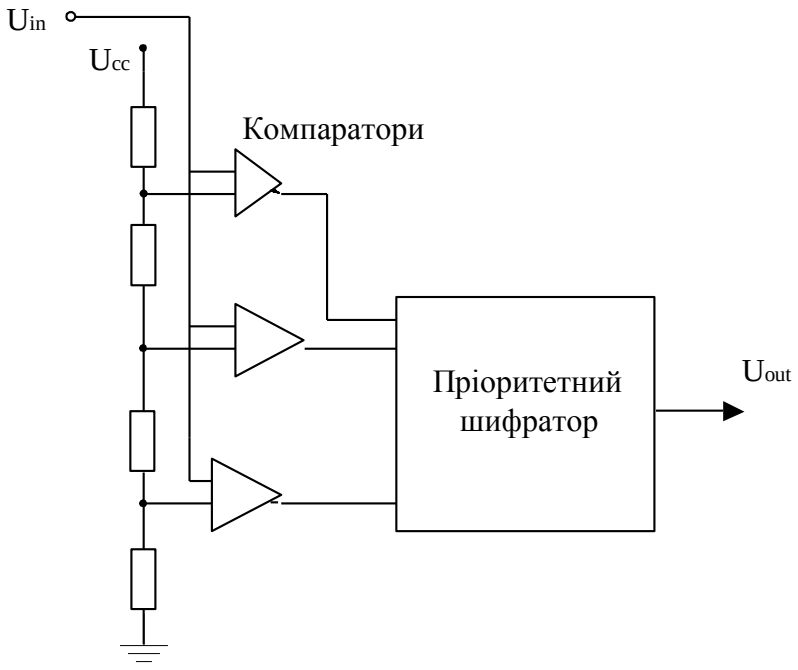


Рисунок 2.49 – Паралельний цифроаналоговий перетворювач

Цей метод забезпечує дуже високу точність перетворення. Однак при його реалізації виникає ряд проблем. Перша – це час, необхідний для перетворення. Чим вища напруга U_{in}, тим більший час перетворення. Деякі мікроконтролери мають вбудовані генератори пилоподібної напруги, що за-

безпечують різну швидкість наростання при різних рівнях вхідної напруги. Однак при збільшенні швидкості наростання знижується точність вимірювання.

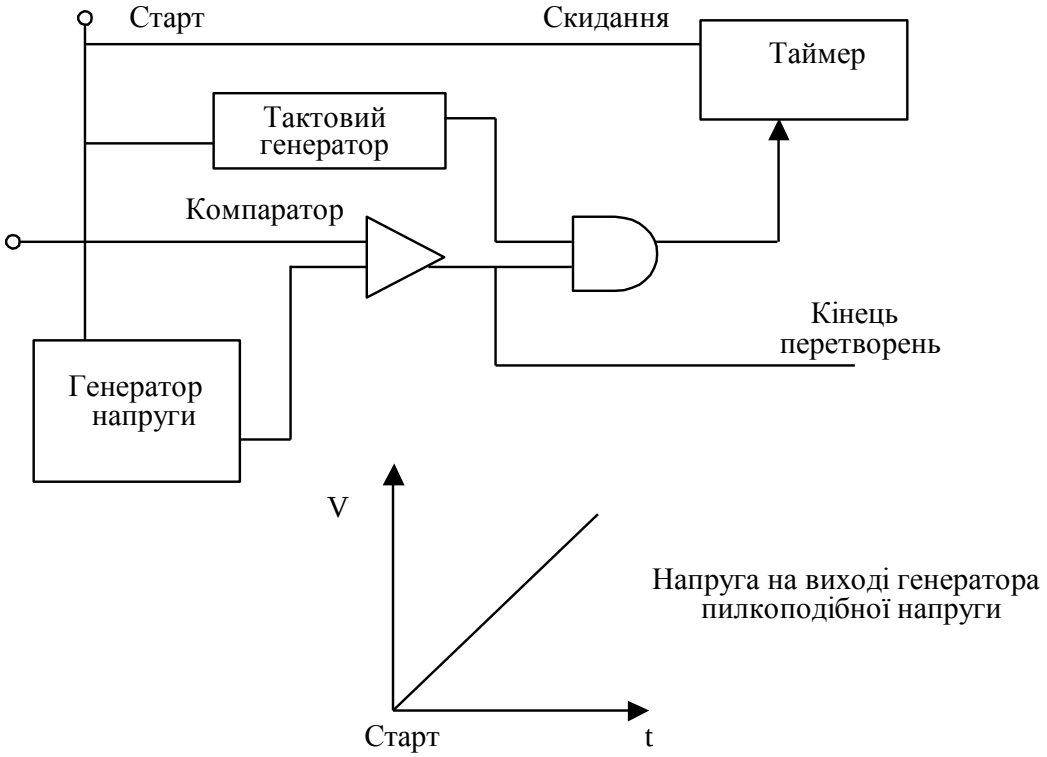


Рисунок 2.50 – Інтегруючий аналого-цифровий перетворювач

Інша проблема – зміна значення вхідного сигналу під час перетворення. Ця проблема вирішується шляхом використання конденсатора, що швидко заряджається до рівня вхідної напруги. Потім напруга на конденсаторі порівнюється з напругою на виході генератора.

Остання проблема, яку залишилося обговорити, це вибір часу перетворення при вимірюванні сигналів, що дуже швидко змінюються. У прикладі, показаному на рис. 2.51, отримана в результаті перетворення форма сигналу істотно відрізняється від реальної. Пояснення як правильно вибрати частоту вибірки сигналу, виходить за рамки даної книги. Інформацію з цього питання можна знайти в книгах, присвячених цифровій обробці сигналів.

Глянувши на специфікації мікроконтролерів, можна знайти, що тільки деякі з них мають аналоговий вихід. Найчастіше мікроконтролери обмежуються формуванням вихідної опорної напруги, одержуваної від резистивного дільника. Причина полягає в тому, що різні аналогові зовнішні пристрої вимагають видачі вихідних сигналів з різними параметрами. Якщо ці параметри не забезпечуються вихідними драйверами мікроконтролера, то відбувається їхнє перевантаження, у результаті чого рівень вихідно-

го сигналу не відповідає необхідному значенню. Крім того, при швидкій зміні напруги у вихідному ланцюзі можуть виникати відображення, що також спотворюють значення вихідного сигналу. Звичайно, для одержання вихідної аналогової напруги використовуються зовнішні цифроаналогові перетворювачі, цифрові потенціометри, включені як подільники напруги чи широтно-модульовані імпульсні сигнали, відфільтровані за допомогою RC-ланцюга. Ці засоби забезпечують дуже високу точність вихідної напруги (у діапазоні декількох мілівольтів).

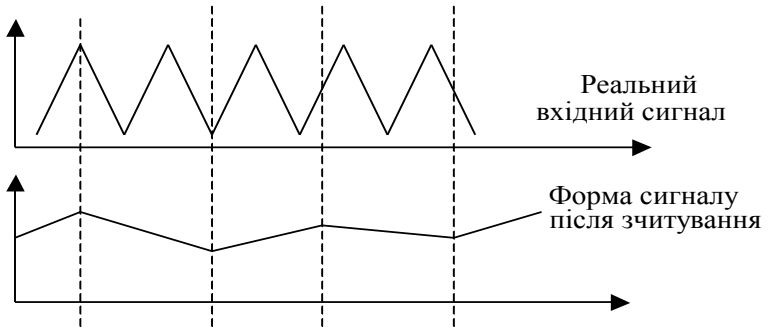


Рисунок 2.51 – Неправильна вибірка сигналу

Існує також одна проста схема, що може бути використана разом з мікроконтролером для реалізації аналогового виходу (рис. 2.52). У цій схемі виводи мікроконтролера використовуються як виходи. Значення напруги, що надходить на вхід операційного підсилювача, залежить від того, на яких виходах встановлений низький рівень сигналу (логічний “0”). У такий спосіб можна одержати різні значення напруги на вході операційного підсилювача. Операційний підсилювач працює в режимі аналогового повторювача, забезпечуючи розв’язку ланцюга навантаження від резистивного подільника. Значення опорів подільника вибираються в залежності від кількості виводів, використовуваних для керування вихідною напругою.

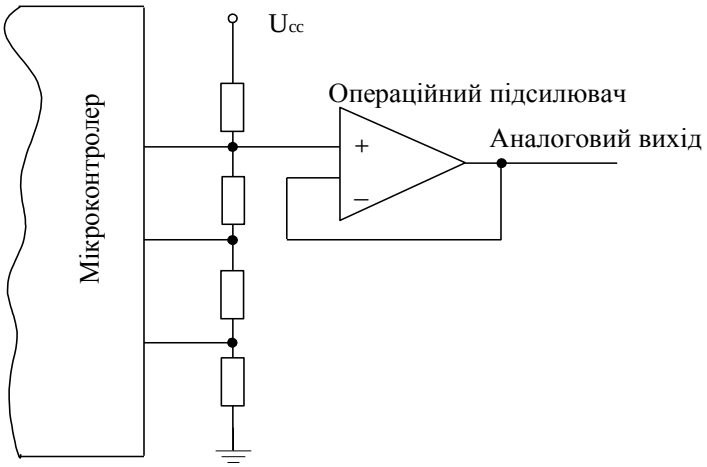


Рисунок 2.52 – Схема реалізації аналогового виходу

Існує багато різних способів аналогового введення-виведення. Більшість мікроконтролерів спроектована таким чином, щоб простими засоба-

ми реалізувати аналогове введення-виведення у випадку повільної зміни аналогової напруги. Наприклад, вибір вхідного сигналу в інтегрувальних АЦП часто здійснюється просто шляхом встановлення біта «Старт АЦП» у регістрі керування, а після виконання перетворення в цьому регістрі встановлюється біт «Кінець АЦП». Для оброблення високочастотних аналогових сигналів варто використовувати зовнішній паралельний АЦП разом з цифровим процесором сигналів (DSP).

2.17 Slave - пристрої

Один з найбільш цікавих і корисних режимів роботи мікроконтролера – функціонування в якості відомого (slave), коли мікроконтролер під'єднується до іншого процесору як периферійний пристрій (рис. 2.53).

У цьому режимі деяка кількість виводів мікроконтролера виділяється для зв'язку із шиною ведучого процесора. При цьому ведений мікроконтролер поводиться як інтелектуальна спеціалізована мікросхема (ASIC), увімкнута в систему, що керується ведучим пристроєм.

Наприклад, персональний комп'ютер, у якому мікроконтролер Intel 8042 слугує як інтерфейс клавіатури, а також виконує деякі інші функції. Це дозволяє процесору просто зчитувати і записувати дані, не реалізуючи протокол послідовного обміну з клавіатурою, що забезпечується мікроконтролером.

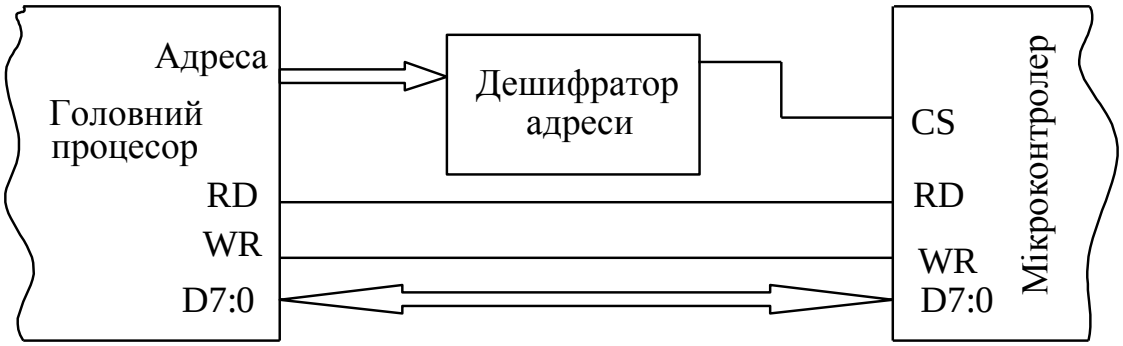


Рисунок 2.53 – Мікроконтролер як ведений пристрій

Мікроконтролер може переривати ведучий пристрій, використовуючи одну зі своїх ліній введення-виведення для подачі запиту переривання. У деяких пристроях відомі мікроконтролери можуть за певних умов працювати як ведучий пристрій, забезпечуючи керування шиною.

3 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ МІКРОКОНТРОЛЕРІВ

Найбільш ефективні варіанти застосування мікроконтролерів реалізуються в тих випадках, коли забезпечується ефективна взаємодія апаратного і програмного забезпечення. Часто проектувальники апаратної части-

ни не залучаються до розробки програмного забезпечення, тому що вважається, що вони не дуже компетентні в області програмування. Але гарне розуміння того як програмується пристрій, не менш важливе ніж розуміння того як працюють апаратні інтерфейси. Сподіваємось, що після знайомства з цим розділом читач зрозуміє, що апаратні і програмні засоби мають однакову важливість для роботи мікроконтролерів, і при розробці цих засобів використовуються схожі інструменти.

3.1 Засоби розробки

Ідея єдності програмного й апаратного забезпечення систем на базі мікроконтролерів є дуже важливою. Об'єднання інструментальних засобів розробки програмного забезпечення з інструментальними засобами розробки апаратного забезпечення може стати важливою перевагою при розробці пристрою. Існують п'ять різних інструментів, що використовуються для розробки програм на базі мікроконтролерів, і об'єднання їхніх функцій може істотно полегшити процес розробки.

У число п'яти основних інструментів розробника входять редактор вихідних текстів, компілятор/асемблер, програмний симулятор, апаратний емулятор і програматор. Хоча не всі з цих інструментів є необхідними, і кожний з них може виконуватися окремо, але їхнє спільне використання спрощує розробку і налагодження програми.

Редактор використовується для створення вихідного коду програми. Існує безліч найрізноманітніших редакторів від простих, котрі просто копіюють код, що вводиться з клавіатури у файл, до спеціалізованих редакторів, реакція яких на натискання визначених клавіш може програмуватися користувачем.

Наприклад, редактор, використовуваний для написання програм мовою С, у дійсності може виконувати програму «REXX» після кожного натискання на клавішу. Ця програма обробляє натискання клавіатури і забезпечує спеціальні види відгуку в різних ситуаціях. Наприклад, після введення оператора «if», редактор видасть наступний відгук, що відтворює формат мови С для цього оператора:

```
if ( )
{ }
else
{ } /* endif */
```

Така реакція редактора рятує від необхідності піклування про правильний синтаксис оператора.

Цей редактор працює з командного рядка DOS, що робить його менш корисним у середовищах із графічним інтерфейсом таких, як Windows, тому що дані не можуть вирізатися та вставлятися стандартним чином. Однак для розробки програми мовою С такий редактор досить комфортний у

використанні. Важливо підкреслити, що редактор, у першу чергу, повинний бути зручний для користувача.

Компілятор/асемблер використовується для перетворення вихідного тексту в асемблерні команди мікроконтролера і потім у формат, що може бути завантажений у пам'ять програм. Далі мова йтиме про асемблер і мови високого рівня, але поки поговоримо про інтерфейс компілятора/ асемблера.

Інтерфейс, про який піде мова, – це інтерфейс між редактором і компілятором/асемблером. Можливість передавати інформацію про помилки компіляції і відображати неправильні рядки на дисплеї може зробити процес розробки більш простим і ефективним. Для описаного вище редактора можна написати REXX-програму, що у процесі компіляції буде зберігати вихідний текст, виконувати компіляцію, знаходити помилкові рядки і виділяти їх у вихідному тексті. Фактично ця REXX-програма робить комплексну обробку вихідного тексту.

Симулятори – це програми, що виконують відкомпільований програмний код в інструментальному комп'ютері системи розробки (host) так, ніби він виконувався в цільовій системі (target). Це дозволяє здійснювати спостереження за програмою і реакцією мікроконтролера на різні події. Симулятор може бути неоціненним інструментом у процесі розробки програмного забезпечення, дозволяючи досліджувати різні ситуації, що важко відтворити на реальній апаратурі.

Щоб імітувати зовнішні умови і ситуації, звичайно використовується спеціальний файл вхідних впливів. Цей файл задає послідовність вхідних сигналів, що надходять на модулювальний пристрій. Звичайно це виглядає в такий спосіб:

Крок	**Bit0**	**Bit1**	**Визначення вхідних бітів**
1	1	1	Початкові умови – всі одиниці
100	0	1	Імітація зрушення даних
101	0	0	
102	0	1	
105	1	1	Перехід до наступного біта
106	1	0	
105	1	1	

І так далі...

Розробка такого файлу вимагає багато часу і великих зусиль. Але для розуміння того, як працюють мікроконтролер і програма у зазначених ситуаціях, використання симулятора і файлу вхідних впливів є найкращим методом. Звичайно, такий файл не може бути використаний для імітації аналогових входів.

У більшості випадків варто використовувати симуляцію перед складанням і увімкненням реальної схеми. Краще спочатку імітувати зазначену ситуацію в пристрої, ніж спалити його під час іспиту. Якщо пристрій не працює очікуваним чином, то варто змінити файл вхідних впливів і спро-

бувати зрозуміти, у чому полягає проблема, використовуючи для цього симулятор, що дозволяє спостерігати за процесом виконання програми на відміну від реальної апаратури, де можна побачити тільки кінцеві результати.

Інтегрована система розробки UMPS використовує графічний симулятор замість зміни у файлі вхідних впливів, щоб імітувати увімкнуте до мікроконтролера устаткування. Використовуючи цю систему, ви можете проаналізувати як буде працювати програма при увімкнутому устаткуванні. Такий симулятор може заощадити багато годин, що витрачаються на створення файлу впливів, і дозволити швидко випробувати визначені комбінації вхідних сигналів, щоб виявити помилку і знайти її причину.

Спеціальна схема, що реалізує інтерфейс із мікроконтролером у реальному масштабі часу, називається «схемний емулятор» (In-Circuit Emulator – ICE). Емулятор використовує мікросхему мікроконтролера, увімкнену не до ПЗП, а до ОЗП програм, що виконує прикладні задачі з реальною швидкістю. Багато емуляторів можуть використовуватися для запису команд, виконуваних процесором у визначений час. Це допомагає зрозуміти, як реагує процесор на дану ситуацію.

Схемний емулятор має два значних недоліки. Перший – ціна, тому що дійсний емулятор коштує тисячу доларів і вище. Другий недолік стосується електричного з'єднання емулятора з системою. Використовувані для цього емулятори мають неміцні виводи, що легко ламаються, особливо якщо багаторазово вставляються і виймаються з рознімів. Але ці недоліки не настільки істотні. На практиці добре спроектований емулятор є могутнім інструментом при розробці програм.

Хоча симулятори не забезпечують виконання програми в режимі реального часу, багато розробників воліють створювати власні файли вхідних впливів і використовувати симулятори для моделювання роботи пристроїв у процесі їхнього проектування. Такий метод дозволяє виявити можливі порушення функціонування пристрою до того, як вони проявляться в реальному виробі. Слід також зазначити, що розробнику істотно легше робити налагодження програмного забезпечення, використовуючи вихідний текст, а не файли, що генеруються компілятором. Цей метод називається «символічним налагодженням». Стежити за виконанням програми за допомогою вихідного тексту набагато простіше, ніж контролювати скомпільований об'єктний код.

Останній інструмент розробника – це програматор пам'яті програм мікроконтролера. Хоча деякі виробники мікроконтролерів воліють випускати їх з однократно програмовною пам'яттю програм, вони, звичайно, випускають також аналогічні версії мікроконтролерів з E(E)PROM пам'яттю для розробки програм. Це значить, що існує можливість безпосереднього програмування мікроконтролера при розробці програми.

Для деяких мікроконтролерів потрібно спеціальний програматор, але найчастіше використовуються можливості внутрішньосистемного програ-

мування ISP. У такому випадку програматор є частиною проектованого пристрою.

Дуже важливо, щоб при програмуванні використовувалися правильні дані. Це особливо необхідно забезпечувати в пристроях із програмовною конфігурацією, коли неправильна установка бітів приведе до того, що пристрій не буде виконувати задані функції.

Деякі програматори реалізують функції схемного емулятора. При цьому встановлений в програматорі мікроконтролер під'єднується до системи і керує її роботою аналогічно тому, як це виконується в емуляторі.

3.2 Мова Асемблер

Перш ніж почати розробку якого-небудь пристрою на базі мікроконтролера, дуже важливо ознайомитися з основами програмування мовою Асемблера. Якщо Ви вивчили мову Асемблер для якого-небудь мікропроцесора чи мікроконтролера й освоїли процес розробки асемблерних програм, то у Вас не виникнуть великі проблеми при написанні програм для іншого процесора. Навіть якщо Ви розробляєте програмне забезпечення мовою високого рівня, то знання Асемблера дозволить визначити, які команди реалізує процесор, щоб зрозуміти, що відбувається в системі при виконанні програми.

Програмування мовою Асемблер, зазвичай, вивчають у школі. Після проходження цього курсу про нього, звичайно, не згадують або згадують з жахом. Однак при розробці програм для мікроконтролерів варто не тільки освоїти цей метод програмування, але і навчитися добре розуміти, як крок за кроком виконується ваша програма, і що при цьому відбувається в пристрої.

Щоб процес вивчення мови, написання і налагодження програм на Асемблері був більш простим і зрозумілим, використовується два прийоми. Перший – візуалізація процедур виконання команд процесором. Другий – використання методів структурного програмування, щоб зробити програми більш простими для читання і розуміння.

Візуалізацію виконання команд найкраще здійснити, використовуючи структурну схему процесора чи мікроконтролера, на якій відзначається проходження даних при виконанні кожної команди. В результаті забезпечується гарне візуальне зображення процесу виконання команд. Для всіх процесорів, представлених у цьому посібнику, наведено такий опис для кожної команди.

Як приклад, розглянемо опис команди *ADC* (Додавання з переносом) для мікроконтролера 68HC05, структура якого показана на рис. 3.1. Відзначимо на цій структурі проходження даних при виконанні команди рис. 3.2.

Зверніть увагу, що на рисунку потоки даних для джерела і приймача розділені. У такий спосіб забезпечується краще уявлення про те, що відбу-

вається в процесорі і якими засобами це забезпечується. Повне розуміння того, що відбувається при виконанні команди дуже важливо, тому що можуть змінюватися біти чи стан вмісту деяких регістрів, на які дана команда впливає неявно. За допомогою цих рисунків простіше запам'ятати, як реалізується команда, тому не виникає необхідність часто заглядати в посібник із програмування.

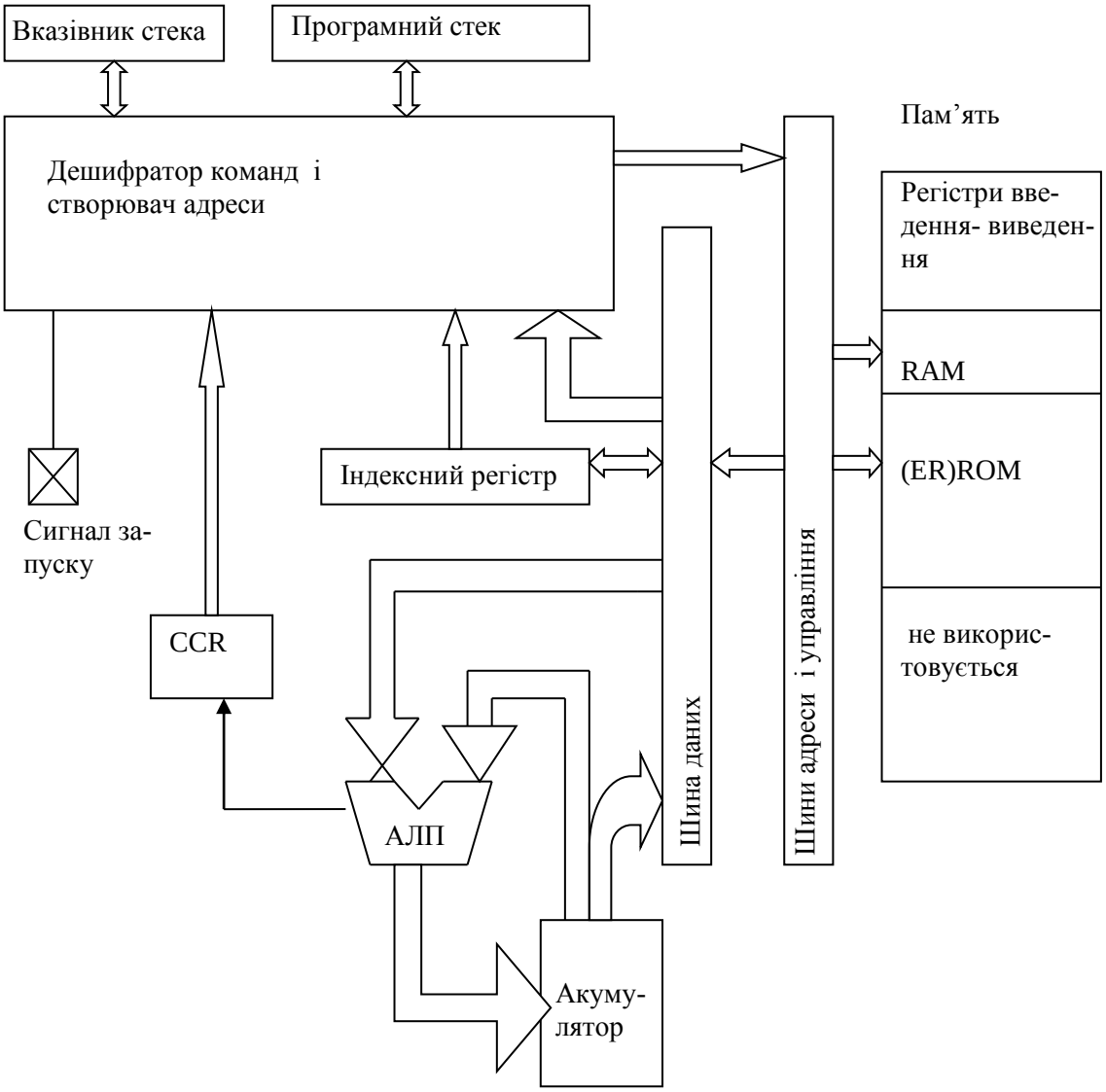


Рисунок 3.1 – Структура мікропроцесора 68HC05

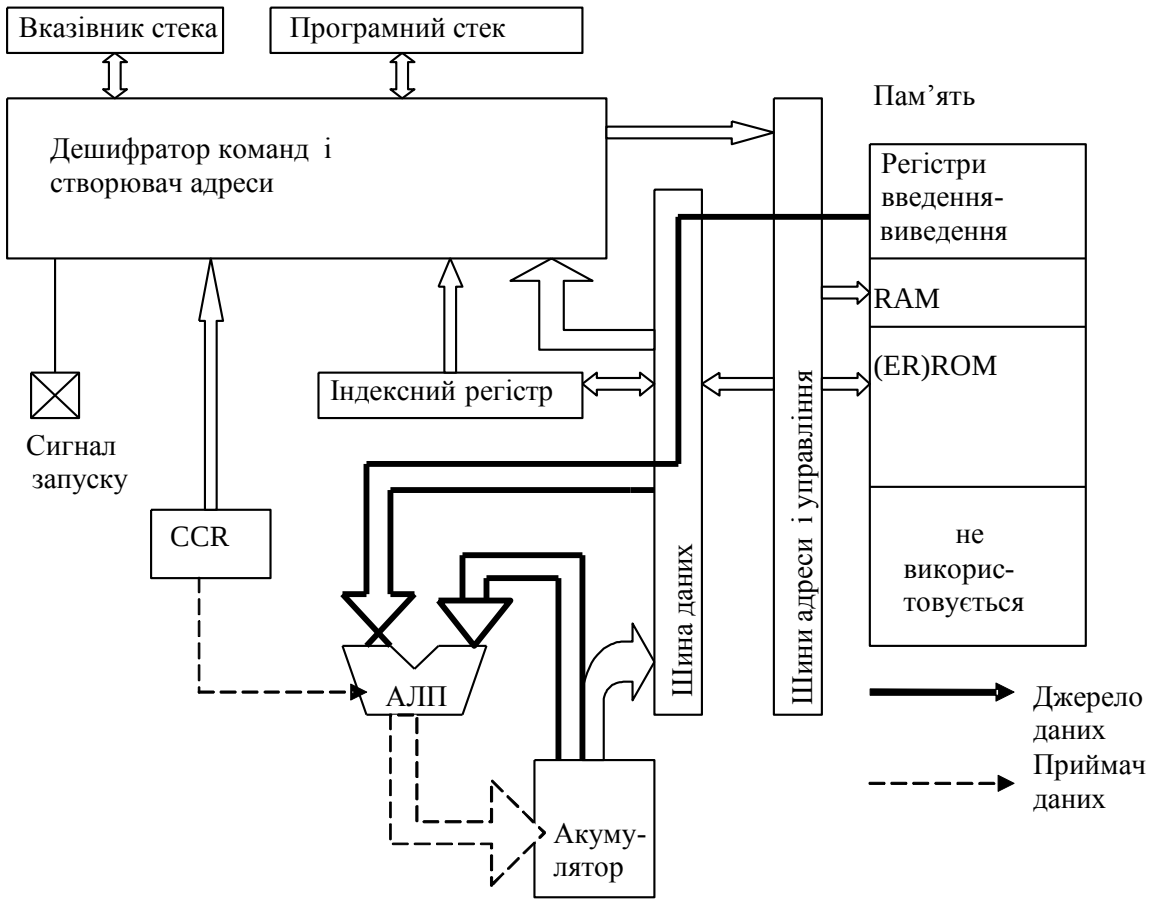


Рисунок 3.2 – Структура мікроконтролера 68HC05 із вказанням шляху проходження даних при виконанні команди ADC

Рекомендація про доцільність використання методів структурного програмування виглядає не зовсім обґрунтованою, однак, застосування деяких з цих методів робить асемблерні програми більш зручними для читання.

Перший з цих методів полягає в тім, щоб розділяти функціональні блоки програмного тексту порожніми рядками. Наприклад, якщо потрібно прочитати 8-бітне число з порту введення-виведення мікроконтролера PIC, а потім видати значення 0x7F в інший порт, якщо введене число дорівнює нулю, і повторити цю процедуру вісім разів, то можна використовувати таку програму:

```
movlw   8               Цикл 8 повторів
movwf   Count
Loop                    Вхід у цикл
  movlw  0              Port == 0?
  iorwf  PORTB, w
  btfss  STATUS, Z
goto    Skip            Ні, пропустити команди
movlw   0x07F           Так, вивести відповідь 0x7F
```

```
movwf     PORTC
Skip
  decfsz  Count        Зменшити Count і якщо != 0
goto      Loop         повернутися до Loop
```

Якщо додати порожні рядки для виділення функціональних блоків, то отримаємо таку програму:

```
movlw     8            Цикл 8 повторів
movwf     Count

Loop                   Вхід у цикл
                       PortB == 0?
  movlw   0
  iorwf   PORTB, w
  btfss   STATUS, Z
goto      Skip         Ні, пропустити команди

movlw     0x07F        Так, вивести відповідь 0x7F
movwf     PORTC

Skip
  decfsz  Count        Зменшити Count і якщо
  goto    Loop         != 0 повернутися до Loop
```

Тепер виділимо умовний фрагмент програми, що також спрощує читання програми:

```
    movlw     8            Цикл 8 повторів
    movwf     Count

    Loop                   Вхід у цикл

      movlw   0            PortB == 0?
      iorwf   PORTB, w
      btfss   STATUS, Z
      goto    Skip         Ні, пропустити команди

      movlw   0x07F        Так, Вивести відповідь 0x7F
      movwf   PORTC

    Skip

      decfsz  Count        Зменшити Count і якщо != 0 по-
      goto    Loop         вернутися до Loop
```

Код стане ще більш зручним для читання, якщо забрати всі непотрібні і зайві коментарі. У результаті отримаємо кінцевий варіант:

```
movlw     8
movwf     Count

Loop                          Опитати Port 8 разів

   Movlw    0                 Якщо Port = 0, то
   iorwf    PORTB, w                Port = 0x7F
   btfss    STATUS, Z
   goto     Skip

   movlw    0x07F
   movwf    PORTC

Skip

   decfsz   Count
   goto     Loop
```

Після розгляду системи команд і прикладів застосування кожного мікроконтролера будуть дані пояснення як виконуються команди, вказівки, на що варто звернути увагу, і рекомендації як зробити програмний код більш ефективним.

3.3 Інтерпретатори

Деякі розробники ще пам’ятають старі часи, коли персональні комп’ютери Apple II чи IBM PC програмувалися за допомогою мови BASIC. При цьому вихідний текст виконувався безпосередньо процесором без проміжної компіляції. Програма, що аналізує кожен рядок програми і потім виконує зазначену в ній команду, називається «інтерпретатор». Ця назва відбиває принцип її роботи, коли кожен рядок спочатку інтерпретується і потім виконується.

Інтерпретація програмного коду звичайно є менш ефективним способом виконання. Перш, ніж виконати командний рядок, потрібно вибрати її з пам’яті, проаналізувати і тільки потім реалізувати задану операцію. Інтерпретувальний код програми завжди буде виконуватися довше, ніж відкомпільований, тому що він має більший обсяг. Багато інтерпретаторів працюють як компілятори – вони спочатку конвертують усі командні рядки в «символи» (tokens), потім ці символи використовуються інтерпретатором для виконання необхідних функцій без необхідності оброблення кожного рядка окремо. Інтерпретатори, що спочатку компілюють програму в символи, працюють набагато швидше, ніж ті, котрі цього не роблять.

Деякі мікроконтролери мають вбудований интерпретатор: наприклад, Intel 8052 містить вбудований інтерпретатор мови BASIC і пам'ять RAM для завантаження вихідного тексту. Ці інтерпретатори не перетворюють текст у символи, тому працюють повільно і потребують великого обсягу пам'яті для збереження вихідних програмних рядків. Цей вихідний програмний код може бути використаний для налагодження прикладної програми безпосередньо всередині мікроконтролера.

3.4 Мови високого рівня

Для програмування мікроконтролерів можна використовувати різні мови високого рівня. Термін «мова високого рівня» слугує для означення мов, використовуваних для написання програм, що легко читаються, і які конвертуються (компілюються) у мову асемблера, а потім перетворюються в об'єктний код (біти і байти) для їхнього виконання мікроконтролером.

Це звичайні мови загального призначення, що конвертовані («портировані») для створення об'єктного коду, виконуваного наявним у мікроконтролері процесором, і забезпечують реалізацію деяких специфічних функцій мікроконтролера. Є безліч компіляторів, розроблених для різних мікроконтролерів, причому ефективність об'єктного коду, що вони генерують, варіюється від дуже поганої до дуже гарної. Кращими є компілятори, що надають користувачу велику частину ресурсів мікроконтролера. Ефективність отриманого об'єктного коду визначається обсягом зайнятої пам'яті програм, необхідним обсягом оперативної пам'яті для збереження даних, і кількістю ресурсів, необхідних для підтримки відкомпільованого коду.

Найбільш популярні мови високого рівня — це C, BASIC і Forth. Дані мови доступні для більшості мікроконтролерів, описаних у посібнику. Існують також деякі спеціалізовані мови для визначених типів мікроконтролерів, що не є універсальними. Мови програмування, використовувані для мікроконтролерів, можуть значно відрізнятися від стандартних варіантів, навіть коли вони мають однакові назви.

Основні характеристики мов високого рівня:

- наявність вбудованих функцій (наприклад, консольне введення-виведення) з бібліотеками, що підключаються;
- різні типи даних (8-, 16-, 32-бітні цілі і з плаваючою точкою);
- виконання арифметичних операцій з використанням стека;
- «локальні» і «глобальні» змінні;
- покажчики і структури даних;
- розподіл пам'яті;
- доступ до апаратних регістрів;
- символічна інформація для симулятора/емулятора.

Усі ці характеристики забезпечуються в стандартному середовищі програмування, що функціонує на персональних комп'ютерах і робочих

станціях. Однак реалізація цих характеристик може бути проблематичною в мікроконтролерах, що вбудовуються, які мають такі особливості:

- обмежений обсяг пам'яті програм ROM;
- обмежений обсяг пам'яті даних RAM;
- відсутність BIOS чи операційної системи;
- перемикальний вхід-вихід (коли порт може використовуватися як цифровий/аналоговий/послідовний вхід-вихід).

Ці особливості можуть обмежити можливість застосування стандартних мов високого рівня й вбудованих бібліотек. Наприклад, функції, вбудовані у бібліотеки, що корисні для деяких програм, у багатьох інших випадках можуть виявитися непотрібними. У мові С стандартні бібліотеки містять функції для консольного введення-виведення, перенесення даних, математичних операцій, перетворень даних і т.д. Хоча всі ці функції є корисними, навряд чи знайдеться додаток, що використовує всі ці функції одночасно. Це означає, що програма, скомпонована з бібліотекою, буде містити функції, що не використовуються. У персональних комп'ютерах, що мають, власне кажучи, необмежений обсяг пам'яті, тому що сегменти прикладних програм можуть завантажуватися і вивантажуватися в пам'ять при необхідності, не виникає проблем із збереженням бібліотек. Однак у вбудованих мікроконтролерах з обмеженим обсягом пам'яті використання бібліотек може стати дуже проблематичним.

Інша проблема зі стандартними бібліотеками полягає в тім, що різні мікроконтролери мають різні особливості. Наприклад, деякі моделі мікроконтролера містять вбудований послідовний порт, тоді як інші його не мають. Часто в таких випадках функція спочатку перевіряє наявність устаткування з визначеними характеристиками, а потім виконує необхідну програму. При цьому погіршується загальна ефективність програми, тому що використовувані функції вимагають більший обсяг пам'яті і більше часу для них на виконання.

Ці проблеми можна вирішити шляхом аналізу використання наявних у бібліотеці функцій і компонування в програмний код тільки тих з них, що необхідні для конкретного застосування і враховують специфіку використовуваного мікроконтролера. Часто в мікроконтролерах компонування різних об'єктних файлів (якими в дійсності є бібліотеки) не застосовується. Замість цього весь код компілюється одночасно як частина однієї програми, а включення в об'єктний код необхідних фрагментів реалізується за допомогою умовної компіляції.

Використання занадто великого числа різних типів даних може викликати проблеми через відсутність вільної пам'яті при використанні 8-розрядних мікроконтролерів. Для оброблення даних з розрядністю більше 8 біт необхідно вводити додаткові команди для виконання заданих операцій. Наприклад, вираз, написаний мовою С :

```
FirstVar = FirstVar + SecondVar;
```

виконується мікропроцесором 8051 за допомогою такої послідовності команд, що виконують оброблення 8-розрядних даних:

```
mov A, FirstVar      Вибрати першу змінну
add A, SecondVar     Скласти з другою змінною
mov FirstVar, A
```

При використанні 16-розрядних змінних програмний код ускладнюється:

```
mov A, FirstVar              Скласти молодші 8 біт
add A, SecondVar
mov FirstVar, A
mov A, FirstVar +1           Скласти старші 8 біт
addc A, SecondVar + 1        Додавання з переносом
mov FirstVar + 1, A
```

Цей програмний код реалізується мікроконтролерами, що виконують команди додавання з переносом (addc). Для мікроконтролерів, які не виконують команди додавання з переносом, програмний код ще більше ускладнюється:

```
mov A, FirstVar          Скласти молодші 8 біт
add A, SecondVar
mov FirstVar, A
jnc   Skip
                         Якщо перенос не встановлений, то пропустити збільшення
inc   FirstVar + 1
                         Збільшити старші 8 біт результату
Skip
mov A, FirstVar + 1
add A, SecondVar + 1
mov FirstVar + 1, A
```

У даному прикладі представлений відносно простий варіант виконання таких операцій. Якщо операція стає більш складною (наприклад, «*FirstVar = SecondVar + (ThirdVar * FourthVar)*»), то ускладнення програмного коду має експонентний характер. При цьому використання даних великої розрядності може привести до надмірного уповільнення чи обчислень, зажадати більшого обсягу пам'яті, ніж є в системі.

Операції над числами, що мають розрядність більше базової (8 біт), часто реалізуються за допомогою бібліотечних функцій, що збільшують час виконання й обсяг необхідної пам'яті. Для багатьох застосувань дуже корисне використання 16-розрядних даних, оброблення яких забезпечується за допомогою невеликого збільшення обсягу програмного коду і пам'яті

даних. У той же час 16-розрядні дані забезпечують досить великий діапазон подання оброблюваних даних. При правильній організації обчислень обробка даних практично для всіх вбудованих додатків, що використовують 8-розрядні мікроконтролери, може бути виконана з використанням 8- і 16-розрядних цілих чисел.

У складних операціях (таких, як «*FirstVar = SecondVar + (ThirdVar * FourthVar)*») програмний код може бути реалізований за допомогою ряду стекових операцій

```
push   SecondVar
push   ThirdVar
push   FourthVar
mul
add
pop    FirstVar
```

Ці стекові операції не є командами процесора, а є послідовністю операцій, що забезпечують одержання необхідного результату. Компілятори часто роблять оптимізацію цієї послідовності операцій, перш ніж перетворити їх у команди процесора. У деяких процесорах такий код може бути дуже ефективним. У залежності від якості компілятора отриманий об'єктний код може бути також ефективний як код, написаний вручну, або в 100 разів (за кількістю необхідних команд і циклів їхнього виконання) менш ефективним.

Для деяких компіляторів подання реалізованої процедури за допомогою декількох командних рядків може бути більш ефективним у порівнянні з їх поданням у вигляді одного рядка. Наприклад, процедура, описана рядком

```
FirstVar = SecondVar + ( ThirdVar * FourthVar );
```

більш ефективно реалізується у вигляді такої послідовності операцій:

```
Temp = ThirdVar * FourthVar;
FirstVar = SecondVar + Temp;
```

Інше важливе розуміння стосується того, як компілятор перетворить числа й обробляє проміжні змінні. Якщо в приведеному вище прикладі змінна «FirstVar» не визначена як порт введення-виведення, то можна подати програмний код у такому вигляді:

```
FirstVar = ThirdVar * FourthVar;
FirstVar = FirstVar + SecondVar;
```

Таке подання є оптимізацією програми, тому що компілятор враховує, що змінна «FirstVar» не використовується при виконанні операції, і її можна використовувати для збереження проміжних результатів. Проблеми можуть виникнути, якщо змінна «FirstVar» слугує для подання вмісту регістрів введення-виведення. У цьому випадку запис проміжного результату може бути сприйнято зовнішнім устаткуванням, як змінна «FirstVar».

У багатьох мовах високого рівня існує два типи змінних. «Глобальні» змінні визначені для використання протягом усієї програми, і їхні значення не можуть перевизначатися. Як глобальні, звичайно, задаються змінні, котрі використовуються в основному тілі програми чи потрібні при виконанні багатьох підпрограм (у цьому випадку передавання значення змінної як параметр підпрограми не ефективна). Глобальні змінні – це єдиний тип змінних, використаних при програмуванні мовою асемблера доти, поки не будуть визначені локальні змінні.

«Локальні» («автоматичні») змінні використовуються при виконанні конкретної підпрограми і створюються при звертанні до неї. Призначення однієї локальної змінної в двох підпрограмах, що не вкладені одна в одну, означає, що компілятор буде використовувати тільки одне значення змінної. Звичайно, локальні змінні завантажуються в стек при виклику підпрограми і витягаються з нього, коли керування повертається програмі, що викликала. Значення локальної змінної губиться після виходу з підпрограми. Параметри, передані підпрограмі, звичайно (але не завжди) є локальними змінними. Це означає, що вони можуть модифікуватися при виконанні підпрограми, але їхні первісні значення, передані підпрограмі, будуть зберігатися.

Покажчики і структури даних є важливими засобами підвищення ефективності програмування при використанні великих обсягів пам’яті. Покажчики звичайно використовуються в системах, де пам’ять є керованим ресурсом. Звичайно, вони не використовуються при програмуванні мікроконтролерів, що мають малий обсяг пам’яті. При розробці прикладних програм для цих мікроконтролерів замість покажчиків використовуються інші методи. Основний метод заміни покажчиків полягає в створенні масиву на початку розділу пам’яті, що передбачається використовувати. Потім для звертання до будь-якого елемента в цій пам’яті використовуються індекси. Організація структур даних дуже корисна при багатьох застосуваннях мікроконтролерів. Структурою даних називається блок пам’яті, що використовується для визначення стандартного записування даних. Наприклад, структуру процесорної команди мовою С можна визначити в такий спосіб:

```
struct instruct  // Формат команди
int address;     // Адреса команди
char instruct;   // Команда
int value;       // 16- бітні дані
```

Звичайно, на структуру посилаються за допомогою покажчика, але ті ж дані можна помістити в масив 16-розрядних слів (цілочислові дані типу ***int*** мовою С) і посилатися на нього за допомогою індексу. Щоб вибрати необхідний елемент у масиві даних, звертаються до значення індексу, що задає адресу початкового елемента масиву, додаючи зсув, що вказує позицію обираного елемента.

Покажемо як це реалізується на практиці. Якщо потрібно вибрати деяку величину з області даних, то при використанні покажчика необхідний програмний код має такий вигляд:

```
struct instruct * Ptr   // Визначення покажчика на структуру
i = Ptr-> value;        //Прочитати «value» з поточного елемента
```

При організації масиву ця процедура виглядає в такий спосіб:

```
struct instruct Array[100]    // Визначення масиву структур
:
i == Array[ Index ].value;
```

Цей код набагато складніший коду з покажчиком, але має одну перевагу: в ньому не використовується покажчик, за яким необхідно стежити, що б він завжди коректно звертався до необхідного елемента структури.

Інша структура даних, яку необхідно розглянути – це таблиця. Таблицю можна подати як одновимірний масив незмінних рядків даних. Звичайно, це інформація про стан системи чи повідомлення для користувача мовою С, і виглядає в такий спосіб:

```
char Greeting [13] = "Hello there I";
```

Компілятор мови повинний розмістити ці таблиці в пам'яті програм (ROM), щоб зберегти вільну оперативну пам'ять RAM, при використанні якої треба було б розміщати в ній елементи масиву і робити їхню ініціалізацію. Інформація в таблиці вибирається як масив, ідентичний масивам, що зчитується з RAM.

У мікроконтролерах програмний код повинний мати доступ до апаратних регістрів. Звичайно, це робиться двома методами. Перший – дозволити вставку асемблерних інструкцій у тіло програми, написаної мовою високого рівня. Другий – дозволити користувачу визначити деякі позиції в адресному просторі даних для звертання до цих регістрів. Обидва методи є відхиленням від стандарту мови, прийнятого для персональних комп'ютерів і робочих станцій. Але ці методи, звичайно, підтримуються мовами високого рівня, використовуваними для програмування мікроконтролерів.

Останнє, що залишилося розглянути – це спосіб передачі символічної інформації до симулятора чи емулятора. Символічна інформація – це посилання на використовувані рядки, мітки, змінні і вказання на їхнє розміщення в пам'яті. Ця інформація використовується симуляторами і емуляторами, щоб відображати виконання вихідного програмного коду. У такий спосіб налагодження ведеться на рівні вихідного тексту, а не на рівні асемблера.

Не слід думати, що всі компілятори видають однакову символічну інформацію (чи видають її взагалі). Може скластися така ситуація, коли не

вдається знайти версію компілятора для обраної мови, що передавала б необхідну символічну інформацію для використовуваного симулятора.

Існує ряд основних особливостей мов високого рівня, які варто враховувати при їхньому виборі для програмування мікроконтролера. Приведені тут коментарі стосуються як проектування програмного забезпечення, так і особливостей функціонування мікроконтролера. Облік цих особливостей допоможе вибрати найбільш ефективний варіант компілятора.

Залишається ще одне питання: навіщо взагалі використовувати мову високого рівня? Існує кілька причин для цього.

Перша – перенос коду. При використанні мов високого рівня існує безліч готових програм і алгоритмів, який можна безпосередньо вставити в розроблювальну прикладну програму. Може виникнути необхідність перенесення програм на інші мікроконтролери даного сімейства чи на інші сімейства мікроконтролерів. При використанні мов високого рівня таке перенесення виконується набагато простіше.

Інша причина, чому варто розробляти додаток мовою високого рівня - відсутність обмежень, властивих програмуванню на асемблері. При цьому користувач одержує різні додаткові можливості, наприклад, автоматично вводити в програму коди складних операцій чи робити ефективний контроль синтаксичних помилок. Необхідно відзначити, що програмна реалізація складних операцій через труднощі й об’ємність програмування мовою асемблер може привести до менш ефективного програмного коду, ніж код, отриманий компілятором з мови високого рівня. Остання і ймовірно найбільш важлива причина використання мов високого рівня – необхідність підтримки програми. Часто потрібно модифікувати чи налагодити написаний код через декілька місяців чи років. При розробці програм мовою високого рівня ця задача значно спрощується.

3.5 Програми, критичні до часу виконання

Хоча мікроконтролери мають багато вбудованих апаратних засобів для виконання точних тимчасових операцій, існують випадки, коли необхідно створити програмний код, що буде робити операції введення-виведення за графіком. Прикладом цього може служити керування асинхронним послідовним обміном без використання апаратних засобів послідовного зв’язку. Це означає, що інтервали часу між посилкою чи опитуванням бітів повинні бути точно витримані.

Перше, що приходить у голову – це використання таймера і переривань, але такий спосіб часто не дає необхідної точності для розроблювальної програми. Розглянемо приклад програми для читання-запису послідовних даних зі швидкістю 9600 бод, що реалізується мікроконтролером, який має тривалість командного циклу 1,0 мкс. При ілюстрації деяких виникаючих проблем припустимо, що всі команди виконуються за один цикл, крім команд переходів, що займають три цикли.

Швидкість 9600 бод відповідає періоду часу 104,167 мкс. Це означає, що часовий інтервал для читання і запису даних повинний займати точно 104 цикли (помилка порядку 0,16%). Нехай для читання даних (прийняття) використовується програмний код:

```
  mov       Count, 8
Loop                          Одержати 8 біт і зберегти в «Char»
  mov       A, Delay          Затримка на 104 циклу
DelayLoop:
  dec       A
  jnz       DelayLoop
  rr        Char              Зрушити "Char" вправо
  mov       A, Port           Останній прочит.біт тепер на 6-й позиції
  and       A, 0x001          Одержати дані з порту А
  jz        Skip              Перевірити біт 0 у порту А
  or        Char, 0x080       Установити старший біт у Char
Skip
  dec       Count             Прочитано 8 біт ?
  jnz       Loop              Якщо «не нуль», то «ні»
```

Тепер треба визначити значення змінної «Delay».

Спочатку обчислимо «власне» час циклу — час, необхідний для організації його запуску. Дивлячись на дану програму можна помітити таку проблему: існує різниця в один командний цикл між двома вітками програми – коли біт встановлений у “1” чи “0”.

Якщо вхідний біт встановлений у “0”, то виконується перехід за командою «jz» (умова правильна), що займає три командних цикли. Якщо цей біт встановлений у “1”, то команди «jz» (умова помилкова, і перехід не відбувається) і «or» виконуються за два командних цикли. Щоб виконання програми завжди забирало однаковий час, необхідно додати дві команди після команди «or Char, 0x080».

Після цього час програмного циклу складе 14 командних циклів. Необхідно додатково виконати 90 циклів для реалізації необхідної затримки. Кожен прохід програми затримки вимагає чотири цикли (один для команди «dec A» і три – для команди «jnz DelayLoop»). У такий спосіб значення «Delay» варто прийняти рівним 23 (при цьому програма затримки буде пройдена 22 рази), а загальний час затримки містить 88 командних циклів. Далі додаємо до цього 13, і усе разом складе 101 командний цикл (помилка 2.98%). Додаючи три команди «or» чи один перехід, одержуємо точно 104 командних циклів.

Щоб визначити, який рівень помилки є прийнятним, необхідне розуміння того, що виконує програма. Для асинхронного інтерфейсу ця помилка збільшується, як мінімум на десять, тому що передаються вісім бітів даних, один старт-біт і один стоп-біт. У такий спосіб помилка в 0.86% при передачі одного біта приводить до помилки 8,6% при передаванні всього

пакета, що має довжину 10 біт. Імовірно, це прийнятно. Помилка в 2.98% на один біт приведе до майже 30% помилки наприкінці передавання байта, що може бути неприйнятно для ряду додатків.

Створення програми для послідовного передавання (записування) даних, багато в чому, аналогічно створенню програмного коду для приймання (читання), але з одним важливим зауваженням. Кожна подія, виведення “1” чи “0”, повинна відбуватися протягом того самого проміжку часу, інакше приймач може пропустити дані. Нижче даний приклад програмного коду для передавання байта зі швидкістю 9600 біт.

```
  mov         Char, 8
Loop                          Вивести 8 біт
  mov         A, Delay        Затримка на 104 циклу
DelayLoop
  dec         A
  jnz         DelayLoop
  mov         A, Char         Вивести 0 чи 1
  and         A, 1            Послати молодші біт першим
  jz          SendZero        Вивести 0
  or          Port, 2         Використовувати біт 1 для виводу
  goto        SentBit
SentZero                      Вивести нуль
  and         PortO,0x0FD
SentBit
  rr          Char            Наступний біт на місце молодшого
  dec         Count           Зробити 8 разів
  jnz         Loop
```

У прикладі читання дані, що надходять, запитувались програмно. При записуванні передані дані видаються в двох різних точках. Значення „1” видається на три цикли раніш, ніж значення „0”. Це можна легко виправити шляхом введення двох команд «jz» перед виконанням команди «or Port, 2». Однак у цьому випадку при посиланні „0” перехід до мітки «SentBit» реалізується на три команди раніш, ніж при посиланні „1”. Можна поставити команду безумовного переходу «goto SentBit» слідом за командою «and Port, 0x0FD». У такому випадку при будь-якому значенні даних вони будуть йти на вихід порту протягом того самого числа циклів після початку процедури «Loop». Після того, як усі можливі вітки програми збалансовані за часом виконання, і виведення даних забезпечене у ті самі моменти часу, можна зробити розрахунок параметра «Delay», як це було виконано в прикладі читання даних.

Цей приклад ймовірно покажеться трохи надуманим і надмірно складним. У дійсності ж він є відносно простим, якщо врахувати як реально виконуються в мікроконтролерах командні цикли й умовні розгалужен-

ня. Деякі процесори мають набагато ширший діапазон необхідних командних циклів, ніж прийнято в розглянутих прикладах.

В області додатків, що вимагають точного обліку часу виконання програм, використання RISC-системи команд і Гарвардської архітектури може дати значні переваги. У цьому випадку команди завантажуються протягом одного циклу (звичайно при виконанні попередньої команди) і виконуються в наступному циклі. Тому різниця в часі виконання команд виявляється істотно менша, і розробка програмного забезпечення для формування необхідних затримок значно спрощується.

3.6 Макроси й умовна компіляція

Застосування процедур макросів і умовної компіляції дозволяє спростити розробку і читання (а виходить, і розуміння) програм. Ці засоби дозволяють писати узагальнений програмний код, що може бути використаний для багатьох специфічних випадків, замість того, щоб створювати спеціальне програмне забезпечення для кожного з цих випадків окремо.

Макрос може розглядатися як функція, що заміщає в програмі її оператор (макровиклик), у той час як підпрограма розміщується поза основним програмним кодом. Наприклад, при програмуванні інтерфейсу, що вимагає подачі тактового сигналу E, можна використовувати код:

```
push  A            Зберегти акумулятор
mov   A, 0x0001    Установити високий рівень сигналу E
or    Port, A
xor   Port, A      Установити знову низький рівень сигналу E
pop   A            Відновити акумулятор
```

Якщо ця процедура використовується багаторазово, то вихідний текст програми стає громіздким і важким для розуміння. Щоб спростити вихідний текст, приведений програмний код можна подати у вигляді макросу:

```
   pulse_E macro       Імпульс по лінії «E»
push       A           Зберегти акумулятор
mov        A,0x000     Установити високий рівень сигналу E
or         1
xor        Port, A     Установити знову низький рівень сигналу E
pop A      Port, A
macroend
```

Тепер щоразу, коли треба сформувати імпульс на лінії E, вищенаведений код буде вставлений у вихідний текст замість макровиклику: pulse_E

Макроси можуть також містити параметри. Якщо в приведеному прикладі потрібно забезпечити видачу імпульсу на довільний вивід порту, то варто виконати необхідні маніпуляції з відповідним бітом у регістрі порту. Для цього макрос перетвориться:

```
pulse macro bit          Імпульс на лінії E
push      A              Зберегти акумулятор
mov       A, 1 « bit     Визначити біт
or        Port, A        Установити біт у 1
xor       Port, A        Скинути біт у ПРО
pop A
macroend
```

Тепер замість макросу, що призначений для роботи з додатками, де лінія E підімкнена до виводу „0” порту, можна використовувати макрос, що формує імпульс на будь-якому заданому виводі порту. Наприклад, макровиклик : pulse 2 вставить у програму макрос, що забезпечить видачу імпульсу на вивід порту. Поряд з макросами в програмі може використовуватися умовно комплексний код, що визначає, чи треба виконувати задану процедуру, у залежності від значення спеціальних параметрів. Ці параметри використовуються тільки під час компіляції і вказують, чи треба включати визначену частину програмного коду у вихідний текст.

Повертаючись до першого прикладу, можна ввести макрозмінну «debug», що дозволить виконувати спеціальні процедури в процесі налагодження. У нашому прикладі формування сигналу E для ЖК-індикатора. Можна реалізувати інверсію сигналу на виводі, що підімкнений до світлодіоду, щоразу, коли подається імпульс E. У такий спосіб буде забезпечена візуальна *індикація* (засвічення світлодіода), коли відбувається записування даних у ЖКІ.

Більшість трансляторів перевіряють виконання умов компіляції перед тим, як компілювати файл. В асемблерах ці умови, звичайно, подані у формі «If (Умова)/else/end» чи у формі «ifdef (Параметр)». Остання форма дозволяє компілювати програмний код за умови, що параметр визначений раніше. У мовах високого рівня, що використовують формат «If(Умова)/else/end» використовуються трохи інші оператори умовної компіляції, наприклад, у мові C використовується оператор «%if».

Тепер перетворимо наш макрос таким чином, щоб інвертувати біт 7 порту A, до виводу якого підімкнутий світлодіод, коли параметр «Debug» визначений. Одержуємо макрос:

```
  pulse macro bit        Імпульс по лінії E
push      A              Зберегти акумулятор
mov       A, 1<< bit     Визначити біт
or        Port, A        Установити біт у 1
```

```
  xor       Port, A       Скинути біт у 0
ifdef       Debug         Якщо "Debug" визначений
  mov       A, 0x080      Інвертувати біт 7
  xor       Port, A
endif       A
  pop
  macroend
```

Умовно комплексний програмний код можна поміщати не тільки в макросах. Вони можуть бути використані й у тілі основної програми. Умовно комплексні коди обробляються в один час з макросами, тому вони, звичайно, розглядаються разом з ними. Навіть якщо Ви ніколи раніше не використовували асемблер, макроси й умовне компілювання, імовірно, Вам знайомі. Що Ви могли не знати – це те що макроси й умовне компілювання широко застосовуються в мовах високого рівня. Твердження «#define» у мові С – фактично макрос, а оператор «#if/#else/ #end» використовується для умовної компіляції.

4 МІКРОКОНТРОЛЕРИ ФІРМИ ATMEL

Отже, тепер, коли ми маємо первинне уявлення про мікроконтролери і про те, що можна зробити з їх допомогою, прийшов час поговорити про створення конструкцій на мікроконтролерах.

Один з варіантів послідовності дій при розробці конструкцій на мікроконтролерах приведений нижче.

1. По-перше, дуже важливо точно визначити технічні вимоги до конструкції, причому робити це бажано письмово – звичайно в процесі записування виявляється багато нюансів, не відразу помітних при обмірковуванні конструкцій.

2. Скласти докладний опис конструкції так званого верхнього рівня – і на цьому етапі ще невідомо ні типу мікроконтролера, ні типу використаних мікросхем і схемних рішень, тому структурна схема являє собою набір прямокутників з надписами найменування вузла, наприклад, АЦП. Складається узагальнена блок-схема, що описує роботу програми. Якщо потрібно – тимчасові діаграми.

3. Визначитися з вибором апаратних вузлів (мікросхем і т.д.) для схеми.

4. Вибрати тип мікроконтролера.

5. Переконатися в тім, що мікроконтролер підходить для реалізації схеми. Варто враховувати швидкодію мікроконтролера, наявність потрібної периферії, число ліній введення-виведення, споживану потужність і інші, істотні для конкретної конструкції параметри. Не є доцільним «бити з гармати по горобцях» – використовувати більш могутній мікроконтролер

для найпростішої задачі, з яким може справитися і більш простий, і більш дешевий. З іншого боку, не слід захоплюватися слабкими мікроконтролерами, ускладнюючи схему, додаючи схему збільшення числа виводів, тому що досить часто (але не завжди) економія, отримана за рахунок застосування більш дешевого мікроконтролера, цілком втрачається через вартість друкованої плати, що збільшилися, (адже її розміри збільшилися), вартості додаткових елементів і т.д.

6. Тепер варто визначитися, які інструменти (програми) будуть використовуватися для розробки програми для мікроконтролера. Це може бути транслятор мови асемблер чи компілятор мови високого рівня, найчастіше С.

7. Після того як стали відомі використовувані вузли мікроконтролера і зовнішні схеми, що під'єднуються до нього, можна розпочинати написання і налагодження програми. Доцільно розділити конструкцію на функціональні вузли і налагоджувати їх у такий спосіб: виготовити частину схеми, що реалізує собою один з вузлів, написати фрагмент програми, що керує цим вузлом, і налагодити його. Після цього аналогічно працювати з наступним вузлом, і так доти, поки всі частини схеми не будуть налагоджені окремо одна від одної. При цьому можна користуватися вже налагодженими вузлами для полегшення перевірки правильності роботи наступних, тільки налагоджених вузлів. Наприклад, для простого калькулятора можна виділити такі вузли: індикатор, клавіатура. Налагоджуючи індикатор, можна написати програму, що виводить на індикатор яке-небудь число. Потім, налагоджуючи клавіатуру, можна використовувати індикатор для виведення, наприклад, номера натиснутої клавіші. І тільки переконавшись, що обидва вузли працюють правильно, варто переходити до реалізації програми, власне, калькулятора.

8. Тепер варто об'єднати всі частини схеми в одне ціле і налагодити їхню роботу спільно. Якщо в процесі об'єднання виявиться, що який-небудь з вузлів реалізований не зовсім вдало, варто повернутися до попереднього пункту.

9. Дуже важливо в процесі складання схеми конструкції і написання програми для неї якомога докладніше документувати всі зміни в схемі чи в програмі. Це дуже важливо не тільки для запису проробленої роботи (а в добре документованій схемі і програмі можна швидко розібратися при необхідності її повторного використання), але і для полегшення подальшого удосконалення чи обслуговування зібраної системи.

10. Завершальний етап відноситься до випадків, коли спроектована конструкція буде виготовлятися, – підготовка креслень принципової електричної схеми, друкованої плати, специфікацій у відповідності зі стандартами, прийнятими на місці, де буде здійснюватися виробництво конструкції.

4.1 Мікроконтролери сімейства AVR

AVR – це нове сімейство 8-розрядних RISC-мікроконтролерів фірми Atmel. Ці мікроконтролери дозволяють вирішувати безліч задач убудованих систем. Вони відрізняються від інших розповсюджених у наш час мікроконтролерів більшою швидкодією, більшою універсальністю. Швидкодія даних мікроконтролерів дозволяє в ряді випадків застосовувати їх у пристроях, для реалізації яких раніше можна було застосовувати тільки 16-розрядні мікроконтролери, що дозволяє відчутно знизити ціну готової системи. Крім того, мікроконтролери AVR дуже легко програмуються – найпростіший програматор можна виготовити самостійно буквально протягом 30 хвилин!

За заявою фірми-виробника мікроконтролерів (www.atmel.com) мікроконтролери сімейства AVR можна перепрограмувати до 1000 разів, причому безпосередньо в зібраній схемі. Все це робить ці мікроконтролери дуже привабливими для створення нових розробок.

Фірма Atmel випускає великий спектр 8-розрядних мікроконтролерів (аналогів 8051) – це сімейство AT89 з вбудованою програмувальною флеш-пам’яттю і AT87 (з пам’яттю типу OTP (одноразовий запис)), а також мікроконтролери власної розробки на базі вдосконаленої RISC- архітектури – AVR-мікроконтролери сімейства AT90 з убудованою флеш-пам’яттю.

В останні роки фірма Atmel освоїла випуск нових мікроконтролерів – AVR ATtiny 11/12/15/22/28 і ATmega 83/161/163/103.

AT89. Сімейство AT89 представлене великою кількістю мікроконтролерів, що відрізняються функціональними можливостями і кількістю інтегрованих на кристалі периферійних пристроїв; максимальною тактовою частотою (від 12 до 33 МГц); кількістю виводів і типом корпусу; діапазоном робочих температур; обсягом вбудованої флеш-пам’яті (від 1 до 32 кбайт) і RAM-пам’яті (від 64 до 512 байт). Деякі з них мають вбудовану пам’ять типу EEPROM (AT89S8252 – 2 кбайта). Усі мікроконтролери сімейства AT89 мають режими роботи зі зниженим енергоспоживанням і можуть працювати при зменшенні тактової частоти аж до 0 Гц. Як і в більшості аналогів 8051, в AT89 передбачені два режими роботи зі зниженим енергоспоживанням: режим idle, у якому здійснюється вимкнення центрального процесорного пристрою (CPU), а струм споживання складає порядку 15 % струму споживання в активному режимі; режим power down, у якому струм споживання знижується до рівня 0.6-15 мкА. Тривалість виконання більшості інструкцій складає $12/f_{такт}$. Більшість мікроконтролерів сімейства AT89 сумісні за розташуванням виводів з аналогічними мікроконтролерами фірми Intel (i80C31, i87C51, i87C54, i80C52, i87C52 та ін.), фірми Philips (PC80C31, PCx80C51, P80C54, P87C54, P80C52, P87C52 та ін.), фірми AMD (87C51, 87C52T2, 8753 та ін.), фірми Matra (80C51, 80C52 та ін.).

Мікроконтролери AT89C1051 (1 кбайт флеш-пам’яті) і AT89C2051

(2 кбайта флеш-пам’яті) випускаються в корпусах із двадцятьма виводами і мають обмежений набір вбудованих пристроїв. В AT89C1051 і AT89C2051 не передбачена можливість реалізації зовнішньої шини програм/даних. Внаслідок обмежених можливостей AT89C1051 і AT89C2051 застосовуються в порівняно простих системах керування/контролю. В одному із самих потужних мікроконтролерів (AT89C55) міститься: флеш-пам’ять (20 кбайт); RAM-пам’ять (256 байт); три таймери/лічильники (чотири є тільки в AT89S8252); UART-контролер; 32 зовнішніх входи/виходи; контролер переривань (вісім джерел). Крім того, є можливість реалізації зовнішньої шини програм/даних. Максимальна тактова частота AT89C55 складає 33 МГц.

AT90. Крім численних аналогів 8051, фірма Atmel випускає велику кількість AVR-мікроконтролерів (сімейство AT90) власної розробки на базі вдосконаленої RISC-архітектури. У AVR-мікроконтролерах реалізовано від 89 до 120 універсальних RISC-подібних інструкцій, що мають фіксовану довжину 16 розрядів. Гнучкі режими адресації, реалізовані в інструкціях, і реєстровий файл обсягом тридцять два 8-розрядних слова (кожен регістр файлу зв’язаний безпосередньо з ALU) забезпечують виконання більшості інструкцій протягом одного такту. Продуктивність AVR-мікроконтролерів складає 1 MIPS/МГц. Фірма Atmel стверджує, що AVR-мікроконтролери мають у десять разів більшу продуктивність, ніж побудовані на базі класичної CISC-архітектури мікроконтролери, що працюють з тією же тактовою частотою. ALU виконує арифметичні і логічні операції з даними, що зберігаються в регістрах файлу. Передбачено можливість одночасної адресації до будь-яких двох регістрів вбудованої RAM-пам’яті даних. Архітектура AVR-мікроконтролерів нагадує гарвардську архітектуру з окремими шинами для вибірки інструкцій і даних. У той час, як виконується поточна інструкція, конвеєр забезпечує вибірку наступної з пам’яті програм. Усі AVR-мікроконтролери сімейства AT90 сумісні між собою на рівні кодів інструкцій. Регістри керування/контролю вбудованими пристроями розташовані в області адрес пам’яті даних. Усі AVR-мікроконтролери мають програмувальні 8- і 16-розрядні таймери/лічильники і сторожовий таймер, що тактується сигналом вбудованого тактового генератора. Схеми фіксації/порівняння, реалізовані в деяких AVR-мікроконтролерах, дають можливість формувати сигнал широтно-імпульсної модуляції. В усіх AVR-мікроконтролерах реалізовано два режими роботи зі зниженим енергоспоживанням: у режимі idle зупиняється робота процесорного ядра, у той час як таймери/лічильники, сторожовий таймер і контролер переривань продовжують роботу; у режимі power down припиняється робота тактового генератора, а, отже, і всіх периферійних пристроїв. Вихід з режиму power down здійснюється за зовнішніми сигналами чи сигналами переривань. Усі AVR-мікроконтролери працюють при напрузі живлення від 2.7 до 6 В й тактовій частоті від 0 до 12 МГц.

У процесорному ядрі деяких AVR-мікроконтролерів реалізований апаратний помножувач. У багатьох AVR-мікроконтролерах реалізований 10-розрядний АЦП (від 6 до 8 каналів).

ATtiny, ATmega. Заслуговують на увагу нові мікроконтролери сімейств ATtiny і ATmega на базі AVR архітектури.

Крім того, що мікроконтролери сімейства ATtiny мають зменшений обсяг вбудованої флеш-пам'яті (від 1 до 2 кбайт), у кожній модифікації випускаються три версії, що відрізняються напругою живлення і тактовою частотою. Наприклад, версії ATtiny12 мають діапазон тактової частоти від 0 до 1 МГц, від 0 до 4 МГц і 0 до 8 МГц відповідно при напрузі живлення від 1.8 до 5.5 В, від 2.7 до 5.5 В і від 4.0 до 5.5 В. Тільки ATtiny22 має RAM-пам'ять даних обсягом 128 байт. У ATtiny15 реалізований 10-розрядний АЦП (4 канали). Деякі модифікації ATtiny мають вбудовану систему перезапуску (brown out detector/reset) при зниженні напруги живлення.

Основна відмінність мікроконтролерів ATmega (ATmega83/1xx) – збільшений обсяг вбудованої пам'яті (ATmega 103 має 128 кбайт флеш-пам'яті і 4 кбайта RAM-пам'яті) і розширений набір периферійних пристроїв. Як і в ATtiny, у модифікаціях ATmega передбачені версії з різною тактовою частотою і напругою живлення. При діапазоні напруги живлення від 2.7 до 3.6 В і від 4.0 до 5.5 В діапазон тактових частот складає відповідно від 0 до 4 МГц і від 0 до 6 МГц.

У результаті цілком справедливо можна сказати, що, вивчивши в достатній мірі мікросхему AT90S2313, читачі легко зможуть використовувати більш потужні мікроконтролери. Мікроконтролер AT90S1200 не підходить для цієї мети через відсутність у нього оперативної пам'яті даних – SRAM, що значно відрізняє його можливості від інших мікроконтролерів сімейства.

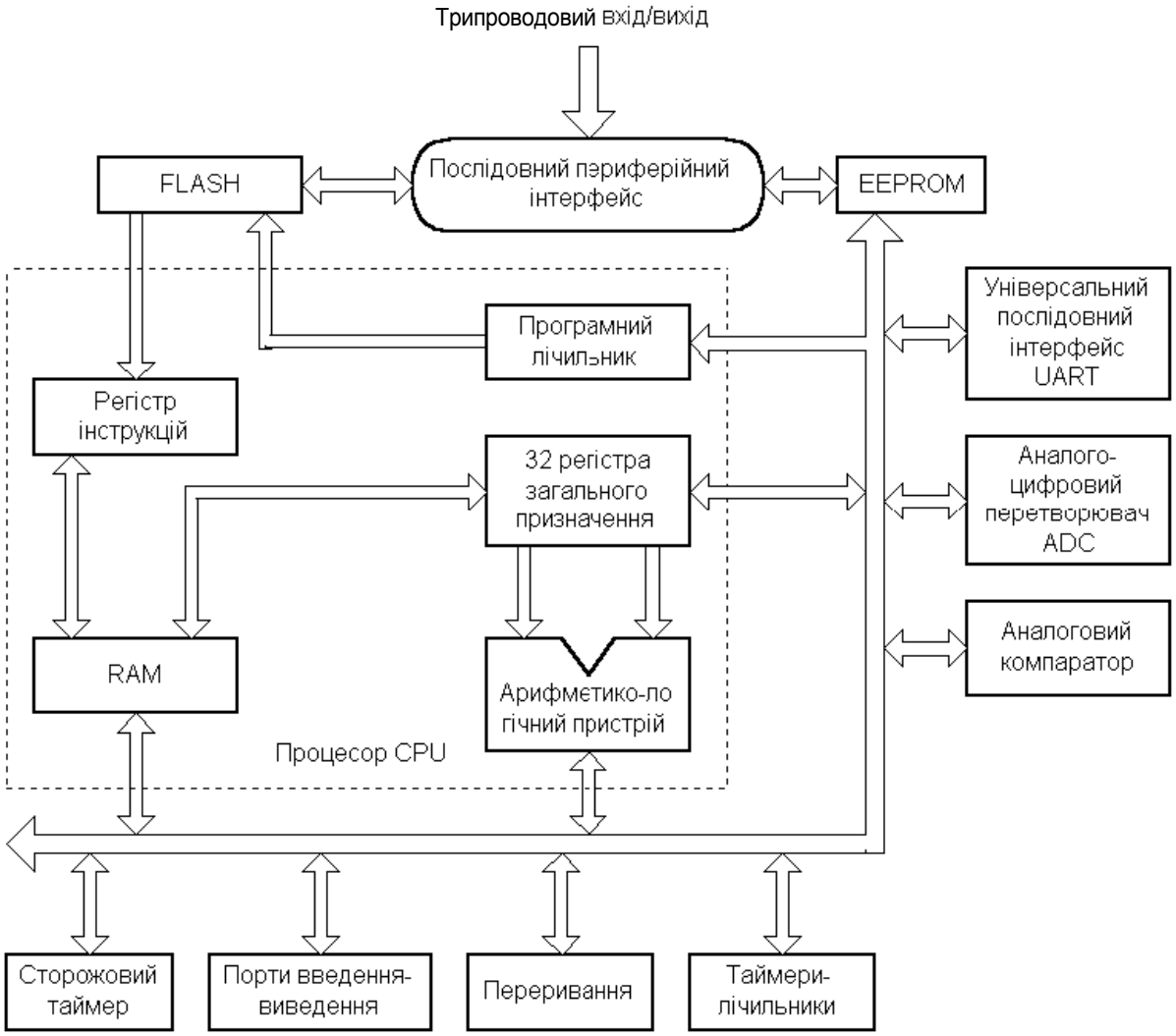


Рисунок 4.1 – Структурна схема AVR-контролерів

Таблиця 4.1 – Характеристики мікроконтролерів сімейства AVR

Тип мікро-контролера	FLASH-пам'ять програм	Пам'ять даних EEPROM	ОЗУ даних SRAM	Кількість команд	Число ліній введення-виведення	Кількість переривань	Кіль-сть зовн. прерив.	SPI-інтерфейс	Послід. інтерфейс	8-розрядний таймер	ШІМ	Сторожовий таймер	Аналоговий компаратор	Число каналів АЦП	Вбудований генератор	Детектор зниж. напруги	Внутрісхемне прогр.	Напруга живлення	Тактова частота	Тип корпусу
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Atiny11L	1	-	-	90	6	4	1	-	-	1	-	Y	Y	-	Y	-	Y	2,7 - 5,5	0 - 2	8-Pin DIP 8-Pin SOIC
Atinyt1	1	-	-	90	6	4	1	-	-	1	-	Y	Y	-	Y	-	Y	4,0 - 5,5	0 - 6	8-Pin DIP 8-Pin SOIC

Продовження таблиці 4.1

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Atiny12L	1	64	-	90	6	5	1	-	-	1	-	Y	Y	-	Y	Y	Y	2,7 - 5,5	0-2	8-Pin DIP 8-Pin SOIC
Atiny12	1	64	-	90	6	5	1	-	-	1	-	Y	Y	-	Y	Y	Y	4,0 - 5,5	0-8	8-Pin DIP 8-Pin SOIC
Atiny15L	1	64	-	90	6	8	1	-	-	2	1	Y	Y	4	Y	Y	Y	2,7 - 5,5	1,6	8-Pin DIP 8-Pin SOIC
Atiny28V	2	-	-	90	20	5	2	-	-	1	-	Y	Y	-	Y	-	Y	1,8 - 5,5	0-1	8-Pin DIP 32-Pin MLF 32-Pin TQFP
Atiny28L	2	-	-	90	20	5	2	-	-	1	-	Y	Y	-	Y	-	Y	2,7 - 5,5	0-4	8-Pin DIP 32-Pin MLF 32-Pin TQFP
AT90S1200	1	64	-	89	15	3	1	-	-	1	-	Y	Y	-	Y	-	Y	2,7 - 6,0	0-12	20-Pin DIP 32-Pin SOIC
AT90S2313	2	128	128	120	15	10	2	-	1	1	1	Y	Y	-	-	-	Y	2,7 - 6,0	0-10	8-Pin DIP 8-Pin SOIC
AT90LS2323	2	128	128	120	3	2	1	-	-	1	-	Y	-	-	-	-	Y	2,7 - 6,0	0-4	8-Pin DIP 8-Pin SOIC
AT90S2323	2	128	128	120	3	2	1	-	-	1	-	Y	-	-	-	-	Y	4,0 - 6,0	0-10	8-Pin DIP 8-Pin SOIC
AT90LS2343	2	128	128	120	4	2	1	-	-	1	-	Y	-	-	Y	-	Y	2,7 - 6,0	0-4	8-Pin DIP 8-Pin SOIC
AT90S2343	2	128	128	120	4	2	1	-	-	1	-	Y	-	-	Y	-	Y	4,0 - 6,0	0-10	8-Pin DIP 8-Pin SOIC
AT90S4433	4	256	128	120	20	14	2	1	1	1	1	Y	Y	6	-	Y	Y	4,0 - 6,0	0-8	28-Pin DIP 32-Pin TQFP
AT90S8515	8	512	512	120	32	11	2	1	1	1	2	Y	Y	-	-	-	Y	4,0 - 6,0	0-8	40-Pin DIP 44-Pin PLCC

Продовження таблиці 4.1

Тип мікро-контролера	FLASH-пам'ять програм	Пам'ять даних EEPROM	ОЗУ даних SRAM	Кількість команд	Число ліній введ.-вивед.	Кількість переривань	SPI-інтерфейс	Послід. інтерфейс UART	Інтерф.TWI, спільн.I2C	8-розрядний таймер	16-розрядний таймер	ШІМ	Сторожовий таймер	Таймер дійсного часу, RTC	Аналоговий компаратор	10-раз.АЦП, число каналів	Вбудований генератор	Детектор зниж. напруги	Внутрісхемне програ.	Напруга живлення	Тактова частота	Тип корпуса
AT90S8535	8	512	512	120	32	15	1	1	-	2	1	3	Y	Y	Y	8	-	-	Y	4,7-6,0	0-8	40-Pin DIP 44-Pin PLCC 44-Pin TQFP
AtmegaBL	8	512	1K	130	23	16	1	1	1	2	1	3	Y	Y	Y	8	Y	Y	Y	2,7-5,5	0-8	28-Pin DIP 32-Pin MLF 32-Pin TQFP
AtmegaB	8	512	1K	130	23	16	1	1	1	2	1	3	Y	Y	Y	8	Y	Y	Y	4,0-5,5	0-16	28-Pin DIP 32-Pin MLF 32-Pin TQFP
Atmega161	16	512	1K	130	35	20	1	2	-	2	1	4	Y	Y	Y	-	Y	Y	Y	4,0-5,5	0-4	40-Pin DIP 44-Pin TQFP
Atmega169L	16	512	1K	130	32	17	1	1	1	2	1	3	Y	Y	Y	8	Y	-	Y	2,7-5,5	0-4	40-Pin DIP 44-Pin TQFP
Atmega163	16	512	1K	130	32	17	1	1	1	2	1	3	Y	Y	Y	8	Y	-	Y	4,0-5,5	0-8	40-Pin DIP 44-Pin TQFP
Atmega16	16	512	1K	130	32	17	1	1	1	2	1	3	Y	Y	Y	8	Y	-	Y	4,0-5,5	0-16	40-Pin DIP 44-Pin TQFP
Atmega323L	32	1K	2K	130	32	19	1	1	1	2	1	4	Y	Y	Y	8	Y	-	Y	2,7-5,5	0-4	40-Pin DIP 44-Pin TQFP
Atmega323	32	1K	2K	130	32	19	1	1	1	2	1	4	Y	Y	Y	8	Y	-	Y	4,0-5,5	0-8	40-Pin DIP 44-Pin TQFP
Atmega103L	128	4K	4K	121	48	16	1	1	-	2	1	4	Y	Y	Y	8	-	-	Y	2,7-3,6	0-4	44-Pin TQFP
Atmega128	128	4K	4K	133	48	27	1	2	1	2	2	6 + 2	Y	Y	Y	8	Y	Y	Y	4,0-5,5	0-16	44-Pin TQFP

AT90S2313 – сучасний 8-бітовий КМОН-мікроконтролер. AT90S2313 має продуктивність близько 1 MIPS на мегагерц за рахунок того, що майже всі команди він виконує за один період тактового генератора.

Мікроконтролери сімейства AVR побудовані на основі розширеної RISC-архітектури, що поєднує розширений набір команд і 32 регістра загального призначення. Усі 32 регістра безпосередньо увімкнені до арифметико-логічного пристрою (АЛУ), що дає доступ до будь-яких двох регістрів протягом одного машинного циклу. Подібна архітектура забезпечує майже десятикратний виграш у продуктивності в порівнянні з традиційними мікроконтролерами, наприклад, серії 8051.

Мікроконтролер AT90S2313 має такі характеристики: 2 Кбайт флеш-пам'яті, що завантажується; 128 байт EEPROM; 15 ліній введення-виведення загального призначення; 32 робітників регістра; два таймери/лічильника, один 8-розрядний, інший 16-розрядний; зовнішнього і внутрішнього переривання; убудований послідовний порт; програмувальний сторожовий таймер з убудованим генератором; послідовний порт SPI для завантаження програм; два режими низького енергоспоживання, які вибираються програмно.

Флеш-пам'ять на кристалі може бути перепрограмована безпосередньо в системі через послідовний інтерфейс SPI.

4.2 Опис виводів

U_{cc} — вивід джерела живлення.

GND — загальний провід («земля»).

PORT B (PB7...PB0) — порт B є 8-бітовим двонаправленим рівнобіжним портом введення-виведення з вбудованими обмежувальними резисторами. У виводів порту передбачені внутрішні резистори, (їх можна вмикати чи вимикати для кожного біта окремо). Виводи PB0 і PB1 також є додатним (AIN0) і від'ємним (AIN1) входами вбудованого аналогового компаратора. Вихідні буфери порту B можуть поглинати струм до 20 мА і безпосередньо керувати світлодіодними індикаторами. Це означає, що мікроконтролер здатен керувати навантаженням до 20 мА при стані логічний "0" на виході порту. Таким чином, для керування світлодіодом його варто приєднати одним виводом до виводу порту мікроконтролера, а іншим – до напруги живлення $+U_{cc}$. Відповідно світитися світлодіод (а виходить, і споживати струм) буде при значенні "0" на відповідній лінії порту. Якщо виводи PB0...PB7 використовуються як входи і ззовні встановлюються в низький стан, вони є джерелами струму, якщо увімкнуті внутрішні обмежуючі резистори. Крім того, порт B обслуговує деякі спеціальні функції, що будуть описані нижче.

PORT D (PD6...PD0) – порт D є 7-бітовим двонапавленим рівнобіжним портом введення-виведення з вбудованими обмежувальними резисторами. Вихідні буфери порту D також можуть поглинати струм до 20 мА.

Якщо входи, встановлені в низький стан, виводи порту D є джерелами струму, якщо задіяні обмежуючі резистори. Крім того, порт D обслуговує деякі спеціальні функції, що будуть описані нижче.

RESET – вхід скидання. Утримання на вході низького рівня протягом двох машинних циклів (якщо працює тактовий генератор), перезапускає мікроконтролер.

XTAL1 – вхід підсилювача генератора, і вхід зовнішнього тактового сигналу.

XTAL2 – вихід підсилювача генератора.

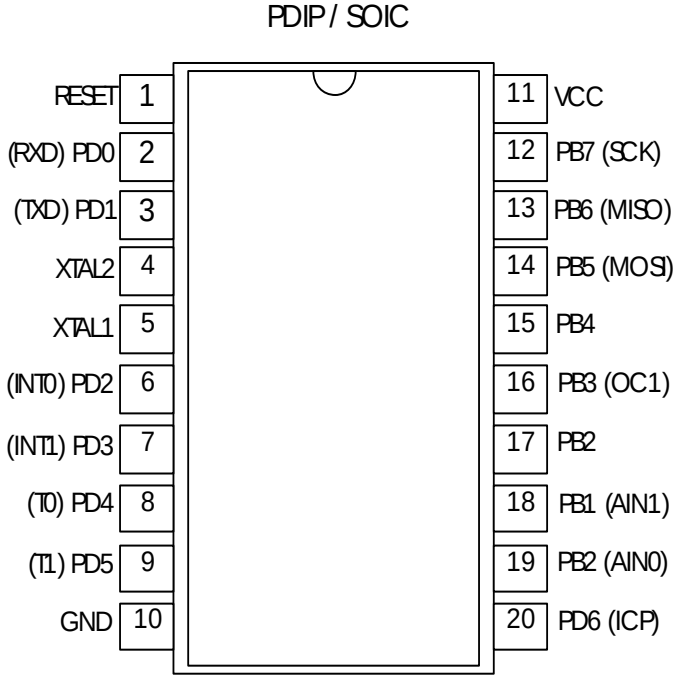


Рисунок 4.1 – Виводи мікроконтролера AT90S2313

Кварцовий генератор

Виходи XTAL1 і XTAL2 є входом і виходом підсилювача, на якому можна зібрати генератор тактових імпульсів. Можна використовувати як кварцові, так і керамічні резонатори. Якщо потрібно використовувати зовнішній тактовий сигнал, він подається на вивід XTAL1, а вивід XTAL2 при цьому залишається непідімкненим.

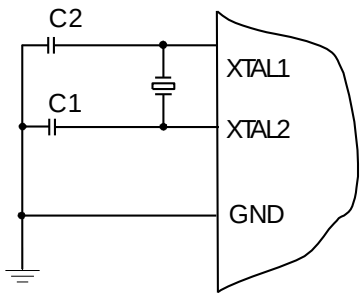


Рисунок 4.2 – Підімкнення кварцового резонатора до мікроконтролера

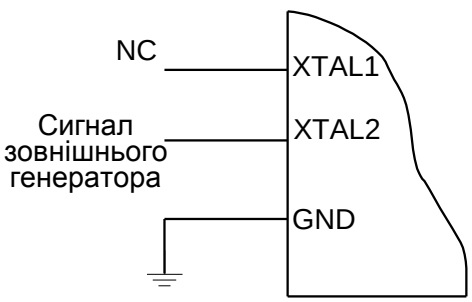


Рисунок 4.3 – Підімкнення зовнішнього джерела тактових імпульсів

4.3 Огляд архітектури AT90S2313

Файл регістрів загального призначення

Реєстровий файл мікроконтролера містить 32 8-розрядних регістра загального призначення, доступ до яких здійснюється за один машинний цикл. Завдяки цьому мікроконтролер може виконати більшість команд за один цикл тактової частоти.

АЛП підтримує арифметичні і логічні операції з регістрами, з константами і регістрами.

Крім реєстрових операцій, для роботи з реєстровим файлом можуть використовуватися доступні режими адресації, тому що реєстровий файл займає адреси $00-$1F в області даних, звертатися до них можна як до комірок пам'яті.

Простір введення-виведення складається з 64 адрес для периферійних функцій процесора таких, як керуючі регістри, таймери/лічильники й ін. Доступ до простору введення-виведення може здійснюватися безпосередньо як до комірок пам'яті, який розташований після реєстрового файлу ($20—$5F).

Більшість команд, що використовують регістри, можуть використовувати будь-які регістри загального призначення. Виняток складають п'ять команд, що оперують з константами: SBCI, SUBI, CPI, ANDI, ORI і команда LDI, що завантажує регістр константою. Ці команди працюють тільки з другою половиною реєстрового файлу – R16...R31.

Кожному регістру привласнена адреса в просторі даних, вони відображаються на перші 32 осередки ОЗП. Хоча реєстровий файл фізично розміщений поза ОЗП, подібна організація пам'яті дає гнучкий доступ до регістрів.

Шість з 32 регістрів – R26...R31 - можна використовувати як три 16-розрядних адресних покажчики в адресному просторі даних. Один із трьох адресних покажчиків (регістр Z) можна використовувати для адресації таблиць у пам'яті програм. Ці регістри позначаються як X, Y, Z і визначені в такий спосіб:

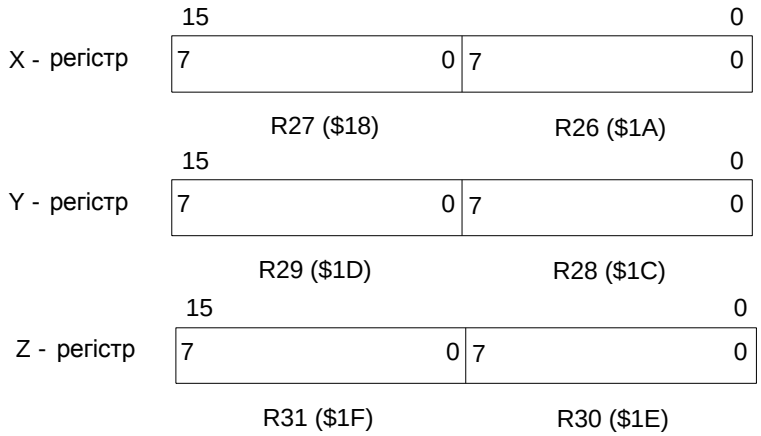


Рисунок 4.4 – Регістри X, Y, Z

При різних режимах адресації ці регістри можуть використовуватися

як фіксована адреса, для адресації з автоінкрементом чи з автодекрементом. При розробці мікроконтролерів сімейства AVR була використана Гарвардська архітектура. На рисунку 4.5 зображена структура пам'яті мікроконтролерів AVR.

Доступ до пам'яті програм здійснюється в такий спосіб: під час виконання однієї команди наступна команда вибирається з пам'яті програм. Це дає можливість виконувати по одній команді за кожен машинний цикл.

За допомогою команд відносних переходів і виклику підпрограм здійснюється доступ до всього адресного простору. Велика частина команд мікроконтролерів AVR має розмір 16-розрядів – одне слово. Кожна адреса в пам'яті програм містить одну 16- чи 32-розрядну команду.

При обробленні переривань і викликах підпрограм адреса повернення запам'ятовується в стеку. Стек розміщується в оперативній пам'яті даних загального призначення (SRAM), його розмір обмежений тільки розміром доступної пам'яті SRAM і її використанням у програмі. Усі програми користувача повинні ініціалізувати покажчик стека (SP) відразу після запуску мікроконтролера, до того як викликаються підпрограми і дозволяються переривання. Виняток складають мікроконтролери, що не мають оперативної пам'яті даних (SRAM), наприклад, AT90S1200. У цих мікроконтролерах реалізований апаратний стек глибиною 3. Це обов'язково варто враховувати при написанні для них програм.

Весь простір пам'яті AVR є лінійним і безупинним. Модуль переривань має власний керуючий регістр у просторі введення-виведеня, і прапорець глобального дозволу переривань у регістрі стану. Кожному перериванню призначений свій вектор у початковій області пам'яті програм. Різні переривання мають пріоритет відповідно до розташування їхніх векторів. За молодшими адресами розташовані вектори з великим пріоритетом.

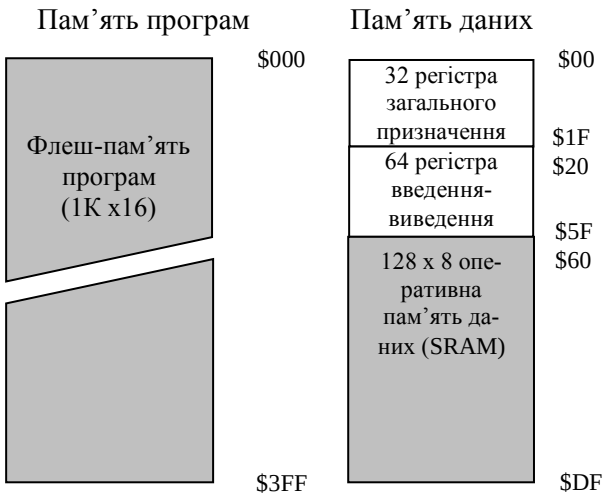


Рисунок 4.5 – Структура пам'яті мікроконтролерів AVR

Режими адресації

Пряма реєстрова адресація з одним регістром Rd

Дані, над якими здійснюється операція (чи використовуються при виконанні операції), знаходяться в регістрі d (Rd).

Пряма реєстрова адресація з двома регістрами – Rd і Rr

Дані, над якими здійснюється операція, знаходяться в регістрах r (Rr) і d (Rd). Результат операції зберігається в регістрі d (Rd).

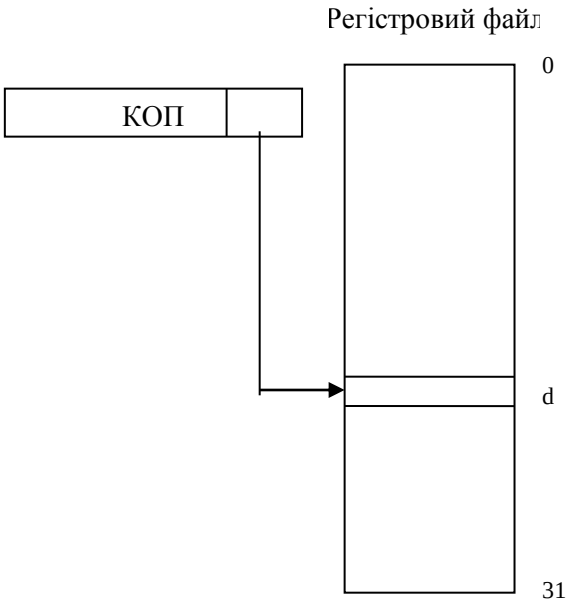


Рисунок 4.6 – Пряма регістрова адресація з одним регістром

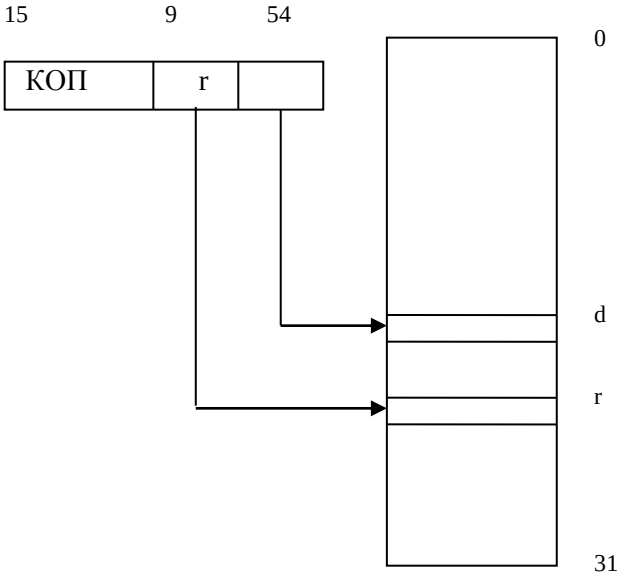


Рисунок 4.7 – Пряма регістрова адресація з двома регістрами

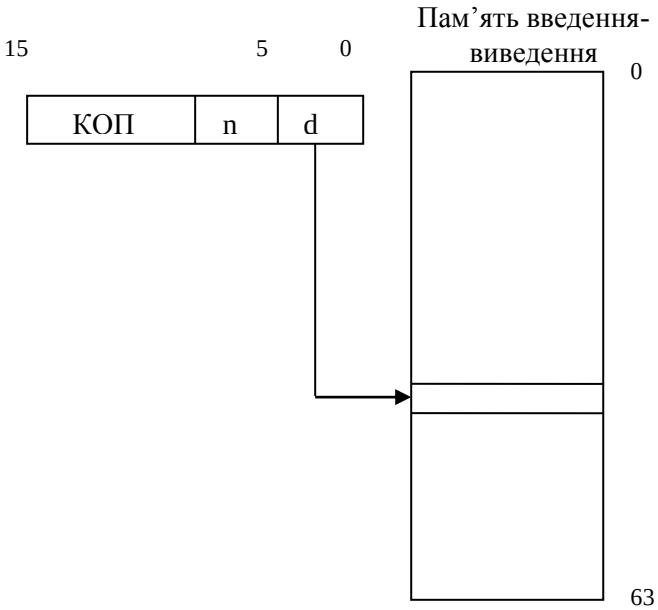


Рисунок 4.8 – Пряма адресація до області введення-виведення

n – адреса регістра, який використовується в операції, знаходиться безпосередньо в коді команди, у бітах 0...5.

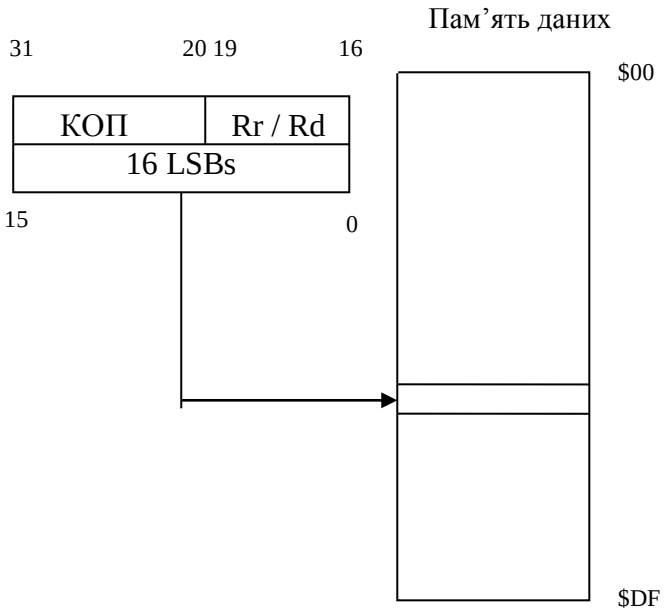


Рисунок 4.9 – Пряма адресація до пам’яті даних

16-розрядна адреса комірки пам’яті даних знаходиться в коді команди, що складається з двох слів. Rr/Rd визначає регістр, використовуваний при роботі з пам’яттю даних (тобто регістр, куди записуються результати операції або звідки вони беруться для виконання операції).

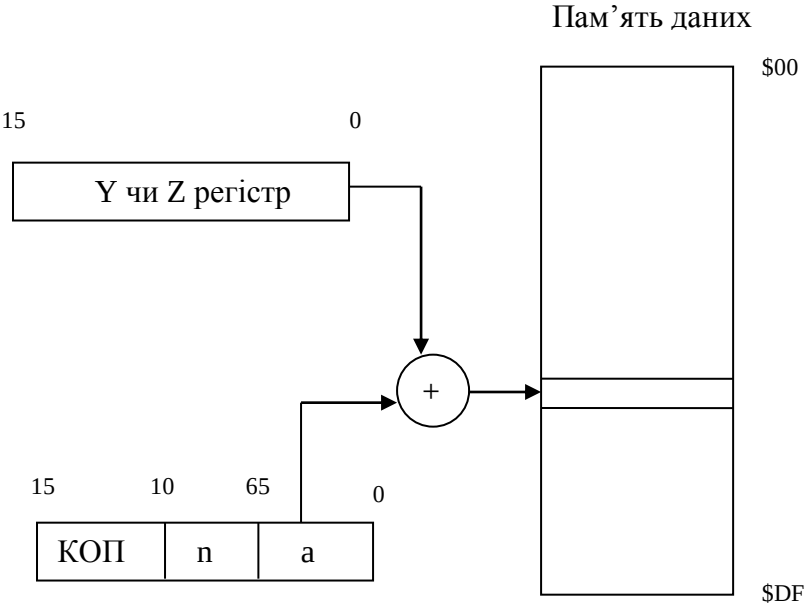


Рисунок 4.10 – Непряма адресація до пам’яті даних зі зсувом

Адреса операнда визначається як сума вмісту Z чи Y регістра і біт 0...5 коду команди.

Пам’ять даних

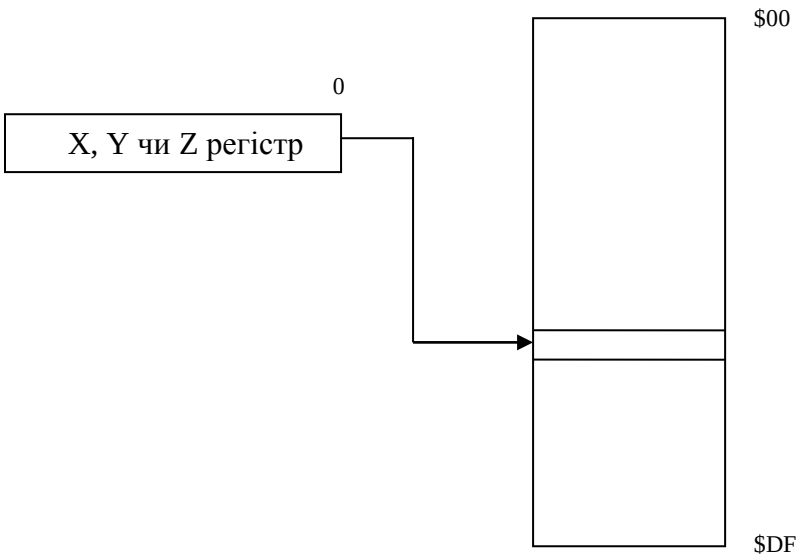


Рисунок 4.11 – Непряма адресація до пам’яті даних

Адреса операнда знаходиться в X-, Y- чи Z- регістрі.

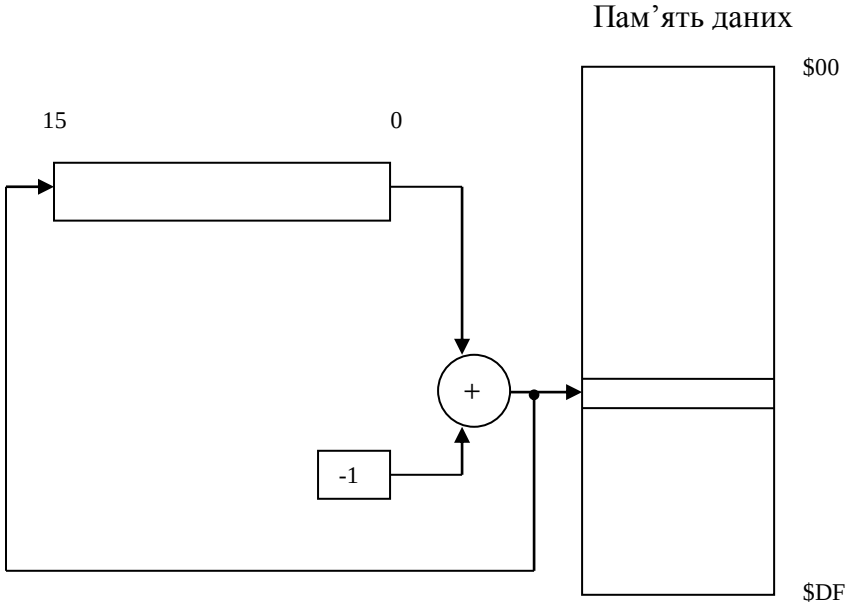


Рисунок 4.12 – Непряма адресація до пам’яті даних з попереднім декрементом

Адреса операнда знаходиться в X-, Y- чи Z-регістрі. Однак перед виконанням операції відповідний індексний регістр X-, Y-чи Z зменшується на одиницю.

Пам’ять даних

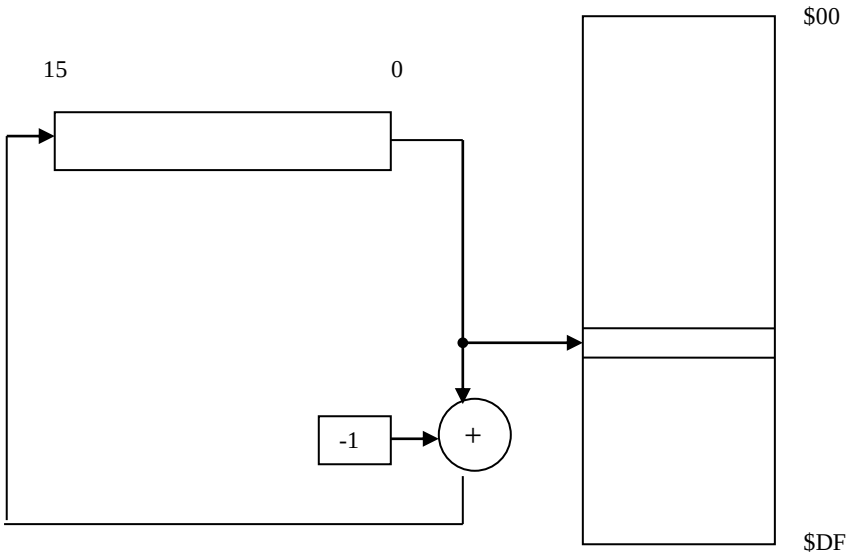


Рисунок 4.13 – Непряма адресація до пам’яті даних з постінкрементом

Адреса операнда знаходиться в X-, Y- чи Z-регістрі. Після виконання операції відповідний індексний регістр X-, Y- чи Z збільшується на одиницю.

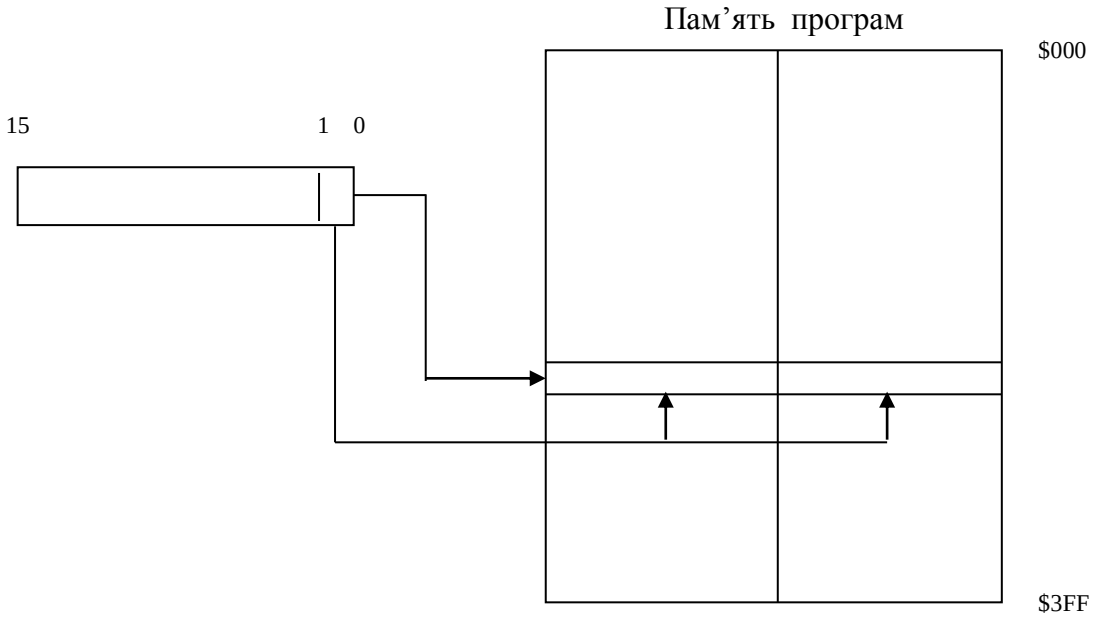


Рисунок 4.14 – Адресація до констант у пам’яті програм

Адреса константи – байта міститься в регістрі Z. 15 старших бітів визначають адресу слова, а молодший (0) біт – молодший чи старший байт константи в пам’яті програм. Якщо в молодшому біті "0" – обраний молодший байт, "1" – старший байт.

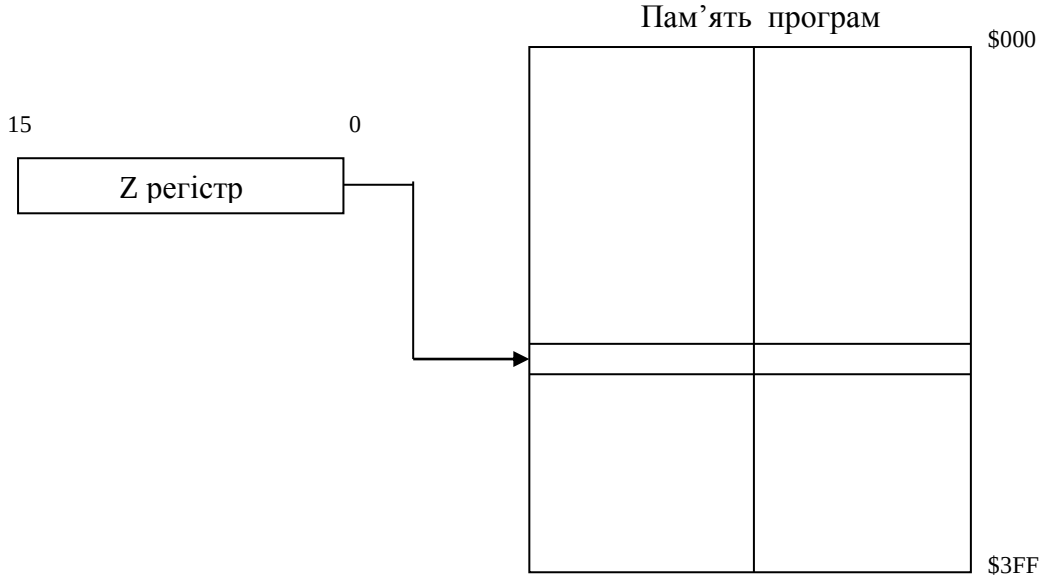


Рисунок 4.15 – Непряма адресація пам’яті програм

Після операцій IJMP або ICALL виконання програми продовжується з адреси, записаного в Z-регістрі (тобто в PC лічильник команд мікроконтролера записується вміст двох регістрів).

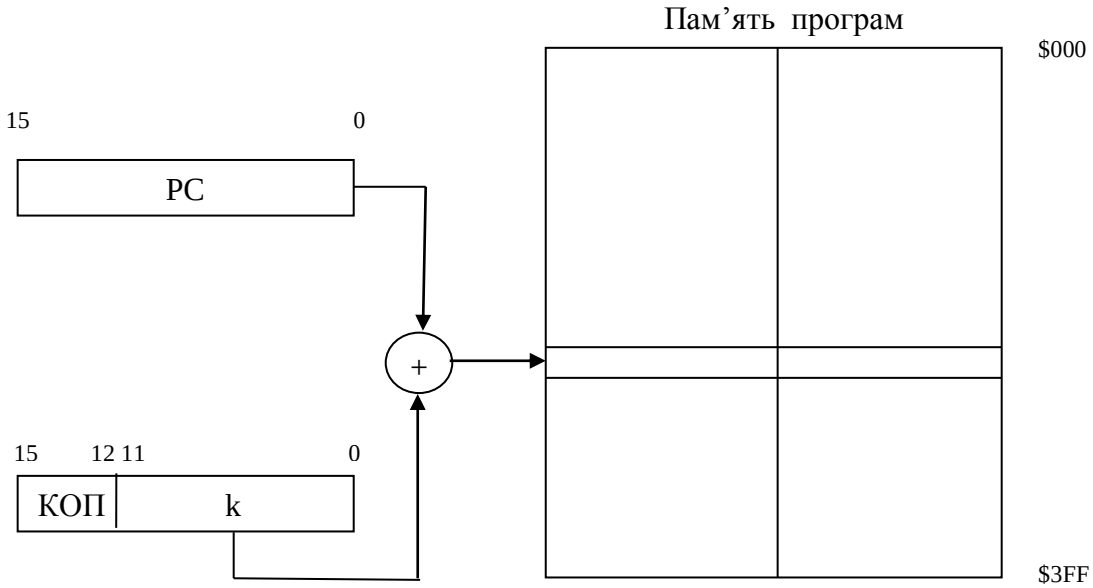


Рисунок 4.16 – Відносна адресація пам’яті програм

Після операцій RJMP або RCALL виконання програми продовжується з адреси PC+k+1. Відносна адреса k може складати від -2048 до 2047.

Арифметико-логічний пристрій

До арифметико-логічного пристрою (АЛП) мікроконтролера безпосередньо увімкнено до 32 регістрів загального призначення. За один машинний цикл АЛП робить операції між регістрами реєстрового файлу. АЛП може виконувати арифметичні, логічні і бітові операції.

Пам’ять програм

AT90S2313 містить 2 Кбайт флеш-пам’яті для збереження програм. Флеш-пам’ять організована як 1Кбайт×16. Програмний лічильник має ширину 10 біт і дозволяє адресувати 1024 слів пам’яті програм.

Способи занесення інформації (тобто програм) у флеш - пам’ять мікроконтролера будуть розглянуті далі.

EEPROM пам’ять даних

AT90S2313 містить 128 байт енергонезалежної пам’яті, що стирається електрично. EEPROM організована як окрема область даних, кожен байт якої може бути прочитаний і при необхідності перезаписаний. EEPROM витримує не менш 100000 циклів записування/стирання. До цієї пам’яті може звертатися програма, зчитуючи чи записуючи які-небудь дані. Крім того, дані в цю пам’ять можна занести за допомогою спеціального пристрою – програматора, на етапі виготовлення і програмування конструкції. Її зручно використовувати для збереження яких-небудь констант.

Оперативна пам’ять даних

На рис. 4.17 показана організація пам’яті даних у AT90S2313. 224 комірки пам’яті містять у собі реєстровий файл, пам’ять введення-виведення й оперативну пам’ять даних.

Перші 96 адрес використовуються для реєстрового файлу і пам’яті введення-виведення, 128 – для ОЗП даних.

При звертанні до пам’яті використовуються п’ять різних режимів адресації: прямої, безпосередній зі зсувом, безпосередній, безпосередній з попереднім декрементом і безпосередній з постінкрементом. Регістри R26...R31 реєстрового файлу використовуються як покажчики для безпосередньої адресації.

Пряма адресація має доступ до всієї пам’яті даних. Безпосередня адресація зі зсувом використовується для доступу до 63 адрес, базова адреса яких задається вмістом регістрів Y чи Z.

Для безпосередньої адресації з інкрементом і декрементом адреси використовуються адресні регістри X, Y і Z.

За допомогою кожного з цих режимів можна здійснювати доступ до 32 регістрів загального призначення, 64 регістрів введення-виведення, 128 адрес ОЗП.

Час виконання команд

ЦПУ процесора AVR керується тактовою частотою, яка генерується зовнішнім резонатором. Внутрішній розподіл частоти генератора в мікроконтролерах сімейства AVR не використовується.

У мікроконтролері процес виконання команд організований так, що при виборі команди з пам’яті програм відбувається виконання попередньої команди. Це дозволяє досягти швидкодії 1 MIPS на МГц.

Рисунок 4.17 – Організація пам’яті даних у мікроконтролері AT90S2313

Таблиця 4.2 – Простір введення-виведення AT90S2313

Адреса	**Назва**	**Призначення**
1	2	3
$3F($5F)	SREG	Регістр стану
$3D($5D)	SPL	Молодший байт покажчика стека
$3B($5B)	GIMSK	Загальний регістр маски переривань
$3A($5A)	GIFR	Загальний регістр прапорців переривань
$39($59)	TIMSK	Регістр маски переривань від таймерів/лічильників
$38($58)	TIFR	Регістр прапорців переривань від таймерів/лічильників
$35($55)	MCURC	Загальний регістр керування мікроконтролера
$33($53)	TCCR0	Регістр керування таймером/лічильником 0
$32($52)	TCNT0	Таймер/лічильник 0 (8 біт)
$2F($4F)	TCCR1A	Регістр А керування таймером/лічильником 1
$2E($4E)	TCCR1B	Регістр У керування таймером/лічильником 1
$2D($4D)	TCNT1H	Старший байт таймера/лічильника 1
$2C($4C)	TCNT1L	Молодший байт таймера/лічильника 1
$2B($4B)	OCR1AH	Вихід регістра збігу, старший байт
$2A($4A)	OCR1AL	Вихід регістра збігу, молодший байт
$25($45)	ICR1H	Регістр захоплення таймера/лічильника 1, старший байт

Продовження таблиці 4.2

1	2	3
$24($4)	ICR1L	Регістр захоплення, молодший байт
$21($41)	WDTCR	Регістр керування сторожовим таймером
$1E($3E)	EEAR	Регістр адреси EEPROM
$1D($3D)	EEDR	Регістр даних EEPROM
$1C($3C)	EECR	Регістр керування EEPROM
$1E($3E)	PORTB	Регістр даних порту B
$18($38)	DDRB	Регістр напрямку даних порту B
$17($37)	PIND	Вхідні лінії порту B
$16($36)	PORTD	Регістр даних порту D
$12($32)	DDRD	Регістр напрямку даних порту D
$11($31)	PIND	Вхідні лінії порту D
$10($30)	UDR	Регістр даних послідовного порту UART
$0C($2C)	USR	Регістр стану послідовного порту UART
$0B($2B)	UCR	Регістр керування послідовного порту UART
$0A($2A)	UBRR	Регістр завдання швидкості послідовного порту UART
$08($28)	ACSR	Регістр керування і стану аналогового компаратора
Примітка. Ділянки, які зарезервовані і не використовуються в AT90S2313, не показані.		

Усі пристрої введення-виведення і периферійні пристрої AT90S2313 розташовуються в просторі введення-виведення. Різні ділянки цього простору доступні через команди IN і OUT, які пересилають дані між одним з 32 регістрів загального призначення і простором введення-виведення. До регістрів $00...$1F можна здійснювати побітовий доступ командами SBI і CBI. Значення окремого біта цих регістрів можна перевірити командами SBIC і SBIS. Додаткову інформацію з цього питання можна знайти в описі системи команд.

При використанні спеціальних команд IN, OUT, SBIS і SBIC повинні використовуватися адреси $00...$3F. При доступі до регістра введення-виведення як до ділянки ОЗП до його адреси необхідно додати $20. У приведеній вище таблиці адреси регістрів у пам’яті даних приведені в дужках.

Регістр стану – SREG

Біт	7	6	5	4	3	2	1	0	
$3F ($5F)	I	T	H	S	V	N	Z	C	SPEG
Чит./зап.	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Поч. знач.	0	0	0	0	0	0	0	0	

Рисунок 4.18 – Регістр стану

Регістр стану розташований за адресою $3F ($5F) простору введення-виведення і визначений у такий спосіб:

Біт 7 – I: загальний дозвіл переривань. Для дозволу переривань цей біт повинний бути встановлений в одиницю. Керування окремими перери-

ваннями виробляється регістрами маски переривань – GIMCK і TIMCK. Якщо біт I = 0, незалежно від стану GIMCK/TIMCK переривання заборонені. Біт I приймає значення, яке дорівнює нулю, апаратно після входу в переривання і відновлюється в стан “1” командою повернення з підпрограми обробки переривань RETI, для дозволу обробки наступних переривань.

Біт 6 – T: збереження біта, який копіюють. Біт з регістра реєстрового файлу може бути скопійований у T командою BST, біт T може бути скопійований у біт реєстрового файлу командою BLD.

Біт 5 – H: прапорець половинного перенесення – ініціює перенесення з молодшої половини байта при деяких арифметичних операціях. Докладніше про це можна прочитати в описі системи команд.

Біт 4 – S: біт знака, S = N XOR V. Біт S дорівнює операції "виключне АБО" між прапорцями N (негативний результат) і V (переповнення доповнення до двох). Якщо після операцій додавання чи віднімання чисел зі знаком прапорець переповнення V буде встановлений у “1”, то результатом буде 9-розрядне число, причому старшим (тобто знаковим) розрядом числа буде прапорець S, а 8 інших бітів результату будуть зберігатися в 8-розрядному регістрі-приймачі операції, що виконувалася.

Біт 3 – V: прапорець переповнення доповнення до двох – його потрібно перевіряти після операцій додавання чи віднімання чисел у додатковому коді. Він встановлюється в тому випадку, якщо результат операції виходить за межі діапазону від -128 до +127. Це – допустимий діапазон чисел зі знаком, які можна зберегти в 8 розрядах. Відповідно, біт V встановиться в “1”, якщо при додаванні двох додатних чисел вийде число більше 127 чи при додаванні двох від’ємних чисел – менше -128.

Біт 2 – N: прапорець негативного результату – встановлюється в “1”, якщо старший (знаковий) розряд (біт 7) дорівнює “1”, що означає, що результат негативний. Необхідно відзначити, що прапорець може бути також встановлений і в результаті логічних операцій.

Біт 1 – Z: прапорець нульового результату – ініціює нульовий результат різних арифметичних і логічних операцій. Докладніше про це можна прочитати в описі системи команд.

Біт 0 – C: прапорець перенесення – ініціює перенесення в арифметичних і логічних операціях. Докладніше про це можна прочитати в описі системи команд.

Покажчик стека SP

Цей 8-розрядний регістр з адресою $3D ($5D) зберігає покажчик стека процесора AT90S2313. Восьми розрядів досить для адресації ОЗП в межах $60 — $DF.

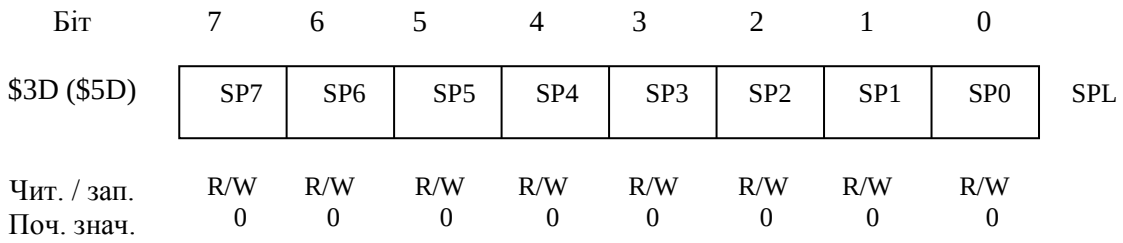


Рисунок 4.19 – Покажчик стека

Покажчик стека вказує на область пам’яті, у якій зберігається адреса повернення з підпрограм і переривань. Також за допомогою стека підпрограми можуть одержувати чи передавати дані.

Область стека в ОЗП повинна бути задана до того, як відбудеться будь-який виклик підпрограми чи будуть дозволені переривання. Покажчик стека зменшується на 1 при записуванні даних у стек командою PUSH, і зменшується на 2 при виклику підпрограми командою CALL або обробці переривання. Покажчик стека збільшується на 1 при виборі даних зі стека командою POP і збільшується на 2 при виконанні команд повернення з підпрограми чи оброблювача переривання (RET чи RETI).

4.4 Перезапуск мікроконтролера і обробка переривань

У AT90S2313 передбачені 10 джерел переривань. Ці переривання і скидання мають різні вектори в області пам’яті програм. Кожному з переривань привласнений окремий біт, що дозволяє дане переривання при встановленні біта в “1” (якщо, звичайно, переривання взагалі дозволені, тобто біт I = 1). Наймолодші адреси пам’яті програм визначені як вектори переривань. Повний список векторів переривань приведений у табл. 4.3. Цей список визначає і пріоритет різних переривань. Менші адреси відповідають більш високому рівню пріоритету. Найвищий рівень у скидання, наступний у INT0 – зовнішньому запиті переривання 0 і т.д.

Нижче наведено типовий фрагмент програми обробки скидання і векторів переривань:

Адреса	**Команда**	**Коментарі**
$000	rjmp RESET	Обробка скидання
$001	rjmp EXT_INT0	Обробка IRQ0
$002	rjmp EXT_INT1	Обробка IRQ1
$003	rjmp TIM_CAPT	Обробка захоплення таймера 1
$004	rjmp TIM_COMP1	Обробка збігу таймера 1
$005	rjmp TIM_OVF1	Обробка переповнення таймера 1
$006	rjmp TIM_OVF0	Обробка переповнення таймера 0
$007	rjmp UART_RXC	Обробка приймання байта
$008	rjmp UART_DRE	Обробка звільнення UDR
$009	rjmp UART_TXC	Обробка передавання байта
$00a	rjmp ANA_COMP	Обробка аналогового компаратора
$00b	**MAIN:**	**Початок основної програми**

Таблиця 4.3 – Скидання і вектори переривань

Вектор	Адреса	Джерело	Найменування переривання
1	$000	RESET	Апаратний вивід Reset, скидання після вмикання живлення і переривання за сторожовим таймером
2	$001	INT0	Зовнішні переривання 0
3	$002	INT1	Зовнішнє переривання 1
4	$003	Timer 1 CAPT1	Захоплення таймера/лічильника 1
5	$004	Timer 1 COMP1	Збіг таймера/лічильника 1
6	$005	Timer 1 OVF1	Переповнення таймера/лічильника 1
7	$006	Timer 0 OVF0	Переповнення таймера/лічильника 0
8	$007	UART,RX	Приймання з послідовного порту завершено
9	$008	UART,UDRE	Регістр даних послідовного порту порожній
10	$009	UART,TX	Передавання з послідовного порту завершено
11	$00A	ANA COMP	Аналоговий компаратор

Джерела скидання

AT90S2313 має три джерела скидання (перезапуску мікроконтролера):

- скидання після вмикання живлення. Процесор скидається при поданні живлення на виводи UCC і GND;
- зовнішнє скидання. Процесор скидається при поданні низького рівня на вивід RESET на час більш двох періодів тактової частоти;
- скидання від сторожового таймера. Процесор скидається після закінчення часу відпрацьовування сторожового таймера, якщо дозволена його робота.

Під час скидання всі регістри введення-виведення встановлюються в початкові значення, програма починає виконуватися з адреси $000, за цією адресою повинна бути записана команда RJMP – відносний перехід на програму обробки скидання. Якщо в програмі не дозволяються переривання і вектори переривань не використовуються, програма може починатися з нульової адреси.

- *Скидання після вмикання живлення*

Спеціальна схема, вбудована в мікроконтролер – ланцюг скидання після вмикання живлення, забезпечує заборону вмикання процесора доти, поки напруга живлення не досягне безпечного рівня. Після того як напруга живлення досягне рівня вмикання, процесор не включається доти, поки вбудований таймер не обробить кілька робочих періодів сторожового таймера.

Оскільки до виводу RESET підімкнений обмежувальний резистор, цей вивід може залишатися непідімкнутим, якщо не потрібно зовнішнє скидання. Час вмикання після подачі живлення може бути збільшено утриманням виводу скидання на низькому рівні.

- *Зовнішнє скидання*

Зовнішнє скидання обробляється за низьким рівнем на виводі RESET. Вивід повинний утримуватися в низькому стані, принаймні, два періоди тактової частоти. Після зняття сигналу “0” з виводу Reset через якийсь час (так само як при скиданні після вмикання живлення) мікроконтролер запускається.

- *Скидання за сторожовим таймером*

Після відпрацьовування заданого при його ініціалізації інтервалу часу сторожовий таймер виробляє імпульс скидання, перезапускаючи при цьому мікроконтролер.

4.5 Обробка переривань

AT90S2313 має два регістри маски переривань GIMCK – загальний регістр маски переривань і TIMCK – регістр маски переривання від таймера/лічильника.

Коли виникає переривання, біт глобального дозволу переривань I скидається (йому привласнюється значення “0”) і всі переривання забороняються. Програма користувача може встановити цей біт для дозволу переривань. Прапорець дозволу переривань I автоматично встановлюється в “1” при виконанні команди виходу з переривання – RETI.

Для переривань, які викликаються статичними подіями, наприклад, переповнення таймера_0, прапорець відповідного переривання встановлюється при виникненні події. Якщо прапорець переривання очищений і є умова виникнення переривання, прапорець не буде встановлений, поки не відбудеться наступна подія.

Коли програмний лічильник встановлюється на поточний вектор переривання для його обробки, прапорець, який згенерований перериванням, апаратно скидається. Деякі прапорці переривання можуть бути скинуті записом логічної одиниці в біт, що відповідає прапорцю.

Біт	7	6	5	4	3	2	1	0	
$3B ($5B)	INT1	INT0	-	-	-	-	-	-	GIMSK
Чит. / зап.	R/W	R/W	R	R	R	R	R	R	
Поч. знач.	0	0	0	0	0	0	0	0	

Рисунок 4.20 – Загальний регістр маски переривань GIMSK

Біт 7 – INT1: запит зовнішнього переривання 1 дозволений. Коли цей біт установлений, а також установлений біт I, дозволяється переривання від зовнішнього виводу. Біти керування запуском переривання (ISC11 і ISC10) у регістрі керування мікроконтролером (MCUCR) визначають за якою подією відпрацьовується переривання – за спадним чи наростаючим фронтом, чи за рівнем. Активність на виводі приводить до виникнення пе-

реривань, навіть якщо вивід має конфігурацію як вихід. При виникненні переривання виконується перехід за адресою $001 для виконання підпрограми обробки переривання.

Біт 6 – INT0: запит зовнішнього переривання 0 дозволений. Коли цей біт установлений, а також установлений біт I, дозволяється переривання від зовнішнього виводу. Біти керування запуском переривання (ISC01 і ISC00) у регістрі керування мікроконтролером (MCUCR) визначають за якою подією відпрацьовується переривання – за спадним чи наростаючим фронтом, чи за рівнем. Активність на виводі приводить до виникнення переривань, навіть якщо вивід має конфігурацію як вихід. При виникненні переривання виконується перехід за адресою $001 для виконання підпрограми обробки переривання.

Біти 5...0 — у AT90S2313 ці біти зарезервовані і завжди читаються як “0”.

Біт	7	6	5	4	3	2	1	0	
$3A ($5A)	INTF1	INTF0	-	-	-	-	-	-	GIFR
Чит. / зап.	R/W	R/W	R	R	R	R	R	R	
Поч. знач.	0	0	0	0	0	0	0	0	

Рисунок 4.21 – Загальний регістр прапорців переривань GIFR

Біт 7 – INTF1: прапорець зовнішнього переривання 1. При виникненні на виводі INT1 події, що викликає переривання, INTF1 встановлюється в “1”. Якщо встановлені біт 1 регістра SREG і біт INT1 у GIMCK, відбувається перехід на вектор переривання за адресою $002. Прапорець очищається після виконання оброблювача переривання. Крім того, прапорець можна очистити, записавши в нього логічну одиницю.

Біт 6 – INTF0: прапорець зовнішнього переривання 0. При виникненні на виводі INT0 події, що викликає переривання, INTF0 встановлюється в “1”. Якщо встановлені біт 1 регістра SREG і біт INT0 у GIMCK, відбувається перехід на вектор переривання за адресою $001. Прапорець очищається після виконання оброблювача переривання. Крім того, прапорець можна очистити, записавши в нього логічну одиницю.

Біти 5...0 – у AT90S2313 зарезервовані і завжди читаються як “0”.

Біт	7	6	5	4	3	2	1	0	
$39 ($59)	TOIE1	OCIE1A	-	-	TICIE1	-	TOIE0	-	TIMSK
Чит. / зап.	R/W	R/W	R	R	R/W	R	R/W	R	
Поч. знач.	0	0	0	0	0	0	0	0	

Рисунок 4.22 – Регістр маски переривань від таймера/лічильника TIMSK

Біт 7 – TOIE1: дозвіл переривання при переповненні тайме-

ра/лічильника 1. Якщо встановлені цей біт і біт дозволу переривань у регістрі стану, дозволені переривання при переповненні таймера/лічильника 1. Відповідне переривання (вектор $005) виконується при переповненні таймера/лічильника 1. У регістрі прапорців таймерів/лічильників (TIFR) встановлюється прапорець переповнення. Якщо таймер/лічильник 1 працює в режимі широтно-імпульсної модуляції (ШІМ), прапорець переповнення встановлюється при зміні напрямку лічби, при значенні $0000.

Біт 6 – OCIE1A: дозвіл переривання при збігу таймера/лічильника 1. Якщо встановлені біт OCIE1A і біт дозволу переривання в регістрі стану, дозволені переривання при збігу таймера/лічильника 1. Переривання (вектор $004) виконується при рівності таймера/лічильника 1 і регістра збігу. В прапорцевому регістрі TIFR встановлюється в “1” прапорець збігу.

Біти 5, 4 – у AT90S2313 зарезервовані і завжди читаються як “0”.

Біт 3 – TICIE1: дозвіл переривання за входом захоплення. Якщо встановлені біт TICIE1 і біт дозволу переривання в регістрі стану, дозволені переривання за входом захоплення. Відповідне переривання (вектор $003) виконується за сигналом захоплення на виводі 11 (PD6/ICP). В прапорцевому регістрі TIFR встановлюється прапорець захоплення.

Біт 2 – у AT90S2313 зарезервований і завжди читається як “0”.

Біт 1 – TOIE0: дозвіл переривання при переповненні таймера/лічильника 0. Якщо цей біт встановлений у “1” і біт I у регістрі стану встановлений у “1”, дозволені переривання при переповненні таймера/лічильника 0. При виникненні переповнення виконується перехід на відповідний вектор переривання ($006). Прапорець переповнення (TOV0) у прапорцевому регістрі переривань (TIFR) таймерів/лічильників встановлюється в “1”.

Біт 0 – у AT90S2313 зарезервований і завжди читається як “0”.

Біт	7	6	5	4	3	2	1	0	
$38 ($58)	TOV1	OCF1A	-	-	ICF1	-	TOV0	-	TIFR
Чит. / зап.	R/W	R/W	R	R	R/W	R	R/W	R	
Поч. знач.	0	0	0	0	0	0	0	0	

Рисунок 4.23 – Регістр прапорців переривань від таймерів/лічильників

Біт 7 – TOV1: прапорець переповнення таймера/лічильника 1. Прапорець TOV1 встановлюється в “1” при виникненні переповнення таймера/лічильника 1. Прапорець TOV1 скидається апаратно при виконанні відповідного вектора обробки переривання. Крім того, прапорець можна скинути, записавши в нього логічну одиницю. Якщо встановлені біт I у SREG і біт TOIE1 у TIMCK, при встановленні біта TOV1 виконується переривання при переповненні таймера/лічильника 1. У режимі ШІМ цей біт встановлюється, коли таймер/лічильник 1 змінює напрямок лічби при значенні

$0000.

Біт 6 – OCF1A: прапорець виходу збігу 1A. Прапорець встановлюється в “1” якщо відбувається збіг значення таймера/лічильника 1 і даних у регістрі OCR1A. Прапорець очищається апаратно при виконанні відповідного вектора переривання. Крім того, прапорець можна скинути, записавши в нього логічну одиницю. Якщо встановлені біт I у SKEG і біт OCIE1A у TIMCK, при встановленні біта OCF1A виконується переривання.

Біти 5,4 – у AT90S2313 зарезервовані і завжди читаються як “0”.

Біт 3 – ICF1: прапорець входу захоплення 1. Біт встановлюється в “1” при виникненні події захоплення за входом. Значення таймера/лічильника 1 скопійовано в регістр захоплення за входом I. ICF1 очищається при виконанні відповідного вектора обробки переривання. Крім того, прапорець можна очистити, записавши в нього логічну одиницю.

Біт 2 – у AT90S2313 зарезервований і завжди читається як “0”.

Біт 1 – TOV0: прапорець переповнення таймера лічильника 1. Прапорець TOV0 установлюється при переповненні таймера/лічильника 0. Прапорець скидається апаратно при виконанні відповідного вектора переривання. Крім того, прапорець можна очистити, записавши в нього логічну одиницю. Якщо встановлені біт I у SREG і біт TOIE0 у TIMCK, при установленні біта TOV0 виконується переривання при переповненні таймера/лічильника 0.

Біт 0 – у AT90S2313 зарезервований і завжди читається як “0”.

Зовнішні переривання

Зовнішні переривання керуються виводами INT0 і INT1. Зверніть увагу на те, що переривання обробляються навіть коли виводи мають конфігурацію як виходи. Це дозволяє генерувати програмні переривання. Зовнішні переривання можуть виникати за спадним чи наростаючим фронтом, а також за низьким рівнем. Це встановлюється в регістрі керування процесором MCUCR. Якщо зовнішні переривання дозволені і мають конфігурацію на відпрацьовування за рівнем, переривання буде вироблятися доти, поки вивід утримується в низькому стані.

Керування роботою зовнішніх переривань розглянуто при описі регістра керування процесором MCUCR.

Час реакції на переривання

Мінімальний час реакції на кожне з передбачених в процесорі переривань — 4 періоди тактової частоти. Після 4 циклів викликається програмний вектор, що обробляє дане переривання. За ці 4 цикли програмний лічильник записується в стек, покажчик стека зменшується на 2. Програмний вектор являє собою відносний перехід на підпрограму обслуговування переривання, і цей перехід займає 2 періоди тактової частоти. Якщо переривання відбувається під час виконання команди, що триває кілька циклів, перед викликом переривання завершується виконання цієї команди.

Вихід із програми обслуговування переривання займає 4 періоди тактової частоти. За ці 4 періоди зі стека відновлюється програмний лічиль-

ник. Після виходу з переривання процесор завжди виконує ще одну команду, перш ніж обслужити будь-яке відкладене переривання.

Помітимо, що регістр стану SREG апаратно не зберігається процесором як при виклику підпрограм, так і при обслуговуванні переривань. Якщо програма вимагає збереження SREG, воно повинно вироблятися програмою користувача.

Біт	7	6	5	4	3	2	1	0	
$35 ($55)	-	-	SE	SM	ISC11	ISC10	ISC01	ISC00	MCUCR
Чит. / зап.	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Поч. знач.	0	0	0	0	0	0	0	0	

Рисунок 4.24 – Регістр керування мікроконтролером MCUCR

Біти 7, 6 – у AT90S2313 ці біти зарезервовані і завжди читаються як “0”.

Біт 5 – SE: дозвіл режиму Sleep. Цей біт повинний бути встановлений у “1”, щоб при виконанні команди SLEEP процесор переходив у режим зниженого енергоспоживання. Цей біт повинний бути встановлений у “1” до виконання команди SLEEP.

Біт 4 – SM: режим Sleep. Цей біт вибирає один із двох режимів зниженого енергоспоживання. Якщо біт скинутий як режим Sleep вибирається холостий режим (Idle mode). Якщо біт установлений – вибирається економічний режим (Power Down). Особливості кожного з режимів будуть розглянуті нижче.

Біти 3, 2 – ISC11, ISC10: біти керування спрацьовуванням переривання 1. Зовнішнє переривання активується виводом INT1, якщо встановлений прапорець I регістра стану SREG і установлена відповідна маска в регістрі GIMCK. Спрацьовування за рівнем і фронтами задається в такий спосіб:

Таблиця 4.4 – Керування спрацьовуванням переривання 1

ISC11	ISC10	Опис
0	0	Запит генерується за низьким рівнем на вході INT1
0	1	Зарезервовано
1	0	Запит на переривання за спадним фронтом на вході INT1
1	1	Запит на переривання за наростаючим фронтом на вході INT1
Примітка. При зміні бітів ISC11/ISC10, переривання INT1 повинно бути заборонено очищенням відповідного біта в регістрі GIMCK. У противному випадку переривання може виникнути під час зміни бітів.		

Біти 1, 0 – ISC01, ISC00: біти керування спрацьовуванням переривання 0. Зовнішнє переривання активується виводом INT0, якщо встановлений прапорець I регістра стану SREG і установлена відповідна маска в регістрі GIMCK. У табл. 4.5 приведено установлення бітів для завдання спрацьовування за рівнем і фронтами.

Таблиця 4.5 – Керування спрацьовуванням переривання 0

ISC01	ISC00	Опис
0	0	Запит генерується за низьким рівнем на вході INT0
0	1	Зарезервовано
1	0	Запит на переривання за спадним фронтом на вході INT0
1	1	Запит на переривання за наростаючим фронтом на вході INT0
Примітка. При зміні бітів ISC01 і ISC00, переривання за входом INT0 повинні бути заборонені скиданням біта дозволу переривання в регістрі GIMCK. У противному випадку переривання може відбутися при зміні значення бітів.		

4.6 Режими пониженого енергоспоживання

Для запуску режиму зниженого енергоспоживання повинний бути встановлений у стан “1” біт SE регістра MCUCR і повинна бути виконана команда SLEEP. Якщо під час перебування в режимі зниженого споживання відбувається одне з дозволених переривань, процесор починає працювати, виконує підпрограму обробки переривання і продовжує виконання програми з команди, що випливає за SLEEP. Вміст реєстрового файлу і пам’яті введення/виведення не змінюється. Якщо в режимі зниженого споживання відбувається скидання, процесор починає виконання програми з вектора скидання.

Якщо для виведення з економічного режиму використовується переривання за рівнем, низький рівень повинний утримуватися на час, достатній для запуску генератора тактових імпульсів — не менше 16 мс. Інакше прапорець переривання може повернутися в “0” до того, як процесор почне роботу.

Режим холостого ходу

Коли біт SM скинутий, команда SLEEP переводить процесор в режим холостого ходу (Idle mode). ЦПУ зупиняється, але таймери/лічильники, сторожовий таймер і система переривань продовжують працювати. Це дозволяє процесору відновляти роботу як від зовнішніх переривань, так і при переповненні таймерів/лічильників, і при скиданні від сторожового таймера. Якщо переривання від аналогового компаратора не потрібно, аналоговий компаратор може бути вимкнутий установленням біта ACD регістра ACSR. Це зменшує споживану потужність.

Економічний режим

Коли біт SM = 1, команда SLEEP переводить процесор в економічний режим (Power Down Mode). У цьому режимі зупиняється генератор тактових імпульсів. Програміст може дозволити роботу сторожового таймера в цьому режимі. Якщо сторожовий таймер дозволений, процесор виходить з економічного режиму після відпрацьовування періоду сторожового таймера. Якщо сторожовий таймер заборонений, вихід з економічного режиму може відбутися тільки при зовнішньому скиданні чи зовнішньому перериванні за рівнем.

4.7 Таймери/лічильники

У AT90S2313 передбачені два таймери/лічильника загального призначення: 8-розрядний і 16-розрядний. Кожний з таймерів індивідуально вмикається до одного з виходів 10-розрядного попереднього подільника частоти. Обидва таймери можуть використовуватися як таймери з внутрішнім джерелом імпульсів чи як лічильники імпульсів, що надходять ззовні. Як джерело імпульсів для таймерів можна вибрати сигнал тактовою частотою мікроконтролера (СК), імпульси попереднього подільника (СК/8, СК/64, СК/256 чи СК/1024) чи імпульси з відповідного зовнішнього виводу. На рис. 4.25 зображена структурна схема попереднього подільника частоти. Крім того, таймери можуть бути зупинені.

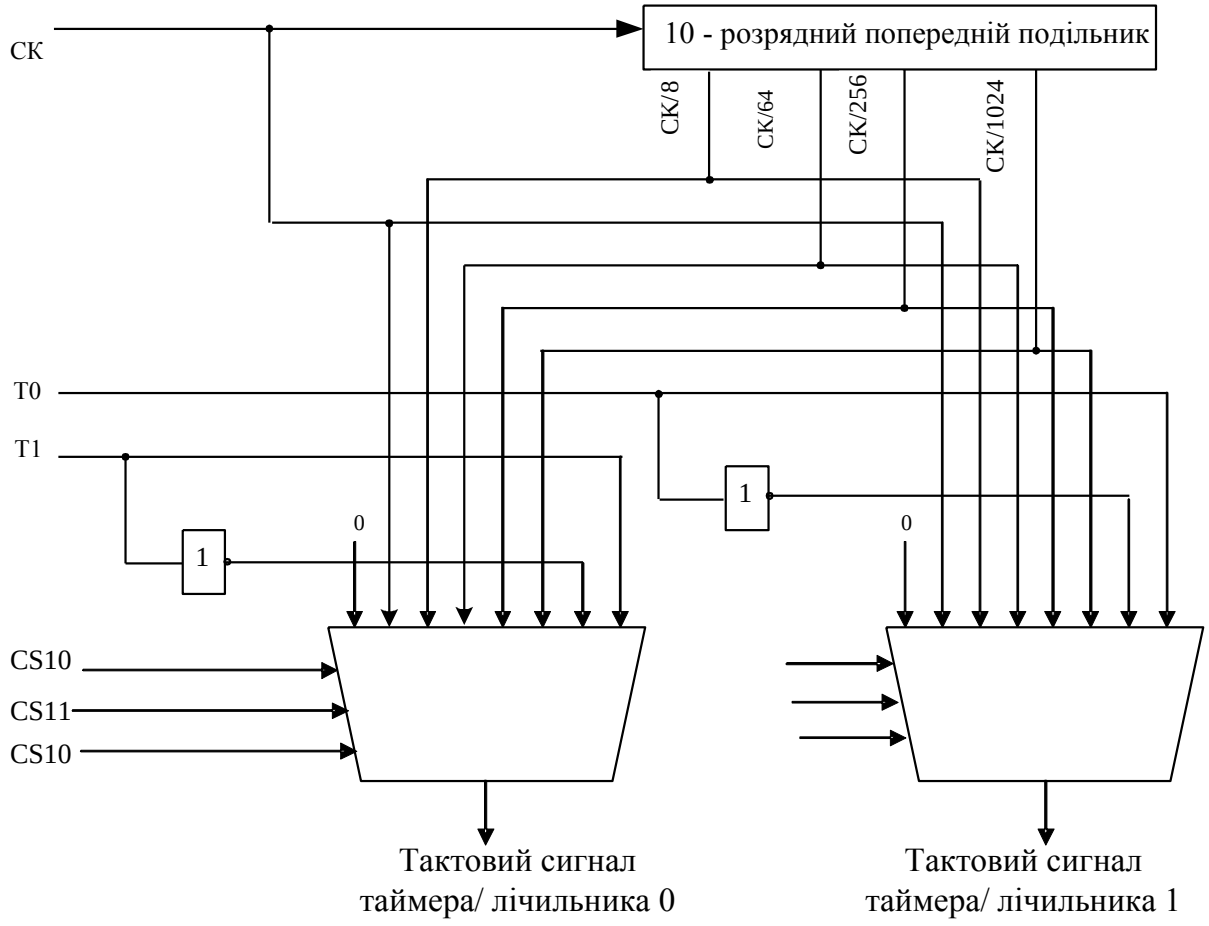


Рисунок 4.25 – Попередній подільник тактової частоти для таймерів

8-розрядний таймер/лічильник може одержувати імпульси тактової частоти – СК, імпульси з попереднього подільника (СК8, СК/64, СК/256 чи СК/1024), імпульси з зовнішнього виводу чи бути зупинений відповідними установленнями регістра TCCR0. Прапорець переповнення таймера знаходиться в регістрі TIFR. Біти керування таймером розташовані в регістрі TCCR0. Дозвіл і заборона переривань від таймера керується регістром TIMCK.

Під час роботи таймера/лічильника від зовнішнього сигналу цей сигнал синхронізується з тактовим генератором мікроконтролера. Для правильної обробки зовнішнього сигналу мінімальний час між сусідніми імпульсами повинний перевищувати період тактової частоти процесора. Сигнал зовнішнього джерела обробляється за спадним фронтом тактової частоти процесора.

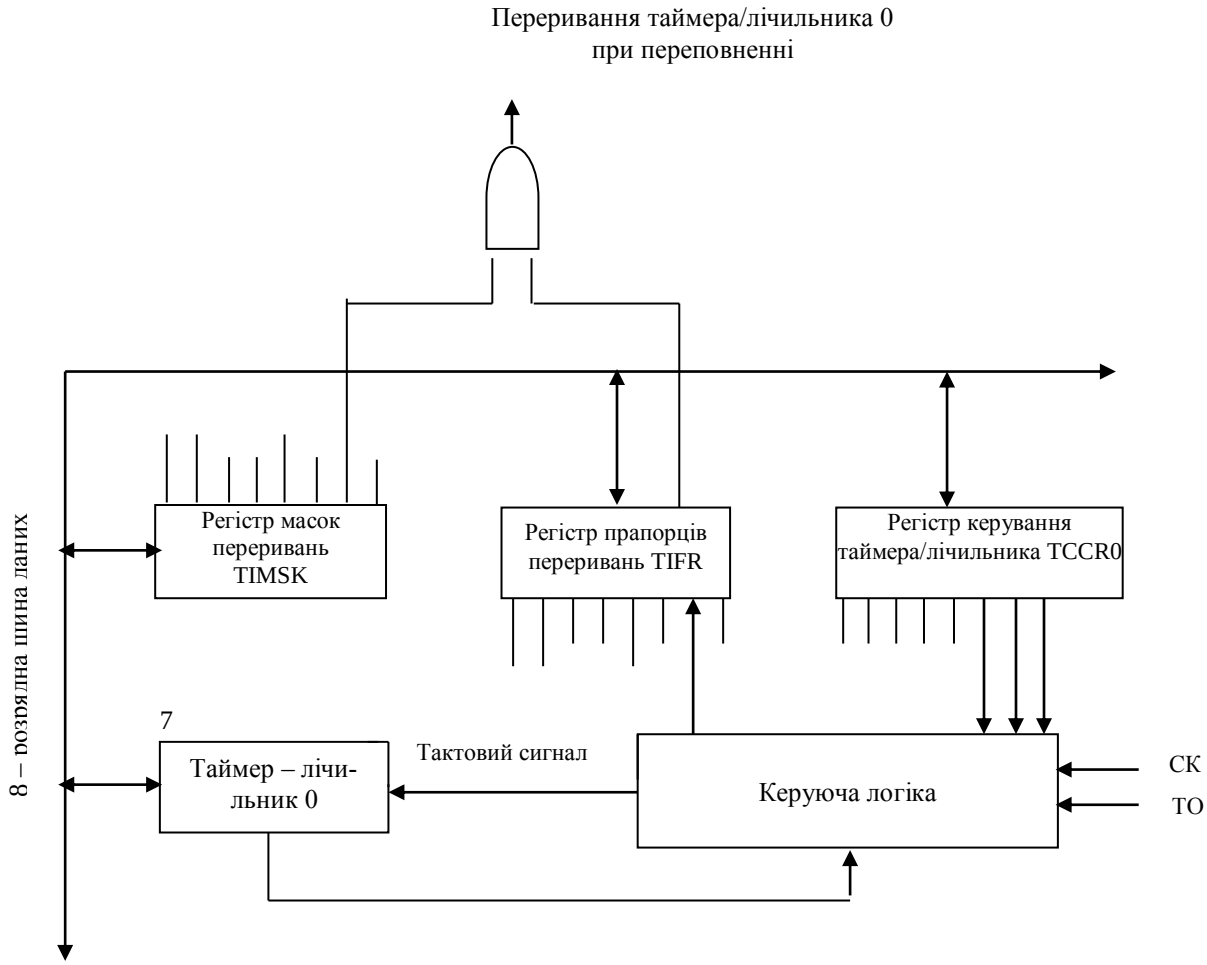


Рисунок 4.26 – Блок-схема таймера/лічильника 0

Біт	7	6	5	4	3	2	1	0	
$33 ($53)	-	-	-	-	-	CS02	CS01	CS00	TCCR0
Чит. / зап.	R 0	R 0	R 0	R 0	R 0	R/W 0	R/W 0	R/W 0	

Рисунок 4.27 – Регістр керування таймером/лічильником 0 TCCR0

Біти 7...3 – у AT90S2313 зарезервовані і завжди читаються як “0”. Біти 2, 1, 0 - CS02, CS01, CS00 — вибір тактової частоти. Ці біти задають коефіцієнт розподілу попереднього подільника.

Таблиця 4.6 – Вибір коефіцієнта попереднього подільника

CS02	CS01	CS00	Опис
0	0	0	Таймер/лічильник зупинений

0	0	1	СК
0	1	0	СК/8
0	1	1	СК/64
1	0	0	СК/256
1	0	1	СК/1024
1	1	0	Зовнішній вивід ТЕ, наростаючий фронт
1	1	1	Зовнішній вивід ТЕ, спадний фронт

Умова «Таймер/лічильник зупинений» забороняє чи дозволяє функціонування таймера/лічильника. У режимах розподілу використовується частота тактового генератора мікроконтролера. При роботі від зовнішнього джерела попередньо повинний бути установлений відповідний біт регістра напрямку даних.

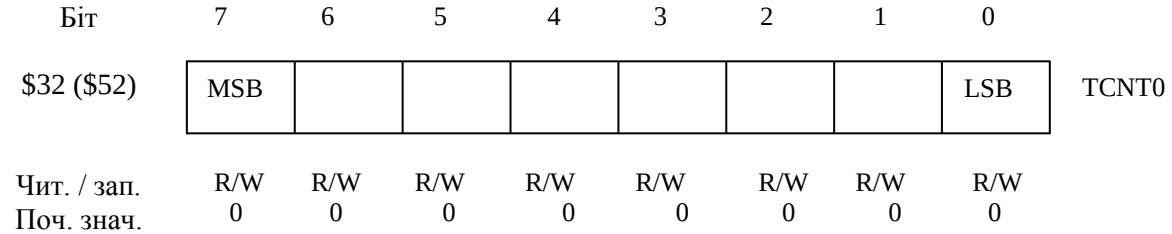


Рисунок 4.28 – Таймер \ лічильник 0 TCNT0

Таймер/лічильник реалізований як наростаючий лічильник з можливістю читання і записування. При записуванні таймера/лічильника, якщо присутні тактові імпульси, таймер/лічильник продовжує лічбу в тактовому циклі таймера, що іде за операцією записування.

16-розрядний таймер/лічильник може одержувати імпульси тактової частоти — *СК* : імпульси з попереднього подільника (*СК/8*, *СК/64*, *СК/256* чи *СК/1024*), імпульси з зовнішнього виводу, що визначаються відповідними установленнями регістра TCCR1A. Прапорці стану таймера (переповнення, збігу і захоплення) і керуючі сигнали знаходяться в регістрі TIFR. Дозвіл і заборона переривань від таймера 1 керується регістром TIMCK.

При роботі таймера/лічильника 1 від зовнішнього сигналу цей сигнал синхронізується з тактовим генератором мікроконтролера. Для правильної обробки зовнішнього сигналу мінімальний час між сусідніми імпульсами повинний перевищувати період тактової частоти процесора. Сигнал зовнішнього джерела обробляється за спадним фронтом тактової частоти процесора.

Таймер/лічильник 1 підтримує функцію збігу, використовуючи регістр збігу OCR1A в якості джерела для порівняння з вмістом лічильника. Функція збігу підтримує очищення лічильника і перемикання виходу при збігу. Таймер/лічильник 1 можна використовувати як 8-, 9- чи 10-розрядний широтно-імпульсний модулятор (ШІМ). У цьому режимі лічильник і регістр OCR1 працюють як захищені від брязкання незалежний ШІМ з відцентрованими імпульсами. Докладніше ця функція буде описана

нижче.

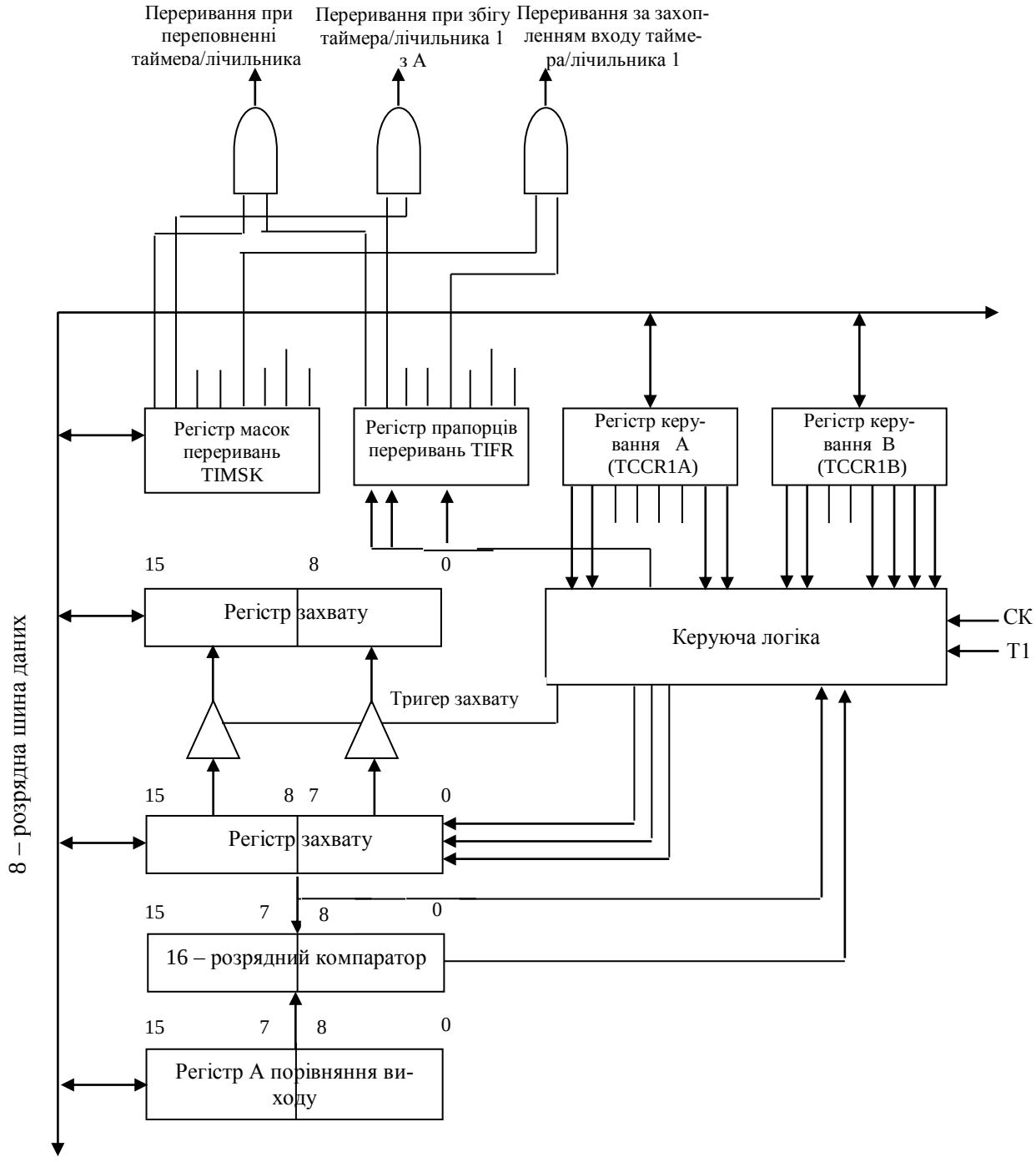


Рисунок 4.29 – Блок-схема таймера/лічильника 1

Функція захоплення за входом передбачає захоплення вмісту таймера/лічильника 1 у регістр захоплення ICR1 і керується зовнішнім сигналом на вході захоплення – ICP. Робота режиму захоплення визначається керуючим регістром TCCR1. При роботі захоплення за входом може бути увімкнута схема подавлення шуму, при цьому сигнал захоплення виникає тільки в тому випадку, якщо подія, що керує захопленням, спостерігається протягом 4 машинних циклів.

Біт	7	6	5	4	3	2	1	0	
$2F ($4F)	COM1A1	COM1A0	-	-	-	-	PWM11	PWM10	TCCR1A
Чит. / зап.	R/W	R/W	R	R	R	R	R/W	R/W	
Поч. знач.	0	0	0	0	0	0	0	0	

Рисунок 4.30 – Регістр А керування таймером/лічильником 1 TCCR1A

Біти 7, 6 – COM1A1, COM1A0: режим виходу збігу, біти 1 і 0. Ці керуючі біти задають відгук виводу OC1 процесора на збіг регістра порівняння і таймера/лічильника 1. Оскільки альтернативна функція порту, що відповідає біту напрямку, необхідно встановлювати вивід на вихід. Конфігурації керуючих бітів показані в таблиці 4.7.

Таблиця 4.7 – Установлення режиму збігу

COM1A1	COM1A0	Опис
0	0	Робота ШІМ заборонена
0	1	8-розрядний ШІМ
1	0	9-розрядний ШІМ
1	1	10-розрядний ШІМ

У режимі ШІМ ці біти мають інші функції, що зазначені в табл. 4.11.

При зміні бітів COM1A1 і COM1A0 переривання за збігом повинно бути заборонено очищенням відповідного біта у регістрі TIMCK. У противному випадку переривання може відбутися під час зміни цих бітів.

Біти 5..2 – у AT90S2313 зарезервовані і завжди читаються як “0”.

Біти 1, 0 – PWM11, PWM10: біти встановлення ШІМ. Ці біти встановлюють режим роботи таймера/лічильника 1 у якості ШІМ (табл. 4.8). Докладніше цей режим буде розглянутий нижче.

Таблиця 4.8 – Установлення режиму роботи ШІМ

PWM11	PWM10	Опис
0	0	Робота ШІМ заборонена
0	1	8-розрядний ШІМ
1	0	9-розрядний ШІМ
1	1	10-розрядний ШІМ

Біт	7	6	5	4	3	2	1	0	
$2F ($4F)	ICNC1	ICES1	-	-	CTC1	CS12	CS11	CS10	TCCR1B
Чит. / зап.	R/W	R/W	R	R	R\W	R\W	R/W	R/W	
Поч. знач.	0	0	0	0	0	0	0	0	

Рисунок 4.31 – Регістр керування таймером/лічильником 1 TCCR1B

Біт 7 – ICNC1: подавлювач вхідного шуму входу захоплення. Якщо ICNC1 = 0, подавлення вхідного шуму входу захоплення заборонено. При цьому захоплення спрацьовує за першим заданим (наростаючим чи спадним) фронтом сигналу на виводі ICP. При встановленні біта обробляються чотири послідовні вибірки сигналу на виводі ICP. Для спрацьовування захоплення усі вибірки повинні відповідати рівню, заданому бітом ICES1. Частота вибірок дорівнює тактовій частоті процесора.

Біт 6 – ICES1: вибір фронту сигналу захоплення. Якщо біт ICES1=0, вміст таймера/лічильника 1 записується в регістр захоплення за спадним фронтом сигналу на виводі ICP. Якщо біт встановлений – за наростаючим фронтом сигналу.

Біти 5, 4 – у AT90S2313 зарезервовані і завжди читаються “0”.

Біт 3 – CTC1: очищення таймера/лічильника 1 за збігом. Якщо біт CTC1 = 1, таймер/лічильник 1 встановлюється в $000 у такті, що випливає за подією збігу. Якщо біт скинутий, таймер/лічильник 1 продовжує рахувати, поки не буде зупинений, скинутий, відбудеться його переповнення чи зміна напрямку лічби. У режимі ШІМ цей біт не працює.

Біти 2, 1, 0 – CS12, CS11, CS10: вибір джерела тактування. Ці біти визначають джерело лічильних імпульсів для таймера/лічильника 1.

Таблиця 4.9 – Вибір джерела лічильних імпульсів

CS12	CS11	CS10	Опис
0	0	0	Таймер/лічильник зупинений
0	0	1	СК
0	1	0	СК/8
0	1	1	СК/64
1	0	0	СК/256
1	0	1	СК/1024
1	1	0	Спадний фронт на виході T1
1	1	1	Наростаючий фронт на виході T1

Біт	15	14	13	12	11	10	9	8	
$2D ($4D)	MSB								TCNT1H
								LSB	TCNT1L
$2C ($4C)	7	6	5	4	3	2	1	0	
Чит./зап.	R/W	R/W	R/W	R/W	R\W	R\W	R/W	R/W	
	R/W	R/W	R/W	R/W	R\W	R\W	R/W	R/W	
	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Рисунок 4.32 – Таймер/лічильник 1 TCNT1H і TCNT1L

Це 16-розрядний регістр, що містить поточне значення таймера/лічильника 1. Щоб читання і запис двох байтів лічильника відбувалося синхронно, для роботи з ним використовується тимчасовий регістр (TEMP).

Запис у таймер/лічильник 1: при записі старшого байта в TCNT1H записувані дані поміщаються в регістр TEMP. Потім, при записуванні молодшого байта, він разом з даними з TEMP переписується в таймер/лічильник 1. Таким чином, при записуванні 16-розрядного значення першим повинний записуватися байт у TCNT1H;

Читання таймера/лічильника 1: при читанні молодшого байта з TCNT1L він посилається в процесор, а дані з TCNT1H переписуються в регістр TEMP, тобто одночасно читаються всі 16 розрядів. При наступному читанні регістра TCNT1H дані беруться з регістра TEMP.

Таймер/лічильник 1 організований як підсумовувальний лічильник (у режимі ШІМ – підсумовувальний/віднімальний) з можливістю читання і записування. Якщо обране джерело тактових імпульсів для таймера/лічильника 1, після записування в нього нового значення він продовжує лічбу у наступному періоді тактової частоти.

Біт	15	14	13	12	11	10	9	8	
$2B ($4B)	MSB								OCR1AH
$2A ($4A)								LSB	OCR1AL
	7	6	5	4	3	2	1	0	
Чит. / зап.	R/W	R/W	R/W	R/W	R\W	R\W	R/W	R/W	
	R/W	R/W	R/W	R/W	R\W	R\W	R/W	R/W	
Поч. знач.	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Рисунок 4.33 – Регістр збігу А таймера/лічильника 1 – OCR1AH і OCR1AL

Регістр збігу – 16-розрядний регістр, доступний для читання і записування. У цьому регістрі зберігаються дані, що безупинно порівнюються з поточним значенням таймера/лічильника 1. Дія за збігом задається регістрами керування таймером/лічильником 1 і регістром стану.

Оскільки регістр OCR1A 16-розрядний, при записуванні нового значення в регістр, для того щоб обидва байти регістра записувалися одночасно, використовується тимчасовий регістр. При записуванні старшого байта дані поміщаються в тимчасовий регістр, що переписується в OCR1AH при записуванні молодшого байта в OCR1AL. Таким чином, для записування в регістр першим повинний записуватися старший байт.

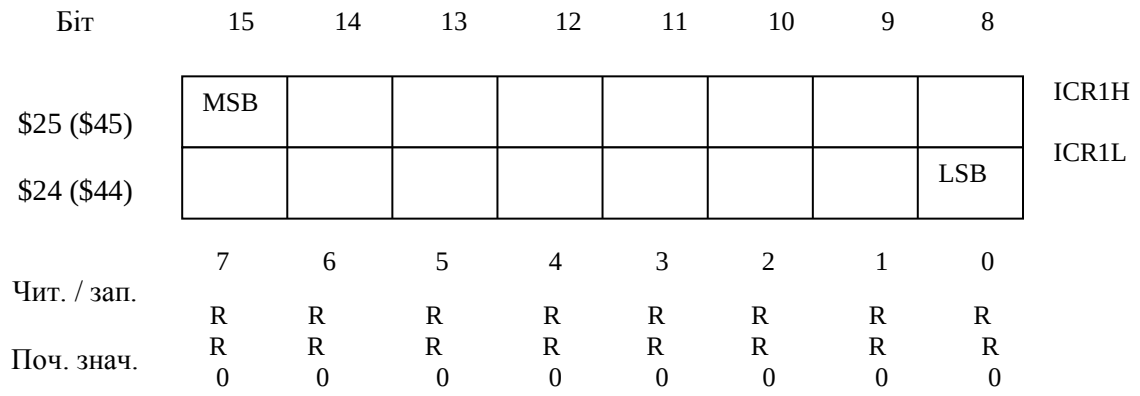


Рисунок 4.34 – Регістр захоплення таймера/лічильника 1 — ICR1H і ICR1L

Регістр захоплення — 16-розрядний регістр, доступний тільки для читання. За наростаючим чи спадним фронтом (відповідно до вибору фронту імпульсу захоплення ICES1) сигналу на виводі ICP поточне значення таймера/лічильника 1 переписується в регістр захоплення ICR1. У цей же час установлюється прапорець захоплення ICF1.

Оскільки регістр захоплення є 16-розрядним, для читання значення необхідно, щоб обидва байти прочиталися одночасно, тому використовується тимчасовий регістр. При читанні молодшого байта ICR1L старший байт регістра ICR1H переписується в тимчасовий регістр. При читанні старшого байта він вибирається з тимчасового регістра. Таким чином, для читання 16-розрядного регістра першим повинний читатися молодший байт.

Таймер/лічильник у режимі ШІМ

При виборі режиму широтно-імпульсної модуляції (ШІМ) таймер/лічильник 1 і регістр збігу OCR1A формують 8-, 9- чи 10-розрядний безупинний, вільний від «дрижання» і правильний за фазою сигнал, виведений на вивід PB3(OC1). Таймер/лічильник 1 працює як реверсивний лічильник, що рахує від 0 до кінцевого значення (табл. 4.10). При досягненні кінцевого значення лічильник починає рахувати в зворотну сторону до нуля, після чого робочий цикл повторюється. Коли значення лічильника збігається з вісьмома, дев’ятьма чи десятьма молодшими бітами регістра OCR1A, вивід PD1(OC1) встановлюється чи скидається відповідно до встановлення бітів COM1A1 і COM1A0 у регістрі TCCR1 (табл. 4.11).

Таблиця 4.10 – Кінцеве значення таймера і частота ШІМ

Дозвіл ШІМ	**Кінцеве значення таймер**	**Частота ШІМ**
8 бітів	$00FF(255)	Ftc/510
9 бітів	$01FF(511)	Ftc/1022
10 бітів	$03FF(1023)	Ftc/2046

У режимі ШІМ при записуванні в регістр OCR1A 10 молодших бітів передаються в тимчасовий регістр і переписуються тільки при досягненні таймером/лічильником кінцевого значення. При цьому усувається поява

несиметричних імпульсів (дрижання), що неминучі при асинхронному записуванні OCR1A.

Таблиця 4.11 – Установлення режиму збігу при роботі ШІМ

COM1A1	COM1A0	Вплив на вихід OC1
0	0	Не підключений
0	1	Не підключений
1	0	Очищається при збігу, для зростання лічильника і скидається для зменшення (ШІМ, який не інвертує)
1	1	Очищається при збігу, для зменшення лічильника і скидається для зростання (ШІМ, що інвертує)

Якщо OCR1A містить значення $000 чи кінцеве значення (TOP), вивід OC1 залишається в тому стані, що визначається станом COM1A1 і COM1A0. Це показано в табл. 4.12.

Таблиця 4.12 – Вихід ШІМ для OCR = $0000 чи TOP

COM1A1	COM1A0	OCR1A	Вихід OC1
1	0	$0000	Низький
1	0	TOP	Високий
1	1	$0000	Високий
1	1	TOP	Низький

У режимі ШІМ прапорець переповнення таймера 1 (TOV1) установлюється, коли лічильник змінює напрямок лічби в точці $0000. Переривання при переповненні таймера 1 працює як при нормальному режимі роботи таймера/лічильника, тобто воно виконується, якщо встановлений прапорець TOV1 і дозволені відповідні переривання. Те ж саме стосується прапорця збігу і переривання за збігом.

Сторожовий таймер

Сторожовий таймер працює від окремого вбудованого генератора, що працює на частоті 1 МГц (типове значення частоти для живлення 5 В). Керуючи попереднім подільником сторожового таймера, можна задавати інтервал скидання таймера від 16 до 2048 мс. Команда WDR скидає сторожовий таймер. Для роботи сторожового таймера можна вибрати одне з восьми значень частоти, що дозволяє в широких межах змінювати час між виконанням команди WDR і скиданням процесора. При відпрацьовуванні періоду роботи сторожового таймера, якщо не надійшла команда WDR, AT90S2313 скидається, виконання програми продовжується з вектора скидання.

Для запобігання небажаного вимкнення сторожового таймера, для його заборони повинна виконуватися визначена послідовність, що описана при розгляді регістра WDTCR.

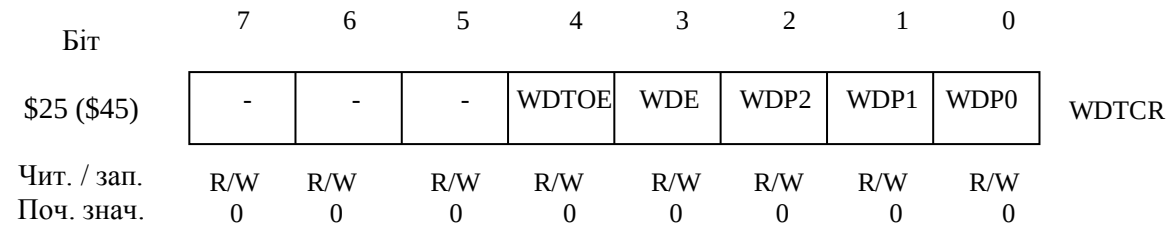


Рисунок 4.35 – Регістр керування сторожовим таймером — WDTCR

Біти 7..5 – у AT90S2313 зарезервовані і завжди читаються як “0”.

Біт 4 – WDT0E – дозвіл вимикання сторожового таймера. При очищенні біта WDE цей біт повинний бути встановлений, інакше робота сторожового таймера не припиняється. Через чотири такти після встановлення цього біта він апаратно скидається.

Біт 3 – WDE – дозвіл сторожового таймера. Якщо біт установлений, робота сторожового таймера дозволена, якщо біт скинутий — заборонена. Скидання біта виробляється тільки в тому випадку, якщо біт WDT0E встановлений у “1”.

Біти 2...0 – WDP2...0 – біти попереднього подільника сторожового таймера. Якщо робота сторожового таймера дозволена, ці біти визначають попередній коефіцієнт розподілу для сторожового таймера. У табл. 4.13 приведені різні значення установлення попереднього подільника і відповідні їм тимчасові інтервали при напрузі живлення U_{cc} = 5 В.

Для заборони увімкнутого сторожового таймера повинна виконуватися така процедура.

1. Однією командою записати 1 у WDT0E і WDE. Одиниця в WDE повинна записуватися навіть у тому випадку, якщо цей біт був установлений перед початком процедури зупинки таймера.

2.Протягом наступних чотирьох тактів процесора необхідно записати в WDE логічний 0, при цьому робота сторожового таймера забороняється.

Таблиця 4.13 – Встановлення подільника сторожового таймера

WDP2	WDP1	WDP0	Період часу, мс	WDP2	WDP1	WDP0	Період часу, мс
0	0	0	16	1	0	0	256
0	0	1	32	1	0	1	512
0	1	0	64	1	1	0	1024
0	1	1	128	1	1	1	2048

4.8 Читання і записування в енергонезалежну пам’ять

Регістри доступу до енергонезалежної пам’яті (EEPROM) розташовані в просторі введення/виведення.

Час записуваня лежить у діапазоні 2,5...4 мс і залежить від напруги живлення. Якщо програма користувача робить запис в енергонезалежну пам’ять, повинні бути виконані деякі запобіжні заходи. При використанні в джерелі живлення конденсаторів великої ємності напруга живлення нарос-

тає і спадає досить повільно. Це приводить до того, що процесор деякий час працює при напрузі живлення нижче мінімуму, достатнього для нормальної роботи схем тактування. При цьому він може робити небажані переходи, потрапляючи на частини програми, що роблять запис у EEPROM. У таких випадках для захисту вмісту EEPROM необхідно використовувати зовнішні схеми, що формують сигнал скидання при зменшенні напруги живлення. При використанні зовнішнього супервайзера живлення (наприклад, К1171СП47 – для нього границя спрацьовування складає 4,7 В), що подає на вивід Reset напругу низького рівня при напрузі живлення нижче допустимого, практично можна бути впевненим у тому, що вміст пам'яті EEPROM не буде пошкоджуватися незалежно від ємності конденсатора на виході джерела живлення.

При записуванні чи читанні EEPROM процесор зупиняється на два машинних цикли до початку виконання наступної команди.

Біт	7	6	5	4	3	2	1	0	
$1E ($3E)	-	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEAR
Чит. / зап.	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Поч.знач.	0	0	0	0	0	0	0	0	

Рисунок 4.36 – Регістр адреси EEPROM – EEAR

Біт 7 – у AT90S2313 зарезервований і завжди читається як "0".

Біти 6...0 – EEAR6...0 – адреса EEPROM. Адресний регістр EEPROM задає адресу в 128-байтовому просторі EEPROM. Байти даних EEPROM адресуються лінійно в діапазоні 0... 127.

Біт	7	6	5	4	3	2	1	0	
$1D ($3D)	MSB							LSB	EEDR
Чит. / зап.	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Поч. знач.	0	0	0	0	0	0	0	0	

Рисунок 4.37 – Регістр даних EEPROM – EEDR

Біти 7...0 – EEDR7...0 – Дані EEPROM. При записуванні регістр EEDR містить дані, що повинні записуватися в EEPROM за адресою, що знаходиться в регістрі EEAR. Після операції читання в цей регістр записуються дані, прочитані з EEPROM за адресою, заданою в регістрі EEAR.

Біт	7	6	5	4	3	2	1	0	
$1C ($3C)	-	-	-	-	-	EEMWE	EEWE	EERE	EECR
Чит. / зап.	R	R	R	R	R	R/W	R/W	R/W	
Поч. знач.	0	0	0	0	0	0	0	0	

Рисунок 4.38 – Регістр керування EEPROM – EECR

Біти 7...3 – у AT90S2313 зарезервовані і завжди читаються як “0”.

Біт 2 – **EEMWE** – керування дозволом запису. Цей біт визначає, чи будуть записані дані при установленні EEWE. Якщо біт EEMWE установлений, при установленні EEWE дані записуються за обраною адресою EEPROM. Якщо цей біт скинутий, установлення EEWE не має ефекту. Після програмного установлення цей біт скидається апаратно через чотири такти процесора.

Біт 1 – **EEWE** – дозвіл запису в EEPROM. Сигнал EEWE є стробом записування в EEPROM. Після установлення правильної адреси і даних для записування в EEPROM необхідно установити біт EEWE. При записуванні “1” у біт EEWE повинний бути встановлений біт EEMWE, тоді відбувається записування у EEPROM. Для записування в EEPROM повинна дотримуватися така послідовність.

1. Чекати обнулення EEWE.
2. Записати адресу в EEAR.
3. Записати дані в EEDR.
4. Установити в 1 біт EEMWE.
5. Не пізніше ніж через 4 такти після установлення EEMWE установити EEWE.

Після того, як час запису мине (приблизно 2,5 мс для U_{cc} = 5 В і 4 мс для U_{cc} = 2,7 В), біт EEWE очищається апаратно. Програміст може перевіряти цей біт і очікувати його установлення в нуль перед тим, як записувати наступний байт. При установленні EEWE мікроконтролер зупиняється на два цикли перед виконанням наступної команди.

Біт 0 – **EERE** – дозвіл читання з EEPROM. Сигнал EERE є стробом читання з EEPROM. Після установлення потрібної адреси в регістрі EEAR повинний бути встановлений біт EERE. Після того як біт EERE буде апаратно очищений, викликані дані будуть знаходитися в регістрі EEDR. Читання EEPROM займає одну команду і не вимагає відстеження біта EERE. Після установлення біта EERE мікроконтролер зупиняється на 4 цикли перед тим, як буде виконана наступна команда. Перед читанням користувач повинний перевірити стан біта EEWE. Якщо змінювати вміст регістрів EEPROM під час операції записування в EEPROM, записування у комірку пам’яті переривається і результат операції записування стає визначеним.

4.9 Універсальний асинхронний приймач-передавач

Мікроконтролер AT90S2313 має вбудований універсальний асинхронний приймач-передавач (UART).

Основні характеристики UART:

- генерація довільних значень швидкості;
- висока швидкість навіть при низьких тактових частотах;
- 8 чи 9 бітів даних;
- фільтрація шуму;
- визначення переповнення;
- детектування помилки кадру;
- визначення неправильного стартового біта;
- три роздільних переривання – завершення передавання, очищення регістра передавання і завершення приймання.

Передача даних

Структурна схема вузла передавання даних UART показана на рис. 4.39.

Передавання даних ініціюється записом переданих даних у регістр введення/виведення даних UART – UDR. Дані пересилаються з UDR у регістр зсуву передавача, коли:

- новий символ записується в UDR після того, як був висунутий стоповий біт для попереднього символа. При цьому регістр зсуву завантажується відразу;
- новий символ записується в UDR до того, як висунуть стоповий біт для попереднього символа. При цьому регістр зсуву записується відразу після того, як буде висунутий стоповий біт попереднього символа.

При цьому в регістрі стану UART - USR встановлюється біт UDRE – ознака очищення регістра даних. Коли цей біт установлений, UART готовий до приймання наступного символа. При перезаписуванні UDR у 10(11) - розрядний регістр зсуву біт 0 регістра приймає значення нуль (стартовий біт), а біт 9 чи 10 установлюється (стоповий біт). Якщо обране 9-бітове слово даних (установлено біт CH9 у регістрі UCR), біт TXB8 з UCR переписується в 9-й біт регістра зсуву передавача.

Після тактового імпульсу, що випливає з частотою передавання, стартовий біт висувається на вивід TXD. Потім висуваються дані починаючи з молодшого біта. Після того як висунуть стоповий біт, у регістр зсуву завантажуються нові дані, якщо вони були записані в UDR під час передачі. При завантаженні встановлюється біт UDRE. Якщо до висування стопового біта в регістр UDR не надходять нові дані, UDRE залишається встановленим до наступного записування UDR. Якщо нові дані не надійшли і на виводі TXD з’являється стоповий біт, у регістрі USR установлюється прапорець і закінчення передавання — TXC.

Установлення біта TXEN у UCR дозволяє роботу передавача. При очищенні біта TXEN вивід PD1 можна використовувати для введен-

ня/виведення даних. Якщо біт TXEN установлений, передавач UART увімкнутий до виводу PD1 незалежно від установлення біта DDD1 у регістрі DDRD.

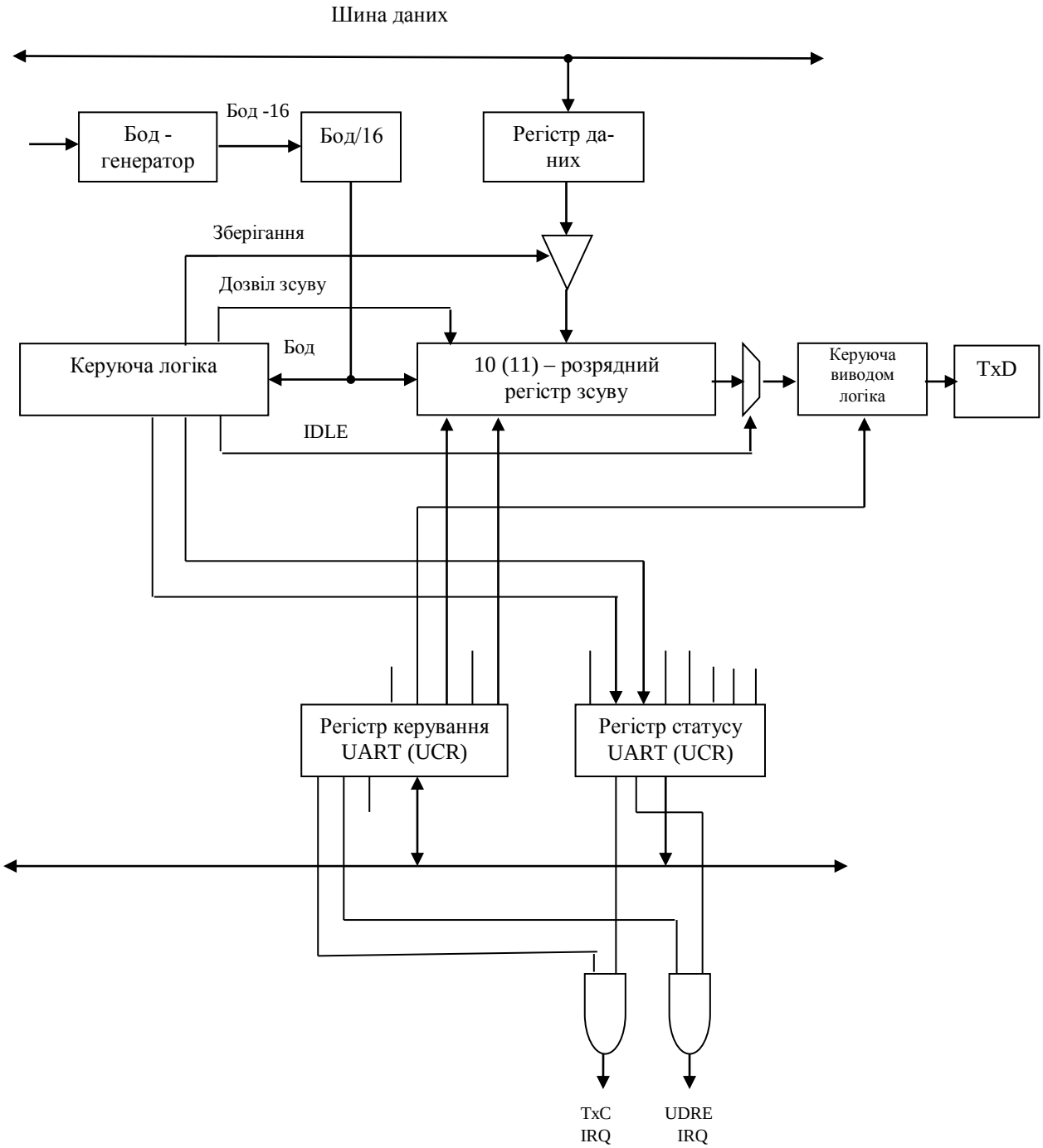


Рисунок 4.39 – Структурна схема вузла передавання даних UART

Приймання даних

Структурна схема вузла приймання даних UART показана на рис. 4.40.

Логічна схема приймача обробляє сигнал на виводі Rx з частотою в 16 разів більше швидкості передавання (для обробки одного біта прийнятої послідовності виробляється 16 вибірок вхідного сигналу). У стані очікування одна вибірка логічного нуля інтерпретується як спадний фронт стар-

тового біта після чого запускається послідовність виявлення стартового біта. Якщо в першій вибірці сигналу виявлений нульовий відлік, приймач обробляє 8, 9 і 10 вибірки сигналу на виводі RXD. Якщо хоча б дві з трьох вибірок дорівнюють логічній одиниці, стартовий біт вважається шумом і приймач чекає наступного переходу з “1” у “0”.

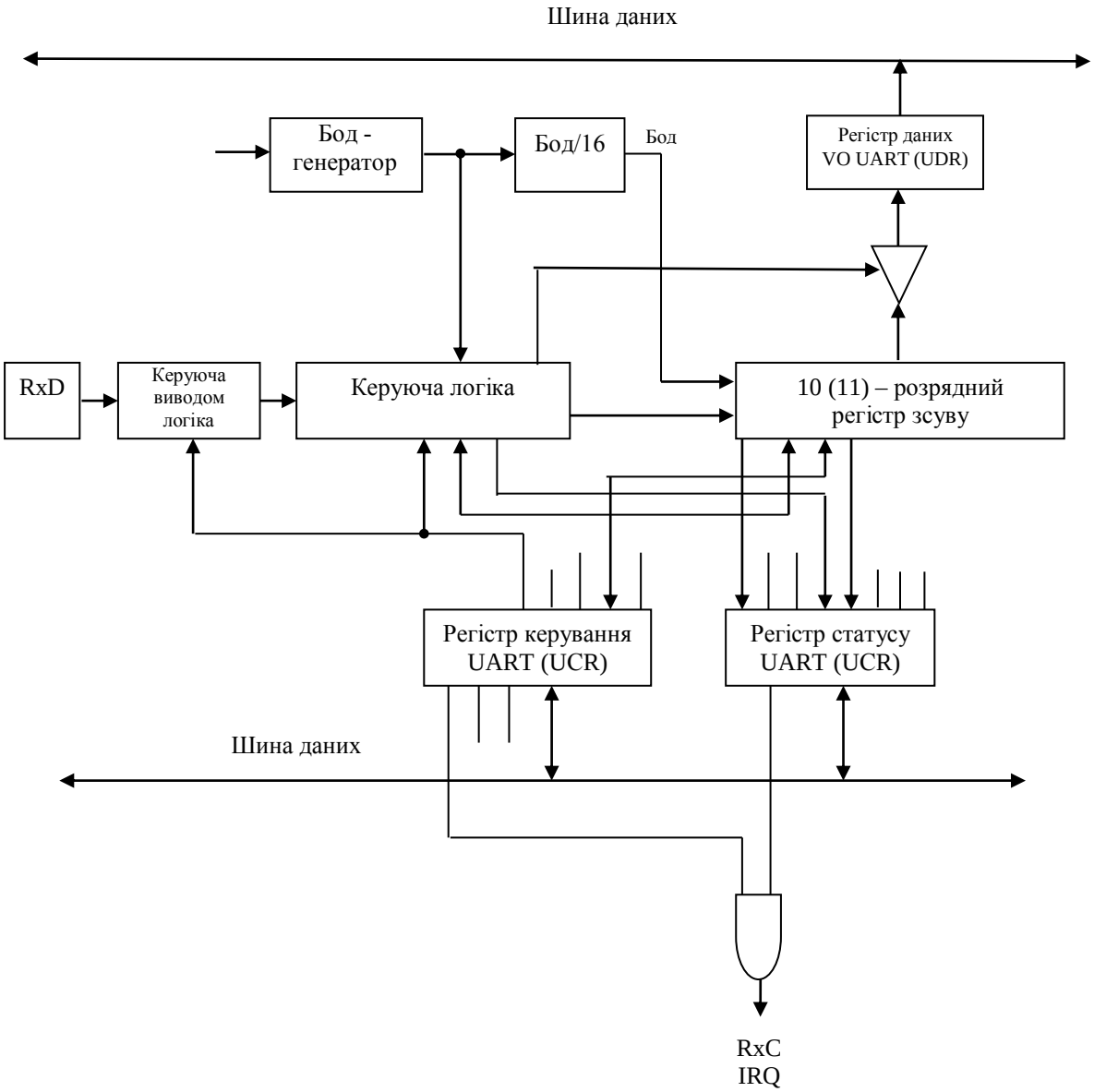


Рисунок 4.40 – Структурна схема вузла приймання даних UART

Якщо виявлений стартовий біт, починається обробка бітів даних. Рішення про рівень даних також виробляється за вісьмома, дев’ятьма і десятьма вибіркам вхідного сигналу, рівень вхідного сигналу визначається за рівністю двох вибірок. Після того як рівень даних визначений, дані зсуваються в регістр зсуву приймача.

Для визначення стопового біта хоча б дві з трьох вибірок вхідного сигналу повинні бути рівні 1. Якщо ця умова не виконується, у регістрі USR встановлюється прапорець помилки кадру FE. Перед читанням даних

з регістра UDR користувач повинний перевіряти біт FE для виявлення помилок кадру.

Незалежно від прийняття правильного стопового біта після закінчення приймання символа прийняті дані переписуються в UDR і встановлюється прапорець RXC у регістрі USR. Фізично регістр UDR складається з двох окремих регістрів, один використовується для передавання даних, інший — для приймання. При читанні UDR відбувається доступ до регістра приймача, при записуванні – до регістра передавача. При обміні 9-бітовими даними 9-й біт прийнятих даних записується в біт RXB8 регістра UCR.

Якщо при прийманні символа з регістра UDR не був прочитаний попередній символ, у регістрі UCR установлюється прапорець переповнення – OR. Установлення цього біта означає, що останній прийнятий байт даних не переписується із регістра зсуву в регістр UDR і буде загублений. Біт OR буферизований і обновляється при читанні правильних даних з UDR. Таким чином, користувач завжди може перевірити стан OR після читання UDR і знайти переповнення, що пройшли.

При скиданні біта RXEN у регістрі UCR приймання даних забороняється. При цьому вивід PD0 можна використовувати для введення/виведення загального призначення. При установленні RXEN приймач підімкнений до виводу PD0 незалежно від стану біта DDD0 у регістрі DDRD.

Керування UART

Біт	7	6	5	4	3	2	1	0	
$0C ($2C)	MSB							LSB	UDR
Чит. / зап.	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Поч. знач.	0	0	0	0	0	0	0	0	

Рисунок 4.41 – Регістр введення-виведення UART (UDR)

Регістр USR доступний тільки для читання, у ньому зберігається інформація про стан UART.

Біт	7	6	5	4	3	2	1	0	
$0B ($2B)	RXC	TXC	UDRE	FE	OR	-	-	-	USR
Чит. / зап.	R	R/W	R	R	R	R	R	R	
Поч. знач.	0	0	0	0	0	0	0	0	

Рисунок 4.42 – Регістр стану UART (USR)

Біт 7 – RXC – приймання завершено. Цей біт встановлюється в 1, коли прийнятий символ переписується із регістра зсуву приймача в регістр UDR. Біт установлюється незалежно від виявлення помилки кадру. Якщо встановлений біт RXC1E у регістрі UCR, при установленні біта виконується переривання після завершення приймання символа. RXC скидається при читанні UDR. При використанні приймання даних за перериванням оброблювач переривання повинний читати регістр UDR для скидання RXC, інакше при виході з переривання воно буде викликано знову.

Біт 6 – TXC – передача завершена. Цей біт встановлюється в "1", якщо символ із регістра зсуву передавача (включаючи стоповий біт) переданий, а в регістр UDR не були записані нові дані. Цей прапорець особливо корисний при півдуплексному зв'язку, коли передавальний пристрій повинний перейти в режим приймання і звільнити лінію зв'язку відразу після закінчення передачі. Якщо встановлений біт TXIE у регістрі UCR, при установленні TXC виконується переривання після закінчення передачі. TXC скидається апаратно при виконанні відповідного вектора переривання. Крім того, біт можна скинути, записавши в нього 1.

Біт 5 – UDRE – регістр даних UART порожній. Цей біт встановлюється в "1", коли дані, записані в UDR, переписуються в регістр зрушення передавача. Установлення цього біта означає, що передавач готовий прийняти наступний символ для передавання. Якщо встановлений біт UDRIE у регістрі UCR, при установленні цього біта виконується переривання закінчення передавання. Біт UDRE скидається при записуванні регістра UDR. При використанні передачі, керованої перериванням, підпрограма обслуговування переривання повинна записувати UDR, щоб скинути біт UDRE, інакше при виході з переривання воно буде викликано знову. При скиданні цей біт встановлюється в "1", щоб показати готовність передавача.

Біт 4 – FE – помилка кадру. Цей біт установлюється при виявленні умови помилки кадру, тобто якщо стоповий біт прийнятого байта дорівнює 0. Біт FE скидається при прийманні одиничного стопового біта.

Біт 3 – OR – переповнення. Цей біт установлюється при виявленні умови переповнення, тобто в тому випадку, коли символ з регістра UDR не був прочитаний до того, як заповнився регістр зсуву приймача. Цей біт буферизований, тобто залишається встановленим доти, поки з регістра UDR не будуть прочитані правильні дані. Біт OR скидається, коли прийняті дані переписуються в UDR.

Біти 2...0 – у AT90S2313 зарезервовані і завжди читаються як 0.

Біт	7	6	5	4	3	2	1	0	
$0A ($2A)	RXCIE	TXCIE	UDRIE	RXEN	TXEN	CHR9	RXB8	TXB8	UCR
Чит. / зап.	R/W	R/W	R/W	R/W	R/W	R/W	R	W	
Поч. знач.	0	0	0	0	0	0	0	0	

Рисунок 4.43 – Регістр керування UART (UCR)

Біт 7 – RXCIE – дозвіл переривання після закінчення приймання. Якщо RXCIE=1, установлення біта RXC у регістрі USR приводить до виконання переривання після закінчення приймання (за умови, що переривання дозволені).

Біт 6 – TXC IE – дозвіл переривання після закінчення передавання. Якщо цей біт встановлений, установлення біта TXC у USR приводить до виконання переривання після закінчення передавання.

Біт 5 – UDRIE – переривання після очищення регістра даних послідовного порту. Якщо цей біт встановлений, установлення біта UDRE у USR приводить до виконання переривання після очищення регістра даних UART.

Біт 4 – RXEN – дозвіл приймача. При установленні цього біта дозволяється робота приймача UART. Якщо приймач вимкнутий, прапорці TXC, OR і FE не встановлюються. Якщо ці прапорці встановлені, скидання RXEN не очищає їх.

Біт 3 – TXEN – дозвіл передавача. При установленні цього біта дозволяється робота передавача UART. При забороні роботи передавача під час передачі символа він продовжує працювати, поки не буде очищено регістр зсуву і не буде переданий символ, поміщений у UDR.

Біт 2 – CHR9 – 9-бітові посилки. Якщо цей біт встановлений, прийняті і передані символи мають довжину 9 бітів. Для передавання і приймання 9-го символа використовуються біти RXB8 і TXB8 відповідно. 9-й біт можна використовувати як додатковий стоповий біт чи як ознака парності.

Біт 1 – RXB8 – 8 бітів прийнятих даних. Якщо встановлений біт CHR9, сюди записується 9-й біт прийнятих даних.

Біт 0 – TXB8 – 8 бітів переданих даних. Якщо встановлений біт CHR9, звідси береться 9-й біт переданих даних.

Генератор швидкості передавання – це подільник частоти, що генерує швидкості відповідно до нижчеподаного виразу:

$$BAUD = Fck / (16 \times (UBRR+1))$$

Тут: *BAUD* – швидкість передавання (бод); *Fck* – частота тактового генератора процесора; *UBRR* – вміст регістра швидкості передавання UART.

У табл. 4.14 приведені значення регістра UBRR та процентне відхи-

лення від стандартної швидкості передавання для стандартних частот кварцових генераторів.

Таблиця 4.14 – Значення регістра UBRR

Швид-кість	**1 МГц**	**Помил-ки, %**	**1.8432 МГц**	**Помил-ки, %**	**2 МГц**	**Помил-ки, %**	**2.4576 МГц**	**Помил-ки, %**
2400	UBRR=25	0.2	UBRR=47	0.0	UBRR=51	0.2	UBRR=63	0.0
4800	UBRR=12	0.2	UBRR=23	0.0	UBRR=25	0.2	UBRR=31	0.0
9600	UBRR=6	7.5	UBRR=11	0.0	UBRR=12	0.2	UBRR=15	0.0
14400	UBRR=3	7.8	UBRR=7	0.0	UBRR=8	3.7	UBRR=10	3.1
19200	UBRR=2	7.8	UBRR=5	0.0	UBRR=6	7.5	UBRR=7	0.0
28800	UBRR=1	7.8	UBRR=3	0.0	UBRR=3	7.8	UBRR=4	6.3
38400	UBRR=1	22.9	UBRR=2	0.0	UBRR=2	7.8	UBRR=3	0.0
57600	UBRR=0	7.8	UBRR=1	0.0	UBRR=1	7.8	UBRR=2	12.5
76800	UBRR=0	22.9	UBRR=1	33.3	UBRR=1	22.9	UBRR=1	0.0
115200	UBRR=0	84.3	UBRR=0	0.0	UBRR=0	7.8	UBRR=0	25.0

Швид-кість	**3.2768 МГц**	**Помил-ки,%**	**3.6864 МГц**	**Помил-ки, %**	**4 МГц**	**Помил-ки, %**	**4.608 МГц**	**Помил-ки, %**
2400	UBRR=84	0.4	UBRR=95	0.0	UBRR=103	0.2	UBRR=119	0.0
4800	UBRR=42	0.8	UBRR=47	0.0	UBRR=51	0.2	UBRR=59	0.0
9600	UBRR=20	1.6	UBRR=23	0.0	UBRR=25	0.2	UBRR=29	0.0
14400	UBRR=13	1.6	UBRR=15	0.0	UBRR=16	2.1	UBRR=19	0.0
19200	UBRR=10	3.1	UBRR=11	0.0	UBRR=12	0.2	UBRR=14	0.0
28800	UBRR=6	1.6	UBRR=7	0.0	UBRR=8	3.7	UBRR=9	0.0
38400	UBRR=4	6.3	UBRR=5	0.0	UBRR=6	7.5	UBRR=7	6.7
57600	UBRR=3	12.5	UBRR=3	0.0	UBRR=3	7.8	UBRR=4	0.0
76800	UBRR=2	12.5	UBRR=2	0.0	UBRR=2	7.8	UBRR=3	6.7
115200	UBRR=1	12.5	UBRR=1	0.0	UBRR=1	7.8	UBRR=2	20.0
Швид-кість	**7.3728 МГц**	**Помил-ки,%**	**8 МГц**	**Помил-ки, %**	**9.216 МГц**	**Помил-ки,%**	**11.059 МГц**	**Помил-ки, %**
2400	UBRR=191	0.0	UBRR= 207	0.2	UBRR=239	0.0	UBRR=287	0.0
4800	UBRR=95	0.0	UBRR= 103	0.2	UBRR=119	0.0	UBRR=143	0.0
9600	UBRR=47	0.0	UBRR=51	0.2	UBRR=59	0.0	UBRR=71	0.0
14400	UBRR=31	0.0	UBRR=34	0.8	UBRR=39	0.0	UBRR=47	0.0
19200	UBRR=23	0.0	UBRR=25	0.2	UBRR=29	0.0	UBRR=35	0.0
28800	UBRR=15	0.0	UBRR=16	2.1	UBRR=19	0.0	UBRR=23	0.0
38400	UBRR=11	0.0	UBRR=12	0.2	UBRR=14	0.0	UBRR=17	0.0
57600	UBRR=7	0.0	UBRR=8	3.7	UBRR=9	0.0	UBRR=11	0.0

Біт	7	6	5	4	3	2	1	0	
$09 ($29)	RXCIE	TXCIE	UDRIE	RXEN	TXEN	CHR9	RXB8	TXB8	UBRR
Чит. / зап.	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R /W	
Поч. знач.	0	0	0	0	0	0	0	0	

Рисунок 4.44 – Регістр швидкості передавання (UBRR)

Це 8-розрядний регістр, що задає швидкість передавання послідовного порту відповідно до виразу, що наведено вище.

4.10 Аналоговий компаратор

Структурна схема аналогового компаратора зображена на рис. 4.45.

Аналоговий компаратор порівнює вхідну напруга на позитивному вході PB0(AIN0) і негативному вході PB1(AIN1). Коли напруга на позитивному вході більша напруги на негативному, встановлюється біт ACO (Analog Comparator Output). Вихід аналогового компаратора можна установити на роботу з функцією захоплення таймера/лічильника 1. Крім того, компаратор може викликати своє переривання. Користувач може установити спрацьовування переривання за наростаючим чи спадним фронтом чи за перемиканням.

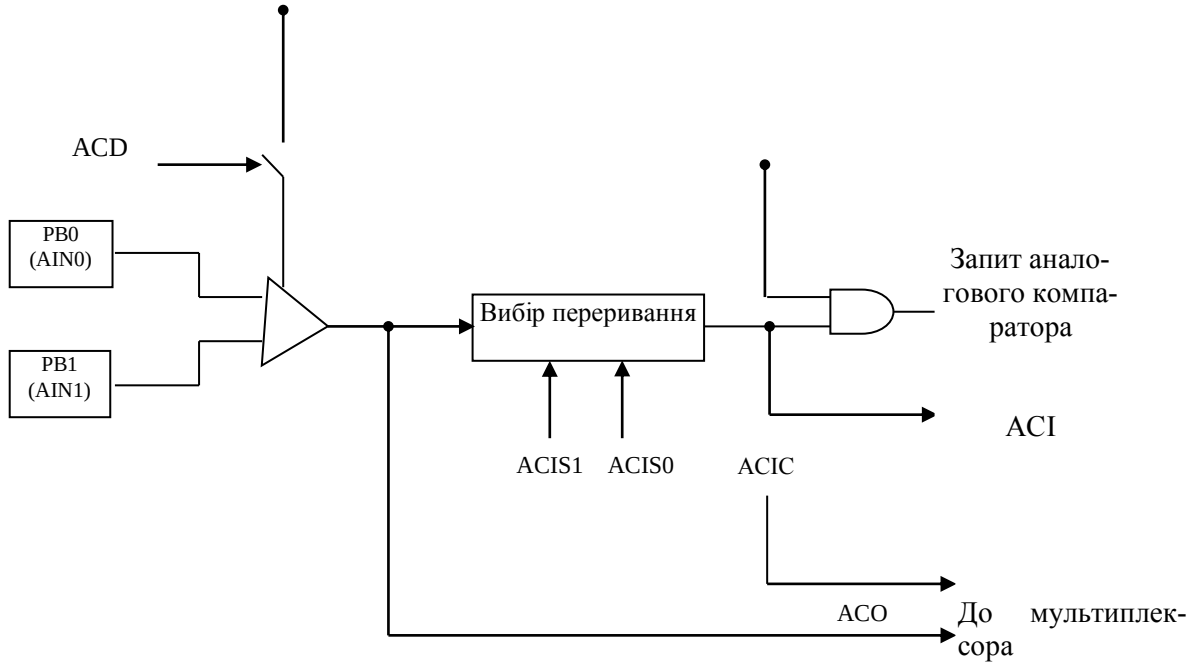


Рисунок 4.45 – Структурна схема аналогового компаратора

Біт	7	6	5	4	3	2	1	0	
$08 ($28)	ACD	-	AC0	AC1	ACIE	ACIC	ACIS1	ACIS0	UBRR
Чит. / зап.	R/W	R	R	R/W	R/W	R/W	R/W	R /W	
Поч. знач.	0	0	N/A	0	0	0	0	0	

Рисунок 4.46 – Регістр керування і стану аналогового компаратора (ACSR)

Біт 7 – **ACD** – заборона аналогового компаратора. Коли цей біт встановлений, живлення від аналогового компаратора вимикається. Для вимкнення компаратора цей біт можна встановити в будь-який час. Звичайно, ця властивість використовується, якщо критичне споживання процесора в холостому режимі і відновлення роботи процесора від аналогового компаратора не потрібно. При зміні біта ACD переривання від аналогового компаратора повинні бути заборонені скиданням ACIE у регістрі ACSR. У противному випадку переривання може відбутися під час зміни біта.

Біт 6 – у AT90S2313 зарезервований і завжди читається як “0”.

Біт 5 – **ACO** – вихід аналогового компаратора. Біт ACO безпосередньо «увімкнутий» до виходу аналогового компаратора.

Біт 4 – **ACI** – прапорець переривання від аналогового компаратора. Цей біт встановлюється, коли перемикання виходу компаратора збігається з режимом переривання встановленим бітами ACIS1 і ACIS0. Програма обробки переривання від аналогового компаратора виконується, якщо встановлений біт AC1E в “1” і встановлений у “1” біт I у регістрі стану. ACI скидається апаратно при виконанні відповідного вектора переривання. Інший спосіб очистити AC1 — записати в прапорець логічну одиницю.

Біт 3 – **ACIE** – дозвіл переривання від аналогового компаратора. Коли встановлений цей біт і біт I регістра стану, переривання від аналогового компаратора відпрацьовуються. Якщо біт ACIE = 0 переривання заборонені.

Біт 2 – **ACIC** – захоплення за виходом аналогового компаратора. Якщо цей біт установлений, функція захоплення таймера/лічильника 1 керується виходом аналогового компаратора. При цьому вихід компаратора вмикається безпосередньо до схеми обробки захоплення, надаючи зручні засоби придушення шуму і вибору фронту, передбачені перериванням захоплення за входом. Коли біт очищений, схема захоплення і компаратор роз’єднані. Щоб компаратор міг керувати функцією захоплення таймера/лічильника 1, повинний бути встановлений біт TICIE1 у регістрі TIMCK.

Біти 1,0 – **ACIS1, ACIS0** – вибір режиму переривання аналогового компаратора. Різні установлення приведені в табл. 4.15.

Таблиця 4.15 – Установлення ACIS1/ACIS0

ACIS1	ACIS0	Опис
0	0	Переривання від компаратора за переключенням виходу
0	1	Зарезервовано
1	0	Переривання від компаратора за спадним фронтом виходу
1	1	Переривання від компаратора за наростаючим фронтом виходу
Примітка. При зміні бітів ACIS1/ACIS0 переривання від аналогового компаратора повинні бути заборонені скиданням біта дозволу переривання в регістрі ACSR. Інакше переривання може відбутися в процесі зміни цих бітів.		

4.11 Порти введення-виведення

Порт B 8-розрядний — двонаправлений для введення-виведення.

Для обслуговування порту відведено три регістри: регістр даних PORTB ($18, $38), регістр напрямку даних — DDRB ($17, $37) та виводи порту B ($16, $36). Адреса виводів порту B призначена тільки для читання, у той час як регістр даних і регістр напрямку даних — для читання і записування.

Усі виводи порту мають обмежувальні резистори, що вмикаються окремо. Виходи порту B можуть споживати струм до 20 мА і безпосередньо керувати світлодіодними індикаторами. Якщо виводи PB0...PB7 використовуються як входи і замикаються на землю, то при увімкнутих внутрішніх обмежувальних резисторах, виводи є джерелами струму. Додаткові функції виводів порту B приведені в табл. 4.16.

Таблиця 4.16 – Альтернативні функції виводів порту B

Вивід	Альтернативна функція
PB0	AIN0(позитивний вхід аналогового компаратора)
PB1	AIN1(негативний вхід аналогового компаратора)
PB3	OC1(вихід збігу таймера/лічильника 1)
PB5	MOS1(вхід даних для SPI)
PB6	MISO(вихід даних для SPI)
PB7	SCK(вхід тактових імпульсів SPI)

При використанні альтернативних функцій виводів регістри DDRB і PORTB повинні бути установлені відповідно до опису альтернативних функцій. При використанні можливості внутрішньосхемного програмування мікроконтролера варто враховувати, що програматор використовує для своєї роботи лінії MOS1, MIS0 і SCK. Відповідно, пристрої, що підімкнені до цих ліній, не повинні заважати роботі програматора.

Біт	7	6	5	4	3	2	1	0	
$17 ($37)	ACD	-	AC0	AC1	ACIE	ACIC	ACIS1	ACIS0	DDRB
Чит. / зап.	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R /W	
Поч. знач.	0	0	0	0	0	0	0	0	

Рисунок 4.48 – Регістр напрямку даних порту B – DDRB

Біт	7	6	5	4	3	2	1	0	
$16 ($36)	ACD	-	AC0	AC1	ACIE	ACIC	ACIS1	ACIS0	PINB
Чт. / зап.	R	R	R	R	R	R	R	R	
Поч. знач.	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Рисунок 4.49 – Виводи порту B — PINB

PINB не є регістром, за цією адресою здійснюється доступ до фізичних значень кожного з виводів порту B. При читанні PORTB читаються дані з регістра, при читанні PINB читаються логічні значення, що відповідають фактичному стану виводів порту.

Порт B як порт введення-виведення загального призначення

Усі 8 бітів порту B при використанні для введення/виведення однакові.

Біт DDBn регістра DDRB вибирає напрямок передавання даних. Якщо біт встановлений (тобто дорівнює одиниці), вивід має конфігурацію як вихід. Якщо біт скинутий (тобто дорівнює нулю) – вивід має конфігурацію як вхід. Якщо PORTBn встановлений і вивід має конфігурацію як вхід, вмикається КМОН і обмежувальний резистор. Для вимкнення резистора PORTBn повинний бути скинутий вивід чи повинний мати конфігурацію як вихід.

Таблиця 4.17 – Вплив DDBn на виводи порту B

DDBn	PORTB	Вхід/вихід	Резистор	Коментар
0	0	Вхід	Немає	Третій стан(Hi-Z)
0	1	Вхід	Є	PBn джерело струму, якщо з’єднаний із землею
1	0	Вихід	Немає	Вихід встановлений у 0
1	1	Вихід	Немає	Вихід встановлений у 1
Примітка. n = 7,6..0 – номер виводу.				

Альтернативні функції PORTB:

Біт 7 SCK – вхід тактової частоти для SPI.

Біт 6 MISO – вихід даних для SPI.

Біт 5 MOSI – вхід даних для SPI.

Біт 3 OC1 – вихід збігу. Цей вивід може мати конфігурацію для зов-

нішнього виводу події збігу таймера 1. Для цього біт DDB3 повинний бути встановлений у 1 (вивід має конфігурацію як вихід).

Біт 1 AIN1 – негативний вхід аналогового компаратора. Якщо цей вивід має конфігурацію як вхід (DDB1=0), і вимкнутий внутрішній обмежувальний резистор, цей вивід працює як негативний вхід внутрішнього аналогового компаратора.

Біт 0 AIN0 – позитивний вхід аналогового компаратора. Якщо цей вивід має конфігурацію як вхід (DDB0=0) і вимкнутий внутрішній резистор, цей вивід працює як позитивний вхід внутрішнього аналогового компаратора.

Порт D

Для порту D зарезервовані 3 комірки пам'яті: регістр PORTD $12 ($32), регістр напрямку даних – DDRD $11($31) і виводи порту D – PIND $10 ($30). Регістри даних і напрямку даних можуть читатися/записуватися, PIND – тільки для читання.

Порт D – 7-розрядний двонаправлений з вбудованими обмежувальними резисторами. Вихідні буфери споживають струм до 20 мА. Якщо виводи використовуються як входи і на них поданий низький рівень, то при підімкненні резисторів, вони є джерелами струму. Деякі з виводів порту мають альтернативні функції, показані в табл. 4.18.

Якщо виводи порту використовуються для обслуговування альтернативних функцій, вони повинні мати конфігурацію на введення-виведення відповідно до опису функції.

Таблиця 4.18 Альтернативні функції порту D

Вивід	**Альтернативна функція**
PD0	RXD(вхід даних UART)
PD1	TXD(вихід даних UART)
PD2	INT0(вхід зовнішнього переривання 0)
PD3	INT1(вхід зовнішнього переривання 1)
PD4	T0(зовнішній вхід таймера/лічильника 0)
PD5	T1(зовнішній вхід таймера/лічильника 1)
PD6	ICP(вхід захоплення таймера/лічильника 0)

Біт	7	6	5	4	3	2	1	0	
$12 ($32)	-	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
Чит. / зап.	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Поч. знач.	0	0	0	0	0	0	0	0	

Рисунок 4.50 – Регістр даних порту D – PORTD

Рисунок 4.51 – Регістр напрямку порту D – DDRD

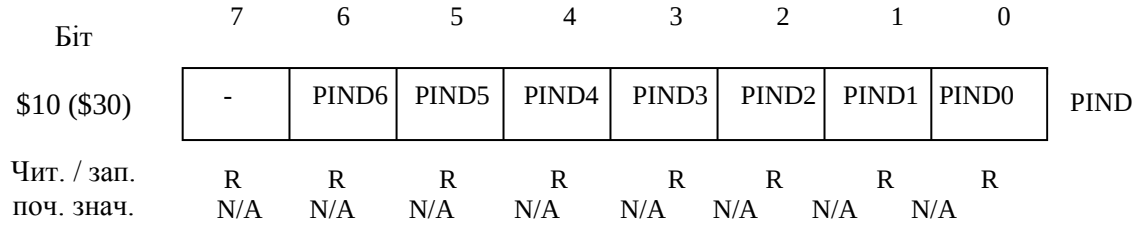


Рисунок 4.52 – Виводи порту B – PINB

PIND не є регістром, за цією адресою здійснюється доступ до фізичних значень кожного з виводів порту D. При читанні PORTD читаються дані з регістра, при читанні PIND читаються логічні значення, що відповідають фактичному стану виводів порту.

Порт D як порт введення-виведення загального призначення

Біт DDDn регістра DDRD вибирає напрямок передавання даних. Якщо біт встановлений, вивід має конфігурацію як вихід. Якщо біт скинутий — вивід має конфігурацію як вхід. Якщо PORTDn встановлений і вивід має конфігурацію як вхід. Для вимкнення резистора PORTDn повинний бути скинутий або вивід повинний мати конфігурацію як вихід.

Таблиця 4.19 – Вплив DDDn на виводи порту D

DDDn	**PORTDn**	**Вхід/вихід**	**Резистор**	**Коментар**
0	0	Вхід	Немає	Третій стан(Hi-Z)
0	1	Вхід	Так	PBn джерело струму, якщо з’єднаний із землею
1	0	Вихід	Немає	Вихід встановлений у 0
1	1	Вихід	Так	Вихід встановлений у 1
Примітка. n = 6..0 – номер виводу.				

Альтернативні функції порту D

Біт 6 ICP – вхід захоплення таймера/лічильника 1.

Біт 5 T1 – тактовий вхід таймера/лічильника 1.

Біт 4 T0 – тактовий вхід таймера/лічильника 0.

Біт 3 INT1 – вхід зовнішніх переривань 1.

Біт 2 INT0 – вхід зовнішніх переривань 0.

Біт 1 TXD – вихід передавача UART. Якщо дозволена робота передавача UART, незалежно від стану DDRD1 цей вивід має конфігурацію як вихід.

Біт 0 RXD – вихід приймача UART. Якщо дозволена робота приймача UART, незалежно від стану DDRD0 цей вивід має конфігурацію як вихід. Коли UART використовує вивід для приймання даних, одиниця в PORTD0 вмикає вбудований обмежувальний резистор.

4.12 Програмування пам'яті

Програмування бітів блокування пам'яті

Мікроконтролер AT90S2313 має два біти блокування, що можуть бути залишені незапрограмованими чи перепрограмуватися, при цьому досягаються властивості, приведені і табл. 4.20.

Таблиця 4.20 – Біти блокування

Режим	Біти блокування		Тип захисту
	LB1	LB2	
1	1	1	Захист не встановлений
2	0	1	Подальше програмування флеш - пам'яті і EEPROM заборонене
3	0	0	Як режим 2, але заборонене і читання
Примітка. Біти блокування стираються тільки при повному стиранні пам'яті.			

Біти конфігурації (Fuse bits)

У AT90S2313 передбачено два біти конфігурації — SPIEN і FSTRT. Коли біт SPIEN = 0, дозволений режим послідовного програмування. За замовчуванням біт SPIEN = 0.

Коли біт FSTRT = 0, використовується скорочений час запуску. За замовчуванням цей біт FSTRT = 1. Можна замовляти мікросхеми з попередньо запрограмованим бітом. Ці біти недоступні при послідовному програмуванні і не змінюються при стиранні пам'яті.

Код пристрою

Усі мікроконтролери фірми Atmel мають 3-байтовий сигнатурний код, за яким ідентифікується пристрій. Цей код можна прочитати в рівнобіжному і послідовному режимах. Це три байти, розміщені в окремому адресному просторі, і для AT90S2313 мають такі значення.

1.$000: $1E — код виробника — Atmel.

2.$001: $91 —2 Кбайт флеш-пам'яті.

3.$002: $01 — при $01=$91 — мікроконтролер AT90S2313.

Якщо запрограмовані біти блокування, байти сигнатури в послідовному режимі не читаються.

4.13 Параметри мікроконтролера AT90S2313

Для правильного застосування будь-якої мікросхеми, а особливо такої складної як мікроконтролер, має сенс при прийнятті рішення про той чи інший варіант схеми уточнити деякі довідкові дані. Щоб у читачів була

така можливість, наведемо тут докладні технічні дані на мікроконтролер AT90S2313. Параметри інших мікроконтролерів цієї серії дуже близькі до зазначеного.

Таблиця 4.21 – Максимально допустимі параметри

Робоча температура	від –55 до +125 ºС
Температура збереження	від –65 до +150 ºС
Напруга на будь-якому виході, крім RESET	від –1,0 до +7,0 В
Максимальна робоча напруга	6,6 В
Постійний струм через вивід порту	40,0 мА
Постійний струм між U_{cc} і GND	140,0 мА
Примітка. Вихід параметрів за межі, вказані в таблиці, може привести до порушення роботоздатності мікросхеми. Це граничні значення параметрів, робочі параметри мікросхеми приведені нижче. Утримання граничних значень на виводах мікроконтролера протягом тривалого часу може привести до втрати його роботоздатності.	

Таблиця 4.22 – Характеристики за постійним струмом

Позн.	**Параметр**	**Умови**	**Мін.**	**Тип.**	**Макс.**	**Од.**
1	2	3	4	5	6	7
VIL	Вхідна напруга низького рівня	(Крім XTAL1)	-0,5	-	$0,3U_{cc}$ (1)	В
vil1	Вхідна напруга низького рівня	(XTAL1)	-0,5	-	$0,3\ U_{cc}$ (1)	В
vih	Вхідна напруга високого рівня	(Крім XTAL1, RESET)	$0,6\ U_{cc}$ (2)	-	U_{cc} +0,5	В
VIH1	Вхідна напруга високого рівня	(XTAL1)	$0,7\ U_{cc}$ (2)	-	U_{cc} +0,5	В
VIH2	Вхідна напруга високого рівня	(RESET)	$0,85\ U_{cc}$ (2)	-	U_{cc} +0,5	В
vol	Вихідна напруга низького рівня (порти B, D)	I_{ol}=20mA, U_{cc} =5В I_{ol} =10mA, U_{cc} =3В	-	-	0,6 0,5	В
voh	Вихідна напруга високого рівня (порти B, D)	I_{oh}= -3mA, U_{cc} =5В I_{oh} = -1,5mA, U_{cc} =3В	4,3 2,3	-	-	В
IIL	Вхідний струм витоку виводу I/O	U_{cc} =6В, низький рівень	-	-	1,5	мкА
IIH	Вхідний струм витоку виводу I/O	Високий рівень	-	-	980,0	нА
RRST	Обмежувальний резистор, RESET		100,0	-	500,0	кОм
ri/o	Обмежувальний резистори ліній, введення/виведення		35,0	-	120,0	кОм
ICC	Струм споживання від джерела живлення	Нормальний режим U_{cc} = 3В, 4МГц	-	-	3,0	мА
		Idle режим U_{cc} = 3В,4МГЦ			1,0	мА

Продовження таблиці 4.22

1	2	3	4	5	6	7
ICC	Power-down Режим	Сторожовий таймер включений, U_{cc}=3В	-	9,0	15,0	мкА
		Сторожовий таймер відключений, U_{cc} =3В	-	<1,0	2,0	мкА
VACIO	Напруга зсуву входу аналогового компаратора	U_{cc} =5В U_{in} = U_{cc} /2	-	-	40,0	мВ
IACLK	Витік за входом аналогового компаратора	U_{cc} =5В U_{in}= U_{cc} /2	-50,0	-	50,0	нА
tACPD	Затримка аналогового компаратора	U_{cc} =2,7В U_{cc} =4,0В	-	750,0 500,0	-	нс

Примітки:
1. У робочому стані струм через виводи повинний обмежуватися такими умовами:
   - максимальний струм через вивід – 20 мА – при стані 0!
   - максимальний струм через усі виводи – 80 мА.
2. Мінімальна напруга для режиму пониженого споживання – 2В.

Таблиця 4.23 – Параметри зовнішнього тактового сигналу

Параметр	**U_{cc}=2,7..6,0 В**		**U_{cc}=4,0..6,0 В**		**Одиниці вимірювань**
	min	max	min	max	
Частота кварцу	0	4	0	10МГц	
Період тактової частоти	250		100		нс
Тривалість «1»	100		40		нс
Тривалість «0»	100		40		нс
Тривалість фронту		1,6		0,5	
Тривалість спаду		1,6		0,5	мкс

Таблиця 4.24 – Варіанти виконання мікроконтролера

Частота	**Напруга живлення**	**Маркування**	**Корпус**	**Діапазон температур**
4МГц	2,7..6,0 В	AT90S2313-4PC	PDIP	Комерційний (0..70 ºС)
		AT90S2313-4SC	SOIC	
		AT90S2313-4PI	PDIP	Промисловий (-40..85 ºС)
		AT90S2313-4SI	SOIC	
10МГц	4,0..6,0 В	AT90S2313-10PC	PDIP	Комерційний (0..70 ºС)
		AT90S2313-10SC	SOIC	
		AT90S2313-10PI	PDIP	Промисловий (-40..85 ºС)
		AT90S2313-10SI	SOIC	

5 ОСОБЛИВОСТІ ВИКОРИСТАННЯ МІКРОКОНТРОЛЕРІВ СІМЕЙСТВА AVR

Яка найпростіша працююча схема на мікроконтролері? Мікроконтролер, джерело живлення, елементи, що забезпечують роботу тактового генератора (рис. 5.1).

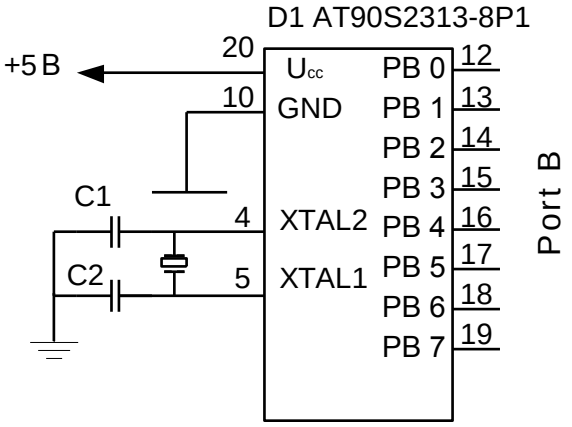


Рисунок 5.1 – Найпростіша схема на мікроконтролері

5.1 Джерело живлення

Джерело живлення схеми на мікроконтролерах — дуже важливий вузол. Жодна схема не може працювати без джерела живлення. На вибір типу джерела живлення впливає безліч факторів, в залежності від створюваної конструкції. У першу чергу, живлення пристрою може бути від батарей чи від мережі змінного струму 220 В.

У деяких випадках навіть тоді, коли поруч є мережа змінного струму, має сенс застосовувати батарейне живлення — якщо споживання пристрою дуже мале і батарей вистачить на тривалий термін роботи. Це дозволяє зробити схему простіше і легше (не потрібний мережний трансформатор і т.д.).

Батарейне живлення

Випускається багато видів батарей. Вони бувають різних розмірів. При виборі батареї для живлення своєї схеми варто керуватися такими параметрами.

1. Ємність батареї — це дуже важливий параметр, вимірюється він у мА/год чи А/год (міліампер-година чи ампер-година). Він визначає тривалість роботи конструкції без заміни батарей. Чим ємність більша, тим довше буде працювати схема. Звичайно, чим більша ємність батареї, тим більші її розміри і маса.
2. Напруга батареї.
3. Термін зберігання батареї.

4. Робоча температура батареї. Особливо важливо враховувати вплив температури в тому випадку, якщо конструкція повинна працювати при низьких чи високих температурах (наприклад, на вулиці).

5. Розміри, форма і вага батареї.

Живлення від мережі

Для живлення від мережі можна застосувати звичайні стабілізатори напруги, наприклад, КР142ЕН5А (аналог закордонної мікросхеми 7805). Ця мікросхема дозволяє одержати стабілізовану напругу величиною 5 В при струмі приблизно до 1 А. Якщо такий великий струм не потрібно, можна застосувати стабілізатор типу 05, що дозволяє одержати струм до 100 мА. Взагалі стабілізатори можуть бути різними, головне, щоб вони забезпечували потрібну напругу і величину струму для роботи схеми.

Перспективний варіант побудови мережних джерел живлення — імпульсні перетворювачі напруги. Вони дозволяють відчутно знизити масу джерела живлення і збільшити його ККД. Але це окрема велика тема, і в цій книзі вона розглядатися не буде.

Живлення від ліній портів

У деяких випадках, якщо схема споживає дуже маленький струм, можливе живлення її від інформаційних ліній портів. Наприклад, якщо пристрій вмикається до порту LPT1 і використовує тільки 3 лінії, інші лінії, що працюють на вивід, можуть використовуватися як джерело живлення (природно, для цього на них потрібно вивести значення 1).

Аналогічно можна використовувати лінію RTS порту RS-232. Для її використання варто зібрати найпростіший параметричний стабілізатор напруги на стабілітроні. Паралельно стабілітрону варто приєднати конденсатор ємністю не менш 10 мкФ.

Увага! Принциповий момент – незалежно від типу джерела живлення якнайближче до виводів живлення мікроконтролера варто приєднати керамічний (ще краще – танталовий) конденсатор ємністю 0,01...0,1 мкФ – для подавлення завад по ланцюгу живлення. До речі, такий конденсатор варто ставити поруч з кожною цифровою мікросхемою.

5.2 Зовнішні елементи тактового генератора

Використання кварцового резонатора

Використання кварцового резонатора найбільш розповсюджений спосіб увімкнення зовнішньої схеми тактового генератора. Ця схема вимагає додатково два конденсатори ємністю від 22 до 33 пФ, щоб полегшити запуск тактового генератора.

На рис. 5.2 показана схема підімкнення кварцового резонатора до мікроконтролера.

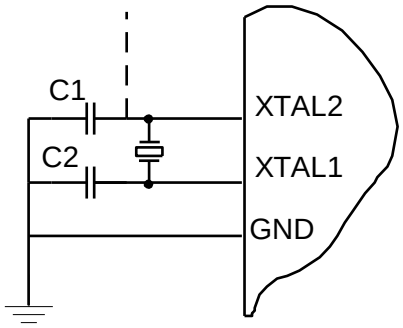


Рисунок 5.2 – Підімкнення кварцового резонатора до мікроконтролера

Використання вбудованого RC-генератора

У деяких типах мікроконтролерів AVR є вбудований RC-тактовий генератор (наприклад, AT90S1200, AT90S2343, Tiny22). Його увімкненням керує біт RCEN. У мікроконтролері AT90S1200 вбудований RC-генератор вимкнений, його можна увімкнути тільки користуючись паралельним програматором. Є варіант мікроконтролера AT90S1200A, у якому заздалегідь увімкнутий вбудований RC-генератор. Мікроконтролери AT90S2343 і Tiny22 поставляються з увімкнутим RC-генератором.

Варто враховувати, що стабільність частоти RC-генератора набагато нижча, ніж кварцового, тому не слід застосовувати його у випадках, коли потрібно точно відміряти інтервали часу.

5.3 Коло скидання

Використовується для затримки запуску мікроконтролера при увімкненні живлення, що потрібно для його правильного запуску, а також для ручного перезапуску мікроконтролера натисканням на кнопку. Найпростіша схема кола скидання показана на рис. 5.3. Якщо в схемі використовується вбудована електрична перепрограмована пам’ять даних мікроконтролера, варто використовувати зовнішню схему, що утримує вивід Reset на низькому рівні до тих пір, поки напруга не підніметься до необхідного рівня для мінімальної роботи мікроконтролера. Ця схема повинна працювати як при збільшенні напруги (тобто рівень на виводі Reset стане високим тільки після досягнення напругою живлення потрібного рівня), так і при зниженні напруги живлення нижче допустимого рівня. Така схема, звичайно, називається супервайзер живлення. На рис. 5.4 показана схема одного з варіантів підімкнення супервайзера живлення.

5.4 Основні типи інтерфейсів мікроконтролерів

Існує ряд інтерфейсів, які найчастіше використовуються з мікроконтролерами. Даний розділ присвячений знайомству з цими пристроями й опису основ їхнього застосування.

Подавлення деренчання контактів

При замиканні і розмиканні перемикачів у колі виникають імпульсні перешкоди (які називаються «шум» чи «дзвін»), викликані деренчанням контактів. Це явище часто виникає в системах на базі мікроконтролерів де використовується клавіатура, і «деренчання» може сприйматися як багаторазове натискання клавіші (рис. 5.5). Деренчання виникає при установленні і розриві контакту шляхом натискання на клавішу.

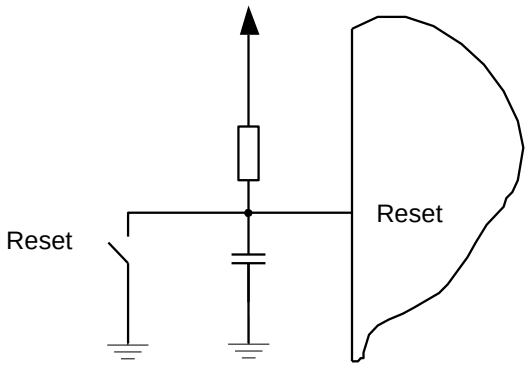


Рисунок 5.3 – Найпростіше коло скиду

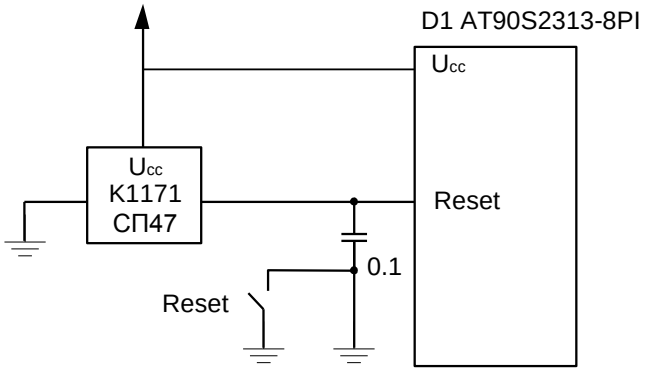


Рисунок 5.4 – Схема підімкнення супервайзера живлення

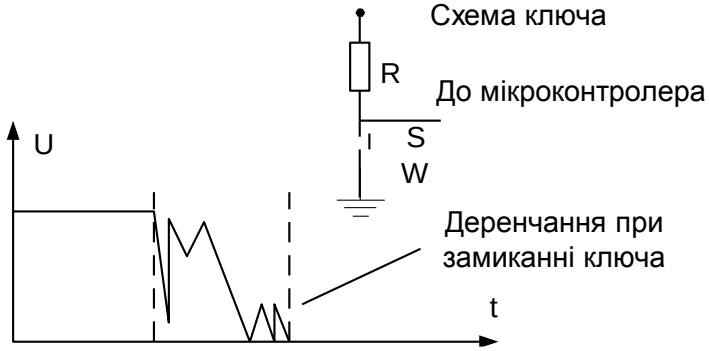


Рисунок 5.5 – Деренчання контактів

У пристроях, що використовують мікроконтролери, перешкоди, викликані деренчанням контактів, можуть інтерпретуватися як кілька розмикань і замикань ключа. Якщо клавіша використовується для керування пристроєм, то цей пристрій сприйме виникаючі перешкоди як багаторазове натискання клавіші, що викликає значні ускладнення при його використанні. Щоб усунути даний ефект, використовуються спеціальні схеми чи програмні методи для подавлення «деренчання».

Один з простих схемотехнічних способів усунення деренчання складається в увімкненні RC-кола (рис. 5.6). У цій схемі час, необхідний для заряду/ розряду конденсатора до граничної напруги, маскує «деренчання» контактів при перемиканні. Можна також встановити тригер Шмітта між схемою ключа і мікроконтролером, щоб підсилити ефект подвалення «деренчання». Недоліки цього методу – додаткові витрати на компоненти, що повинні бути встановлені на платі, і додатковий час, необхідний для заря-

ду/розряду RC-кола. Усе це може ускладнити використання даної схеми, тому що для деяких ключів з великим рівнем шумів додаткова затримка може скласти десяті долі секунди.

Значно кращий спосіб позбутися від деренчання — зробити це програмно. Якщо рівень напруги на виході ключа не змінюється протягом 20 мс, то можна вважати що деренчання закінчилося, і більше змін стану не очікується.

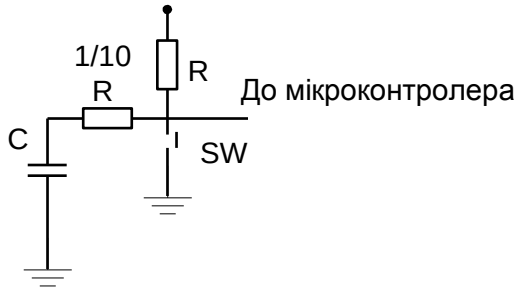


Рисунок 5.6 – Усунення деренчання контактів за допомогою конденсатора

Програма мовою С, що виконує фільтрацію деренчання, може мати такий вигляд:

```
DBounce:                        // Почати при високому рівні
                                сигналу на виході ключа
while (( Port & SW )!=0);       // Очікування низького рівня
                                сигналу на виході ключа
TMR =0;                         // Скинути таймер для чекання
                                20мс.
while ((( Port & SW ) == 0 ) && ( TMR < Twenty_msec ));
if ( TMR < Twenty_msec )
goto DBounce;                   //Деренчання ще не скінчилося,
                                повторити ще раз
```

Цей програмний код може бути модифікований для конкретної програми. Можливо його спрощення шляхом використання переривань для обробки сигналів, що надходять від таймера і при натисканні клавіші.

Увімкнення світлодіодів

Дуже часто вивід даних реалізується за допомогою світлодіодів LED - Light Emitting Diode), що досить дешеві і легко вмикаються до мікроконтролера. Звичайно, для світіння світлодіодів потрібен струм не більше 16мА, що для більшості мікроконтролерів знаходиться в діапазоні допустимих значень вихідних струмів. Варто пам’ятати, що світлодіод пропускає струм тільки в одному напрямку.

Типова схема підімкнення світлодіода до мікроконтролера показана на рис 5.7. У цій схемі світлодіод буде світитися, коли мікроконтролер видає на цей вивід сигнал «0» (низька напруга). Коли вивід працює як вхід даних чи на нього виводиться «1», то світлодіод буде вимкнутий.

Резистор з опором 220 Ом використовується для обмеження струму, тому що занадто великий струм може вивести з ладу мікроконтролер і світлодіод. Деякі мікроконтролери містять обмежники струму на вихідних лініях, що усуває потребу в обмежувальному резисторі.

Але все-таки доцільно, про всякий випадок, увімкнути цей резистор, щоб гарантувати, що коротке замикання на «землю» чи напруга живлення U_{cc} не виведе з ладу мікроконтролер.

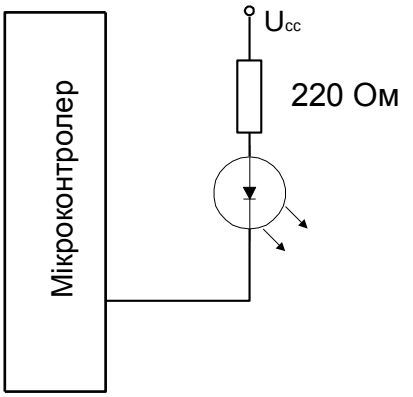


Рисунок 5.7 – Підімкнення світлодіода до виводу мікроконтролера

7-сегментний світлодіод індикатор

Імовірно, найпростіший спосіб виведення числових десяткових і шістнадцятирічних даних — це використання 7-сегментного світлодіодного індикатора. Такі індикатори були дуже популярні в 70-х роках, але згодом їхнє місце зайняли рідкокристалічні індикатори. Але світлодіодні індикатори дотепер є корисними приладами, що можуть бути увімкнуті в схему без великих зусиль для створення програмного забезпечення. Вмикаючи визначені світлодіоди (запалювальні сегменти), можна виводити десяткові числа (рис.5.8)

Кожен світлодіодний індикатор має свій буквений ідентифікатор (A, B, C, D, E, F чи G), і одна з ніжок світлодіода підімкнута до відповідного зовнішнього виводу. Другі ніжки всіх світлодіодів з’єднані разом і підімкнуті до загального виводу. Цей загальний вивід визначає тип індикатора: із загальним катодом чи із загальним анодом. Підімкнення індикатора до мікроконтролера здійснюється дуже просто: звичайно, індикатор вмикають як сім чи вісім (якщо використовується десяткова точка) незалежних світлодіодів.

Найбільш важливою частиною роботи при підімкненні до мікроконтролера декількох 7-сегментних індикаторів є призначення ліній введення-виведення для кожного світлодіоду. Вирішення цієї задачі на початку виконання проекту спростить Вам монтаж, розведення і налагодження пристрою надалі. Типовий спосіб підімкнення декількох індикаторів полягає в тому, щоб увімкнути їх паралельно і потім керувати протіканням струму через загальні виводи окремих індикаторів. Оскільки величина цього струму, звичайно, перевищує допустиме значення вихідного струму мікрокон-

тролера, то для керування струмом вмикаються додаткові транзистори, що вибирають, який з індикаторів буде знаходитися в активному стані.

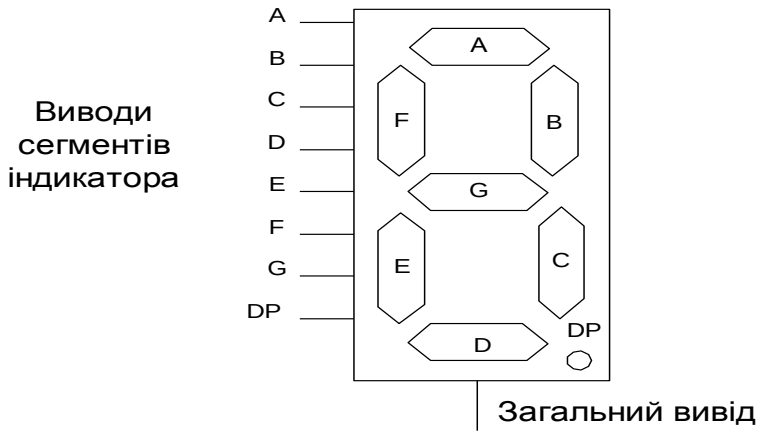


Рисунок 5.8 – 7-сегментний світлодіодний індикатор

На рис. 5.9 показано підімкнення до мікроконтролера чотирьох 7-сегментних індикаторів. У цій схемі мікроконтролер видає дані для індикації, послідовно переходячи від одного індикатора до іншого. Кожна цифра буде висвітлюватися протягом дуже короткого інтервалу часу. Це, звичайно, виконується за допомогою підпрограми обслуговування переривань таймера:

Int

- Зберегти регістри контексту;
- Скинути таймер і контролер переривань;
- `LED_Display = 0;` Вимкнути усі світлодіоди
- `LED_Output = Display[++Cur mod #LEDs];`
- `LED_Display = 1 << Cur` ; Висвітити значення для поточного індикатора
- Відновити вміст регістрів контексту;
- Повернення з переривання.

Ця підпрограма буде циклічно робити виведення цифри на кожен індикатор, дозволяючи протікання струму через транзистор, підімкнутий до його загального виводу. Щоб уникнути мерехтіння зображення, підпрограма повинна виконуватися зі швидкістю, що забезпечує увімкнення індикатора (світіння кожної цифри) принаймні 50 разів у секунду. Чим більше цифр, тим частіше повинні випливати переривання від таймера. Наприклад, при восьми індикаторах цифри повинні виводитися зі швидкістю 400 разів у секунду, тобто в два рази швидше, ніж для чотирьох індикаторів.

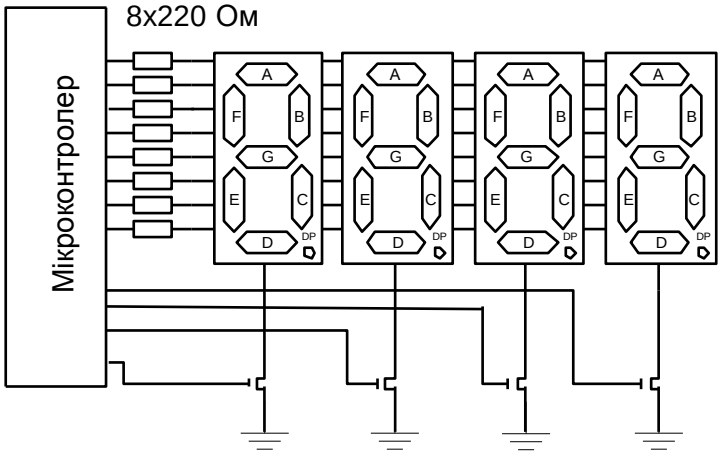


Рисунок 5.9 – Підімкнення до мікроконтролера чотирьох 7-сегментних індикаторів

У деяких ситуаціях може виявитися, що виділення кожному світлодіоду-індикатору окремого виводу мікроконтролера занадто витратно. Можна використовувати демультиплікатор з високим вихідним струмом, наприклад, Тл-мікросхему типу 74S138, замість дискретних транзисторів. Коли на обраному виході демультиплексора встановлюється низький рівень, то він пропускає струм підімкненого до нього індикатора, забезпечуючи виведення цифри. При цьому розведення монтажних з'єднань у пристрої виявляється більш простим. Варто звернути увагу на те, щоб використовуваний демультиплексор був здатний пропускати максимальний струм 140 мА, що протікає через загальний катод індикатора.

Поряд з 7-сегментними індикаторами існують 14- і 16-сегментні індикатори, що дозволяють виводити символи алфавіту («A»-«Z»> і «0»-«9»). При підімкненні цих індикаторів необхідно враховувати ті ж правила, що і для 7-сегментного індикатора.

Введення з матричної клавіатури

У багатьох програмах потрібно робити введення даних з клавіатури. Це може бути реалізовано за допомогою окремих кнопок, але такий підхід занадто витратний з точки зору використання ліній введення-виведення мікроконтролера. Кращим рішенням є використання матричної клавіатури, що являє собою набір ключів, об'єднаних у ряди і стовпці (рис. 5.10).

Для читання стану визначеного ключа на стовпець подається сигнал, а потім зчитується стан рядів. Звичайно, ряди вмикаються до високого потенціалу, а опитуваний стовпець з'єднується з землею. Якщо при скануванні рядів зчитується низький рівень сигналу, то це означає, що ключ у даній позиції ряд/стовпець замкнуто (рис. 5.11). На рис. 5.11 показані два МОН-транзистора, що використовуються для увімкнення стовпців до землі, виводи мікроконтролера можуть працювати в режимі з відкритим колектором. Таким чином, вони можуть імітувати роботу цих транзисторів, ро-

блячи їхнє увімкнення непотрібним. Транзистори, показані на рис 5.11, уведені для кращого розуміння роботи схеми.

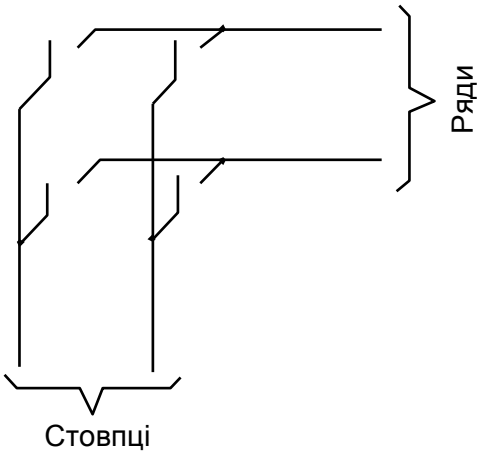


Рисунок 5.10 – З`єднання ключів в матричній клавіатурі

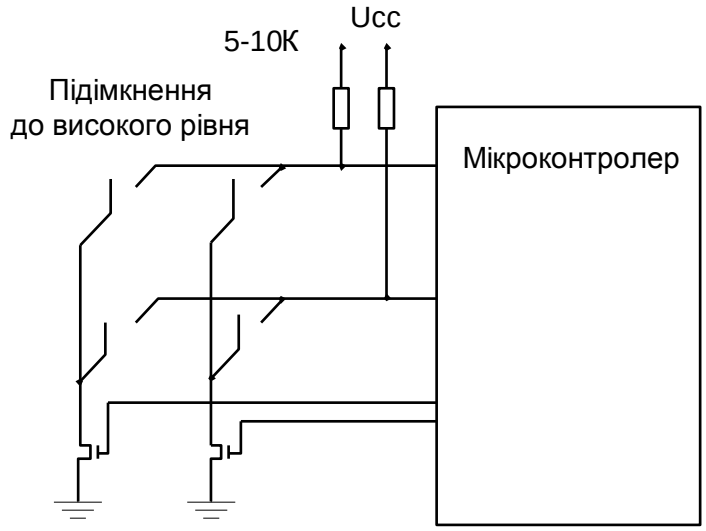


Рисунок 5.11 – Підімкнення до мікроконтролера матричної клавіатури

Матрична клавіатура може бути розширена практично до будь-якого розміру, використовуючи при цьому невелике число виводів мікроконтролера. Наприклад, 104-клавішна клавіатура персонального комп'ютера – це матриця, що містить 13×8 ключів. Необхідне програмне забезпечення практично не змінюється при підімкненні клавіатури різної розмірності.

Керування рідкокристалічним індикатором

Світлодіодні індикатори здатні відображати просту інформацію, але вони не мають того діапазону можливостей, що реалізують рідкокристалічні індикатори (РКІ). Ці індикатори дозволяють виводити дуже специфічні повідомлення, що робить інтерфейс користувача більш дружнім. РКІ також дуже корисні для виведення повідомлень про стан пристрою й іншої необхідної інформації в процесі налагодження програми.

РКІ, що реалізують виведення алфавітно-цифрових символів, мають репутацію пристроїв, з якими важко працювати. Щоб показати що це не так, у цьому розділі буде розказано як працюють РКІ, і як їх підмикати до мікроконтролера.

Більшість алфавітно-цифрових РКІ використовують для керування контролер Hitachi 44780 і реалізують загальний інтерфейс увімкнення. Завдяки цим обставинам РКІ, що забезпечують виведення від 8 до 80 символів (організованих у вигляді двох рядків по сорок символів чи чотири рядкі по двадцять символів), є цілком взаємозамінними, тому що їхнє застосування не вимагає якої-небудь зміни програмного забезпечення чи апаратних засобів.

Найчастіше РКІ, що використовують контролер Hitachi 44780, мають 14-виводні роз'єми з кроком 2,54 мм. Виводи РКІ мають таке призначення.

Вивід 1 — «Земля».

Вивід 2 — Напруга живлення U_{cc}.

Вивід 3 — Вхід регулювання контрастності зображення.

Вивід 4 — Сигнал вибору регістра даних чи команд (R/S).

Вивід 5 — Сигнал вибору режиму «чит./зап.» (R/W).

Вивід 6 — Синхросигнал E.

Вивід 7-14 - Лінії передавання даних.

З даного опису видно, що інтерфейс мікроконтролера з РКІ являє собою паралельну шину, що дозволяє просто і швидко здійснювати читання і записування даних у РКІ. Тимчасові діаграми сигналів на рис. 5.12 ілюструють процес видачі байта, що містить ASCII-код символа, на екран РКІ. ASCII-код містить 8 біт, що посилаються в РКІ по чотирьох чи по восьми бітах за один цикл обміну. Якщо використовується 4-бітний режим обміну, то повний 8-бітний код символа передається у вигляді двох 4-бітних «ніблів» (півбайтів): спочатку 4 старших біти, потім 4 молодших. Кожне посилання супроводжується синхросигналом E, що ініціює приймання даних у РКІ.

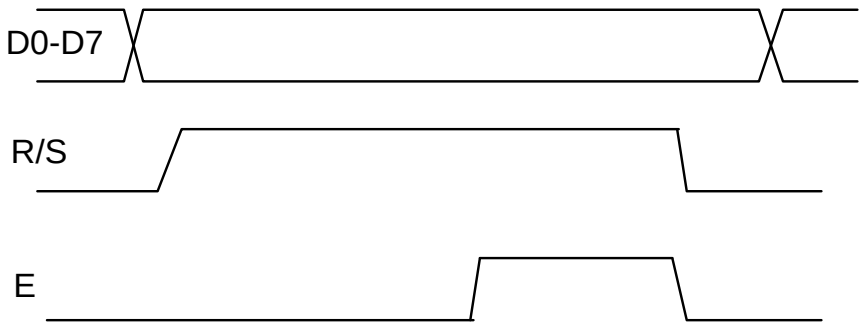


Рисунок 5.12 – Тимчасові діаграми сигналів при виводі символа на РКІ

Передавання 4 чи 8 бітів даних – це два основних режими паралельного обміну. Розглянемо деякі розуміння з приводу вибору того чи іншого режиму. Восьмибітовий режим передавання доцільно використовувати,

коли потрібна висока швидкість обміну і є не менш 10 доступних ліній для введення-виведення даних. Чотирибітовий режим передавання вимагає, як мінімум, 6 ліній введення-виведення. Щоб приєднати мікроконтролер до РКІ при чотирибітовому режимі використовуються тільки 4 старших розряди лінії даних DB7-4 (рис. 5.13).

Подальше скорочення числа необхідних ліній введення-виведення може бути забезпечене шляхом використання пускового регістра: у цьому випадку буде потрібно всього 3 лінії. У якості пускового регістра, звичайно, використовується мікросхема 74x174 (де «x» – чи HC, чи LS). Восьмибітовий режим також можна реалізувати за допомогою пускового регістра, але потрібно передавати дев'ятий біт, що використовується, щоб забезпечити видачу сигналу R/S. Біт R/S вказує, яка інформація передається – команда чи дані. Якщо цей біт встановлений у 1, то передаються дані, що можуть бути лічені чи записані в поточній позиції РКІ, обумовленої положенням курсору. Коли біт скинутий у "0", то при записуванні в РКІ передається команда, при читанні — зчитується стан РКІ після виконання останньої команди.

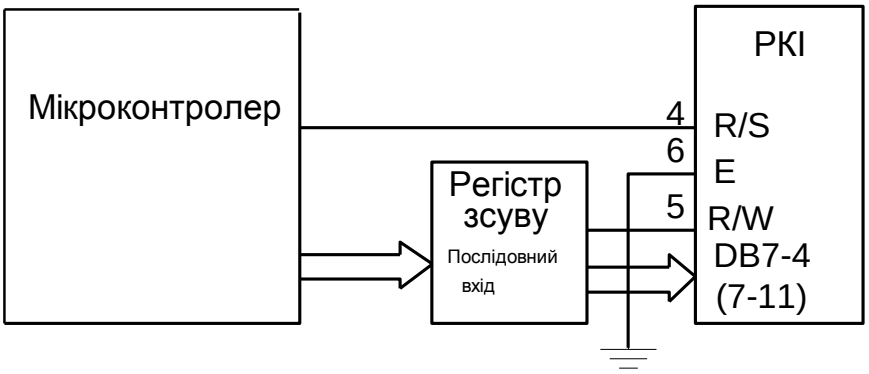


Рисунок 5.13 – Підімкнення РКІ до мікроконтролера при чотирибітовому режимі передавання

Набір символів, що виводиться РКІ під керуванням контролера 44780, в основному аналогічний символам, що подаються в ASCII-коді. Деякі символи РКІ не збігаються з ASCII: найважливіша відмінність – це відсутність символа «\», що має ASCII-код 0x05. Керуючі ASCII-коди з 0x008 по 0x01 не сприймаються РКІ як символи керування і можуть відображатися як японські ієрогліфи. Є вісім програмувальних символів, що виводяться за допомогою кодів з 0x000 по 0x007. Ці символи програмуються за допомогою команд, які встановлюють курсор РКІ на область пам'яті генератора символів («CGRAM») і задають вісім значень адреси для порядкового запису зображення символа. Наступні вісім байтів, записані в пам'ять, являють собою зображення кожного рядка програмувального символа, починаючи зверху.

Таблиця 5.1 – Набір команд, реалізованих РКІ

R/S	R/W	D7	D6	D5	D4	D3	D2	D1	D0	Команда/Опис
0	0	0	0	0	0	0	0	0	1	Очистити індикатор
0	0	0	0	0	0	0	0	1	*	Повернути курсор у початкову позицію Home
0	0	0	0	0	0	0	1	ID	S	Установити напрямок руху курсора
0	0	0	0	0	0	1	D	C	B	Дозволити екран/курсор
0	0	0	0	0	1	SC	RL	*	*	Перемістити курсор/Зрушити екран
0	0	0	0	1	DL	N	F	*	*	Установити розмірність інтерфейсу
0	0	0	1	A	A	A	A	A	A	Перемістити курсор на область CGRAM
0	0	1	A	A	A	A	A	A	A	Помістити курсор на екран
0	0	BF	*	*	*	*	*	*	*	Прочитати прапорець «Зайнято»
1	1	D	D	D	D	D	D	D	D	Вивести ASCII-символ на екран
1	1	D	D	D	D	D	D	D	D	Прочитати ASCII-символ з екрана

Призначення окремих бітів команд.

Вказання напрямку руху курсора:

ID - Переміщення курсора після записування кожного байта, якщо біт встановлений у 1;

S - Зрушення зображення на екрані після записування байта;

D - Екран увімкнути(1)/вимкнути(0);

C - Курсор увімкнути(1)/вимкнути(0);

B - Миготіння курсора увімкнути(1)/вимкнути(0). Переміщення курсора/зрушення екрана;

SC - Зрушення екрана увімкнути(1)/вимкнути(0);

RL - Напрямок зрушення. Вправо(1)/вліво(0). Установлення розмірності інтерфейсу;

DL - Розрядність даних 8(1)/4(0);

N - Число рядків на екрані 1(0)/2(1);

F - Розмір шрифту 5x10(1)/5x7(0). Установлення курсора на CGRAM;

A — Адреса читання /записування ASCII- символів;

D — Дані.

Зверніть увагу, що тип команди визначається числом старших нулів.

Прапорець «Зайнято» («busy») встановлюється на час виконання команди. Для написання програм, що працюють з максимально можливою швидкістю, необхідно опитувати цей прапорець, щоб виключити необхідність реалізації затримки, розрахованої на найгірший випадок виконання команд РКІ. Звичайно, швидкість обміну з РКІ не дуже важлива, і доцільно використовувати програмну затримку, що нескладно реалізувати. Вико-

нання всіх команд займає не більше 160мкс, крім команд «Очистити індикатор» і «Повернути курсор у початкову позицію», що вимагають максимум 4,1 мс. Для найгіршого випадку можна встановити затримку в 5 мс, щоб забезпечити деякий запас для надійного функціонування.

Різні типи РКІ виконують команди з різною швидкістю. Вище зазначені максимальні значення затримок, але деякі індикатори вимагають менший час для виконання команд. Однак, якщо не використовується опитування прапорця «Зайнято», то рекомендується завжди використовувати максимальні затримки.

У більшості застосувань лінію «R/W» приєднують до землі, тому що читання стану РКІ не потрібно. Це значно спрощує додаток, оскільки для зчитування даних необхідно змінювати режим роботи виводів — із записування на читання. У деяких випадках можливість читання стану РКІ буває корисна, наприклад, при прокручуванні даних на екрані. Підімкнення лінії «R/W» до землі також звільняє один вивід мікроконтролера.

РКІ з розміром символів 5x10 точок практично не випускаються, тому біт «F» у команді «Установлення розмірності інтерфейсу» повинний завжди бути рівним 0.

Перед тим, як вводити в РКІ команди або дані, його необхідно ініціалізувати. Це робиться за допомогою такої послідовності дій.

Для 8-бітового режиму.

1. Почекати більше 15мс після подачі живлення.
2. Записати 0x30 у РКІ і чекати 5мс до завершення виконання команди.
3. Записати 0x30 у РКІ і чекати 160мкс до завершення виконання команди.
4. Знову записати 0x30 у РКІ і чекати 160мкс до завершення виконання команди чи опитувати прапорець «Зайнято».
5. Встановити робочі характеристики РКІ:
   - ввести «Установлення розмірності інтерфейсу»;
   - ввести 0x10, щоб вимкнути екран;
   - ввести 0x01, щоб очистити екран;
   - ввести «Установлення напрямку руху курсора», щоб установити поводження курсора;
   - Ввести «увімкнення екрана/курсора», щоб увімкнути екран і, якщо потрібно, курсор.

Для ініціалізації індикатора в 4-бітовому режимі використовується пересилання двох окремих півбайтів, а не повних байтів, що складають команду. Як було відзначено вище, при посиланні байта спочатку посилається старший півбайт, потім молодший. При цьому кожне посилання чотирьох біт супроводжується перемиканням лінії E.

1. Почекати більше 15мс після подачі живлення.
2. Записати 0x3 у РКІ і чекати 5мс до завершення виконання команди.

3. Записати 0x3 у РКІ і чекати 160 мкс до завершення виконання команди.

4. Знову записати 0x3 у РКІ і чекати 160 мкс до завершення виконання команди чи опитувати прапорець «Зайнято».

5. Встановити робочі характеристики РКІ:

• ввести 0x02 у РКІ, щоб дозволити 4-бітовий режим;

• усі наступні команди/дані вимагають пересилання двох півбайтів;

• ввести «установлення розмірності інтерфейсу»;

• ввести 0x1, 0x0, щоб вимкнути екран;

• ввести 0x0, 0x1, щоб очистити екран;

• ввести «установлення напрямку руху курсора», щоб установити поводження курсора;

• ввести «увімкнення екрана/курсора», щоб увімкнути екран і, якщо потрібно, курсор.

Після того, як ініціалізація завершена, РКІ готовий до приймання команд і даних. Останнє питання, що стосується РКІ – як установити контрастність зображення. Звичайно, для цього використовується потенціометр, увімкнутий як подільник напруги (рис. 5.14). У такий спосіб отримуємо напругу, що змінюється у діапазоні від «землі» до U_{cc}, що забезпечує регулювання контрастності зображення символів на екрані РКІ.

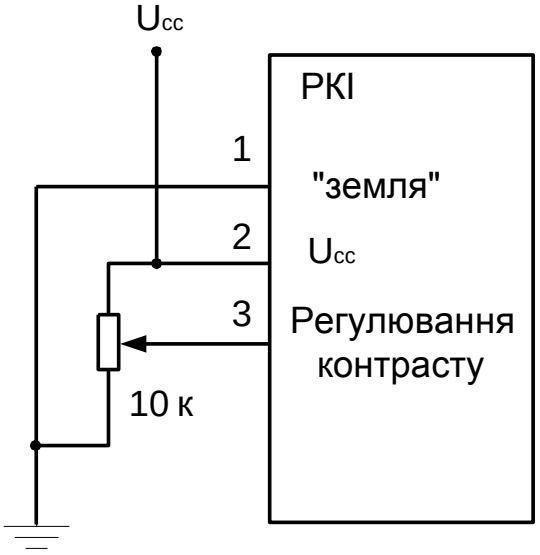


Рисунок 5.14 – Управління контрастністю зображення РКІ

Найбільша проблема з РКІ полягає в тім, щоб правильно його ініціалізувати. Якщо виконувати наведені вище інструкції, то проблем не повинно виникнути. Також варто пам’ятати, що мінімальна тривалість імпульсу на вході E повинна складати 450 нс. Якщо тривалість імпульсу менша, то робота РКІ буде нестійкою.

Даного в цьому розділі опису роботи алфавітно-цифрового РКІ, керованого контролером Hitachi 44780, досить для того, щоб почати експе-

рименти з підімкненням РКІ до мікроконтролера. Є багато цікавих можливостей застосування РКІ, особливо якщо використовувати вивід програмувальних символів. Безліч різних додатків, описаних у цій книзі, використовують РКІ, і дані у відповідних розділах приклади програм можна використовувати в розробках. Однак у дійсності не так важко самостійно спроектувати і запрограмувати пристрій, що використовує РКІ.

Керування двигуном/реле

У цьому розділі мова йтиме про керування електромагнітними реле і моторами за допомогою мікроконтролера. Ці пристрої не можуть безпосередньо керуватися мікроконтролером, тому що вони споживають великий струм і є джерелами перешкод. Необхідно використовувати спеціальні інтерфейси, щоб керувати електромагнітними пристроями.

Найпростіший метод керування цими пристроями полягає в тім, щоб просто вмикати і вимикати їх. На рис. 5.15 мікроконтролер відкриває транзисторну пару Дарлінгтона, що приводить до протікання струму через котушку реле, що замикає контакти. Щоб розімкнути реле, транзистори закриваються подачею "0" на вихід мікроконтролера. Шунтувальний діод використовується для подавлення імпульсу напруги, що виникає при вимиканні струму. Цей імпульс індукується магнітним потоком у котушці і може привести до ушкодження джерела живлення реле і навіть мікроконтролера. Ніколи не слід забувати про вмикання такого діода в схемах керування електромагнітними пристроями. Імпульс напруги має амплітуду порядку декількох сотень вольт і тривалість декілька наносекунд. При цьому діод знаходиться в режимі пробою, і через нього протікає струм, що викликає зниження індукованої напруги.

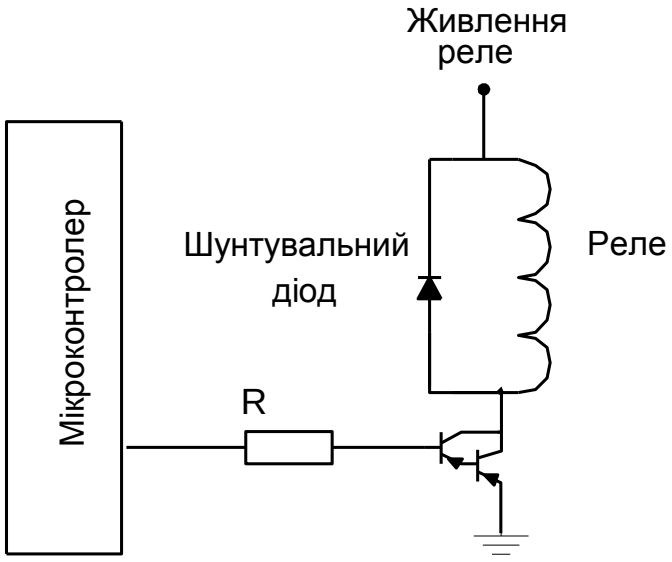


Рисунок 5.15 – Керування реле за допомогою мікроконтролера

Замість розробки схеми на дискретних компонентах можна використовувати для керування електромагнітними приладами спеціальні інтегральні мікросхеми. Дуже корисними приладами є мікросхеми серії ULN200x,

що містять кілька вихідних каскадів-драйверів, що використовують транзисторні пари Дарлінгтона і шунтувальні діоди.

Такі ж апаратні засоби застосовуються для керування електромоторами. Якщо мотор повинний обертатися тільки в одному напрямку, то можна використовувати схему на рис. 5.18.

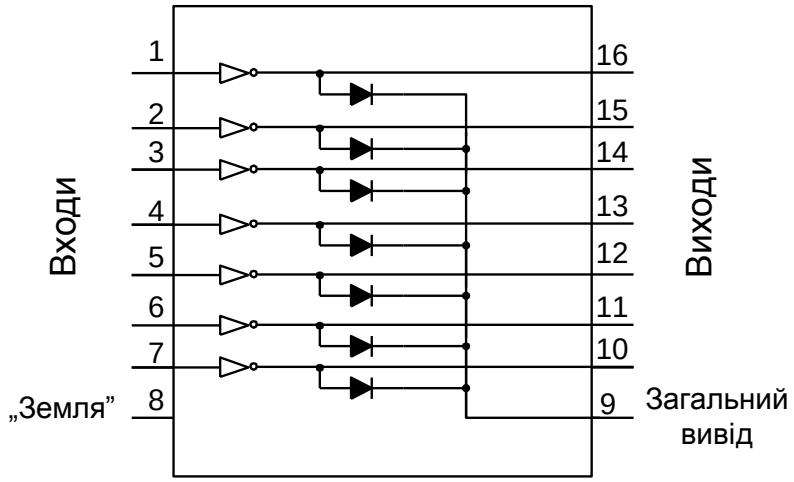


Рисунок 5.16 – Набір вихідних драйверів в мікросхемі ULN2003A

Для керування обертанням мотора в будь-якому напрямку служить мостова схема з’єднання ключів (транзисторів), показана на рис. 5.17. Якщо в цій схемі розімкнуті всі ключі, то струм через мотор не протікає, і він не буде обертатися. Якщо замкнуті ключі 1 і 4, то мотор буде обертатися в одному напрямку, якщо замкнуті ключі 2 і 3 – в іншому. Якщо одночасно замкнути обидва ключі на одній стороні моста, то через коротке замикання може згоріти запобіжник чи джерело живлення.

Керування швидкістю мотора, звичайно, здійснюється за допомогою сигналів із широтно-імпульсною модуляцією (ШІМ). Частота ШІМ-сигналів повинна бути більша 20 кГц, щоб уникнути виникнення при обертанні мотора звукового сигналу, що може бути дуже дратівливим.

Оскільки мікросхема ULN2003A спрощує увімкнення реле, мікросхема 293D може бути використана для керування мотором. Мікросхема 293D, показана на рис. 5.18, може керувати двома моторами, приєднаними до вихідних буферних каскадів (виводи 3, 6, 11 і 14). Виводи 2, 7, 10 і 15 використовуються для керування рівнем напруги на буферних виходах мікросхеми (ключі в мостовій схемі, показаної на рис 5.17). Виводи 1 і 9 керують вмиканням/вимиканням вихідних буферів. На ці виводи можуть подаватися ШІМ-сигнали, що робить керування швидкістю обертання мотора дуже простим.

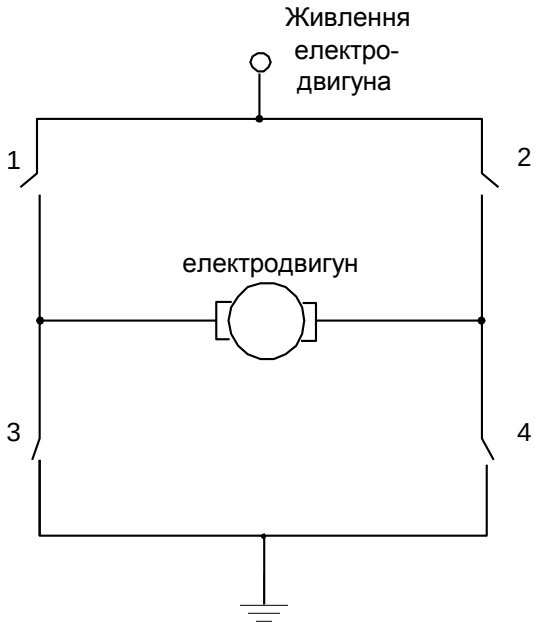


Рисунок 5.17 – Керування електромотором за допомогою мостової схеми

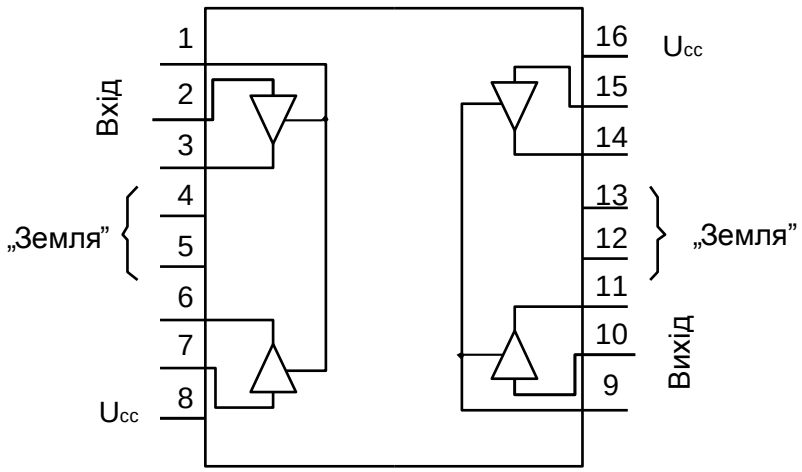


Рисунок 5.18 – Мікросхема 293D для керування електромоторами

Напруга живлення мікросхеми U_{cc} складає +5 В, а напруга живлення електромотора U_{cc} може знаходитися в межах від 4.5В до 36В. Максимальний струм, що протікає через електромотор, складає 500 мА. Як і ULN2003A, мікросхема 293D містить вбудовані шунтувальні діоди. Це означає, що при підімкненні до неї електромотора не потребуються зовнішні шунтувальні діоди.

На рис. 5.19 показані резистор і конденсатор, що іноді використовуються для подавлення перешкод. Ці два компоненти, підімкненні до щіток електромотора, зменшують електромагнітне випромінювання й імпульсні перешкоди, що виникають при його роботі. Звичайно, їхнього підімкнення не потрібно, але якщо спостерігається хибна робота мікроконтролера під час роботи мотора, то можна ввести в схему конденсатор ємністю 0.1 мкФ і резистор 5 Ом.

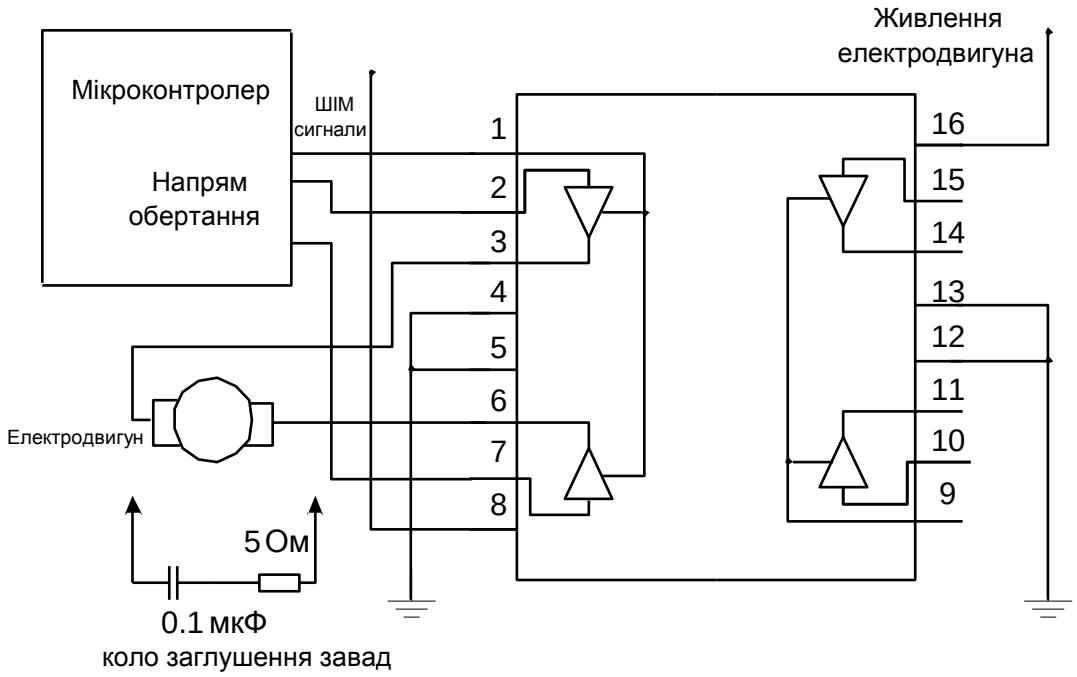


Рисунок 5.19 – Підімкнення електродвигуна до мікросхеми 293D

Мікросхема 293D може також використовуватися для керування чотириполюсним кроковим двигуном – кожний з буферних виходів служить для керування одним з полюсів двигуна. При цьому виводи 1 і 9 повинні бути підімкненні до живлення, тому що не потребуються ШІМ-сигнали для керування швидкістю двигуна.

Кроковий двигун

Для крокових двигунів набагато простіше розробляти програмне забезпечення, ніж для звичайних двигунів постійного струму. Двигун може обертатися на один крок чи обертатися з визначеною швидкістю, що задається частотою виконання кроків. Стосовно апаратного інтерфейсу крокові двигуни не набагато складніші в підімкненні і споживають більший струм (тобто мають більш низький ККД), але ці недоліки компенсуються перевагами, що забезпечує їхнє програмне керування.

Двополюсний кроковий двигун складається з розміщеного на валу постійного магніту, положення якого задається парою котушок (рис. 5.20).

Щоб переміщати магніт і вал, на котушки подається напруга з різними фазами. У такий спосіб забезпечується притягання магніту і його переміщення в необхідному напрямку.

Для обертання двигуна, показаного на рис. 5.20, за годинниковою стрілкою, може бути використана послідовність фаз, приведена в табл. 5.2.

При даній послідовності котушка А притягає північний полюс магніту, щоб перевести магніт у вихідне положення. Потім котушка В притягає південний полюс магніту, повертаючи його на 90 градусів. Далі обертання мотора продовжується, причому за кожен крок мотор повертається на 90 градусів.

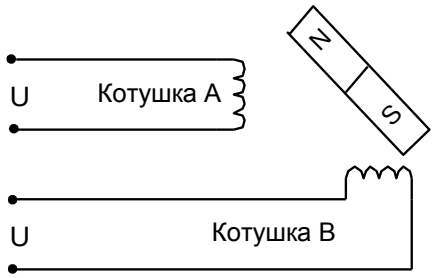


Рисунок 5.20 – Спрощена схема крокового двигуна

Таблиця 5.2 – Послідовність фаз крокового двигуна

Крок	Фаза	Котушка А	Котушка В
1	0	S	
2	90		N
3	180	N	
4	270		S
5	360/0	S	

На кінці валу крокового двигуна часто встановлюється шестеренчаста передача, що перетворює кожен крок у дуже мале кутове переміщення. Таким способом забезпечується одержання більшого значення обертового моменту і більш точний позиційний контроль переміщення вала.

Крокoвий двигун може керуватися мікросхемами, аналогічними 293D, при цьому кожна пари виходів керує однією котушкою. Але існують спеціальні мікросхеми керування кроковим двигуном такі, як UC1517 (рис. 5.21). У цій мікросхемі мікроконтролер посилає кроковий імпульс STEP і задає напрямок обертання сигналом DIR. Вивід INH вимикає вихідні драйвери, даючи можливість обертати двигун вручну. Мікросхема UC1517 може видавати на котушки дворівневі сигнали, що дозволяє підвищити ККД і знизити рівень перешкод. При цьому забезпечується також можливість повороту вала на півкроку, тобто на 45 градусів, а не тільки на 90 градусів. Зазначені можливості реалізуються специфічним способом для різних типів крокових двигунів, тому перш ніж їх використовувати необхідно добре ознайомитися з пристроєм і принципом дії застосовуваного двигуна.

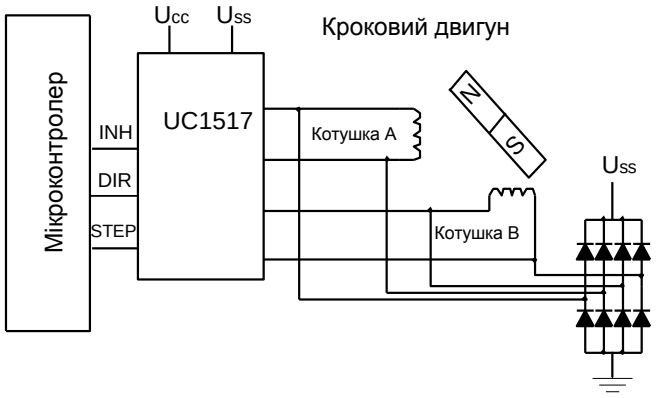


Рисунок 5.21– Керування двигуном за допомогою мікросхеми UC1517

Керування сервоприводом

Сервомотори призначені для використання в радіокеруючих аеропланах, машинах і човнах. Вони легко під'єднуються до мікроконтролера і часто використовуються в робототехніці й інших додатках, де потребуються прості механічні рухи. Це може показатися дивним, але позиційний сервомотор є аналоговим пристроєм.

Виходом R/C сервомотора, звичайно, служить колесо, що повертається на кут від 0 до 90 градусів. Існують сервомотори, здатні повертатися в діапазоні від 0 до 180 градусів, а також сервомотори, що мають дуже великий обертальний момент для спеціальних додатків. Звичайно, до сервомотора досить підімкнути напругу живлення +5В, «землю» і вхідний сигнал.

R/C сервомотор дійсно аналоговий пристрій. На вхід подається ШІМ-сигнал з цифровими рівнями напруги "0" і "1". Тривалість імпульсу від 1.0 мс до 2.0 мс, частота повторень 20 мс (рис. 5.22). Тривалість ШІМ-імпульса визначає положення колеса сервомотора. Імпульс тривалістю 1.0 мс відповідає положенню колеса 0 градусів, імпульс тривалістю 2.0 мс – 90 градусів.

При використанні мікроконтролерів, що реалізують на виході ШІМ-сигнали, керувати сервомотором дуже легко, хоча можна не досягти необхідної точності позиціонування. Для мікроконтролерів, що не мають спеціальних ШІМ-виходів, можна одержати ШІМ-сигнали програмно за допомогою такої процедури:

```
Interrupt                    // Використовується оброблювач переривання
    Зберегти вміст регістрів контексту
    Вивести "1" на ШІМ лінію
    Чекати 1 мс
    Цикл на 1 мс             // Вивести ШІМ-сигнал
    if Loop_Counter > Specified_ServoPos
    Output a, 0              // Завершення ШІМ-сигналу
    Установити таймер на переривання через 18 мс
    Відновити вміст регістрів контексту
    Повернення з переривання
```

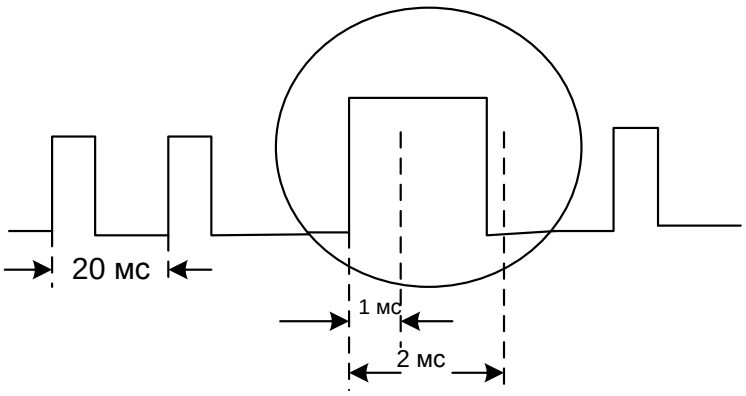


Рисунок 5.22 – ШІМ-сигнал для керування сервомотором

Ця програма може бути легко модифікована для керування більш ніж одним сервомотором шляхом введення додаткових вихідних ліній ШІМ-сигналів і змінних «Specified_ServoPos». Такий спосіб керування сервомоторами непоганий тим, що змінні «Specified_ServoPos» можуть обновлятися, не впливаючи на роботу оброблювача переривань. Обслуговування переривання займає 2 мс із кожних 20 мс. Це значить, що на забезпечення функцій ШІМ потрібно 10 відсотків процесорного часу незалежно від числа керованих сервомоторів.

Перетворення рівнів RS-232

Найбільш часте перетворення рівня сигналу використовується для забезпечення зв'язку по інтерфейсу, за допомогою якого мікроконтролер повинен з'єднуватись з ведучим комп'ютером. Перетворення рівнів можна здійснити застосовуючи різні мікросхеми, але використання деяких рішень дозволяє спростити виконання цієї задачі.

Напруги і логічні рівні інтерфейсу RS-232 трохи незвичайні (рис. 5.23). Перше, що може викликати проблему – це великий розмах напруги. Але напруги і струми можуть бути зменшені за допомогою струмообмежувального резистора, що вмикається між входом мікроконтролера і роз'ємом RS-232. Цей резистор буде захищати фіксуючі діоди на входах мікроконтролера від протікання великого струму, а також утримувати рівні вхідної напруги в допустимих межах.

Цей метод добре працює, тому що від'ємна напруга (логічна «l» в інтерфейсі RS-232) буде сприйматися як "0" В на вході мікроконтролера, а додатня напруга (логічний «0» в інтерфейсі RS-232) навіть після обмеження до рівня +5 В буде вище області перемикання, що знаходиться в діапазоні від -3 В до +З В.

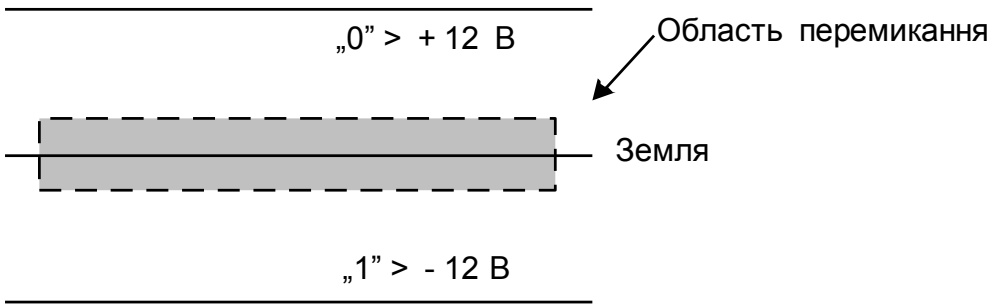


Рисунок 5.23 – Рівні напруг для інтерфейсу RS-232

Таке рішення можна використовувати тільки для приймання сигналів RS-232. Якщо необхідно пересилати дані по інтерфейсу RS-232, то негативну напругу для логічної l не можна одержати за допомогою даної схеми. Можна одержати тільки 0 В, а це значення знаходиться всередині області перемикання приймача RS-232. Часто ця обставина ігнорується, і мікроконтролер просто під'єднується до приймача сигналів RS-232 через струмообмежувальний резистор. Іноді така схема з'єднання нормально працює, але

варто мати на увазі, що можливість передавання даних по інтерфейсу RS-232 з використанням тільки струмообмежувального резистора залежить від використовуваного мікроконтролера і його конкретного застосування. Такий спосіб вирішення задачі не рекомендується.

Замість цього негативний рівень напруги для передавання можна одержати від приймача. Як показано на рис. 5.24, дані надходять у мікроконтролер через резистор 10 Ом. Нормально лінія знаходиться в негативному стані (передається «l»). Коли передається «0», напруга установиться на рівні U_{cc}. Це дуже дешевий і елегантний спосіб реалізації приймання і передавання за стандартом RS-232: 3-провідний інтерфейс RS-232 з «земляним» проводом.

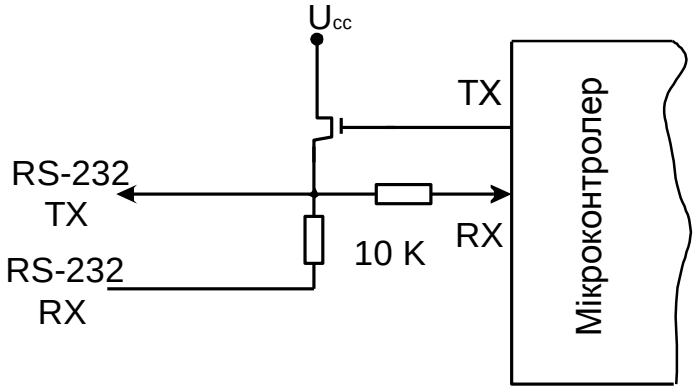


Рисунок 5.24 – Інтерфейс між RS232 та КМОН-мікроконтролером

Застосування описаного методу викликає деякі проблеми, які необхідно вирішити перед його використанням у конкретному додатку. Прийняті і передані дані будуть мати інверсні рівні сигналів "0" і "1" у порівнянні з звичайною «позитивною» логікою, тому мікроконтролер повинен зробити їхню зворотну інверсію перед подальшим використанням. Це може стати проблемою для деяких типів мікроконтролерів, у яких послідовний інтерфейс не виконує інвертування отриманих даних.

Описаний метод дозволяє реалізувати тільки трипровідний інтерфейс RS-232. Якщо ведучий мікроконтролер потребує обміну сигналами квитування (підтвердження готовності) DTR-DSR і CTS-RTS, то прийдеться використовувати інший вид інтерфейсу чи замкнути накоротко лінії сигналів квитування, з’єднавши виводи DTR з DSR і CTS з RTS.

Інша проблема полягає в тому, що дані, передані мікроконтролеру, будуть посилатися назад до передавача. Це відбувається тому, що при високому рівні прийнятого сигналу (коли передається логічний “0”) такий же високий рівень буде встановлений на виході передавача, тобто переданий сигнал ніби відбивається (ефект «луни»). У багатьох випадках такий ефект не викликає яких-небудь проблем, а іноді навіть звільняє від необхідності писати спеціальну програму, імітуючи «луну». Але в програмах, що не очікують приходу «луни», можуть виникнути визначені труднощі. Поява

«луни» приводить до того, що дані не можуть передаватися в дуплексному режимі, тобто одночасно в двох напрямках.

Якщо даний спосіб реалізації інтерфейсу RS-232 викликає проблеми в конкретній програмі, то варто використовувати для організації інтерфейсу мікросхему Maxim MAX232, що містить внутрішній генератор негативної напруги.

Генерація випадкових чисел

Однією з найбільш важких задач при роботі з мікроконтролером є генерація випадкових чисел. Існує безліч кандидатських дисертацій, написаних на цю тему. Проблема полягає в тому, що ці пристрої за своєю природою є детермінованими. Вони спеціально спроектовані таким чином, щоб щоразу вирішувати дану задачу тим самим шляхом, і не мають можливості створювати випадкову інформацію.

Деякі мови програмування і комп'ютерні системи реалізують спеціальні функції, що створюють послідовність випадкових чисел. Однак для їхнього обчислення потрібно одне чи кілька початкових чисел, що повинні бути випадковими, інакше ця функція буде завжди генерувати ті самі числа.

У мікроконтролерах існує три способи створення випадкових чисел і усі вони вимагають використання зовнішніх даних. Перший спосіб застосовується тоді, коли мікроконтролер підімкнений до годинника реального часу. Читання часткою секунди при вмиканні живлення дасть початкове випадкове число.

Імовірність того, що увімкнення буде відбуватися в один і той же час з точністю до сотих і тисячних часток секунди, дуже мала.

Наступний спосіб полягає в тому, щоб записати таблицю випадкових чисел у пам'ять програм і зберігати їх як константи. Хоча ці числа будуть не зовсім випадковими, для деяких додатків це цілком прийнятно. Цей спосіб, однак, вимагає використання деяких засобів для модифікації записаного в пам'ять програм об'єктного коду і розміщення в ній випадкових чисел.

Останній спосіб заснований на фіксації деяких випадкових зовнішніх подій. Найчастіше мікроконтролер вмикається до зворотно-зміщеного діода чи до виходу радіолінії, на яку не надходить несуча частота. Потім відбувається записування числа випадкових перешкод («шуму»), що надходять за визначений період часу.

Персональні комп'ютери

Пристроєм, з яким найчастіше потрібно реалізувати інтерфейс, є персональний комп'ютер. Існує чотири основних інтерфейси, за допомогою яких мікроконтролер зв'язується з комп'ютером: послідовний порт (RS-232), паралельний порт, інтерфейс клавіатури і миші, шина ISA. Існують і інші методи зв'язку з комп'ютером, але перераховані вище є найбільш розповсюдженими. Існують книги, де розказано як підімкнути пристрої, використовуючи дані інтерфейси. У цьому розділі немає можливості розг-

лянути всі ці інтерфейси докладно, особливо шину ISA, що забезпечує переривання і прямий доступ до пам'яті. Тут буде даний тільки загальний огляд цих інтерфейсів і пояснення того, як можна з їхньою допомогою підімкнути до комп'ютера мікроконтролер.

Персональний комп'ютер – досить складний пристрій для реалізації інтерфейсу з ним. Причиною цього є непередбачене поводження різних комп'ютерів, що виявляється в установленні довільних значень окремих бітів і байтів у процесі функціонування комп'ютера. Дана обставина може показатися дивним, з огляду на те, що ці комп'ютери були розроблені майже 20 років тому. Але в міру їхнього розвитку функції, що раніше були реалізовані за допомогою дискретних компонентів, тепер виявилися захована глибоко у внутрішню структуру використовуваних у комп'ютері спеціалізованих замовлених мікросхем (ASIC). Використовувані в комп'ютерах мікропроцесори, що колись мали передбачуваний командний цикл, сьогодні реалізують функції, що не дуже давно виконували тільки великі комп'ютери. У результаті функції, що здаються легкими для розуміння і реалізації, можуть по-різному виконуватися різними комп'ютерами чи узагалі виявляються нездійсненними. Ця обставина може перетворитися в „кошмар" для розробника, особливо коли він намагається зв'язатися з комп'ютером за допомогою нестандартного апаратного інтерфейсу.

Велике значення має операційна система, що установлена на персональному комп'ютері. Програми для персональних комп'ютерів пишуться для операційних систем MS-DOS, Windows 95/98/2000 і Windows XP; у меншій мірі використовуються OS/2, Windows NT і симулятори MS-DOS, працюючі з різними операційними системами. Кожна з цих операційних систем використовує різні ресурси (наприклад, таймери і вектори переривань), що ускладнює написання програмного забезпечення для зв'язку з мікроконтролером.

Після всього сказаного виникає питання, навіщо розробляється апаратура, орієнтована на зв'язок з комп'ютером, за винятком випадків коли продукція буде випускатися мільйонним тиражем чи вона спеціально призначена для роботи з визначеною моделлю комп'ютера. При спробах вирішення зазначених вище проблем у розробника виникає більше питань, ніж існують готові відповіді.

Можна сформулювати деякі рекомендації з розробки програм, що можуть працювати разом з різними моделями персональних комп'ютерів.

1. Використовувати інтерфейси, що завжди працюють на різних моделях комп'ютерів з різними операційними системами.

2. Апаратура зв'язку з комп'ютером повинна бути інтелектуальною, і мати можливість обробляти блоки команд і даних, а не тільки біти і байти.

Далі буде показано як ці правила застосовуються для різних інтерфейсів.

Послідовний інтерфейс RS-232 найкраще задовольняє ці правила. Електричні і тимчасові специфікації даного інтерфейсу не змінюються з

часом. Комп'ютер може мати 9-контактний (DB-9M) чи 25-контактний (DB-25M) розніми, що під'єднуються до інтерфейсу RS-232 як термінальне устаткування DTE (DataTerminal Equipment). Призначення контактів рознімів приведено в табл. 5.3.

Звичайно, ці сигнали виводяться з мікроконтролера за допомогою гнучкого багатожильного кабелю. Рекомендується використовувати 9-контактний рознім, тому що він має менші розміри.

Лінії RTS, CTS, DSR, DCD, DTR і R1 служать для передавання сигналів квитування, що використовуються в деяких видах зв'язку, щоб підтвердити готовність устаткування для посилання даних за допомогою інтерфейсу RS-232. Наприклад, ці лінії використовуються базовою системою введення-виведення (BIOS) деяких комп'ютерів. Щоб ці сигнали не викликали проблем при обміні даними, можна підімкнути лінію DSR до DTR і лінію CTS до RTS. Таке підімкнення «обдурить» апаратне і програмне забезпечення комп'ютера, що у будь-який час будуть одержувати сигнали готовності до приймання і передавання даних.

Таблиця 5.3 – Контакти послідовного інтерфейсу RS-232

DB-25M		**DB-9M**
Вихід 1 — не увімкнутий	Вихід 14 — не увімкнутий	Вихід 1 — вхід DCD
Вихід 2 — вихід TX даних	Вихід 15 — не увімкнутий	Вихід 2 — вихід RX
Вихід 3 — вхід RX даних	Вихід 16 — не увімкнутий	даних
Вихід 4 — вихід RTS	Вихід 17 — не увімкнутий	Вихід 3 — вихід TX
Вихід 5 — вихід CTS	Вихід 18 — не увімкнутий	даних
Вихід 6 — вхід DSR	Вихід 19 — не увімкнутий	Вихід 4 — вихід DTR
Вихід 7 — «земля»	Вихід 20 — вихід DTR	Вихід 5 — «земля»
Вихід 8 — вхід DCD	Вихід 21 — не увімкнутий	Вихід 6 — вхід DSR
Вихід 9 — не увімкнутий	Вихід 22 — вхід R1	Вихід 7 — вихід RTC
Вихід 10 — не увімкнутий	Вихід 11 — не увімкнутий	Вихід 8 — вхід CTS
Вихід 11 — не увімкнутий	Вихід 24 — не увімкнутий	Вихід 9 — вхід RI
Вихід 12 — не увімкнутий	Вихід 25 — не увімкнутий	
Вихід 13 — не увімкнутий		

Ці сигнали були спочатку введені в стандарт RS-232, коли комп'ютери мали набагато менші можливості, ніж зараз. Призначення цих сигналів полягає в тому, щоб зупинити передавач, якщо приймач не готовий до приймання даних, а також інформувати передавач про стан модему. Практично у всіх додатках і пристроях, які можна купити в магазині, лінії підтвердження готовності з'єднані разом (як описано вище), і для обміну даними використовуються тільки три лінії: дві лінії передавання і «земля».

За допомогою функцій BIOS порт RS-232 у персональному комп'ютері має стандартний програмний інтерфейс, що дозволяє програмам без великих проблем працювати з різними комп'ютерами й операційними системами. Крім того операційна система містить програму емуляції термінала, що допомагає при налагодженні програми. Це дає велику перевагу при розробленні програм, що мають інтерфейс із комп'ютером, тому

що при налагодженні програми можна використовувати емулятор термінала. а не частково налагоджений інтерфейс користувача. За допомогою текстових команд емулятор термінала дозволяє розробляти різні програми, при цьому введення команд і читання відповідних повідомлень виконується в простій і зручній формі.

Існує думка, що зручні для людини текстові команди менш ефективні, ніж двійкові коди. Однак, звичайно, ніяких труднощів не виникає. При швидкості обміну 9600 бод необхідність передавання двох байтів замість одного приведе до того, що пересилання даних займе 2 мс замість 1 мс. Більшість мікроконтролерних інтерфейсів не передає великих обсягів даних, тому додаткова затримка буде непомітною для користувача, а при цьому він одержує значні переваги у вигляді більш простої процедури налагодження програми.

Якщо потрібно передавати великі обсяги даних — порядку десятків Кбайтів у хвилину і більше, то краще не використовувати інтерфейс RS-232. Замість цього краще безпосередньо підімкнутися до системної шини комп’ютера чи використовувати мережний інтерфейс.

Доступ до послідовного порту може бути отриманий через файлову систему шляхом «відкриття» імен (від COM1 до COM4) для чотирьох стандартних послідовних портів комп’ютера. Якщо комп’ютер містить графічну карту Super-VGA то порт COM4 буде недоступний, тому що адресний простір введення-виведення порту COM4 використовується графічним адаптером, сумісним з IBM 8514/A.

Використання паралельного (принтерного) порту також є дуже популярним методом з’єднання мікроконтролера з комп’ютером. Цікавою особливістю цього інтерфейсу є безліч можливих способів його використання. Реалізація паралельного інтерфейсу в традиційному варіанті, коли дані передаються від комп’ютера до приймача, показана на рис. 5.25.

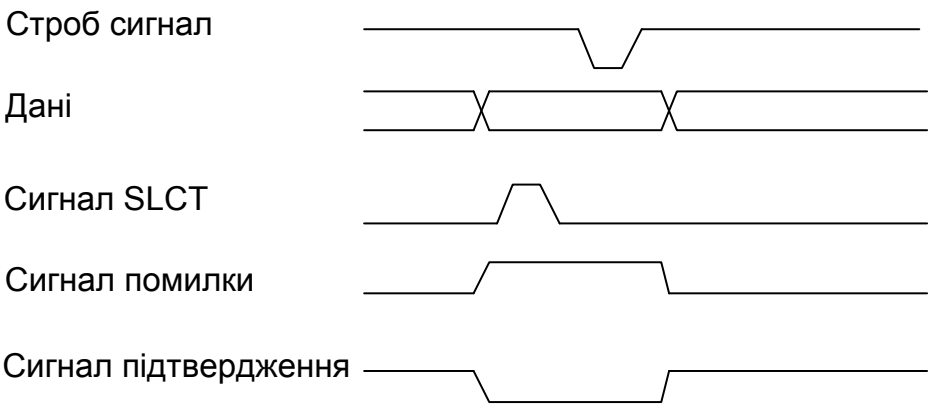


Рисунок 5.25 – Тимчасова діаграма функціонування паралельного порту

При цьому методі інтерфейсу перевіряються різні сигнали підтвердження, щоб визначити готовність зовнішнього пристрою до приймання перед видачею даних комп’ютером. Паралельний порт у даному випадку служить тільки для передавання даних з комп’ютера. У більшості програм,

що використовують принтерний порт, дані передаються в двох напрямках. При цьому лінії підтвердження зв'язку використовуються для вказання напрямку передавання даних, а не для перевірки стану зовнішнього пристрою. У більшості нових моделей персональних комп'ютерів вісім ліній даних мають можливість передавати дані в обох напрямках, але в багатьох старих комп'ютерах принтерний порт може тільки посилати дані. Для принтерного порту використовується 25-контактний рознім (DB-25F), виводи якого мають таке призначення.

Таблиця 5.4 – Контакти LPT порту

Вихід 1 - вихід строб-сигналу	Вихід 14 - вихід —AutoFDxt
Вихід 2 - вхід/вихід даних D0	Вихід 15 - вхід —Error
Вихід 3 - вхід/вихід даних D1	Вихід 16 - вихід —INIT
Вихід 4 - вхід/вихід даних D2	Вихід 17 - вихід —SLCTIN
Вихід 5 - вхід/вихід даних D3	Вихід 18 - «земля»
Вихід 6 - вхід/вихід даних D4	Вихід 19 - «земля»
Вихід 7 - вхід/вихід даних D5	Вихід 20 - «земля»
Вихід 8 - вхід/вихід даних D6	Вихід 21 - «земля»
Вихід 9 - вхід/вихід даних D7	Вихід 22 - «земля»
Вихід 10 - вхід -Ack	Вихід 23 - «земля»
Вихід 11 - вхід Busy	Вихід 24 - «земля»
Вихід 12 - вхід PE	Вихід 25 - «земля»
Вихід 13 - вхід SLCT	

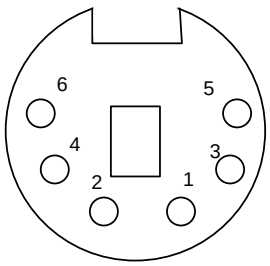
При використанні паралельного порту існує проблема забезпечення тимчасових параметрів. У первісних моделях IBM PC (і навіть у PC/AT) час читання і записування для цього порту було передбачено — наприклад, записування продовжувалося не менше 125нс. У нових комп'ютерах, що працюють значно швидше, читання/записування вмісту регістра може тривати менше 20 нс. Сучасні комп'ютери будуть поводитися так само як старі моделі, якщо їхній паралельний порт підімкнутий до шини ISA, а не використовує інтерфейс, реалізований за допомогою спеціалізованих замовлених мікросхем (ASIC).

Якщо використовуються функції BIOS, то тимчасові параметри будуть підтримуватися на рівні стандартних оригінальних значень. Але в цьому випадку потрібно використання сигналів підтвердження готовності.

Перш ніж використовувати паралельний порт комп'ютера, необхідно уважно проаналізувати тимчасові параметри вашого додатка і переконатися, що вони будуть реалізованими при підімкненні різних моделей комп'ютерів. Бажано уникати застосування паралельного порту, якщо це можливо — принтерний порт порушує перше з даних вище правил вибору інтерфейсу.

Передавання даних і команд через порт клавіатури/миші краще залишити для дуже специфічних додатків таких, як підключення зовнішньої

клавіатури чи спеціальних пристроїв введення. Вихідні дані мікроконтролера повинні в цьому випадку являти собою стандартні скан-коди. Далі буде розглянутий інтерфейс комп'ютера PC/AT із клавіатурою, що здатна посилати і приймати дані (рис. 5.26). Скан-коди клавіатури передаються синхронно, з використанням такого ж формату, як при асинхронному передаванні даних (рис. 5.27). Дані можуть передаватися в обох напрямках, хоча очевидно, що дані від клавіатури надходять набагато частіше.



1 – Введення-виведення даних
2 – Не підімкнутий
3 – "Земля"
4 – U_{cc}
5 – Синхросигнал введення-виведення
6 – Зарезервовано

Рисунок 5.26 – Рознім для підімкнення клавіатури в комп`ютері PC/AT

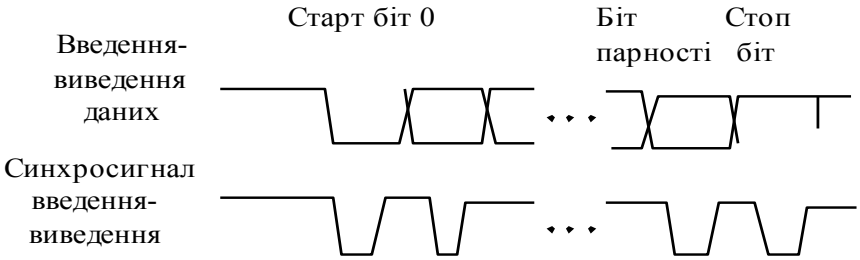


Рисунок 5.27 – Часові діаграми сигналів клавіатури PC/AT

При розробці мікроконтролерного інтерфейсу між комп'ютером і клавіатурою необхідно здійснити керування синхросигналами і сигналами даних для обох пристроїв. Якщо дані послані, то мікроконтролер повинен забезпечити їхнє передавання між пристроями. Якщо дані посилаються від мікроконтролера до комп'ютера, а в цей час із клавіатури надходять нові дані, то вони повинні зберігатися в мікроконтролері, а потім передаватися комп'ютеру.

Вище говорилося про скан-коди, що надходять від клавіатури. У комп'ютері дані, що посилаються у вигляді скан-кодів, визначають позицію ключа на клавіатурі. При одержанні даних від клавіатури необхідно стежити за станом клавіш Shift, Alt, Ctrl, CapsLock, NumLock і ScrollLock. Від цього залежить як буде інтерпретуватися комп'ютером скан-код, посланий із клавіатури. Це також справедливо для команд, що посилаються комп'ютером клавіатурі.

При відпусканні ключа перед скан-кодом посилається значення 0x0. Цей код указує комп'ютеру, що клавіша більше не натиснута, і не треба продовжувати реалізацію функції повторення, якщо така містилася в програмі, яка виконується комп'ютером. Комп'ютер може послати клавіатурі такі команди:

Таблиця 5.5 – Скан-коди комп’ютера PC/AT

F1 - 0x005 F2 - 0x006 " ‘ "- 0x00E "1"- 0x016 "2"- 0x0l "3"- 0x026 "4"- 0x025 "5"- 0x02E "6"- 0x036 "7"- 0x03D "8"- 0x03E "9"- 0x046 "0"- 0x045 "-" - 0x04E "="_ 0x055 BS - 0x05D ESC- 0x066 NL- 0x076 SL - 0x077 SR - 0x084	F3 - 0x004 F4 - 0x00C Tab - 0x00 D "Q"- 0x015 "W"-0x01D "E" - 0x024 "R"-0x02D "T" - 0x02C "Y" - 0x035 "U"-0x03C "I" -0x043 "0"- 0x044 "P" - 0x04D "[" - 0x054 "]" - 0x05B Hme- 0x0бс Up - 0x075 Pg- 0x07D PS - 0x07C	F5 - 0x003 F6 - 0x00 B Ctr -0x014 "A"-0x01C "S"-0x01B "D"-0x023 "F" - 0x028 "G"-0x034 "H"- 0x033 "J" -0x038 "ДО"-0x042 "L" - 0x04B ";" - 0x04C " " " - 0x052 Ent - 0x05A Lft - 0x0бв 5 - 0x073 Rht - 0x074 - - 0x07 B	F7 - 0x083 F8 - 0x00A LSH - 0x012 "Z" – 0x0l "X" - 0x022 "З" - 0x021 "V" - 0x02A "У" - 0x032 "N"-0x031 "M"-0x03A "," -0x041 "." -0x049 "/" - 0x04A RSH- 0x059 End - 0x069 Dwn- 0x072 Pd- 0x07A + - 0x079	F9 - 0x001 F10 - 0x009 Alt - 0x011 " " - 0x029 CPL- 0x058 Ins - 0x070 Del- 0x071
BS — "Backspace" NL- "Num Lock" SL - "Scroll Lock" SR — "System Request" CPL— "Caps Lock" Dwn-"стрілка вниз"	Hme— "Home" PgU-"Page Up" PS — "Print Screen" Ent — "Enter" RSH- правий "Shift" Pg- "PageDown"	Lft - "стрілка вліво" 5 - "5" на клавіатурі Rht - "стрілка вправо" - —"-" на клавіатурі LSH - лівий "Shift" Ctl -"Ctrl"		

0x0FF – Повторний запуск мікроконтролера клавіатури.
0x0FE – Запит на повторне посилання останнього символа.
0x0F7-0x0FD – «Порожні» команди NOP (ці команди ігноруються).
0x0F6 – Встановити початкові значення за замовчуванням (після вмикання живлення).
0x0F5 – Встановити початкові значення за замовчуванням, але заборонити сканування клавіатури.
0x0F4 – Дозволити сканування клавіатури.
0x0F3 – Встановити швидкість сканування. Наступний символ визначає швидкість сканування стану клавіш.
0x0EF-0x0F2 – «Порожні» команди NOP. Ці команди ігноруються.
0x0EE – «Луна». Клавіатура повертає код 0x0EE
0x0ED – Встановити стан світлодіодних індикаторів. Наступний символ визначає стан індикаторів.

Після кожної з цих команд клавіатура видає код підтвердження «Ack» (0xFA).

Порт миші використовує такий же інтерфейс, але, звичайно, комп'ютер не передає по ньому дані. З цієї причини апаратні засоби комп'ютера не дозволяють передавати дані за допомогою даного інтерфейсу. Щоб уникнути помилок, необхідно пам'ятати, що при використанні цього порту дані повинні передаватися тільки в комп'ютер.

Останній з розглянутих інтерфейсів — шина ISA (Industry Standard Architecture). Це мікропроцесорна шина, що забезпечує доступ до багатьох команд процесора.

Таблиця 5.6 – Призначення контактів слота шини ISA

Зворотна сторона плати		**Монтажна сторона плати**	
B1:	«Земля»	A1:	-CHKCHK
B2:	RESDRV	A2:	SD7
B3:	+5V	A3:	SD6
B4:	IRQ2	A4:	SD5
B5:	-5V	A5:	SD4
B6:	DRQ2	A6:	SD3
B7:	-12V	A7:	SD2
B8:	-NOWS	A8:	SD1
B9:	+12V	A9:	SD0
B10:	«Земля»	A10:	CHRDY
B11:	-MEMW	A11:	AEN
B12:	-MEMR	A12:	SA19
B13:	-OW	A13:	SA18
B14:	-IOR	A14:	SA17
B15:	-DAC1	A15:	SA16
B16:	DRQ3	A16:	SA15
B17:	-DAC1	A17:	SA14
B18:	DRQ1	A18:	SA13
B19:	-REFRESH	A19:	SA12
B20:	BLCK	A20:	SA11
B21:	IRQ7	A21:	SA10
B22:	IRQ6	A22:	SA9
B23:	IRQ5	A23:	SA8
B24:	IRQ4	A24:	SA7
B25:	IRQ3	A25:	SA6
B26:	-DAC2	A26:	SA5
B27:	TC	A27:	SA4
B28:	ALE	A28:	SA3
B29:	+5V	A29:	SA2
B30:	OSC	A30:	SA1
B31:	«Земля»	A31:	SA0

У сучасних комп'ютерах застосовуються також інші шини, наприклад PCI і EISA, але шина ISA представляє найбільш традиційний інтерфейс і є досить простою для використання. Далі розглядаються сигнали

для реалізації 8-розрядного інтерфейсу, тому що 16-розрядна шина навряд чи буде використовуватися 8-розрядними мікроконтролерами.

Далі будуть розглянуті тільки можливості читання і записування даних для пристроїв, підімкнених до шини ISA. Якщо Вас цікавить більш детальне ознайомлення з реалізацією інтерфейсу мікроконтролерів за допомогою шини ISA, особливо якщо необхідно використовувати переривання чи прямий доступ до пам'яті (DMA), — скористайтеся посиланнями наприкінці книги.

Таблиця 5.7 – Сигнали шини ISA мають таке призначення:

Сигнал	Призначення сигналу
-CHKCHK	Перевірка каналу. Коли сигнал активний, реалізується немаскуєме переривання NMI
SD7-SD0	Системна шина даних
CHRDY	Готовність каналу. Коли сигнал активний, уводяться такти чекання обміну по шині
AEN	Індикатор активності DMA. Коли сигнал активний, DMA контролер працює в активному режимі
SA19-SA0	Системна шина адреси
-RESDRV	Системне скидання (Reset)
IRQx	Лінії запиту переривань. Активний рівень високий
DRQx	Лінії запиту DMA. Активний рівень низький
DACx	Лінії підтвердження DMA. Активний рівень низький
-MEMR	Читання пам'яті
-MEMW	Записування у пам'ять
-10R	Читання пристроїв введення-виведення
-I0W	Записування у пристрої введення-виведення
-REFRESH	Активний при регенерації динамічної пам'яті
-N0WS	Немає станів чекання. Операції введення-виведення виконуються без тактів чекання з боку процесора
TC	Закінчення DMA. Активний рівень сигналу вказує на завершення DMA
ALE	Дозволено приймання адреси. Активний, коли сигнали SA19-SA0 достовірні

Лінії адреси і даних використовуються при виконанні циклів читання і записування як показано на рис. 5.28 і 5.29.

Слід зазначити, що сигнал на лінії AEN активний, коли шиною керує контролер прямого доступу до пам'яті. Стан цієї лінії необхідно перевіряти при підімкненні мікроконтролера до шини ISA (рис. 5.30). Сигнал на лінії повинний мати низький рівень, щоб не відбувалося читання чи записування неправильних даних.

Якщо при налагодженні мікроконтролера чи будь-якого іншого пристрою, підімкненого до шини ISA, використовується логічний аналізатор чи осцилограф, то для їхнього запуску використовується сигнал ALE чи вихідний сигнал адресного дешифратора.

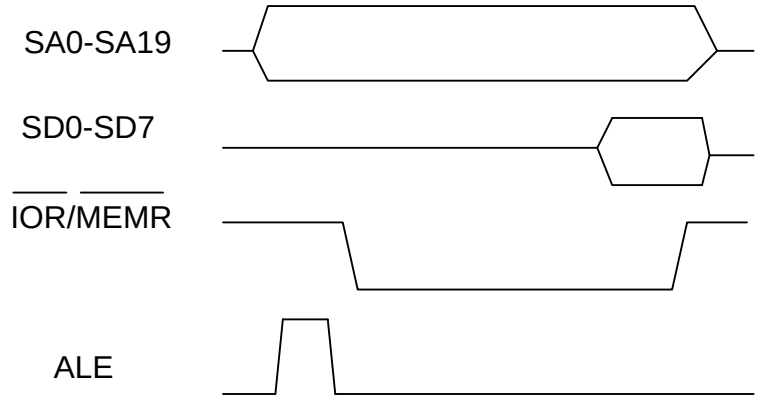


Рисунок 5.28 – Сигнали читання даних по шині ISA

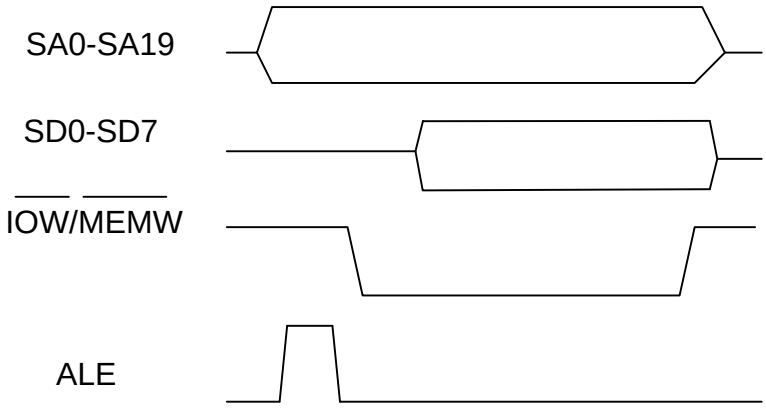


Рисунок 5.29 – Сигнали записування по шині ISA

При використанні мікроконтролера як адаптера на шині ISA дуже важливо, де розміщується його адреса: у просторі пристроїв виведення чи в просторі пам’яті. Архітектура мікропроцесорів Intel x86 має окремий адресний простір для пристроїв введення-виведення (їхнє відображення на пам’яті не використовується). При 8-розрядній шині ISA доступний один Мбайт пам’яті чи адреси 1024 пристроїв введення-виведення. Доступний адресний простір введення-виведення для цих мікропроцесорів має розмірність 64 Кбайт, але при розробці первісної специфікації на комп’ютери IBM PC 16-розрядна адреса пристроїв введення-виведення була зменшена на 2 біти, тому доступними виявилися тільки 4096 адрес. Коли проектувалися ці комп’ютери, здавалося, що 1024 різних адрес більш ніж достатньо для пристроїв введення-виведення, тому 4 старших біти адреси ігноруються адресним дешифратором.

Звичайно, адаптери шини ISA проектуються, щоб працювати з адресним простором пристроїв введення-виведення. Однак більшість з цих 4096 адрес вже визначені для різних стандартних пристроїв. Це означає, що ви будете мати труднощі з виділенням для мікроконтролера вільної адреси, що під’єднується до шини, і не використовується іншим пристроєм. Така одна з основних причин, за якою не рекомендується проектувати мікроконтролерні пристрої, що під’єднуються безпосередньо до шини ISA.

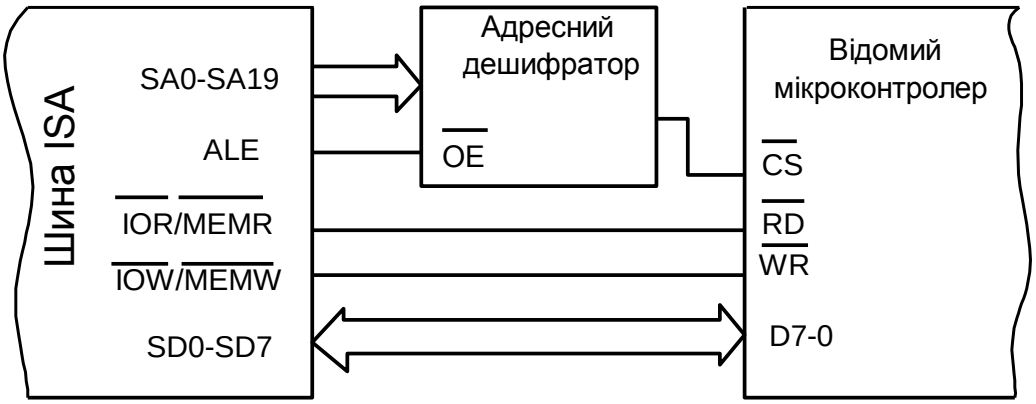


Рисунок 5.30 – Мікроконтролер у якості відомого пристрою на шині ISA

Може виникнути питання, яким же чином потрібно здійснити інтерфейс із персональним комп'ютером. В наш час на ринку з'явилися мікроконтролери, що мають інтерфейс із шиною USB («universal serial bus»), що не згадувалася в цій книзі. Популярність цих мікроконтролерів буде рости при розширенні використання шини USB у комп'ютерах і її програмній підтримці різними операційними системами. Шина USB працює подібно CAN інтерфейсу. Як і CAN, вона призначена для реалізації малих локальних мереж, забезпечуючи при цьому швидке передавання великих обсягів даних.

5.5 Основні схемні рішення інтерфейсів

5.5.1 Паралельні виходи

Одним з найпростіших, але одночасно і найважливіших і частих застосувань паралельних портів мікроконтролера можна назвати керування різними пристроями. У даному випадку мова йтиме про керування типу «увімкнути/вимкнути».

Як виходи паралельні порти можуть застосовуватися для керування реле, семісторами, світлодіодними індикаторами і т.д.

Керування світлодіодами чи оптронами

Керування світлодіодами — найпростіше, що може зустрітися при виготовленні схем на мікроконтролерах. Як відомо, світлодіоди споживають досить маленький струм — у залежності від типу світлодіода цей струм може складати від 3 до 20 мА. Робоча напруга світлодіодів складає приблизно від 1,5 до 4 В.

Оскільки струм, який мікроконтролери сімейства AVR можуть віддавати при напрузі «логічний нуль» на вихідній лінії, може досягати 20 мА, можна керувати світодіодом просто підімкнувши його до вихідної лінії порту послідовно з обмежувальним резистором. Другий вивід цього ланцюга варто приєднати до позитивної лінії живлення.

Варто звернути увагу на те, що увімкнутий саме в такий спосіб мікроконтролер, при напрузі «логічна одиниця» може віддавати набагато

менший струм. А виходить, його не можна буде застосувати для керування світлодіодом безпосередньо. Більш докладно можна ознайомитися з величинами допустимих струмів, скориставшись фірмовою документацією на мікроконтролери.

Керувати світлодіодом дуже просто, тому що один його вивід підімкнений до позитивного проводу живлення для того, щоб він став світитися (тобто спад напруги на ньому став достатнім для засвічування), потрібно сформувати на другому виводі ланцюга зі світлодіодом напругу низького рівня «0». Простіше кажучи, для того, щоб засвітити світлодіод, треба записати у вихідний порт значення «0». Щоб погасити — записати «1».

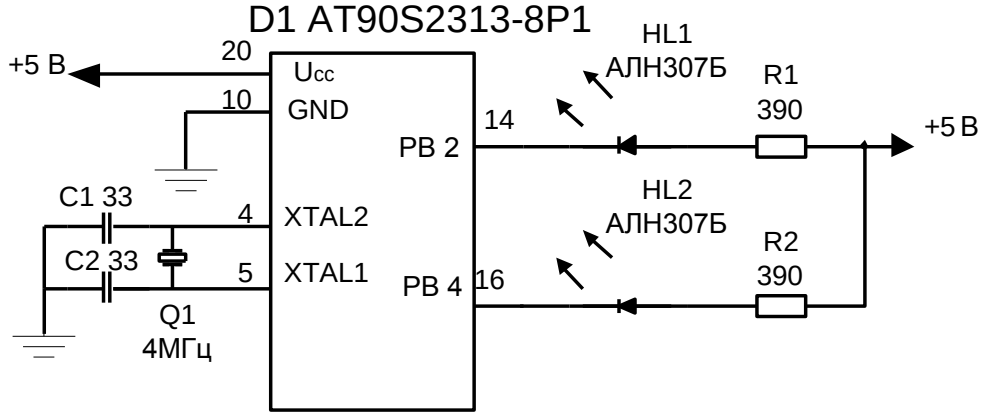


Рисунок 5.31 – Найпростіша схема для керування двома світлодіодами

У такий же спосіб можна приєднати і більшу кількість світлодіодів — стільки, скільки всіх ліній портів введення-виведення. Однак варто мати на увазі дуже важливий факт — хоча кожен вихід мікроконтролера може керувати навантаженням до 20 мА, загальний споживаний струм від усіх ліній портів введення-виведення не повинний перевищити визначеного значення. У залежності від типу корпусу мікроконтролера і числа його ліній портів введення-виведення його величина може бути різною. Точно з цими значеннями струму можна ознайомитися у фірмовій документації на мікроконтролер.

Наприклад, для мікроконтролера AT90S2313 є такі обмеження: сумарний струм навантаження при «0» на виходах не повинний перевищувати 200 мА, причому сумарний струм ліній D0—D5 не більше 100 мА і сумарний струм ліній D7 і D6 також не повинний перевищувати 100 мА. Легко побачити, що при навантаженні всіх виходів струмом 20 мА, можна перевищити допустимий струм, що може зашкодити мікросхемі.

Аналогічно можна керувати оптопарами, адже вони являють собою розміщені в одному корпусі один навпроти одного світлодіод і фоточутливий елемент – фоторезистор, фототранзистор, і т.д. Наприклад, використовуючи оптопару з вбудованим фотосемістором можна керувати високовольтним навантаженням. При цьому досягаються такі важливі цілі, як галь-

ванічна розв’язка високовольтних ланцюгів і схеми керування, відсутність іскрового проміжку і т.д.

Керування реле

Для живлення обмотки реле потрібен струм, що перевищує 20 мА, тому безпосередньо підімкнути до мікроконтролера його не можна. Для керування реле, можна застосовувати найпростіший підсилювач — транзисторний ключ. На рис. 5.32 показаний приклад схеми з реле. Зверніть увагу на наявність діода, підімкненого паралельно обмотці реле — він потрібний для захисту схеми від ЕРС самоіндукції, що з’являється в процесі комутації обмотки.

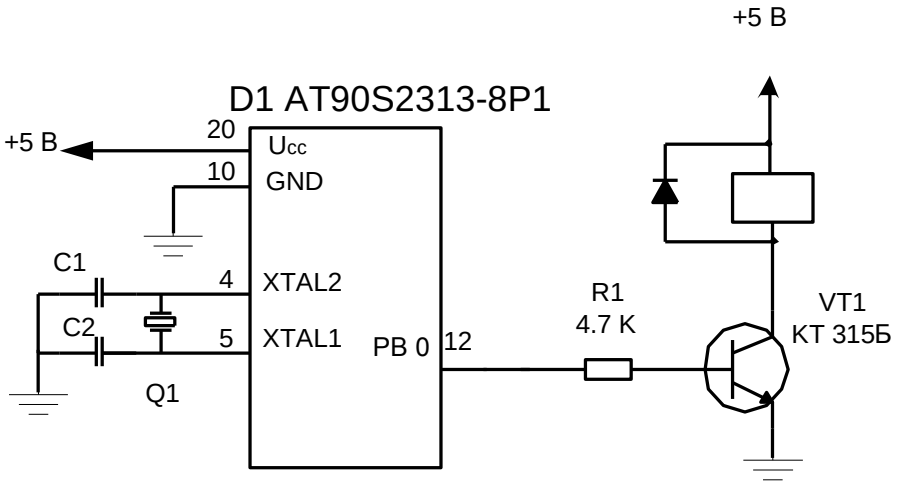


Рисунок 5.32 – Приклад схеми з реле

Аналогічно можна вмикати не реле, а яке-небудь інше навантаження, наприклад, лампу розжарювання і т.д.

У випадку, якщо необхідно керувати великим числом реле, чи інших потужних навантажень, зручно застосовувати мікросхеми ULN2003 чи ULN2803. Ці мікросхеми містять відповідно, 7 і 8 транзисторних ключів на складених транзисторах (схема Дарлінгтона). Вони дозволяють керувати навантаженням до 500 мА при напрузі до 50 В. При цьому входи цих мікросхем можуть бути підімкненні безпосередньо до ліній портів введення-виведення мікроконтролера. Всередині мікросхем уже є вбудований захисний діод, який можна вмикати чи вимикати, здійснюючи зовнішні з’єднання. На рис. 5.33 показаний приклад схеми з використанням мікросхеми ULN2003.

Для вмикання навантаження варто сформувати на відповідному виводі мікроконтролера рівень «1». При цьому струм, споживаний від виводу порту мікроконтролера, не перевищує допустимий, одночасно здійснюючи керування достатньо великим навантаженням.

Керування світлодіодними цифровими індикаторами

Оскільки світлодіодні цифрові індикатори являють собою набір світлодіодів спеціальної форми, і розташовані так, щоб при засвічуванні різних їхніх комбінацій виходили цифри, то керування ними принципово не відрі-

зняється від керування окремими світлодіодами. На рис. 5.34 зображений приклад схеми керування семисегментним світлодіодним індикатором.

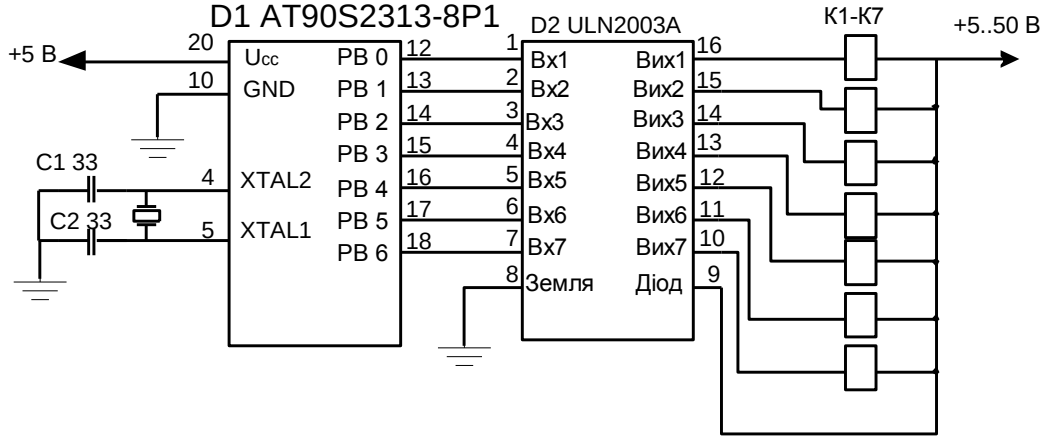


Рисунок 5.33 – Застосування мікросхеми ULN2003

Легко побачити, що при потребі керувати великим числом індикаторів, кількості виводів портів введення-виведення буде недостатньо. Для подолання цієї перешкоди застосовується динамічна індикація. На рис. 5.35 показаний приклад схеми динамічної індикації. Ідея, що лежить в основі роботи цієї схеми дуже проста — людське око досить інерційне, тому можна засвічувати не всі індикатори одночасно, а тільки один з них, потім через короткий час інший і т. д. Оскільки перемикання індикаторів відбувається досить швидко, людині здається, що всі індикатори світяться.

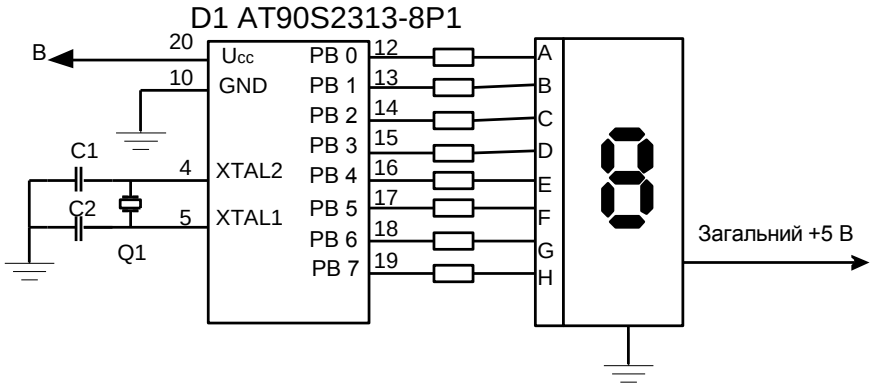


Рисунок 5.34 – Керування цифровим індикатором

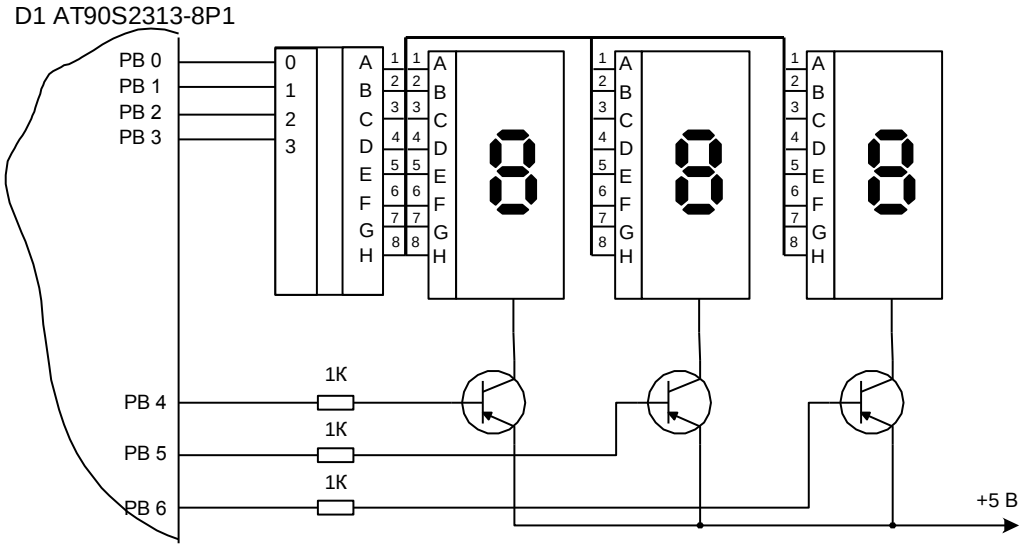


Рисунок 5.35 – Динамічна індикація

5.5.2 Паралельні входи

Паралельні входи, звичайно, застосовуються для контролю стану різних комутаційних елементів: кнопок, перемикачів, блоку контактів і т.д. Також можна перевіряти стан деяких видів датчиків, але при цьому може знадобитися додаткова схема, що перетворить стан датчика до логічних рівнів (наприклад, рівень води в баці нижче чи вище певної висоти і т.д.). Дуже часто входи паралельних портів застосовуються для контролю стану кнопок керування пристроєм.

Кнопки і перемикачі

Перевіряти стан кнопок чи перемикачів досить просто. Досить під'єднати, наприклад, кнопку одним виводом до загального проводу, а іншим — до вхідної лінії порту введення-виведення, налаштованої для роботи в режимі читання. Також ця лінія повинна бути з'єднана через резистор опором приблизно 4,7... 100 КОм з проводом «плюс» живлення. При більшому опорі сумарний споживаний струм менший. При розімкнутих контактах, на відповідному виводі мікроконтролера буде «1», при замиканні контактів — «0».

Усі механічні вимикачі мають недолік — при роботі з ними спостерігається, так зване, деренчання контактів, коли при натисканні на кнопку відбувається багато замикань і розмикань контактів через те, що вони, як правило, пружинять. Тривалість періоду деречання залежить від якості контактів, і звичайно, складає від 10 до 100 мс. Боротися з цим ефектом простіше програмним способом. На рис. 5.36 показані графіки, що ілюструють деренчання контактів, а на рис. 5.37 приведена найпростіша схема з кнопкою.

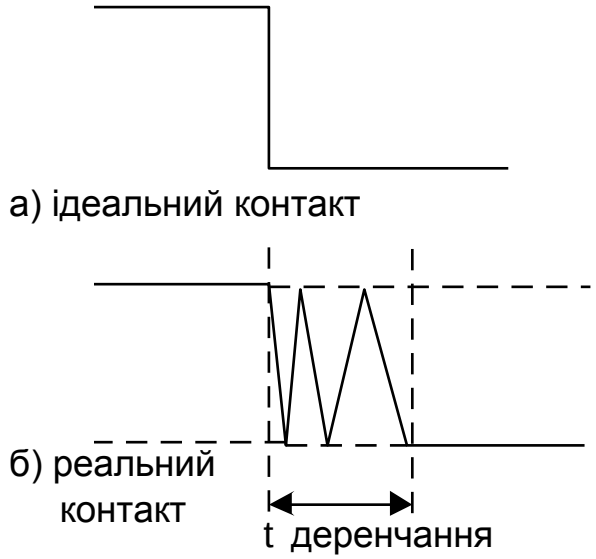


Рисунок 5.36 – Явище „деренчання” контактів

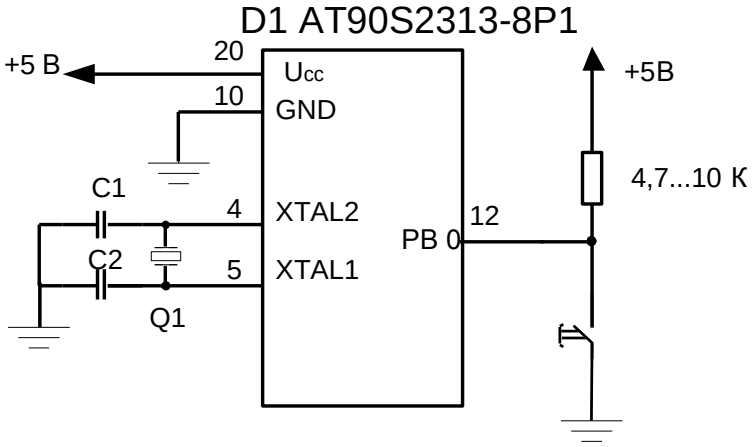


Рисунок 5.37 – Підімкнення кнопки до мікроконтролера

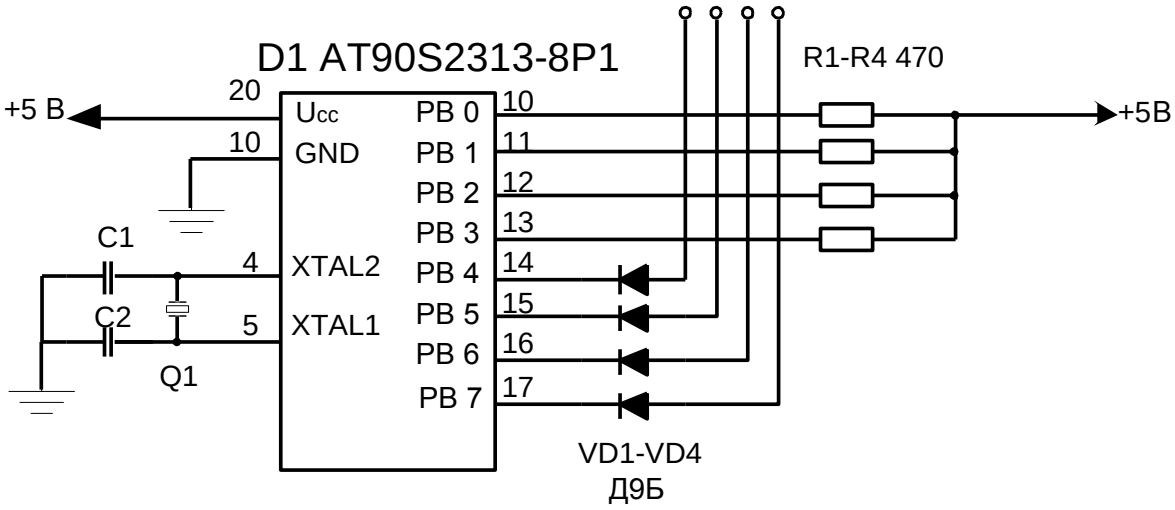


Рисунок 5.38 – Матричну схему з’єднання клавіатури з мікроконтролером

6 ПРИКЛАДИ РЕАЛІЗАЦІЇ МІКРОКОНТРОЛЕРНИХ ПРИСТРОЇВ

6.1 Мікропроцесорний частотомір

При побудові приладів часто виникає необхідність у реалізації функції вимірювання частотночасових параметрів сигналів (період, частота).

При безпосередньому (прямому) вимірюванні частоти періодичного сигналу найвагомішими є дві складові похибки – міри і порівняння. Похибка міри характеризується нестабільністю частоти кварцового генератора. Ця складова похибки може бути відчутною при вимірюванні дуже високих частот. Похибка порівняння характеризується, головним чином, похибкою квантування $\delta_к$. При вимірюванні низьких частот похибка квантування є визначальною складовою похибки вимірювання.

Наприклад, якщо вимірюється частота $f_x = 10$ Гц при $T_0 = 1$ с, то максимальна похибка квантування $\delta_{кч} = \frac{100\%}{f_x T_0} = \frac{100\%}{10 \cdot 1} = 10\%$, що недопустимо.

Таким чином, через великі похибки квантування низькі частоти безпосередньо вимірюються цифровим частотоміром із невисокою точністю. Тому вирішення проблеми зменшення впливу похибки квантування на результати вимірювання завжди було одним із важливих напрямів розробки цифрової частотовимірювальної техніки. Перед тим, як розглядати мікропроцесорний частотомір, який радикально вирішує вказану проблему, зупинимось на чотирьох способах зменшення похибки квантування при вимірюванні частоти.

1. Збільшення тривалості еталонного часового інтервалу Т, тобто часу вимірювання. Але можливості такого способу обмежені, оскільки для одержання малої похибки квантування (наприклад, $\delta_к$ =0,01%; $f_x = 10$ Гц) потрібний дуже великий час вимірювання:

$$T_0 = \frac{100\%}{\delta_{кч} f_x} = \frac{100\%}{0{,}01 \cdot 10} = 1000c. \qquad (6.1)$$

2. Збільшення числа імпульсів, що квантують еталонний часовий інтервал T_0 , який досягається множенням вимірюваної частоти f_x. Виконання даного способу поєднано із застосуванням додаткового блока помножувача частоти, що ускладнює і підвищує вартість апаратної частини.

3. Врахування випадкової природи похибки квантування. Припускається проведення багаторазових вимірювань і усереднення їх результатів. Це ефективний шлях зменшення впливу випадкової похибки на результат вимірювання.

4. Безпосереднє вимірювання періоду досліджуваного сигналу з наступним обчисленням частоти $f_x = 1/T_x$. Цей шлях дозволяє різко зменшити похибку квантування при вимірюванні низьких частот.

Щоб побачити ефект, який досягається, скористаємось наведеним раніше прикладом. Перейдемо до вимірювання періоду.

Частота f_x = 10Гц. Відповідний період T_x = 0,1 с. Сформуємо стробуючнй імпульс тривалістю, що дорівнює періоду T_x і проквантуємо його імпульсами, частота проходження яких f_0 = 10 МГц (що, звичайно, має місце в цифрових частотомірах). У цьому разі похибка квантування

$$\delta_{kn} = \frac{100\%}{T_x f_0} = \frac{100\%}{0{,}1 \cdot 10^7} = 10^{-4}\%. \qquad (6.2)$$

Можна зробити висновок, що непряме вимірювання частоти $f_x = 1/T_x$ у даному разі дозволило різко підвищити точність порівняно з прямим вимірюванням частоти: похибка квантування зменшилась у 100000 разів.

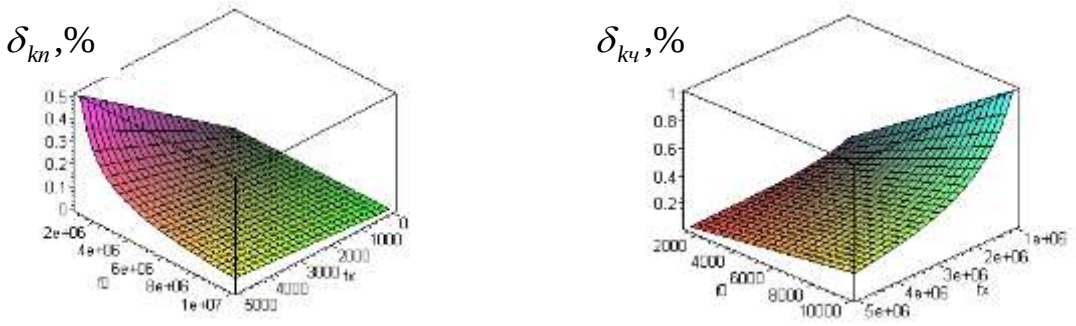
Однак при вимірюванні високих частот (наприклад, f_x = 106 Гц, T_0 = 1 с, f_0 = 10 Гц) похибка квантування цифрового частотоміра

$$\delta_{kч} = \frac{100\%}{T_0 f_x} = \frac{100\%}{0{,}1 \cdot 10^6} = 10^{-4}\%, \qquad (6.3)$$

а похибка квантування цифрового періодоміра надмірно зросте:

$$\delta_{kч} = \frac{100\% \cdot f_x}{f_0} = \frac{100\% \cdot 10^6}{10^7} = 10\%. \qquad (6.4)$$

Таким чином, при дослідженнях періодичних процесів у широкому діапазоні частот для досягнення заданої точності доцільно в діапазоні високих частот застосовувати цифровий частотомір, а в діапазоні низьких частот переходити до вимірювання періоду (рис. 6.1. а, б).



а) вимірювач періоду б) вимірювач частоти

Рисунок 6.1 – Визначення похибки мікропроцесорного частотоміра

Алгоритм роботи мікропроцесорного частотоміра, що дозволяє досліджувати періодичні процеси у широкому діапазоні, наведено на рис. 6.2. Спочатку настроюють програму на режим періодоміра. Встановлюють коефіцієнт подільника частоти К=1 і проводять вимірювання невідомої частоти f_x. Вимірювана частота подається на вхід аналогового компаратора. Аналоговий компаратор вибраний з тієї причини, що він має досить гнучку програмну обробку інформації: програміст може вибрати пряму програмну обробку або обробку за перериванням. За переднім фронтом імпульсу на вході AIN0 аналогового компаратора запускають таймер на рахування імпульсів f_0/K. За наступним переднім фронтом імпульсу на вході AIN0 аналогового компаратора (після закінчення періоду *Tx*) таймер мікроконтролера зупиняють і підраховують кількість імпульсів N_x.

Якщо $N_x = 0$ (частота f_0 недостатня для спрацювання періодоміра), то задають за допомогою таймера часовий інтервал T_0 (наприклад, T_0 = 1с) і переходять в режим вимірювання частоти. Частоту обчислюють за формулою

$$f_x = N_x / T_0 . \qquad (6.5)$$

Якщо $N_x \neq 0$, то перевіряють переповнення таймера мікроконтролера. При невиконанні цієї умови обчислюють частоту, інакше збільшують коефіцієнт подільника частоти $K = K + \Delta K$ і повертаються на вимірювання періоду. Частоту обчислюють за формулою

$$f_x = f_0 / (N_x \cdot K). \qquad (6.6)$$

Структурна схема мікропроцесорного частотоміра, яка дозволяє реалізувати наведений алгоритм, подана на рис. 6.3.

6.2 Мікропроцесорний фазометр

Принцип дії мікропроцесорного фазометра ґрунтується на перетворенні різниці фаз двох електричних сигналів у часовий інтервал t_x з його наступним квантуванням імпульсами опорної частоти f_0 (цифровий фазометр середніх значень). На рис. 6.4 наведено структурну схему мікропроцесорного фазометра, а на рис. 6.5 – алгоритм його роботи.

Основними елементами фазометра є блоки узгодження сигналів U_1 і U_2, мікроконтролер MCU, кварцевий резонатор частотою f_0 і пристрій індикації.

Перед початком вимірювань встановлюють час вимірювань T_B і коефіцієнт подільника частоти К=1.

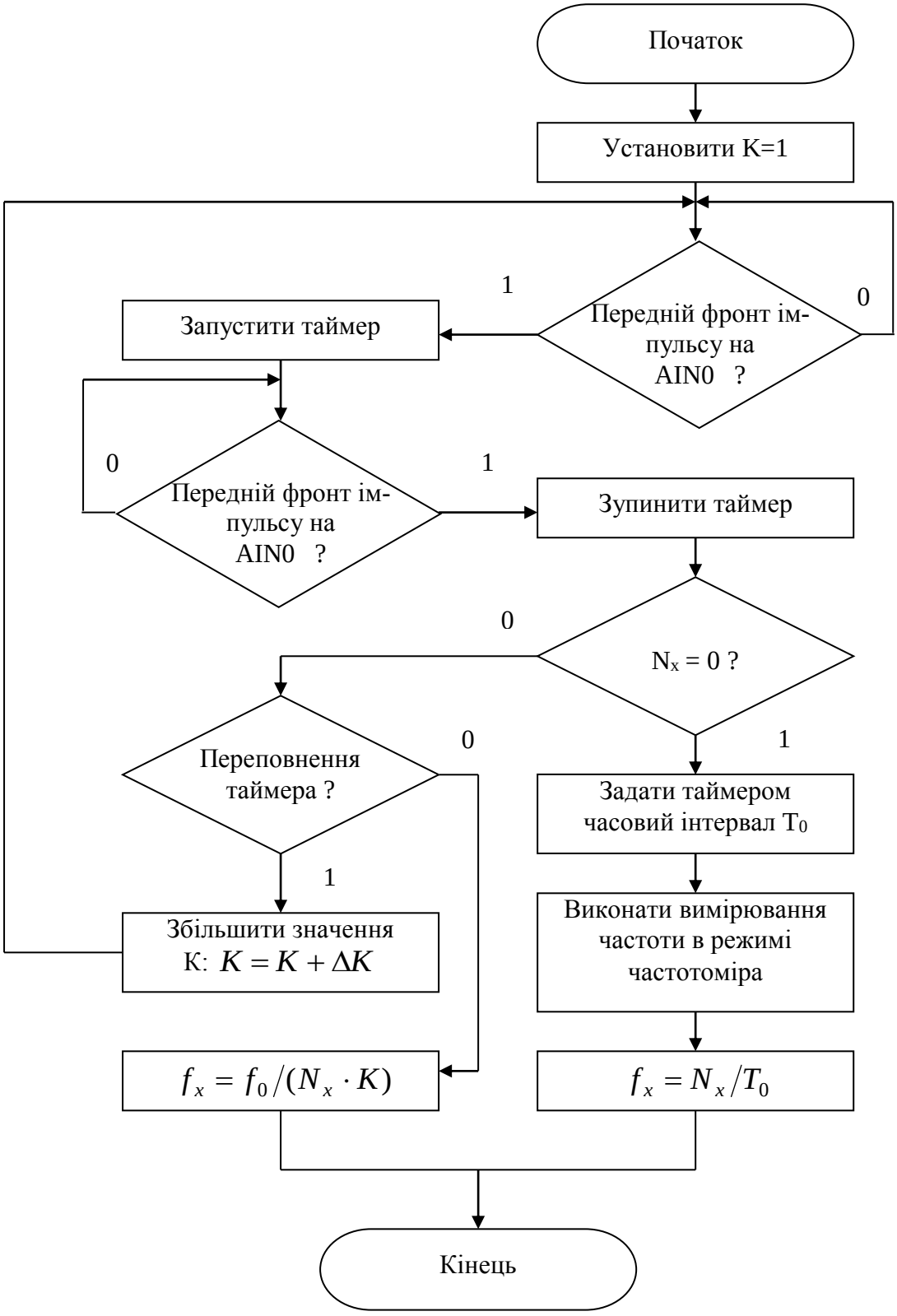


Рисунок 6.2 – Алгоритм виконання вимірювальної процедури у мікропроцесорному частотомірі

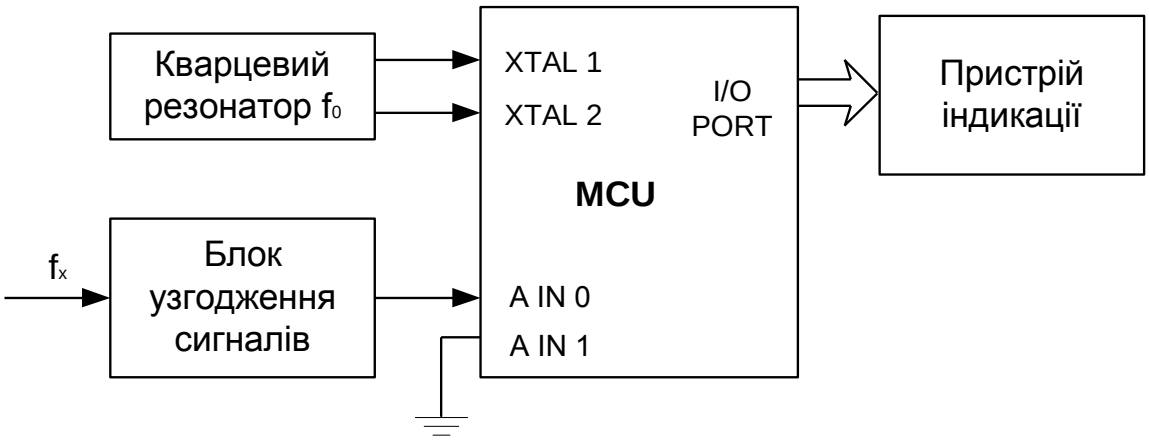


Рисунок 6.3 – Структурна схема мікропроцесорного частотоміра

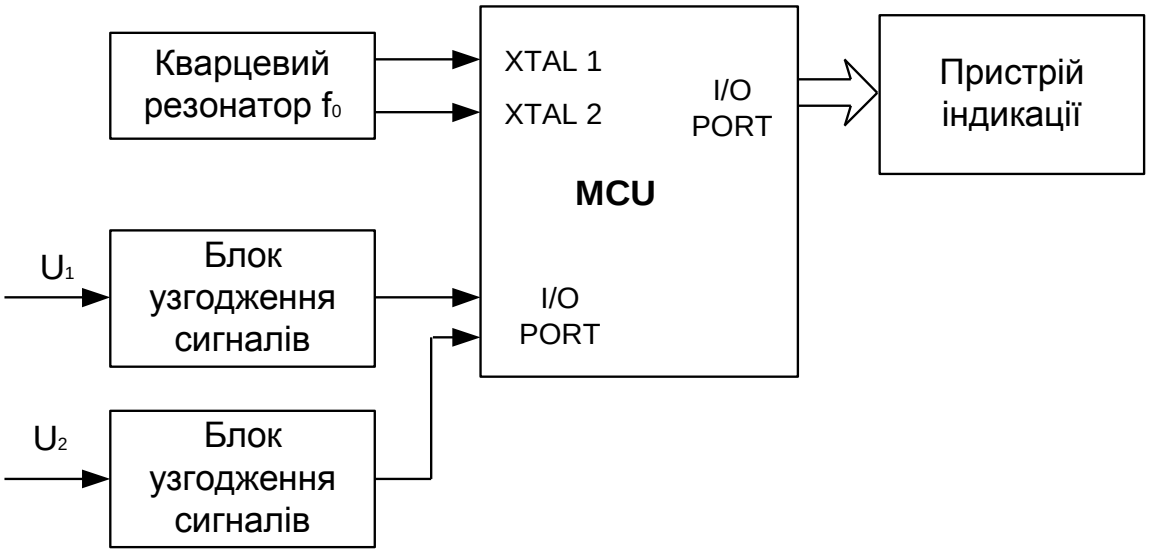


Рисунок 6.4 – Структурна схема мікропроцесорного фазометра

У момент переходу напруги U_1 через рівень нуля запускають таймер на підрахунок імпульсів f_0/K. Рахування проходить до переднього фронту імпульсу напруги U_2. При цьому робота таймера зупиняється і підраховують кількість імпульсів N_x. Кількість імпульсів усереднюється за проміжок часу

$$T_B = n \cdot f_x. \tag{6.7}$$

При виникненні переповнення таймера збільшують коефіцієнт подільника частоти $K = K + \Delta K$ і повертаються на початок вимірювань.

Фазовий зсув обчислюють за формулою

$$\varphi_x = \frac{2\pi N_x}{n}. \tag{6.8}$$

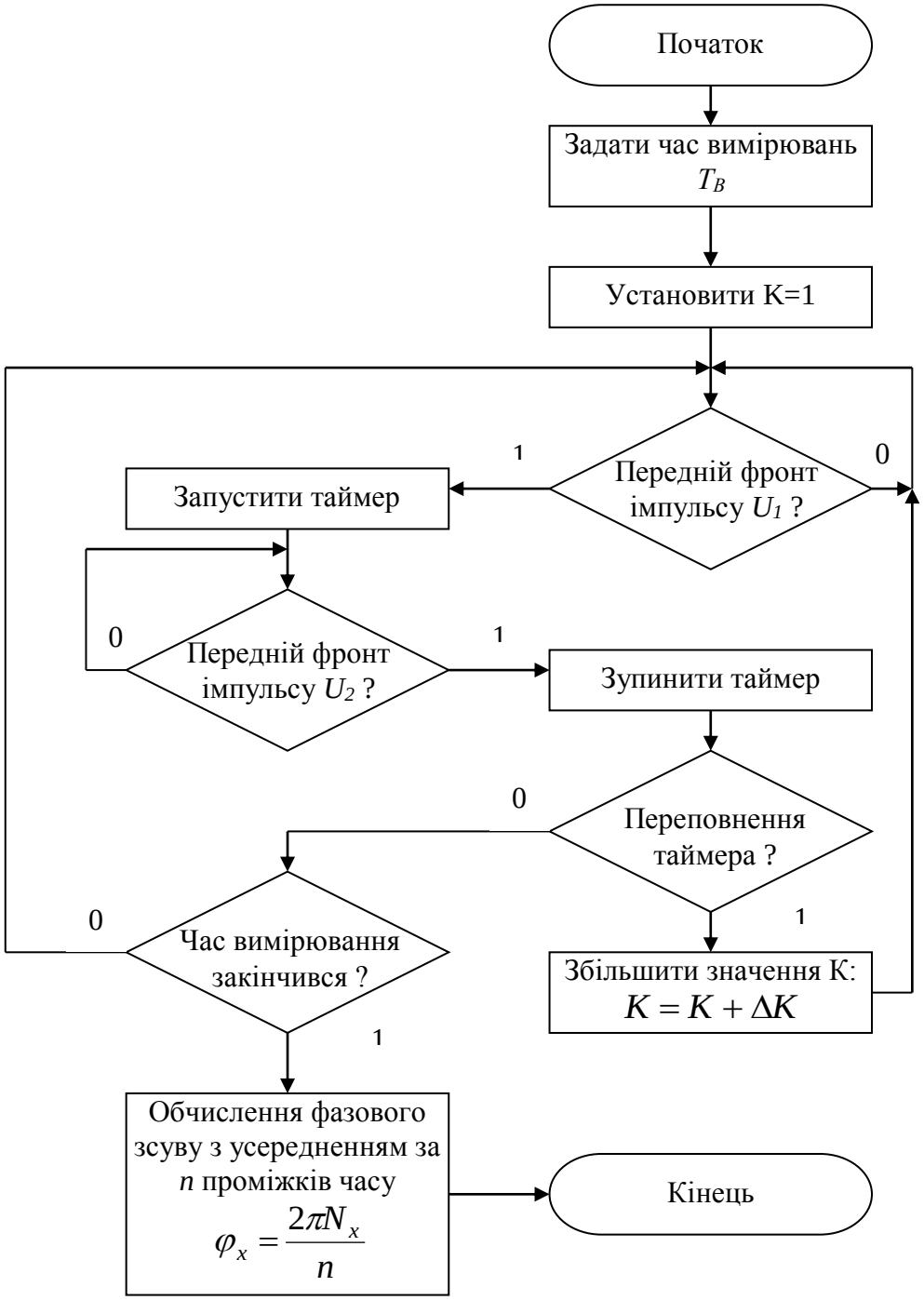


Рисунок 6.5 – Алгоритм виконання вимірювальної процедури у мікропроцесорному фазометрі

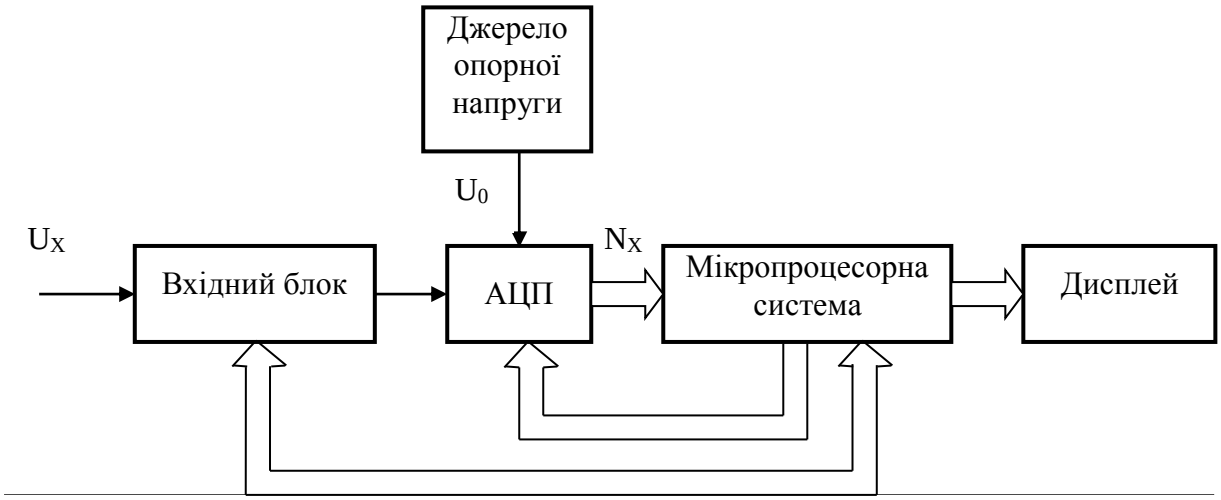
6.3 Мікропроцесорний вимірювач струму та напруги

Мікропроцесорні вольтметри та амперметри отримали широке поширення в техніці вимірювання на постійному і змінному струмі. У них найбільш повно втілені переваги мікропроцесорних вимірювальних приладів: подальше підвищення точності, розширення вимірювальних можливостей, спрощення і полегшення керування, можливість одержання різних

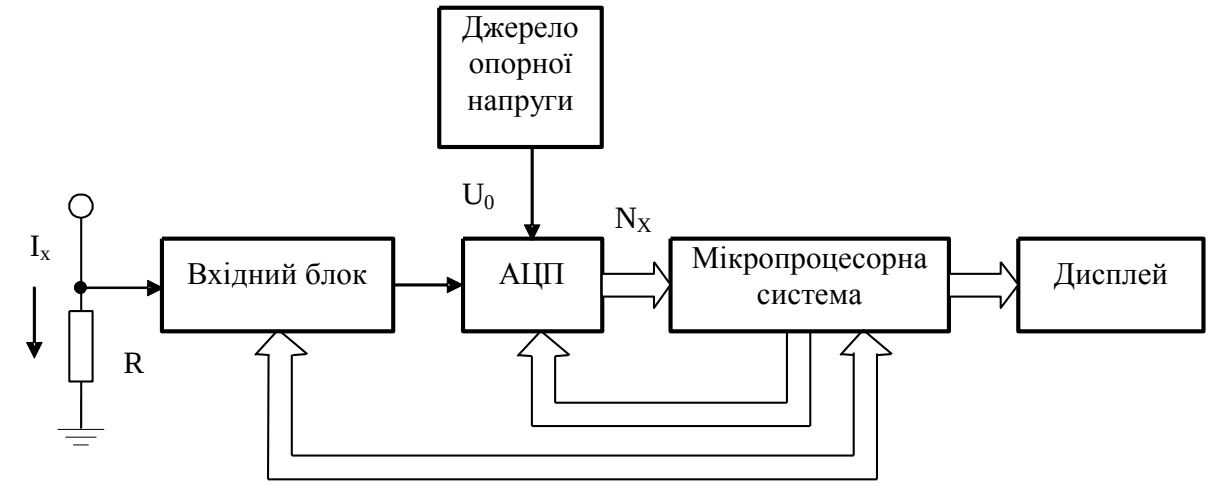
математичних функцій виміряних значень, статистична обробка результатів спостережень, самокалібрування і самодіагностика, підвищення надійності та економічності, можливість побудови програмувальних багатофункціональних приладів.

У самому загальному вигляді структурні схеми мікропроцесорних вольтметра та амперметра подані на рис. 6.6, а, б.

Вхідний блок містить аналогові перетворювачі — це насамперед атенюатор і підсилювач, але в деяких приладах до складу блока може входити також вимірювальний перетворювач напруги змінного струму в напругу постійного струму.



а) мікропроцесорний вольтметр



б) мікропроцесорний амперметр

Рисунок 6.6 – Структурні схеми мікропроцесорних вольтметра та амперметра

Як видно із рис. 6.6, амперметр відрізняється від вольтметра лише наявністю перетворювача струму в напругу (вимірювального шунта R). Тому в подальшому будемо розглядати мікропроцесорний вольтметр.

Обов’язковим вузлом кожного мікропроцесорного вольтметра є аналого-цифровий перетворювач (АЦП). Сучасна мікроелектронна техніка представляє розробнику вимірювальних приладів АЦП в інтегральному виконанні, що випускаються у вигляді інтегральних схем. Але не слід думати, що наявність АЦП і мікропроцесора цілком гарантує успішне створення приладу. Побудова мікропроцесорного вольтметра вимагає правильного вибору АЦП, що служить основним вимірювальним перетворювачем, раціонального вибору МП, здійснення їхнього з’єднання, визначення необхідних характеристик інших модулів МПС, розробки програмного забезпечення.

Джерело опорної напруги використовується для завдання високостабілізованої точної опорної напруги U_0, відносно якої проводяться вимірювання в АЦП. Рівняння перетворення АЦП має вигляд:

$$N_X = \frac{U_X}{U_0} \cdot \left(2^n - 1\right), \tag{6.9}$$

де n – розрядність АЦП.

Деякі мікроконтролери Atmel (Atmega603/103, AT90S4433, AT90S8535, ATtiny15) мають у своєму складі вбудований 10-розрядний АЦП із вхідним багатоканальним мультиплексором, що значно спрощує реалізацію вольтметрів. Для живлення АЦП в них використовуються два окремих виводи AVCC, AGND. Опорна напруга U_0 подається на вивід AREF.

Структурна схема вольтметра на основі мікроконтролерів ATMEL подана на рис. 6.7. АЦП може працювати в двох режимах: режимі однократного перетворення і режимі циклічного перетворення. В режимі однократного перетворення кожне перетворення ініціалізується програмою. В режимі циклічного перетворення АЦП здійснює вибірку і поновлення вмісту регістра даних АЦП неперервно. Вибір режиму здійснюється установленням певних бітів в регістрі керування АЦП. На рис. 6.8 подано алгоритм виконання вимірювальної процедури у мікропроцесорному вольтметрі.

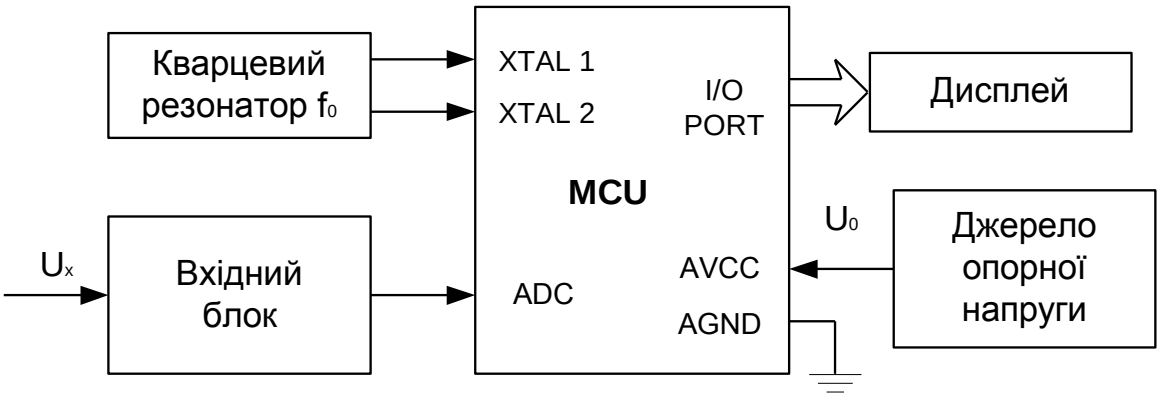


Рисунок 6.7 – Структурна схема мікропроцесорного вольтметра

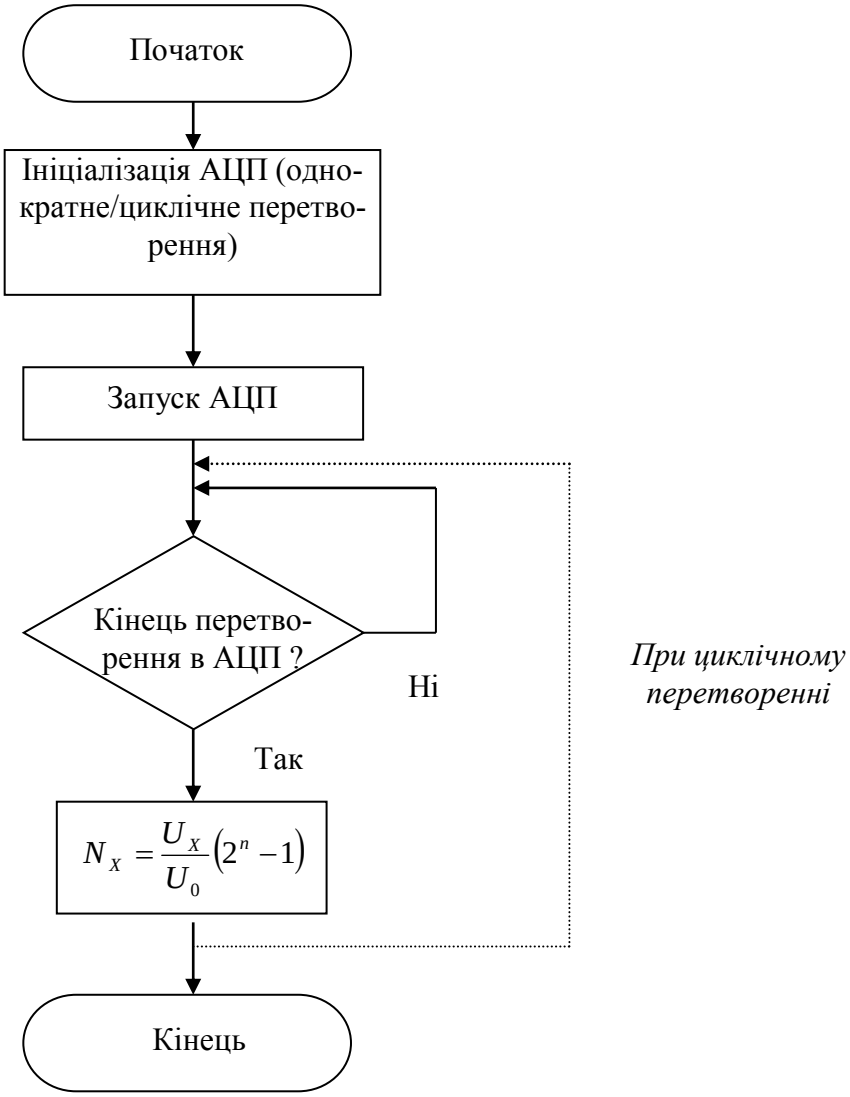


Рисунок 6.8 – Алгоритм виконання вимірювальної процедури у мікропроцесорному вольтметрі

6.4 Вимірювальний канал потужності

Для визначення потужності у колах постійного і змінного струму необхідно проводити вимірювання струму і напруги. Найбільш просто схемотехнічно це реалізується увімкненням сенсорів струму і напруги через АЦП до мікроконтролера, який проводить вибірку миттєвих значень струму і напруги в дискретні моменти часу. Точність вимірювань росте із збільшенням частоти дискретизації, що, в свою чергу, веде до ускладнення програмного забезпечення, оскільки обробка сигналів (фільтрація, усереднення) проводиться в реальному часі. Крім того, недоліком таких систем є їх відносно велика вартість.

Спростити алгоритми обробки інформації і понизити вартість на комплектацію дозволяє схема, наведена на рис. 6.9. Тут функцію вимірювання здійснює спеціалізована мікросхема вимірювача потужності. Мікро-

контролер виконує лише функцію частотоміра.

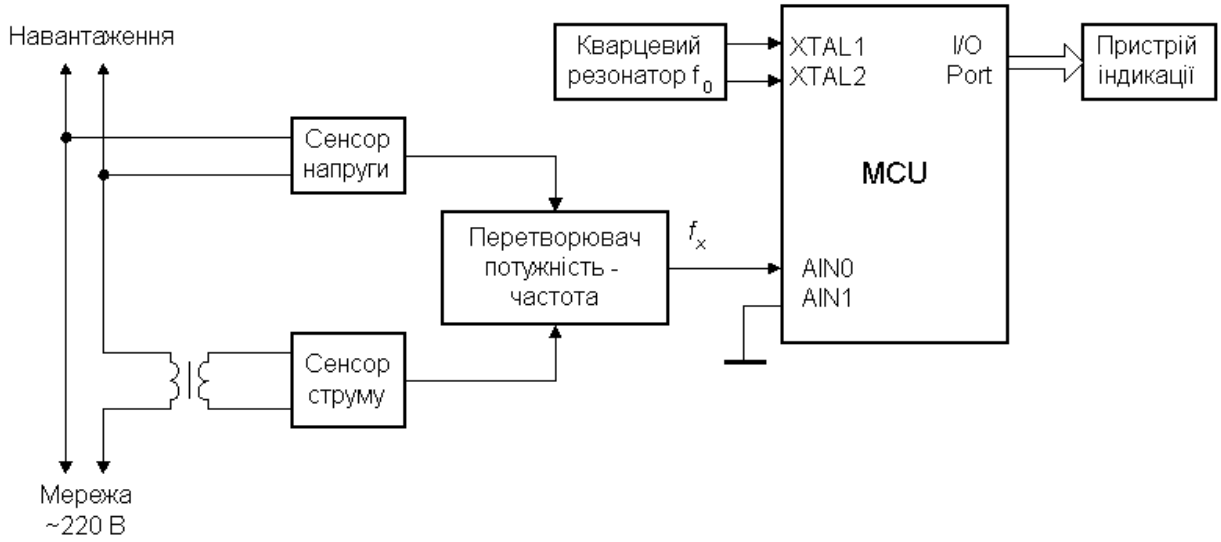


Рисунок 6.9 – Структурна схема мікропроцесорного вимірювача потужності

Прикладом перетворювача потужності є AD7750 – мікросхема перетворювача добутку напруг в частоту слідування імпульсів (Product to Frequency Converter) із похибкою менше 0,3%, розроблена фірмою Analog Devices. На рис. 6.10 наведено структурну схему мікросхеми AD7750.

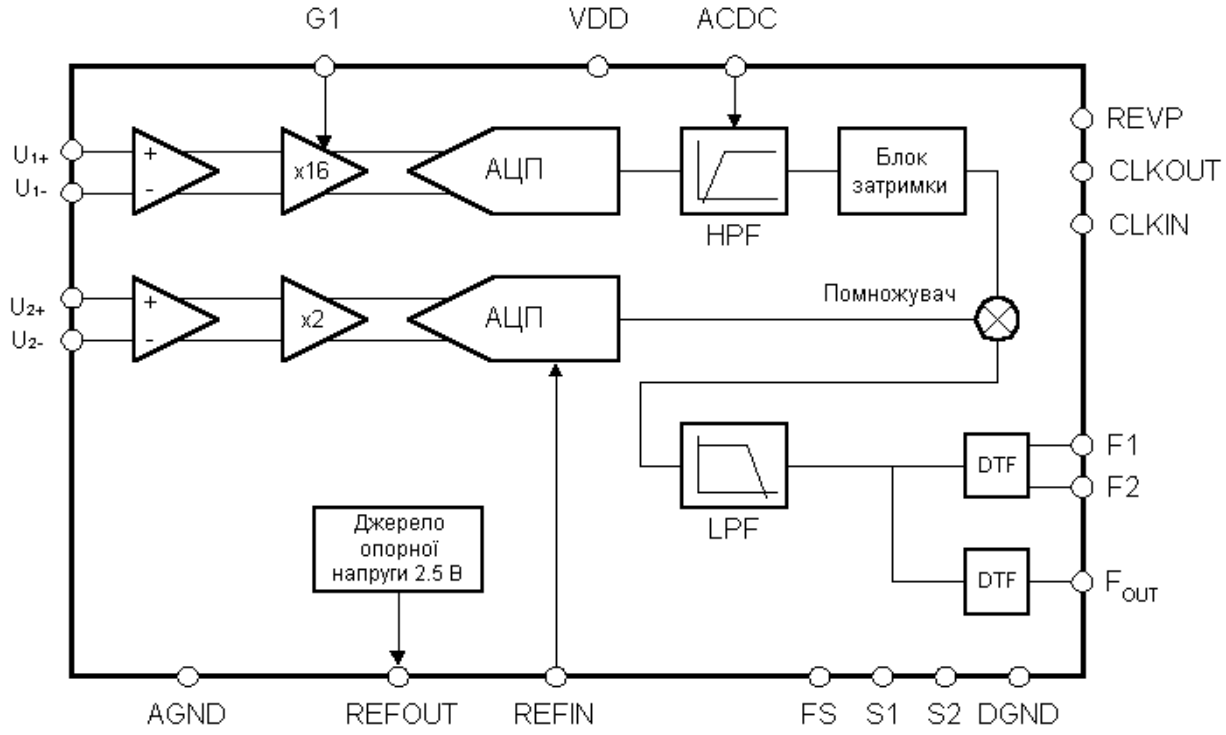


Рисунок 6.10 – Структурна схема мікросхеми AD7750

Входи каналів струму та напруги виконані диференціальними, кожний вхід розрахований на напругу не більше 1 В. Канал струму вміщує підсилювач із змінним коефіцієнтом підсилення (1 чи 16), канал напруги вміщує підсилювач з коефіцієнтом підсилення 2. Після підсилення обидва сигнали перетворюються в цифровий код АЦП і перемножуються, високочастотні складові відфільтровуються цифровим фільтром нижніх частот LPF, потім код потужності подається в перетворювач коду в частоту (Digital to Frequency Converter, DTF) і перетворюється в частотно-імпульсні низькочастотні сигнали F1, F2 для керування кроковим двигуном або лічильником імпульсів і високочастотний сигнал Fout для вимірювання струму чи напруги, а також перевірки лічильника.

Для зменшення похибки вимірювання потужності із-за наявності постійного струму в проводі нейтралі можна ввімкнути режим фільтрації струму фільтром верхніх частот HPF. Одночасно з фільтром вмикається лінія затримки, яка коригує фазову характеристику фільтра для мінімізації похибки при частоті мережі 50 Гц.

Режим роботи мікросхеми задають сигналами на входах FS, S1 і S2. Опис режимів наведено в табл. 6.1. Залежність вихідних сигналів від різниці фаз струму і напруги в двоквадрантному і чотириквадрантному режимах показано на рис. 6.11.

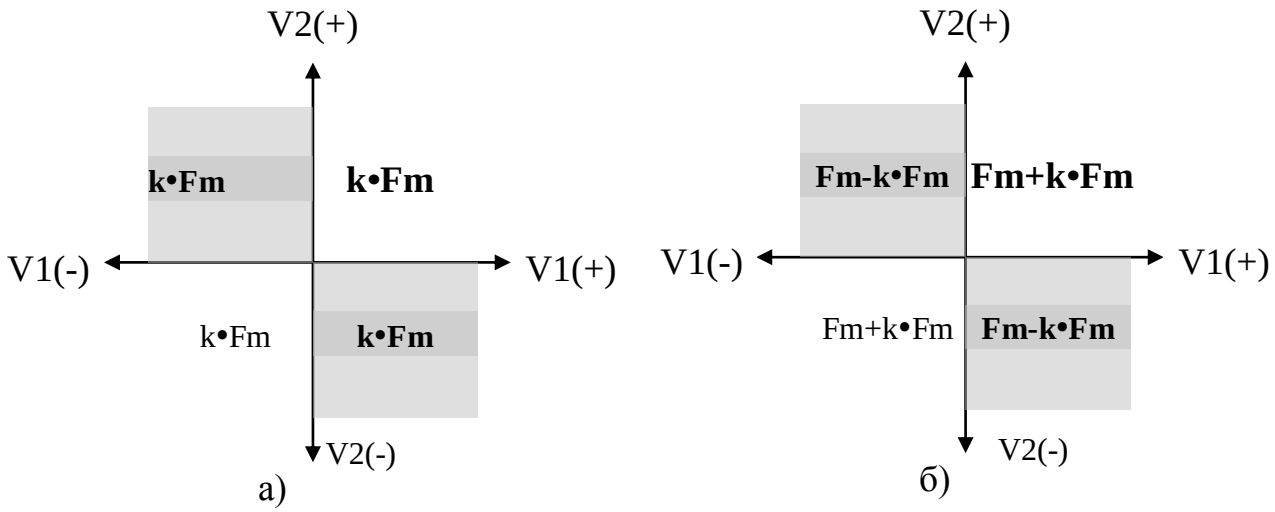


Рисунок 6.11 – Залежність вихідних сигналів від різниці фаз струму і напруги для двоквадрантного (а) і чотириквадрантного (б) вимірювання потужності

6.5 Мікропроцесорний вимірювач кутової швидкості

Розглянемо основні принципи побудови мікропроцесорного засобу вимірювання кутової швидкості електричних машин із використанням фотоелектричного сенсора кутової швидкості. Його структурна схема подана на рис. 6.12.

Сенсор кутової швидкості перетворює змінну кутову швидкість

$\omega_x(t)$ у частоту слідування електричних імпульсів $f_x(t)$.

Визначимо нижню межу вимірювання цифрового частотоміра миттєвих значень (періодоміра), рівняння перетворення якого має вигляд

$$N_\omega(t) = \frac{2\pi \cdot f_0}{\omega_x(t) \cdot z}, \qquad (6.10)$$

де z – кількість міток в фотоелектричному сенсорі кутової швидкості.

Похибка квантування визначається

$$\delta_\omega(t) = \frac{\omega_x(t) \cdot z}{2\pi \cdot f_0} \cdot 100\%, \qquad (6.11)$$

де f_0 – частота імпульсів квантування; z – розрізнювальна здатність сенсора кутової швидкості.

Таблиця 6.1 – Режими роботи мікросхеми AD7750

Режим	FS	S1	S2	F1, F2, Гц	Fout, Гц	Fmax	Опис режиму
0	0	0	0	$Fm1 \pm k \cdot Fm1$	16×F1	$Fm1 = \frac{f_0}{2^{19}}$	Вимірювання потужності в чотирьох квадрантах
1	0	0	1	$k \cdot Fm1$	8×F1	$Fm1 = \frac{f_0}{2^{19}}$	Вимірювання потужності в двох квадрантах
2	0	1	0	$k \cdot Fm1$	16×F1	$Fm1 = \frac{f_0}{2^{19}}$	Вимірювання потужності в двох квадрантах
3	0	1	1	$Fm1 \pm k \cdot Fm1$	32×F1	$Fm1 = \frac{f_0}{2^{19}}$	На виході Fout частота пропорційна напрузі на першому вході, на виходах F1, F2 частота пропорційна потужності в чотирьох квадрантах
4	1	0	0	$Fm1 \pm k \cdot Fm1$	16×F1	$Fm1 = \frac{f_0}{2^{18}}$	Вимірювання потужності в чотирьох квадрантах
5	1	0	1	$k \cdot Fm1$	16×F1	$Fm1 = \frac{f_0}{2^{18}}$	Вимірювання потужності в двох квадрантах
6	1	1	0	$k \cdot Fm1$	32×F1	$Fm1 = \frac{f_0}{2^{18}}$	Вимірювання потужності в двох квадрантах
7	1	1	1	$Fm1 \pm k \cdot Fm1$	16×F1	$Fm1 = \frac{f_0}{2^{18}}$	На виході Fout частота пропорційна напрузі на другому вході, на виходах F1, F2 частота пропорційна потужності в чотирьох квадрантах

Примітка. Коефіцієнт k пропорційний добутку середньоквадратичних значень напруг в каналах 1 і 2: $k = \dfrac{1.32 \cdot V1 \cdot V2 \cdot G}{U_{ref}^2}$,

де G – коефіцієнт підсилення в каналі 1, U_{ref}^2 – величина опорної напруги.

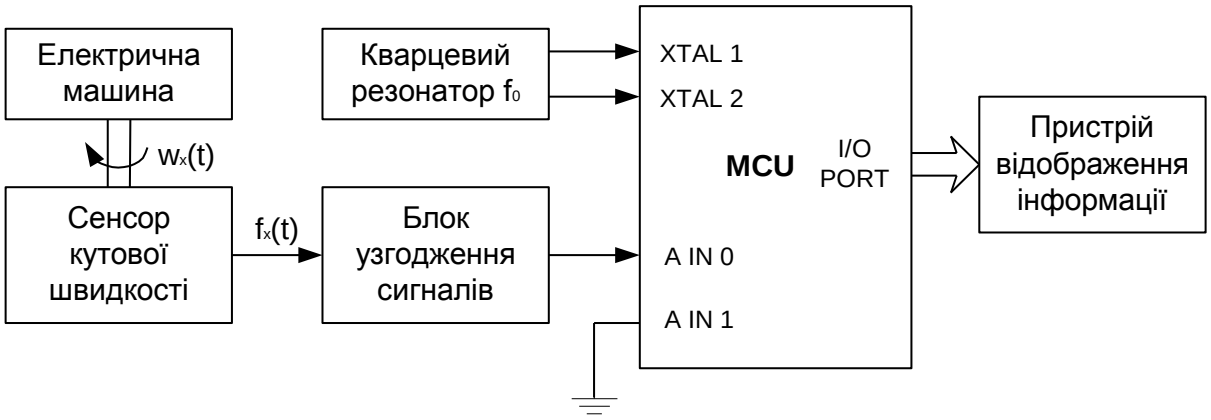


Рисунок 6.12 – Структурна схема вимірювача кутової швидкості

Максимальна ємність бінарного лічильника мікроконтролера складає

$$N_{\max} = 2^{n+1} - 1, \tag{6.12}$$

де n – розрядність лічильника.

Враховуючи (6.12), рівняння (6.10) запишемо

$$2^{n+1} - 1 = \frac{2\pi \cdot f_0}{z \cdot \omega_{x\min}}. \tag{6.13}$$

Тоді з рівняння (6.13) визначимо нижню межу вимірювання

$$\omega_{x\min} = \frac{2\pi \cdot f_0}{z \cdot \left(2^{n+1} - 1\right)}. \tag{6.14}$$

Верхня межа вимірювання $\omega_{x\max}$ для частотоміра миттєвих значень визначається із рівняння похибки квантування (6.11):

$$\omega_{x\max} = \frac{2\pi \cdot \delta_{\omega n} \cdot f_0}{z \cdot 100\%}, \tag{6.15}$$

де $\delta_{\omega n}$ – нормоване значення похибки квантування (6.11).

Задаючись величиною $\delta_{\omega n} = 1\%$, отримаємо

$$\omega_{x\,\max} = 314\,\frac{рад}{сек}\quad \left| \begin{array}{l} f_0 = 5\ МГц \\ z = 1000 \end{array} \right|. \tag{6.16}$$

На рис. 6.13 подано розраховану залежність $N_x(t)$ в режимі пуску електричної машини, а на рис. 6.14 – похибку квантування.

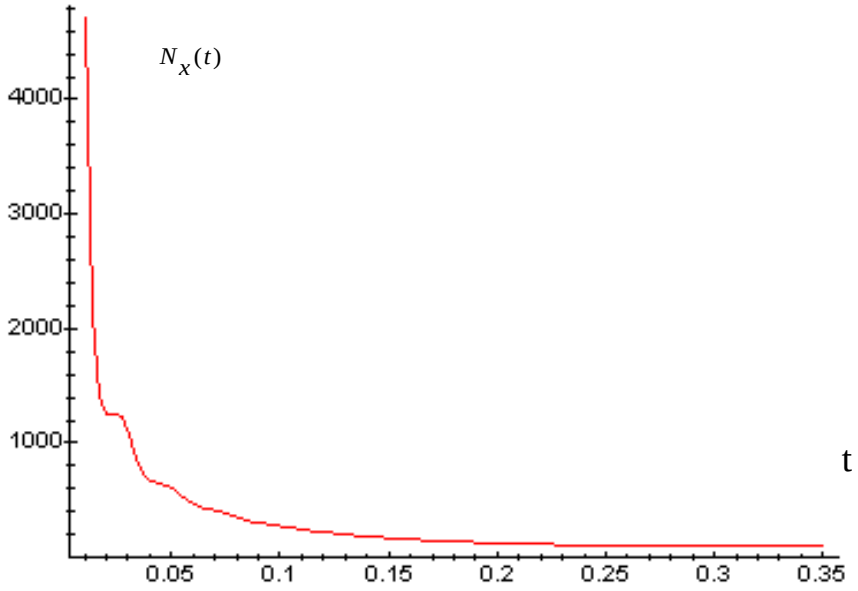


Рисунок 6.13 – Залежність $N_x(t)$

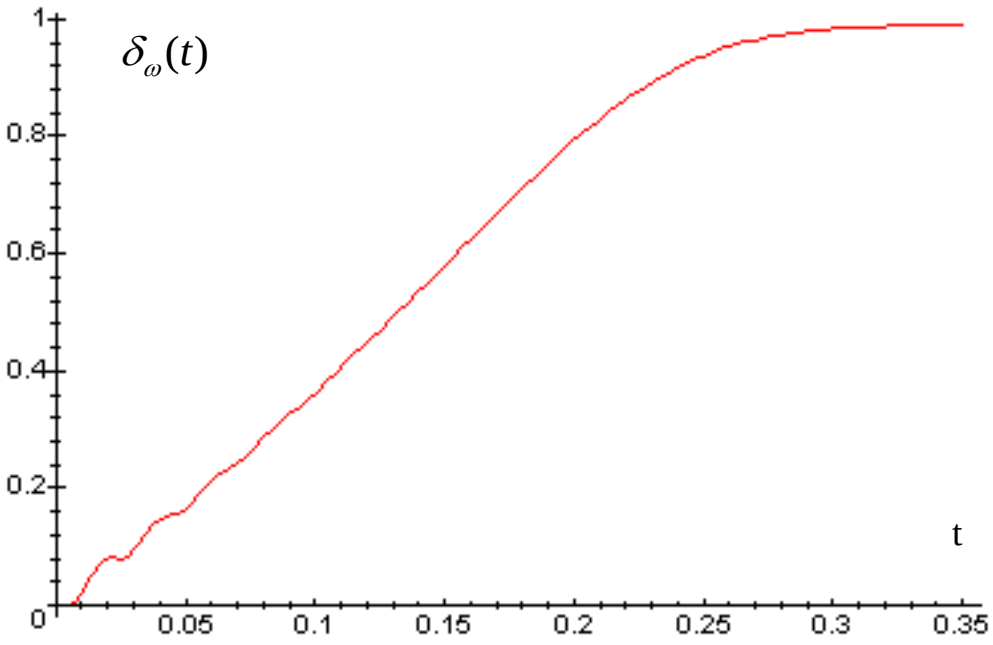


Рисунок 6.14 – Залежність $\delta_\omega(t)$

З рисунку 6.14 видно, що значення похибки квантування не перевищує 1% при f_0 = 5 МГц та z=1000. Таким чином, для вимірювання кутової швидкості як в перехідних режимах роботи електричної машини, так і в статичному режимі можна використовувати частотомір миттєвих значень. Алгоритм роботи мікропроцесорного вимірювача кутової швидкості подано на рис. 6.15. Рівняння перетворення мікропроцесорного вимірювача кутової швидкості має вигляд:

$$\omega_x = \frac{2\pi \cdot f_0}{N_x \cdot z \cdot K} \qquad (6.17)$$

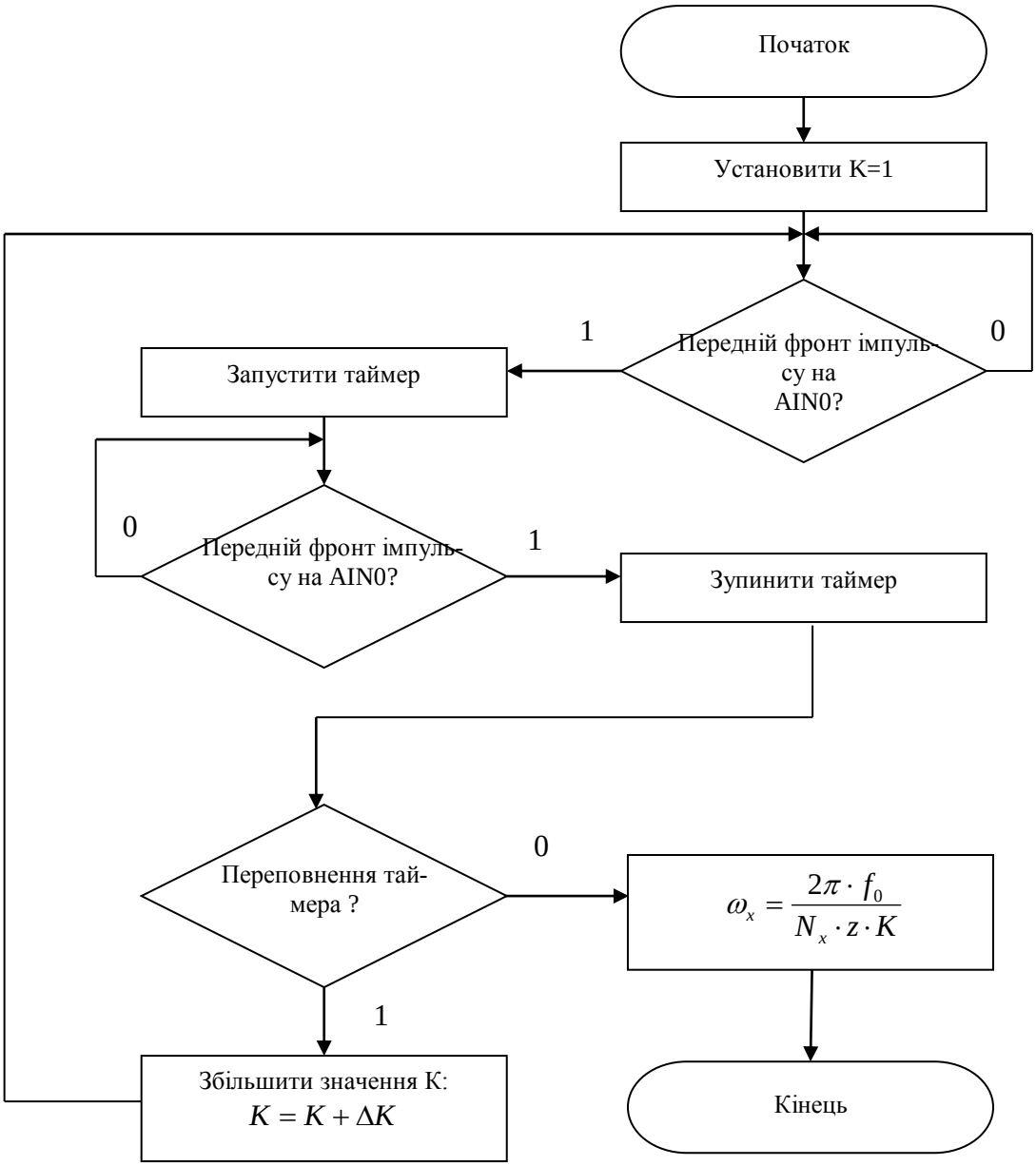


Рисунок 6.15 – Алгоритм роботи мікропроцесорного вимірювача кутової швидкості

6.6 Мікропроцесорний вимірювач ковзання

Ковзання S в асинхронній електричній машині характеризує відставання частоти поля статора f_s від частоти поля ротора f_r:

$$S(t) = \frac{f_s(t) - f_r(t)}{f_s(t)} = 1 - \frac{f_r(t)}{f_s(t)} \qquad (6.18)$$

і є досить важливим параметром, який значною мірою визначає якість електричної машини.

Як видно з формули (6.18), для визначення ковзання S потрібно проводити вимірювання двох складових: частоти поля статора f_s і частоти поля ротора f_r.

На рис. 6.16 наведено структурну схему мікропроцесорного вимірювача ковзання, яка складається з двох основних вимірювальних каналів – кутової швидкості $\omega_x(t)$ і частоти мережі живлення f_m.

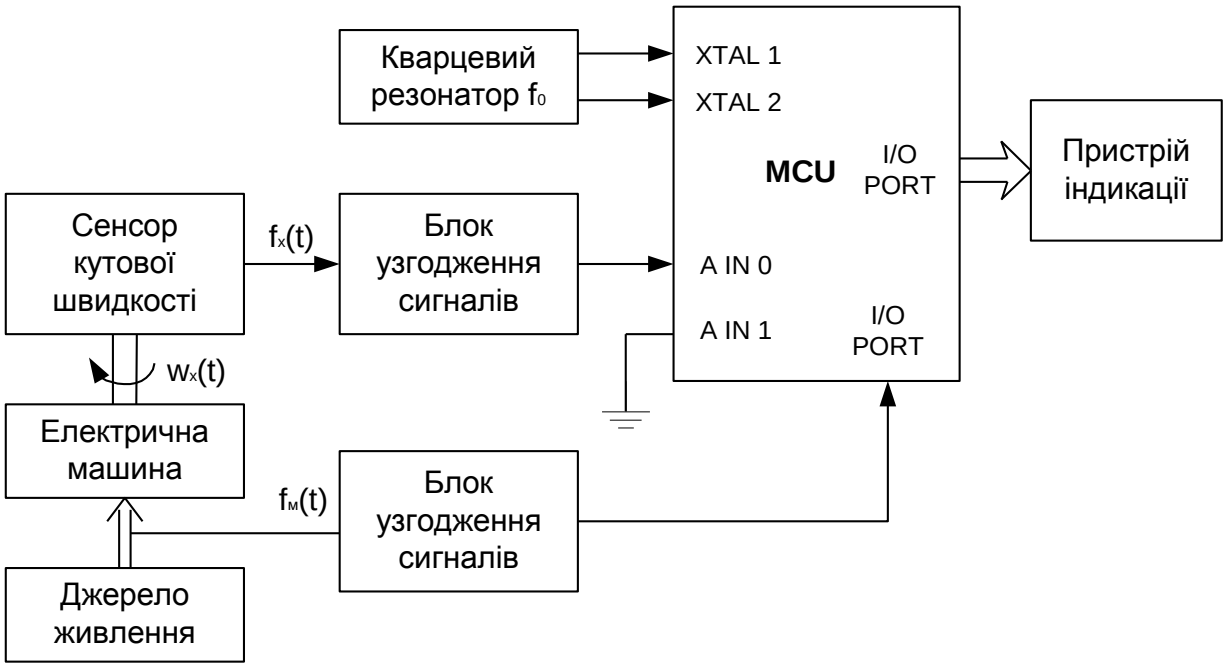


Рисунок 6.16 – Структурна схема мікропроцесорного ковзиметра

Рівняння перетворення вимірювальних каналів кутової швидкості $\omega_x(t)$ і частоти мережі живлення f_m матимуть вигляд:

$$\omega_x = \frac{2\pi \cdot f_0}{N_{x\,r} \cdot z \cdot K_r}; \qquad (6.19)$$

$$f_m = \frac{f_0}{N_{xs} \cdot K_s}, \tag{6.20}$$

де K_s, K_r – коефіцієнти подільника частоти у вимірювальних каналах частоти мережі та кутової швидкості відповідно; N_{xr}, N_{xs} - кількість імпульсів у вимірювальних каналах частоти мережі та кутової швидкості відповідно.

Враховуючи, що $\omega_r = 2\pi \cdot f_r$, із рівнянь (6.18) – (6.20) отримаємо рівняння перетворення мікропроцесорного вимірювача ковзання:

$$S = 1 - \frac{N_{xs} \cdot K_s}{N_{xr} \cdot K_r \cdot z}. \tag{6.21}$$

Алгоритм роботи мікропроцесорного вимірювача ковзання наведено на рис. 6.17.

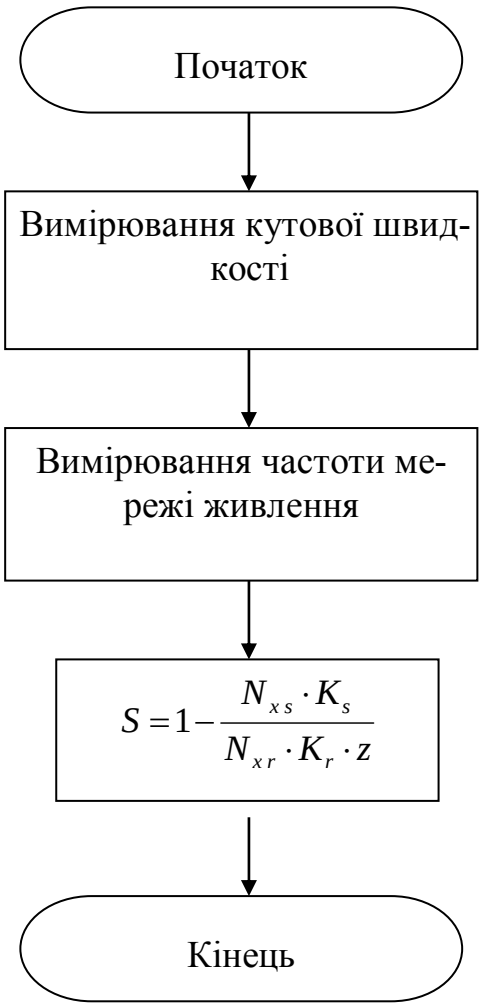


Рисунок 6.17 – Алгоритм роботи мікропроцесорного вимірювача ковзання

6.7 Мікропроцесорний вимірювач моменту інерції

В даному параграфі розглядається метод визначення моменту інерції, який використовує зразкові моменти інерції J_1, J_2 та обмежений діапазон кутових швидкостей обертання в режимі самогальмування електричної машини, в якому момент опору M_0 лінійно залежить від кутової швидкості ω_r:

$$M_0(\omega_r) = a \cdot \omega_r, \tag{6.22}$$

де a – тангенс кута нахилу характеристики опору.

Тоді рівняння руху електричної машини відповідно з першим і другим зразковими моментами інерції в процесах самогальмування мають вигляд

$$\omega_{r1}(t) = \omega_n \cdot e^{-\frac{at}{J+J_1}},\ \omega_{r2}(t) = \omega_n \cdot e^{-\frac{at}{J+J_2}}, \tag{6.23}$$

де ω_n – номінальна кутова швидкість обертання; J – момент інерції електричної машини; t – час.

Після логарифмування рівнянь (6.23)

$$at = (J + J_1) \cdot \ln\frac{\omega_{r1}(t)}{\omega_n},\quad at = (J + J_2) \cdot \ln\frac{\omega_{r2}(t)}{\omega_n}, \tag{6.24}$$

і виключення складової at з рівнянь (6.24) рівняння для визначення моменту інерції запишеться

$$J = \frac{J_2 \cdot \ln\frac{\omega_{r2}(t)}{\omega_n} - J_1 \cdot \ln\frac{\omega_{r1}(t)}{\omega_n}}{\ln\frac{\omega_{r1}(t)}{\omega_{r2}(t)}} \tag{6.25}$$

Структурна схема для реалізації запропонованого способу вимірювання моменту інерції наведена на рисунку 6.18. Мікропроцесорна система MCU проводить вимірювання кутової швидкості ω_r і за допомогою таймера обчислює час t, який необхідний для визначення моменту інерції.

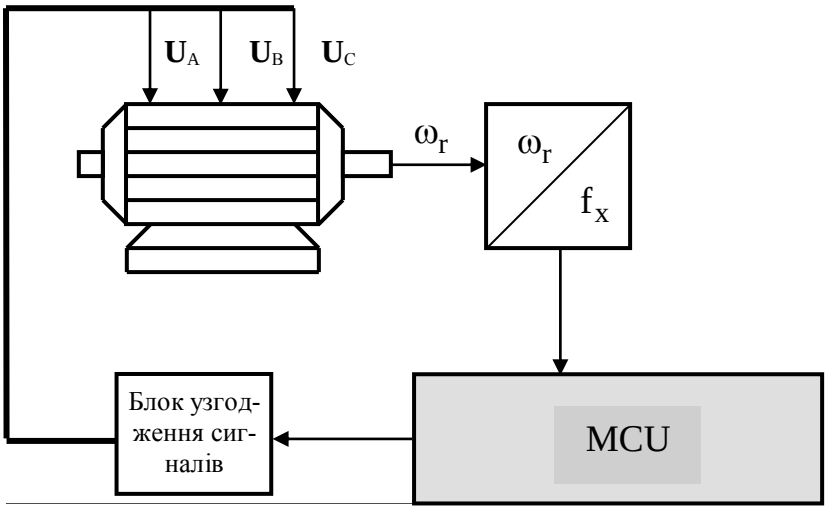


Рисунок 6.18 – Структурна схема мікропроцесорного вимірювача моменту інерції

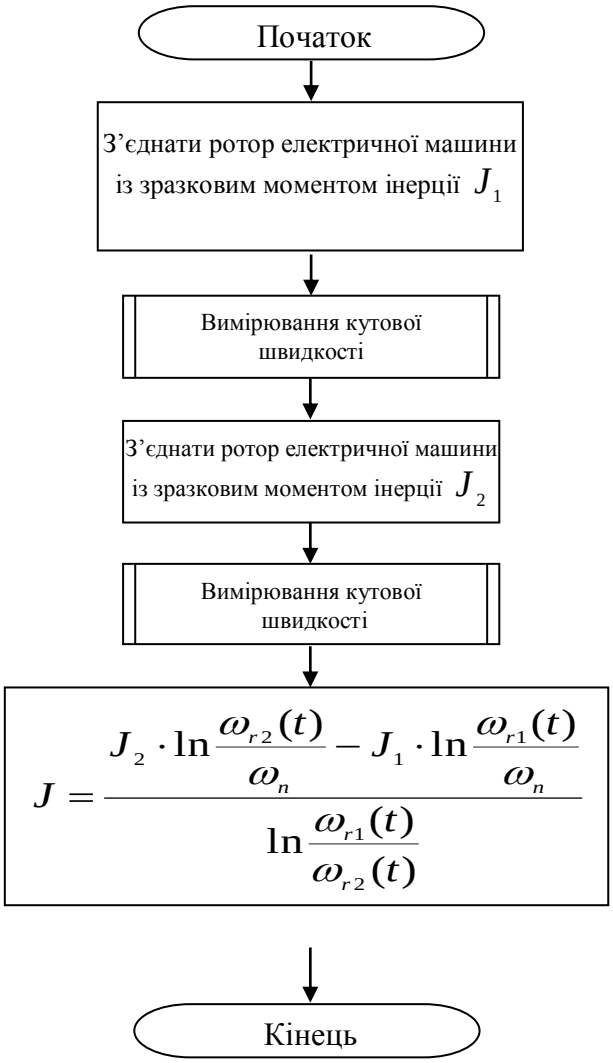


Рисунок 6.19 – Алгоритм роботи мікропроцесорного вимірювача моменту інерції

6.8 Мікропроцесорний вимірювач пускового моменту

Залежність пускового моменту M_n від кута повороту ротора α є досить важливим параметром, який значною мірою визначає якість пускових характеристик електричної машини.

Структурна схема засобу вимірювання залежності $M_n(\alpha)$ наведена на рис. 6.20.

Зовнішній приводний двигун ПД із невеликою наперед заданою швидкістю обертає ротор електричної машини ЕМ. Одночасно вимірюється момент на його роторі та кутове положення ротора протягом одного повного оберту ротора ЕМ. Швидкість обертання ротора ЕМ має бути такою, щоб за час вимірювання залежності $M_n(\alpha)$ температура обмоток статора ЕМ не перевищила розрахункове робоче значення.

Пристрій містить основу, приводний двигун ПД, вихідний вал якого встановлено на підшипниках, сенсор зусилля, що встановлено на основі перетворювача і з'єднано зі статором приводного двигуна вимірювальним важелем, пускові пристрої приводного двигуна і ЕМ, мікропроцесорну систему MCU.

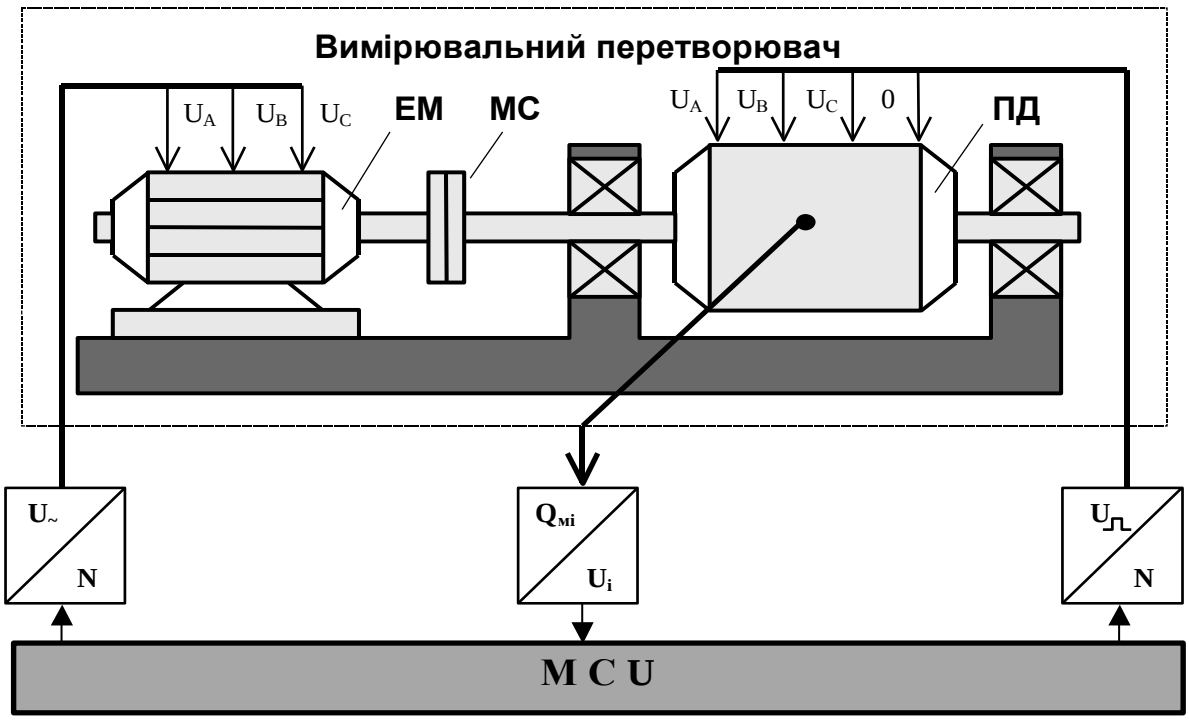


Рисунок 6.20 – Структурна схема засобу вимірювання залежності $M_n(a)$

Вимірювання здійснюють у відповідності з таким алгоритмом. ЕМ встановлюють на основі вимірювального перетворювача жорстко і його ротор за допомогою муфти спряження МС з'єднують із вихідним валом

приводного двигуна, а живлення ЕМ і ПД відбувається від електричної мережі за допомогою пускових пристроїв.

Перед вмиканням приводного двигуна і ЕМ до електричної мережі в оперативній пам'яті MCU встановлюється початкова адреса і довжина буфера, в якому будуть зберігатися масиви виміряних значень моменту і кута повороту ротора. Потім MCU переходить на програму запуску ЕМ і ПД. Приводний двигун обертає через муфту МС ротор ЕМ. В результаті вимірюваний параметр $M_{n\,EM}$ передається з ротора ЕМ на муфту МС і на корпус приводного двигуна. При цьому на сенсор зусилля через вимірювальний важіль діє сила Q_i, пропорційна $M_{n\,EM}$. Напруга U_i на виході сенсора зусилля, пропорційна Q_i, через аналого-цифровий перетворювач MCU записується в оперативну пам'ять. Після завершення одного повного оберту ротора ЕМ приводний двигун і ЕМ знеструмлюються, а в оперативній пам'яті MCU сформується масив виміряних значень сили Q_i. Причому адреса масиву виміряних значень упорядкована за кутовим положенням ротора. Після цього визначається масив результатів пускового моменту

$$M_{ni} = Q_i \cdot g \cdot l, \qquad (6.26)$$

де g – прискорення вільного падіння; l – довжина вимірювального важеля.

Перелік літератури

1. Бродин В.Б., Шагурин И.И. Микроконтроллеры. Архитектура, программирование, интерфейс. –М.:ЭКОМ, 1999.

2. Козаченко В.Ф. Микроконтроллеры: Руководство по применению 16-разрядных микроконтроллеров Intel MCS-196/296 во встроенных системах управления. –М.:ЭКОМ, 1997.

3. Современные микроконтроллеры: Архитектура, средства проектирования, примеры применения, ресурсы сети Интернет. «Телесистемы» / Под ред. И.В.Коршуна. Сост.: Пер. с англ. Б.Б.Горбунова –М.: Аким, 1998.

4. AT90S2313 Embedded Microcontroller User's Manual. Atmel Corp. 1999.

5. Чумаченко І.В., Кошовий М.Д., Лопатин В.В. Мікроконтролерні прилади: структура і використання: Навчальний посібник. – Харків: Нац. аерокосмічний ун-т «ХАІ», 2001. –277с.

Міністерство освіти і науки України
Вінницький національний технічний університет

В.О. Поджаренко, В.Ю. Кучерук, В.М. Севастьянов

Основи мікропроцесорної техніки

НАВЧАЛЬНИЙ ПОСІБНИК

Усі цитати, цифровий, фактичний матеріал та бібліографічні відомості перевірені, написання одиниць відповідає стандартам.

Зауваження рецензентів враховані.

Автори: _____ В.О. Поджаренко

_____ В.Ю. Кучерук

_____ В.М. Севастьянов

Вимогам, які висуваються до навчальної літератури, відповідає.

До друку і в світ дозволяю на підставі § 2 п. 15 "Єдиних правил .... "

Проректор з навчальної та науково-методичної роботи

В.О. Леонтьєв

Затверджено
на засіданні кафедри МПА
Протокол № ___ від __.__.200__р.
Зав. кафедрою

_____ В.О. Поджаренко

Вінниця ВНТУ 2006
Навчальне видання

Володимир Олександрович Поджаренко
Володимир Юрійович Кучерук
Володимир Миколайович Севастьянов

Основи мікропроцесорної техніки

Навчальний посібник

Оригінал-макет підготовлено авторами

Редактор О.Д. Скалоцька

Науково-методичний відділ ВНТУ
Свідоцтво Держкомінформу України
серія ДК № 746 від 25.12.2001
21021, м. Вінниця, Хмельницьке шосе, 95, ВНТУ

Підписано до друку
Формат 29,7х42¼ Гарнітура Times New Roman
Папір офсетний
Друк різографічний Ум. друк. арк.
Тираж _____ прим.

Зам. №

Віддруковано в комп'ютерному інформаційно-видавничому центрі Вінницького національного технічного університету
Свідоцтво Держкомінформу України
серія ДК № 746 від 25.12.2001
21021, м. Вінниця, Хмельницьке шосе, 95, ВНТУ