

МІНІСТЕРСТВО ОСВІТИ І НАУКИ,
МОЛОДІ ТА СПОРТУ УКРАЇНИ

В.М. РЯБЕНЬКИЙ, О.О. УШКАРЕНКО, В.С. БУРЯК

СХЕМОТЕХНІКА ЕЛЕКТРОННИХ ПРИСТРОЇВ ТА СИСТЕМ

*Рекомендовано Міністерством освіти і науки України
як підручник для студентів вищих навчальних закладів*

У ШЕСТИ ТОМАХ

ТОМ 3

МІКРОПРОЦЕСОРНА ТЕХНІКА

Миколаїв
“Гліон”
2012

УДК 681.3:621.38
ББК 32.884.1
Р 98

Р е ц е н з е н т и:

Т.О. Терещенко, доктор технічних наук, професор
І.І. Коваленко, доктор технічних наук, професор

*Рекомендовано Міністерством освіти і науки України
як підручник для студентів вищих навчальних закладів
(лист № 1.4/Г8–Г–3007 від 31.12.2008 р.)*

Рябенський В. М., Ушкаренко О. О., Буряк В. С.

Р 98 Схемотехніка електронних пристроїв та систем : підручник : в 6 т. /
В. М. Рябенський, О. О. Ушкаренко, В. С. Буряк. – Миколаїв : Іліон,
2012 – .

ISBN 978-617-534-066-0

Т. 3 : Мікропроцесорна техніка. – 2012. – 446 с.

ISBN 978-617-534-084-4

У підручнику розглядаються основи проектування мікропроцесорних систем на базі 8-розрядних мікроконтролерів AVR та 32-розрядних мікроконтролерів з архітектурою ARM. Приводиться велика кількість прикладів програм для мікроконтролерів на мові Сі, а також прикладів опису цифрових пристроїв для пограмованих логічних інтегральних схем на мові Verilog.

Підручник призначений для студентів і фахівців електротехнічних спеціальностей.

**УДК 681.3:621.38
ББК 32.884.1**

ISBN 978-617-534-066-0

ISBN 978-617-534-084-4 (Т. 3)

© Рябенський В. М., Ушкаренко О. О., Буряк В. С., 2012

ВСТУП

Мікроконтролери AVR по праву вважаються одним із самих цікавих напрямків, які активно розвиває корпорація Atmel. Вони являють собою потужний інструмент для створення сучасних високопродуктивних і економічних багатоцільових контролерів. Галузі застосування AVR багатогранні – від найпростіших іграшок і інтелектуальних датчиків до складних промислових систем керування й контролю і сучасного телекомунікаційного обладнання. Сімейство інтенсивно розвивається, ускладнюється, удосконалюється.

Архітектура Flash-мікроконтролерів AVR 8-bit RISC є однією із самих вдалих на світовому ринку мікроконтролерів. Наявність трьох типів пам'яті на кристалі, висока продуктивність, низьке енергоспоживання, відмінна адаптивність до мов програмування високого рівня (C/C++), різноманітні периферійні модулі, широкий діапазон напруг живлення й сумісність кристалів сімейства "знизу нагору" здобули заслужену повагу й авторитет в усьому світі.

Мікроконтролери AVR – це досить потужні мікросхеми, які можуть бути використані для вирішення широкого кола задач. Однак завжди знайдуться задачі, коли потрібна ще більш висока обчислювальна потужність та більш розвинена периферія. В такому випадку увага розробників повертається до контролерів на ядрі ARM7. В підручнику розглядається сімейство мікроконтролерів SAM7 фірми Atmel на прикладі найбільш функціонально насиченого представника даного сімейства – AT91SAM7X256.

При розробці цифрових пристроїв широке поширення знайшли програмовані логічні інтегральні схеми, які дуже вдало доповнили мікропроцесорні засоби. Це зручна елементна база, яка використовується в системах цифрової обробки сигналів, телекомунікації, для побудови різноманітних систем керування та ін. Існує можливість створювати на кристалі спеціалізовані процесорні ядра, що дозволяє вирішувати будь-які науково-практичні задачі. Однак схемна реалізація подібних систем – задача досить складна, а іноді і неможлива. Для вирішення подібних задач може бути використана мова опису апаратури (HDL), однією з яких є Verilog. Він вигідно відрізняється загальноалгоритмічною базою, гнучкістю, широким діапазоном охоплення систем та рівнів їх опису. Verilog є офіційно признаним стандартом опису цифрової апаратури. В підручнику розглядається велика кількість прикладів цифрових моделей, описаних за допомогою Verilog, які можуть бути синтезовані та запрограмовані в мікросхему. Детально описуються принципи проектування цифрових пристроїв на ПЛІС фірми ALTERA на основі САПР Quartus.

Підручник призначений для студентів, які досконало вивчають мікропроцесорну техніку, а також буде корисним розробникам електронних пристроїв на сучасній елементній базі.

УМОВНІ СКОРОЧЕННЯ

АЛП – Арифметико–логічний пристрій
АЦП – Аналого–цифровий перетворювач
БА – Блок адресації
БК – Блок керування
ЕОМ – Електронно–обчислювальна машина
ЛАК – Лічильник адреса команди
МК – Мікроконтролер
МП – Мікропроцесор
МЦ – Машинний цикл
НПЗСІ – Напівпровідникові знакосинтезуючі індикатори
ОЗП – Оперативний запам’ятовуючий пристрій
ПВВ – Пристрої вводу/виведення інформації
ПДП – Прямий доступ до пам’яті
ПЗП – Постійний запам’ятовуючий пристрій
ПК – Персональний комп’ютер
ПЛІС – Програмована інтегральна логічна схема
ПС – Показчик стеку
РА – Регістр адреси
РЗП – Регістри загального призначення
РК – Регістр команди
РКІ – Рідкокристалічний індикатор
РО – Регістр операнда
САПР – Система автоматизованого проектування
ССП – Слово стану процесора
ФАПЧ – Фазове автопідлаштування частоти
ЦАП – Цифро–аналоговий перетворювач

1. ЗАГАЛЬНІ ВІДОМОСТІ ПРО МІКРОПРОЦЕСОРИ

1.1. Основні поняття та визначення

Характерна риса традиційної цифрової системи полягає в тому, що алгоритми обробки і зберігання інформації в ній жорстко зв'язані зі схемотехнікою системи. Зміна цих алгоритмів можлива тільки шляхом зміни структури системи, заміни електронних вузлів, що входять у систему, і зв'язків між ними. Наприклад, якщо потрібна додаткова операція додавання, то необхідно додати в структуру системи суматор. Якщо потрібна додаткова функція зберігання коду протягом одного такту, то необхідно додати в структуру ще один регістр. Це практично неможливо зробити в процесі експлуатації, обов'язково потрібний новий виробничий цикл проектування, виготовлення, налагодження всієї системи. Саме тому традиційна цифрова система часто називається системою на "твердій логіці".

Розглянемо простий приклад. Необхідно розробити цифровий пристрій, призначений для виконання операції множення двох 4-х розрядних слів.

Розв'язання. Множимо $B=1101$. Множник $A=1011$. Послідовність виконання операції перемноження приведена на рис. 1.1.

B	1101	множимо
A	1011	множник
	<u>1101</u>	частковий добуток 1
	1101	частковий добуток 2
	<u>100111</u>	частковий добуток 3
	1101	частковий добуток 4
	<u>10001111</u>	добуток

Рис. 1.1.

Аналіз послідовності виконання операції у відповідності до рис. 1.1 показує, що операція множення полягає в послідовному виконанні двох простих операцій – знаходження суми двох слів і зсув праворуч часткового добутку, або ліворуч множника на один розряд. Звідси випливає, що для побудови пристрою, що виконуватиме вказану операцію, необхідно мати такі цифрові елементи:

- послідовний регістр;
- суматор;
- паралельні регістри для запам'ятовування даних;
- пристрій для керування роботою попередніх елементів.

На рис. 1.2 представлений можливий взаємозв'язок між елементами, які утворюють схему перемножувача.

Множимо B записується в регістр $DD1$ з відкритими виходами, які підключаються до входів B суматора $DD2$. Множник A з регістра $DD3$, де він зберігається, записується в старші розряди D_4-D_7 послідовного регістра $DD4$, а виходи суматора підключаються до його молодших розрядів D_0-D_3 . Чотири молодші розряди виходу послідовного регістру з'єднані з входами слова A суматора $DD2$. Блок керування (БК) має два входи та два виходи.

Вхід синхронізації T від зовнішнього тактового генератора (годинника) задає частоту виконання операцій. Сигнал D з виходу D_7 старшого розряду послідовного регістра використовується для організації виконання операції додавання. Виходи БК V та C призначені відповідно для виконання операцій знаходження суми та операції зсуву інформації на 1 розряд в послідовному регістрі. Виконання операції перемноження відбувається в наступній послідовності.

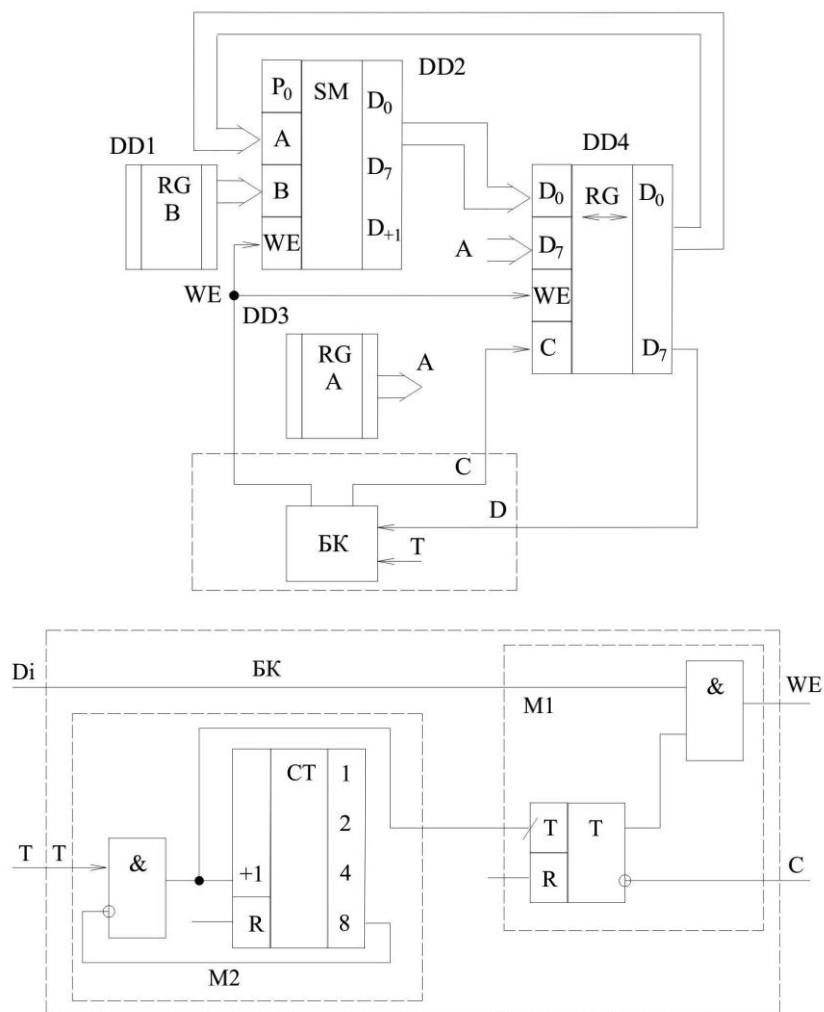


Рис. 1.2.

При наявності першого тактового сигналу T повинна виконуватись операція додавання множеного B з двійковим кодом, що розміщений в молодших розрядах послідовного регістру. В перший момент часу там записаний нуль. При наявності сигналу "1" на виводі D_7 послідовного регістра в БК формується сигнал WE (Write enable), який дозволяє

суматору SM виконати операцію і одночасно записати результат в молодші розряди послідовного регістру. При другому тактовому сигналі формується сигнал C , що зсуває інформацію послідовного регістру на 1 розряд, і тим самим замінює в D_7 значення останнього розряду множника на передостанній. При наступному тактовому сигналі знову повинен формуватись сигнал WE , але він може з'явитись лише у випадку, якщо в D_7 на цей час знаходиться "1". При наявності "0" в цьому розряді регістру сигнал WE не з'являється і операція знаходження поточної суми не виконується. Далі знову формується сигнал C . БК – це керуючий цифровий автомат, який складається з двох модулів. Перший – $M1$ – містить всього два елементи – T -тригер, на тактовий вхід якого подається сигнал T , та логічний елемент $2I$, на один вхід якого подається сигнал з прямого виходу T -тригера, а на другий – сигнал D . Інверсний вихід тригера формує сигнал C . Другий – $M2$ – містить логічний ключ та лічильник тактів T , щоб забезпечити зупинку автомата після завершення операції.

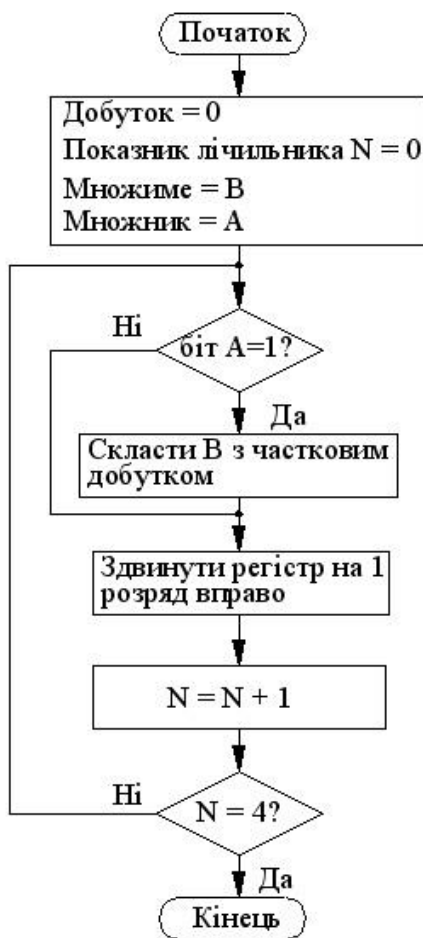


Рис. 1.3.

По–перше, спеціалізована система (на відміну від універсальної) ніколи не має апаратної надмірності, тобто кожен її елемент обов'язково працює на повну силу, якщо ця система грамотно спроектована.

По–друге, саме спеціалізована система може забезпечити максимально високу швидкодію, тому що швидкість виконання алгоритмів обробки інформації визначається в ній тільки швидкодією окремих логічних елементів і обраною схемою шляхів проходження інформації. Саме логічні елементи завжди володіють максимальним на даний момент швидкодією.

Але, в той же час, недоліком цифрової системи на "жорсткій логіці" є те, що для кожного нового завдання її потрібно проектувати та виготовляти заново. Це процес тривалий, коштовний, і потребує високої кваліфікації виконавців.

Шлях подолання цього недоліку наступний: потрібно побудувати таку систему, яка могла б легко адаптуватися під будь-яке завдання, перебудовуватися з одного алгоритму роботи на інший без зміни апаратної реалізації, а задавати той або інший алгоритм можна було б шляхом введення в систему додаткової керуючої інформації – програми роботи системи (рис. 1.4). Тоді система стане універсальною, або програмованою, гнучкою. Саме це і забезпечує мікропроцесорна система (МПС).

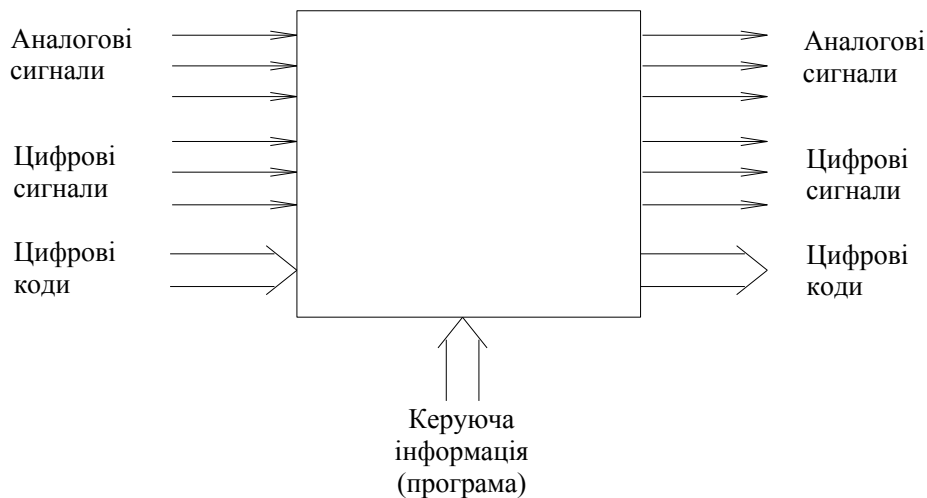


Рис. 1.4.

МПС може розглядатися як окремий випадок електронної системи, призначеної для обробки входних сигналів і видачі вихідних сигналів. У якості входних і вихідних сигналів можуть використовуватися аналогові сигнали, одиночні цифрові сигнали, цифрові коди, послідовності цифрових кодів. Всередині системи може виконуватися зберігання, накопичення сигналів. Входні аналогові сигнали перетворюються в послідовності кодів вибірок за допомогою аналого–цифрового перетворювача (АЦП), а вихідні аналогові сигнали формуються з послідовності кодів вибірок за допомогою цифро–аналогового перетворювача (ЦАП). Обробка та зберігання інформації виконується в цифровому вигляді.

Таким чином, можна зробити наступний висновок. Системи на "жорсткій логіці" зручно використовувати в тих випадках, коли задача, що розв'язується, не змінюється тривалий час, не потрібна висока швидкодія, алгоритми обробки інформації досить прості. Універсальні, програмовані системи зручні тоді, коли часто змінюються задачі, які потребують розв'язання, висока швидкодія не надто важлива, алгоритми обробки інформації досить складні.

Однак за останні десятиліття швидкодія універсальних (мікропроцесорних) систем значно виросла (на декілька порядків). До того ж, великий обсяг випуску мікросхем для таких систем призвів до різкого

зниження їх вартості, а властивості сучасних програмованих логічних матриць відкривають широкі можливості до створення власних спеціалізованих мікропроцесорних структур. Тому слід більш детально розглянути особливості схеми, що наведена на рис. 1.2, та визначитися з термінологією, що використовується в подібній схемотехніці.

Перш за все слід знати, що будь-який обчислювальний пристрій характеризується форматом слів, що приймають участь в обробці інформації. **Словом** називається впорядкована кількість біт, яка характеризується як визначена кількість інформації. **Формат слова** – це стандартизована кількість біт, що є кратною чотирьом. На сьогодні слова здебільшого характеризуються кількістю байт інформації (від 1 до 8). Слова, інформація яких підлягає обробці, називаються **операндами** (множимо B). слова у відповідності до яких виконується обробка операндів називаються **командами** (множник A). Кожен біт команди виконує конкретну дію для її виконання і називається **мікрокомандою**. Послідовність мікрокоманд, що створюють команду називається **мікропрограмою**.

Одним з найбільш важливих елементів обчислювального пристрою є реєстри. Їх головне завдання – тимчасове зберігання різноманітної інформації. Реєстри, в яких зберігаються операнди перед їх обробкою, називаються **реєстрами операндів (PO)**. Реєстри, в яких зберігається код (слово) команди, називається **реєстром(ами) команд (PK)**. Реєстр, в який поміщається слово після обробки, називається **реєстром-акумулятором (PA)**. Комбінаційна схема, в якій виконується та чи інша операція над операндом у відповідності до коду команди, називається **арифметико-логічним пристроєм (АЛП)**. Прикладом АЛП є відома з курсу цифрової схемотехніки мікросхема К1533ІПЗ, що дозволяє виконувати 32 операції над двома чотирьохрозрядними словами у відповідності до чотирьохрозрядного коду команди [9]. Приклад її використання для побудови мікропрограмного автомата для обробки логічних функцій наведено в [11].

В загальному вигляді арифметико–логічний пристрій (АЛП), в якому виконуються окремі команди має вигляд, поданий на рис. 1.5.

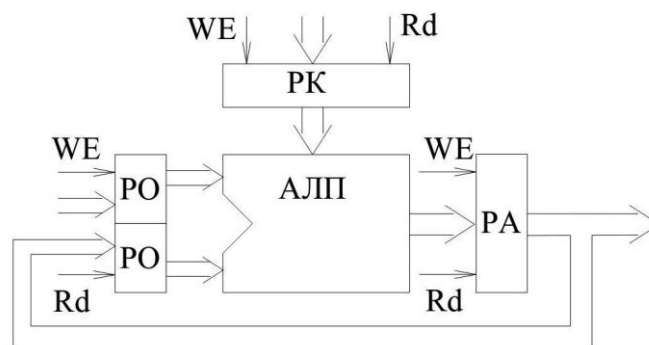


Рис. 1.5.

Розглянемо його роботу на прикладі виконання однієї операції.

По сигналах запису WE в регістри команд і операндів по черзі, або одночасно, виконується запис операндів та коду команди. Для виконання операції на вказані регістри подається сигнал зчитування Rd, а на регістр RA – сигнал WE. Комбінаційна схема АЛП виконає відповідні логічні перетворення і по завершенні дії сигналів зчитування та запису в RA буде записаний результат виконання операції.

Для більшості процесорів АЛП може виконувати арифметичні операції (додавання, віднімання, множення, іноді ділення), побітні операції типу «І», «АБО», «ВИКЛ. АБО» над словами. Швидкодія АЛП в найбільшій мірі визначає ефективність процесора, тому частоту виконання операцій завжди роблять максимально високою.

1.2. Загальна структура мікропроцесорної системи

Розглянута схема, що наведена на рис. 1.5, дозволяє розкрити питання про те, як виконати найпростішу операцію, яка по відношенню до RA може розглядатись як заміна одного слова іншим. Більш повно суть виконання операцій пояснює схема, приведена на рис. 1.6.

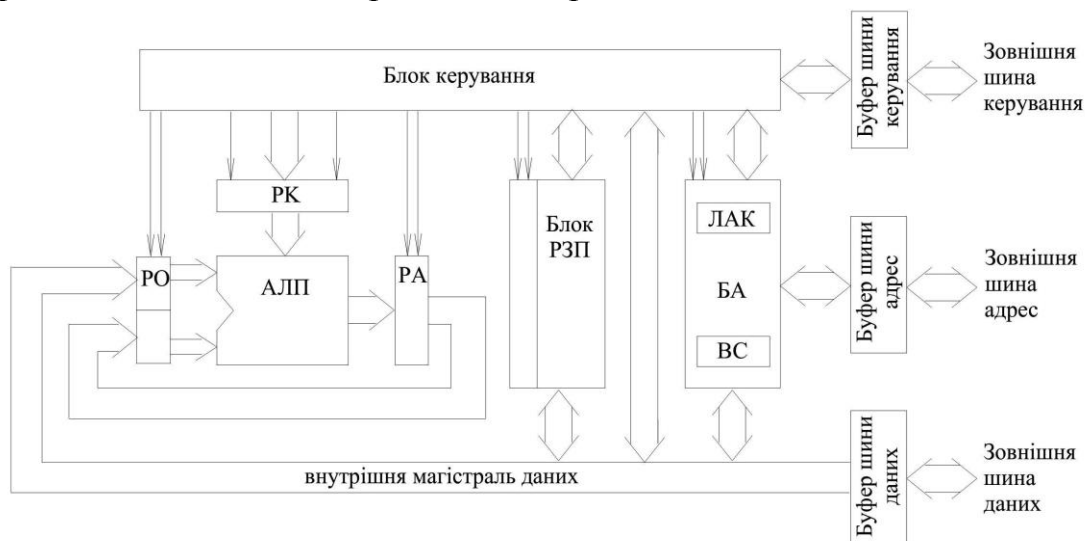


Рис. 1.6.

Блок регістрів загального призначення (РЗП) – це регістри, що призначені для тимчасового збереження інформації, що підлягає обробці. Це обмежена група регістрів, що пов'язані між собою одною загальною шиною даних, та загальною шиною керування. Головна особливість РЗП полягає в тому, що записати в ці регістри можливо тільки слова з внутрішньої магістралі даних (шини даних). Шина – це група провідників, яка об'єднує на паралельній основі всі інформаційні входи і виходи регістрів. В цих регістрах зберігаються операнди, що підлягають обробці, коди операцій, сюди заносяться слова після виконання операцій. Слід зазначити, що в блоці РЗП регістри можуть бути нерівнозначними. В ряді процесорів кожному регістру відводиться строго визначена функція. Такий підхід спрощує організацію процесора і іноді зменшує час виконання

команд. Але, наприклад, наявність одного регістра – акумулятора приводить до уповільнення роботи, оскільки всі операції повинні проходити через нього. Другий підхід полягає в тому, що всі, або майже всі регістри мають однакове призначення, що розширює можливості програміста, якому всі регістри є доступними, але ускладнює архітектуру процесора.

Виконання будь-якої операції з участю РЗП проходить в наступній послідовності. З БК поступають сигнали Rd на вибраний РЗП і сигнал WE на РО. Необхідне слово проходить через внутрішню магістраль даних з вибраного РЗП в РО. Одночасно з БК поступає код операції та сигнал WE на РК. Після виконання операції у відповідності з вище описаною послідовністю з БК поступає сигнал Rd на РА і сигнал WE на один з РЗП, куди поступає результат виконання операції через внутрішню магістраль даних.

Наявність блоку РЗП та його місткість є дуже важливими для будь-якого процесора – від найпростіших контролерів, до потужних процесорів, що використовуються в комп'ютерній техніці. Переваги їх використання полягають в тому, що вони забезпечують найбільш короткий та найбільш швидкий шлях підготовки даних та команд для роботи АЛП. Тому розміри блоку РЗП суттєво впливають на швидкість виконання операцій. До того ж на швидкість обробки інформації процесором впливає розрядність регістрів та АЛП, яка називається внутрішньою розрядністю процесора. Вона може не співпадати з зовнішньою розрядністю.

Серед регістрів загального призначення особливе місце займає регістр ознак (регістр стану). Інформація, що міститься в ньому, називається словом стану програми (ССП) (PSW – program status word). Кожен біт цього регістра має строго визначене призначення і називається ознакою, або прапорцем. Наприклад, при виконанні операції додавання двох однобайтних слів результат має 9 розрядів. Оскільки в АЛП тільки 8 розрядів, то в відповідному біті регістра ознак, який називається ознакою переносу, з'явиться 1, яку необхідно контролювати і враховувати програмісту. Більш детально призначення кожного біта регістра ознак буде розглянуто на конкретних прикладах контролерів.

Якщо необхідне слово відсутнє в РЗП, то його необхідно знайти в зовнішній пам'яті (ОЗП) за конкретною адресою. Формуванням адресів ОЗП займається спеціальний блок – блок адресації (БА).

З використанням добавлених блоків виконання окремої операції проходить в наступній послідовності:

1. Чергова команда з сформованої в БА адреси ОЗП через буфер шини даних подається на внутрішню магістраль даних. Буфер даних – в даному випадку це паралельний регістр, що вирішує задачі синхронізації при обміні даними процесора з зовнішнім середовищем, а також узгоджує рівні та потужності сигналів. Відповідні сигнали синхронізації для звернення до ОЗП формуються в БК.

2. З внутрішньої магістралі даних команда подається в БК, який перетворює її в код операції та необхідний набір сигналів керування. Формування коду операції та керуючих сигналів – головна функція БК. У випадку великих форматів команд, які розміщуються в декількох послідовних елементах ОЗП, частина описання команди блоком керування перетворюється в адресу операнда і подається в БА.

3. На основі сформованих на попередньому етапі адресів з пам'яті або, як виключення, з РЗП достається операнд, який по внутрішній магістралі даних подається в один з РО. БК при цьому формує всі необхідні сигнали.

4. АЛП виконує операцію в відповідності до раніше описаної послідовності.

5. Результат операції через внутрішню магістраль даних подається в РЗП, або через буфер даних в ОЗП по адресі, що формується в БА.

6. БА формує адресу, по якій з БА виймається описання чергової команди.

Отже, для мікропроцесорів характерні наступні базові принципи їх побудови:

1. Процесор реалізує алгоритм, описання якого зберігається в тому чи іншому пристрої пам'яті, зовнішньому по відношенню до процесора (говорять, процесор працює по програмі, що зберігається в ОЗП).

2. Принцип мікропрограмування – кожна операція керується відповідною мікрокомандою. Це найважливіший принцип, що забезпечує універсальність процесора.

3. Наявність системи внутрішньої та зовнішньої синхронізації.

4. Принцип магістральної (шинної) організації зв'язків між окремими пристроями.

Виконання того чи іншого алгоритму можливо при наявності мікропроцесора (МП) та пристроїв, в яких зберігається програма та через які вводяться дані, що підлягають опрацюванню. Програма – це сукупність команд (правил), що виконуються в послідовності, заданій алгоритмом. Команди вибираються з пам'яті в послідовності, що задається процесором. Процесор визначає адреси елементів пам'яті, в яких зберігаються необхідні данні. Данні передаються в процесор, де обробляються у відповідності з командами, і результати операції передаються знову в пам'ять.

1.3. Мікропроцесорна система та її особливості

Мікропроцесорна система – це комплекс електронних пристроїв, що включає в себе мікропроцесор, обслуговуючі його пристрої пам'яті та вводу/виводу інформації (ПВВ), а також периферійні пристрої, що є генераторами інформації, яка підлягає обробці, та споживачами результатів обробки цієї інформації. Для зв'язку процесора з зовнішніми пристроями існує інтерфейс (interface). Цим терміном позначається весь комплекс пристроїв, правил та технічних засобів, що регламентують та забезпечують обмін інформацією між мікропроцесором (включаючи

пам'ять) та зовнішніми пристроями. Головними в інтерфейсі є шини, або, як їх часто називають, магістралі (системні магістралі). Магістраль (шина) – це сукупність визначеної кількості провідників (розрядність шини), для якого строго нормовані рівні «0» та «1».

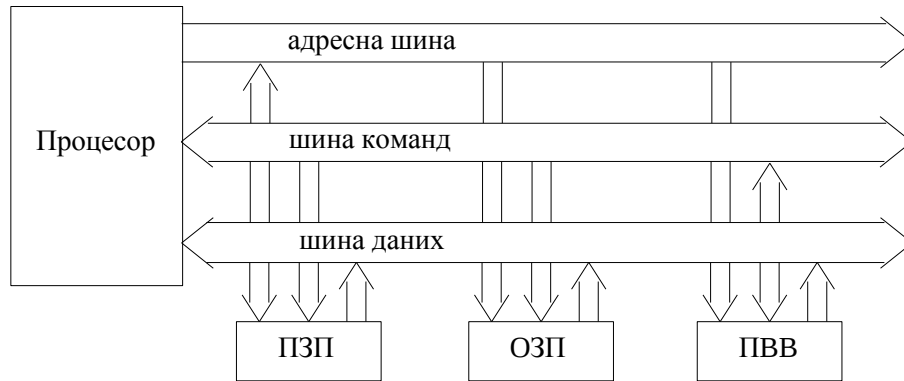


Рис. 1.7.

Всі пристрої мікропроцесорної системи поєднуються загальною системною шиною (вона ж називається системною магістраллю або каналом). Системна магістраль містить у собі чотири основні шини:

- шани адресу (Address Bus);
- шина даних (Data Bus);
- шина команд (Control Bus);
- шина живлення (Power Bus).

Велика перевага шинної структури зв'язків полягає в тому, що всі пристрої, підключені до шини, повинні приймати і передавати інформацію, використовуючи одні і ті ж правила (протоколам обміну інформацією із шиною). Відповідно, всі вузли, відповідальні за обмін із шиною в цих пристроях, повинні бути уніфіковані.

При шинній структурі зв'язків (рис. 1.7) всі сигнали між пристроями передаються по тих самих лініях зв'язку, але в різний час (це називається мультиплексованою передачею). Причому передача по всіх лініях зв'язку може здійснюватися в обох напрямках. У результаті кількість ліній зв'язку істотно скорочується, а правила обміну (протоколи) спрощуються. Вказана взаємодія процесора з зовнішньою пам'яттю – оперативною та постійною, показана на рис. 1.7.

Шина адреси призначена для передачі адреси пристрою, або елементів пам'яті, з якими процесор обмінюється інформацією в поточний момент часу. Кожен зовнішній пристрій, за винятком процесора має свою адресу. Коли блок адресації процесора генерує адресний код, то пристрій, до якого йде звернення, активізується (підготовлюється до обміну даними). Кількість адресів A , по яких процесор може звертатись до зовнішніх пристроїв та пам'яті пов'язано з кількістю розрядів шини формулою $A=2^N$. Розрядність адресної шини N завжди кратна 4 і досягає 64 і більше.

Шина даних – це головна шина по якій відбувається обмін інформацією між всіма пристроями мікропроцесорної системи. Шина завжди двонапрявлена і може використовуватись як для обміну інформацією між процесором і будь-яким пристроєм, в тому числі з запам'ятовуючими пристроями, так і між окремими зовнішніми пристроями. Кількість розрядів шини визначає швидкість та ефективність інформаційного обміну і вона визначає розрядність системної магістралі. Кількість розрядів шини даних сучасних процесорів коливається від 8 до 64. Створені процесори і з більшою розрядністю шини даних. Збільшення розрядності шини даних дозволяє збільшити кількість інформації, що передається за один такт.

В ряді мікропроцесорних систем адресна шина та шина даних з метою скорочення кількості виводів об'єднуються і працюють в режимі мультиплексування. В таких випадках спочатку по шині подається адреса, а потім передаються, або приймаються дані. Для керування режимом мультиплексування і, відповідно, режимом обміну, використовуються окремі виводи шини керування. Для підвищення ефективності роботи системних шин, та швидкості обміну, досить часто вслід за адресним кодом передається (приймається) не один код даних, а цілий масив (блок). Такий режим використовується в швидкодіючих шинах персональних комп'ютерів. На рис. 1.8 приведено схематичне зображення передачі інформації в немультимплексованих системних шинах, (рис. 1.8 а), мультимплексованих з одним кодом даних (рис. 1.8 б) та мультимплексованих з пакетною формою передачі даних (рис. 1.8 в)

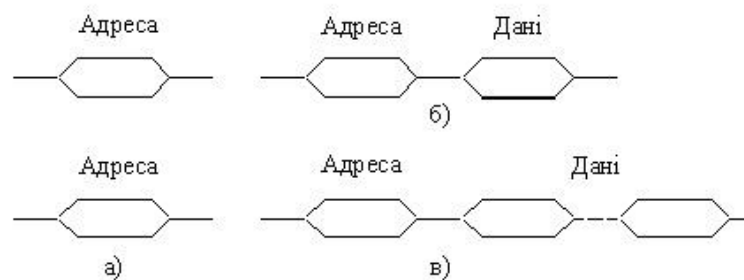


Рис. 1.8.

Шина керування (команд) на відміну від шини адреси і шини даних, складається з окремих керуючих сигналів. Деякі сигнали обслуговують дані, що передаються або приймаються (тобто визначають моменти часу, коли інформаційний код виставлений на шину даних). Інші керуючі сигнали можуть використовуватися для підтвердження прийому даних, для скидання всіх пристроїв у вихідний стан, для тактування всіх пристроїв і т.д. Лінії шини керування можуть бути однонаправленими або двонаправленими. Найбільш важливими керуючими сигналами є строби обміну, тобто сигнали, що формуються процесором для визначення моментів часу в які відбувається пересилка даних по шині. Практично в магістралі виділяють два строби обміну:

– строб запису (виводу), який визначає момент часу, коли пристрій – приймач може приймати дані, які виставлені процесором на шину даних;

– строб читання (вводу), що визначає момент часу, коли пристрій – виконавець повинен видати на шину даних код даних для зчитування процесором.

При цьому важливо, як процесор закінчує обмін, в який момент часу він знімає з відповідного провідника шини керування свій строб обміну. Використовуються два способи:

– при *синхронному обміні* процесор закінчує обмін даними самостійно, через раз і назавжди встановлений інтервал витримки часу t_B , тобто без врахування часових властивостей зовнішнього пристрою;

– при *асинхронному обміні* процесор закінчує обмін лише тоді, коли зовнішній пристрій підтверджує виконання операції спеціальним сигналом H_s (handshake – рукоштовпання).

Часові діаграми, що пояснюють обидва режими, наведені на рис. 1.9, а, б, відповідно.

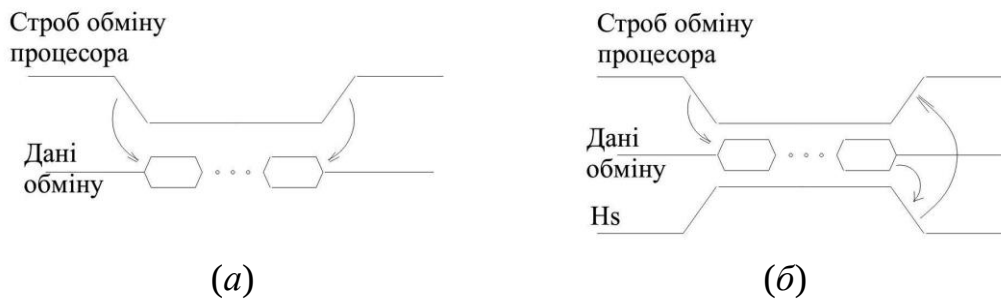


Рис. 1.9.

Синхронний режим має більш простий протокол обміну, але при цьому відсутня гарантія, що зовнішній пристрій виконав необхідну операцію. До того ж зовнішні пристрої при таких протоколах обміну повинні мати високу швидкодію, або тривалість стробу повинна бути настільки високою, щоб забезпечувався надійний обмін з найменш швидкими зовнішніми пристроями.

Асинхронний обмін більш надійний, дає можливість працювати з різними по швидкодії зовнішніми пристроями, але в таких випадках необхідно, щоб всі пристрої, що приймають участь в обміні інформацією з процесором могли подавати сигнали підтвердження. Це призводить до ускладнення апаратних засобів.

Переваги того чи іншого способу визначаються розробником мікропроцесорної системи. Іноді передбачається можливість використання обох способів обміну – синхронний при цьому повинен бути основним і працювати з швидкими периферійними пристроями, а асинхронний – розрахований на повільних виконавців.

Шина живлення призначена для живлення системи. Вона складається з ліній живлення і загального проводу. У мікропроцесорній системі може

бути одне джерело живлення (частіше +5 В) або кілька джерел живлення (зазвичай це +5 В, +12 В та –12 В). Кожній напрузі живлення відповідає своя лінія зв'язку. Всі пристрої підключені до цих ліній паралельно.

Якщо в мікропроцесорну систему треба ввести вхідний код (або вхідний сигнал), то процесор по шині адреси звертається до потрібного пристрою вводу/виводу і приймає по шині даних вхідну інформацію. Якщо з мікропроцесорної системи треба вивести вихідний код (або вихідний сигнал), то процесор звертається по шині адреси до потрібного пристрою вводу/виводу і передає йому по шині даних вихідну інформацію.

Якщо інформація повинна пройти складну багатоступінчасту обробку, то процесор може зберігати проміжні результати в системній оперативній пам'яті. Для звертання до будь-якої комірки пам'яті процесор виставляє її адрес на шину адреси і передає інформаційний код по шині даних або ж приймає з неї інформаційний код. У пам'яті (оперативній і постійній) знаходяться також і керуючі коди (команди програми), які процесор також читає по шині даних з адресацією по шині адресу. Постійна пам'ять використовується в основному для зберігання програми початкового пуску мікропроцесорної системи, що виконується щоразу після вмикання живлення.

Таким чином, у мікропроцесорній системі всі інформаційні коди і коди команд передаються по шинах послідовно, по черзі. Це визначає порівняно невисоку швидкість мікропроцесорної системи. Вона обмежена навіть не швидкістю процесора (яке теж дуже важлива) і не швидкістю обміну по системній магістралі, а саме послідовним характером передачі інформації по системній магістралі.

1.4. Режими роботи мікропроцесорної системи

Практично будь-яка мікропроцесорна система підтримує три основних режими обміну по магістралі:

- програмний обмін інформацією;
- обмін з використанням переривань (Interrupts);
- обмін з використанням прямого доступу до пам'яті (ПДП, DMA – Direct Memory Access).

1.4.1. Програмний обмін інформацією

Програмний обмін інформацією є основним у будь-якій мікропроцесорній системі. Він передбачений завжди, без нього неможливі інші режими обміну. Всі операції (цикли) обміну інформацією ініціюються тільки процесором, всі вони виконуються строго в порядку, що визначається програмою.

Процесор зчитує (вибирає) з пам'яті коди команд і виконує їх, читаючи дані з пам'яті або із пристрою вводу/виводу, обробляючи їх, записуючи дані у пам'ять або передаючи їх у пристрій вводу/виводу. Шлях процесора по програмі може бути лінійним, циклічним, може містити переходи (стрибки), але він завжди безперервний і повністю перебуває під

контролем процесора. Ні на які зовнішні події, не пов'язані із програмою, процесор не реагує. Всі сигнали на магістралі в цьому випадку контролюються процесором. На рис. 1.10 представлена загальна схема програмного обміну інформацією.

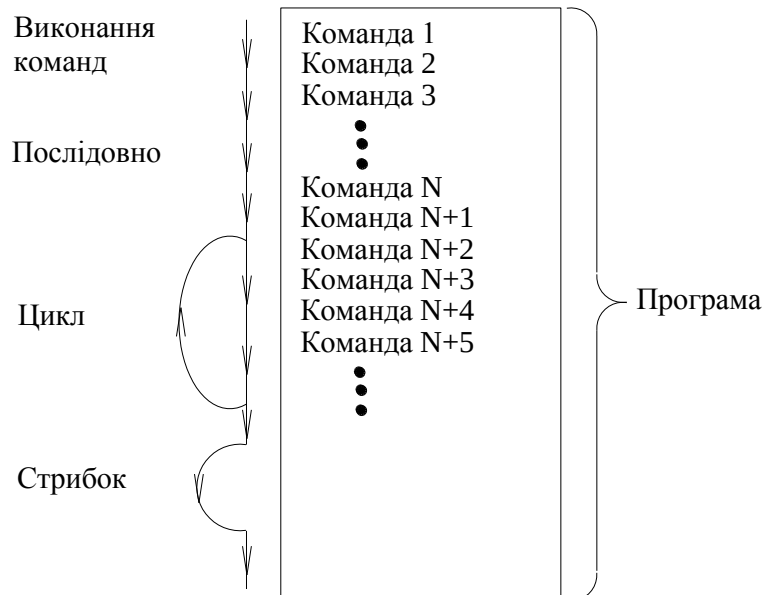


Рис. 1.10.

1.4.2. Обмін по перериваннях

Обмін по перериваннях використовується тоді, коли необхідна реакція мікропроцесорної системи на якусь внутрішню або зовнішню подію, на прихід зовнішнього сигналу. У випадку комп'ютера зовнішньою подією може бути, наприклад, натискання на клавішу клавіатури або прихід по локальній мережі пакета даних. Комп'ютер повинен реагувати на це, відповідно, виводом символу на екран або ж зчитуванням і обробкою прийнятого по мережі пакета.

У загальному випадку організувати реакцію на зовнішню подію можна трьома різними шляхами:

- за допомогою постійного програмного контролю факту настання події (так званий метод опитування прапорця);
- за допомогою переривання, тобто переводу процесора з виконання поточної програми на негайне виконання необхідної програми;
- за допомогою прямого доступу до пам'яті, тобто без участі процесора при його відключенні від системної магістралі.

Перший випадок з опитуванням прапорця реалізується в мікропроцесорній системі постійним зчитуванням інформації процесором із пристрою вводу/виводу, пов'язаного з тим зовнішнім пристроєм, на поведінку якого необхідно терміново реагувати.

У другому випадку в режимі переривання процесор, одержавши запит переривання від зовнішнього пристрою (часто називаний IRQ – Interrupt

ReQuest), закінчує виконання поточної команди і переходить до програми обробки переривання. Закінчивши виконання програми обробки переривання, він повертається до перерваної програми з того місця, де його перервали (рис. 1.11).

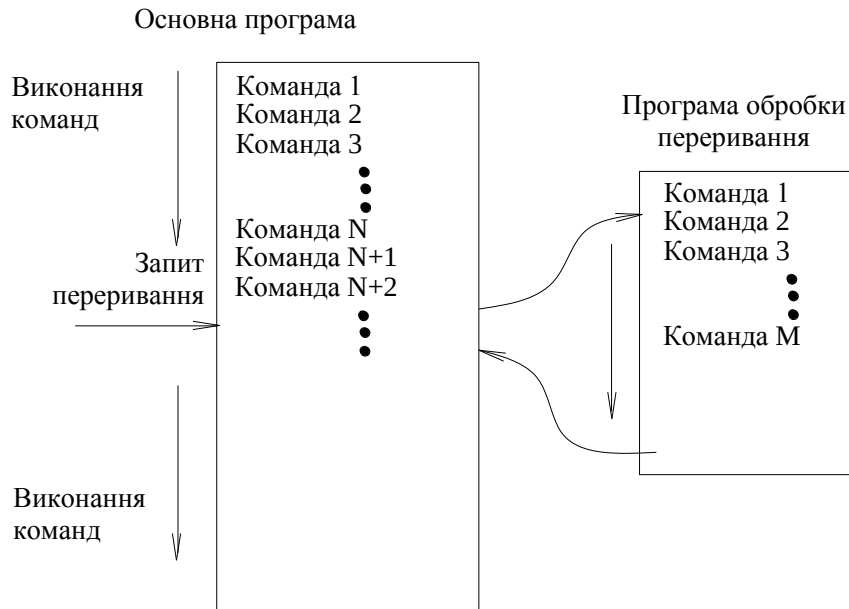


Рис. 1.11.

Важливим є те, що вся робота, як і у випадку програмного режиму, здійснюється самим процесором, зовнішня подія просто тимчасово відволікає його. Реакція на зовнішню подію по перериванню в загальному випадку повільніша, ніж при програмному режимі. Як і у випадку програмного обміну, тут всі сигнали на магістралі виставляються процесором, тобто він повністю контролює магістраль. Для обслуговування переривань у систему іноді вводиться спеціальний модуль контролера переривань, але він в обміні інформацією не бере участь. Його завдання полягає в тому, щоб спростити роботу процесора із зовнішніми запитами переривань. Цей контролер зазвичай програмно керується процесором по системній магістралі.

1.3.3. Прямий доступ до пам'яті

Прямий доступ до пам'яті – це режим, що принципово відрізняється від двох раніше розглянутих режимів тим, що обмін по системній шині йде без участі процесора. Зовнішній пристрій, що вимагає обслуговування, сигналізує процесору, що необхідний режим ПДП, у відповідь на це процесор закінчує виконання поточної команди і відключається від всіх шин, сигналізуючи пристрою, що обмін у режимі ПДП можна починати.

Операція ПДП зводиться до пересилання інформації із пристрою вводу/виводу в пам'ять, або ж з пам'яті в пристрій вводу/виводу. Коли пересилання інформації буде закінчено, процесор знову повертається до перерваної програми, продовжуючи її з того місця, де його перервали (рис.

1.12). Це схоже на режим обслуговування переривань, але в цьому випадку процесор не бере участь в обміні. Як і у випадку переривань, реакція на зовнішню подію при ПДП істотно повільніша, ніж при програмному режимі.

Зрозуміло, що в такому випадку зовнішній пристрій повинен бути забезпечений більш функціональним пристроєм, оскільки він повинен «розпізнавати» ряд спеціальних сигналів, генерувати адресні сигнали і забезпечувати обмін інформацією з зовнішньою пам'яттю, подібно центральному процесорові. Причому процесор попередньо повинен повідомити контролер ПДП, звідки йому брати інформацію і куди її передавати. Контролер ПДП може вважатися спеціалізованим процесором, який відрізняється тим, що сам не бере участь в обміні, не приймає в себе інформацію і не видає її (рис. 1.13).

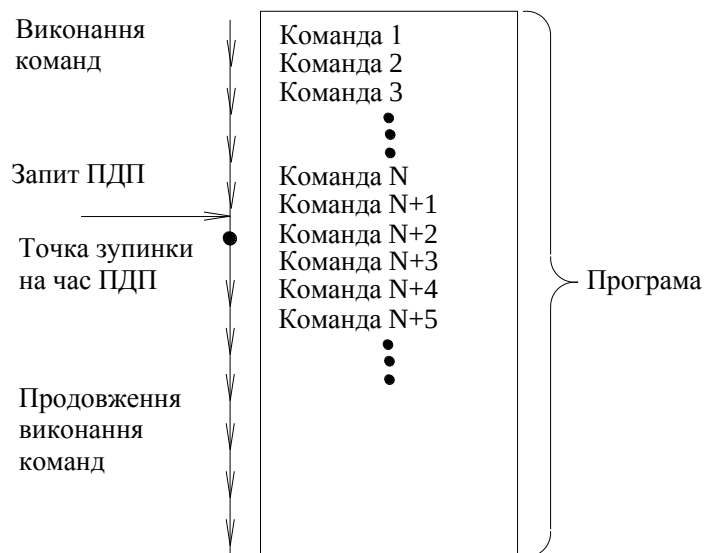


Рис. 1.12.

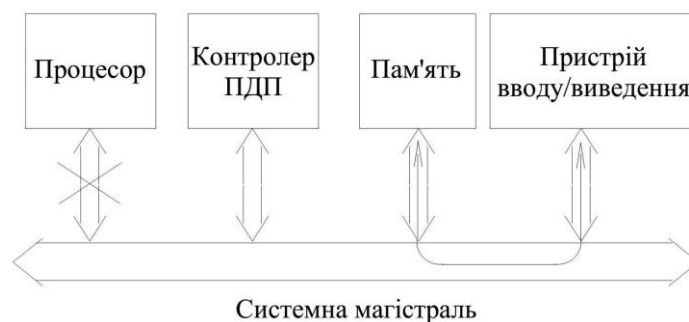


Рис. 1.13.

Теоретично обмін за допомогою прямого доступу до пам'яті може забезпечити більш високу швидкість передачі інформації, ніж програмний обмін, тому що процесор передає дані повільніше, ніж спеціалізований контролер ПДП. Однак на практиці ця перевага реалізується далеко не

завжди. Швидкість обміну в режимі ПДП зазвичай обмежена можливостями магістралі. До того ж, необхідність програмного завдання режимів контролера ПДП може звести нанівець вигоду від більш високої швидкості пересилання даних у режимі ПДП. Тому режим ПДП застосовується рідко. Якщо в системі вже є самостійний контролер ПДП, то це може в ряді випадків істотно спростити апаратну реалізацію пристроїв вводу/виводу, що працюють у режимі ПДП. В цьому є перевага режиму ПДП.

1.5. Загальна характеристика пристроїв пам'яті мікропроцесорної системи

Запам'ятовуючі пристрої будь – якої мікропроцесорної системи відіграють практично не менш важливу роль, ніж самі процесори. Обсяг пам'яті, що використовується процесором, та її типи визначають складність тих задач, які може розв'язувати система, швидкість обробки інформації. Розрядність пристроїв пам'яті завжди співпадає з розрядністю шини даних і практично завжди кратна 8. Обсяги пам'яті вимірюються в байтах, або в відповідних пропорційних одиницях: Кбайт (кілобайт) – 2^{10} байт; Мбайт (мегабайт) – 2^{20} байт; Гбайт (гігабайт) – 2^{30} байт; Тбайт (терабайт) – 2^{40} байт) Практично всі мікропроцесорні системи в своєму розпорядженні мають постійні запам'ятовуючі пристрої, та оперативні (ОЗП)

ПЗП виготовляються по технології EEPROM (з електричним записом і стиранням), які в контролерах розміщуються на тому ж кристалі, що і процесор, а для потужних мікропроцесорних систем використовуються окремі мікросхеми. В ПЗП записується та частина програми, що не змінюється в процесі роботи системи. Наприклад, для потужних систем в ПЗП записується програма початкового запуску.

Зв'язок ПЗП з системною шиною досить простий. В системах на основі мікроконтролерів для звернення до елементів пам'яті необхідно мати лише дешифратори адреси, які здебільшого розміщуються в мікросхемах пам'яті. В складних системах іноді необхідними елементами є схема обробки керуючих сигналів та буфери даних.

В залежності від складності мікропроцесорної системи, в якості оперативних запам'ятовуючих пристроїв використовується статичний та динамічний тип пам'яті. Оскільки динамічна пам'ять вимагає постійної регенерації, що ускладнює роботу з нею, то вона використовується в потужних мікропроцесорних системах, в персональних комп'ютерах. В контролерах використовується статична пам'ять. ОЗП взаємодіють з системою в циклах запису і читання. Одночасно з адресацією до ОЗП по шині керування подаються сигнали дозволу на запис або читання. Зв'язок з шиною даних забезпечується через відповідні буфери даних. Таким чином, для приєднання модуля пам'яті до системної шини використовуються відповідні блоки, що містять в собі дешифратор адреси (селектор адреси), схему обробки керуючих сигналів і буфери даних. В будь – якій

мікропроцесорній системі простір пам'яті строго розділений на окремі області визначеного розміру. Вони залежать від складності і призначення мікропроцесорної системи і будуть детально розглянуті пізніше.

1.6. Методи адресації до зовнішньої пам'яті

Більша частина команд та даних, над якими виконуються операції, знаходяться в зовнішній по відношенню до процесора пам'яті – в ОЗП або ПЗП. Команди, що виймаються з зовнішньої пам'яті для виконання повинні працювати не з будь – якими даними, а з конкретними, які також необхідно виймати з зовнішньої пам'яті. Тому методологія взаємодії процесора з зовнішньою пам'яттю повинна бути направлена на те, щоб в максимальній мірі зменшити кількість звернень процесора до ОЗП та ПЗП. Це досягається відповідно організованою системою адресації в якій закладається можливість через код команди визначати, звідки брати вхідний операнд і куди поміщати вихідний. Така строго формалізована методологія, що є характерною для кожного конкретного типу мікропроцесорів і називається методами адресації. Ефективність вибраних методів адресації суттєво впливає на ефективність роботи мікропроцесорної системи в цілому. Використовуються наступні способи формування адресів ОЗП.

Безпосередня адресація – адреса елемента пам'яті або декількох адресів формується безпосередньо в БА. Операнд, що використовується командою за даним способом адресації знаходиться в пам'яті за адресою, що безпосередньо слідує за елементом пам'яті команди. (рис. 1.14)



Рис. 1.14.

Пряма (абсолютна) адресація – передбачає, що операнд, що використовується командою, або отримується в результаті виконання команди знаходиться за адресою, що розміщена в адресному просторі програм безпосередньо за командою (рис. 1.15).

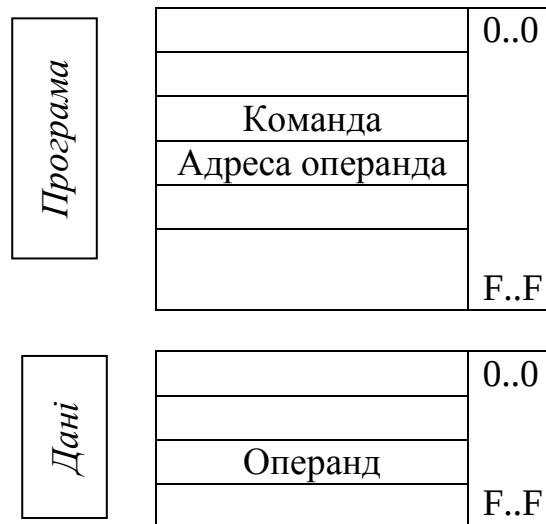


Рис. 1.15.

Регістрова адресація (двоадресні команди, команди між регістрових пересилок) – передбачає, що операнд знаходиться в одному з РЗП. Код операції містить в собі як адресу регістра з операндом, так і регістра, в який необхідно переслати цей операнд.

Відносна адресація – з БА виймається частина адресного слова і до нього додається (за правилами арифметики) друга його частина, яка може знаходитись, наприклад, в одному з РЗП.

Побічна адресація. В блоці РЗП завжди є декілька регістрів спеціального призначення. В цих регістрах розміщується не операнд, а його адреса в ОЗП. Тому використання цих регістрів говорить про те, що в таких операціях приймає участь блок адресації.

Автоінкрементна адресація подібна до побічної адресації. Різниця полягає в тому, що після виконання команди вміст регістра, що використовується, інкрементується на 2 (при виконанні операцій над словами) і на 1 при виконанні операцій над байтами (в мікропроцесорних системах, що використовуються в обчислювальній техніці операнди розділяються на байт, слово (два байти) і подвійне слово (4 байти)).

Автодекрементна адресація – подібна до попередньої з різницею в тому, що вміст регістра не збільшується, а зменшується.

Індексна адресація – відрізняється від автоінкрементної тим, що вміст регістра, що використовується збільшується на постійну величину, яка називається індексом, і розміщується в ОЗП безпосередньо за кодом команди

Від вибору того чи іншого способу адресації залежить швидкість виконання команди. Найбільш швидка адресація – це регістрова, оскільки вона не вимагає використання системної магістралі. Якщо ж адресація вимагає звернення до ОЗП, то тривалість виконання операції буде зростати на кількість необхідних циклів системної магістралі. Тривалість виконання операцій буде зростати ще більше, якщо в формуванні адреси буде

приймати участь АЛП. Зрозуміло, що чим більше внутрішніх регістрів, тим більше для виконання програми буде використовуватись команд з між регістровою адресацією і тим швидше буде виконуватись програма.

1.7. Обмін інформацією в мікропроцесорних системах

Самим низьким рівнем, який дозволяє описувати роботу цифрових пристроїв – це рівні логічних станів їх входів та виходів – таблиці станів. Наступним рівнем способу описання – це мова значень вхідних та вихідних сигналів, що складають мову мікрокоманд та записуються багаторозрядними кодами. Сукупність коду операції та керуючих сигналів називають мікрокомандою. Третій рівень формалізації описання роботи мікропроцесора – це мова команд – тобто строга послідовність мікрокоманд, що записується в пам'яті мікропроцесора. Тобто, команда, це слово, або набір слів, які дешифруються в блоці керування в послідовність мікрокоманд. Звідси витікає, що будь-який процесор має строго фіксований і обмежений набір команд, який є характерним для даного процесора.

Будь-яка команда характеризується своїм форматом. Під форматом команди розуміється її протяжність та призначення кожного біта або їх групи.

Команди, також мають свій фіксований формат. (Протяжність мікрокоманди – це стандартна для даного процесора кількість біт в слові АЛП). В залежності від протяжності команди, вона може складатися з одного, двох, та трьох слів.

Структура команд повністю залежить від структури мікропроцесора, але незалежно від типу процесора прийнято вважати, що однослівні команди повністю складаються з коду операції. До таких операцій відносяться, наприклад, команди міжрегістрових пересилок. Двослівні команди складаються з коду операції та однослівного операнда D^N (N – кількість розрядів в слові). Це може бути операнд, над яким повинна виконуватись операція. Трислівні команди також складаються з двох частин: перша частина – код операції, а друга – адреса (величина, адресного коду завжди перевищує розміри операнда), або двослівний операнд D^{2N} .

Формат пам'яті мікропроцесорної системи також тісно пов'язаний з довжиною слова. Тому при зберіганні таких команд відповідно використовується адресний простір та пам'ять. Якщо, наприклад, команда складається з трьох слів, а використовується з послідовною адресацією, то для зберігання такої команди використовуються три послідовні адреси. Для того, щоб таку команду вибрати з пам'яті, необхідно мати спеціальні засоби, щоб забезпечити її представлення в ПК як єдине ціле.

Типи команд що використовуються мікропроцесором тісно пов'язані з внутрішньою організацією, алгоритмом функціонування

мікропрограмного автомата (БК) процесора, та внутрішньою системою синхронізації.

Мікропроцесорна система функціонує синхронно з частотою тактових сигналів зовнішнього генератора. В залежності від типу мікропроцесорів використовується одно— або двохфазна синхронізація. Незалежно від цього, в мікропроцесорних системах використовуються більш тривалі інтервали часу, ніж тактовий інтервал зовнішнього генератора. Одним з таких інтервалів є **машинний цикл** – це інтервал часу, протягом якого мікропроцесор звертається до пам'яті або пристрою вводу/виводу. Машинний цикл (МЦ) складає тільки частину циклу команди. На початку кожного МЦ на одному з виходів МП з'являється сигнал синхронізації. Він передається по лінії шини керування в ОЗП або пристрої вводу/виводу і “сповіщає” про початок нового МЦ. В результаті досягається узгодження в часі зовнішніх пристроїв з роботою мікропроцесора.



Рис. 1.16.

Цикл команди – це інтервал часу, необхідний для вибірки з пам'яті команди, та її виконання. Він складається з 1–5 машинних циклів. Їх конкретне число залежить від складності операції, яка виконується в даній команді і дорівнює числу звернень мікропроцесора до пам'яті. Тривалість виконання команди визначається кількістю тактів в циклі команди та тривалістю такту (рис. 1.16).

Протягом циклу команди, що ділиться на дві фази, робота МП виконується в такій послідовності. Пристрій керування задає початок чергового циклу шляхом формування сигналу, по якому число, що знаходиться в лічильнику команд, відправляється в буферний регістр адреси і через нього направляється в зовнішню пам'ять (ЗП), де дешифрується. Після приходу від МП сигналу керування RD з елемента пам'яті, що знаходиться по вказаній адресі, зчитується слово команди, яке подається по шині даних в буферний регістр даних, а потім в пристрій керування, де дешифрується з допомогою коду операції. Ця послідовність операцій називається **фазою вибірки**. За нею слідує виконавча фаза, в якій пристрій керування формує послідовність сигналів, необхідних для виконання команди. За цей час число, що знаходиться в лічильнику команд, збільшується на 1 і формується адреса команди, що стоїть слідом

за тою, що виконується. Вона зберігається в лічильнику до приходу сигналу, що задає початок чергового циклу команди.

Окрім адреси елемента в якому зберігається необхідний байт від МП до ЗП поступає сигнал по шині керування, який визначає характер операції – запис, або зчитування. Виконання вказаних операцій проходить протягом інтервалу часу, що називається часом доступу. По закінченні цього інтервалу від ЗП в МП подається сигнал готовності, який є сигналом початку прийому, або, відповідно, передачі сигналів в ЗП. До одержання сигналу готовності МП знаходиться в стані очікування. Інтервал часу між імпульсами звернення до зовнішніх пристроїв та одержання від них відповіді називається **циклом очікування**.

Якщо, наприклад, цикл команди розглядати відповідно до команди вводу даних, то перші два машинних цикли будуть відноситись до фази вибірки, а третій – до **фази виконання команди**. В усіх машинних циклах передається адреса, але в кожному циклі адреса належить своєму адресату, в першому – це адреса елемента, де зберігається код операції, в другому – адреса порту, що зберігає байт даних, в третьому – адреса акумулятора мікропроцесора, куди повинен поступити байт даних з порту.

Використовуються різні способи організації послідовності виконання команд, наприклад переривання Організація обміну по перериванням будується на основі тих же принципів, що і при програмному обміні, але мають ряд особливостей.

Перериванням називається послідовність дій, в результаті яких виконання даного алгоритму тимчасово припиняється і замість цього починає виконуватись інший алгоритм, що призначений для обслуговування переривання. Для процесора переривання – це двійковий код, що поступає від зовнішніх пристроїв через порт керування. В залежності від складності процесора може існувати від 1-2-х до декількох десятків таких кодових комбінацій. Кожному з вказаних кодів, які часто називають **векторами переривання**, відповідає спеціально створена і записана в оперативній пам'яті програма, яку повинен виконати процесор, якщо такий код появиться на входах порту керування. Щоб потім повернутись до виконання перерваного алгоритму, в перелік дій по реалізації переривань входять дії по перезапису вихідного коду лічильника адресів команд (ЛАК) в спеціальний буферний регістр, що називається **стек**. В якості стека використовуються відомі буфери LIFO. В ряді випадків стек реалізується програмним шляхом з вибраної ділянки оперативної пам'яті. При багатосходиновому перериванні, тобто у випадку, коли під час обслуговування програми переривання, в процесор поступає сигнал на обслуговування іншого переривання, створюється послідовність кодів ЛАК, яка розміщується в стеку в заданому порядку. Цей порядок забезпечується спеціальним пристроєм – **показчиком стеку** (ПС), який також входить до складу блоку адресації.

Поява на вході порту керування процесора коду переривання розглядається як „**запит на переривання**”. Оскільки вектори переривання і, відповідно, „запити на переривання” можуть поступати на входи процесора в довільні інтервали часу, то створюється спеціальний порядок обслуговування переривань. Його особливість полягає в тому, що на програмному рівні встановлюються пріоритети векторів переривань. При появі на вході процесора запиту на переривання від зовнішнього пристрою процесор закінчує виконання поточної команди і в блоці адресації формується адреса чергової команди. Але остання команда не виймається з ОЗП, а її адреса розміщується в стек. Після „розпізнавання” отриманого запиту на переривання в ЛАК завантажується початкова адреса програми, що призначена для обслуговування даного переривання. В подальшому, після завершення виконання вказаної програми, в ЛАК завантажується зі стеку адреса останньої невиконаної команди основної програми і продовжується її виконання.

Але часто мають місце ситуації, коли підчас обслуговування програми переривання, на керуючі входи процесора поступають сигнали нового переривання. В такому випадку спрацьовує пріоритетність векторів переривань. Можуть бути наступні ситуації:

- пріоритети однакові для обох векторів;
- пріоритет вектора, що обслуговується, вище, ніж того, що поступив пізніше;
- пріоритет вектора, що обслуговується, нижче, ніж того, що поступив пізніше.

В перших двох випадках спрацьовує послідовність виконання операцій, що вже описана вище. В останньому випадку процесор повинен перейти на обслуговування переривання з вищим пріоритетом, тобто, він закінчує виконання чергової команди програми обслуговування переривання з нижчим пріоритетом, завантажує адресу чергової команди в стек і після цього переходить на обслуговування вектора переривання з вищим пріоритетом. Після виконання останньої програми з стеку виймається адреса команди, що була завантажена останньою і продовжується її виконання до завершення. Кількість програм, що обслуговуються по перериванню, відображається покажчиком стеку. Всі ці програми називаються вкладеними, а їх кількість, що характеризує ємність стеку – **глибиною вкладки**.

Таким чином обмін по перериванням використовується в тих випадках, коли необхідна реакція мікропроцесорної системи на зовнішні події, які проявляються у вигляді зовнішніх сигналів. Наприклад для комп'ютера зовнішніми подіями можуть бути натиснення кнопки „Esc”, прихід даних по мережі і т.п.

Іноді для обслуговування зовнішніх сигналів переривань використовують спеціальний модуль контролера переривань. Такий контролер може брати на себе ряд функцій по визначенню пріоритетів переривань, їх кодуванню, та ін.

Радіальні переривання не вимагають ніяких циклів обміну по магістралі. В магістралі існує стільки ліній запитів на переривання, скільки всього може бути переривань. Кожен зовнішній пристрій, що використовує переривання, подає сигнал запиту по окремій лінії. Процесор знаходить номер переривання по номеру лінії, по якій прийшов сигнал запиту на переривання. Ніяких циклів обміну по магістралі при цьому не потрібно. Досить часто в випадку радіальних переривань в систему включається допоміжна мікросхема – контролер переривань, що опрацьовує сигнали переривань і забезпечує їх доступ до процесора в відповідності з пріоритетами, що встановлюються в контролері. Приклад структури зв'язків при радіальних перериваннях приведено на рис. 1.17.

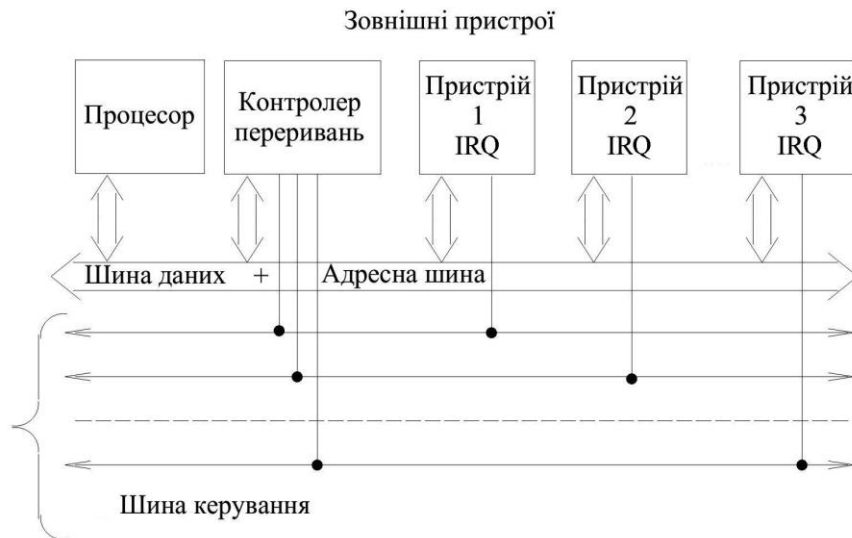


Рис. 1.17.

В загальному плані процесор може обмінюватись інформацією з контролером як через системну магістраль (коли задає йому режим роботи), так і по окремих лініях зв'язку (при обробці запитів на переривання). Кожен зовнішній пристрій має свою лінію запиту на переривання, по якій йде звернення до контролера переривань. При отриманні запиту контролер „розпізнає” переривання, опрацьовує його в відповідності до встановлених пріоритетів і передає в процесор інформацію, на основі якої процесор переходить на опрацювання відповідної підпрограми переривання. Подібний спосіб обслуговування переривань використовується в ISA –шині персональних комп'ютерів.

При **векторному перериванні** код номера переривання передається процесору тим пристроєм вводу /виводу, який подав запит на переривання. Для цього виконує цикл читання по магістралі і по шині даних отримує код номера переривання. Адресна шина в даному циклі не використовується, оскільки пристрій, що подав сигнал запиту на переривання, активізується, а процесор по запиту на переривання переходить в режим читання по магістралі і зчитує номер вектора переривань з шини даних пристрою, що

подало запит. Структура зв'язків при векторному перериванні зображена на рис. 1.18.

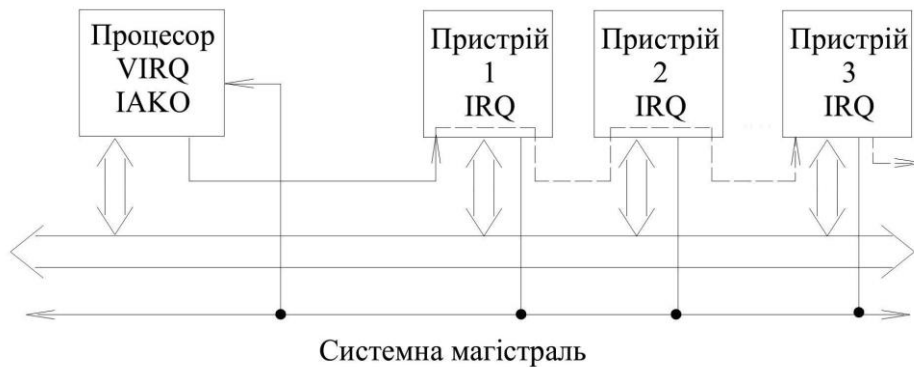


Рис. 1.18.

А часові діаграми, що пояснюють цикл отримання процесором номера вектора переривань пояснюються рис. 1.19.

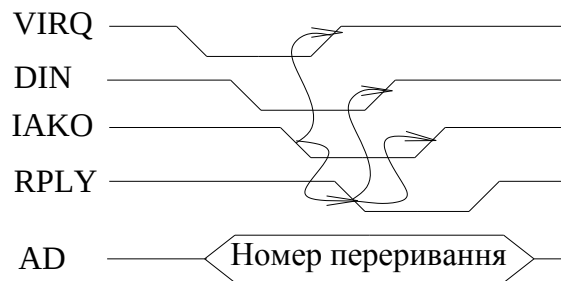


Рис. 1.19.

Запит на переривання виконується негативним сигналом VIRQ, що може формуватись кожним з пристроїв, що подає запит на переривання. Отримавши сигнал VIRQ, процесор в відповідності до описаних вище правил переходить на обслуговування переривання. Для цього він виставляє сигнал читання DIN і сигнал підтвердження IAKO, який послідовно проходить через всі зовнішні пристрої, що можуть запитувати переривання. Якщо відповідний пристрій подав сигнал запиту на переривання, то він не пропустить через себе цей сигнал. В результаті, якщо переривання одночасно запитали два і більше пристроїв, то сигнал IAKO отримає лише пристрій, що ближче до процесора. Такий механізм розв'язання конфліктів називається **географічним пріоритетом**. Отримавши сигнал IAKO, пристрій, що подав запит на переривання, повинен зняти свій сигнал VIRQ. Потім процесор проводить цикл читання номера переривання. В відповідь на отримання сигналів DIN і IAKO пристрій, якому надано переривання, повинен видати на шину даних AD код номера переривання (адресу вектора переривання) і виставити сигнал підтвердження RPLY. Процесор зчитує код номера переривання і закінчує цикл безадресного читання зняттям сигналів DIN і IAKO. Близький до описаного векторного переривання використовувався в перших

вітчизняних мікро контролерах серії 580 і знаходить використання в шині Q-bus.

До типу радіальних переривань слід віднести вхід системної установки – сигнал RESET, який використовується в тих випадках, коли процесор внаслідок якихось причин загубив контроль над виконанням програми і зупинився на якійсь адресі.

Обмін по системній магістралі в режимі ПДП по своїй організації близький до обміну по перериваннях. В малопотужних мікроконтролерних структурах цей режим обміну безпосереднього використання не знайшов. Знаходить досить широке використання в мікропроцесорних системах персональних комп'ютерів, де він використовується для забезпечення обміну інформацією повільних зовнішніх пристроїв з оперативною пам'яттю без безпосередньої участі центрального процесора. Для цього зовнішні пристрої, що приєднуються до комп'ютера, мають власні контролери, що можуть працювати в режимі ПДП, а в адресному просторі оперативної пам'яті виділяється ділянка, в якій розміщується керуюча програма відповідного контролера (драйвер) та область для запису/зчитування даних. Різні контролери, як і їх драйвери, мають свої особливості, і іноді суттєво відрізняються як архітектурою, так і алгоритмом роботи. Але в загальному плані взаємодія контролера ПДП з мікропроцесорною системою полягає в наступному.

При необхідності обміну контролер ініціює запит на захоплення шин, виставляючи на відповідну лінію шини керування сигнал HRQ. Отримавши цей сигнал, центральний процесор, подібно до режиму переривання, переходить на режим внутрішньої обробки даних, і при цьому виставляє сигнал HDLA дозволу на захоплення шин, а також переводить порти адресної шини та шини даних у високоімпедансний стан. В інтервалі часу дії сигналу HDLA контролер ПДП забезпечує обмін інформацією з основною пам'яттю у визначеному адресному просторі. Час обміну характеризується кількістю циклів, які контролюються лічильником контролера ПДП. Якщо сигнал HDLA знімається, то обмін призупиняється і відновлюється знову, якщо сигнал HRQ не знятий. Контролери ПДП також мають свої пріоритети, подібно до режиму переривань.

Особливості протікання обміну по магістралі. Незважаючи на зовнішню простоту, системна магістраль завжди завдає великого клопоту розробникам мікропроцесорних систем. Досить велика протяжність магістралей приводить до необхідності врахування реальних розподілених індуктивностей, конденсаторів, активних опорів. Все це приводить не лише до значних затримок сигналів, що проходять по магістралі, а і до спотворення форми самих сигналів, що суттєво впливає на синхронізацію обміну. Реально, виходячи з відомих розділів курсу ТОЕ, на проходження сигналів по магістралі впливають наступні фактори:

- реальні величини затримок в розповсюдженні сигналів по лініях магістралі;
- різниці в затримках сигналів по різних лініях шини;
- неодночасність „виставлення” сигналів на лінії шини;
- спотворення фронтів сигналів, що проходять по магістралі;
- відбиття сигналів від кінців лінії зв’язку.

Для усунення можливих проблем, обумовлених вказаними факторами, розробники стандартних магістралей обміну і стандартних протоколів завжди передбачають затримки між сигналами. Сучасні засоби САПР радіоелектронних пристроїв також мають можливість підрахувати можливі величини затримок сигналів по магістралі. Затримки вибираються так, щоб сигнали уже були підготовлені, перш ніж буде поданий синхросигнал, а також, щоб будь – який пристрій мав досить часу для його опрацювання. З такої точки зору спроби якимось чином модернізувати стандартний протокол обміну, наприклад, зменшити величини затримок, підняти частоту тактового генератора процесора, збільшити допустиму довжину магістралі недопустимі.

Для покращення форми сигналів, що розповсюджуються по магістралі іноді на кінцях ліній магістралі встановлюють узгоджуючі опори. Це дає можливість усунути можливість появи відбитої хвилі, особливо в тих випадках, коли використовується магістраль значної довжини. Але такий підхід приводить до допоміжного навантаження буферів, що працюють на шинах.

1.7.1. Архітектури мікропроцесорних систем

Розглянута структура мікропроцесорної системи характеризується системною шиною, що містить в собі три окремі шини – адресну, керування, та команд і даних. Архітектура з двома шинами, коли об’єднуються адресна шина та шина команд і даних не вносить принципово ніяких особливостей в функціонування системи, оскільки в інтервалі обміну командами і даними між процесором та пам’яттю або зовнішніми пристроями, адресна шина не функціонує. Описана архітектура має назву Прістонська, або фон–Нейманівська (рис. 1.20). Вона характеризується також тим, що пам’ять в ній також об’єднана, а команди і дані можуть розміщуватись довільним шляхом в рамках адресного простору ЗП.

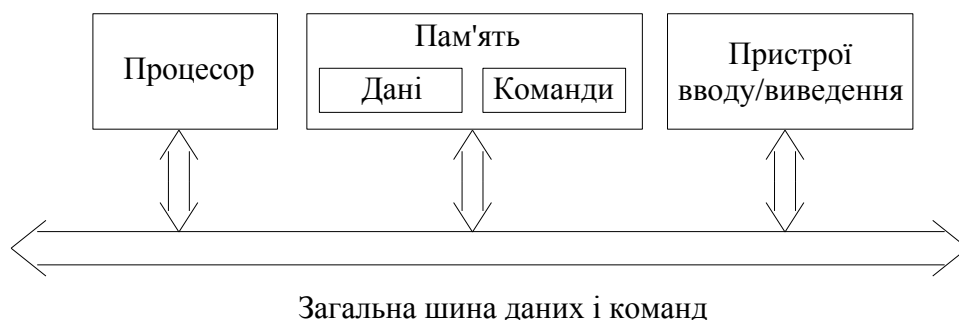


Рис. 1.20.

Недолік такої архітектури очевидний, оскільки команди і дані виймаються з ОЗП по – черзі, збільшуючи тим самим тривалість циклу команди. Незважаючи на вказаний недолік, така архітектура знаходить широке використання в персональних комп'ютерах, та інших обчислювальних системах. Вона має ряд переваг, оскільки вона простіша, не вимагає від процесора одночасного обслуговування двох шин. Наявність одної пам'яті дозволяє гнучко розподіляти її між командами та даними, вільно розміщати як складні програми, що працюють з невеликою кількістю даних, так і програми, що мають невелику кількість команд, а великі масиви даних. Обсяг оперативної пам'яті в таких системах досить великий (досягає одиниць гігабайт), що дозволяє розв'язувати самі різноманітні задачі.

Для усунення вказаного недоліку запропонована архітектура мікропроцесорної системи з розділеними шинами команд та даних (гарвардська архітектура). Вона також передбачає розділення пам'яті для даних і для команд.

Архітектура з розділеними шинами даних та команд (рис. 1.21) більш складна, вона вимагає від процесора обслуговувати дві шини, два потоки кодових послідовностей одночасно. При цьому програма розміщується лише в пам'яті команд, а дані – лише в пам'яті даних. Перерозподіл пам'яті при цьому неможливий, що реально обмежує обсяги пам'яті, що використовується. Тому така архітектура може забезпечити швидке розв'язання порівняно нескладних задач, тобто, вона знаходить своє використання в швидкодіючих мікроконтролерах, що виготовляються в рамках однієї мікросхеми.

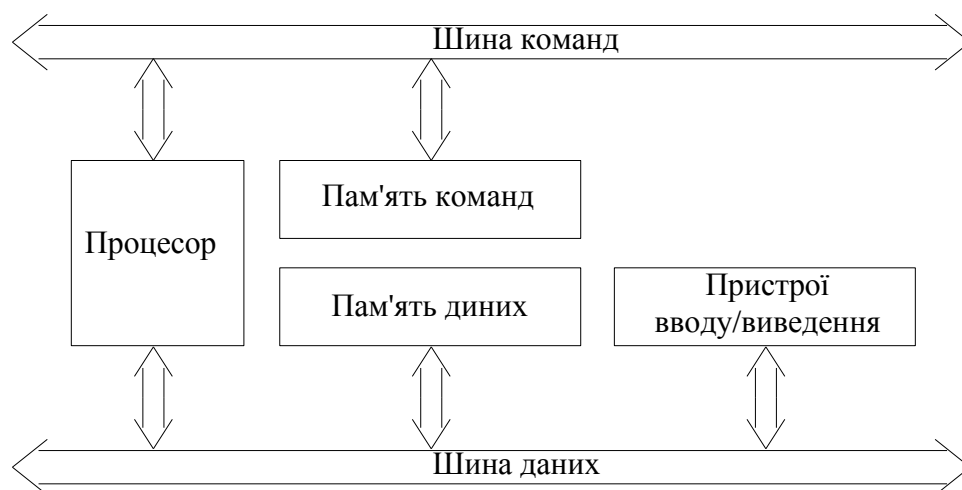


Рис. 1.21.

Архітектура з розділеними шинами команд та даних має ряд переваг. При єдиній шині команд і даних процесор змушений по одній шині приймати дані (з пам'яті або пристрою вводу/виводу) і передавати дані (в пам'ять або пристрій вводу/виводу), а також читати команди з пам'яті. Одночасно ці пересилання кодів по магістралі відбуватися не можуть, вони

повинні виконуватися по черзі. Сучасні процесори здатні поєднати в часі виконання команд і проведення циклів обміну по системній шині. Використання конвеєрних технологій і швидкої кеш-пам'яті дозволяє їм прискорити процес взаємодії з порівняно повільною системною пам'яттю. Підвищення тактової частоти та удосконалювання структури процесорів дають можливість скоротити час виконання команд. Але подальше збільшення швидкодії системи можливо тільки при сполученні передачі даних і зчитування команд, тобто при переході до архітектури із двома шинами.

У випадку архітектури з роздільними шинами команд та даних обмін по обох шинах може бути незалежним, паралельним у часі. Відповідно, структури шин (кількість розрядів коду адреси та коду даних, порядок і швидкість обміну інформацією) можуть бути обрані оптимально для того завдання, що вирішується кожною шиною. Тому за інших рівних умов перехід на архітектуру з двома шинами прискорює роботу мікропроцесорної системи, хоча і вимагає додаткових апаратних витрат, ускладнення структури процесора. Пам'ять даних у цьому випадку має свій розподіл адресів, а пам'ять команд – свій.

1.7.2. Типи мікропроцесорних систем

Діапазон застосування мікропроцесорної техніки зараз дуже широкий, вимоги до мікропроцесорних систем пред'являються самі різні. Тому сформувалося кілька типів мікропроцесорних систем, що розрізняються потужністю, універсальністю, швидкістю та структурними відмінностями. Основні типи наступні:

- мікроконтролери – найбільш простий тип мікропроцесорних систем, у яких всі або більшість вузлів системи виконані у вигляді однієї мікросхеми;
- контролери – керуючі мікропроцесорні системи, виконані у вигляді окремих модулів;
- мікрокомп'ютери – потужні мікропроцесорні системи з розвиненими засобами сполучення із зовнішніми пристроями;
- комп'ютери (у тому числі персональні) – самі потужні та найбільш універсальні мікропроцесорні системи.

Чітку границю між цими типами провести досить складно. Швидкодія всіх типів мікропроцесорів постійно росте, і нерідкі ситуації, коли новий мікроконтролер виявляється швидшим за персональний комп'ютер. Але деякі принципові відмінності все-таки є.

Мікроконтролери представляють собою універсальні пристрої, які практично завжди використовуються в більш складних пристроях, у тому числі і контролерах. Системна шина мікроконтролера схована від користувача всередині мікросхеми. Можливості підключення зовнішніх пристроїв до мікроконтролера обмежені. Пристрої на мікроконтролерах зазвичай призначені для вирішення одного завдання.

Контролери, як правило, створюються для вирішення якогось окремого завдання або групи близьких завдань. Вони не мають можливості підключення додаткових вузлів і пристроїв, наприклад, великої пам'яті, засобів вводу/виводу. Їх системна шина найчастіше недоступна користувачу. Структура контролера проста і оптимізована під максимальну швидкодію. У більшості випадків програми, що виконуються, зберігаються в постійній пам'яті і не змінюються. Конструктивно контролери випускаються в одноплатному варіанті.

Мікрокомп'ютери відрізняються від контролерів більш відкритою структурою, вони допускають підключення до системної шини декількох додаткових пристроїв. Виробляються мікрокомп'ютери в каркасі, корпусі з роз'ємами системної магістралі, доступними користувачеві. Мікрокомп'ютери можуть мати засоби зберігання інформації на різних носіях і досить розвинені засоби зв'язку з користувачем (монітор, клавіатура). Мікрокомп'ютери розраховані на широке коло завдань, але на відміну від контролерів, до кожної нової задачі його треба пристосовувати заново.

Комп'ютери і найпоширеніші з них – персональні комп'ютери – це самі універсальні з мікропроцесорних систем. Вони обов'язково передбачають можливість модернізації, а також широкі можливості підключення нових пристроїв. Їх системна шина, зазвичай, доступна користувачу. Крім того, зовнішні пристрої можуть підключатися до комп'ютера через кілька вбудованих портів зв'язку. Комп'ютер завжди має розвинені засоби зв'язку з користувачем, засоби тривалого зберігання інформації великого обсягу, засоби зв'язку з іншими комп'ютерами по інформаційним мережам. Області застосування комп'ютерів можуть бути самими різними: математичні розрахунки, обслуговування доступу до баз даних, керування роботою складних електронних систем, комп'ютерні ігри, підготовка документів і т.д.

Будь-яке завдання можна виконати за допомогою кожного з перерахованих типів мікропроцесорних систем. Але при виборі типу треба по можливості уникати надмірності і передбачати необхідну для даного завдання гнучкість системи.

В існуючий час при розробці нових мікропроцесорних систем найчастіше вибирають шлях використання мікроконтролерів (приблизно у 80% випадків). При цьому мікроконтролери застосовуються або самостійно, з мінімальними додатковими вузлами, або в складі більш складних контролерів з розвиненими засобами вводу/виводу.

1.7.3. Функції процесора

Процесор (рис. 1.22) зазвичай представляє собою окрему мікросхему або ж частину мікросхеми (у випадку мікроконтролера). Мікросхема процесора обов'язково має виводи трьох шин: шини адресу, шини даних і

шини керування. Іноді деякі сигнали і шини мультиплексуються, щоб зменшити кількість виводів мікросхеми процесора.

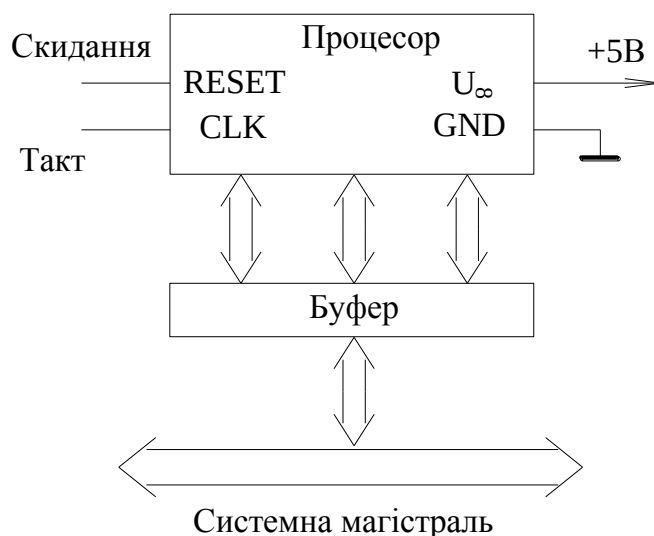


Рис. 1.22.

Найважливішими характеристиками процесора є кількість розрядів його шини даних, кількість розрядів його шини адресу і кількість керуючих сигналів у шині керування. Розрядність шини даних визначає швидкість роботи системи. Розрядність шини адресу визначає припустиму складність системи. Кількість ліній керування визначає різноманітність режимів обміну та ефективність обміну даними між процесором та іншими пристроями системи.

Крім виводів для сигналів трьох основних шин, процесор завжди має вивід (або два виводи) для підключення зовнішнього тактового сигналу або кварцового резонатора (CLK), тому що процесор завжди представляє собою тактований пристрій. Чим більше тактова частота процесора, тим він швидше працює, тобто тим швидше виконує команди. Однак швидкодія процесора визначається не тільки тактовою частотою, але і особливостями його структури. Сучасні процесори виконують більшість команд за один такт і мають засоби для паралельного виконання декількох команд. Тактова частота процесора не зв'язана прямо і жорстко зі швидкістю обміну по магістралі, тому що швидкість обміну по магістралі обмежена затримками поширення сигналів. Тобто тактова частота процесора визначає тільки його внутрішню швидкодію, а не зовнішню. Іноді тактова частота процесора має нижню і верхню межі. При перевищенні верхньої межі частоти можливий перегріву процесора, а також збої, причому, які виникають не завжди і нерегулярно.

Ще один важливий сигнал, що є в кожному процесорі – це сигнал початкового скидання RESET. При включенні живлення, при аварійній ситуації або зависанні процесора, подача цього сигналу призводить до ініціалізації процесора, змушує його виконати програму початкового запуску. Аварійна ситуація може бути викликана перешкодами по

ланцюгах живлення, збоями в роботі пам'яті та іншими причинами. В результаті процесор може втратити контроль над програмою, що виконується, і зупинитися в якійсь адресі. Для виходу із цього стану саме і використовується сигнал початкового скидання. Цей же вхід початкового скидання може використовуватися для оповіщення процесора про те, що напруга живлення нижче встановленої межі. В такому випадку процесор переходить до виконання програми збереження важливих даних. По суті, цей вхід представляє собою особливий різновид переривання.

Для підключення процесора до магістралі використовуються буферні мікросхеми, що забезпечують при необхідності демультіплексування сигналів та електричне буферування сигналів магістралі. Іноді протоколи обміну по системній магістралі і по шинах процесора не збігаються між собою, тоді буферні мікросхеми ще й узгоджують ці протоколи один з одним. Іноді в мікропроцесорній системі використовується кілька магістралей (системних і локальних), тоді для кожної з магістралей застосовується свій буферний вузол. Така структура характерна, наприклад, для персональних комп'ютерів.

Після включення живлення процесор переходить в першу адресу програми початкового пуску і виконує цю програму. Дана програма попередньо записана в постійну (енергонезалежну) пам'ять. Після завершення програми початкового пуску процесор починає виконувати основну програму, що перебуває в постійній або оперативній пам'яті, для чого вибирає по черзі всі команди. Від цієї програми процесор можуть відволікати зовнішні переривання або запити на ПДП. Команди з пам'яті процесор вибирає за допомогою циклів зчитування по магістралі. При необхідності процесор записує дані в пам'ять або в пристрої вводу/виводу за допомогою циклів запису або ж зчитує дані з пам'яті або із пристроїв вводу/виводу за допомогою циклів зчитування.

Таким чином, основні функції будь-якого процесора наступні:

- вибірка (зчитування) команд, що виконуються;
- ввід (зчитування) даних з пам'яті або пристроїв вводу/виводу;
- вивід (запис) даних в пам'ять або в пристрої вводу/виводу;
- обробка даних (операндів), в тому числі арифметичні операції над ними;
- адресація пам'яті, тобто завдання адреси пам'яті, з якою буде виконуватися обмін;
- обробка переривань і режиму прямого доступу.

Основні функції показаних на рис. 1.23 вузлів наступні. Схема управління вибіркою команд виконує зчитування команд із пам'яті та їх дешифрацію. У перших мікропроцесорах було неможливим одночасне виконання попередньої команди і вибірка наступної команди, тому що процесор не міг сполучати ці операції. Але вже в 16-розрядних процесорах з'являється так званий конвеєр (черга) команд, що дозволяє вибирати

кілька наступних команд, поки виконується попередня. Два процеси виконуються паралельно, що прискорює роботу процесора.

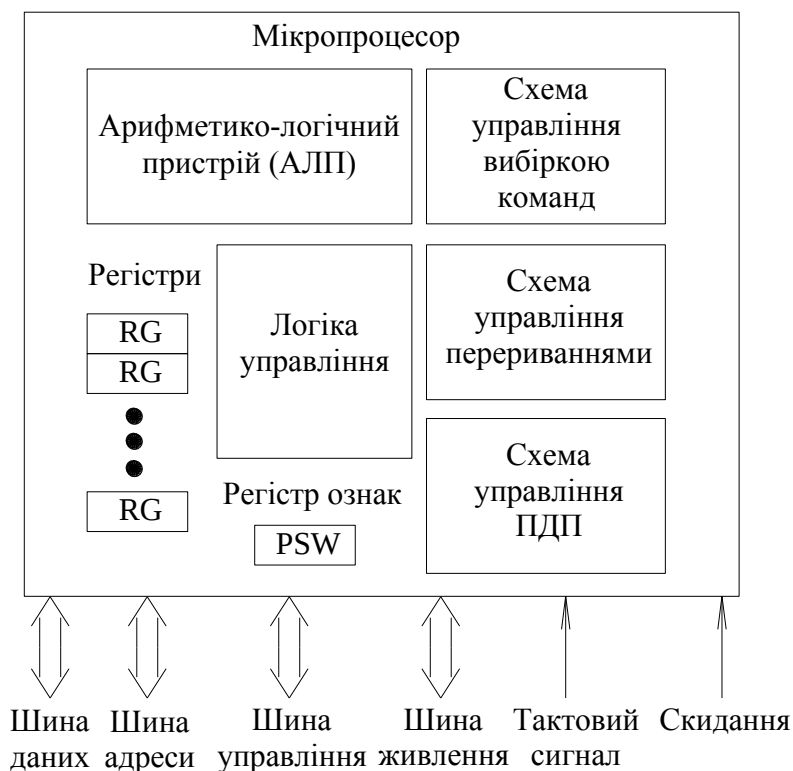


Рис. 1.23.

Конвеєр представляє собою невелику внутрішню пам'ять процесора, у яку при найменшій можливості (при звільненні зовнішньої шини) записується кілька команд. Зчитуються ці команди процесором у тому ж порядку, що і записуються в конвеєр (це пам'ять типу FIFO).

Розвитком ідеї конвеєра стало використання внутрішньої кеш-пам'яті процесора, що заповнюється командами, поки процесор зайнятий виконанням попередніх команд. Обробляти команди, що перебувають у внутрішній пам'яті, процесор може набагато швидше, ніж ті, які розташовані у зовнішній пам'яті. У кеш-пам'яті можуть зберігатися і дані, що також прискорює роботу.

Арифметико-логічний пристрій (або АЛП, ALU) призначений для обробки інформації відповідно до отриманої процесором команди. Прикладами обробки можуть служити логічні операції (типу логічного "І", "АБО", "Викл. АБО" і т.д.) тобто побітові операції над операндами, а також арифметичні операції (типу додавання, множення, ділення та ін.). Команда визначає, над якими даними буде виконуватися операція і куди буде розміщений результат. Якщо команда виконує пересилання даних без їх обробки, то АЛП не бере участь у її виконанні.

Швидкодія АЛП багато в чому визначає продуктивність процесора. Причому важлива не тільки частота тактового сигналу, але й кількість тактів, необхідна для виконання тієї або іншої команди.

Регістри процесора представляють собою по суті осередок дуже швидкої пам'яті і служать для тимчасового зберігання різних кодів: даних, адрес, службових кодів. Операції із цими кодами виконуються гранично швидко, тому, у загальному випадку, чим більше внутрішніх реєстрів, тим краще. Крім того, на швидкодію процесора сильно впливає розрядність реєстрів. Саме розрядність реєстрів та АЛП називається внутрішньою розрядністю процесора, що може не збігатися із зовнішньою розрядністю.

Регістр ознак (регістр стану) займає особливе місце, хоча він також є внутрішнім реєстром процесора. Інформація, що міститься в ньому – це не дані і не адреси, а слово стану процесора (ССП, PSW – Processor Status Word). Кожен біт цього слова (прапорець) містить інформацію про результат попередньої команди. Наприклад, є біт нульового результату, що встановлюється в тому випадку, коли результат виконання попередньої команди – нуль, і очищається в тому випадку, коли результат виконання команди відмінний від нуля. Ці біти (прапори) використовуються командами умовних переходів, наприклад, командою переходу у випадку нульового результату. У цьому ж реєстрі іноді утримуються прапори керування, що визначають режим виконання деяких команд.

Схема керування перериваннями обробляє поступаючий на процесор запит переривання, визначає адресу початку програми обробки переривання (адресу вектора переривання), забезпечує перехід до цієї програми після виконання поточної команди і збереження в пам'яті (у стеці) поточного стану реєстрів процесора. По закінченні програми обробки переривання процесор повертається до перерваної програми з відновленими з пам'яті (зі стека) значеннями внутрішніх реєстрів.

Схема керування прямим доступом до пам'яті служить для тимчасового відключення процесора від зовнішніх шин і припинення роботи процесора на час надання прямого доступу пристрою, що його запросив.

Логіка керування організує взаємодію всіх вузлів процесора, перенаправляє дані, синхронізує роботу процесора із зовнішніми сигналами, а також реалізує процедури вводу і виводу інформації.

Таким чином, в процесі роботи процесора схема вибірки команд вибирає послідовно команди з пам'яті, потім ці команди виконуються, причому, якщо буде потреба обробки даних, підключається АЛП. На входи АЛП можуть подаватися дані з пам'яті або із внутрішніх реєстрів. У внутрішніх реєстрах зберігаються також коди адрес оброблюваних даних, розташованих у пам'яті. Результат обробки в АЛП змінює стан реєстра ознак і записується у внутрішній.

Внутрішні реєстри будь-якого мікропроцесора обов'язково виконують дві службові функції:

- визначають адресу в пам'яті, де перебуває команда, що виконується в даний момент (функція лічильника команд або вказівника команд);

– визначають поточну адресу стека (функція вказівника стека).

У різних процесорах для кожної із цих функцій може приділятися один або два внутрішніх регістри. Ці два вказівники відрізняються від інших не тільки своїм специфічним, службовим, системним призначенням, але і особливим способом зміни вмісту.

Вміст вказівника (лічильника) команд змінюється в такий спосіб. На початку роботи системи (при включенні живлення) у нього заноситься раз і назавжди встановлене значення. Це перша адреса програми початкового запуску. Потім після вибірки з пам'яті кожної наступної команди значення покажчика команд автоматично збільшується (інкрементується) на одиницю (або на два залежно від формату команд і типу процесора). Тобто наступна команда буде вибиратися з наступної адреси пам'яті. При виконанні команд переходу, що порушують послідовний перебір адрес пам'яті, у вказівник команд примусово записується нове значення – нова адреса в пам'яті, починаючи з якої адреси команд знову ж будуть перебиратися послідовно. Така ж зміна вмісту вказівника команд виконується при виклику підпрограми і поверненні з неї, або при початку обробки переривання і після його закінчення.

1.7.4. Функції пам'яті

Пам'ять мікропроцесорної системи виконує функцію тимчасового або постійного зберігання даних і команд. Обсяг пам'яті визначає припустиму складність виконуваних системою алгоритмів, а також швидкість роботи системи в цілому. Модулі пам'яті виконуються на мікросхемах пам'яті (оперативної або постійної). Все частіше в складі мікропроцесорних систем використовується флеш–пам'ять (англ. – flash memory), що представляє собою енергонезалежну пам'ять із можливістю багаторазового перезапису вмісту.

Інформація в пам'яті зберігається в комірках, кількість розрядів яких дорівнює кількості розрядів шини даних процесора. Сукупність комірок пам'яті називається звичайно простором пам'яті системи.

Для підключення модуля пам'яті до системної магістралі використовуються блоки сполучення, які містять у собі дешифратор (селектор) адреси, схему обробки керуючих сигналів магістралі та буфери даних (рис. 1.24).

Оперативна пам'ять спілкується із системною магістраллю в циклах читання і запису, постійна пам'ять – тільки в циклах читання. Зазвичай в складі системи є кілька модулів пам'яті, кожний з яких працює у своїй області простору пам'яті. Селектор адреси саме і визначає, яка область адресів простору пам'яті відведена даному модулю пам'яті. Схема керування виробляє в потрібні моменти сигнали дозволу роботи пам'яті (CS) і сигнали дозволу запису (WR). Буфери даних передають дані від пам'яті до магістралі або від магістралі до пам'яті.

У просторі пам'яті мікропроцесорної системи виділяють кілька особливих областей, які виконують спеціальні функції.

Пам'ять програми початкового запуску завжди виконується на ПЗП або флеш-пам'яті. Саме із цієї області процесор починає роботу після вмикання живлення і після скидання його за допомогою сигналу RESET.

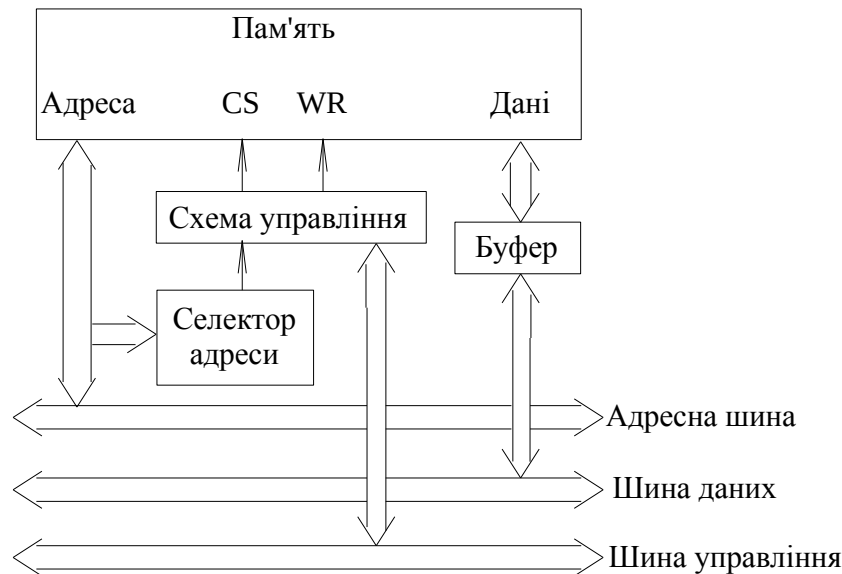


Рис. 1.24.

Пам'ять для стека або стек (Stack) – це частина оперативної пам'яті, призначена для тимчасового зберігання даних у режимі LIFO (Last In – First Out).

Особливість стека в порівнянні з іншою оперативною пам'яттю – це заданий і незмінний спосіб адресації. При записі будь-якого числа (коду) у стек число записується за адресою, що розташована у регістрі вказівника стеку, попередньо зменшене (декрементоване) на одиницю (або на два, якщо 16-розрядні слова розташовані в пам'яті по парних адресах). При зчитуванні зі стека, число зчитується з адреси, що міститься в регістрі вказівника стеку, після чого значення вказівника стека збільшується (інкрементується) на одиницю (або на два). В результаті виходить, що число, записане останнім, буде прочитано першим, а число, записане першим, буде прочитано останнім. Принцип дії стека показаний на рис. 1.25 (адреси комірок пам'яті обрані умовно).

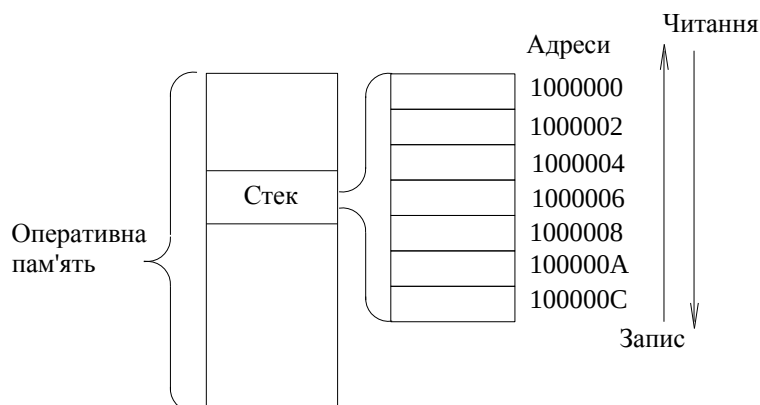


Рис. 1.25.

Нехай, наприклад, що поточний стан покажчика стека 1000008, і в нього треба записати два числа (слова). Перше слово буде записано за адресою 1000006 (перед записом покажчик стека зменшиться на два). Друге – за адресою 1000004. Після запису вміст вказівник стека – 1000004. Якщо потім прочитати зі стека два слова, то першим буде прочитане слово з адреси 1000004, а після зчитування вказівник стека стане рівним 1000006. Другим буде прочитане слово з адреси 1000006, а вказівник стека стане рівним 1000008. Все повернулося до вихідного стану. Перше записане слово зчитується другим, а друге – першим.

Необхідність такої адресації стає очевидною у випадку багаторазово вкладених підпрограм. Нехай, наприклад, виконується основна програма, і з неї викликається підпрограма 1. Якщо нам треба зберегти значення даних і внутрішніх реєстрів основної програми на час виконання підпрограми, ми перед викликом підпрограми збережемо їх у стеці (запишемо в стек), а після її закінчення витягнемо (прочитаємо) їх зі стека. Якщо ж з підпрограми 1 викликається підпрограма 2, то ту ж саму операцію ми проробимо з даними і вмістом внутрішніх реєстрів підпрограми 1. Зрозуміло, що всередині підпрограми 2 крайніми в стеці будуть дані з підпрограми 1, а дані з основної програми будуть глибше. При цьому у випадку зчитування зі стека автоматично буде дотримуватися потрібний порядок зчитування інформації. Те ж саме буде і у випадку, коли таких рівнів вкладення підпрограм набагато більше.

У системі команд будь-якого процесора для обміну інформацією зі стеком передбачені спеціальні команди запису в стек (PUSH) і зчитування зі стека (POP). У стеці можна зберігати не тільки вміст всіх внутрішніх реєстрів процесора, але і вміст реєстра ознак (слово стану процесора, PSW). Це дозволяє, наприклад, при поверненні з підпрограми контролювати результат останньої команди, виконаної безпосередньо перед викликом цієї підпрограми. Можна також зберігати в стеці і дані, для того щоб зручніше було передавати їх між програмами та підпрограмами.

Наступна спеціальна область пам'яті – це таблиця векторів переривань. Взагалі, поняття переривання досить багатозначне. Під перериванням у загальному випадку розуміється не тільки обслуговування запиту зовнішнього пристрою, але і будь-яке порушення послідовної роботи процесора. Наприклад, може бути передбачене переривання по факту некоректного виконання арифметичної операції типу ділення на нуль. Або ж переривання може бути програмним, коли в програмі використовується команда переходу на якусь підпрограму, з якої потім піде повернення в основну програму.

Будь-яке переривання обробляється через таблицю векторів (вказівників) переривань. У цій таблиці в найпростішому випадку перебувають адреси початку програм обробки переривань, які й називаються векторами. Довжина таблиці може бути досить великою (до кількох сотень елементів). Зазвичай таблиця векторів переривань

розташовується на початку простору пам'яті (у комірках пам'яті з малими адресами). Адреса кожного вектора (або адреса початкового елемента кожного вектора) представляє собою номер переривання.

У випадку апаратних переривань номер переривання задається пристроєм, що запросив переривання (при векторних перериваннях), або ж задається номером лінії запиту переривань (при радіальних перериваннях). Процесор, одержавши апаратне переривання, закінчує виконання поточної команди і звертається до пам'яті в область таблиці векторів переривань, у той її рядок, що визначається номером запитаного переривання. Потім процесор зчитує вміст цього рядка (код вектора переривання) і переходить на адресу пам'яті, що задається цим вектором. Починаючи із цієї адреси в пам'яті повинна розташовуватися програма обробки переривання з даним номером. Наприкінці програми обробки переривання обов'язково повинна розташовуватися команда виходу з переривання, виконавши яку процесор повертається до виконання перерваної основної програми. Параметри процесора на час виконання програми обробки переривання зберігаються в стеці. На рис. 1.26 представлений алгоритм обробки переривання.

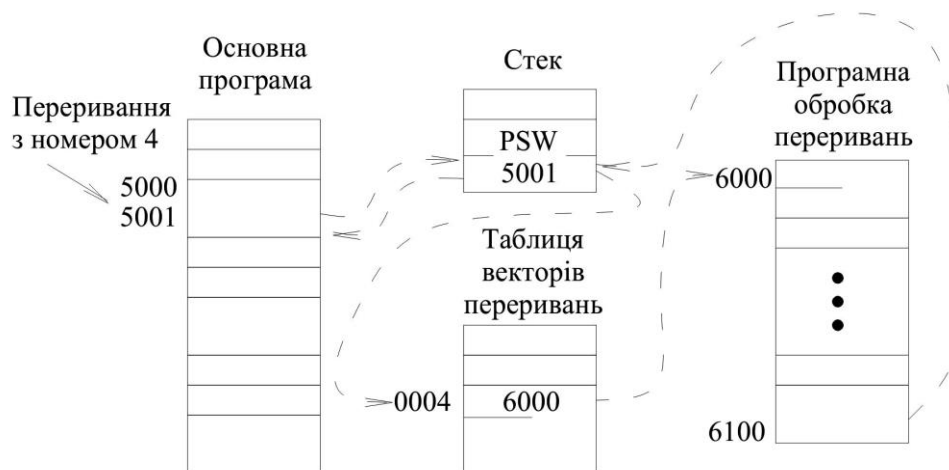


Рис. 1.26.

Нехай, наприклад, процесор виконував основну програму і команду, що перебуває в адресі пам'яті 5000 (умовно). У цей момент він одержав запит переривання з номером (адресою вектора) 4. Процесор закінчує виконання команди з адреси 5000. Потім він зберігає в стеці поточне значення лічильника команд (5001) і поточне значення PSW. Після цього процесор зчитує з адреси 4 пам'яті код вектора переривання. Нехай цей код дорівнює 6000. Процесор переходить на адресу пам'яті 6000 і приступає до виконання програми обробки переривання, що починається із цієї адреси. Нехай ця програма закінчується в адресі 6100. Дійшовши до цієї адреси, процесор повертається до виконання перерваної програми. Для цього він витягає зі стека значення адреси (5001), на якому його перервали, і значення PSW. Потім процесор зчитує команду з адреси 5001 і далі послідовно виконує команди основної програми.

Ще одна спеціальна область пам'яті мікропроцесорної системи – це пам'ять пристроїв, підключених до системної шини. Таке рішення зустрічається нечасто, але іноді воно дуже зручно. Тобто процесор одержує можливість звертатися до внутрішньої пам'яті пристроїв вводу/виводу або якихось ще підключених до системної шини пристроїв, як до своєї власної системної пам'яті. Звичайне вікно в просторі пам'яті, що виділяється для цього, не занадто велике.

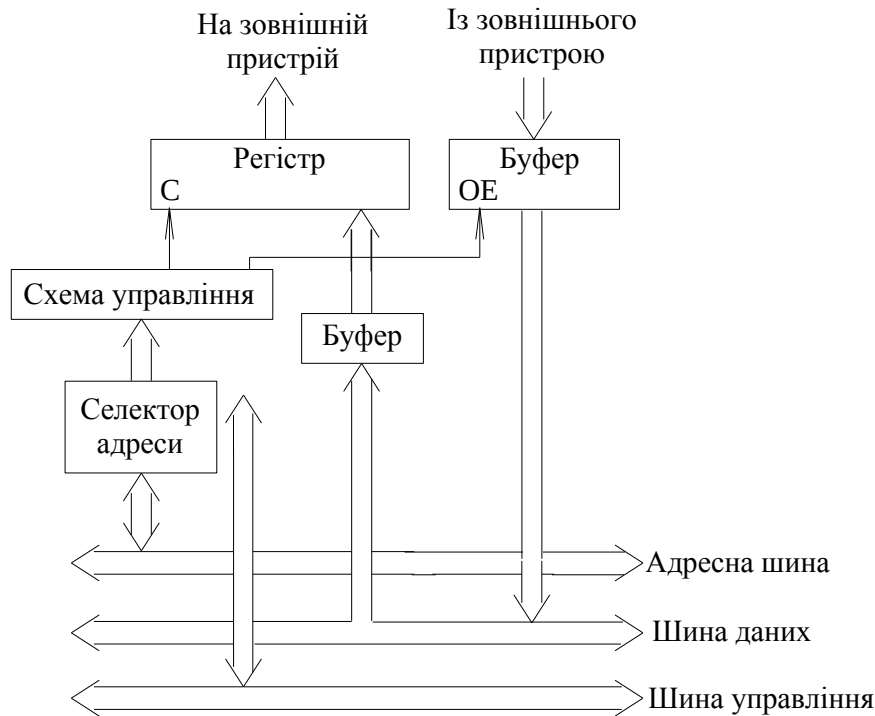
Всі інші частини простору пам'яті, як правило, мають універсальне призначення. У них можуть розташовуватися як дані, так і програми. Іноді цей простір пам'яті використовується як єдине ціле. А іноді простір пам'яті ділиться на сегменти із програмно змінюваною адресою початку сегмента і з встановленим розміром сегмента. Обидві підходи мають свої плюси та мінуси. Наприклад, використання сегментів дозволяє захистити область програм або даних, але границі сегментів можуть ускладнювати розміщення великих програм і масивів даних.

1.7.5. Функції пристроїв вводу/виводу

Пристрої вводу/виводу обмінюються інформацією з магістраллю по таким самим принципам, що і пам'ять. Найбільш істотна відмінність з погляду організації обміну полягає в тому, що модуль пам'яті має в адресному просторі системи багато адресів (до декількох десятків мільйонів), а пристрій вводу/виводу зазвичай має небагато адресів, а іноді всього одну адресу.

Поєднує всі пристрої вводу/виводу загальні принципи обміну з магістраллю і, відповідно, загальні принципи організації вузлів, які здійснюють сполучення з магістраллю. Спрощена структура пристрою вводу/виводу (його інтерфейсна частина) представлена на рис. 1.27. Вона містить схему селектора адреси, схему керування для обробки стробів обміну та буфери даних.

Самі найпростіші пристрої вводу/виводу видають на зовнішній пристрій код даних у паралельному форматі і приймають із зовнішнього пристрою код даних у паралельному форматі. Такі пристрої вводу/виводу часто називають паралельними портами вводу/виводу. Вони найбільш універсальні, тобто задовольняють потреби з'єднання з великим числом зовнішніх пристроїв, тому їх часто вводять до складу мікропроцесорної системи як стандартні пристрої. Паралельні порти звичайно є в складі мікроконтролерів. Саме через паралельні порти мікроконтролер зв'язується із зовнішніми пристроями (з датчиками та виконавчими пристроями).



Puc. 1.27.

Вхідний порт (порт вводу) в найпростішому випадку представляє собою паралельний регістр, у який процесор може записувати інформацію. Вихідний порт (порт виводу) представляє собою просто однонаправлений буфер, через який процесор може зчитувати інформацію від зовнішнього пристрою. Порт може бути і двонаправленим (вхідним/вихідним). У цьому випадку процесор пише інформацію в зовнішній пристрій і зчитує інформацію із зовнішнього пристрою по тому самому адресу в адресному просторі системи. Вхідні та вихідні лінії для зв'язку із зовнішнім пристроєм при цьому можуть бути об'єднані порозрядно, утворюючи двонаправлені лінії.

При звертанні з боку магістралі селектор адреси розпізнає адресу, що відповідає даному пристрою вводу/виводу. Схема керування видає внутрішні строби обміну у відповідь на магістральні строби обміну. Вхідний буфер даних забезпечує електричне узгодження шини даних із цим пристроєм. Дані із шини даних записуються в регістр по сигналу С і видаються на зовнішній пристрій. Вихідний буфер даних передає вхідні дані із зовнішнього пристрою на шину даних магістралі в циклі зчитування з порту.

Більш складні пристрої вводу/виводу мають у своєму складі внутрішню буферну оперативну пам'ять і навіть можуть мати мікроконтролер, на який покладене виконання функцій обміну із зовнішнім пристроєм.

Кожному пристрою вводу/виводу призначається своя адреса в адресному просторі мікропроцесорної системи. Дублювання адрес повинне бути виключене.

У складі мікропроцесорних систем, як правило, виділяються три спеціальні групи пристроїв вводу/виводу:

- пристрої інтерфейсу користувача (введення інформації користувачем і виводу інформації для користувача);
- пристрої вводу/виводу для тривалого зберігання інформації;
- таймерні пристрої.

До пристроїв вводу для інтерфейсу користувача ставляться контролери клавіатури, тумблерів, окремих кнопок, миші, джойстика і т.д. До пристроїв виводу для інтерфейсу користувача ставляться контролери світлодіодних індикаторів, табло, рідкокристалічних, плазмових екранів і т.д. У найпростіших випадках – керуючих контролерів або мікроконтролерів ці засоби можуть бути відсутніми. У складних мікропроцесорних системах вони є обов'язково.

Таймерні пристрої відрізняються від інших пристроїв вводу/виводу тим, що вони можуть не мати виводів для підключення до зовнішніх пристроїв. Ці пристрої призначені для того, щоб мікропроцесорна система могла витримувати задані часові інтервали, стежити за реальним часом, рахувати імпульси і т.д. В основі будь-якого таймера лежить кварцовий тактовий генератор і багато розрядні двійкові лічильники, які можуть перезапускати один одного. Процесор може записувати в таймер коефіцієнти розподілу тактової частоти, кількість відлічуваних імпульсів, задавати режим роботи лічильників таймера, а зчитує процесор вихідні коди лічильників. Виконати практично всі функції таймера можна і програмним шляхом, тому іноді таймери в системі відсутні. Але включення в систему таймера дозволяє вирішувати більш складні завдання і будувати більше ефективні алгоритми.

1.8. Організація однокристальних мікроконтролерів

Сучасний ринок мікроконтролерів насичений виробами великої кількості фірм. Кожен з типів мікроконтролерів має свої особливості, які виділяють його серед інших і забезпечують йому конкретні галузі використання. Але всі ці прилади можна розділити на три основні групи:

- 8-ми розрядні мікро контролери;
- 16 і 32-х розрядні;
- цифрові процесори для обробки сигналів.

Особливість всіх цих архітектур на сучасному рівні розвитку мікропроцесорної техніки полягає в тому, що всі вони виготовляються на одному кристалі і об'єднують в собі всі елементи мікропроцесорної системи, тобто, центральний процесор, ПЗП, ОЗП, порти вводу / виводу, таймери та інше.

Найбільш широке використання знаходять 8-ми розрядні прилади, які широко використовуються в промисловості, побутовій техніці, для керування реальними об'єктами, де використовуються в основному алгоритми з переважною кількістю логічних операцій та таймери. Росту популярності 8-ми розрядних мікро контролерів, що виготовляються

такими відомими фірмами, як Motorola, Microchip, Intel, Zilog, Atmel сприяє той факт, що номенклатура їх постійно зростає, а функціональні можливості розширюються. Відмінними рисами сучасних мікроконтролерів є:

- модульна організація, при якій на основі одного процесорного ядра проектується ряд мікроконтролерів з різними об'ємами пам'яті програм і пам'яті даних, набором периферійних модулів, тактовою частотою;
- використання закритої архітектури, що характеризується відсутністю ліній системної магістралі на виводах корпусів;
- використання типових функціональних периферійних модулів (таймери, процесори подій, контролери послідовних інтерфейсів, аналого – цифрові перетворювачі та інше);
- розширення числа режимів роботи периферійних модулів, що задаються в процесі ініціалізації регістрів спеціальних функцій мікро контролера.

На рис. 1.28. приведена типова структура мікроконтролера, що вміщує в себе як вже відомі модулі, так і допоміжні, якими можуть бути аналого–цифрові перетворювачі, широтно–імпульсні модулятори, компаратори напруги та ін.

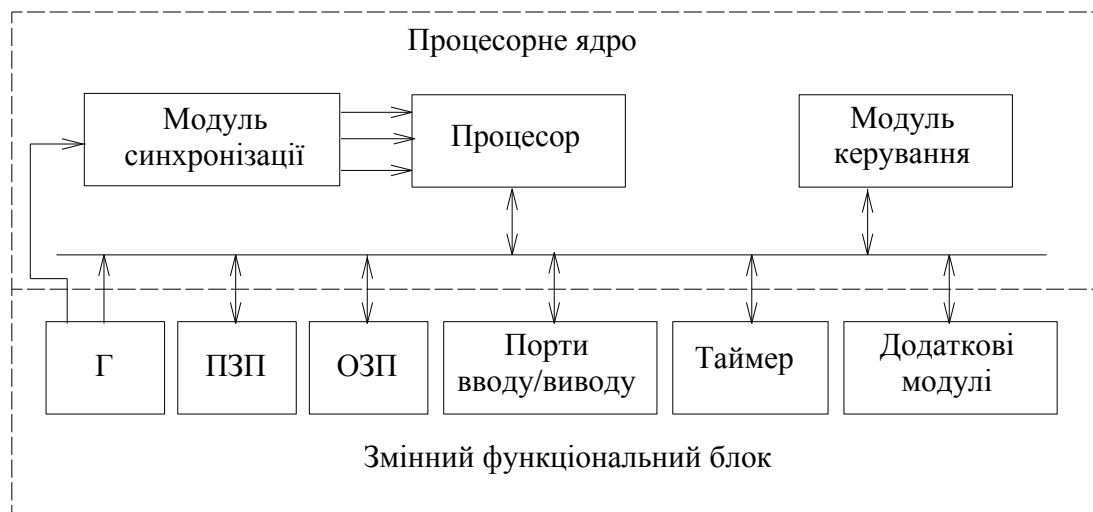


Рис. 1.28.

Процесорне ядро сучасних мікроконтролерів будується як на основі фон–Неймановської архітектури, так і на основі гарвардської. Але оскільки мікроконтролери – це спеціалізовані пристрої, то універсальність фон–Неймановської архітектури здебільшого не має суттєвого значення. До того ж програми, що виконуються на МК вимагають незначних обсягів пам'яті даних, значно менших, ніж обсяги пам'яті програм. Тому використання єдиного адресного простору приводить до збільшення формату команд за рахунок збільшення розрядів для адресації операндів. В таких випадках переваги гарвардської архітектури з невеликим обсягом пам'яті даних і зменшеною довжиною команд очевидні. До того ж

гарвардська архітектура може забезпечити більш високу швидкість виконання програм за рахунок можливості реалізації паралельних операцій. Вибір чергової команди може виконуватись одночасно з виконанням попередньої, що може забезпечити виконання різних команд за однакову кількість тактів. Це дає можливість більш просто визначити час виконання циклів. Тому більшість сучасних 8–ми розрядних МК використовують гарвардську архітектуру.

1.9. Система команд процесора

Система команд процесора включає в себе три основні групи:

- команди пересилок;
- арифметичні та логічні команди;
- команди переходів.

Команди пересилок займають дуже важливе місце в системі команд будь–якого процесора, оскільки виконують наступні функції:

- завантаження внутрішніх регістрів процесора (пересилка безпосереднього операнда в регістр);
- копіювання даних з одної пам'яті процесора в іншу;
- запис в пристрої вводу/виводу та зчитування з них.

В різних типах процесорів використовується різна кількість команд пересилок. Мінімальна кількість – це одна команда MOV, а функції, які вона виконує, задаються лише адресами портів, або регістрів. В інших процесорах використовується досить велика кількість типів команд пересилок. Вони можуть характеризуватись як об'ємом інформації, що пересилається (від одного байта до цілих пакетів, наприклад, команди MOVS процесора 8086), так і конкретним призначенням слів, що пересилаються, наприклад, команди IN, OUT – для обміну з портами; команди PUSH, POP – для роботи зі стеком. До команд пересилки відносяться команди обміну інформацією, які передбачають процедури обміну між двома регістрами, між регістром і зовнішньою пам'яттю, між двома половинами одного регістра.

Арифметичні команди в мікроконтролерах включають наступні групи:

- команди операцій з фіксованою комою (додавання (ADD), віднімання (SUB), множення (MUL), ділення (DIV)). При цьому деякі команди, особливо множення та ділення, в малопотужних МК відсутні, а ті що використовуються, можуть бути розширеними, наприклад, командами з фіксацією переносу (ADC).
- Ті ж команди, але для операцій з плаваючою комою. Вони мають місце лише в спеціалізованих МК, або в потужних мікропроцесорах. В останніх кількість команд може бути суттєво розширена за рахунок операцій обчислення тригонометричних, логарифмічних функцій, спеціалізованих функцій обробки сигналів.

- Команди інкремента (INC – збільшення вмісту регістра на 1) та декремента (DEC – зменшення вмісту регістра на 1). Їх різниця між операціями додавання (віднімання) одиниці полягає в тому, що вони обчислюються значно швидше.
- Команда порівняння двох вхідних операндів (CMP) виділяється окремо від арифметичних команд тому, що вона не формує вихідного операнда, як подібна команда віднімання, а лише змінює стан відповідних бітів в регістрі стану процесора (PSW). В програмах за командою CMP слідує, як правило, команда переходу, яка контролює стан вказаних бітів в регістрі стану і виконує дії в залежності від їх значень.

Логічні команди виконують над операндами побітові операції, тобто операнд розглядається не як єдине слово, а як набір окремих бітів. В цьому полягає їх суттєва відмінність від арифметичних операцій. В мікроконтролерах логічні команди виконують наступні операції:

- Логічні операції "І" (AND) "АБО" (OR), "ВИКЛ.АБО" (XOR) призначені для побітового обчислення логічних функцій від двох операндів. Вказані логічні операції також широко використовуються для інших побітових операцій. Наприклад, операція AND використовується для примусового встановлення відповідного розряду операнда в нуль. Для цього виконується операція "І" операнда, що змінюється з операндом – маскою, в якому необхідний розряд встановлений в нуль. Аналогічно операція "АБО" використовується для встановлення відповідного розряду операнда в одиницю, а операція "ВИКЛ.АБО" використовується для інверсії заданих бітів (при цьому відповідні біти в коді маски встановлюються в одиницю).
- Команди очистки регістра, або елемента пам'яті (CLR). В деяких МК вони відсутні і замінюються командами пересилки нульового коду. Антиподом команди очистки використовується команда установки (SEC). Для керування портами мікро контролерів використовують команди установки – очищення регістрів портів. Для цього використовують команди SBI (установки) та CBI (очищення).
- Команди зсуву (LSL – логічний зсув вліво), (LSR – логічний зсув вправо), (ASR – арифметичний зсув в право) та інші що забезпечують побітовий зсув розрядів операнда вправо (в сторону молодших розрядів) або вліво (в сторону старших розрядів). При цьому використовується декілька типів зсуву: логічний, арифметичний, циклічний, які визначають нове значення старшого або молодшого біту в операнді після виконання операції. Команди циклічного зсуву дозволяють переміщувати біти операнда по кільцю, як вправо так і вліво. При

цьому в кільце може включатись біт ознаки переносу PSW. Команди зсуву вихідного операнда не формують.

- Команди переходів (розгалуження) призначені для створення різноманітних циклів, розгалужень в програмі, виклику підпрограм і т.п. Від уміння використовувати команди переходів залежить компактність програм. Вони розділяються на команди безумовних переходів, та команди умовних переходів. Їх назви походять від двох слів –branch (розгалуження) та jump (скачок), наприклад, RJMP– (відносний перехід), BRGE (перехід по умові). Команди безумовних переходів забезпечують обов'язковий перехід на нову адресу. Нова адреса при цьому може вказуватись різними способами, наприклад, фіксованим зміщенням, міткою, або вхідним операндом. Команди умовних переходів виконуються по результатах операції попередньої команди, що змінює біт ознаки в PSW. Попередньою командою, що використовується для перевірки умови є команда порівняння CMP, Кількість таких умов, і, відповідно, кількість команд по умові, залежить від мікропроцесора. Приведемо деякі приклади переходів по умовах, що використовуються в мікроконтролерах:

- якщо біт в регістрі скинуто;
- якщо біт в регістрі встановлений;
- перехід на мітку, якщо встановлений біт ознаки в PSW;
- перехід на мітку якщо не рівно;
- перехід на мітку, якщо мінус;
- перехід на мітку, якщо заборонено переривання.

Таких команд переходів по умовах в сучасних МК використовується до декількох десятків. Вони використовуються як для перевірки умов виконання операцій, так і для перевірки наявності чи відсутності зовнішніх сигналів, наприклад, запитів на переривання, сигналів таймерів і т.п. Сумісне використання команд умовних та безумовних переходів дозволяє компактно будувати і виконувати розгалужені алгоритми будь – якої складності.

Поряд з командами переходів використовуються команди з поверненням в точку, з якої був виконаний перехід (CALL – визвати підпрограму, RET – вихід з підпрограми, RETI – вихід з обслуговування переривання). Це ще один спосіб побудови компактних програм. Команда CALL має вхідний операнд, який вказує або абсолютне значення адреси підпрограми, або забезпечує інші шляхи отримання адреси. Перед переходом на виконання підпрограми чергове значення лічильника команд зберігається в стеку. Команди повернення з підпрограми, які записуються в кінці підпрограми, виймають зі стеку значення адреси чергової команди і записують її в лічильник команд. Використання підпрограм дозволяє суттєво спростувати структуру основної програми, робити її більш логічною, гнучкою, більш легкою для від лагодження. Але використання підпрограм дещо збільшує час виконання програми.

Команди обслуговування переривань (прийнята назва INT) також можна віднести до команд переходів з поверненням. Ці команди в якості вхідного операнда використовують адресу вектора переривання. Для переходу на виконання підпрограми обслуговування переривання процесор звертається до таблиці векторів переривань і отримує з неї по номеру переривання адресу пам'яті, на яку йому необхідно перейти. Ця адреса завантажується в лічильник адресів команд, а адреса чергової команди і вміст регістру PSW завантажуються в стек. В зберіганні вмісту PSW є суттєва різниця між командами переривання і командами переходів з поверненням. По закінченню обслуговування підпрограми переривання, що виконується командами IRET, або RTI, останні виймають зі стеку як адресу чергової команди, так і записаний вміст PSW. Адреса переписується в лічильник адресів команд, а вміст – в регістр стану процесора.

Незважаючи на дещо більшу складність виконання команди переходу по перериваннях, їх використання в ряді випадків є більш доцільним, ніж використання команд переходу з поверненням. Таблиця векторів переривань може бути сформованою попередньо і до неї легко звертатись по необхідності. Номер переривання буде відповідати номеру підпрограми, до якої йде звернення.

В окремих МК виділяється група команд для керування ресурсами контролера, яка використовується для налагодження режимів роботи портів вводу/виводу, керування таймером, і т.п. Внутрішні ресурси МК відображаються на пам'ять даних, тому для керування ресурсами використовуються команди пересилки даних.

В табл. 1.1 приводиться перелік команд, їх кодування і опис. Поле COP означає код операції, в полях Rd, Rs представлені адреси регістрів загального призначення, поле Func кодує тип арифметичної операції або умову переходу. При виконанні команд переходу аналізуються біти регістра статусу і якщо умова переходу виконується, відбувається зміна значення лічильника команд, тобто перехід на задану комірку пам'яті програм. Для перевірки умови використовуються команди порівняння вмісту двох регістрів (CP) та порівняння регістра з константою (CPI). В результаті виконання зазначених операцій змінюються біти в регістрі статусу SREG. Команди пересилки даних виконують дії над вмістом регістрів (завантаження константи, копіювання вмісту одного регістра в інший). Арифметичні і побітові операції виконуються арифметико-логічним пристроєм.

Табл. 1.1.

COP [15..12]	[11..8]	[7..4]	[3..0]	Команда
0000	0000	0000	0000	NOP (пуста операція)
0001	Адреса порту		D	OUT portX, Rd (вивід значення в порт)
0010	Rd	Адреса порту		IN Rd, pinX (зчитування з порту)
0011	1000	Адреса комірки пам'яті програм (ПП)		BRGE Addr (перехід якщо більше)
	0100	Адреса комірки ПП		BRLO Addr (перехід якщо менше)
	0010	Адреса комірки ПП		BREQ Addr (перехід якщо дорівнює)
	0001	Адреса комірки ПП		BRNE Addr (перехід якщо не дорівнює)
	1010	Адреса комірки ПП		BRSN Addr (перехід якщо більше або дорівнює)
	0110	Адреса комірки ПП		BRLT Addr (перехід якщо менше або дорівнює)
0100	0000	Адреса комірки ПП		JMP Addr (абс. перехід)
	0001	Зсув		RJMP Addr (відн. перехід)
	0010	—		RETI (вихід з переривання)
	0011	Адреса комірки ПП		RCALL (виклик підпрограми)
	0100	—		RET (вихід з підпрограми)
0110	Rd	Rs	0000	MOV Rd, Rs (копіювання вмісту регістра)
0110	Rd	Rs	0001	CP Rd, Rs (порівняння регістрів)

1110	Rd	константа		CPI Rd, K (порівняння регістру і константи)
1010	Rd	константа		LDI Rd, K (завантаження константи в регістр)
0111	Rd	Rs	0000	AND Rd, Rs (логічне І)
	Rd	Rs	0001	XOR Rd, Rs (логічне ВИКЛ.АБО)
	Rd	Rs	0010	OR Rd, Rs (логічне АБО)
	Rd	—	0011	NOT Rd (інверсія)
	Rd	—	0100	LSR Rd (лог.зсув праворуч)
	Rd	—	0101	ASR Rd (арифм.зсув праворуч)
	Rd	Rs	0110	ADD Rd, Rs (додавання значень регістрів)
	Rd	Rs	0111	SUB Rd, Rs (операція віднімання)
	Rd	Rs	1000	ADC Rd, Rs (додавання з переносом)
	Rd	Rs	1001	SBC Rd, Rs (віднімання з переносом)
	Rd	—	1010	INC Rd (інкремент)
	Rd	—	1011	DEC Rd (декремент)
	Rd	—	1100	LSL Rd (лог.зсув ліворуч)
	Rd	—	1101	ASL Rd (арифм.зсув ліворуч)
	Rd	Rs	1110	MUL Rd, Rs (перемноження)
	Rd	—	1111	SWAP Rd (обмін тетрадами)

В залежності від типу команди, в окремих розрядах міститься різна інформація. Так, для арифметичних операцій в розрядах [11:8] та [7:4]

команди розташовані адреси регістрів, над вмістом яких виконується операція. При виконанні арифметичних операцій результат розташовується в регістрі Rd. Якщо виконується операція множення, молодший та старший байти добутку розміщуються в регістрах R0 та R1. При виконанні команд з одним операндом значення бітів [7:4] не враховуються. Для команд роботи з константами (LDI та CPI) в бітах [11:8] міститься адрес регістра, а в молодших 8 бітах – значення константи. У командах переходу біти [11:8] визначають умову переходу, а в розрядах [7:0] міститься адреса комірки пам'яті, до якої буде виконаний перехід. Перехід відбувається якщо одна з умов переходу, яка передається разом з командою, співпадає зі значенням окремого біту (або декількох бітів) в регістрі–статусі. Перевірка виконання умови переходу відбувається шляхом виконання операції лог. „І” полів [11:8] команди і значення в регістрі – статусі. Якщо результат не дорівнює 0, то значення лічильника команд приймає нове значення.

1.10. Структура мікроконтролерів сімейства AVR

Мікроконтролери цього сімейства фірми Atmel – це 8–розрядні RISC – однокристальні ЕОМ широкого призначення. Для розробників електронних систем вони є привабливими завдяки найкращому співвідношенню показників “швидкодія/енергоспоживання”, широкою номенклатурою мікросхем, доступністю і простотою апаратно–програмних засобів підтримки.

На теперішній час фірма Atmel виготовляє різноманітні типи мікроконтролерів, які розділяються нею на чотири групи:

- Mega AVR (ATMEGA...);
- Classic AVR (AT90S...);
- Tiny AVR (ATiny...);
- AVR Smart Cards (AT90SEE...).

Найбільш різноманітне використання знаходять мікроконтролери групи Mega AVR завдяки досить високим обсягам пам'яті, що доступна для запису і роботи програм, розвиненій периферії та низькій вартості.

Мікроконтролери Tiny AVR, що появились пізніше, виготовляються в малих за розмірами корпусах (на 8 виводів), мають низьку вартість і найбільш важливі для практики периферійні пристрої.

Розробники AVR контролерів заклали в їх архітектуру ряд ідей, які дозволили забезпечити максимально вигідні співвідношення “вартість–продуктивність–енергоспоживання”:

- використання сучасної найбільш швидкодіючої КМОН–технології в поєднанні з RISC архітектурою для виробництва швидких 8–розрядних мікроконтролерів, які б мали продуктивність, близьку до 16–розрядних мікропроцесорів і значно перевищували по швидкодії стандартну логіку КМОН.
- розробка архітектури і системи команд AVR в максимальному узгодженні до принципів побудови мови програмування C з тим,

щоб апаратна частина мікро контролера та його система команд використовувались з найвищою ефективністю. Такий підхід створює розумну альтернативу асемблеру не тільки в формуванні компактних кодів, а і в тому плані, що мова програмування C користується високою популярністю серед програмістів, а тому контролери AVR будуть привабливими для великої кількості фахівців близьких професій.

- мікроконтролер функціонально розширюється можливістю програмування в системі ISP (In System Programming) шляхом поєднання Flash-технології з швидкодіючим інтерфейсом SPI.

Підвищення швидкодії мікроконтролера досягається рядом архітектурних особливостей. Перша з них полягає в тому, що АЛП має безпосередній доступ до всіх 32-х регістрів загального призначення, які об'єднані в регістровий файл. Завдяки цьому вибірка вмісту регістрів, виконання операції (команди) та розміщення результату знову в регістровий файл виконується за один цикл, оскільки відповідають проміжні операції з регістром-акумулятором.

Контролер побудований на основі RISC-архітектури. Відомо, що основна ідея архітектур цього типу полягає в збільшенні швидкодії за рахунок скорочення кількості операцій обміну з пам'яттю програм. Для цього кожен команду намагаються розмістити в один елемент пам'яті програм. При обмеженій розрядності елементів в пам'яті це вимушено приводить до скорочення кількості команд процесора. В AVR-контролерах таке протиріччя розв'язане завдяки тому, що елементи пам'яті програм були розширені до 16 розрядів. Як наслідок, на відміну від PIC-контролерів фірми Microchip, з 35 командами, в AVR-контролерах кількість досягає 120, що значно спрощує програмування різноманітних задач.

Гарвардська архітектура AVR-контролерів реалізує логічне і фізичне розділення адресних просторів пам'яті даних, а також типи доступу і способи адресації до них також різні. Така побудова близька до структури швидкісних цифрових сигнальних процесорів і забезпечує суттєве підвищення продуктивності як за рахунок одночасної роботи процесора з пам'яттю програм та даних, так і за рахунок використання технології конвейеризації. Остання особливість дає можливість скоротити тривалість машинного циклу (вибір команди та її виконання) до одного такту тактового генератора. Саме ця можливість дозволяє AVR-мікроконтролерам мати найкраще співвідношення енергоспоживання – продуктивність, оскільки забезпечує таку ж швидкість виконання операцій як, наприклад, PIC-контролери при значно нижчій частоті. Енергоспоживання КМОН-мікросхем, як відомо, визначається їх тактовою частотою. Оскільки мікроконтролери мають повністю статичну структуру, то вони можуть працювати на будь-якій частоті в межах від 0 до максимальної (4–12 МГц), що задається для кожного типу контролера.

(Для більшості мікроконтролерів сімейства Classic AVR частота тактового генератора задається зовнішнім кварцовим, або керамічним резонатором, що приєднується до виводів XTAL1 та XTAL2, які одночасно через конденсатори 20–30 пФ з'єднуються з загальною шиною).

Мікроконтролери AVR мають єдину базову структуру, яку часто називають ядром мікро контролерів AVR, а також різноманітні периферійні пристрої. Склад периферійних пристроїв змінюється від моделі до моделі, але є такі, що присутні в усіх контролерах сімейства. До них відносяться: сторожовий таймер, 8-розрядний таймер-лічильник реального часу (TO), порти вводу/виводу. Серед контролерів сімейства Classic AVR найбільш розвинену архітектуру має AT90S8535, структурна схема якої приведена на рис. 1.29.

Порівняння структурної схеми AVR-контролера з структурою процесора, показує, що сучасний контролер поєднує в собі об'єднані в єдине ціле процесор, постійну та оперативну пам'ять, розвинену систему керування та порти, що мають ряд виводів багатофункціонального призначення.

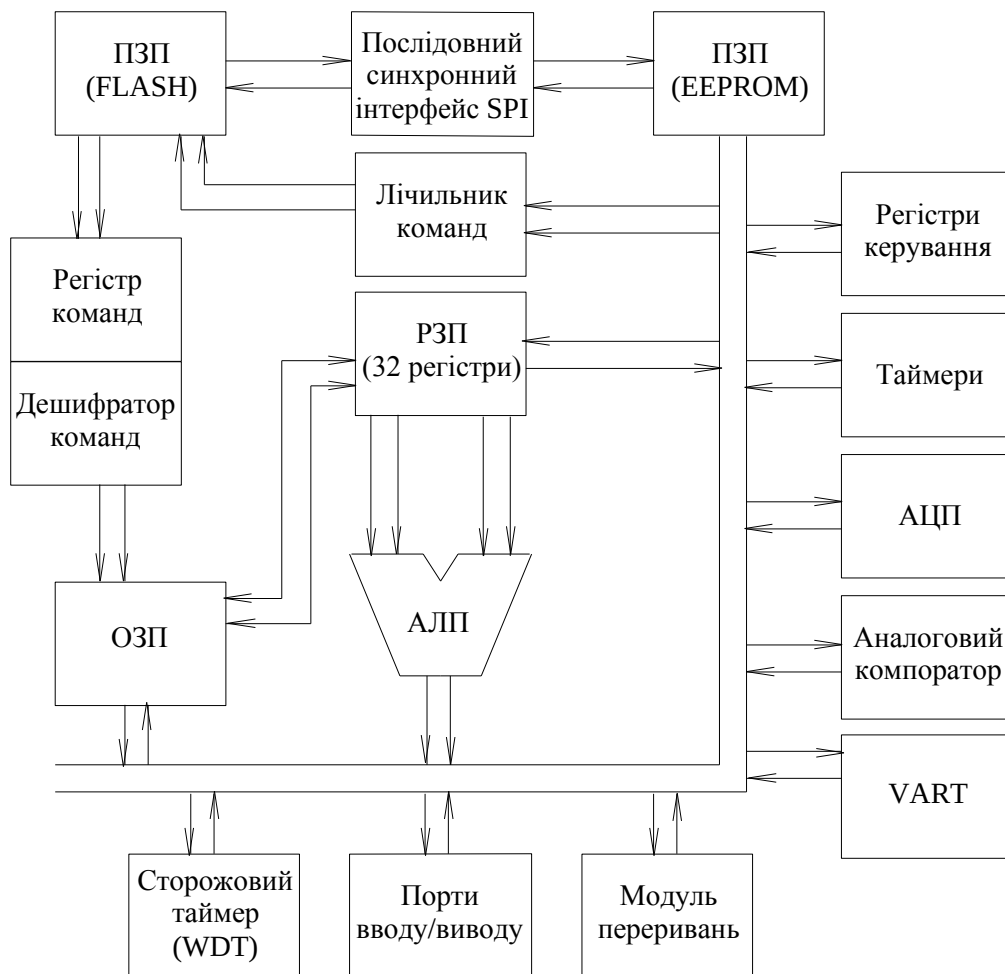


Рис. 1.29.

1.10.1. Основні характеристики процесора

Власне процесор об'єднує в собі наступні модулі:

- лічильник команд(program counter (pc));
- арифметично–логічний пристрій (arithmetral and logic unit (ALU));
- блок регістрів загального призначення (general purpose registers (GPR));
- регістр стану мікроконтролера (status register (SREG));
- регістр–вказівник стеку (stack pointer (SP)). Оскільки він містить адреси, то має 16 розрядів і розподіляється на SPL і SPH, тобто регістри молодших (low) та старших (hith) розрядів).

Лічильник адресів команд є 16–розрядним двійковим лічильником, в якому перед початком роботи контролера заноситься код нульової адреси, з якої і починається звернення до ПЗП. Адресс чергової команди формується шляхом інкременту (додавання 1) до числа, код якого зберігається в лічильнику.

АЛП виконує арифметичні та логічні операції. Деякі мікроконтролери AVR мають апаратний перемножувач, розміщений в АЛП. Операнди в АЛП подаються з блоку РЗП (GPR). Результат виконання операції також розміщується в РЗП. РЗП містить 32 восьмирозрядних регістрів з іменами R0, R1, R2,...R31 і однаковий для всіх AVR. Всі регістри безпосередньо можуть бути доступними для АЛП на відміну від архітектур інших типів процесорів, в яких результат записується в регістр спеціального призначення – акумулятор. Всі регістри об'єднані в файл, структура якого приведена на рис. 1.30. Регістри R26–R31 використовуються як покажчики при використанні непрямої адресації. Для цього вони об'єднуються в пари X,Y,Z для створення коду 16–розрядної адреси. Для AT90S1200 в якості регістра покажчика використовується один регістр R30. Регістровий файл має свій власний діапазон адрес в адресному просторі пам'яті даних, тому до регістрів R0–R31 можна звертатись як до пам'яті даних, незважаючи на те, що фізично вони відносяться до елементів ОЗП.

	7	0
R0		\$00
...		
R15		\$0F
R16		\$10
R17		\$11
...		
R26		\$1A → Reg XL
R27		\$1B → Reg XH
R28		\$1C → Reg YL
R29		\$1D → Reg YH
R30		\$1E → Reg 2L
R31		\$1F → Reg 2H

Рис. 1.30.

Регістр стану мікроконтролера (регістр прапорців) SREG має свою власну адресу \$3F. Більшість його розрядів використовується для зберігання прапорців результатів виконання арифметичних та логічних операцій, що виконуються в АЛП. Сьомий розряд регістра SREG–7 використовується для забезпечення дозволу чи заборони переривань (I–interrupt).

При I = 0 всі переривання заборонені, при I = 1 – дозволяється будь-яке переривання.

Розряд SREG–6 (T) використовується для зберігання біта при виконанні операцій з бітами.

Решта розрядів буде розглянута пізніше при вивченні системи команд.

Регістр – показчик стеку SP зберігає і формує адреси при зверненні до стеку (буфер типу FIFO). В мікроконтролерах типу AVR, а також в AT90S1200 використовується апаратний стек, який призначений для зберігання адрес повернення при перериваннях та зверненнях до підпрограм. В системі команд цих контролерів відсутні команди звернення до стеку.

В інших типах мікроконтролерів AVR в якості стеку використовується область ОЗП, що вибирається користувачем. В системі команд передбачені команди звернення до стеку. При запуску мікроконтролера в регістр SP заноситься під число 0. Для роботи стека в нього необхідно занести початкову адресу вибраного стеку. Запис байтів в стек виконується по інкрементуванню адресів RAM.

В мікроконтролерах з великим обсягом RAM SP складається з двох регістрів – SPL і SPH з адресами відповідно \$3D і \$3E.

1.10.2. Організація пам'яті

Організація пам'яті мікроконтролерів AVR виконана по Гарвардській архітектурі в відповідності до якої розділеними є не тільки адресні простори пам'яті програм і пам'яті даних, а і шини доступу до них. При цьому адресний простір пам'яті даних розподілений на три області: регістрова пам'ять, статичний ОЗП та ОЗП на основі EEPROM знаходиться в адресному просторі ОЗП, то їх завжди розглядають як одну, EEPROM знаходиться в своєму адресному просторі.

На рис. 1.31 приводиться карта пам'яті даних для AT90S8535.

Регістрова пам'ять. Адресація розписана в відповідності до технічної документації на AVR. По суті регістри процесора є елементами швидкої пам'яті для тимчасового зберігання різних кодів, даних, адрес. Операції з внутрішніми регістрами виконуються максимально швидко, тому збільшення кількості внутрішніх регістрів сприяє підвищенню швидкодії процесора. Існує два крайні погляди на призначення внутрішніх регістрів.

R0	\$0000
GPR	
R31	\$001F
I/O RO	\$0020
64 I/O	
I/O R63	\$005F
	\$0060
INT SRAM (512*8)	
	\$025F
EEPROM (512*8)	\$000
	\$1FF

Рис. 1.31.

Один з них – це строго функціональне призначення кожного з них, або окремих груп. Такий принцип закладено в процесори фірми Intel. В деякій мірі він спрощує організацію процесора та підвищує час виконання окремих команд. Але в ряді випадків такий підхід приводить до появи вузьких місць при виконанні програм, оскільки може використовуватись лише обмежена кількість регістрів. Прикладом може служити використання єдиного регістра – акумулятора при виконанні арифметико – логічних операцій та операцій вводу – виводу. Другий підхід полягає в тому, що всі, або майже всі регістри є рівноправними при виконанні різних операцій. Такий підхід використовується в процесорах фірм Atmel, DEC. Фірма Motorola використовує різного рівня компромісні варіанти.

Адресний простір регістрів вводу–виводу \$0020 \$005F включає в себе керуючі регістри таймера–лічильника, аналого–цифрового перетворювача та інші. Звернення до цих регістрів може бути безпосереднім, або як до оперативної пам'яті.

Всі регістри вводу–виводу умовно можуть бути розділені на дві групи: службові регістри мікроконтролера і регістри, що безпосередньо відносяться до периферійних пристроїв. Всі регістри мають однаковий розмір – 8 розрядів. Подібно до РЗП до регістрів вводу–виводу можна звертатись як безпосередньо до регістрів за допомогою відповідних команд, так і як до елементів ОЗП з використанням їх адресів в загальному адресному просторі.

В табл. 1.2 приведені назви регістрів, що використовуються в мікроконтролері AT90S8535. Слід зазначити, що це повний перелік регістрів сімейства AVR і їх адреси, зазначені в таблиці для всіх контролерів, однакові.

Табл. 1.2.

№	Назва регістра	Адреса	Призначення
1	SPEG	\$5F	Регістр стану
2	SPH	\$5E	Показчик стеку (старші розряди)
3	SPL	\$5D	Показчик стеку (молодші розряди)
4	GIMSK	\$5B	Керування зовнішніми перериваннями
5	GIFR	\$5A	Керування зовнішніми перериваннями
6	TIMSK	\$59	Регістр переривання від таймерів
7	TIFR	\$58	Регістр переривання від таймерів
8	MCVCR	\$55	Регістр керування мікро контролера
9	MCVSR	\$54	Регістр стану мікро контролера

SREG (регістр стану). В розрядах регістра розміщуються прапорці (“1” або “0”), які характеризують поточний стан мікроконтролера. Відповідні розряди автоматично встановлюються в “1” або “0” при появі деяких ситуацій під час виконання команд. Всі розряди регістра доступні для читання і запису в будь-який момент часу. Початковий стан кожного біта регістра – нульовий – 00h.

Призначення кожного біта регістра приведено на рис. 1.32.

7	6	5	4	3	2	1	0
I	T	H	S	V	N	Z	C

Рис. 1.32.

C – прапорець переносу. Цей розряд встановлюється в 1, якщо розмір отриманого за результатом виконання операції, перевищує байт.

Z – прапорець нуля. Розряд встановлюється в “1”, якщо результат виконання операції дорівнює нулю.

N – Прапорець від’ємного результату. Розряд встановлюється в “1”, якщо старший розряд слова (7-й), що отримане по результату виконання операції, дорівнює одиниці.

V – Прапорець переповнення допоміжного коду. Біт встановлюється в “1” при переповненні розрядної сітки знакового розряду. Використовується при роботі зі знаковими числами, що представлені в допоміжному коді.

S – Прапорець знака. Біт дорівнює результату операції $N \oplus V$. Біт встановлюється в “1”, якщо результат виконання арифметичної операції менше нуля.

H – Прапорець половинного переносу. Біт встановлюється в “1”, якщо має місце перенос між тетрадами байта.

T – зберігання біта, що копіюється. Розряд використовується в якості джерела або приймача командами копіювання бітів. Заданий розряд будь-

якого РЗП може бути скопійованим в цей розряд командою BST, або встановлений в відповідності до вмісту даного розряду командою BLD.

I – загальний дозвіл переривань. Для дозволу переривань біт повинен бути встановленим в “1”. Якщо I = 0, то переривання заборонені. Біт встановлюється в “1” командою RETI для дозволу послідовних переривань, і в “0” після входу в переривання.

Регістр SP реалізований на двох регістрах SPH і SPL. Оскільки для об’єму ОЗП 512 байт необхідно мати 10 розрядів, то в SPH задіяні лише два молодші розряди. Тільки в AT90S4414 і AT90S8515 регістр SPH використовується повністю, оскільки доступний максимальний обсяг пам’яті в них досягає 64 кБ. Всі розряди регістрів доступні як для читання так і для запису. Початковий стан вмісту регістрів дорівнює “0”, тому перед початком роботи програми з використанням стека необхідно передбачити запис нового будь-якого числа. Як правило – це найбільша адреса пам’яті даних.

Регістри GIMSK, GIFR призначені для керування зовнішніми перериваннями. Загальний регістр маски переривань GIMSK використовується для дозволу та заборони окремих переривань, а загальний регістр прапорців переривань GIFR містить прапорці, що показують, відбувалось чи ні відповідне переривання.

Регістри TIMSK та TIFR призначені для керування перериваннями від таймерів. Конкретне призначення регістру TIMSK аналогічне GIMSK, а TIFR, відповідно – GIFR.

Регістр керування мікроконтролером MCVCR містить в собі ряд ознак, що використовуються для загального керування. Структура регістра для AT90S8535 приведена на рис. 1.33.

Всі біти, за винятком 7, доступні для запису та зчитування в будь-який момент часу. Початкове значення вмісту регістра – 00h.

7	6	5	4	3	2	1	0
–	SE	SMI	SMO	ISC11	ISC10	ISC01	ISC00

Рис. 1.33.

Функціональне призначення розрядів регістра наступне. Розряди 0 та 1 визначають умову генерації зовнішнього переривання INT0 відповідно до табл. 1.3.

Табл. 1.3.

№	ISCM	ISC20	Умова
0	0	0	На низькому рівні на виводі INT0
1	0	1	Резерв
2	1	0	На зрізу сигналу на виводі INT0
3	1	1	На фронту сигналу на виводі INT0

Розряди 2, 3 визначають умову генерації зовнішнього переривання INT0 в відповідності до табл. 1.4.

Табл. 1.4.

№	ISC1	ISC10	Умова
0	0	0	По низькому рівню на виводі INT1
1	0	1	Резерв
2	1	0	На зрізу сигналу на виводі INT1
3	1	1	По фронту сигналу на виводі INT1

Розряди 4 та 5 забезпечують вибір режиму пониженого енергоспоживання. Кодові значення розрядів (табл. 1.5) визначають, в який режим перейде мікроконтролер після виконання команди SLEEP.

Табл. 1.5.

№	SMI	SMO	Режим
0	0	0	IDLE
1	0	1	Резерв
2	1	0	POWER DOWN
3	1	1	POWER SAVE

Розряд 6 використовується для забезпечення дозволу режиму пониженого енергоспоживання. Якщо розряд встановлено в 1, то по команді SLEEP мікроконтролер переходить в “сплячий” режим.

Розряд 7 не використовується.

Регістр MCVSR містить в своїх розрядах прапорці, стан яких дозволяє визначити причину, по якій мав місце зрив в роботі програми.

Оперативна пам'ять. Оперативний запам'ятовуючий пристрій статичного типу (SRAM – static access memory) ОЗП, призначений для зберігання даних, що підтримуються як проміжні при роботі процесора. Статична пам'ять, як відмічалось вище, не залежить від частоти, але при відключенні живлення інформація в них губиться.

Елемент пам'яті, як і регістри, містять 8 розрядів, а ємність залежить від типу мікроконтролера, що використовується. AT90S8535 має ємність 512 Байт. Адреса байта при зверненні до ОЗП може бути вказаним в коді команди (пряма адресація) або може бути записана в пару регістрів X, Y, Z (непряма адресація). Звернення до ОЗП може виконуватися з використанням адреси, що зберігається в регістрі стека.

Запис та зчитування даних ОЗП відбувається через РЗП.

Запис в стек виконується по порядку зменшення адресів. В якості початкової адреси для звернення до стеку, яка заноситься в регістр – показчик стеку на початку програми рекомендується записати старшу адресу ОЗП (\$625F).

Запам'ятовуючий пристрій EEPROM. Постійний запам'ятовуючий пристрій EEPROM призначений для зберігання енергонезалежних даних,

що записуються при програмуванні мікроконтролера, а також отримані в процесі виконання програми. Це можуть бути різноманітні коефіцієнти, установки, конфігураційні параметри системи і т.п. Тобто EEPROM – пам'ять даних може бути програмним шляхом зчитуватись та записуватись. Вона має відокремлений адресний простір, діапазон якого приведений на рис. 1.31. Спеціальних команд звернення до EEPROM не існує. Операції запису (зчитування виконуються через регістри вводу–виводу EEAR (регістр адреси), EEDR (регістр даних) та EECR (регістр керування). Ці регістри мають, відповідно, адреси для мікроконтролера AT90S8535.

EEARH – \$3F;

EEARL – \$3E;

EEDR – \$30;

EECR – \$3C.

Особливості програмування EEPROM і FLASH – пам'яті будуть розглянуті пізніше.

Запам'ятовуючий пристрій FLASHROM. FLASHROM призначений для зберігання кодів команд програм та констант. Оскільки всі команди AVR мають 16–розрядний формат, то FLASH – пам'ять організована як послідовність 16–розрядних елементів і для AT90S8535 має обсяг 8 кБайт (4K×16) (рис. 1.34)

Крім програмного коду в FLASH–пам'ять записуються також постійні дані, що не змінюються за час виконання програми. Це можуть бути таблиці знака генераторів, таблиці лінеаризації датчиків і т.п

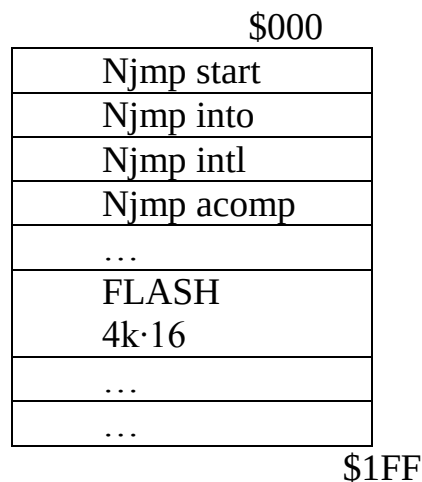


Рис. 1.34.

Молодші адреси пам'яті програм мають спеціальне призначення. По нульовій адресі розміщується код, з якого починається виконання програми. Послідовні елементи пам'яті створюють в собі команди відносного переходу RYNP на підпрограми обробки переривань. Запис кодів в FLASHROM виконується в процесі програмуванні побайтно. В залежності від типу мікроконтролера байт може вводиться послідовно, або паралельно. Програмування може виконуватись з використанням допоміжного джерела або без нього. Послідовне програмування без

використання допоміжного джерела живлення забезпечується через послідовний порт вводу–виводу SPI (Serial Peripheral Interface). Такий спосіб програмування може виконуватись після установки мікроконтролера в апаратури (Downloading).

Зчитування даних з FLASHROM програмним шляхом забезпечується в регістровий файл за допомогою команд передачі даних.

1.11. Характеристика периферійних пристроїв

Всі периферійні пристрої мікроконтролера безпосередньо з'єднані з внутрішньою шиною даних, що, з однієї сторони, забезпечує їм рівні можливості в організації обміну інформацією, а з іншої, можлива лише в режимі демультіплексування. Розглянемо коротко основні функціональні можливості кожного з них

Паралельні порти вводу–виводу (містять в собі пристрої керування, регістри DDRX, регістри напрямку. При вмиканні живлення всі порти налаштовуються на ввід байта, а для забезпечення виводу необхідно попередньо налаштувати кожен з портів шляхом посилки в регістр напрямку коду FFh.

Виводи порту можуть виконувати альтернативні функції при роботі інших пристроїв.

Послідовний порт вводу–виводу SPI (Serial Peripheral interface) призначений для обміну байтами інформації з іншими пристроями, що мають такі ж порти. Обмін відбувається під керуванням тактового сигналу порту. Пристрій, що ініціює обмін, виробляє тактовий сигнал і є ведучим (master). Пристрій, що виконує обмін при прийомі тактового сигналу, є веденим (slave). За час обміну обидва пристрої послідовно, біт за бітом одночасно видають та приймають байт. Обмін забезпечується по 3–х провідній шині. Швидкість обміну дорівнює $\frac{1}{4}$ тактової частоти мікроконтролера.

Ведучий і ведений контролери можуть змінюватись під час роботи системи. До одного ведучого мікроконтролера може бути приєднано декілька ведених.

Послідовний порт вводу–виводу UART (Universal Asynchronous Receiver Transmitter) призначений для передачі та прийому байтів даних по двохпровідних лініях зв'язку, наприклад, по інтерфейсу RS232C або “струмова петля”. Порт UART містить в собі передавач, приймач, тактовий генератор і апаратуру керування передачею та прийомом. Передавач приймає байт з внутрішньої шини даних мікроконтролера, формує і видає послідовність бітів (кадр), яка містить в собі інформаційний байт, стартовий і допоміжний біти (всього 11 розрядів).

Приймач виділяє стартовий біт, приймає кадр і виділяє з нього дані для передачі на шину даних мікроконтролера.

Тактовий генератор формує сигнал, що визначає швидкість передачі та прийому бітів кадру і дозволяє виділяти біти при прийомі кадру.

Послідовний порт вводу–виводу TWSI (Two wire serial Interface) призначений для обміну байтами з іншими пристроями по двохпровідній шині IC (Integrated Circuit). До шини може бути приєднано до 127 пристроїв. Пристрої, що приєднуються до шини IC, можуть працювати як ведучі, так і ведені. Ведучий пристрій при звільненні шини від обміну між іншими посилає в шину сигнал початку обміну і займає шину. Потім посилає адресний байт для виводу одного з ведених пристроїв і завдання напрямку обміну. Після цього передає або приймає байти даних і посилає сигнал закінчення обміну.

Ведений пристрій після розпізнавання адреси і визначення напрямку передачі передає або приймає дані, що підлягають обміну. Пристрій, що приймає дані, після кожного прийнятого байта подає в шину сигнал, що підтверджує прийом. Ведені пристрої можуть мати загальну адресу. В такому випадку всі вони можуть сприймати дані.

Таймер–лічильник загального призначення (General purpose Timer/Counter) призначений для формування захисту на переривання по закінченню заданого інтервалу часу (режим таймера), або появи деякої кількості подій (режим лічильника). Мікроконтролери AVR можуть мати від одного до трьох таймерів–лічильників.

Основним елементом таймера–лічильника є базовий лічильник, який веде відлік на додавання. При його переповненні формується запит на переривання. Таймер–лічильник може виконувати допоміжні функції: захоплення, порівняння, широтно–імпульсного модулятора, підрахунку реального часу.

Функція захоплення (capture) полягає в запам'ятовуванні коду, що формується в базовому лічильнику, в спеціальному регістрі захоплення при зміні значення деякого зовнішнього або внутрішнього сигналу. При цьому формується запит на переривання.

Функція порівняння (compare) полягає в зміні значення сигналу на деякому виході мікроконтролера при співпадінні коду, що формується в базовому лічильнику, з кодом в спеціальному регістрі порівняння. При цьому формується запит на переривання.

Функція широтно–імпульсного модулятора (PWM) полягає в формуванні на деякому виході мікроконтролера імпульсної послідовності з заданим періодом повторення і тривалістю імпульсів.

Функції порівняння PWM реалізується з використанням одного і того ж обладнання. Вибір необхідної функції забезпечується програмними засобами.

Функція відліку реального часу (Real Time Clock) реалізується в таймері–лічильнику при використанні допоміжного внутрішнього генератора з зовнішнім кварцовим резонатором з частотою 32768 Гц (годинний кварц). При цьому параметри процесів в таймері–лічильнику з високою точністю прив'язані до одиниці вимірювання реального часу – секунд.

В залежності від розрядності лічильника і допоміжних функцій, що виконуються, можуть бути виділені п'ять типів таймерів–лічильників загального призначення, що входять в групу периферійних пристроїв мікроконтролерів сімейства Classic AVR.

Тип А – восьмирозрядний таймер–лічильник без допоміжних функцій.

Тип В – восьмирозрядний таймер–лічильник з функцією порівняння – PWM.

Тип С – восьмирозрядний таймер–лічильник з функцією порівняння – PWM і функцією підрахунку реального часу,

Тип D – шістнадцятирозрядний таймер–лічильник з функціями захоплення і порівняння – PWM.

Тип Е – шістнадцятирозрядний таймер–лічильник з функцією захоплення і двома каналами для виконання функцій порівняння – PWM.

Сторожовий таймер (Watchdog Timer, WDT) призначений для усунення результату збою по ходу виконання програми шляхом перезапуску мікроконтролера при його виявленні. WDT вмонтований в усі типи мікроконтролерів AVR.

Аналого–цифровий перетворювач (Analog-to-Digit Converter) ADC призначений для формування десятирозрядного двійкового коду числа, пропорційного величині напруги аналогового сигналу на вході мікроконтролера. В мікроконтролерах AVR до ADC може приєднуватись від 4–х до 8–ми входів.

Аналогово–цифровий компаратор (Analog Comparator, AC) призначений для порівняння по величині аналогові сигнали, що поступають на два входи мікроконтролера і формування запиту на переривання ANACOMP, якщо різниця між їх значеннями змінює знак. При цьому також видається сигнал для виконання функції захоплення в таймері–лічильнику загального призначення.

Блок переривань (Interrupt Unit, IU) призначений для організації програмного переходу до виконання перекриваючої програми при появі запиту на переривання, якщо він дозволений по даному запиту з урахуванням пріоритетів. З попереднього матеріалу відомо, що дозвіл на переривання дається, якщо в розряді I (7) регістру SREG встановлене “1”: “1” встановлена в відповідному регістрі вводу–виводу, що дозволяє переривання по даному запиту.

При переході до обслуговування програми по перериванню розряд I SREG встановлюється в “0”, і залишається в цьому стані до виконання програми. Він може бути переведений в одиничний стан по команді з програми, що виконується. Запити в блок переривань поступають як від внутрішніх, так і зовнішніх по відношенню до мікроконтролера пристроїв.

Питання до розділу 1

1. Принципи побудови мікропроцесорних систем.
2. Фон-Нейманівська та гарвардська структури мікропроцесорів.
3. Особливості виконання команд у 8-розрядному мікропроцесорі.
4. Взаємодія мікропроцесорів із зовнішніми пристроями. Поняття інтерфейсу.
5. Призначення та принцип дії основних блоків мікроконтролера.
6. Способи адресації операндів.
7. Архітектура однокристального мікропроцесора.
8. Байти стану для різних типів машинних циклів.
9. Організація підсистеми вводу/виводу, система переривань.
10. Системи команд мікропроцесора.
11. Поняття про програмування мовою Асемблер.
12. Призначення внутрішніх вузлів мікроконтролера.
13. Призначення та робота з внутрішньою пам'яттю мікроконтролера.
14. Призначення стеку.
15. Команди роботи зі стеком, послідовність дій при роботі зі стеком.
16. Час виконання команди, поняття машинного циклу.
17. Режими роботи мікропроцесорних систем.
18. Режим прямого доступу до пам'яті (DMA).
19. Основні характеристики мікропроцесора.
20. Структура мікроконтролерів сімейства AVR.
21. Система команд мікропроцесора.
22. Функції пристроїв вводу/виводу.

2. ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ НА МОВІ СІ

Історично першою мовою програмування була машинна мова. Вона була дуже незручною для користувача, тому що потрібно було оперувати машинними кодами у двійковому або шіснадцятковому представленні. При вдосконаленні апаратної частини з'явилася інша, більш орієнтована для роботи з користувачем мова – асемблер. Для написання програм в асемблері використовуються мнемонічні коди машинної мови. Асемблер є досить незручним з погляду читабельності при написанні великих програм, для налагодження й пошуку помилок. Однак асемблер активно використовується для написання програм під мікроконтролери у зв'язку з компактністю коду та швидкістю його виконання. Як у програмуванні для комп'ютера, так і для програмування контролерів, на заміну асемблеру приходять мови програмування високого рівня. У цей час існує досить велике число компіляторів для роботи з мікроконтролерами, що реалізують різні мови програмування, такі як Basic, Pascal, Сі. Використання мови Сі є перспективним і дуже зручним для користувача. Переваги мови Сі у порівнянні з асемблером:

- легкість вивчення мови для початківців;
- простота написання програм;
- легкість читання;
- структурованість коду;
- можливість написання власних бібліотек;
- простота налагодження й пошуку помилок.

До недоліків можна віднести те, що код буде не оптимальним.

Існує велика кількість компіляторів мови Сі для AVR мікроконтролерів, які розроблялися для роботи з мовами високого рівня. Деякі з них є платними, інші безкоштовними або умовно безкоштовними. Основним критерієм при виборі того або іншого компілятора є наявність додаткових заголовних файлів для полегшення роботи з периферією та зовнішніми пристроями, а також умови поширення.

2.1. Компілятор мови Сі ImageCraft

Середовище розробки ImageCraft представляє собою компілятор мови Сі, графічну оболонку та автоматичний генератор програм, орієнтований на роботу із сімейством мікроконтролерів AVR фірми Atmel. Зовнішній вигляд вікна програми показаний на рис. 2.1.

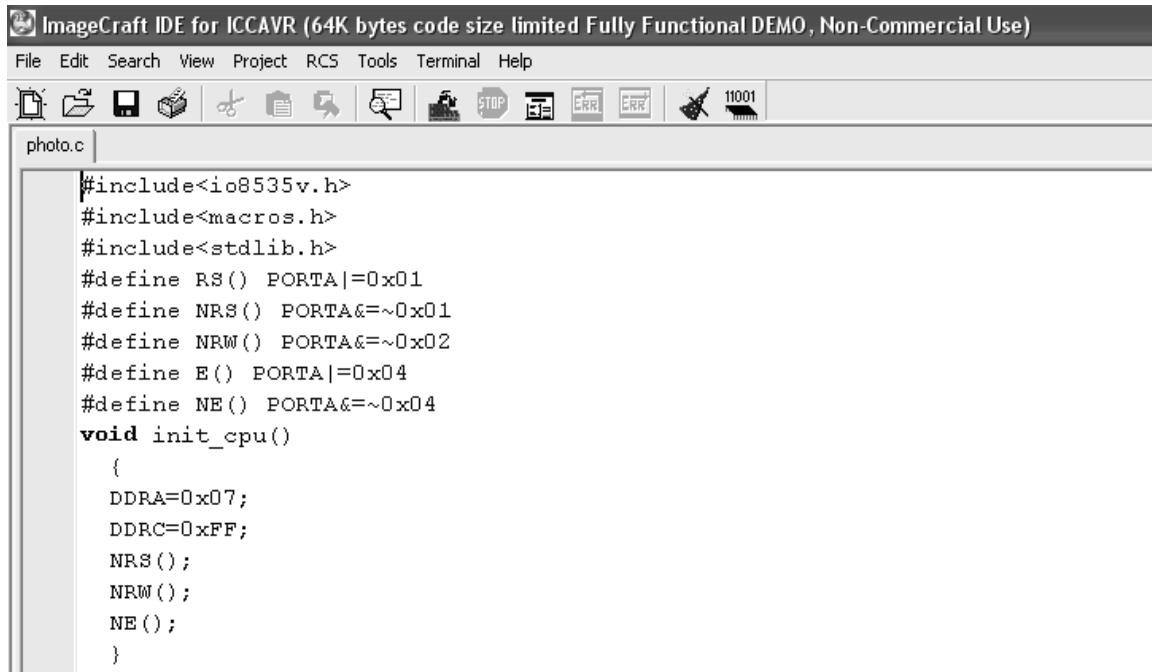


Рис. 2.1.

Програма представляє собою 32-розрядний додаток для роботи в операційних системах Windows 2000, XP.

Компілятор містить у собі майже всі елементи, що відповідають стандарту ANSI. Також у компілятор включені додаткові можливості, орієнтовані на використання архітектурних особливостей мікроконтролерів цього сімейства.

Об'єктні файли COFF дозволяють здійснювати налагодження програм з переглядом вмісту змінних. Для цього слід застосовувати вільно розповсюджуваний фірмою Atmel (www.atmel.com) відлагоджувач AVR Studio.

В ImageCraft є автоматичний генератор програм, що дозволяє протягом лічених хвилин одержати готовий код для наступних функцій:

- налаштування доступу до зовнішньої пам'яті;
- визначення джерела переривання Reset;
- ініціалізація портів введення/виведення;
- ініціалізація зовнішніх переривань;
- ініціалізація таймерів/лічильників;
- ініціалізація сторожового таймера;
- ініціалізація UART;
- ініціалізація аналогового компаратора;
- ініціалізація вбудованого АЦП;
- ініціалізація інтерфейсу SPI.

Середовище ImageCraft містить у собі програмне забезпечення для роботи з платою STK200. Після компіляції вихідної програми мовою Cі, отриманий код може бути безпосередньо запрограмований у

мікроконтролер. Цей програматор використовує всього чотири сигнали: MOSI, MISO, SCK, RESET. Відомо багато найпростіших програматорів, що з'єднуються з послідовним або паралельним портом IBM сумісного персонального комп'ютера.

Для запуску компілятора ImageCraft вибираємо – меню "ПУСК" → "Програми" → "ImageCraft Development Tools" → "ICCV7 for AVR". Основне вікно містить стандартні елементи Windows-вікон – "Головне меню", "Панель інструментів", "Рядок стану". Основна робоча область розділена на три частини: "Редактор", "Вікно навігації" та "Вікно повідомлень". При першому запуску програми, всі три області порожні, але при наступних запусках програми можливий інший їхній стан. "Основне меню" містить кнопки "File", "Edit", "Search", "View", "Project", "RCS", "Tools", "Terminal" та "Help". На рис. 2.2 представлені меню програми: а – меню File; б – меню Edit; в – меню Search.

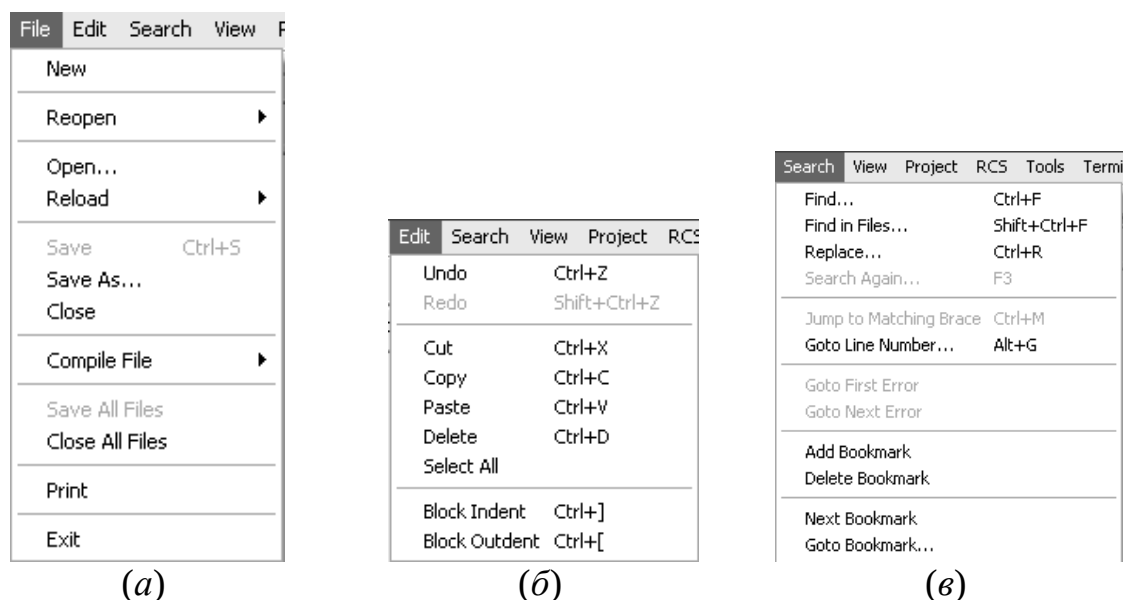


Рис. 2.2.

Меню "File" (рис. 2.2, а) містить наступні підменю:

- "New" – використовується для створення нового файлу;
- "Reopen" – для відкриття останніх файлів, що використовувалися;
- "Open" – відкрити файл;
- "Save" – збереження;
- "Save as..." – зберегти як (можливо в іншому місці або з іншим ім'ям);
- "Close" – закриття поточного файлу;
- "Compile File" – компілювати файли;
- "Save All Files" – зберегти всі файли;
- "Close All Files" – закрити всі файли;
- "Print" – друкування поточного файлу;
- "Exit" – вихід із програми.

Меню "Edit" (рис. 2.2, б) містить команди для роботи з текстом:

"Undo" – скасувати останню дію;
 "Redo" – повернути скасовану дію;
 "Cut", "Copy", "Paste", "Delete" – вирізати, копіювати, вставити, видалити виділений фрагмент тексту;
 "Select All" – виділити все;
 "Block Indent" / "Block Outdent" – зрушити виділений фрагмент праворуч/ліворуч.

Меню "Search" (рис. 2.2, в) містить команди для роботи з текстом, пошуку заміни й переходу:

"Find" – знайти текст у поточному файлі;
 "Find in Files" – знайти текст у всіх файлах проекту;
 "Replace" – замінити фрагмент тексту;
 "Go to Line Number" – перейти на рядок із зазначеним номером;
 "Add Bookmark" / "Delete Bookmark" – додати / видалити закладку.

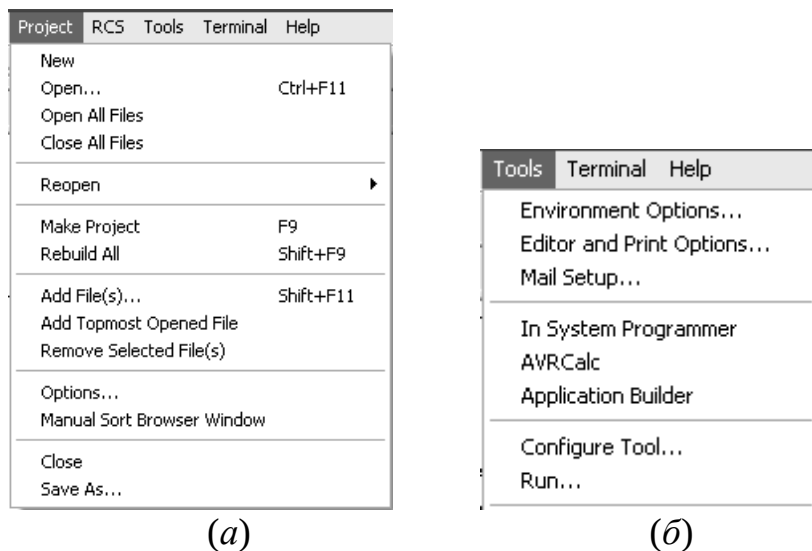


Рис. 2.3.

Меню "Project" (рис. 2.3, а) дозволяє робити різні дії із проектом:

"New" – створити новий проект;
 "Open" – відкрити існуючий проект;
 "Reopen" – відкрити один з останніх проектів;
 "Make project" – компілювати проект;
 "Rebuild All" – перекомпілювати проект;
 "Add Files" – додати файли в проект;
 "Options" – налаштування властивостей проекту;
 "Close" – закрити проект,
 "Save As..." – зберегти проект.

Меню "Tools" (рис. 2.3, б) містить інструменти та різні утиліти для полегшення роботи з компілятором:

"Application Builder" – майстер, що допомагає налаштувати проект під конкретний пристрій, забезпечує генерацію коду для початкової ініціалізації пристрою;

"AVR Calc" – спеціальний калькулятор, що дозволяє переводити числа з однієї системи счислення в іншу, а також розраховувати значення дільників частоти для таймерів/лічильників, UART;

"In System Programmer" – вбудований програматор контролерів для програмування через SPI-інтерфейс;

"Configure Tool" – налаштування меню, дозволяє додавати в дане меню зовнішні програми для запуску (наприклад альтернативний програматор);

Меню "Terminal" – вікно термінала для роботи з UART, дозволяє налаштувати COM-порт.

Меню "Help" – допомога, містить розділи допомоги.

Панель інструментів забезпечує швидкий доступ до елементів меню, що найбільш часто використовуються. Панель інструментів містить наступні кнопки, розташування й розміри яких не змінюються.



– створити новий файл;



– відкрити файл;



– зберегти файл (Ctrl+S);



– друк файлу;



– вирізати виділений фрагмент тексту (Ctrl+X);



– копіювати виділений фрагмент тексту (Ctrl+C);



– вставити (Ctrl+V);



– знайти (Ctrl+F);



– побудувати проект (Shift+F9);



– налаштування проекту;



– перейти до місця першої помилки;



– знайти наступну помилку;



– запустити майстер Application Builder для створення нового файлу;



– запустити вбудований програматор.

Основне вікно програми складається із трьох дочірніх вікон: "Редактор", "Вікно проекту", "Вікно стану". У вікні "Редактора" відображається текст файлу, що редагується. У "Вікні проекту" відображається у вигляді дерева список включених у проект файлів (заголовних і файлів з кодом програми), список всіх функцій, список

глобальних змінних, файл із позначками (Notes). Всі попередження й помилки з посиланнями на рядок, у якому вони виникають, з'являються у "Вікні стану". Між всіма елементами можна перемикатися за допомогою миші.

Як і більшість C1-компіляторів, ImageCraft працює не з окремим C1-файлом, а з проектом. Для створення нового проекту вибираємо Project → New, і у діалоговому вікні, що відкрилося, вибираємо проект. Бажано кожен новий проект зберігати в окремому каталозі, тому що компілятор створює багато файлів. Після створення проекту, необхідно створити файл, у якому буде написана програма, а потім додати цей файл у проект. Для цього з меню File вибирається команда New. Потім бажано відразу файл зберегти, вибравши команду File → Save As... У діалоговому вікні слід ввести назву файлу, і обов'язково вказати розширення файлу «.c». Після цього натисканням правої кнопки миші на назві проекту у вікні проекту викликати контекстне меню, з якого вибрати команду «Add File(s)». Після цього можна писати програму.

Майстер Application Builder (рис. 2.4) використовується для швидкого та зручного налаштування периферії та зовнішніх підключених пристроїв. Майстер дозволяє вибрати тип процесора (Target CPU), тактову частоту (Xtal Speed), режим роботи портів вводу-виводу (Ports), налаштувати роботу аналогового компаратора та АЦП (Analog), роботу SPI, налаштувати зовнішні переривання (External interrupts), роботу таймерів-лічильників (Timers), протокол UART. Зовнішній вигляд вікна залежить від типу обраного контролера та периферії, що використовується.

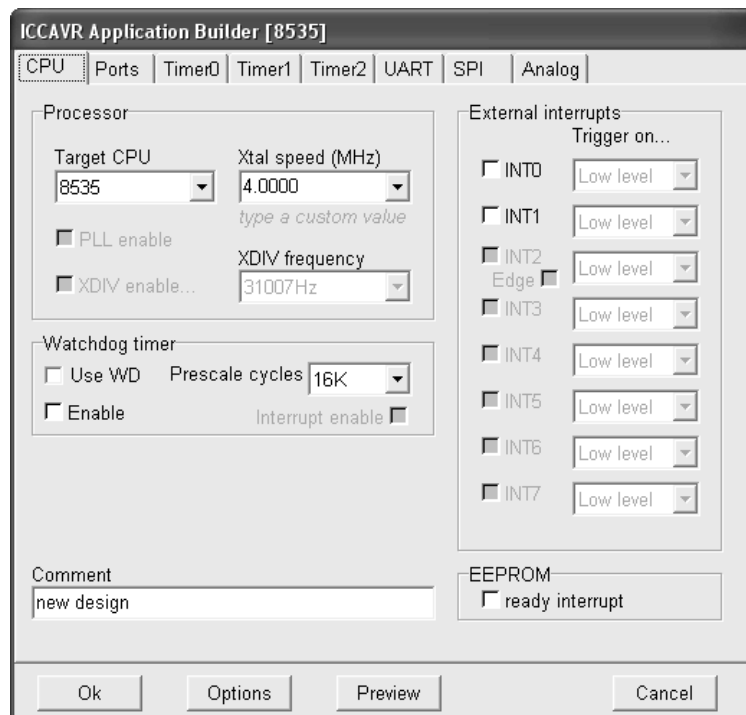


Рис. 2.4.

Для перегляду коду програми можна натиснути на кнопку Preview. При використанні переривань, Application Builder автоматично генерує код прототипів функцій обробки переривань.

Перед написанням програми, властивості кожного проекту необхідно налаштувати. Для цього вибираємо Project → Options. З'явиться діалогове вікно, показане на рис. 2.5. На першій вкладці (Paths) вказуються шляхи до бібліотек, включених в проект, для побудови мульти-файлового проекту; на третій вкладці (Target) вибираємо тип контролера, розмір стека. Налаштування на другій і четвертій закладках можна залишити без змін.

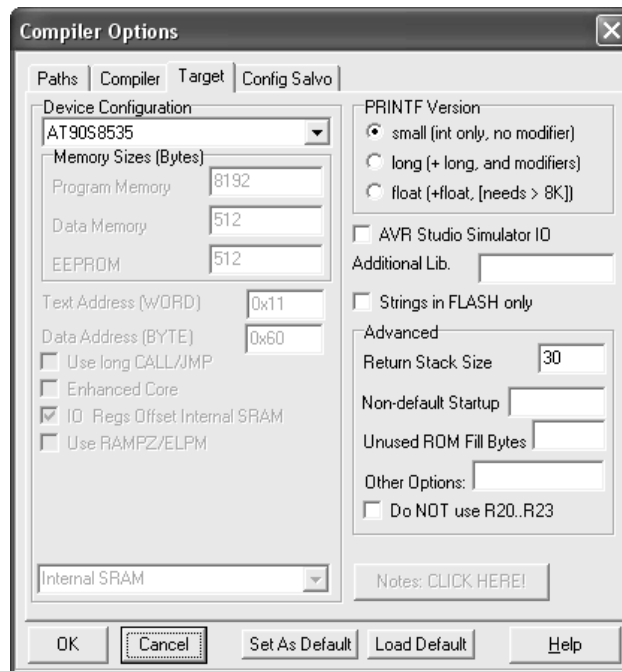


Рис. 2.5.

2.2. Синтаксис мови

Синтаксис мови AVR Cі компілятора ImageCraft практично повністю відповідає міжнародному стандарту ANSI-C, але оскільки він призначений для роботи з мікроконтролерами, то має деякі особливості.

2.2.1. Коментарі в програмі

Всі символи, розташовані між /* та */, ігноруються компілятором, і цим можна вільно користуватися, щоб зробити програму більш зрозумілою. Коментар можна розташовувати в будь-якому місці, де можуть стояти символи пробілу, табуляції або символ нового рядка. Так само підтримуються коментарі стандарту мови C++, наприклад:

```
/* Це коментарі */  
// Це коментарі
```

2.2.2. Зарезервовані слова

Компілятор резервує деякі слова, наприклад, такі як `break`, `double`, `for`, необхідні для керування програмою, що представляють собою ідентифікатори типів, деякі стандартні функції. У цей список входять як зарезервовані слова стандарту ANSI, так і специфічні слова, що відносяться тільки до мікроконтролерів, такі як `eeprom`, `flash`. Список всіх зарезервованих слів можна знайти в розділах допомоги.

Імена складаються з букв латинського алфавіту (A...Z; a...z) і цифр (0...9); першим символом повинна бути буква. Символ підкреслення "_" вважається буквою; його іноді зручно використовувати, щоб поліпшити сприйняття довгих імен змінних. Не слід починати імена змінних з підкреслення, тому що багато змінних бібліотечних функцій починаються саме із цього знака. Великі (прописні) і малі (рядкові) букви розрізняються, так що `x` та `X` – це два різних ідентифікатори.

Для імен змінних значимими є 31 символ. Бажано давати змінним осмислені імена відповідно до їх призначення, причому такі, щоб їх було важко сплутати один з одним.

Для компілятора ImageCraft існує також список змінних, зарезервованих за периферією контролера, аналогічних відповідним регістрам асемблера. Список зарезервованих слів залежить від типу контролера й описується в його заголовному файлі.

2.2.3. Типи даних

Компілятор ImageCraft працює з наступними типами даних (табл. 2.1):

Табл. 2.1.

Тип	Розмір (Біт)	Діапазон
bit	1	0 or 1
char	8	-128 to 127
unsigned char	8	0 to 255
signed char	8	-128 to 127
int	16	-32768 to 32767
short int	16	-32768 to 32767
unsigned int	16	0 to 65535
signed int	16	-32768 to 32767
long int	32	-2147483648 to 2147483647
unsigned long int	32	0 to 4294967295
signed long int	32	-2147483648 to 2147483647
float	32	$\pm 1.175 \times 10^{-38}$ to $\pm 3.402 \times 10^{38}$
double	32	$\pm 1.175 \times 10^{-38}$ to $\pm 3.402 \times 10^{38}$

2.2.4. Константи

Константи можуть бути записані в десятковій формі (наприклад 123456), шістнадцятковій з використанням префікса `0x` (наприклад

0x123abc або 0x123ABC), або у двійковій з використанням префікса 0b (наприклад 0b1010010).

Символьна константа є ціле, записане у вигляді символу, обрамленого одиночними лапками, наприклад 'k'. Строкові константи повинні записуватися в обрамленні подвійних лапок, наприклад "Hello world". Якщо текстовий рядок записується в лапках у вигляді параметра функції, то вона сприймається у вигляді строкової константи й міститься у flash пам'яті.

```
/* Ця функція показує рядок, розміщений SRAM */
void display_ram(char *s)
{

}

/* Ця функція показує рядок, розміщений FLASH */
void display_flash(char flash *s)
{

}

void main(void)
{
/* Цей код працювати не буде, тому що функція адресує рядок, розміщений в
SRAM, а рядок "Hello world" - константа, що знаходиться у FLASH */
display_ram("Hello world");

/* Цей код буде працювати тому що функція адресує рядок, розміщений у FLASH
*/
display_flash("Hello world");
}
```

Приклад 2.1. Написати програму, що виводить в порт (наприклад, PORTD) задане число $170_{10}=10101010_2=AA_{16}$.

Створюємо новий проект із назвою First. Для цього вибираємо Project → New і зберігаємо його в каталозі First з назвою First.prj. Майстром Application Builder користуватися не будемо, створюємо новий файл File → New і зберігаємо його в каталозі First з назвою first.c, додаємо сі-файл до проекту та вибираємо тип контролера, що використовується. У редакторі пишемо код програми:

```
#include <io8535v.h>           //підключаємо заголовний файл контролера
void main ()                   // основна функція
{
    DDRD=0xFF;                 //установка напрямку роботи порту D на вихід
    PORTD=0b10101010;         //вивід на порт комбінації
}
```

Перший рядок в програмі – підключення заголовного файлу, у якому описана структура та периферія контролера. Другий рядок – оголошення основної функції програми: DDRD – налаштування всіх виводів порту D на вихід; PORTD – вивід комбінації 10101010 на порт.

2.2.5. Змінні

Однією з основних переваг C1 компілятора, у порівнянні з асемблером, є використання у вихідному коді програми змінних. При цьому користувачеві немає необхідності стежити за розподілом регістрів загального призначення, FLASH та EEPROM пам'яті. Компілятор сам "стежить" за розподілом пам'яті, рятуючи програміста від цього завдання. Але при цьому можуть виникати проблеми, пов'язані з оптимальністю розподілу пам'яті та швидкодією програм, особливо при приведенні типів або обчисленнях із плаваючою крапкою.

Всі змінні повинні бути оголошені раніше, ніж будуть використовуватися. Оголошення передбачає вказівку типу і список з однієї або декількох змінних цього типу, наприклад:

```
int i, j, k;
char c;
```

При своєму оголошенні змінній може бути привласнене початкове значення, наприклад:

```
int i=0;
```

На відміну від стандарту ANSI, всі змінні повинні бути оголошені на початку блоку, у якому будуть використовуватися. Наприклад:

```
void main ()           // основна функція
{
    int i;              // правильне оголошення
    DDRD=0xFF;          // установка напрямку роботи порту D на вихід
    int j, k;           // неправильне оголошення
    PORTD=0b10101010; // вивід на порт комбінації
}
```

Програмні змінні можуть бути глобальними (доступні для всіх функцій всієї програми) і локальними (доступні тільки всередині тіла функцій, де вони оголошені). Глобальні змінні оголошуються перед всіма функціями. Глобальним змінним при оголошенні автоматично встановлюється значення 0. Локальні змінні автоматично не ініціюються при виклику функцій.

До будь-якої змінної в оголошенні може бути застосований оператор const для вказівки того, що її значення далі не буде змінюватися.

```
const double e = 2.71828182845905;
const char msg[] = "Error number:";
```

Змінні можуть групуватися в масиви. Масив може мати розмір до 8 вимірів. Перший елемент масиву має індекс 0. Елементом глобально оголошених масивів автоматично привласнюється значення 0.

Стосовно до масиву ключове слово `const` вказує на те, що жоден з його елементів не буде змінюватися. Вказівку `const` можна також застосовувати до аргументу-масиву, щоб повідомити, що функція не змінює цей масив:

```
int strlen(const char[]);
```

При спробі змінити змінну, позначену ключовим словом `const`, компілятор повідомляє про помилку: "незмінне значення" (unmodifiable value).

При використанні в програмі асемблера, часто доводиться знати адресу змінної, що розташована в області SRAM пам'яті. При кожній зміні програми, адреса змінної може змінюватися, тому можна жорстко вказати компілятору розташувати змінну по певній адресі.

Глобальні змінні можуть бути оголошені в області SRAM під час написання програми з використанням директиви `#pragma`. Наприклад:

```
#pragma abs_address:0x1000
unsigned LCD_control_register;
#pragma end_abs_address
```

2.2.6. Організація SRAM-пам'яті

Однією з основних переваг Сі-компіляторів у порівнянні з асемблером, є автоматичний розподіл регістрів, SRAM, FLASH та EEPROM пам'яті. При цьому різні типи змінних оголошуються в різних частинах пам'яті контролера. Оскільки контролер має досить обмежені ресурси, необхідно знати, як розподіляються змінні. Використання SRAM пам'яті контролера зазначене на рис. 2.6.

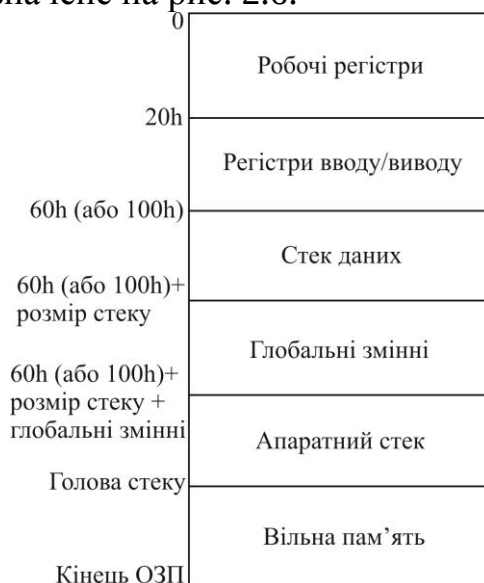


Рис. 2.6.

Working Registers (робочі регістри) – область пам'яті, що містить 32 8-бітних регістра загального призначення.

Компілятор використовує наступні регістри: R0, R1, R15, R22, R23, R24, R25, R26, R27, R28, R29, R30 та R31.

Також, деякі з регістрів від R2 до R14 можуть бути зарезервовані компілятором для глобальних змінних. Інші регістри з тих, що не використовуються у цьому діапазоні, резервуються для глобальних змінних типу `char` та `int`, а також для глобальних вказівників.

Регістри з R16 по R21 резервуються для локальних `char` та `int` змінних.

I/O Registers (регістри вводу/виводу) – діапазон, що містить 64 адреси для функцій периферійних пристроїв, таких як регістри керування напрямком роботи порту, регістри таймерів-лічильників та інші функції вводу-виводу. Можливо вільно використовувати ці регістри в асемблерних вставках.

Data Stack area (стек даних) використовується для динамічного зберігання локальних змінних, передачі параметрів функцій і зберігання регістрів R0, R1, R15, R22, R23, R24, R25, R26, R27, R30, R31 та SREG під час обробки переривань.

Доступ до вказівника стека відбувається за допомогою Y-регістра.

При запуску вказівник стека ініціюється значенням 5Fh (FFh для деяких контролерів) + Розмір стека (Data Stack Size).

При налаштуванні компілятора, на вкладці **Project** → **Options** → **Target** → **Return Stack Size**, необхідно вказати достатній розмір стека, такий, щоб він не перекривав діапазон регістрів вводу-виводу при виконанні програми.

Global Variables (глобальні змінні) – область пам'яті, що використовується для статичного зберігання глобальних змінних. Розмір цієї області обчислюється як сума розмірів всіх оголошених глобальних змінних.

Hardware Stack (апаратний стек) – діапазон, що використовується для зберігання адреси повернення функцій.

Регістр вказівника стека (SP) використовується як вказівник стека та ініціюється при запуску зі значенням `_HEAP_START_ - 1`.

При виконанні програми апаратний стек (Hardware Stack) зміщується донизу, в область глобальних змінних (Global Variables).

HEAP ("купа") – вільна область пам'яті, розташована між апаратним стеком та SRAM. Ця пам'ять використовується для виділення функцій зі стандартної бібліотеки (Standard Library), таких як: `malloc`, `calloc`, `realloc` та `free`.

Компілятор підтримує наступні типи контролерів, для яких існують заголовочні файли (табл. 2.2):

Табл. 2.2.

Сімейство		
Tiny	Classic	Mega
tiny13.h	90s2313.h	mega103.h
tiny22.h	90s2323.h	mega128.h
tiny2313.h	90s2333.h	mega16.h
tiny26.h	90s2343.h	mega161.h
tiny13.h	90s4414.h	mega162.h
tiny22.h	90s4433.h	mega163.h
	90s4434.h	mega168.h
	90s8515.h	mega169.h
	90s8534.h	mega32.h
	90s8535.h	mega323.h
		mega48.h
		mega603.h
		mega64.h
		mega8.h
		mega8515.h
		mega8535.h
		mega88.h
		43usb355.h
		76c711.h
		86rf401.h
		94k.h

Список регістрів та адреси SRAM для ATmega603/103 наведений у табл. 2.3.

Табл. 2.3.

ПОЗНАЧЕННЯ	АДРЕСА SRAM	ФУНКЦІЯ
SREG	0x3F (0x5F)	Регістр статусу (Status REGister)
SPH	0x3E (0x5E)	Старший байт вказівника стека (Stack Pointer High)
SPL	0x3D (0x5D)	Молодший байт вказівника стека (Stack Pointer Low)
XDIV	0x3C (0x5C)	Регістр керування розподілом тактової частоти (XTAL Divide Control Register)
RAMPZ	0x3B (0x5B)	Регістр вибору сторінки Z RAM (RAM Page Z Select Register)
EICR	0x3A (0x5A)	Регістр керування зовнішніми перериваннями (External Interrupt Control Register)
EIMSK	0x39 (0x59)	Регістр масок зовнішніх переривань (External Interrupt MaSK register)
EIFR	0x38 (0x58)	Регістр прапорів зовнішніх переривань (External Interrupt Flag Register)
TIMSK	0x37 (0x57) Counter	Регістр масок переривань по таймерах/лічильниках (Timer/Interrupt MaSK register)
TIFR	0x36 (0x56)	Регістр прапорів переривань по таймерах/лічильниках (Timer/Counter Interrupt Flag register)
MCUCR	0x35 (0x55)	Регістр керування MCU (MCU General Control Register)
MCUSR	0x34 (0x54)	Регістр статусу MCU (MCU Status Register)
TCCR0	0x33 (0x53)	Регістр керування таймером/лічильником 0 (Timer/Counter0)

		Control Register)
TCNT0	0x32 (0x52)	Таймер/лічильник0 (Timer/Counter0 (8-bit))
OCR0	0x31 (0x51)	Регістр порівняння виходу таймера/лічильника 0 (Timer/Counter0 Output Compare Register)
ASSR	0x30 (0x50)	Регістр статусу асинхронного режиму (Asynchronous Mode Status Register)
TCCR1A	0x2F (0x4F)	Керуючий регістр А таймера/лічильника 1 (Timer/Counter1 Control Register A)
TCCR1B	0x2E (0x4E)	Керуючий регістр В таймера/лічильника 1 (Timer/Counter1 Control Register A)
TCNT1H	0x2D (0x4D)	Старший байт таймера/лічильника 1 (Timer/Counter1 High Byte)
TCNT1L	0x2C (0x4C)	Молодший байт таймера/лічильника 1 (Timer/Counter1 Low Byte)
OCR1AH	0x2B (0x4B)	Старший байт регістра А порівняння виходу таймера/лічильника 1 (Timer/Counter1 Output Compare Register A High Byte)
OCR1AL	0x2A (0x4A)	Молодший байт регістра А порівняння виходу таймера/лічильника 1 (Timer/Counter1 Output Compare Register A Low Byte)
OCR1BH	0x29 (0x49)	Старший байт регістра В порівняння виходу таймера/лічильника 1 (Timer/Counter1 Output Compare Register B High Byte)
OCR1BL	0x28 (0x48)	Молодший байт регістра В порівняння виходу таймера/лічильника 1 (Timer/Counter1 Output Compare Register B Low Byte)
ICR1H	0x27 (0x47)	Старший байт регістра захоплення таймера/лічильника 1 (Timer/Counter1 Input Capture Register High Byte)
ICR1L	0x26 (0x46)	Молодший байт регістра захоплення таймера/лічильника 1 (Timer/Counter1 Input Capture Register Low Byte)
TCCR2	0x25 (0x45)	Регістр керування таймером/лічильником 2 (Timer/Counter2 Control Register)
TCNT2	0x24 (0x44)	Таймер/лічильник 2 (Timer/Counter2 (8-bit))
OCR2	0x23 (0x43)	Регістр порівняння виходу таймера/лічильника 2 (Timer/Counter2 Output Compare Register)
WDTCR	0x21 (0x41)	Регістр керування сторожовим таймером (Watchdog Timer Control Register)
EEARH	0x1F (0x3F)	Старший байт регістра адреси EEPROM (EEPROM Address Register High)
EEARL	0x1E (0x3E)	Молодший байт регістра адреси EEPROM (EEPROM Address Register Low)
EEDR	0x1D (0x3D)	Регістр даних EEPROM (EEPROM Data Register)
EECR	0x1C (0x3C)	Регістр керування EEPROM (EEPROM Control Register)
PORTA	0x1B (0x3B)	Регістр даних порту А (Data Register, Port A)
DDRA	0x1A (0x3A)	Регістр напрямку даних порту А (Data Direction Register, Port A)
PINA	0x19 (0x39)	Виводи входів порту А (Input Pins, Port A)
PORTB	0x18 (0x38)	Регістр даних порту В (Data Register, Port B)
DDRB	0x17 (0x37)	Регістр напрямку даних порту В (Data Direction Register, Port B)
PINB	0x16 (0x36)	Виводи входів порту В (Input Pins, Port B)
PORTC	0x15 (0x35)	Регістр даних порту С (Data Register, Port C)
PORTD	0x12 (0x32)	Регістр даних порту D (Data Register, Port D)
DDRD	0x11 (0x31)	Регістр напрямку даних порту D (Data Direction Register, Port D)
PIND	0x10 (0x30)	Виводи входів порту D (Input Pins, Port D)
SPDR	0x0F (0x2F)	Регістр даних SPI I/O (SPI I/O Data Register)
SPSR	0x0E (0x2E)	Регістр статусу SPI (SPI Status Register)

SPCR	0x0D (0x2D)	Регістр керування SPI (SPI Control Register)
UDR	0x0C (0x2C)	Регістр даних UART I/O (UART I/O Data Register)
USR	0x0B (0x2B)	Регістр статусу UART (UART Status Register)
UCR	0x0A (0x2A)	Регістр керування UART (UART Control Register)
UBRR	0x09 (0x29)	Регістр керування швидкістю UART (UART Baud Rate Register)
ACSR	0x08 (0x28)	Регістр статусу й керування аналогового компаратора (Analog Comparator Control and Status Register)
ADMUX	0x07 (0x27)	Регістр вибору мультиплексора ADC (ADC Multi plexer Select Register)
ADCSR	0x06 (0x26)	Регістр статусу й керування ADC (ADC Control and Status Register)
ADCH	0x05 (0x25)	Старший байт регістра даних ADC (ADC Data Register High)
ADCL	0x04 (0x24)	Молодший байт регістра даних ADC (ADC Data Register Low)
PORTE	0x03 (0x23)	Регістр даних порту E (Data Register, Port E)
DDRE	0x02 (0x22)	Регістр напрямку даних порту E (Data Direction Register, Port E)
PINE	0x01 (0x21)	Виводи входів порту E (Input Pins, Port E)
PINF	0x00 (0x20)	Виводи входів порту F (Input Pins, Port F)

2.2.7. Математичні операції

Для обробки інформації, що надходить, мікроконтролер повинен виконувати різні математичні перетворення над даними. У табл. 2.4 наведені основні математичні операнди компілятора. Якщо контролер не має можливості апаратного виконання функції, то функція виконується програмно.

Табл. 2.4.

Операнд	Значення	Результат
+	Додавання	
-	Віднімання	
*	Множення	
/	Ділення	float
%	Ділення по модулю	
>>	Побітовий зсув праворуч	
<<	Побітовий зсув ліворуч	
	Логічне (побітове) АБО	
&	Логічне (побітове) І	

2.2.8. Функції

Мова Сі заснована на понятті функції. Функція – це частина програми, що зберігається в певній області пам'яті і яку можна викликати необмежене число раз.

Будь-яка програма – це сукупність визначень змінних і функцій. Зв'язки між функціями здійснюються через аргументи, що повертають значення, й зовнішні змінні. У вихідному файлі функції можуть розташовуватися в будь-якому порядку; вихідну програму можна розбивати на будь-яке число файлів, але так, щоб жодна з функцій не виявилася розрізаною.

Інструкція `return` реалізує механізм повернення результату від функції, що викликається, в основну програму. За ключовим словом `return` може розташовуватися будь-який вираз або змінна:

return *вираз*;

Якщо буде потрібно, вираз буде приведено до типу даних, що повертає функція. Часто вираз беруть у дужки, але вони не обов'язкові.

Вираз в `return` може бути відсутнім, і тоді взагалі ніяке значення не буде повернуто в основну програму з функції. Керування повертається без результуючого значення також і у тому випадку, коли обчислення досягли "кінця" (тобто останньої закриваючої фігурної дужки функції).

Для використання функцій, необхідно використовувати прототипи. Це оголошення включає інформацію про параметри функцій.

Наприклад:

int alfa(**char** par1, **int** par2, **long** par3);

Дійсне визначення функції повинне бути описане в іншому місці, наприклад:

```
int alfa(char par1, int par2, long par3)
{
/* Тіло функції */
}
```

Параметри у функції передаються через стек (Data Stack). Функції повертають свої значення в регістри R30, R31, R22 та R23 (спочатку молодші розряди –LSB, потім старші – MSB).

Для створення функції, що обробляє переривання, використовується специфікатор `#pragma interrupt_handler`, після якого вказується джерело переривання (вектор). Функції, що обробляють переривання не повертають і не приймають ніяких змінних, тому мають тип **void**. Для кожного контролера існує список доступних переривань, описаних у його заголовному файлі. У табл. 2.5 наведений список векторів переривань для контролера At90S8535.

Табл. 2.5.

Вектор	Адреса	Значення
EXT_INT0	0x02	Зовнішнє int0
EXT_INT1	0x04	Зовнішнє int1
TIM2_COMP	0x12	Збіг при порівнянні таймера-лічильника 2
TIM2_OVF	0x	Переповнення таймера-лічильника 2
TIM1_CAPT	0x14	Захват таймера-лічильника 1
TIM1_COMPA	0x18	Збіг А при порівнянні таймера-лічильника 1
TIM1_COMPB	0x1A	Збіг В при порівнянні таймера-лічильника 1
TIM1_OVF	0x0C	Переповнення таймера-лічильника 1
TIM0_OVF	0x20	Переповнення таймера-лічильника 0

SPI_STC	0x22	Завершення пересилання SPI
UART_RXC	0x24	Прийом символу UART
UART_DRE	0x26	Спустошення регістра даних UART
UART_TXC	0x28	Передача символу UART
ADC_INT	0x2A	Завершення перетворення АЦП
EE_RDY	0x2C	Готовність АЦП
ANA_COMP	0x2E	Спрацьовування аналогового компаратора

Наприклад, обробка зовнішнього переривання int0:

```
#pragma interrupt_handler int0_isr:2
void int0_isr(void)
{
    //external interrupt on INT0
}
```

Часто при написанні програм на Сі виникає необхідність використати асемблерні коди. Це може бути пов'язане з необхідністю оптимізації швидкості роботи або розміром програми, або можливо існує код, що виконує завдання і написаний на асемблері, і його можна просто вставити в текст програми. Асемблерний код може бути вставлений за допомогою команди:

```
#asm("asm_code");
```

Для роботи з периферією, зовнішніми пристроями та для полегшення керування контролером, використовуються заголовні файли. Заголовні файли можна включати в будь-якому порядку. Для включення заголовного файлу, використовується директива компілятора:

```
#include <заголовний файл>
```

Рядок `#include` не повинний бути всередині зовнішнього оголошення або визначення, і повинний зустрітися раніше, ніж що-небудь із заголовного файлу буде використано.

Для компілятора ImageCraft у заголовних файлах описуються тільки прототипи функцій, а сама реалізація функцій перебуває у файлах в каталозі lib, описана на асемблері.

ImageCraft містить наступні заголовні файли для роботи із зовнішніми пристроями:

Табл. 2.6.

Бібліотека	Опис	Заголовний файл
Character Type Functions	Функції для роботи із символами	ctype.h
Standard C Input/Output	Стандартні функції вводу-	stdio.h

Functions	виводу	
Standard Library Functions	Бібліотека стандартних функцій	stdlib.h,
Mathematical Functions	Математичні функції	math.h
String Functions	Функції для роботи з рядками	string.h

Розглянемо деякі функції з тих, що найбільш часто використовуються. Заголовний файл «ctype.h» призначений для роботи з окремими символами, використовується при роботі з рядками, при обробці керуючих байтів пристроїв. Файл «ctype.h» містить наступні функції:

unsigned char isalnum(**char** c) – повертає 1 якщо змінна c алфавітно-цифровий символ;

unsigned char isalpha(**char** c) – повертає 1 якщо змінна c буквенний символ;

unsigned char isascii(**char** c) – повертає 1 якщо змінна c ASCII символ (0..127);

unsigned char iscntrl(**char** c) – повертає 1 якщо змінна c керуючий символ (0..31 або 127);

unsigned char isdigit(**char** c) – повертає 1 якщо змінна c десяткова цифра (від 0 до 9);

unsigned char islower(**char** c) – повертає 1 якщо змінна c буквенний символ нижнього регістра;

unsigned char isprint(**char** c) – повертає 1 якщо змінна c друкований символ (32..127);

unsigned char ispunct(**char** c) – повертає 1 якщо змінна c символ пунктуації (всі, крім керуючих і буквено-цифрових);

unsigned char isspace(**char** c) – повертає 1 якщо змінна c символ пробілу (пробіл, новий рядок, табулятор);

unsigned char isupper(**char** c) – повертає 1 якщо змінна c буквенний символ у верхньому регістрі;

unsigned char isxdigit(**char** c) – повертає 1 якщо змінна c шістнадцяткове число;

char toascii(**char** c) – повертає ASCII еквівалент символу c;

unsigned char toint(**char** c) – інтерпретує c як шістнадцяткову цифру й повертає **unsigned char** від 0 до 15;

char tolower(**char** c) – повертає символ у нижньому регістрі, якщо c у верхньому регістрі, інакше – c;

char toupper(**char** c) – повертає символ у верхньому регістрі, якщо c у нижньому регістрі, інакше – c.

Заголовний файл «stdio.h» містить функції вводу-виводу. Оскільки контролери не мають стандартного потоку вводу-виводу, функції заголовного файлу оптимізовані для роботи з обмеженими ресурсами. Файл «stdio.h» містить наступні функції:

char getchar(**void**) – повертає символ отриманий з UART, використовуючи послідовне опитування;

void putchar(**char** c) – передає символ в UART, використовуючи послідовне опитування;

void puts(**char** *str) – виводить (в UART), використовуючи putchar, символьний рядок str (null terminated), розташований в SRAM пам'яті, включаючи символ нового рядка '\n';

void putsf(**char** *str) – виводить (в UART), використовуючи putchar, символьний рядок str (null terminated), розташований у FLASH пам'яті, включаючи символ нового рядка '\n';

void printf(**char** *fmtstr [arg1, arg2, ...]) – виводить (в UART) форматований текст, використовуючи putchar, так само як він форматований у рядку fmtstr;

signed char sscanf(char *str, char flash *fmtstr [arg1 address, arg2 address, ...]) – ця функція аналогічна scanf, за винятком того, що форматує текст вводиться в рядку str, розташованому в SRAM.

Заголовний файл «math.h» містить опис більш ніж двадцяти математичних функцій. Тут наведені найбільш часто вживані. Кожна з функцій має один або два аргументи типу float і повертає результат також типу float:

sin(x) – повертає значення синуса x, x – у радіанах;

cos(x) – повертає значення косинуса x, x – у радіанах;

atan2(y,x) – повертає значення арктангенса відношення y/x у радіанах;

$\exp(x)$ – експонентна функція e у ступені x ;
 $\log(x)$ – натуральний логарифм x ($x > 0$);
 $\log_{10}(x)$ – десятковий логарифм x ($x > 0$);
 $\text{pow}(x,y)$ – x у ступені y ;
 $\text{sqrt}(x)$ – корінь квадратний x ($x > 0$);
 $\text{fabs}(x)$ – абсолютне значення (модуль) x .

Компілятор ImageCraft містить також інші заголовні файли, які не розглянуті в даному розділі, тому що застосовуються досить рідко для специфічних пристроїв. Всі заголовні файли докладно описані в розділах вбудованої довідки.

2.3. Приклади програм

Приклад 2.2. Написати програму, що виводить на порт D інформацію, яка надходить на порт A мікроконтролера з інверсією.

Текст програми представлений нижче.

```
#include "io8535v.h"
void main()
{
    DDRD=0xff;           // налаштування порту D на вихід
    DDRA=0x00;           // налаштування порту A на вхід
    while(1)
    {
        PORTD=~PINA;     // вивід на порт D інформації з порту A
    }
}
```

У першому рядку програми виконується підключення заголовного файлу, у якому зберігається інформація про адресний простір мікроконтролера і його ресурсів. Робота програми починається з функції main. У цій функції виконується налаштування порту D на вихід шляхом запису в регістр напрямку даних DDRD значення 0xFF. Порт A налаштовується на вхід. У циклі виконується зчитування значення з виводів порту A та передача цього значення в порт D.

Приклад 2.3. Написати програму, що виводить на порт D добуток молодшого й старшого напівбайтів, що подаються на порт B.

```
#include <io8535v.h>
void main(void)           // головна функція програми
{
    int i,j,rez;           // оголошення змінних
    PORTB=0;               // вивід значення 0 в порт B
    DDRB=0;                // налаштування порту B на вхід
    PORTD=0;               // вивід значення 0 в порт D
    DDRD=0xff;             // налаштування порту D на вихід
    while (1)              // вічний цикл
    {
        i=PINB;             // запис значення з порту B у змінну
        j=i;                // записати значення i в змінну j
    }
```

```
i=i&0b00001111;    // виділення 4-х молодших біт
j=j&0b11110000;    // виділення 4-х старших біт
j=j>>4;            // зсув 4-х старших біт у молодші
rez=i*j;            // обчислення добутку
PORTD=rez;          // вивід результату в порт
};
}
```

Приклад 2.4. Програма для роботи з вбудованим АЦП АТ90S8535 без переривань.

```
#include <io8535v.h>
void InitADC(void)    // функція ініціалізації АЦП
{
    ADMUX = 0;        // вибрати вхід номер 0
    ADCSR = 0x0;      // включити АЦП і запустити перше
                      // "порожнє" перетворення
}
int ReadADC(unsigned char channel)
{
    int i;
    ADMUX = channel;   // вибрати номер входу
    ADCSR |= 0x40;     // почати перетворення
    while (!(ADCSR & 0x10)); // перевірка завершення
                          // перетворення
    ADCSR |= 0x10;     // очищення біта "Перетворення
                      // завершене" за допомогою запису в нього 1
    i = ADCL;          // зчитування молодших 8 біт
    i=i+(int)ADCH<<8;  // зчитування старших 2 біт, множення
                      // їх на 256 і додавання з молодшим байтом
    return i;          // повернення результату
}

void main(void)       // головна функція програми
{
    unsigned int temp; // додаткова змінна
    InitADC();          // ініціалізація АЦП
    temp=ReadADC(0);    // виміряти напругу на вході АЦП
}
```

Завдяки вбудованим модулям підтримки широкого спектра периферійних пристроїв, процес створення програм істотно спрощується, що робить доступним застосування AVR мікроконтролерів навіть починаючим розробникам і студентам.

Вбудований автоматичний генератор програмного коду істотно підвищує продуктивність праці розробника та значно полегшує переведення програм з одного типу мікроконтролера на інший, оскільки звільняє програміста від турбот про внутрішню архітектуру процесора та адресації регістрів.

Досвід використання ImageCraft з урахуванням постійного вдосконалювання й технічної підтримки дозволяє поставити його в нечисленний ряд продуктів, ціна яких дійсно відповідає якості.

Питання до розділу 2

1. Які переваги має програмування на мові високого рівня?
2. Яке призначення програмних комплексів ImageCraft, Proteus та AVR Studio?
3. Для чого потрібний дизасемблер?
4. Що таке “змінна” у мові Cі. Дати характеристику основним типам чисельних даних та навести приклади їх використання у програмах.
5. Що таке арифметичний вираз? Навести приклади арифметичних виразів у мові Cі.
6. Для чого використовуються цикли. Навести приклади інструкцій while і for.
7. Дати характеристику поняттю “іменована константа”.
8. Що таке – масив? навести приклади масивів.
9. Дати характеристику поняттю “функція у мові Cі”. Навести приклади.
10. Що означає термін “аргумент функції”? Що означає вираз “виклик по значенню”.
11. Дати характеристику термінам “зовнішня змінна” і “область бачення”.
12. Навести правила формування імен змінних у мові Cі.
13. Навести основні арифметичні оператори, оператори відношення і логічні оператори.
14. Дати характеристику операторам інкремента і декремента та навести приклади їх використання.
15. Дати характеристику побітовим операторам та навести приклади їх використання.
16. Дати характеристику операторам і виразам присвоювання і навести приклади їх використання.
17. Дати характеристику умовним виразам у мові Cі і навести приклади їх використання.
18. Дати характеристику конструкції if-else і навести приклади її використання.
19. Дати характеристику перемикачу switch і навести приклади його використання.
20. Дати характеристику циклам while і for і навести приклади їх використання.
21. Навести приклади використання зовнішніх змінних у програмах на мові Cі.
22. Що таке статичні і регістрові змінні?
23. Розкрити зв'язок між покажчиками і масивами.

3. ПЕРИФЕРІЙНІ ПРИСТРОЇ МІКРОКОНТРОЛЕРА

3.1. Лабораторний стенд для вивчення мікроконтролерів

Існує велика кількість програм для моделювання електронних пристроїв, які відрізняються між собою складністю інтерфейсу, наявністю бібліотек різних компонентів, точністю результатів моделювання. Одним з найпоширеніших пакетів є Electronic WorkBench, який має зручний інтерфейс, велику кількість бібліотек елементів. Однак суттєвим обмеженням є неможливість моделювання схем, в яких є мікропроцесори, програмовані логічні матриці. Теж саме відноситься до системи MicroCap. Також суттєвим недоліком цих програм є неможливість створення друкованої плати для схеми, що моделюється. Система автоматизованого проектування PCAD призначена для розробки схем та друкованих плат, однак немає засобів моделювання.

САПР Proteus дозволяє досить достовірно моделювати та налагоджувати складні пристрої, в яких може міститися декілька мікроконтролерів одночасно, причому різних сімейств. В бібліотеці наявні мікроконтролери AVR, 8051, Microchip PIC10, PIC16, PIC18, Philips ARM7, Motorola MC68HC11. Для моделювання мікроконтролера у вікні його параметрів потрібно вказати шлях до файлу прошивки формату .hex, який може бути створений будь яким компілятором (IAR, CodeVision, ICC, WinAVR), а також вказати тактову частоту. Крім того, є велика кількість різноманітних віртуальних приладів: осцилограф, логічний аналізатор, лічильник імпульсів, вимірник частоти, налагоджувачі шин SPI та I2C, термінал, генератор складних сигналів та інші. В генераторі сигналів можна вказати шлях до WAVE-файлу, інформація з якого буде використана під час моделювання. На виході цього блоку буде напруга, що відповідає даним у файлі. За допомогою цих інструментів можна отримати всі дані про роботу схеми. Дані і результати роботи віртуально створеного пристрою можуть бути збережені на комп'ютері і доступні в цифровому вигляді для подальшого аналізу в пакетах Excel або MATLAB.

Суттєвою перевагою Proteus є можливість створення власних компонентів. Для цього на сайті виробника треба знайти SPICE-модель та додати її в потрібний корпус. SPICE-модель можна взяти з бібліотек інших пакетів симуляції. Є моделі електромеханічних пристроїв: електродвигуни з квадратурним енкодером зворотного зв'язку, сервоприводи – servo – с налаштуваннями, крокові двигуни, безщітковий трьохобмотковий електродвигун. В цих моделях можна встановлювати інерційність якоря з підключеними до нього механізмами, навантаження, опір та індуктивність пристрою, номінальну напругу живлення та швидкість роботи без навантаження. Моделі інтерактивних кнопок, перемикачів, ламп, світлодіодів, рідкокристалічних індикаторів, сенсорних панелей, змінних резисторів дозволяють змінювати їх параметри прямо в процесі роботи пристрою, що моделюється.

Для вивчення мікропроцесорної техніки зручно використовувати спеціалізовані стенди. На рис. 3.1 представлений зовнішній вигляд стенду, який розроблений на кафедрі Теоретичної електротехніки та електронних систем Національного університету кораблебудування ім. адм. Макарова та використовується у навчально-дослідній роботі.

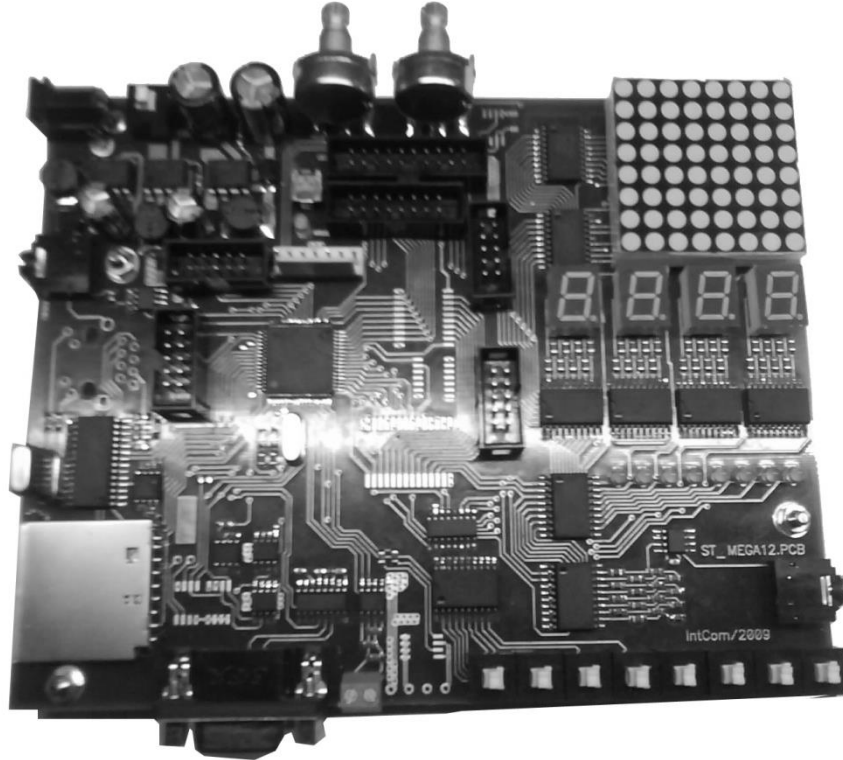


Рис. 3.1.

Лабораторний стенд на базі мікроконтролера ATMEGA128 має в своєму складі:

- мікроконтролер ATMEGA128;
- цифровий датчик температури DS18B20 з інтерфейсом 1-Wire;
- цифровий датчик температури DS1621 з інтерфейсом I2C;
- Flash-пам'ять з інтерфейсом SPI, роз'єм для карт SD-MMC;
- інтерфейс RS-232, RS-485;
- Ethernet-контролер;
- 4 семисегментні індикатори, 8 світлодіодів, 8 кнопок з фіксацією положення;
- інтерфейси підключення графічного та алфавітно-цифрового рідкокристалічних індикаторів;
- інтерфейс клавіатури 4×4;
- 10 – бітний аналого-цифровий перетворювач;
- матричний індикатор 8×8.

В пакеті Proteus створена точна модель такого стенду, за допомогою якої можна перевіряти та налагоджувати програми для мікроконтролерів. На рис. 3.2 представлена схема моделі стенду.

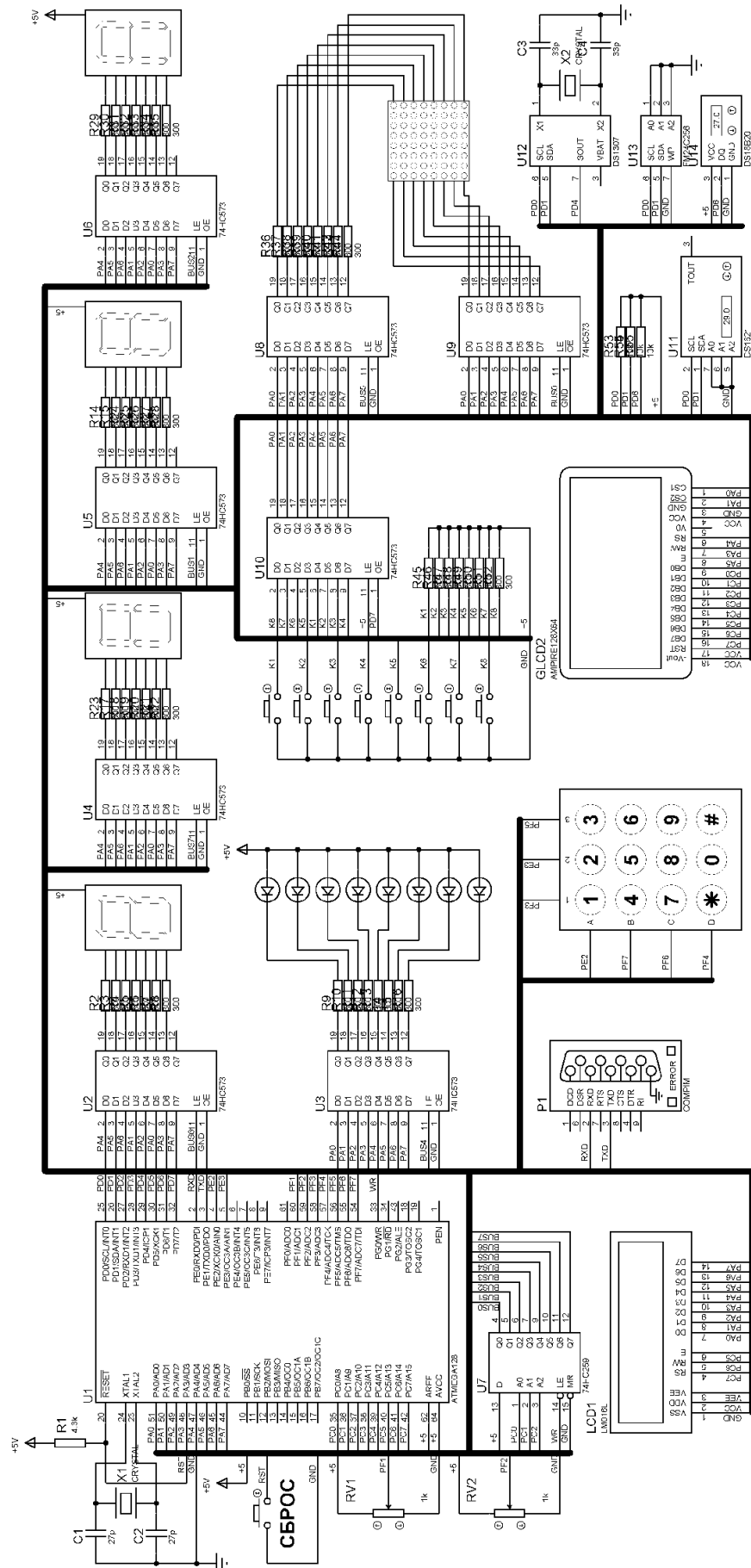


Рис. 3.2.

Основним елементом стенду є мікроконтролер ATMEGA128. Він має велику кількість периферійних пристроїв тому може бути використаний в різних системах. Стенд має засоби для відображення даних: вісім світлодіодів, чотири семисегментні індикатори, які під'єднані до порту А мікроконтролера. Кожен індикатор керується окремим паралельним регістром 74HC573. Дані в регістр записуються при наявності високого рівня на вході LE. Керування регістрами відбувається за допомогою дешифратора двійкового коду в десятковий (мікросхема 74HC259). Входи дешифратора під'єднані до порту С (PC0, PC1, PC2). Для того, щоб керувати визначеним індикатором, потрібно вивести в порт С його адрес, а потім в порт А мікроконтролера вивести дані. В стенді також є матричний індикатор розмірами 8×8, алфавітно-цифровий рідкокристалічний індикатор 2×16 (на базі контролера Hitachi HD44780), а також графічний рідкокристалевий індикатор розмірами 128×64 точок. На ці індикатори може виводитися різноманітна інформація, як текстова, так і графічна, створюватися анімація, меню користувача. Система команд рідкокристалічних індикаторів повністю відповідає їх реальним зразкам. За допомогою інтерактивних кнопок є можливість в процесі моделювання формувати на входах мікроконтролера зміну сигналів. Це зручно при створенні систем, керування якими буде здійснювати оператор. Матричний індикатор дозволяє вивчати принципи створення знаків, динамічної індикації. Модель стенду містить компонент COMPM, який дозволяє віртуальному пристрою підключитися до реального COM-порту комп'ютера. В моделі можна не використовувати перетворювач рівнів RS232 типу MAX232. Також в схемі присутні датчики температури з інтерфейсами 1-Wire (DS18B20) та I2C (DS16C01), мікросхема годинника реального часу з інтерфейсом I2C, карта пам'яті SD/MMC з інтерфейсом SPI. При необхідності можна додати в модель ще кілька датчиків та об'єднати їх в мережу.

Таким чином з'являється можливість створення та відлагодження програмних засобів ще до того, як виготовлений реальний пристрій. Симулятор Proteus дозволяє редагувати та досліджувати програми без особливих матеріальних затрат, додаткових приладів та лабораторних стендів. Найбільша перевага – це додаткові елементи, що дозволяють моделювати роботу мікроконтролера зі світлодіодними індикаторами, рідкокристалічними дисплеями, клавіатурою, кроковим двигуном, датчиками з різними інтерфейсами, програмованою логікою, іншими мікроконтролерами та зовнішніми пристроями. Однак слід зауважити, що віртуальні стенди не переслідують мету повної заміни реальних стендів.

3.2. Паралельні порти вводу-виводу

Всі порти вводу-виводу (ПВВ) AVR-мікроконтролерів працюють за принципом зчитування-модифікація-запис при використанні їх як порти універсального вводу/виводу. Кожен вихідний буфер має симетричну характеристику керування з високим вихідними струмами. Вихідний

драйвер має навантажувальну здатність, що дозволяє безпосередньо керувати світлодіодними індикаторами. До всіх ліній портів може бути підключений резистор, що підтягує до лінії живлення, опір якого не залежить від напруги живлення. На всіх лініях ПБВ установлені захисні діоди, які підключені до шини живлення VCC і загального (GND) провідника, як показано на рис. 3.3.

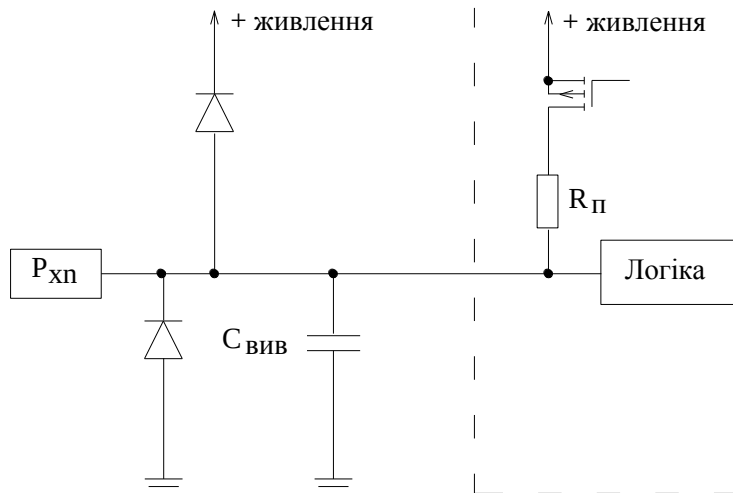


Рис. 3.3.

Для кожного порта вводу/виводу в пам'яті зарезервовано три комірки: одна під реєстр даних – PORTx, друга під реєстр напрямку даних – DDRx і третя під стан входів порту – PINx. Реєстр, що зберігає стан на входах портів, доступний для зчитування, а реєстри даних і напрямку роботи порта мають двонапрямлений доступ.

Режим і стан для кожного виводу визначається значенням відповідних розрядів трьох реєстрів: DDRxn, PORTxn та PINxn. При цьому, символ “x” заміняє найменування ПБВ, а символ “n” заміняє номер розряду ПБВ.

Біти DDRxn реєстра DDRx визначають напрямок лінії вводу-виводу. Якщо DDRxn = 1, то Pxn конфігурується на вивід. Якщо DDRxn=0, то Pxn конфігурується на ввід даних. Якщо PORTxn = 1 при конфігурації лінії порту на введення, то дозволяється підключення підтягуючого резистора. Для вимикання даного резистора необхідно записати в PORTxn лог. 0 або налаштувати лінію порту на вихід. Під час скидання всі лінії портів перебувають у третьому (високоімпедансному) стані, навіть якщо не працює синхронізація. У табл. 3.1 приводиться пояснення дії керуючих сигналів на стан виводу.

Табл. 3.1.

DDxn	PORTxn	PUD (SFIO)	Ввід-вивід	Резистор, що підтягує	Коментар
0	0	X	Введення	Немає	Третій стан (Z-стан)
0	1	0	Введення	Так	Rxn буде джерелом струму при подачі зовнішнього низького рівня
0	1	1	Введення	Немає	Третій стан (Z-стан)
1	0	X	Вивід	Немає	Вивід лог. 0 (струм, що витікає)
1	1	X	Вивід	Немає	Вивід лог. 1 (струм, що витікає)

Незалежно від значення біта напрямку даних DDRxn стан виводу порту може бути опитаний через регістровий біт PINxn.

Регістр даних порта A – PORTA

Розряд	7	6	5	4	3	2	1	0
	PORT A7	PORT A6	PORT A5	PORT A4	PORT A3	PORT A2	PORT A1	PORT A0
Зчитування/запис	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Поч. значення	0	0	0	0	0	0	0	0

Регістр напрямку даних порта A – DDRA

Розряд	7	6	5	4	3	2	1	0
	DDRA 7	DDRA 6	DDRA 5	DDRA 4	DDRA 3	DDRA 2	DDRA 1	DDRA 0
Зчитування /запис	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Поч. значення	0	0	0	0	0	0	0	0

Адреса входів порта A – PINA

Розряд	7	6	5	4	3	2	1	0
	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0
Зчитування /запис	R	R	R	R	R	R	R	R
Поч. значення	-	-	-	-	-	-	-	-

Аналогічні регістри має кожний з портів мікроконтролера. Порти мікроконтролера використовуються для керування зовнішніми пристроями, що підключені до нього.

3.2.1. Приклади програм виведення даних на індикатори різних типів

Напівпровідникові знакосинтезуючі індикатори (НПЗСІ) – низьковольтні пристрої, які зручно сполучаються з джерелами живлення і рівнями струмів мікросхем. Вони мініатюрні і дозволяють створювати пристрої, призначені для відображення інформації різної складності – від точки до текстів та графіків. НПЗСІ мають малий час перемикання –

менше за 50 нс. Промисловістю випускається декілька сотень типів багатоелементних напівпровідникових індикаторів. Вони відрізняються числом, розмірами і конфігурацією світло випромінюючих елементів, кольором, конструктивними рішеннями.

Знакові індикатори в основному призначені для відображення інформації у вигляді цифр та букв. На робочому полі може одночасно відображатися одне або декілька знакомісць. По кількості елементів та їх взаємному розташуванню у межах одного поля розрізняють семисегментні, дев'ятисегментні, п'ятисегментні, 35-сегментні матричні НПЗСІ та ін. На рис. 3.4 представлено умовне розміщення світлодіодів матриці семисегментного індикатора.

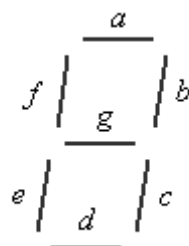


Рис. 3.4.

Для відображення цифр десяткової системи счислення на індикаторі необхідно виконати перетворення двійкового коду в прямий код семисегментних індикаторів.

Приклад 3.1. Зчитати з порту D число, додати до нього константу і результат вивести на порт C. Принципова схема представлена на рис. 3.5.

Текст програми для мікроконтролера на асемблері має вигляд:

```
.device AT90S8535
.include "8535def.inc"
ser r16                ; всі біти регістра R16 встановлюються в 1
out DDRC, r16          ; налаштування порту B на вивід
in r17, PIND           ; запис в регістр R17 значення з порту D, яке задаємо за
                        ; допомогою вимикачів
ldi r18, 23            ; присвоюємо значення константи регістру R18
add r17, r18           ; додаємо значення регістрів R17 та R18
out PORTC, r17         ; результат виводимо на порт D
```

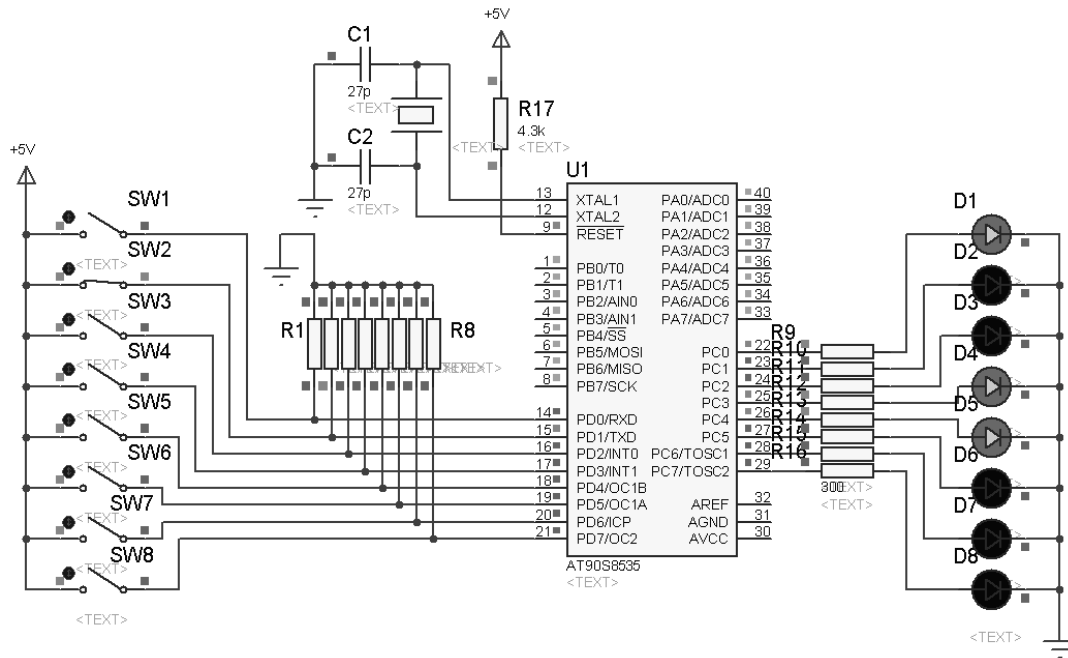


Рис.3.5.

Текст програми на мові C представлений нижче.

```
#include "io8535v.h"
void main()                // початок програми
{
    char data;              // оголошення змінної
    DDRC=0b11111111;       // налаштування порту C на вивід
    while(1)               // вічний цикл
    {
        data=PIND;         // в змінну data записується значення на порту D
        data=data+23;      // до зчитаного значення додається 23
        PORTC=data;        // результат виводиться в порт C
    }
}
```

Перед написанням програми необхідно підключити заголовочні файли. В даному прикладі це файл "io8535v.h", в якому зберігається інформація про ресурси мікроконтролера, адреси портів вводу/виводу, об'єм доступної пам'яті, назви бітів регістрів. Виконання програми починається з функції main(). В тілі функції оголошується додаткова змінна data, в яку потім записується значення, зчитане з порта вводу/виводу мікроконтролера (регістр PIND). Після цього до цієї змінної додається число 23 і результат виводиться в порт C. Особливість програми для мікроконтролера полягає в тому, що вона не має закінчення. Після того, як значення змінної буде виведено в порт C, цикл повторюється. Виконання програми припиниться коли буде відключене живлення схеми.

Приклад 3.2. Розробити програму, яка реалізує роботу мультиплексора 4:1. Данні зчитуються з розрядів $d_0 \div d_3$ порта D, а адресні сигнали з $d_4 \div d_5$ порта D. Дозволяючий вхід d_6 .

Текст програми для мікроконтролера на асемблері має вигляд:

```
.device at90s8535
.include "8535def.inc"
ser r20          ; всі біти регістра R20 встановлюються в одиницю
out DDRC, r20; налаштування порту C на вивід
main:
clr r21          ; очищення регістра R21
set              ; установка біта-акумулятора T регістрі статусу
sbic PIND, 0     ; якщо нульовий біт порту D дорівнює 0, то програма переходить
                  ; через одну команду, що йде за даною командою
bld r21,0        ; завантаження в нульовий біт регістра R21 значення
                  ; біта-акумулятора
sbic PIND, 1     ; якщо перший біт порту D дорівнює 0, то програма переходить
                  ; через одну команду, що йде за даною командою
bld r21, 1       ; завантаження в перший біт регістра R21 значення біта-акумулятора
sbic PIND, 2     ; якщо другий біт порту D дорівнює 0, то програма переходить через
                  ; одну команду, що йде за даною командою
bld r21, 2       ; завантаження в другий біт регістра R21 значення біт-
                  ; акумулятора
sbic PIND, 3     ; якщо третій біт порту D дорівнює 0, то програма переходить через
                  ; одну команду, що йде за даною командою
bld r21, 3       ; завантаження в третій біт регістра R21 значення біта-акумулятора
sbic PIND, 4     ; якщо четвертий біт порту D дорівнює 0, то програма переходить
                  ; через одну команду, що йде за даною командою
bld r21, 4       ; завантаження в четвертий біт регістра R21 значення біт-
                  ; акумулятора
sbic PIND, 5     ; якщо п'ятий біт порту D дорівнює 0, то програма переходить через
                  ; одну команду, що йде за даною командою
bld r21,5        ; завантаження в п'ятий біт регістра R21 значення біта-акумулятора
sbis PIND, 6     ; якщо шостий біт порту D дорівнює 1, то програма пропускає
                  ; одну команду
rjmp m1          ; перехід на мітку m1
out PORTC, r21   ; вивід значення регістра R21 на порт C
rjmp main        ; перехід на мітку main (початок програми)
m1:
bld r21, 6       ; завантаження в шостий біт регістра R21 значення біта-акумулятора
out PORTC, r21   ; вивід значення регістра R21 на порт C
clr r22          ; очищення регістра R22
clr r23          ; очищення регістра R23
bst r21, 4       ; копіювання четвертого біта регістра R21 у біт-акумулятор
bld r22, 0       ; завантаження в нульовий біт регістра R22 значення біта-акумулятора
bst r21, 5       ; копіювання п'ятого біта регістра R21 у біт-акумулятор
bld r22, 1       ; завантаження в перший біт регістра R22 значення біта-акумулятора
cpi r22, 0       ; порівняння значення регістра R22 з нулем
breq n1          ; якщо R22=0, то програма переходить на мітку n1
cpi r22, 1       ; порівняння значення регістра R22 з одиницею
breq n2          ; якщо R22=1, то програма переходить на мітку n2
```

```

cpi r22, 2    ; порівняння значення регістра R22 із двійкою
breq n3       ; якщо R22=2, то програма переходить на мітку n3
bst r21, 3    ; копіювання третього біта регістра R21 у біт- акумулятор
rcall pp      ; виклик підпрограми pp
    n1:
bst r21, 0    ; копіювання нульового біта регістра R21 у біт- акумулятор
rcall pp      ; виклик підпрограми pp
    n2:
bst r21, 1    ; копіювання першого біта регістра R21 у біт- акумулятор
rcall pp      ; виклик підпрограми pp
    n3:
bst r21, 2    ; копіювання другого біта регістра R21 у біт- акумулятор
rcall pp      ; виклик підпрограми pp
    pp:
bld r23, 0    ; завантаження в нульовий біт регістра R23 значення біт- акумулятора
cpi r23, 0    ; порівняння значення регістра R23 з нулем
breq main     ; якщо R23=0, то програма переходить на мітку main
                ; (початок програми)
sbi PORTC, 7  ; установка сьомого біта порту C в одиницю
rjmp main     ; перехід на початок програми
ret          ; повернення з підпрограми в те місце, де дана підпрограма
                ; викликалася

```

Програма на мові Cі.

```

#include "io8535v.h"    // підключення заготовочних файлів
void main()            // головна функція
{
    char data;          // оголошення змінних
    char addr;
    DDRC=0b00000001;   // налаштування виводу 0 порту C на вихід
    while(1)           // вічний цикл
    {
        data=PIND;      // зчитування даних з порту D
        if((PIND&0b01000000)!=0) // якщо на виводі 6 (вхід дозволу) рівень лог.1
        {
            addr=PIND;   // зчитування значення з порту D в змінну addr
            addr=addr&0b00110000; // виділення 2-х адресних бітів
            addr=addr>>4; // зсув значення змінної на 4 розряди праворуч
            if((addr==0) && (PIND&0b00000001)) PORTC=1; // перевірка адреси
            else if((addr==1) && (PIND&0b00000010)) PORTC=1;
            else if((addr==2) && (PIND&0b00000100)) PORTC=1;
            else if((addr==3) && (PIND&0b00001000)) PORTC=1;
            else PORTC=0;
        }
        else PORTC=0;
    }
}

```

В даному прикладі на базі мікроконтролера реалізований мультиплексор. Вихід мультиплексора – молодший розряд порту C. Оскільки цей вивід є виходом, то в функції main після оголошення

додаткових змінних, молодший біт порта C налаштовується на вихід шляхом запису значення 1 в потрібну позицію регістра напрямку роботи порта DDRC. Після цього розпочинається цикл, в якому реалізований алгоритм роботи програми. В змінну data зчитується стан ліній порта D (регістр PIND). Далі за допомогою умовного оператора if перевіряється стан входу дозволу (6-й біт порта). Якщо на цьому вході присутній рівень лог.1, то вираз в дужках умовного оператора приймає значення істина. В змінну addr зчитується стан ліній порта D. За допомогою бітової операції «I» виділяються 4 та 5 біти, які є адресними для розробленого мультиплекса. Для зручності перевірки значення цих біт виконується логічний зсув на 4 розряди праворуч для того, щоб 4 та 5 біти розташувались в розрядах 0 та 1. Далі за допомогою умовних операторів перевіряється рівень сигналу на вході порта D, і якщо номер розряду, на якому присутня лог.1 співпадає з числом, що задане в двійковому коді на адресних входах (біти 4 та 5), то на виході (молодший розряд порта C) встановлюється рівень лог.1.

Приклад 3.3. Розробити програму двійкового лічильника імпульсів з $k=156$. Кількість імпульсів відображати в двійковому коді світлодіодами, які приєднані до розрядів $b_0 \div b_7$ порту C. Принципова схема представлена на рис. 3.6.

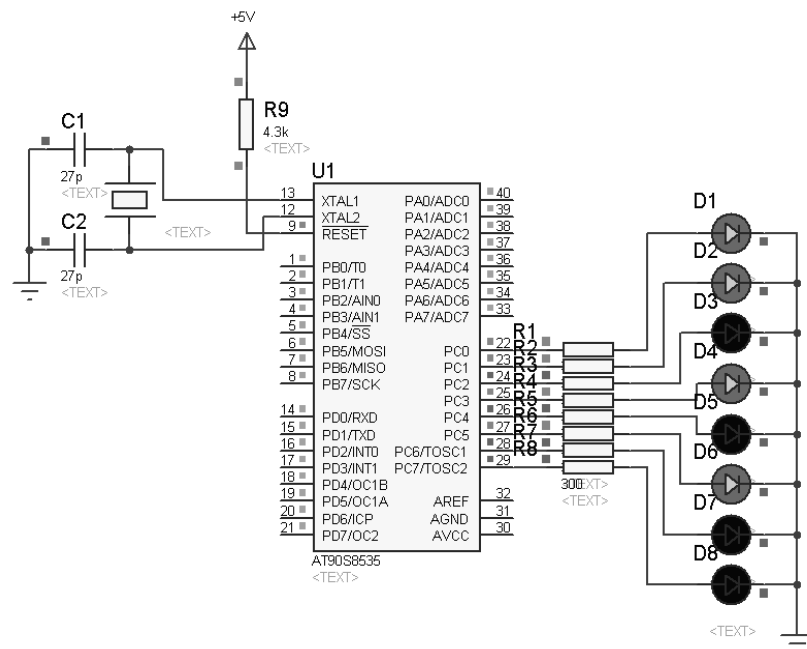


Рис. 3.6.

Текст програми представлений нижче.

```
#include "io8535v.h"
#include "macros.h"
void delay_ms(int ms)           // підпрограма затримки
{
    int c1, c2;                 // оголошення додаткових змінних
    for(c1=0;c1<ms;c1++)        // в циклі виконується «пуста» операція
```

```

    for(c2=0;c2<970;c2++) NOP();
}

void main()                                // головна функція
{
    char cnt=0;                             // оголошення змінної-лічильника
    DDRC=0xFF;                             // налаштування порту C на вихід
    while(1)                               // вічний цикл
    {
        cnt=cnt+1;                         // збільшення значення лічильника на 1
        PORTC=cnt;                         // вивід значення в порт C
        if(cnt==156) cnt=0;                // якщо значення лічильника дорівнює 156,
                                           // відбувається обнуління змінної
        delay_ms(250);                     // виклик підпрограми затримки 250 мс
    }
}

```

В схемі на рис. 3.6 мікроконтролер тактується від кварцового резонатора частотою 8 МГц. При такій частоті нможливо побачити стан ліній порта мікроконтролера. Тому в програмі реалізована функція часової затримки `delay_ms`. В якості параметра в цю функцією передається значення затримки в мілісекундах. Алгоритм реалізації часової затримки досить простий. Оскільки команда в мікроконтролері виконується за 1 такт, і частота кварцового резонатора відома, то можна розрахувати час виконання однієї команди, а також кількість команд, що виконуються за 1 мс. В функції затримки цикли організовані таким чином, що впродовж вказаного часу мікроконтролер виконує «пусту» операцію `NOP()` (No Operation). Відзначимо, що вказаний підхід не є точним, тому не рекомендується використовувати вказаний підхід в програмах, де є важливим формування точних часових інтервалів. Для цього потрібно використовувати таймери-лічильники.

Перед тим, як реалізувати безпосередньо лічильник, порт C налаштовується на вивід шляхом запису значення $0xFF_{16}=255_{10}=11111111_2$ в регістр напрямку порта DDRC. Шістнадцятковий формат представлення чисел використаний для скорочення запису. Не має принципового значення в якому вигляді записуються числа, і це визначається звичками розробника. Однак зауважимо, що при вирішенні окремих задач використання визначеного коду дозволяє спростити розуміння програми та зменшити кількість логічних помилок, які можуть бути допущені при написанні програми. В якості лічильника в програмі використовується змінна `cnt`. В циклі відбувається інкремент цієї змінної. Після цього значення виводиться в порт C, і за допомогою світло діодів, підключених до порту C, можна побачити у двійковому коді значення лічильника. Далі перевіряється умова рівності лічильника значенню 156. Якщо умова виконується. Відбувається онулення змінної. Потім викликається функція часової затримки (250 мс).

Приклад 3.4. Розробити програму для мікроконтролера, що створює ефект «бігучий вогник» на світлодіодах (частота роботи мікроконтролера 8МГц). Схема представлена на рис. 3.7.

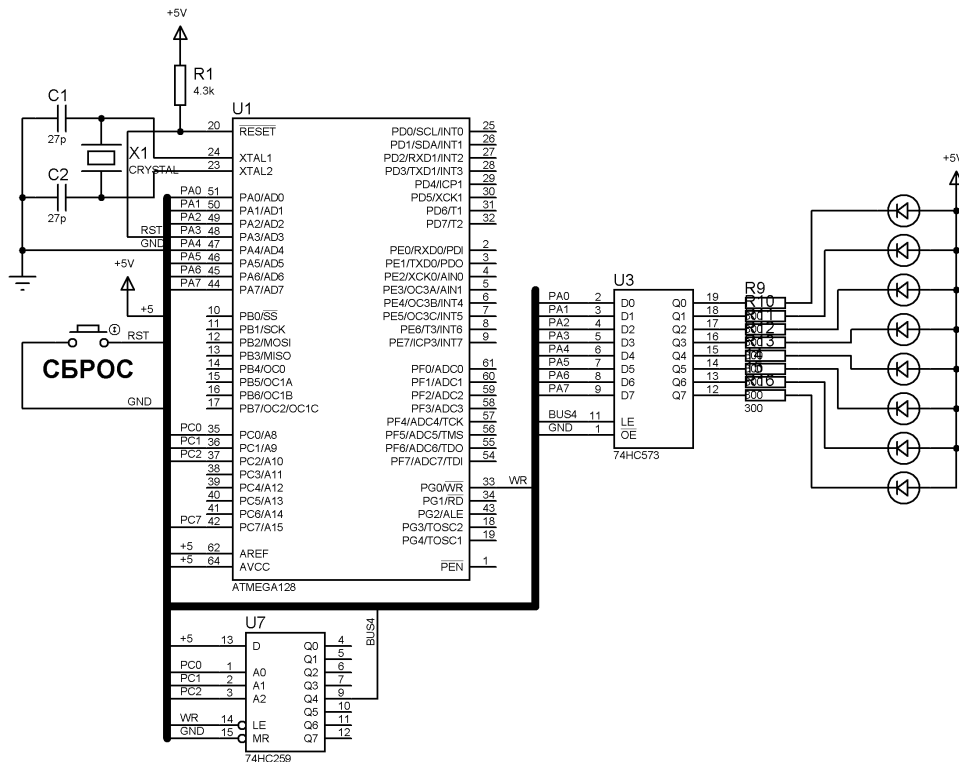


Рис. 3.7.

Текст програми представлений нижче.

```
#include "iom128v.h"
#include "macros.h"
void delay(int ms) // функція формування часової затримки
{
    int c1, c2; // оголошення додаткових змінних
    for(c1=0;c1<ms;c1++)
        for(c2=0;c2<970;c2++) NOP(); // в циклі виконується «холоста» операція
}

void main() // головна функція програми
{
    char c;
    DDRC=0b00000111; // керування дешифратором шини даних
    DDRA=0xFF; // шина даних
    PORTC=4; // шина даних підключається до світлодіодів
    while(1) // програма виконується циклічно
    {
        for(c=0;c<8;c++)
        {
            PORTA=~(1<<c); // в циклі відбувається зсув 1 на визначену кількість розрядів
            delay(300); // формування затримки 300 мс
        }
    }
}
```


Як і в попередньому прикладі, для формування часової затримки використовується функція `delay`. Однак на відміну від попереднього прикладу, в схемі на рис. 3.7 для керування світло діодами використовується окремий зовнішній регістр. Для керування регістром використовується мікросхема дешифратора. Таке рішення дозволить в подальшому підключити до порту А додаткові засоби відображення інформації без суттєвого ускладнення алгоритмів керування ними. Отже, для виводу даних на світлодіодні індикатори необхідно в порт С, який керує дешифратором, записати значення $00000100_2 = 4_{10}$. На виході Q4 дешифратора з'явиться сигнал високого рівня, який дозволить запис даних паралельний регістр. Тому стан виводів порта А можна спостерігати за допомогою світлодіодів. Для керування дешифратором необхідно налаштувати три молодших біти порта С на вихід (регістр DDRC), а для виводу даних на індикатори всі лінії порта А налаштовуються на вихід. В циклі відбувається зсув 1 на визначену кількість розрядів ліворуч, вивід цього значення в порт А та формування часової затримки. При кожній наступній ітерації змінюється значення кількості біт, на яке відбувається зсув 1, і на світло діодах можна спостерігати ефект «бігучий вогник». Оскільки світло діоди підключені таким чином, що на аноди подається живлення, а потенціал катодів визначається виводами регістра, то для того, щоб світлодіод загорівся потрібно підключит катод до загальної лінії, встановивши на ньому низький потенціал. Тому при виводі даних в порт А також виконується їх інверсія.

Приклад 3.5. Створити на світлодіодах ефект вогників, що зіштовхуються (схема представлена на рис. 3.7).

Текст програми представлений нижче.

```
#include "iom128v.h"
#include "macros.h"

void delay(int ms)           // функція формування часової затримки
{
    int c1, c2;               // оголошення додаткових змінних
    for(c1=0;c1<ms;c1++)
        for(c2=0;c2<970;c2++) NOP(); // в циклі виконується "пуста" операція
}

void main()
{
    char i;                   // оголошення додаткової змінної
    char arr[6]={0x81, 0x42, 0x24, 0x18, 0x24, 0x42}; // в масиві зберігаються
                                                // значення, що виводяться в порт
    DDRC=0b00000111;         // керування дешифратором шини даних
    DDRA=0xFF;                // шина даних
    PORTC=4;                  // шина даних підключається до світлодіодів
    while(1)                  // програма виконується циклічно
    {
        for(i=0;i<6;i++)
```

```
{  
  PORTA=~arr[i];           // в циклі виводиться в порт по черзі елементи масиву  
  delay(300);              // формування затримки 300 мс  
}  
}  
}
```

Для реалізації цього світлового ефекту зручно використовувати масив чисел, в якому зберігаються значення для виводу в порт. Послідовний вивід даних в порт призведе до появи вказаного ефекту. Для скорочення запису дані в масиві представлені в шістнадцятковому форматі.

Приклад 3.6. Розробити програму - лічильник від 0 до 9999, що виводить значення на чотири семисегментні індикатори. При досягненні максимального значення лічильник скидається.

Схема представлена на рис. 3.8. Для керування семисегментними індикаторами використовуються додаткові зовнішні паралельні регістри. Таке рішення дозволяє підключити всі індикатори на одну шину, тим самим зменшивши кількість задіяних виводів мікроконтролера, а керування позицією виводу реалізувати за допомогою дешифратора. Таким чином вдалося зменшити кількість виводів з 35 до 10, що використовуються на схемі індикації.

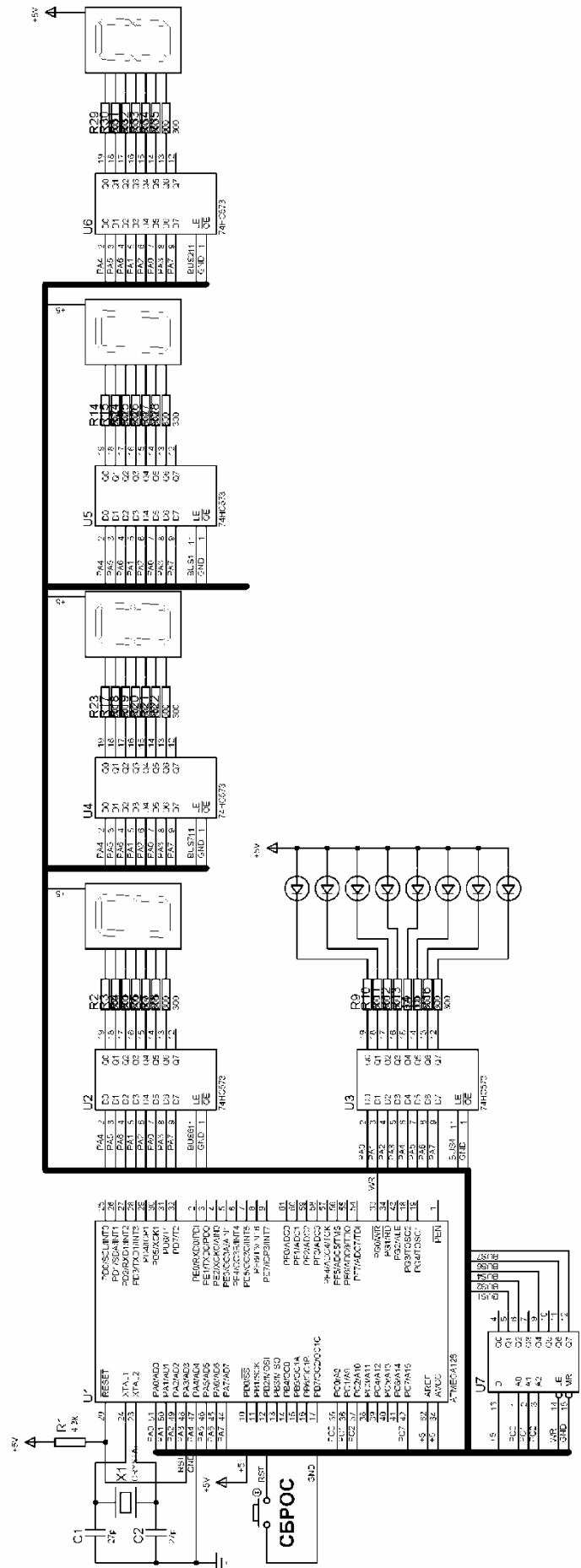


Рис. 3.8.

Текст програми має вигляд:

```
#include "iom128v.h"
#include "macros.h"
void delay(int ms)                // функція часової затримки
{
    int c1, c2;
    for(c1=0; c1<ms; c1++)
        for(c2=0; c2<970; c2++) NOP();
}

void main()                       // головна функція програми
{
    char seg[10]={0b01110111, 0b01100000, 0b00111110, 0b01111010, 0b01101001,
        0b01011011, 0b01011111, 0b01110000, 0b01111111, 0b01111011};
    int i;
    DDRC=0b00000111;             // керування дешифратором шини даних
    DDRA=0xFF;                   // шина даних – порт налаштований на вихід
    while(1)                     // вічний цикл
    {
        for(i=0; i<10000; i++)
        {
            PORTC=6;              // включення індикатора H6
            PORTA=~seg[i/1000];   // вивід в порт значення тисяч
            PORTC=7;              // включення індикатора H7
            PORTA=~seg[(i%1000)/100]; // вивід в порт значення сотень
            PORTC=1;              // включення індикатора H8
            PORTA=~seg[((i%1000)%100)/10]; // вивід в порт значення десятків
            PORTC=2;              // включення індикатора H9
            PORTA=~seg[((i%1000)%100)%10]; // вивід в порт значення одиниць
            delay(100);           // формування часової затримки
        }
    }
}
```

Для виводу даних на семисегментні індикатори потрібно реалізувати дешифратор двійкового коду в код семисегментних індикаторів. Це досить легко зробити програмним шляхом у вигляді масиву, елементами якого є коди семисегментних індикаторів, а індекс елемента відповідає цифрі, що буде відображатися на індикаторі. Для виділення тисяч, значення змінної ділиться на 1000. Залишок від ділення не враховується, тому залишається лише ціла частина. Аналогічні дії виконуються для виділення сотень, десятків та одиниць. Синхронно з виводом даних в порт змінється код, що подається на входи дешифратора. Таким чином вибирається потрібний регістр для виводу даних. При досягненні значення 9999, значення лічильника онулюється. Оскільки семисегментні індикатори підключені по схемі з загальним анодом, то при виводі даних виконується інверсія елемента масиву.

При необхідності індикації великої кількості станів на обмеженій площі лицьової панелі одного приладу, або при створенні інформаційних таблиць, загальний об'єм устаткування для управління індикаторами може бути значно зменшений за рахунок використання їх матричного включення.

При необхідності одночасної індикації декількох одиничних індикаторів у складі матриці або відображення букв та цифрової інформації, використовується загальний принцип управління матрицею – мультиплексний, який дозволяє включати будь-яку комбінацію елементів матриці.

Матричність структури індикаторів дозволяє здійснити вивід символу лише в режимі стробування, при цьому кожен символ має бути утворений з п'яти або семи під наборів, в залежності від способу адресації.

Приклад 3.7. На матричний індикатор вивести букву М.

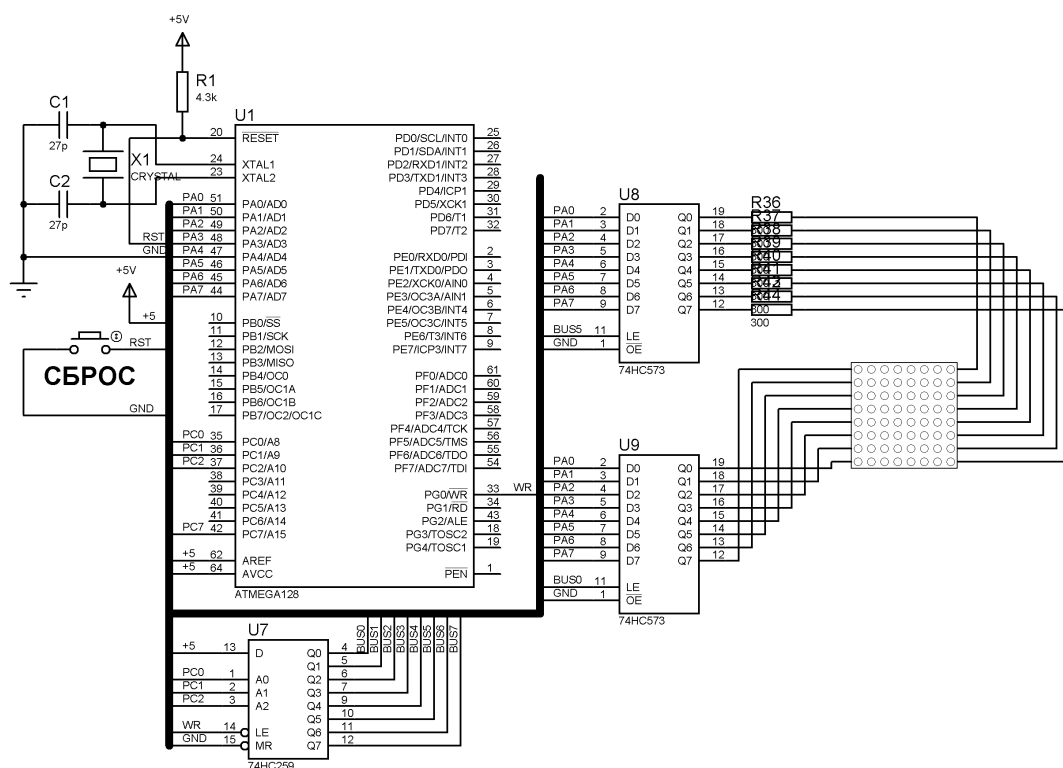


Рис. 3.9.

Керування стовбцями та строками матриці вионується за допомогою регістрів. При подачі лог.1 на вхід LE регістрів, вони працюють в режимі повторювача, виконують роль підсилювального елементу. Тому немає необхідності формувати додатковий синхросигнал для запису даних в ці регістри.

Текст програми для мікроконтролера має вигляд:

```
#include "iom128v.h"
#include "macros.h"
```

```
void delay(int ms) // функція формування часової затримки
{
```

```
int c1, c2; // оголошення додаткових змінних
for(c1=0;c1<ms;c1++)
  for(c2=0; c2<970; c2++) NOP(); // в циклі виконується "пуста" операція
}

void main() // головна функція програми
{
  char i;
  char sym[8]={0b11111111, 0b00000010, 0b00000100, 0b00001000, 0b00001000,
  0b000000100, 0b000000010, 0b11111111};
  DDRC=0b00000111; // керування дешифратором шини даних
  DDRA=0xFF; // шина даних
  while(1)
  {
    for(i=0; i<8; i++)
    {
      PORTC=5;
      PORTA=sym[i];
      PORTC=0;
      PORTA=~(1<<i);
      delay(2);
    }
  }
}
```

В масиві `sym` зберігаються коди для формування символу «М». Значення кожного елемента масиву виводиться в порт А, виводи якого керують станом ліній матричного індикатора. При цьому дані записуються в регістр, який керується лінією BUS5 дешифратора. Синхронно з установкою сигналів на лініях матричного індикатора відбувається керування відповідним стовпчиком. Послідовний вивід даних в порт з високою частотою зі зміною номера стовпчика формує на індикаторі заданий символ.

3.2.2. Введення даних в мікроконтролер

Порти мікроконтролера – це пристрої вводу/виводу, що дозволяють мікроконтролеру передавати або приймати дані. Стандартний порт мікроконтролера AVR має вісім розрядів даних, які можуть передаватися або прийматися паралельно. Кожному розряду (або біту) відповідає вивід (ніжка) мікроконтролера. Ніжки мікроконтролера також називають пінами. Для позначення портів використовуються латинські букви А, В, С і т.д. Кількість портів вводу/виводу варіюється залежно від моделі мікроконтролера.

Будь-який порт мікроконтролера можна налаштувати як вхід, або як вихід. Для того, щоб це зробити, потрібно записати у відповідний регістр `DDRx` необхідне значення. Крім того, як вхід або вихід можна налаштувати окремо будь-який вивід (пін) порту.

DDRx – реєстр напрямку передачі даних. Цей реєстр визначає напрямок роботи порту – вхід або вихід. Якщо деякий розряд реєстра DDRx містить логічну одиницю, то відповідний вивід порту налаштований як вихід, у протилежному випадку – як вхід. Літера “x” у цьому випадку позначає ім'я порту, з яким відбувається робота. Таким чином, для порту A це буде реєстр DDRA, для порту B – реєстр DDRB і т.д. Наприклад, у реєстрі DDRD можна записати число:

DDRD = 0xb3;

0xb3 – шістнадцяткове представлення числа 179. У двійковому вигляді воно буде представлене як 10110011. Частина виводів порту D буде налаштована як виходи, а частина – як входи:

PD0 – 1 (вихід)

PD1 – 1 (вихід)

PD2 – 0 (вхід)

PD3 – 0 (вхід)

PD4 – 1 (вихід)

PD5 – 1 (вихід)

PD6 – 0 (вхід)

PD7 – 1 (вихід)

У реєстрі PINx міститься інформація про реальний поточний логічний рівень на виводах порту, не залежно від налаштувань порту.

Існує дві межі: межа гарантованого нуля і межа гарантованої одиниці – пороги за якими ми можемо однозначно чітко визначити поточний логічний рівень. При живленні мікроконтролера 5 В це 1,4 В та 1,8 В відповідно. При зниженні напруги від максимуму до мінімуму, біт у реєстрі PIN перемкнеться з 1 на 0 тільки при зниженні напруги нижче 1,4 В, а коли напруга наростає від мінімуму до максимуму, перемикавання біта з 0 на 1 буде тільки по досягненні напруги в 1,8 В, тобто є гістерезис перемикавання з 0 на 1, що виключає хаотичні перемикавання під дією перешкод і завад, а також виключає помилкове зчитування логічного рівня між порогамі перемикавання.

Майже кожна ніжка мікроконтролера має додаткові функції. На зображенні мікроконтролера в технічній документації вони підписані в дужках. Це можуть бути виводи приймача, різні послідовні інтерфейси, аналогові входи, виходи ШІМ – генераторів. За замовчуванням всі ці функції відключені, а вивід керується винятково парою DDR та PORT, але якщо включити яку-небудь додаткову функцію, то керування може повністю або частково перейти під контроль периферійного пристрою.

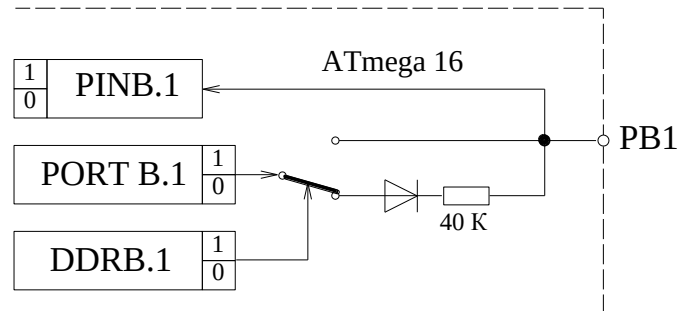


Рис. 3.10.

На рис. 3.11 представлена принципова схема для демонстрації роботи програм.

Приклад 3.8. Написати програму, що відображає на світлодіодних індикаторах стан кнопок.

```
#include "iom128v.h"
#include "macros.h"

void delay(int ms)          // функція формування часової затримки
{
    int c1, c2;              // оголошення додаткових змінних
    for(c1=0;c1<ms;c1++)
        for(c2=0;c2<970;c2++) NOP();// в циклі виконується "пуста" операція
}

void main()
{
    char t;
    DDRC=0b00000111;        // керування дешифратором шини даних
    DDRA=0xFF;               // порт А налаштовується на вихід
    DDRD=0x80;               // лінія керування регістром тактових кнопок
    while(1)
    {
        DDRA=0;              // порт А налаштований на вхід для зчитування даних
        PORTD=0x80;          // уст. лог. 1 на виводі PD7 для керування регістром
        t=PINA;              // зчитування значення з шини даних
        PORTD=0;             // переведення регістра клавіатури у стан Z
        DDRA=0xFF;           // налаштування порту А на вихід
        PORTC=4;             // включення регістра світлодіодів
        PORTA=t;             // вивід значення в порт
        delay(100);          // пауза 100 мс
    }
}
```


Для керування дешифратором три виводи порту C налаштовуються на вихід. Світлодіоди підключені до порту A мікроконтролера, тому саме цей порт налаштовується на вихід шляхом запису значення 1 в усі розряди (DDRA=0xFF). Оскільки світлодіодні та семисегментні індикатори керуються загальною шиною, до якої через регістр підключені тактові кнопки, тому для зчитування стану кнопок порт A налаштовується на вхід шляхом запису значення 0 в усі розряди регістра напрямку порту DDRA. Після цього вмикається регістр, до якого підключені кнопки. На шині даних встановиться код, що задається кнопками. В локальну змінну t зчитується значення з вхідного регістра порту A – PINA. Після цього регістр до якого підключені тактові кнопки вимикається (PORTD), порт A налаштовується на вихід, і значення зі змінної t виводиться на шину даних, до якої через регістр підключаються світлодіоди (PORTC=4). Далі описана послідовність дій повторюється циклічно з періодом 100 мс.

Приклад 3.9. Написати програму, що виводить на семисегментних індикаторах десяткове значення коду, заданого кнопками у двійковому вигляді.

```
#include "iom128v.h"
#include "macros.h"

void delay(int ms)                // функція формування часової затримки
{
    int c1, c2;                   // оголошення додаткових змінних
    for(c1=0;c1<ms;c1++)
        for(c2=0;c2<970;c2++) NOP(); // в циклі виконується "пуста" операція
}

char scan_buttons()               // функція зчитування стану кнопок
{
    char t, d=255;
    DDRA=0;                       // порт A налаштований на вхід для зчитування даних с кнопок
    PORTD=0x80;                   // уст. лог. 1 на виводі PD7 для керування регістром клавіатури
    t=PINA;                       // зчитування значення з шини даних
    PORTD=0;                       // переведення регістра клавіатури у високоімпедансний стан
    DDRA=0xFF;                    // налаштування порту A на вихід
    // перетворення даних, отриманих з кнопок для зручності використання
    if((t&0x10)==0) d&=~0x80;
    if((t&0x20)==0) d&=~0x40;
    if((t&0x40)==0) d&=~0x20;
    if((t&0x80)==0) d&=~0x10;
    if((t&0x01)==0) d&=~0x01;
    if((t&0x02)==0) d&=~0x02;
    if((t&0x04)==0) d&=~0x04;
    if((t&0x08)==0) d&=~0x08;
    return ~d;                    // значення повертається із функції з інверсією
}

void main()
{
    char i;
    char seg[10]={0b01110111, 0b01100000, 0b00111110, 0b01111010, 0b01101001,
```

```

0b01011011, 0b01011111, 0b01110000, 0b01111111, 0b01111011};
DDRC=0b00000111;           // керування дешифратором шини даних
DDRA=0xFF;                  // порт А налаштований на вихід
PORTA=0;                    // вивід значення 0 на шину
DDRD=0x80;                  // лінія керування регістром тактових кнопок
PORTC=4;                    // включення регістра світлодіодів
while(1)
{
    i=scan_buttons();        // отримання даних з кнопок
    PORTC=6;                  // включення індикатора Н6
    PORTA=~seg[i/1000];       // вивід в порт значення тисяч
    PORTC=7;                  // включення індикатора Н7
    PORTA=~seg[(i%1000)/100]; // вивід в порт значення сотень
    PORTC=1;                  // включення індикатора Н8
    PORTA=~seg[((i%1000)%100)/10]; // вивід в порт значення десятків
    PORTC=2;                  // включення індикатора Н9
    PORTA=~seg[((i%1000)%100)%10]; // вивід в порт значення одиниць
    delay(100);               // пауза 100 мс
}
}

```

Дослідивши попередній приклад можна побачити, що розташування тактових кнопок не співпадає з розташуванням світлодіодів. Тому фрагмент коду для зчитування стану кнопок оформлений у вигляді функції `scan_buttons()`. За допомогою умовного оператора `if` та бітових операцій виконується перерозташування бітів таким чином, щоб розташування їх було з молодшого розряду праворуч до старшого розряду ліворуч. В масиві `seg` зберігається коди цифр від 0 до 9 для семисегментного індикатора. При виводі значень на індикатори спочатку вибирається відповідний регістр, який керує індикатором, і значення коду встановлюється на шині шляхом запису даних в регістр порта А.

Приклад 3.10. Написати програму керування за допомогою кнопок положенням точки, що світиться, на матричному індикаторі.

```

#include "iom128v.h"
#include "macros.h"

void delay(int ms)           // функція формування часової затримки
{
    int c1, c2;               // оголошення додаткових змінних
    for(c1=0; c1<ms; c1++)
        for(c2=0; c2<970; c2++) NOP(); // в циклі виконується "пуста" операція
}

char scan_buttons() // функція зчитування стану з кнопок
{
    char t, d=255;
    DDRA=0;                  // порт А налаштований на вхід для зчитування даних з кнопок
    PORTD=0x80;              // уст. лог. 1 на виводі PD7 для керування регістром клавіатури
    t=PINA;                  // зчитування значення з шини даних
    PORTD=0;                  // переведення регістра у високоімпедансний стан
}

```

```
DDRA=0xFF;          // налаштування порту А на вихід
// перетворення даних, отриманих з кнопок для зручності використання
if((t&0x10)==0) d&=~0x80;
if((t&0x20)==0) d&=~0x40;
if((t&0x40)==0) d&=~0x20;
if((t&0x80)==0) d&=~0x10;
if((t&0x01)==0) d&=~0x01;
if((t&0x02)==0) d&=~0x02;
if((t&0x04)==0) d&=~0x04;
if((t&0x08)==0) d&=~0x08;
return ~d;           // значення повертається із функції з інверсією
}
void main()
{
    char t;
    char c=0, r=0;
    DDRC=0b00000111; // керування дешифратором шини даних
    DDRA=0xFF;        // порт А налаштовується на вихід
    DDRD=0x80;        // лінія керування регістром тактових кнопок
    PORTC=4;          // включення регістра світлодіодів
    while(1)
    {
        t=scan_buttons(); // отримання даних с кнопок
        if(((t&0x01)!=0)&& (c<7)) c++;
        if(((t&0x02)!=0)&& (c>0)) c--;
        if(((t&0x80)!=0)&& (r>0)) r--;
        if(((t&0x40)!=0)&& (r<7)) r++;
        PORTC=5;
        PORTA=(1<<r);
        PORTC=0;
        PORTA=~(1<<c);
        delay(500);
    }
}
```

В основному циклі програми для визначення положення точки на матричному індикаторі, що світиться, використовуються дві змінні – *c* та *r*. Оскільки розмір матричного індикатора 8×8 , то необхідно обмежити значення, які можуть приймати вказані змінні. Зрозуміло, що значення повинно лежати в межах від 0 до 7. За допомогою умовного оператора перевіряється чи натиснута кнопка і виконується інкремент або декремент змінних. Після цього вмикається регістр строчок матричного індикатора шляхом запису в порт С значення 5, а потім вмикається регістр, що керує стовбцями матричного індикатора. Таким чином, у заданій позиції загоряється сегмент.

Приклад 3.11. При натисканні кнопок на клавіатурі, номер натиснутої кнопки виводиться на семисегментному індикаторі. У поточній позиції виводу горить десяткова крапка. Використовувати принцип динамічної індикації при виводі значень на індикатор із загальним катодом. На рис. 3.12 представлена принципова схема пристрою.

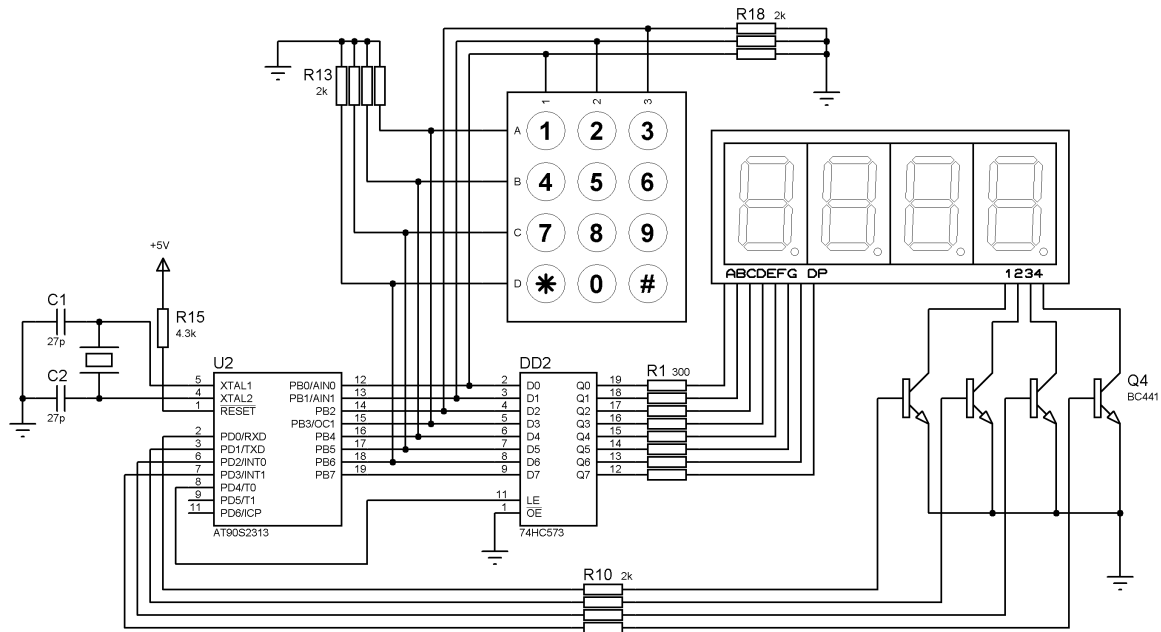


Рис. 3.12.

Текст програми для мікроконтролера представлений нижче.

```
#include <io2313v.h>           // підключення заголовочних файлів
#include <macros.h>
char prev[4]={0,0,0,0};       // оголошення масиву з 4-х елементів
char index;                   // оголошення додаткової змінної
void delay(int ms)            // функція затримки
{
    int c1, c2;               // оголошення додаткових змінних
    for(c1=0; c1<ms; c1++)
        for(c2=0; c2<970; c2++) NOP(); // у циклі виконується «порожня» операція
}

char scan_keyboard()          // функція сканування клавіатури
{
    char col=0, row=0;        // додаткові змінні
    char key=13;              // оголошення змінної з ініціалізацією
    PORTD=0;                  // вивід значення 0 у порт D
    PORTB=0;                  // вивід значення 0 у порт B
    DDRB=0b00000111;         // налаштування виводів PB0-PB2 на вихід
    PORTB=0b00000111;        // установка на виводах PB0-PB2 сигнали лог.1.
    delay(2);                 // затримка 2 мс
    row=PINB;                 // зчитування значення з порту B
    row=row>>3;               // зсув значення на 3 розряди праворуч
    row=row&0x0F;             // виділення 4-х молодших біт
    DDRB=0b01111000;         // налаштування виводів PB6-PB3 на вихід
    PORTB=0b01111000;        // установка на виводах PB6-PB3 сигналу лог.1.
    delay(2);                 // затримка 2 мс
    col=PINB;                 // зчитування значення з порту B
    col=col&0x07;             // виділення 3-х молодших біт
    if(row==1) key=1;         // визначення ряду, у якому перебуває натиснута кнопка
    else if(row==2) key=4;
```

```
else if(row==4) key=7;
else if(row==8) key=10;
if(col==2) key=key+1;      // визначення колонки, у якій перебуває натиснута кнопка
else if(col==4) key=key+2;
PORTB=0;                  // вивід значення 0 у порт В
DDRB=0xFF;               // налаштування порту В на вихід
PORTD|=0x10;             // установка сигналу лог.1 на виводі PD4
return key-1;            // повернення номера натиснутої кнопки
}

void main()              // головна функція програми
{
    char c=0;             // оголошення додаткових змінних
    char button;
    // оголошення масиву з кодами натиснутих клавіш (0-9, A, B).
    char seg[12]={0b00000110, 0b01011011, 0b01001111, 0b01100110,
    0b01101101, 0b01111101, 0b00000111, 0b01111111, 0b01101111, 0b01110111,
    0b00111111, 0b01111100};
    index=0;              // обнуління змінної
    DDRB=0b11111111;      // налаштування порту В на вихід
    DDRD=0b00011111;      // налаштування 5 молодших виводів порту D на
    вихід
    PORTD=0;              // вивід значення 0 у порт D
    while(1)              // вічний цикл
    {
        if(c>50)          // якщо значення змінної з більше 50
        {
            c=0;           // онулення змінної
            button=scan_keyboard(); // сканування клавіатури
            if(button<12)   // якщо натиснута яка-небудь клавіша
            {
                prev[index]=button; // запис у масив номера натиснутої клавіші
                if(index<3) index++; // якщо значення змінної менше 3 – інкремент
                else index=0;       // інакше обнуління змінної
            }
        }
        else c++;          // збільшення значення змінної на 1
        PORTD|=1;          // установка на виході PD0 значення лог.1.
        PORTB=seg[prev[0]]; // вивід у порт В елемента масиву
        if(index==0) PORTB|=0x80; // якщо значення індексу дорівнює 0, PB7=1
        else PORTB&=~0x80;  // інакше PB7=0
        delay(5);          // затримка 5 мс
        PORTD&=~1;         // установка на виводі PD0 низького рівня сигналу
        PORTD|=2;          // установка на виводі PD1 високого рівня сигналу
        PORTB=seg[prev[1]]; // вивід у порт В елемента масиву
        if(index==1) PORTB|=0x80; // якщо значення індексу дорівнює 1, PB7=1
        else PORTB&=~0x80;  // інакше PB7=0
        delay(5);          // затримка 5 мс
        PORTD&=~2;         // установка на виводі PD1 низького рівня сигналу
        PORTD|=4;          // установка на виводі PD2 високого рівня сигналу
        PORTB=seg[prev[2]]; // вивід у порт В елемента масиву
        if(index==2) PORTB|=0x80; // якщо значення індексу дорівнює 1, PB7=1
```

```

else PORTB&=~0x80;      // інакше PB7=0
delay(5);               // затримка 5 мс
PORTD&=~4;              // установка на виводі PD2 низького рівня сигналу
PORTD|=8;               // установка на виводі PD3 високого рівня сигналу
PORTB=seg[prev[3]];     // вивід у порт В елемента масиву
if(index==3) PORTB|=0x80; // якщо значення індексу дорівнює 1, PB7=1
else PORTB&=~0x80;      // інакше PB7=0
delay(5);               // затримка 5 мс
PORTD&=~8;              // установка на виводі PD3 низького рівня сигналу
}
}

```

Опитування клавіатури (функція `scan_keyboard()`) відбувається в проміжку часу між змінами розрядів у процесі динамічної індикації. У масив `prev` записуються коди натиснутих клавіш. При записі чергового значення змінна `index` збільшується на 1. Таким чином, при кожному новому натисканні на клавішу елементи масиву `prev` оновлюються і виводяться на чотирьохрозрядний індикатор. У програмі відслідковується значення змінної `index` і встановлюється або відключається старший біт порту В, який керує станом десяткової крапки в кожному розряді.

3.2.3. Керування алфавітно-цифровими РКІ

Контролер HD44780 фірми Hitachi фактично є промисловим стандартом і часто використовується в алфавітно-цифрових РКМ. Багато фірм (Optrex Corporation, Powertip, Seiko Instruments, Batron та ін.) випускають алфавітно-цифрові рідкокристалічні індикатори з вбудованими контролерами HD44780. Конструктивні модифікації алфавітно-цифрових РКІ представлені на рис. 3.13.



Рис. 3.13.

Рідкокристалічний модуль, що розглядається, за допомогою стандартного 14-контактного роз'єму (табл. 3.2) обмінюється інформацією з керуючим мікроконтролером (у нашому випадку з AVR). AVR-

мікроконтролер посилає в РКІ команди (табл. 3.3), керуючи режимами його роботи, а також ASCII-коди виведених символів. В свою чергу, РКІ може посилати мікроконтролеру по його запиті інформацію про свій стан і дані зі своїх внутрішніх блоків пам'яті.

Табл.3.2.

N	Назва виводу	Опис
1	V _{SS}	(-) Живлення. 0 В.
2	V _{DD}	(+) Живлення. +5 В.
3	V ₀	Напруга зсуву, що управляє контрастністю
4	RS	Вхід. Високий рівень - дані; низький - команди
5	R/W	Вхід. Високий - зчитування, низький-запис
6	E	Вхід. Строб, що супроводжує сигнали на шині "команди/дані"
7	DB ₀	Шина "команди/дані"
8	DB ₁	
9	DB ₂	
10	DB ₃	
11	DB ₄	
12	DB ₅	
13	DB ₆	
14	DB ₇	

Три виводи 14-контактного роз'єму призначені для подачі напруги живлення і напруги зсуву, що управляє контрастністю дисплею. З 11 виводів, що залишилися, 8 (DB₀ – DB₇) використовуються для організації мультиплексованої шини "команди/дані", і на 3 виводи (RS, R/W, E) мікроконтролер виставляє керуючі сигнали. За допомогою сигналу на лінії RS мікропроцесор повідомляє контролеру індикатора про те, що саме передається по шині: команда або дані. Сигнал на лінії E є стробом, що супроводжує сигнали на шині "команди/дані". Запис інформації в РКІ відбувається по спаду цього сигналу. Потенціал на керуючому виводі R/W задає напрямок передачі даних: запис в ОЗП індикатора (R/W=0) або зчитування (R/W=1). Для випадку, коли мікроконтролер має обмежену кількість ліній вводу/виводу, передбачений другий варіант підключення РКІ з використанням 4-х розрядної шини "команди/дані". При цьому кожен байт даних передається по лініях DB₄ – DB₇ послідовно двома тетрадами, починаючи зі старшої. Контролер РКІ після прийому байта команди або байта даних потребує деякого часу (табл. 3.3) для обробки отриманої інформації, протягом якого AVR-мікроконтролер не повинен виконувати нових передач.

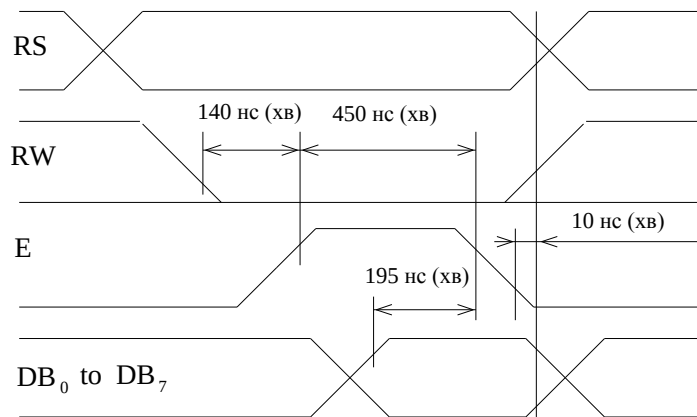


Рис. 3.14.

Для того, щоб визначити, коли контролер PKI закінчить свої внутрішні операції, AVR може опитувати BUSY-прапор індикатора (команда "зчитування busy-прапора"), що скинеться тільки тоді, коли контролер PKI звільниться. Другий, більш простий спосіб полягає в тому, що керуючий мікроконтролер, знаючи скільки часу потрібно PKI на обробку тієї або іншої команди, просто виконує часову затримку після кожної передачі інформації. Якщо під час циклу запису AVR-мікроконтролер передає в контролер індикатора код команди, то цей код записується в регістр команд контролера PKI, і команда відразу ж починає виконуватися. Якщо AVR-мікроконтролер передає в контролер PKI дані, які представляють собою ASCII-коди відображуваних символів, то вони записуються в буфер даних (DDRAM), що зазвичай містить 80 комірок (рис. 3.15). При записі або зчитуванні буфера даних операція здійснюється до комірки, на яку в цей момент вказує курсор.

Табл. 3.3.

Код										Опис команди	Час виконання команди
R S	R/ W	D B7	D B6	D B5	D B4	D B3	D B2	D B1	D B0		
0	0	0	0	0	0	0	0	0	1	Очистити дисплей і установити курсор в нульову позицію (адреса 0)	від 82 мкс до 1.64мс
0	0	0	0	0	0	0	0	1	*	Установити курсор в нульову позицію (адреса 0). Установити дисплей відносно буфера DDRAM у початкову позицію. Вміст DDRAM при цьому не змінюється.	від 40 мкс до 1.6 мс

0	0	0	0	0	0	0	1	I/ D	S	Установити напрям зсуву курсора праворуч (I/D=1) або ліворуч (I/D=0) при записі/зчитуванні чергового коду в DDRAM. Дозволити (S=1) зсув дисплею разом зі зсувом курсора.	40 мкс
0	0	0	0	0	0	1	D	C	B	Включити (D=1) / виключити (D=0) дисплей. Запалити (C=1) / погасити (C=0) курсор. Зображення курсора зробити мигаючим (B=1).	40 мкс
0	0	0	0	0	1	S/ C	R/ L	*	*	Перемістити курсор (S/C=0) або зрушити дисплей (S/C=1) праворуч (R/L=1) або ліворуч(R/L=0).	40 мкс
0	0	0	0	1	D L	N	F	*	*	Установити розрядність шини даних 4 біти (DL=0) або 8 біт (DL=1), кількість рядків дисплея – один (N=0) або два (N=1), шрифт – 5x7 крапок (F=0) або 5x10 крапок (F=1).	40 мкс
0	0	0	1	A _{CG}					Установка адреси CGRAM. Після цієї команди дані будуть записуватися/зчитувати ся в/з CGRAM.	40 мкс	
0	0	1	A _{DD}					Установка адреси DDRAM. Після цієї команди дані будуть записуватися/зчитувати ся в/з DDRAM.	40 мкс		
0	1	B F	AC					Зчитування стану busy-прапора (BF) і лічильника адреси.	1 мкс		
1	0	Дані					Запис даних в DDRAM або CGRAM.			40 мкс	
1	1	Дані					Зчитування даних з DDRAM або CGRAM.			40 мкс	

DDRAM – Display data RAM – ОЗП ASCII-кодів символів, що відображаються.

CGRAM – Character generator RAM – ОЗП знакогенератору.

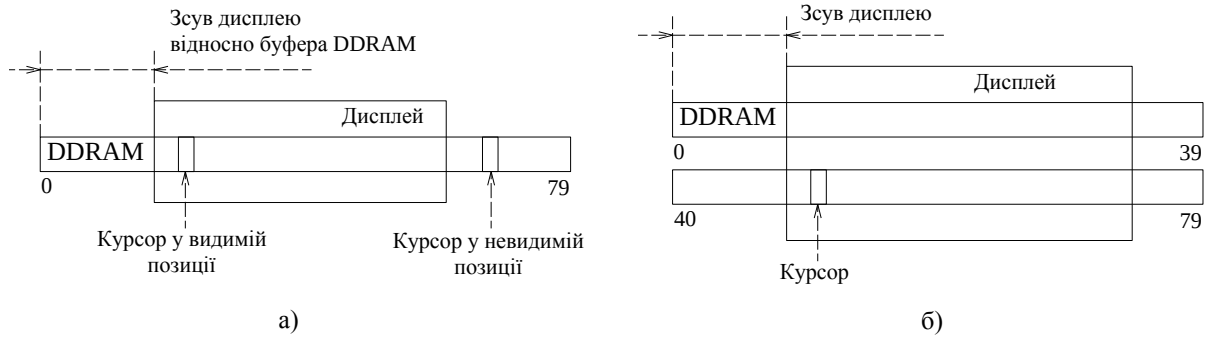
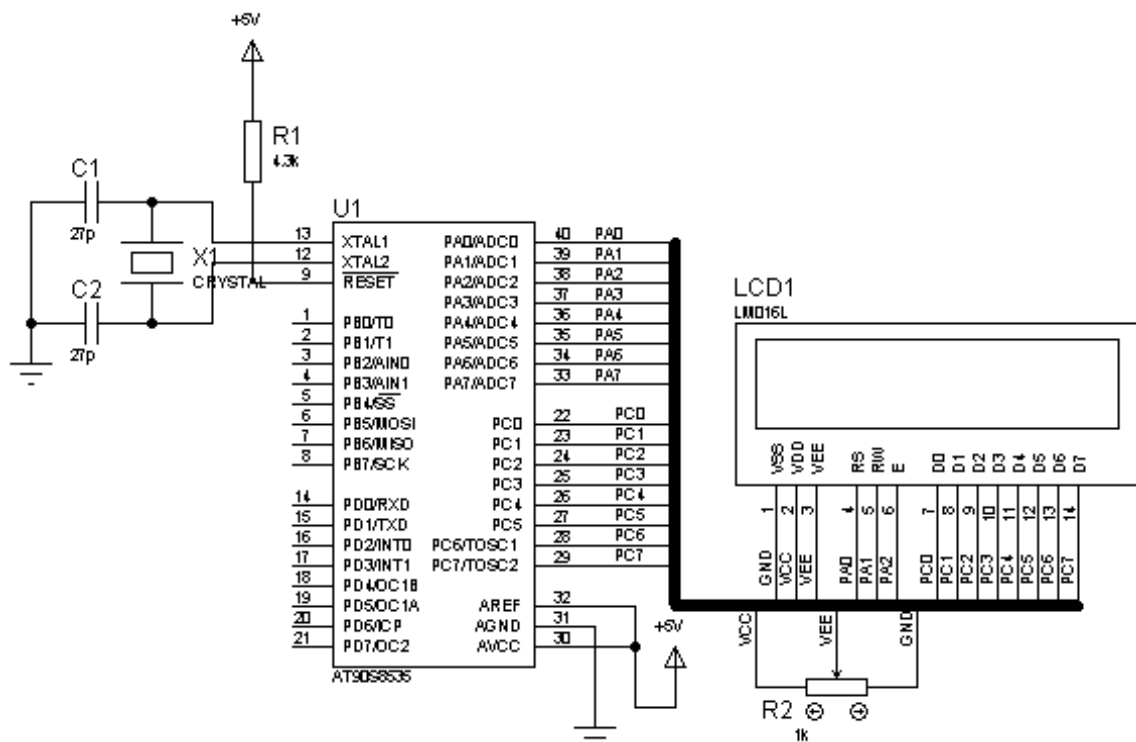


Рис. 3.15.

Буфер даних має більше комірок, ніж кількість знакомісць дисплею. Зміщуючи вікно індикатора відносно буфера даних (див. систему команд), можна відображати на дисплеї різні області буфера. У дворядкових індикаторів перші 40 комірок буфера даних зазвичай відображаються на верхньому рядку дисплею, а другі 40 комірок – на нижньому рядку. Зсув вікна дисплея відносно буфера даних для верхнього і нижнього рядків відбувається синхронно. Курсор буде видний на індикаторі тільки в тому випадку, якщо він потрапив у зону видимості дисплея (і якщо попередньо була подана команда відображати курсор).

На початковому етапі необхідно подати живлення на РКІ-модуль і домогтися від нього ознак працездатності. Схема включення індикатора показана на рис. 3.16.



Змінний резистор R2 дозволяє плавно керувати напругою живлення драйвера РКІ, що призводить до зміни кута повороту рідких кристалів.

Цим резистором можна відрегулювати фактичну контрастність при деякому переважному куті спостереження.

Після закінчення циклу внутрішньої ініціалізації модуль включається в режим розгорнення одного верхнього рядка. При зміні напруги на виводі V_0 сегменти цього рядка повинні змінювати свій стан від прозорого до непрозорого, що є свідченням правильного підключення живлення модуля і працездатності контролера та драйверів РКІ.

Послідовність дій, які необхідно виконувати керуючій системі при здійсненні операцій запису та зчитування для 8-ми розрядної шини демонструє часова діаграма, показана на рис. 3.17.

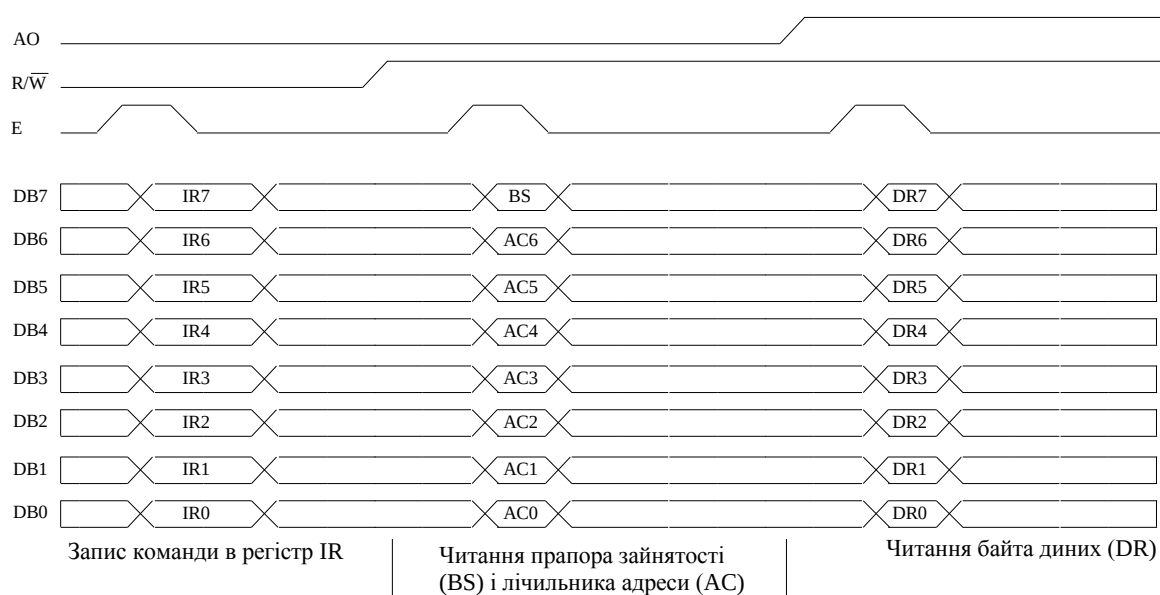


Рис. 3.17.

Операції запису для 8-ми розрядної шини полягають у наступному:

1. Установити значення лінії RS.
2. Вивести значення байта даних на лінії шини DB0...DB7.
3. Установити лінію $E = 1$.
4. Установити лінію $E = 0$.
5. Установити лінії шини DB0...DB7 = HI.

Для зчитування значення з модуля виконуються наступні операції:

1. Установити значення лінії RS.
2. Установити лінію $R/W = 1$.
3. Установити лінію $E = 1$.
4. Зчитати значення байта даних з ліній шини DB0...DB7.
5. Установити лінію $E = 0$.
6. Установити лінію $R/W = 0$.

Приклад 3.12. Написати програму для виводу статичного тексту на алфавітно-цифровий рідкокристалічний індикатор по 8-бітній шині. Принципова схема представлена на рис. 3.16.

Текст програми для мікроконтролера представлений нижче.

```
#include <io8535v.h>           // підключення заголовних файлів
#include <macros.h>             // бібліотека макросів
#include <string.h>              // бібліотека функцій для роботи з рядками
#define RS() PORTA|=0b00000001  // установити сигнал команда/дані
#define NRS() PORTA&=~0b00000001 // скинути сигнал команда/дані
#define RW() PORTA|=0b00000010  // установити сигнал зчитування/запис
#define NRW() PORTA&=~0b00000010 // скинути сигнал зчитування/запис
#define E() PORTA|=0b00000100   // установити сигнал тактування
#define NE() PORTA&=~0b00000100 // скинути сигнал тактування
#define LCD PORTC                // PKI модуль підключений до порту C
#define DDLCD DDRC               // псевдонім регістра напрямку даних
// оголошення прототипів функцій
void init_cpu();                // функція ініціалізації мікроконтролера
void pause();                  // функція часової затримки
void init_lcd();               // функція ініціалізації PKI
void write();                  // функція, при виклику якої формується тактовий сигнал
// реалізація функцій
void init_cpu()                // функція ініціалізації мікроконтролера
{
    DDRA=0b00000111;           // налаштування трьох молодших біт порту A на вивід
    DDLCD=0x00;                // порт, до якого підключений PKI, переводитися
                                // у стан високого імпедансу
    NRS();                     // скидання сигналу команда/дані
    NRW();                     // скидання сигналу зчитування/запис
    NE();                       // скидання тактового сигналу
}
void pause()                   // функція формування часової затримки
{
    int tmp;                    // додаткова змінна
    for(tmp=0;tmp<350;tmp=tmp+1) NOP(); // у циклі контролер виконує порожню
    операцію
}
void init_lcd()                // функція ініціалізації PKI
{
    DDLCD=0xFF;                // налаштування порту на вивід даних
    LCD=0x38;                  // вивід команди 0x38 на порт контролера
    write();                   // запис команди в модуль
    LCD=0x0C;                  // вивід команди 0x0C на порт контролера
    write();                   // запис команди в модуль
    LCD=0x06;                  // вивід команди 0x06 на порт контролера
    write();                   // запис команди в модуль
    LCD=0x02;                  // вивід команди 0x02 на порт контролера
    write();                   // запис команди в модуль
    LCD=0x01;                  // вивід значення 0 на порт контролера
    write();
}
```

```

void write()           // функція, що видає тактовий сигнал
{
    E();                // установка тактового сигналу
    pause();            // часова затримка
    NE();               // скидання тактового сигналу
    pause();            // часова затримка
}
void main()           // основна програма
{
    char text1[]="Hello students!"; // оголошення змінних,
    char text2[]="How are you?";   // інформація, що буде виводитися на екран РКІ
    int index;                     // змінна для використання в циклі
    init_cpu();                   // виклик функції ініціалізації мікроконтролера
    init_lcd();                   // виклик функції ініціалізації РКІ модуля
    DDLC=0xFF;                   // налаштування порту на вивід даних
    RS();                         // установка в 1 сигналу команда/дані; у РКІ передаються дані
    for(index=0; index<strlen(text1); index++) // у циклі виконується запис даних у РКІ
    {
        LCD=text1[index]; // установка ASCII-коду символу на порту
        write();           // запис символу в РКІ; курсор зміщується автоматично
    }
    NRS();                  // скидання сигналу команда/дані; у РКІ записуються команди
    LCD=0;                  // установка значення 0 на порту
    DDLC=0x00;             // переведення порту в стан високого імпедансу
    while(1);              // зациклення програми
}

```

Функція `init_lcd()` виконує початкову ініціалізацію рідкокристалічного модуля у відповідності до системи команд (див. табл. 3.3). У змінних `text1` та `text2` зберігаються строки, що будуть виведені на екран. Оскільки в мові Сі строка – це масив символів, то для виводу кожного символу на екран РКІ в циклі по чергово на порт мікроконтролера видається ASCII-код відповідного символу, після чого викликається функція `write()`, яка формує тактовий сигнал.

На рис. 3.18 представлений результат виконання програми.

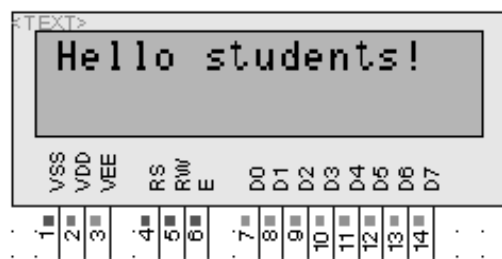


Рис. 3.18.

Приклад 3.13. Написати програму для мікроконтролера, що виводить на рідкокристалічний індикатор рядок, що біжить. Обмін даними із РКІ-модулем виконується по 8-бітній шині.

Текст програми для мікроконтролера представлений нижче.

```

#include<io8535v.h>           // підключення заголовних файлів
#include<macros.h>           // бібліотека макросів
#define RS() PORTA|=0x01     // установити сигнал команда/дані
#define NRS() PORTA&=~0x01   // скинути сигнал команда/дані
#define RW() PORTA|=0x02     // установити сигнал зчитування/запис
#define NRW() PORTA&=~0x02   // скинути сигнал зчитування/запис
#define E() PORTA|=0x04      // установити сигнал тактування (біт 7 порту A)
#define NE() PORTA&=~0x04    // скинути сигнал тактування
// оголошення прототипів функцій
void init_cpu(void);         // функція ініціалізації контролера
void pause();               // часова затримка
void init_lcd(void);        // функція ініціалізації РКІ
void delay(int ms);
// реалізація функцій
void init_cpu()              // функція ініціалізації контролера
{
    DDRA=0b00000111;        // налаштування трьох молодших біт порту A на вивід
    DDRC=0xFF;
    NRS();                  // скидання сигналу команда/дані
    NRW();                 // скидання сигналу зчитування/запис
    NE();                   // скидання тактового сигналу
}
void pause()                // функція часової затримки
{
    int t;                  // додаткова змінна
    for (t=0;t<800;t++)NOP(); // у циклі виконується порожня операція
}
void write()                // функція, що видає тактовий сигнал
{
    E();                    // установка тактового сигналу
    pause();                // часова затримка
    NE();                   // скидання тактового сигналу
    pause();                // часова затримка
}

void init_lcd()             // функція ініціалізації РКІ
{
    PORTC=0b00111000;       // команда – вст. розрядність шини 8 біт,
                             // 2 рядка, шрифт 5×7 крапок
    write();                 // запис команди в модуль
    PORTC=0b00001100;       // команда включити дисплей,
                             // погасити курсор
    write();
    PORTC=0x06;             // команда дозволу зсуву курсору вправо
                             // заборонити зсув дисплея з курсором
    write();
    PORTC=0x01;             // команда очистити дисплей й установити
                             // курсор у нульову позицію
    write();
}

```

```

void delay(int ms)           // функція затримки
{
int t1,t2;                   // додаткові змінні
for (t1=0;t1<ms;t1++)       // у циклі виконується «пуста» операція
for (t2=0;t2<970;t2++)NOP();
}
void main()                   // основна програма
{
char text1[]="Running string"; // оголошення змінних
int i=0;                     // змінна для використання в циклі
init_cpu();                   // виклик функції ініціалізації контролера
init_lcd();                   // виклик функції ініціалізації РКІ-модуля
RS();                         // установка в 1 сигналу команда/дані;
                               // у РКІ передаються дані
pause();                     // затримка
for(i=0;i<15;i++)            // у циклі виконується запис даних у РКІ
{
PORTC=text1[i];              // установка ASCII-коду символу на порт C
write();                     // запис символу в РКІ; курсор зміщається
автоматично
}
NRS();                       // скидання сигналу команда/дані;
                               // у РКІ записуються команди

while(1)
{
PORTC=0b00011100;           // команда зсуву дисплея на одну позицію
write();                     // формування тактового сигналу
delay(300);                  // затримка, регулюється швидкість руху тексту
}
}

```

У програмі функція `init_cpu()` виконує ініціалізацію мікроконтролера. У цій функції налаштовуються 3 молодших біти порту А и порт С на вихід, а також установлюються сигнали низького рівня на виводах PA0-PA2. Функція `init_lcd()` виконує ініціалізацію РКІ-модуля: включення, установка 8-розрядної шини, режим відображення курсору, очищення дисплея та установка курсору в початкову позицію. Для створення ефекту «рядок, що біжить» у циклі виконується зсув екрана праворуч. На рис. 3.19 представлений результат виконання програми.

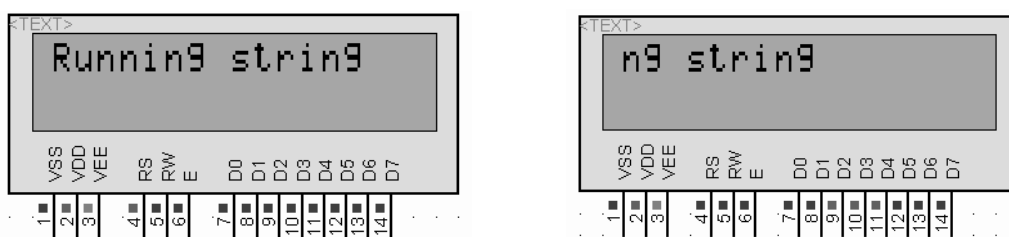


Рис. 3.19.

3.2.4. Керування графічними РКІ

Рідкокристалічний модуль MTG12863B складається з контролеру керування та рідкокристалічної панелі. Кожній точці, що світиться на РКІ, відповідає „1” в ОЗП модуля. Зовнішній вигляд модуля MTG12863B представлений на рис. 3.20.

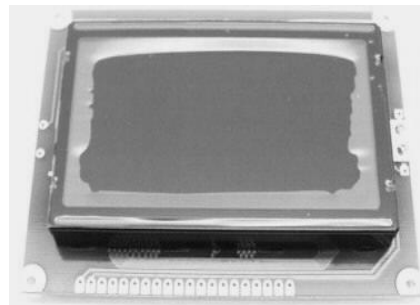


Рис. 3.20.

Також випускається широкий набір графічних рідкокристалічних модулів, що відрізняються конструктивним оформленням, розмірами, розрішаючою здатністю (рис. 3.21)



MT-12232A



WG12864A



YMS160160-01



YMS240128-01

Рис. 3.21.

Модуль дозволяє приймати команди з шини DB7-DB0 (перелік команд представлений в табл. 3.5), записувати дані в ОЗП по 8-розрядній шині даних (DB7-DB0). Для початкової установки модуля потрібно подати сигнал RES. При цьому модуль виконує команди установки Display Start Line в 0, Display Off. Після деактивації сигналу RES необхідно дочекатися

сбросу бітів BUSY та RESET в регістрі статусу. Після цього модуль нормально функціонує. Опис виводів модуля MTG-S12864BFGHSGY представлений в табл. 3.4.

Табл. 3.4.

№ виводу	Назва виводу	Напрямок даних	Опис
1	V _{dd}	—	Напруга живлення
2	V _{ss}	—	Земля
3	V _{ee}	—	Керування контрастністю
4-11	DB0-DB7	I/O	Шина даних з трьома станами
12	CS1	I	Вибір контролера лівої половинки дисплею. Активний рівень – низький
13	CS2	I	Вибір контролера правої половинки дисплею. Активний рівень – низький
14	RES	I	Установка сигналу в низький рівень ініціалізує графічний модуль
15	R/W	I	R/W=1 – дані можуть бути зчитані з модуля в контролер R/W=0 – дані можуть бути записані в дисплей
16	D/I	I	D/I=1 – дані на DB7-DB0 – дані для відображення D/I=0 – дані на DB7-DB0 – команда
17	E	I	Сигнал дозволу. Якщо запис (R/W=0), дані записуються по зрізу сигналу E. Якщо зчитування (R/W=1), дані зчитуються по фронту сигналу E
18	FGND	—	Земля
19	BKL _A	—	Живлення для підсвітки
20	BKL _K	—	

Модуль містить ОЗП для зберігання даних, які виводяться на РКІ, розміром 64×64×2 біт (по 64×64 біт на кристал). Для вибору потрібного кристалу використовуються виводи CS1 та CS2. Далі вся інформація відноситься до кожного кристалу. ОЗП розбито на 8 сторінок розмірами 64×8 біт кожна. Кожній точці, що світиться на РКІ, відповідає лог. 1 в комірниці ОЗП модуля. Відповідність між адресами в ОЗП і точками, що відображаються на РКІ, приведено на рис. 3.22.

Зчитування (запис) даних з (в) модуль виконується по сторінках (64×8 біт або 64×1 байт). Кожна сторінка представлена як 64 байти. Для читання або запису даних по заданому адресу необхідно попередньо встановити сторінку ОЗП і вибрати стовбець всередині сторінки ОЗП. Це виконується командами „Set Page” та „Set Address” відповідно. Після цього можна записати або прочитати байт даних. Модуль підтримує безперервну послідовність операцій читання та запису: після зчитування (запису)

одного байту лічильник стовпця автоматично збільшується на 1 і модуль готовий до нової операції зчитування (запису) по наступному адресу без попередньої установки сторінки ОЗП та адреси стовпця. Лічильник стовпця рахує в межах однієї сторінки. При досягненні лічильником адреси 63, наступним буде значення 0 і т.д.

Табл. 3.5.

Команда	RD/WR	A0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Функція		
Diplay ON/OFF	0	0	0	0	1	1	1	1	1	0/1	Вмикає або вимикає РКІ, незалежно від даних в екранному ОЗП і внутрішнього стану		
											“1” – увімкнути дисплей		
											“0” – виммкнути дисплей		
Diplay START Line	0	0	1	1	Diplay START Line (0...63)						Визначають строчку ОЗП, яка буде відображуватися в верхній строчці РКІ (стартова строка РКІ).		
Set page	0	0	1	0	1	1	1	Page (0...7)			Встановлює сторінку ОЗП (стор. 0...7)		
Set Address	0	0	0	1	Column address (0...63)						Встановлює адресу стовпця		
Status Read	1	0	BUSY	0	ON/OFF	RESET	0	0	0	0	Читання режиму стану		
											BUSY	1 0	модуль зайнятий внутрішньою обробкою
													модуль готовий до роботи з зовнішнім МП
											ON/OFF	1 0	РКІ вимкнено
													РКІ ввімкнено
											RESET	1 0	стан скидання
													нормальний стан
Write Display Data	0	1	Write Data								Запис даних в ОЗП модуля	Ці команди обирають ОЗП по раніше заданій адресі, після чого адреса стовпця інкрементується	
Read Display Data	1	1	Read Data								Зчитування даних з ОЗП модуля		

Модуль підтримує команду “Display START Line”, яка встановлює номер верхньої строки, що відображається. Номер може бути в інтервалі від 0 до 63, що відповідає інтервалу від першої строки нульової сторінки

ОЗП до останньої строки сьомої сторінки ОЗП. Відразу за сьомою сторінкою відображається знову нульова сторінка ОЗП. Це дозволяє реалізувати плавний зсув інформації на РКІ по вертикалі шляхом зміни першої строчки, що буде відображатися.

На відміну від алфавітно-цифрових рідкокристалевих модулів, в яких для виводу тексту достатньо було записати ASCII-коди символів в пам'ять, в графічних модулях така можливість відсутня. Тому задача по відображенню інформації (як текстової, так і графічної) вирішується програмним шляхом. Кожна сторінка пам'яті має 8 біт (точок) в висоту. При виводі текстової інформації необхідно витримати проміжки між строчками для зручності сприйняття тексту (хоча це не обов'язково). В алфавітно-цифрових РКІ найчастіше розміри символу дорівнюють 5×7 точок. Розміри знакомісця дорівнюють 6×8 точок. Нижня строчка та крайня права завжди погашені. Це зроблено для формування відступів між символами. Для відображення одного знакомісця вказаних розмірів, потрібно записати в ОЗП модуля 6 байт даних, попередньо вказавши адрес сторінки та стовбця, де потрібно вивести символ (рис. 3.22). Отже, всі дані, які утворюють шрифт, зручно оформити у вигляді двовимірного масиву, наприклад `font[256][6]`. В цьому масиві будуть зберігатися графічна інформація про 256 символів ASCII, кожний з яких представлений 6 байтами. Керування виводом текстової інформації значно полегшується, якщо індекс елементу масиву співпадає з ASCII-кодом символу. В такому разі достатньо звернутися до елементу масиву по коду символу з таблиці ASCII і розробник отримує доступ до 6 байт даних, які власне і сформують графічне зображення при виводі їх на індикатор. Звичайно, процес створення власного шрифту – задача досить складна і тривала. Але цей процес можна автоматизувати. Один з варіантів програм, за допомогою якої можна отримати потрібні дані для формування символу, представлена на рис. 3.23.

Адреса сторінки D2, D1, D0		Адреса строки
0.0.0	D0	00н
	D1	01
	D2	02
	D3	03
	D4	04
	D5	05
	D6	06
	D7	07
0.0.1	D0	08
	D1	09
	D2	0A
	D3	0B
	D4	0C
	D5	0D
	D6	0E
	D7	0F
0.1.1	D0	30
	D1	31
	D2	32
	D3	33
	D4	34
	D5	35
	D6	36
	D7	37
1.1.1	D0	38
	D1	39
	D2	3A
	D3	3B
	D4	3C
	D5	3D
	D6	3E
	D7	3F
Адреса колонки (адреса байта ОЗП в сторінці HEX)	00 01 02 03 04 05 06 07.....3E 3F	
Номер колонки на РКІ	0 1 2 3 4 5 6 7.....62 63	

Для лівої половини відображуваного поля точок (1-й кристал)

Адреса сторінки D2, D1, D0		Адреса строки
0.0.0	D0	00н
	D1	01
	D2	02
	D3	03
	D4	04
	D5	05
	D6	06
	D7	07
0.0.1	D0	08
	D1	09
	D2	0A
	D3	0B
	D4	0C
	D5	0D
	D6	0E
	D7	0F
0.1.1	D0	30
	D1	31
	D2	32
	D3	33
	D4	34
	D5	35
	D6	36
	D7	37
1.1.1	D0	38
	D1	39
	D2	3A
	D3	3B
	D4	3C
	D5	3D
	D6	3E
	D7	3F
Адреса колонки (адреса байта ОЗП в сторінці HEX)	00 01 02 03 04 05 06 07.....3E 3F	
Номер колонки на РКІ	0 1 2 3 4 5 6 7.....62 63	

Для правої половини відображуваного поля точок (2-й кристал)

Рис. 3.22.

Шляхом натискання на кнопки в лівій частині діалогового вікна, розробник формує зображення символу. Потім, натиснувши кнопку „Преобразовать” у списку з’явиться 6 чисел, які і утворюють зображення. Записавши послідовно ці числа в ОЗП індикатора, можна буде побачити зображення букви „А”. Зображення легко інвертувати, натиснувши на кнопку „Инверсия”. Якщо вивести в індикатор інверсні дані, то буква „А” буде зображена світлим на темному фоні. Зрозуміло, що таким способом можна надати буквам будь-яке начертання за бажанням, створити власні, унікальні символи. Для виводу текстової інформації не обов’язково формувати всі символи таблиці ASCII. Достатньо букв алфавіту (російського, українського, англійського), цифр та деяких знаків (оклику, питання, коми, крапки і т.д.).

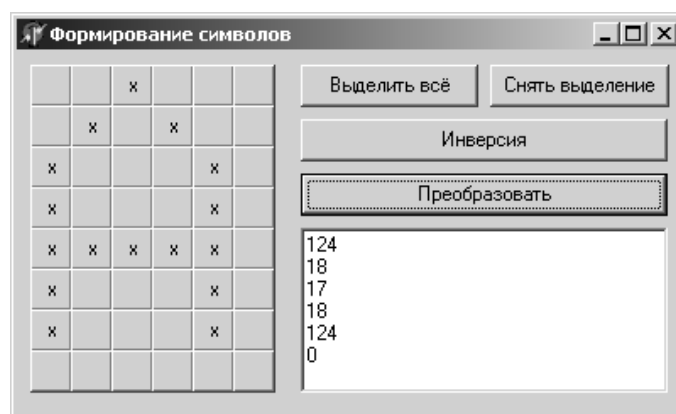


Рис. 3.23.

Крім тексту можна виводити також різні зображення. Принцип формування зображення залишається таким самим. Потрібно сформувати масив даних у пам'яті контролера (Flash, EEPROM), а потім в циклі записати ці дані в індикатор. Звичайно, потрібно слідкувати за значенням стовпця, в який виводяться дані в поточний момент, і при досягненні значення 63, змінювати номер сторінки пам'яті.

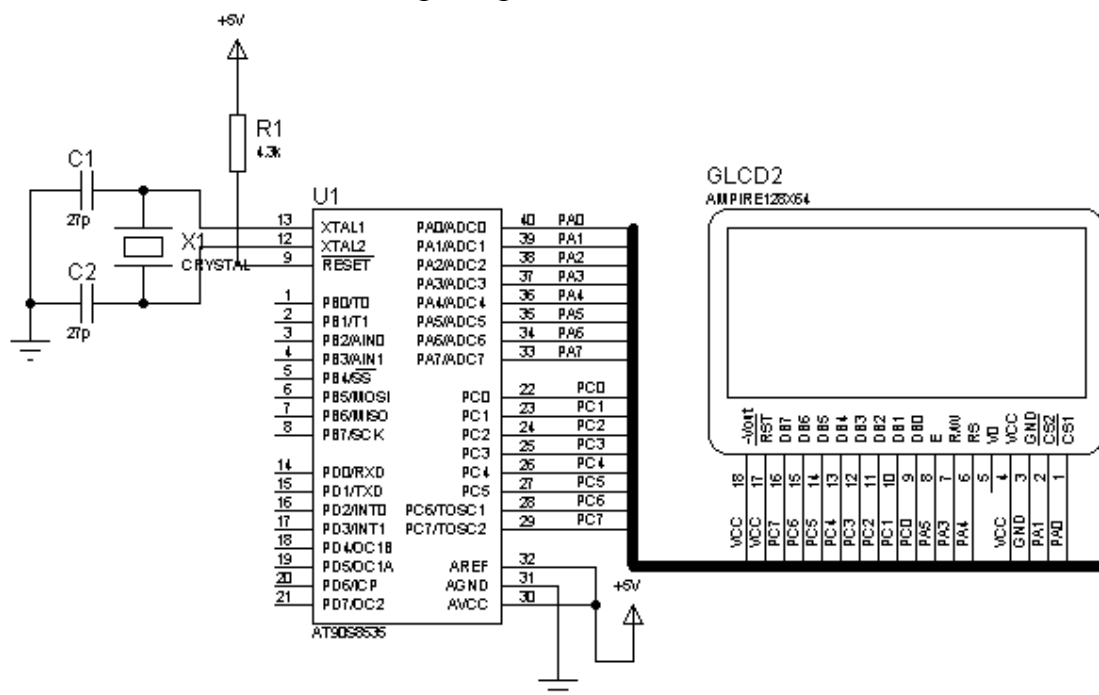


Рис. 3.24.

Приклад 3.14. Написати програму, яка виводить текстову інформацію в лівій частині індикатора та фотографію в правій частині індикатора.

Текст програми з коментарями представлений нижче.

```
#include <io8535v.h> // підключення заготовочних файлів
#include <macros.h> // підключення бібліотеки макросів
```


132


```

NCS1();           // сброс сигнала CS1
NCS2();           // сброс сигнала CS2
NRS();            // сброс сигнала RS
PORTC=0;          // вивід в порт значення 0
DDRC=0;           // перевід порту C в стан високого імпедансу
}

void clear_lcd(void) // функція очистки екрану індикатора
{
    char i;         // додаткова змінна для використання в циклі
    for(i=0;i<8;i++) // цикл виконується 8 разів
        clear_row(i,0); // виклик функції очистки рядку обох кристалів дисплею (параметр 0)
    DDRC=0xFF;       // налаштування порту C на вивід даних
    NRS();            // сброс сигнала RS – передаються команди
    PORTC=0b11000000; // виклик команди Display Start Line; зверху екрану
                     // відображається 0 строчка
    write();          // запис команди в індикатор
    hi_z();           // перевод порту в стан високого імпедансу
}

void clear_row(char row, char half) // функція очистки окремого рядку індикатора
{
    char i;         // додаткова змінна для використання в циклі
    PORTC=0;        // вивід в порт C значення 0
    DDRC=0xFF;       // налаштування порту C на вивід даних
    NRS();            // сброс сигнала RS – передаються команди
    NRW();           // сброс сигнала RW
    if(half==1)       // якщо вибрана ліва половина екрану
    {
        NCS1();       // сброс сигнала CS1
        CS2();         // установка сигнала CS2
    }
    else if(half==2)  // якщо вибрана права половина екрану
    {
        CS1();         // установка сигнала CS1
        NCS2();         // сброс сигнала CS2
    }
    else              // якщо вибрані обидві половинки екрану
    {
        NCS1();        // сброс сигнала CS1
        NCS2();        // сброс сигнала CS2
    }
    row&=0x07; // виділення трьох молодших біт змінної row (макс. можливе значення 7)
    row|=0b10111000; // формування команди Set Page
    PORTC=row;       // вивід даних на порт C
    write();          // запис команди в індикатор
    PORTC=0b01000000; // передача команди Set address – установка значення
                     // лічильника стовбців в 0
    write();          // запис команди в модуль
    RS();             // установка сигнала RS – передаються дані
    PORTC=0;          // установка значення 0 на порту C
    for(i=0;i<64;i++) // цикл виконується 64 рази

```

```
{
    write();                // запис 0 в пам'ять індикатора – очистка рядку
}
hi_z();                    // перевід порту у високоімпедансний стан
}

void init_lcd(void)        // функція ініціалізації PKI
{
    char i;                // додаткова змінна
    NRES();                // сброс сигналу RES
    delay_us(500);         // часова затримка 500 мкс
    RES();                 // установка сигналу RES
    delay_us(500);         // часова затримка 500 мкс
    NCS1();                // сброс сигналу CS1
    NCS2();                // сброс сигналу CS2
    NRW();                // сброс сигналу R/W

    NDI();                 // сброс сигналу D/I (RS) – передаються команди
    PORTC=0b00111111;      // виклик команди Display ON
    write();               // запис команди в індикатор
    PORTC=0b11000000;      // виклик команди Display StartLine
    write();               // запис команди в індикатор
    PORTC=0b10111000;
    // виклик команди Set Page – установка сторінки пам'яті з номером 0
    write();               // запис команди в модуль
    PORTC=0b01000000;      // виклик команди Set Address
    write();               // запис команди в індикатор
    hi_z();                // перевод порту в стан високого імпедансу
}

void delay_ms(int ms)      // функція затримки; передається параметр – затримка в мс
{
    char i,j;              // додаткові змінні
    int k;
    for(k=0;k<ms;k++)      // в циклі контролер виконує холості операції
    {
        for(i=0;i<20;i++)
        for(j=0;j<75;j++);
    }
}

void delay_us(int us)      // функція затримки; передається параметр – затримка в мкс
{
    int k;                 // додаткова змінна
    for(k=0;k<=us;k++)     // в циклі контролер виконує холості операції
    {
        NOP();             // пуста операція
    }
}

void out_string(char * str, char row, char stpos) // функція виводу текстової строчки в
задану позицію
```

```

{
    char len=strlen(str);           // визначення довжини виводимої тестової строчки
    char i,j;                       // додаткові змінні для роботи алгоритму
    char curpos=stpos;              // запам'ятовування стартової позиції
    clear_row(row,0);               // виклик функції очистки строки
    PORTC=0;                        // вивід значення 0 в порт C
    DDRC=0xFF;                     // налаштування порту C на вивід
    NRS();                          // сброс сигналу RS – передаються команди
    delay_us(2);                   // затримка 2 мкс
    NCS1();                        // сброс сигналу CS1
    NCS2();                        // сброс сигналу CS2
    row&=0x07;                     // виділення трьох молодших бітів змінної row
    row=row|=0b10111000;           // формування команди Set Page
    PORTC=row;                     // вивід команди в порт
    write();                       // запис команди в порт
    CS2();                         // установка сигналу SC2
    stpos&=0b00111111;
    // виділення 6 молодших бітів значення стартової позиції (макс. 63)
    stpos|=0b01000000;            // формування команди Set Address
    PORTC=stpos;                  // вивід команди в порт
    write();                      // запис команди в порт
    RS();                         // установка сигналу RS – передаються дані
    delay_us(2);                 // затримка 2 мс
    for(i=0;i<len;i++)           // цикл виконується стільки разів, скільки букв в виводі мій
    строчці
    {
        for(j=0;j<6;j++)         // вкладений цикл; виводяться дані про символ з
        двовимірною масиву
        {
            PORTC=currfont[str[i]-32][j]; // установка значення j-елементу масиву на порту C
            write();                 // запис даних в PKI
            curpos++;               // збільшення на 1 внутрішнього лічильника
            if(curpos==64)          // якщо досягнутий край половинки екрану
            {
                CS1();              // сброс сигналу CS1
                NCS2();            // установка сигналу CS2
                NRS();             // сброс сигналу RS – передаються команди
                PORTC=0b01000000; // установка команди Set Address
                write();            // запис команди в PKI
                RS();              // установка сигналу RS – передаються дані
            }
        }
    }
    hi_z();                       // перевід порту в стан високого імпедансу
}

// функція виводу на екран даних ,що зберігаються у Flash-пам'яті контролера
void draw_flash(char half)
{
    char i=0;                    // додаткова змінна
    char page=0;                 // змінна – номер сторінки пам'яті; початкове значення 0
    int a=0;                    // додаткова змінна

```

```
char data;
DDRC=0xFF;           // налаштування порту C на вивід
if(half==1) // якщо значення змінної half дорівнює 1 (ліва половина екрану)
{
    CS2();             // установка сигналу CS2
    NCS1();            // сброс сигналу CS1
}
else if(half==2) // якщо значення змінної half дорівнює 2 (права половина екрану)
{
    NCS2();            // сброс сигналу CS1
    CS1();             // установка сигналу CS2
}
else // якщо значення змінної half дорівнює 0 (або інше) (обидві половинки екрану)
{
    NCS2();            // сброс сигналу CS2
    NCS1();            // сброс сигналу CS1
}
for(page=0;page<8;page++) // в циклі виконується запис даних в кожен сторінку пам'яті
почергово
{
    NRS();             // сброс сигналу RS – передається команда
    PORTC=0b11000000; // вивід команди Display Start Line
    write();           // запис команди в індикатор
    PORTC=0b10111000|page; // вивід команди Set Page
    write();           // запис команди в індикатор
    PORTC=0b01000000; // вивід команди Set Address
    write();           // запис команди в індикатор
    for(i=0;i<64;i++) // цикл повторюється 64 рази
    {
        RS();          // установка сигналу RS – запис даних
        PORTC=~image[a]; // установка даних на порту C
        write();        // запис даних в індикатор
        a++;            // інкремент змінної
    }
}
hi_z();              // перевід порту в стан високого імпедансу
}

void main(void)
{
    char name[]="Photo on"; // оголошення змінних, в яких зберігається текстова
інформація
    char sname[]="Graphic"; // кожна змінна – одновимірний масив
    char surname[]="Display"; // елементи масиву – окремі букви
    char ed1[]="Magister of"; // розмір кожного з масивів дорівнює кількості букв в слові
    char ed2[]="electronics";
    char vuz[]="TEES 2011";

    init_device();      // виклик функції ініціалізації контролера
    init_lcd();         // виклик функції ініціалізації рідкокристалового індикатора
    clear_lcd();        // очистка екрану РКІ
}
```

```

out_string(name,0,0);    // виклик функції виводу тексту;
                        // 0 строчка починаючи з позиції 0

out_string(sname,1,0);   // виклик функції виводу тексту;
                        // 1 строчка починаючи з позиції 0
out_string(surname,2,0); // виклик функції виводу тексту;
                        // 2 строчка починаючи з позиції 0
out_string(ed1,4,0);     // виклик функції виводу тексту; 4 строчка починаючи з позиції 0
out_string(ed2,5,0);     // виклик функції виводу тексту; 5 строчка починаючи з позиції 0
out_string(vuz,7,0);     // виклик функції виводу тексту;
                        // 7 строчка починаючи з позиції 0
draw_flash(2);           // виклик функції малювання зображення в правій частині
                        // екрану (параметр дорівнює 2)

while(1);                // зациклення мікроконтролера
}

```

В результаті виконання програми на графічному індикаторі формується зображення, представлене на рис. 3.25.



Рис. 3.25.

3.3. Універсальний асинхронний приймач-передавач UART

Мікроконтролери Atmel оснащені повнодуплексним універсальним приймачем-передавачем (UART). Його основні можливості наступні:

- Повнодуплексна робота (окремі регістри послідовного прийому та передачі);
- Асинхронна або синхронна робота;
- Робота в режимі ведучого або веденого;
- Висока розв'язна здатність генератора швидкості зв'язку;
- Підтримка формату переданих даних з 5, 6, 7, 8 або 9 бітами даних та 1 або 2 стоп-бітами;
- Апаратна генерація та перевірка біта паритету (парність/непарність)
- Визначення переповнення даних;
- Визначення помилки в структурі посилки;

- Фільтрація шуму з детекцією помилкового старт-біта та цифровим ФНЧ;
- Три роздільних переривання по завершенні передачі, звільнення регістра переданих даних і завершення прийому;
- Режим багатопроцесорного зв'язку;
- Режим подвоєння швидкості зв'язку в асинхронному режимі.

3.3.1. Формати передачі даних

Послідовна посилка складається з біт даних, біт синхронізації (старт і стоп-біти), а також біта паритету для пошуку помилок. Універсальний асинхронний приймач-передавач (УАПП) підтримує всі 30 комбінацій наступних форматів посилки:

- 1 старт-біт;
- 5, 6, 7, 8 або 9 біт даних;
- без паритету, з бітом парності чи бітом непарності;
- 1 або 2 стоп-біта.

Посилка починається зі старт-біта, а за ним відбувається передача біт даних, починаючи із самого молодшого розряду. Потім виконується передача інших біт даних (макс. число біт даних 9), яка закінчується передачею старшого розряду даних. Якщо дозволена функцію контролю паритету, то відразу після біт даних передається біт паритету, а потім стоп-біти. Після завершення передачі посилки є можливість або передавати наступну посилку, або перевести лінію зв'язку в стан очікування (високий рівень). Рис. 3.26 ілюструє можливість сполучення форматів посилки.

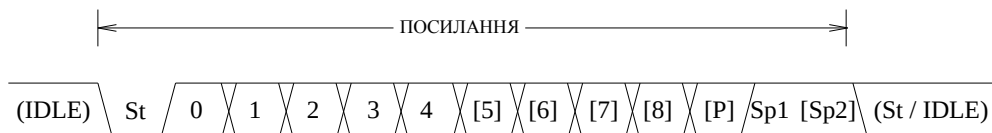


Рис. 3.26.

St – Стартовий біт має завжди низький рівень.

0...8 – Номер біта даних.

P – біт паритету: парність або непарність.

Sp1, Sp2 – Стоп-біти мають завжди високий рівень.

IDLE – стан очікування, у якому припинена передача на Rx або Tx. У стані очікування на лінії повинен бути високий рівень.

Біт паритету обчислюється шляхом виконання логічної операції «Виключаюче АБО» над всіма бітами даних. Якщо використовується непарність, то результат цієї операції інвертується. Наступні вирази демонструють це:

$$P_{\text{ЧЕТН}} = d_{n-1} \oplus \dots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0$$

$$P_{\text{НЕЧЕТН}} = d_{n-1} \oplus \dots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1,$$

де $P_{\text{четн}}$ – біт парного паритету; $P_{\text{нечетн}}$ – біт непарного паритету; d_n – n -ий біт даних у посилці.

Після дозволу, біт паритету передається між останнім бітом даних і першим стоп-бітом.

3.3.2. Ініціалізація та налаштування швидкості зв'язку

Бод-генератор представляє собою дільник, що генерує імпульси передачі із частотою, відповідно до формули:

$$BAUD = \frac{f_{\text{СК}}}{16(UBRR + 1)},$$

де $BAUD$ – швидкість передачі даних у бодах; $f_{\text{СК}}$ – частота кварцового генератора, $UBRR$ – зміст регістра $UBRR$ (0 – 255).

Регістр бод-генератора UART – $UBRR$

Біт	7	6	5	4	3	2	1	0
$UBRR$	MSB							LSB
Зчитування/ запис	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Початкове значення	0	0	0	0	0	0	0	0

Регістр $UBRR$ – це 8-розрядний регістр із можливістю зчитування/запису даних. Швидкість передачі даних UART визначається значенням, записаним у цей регістр.

3.3.3. Передача та приймання даних

Передача даних ініціюється записом байту у регістр даних UART (UDR). Дані пересилаються з регістра UDR у регістр зсуву передачі в наступних випадках:

- новий символ записаний в UDR після того як був переданий з регістра зсуву стоповий біт попереднього символу. Регістр зсуву завантажується негайно;

- новий символ записаний в UDR до того, як був переданий стоповий біт попереднього символу. Дані в регістр зсуву завантажуються після виходу стопового біта переданого символу.

- якщо з 10 (11) – розрядного регістра зсуву передачі виведена вся інформація, у нього пересилаються дані з UDR . У цей час встановлюється біт $UDRE$ (UART Data Register Empty) регістра стану UART (USR). При встановленому в стан 1 біті $UDRE$ (регістр даних порожній), UART готовий прийняти наступний символ. Прапор $UDRE$ скидається при записі даних у регістр UDR . У той же час, коли дані пересилаються з регістра UDR в 10 (11) – розрядний регістр зсуву, біт 0 регістра зсуву встановлюється в стан 0 (стан 0 – стартовий біт), а біт 9 або 10 встановлюється в стан 1 (стан 1 – стоповий біт). Якщо в регістрі керування UART (UCR) установлений біт $CHR9$ (тобто обраний режим 9-розрядного

слова даних), то біт TXB8 регістра UCR пересилається в біт 9 регістра зсуву.

Відразу після пересилання даних у регістр зсуву, стартовий біт передається на вивід TXD. За ним відбувається передача інших біт даних. Коли буде виданий стоповий біт, регістр зсуву завантажується новою порцією даних, якщо вони були записані в UDR під час передачі. У процесі завантаження біт UDRE перебуває у встановленому стані. Якщо ж нові дані не будуть завантажені в UDR до видачі стопового біта, прапор UDRE залишається встановленим. У цьому випадку, після того, як стоповий біт буде присутній на виводі TXD протягом одного такту, у регістрі стану UART (USR) установлюється прапор завершення передачі TXC (TX Complete Flag).

Установлений у стан 1 біт TXEN регістра UCR дозволяє передачу UART. При очищенні біта TXEN (скидання в стан 0), вивід PD1 може бути використаний як порт вводу/виводу загального призначення. При встановленому біті TXEN передавач UART підключається до PD1 і використовує його як вивід виходу, незалежно від установки біта 1 у регістрі напрямку DDRD. На рис. 3.27 представлена програмна модель передавача UART.

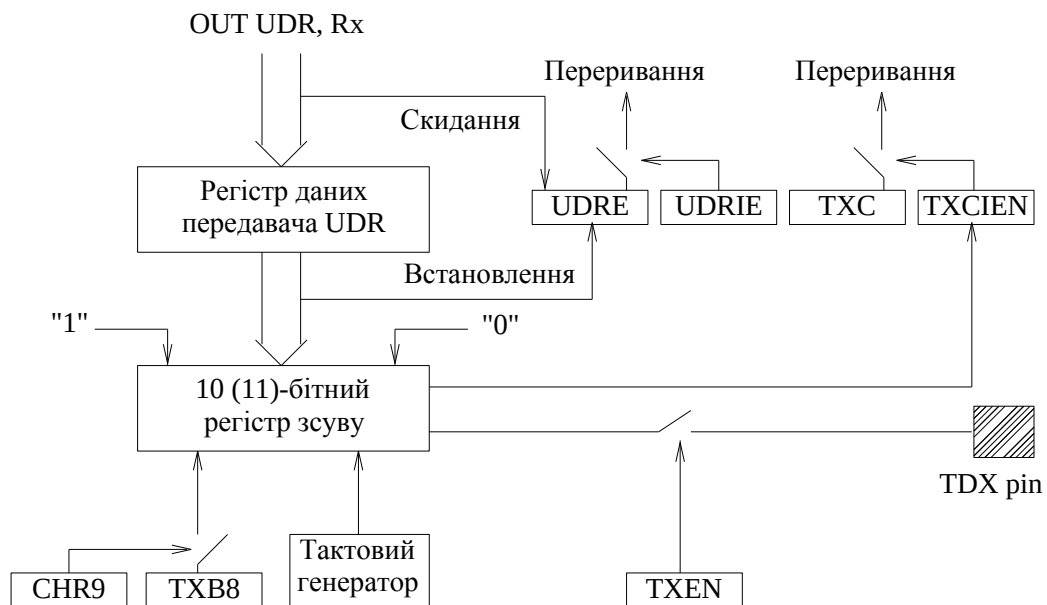


Рис. 3.27.

Логіка відновлення даних робить вибірку станів виводу RXD із частотою в 16 разів більшою, ніж швидкість обміну. Коли лінія перебуває в пасивному стані, одиночна вибірка нульового логічного рівня буде інтерпретуватися як падаючий фронт стартового біта і буде запущена послідовність детектування стартового біта. Вважається, що перша вибірка знайшла перший нульовий логічний рівень імовірного стартового біта. На вибірках 8, 9 й 10 приймач знову перевіряє вивід RXD на зміну логічних станів. Якщо дві або більше із цих трьох вибірок знайдуть логічні 1, то

даний імовірний стартовий біт відкидається як шумовий сплеск і приймач почне виявляти та аналізувати наступні переходи з 1 в 0.

Якщо ж був виявлений дійсний стартовий біт, то починає генеруватися вибірка наступних за стартовим бітом інформаційних біт. Ці біти також перевіряються на вибірках 8, 9 й 10. Логічний стан біта приймається по двох або трьох однакових станах вибірок. Всі біти вводяться в регістр зсуву приймача з тим значенням, що було визначено тестуванням вибірок.

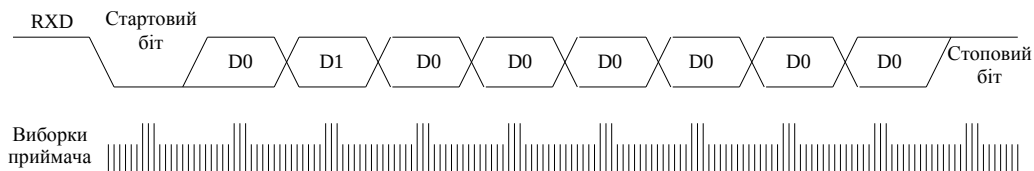


Рис. 3.28.

При надходженні стопового біта необхідно, щоб не менше двох вибірок із трьох підтвердили прийом стопового біта (показали високий рівень). Якщо ж дві або більше вибірки покажуть значення 0, то при пересиланні прийнятого байта в UDR, у регістрі статусу UART (USR) встановлюється біт помилки кадру FE (Framing Error). Для виявлення помилки кадру користувач перед зчитуванням регістра UDR повинен перевіряти стан біта FE. Прапор FE очищується при зчитуванні змісту регістра даних UART (UDR).

Не залежно від того, прийнятий правильний стоповий чи біт ні, дані пересилаються в регістр UDR і встановлюється прапор RXC у регістрі стану UART (USR). Регістр UDR фактично є двома фізично окремими регістрами, один із яких служить для передачі даних, а інший – для прийому. При зчитуванні UDR звернення ведеться до регістра прийому даних, при записі звернення ведеться до регістра передачі. Якщо обрано режим обміну 9-розрядними словами даних (установлений біт CHR9 регістра UCR), при пересиланні даних в UDR біт RXB8 регістра UCR завантажується в біт 9 регістра зсуву передачі. Якщо після одержання символу до регістра UDR не було звернення, починаючи з останнього прийому, у регістрі UCR встановлюється прапор переповнення (OR). Це означає, що нові дані, які пересилаються в регістр зсуву, не можуть бути передані в UDR і будуть загублені. Користувачу для виявлення переповнення необхідно завжди перевіряти прапор OR після зчитування вмісту регістра UDR.

При скинутому біті RXEN регістра UCR робота приймача заборонена. Це означає, що вивід PD0 може використовувати як порт вводу/виводу загального призначення. При встановленому біті RXEN, приймач UART підключається до виводу PD0, який працює як вхід, не залежно від значення відповідного біта в регістрі напрямку передачі даних DDRD. На рис. 3.29 представлена програмна модель приймача UART.

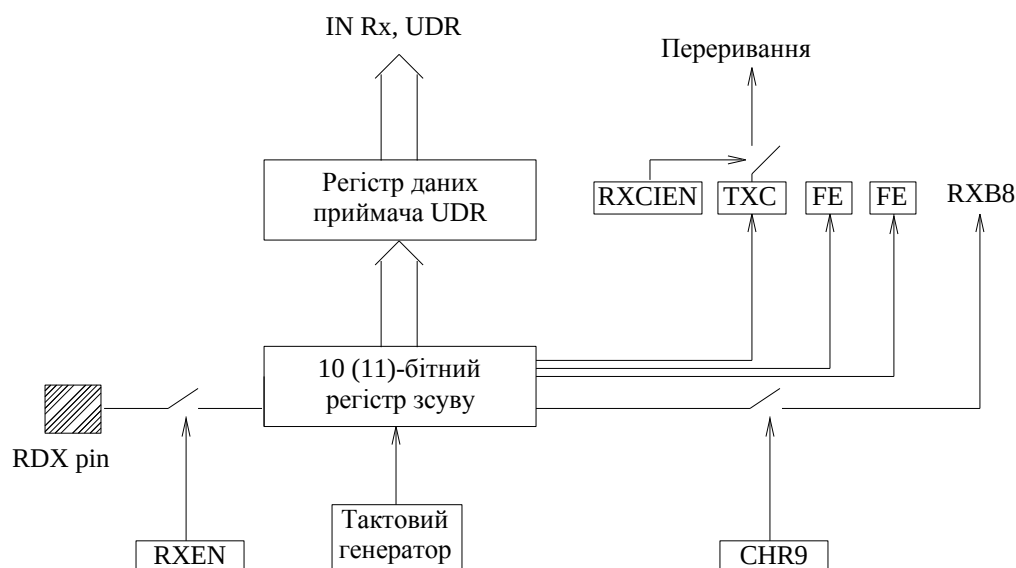


Рис. 3.29.

Регістр даних UART – UDR

Біт	7	6	5	4	3	2	1	0
UDR	MSB							LSB
Зчитування/ запис	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Початкове значення	0	0	0	0	0	0	0	0

У дійсності регістр UDR є двома фізично розділеними регістрами – регістром передачі даних і регістром прийому даних, які використовують ті самі адреси I/O.

Регістр стану UART – USR

Біт	7	6	5	4	3	2	1	0
USR	RXC	TXC	UDRE	FE	DOR	-	-	-
Зчитування/ запис	R/W	R/W	R	R	R	R	R	R
Початкове значення	0	0	0	0	0	0	0	0

Регістр USR забезпечує тільки зчитування інформації про стан UART.

– Біт 7 – RXC: UART Receive Complete – Прийом завершений. Даний біт встановлюється в стан 1 при пересиланні прийнятого символу з регістра прийому в UDR. Біт встановлюється незалежно від того, чи є помилки при прийомі кадру. При встановленому в UCR біті RXCIE і встановленому біті RXC виконується переривання по завершенню прийому UART. Біт RXC очищається при зчитуванні UDR. При прийомі

даних виникає переривання. Підпрограма обробки переривання по завершенню прийому UART повинна зчитувати значення з регістра даних UDR для того, як очиститься біт RXC, інакше при виході з підпрограми обробки переривання відбудеться нове переривання.

– Біт 6 – TXC: UART Transmit Complete – Передача завершена. Даний біт встановлюється в стан 1 коли весь символ (включаючи стоповий біт) переданий з регістра зсуву передачі, а в UDR не записані нові дані. Цей прапор використовується при напівдуплексному інтерфейсі зв'язку, коли устаткування передачі повинне встановити режим прийому та звільнити комунікаційну шину відразу після завершення передачі. При встановленому в регістрі UCR біті TXCIE, установка TXC призведе до виконання переривання по завершенню передачі UART. Прапор TXC очищається апаратно при виконанні обробки відповідного вектора переривання. Очистити біт TXC можна записом значення 1 у цю позицію.

– Біт 5 – UDRE: UART Data Register Empty – Регістр даних порожній. Даний біт встановлюється в стан 1 коли символ, записаний в UDR, пересилається в регістр зсуву передачі. Установка цього біта означає, що передавач готовий до одержання нового символу для передачі. Якщо біт UDRE в UCR встановлений, виконується переривання по завершенню передачі UART доти, поки буде встановлений біт UDRE. Біт UDRE скидається при записі даних у регістр UDR. При прийомі даних виникає переривання. Підпрограма обробки переривання при порожньому регістрі даних UART повинна записати дані в регістр UDR для того, щоб очистити UDRE, інакше по закінченню підпрограми переривання виникне нове переривання.

– Біт 4 – FE: Framing Error – Помилка кадрів. Даний біт встановлюється в стан 1 при виявленні умов помилкового прийому кадру, тобто коли стоповий біт вхідного символу в стані 0. Біт FE очищається при прийомі стопового біта з логічним рівнем 1.

– Біт 3 – DOR: Data OverRun – Переповнення даних. Біт DOR встановлюється в стан 1 при виявленні умов переповнення, тобто коли символ, що вже перебуває в регістрі UDR не був зчитаний перед пересиланням нового символу з регістра зсуву приймача. Біт DOR буферизований. Це означає, що він буде залишатися встановленим поки не будуть правильно зчитані дані з регістра UDR. Біт DOR скидається коли дані прийняті і переслані в регістр UDR.

– Біти 2..0 – Res: Reserved bits – Зарезервовані біти. Ці біти в мікроконтролерах ATmega зарезервовані і при зчитуванні завжди мають значення 0.

Регістр керування UART – UCR

Біт	7	6	5	4	3	2	1	0
UCR	RXCIE	TXCIE	UDRIE	RXEN	TXEN	CHR9	RXB8	TXB8
Зчитування/ запис	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Початкове значення	0	0	0	0	0	0	0	0

– Біт 7 – RXCIE: RX Complete Interrupt Enable – Дозвіл переривання по завершенню прийому. При встановленому в стан 1 біті RXCIE і встановленому дозволі глобального переривання, установка біта RXC у регістрі USR призведе до виконання переривання по завершенню прийому.

– Біт 6 – TXCIE: TX Complete Interrupt Enable – Дозвіл переривання по завершенню передачі. При встановленому в стан 1 біті TXCIE і встановленому дозволі глобального переривання, установка біта TXC у регістрі USR призведе до виконання переривання по завершенню передачі.

– Біт 5 – UDRIE: UART Data Register Empty Interrupt Enable – Дозвіл переривання при порожньому регістрі даних. При встановленому в стан 1 біті UDRIE і встановленому дозволі глобального переривання, установка біта UDRE у регістрі USR призведе до виконання переривання при порожньому регістрі даних UART.

– Біт 4 – RXEN: Receiver Enable – Дозвіл приймача. Встановлений у стан 1 біт RXEN дозволяє роботу приймача UART. Якщо приймач заборонений, то прапори стану TXC, DOR та FE установити неможливо. Якщо прапори встановлені, то скидання біта RXEN не призведе до скидання цих прапорів.

– Біт 3 – TXEN: Transmitter Enable – Дозвіл передавача. Встановлений у стан 1 біт TXEN дозволяє роботу передавача UART. При забороні роботи передавача під час передачі символу, передавач не буде заблокований до того, як буде повністю переданий символ з регістра зсуву плюс будь-який наступний символ, що перебуває в UDR.

– Біт 2 – CHR9:9 Bit Characters – Режим 9-розрядних символів. При встановленому в 1 біті CHR9 передаються та приймаються 9-розрядні символи, а також стартовий і стоповий біти. Дев'яті біти зчитуються і записуються з використанням бітів RXB8 та TXB8 регістра UCR. Дев'ятий біт даних може використовуватися як додатковий стоповий, або біт контролю парності.

– Біт 1 – RXB8: Receive Data Bit 8 – Приймання 8-розрядних даних. При встановленому в стан 1 біті CHR9, біт RXB8 є дев'ятим бітом даних прийнятого символу.

– Біт 0 – TXB8: Transmit Data Bit 8 – Передача 8-розрядних даних. При встановленому в стан 1 біті CHR9, біт TXB8 є дев'ятим бітом у переданому символі.

3.3.4. Багатопроцесорний режим зв'язку

Установка біта багатопроцесорного режиму зв'язку MPCM у регістрі UCSRA активізує функцію фільтрації вхідних посилок приймачем УАПІ. Посилки, які не містять інформації про адресу ігноруються і не переміщуються в буфер приймання. Це дозволяє істотно зменшити кількість вхідних посилок багатопроцесорних системах, зв'язок між процесорами в яких організований через одну послідовну шину. Значення біта MPCM не виконує ніякого впливу на роботу передавача, але при цьому передавач повинен бути використаний інакше, якщо використовується режим багатопроцесорного зв'язку.

Якщо приймач налаштовується на прийом посилок з 5...8 бітами даних, то перший стоп-біт дозволяє відрізнити призначення прийнятих даних: адреса або дані. Якщо приймач налаштований на прийом 9 біт даних, то значення 9-го біта (RXB8) використовується для ідентифікації адреси або даних. Якщо ідентифікатор типу посилки (перший стоп-біт або 9-ий біт даних) дорівнює 1, то в посилці утримується адреса. У протилежному випадку в посилці передані дані.

Режим багатопроцесорного зв'язку дозволяє декільком підлеглим мікроконтролерам приймати дані від одного ведучого. При цьому підлеглі мікроконтролери при першій адресній посилці визначають, до якого мікроконтролера адресується ведучий. Якщо один з підлеглих мікроконтролерів виявляє свою адресу, то наступні посилки даних він буде приймати в нормальному режимі, а інші підлеглі мікроконтролери ці дані ігнорують доти, поки не буде виявлена наступна адресна посилка.

Якщо мікроконтролер діє як ведучий, то він може використовувати 9-бітний формат даних у посилці (UCSZ = 7). 9-ий біт даних (TXB8) установлюється при передачі адресної посилки (TXB8 = 1) і скидається при передачі посилки даних (TXB = 0). У цьому випадку підлеглі мікроконтролери також повинні встановлювати 9-бітний формат.

Для обміну даними в багатопроцесорному режимі зв'язку необхідно використовувати наступні процедури:

1. Всі підлеглі мікроконтролери (МК) переводяться в багатопроцесорний режим зв'язку (MPCM = 1 в UCSRA).
2. Ведучий МК відправляє адресну посилку, а всі підлеглі приймають і зчитують цю посилку. У підлеглих МК прапор RXC у регістрі UCSRA встановлюється як зазвичай.
3. Кожен підлеглий МК зчитує регістр UDR і визначає до кого адресується ведучий МК. Адресований МК повинен очистити біт MPCM в UCSRA, у протилежному випадку він очікує наступного адресного байта і зберігає налаштування MPCM.
4. Адресований МК приймає всі дані до наступної адресної посилки. Інші підлеглі МК, у яких біт MPCM залишився встановленим будуть ігнорувати посилки даних.

5. Після прийому адресованим МК останньої посилки даних, встановлюється біт МРСМ і очікується прийом нової адресної посилки від ведучого МК. Далі процес повторюється з пункту 2.

Використовувати 5..8 – розрядні формати даних можливо, але незручно, тому що приймач повинен перемикається між n та $n+1$ форматами посилки. При використанні 5...8 – розрядних даних у посилці передавач повинен використати два стоп-біти, оскільки перший стоп-біт буде задіяний для індикації типу посилки.

Приклад 3.15. Розробити схему мережі з двох мікроконтролерів, що з'єднані між собою по інтерфейсу RS232 (рис. 3.30). Розробити простий протокол обміну даними між ними та написати програму для кожного з мікроконтролерів.

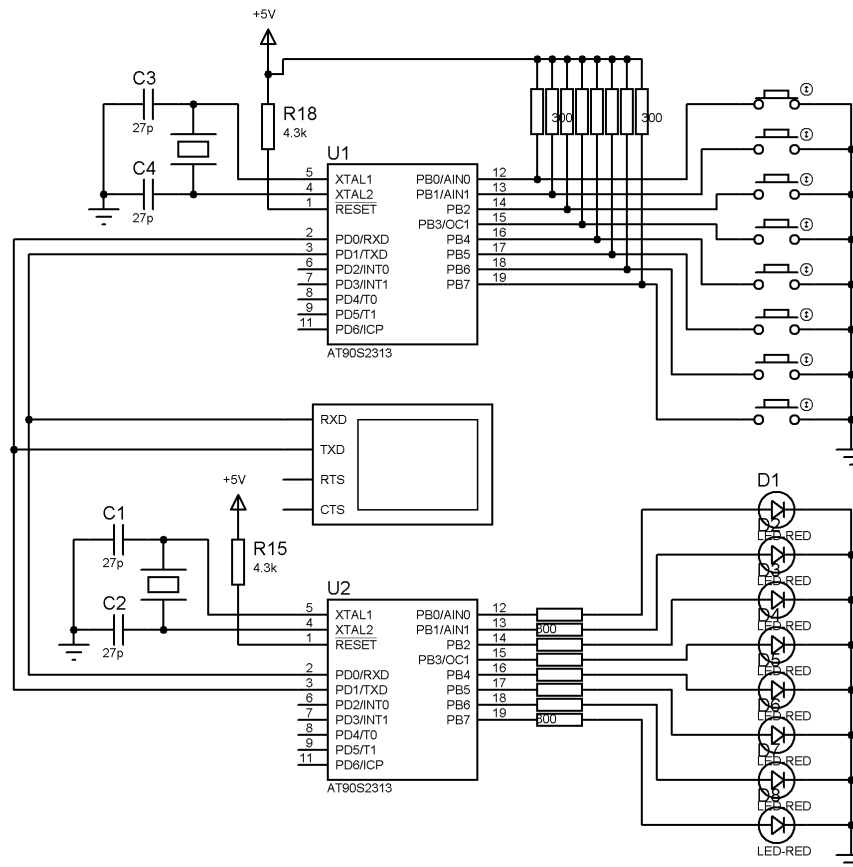


Рис. 3.30.

Для ведучого мікроконтролера текст програми має вигляд:

```
#include <io2313v.h>           // підключення заготовочних файлів
#include <macros.h>
void delay(int ms);           // прототип функції часової затримки
void init_cpu(void);          // функція ініціалізації мікроконтролера
char status;                  // додаткові глобальні змінні
char bcnt;
char rbytes[4];

// обробник переривання по прийманню даних по UART
```

```

#pragma interrupt_handler uart0_rx_isr:8
void uart0_rx_isr(void)
{
    char d;                // додаткова змінна
    d=UDR;                 // зчитування даних з регістру даних
    if(bcnt<2)             // перевірка значення змінної
    {
        rbytes[bcnt]=d;    // запис даних у масив (буфер приймання)
        bcnt++;           // інкремент значення змінної
    }
    else if(bcnt==2)       // якщо значення змінної дорівнює 2
    {
        rbytes[bcnt]=d;    // запис даних у масив (буфер приймання)
        // перевірка прийнятих даних на відповідність формату
        if((rbytes[0]=='Y') && (rbytes[1]=='F') && (rbytes[2]=='R')) status=0;
        bcnt=0;
        delay(100);        // затримка 100 мс
    }
    else bcnt=0;           // обнуління змінної
}

void delay(int ms)        // функція часової затримки
{
    int c1, c2;            // оголошення додаткових змінних
    for(c1=0;c1<ms;c1++)   // в циклі виконується «пуста» операція
        for(c2=0;c2<970;c2++) NOP();
}

void init_cpu(void)       // функція ініціалізації контролера
{
    PORTB = 0;
    DDRB = 0;
    PORTD = 0;
    DDRD = 0;

    UBRR = 0x33;           // установка швидкості 9600 біт/с
    UCR = 0x98;            // дозвіл роботи передавача та приймача
    status=0;              // онулення змінних
    bcnt=0;
    SEI();                 // дозвіл обробки переривань
}

void main()               // головна функція програми
{
    char data[4]={'A', 'N',0,'!'};
    int i;
    init_cpu();            // виклик функції ініціалізації мікроконтролера
    while(1)
    {
        if(status==0)      // якщо значення змінної дорівнює 0
        {
            data[2]=~PINB;  // зчитування даних з порту

```

```
        for(i=0;i<4;i++)
        {
            UDR=data[i];          // передача даних по UART
            while(!(USR&(1<<UDRE))); // завершення передачі чергового байта
        }
        status=1;                // присвоєння змінній значення 1
    }
}
```

Перший мікроконтролер формує інформаційний пакет для передачі даних. Перші два байти, що передаються, є ASCII-кодами символів A та N, і дозволяють ідентифікувати початок пакету даних. Третім байтом передаються дані про стан кнопок, що підключені до порту В мікроконтролера. Четвертий байт є завершальним, і за допомогою нього визначається кінець пакету. В циклі почергово передаються всі байти, після чого змінній status присвоюється значення 1. Ведений мікроконтролер після прийому даних надсилає відповідь, що підтверджує отримання даних. Якщо ведений мікроконтролер прийняв дані і надіслав відповідь, то у головному мікроконтролері змінна status приймає значення 0, і формується новий пакет даних для відправлення. Текст програми для веденого мікроконтролера має вигляд:

```
#include <io2313v.h>          // підключення заголовних файлів
#include <macros.h>
void delay(int ms);           // прототип функції часової затримки
void init_cpu(void);           // функція ініціалізації мікроконтролера
char bcnt;                    // оголошення глобальних змінних
char rbytes[4];
// обробник переривання по прийманню символу
#pragma interrupt_handler uart0_rx_isr:8
void uart0_rx_isr(void)
{
    char d, i, data[3]={ 'Y', 'F', 'R' }; // оголошення масиву символів
    d=UDR;                               // зчитування даних з регістра приймача
    if(bcnt<3)                           // якщо значення змінної менше 3
    {
        rbytes[bcnt]=d;                  // запис даних у масив (буфер приймання)
        bcnt++;                          // інкремент значення змінної
    }
    else if(bcnt==3)                    // якщо умова виконується
    {
        rbytes[bcnt]=d;                  // запис даних у масив (буфер приймання)
        // перевірка прийнятого пакета на відповідність протоколу
        if((rbytes[0]=='A') && (rbytes[1]=='N') && (rbytes[3]!='!')) PORTB=rbytes[2];
        for(i=0;i<3;i++)
        {
            UDR=data[i];                  // передача відповіді у ведучий мікроконтролер
            while(!(USR&(1<<TXC))); // очікування завершення передачі байта даних
            delay(2);                      // затримка 2 мс
        }
    }
}
```



```

    }
    bcnt=0;           // онулення змінної
}
else bcnt=0;
}

void delay(int ms)    // функція часової затримки
{
    int c1, c2;        // оголошення додаткових змінних
    for(c1=0;c1<ms;c1++)
        for(c2=0;c2<970;c2++) NOP();    // в циклі виконується «пуста» операція
}

void init_cpu(void)
{
    PORTB = 0;
    DDRB = 0xFF;       // налаштування порту В на вихід
    PORTD = 0;
    DDRD = 0;
    UBRR = 0x33;        // установка швидкості 9600 біт/с
    UCR = 0x98;         // дозвіл роботи передавача та приймача
    bcnt=0;             // обнуління змінних
    SEI();              // дозвіл обробки переривань
}

void main()           // головна функція програми
{
    init_cpu();        // виклик функції ініціалізації мікроконтролера
    while(1);          // вічний цикл
}

```

В наведеній програмі використовується переривання по прийманню даних з UART. Глобальна змінна `bcnt` використовується як лічильник прийнятих байт. Якщо значення прийнятих байт менше 4, то кожний наступний байт записується в масив. Після прийому 4-х байт виконується їх перевірка відповідності формату пакета. У разі вдалої перевірки на порт мікроконтролера виводиться прийнятий байт і на світло діодах відображається стан кнопок ведучого мікроконтролера. Також формується пакет-відповідь у ведучий мікроконтролер, що підтверджує прийом даних.

3.3.5. Сполучення мікроконтролера з комп'ютером

На рис. 3.31 представлена схема мікропроцесорної системи контролю температури.

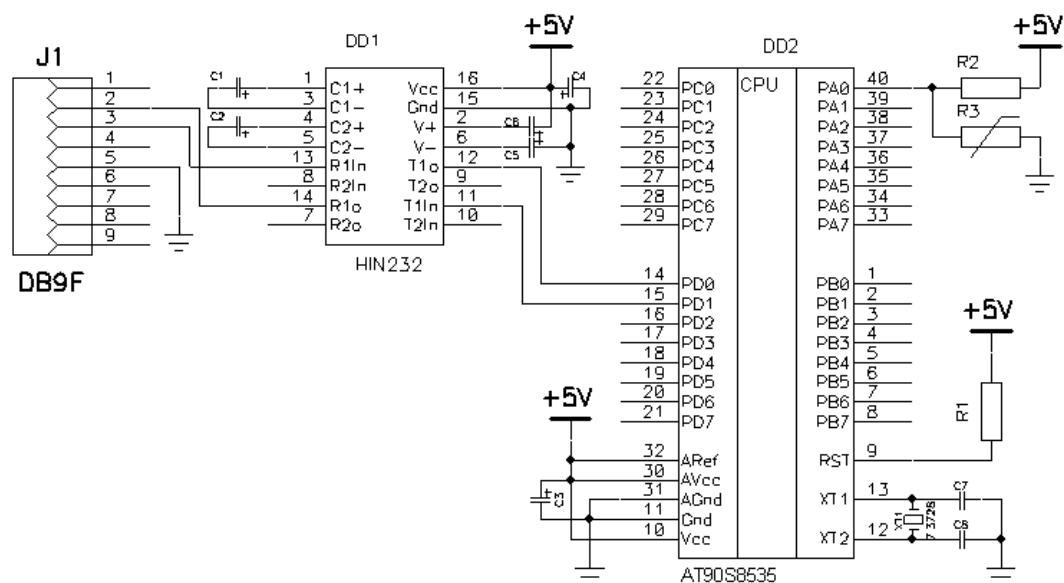


Рис. 3.31.

Для узгодження інтерфейсів COM-порта та мікроконтролера використана мікросхема перетворювача рівнів HIN232 (MAX232). Для її роботи необхідно підключити 5 зовнішніх елементів – конденсаторів ємністю 1 мкФ. Лінії Rx і Tx з'єднуються з виводами Tx і Rx мікроконтролера. Частота кварцу обрана рівною 7,3728 МГц. На такій частоті роботи мікроконтролера, при використанні вбудованого UART, не виникає помилка кадровання. Ємності конденсаторів C7 і C8 рівні 27 пФ, C3 – 1 мкф. Для роботи мікроконтролера необхідно встановити рівень лог.1 на вході RST, тому цей вхід підтягнутий до лінії живлення через опір 4,3 кОм. Опорна напруга аналого-цифрового перетворювача дорівнює 5В.

Приклад 3.16. Написати програму для мікроконтролера ATМega8535, що виконує оцифровування аналогового сигналу (з датчика температури), що подається на вхід PA0. Отримане значення перетворити до 8 біт і відправити в комп'ютер. Перетворення АЦП починається по прийому команди 0xBC.

Текст програми приводиться нижче.

```
#include <iom8535v.h>    // підключення заголовочних файлів
#include <macros.h>
void init_device(void)   // функція ініціалізації контролера
{
    DDRD=0b00000010;     // налаштування другого виводу порту D на вихід
    ADCSRA=0x86;          // налаштування АЦП; частота роботи – 125 кГц;
                          // перетворення виконується за 14 тактів
    ADMUX=0;              // канал мультиплексора АЦП - 0
    UCSRA=0x00;
    UCSRB=0x98;           // налаштування UART – дозволена робота приймача
                          // і передавача, а також переривання по прийому даних
    UCSRC=0x86;
    UBRRH=0x00;
```

```

UBRRL=0x30;          // 9600 біт/із при 7,3728 МГц
SEI();               // дозвіл переривань
}
// оброблювач переривання по прийому даних
#pragma interrupt_handler uart0_rx_isr:12
void uart0_rx_isr(void)
{
    char rc;          // додаткові змінні
    char data;
    rc=UDR;           // зчитування прийнятого байта даних
    if(rc==0xBC)      // якщо прийнято команду 0xBC
    {
        data=read_adc(0); // виклик функції перетворення АЦП
        UDR=data;        // передача байта в комп'ютер
    }
}

// функція перетворення АЦП
unsigned char read_adc(char adc_input)
{
    unsigned char adcw; // змінна для зберігання даних
    char a;             // додаткові змінні
    char b;
    ADMUX=adc_input;    // установка каналу мультиплексора
    ADCSRA|=0x40;        // початок перетворення АЦП
    while(!(ADCSRA&(1<<ADIF))); // очікування завершення перетворення
    ADCSRA&=~0x10;      // скидання прапора завершення перетворення
    a=ADCL;             // зчитування молодшого байта
    b=ADCH;             // зчитування старшого байта
    adcw=b<<6;          // формування 8-бітного значення
    a=a>>2;             // 2 молодших біти відкидаються
    adcw=adcw+a;
    return adcw;        // вихід з функції
}

void main(void)        // головна функція
{
    init_device();      // виклик функції ініціалізації контролера
    while(1);           // вічний цикл – очікується переривання
}

```

Весь алгоритм роботи програми реалізований в оброблювачі переривання по прийому даних. Отримане значення порівнюється зі значенням 0xBC (початок перетворення). Якщо умова виконується, то викликається функція `read_adc(0)`, після виконання якої повертається значення, пропорційне амплітуді аналогового сигналу на вході. Параметр функції – номер каналу АЦП. У наведеному прикладі – це 0 канал. Запуск перетворення АЦП виконується установкою біта ADSC. Потім виконання програми припиняється доти, поки не встановиться біт ADIF у регістрі стану. Це говорить про те, що перетворення завершилося і дані можуть бути зчитані з регістрів. Розрядність АЦП у мікроконтролера 10 біт. Тому необхідно привести 10-бітне значення до 8 бітного. У прикладі це

виконано відкиданням молодших 2 біт. Отримане значення записується в регістр UDR, після чого воно відразу ж передається в комп'ютер.

Приклад 3.17. Написати програму для мікроконтролера ATmega128, що відправляє в комп'ютер деякий символ.

```
#include "iom128v.h"
#include "macros.h"

void init_cpu()
{
    DDRC=0b00000111;    // керування дешифратором шини даних
    DDRA=0xFF;           // шина даних
    UCSR0C = 0x06;
    UBRRL = 0x33;        // швидкість передачі даних 9600 біт/с
    UCSR0B = 0x18;       // дозвіл роботи приймача і передавача
}

void delay(int ms)
{
    int c1, c2;
    for(c1=0;c1<ms;c1++)
        for(c2=0;c2<970;c2++) NOP();
}

void main()
{
    init_cpu();           // виклик функції ініціалізації мікроконтролера
    while(1)
    {
        UDR0='A';         // відправлення символу в комп'ютер
        delay(500);       // формування часової затримки
    }
}
```

В функції `init_cpu()` виконується налаштування UART мікроконтролера – дозволяється робота приймача та передавача, а також встановлюється швидкість передачі даних. У циклі в регістр UDR0 записується ASCII-код символу А, який передається в комп'ютер. За допомогою програми Terminal можна побачити прийняті дані, які в даному прикладі відправляються з інтервалом 500 мс.

Приклад 3.18. Написати програму для мікроконтролера ATmega128, що відправляє в комп'ютер визначене слово (масив символів).

```
#include "iom128v.h"
#include "macros.h"
#include "string.h"

void init_cpu()
{
    DDRC=0b00000111;    // керування дешифратором шини даних
```

```

DDRA=0xFF;           // шина даних
UCSR0C = 0x06;
UBRR0L = 0x33;       // швидкість передачі даних 9600 біт/с
UCSR0B = 0x18;       // дозвіл роботи приймача й передавача
}

void delay(int ms)
{
    int c1, c2;
    for(c1=0;c1<ms;c1++)
        for(c2=0;c2<970;c2++) NOP();
}

void main()
{
    char txt[]="Alexander "; // слово для передачі
    char i;                  // додаткова змінна
    init_cpu();              // виклик функції ініціалізації мікроконтролера
    while(1)
    {
        for(i=0;i<strlen(txt);i++)
        {
            UDR0=txt[i];           // відправлення символу
            while(!(UCSR0A&(1<<UDRE0))); // очікування завершення передачі
        }
        delay(500);               // формування часової затримки
    }
}

```

Як і в попередніх прикладах, в функції `init_cpu()` виконується налаштування UART мікроконтролера – дозволяється робота приймача та передавача, а також встановлюється швидкість передачі даних. В циклі відбувається передача даних в комп'ютер шляхом запису в регістр UDR0 кожного символу по чергові. Необхідно контролювати завершення передачі даних в комп'ютер і лише після цього відправляти наступний байт даних. Це можна зробити шляхом контролю стану біта UDRE0 регістра UCSR0A, який встановлюється коли регістр даних пустий, і в нього можна записати наступний байт для передачі. Після передачі всього слова формується затримка 500 мс.

Приклад 3.19. Написати програму для мікроконтролера ATmega128, що відправляє в комп'ютер ASCII-код номера натиснутої кнопки.

```

#include "iom128v.h"
#include "macros.h"

void delay(int ms)           // функція формування часової затримки
{
    int c1, c2;              // оголошення додаткових змінних
    for(c1=0;c1<ms;c1++)

```

```
for(c2=0;c2<970;c2++) NOP();    // у циклі виконується "порожня" операція
}

char scan_buttons()             // функція зчитування стану із кнопок
{
    char t, d=255;
    DDRA=0;                     // порт А налаштований на вхід для зчитування даних із кнопок
    PORTD=0x80;                 // уст. лог. 1 на виводі PD7 для керування регістром клавіатури
    t=PINA;                     // зчитування значення із шини даних
    PORTD=0;                     // переведення регістра клавіатури у третій стан
    DDRA=0xFF;                  // налаштування порту А на вихід
    // перетворення даних, отриманих із кнопок для зручності використання
    if((t&0x10)==0) d&=~0x80;
    if((t&0x20)==0) d&=~0x40;
    if((t&0x40)==0) d&=~0x20;
    if((t&0x80)==0) d&=~0x10;
    if((t&0x01)==0) d&=~0x01;
    if((t&0x02)==0) d&=~0x02;
    if((t&0x04)==0) d&=~0x04;
    if((t&0x08)==0) d&=~0x08;
    return ~d;                  // значення повертається з функції з інверсією
}

void main()
{
    char t;
    DDRC=0b00000111;           // керування дешифратором шини даних
    DDRA=0xFF;                  // порт А налаштовується на вихід
    DDRD=0x80;                  // лінія керування регістром тактових кнопок
    UCSR0C = 0x06;
    UBRRL = 0x33;               // швидкість передачі даних 9600 біт/с
    UCSR0B = 0x18;
    PORTC=4;                     // включення регістра світлодіодів
    while(1)
    {
        t=scan_buttons();       // одержання даних із кнопок
        if((t&0x01)!=0) UDR0='1';
        if((t&0x02)!=0) UDR0='2';
        if((t&0x04)!=0) UDR0='3';
        if((t&0x08)!=0) UDR0='4';
        if((t&0x10)!=0) UDR0='5';
        if((t&0x20)!=0) UDR0='6';
        if((t&0x40)!=0) UDR0='7';
        if((t&0x80)!=0) UDR0='8';
        delay(500);              // пауза 500 мс
    }
}
```

У прикладі функція `scan_buttons()` виконує зчитування даних з кнопок та повертає число, в якому значення кожного біта відображає стан відповідної кнопки. В основному циклі програми аналізується стан кожного біта, і в комп'ютер відправляється ASCII-код символу від 1 до 8.

Приклад 3.20. Написати програму керування станом світлодіодних індикаторів. Номер світлодіоду, що загоряється, відповідає значенню, прийнятому з комп'ютера.

```
#include "iom128v.h"
#include "macros.h"
void init_cpu()
{
    DDRC=0b00000111;    // керування дешифратором шини даних
    DDRA=0xFF;           // шина даних
    UCSR0C = 0x06;
    UBRRL = 0x33;        // швидкість передачі даних 9600 біт/с
    UCSR0B = 0x18;        // дозвіл роботи приймача й передавача
}

void main()
{
    char d;               // додаткова змінна
    init_cpu();           // виклик функції ініціалізації мікроконтролера
    PORTC=4;              // вибір регістра світлодіодів
    PORTA=255;            // виключаємо всі світлодіоди
    while(1)
    {
        while(!(UCSR0A&(1<<RXC0))); // очікування даних з комп'ютера
        d=UDR0;           // зчитування прийнятого значення в змінну
        if(d=='0') PORTA=255; // залежно від прийнятого значення
        if(d=='1') PORTA=~0x01; // запалюється відповідний світлодіод - H10
        if(d=='2') PORTA=~0x02; // H11
        if(d=='3') PORTA=~0x04; // H12
        if(d=='4') PORTA=~0x08; // H13
        if(d=='5') PORTA=~0x10; // H14
        if(d=='6') PORTA=~0x20; // H15
        if(d=='7') PORTA=~0x40; // H16
        if(d=='8') PORTA=~0x80; // H17
    }
}
```

В основному циклі програми відбувається аналіз біта `RXC0` регістра `UCSR0A`, який встановлюється в 1 коли прийнятий байт даних, і він може бути зчитаний з регістра `UDR0`. Далі відбувається перевірка прийнятого байта з ASCII-кодом символів 0-8. Якщо умова виконується, то в порт А мікроконтролера виводиться значення, яке змінює стан відповідного виводу, що можна побачити за допомогою світлодіодів.

Приклад 3.21. Розробити принципову схему пристрою та написати програму для мікроконтролера ATmega8, що виконує функції керування станом 3-х виводів мікроконтролера та передачу даних про стан 3-х кнопок, що підключені до мікроконтролера, в комп'ютер. Розробити власний протокол обміну даними.

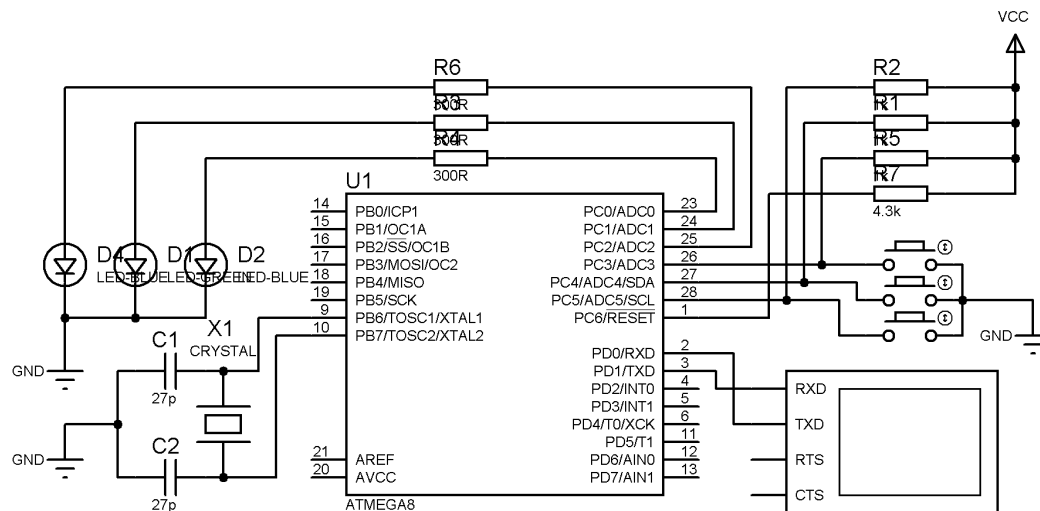


Рис. 3.32.

```
#include "iom8v.h"          // підключення заголовочних файлів
#include "macros.h"

// присвоєння виводам альтернативних імен
#define SET_LED1() PORTC|=0x01
#define SET_LED2() PORTC|=0x02
#define RST_LED1() PORTC&=~0x01
#define RST_LED2() PORTC&=~0x02
#define SET_LED3() PORTC|=0x04
#define RST_LED3() PORTC&=~0x04

void init_cpu()             // функція ініціалізації мікроконтролера
{
    DDRC=0x07;
    // налаштування UART
    UCSRC = 0x80;
    UBRRL = 0x33;           // швидкість передачі 9600 біт/с при 8 МГц
    UBRRH = 0x00;
    UCSRB = 0x18;           // дозвіл роботи приймача та передавача
    UCSRC=0x86;             // 8 біт даних, 1 стоповий біт
}

void main()                 // голова функція програми
{
    char d;                 // додаткова змінна
    init_cpu();             // виклик функції ініціалізації контролера
    while(1)
    {
        while(!(UCSRA&(1<<RXC))); // очікування приймання символу
        d=UDR;              // зчитування прийнятого символу з регістра даних
        // перевірка прийнятого символу
        if(d=='A') SET_LED1(); // загоряється світлодіод
        else if(d=='a') RST_LED1(); // вимикається світлодіод
        else if(d=='B') SET_LED2();
        else if(d=='b') RST_LED2();
    }
}
```



```

else if(d=='C') SET_LED3();
else if(d=='c') RST_LED3();
else if(d=='q')          // запит на передачу стану кнопок
{
d=PINC;                  // зчитування стану порту
d=~(d>>3);               // форматування
d=d&0x07;                // виділення 3-х молодших біт
d=d+48;                  // формування ASCII-коду для зручності
UDR=d;                   // передача даних
}
}
}

```

Після налаштування портів мікроконтролера та UART, відбувається очікування приймання даних. Виконання програми зупиняється доти, доки не буде встановлений біт RXC регістра UCSRA. Якщо цей біт встановлений, то це означає, що був прийнятий байт даних і він може бути зчитаний з регістра UDR. Далі аналізується значення прийнятого байта. Якщо прийнятий ASCII-код одного з символів A, B, C, то загоряється відповідний світлодіод. Якщо прийнятий ASCII-код одного з символів a, b, c, то відповідний світлодіод погашений. При надходженні ASCII-кода символу q відбувається форматування байту даних, в якому передається номер натисненої кнопки.

3.4. Аналого-цифровий перетворювач

ATmega128 містить 10-розрядний АЦП послідовного наближення. АЦП пов'язаний з 8-канальним аналоговим мультиплексором, 8 однополярних входів якого з'єднані з лініями порту

Відмінні риси:

- 10-розрядне представлення результату перетворення;
- Інтегральна нелінійність 0,5 мол. розряду;
- Абсолютна похибка ± 2 мол. розряду;
- Час перетворення 65 – 260 мкс.;
- Частота до 15 тис. перетворень в секунду;
- 8 мультиплексованих однополярних входів;
- 7 диференціальних вхідних каналів;
- 2 диференціальних вхідних канали з опціональним підсиленням на 10 і 200;
- Представлення результату вирівнюванням в 16-розрядному слові ліворуч або праворуч;
- Діапазон вхідної напруги АЦП 0...VCC;
- Внутрішнє джерело опорної напруги на 2,56 В;
- Режими одиночного перетворення і автоматичного перезапуску;
- Переривання по завершенню перетворення АЦП;
- Механізм фільтрації шумів у режимі очікування.

Два диференціальних входи (ADC1, ADC0 і ADC3, ADC2) містять каскад зі східчастим програмованим посиленням: 0 дБ (1x), 20 дБ (10x),

або 46 дБ (200х). Сім диференціальних аналогових каналів використовують загальний інвертуючий вхід (ADC1), а всі інші входи АЦП виконують функцію неінвертуючих входів.

Функціональна схема АЦП показана на рис. 3.33. АЦП має окремий вивід живлення AVCC (аналогове живлення). Напруга на виводі AVCC не повинна відрізнятись більш ніж на $\pm 0,3$ від VCC.

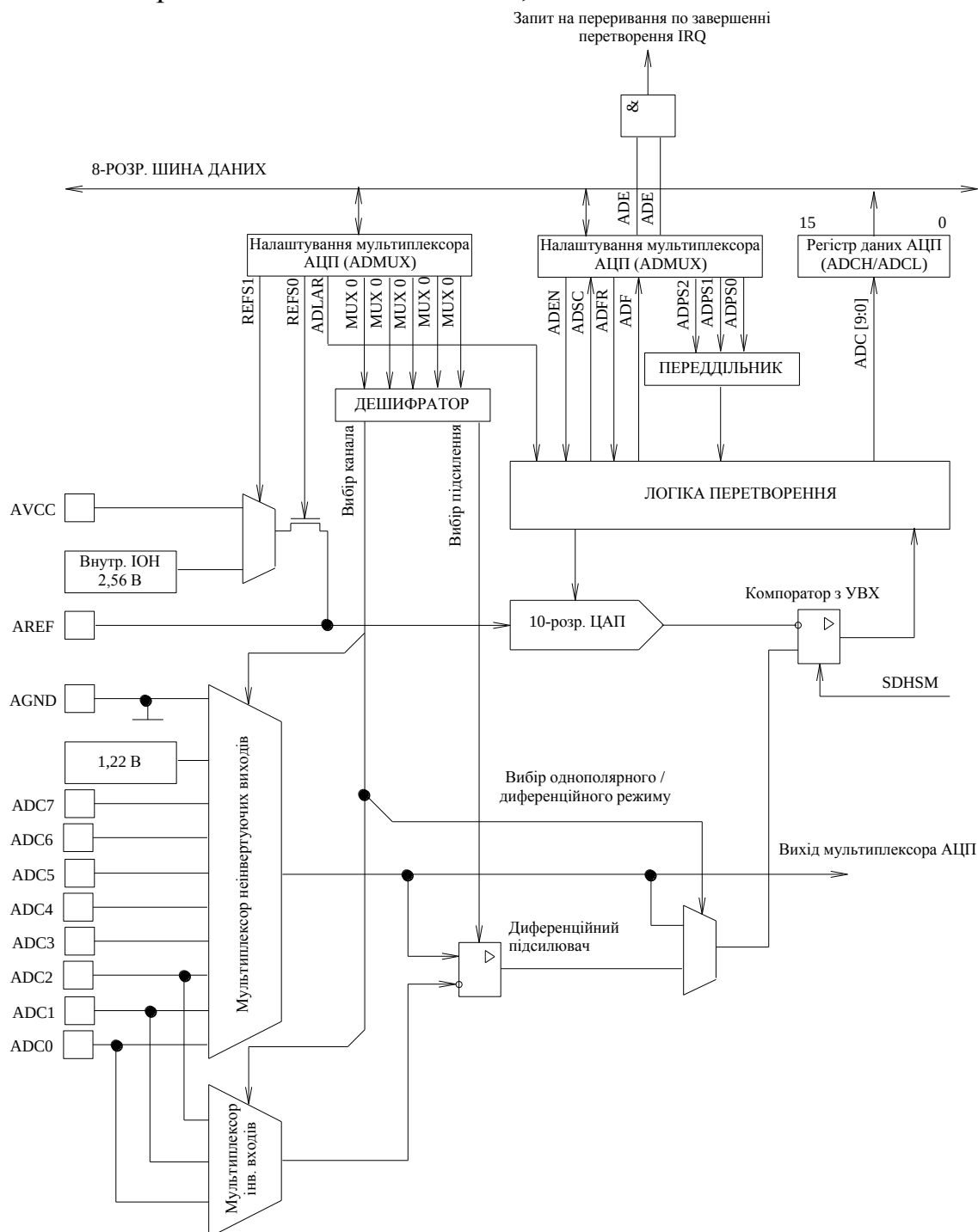


Рис. 3.33.

АЦП перетворює вхідну аналогову напругу в 10-розрядний код методом послідовних наближень. Мінімальне значення відповідає рівню GND, а максимальне рівню AREF мінус 1 молодший розряд. До виводу

AREF опціонально може бути підключена напруга AVCC або внутрішнє джерело опорної напруги (ДОН) на 1,22 В шляхом запису відповідних значень у біти REFSn у регістр ADMUX. Незважаючи на те, що ДОН на 2,56 В розташоване всередині мікроконтролера, до його виходу може бути підключений конденсатор для зниження чутливості до шумів, оскільки він пов'язаний з виводом AREF.

Канал аналогового вводу і каскад диференціального підсилення вибираються шляхом запису біт MUX у регістр ADMUX. У якості однополярного аналогового входу АЦП може бути обраний один із входів ADC0...ADC7, а також GND, і вихід фіксованого джерела опорної напруги 1,22 В. У режимі диференціального вводу передбачена можливість вибору інвертуючих та неінвертуючих входів до диференціального підсилювача.

Якщо обрано диференціальний режим аналогового введення, то диференціальний підсилювач буде підсилювати різницю напруг між обраною парою входів на заданий коефіцієнт підсилення. Підсилене в такий спосіб значення надходить на аналоговий вхід АЦП. Якщо вибирається однополярний режим аналогового вводу, то каскад підсилення пропускається.

Робота АЦП дозволяється шляхом установки біта ADEN в ADCSRA. Вибір опорного джерела й каналу перетворення не можливо виконати до установки ADEN. Якщо ADEN = 0, то АЦП не споживає струм, тому, при переведенні в економічні режими сну рекомендується попередньо відключити АЦП.

АЦП генерує 10-розрядний результат, що міститься в парі регістрів даних АЦП ADCH й ADCL. За замовчуванням результат перетворення розташовується в молодших 10-ти розрядах 16-розрядного слова (вирівнювання праворуч), але може бути опціонально розміщений у старших 10-ти розрядах (вирівнювання ліворуч) шляхом установки біта ADLAR у регістрі ADMUX.

АЦП генерує власний запит на переривання по завершенні перетворення. Якщо між зчитуванням регістрів ADCH та ADCL заборонений доступ до даних для АЦП, то переривання виникне, навіть якщо результат перетворення буде загублений.

Якщо потрібна максимальна розв'язна здатність (10 розрядів), то частота на вході схеми послідовного наближення повинна бути в діапазоні 50...200 кГц. Якщо достатня розв'язна здатність менша 10 розрядів, але потрібна більш висока частота перетворення, то частота на вході АЦП може бути встановлена понад 200 кГц.

По завершенні перетворення (ADIF = 1) результат може бути зчитаний з пари регістрів результату перетворення АЦП (ADCL, ADCH).

Для однополярного перетворення:

$$ADC = \frac{V_{IN} \cdot 1024}{V_{REF}}$$

де $V_{\text{вх}}$ – рівень напруги на підключеному до АЦП вході, V_{ref} – напруга обраного опорного джерела.

3.4.1. Регістри керування АЦП

Регістр керування мультиплексором АЦП – ADMUX

Розряд	7	6	5	4	3	2	1	0
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0
Зчитування /запис	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.
Початкове значення	0	0	0	0	0	0	0	0

Розряд 7:6 – REFS1:0: Біти вибору джерела опорної напруги.

Дані біти визначають, яка напруга буде використовуватися в якості опорної для АЦП. Якщо змінити значення даних біт у процесі перетворення, то нові установки набудуть чинності тільки по завершенні поточного перетворення (тобто коли встановиться біт ADIF у регістрі ADCSRA). Внутрішнє джерело опорної напруги можна не використовувати, якщо до виводу AREF підключене зовнішнє опорне джерело.

REFS1	REFS0	Опорне джерело
0	0	AREF, внутрішнє джерело відключене
0	1	AVCC із зовнішнім конденсатором на виводі AREF
1	0	Зарезервовано
1	1	Внутрішнє джерело опорної напруги 2,56В з зовнішнім конденсатором на виводі AREF

Розряд 5 – ADLAR: Біт керування представленням результату перетворення.

Біт ADLAR впливає на представлення результату перетворення в парі регістрів результату перетворення АЦП. Якщо ADLAR = 1, то результат перетворення буде вирівняний ліворуч, у протилежному випадку – праворуч. Дія біта ADLAR відбувається відразу після зміни, незалежно від перетворення, що виконується паралельно.

Розряд 4:0 – MUX4:0: Біти вибору аналогового каналу й коефіцієнта підсилення.

Дані біти визначають, які з наявних аналогових входів підключаються до АЦП. Крім того, з їхньою допомогою можна вибрати коефіцієнт підсилення для диференціальних каналів. Якщо значення біт змінити в процесі перетворення, то механізм їхньої дії набуде чинності тільки після завершення поточного перетворення (після установки біта ADIF у регістрі ADCSRA).

Регістр керування й статусу АЦП – ADCSRA

Розряд	7	6	5	4	3	2	1	0
	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Зчитування /запис	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.
Початкове значення	0	0	0	0	0	0	0	0

Розряд 7 – ADEN: Дозвіл роботи АЦП.

Запис у даний біт лог. 1 дозволяє роботу АЦП. Якщо в даний біт записати лог 0, то АЦП відключається, навіть якщо він перебував у процесі перетворення.

Розряд 6 – ADSC: Запуск перетворення АЦП.

У режимі одиночного перетворення установка даного біта ініціює старт кожного перетворення. У режимі автоматичного перезапуску установкою цього біта ініціюється тільки перше перетворення, а всі інші виконуються автоматично. Перше перетворення після дозволу роботи АЦП, ініційоване бітом ADSC, виконується по розширеному алгоритму і триває 25 тактів синхронізації АЦП, замість звичайних 13 тактів. Це пов'язане з необхідністю ініціалізації АЦП.

У процесі перетворення при опитуванні біта ADSC повертається лог. 1, а по завершенні перетворення – лог. 0. Запис лог. 0 у даний біт не передбачене і не виконує ніякої дії.

Розряд 5 – ADFR: Вибір режиму автоматичного перезапуску АЦП.

Якщо в даний біт записати лог. 1, то АЦП перейде в режим автоматичного перезапуску. У цьому режимі АЦП автоматично виконує перетворення і модифікує регістри результату перетворення через фіксовані проміжки часу. Запис лог. 0 у цей біт припиняє роботу в даному режимі.

Розряд 4 – ADIF: Прапор переривання АЦП.

Даний прапор установлюється після завершення перетворення АЦП і відновлення регістрів даних. Якщо встановлені біти ADIE й I (регістр SREG), то відбувається переривання по завершенні перетворення. Прапор ADIF скидається апаратно при переході на відповідний вектор переривання. Альтернативно прапор ADIF скидається шляхом запису лог. 1 у нього. При виконанні команди "зчитування-модифікація-запис" з регістром ADCSRA очікуване переривання може бути відключено.

Розряд 3 – ADIE: Дозвіл переривання АЦП.

Після запису лог. 1 у цей біт, за умови, що встановлено біт I у регістрі SREG, дозволяється переривання по завершенні перетворення АЦП.

Розряди 2:0 – ADPS2:0: Біти керування дільником АЦП.

Дані біти визначають на яке значення тактова частота мікроконтролера буде відрізнятися від частоти вхідної синхронізації АЦП.

Регістри даних АЦП – ADCL й ADCH

ADLAR = 0:

Розряд	15	14	13	12	11	10	9	8
	-	-	-	-	-	-	ADC9	ADC8
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0
	7	6	5	4	3	2	1	0
Зчитування /запис	Зчит.	Зчит.	Зчит.	Зчит.	Зчит.	Зчит.	Зчит.	Зчит.
	Зчит.	Зчит.	Зчит.	Зчит.	Зчит.	Зчит.	Зчит.	Зчит.
Початкове значення	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0

ADLAR = 1:

Розряд	15	14	13	12	11	10	9	8
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2
	ADC1	ADC0	-	-	-	-	-	-
	7	6	5	4	3	2	1	0
Зчитування /запис	Зчит.	Зчит.	Зчит.	Зчит.	Зчит.	Зчит.	Зчит.	Зчит.
	Зчит.	Зчит.	Зчит.	Зчит.	Зчит.	Зчит.	Зчит.	Зчит.
Початкове значення	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0

По завершенні перетворення результат міститься в цих двох регістрах. При використанні диференціального режиму перетворення результат представляється в коді двійкового доповнення.

Якщо виконано зчитування ADCL, то доступ до цих регістрів для АЦП буде заблокований (тобто АЦП не зможе надалі модифікувати результат перетворення), поки не буде зчитаний регістр ADCH.

Лівосторонній формат представлення результату зручно використовувати, якщо достатньо 8 розрядів. У цьому випадку 8-розрядний результат зберігається в регістрі ADCH, і зчитування регістра ADCL можна не виконувати. При форматі вирівнювання праворуч необхідно спочатку зчитати ADCL, а потім ADCH.

ADC9:0: Результат перетворення АЦП.

Дані біти представляють результат перетворення.

3.4.2. Приклади програмування АЦП

Для демонстрації прикладів роботи з АЦП використовується схема, представлена на рис. 3.34.

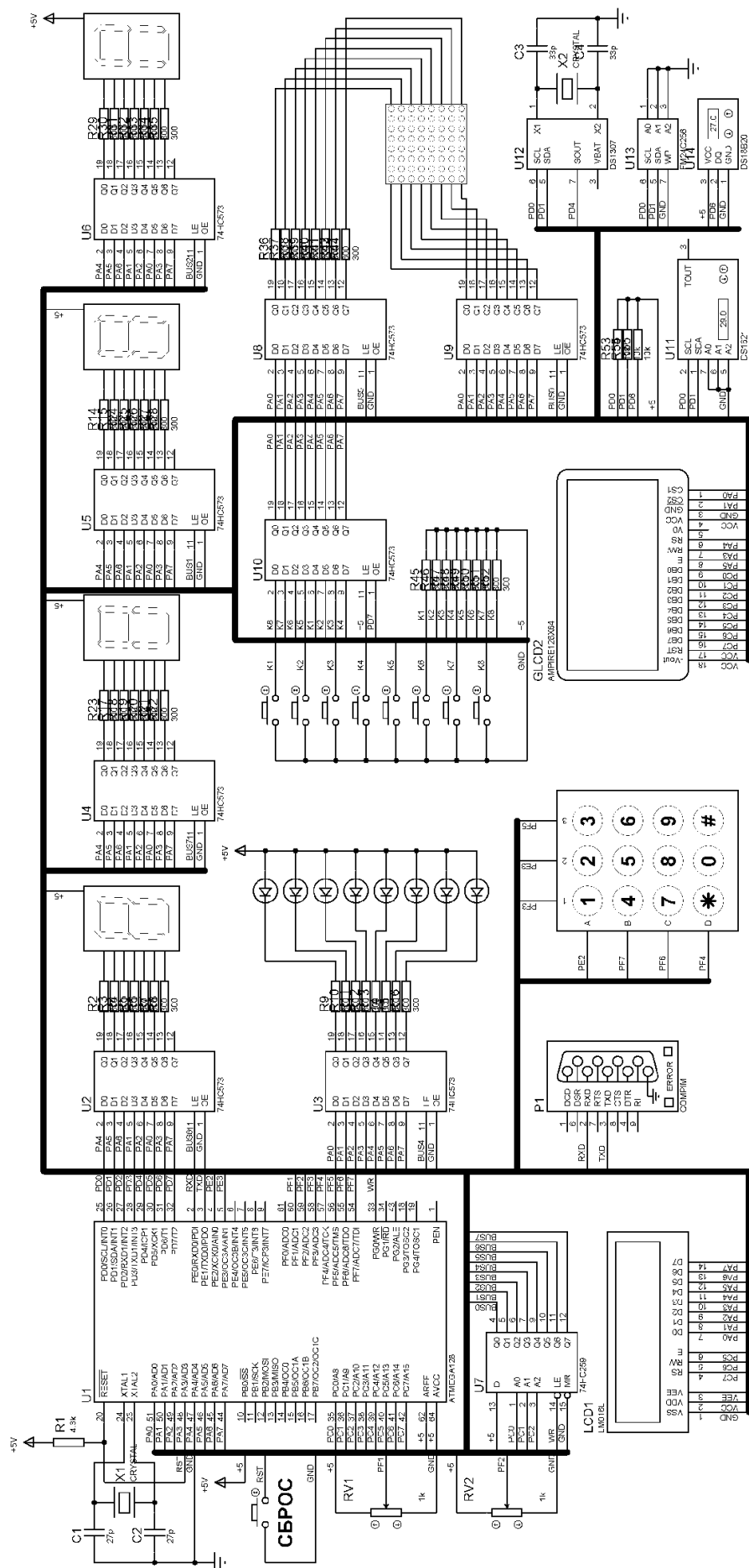


Рис. 3.34.

Приклад 3.22. Програма, що виконує вивід значення, отриманого з АЦП, на семисегментні індикатори.

```
#include "iom128v.h"
#include "macros.h"

void delay(int ms)           // функція часової затримки
{
    int c1, c2;              // оголошення додаткових змінних
    for(c1=0;c1<ms;c1++)
        for(c2=0;c2<970;c2++) NOP(); // виконання «пустої» операції
}

unsigned int read_adc(char ch) // функція зчитування даних з АЦП
{
    char dh,dl;              // додаткові змінні
    int res;
    ADMUX=ch|0x40;           // вибір каналу мультиплексора
    ADCSRA=(1<<ADSC);        // запуск перетворення АЦП
    while(!(ADCSRA&(1<<ADIF))); // очікування завершення перетворення
    dl=ADCL;                  // зчитування результату з регістрів у змінні
    dh=ADCH;
    res=dh;                   // приведення результату до 10-бітного
    res=res<<8;
    res=res+dl;
    return res;               // повернення 10-бітного значення із функції
}

void main()                  // головна функція програми
{
    char seg[10]={0b01110111, 0b01100000, 0b00111110, 0b01111010, 0b01101001,
        0b01011011, 0b01011111, 0b01110000, 0b01111111, 0b01111011};
    unsigned int i;

    DDRC=0b00000111;         // керування дешифратором шини даних
    DDRA=0xFF;                // шина даних
    // дозвіл роботи АЦП, налаштування дільника
    ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
    while(1)
    {
        i=read_adc(1);
        PORTC=6;              // включення індикатора Н6
        PORTA=~seg[i/1000];    // вивід у порт значення тисяч
        PORTC=7;              // включення індикатора Н7
        PORTA=~seg[(i%1000)/100]; // вивід у порт значення сотень
        PORTC=1;              // включення індикатора Н8
        PORTA=~seg[((i%1000)%100)/10]; // вивід у порт значення десятків
        PORTC=2;              // включення індикатора Н9
        PORTA=~seg[((i%1000)%100)%10]; // вивід у порт значення одиниць
        delay(400);
    }
}
```


Для зручності отримання даних з АЦП в програмі реалізована функція `read_adc()`, параметром якої виступає номер каналу АЦП. Результатом перетворення є 10-бітне значення, яке повертається вказаною функцією після завершення перетворення. При виклику цієї функції виконується установка каналу АЦП шляхом запису в регістр `ADMUX` переданого в функцію значення. Після цього виконується запуск перетворення установкою біта `ADSC` в регістрі `ADCSRA`. Виконання програми припиняється доти, доки біт `ADIF` регістра `ADCSRA` не прийме значення 1. Після цього дані можуть бути зчитані з регістрів `ADCL` та `ADCH`. В основному циклі програми з отриманого значення виділяються тисячі, сотні, десятки та одиниці, і виводяться на відповідні семисегментні індикатори.

Приклад 3.23. Реалізувати шкальний індикатор на світлодіодах, який виконує індикацію рівня напруги на вході АЦП.

```
#include "iom128v.h"
#include "macros.h"

void delay(int ms)           // функція часової затримки
{
    int c1, c2;              // оголошення додаткових змінних
    for(c1=0;c1<ms;c1++)
        for(c2=0;c2<970;c2++) NOP(); // виконання «пустої» операції
}

unsigned int read_adc(char ch) // функція зчитування даних з АЦП
{
    char dh,dl;              // додаткові змінні
    int res;
    ADMUX=ch|0x40;           // вибір каналу мультиплексора
    ADCSRA|=(1<<ADSC);       // запуск перетворення АЦП
    while(!(ADCSRA&(1<<ADIF))); // очікування завершення перетворення
    dl=ADCL;                  // зчитування результату з регістрів у змінні
    dh=ADCH;
    res=dh;                   // приведення результату до 10-бітного
    res=res<<8;
    res=res+dl;
    return res;               // повернення 10-бітного значення із функції
}

void main()
{
    char arr[9]={0,0x01,0x03,0x07,0x0F,0x1F,0x3F,0x7F,0xFF};
    unsigned int i;
    DDRC=0b00000111;        // керування дешифратором шини даних
    DDRA=0xFF;               // шина даних
    ADCSRA = 0x87;           // дозвіл роботи АЦП, налаштування дільника
    PORTC=4;                 // включення регістра світлодіодів
    while(1)
```

```
{
    i=read_adc(1);           // зчитування значення з каналу 1 АЦП
    PORTA=~arr[i/114];       // вивід в порт елементу масиву
    delay(100);              // часова затримка 100 мс
}
}
```

Максимальне значення, яке зчитується з АЦП, дорівнює $1023 (2^{10})$. В масиві `arr[]` зберігаються значення, які будуть виведені в порт. При виводі в порт елементу масива будуть увімкнені світло діоди, кількість яких відповідає індексу елемента масива. Оскільки на стенді є 8 світлодіодів, то для того, щоб не вийти за межі масиву, отримане значення з АЦП ділиться на 114. Таким чином, максимальний індекс елемента масива, що може бути виведений в порт, дорівнює 8 (всі світло діоди увімкнені).

Приклад 3.24. Програма, що виконує зчитування значення з АЦП і передачу його в комп'ютер по UART.

```
#include "iom128v.h"
#include "macros.h"
void init_cpu()
{
    DDRC=0b00000111;        // керування дешифратором шини даних
    DDRA=0xFF;               // шина даних
    ADCSRA = 0x87;           // дозвіл роботи АЦП, налаштування дільника
    UCSR0C = 0x06;
    UBRRL = 0x33;            // швидкість передачі даних 9600 біт/с
    UCSR0B = 0x18;           // дозвіл роботи приймача й передавача
}

void delay(int ms)          // функція часової затримки
{
    int c1, c2;              // оголошення додаткових змінних
    for(c1=0;c1<ms;c1++)
        for(c2=0;c2<970;c2++) NOP(); // виконання «пустої» операції
}

unsigned int read_adc(char ch) // функція зчитування даних з АЦП
{
    char dh,dl;              // додаткові змінні
    int res;
    ADMUX=ch|0x40;           // вибір каналу мультиплексора
    ADCSRA|=(1<<ADSC);       // запуск перетворення АЦП
    while(!(ADCSRA&(1<<ADIF))); // очікування завершення перетворення
    dl=ADCL;                  // зчитування результату з регістрів у змінні
    dh=ADCH;
    res=dh;                   // приведення результату до 10-бітного
    res=res<<8;
    res=res+dl;
    return res;               // повернення 10-бітного значення із функції
}
```

```

void main()
{
    unsigned int i;           // додаткова змінна
    init_cpu();              // виклик функції ініціалізації контролера
    while(1)
    {
        i=read_adc(1);       // зчитування даних з каналу АЦП
        UDR0=(i>>2);        // передача 8-бітного значення в комп'ютер
        delay(50);          // затримка 50 мс
    }
}

```

Як було сказано вище, розрядність АЦП дорівнює 10. Тому для передачі даних в комп'ютер виконується зсув отриманого з АЦП значення на 2 розряди ліворуч, після чого виконується передача в комп'ютер. За допомогою програми Terminal можна побачити у вигляді осцилограми прийняті дані. Якщо необхідно передавати 10-бітне значення, то необхідно виконати передачу 2-х байт. В першому байті передати 2 старші біти результату перетворення, а в другому байті – 8 молодших бітів. В такому випадку клієнтська програма повинна виконати форматування прийнятих байт в одну змінну для подальшого використання.

Приклад 3.25. Програма реалізує наступний алгоритм: напругою на входах АЦП (канали 1 та 2) виконується керування положенням сегменту, що світиться, на матричному індикаторі.

```

#include "iom128v.h"
#include "macros.h"

void init_cpu()
{
    DDRC=0b00000111;        // керування дешифратором шини даних
    DDRA=0xFF;               // шина даних
    DDRF=0;
    ADCSRA = 0x87;
}

void delay(int ms)         // функція часової затримки
{
    int c1, c2;              // оголошення додаткових змінних
    for(c1=0;c1<ms;c1++)
        for(c2=0;c2<970;c2++) NOP(); // виконання «пустої» операції
}

unsigned int read_adc(char ch) // функція зчитування даних з АЦП
{
    char dh,dl;              // додаткові змінні
    int res;
    ADMUX=ch|0x40;           // вибір каналу мультиплексора
    ADCSRA|=(1<<ADSC);       // запуск перетворення АЦП
}

```

```
while(!(ADCSRA&(1<<ADIF))); // очікування завершення перетворення
dl=ADCL; // зчитування результату з регістрів у змінні
dh=ADCH;
res=dh; // приведення результату до 10-бітного
res=res<<8;
res=res+dl;
return res; // повернення 10-бітного значення із функції
}

void main()
{
unsigned int r, c;
init_cpu(); // виклик функції ініціалізації контролера
while(1)
{
r=read_adc(1); // зчитування значення з каналу 1 АЦП
c=read_adc(2); // зчитування значення з каналу 2 АЦП
PORTC=0; // вибір регістра керування стовбцями індикатора
PORTA=~(1<<(r/146)); // вивід значення на шину даних
PORTC=5; // вибір регістра керування строками індикатора
PORTA=(1<<(c/146)); // вивід значення на шину даних
delay(200); // часова затримка
}
}
```

Для зручності отримання даних з АЦП в програмі реалізована функція `read_adc()`, параметром якої виступає номер каналу АЦП. Значення, що повертає функція, пропорційне вхідній напрузі на вході аналого-цифрового перетворювача. Максимальне значення, яке отримується після завершення перетворення, дорівнює $1023 (2^{10})$. Це значення ділиться на 146 для того, щоб отримати число, яке не перевищує 7. За допомогою операції зсуву ліворуч встановлюються відповідні біти регістра порта А, який керує матричним індикатором.

3.5. Аналоговий компаратор

Аналоговий компаратор порівнює рівні напруг на неінвертуючому вході AIN0 та інвертуючому вході AIN1. Якщо напруга на неінвертуючому вході AIN0 перевищує напругу на інвертуючому вході AIN1, то вихід аналогового компаратора АСО приймає одиничний стан. Вихід компаратора може бути налкштований для використання як джерело вхідного сигналу для схеми захвату фронтів таймера-лічильника 1. Крім того, компаратор може генерувати власний запит на обробку переривання. Користувач може вибрати кілька подій, по яких виникає переривання: фронт або зріз на виході компаратора, або будь-яка його зміна. Функціональна схема компаратора та пов'язаної з ним логіки представлена на рис. 3.35.

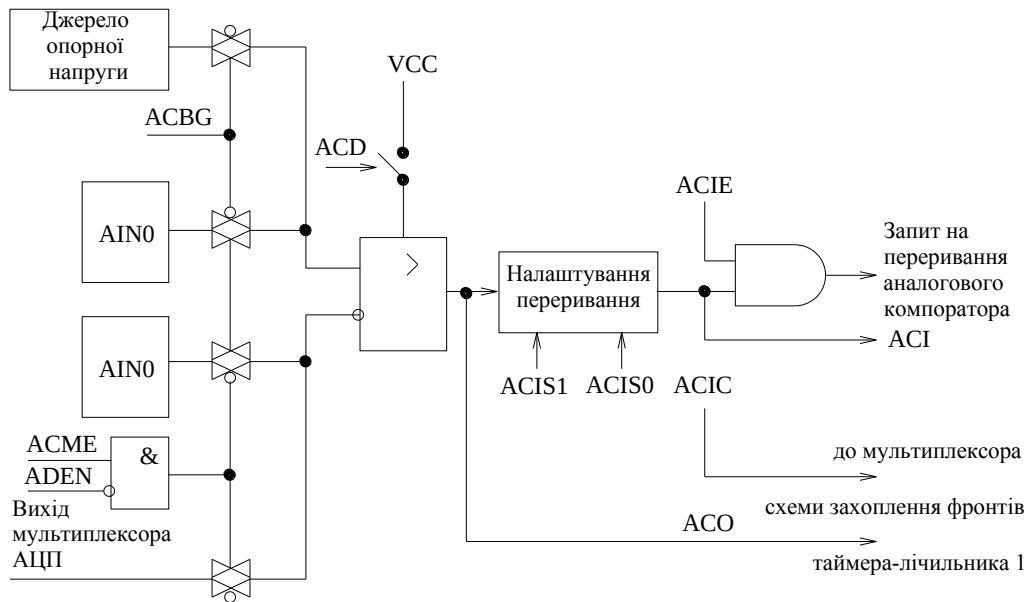


Рис. 3.35.

Регістр стану й керування аналогового компаратора – ACSR:

Розряд	7	6	5	4	3	2	1	0
	ACD	ACBG	AC0	ACI	ACIE	ACIC	ACIS1	ACIS0
Зчитування /запис	Зч. /Зап.	Зч. /Зап.	Зчит.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.
Початкове значення	0	0	x	0	0	0	0	0

Розряд 7 – ACD: Відключення аналогового компаратора.

Запис у даний розряд лог. 1 приводить до зняття живлення з аналогового компаратора. Даний розряд можна встановлювати в будь-який момент при необхідності відключення аналогового компаратора. Його використання дозволяє знизити енергоспоживання в активному режимі та режимі холостого ходу. Перед зміною біта ACD необхідно відключити переривання від аналогового компаратора шляхом скидання біта ACIE у регістрі ACSR. У протилежному випадку може виникнути переривання після зміни значення даного біта.

Розряд 6 – ACBG: Підключення джерела опорної напруги до аналогового компаратора.

Після установки даного біта до неінвертуючого входу компаратора підключається джерело опорної напруги. Після скидання даного розряду неінвертуючий вхід компаратора з'єднаний з виводом AIN0 мікроконтролера.

Розряд 5 – ACO: Вихід аналогового компаратора.

Даний біт виходу аналогового компаратора зв'язаний безпосередньо з виводом ACO через ланцюг синхронізації. Синхронізація реалізована як часова затримка на 1 – 2 машинних цикли.

Розряд 4 – ACI: Прапор переривання аналогового компаратора.

Даний розряд встановлюється апаратно, при виникненні події відповідно до налаштувань бітів ACIS1 й ACIS0. Запит на обробку переривання аналогового компаратора виконується, якщо встановлені біти ACIE та I у регістрі SREG. ACI скидається апаратно при переході на відповідний вектор обробки переривання. Альтернативно, біт ACI можна скинути програмно шляхом запису лог. 1 у даний розряд.

Розряд 3 – ACIE: Дозвіл переривання аналогового компаратора.

Якщо в даний розряд записана лог. 1 і встановлений біт I у регістрі статусу, то переривання по аналоговому компаратору активізується. Запис у даний розряд лог. 0 призводить до відключення даного переривання.

Розряд 2 – ACIC: Підключення аналогового компаратора до схеми захвата фронтів.

Установка даного розряду призводить до дозволу спільної роботи схеми захвату фронтів таймера-лічильника 1 й аналогового компаратора. У цьому випадку, вихід аналогового компаратора безпосередньо підключений до вхідного каскаду схеми захвату фронтів, дозволяючи до компаратора додати функції придушення шумів і настроювання фронтів переривання по захвату фронту таймером-лічильником 1. Після запису в даний розряд лог. 0 зв'язок між аналоговим компаратором і схемою захвату фронтів розривається. Для активізації переривання схеми захвату фронтів таймера-лічильника 1 по спрацьовуванню аналогового компаратора необхідно встановити біт TICIE1 у регістрі маски переривання таймера (TIMSK).

Розряди 1, 0 – ACIS1, ACIS0: Вибір події переривання аналогового компаратора.

Дані розряди визначають яка подія призводить до генерації запиту на переривання аналогового компаратора. Варіанти установок даних розрядів та їхнє призначення представлені в табл. 3.6.

Табл. 3.6.

ACIS1	ACIS0	Подія
0	0	Переривання при будь-якій зміні на виході компаратора
0	1	Зарезервовано
1	0	Переривання по зрізу сигналу на виході компаратора
1	1	Переривання по фронту на виході компаратора

Перед зміною біт ACIS1/ACIS0 необхідно відключити переривання від аналогового компаратора шляхом скидання біта дозволу переривання в регістрі ACSR. У протилежному випадку може виникнути переривання при зміні значень даних біт.

Є можливість використати виводи ADC7..0 як неінвертуючі входи аналогового компаратора. Для організації такого вводу використовується мультиплексор АЦП і, отже, у цьому випадку АЦП повинен бути відключений. Якщо встановлено біт дозволу підключення мультиплексора до аналогового компаратора (біт ACME в SFIOR) і виключений АЦП

(ADEN=0 у регістрі ADCSRA), то стан розрядів MUX2..0 регістра ADMUX визначають який вивід мікроконтролера підключений до неінвертуючого входу аналогового компаратора. Якщо АСМЕ скинутий або встановлений ADEN, в якості неінвертуючого входу аналогового компаратора використовується вивід мікроконтролера AIN1.

3.6. Таймери-лічильники загального призначення

Таймер-лічильник 0 – модуль багатофункціонального одноканального 8-розрядного таймера-лічильника з апаратним виходом для генерації ШІМ-сигналу й вбудованим асинхронним тактовим генератором, що оптимізований під використання годинникового кварцу (32768 Гц) для асинхронного відносно системної синхронізації тактування.

Основні відмінні риси:

- одноканальний лічильник;
- опціональний режим скидання таймера при збігу (автоматичне перезавантаження);
- широтно-імпульсна модуляція без генерації помилкових імпульсів при записі нового порога порівняння в OCR0 (подвійна буферизація) із фазовою корекцією;
- генератор частоти;
- 10-розрядний дільник тактової частоти;
- генерація переривань по переповненню й виконання умови порівняння (TOV0 й OCF0);
- можливість асинхронного тактування разом із зовнішнім кварцовим резонатором частотою 32 кГц незалежно від частоти синхронізації вводу-виводу.

16-розрядні таймери-лічильники призначені для точного задання часових інтервалів, генерації прямокутних імпульсів і виміру часових характеристик імпульсних сигналів.

3.6.1. Функціональні схеми 8- та 16-розрядних таймерів

Функціональна схема 8-розрядного таймера-лічильника представлена на рис. 3.36.

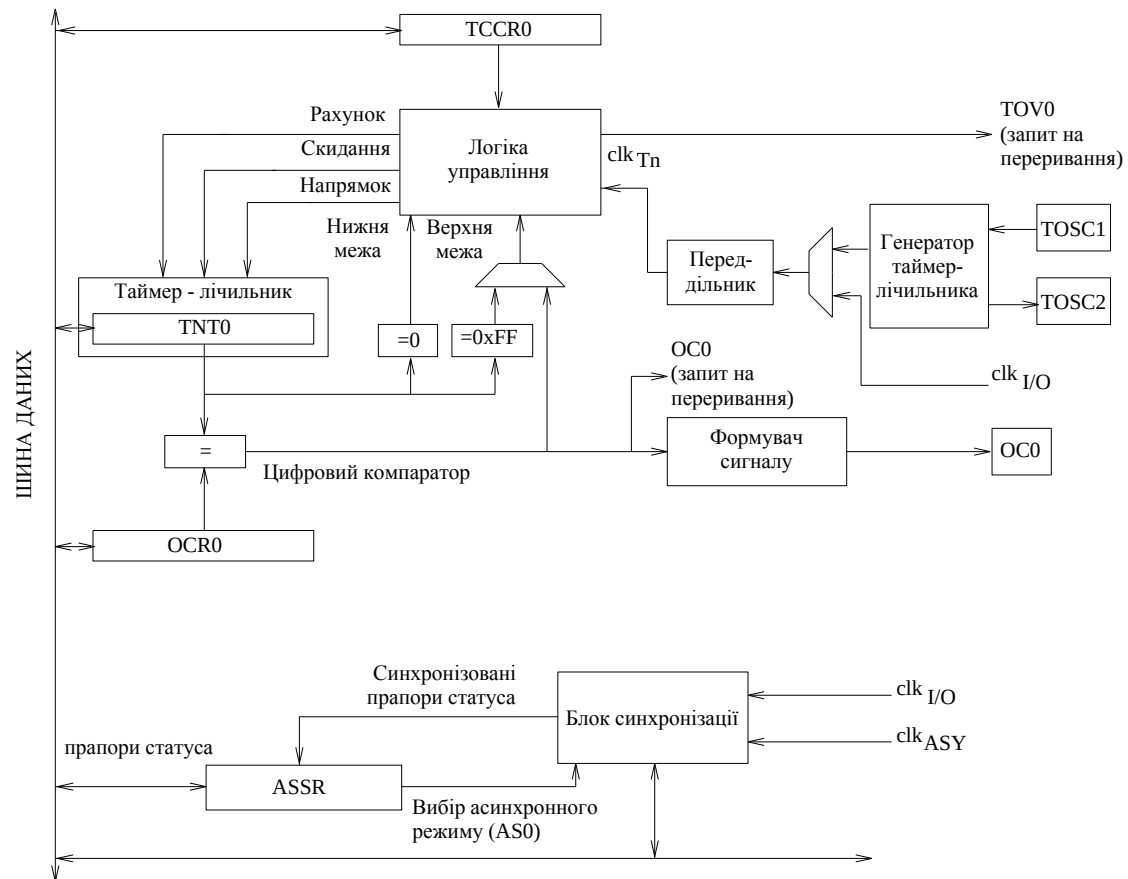


Рис. 3.36.

Регістр таймера-лічильника (TCNT0) і регістр порога порівняння (OCR0) – 8-розрядні регістри. Сигнали запиту на переривання представлені як прапори переривань таймера в регістрі TIFR. Всі переривання індивідуально маскуються за допомогою регістра маски переривань таймерів (TIMSK). Регістри TIFR й TIMSK не представлені на функціональній схемі, оскільки вони спільно використовуються з іншими таймерами мікроконтролера.

Значення регістра порога порівняння з подвійною буферизацією (OCR0) безперервно порівнюється зі значенням таймера-лічильника. Результат порівняння може використовуватися для генерації сигналів із ШІМ, або прямокутних імпульсів змінної частоти на виводі OC0.

8-розрядний цифровий компаратор безперервно виконує порівняння вмісту регістра таймера-лічильника TCNT0 з регістром порога порівняння OCR0. Коли значення TCNT0 збігається зі значенням OCR0 компаратор установлює прапор збігу OCF0 наступним тактом синхронізації таймера. Якщо дозволено переривання бітом OCIE0 = 1, то установка прапора збігу викликає запит на переривання. Прапор OCF0 автоматично скидається під час виконання процедури обробки переривання. Альтернативно, прапор OCF0 можна скинути програмно, шляхом запису лог. 1 у позицію даного біта. Генератор сигналу використовує сигнал результату порівняння для генерації прямокутних імпульсів по одному з алгоритмів, що вибирається

бітами завдання режиму роботи таймера WGM01, WGM00 і бітами завдання режиму формування вихідного сигналу (COM01, COM00).

Основним елементом 16-розрядного таймера-лічильника є програмований реверсивний 16-розрядний лічильник. На рис. 3.37 представлена функціональна схема лічильника та його елементів.

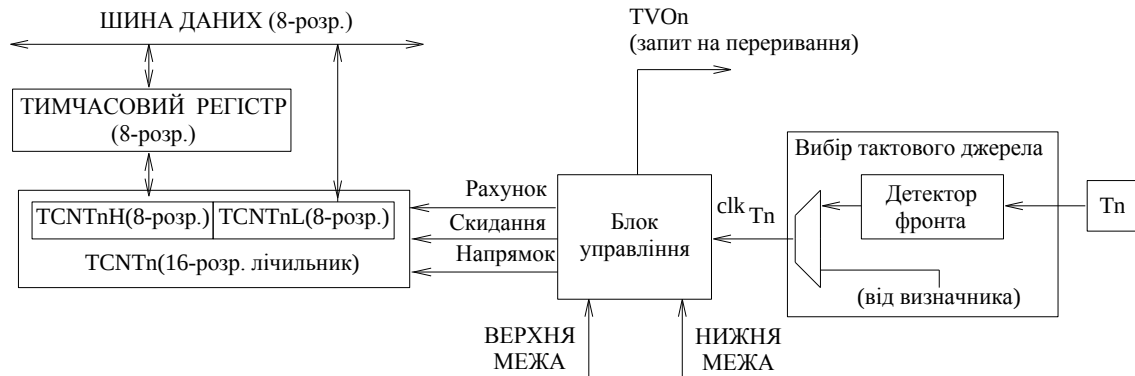


Рис. 3.37.

Опис внутрішніх сигналів:

Рахунок – інкрементує або декрементує стан TCNTn на 1.

Напрямок – задає прямий рахунок (інкремент) або зворотний рахунок (декремент).

Скидання – скидання TCNTn (установка всіх розрядів до лог. 0).

clkTn – синхронізація таймера-лічильника.

Верхня межа – сигналізує про досягнення TCNTn максимального значення.

Нижня межа – сигналізує про досягнення TCNTn мінімального значення (нуля).

Вміст 16-розрядного лічильника розбито на два 8-розрядних регістри, розташованих у пам'яті вводу-виводу: старший байт лічильника (TCNTnH), у якому зберігаються старші 8 розрядів лічильника, і молодший байт лічильника (TCNTnL), у якому зберігаються молодші 8 розрядів.

Регістр таймера-лічильника (TCNTn), регістри порогів порівняння (OCRn/B/C), а також регістр захоплення (ICRn) є 16-розрядними регістрами. У зв'язку із цим, під час доступу до цих регістрів повинна бути дотримана спеціальна процедура. Регістри керування таймером (TCCRn/B/C) є 8-розрядними регістрами, тому доступ до них з боку процесора не пов'язаний з якими-небудь обмеженнями. Всі сигнали запитів на переривання представлені в регістрі прапорів переривань таймерів (TIFR) і регістрі прапорів розширених переривань (ETIFR). Всі переривання індивідуально маскуються регістром маски переривань таймерів (TIMSK) і регістром маски розширених переривань (ETIMSK).

Таймер-лічильник може тактуватися внутрішньо через дільник або зовні тактовим джерелом, підключеним до виводу Tn. Блок вибору тактового джерела дозволяє вибрати тактове джерело і фронт, по якому

буде змінюватися стан таймера-лічильника. Якщо тактове джерело не задане, то таймер-лічильник перебуває в неактивному стані.

Значення регістрів порогів порівняння (OCR_n/B/C) безупинно порівнюється зі значенням лічильника. Результат порівняння може використовуватися для генерації прямокутних імпульсів із ШІМ або зі змінною частотою на виходах ОС_n/B/C. У випадку визначення збігу значень регістрів, що порівнюються, встановлюється відповідний прапор переривань (OCF_n/B/C), що у свою чергу може служити джерелом переривання.

3.6.2. Регістри керування таймерами

Найпростішим режимом роботи є нормальний режим (WGM01, WGM00 = 0). У даному режимі лічильник працює як додаючий, при цьому скидання лічильника не виконується. Переповнення лічильника відбувається при переході через максимальне 8-разрядне значення (верхня межа = 0xFF) до нижньої межі рахунку (0x00). У нормальному режимі роботи прапор переповнення таймера-лічильника TOV0 буде встановлений на тому ж такті синхронізації, коли TCNT0 прийме нульове значення. Фактично, прапор переповнення TOV0 є 9 бітом таймера-лічильника за тим виключенням, що він тільки встановлюється і не скидається.

Регістр керування таймером-лічильником 0 – TCCR0

Розряд	7	6	5	4	3	2	1	0
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
Зчитування/запис	Зч.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.
Початкове значення	0	0	0	0	0	0	0	0

Розряд 7 – FOC0: Примусова установка результату порівняння.

Функція біта FOC0 активна, якщо за допомогою біт WGM заданий один з режимів, де немає широтно-імпульсної модуляції. Однак з метою сумісності з наступними мікроконтролерами рекомендується під час запису в регістр TCCR0 у позиції даного біта вказувати лог. 0, якщо таймер працює в одному з режимів із широтно-імпульсною модуляцією. Якщо записати лог. 1 у біт FOC0, це приводить до примусової установки результату порівняння на вході блоку формування вихідного сигналу. Вихід ОС0 змінюється відповідно до установок біт COM01, COM00. Отже, значення записане в COM01, COM00 визначає ефект дії примусової установки результату порівняння.

Біт FOC0 завжди зчитується як 0.

Розряд 6, 3 – WGM01:0: Режим роботи таймера-лічильника 0.

Дані біти визначають алгоритм рахунку лічильника, джерело, що задає верхню межу рахунку і тип прямокутних імпульсів, що генеруються. Даним таймером підтримуються наступні режими роботи: нормальний

режим, режим скидання при збігу і два режими із широтно-імпульсною модуляцією.

Розряд 5:4 – COM01, COM00: Режим формування вихідного сигналу.

Дані біти визначають алгоритм зміни сигналу на виводі OC0. Якщо значення даних біт ненульове, то функція виводу OC0 як звичайного порту вводу-виводу, замінюється на альтернативну. Однак, варто враховувати, що напрямок цього виводу також керується через регістр напрямку даних порту B (DDRB). Тому, для дозволу альтернативної функції виводу OC0 також необхідно встановити біт 4 (OC0) у регістрі DDRB для установки вихідного напрямку.

Розряд 2:0 - CS02:0: Налаштування частоти синхронізації таймера.

За допомогою трьох біт є можливість вибрати різні тактові частоти, кратні вихідній частоті синхронізації (див. табл. 3.7).

Табл. 3.7.

CS02	CS01	CS00	Опис
0	0	0	Немає синхронізації. Таймер-лічильник 0 зупинений
0	0	1	clk0S/1
0	1	0	clk0S/8
0	1	1	clk0S/32
1	0	0	clk0S/64
1	0	1	clk0S/128
1	1	0	clk0S/256
1	1	1	clk0S/1024

Регістр таймера-лічильника – TCNT0

Розряд	7	6	5	4	3	2	1	0
	TCNT0[7:0]							
Зчитування/запис	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.
Початкове значення	0	0	0	0	0	0	0	0

Регістр таймера-лічильника характеризується двонаправленістю доступу до 8-розрядного таймера-лічильника 0. Запис у регістр TCNT0 блокує відпрацьовування виникаючого збігу на наступному після запису такті синхронізації таймера. Зміна вмісту лічильника (TCNT0) під час рахунку пов'язане з ризиком втрати результату порівняння між TCNT0 і регістром OCR0.

Регістр порогу порівняння – OCR0

Розряд	7	6	5	4	3	2	1	0
	OCR0[7:0]							
Зчитування/запис	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.
Початкове значення	0	0	0	0	0	0	0	0

Регістр порогу порівняння містить 8-розрядне значення, що безупинно порівнюється цифровим компаратором зі значенням 8-розрядного лічильника (TCNT0). Факт збігу значень може використовуватися для генерації переривання по виконанню умови порівняння, або для генерації прямокутних імпульсів на виводі OSC0.

Регістр маски переривань таймерів-лічильників – TIMSK

Розряд	7	6	5	4	3	2	1	0
	OSIE 2	TOIE 2	TICIE 1	OSIE1 A	OSIE1 B	TOIE 1	OSIE 0	TOIE 0
Зчитування/запис	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.
Початкове значення	0	0	0	0	0	0	0	0

Розряд 5 – TICIE1: Дозвіл переривання по захвату стану таймера-лічильника 1.

Якщо в даний біт записана лог. 1, а також установлений прапор I у регістрі статусу (загальний дозвіл переривань), то дозволяється переривання по захвату стану таймера-лічильника 1. Якщо встановлюється прапор у регістрі TIFR, програма переходить на відповідний вектор переривання.

Розряд 4 – OSIE1A: Дозвіл переривання по результату порівняння каналу A таймера-лічильника 1.

Якщо в даний біт записана лог. 1 і встановлений прапор I у регістрі статусу, то дозволяється робота переривання по результату порівняння каналу A. Якщо встановлюється прапор OCF1A у регістрі TIFR, то програма переходить на відповідний вектор.

Розряд 3 – OSIE1B: Дозвіл переривання по результату порівняння каналу B таймера-лічильника 1.

Дія аналогічно попередньому, але відносно каналу порівняння B.

Розряд 2 – TOIE1: Дозвіл переривання при переповненню таймера-лічильника 1.

Якщо в даний біт записана лог. 1 і встановлений прапор I у регістрі статусу, то дозволяється переривання по переповненню таймера-лічильника 1. Після цього установка прапора TOV1 у регістрі TIFR призведе до переходу на відповідний вектор переривання.

Розряд 1 – OSIE0: Дозвіл переривання по результату порівняння таймера-лічильника 0.

Якщо OSIE0=1, а також установлений біт I у регістрі статусу, то активізується переривання по результату порівняння таймера-лічильника 0. У цьому випадку переривання виникає, якщо відбувається збіг значення таймера-лічильника 0 з порогом порівняння, тобто коли встановлений прапор OCF0 у регістрі прапорів переривань таймерів-лічильників TIFR.

Розряд 0 – TOIE0: Дозвіл переривання по переповненню таймера-лічильника 0.

Якщо $TOIE0=1$, а також установлений біт I у регістрі статусу, то дозволяється переривання по переповненню таймера-лічильника 0. У цьому випадку запит на переривання генерується, якщо відбувається переповнення таймера-лічильника 0, тобто коли встановлений прапор $TOV0$ у регістрі прапорів переривань таймерів-лічильників TIFR.

Регістр прапорів переривань таймерів-лічильників – TIFR

Розряд	7	6	5	4	3	2	1	0
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
Зчитування/запис	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.
Початкове значення	0	0	0	0	0	0	0	0

Розряд 5 – ICF1: Прапор захоплення стану таймера-лічильника 1.

Прапор встановлюється, якщо на вході ICP1 визначається умова захвата. Якщо регістр захвата ICR1 обраний за допомогою бітів $WGMn3:0$ як джерело верхньої межі рахунку, прапор ICF1 встановлюється при досягненні верхньої межі рахунку.

ICF1 автоматично скидається при переході на вектор переривання по захвату стану таймера-лічильника. Альтернативно прапор ICF1 можна скидати шляхом запису в нього лог. 1.

Розряд 4 – OCF1A: Прапор результату порівняння каналу A таймера-лічильника 1.

Даний прапор встановлюється наступним тактом після збігу значення TCNT1 з регістром A порогу порівняння (OCR1A).

Прапор OCF1A автоматично скидається при переході на відповідний вектор переривання. Альтернативно, прапор OCF1A скидається шляхом запису в нього лог. 1.

Розряд 3 – OCF1B: Прапор результату порівняння каналу B таймера-лічильника 1.

Даний прапор діє аналогічно попередньому, але відносно каналу порівняння B.

Розряд 2 – TOV1: Прапор переповнення таймера-лічильника 1.

Установка даного прапора залежить від значень біт $WGMn3:0$.

Розряд 1 – OCF0: Прапор збігу таймера-лічильника 0.

OCF0 дорівнює лог. 1, якщо відбувається збіг між значенням таймера-лічильника 0 і даними в регістрі OCR0 (регістр порога порівняння). OCF0 скидається апаратно при переході на відповідний вектор переривання. Альтернативно, прапор OCF0 може бути скинутий шляхом запису в нього лог. 1. Якщо встановлені біт I у регістрі SREG, біт $OSIE0$ (дозволений переривання по виконанню умови порівняння таймера-лічильника 0) і прапор OCF0, то генерується переривання по виконанню умови порівняння таймера-лічильника 0.

Розряд 0 – TOV0: Прапорець переповнення таймера-лічильника 0.

Прапорець TOV0 встановлюється, якщо в таймері-лічильнику 0 виникає переповнення. Прапор TOV0 скидається апаратно при переході на відповідний вектор переривання. Альтернативно, прапор TOV0 скидається шляхом запису в нього лог. 1. Якщо встановлені біт I у регістрі SREG, біт TOIE0 (дозволене переривання по переповненню таймера-лічильника 0) та біт TOV0, то генерується переривання по переповненню таймера-лічильника 0. У режимі ШІМ даний прапор встановлюється, якщо таймер-лічильник 0 змінює напрямок рахунку на значенні 0.

Регістр А керування таймером-лічильником 1 – TCCR1A

Розряд	7	6	5	4	3	2	1	0
	COM1 A1	COM1 A0	COM1 B1	COM1 B0	COM1 C1	COM1 C0	WGM 11	WGM 10
Зчитуван ня /запис	Зч. /Зап	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.
Початков е значення	0	0	0	0	0	0	0	0

Розряди 7:6 – COMn1:0: Режим формування вихідного сигналу каналу А.

Розряди 5:4 – COMn1:0: Режим формування вихідного сигналу каналу В.

Розряди 3:2 – COMn1:0: Режим формування вихідного сигналу каналу С.

Біти COMn1:0, COMn1:0 та COMn1:0 впливають на роботу виводів OCn, OCn та OCn, відповідно. Якщо один або обидва біти COMn1:0 дорівнюють 1, то вивід OCn переходить до виконання альтернативної функції, забороняючи його роботу як звичайного порту вводу-виводу. Аналогічні зміни відбуваються з виводами OCn та OCn під час запису лог. 1 в один з бітів COMn1:0 й COMn1:0 відповідно. Однак необхідно враховувати, що залишається вплив на роботу даних виводів з боку регістра напрямку даних (DDR) і у відповідних розрядах цього регістра повинно бути задано вихідний напрямок для виводів OCn, OCn або OCn.

Якщо обрано підключення сигналів OCn, OCn або OCn до виводів мікроконтролера, то призначення біт COMnx1:0 визначається обраним за допомогою біт WGMn3:0 режиму роботи таймера-лічильника.

Розряд 1:0 - WGMn1:0: Режим роботи таймера-лічильника.

У сполученні з бітами WGMn3:2 з регістра TCCRn дані біти визначають алгоритм рахунку, джерело для завдання вершини рахунку і тип сигналу, що генерується.

Регістр В керування таймером-лічильником 1 – TCCR1B

Розряд	7	6	5	4	3	2	1	0
	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10
Зчитування /запис	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.
Початкове значення	0	0	0	0	0	0	0	0

Розряд 7 – ICNCn: Фільтр шуму на вході захоплення.

Установка даного біта (запис лог. 1) активізує фільтр шуму на вході захвата. Логіка роботи фільтра складається у визначенні чотирьох підряд рівних за значенням вибірок і тільки в цьому випадку зміни свого вихідного стану. Отже, після дозволу придушення шумів сигнал із входу захоплення буде затримуватися на 4 такти системної синхронізації.

Розряд 6 – ICESn: Вибір типу фронту на вході захвата.

Даний біт дозволяє задати, який фронт на вході захвату ICPn призведе до захоплення стану таймера. Якщо ICESn=0, то зріз сигналу призводить до захоплення стану таймера, а якщо ж ICESn=1, то фронт призводить до виникнення захоплення.

Якщо відповідно до установки ICESn виникає умова захоплення, то вміст лічильника копіюється в регістр захвату ICRn. При цьому також встановлюється прапор захоплення ICFn, що може використовуватися для генерації переривання по захопленню (якщо дане переривання дозволене).

Якщо регістр ICRn використовується для зберігання значення верхньої межі рахунку, то вхід ICPn відключається від відповідного виводу мікроконтролера і функція захоплення блокується.

Розряд 5 – Зарезервований біт.

Даний біт зарезервований для подальшого використання. З метою сумісності з майбутніми розробками рекомендується під час запису в регістр TCCRn у даному розряді вказувати лог. 0.

Розряд 4:3 – WGMn3:2: Режим роботи таймера-лічильника.

Розряд 2:0 – CSn2:0: Вибір тактового джерела.

Регістр С керування таймером-лічильником 1 – TCCR1C

Розряд	7	6	5	4	3	2	1	0
	FOC1A	FOC1B	FOC1C	-	-	-	-	-
Зчитування /запис	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.
Початкове значення	0	0	0	0	0	0	0	0

Розряд 7 – FOCn: Біт примусової установки результату порівняння для каналу А.

Розряд 6 – FOCn: Біт примусової установки результату порівняння для каналу В.

Розряд 5 – FOCn: Біт примусової установки результату порівняння для каналу C.

Біти FOCn/FOCn/FOCn стають активними, коли за допомогою біт WGMn3:0 обраний режим без ІІМ. У цьому випадку запис в FOCn/FOCn/FOCn лог. 1 призводить до негайної установки результату порівняння на вході блоку формувача сигналу. Вихід OCn/OCn/OCn змінюється відповідно до установок біт COMnx1:0. Зверніть увагу, що біти FOCn/FOCn/FOCn реалізовані як строби. Строби FOCn/FOCn/FOCn не генерують яких-небудь переривань і скидають лічильник у режимі скидання таймера при збігу (СТС), де OCRn використовується для задання вершини рахунку.

При зчитуванні біт FOCn/FOCn/FOCn завжди повертається нульове значення.

Розряди 4:0 – Зарезервовані біти.

Дані біти зарезервовані для подальшого використання. З метою сумісності з наступними розробками під час запису в регістр TCCRn необхідно записувати лог. 0.

Таймер-лічильник 1 – TCNT1H й TCNT1L

Розряд	7	6	5	4	3	2	1	0
	TCNT1[15:8]							
	TCNT1[7:0]							
Зчитування /запис	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.
Початкове значення	0	0	0	0	0	0	0	0

Два регістри в області вводу-виводу (TCNTn й TCNTn, разом TCNTn) дають повний доступ, як на зчитування, так і на запис до 16-розрядного лічильника.

Регістр порівняння 1 A – OCR1AH й OCR1AL

Розряд	7	6	5	4	3	2	1	0
	OCR1A [15:8]							
	OCR1A [7:0]							
Зчитування /запис	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.
Початкове значення	0	0	0	0	0	0	0	0

У регістрах порівняння зберігається 16-розрядне значення, що постійно порівнюється зі значенням лічильника (TCNTn). Виникаючий збіг може використовуватися для генерації переривання по результату порівняння і генерації прямокутних імпульсів на виводі OCnx.

3.6.3. Приклади програм з таймером-лічильником

В даному параграфі приводяться декілька прикладів програм, в яких використовується таймер/лічильник в різних режимах роботи.

Приклад 3.26. До виводу PD3 мікроконтролера AT90S8535 підключена кнопка (схема на рис. 3.38). Написати програму, яка виводить на семисегментні індикатори кількість натискань на кнопку за проміжок часу 10 секунд.

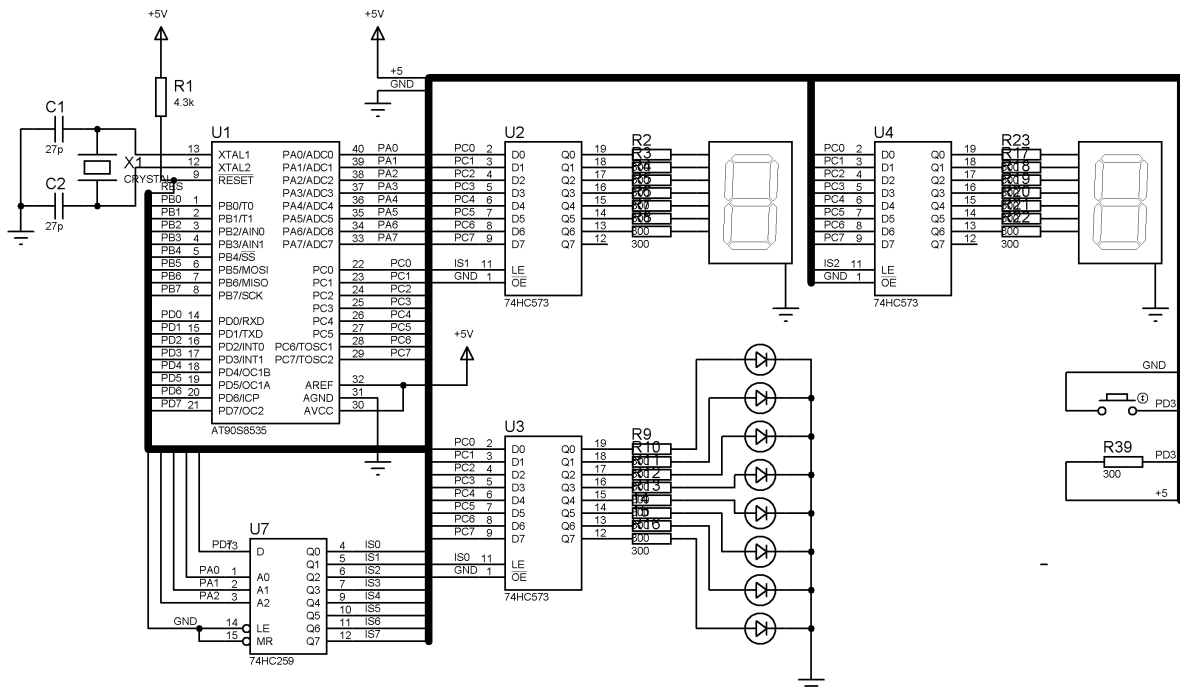


Рис. 3.38.

```
#include<io8535v.h>           // підключення бібліотеки
#include<macros.h>
int i, c, l;                  // додаткові змінні
// в масиві зберігаються коди семисегментного індикатора
char seg[10]={0b00111111,0b00000110,0b01011011,0b01001111,0b01100110,
0b01101101, 0b01111101,0b00000111,0b01111111,0b01101111};
// обробник зовнішнього переривання
#pragma interrupt_handler int1_isr:3
void int1_isr()
{
    i++;                       // лічильник натискань на кнопку
}
#pragma interrupt_handler timer0_ovf:10
void timer0_ovf()
{
    c++;                       // інкремент значення змінної
    if(c==30)                  // якщо значення дорівнює 30 (пройшла 1 секунда)
    {
        c=0;                  // онулення змінної
        PORTA=0;              // вибір регістру керування світлодіодами
    }
}
```

```
PORTC^=0b11111111;    // індикація відліку часу
l++;                  // лічильник секунд
}
if(l==10)              // якщо пройшло 10 секунд
{
PORTA=1;              // вибір регістра першого індикатора
PORTC=seg[i/10];      // вивід значення десятків
PORTA=2;              // вибір регістра другого індикатора
PORTC=seg[i%10];      // вивід одиниць
PORTA=0;              // вибір регістра світлодіодів
PORTC=0;              // вимикання всіх світлодіодів
l=0;                  // онулення лічильника секунд
i=0;                  // онулення лічильника натискань
}
}

void init_cpu()        // функція ініціалізації мікроконтролера
{
DDRA=0x07;            // налаштування 3-х біт порту А на вихід
DDRC=0xFF;            // налаштування порту С на вихід
DDRD=0x80;            // вивід 7 порту D налаштований на вихід
PORTD=0x80;           // на виводі 7 порту D встановлений рівень лог.1
MCUCR=0b00001010;    // переривання виникатиме по фронту сигналу
GIMSK=0b11000000;    // дозвіл переривань INT0, INT1
TIMSK=0b00000001;    // дозвіл переривання по переповненню таймера 0
TCNT0=0;              // значення таймера лічильника встановлюється в 0
TCCR0=0b00000101;    // дільник частоти таймера дорівнює 1024
SEI();                // дозвіл обробки переривань
i=0;                  // онулення змінної
}

void main()            // головна функція програми
{
init_cpu();           // виклик функції ініціалізації мікроконтролера
while(1);             // вічний цикл
}
```

При натисканні на кнопку, яка підключена до входу зовнішнього переривання (вивід PD3), виникє переривання і викликається функція `int1_isr()`, в якій відбувається інкремент додаткової глобальної змінної. Таймер/лічильник налаштований таким чином, що за 1 секунду виникає його переповнення 30 разів. При переповненні таймера-лічильника виникає переривання і викликається функція `timer0_ovf()`, в якій за допомогою додаткової змінної підраховується кількість подій переповнення таймера/лічильника. За 10 секунд відбувається 300 переповнень таймера/лічильника, після чого на семсегментні індикатори виводиться кількість натискань на кнопку за цей проміжок часу.

Приклад 3.27. Написати програму (схема представлена на рис. 3.39), яка формує синусоїдальний ШІМ-сигнал по двох каналах. Використовується таймер-лічильник у режимі широтно-імпульсної модуляції.

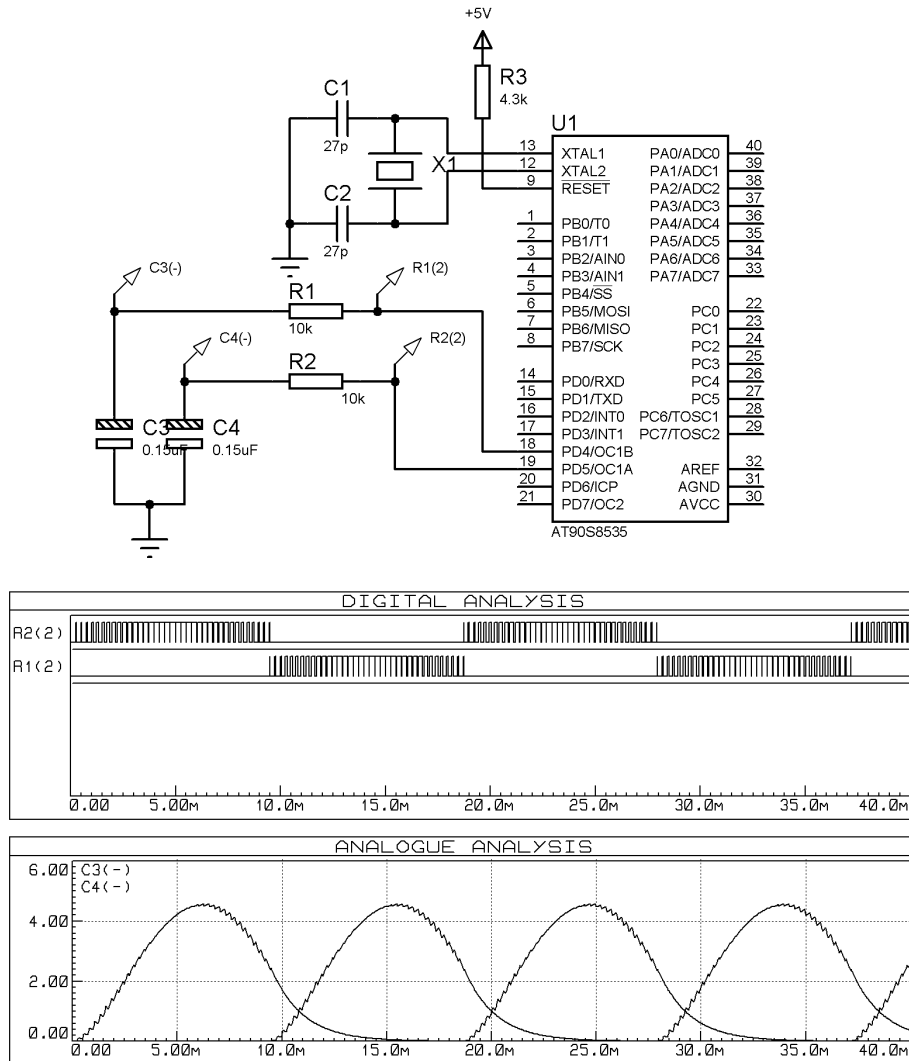


Рис. 3.39.

Текст програми представлений нижче.

```
#include <io8535v.h>          // підключення бібліотеки
#include <macros.h>
// в масиві зберігається значення, пропорційні функції синуса
const int sinarr[40]={82,164,244,323,400,475,546,614,678,737,792,841,885,
923,956,982,1002,1015,1022,1022,1015,1002,982,956,923,885,841,792,737,678,
614,546,475,400,323,244,164,82,1,0};

char flag;                    // оголошення додаткових глобальних змінних
char flag1;
char index;

void port_init(void)          // функція налаштування портів
{
```

```
DDRD = 0x30;           // виводи PD4 та PD5 налаштовані на вихід
flag=0;                // обнуління змінних
flag1=0;
index=0;
}

void timer1_init(void)  // функція налаштування таймера
{
    TCCR1A = 0x83;      // дозвіл переривання таймера по порівнянню
    TCCR1B = 0x01;      // запуск таймера
}
// обробник переривання при співпаданні значення таймера-лічильника
// з регістром порівняння OCR1A
#pragma interrupt_handler timer1_compa_isr:7
void timer1_compa_isr(void)
{
    if(flag==0) flag=1;    // зміна значення прапорця
    else if(flag==1)       // якщо значення змінної дорівнює 1
    {
        flag=0;           // онулення змінної
        index=index+1;     // інкремент індексу
        OCR1A=sinarr[index]; // завантаження в регістр порівняння А нового значення
        if(index>=38)      // якщо всі значення були завантажені
        {
            CLI();         // заборона обробки переривань
            index=0;       // онулення змінної
            TCNT1=0;       // онулення таймера-лічильника
            TCCR1A = 0x23; // зміна режиму роботи таймера
            OCR1B=sinarr[0]; // завантаження нового значення в регістр порівняння В
            TIMSK = 0x08;  // дозвіл обробки переривання по співпадінню
            SEI();         // дозвіл обробки переривань
        }
    }
}

// обробник переривання при співпаданні значення таймера-лічильника
// з регістром порівняння OCR1B
#pragma interrupt_handler timer1_compb_isr:8
void timer1_compb_isr(void)
{
    if(flag1==0) flag1=1; // зміна значення прапорця
    else if(flag1==1)     // якщо значення змінної дорівнює 1
    {
        flag1=0;         // онулення змінної
        index=index+1;    // інкремент індексу
        OCR1B=sinarr[index]; // завантаження в регістр порівняння В нового значення
        if(index>=38)    // якщо всі значення були завантажені
        {
            CLI();       // заборона обробки переривань
            index=0;     // онулення змінної
            TCNT1=0;     // онулення таймера-лічильника
            TCCR1A = 0x83; // зміна режиму роботи таймера
            OCR1A=sinarr[0]; // завантаження нового значення в регістр порівняння А
            TIMSK = 0x10;  // дозвіл обробки переривання по співпадінню
        }
    }
}
```

```

    SEI();                // дозвіл обробки переривань
    }}}

void init_devices(void)   // функція ініціалізації контролера
{
    CLI();                // заборона обробки переривань
    port_init();           // виклик функції налаштування портів
    timer1_init();         // виклик функції налаштування таймера
    TIMSK = 0x10;
    SEI();                // дозвіл обробки переривань
}
void main()               // головна функція програми
{
    init_devices();        // виклик функції ініціалізації контролера
    OCR1A=sinarr[index];   // завантаження значення в регістр порівняння
    while(1);             // вічний цикл
}

```

В масиві `sinarr` зберігаються значення половини періоду синусоїдального сигналу. Кожного разу при співпадині значення таймера/лічильника з регістрами переривання OCR1A та OCR1B відбувається виклик функції – обробника переривання (`timer1_compa_isr()` та `timer1_compb_isr()` відповідно). В цих функціях в регістри OCR1A та OCR1B завантажуються нові значення з масиву `sinarr`, формуючи на виходах PD4 та PD5 мікроконтролера ШІМ-сигнал, в якому скважність імпульсів змінюється по синусоїдальному закону. Це демонструється осцилограмою, що представлена на рис. 3.39.

3.8. Обробка зовнішніх переривань

Зовнішні переривання здійснюються через виводи INT7:0. Після дозволу зовнішні переривання будуть генеруватися, навіть якщо лінії INT7:0 налаштовані як виходи. Дана особливість може використовуватися для програмної генерації переривання. Зовнішні переривання можуть генеруватися по фронту або зрізу імпульсу, а також по низькому логічному рівні сигналу. Ці налаштування задаються в регістрах керування зовнішніми перериваннями EICRA (INT3:0) і EICRB (INT7:4). Якщо зовнішнє переривання дозволене й налаштоване на спрацьовування при низькому рівні, то переривання буде ініціюватися постійно, поки на виводі буде залишатися низький рівень.

Регістр А керування зовнішніми перериваннями – EICRA

Розряд	7	6	5	4	3	2	1	0
	ICS31	ICS30	ICS21	ICS20	ICS11	ICS10	ICS01	ICS00
Зчитування/запис	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.
Початкове значення	0	0	0	0	0	0	0	0

Розряди 7..0 – ICSC31, ICS30 – ICS00: Біти вибору умови генерації зовнішнього переривання 3 – 0.

Зовнішні переривання 3 – 0 активізуються через зовнішні виводи INT3:0, якщо встановлені прапор I у регістрі статусу SREG і відповідна маска переривання в EIMSK. Вибір рівня або фронту для активізації зовнішнього переривання здійснюється відповідно до табл. 3.8. Переривання по виводах INT3:0 буде згенеровано, якщо тривалість імпульсу буде більше мінімально необхідної. При виникненні імпульсів меншої тривалості генерація переривання не гарантується. Якщо обрано переривання по низькому рівні, то для генерації переривання необхідно, щоб цей рівень залишався на низькому рівні до моменту завершення виконання поточної інструкції. Після дозволу переривання за рівнем воно буде генеруватися безупинно доти, поки на вході присутній низький рівень. При зміні біт ISC_n може виникнути переривання. Тому, рекомендується спочатку відключити переривання INT_n шляхом скидання біта дозволу переривання в регістрі EIMSK. Після цього, значення біт ISC_n може бути змінено. Перед відновленням роботи переривань необхідно скинути прапор переривання INT_n шляхом запису лог. 1 у прапорець переривання (INTF_n) в регістрі EIFR.

Табл. 3.8.

ISC _{n1}	ISC _{n0}	Опис
0	0	Низький рівень на INT _n генерує запит на переривання
0	1	Зарезервовано
1	0	Зріз сигналу на INT _n генерує запит на переривання
1	1	Фронт сигналу на INT _n генерує запит на переривання

Примітка: $n = 3, 2, 1$ або 0.

Перед зміною бітів ISC_{n1}/ISC_{n0} необхідно заборонити роботу переривання шляхом очищення біта дозволу переривання в регістрі EIMSK. У протилежному випадку переривання може виникнути після зміни даних біт.

Регістр В керування зовнішніми перериваннями – EICRB

Розряд	7	6	5	4	3	2	1	0
	ICS71	ICS70	ICS61	ICS60	ICS51	ICS50	ICS41	ICS40
Зчитування/запис	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.
Початкове значення	0	0	0	0	0	0	0	0

Розряди 7..0 – ICS71, ICS70 – ICS41, ICS40: Біти вибору умови генерації зовнішнього переривання 7 – 4.

Зовнішні переривання 7 – 4 активізуються через зовнішні виводи INT7:4, якщо встановлені прапор I у регістрі статусу SREG і відповідна маска переривання в регістрі EIMSK. Для визначення фронтів на виводах

INT7:4 здійснюється вибірка їхніх станів. Якщо обрано переривання по фронту або зміні рівня, то переривання буде згенеровано, якщо на вході з'являється імпульс, тривалість якого більше одного періоду синхронізації. При дії на вході більш коротких імпульсів генерація переривання не гарантується.

Регістр маски зовнішнього переривання – EIMSK

Розряд	7	6	5	4	3	2	1	0
	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0
Зчитування/запис	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.
Початкове значення	0	0	0	0	0	0	0	0

Розряди 7..0 – INT7 – INT0: Дозвіл запиту на зовнішні переривання 7 – 0.

Якщо в біти INT7 – INT0 та у біт I регістра статусу SREG записати лог. 1, то дозволяється робота зовнішнього переривання при зміні сигналу на відповідному виводі. Біти вибору умови генерації переривання в регістрах керування зовнішніми перериваннями EICRA та EICRB визначають, по якій умові генерується переривання: по фронту, по зрізу або за рівнем. Будь-який з даних виводів зберігає активність, навіть якщо він налаштований на вихід. Дана особливість може використовуватися для програмної генерації переривання.

Регістр прапорів переривання – EIFR

Розряд	7	6	5	4	3	2	1	0
	INTF7	INTF6	INTF5	INTF4	INTF3	INTF2	INTF1	INTF0
Зчитування/запис	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.
Початкове значення	0	0	0	0	0	0	0	0

Розряди 7..0 – INTF7 – INTF0: Прапори зовнішніх переривань 7 – 0.

Якщо фронт або зміна логічного стану на INT7:0 викликає переривання, то відповідний прапор INTF7:0 приймає одиничний стан. Якщо встановлені біт I регістра статусу SREG і відповідний біт дозволу переривання INT7:0 у регістрі EIMSK, то мікроконтролер виконає перехід на вектор переривання. Прапор скидається апаратно після виконання процедури обробки переривання. Альтернативно прапор може бути скинутий програмно шляхом запису лог. 1 у відповідний біт. Якщо INT7:0 налаштовані на генерацію переривання за рівнем, то прапори постійно перебувають у скинутому стані.

Приклад 3.28. На вхід зовнішнього переривання INT0 поступає імпульсний сигнал. Написати програму, яка вимірює частоту вхідних імпульсів з використанням таймера-лічильника.

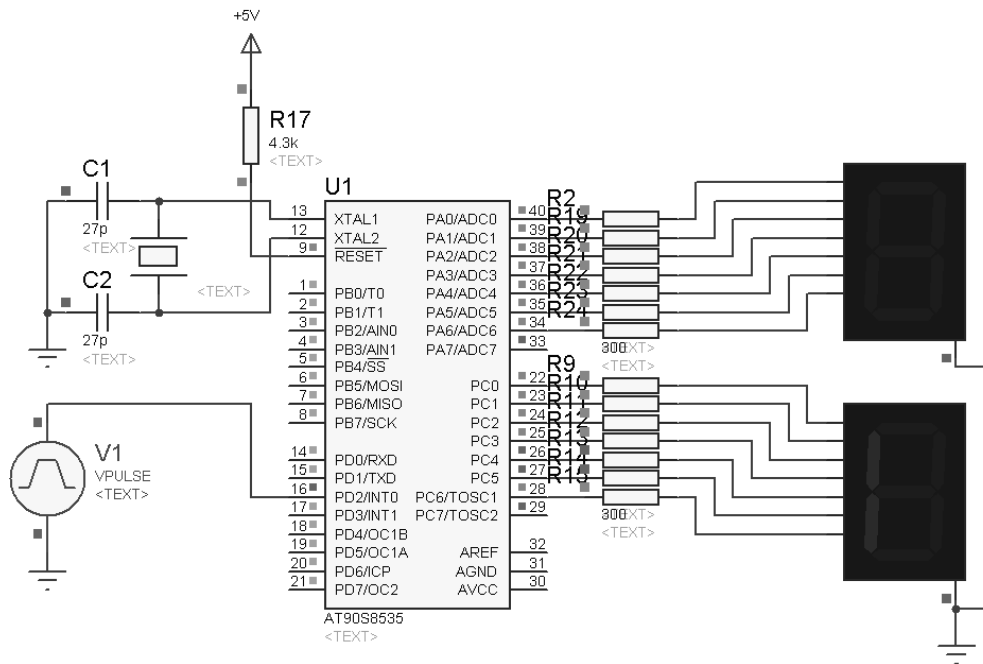


Рис. 3.40.

Текст програми представлений нижче.

```
#include "io8535v.h"
#include "macros.h"
char flag; // оголошення глобальної змінної

#pragma interrupt_handler int0_isr:2 // обробник зовнішнього переривання
void int0_isr(void)
{
    int freq; // додаткові змінні
    int pulses;
    char segment[10]={0b00111111, 0b00000110, 0b01011011, 0b01001111,
    0b01100110, 0b01101101, 0b01111101, 0b00000111, 0b01111111, 0b01101111};
    char d1, d2;
    if(flag==0) // якщо значення змінної дорівнює 0
    {
        flag=1; // змінній присвоюється значення 1
        TCCR1B=0x02; // запуск таймера-лічильника
    }
    else if(flag==1) // якщо значення змінної дорівнює 1
    {
        flag=0; // змінній присвоюється значення 1
        TCCR1B=0; // зупинка таймера-лічильника
        pulses=TCNT1; // зчитування значення таймера-лічильника
        TCNT1=0;
        freq=1000000/pulses; // визначення частоти імпульсів
        PORTA=segment[freq/10]; // вивід значення на семи сегментні індикатори
        PORTC=segment[freq-(freq/10)*10];
    }
}
```



```

void main()           // головна функція
{
    flag=0;            // оголошення змінної
    DDRA=0xFF;         // налаштування порта А на вивід
    DDRC=0xFF;         // налаштування порта С на вивід
    TCCR1A=0;          // ініціалізація регістрів таймера-лічильника
    TCCR1B=0;
    TCNT1=0;
    MCUCR = 0x03;      // дозвіл зовнішнього переривання
    GIMSK = 0x40;      // установка маски переривання: по фронту імпульсу
    SEI();              // дозвіл обробки переривань
    while(1);           // вічний цикл
}

```

В програмі використовується функція-обробник зовнішнього переривання `int0_isr()` для запуску та зупинки таймера/лічильника. За допомогою змінної `flag` відслідковується режим роботи мікроконтролера. Якщо значення цієї змінної дорівнює 0, відбувається запуск таймера/лічильника, і змінній присвоюється значення 1. При наступному виклику функції-обробника переривання аналізується значення змінної `flag`, і якщо значення дорівнює 1, то таймер зупиняється і на семисегменті індикатори виводиться значення частоти.

3.9. Сторожовий таймер

Сторожовий таймер тактується від окремого вбудованого генератора частотою 1 МГц (рис. 3.41). Дане значення типове для напруги живлення $VCC = 5$ В. Період переповнення сторожового таймера можна задавати шляхом керування дільником частоти.

Сторожовий таймер також скидається при вимиканні або під час скидання мікроконтролера. Період переповнення визначають вісім різних коефіцієнтів дільника. Якщо команда скидання сторожового таймера `WDR` не виконується протягом часу, що дорівнює періоду переповнення сторожового таймера, то `ATmega128` скидається і починає виконання програми по вектору скидання.

Регістр керування сторожового таймера – `WDTCR`

Розряд	7	6	5	4	3	2	1	0
	–	–	–	WDCE	WDE	WDP2	WDP1	WDP0
Зчитування /запис	Зч.	Зч.	Зч.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.
Початкове значення	0	0	0	0	0	0	0	0

Розряди 7..5 – Зарезервовані.

Дані розряди є зарезервованими і завжди зчитуються як 0.

Розряд 4 – `WDCE`: Дозвіл зміни сторожового таймера.

Даний біт необхідно встановити безпосередньо перед записом лог. 0 у біт WDE. У протилежному випадку заборонити роботу сторожового таймера неможливо. Після запису в даний біт лог. 1 він автоматично апаратно скинеться після закінчення чотирьох тактів синхронізації мікроконтролера. На рівнях безпеки 1 та 2 даний біт необхідно встановлювати також перед зміною налаштувань дільника.

Розряд 3 – WDE: Дозвіл сторожового таймера.

Робота сторожового таймера дозволяється (забороняється) шляхом установки (скидання) біта WDE. Скинути біт WDE можливо, тільки якщо попередньо встановити біт WDCE. Для вимикання сторожового таймера необхідно виконати наступну послідовність:

1. Записати лог. 1 в WDCE та WDE однією командою. Логічна 1 повинна бути записана в біт WDE, навіть якщо до виконання даної операції в ньому вже була записана лог. 1.
2. Протягом наступних чотирьох тактів записати лог. 0 в WDE, що приводить до відключення сторожового таймера.

На 2-ому рівні безпеки заборонити роботу сторожового таймера неможливо навіть за допомогою зазначеної вище послідовності.

Розряд 2..0 – WDP2, WDP1, WDP0: Вибір коефіцієнта дільника.

Біти WDP2, WDP1 й WDP0 задають коефіцієнт ділення частоти генератора сторожового таймера після дозволу роботи останнього.

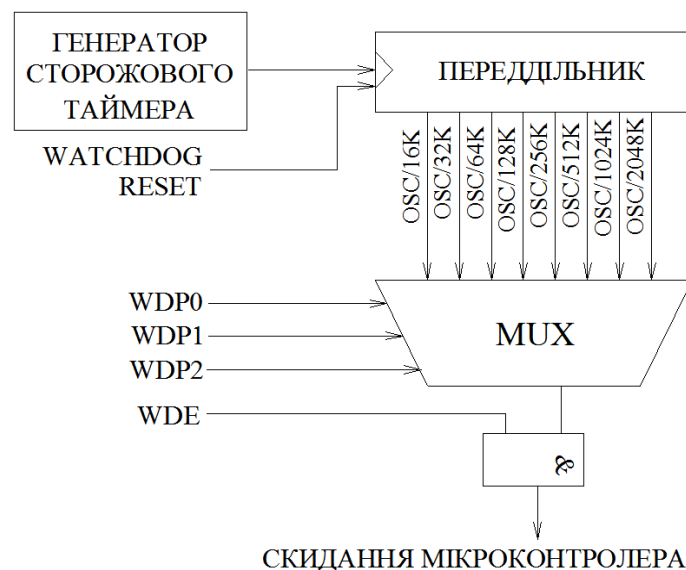


Рис. 3.41.

Послідовність зміни налаштування сторожового таймера плавно змінюється між трьома рівнями безпеки. Нижче описані процедури зміни налаштувань для кожного з рівнів.

Рівень безпеки 0. Сторожовий таймер спочатку відключений, але може бути активізований шляхом запису в біт WDE лог. 1 без яких-небудь обмежень. Період переповнення таймера також може бути змінений без обмежень.

Рівень безпеки 1. У даному режимі сторожовий таймер спочатку відключений. Його робота може бути дозволена шляхом запису лог. 1 у біт WDE без яких-небудь обмежень. Повинна бути виконана наступна послідовність:

1. За допомогою однієї й тієї ж команди записати лог. 1 в WDCE та WDE. Лог. 1 повинна бути записана в WDE незалежно від попереднього значення біта WDE.
2. Протягом наступних 4-х тактів однією командою записати бажане значення біт WDE та WDP, але зі скинутим значенням біта WDCE.

Рівень безпеки 2. У даному режимі сторожовий таймер завжди включений а значення біта WDE завжди зчитується як лог. 1. Повинна бути виконана наступна послідовність:

1. Записати лог. 1 в WDCE та WDE однією інструкцією. Незважаючи на те, що WDE завжди встановлений, для запуску часової послідовності в нього однаково необхідно записувати лог. 1.
2. Протягом наступних 4-х тактів записати бажане значення WDP однією інструкцією при скинутому значенні біта WDCE. Записуване значення в біт WDE не призводить до яких-небудь наслідків.

Після закінчення періоду переповнення сторожового таймера генерується короткий імпульс скидання тривалістю, що дорівнює одному періоду системної синхронізації (1 CK) (рис. 3.42). Падаючим фронтом цього імпульсу запускається лічильник затримки t_{TOUT} .

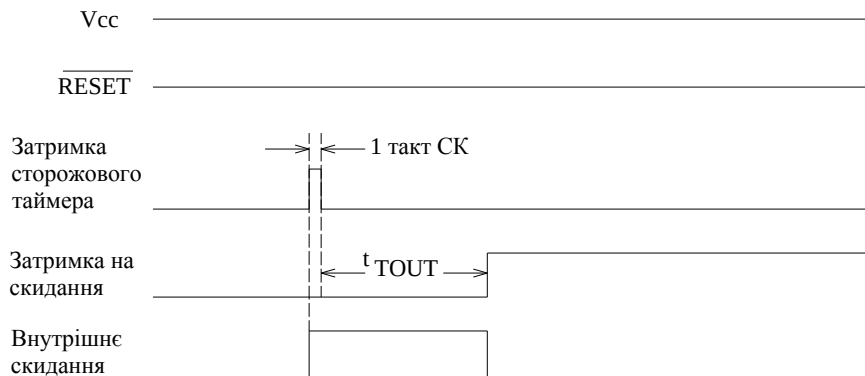


Рис. 3.42.

3.10. Налаштування програм в AVR-Studio

При старті AVR Studio відразу відкривається вікно майстра створення нового проекту. Натиснувши на кнопку Open і вказавши шлях до файлу з розширенням COFF, з'явиться вікно, показане на рис. 3.43. У цьому вікні ліворуч необхідно вибрати обладнання, з яким буде виконуватися робота. Для моделювання та налагодження програм необхідно вибрати пункт AVR Simulator і вказати модель мікроконтролера, для якого була написана програма.

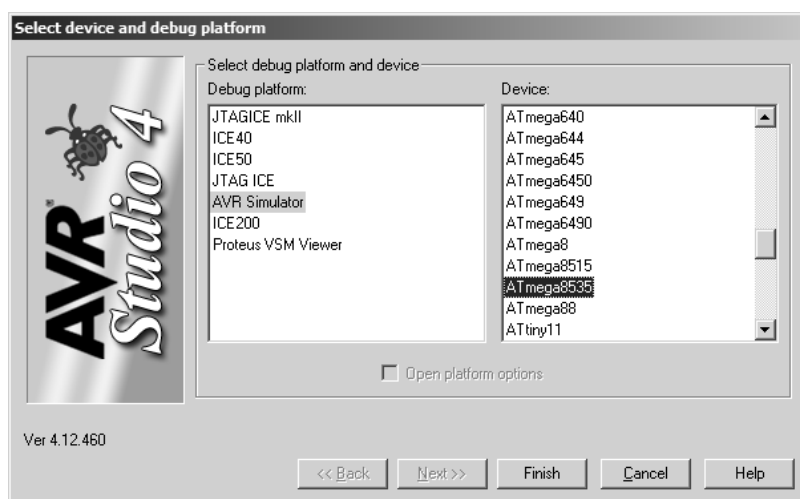


Рис. 3.43.

Після натискання на кнопку Finish з'являється головне вікно програми (рис. 3.44).

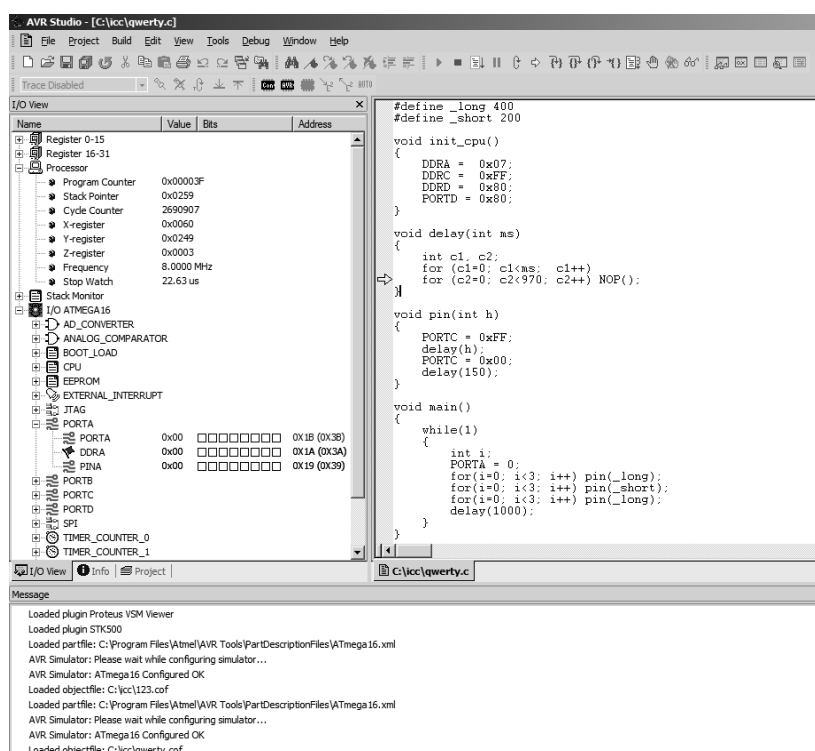


Рис. 3.44.

AVR Studio має дуже гнучкий інтерфейс, і зовнішній вигляд може сильно відрізнитися від варіанту, як показано на малюнку. Головна панель програми AVR Studio розділена на три основних вікна. Зліва розташоване вікно ресурсів мікроконтролера. Воно надає повну інформацію про поточний проект. За замовчуванням це вікно містить в собі три вкладки. Перша вкладка називається «Info». Вона містить довідкову інформацію про мікроконтролер, що використовується, а саме опис векторів переривань, опис виводів для різних корпусів та короткий опис регістрів.

Наступна вкладка називається «Project». Вона містить повну інформацію про поточний проект. Інформація представлена у вигляді дерева. Різні гілки цього дерева описують всі вихідні і результуючі файли проекту, всі мітки, процедури і файли.

Остання вкладка вікна називається «I/O View» (перегляд вводу/виведення). Це найкорисніша вкладка. На ній можна побачити, що відбувається з мікроконтролером: стан біт регістрів керування периферійними пристроями, регістрів порту, регістрів конфігурації. Вся інформація також представлена у вигляді дерева. Кожна гілка цього дерева – це окремий елемент. Якщо який-небудь елемент складається з інших елементів, то його можна розкрити і побачити ці елементи.

Якщо яка-небудь гілка може бути розкрита, то в своєму корені вона має квадратик з «+» всередині. Подвійне натискання на цій позначці розкриває гілку. Повторне подвійне натискання закриває розкриту гілку.

У центрі розташоване вікно з кодом програми. Внизу, під вікном коду, вікно повідомлень компілятора. Всі помилки, попередження, а також інша інформація, відображаються у цьому вікні.

На рис. 3.45 для наочності розкрито гілки всіх портів вводу/виводу і регістри, пов'язані з EEPROM. На рисунку можна бачити:

- повний склад керуючих регістрів для кожного з пристроїв;
- їх назви та адреси;
- склад та назву кожного біта (якщо біти мають свої назви).

У процесі налагодження в цьому вікні можна побачити не тільки назву і склад всіх ресурсів, але і їх вміст. Вміст буде відображатися як у шістнадцятковому вигляді, так і шляхом затемнення квадратиків, що відображають окремі біти конкретних регістрів.

Затемнений квадратик означає, що біт дорівнює одиниці. Світлий квадратик говорить про те, що біт дорівнює нулю. Також можна оперативно змінювати цей вміст прямо в цьому вікні. Для зміни значення біта досить натискання лівою кнопкою миші у відповідному квадратику.

Вікно порту має три різні регістри вводу-виводу (Port – регістр даних, Pin – ніжки порту, DDR – Data Direction Registers (регістри напрямку передачі даних)), які звичайно зв'язані з портом. Ніжки порту використовуються тільки для зчитування, в той час як регістр даних і регістри напрямку передачі даних – для зчитування/запису.

Pin не являється регістром, по цьому адресу здійснюється доступ до фізичних значень кожного із виводів порту. При зчитуванні з регістру даних Port, зчитуються дані з “регістру-защипку”, при зчитуванні з Pin зчитуються логічні значення, які знаходяться на виводах порту. Біти регістру DDR вибирають напрямок передачі даних. Якщо біт встановлений (1), вивід сконфігурований як вихід. Якщо біт скинутий (0) – вивід сконфігурований як вхід.

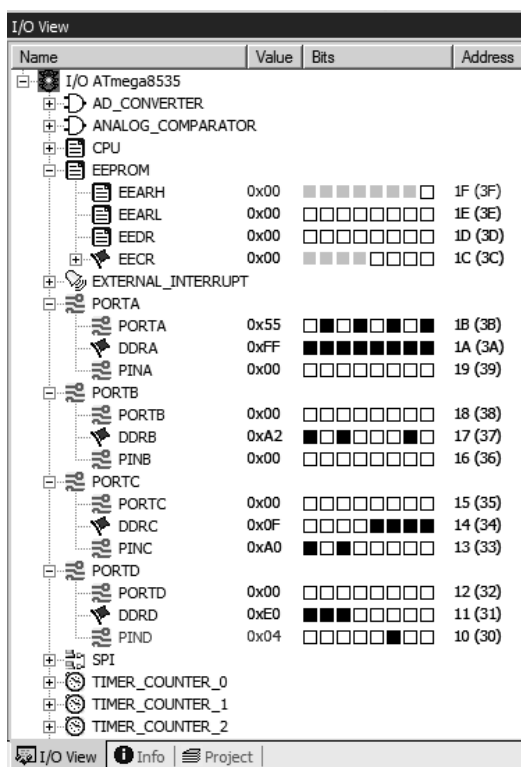


Рис. 3.45.

В центральному вікні відображається код програми. Кожна наступна команда, яка буде виконуватись, відмічена жовтим маркером. Можна відкрити вікно реєстрів (меню View → Registers), вікно для спостереження за змінними (View → Processor), вікно пам'яті (View → Memory). Вікно реєстрів складається з 32-х реєстрів і має вигляд, як показано на рис. 3.46.

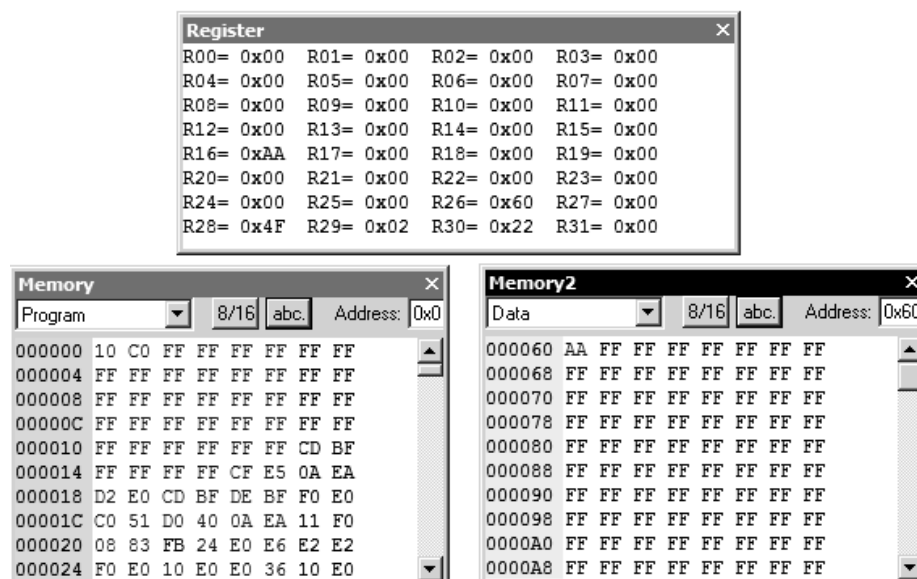


Рис. 3.46.

Вміст реєстрів можуть змінюватись після виконання команди. Значення реєстрів можна змінювати самотійно. Для цього потрібно поставити курсор на значення реєстру і нажати двічі ліву кнопку миші з

невеликою паузою між натисканням кнопки. Після цього вписати нове значення регістру у шістнадцятиричному коді.

Вікно пам'яті використовується користувачем для огляду і модифікації змісту змінних пам'яті, які використовуються у мітках (рис. 3.4). Одне й теж вікно використовується для огляду різних типів пам'яті: пам'яті даних, програмної пам'яті, пам'яті вводу-виводу і енергонезалежної пам'яті EEPROM. Тип пам'яті, який потрібно продивитись, може бути змінений в меню вибору пам'яті, яке знаходиться в верхньому лівому куті вікна пам'яті. Коли створюється нове вікно пам'яті, пам'ять даних (Data memory) створює стандартний тип пам'яті.

Адресація і зміст пам'яті завжди відображаються в 16-річному коді. Користувач може увімкнути огляд змісту пам'яті в символах ASCII, а також продивитись зміст пам'яті в 8-річному коді. Користувач може змінити зміст пам'яті подвійним натисканням кнопки миші (double click).

Вікно процесора вказує всю необхідну інформацію про команду, яка виконується в даний момент (рис. 3.47).

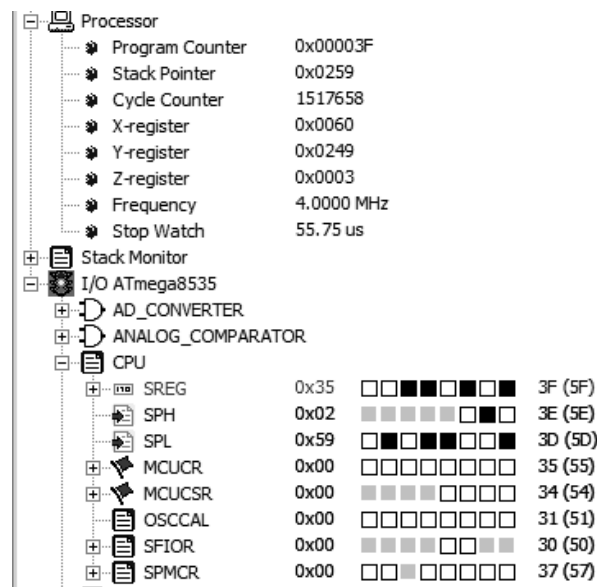


Рис. 3.47.

Програмний лічильник (Program Counter) вказує адресу команди, яка буде виконуватись. Він вказує адрес в 16-річному коді і цей код може бути змінений після зупинки команди. Коли значення програмного лічильника змінюється, поточна команда відкидається. Після зміни значення програмного лічильника програма переходить на наступну адресу.

Стековий вказівник (Stack Pointer) зберігає поточне значення стекового вказівника, які розташовані в області вводу-виводу. Якщо мітка має апаратний стек замість базового стеку SRAM, вона вказується в полі стекового вказівника. Значення стекового вказівника може бути змінено після того, як зупинено виконання команди.

Лічильник циклів (Cycle Counter) дає інформацію про кількість циклів тактового генератора, які пройшли після останнього скидання. Значення

лічильника циклів записується в десятковому коді і може бути змінений після зупинки команди.

Прапорці (Flags) показують поточне значення бітів статус регістру (Status register). Коли виконання команди зупинено, ці біти можуть бути змінені натисканням кнопки миші на визначений біт. Відмічений прапорець означає, що відповідний біт статус регістру має значення 1.

Команди керування виконанням програми використовуються для керування виконанням програми. Усі вони мають меню, так називаємі гарячі кнопки виклику і панель наладки (Debug toolbar).

Команда **Go** (пуск) викликається в меню Debug → Go (F5) або на панелі наладки натисканням кнопки . Вона запускає виконання програми. Виконується до тих пір, доки не користувач не зупинить виконання або доки не дійде до крапки переривання.

Команда **Break** (зупинити) викликається в меню Debug → Break (Ctrl+F5) або на панелі наладки натисканням кнопки . Зупиняє виконання програми. Після зупинки програми вся інформація в усіх вікнах відновлюється.

Команда **Trace info** (відслідкувати) викликається в меню Debug → Trace info (F11) або на панелі наладки натисканням кнопки . Ця команда виконує одну команду програми. За допомогою неї можна по командам перевірити роботу програми.

Команда **Step over** викликається в меню Debug → Step Over (F10) або на панелі наладки натисканням кнопки . При виклиці цієї команди програма буде виконуватись до тих пір, доки не дійде до команди виклику підпрограми чи функції (якщо така є), також це стосується крапок переривання.

Команда **Step out** викликається на панелі наладки натисканням кнопки . Якщо курсор знаходиться в функції чи підпрограмі, то при натисканні даної команди програма буде виконуватись, доки не дійде до команди виходу з функції чи підпрограми. Це також стосується крапок переривання.

Команда **Goto cursor** викликається в меню Debug → Run to Cursor (F7) або на панелі наладки натисканням кнопки . Якщо в програмі в якомусь місці поставити курсор і викликати цю команду, то програма виконується доти, доки не дійде до курсору. Команда припинить виконуватись, коли також дійде до крапки переривання.

Команда **Reset** викликається в меню Debug → Reset (Shift-F5) або на панелі наладки натисканням кнопки . При виклиці цієї команди курсор переходить на начало програми, значення статус-регістру, портів, процесору, таймера-лічильника обнуляються.

Для вибору контролера заходимо в меню Debug → AVR Simulator Options (рис. 3.48).

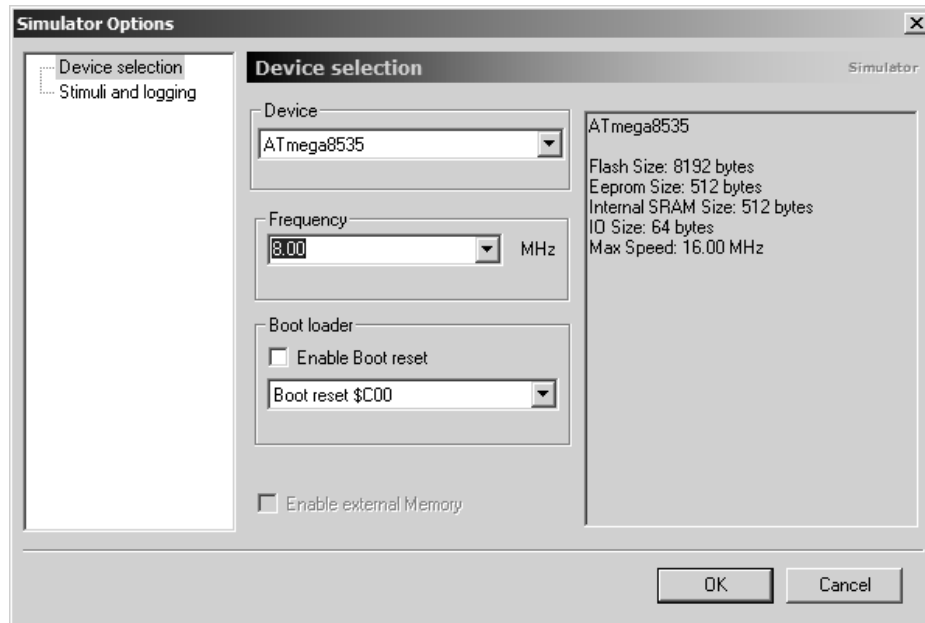


Рис. 3.48.

В меню Device вибираємо потрібний контролер. В меню Frequency задаємо частоту роботи мікроконтролера. У вікні праворуч можна побачити характеристики вибраного мікроконтролера та максимальну частоту його роботи.

3.11. Апаратні засоби підтримки розробок для мікроконтролерів

Мікроконтролери AVR мають кілька режимів програмування: паралельний режим з використанням підвищеної програмуючої напруги, послідовний режим з використанням підвищеної програмуючої напруги і режим послідовного внутрішньосхемного програмування.

Можливість внутрішньосхемного програмування мікроконтролерів AVR по послідовному синхронному інтерфейсу SPI дозволяє створювати прості і недорогі програматори. Такі програматори називають внутрішньосхемними (In-System Programmer, ISP).

Для реалізації внутрішньосхемного програмування компанія Atmel випускає і підтримує програматор ATAVRISP. ATAVRISP підтримує всі мікроконтролери, що випускаються на сьогоднішній день. Керування цим програматором здійснюється через COM-порт персонального комп'ютера. Живлення на ATAVRISP подається від цільового пристрою.



Рис. 3.49.

Програматор ATAVRISP реалізований на двох мікроконтролерах – 90S1200-12SC і 90S8535-8AC, і працює під керуванням AVR Studio. Керуюча програма викликається в меню Tools → STK500/AVRISP/JTAGICE → STK500/AVRISP/JTAGICE.

Вибравши одну з закладок у вікні керуючої програми, користувач одержує доступ до різних функцій програмування:

- Program. Вікно керуючої програми в цьому режимі представлене на рис. 3.50 Тут користувач вказує тип цільового мікроконтролера (Device), режим програмування (у нашому випадку ISP), а також файли, що містять дані для програмування Flash-ПЗП та EEPROM мікроконтролера. Тут же користувач може вказати керуючій програмі на необхідність автоматичного стирання пам'яті програм мікроконтролера перед черговим програмуванням, а також верифікацію записаного коду – після.
- Fuses. Дозволяє зчитати стан керуючих бітів мікроконтролера, і при необхідності, запрограмувати їх. Операцію установки FUSE-бітів потрібно завершувати натисканням кнопки Program на закладці Fuses.
- LockBits. Надає доступ до бітів захисту пам'яті програм мікроконтролера від несанкціонованого зчитування або модифікації. Операцію установки LOCK бітів потрібно завершувати натисканням кнопки Program на закладці LockBits.
- Advanced. Дозволяє зчитати сигнатуру цільового мікроконтролера. Тут може бути задане значення каліброваного байта для внутрішнього RC-генератора мікроконтролера.
- Board. Ця закладка призначена для керування STK500.
- Auto. Дозволяє вибрати кілька функцій програмування і верифікації і запускати їх у пакетному режимі.

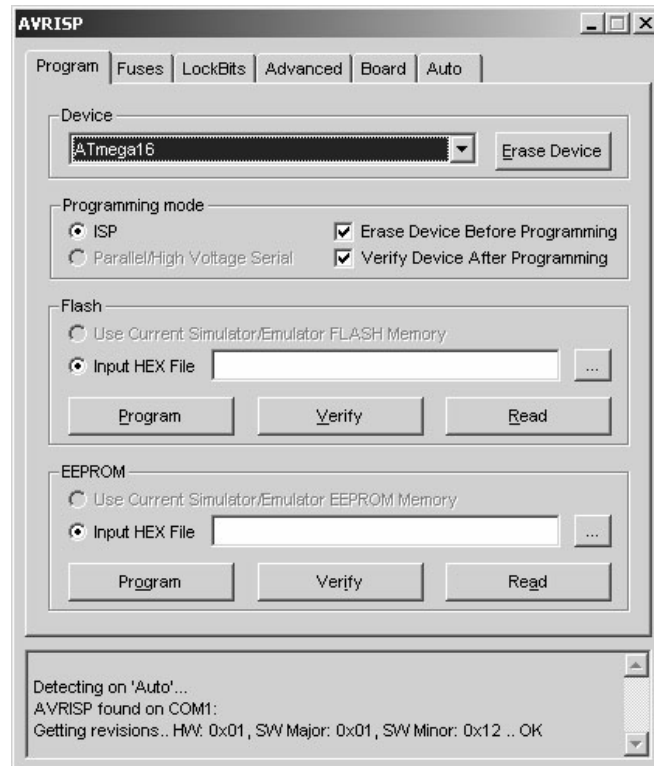


Рис. 3.50.

При кожному перемиканні з однієї закладки на іншу керуюча програма перевіряє зв'язок комп'ютера з програматором. Якщо програматор не відповідає на запит комп'ютера, у поле повідомлень вікна керуючої програми видається відповідне попередження.

При роботі з цільовим мікроконтролером, програматор ATAVRISP крім ліній внутрішньосхемного програмування (MOSI, MISO, CLK) використовує вивід RESET, GND, а також лінію живлення (VCC) по який здійснюється живлення програматора.

Уся необхідна інформація з підключення і використання програматора AVRISP включена у вбудований файл довідки AVR Studio.

Принцип внутрішньосхемної емуляції складається в заміні цільового мікроконтролера пристроєм, поведження якого з погляду системи, що відлагоджується, відповідає поведженню цільового мікроконтролера. Цей пристрій, що називається внутрішньосхемним емулятором, повинен не тільки адекватно відображати поведження цільового мікроконтролера, але і надавати розробнику можливість керування процесом налагодження і контролю за станом реальної мікропроцесорної системи. Для керування процесом налагодження при використанні сучасних емуляторів, як правило, використовується персональний комп'ютер з відповідним програмним забезпеченням.

На даний момент компанія Atmel пропонує 2 типи внутрісхемних емуляторів. Перший тип має на меті повну заміну мікроконтролера на якийсь пристрій, що емулює поведження реального мікроконтролера на

цільовій платі. Даний тип представлений двома внутрішньосхемними емуляторами ICE40 і ICE50.

Другий тип має на меті використання мікроконтролера AVR, що має всі необхідні для налагодження апаратні блоки і керується зовнішнім налагоджувачем. На даний момент цей тип налагоджувачів представлений пристроєм ATJTAGICE mkII.

Використання внутрішньосхемних емуляторів дозволяє налагоджувати пристрої, у яких задіяні периферійні вузли і режими мікроконтролерів, підтримка яких відсутня у програмному симуляторі.



Рис. 3.51.

Для підтримки різних мікроконтролерів є набір виносних плат (адаптерів) для пайки або установки в DIP-панель для мікроконтролера на цільовій платі (рис. 3.52). Для мікроконтролерів AVR, що не випускаються в DIP-корпусах, адаптер (ATADAP64BOT) призначений для припаювання на площадку під корпус TQFP. По типу підключеного адаптера ICE40/50 автоматично визначає, який мікроконтролер буде емулюватися.

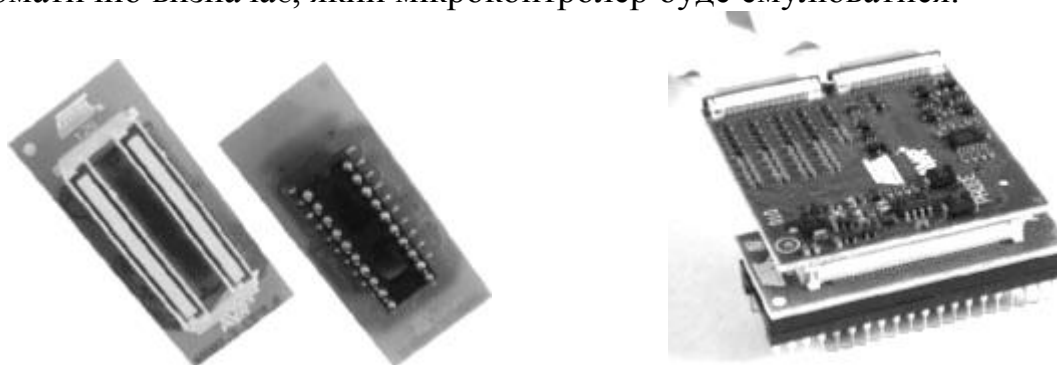


Рис. 3.52.

Емулятори ICE40 і ICE50 виконують наступні функції:

- емуляція аналогової і цифрової периферії мікроконтролера;
- робота в діапазоні напруг від 2,2 В до 5,5 В;
- перегляд значень регістрів вводу-виводу;
- запис траси;

- перегляд реєстрового файлу, пам'яті програм, даних і EEPROM;
- необмежена кількість точок останова;
- налагодження по тексту програми мовою високого рівня;
- лічильник часу виконання програми.

Керування налагодженням виконується в інтегрованому середовищі розробки AVR Studio. В якості програми для роботи мікроконтролера може використовуватися об'єктний код у форматі UBROF8 або COFF, а також програма на асемблері. Підключення емулятора до комп'ютера виконується через COM-порт або USB.

Однією зі зручних функцій, реалізованих у ICE40/50, є можливість запису траси, тобто збереження послідовності виконання інструкцій мікроконтролером. Аналіз записаної траси дозволяє виявити ділянки коду, виконання яких по тих або інших причинах не відбувається (Code Coverage Analysis). Такий аналіз дуже корисний при налагодженні програм, що містять кілька процедур обробки переривань, особливо якщо вкладеність переривань не дозволена.

У ICE50 реалізований режим діагностики вузлів внутрішньосхемного емулятора. Для здійснення діагностики необхідно установити тестову панель на плату відлагоджувального кристала та в AVR Studio вибрати пункт меню Tools → ICE50 Selftest. Після запуску процесу діагностування (кнопка Run) послідовно перевіряються всі вузли і режими внутрішньосхемного емулятора. Якщо при цьому визначається, що на платі керування утримується застарілий варіант внутрішнього мікрокоду (firmware), то програма діагностики запропонує автоматично оновити конфігурацію ПЛІС і пам'яті програм мікроконтролера. Також можливо зробити ручне відновлення мікрокоду ICE50, вибравши пункт меню Tools → ICE50 Upgrade.

Альтернативою внутрішньосхемної емуляції є *режим фонового налагодження*. В англomовній літературі цей режим позначається терміном "On-Chip Debugging", або OCD.

Різниця між внутрішньосхемною емуляцією і фоновим налагодженням полягає в наступному. Внутрішньосхемний емулятор з тим або іншим ступенем вірогідності імітує поведінку цільового мікроконтролера в відлагоджуваній системі, у той час як у режимі фонового налагодження програмний код виконується самим цільовим мікроконтролером. Таким чином, досягається повний збіг часових і електричних параметрів системи в налагоджувальному і штатному режимах.

Для підтримки режиму "On-Chip Debugging" (OCD) блок керування налагодженням повинен бути частиною самого відлагоджуваного мікроконтролера. Керування фоновим налагодженням у мікроконтролерах AVR здійснюється по JTAG-інтерфейсу, сумісному зі стандартом IEEE1149.1, або однопровідному інтерфейсу debugWIRE, що є власною розробкою компанії Atmel.

Блок OCD з керуванням по інтерфейсу JTAG міститься в мікроконтролерах ATmega128/CAN128, ATmega16/32/64, ATmega162, ATmega165/325, ATmega169/329, ATmega3250/3290. Блок OCD з керуванням по інтерфейсу debugWIRE міститься в мікроконтролерах ATmega48/88/168, ATtiny13, ATtiny2313, ATtiny45.

Під керуванням блоку OCD у мікроконтролерах AVR виконується виконання програмного коду і доступ до усіх регістрів мікроконтролера в режимі фонового налагодження.

Для роботи мікроконтролера в режимі внутрішньосхемного налагодження необхідно дозволити інтерфейс JTAG або debugWIRE. Для цього необхідно установити відповідні FUSE-біти. При дозволі інтерфейсу debugWIRE слід пам'ятати, що інтерфейс debugWIRE використовує вивід RESET мікроконтролера і, при дозволі інтерфейсу, стандартна функція виводу (скидання мікроконтролера) відключається, а значить, стає недоступною функція послідовного внутрішньосхемного програмування, оскільки вивід RESET використовується для введення мікроконтролера в режим послідовного програмування. Перепрограмування FUSE-бітів мікроконтролера можна зробити по інтерфейсу debugWIRE або в режимі паралельного програмування.

Пристроєм, що реалізує протокол керування, є ATJTAGICE mkII (ATJTAGICE2). Внутрішньосхемний емулятор ATJTAGICE mkII підключається до персонального комп'ютера по COM або USB інтерфейсах. При підключенні по USB інтерфейсу не потрібне використання зовнішнього джерела живлення. Робити з'єднання емулятора з комп'ютером і цільовим мікроконтролером необхідно при виключеному живленні всіх трьох пристроїв (при підключенні по інтерфейсі USB досить установити вимикач живлення ATJTAGICE mkII у положення ВИКЛЮЧЕНЕ). У протилежному випадку можна вивести з ладу вихідні каскади емулятора.

При використанні інтерфейсу USB для підключення ATJTAGICE mkII до комп'ютера, передача команд виконується істотно швидше, ніж при використанні COM-порту.

Як керуюча програму для роботи ATJTAGICE mkII і OCD мікроконтролера використовується інтегроване середовище розробки AVR Studio. Більш докладну інформацію з можливостей і використання ATJTAGICE mkII можна знайти у вбудованому файлі довідки AVR Studio.

Крім забезпечення режиму фонового налагодження, емулятор ATJTAGICE mkII може бути використаний у якості внутрішньосхемного програматора для мікроконтролерів AVR. Програмування може здійснюватися по інтерфейсах JTAG або SPI. Для програмування, а також налагодження, по інтерфейсу JTAG одночасно декількох пристроїв потрібно з'єднати пристрої в дейзі-ланцюжок. Після цього необхідно вказати AVR Studio наявність декількох мікроконтролерів у меню Tools → STK500/AVRISP/JTAGICE → STK500/AVRISP/JTAGICE → Advanced → JTAG Daisy Chain Settings.

Альтернативою описаним вище емуляторам і відлагоджувачу може служити інтегрований у AVR Studio програмний симулятор, що робить емуляцію програмним способом, або спеціалізовані середовища моделювання електронних пристроїв, наприклад Proteus Isis.

Питання до розділу 3

1. Яка максимальна навантажувальна здатність виводу мікроконтролера?
2. Що відбудеться якщо в порт мікроконтролера, який налаштований на вхід, записати дані?
3. Як можна вийти із ситуації, якщо кількості виводів мікроконтролера не вистачає для підключення тактових кнопок та індикаторів?
4. Для чого у схемі стенда використовуються додаткові регістри?
5. Яким чином можна вирішити проблему тремтіння контактів? Які способи усунення тремтіння контактів існують?
6. Розрахунок часу регенерації для динамічного методу відображення.
7. Обґрунтувати необхідність застосування різних методів відображення інформації.
8. Схемотехнічні рішення для побудови схем відображення інформації.
9. Яким чином здійснюється виведення на індикатори дійсних чисел?
10. Принцип програмного регулювання яскравості семисегментних індикаторів.
11. Особливості управління семисегментними індикаторами при їх підключенні на загальну шину.
12. У чому полягає відмінність в управлінні семисегментними індикаторами із загальним анодом та загальним катодом?
13. Яким чином можна керувати сегментами індикатора, якщо номінальний струм індикатора перевищує максимально допустимий струм виводу мікроконтролера?
14. Призначення та сфера застосування передачі даних по послідовному інтерфейсу.
15. Фізична характеристика сигналів в інтерфейсі RS232.
16. Часові параметри, визначення швидкості обміну інформацією по інтерфейсу RS232.
17. Протокол обміну даними по інтерфейсу RS232.
18. Синхронні та асинхронні канали послідовної передачі даних.
19. Регістри, що використовуються при ініціалізації послідовного порту.
20. Методи і типи АЦП.
21. Статичні параметри АЦП.
22. Поняття дискретності, квантування, роздільна здатність АЦП.
23. Характеристика перетворення, диференційна нелінійність АЦП, відхилення коефіцієнта перетворення.
24. Час перетворення АЦП, час затримки запуску, час циклу перетворення, максимальна частота перетворення.
25. Фактори, що впливають на похибку АЦП.

- 26.Опишіть послідовність дій для визначення номера натиснутої кнопки на матричній клавіатурі.
- 27.У яких режимах працюють порти мікроконтролера при скануванні матричної клавіатури?
- 28.Чому при підключенні клавіатури до мікроконтролера необхідно використовувати підтягуючі резистори?
- 29.Чи можливо одночасно підключити матричну клавіатуру і семисегментний індикатор до одного порту мікроконтролера? Обґрунтуйте відповідь.
- 30.В чому полягають переваги та недоліки матричної клавіатури в порівнянні з дискретними тактовими кнопками?
- 31.Яку кількість виводів мікроконтролера необхідно задіяти для підключення клавіатури 4×4 ?
- 32.Структура системи переривань.
- 33.Призначення, приклади застосування системи переривання.
- 34.Регістри управління, розподіл пам'яті в ATmega128.
- 35.Переривання від таймерів, переривання від послідовного приймача/передавача.
- 36.Програмне усунення явища тремтіння контактів.
- 37.Методи і типи частотного та часового перетворення.
- 38.Параметри частотного перетворення.
- 39.Фактори, що впливають на похибку частотного перетворення.
- 40.Поняття роздільної здатності частотного перетворення.
- 41.Алгоритм вимірювання періоду імпульсного сигналу.
- 42.Характеристика перетворення, нелінійність частотного перетворення.
- 43.Апаратні реалізації частотного та часового перетворення.
- 44.Приклади практичного застосування частотного та часового перетворення.
- 45.Опишіть структуру РКІ. Які види пам'яті є в РКІ?
- 46.Особливості роботи та управління РКІ в різних режимах (4-х і 8-бітні шини).
- 47.Намалюйте схему підключення РКІ до мікроконтролера.
- 48.Яку послідовність команд необхідно сформувати для ініціалізації РКІ? В чому полягає суть процесу ініціалізації?
- 49.Наведіть приклади використання алфавітно-цифрових РКІ.
- 50.Яка організація адресного простору в графічному РКІ?
- 51.Назвіть послідовність команд, яку необхідно сформувати для виведення інформації на екран графічного РКІ.
- 52.У чому полягає відмінність у принципах виведення текстової та графічної інформації?
- 53.Яким чином можна виконати інверсію зображення на екрані?

Задачі до розділу 3

1. Написати програму, що виконує додавання 2-х чисел. Результат виводити в двійковому коді на світлодіодні індикатори.
2. Вивести в порт з інтервалом 1 секунда послідовність чисел, що представлені у десятковому коді: 1, 2, 4, 8, 16, 32, 64, 128. Відображати стан порту на світлодіодах.
3. Реалізувати двійковий лічильник. Значення лічильника відображати на світлодіодних індикаторах. Інкремент здійснюється автоматично з періодом 200 мс. При досягненні значення 100 лічильник починає рахувати у зворотньому напрямку до 0. Потім цикл повторюється.
4. Змінити попередню програму таким чином, щоб при збільшенні значення лічильника швидкість інкременту збільшувалася, а при зменшенні значення лічильника швидкість декременту зменшувалася.
5. Реалізувати ефект «гусениця» на світлодіодних індикаторах.
6. Реалізувати лічильник з коефіцієнтом перерахунку $M = 147$.
7. Створити ефект «хвиля» на семисегментних індикаторах.
8. Реалізувати біжучий рядок на семисегментних індикаторах. Виводити наступний текст: 123456789AbCdEF.
9. Написати програму, яка виводить число 4321 на семисегментний індикатори. Кожна цифра «виїжджає» справа і займає свою позицію. Нова цифра не з'являється до тих пір, поки попередня не опиниться на своєму місці.
10. Відображати на матричному індикаторі числа від 0 до 9, дублювати ці числа на семисегментного індикаторі.
11. Реалізувати біжучий рядок на матричному індикаторі. Виводити своє ім'я.
12. Написати програму, яка послідовно запалює всі елементи матричного індикатора, а потім також послідовно, один за іншим їх гасить.
13. Реалізувати таку програму: по контуру матричного індикатора рухається змійка (довжина 5 точок).
14. Написати програму – кодовий замок. Спочатку на семисегментних індикаторах відображається CLOS, на світлодіодах горить крайній правий світлодіод. Після введення правильного коду (за допомогою кнопок), на семисегментних індикаторах відображається OPEN. При натисканні правильної кнопки на світлодіодних індикаторах запалюється наступний світлодіод. При натисканні помилкової кнопки загоряється крайній правий світлодіод.
15. Розробити реверсивний лічильник. При натисканні на одну кнопку лічильник інкрементується, при натисканні на іншу кнопку значення лічильника зменшується. Натискання третьої кнопки обнуляє лічильник.

16. Розробити суматор напівбайтних слів. Старша тетрада додається до молодшої тетради, результат виводиться на семисегментні індикатори.
17. Реалізувати 5 світлових ефектів: 3 на світлодіодах, по 1 на семисегментному та матричному індикаторі. Вид ефекту вибирається натисканням відповідних кнопок.
18. Реалізувати гру «змійка». Напрямок руху задається 4 кнопками.
19. Написати програму, яка посилає в комп'ютер номер натиснутої кнопки.
20. Написати програму, яка на семисегментних індикаторах виводить прийняті значення (цифри від 0 до 9). Після прийому чергового байта, він виводиться на наступний семисегментний індикатор.
21. Написати програму, яка формує синусоїдальний сигнал і відправляє значення в комп'ютер.
22. Написати програму, яка після прийому значення з комп'ютера (від 0 до 9) встановлює відповідну яскравість / частоту мерехтіння світлодіодів.
23. Написати програму, яка на чотирьох семисегментних індикаторах виводить значення напруги, зчитане з АЦП (у вольтах).
24. При натисканні на кнопку запустити перетворення АЦП і вивести результат на чотирьох семисегментних індикаторах.
25. Написати програму, яка управляє яскравістю / частотою мерехтіння світлодіодів в залежності від значення, зчитаного з АЦП.
26. Виміряти час, протягом якого значення напруги на вході АЦП перевищує деякий заданий рівень. Відобразити значення інтервалу часу на семисегментних індикаторах.
27. Написати програму, яка виконує аналого-цифрове перетворення і відправляє отримане значення в комп'ютер.
28. Написати програму – калькулятор. Вид операції змінюється натисканням на кнопку * клавіатури, введення здійснюється натисканням кнопки #. На матричному індикаторі відображати вид операції.
29. Написати програму управління яскравістю світлодіодів. Значення яскравості задається за допомогою клавіатури і відображається на семисегментних індикаторах у відсотках (від 0 до 100). Після натискання на кнопку # виконується зміна яскравості світлодіодів в залежності від введенного числа. Передбачити захист від введення неправильних даних.
30. Написати програму, яка збільшує значення змінної на 1 кожен секунду, і виводить результат на панель світлодіодів кожен секунду. Використовувати таймер-лічильник.
31. Написати програму, яка працює як зворотний таймер, зменшує значення змінної кожен секунду на 1. Використовувати таймер-лічильник.

32. Написати програму, яка дозволяє виміряти швидкість реакції людини. Після натискання на першу кнопку всі світлодіоди гаснуть, і через випадковий проміжок часу (не більше 5 секунд) вони спалахують. Користувач натискає другу кнопку й на семисегментних індикаторах відображається час (в мілісекундах) від моменту запалювання світлодіодів до моменту натискання на кнопку.
33. Написати програму – годинник. В якості початкового значення часу прийняти поточний час. Час має бути представлено у форматі: ГГ: ХХ: СС: МС і виводитися у верхньому рядку алфавітно-цифрового рідкокристалічного індикатора.
34. Написати функцію, яка перетворює число типу `int` в рядок символів (за власним алгоритмом). Порівняти швидкість виконання стандартної функції з бібліотеки (`itoa(...)`) з розробленою.
35. Створити власні 8 символів і вивести їх через пробіл в нижній частині рідкокристалічного індикатора.
36. Розробити вольтметр. У верхній частині РКІ відображати напругу на першому каналі АЦП, в нижньому рядку відображати значення напруги на другому каналі АЦП. Змінювати рівень напруги можна за допомогою змінних резисторів.
37. Розробити український або російський шрифт. Вивести інформацію про себе з використанням розробленого шрифту.
38. Написати функцію, яка інвертує зображення на екран графічного РКІ.
39. Написати функцію, яка очищає зазначену область екрану. Параметри функції – координати верхнього лівого і нижнього правого кутів.
40. Написати програму що виводить на екран графічного РКІ вашу фотографію.

4. ПОСЛІДОВНИЙ ПЕРИФЕРІЙНИЙ ІНТЕРФЕЙС SPI

Схеми послідовного периферійного інтерфейсу забезпечують синхронний послідовний зв'язок, в якому можлива передача даних у зовнішні пристрої в режимі ведучого й веденого. Якщо в системі присутній зовнішній процесор, за допомогою цього інтерфейсу може бути забезпечений міжпроцесорний зв'язок.

Основні характеристики інтерфейсу SPI:

- повнодуплексний 3-провідний синхронний обмін даними;
- режим роботи ведучий або ведений;
- обмін даними з переданими першими старшим або молодшим бітами;
- чотири програмовані швидкості обміну даними;
- переривання по закінченню передачі;
- вихід з режиму зниженого споживання енергії (тільки в режимі веденого).

4.1. Блок-схема SPI

Апаратна частина послідовного периферійного інтерфейсу представляє собою регістр зсуву, що послідовно передає біти даних в аналогічні інтерфейси інших пристроїв. Під час процесу передачі один із пристроїв працює як ведучий, забезпечуючи контроль над потоком даних, у той час як інші, ведені пристрої, приймають або передають дані під керуванням ведучого. Функції провідного пристрою інтерфейсу можуть передаватися від одного процесора іншому (мультимастерний протокол у порівнянні з одномастерним, коли тільки один процесор завжди працює в якості ведучого, а всі інші в якості ведених), і ведучий пристрій може вести передачу даних декільком веденим пристроям одночасно.

З'єднання між ведучим й веденим пристроями показано на рис. 4.1.

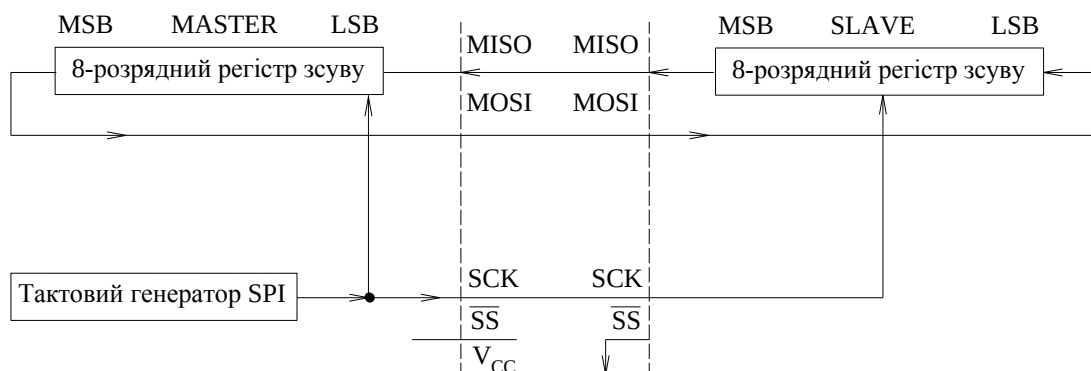


Рис. 4.1.

Два регістри зсуву ведучого і веденого можна розглядати як один рознесений 16-розрядний циклічний регістр зсуву. При зсуві даних із ведучого мікроконтролера у ведений одночасно відбувається зсув даних з веденого мікроконтролера у ведучий, тобто протягом одного циклу зсуву відбувається обмін даними між ведучим й веденим мікроконтролерами.

Вивід SCK є виходом тактового сигналу ведучого мікроконтролера і входом тактового сигналу веденого. При записі ведучим мікроконтролером даних у регістр SPI починає працювати тактовий генератор, і записані дані передаються з виходу MOSI ведучого мікроконтролера на вхід MOSI веденого. Після передачі одного байта генератор зупиняється, встановлюючи прапор завершення передачі SPIF. Якщо в регістрі SPCR встановлений прапор дозволу переривання (SPIE), то відбудеться запит переривання. Вхід вибору веденого SS, для вибору індивідуального SPI пристрою в якості веденого, встановлюється на низький рівень. При установці високого рівня на виводі SS порт SPI деактивується. Режим ведучий/ведений може бути встановлений програмним способом установкою або очищенням біта MSTR у регістрі керування SPI. На рис. 4.2 представлена блок-схема SPI.

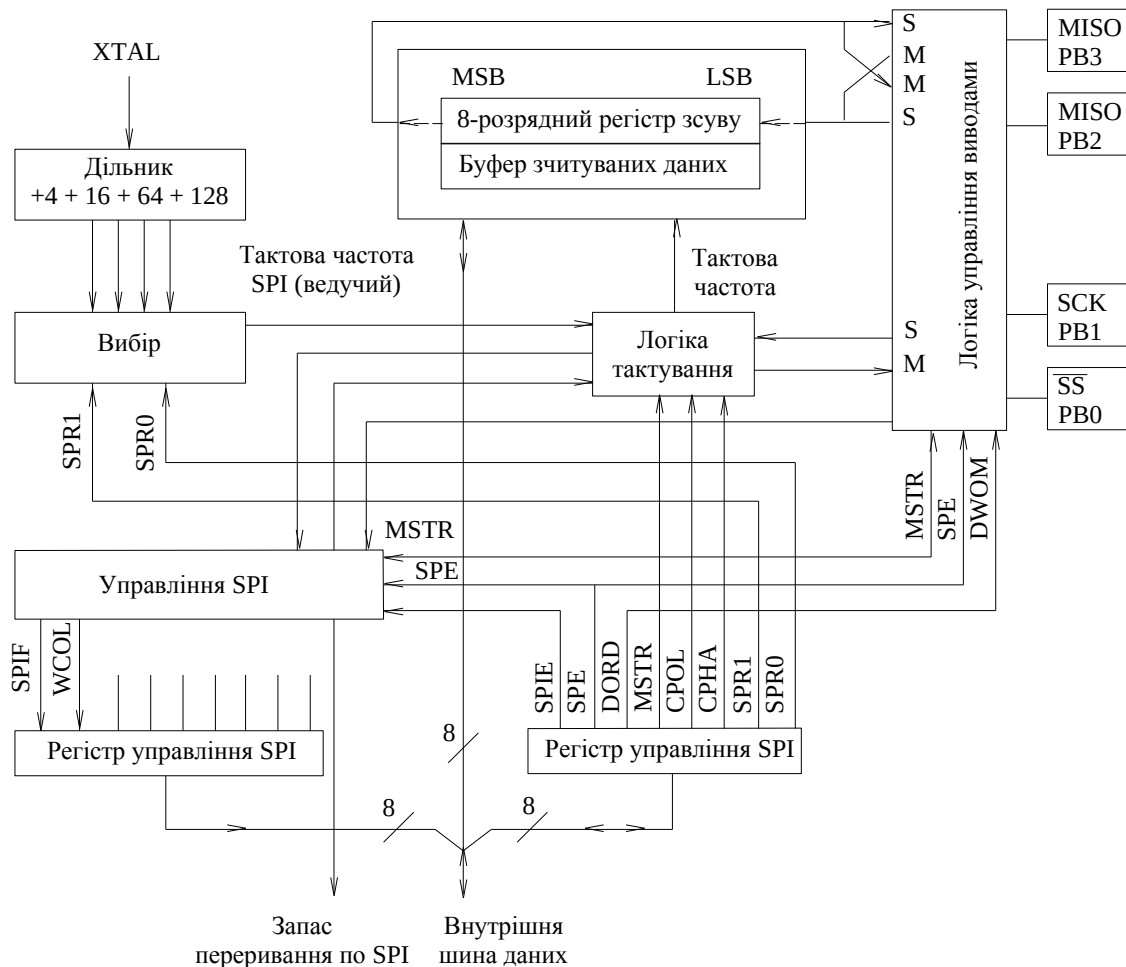


Рис. 4.2.

При роботі SPI ведучим (біт MSTR регістра SPCR встановлений), користувач має можливість установити напрямок роботи виводу #SS. Якщо вивід #SS налаштований як вихід, то вивід є виводом загального призначення і він не активується системою SPI. Якщо ж вивід #SS налаштований як вхід, то для забезпечення роботи ведучого SPI він повинен утримуватися на високому рівні. Якщо в режимі ведучого вивід #SS є входом, і зовнішньою периферійною системою на нього поданий низький рівень, то SPI сприйме його як звернення іншого провідного SPI до себе, як до веденого. Щоб уникнути конфліктної ситуації на шині, система SPI виконує наступні дії:

1. Біт MSTR у регістрі SPCR очищається й SPI система стає веденою. Результатом цього є те, що виводи MOSI та SCK стають входами.
2. Встановлюється прапор SPIF регістра SPSR, і якщо дозволено переривання SPI, починається виконання підпрограми обробки переривання.

Якщо SPI працює в режимі веденого, то вивід #SS постійно працює на вхід. Якщо на вивід #SS поданий низький рівень, то SPI активується і вивід MISO стає виходом. Всі інші виводи є входами. Якщо вивід #SS утримується на високому рівні, то всі виводи є входами, SPI пасивний, і це означає, що він не буде одержувати вхідних даних.

Існує чотири варіанти комбінації фази й полярності SCK відносно послідовних даних, які встановлюються керуючими бітами CPHA та CPOL. Формати передачі даних показані на рис. 4.3 і рис. 4.4.

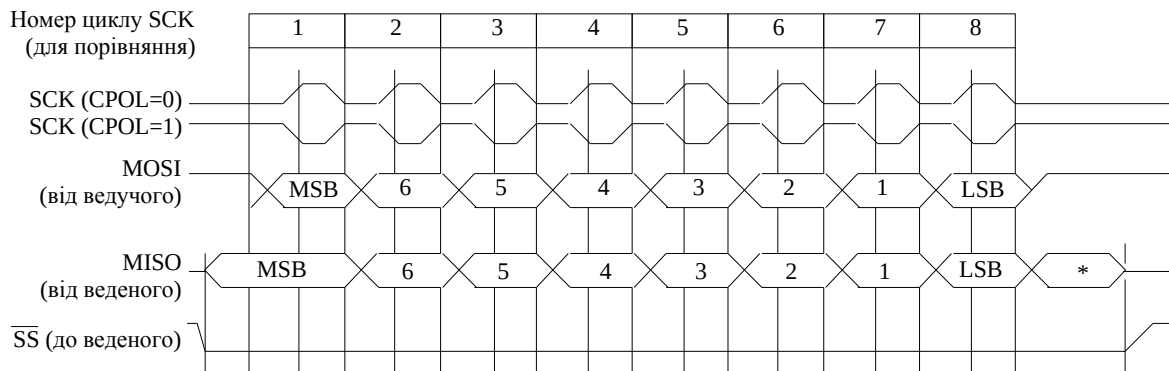


Рис. 4.3.

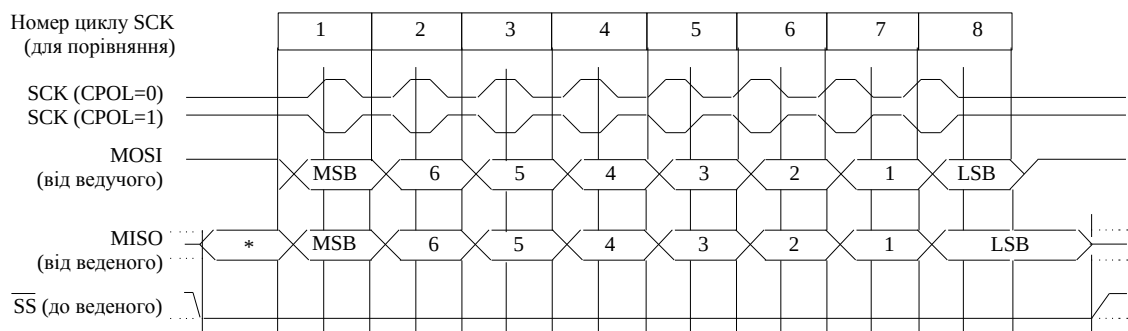


Рис. 4.4.

4.2. Конфігурація SPI

Після переходу SPI у режим веденого вивід SS завжди працює як вхід. У цьому випадку SPI активується, якщо на вхід SS подати низький рівень, а вивід MISO стає виходом. Всі інші виводи працюють як входи. Якщо на вхід SS подати високий рівень, то всі виводи стануть входами і SPI перейде в пасивний стан, у якому блокується прийом вхідних даних. Логіка SPI скидається, як тільки на вивід SS подається високий логічний рівень.

Опис регістрів SPI для контролерів AVR приводиться нижче.

Регістр SPCR – регістр керування:

Розряд	7	6	5	4	3	2	1	0
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
Зчитування /запис	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.
Початкове значення	0	0	0	0	0	0	0	0

SPIE – дозвіл переривання SPI.

SPE – SPI включений. Якщо встановлено цей біт, то виводи #SS, MISO, MOSI та SCK працюють як виводи SPI, інакше – як прості виводи порту.

DORD – напрямок передачі даних. Якщо цей біт встановлено, то передача починається з молодшого біта, якщо скинуто – зі старшого.

MSTR – якщо встановлено, контролер працює як Master, якщо скинуто – як Slave. Керується також виводом #SS, якщо він налаштований на вхід – при подачі "0" на #SS біт MSTR скидається.

CPOL – визначає рівень на виході SCK у режимі очікування – SCK = CPOL.

CPHA – якщо встановлено, передача і прийом біта відбуваються по спаду сигналу, якщо скинуто – по фронту.

SPR1, SPR0 – дільник тактової частоти (F – частота генератора контролера).

SPR1	SPR0	Частота
0	0	F/4
0	1	F/16
1	0	F/64
1	1	F/128

Регістр SPSR – регістр стану:

Розряд	7	6	5	4	3	2	1	0
	SPIF	WCOL	–	–	–	–	–	SPI2X
Зчитування /запис	Зч.	Зч.	Зч.	Зч.	Зч.	Зч.	Зч.	Зч./Зап.
Початкове значення	0	0	0	0	0	0	0	0

SPIF – прапор переривання по закінченню обміну даними. Встановлюється й скидається апаратно. Якщо біт SPIE у регістрі SPCR встановлений, і дозволене глобальне переривання, генерується сигнал переривання. Біт SPIF може бути очищений також при першому зчитуванні регістра стану.

WCOL – прапор помилки при записі. Встановлюється, коли прийнятий байт накладається на ще не зчитаний з регістра даних. Скидається зчитуванням регістра стану.

Біти 5..1 – зарезервовані біти. При зчитуванні завжди покажуть 0.
SPI2X – біт подвоєння швидкості SPI.

Регістр SPDR – регістр даних:

Розряд	7	6	5	4	3	2	1	0
	Ст. розр.							Мол. розр.
Зчитування /запис	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.
Початкове значення	х	х	х	х	х	х	х	х

Регістр даних SPI представляє собою регістр із можливістю зчитування/запису і призначений для пересилання даних. Запис у регістр SPDR ініціює передачу даних, зчитування регістра призводить до зчитування даних із регістра зсуву приймача. Передача та прийом тактуються по лінії SCK відповідно до конфігурації, що задана в регістрі керування. Після того, як байт переданий, тактування припиняється, у регістрі стану встановлюється прапор SPIF і викликається переривання по закінченню передачі, якщо воно дозволено. Запис у регістр даних передаючого контролера неможливий, поки не звільнений регістр зсуву SPI.

В SPI установка й передача даних відбувається по протилежних фронтах тактових імпульсів. Такий поділ операцій зсуву та передачі має певні переваги, дозволяючи уникати жорсткого часового режиму між цими двома операціями. Це дозволяє полегшити задачу синхронізації при розробці інтегральних схем або плат. Але, з іншого боку, через комбінації полярності й фази синхроімпульсів виникає чотири режими роботи, і ведучому пристрою доводиться налаштовуватися на той режим, що використовується веденим.

4.3. Мікросхеми пам'яті SD/MMC

Зараз SD-карти пам'яті (Secure Digital Memory Card) є найбільш популярними картами пам'яті для мобільних пристроїв. SD-карти (далі SDC) були розроблені як сумісні з MMC-картами (Multi Media Card) (далі MMC), при цьому SDC-сумісні пристрої також можуть використовувати MMC, але з деякими застереженнями. Також існують зменшені версії, такі як RS-MMC, miniSD й microSD, з аналогічними функціями. MMC/SDC має

вбудований мікроконтролер, всі засоби керування flash-пам'яттю (стирання, зчитування, запис і захист від помилок) повністю зосереджені усередині карти. За замовчуванням дані передаються між картою й ведучим контролером блоками по 512 байт, так що з погляду прикладних програм її можна розглядати як звичайний жорсткий диск. На рис. 4.5 представлений зовнішній вигляд карт пам'яті SDC та MMC.



Рис. 4.5.

На рис. 4.6, показані контактні поверхні SDC й MMC. MMC має сім контактних площадок, а в SDC їх дев'ять, дві з яких додані до семи площадок MMC. Три контакти в кожній картці зайняті як виводи живлення, тому кількість робочих сигналів становить відповідно чотири або шість. Обмін даними між ведучим контролером і картою здійснюється у вигляді послідовної синхронної передачі.

Діапазон робочої напруги живлення зазначений у регістрі OCR. Цого необхідно зчитати, щоб переконатися в правильності обраної напруги. Однак живляча напруга може приймати фіксоване значення, оскільки MMC/SDC працюють у діапазоні напруг від 2,7 до 3,6 В. Оскільки струм споживання може досягати декількох десятків міліампер, ведучий контролер повинен бути здатний забезпечити струм 100 мА.

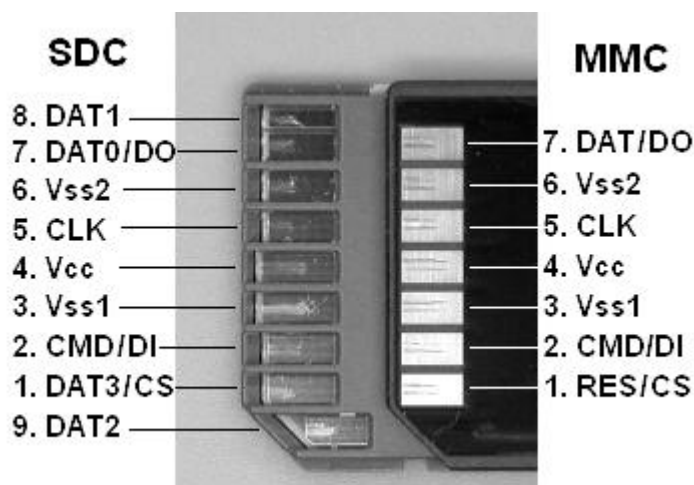


Рис. 4.6.

Режим SPI є альтернативним режимом роботи з MMC/SDC, більш простим у порівнянні з використанням їхнього рідного інтерфейсу. MMC/SDC можна підключити через універсальний порт SPI або порт GPIO (універсальний порт вводу/виводу), вбудований у більшість мікроконтролерів. Найбільш підходящим режимом для роботи з MMC/SDC по SPI є режим "SPI 0" (позитивний синхроімпульс, передача по фронту синхроімпульсу, зсув по зрізу). Режим "SPI 3" теж підходить для цих цілей.

У режимі SPI напрямок даних на сигнальній лінії фіксований, і дані передаються послідовно по байтах. Кадр команди, переданий від ведучого контролера до карти, має фіксовану довжину в 6 байтів і формат, представлений на рис. 4.7. Оскільки вся передача даних керується послідовними синхроімпульсами, що генеруються ведучим, останній повинен продовжувати зчитування байтів, поки не одержить дійсну відповідь. Час відповіді на команду (NCR) становить від 0 до 8 байтів для SDC, і від 1 до 8 байтів для MMC.

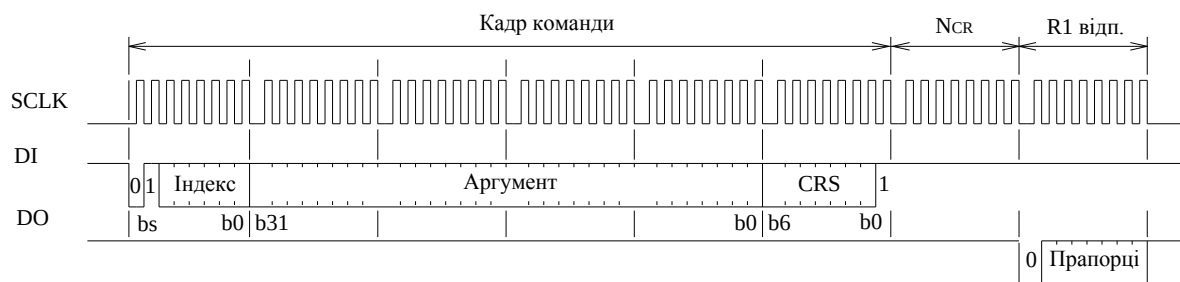


Рис. 4.7.

Сигнал CS повинен утримуватися в стані низького рівня протягом всієї транзакції (команда, відповідь і передача даних, якщо такі є). Поле CRC в режимі SPI не обов'язкове, але його присутність необхідна, щоб скласти кадр команди.

4.3.1. Команди керування

Кожна команда представлена у вигляді аббревіатури (наприклад, GO_IDLE_STATE), або у вигляді позначення CMD#, де # – номер індексу команди, що може приймати значення від 0 до 63. У наведеній нижче таблиці описана тільки група команд, що використовується для зчитування/запису та ініціалізації карти. Більш детальну інформацію із всіх команд ви можете знайти в специфікаціях від MMCA та SDCA.

Табл. 4.1.

Індекс команди	Аргумент	Відп.	Дані	Абревіатура	Опис
CMD0	Hi(0)	R1	Hi	GO_IDLE_STATE	Програмне скидання.
CMD1	Hi(0)	R1	Hi	SEND_OP_COND	Ініціювати процес ініціалізації.
ACMD41(*1)	Hi(0)	R1	Hi	APP_SEND_OP_COND	Тільки для SDC. Ініціювати процес ініціалізації.
CMD9	Hi(0)	R1	Так	SEND_CSD	Зчитати регістр CSD.

CMD10	Hi(0)	R1	Так	SEND_CID	Зчитати регістр CID.
CMD12	Hi(0)	R1b	Hi	STOP_TRANSMISSION	Запинити зчитування даних.
CMD17	Адреса [31:0]	R1	Так	READ_SINGLE_BLOCK	Зчитати блок.
CMD18	Адреса [31:0]	R1	Так	READ_MULTIPLE_BLOCK	Зчитати декілька блоків.
CMD23	Кількість блоків [15:0]	R1	Hi	SET_BLOCK_COUNT	Тільки для MMC. Визначити кількість блоків для передачі з наступною командою багатоблокового зчитування/запису.
ACMD23(*1)	Кількість блоків [15:0]	R1	Hi	SET_WR_BLOCK_ERASE_COUNT	Тільки для SDC. Визначити кількість блоків для попереднього стирання для наступної багато блокової команди запису.
CMD24	Адреса [31:0]	R1	Так	WRITE_BLOCK	Записати блок.
CMD25	Адреса [31:0]	R1	Так	WRITE_MULTIPLE_BLOCK	Записати декілька блоків.
CMD55(*1)	Hi(0)	R1	Hi	APP_CMD	Команда, що визначається програмою.
CMD58	Hi(0)	R3	Hi	READ_OCR	Зчитати OCR.
*1: ACMD означає послідовність команд CMD55, CMD.					

У стані очікування карта приймає тільки команди CMD0, CMD1 та CMD58, будь-які інші команди будуть відхилені. Коли карта виявляє команду CMD1, вона починає ініціалізацію. Щоб визначити, чи завершена ініціалізація, ведучий контролер повинен повторювати посилку команди CMD1 і перевіряти відповідь. Як тільки карта успішно була ініційована, біт "In Idle State" у відповіді R1 буде скинутий (R1 = 0x00). Процес ініціалізації може зайняти кілька сотень мілісекунд (більші за об'ємом карти мають тенденцію до більших витрат часу). Після закінчення ініціалізації будуть прийматися й команди зчитування/запису. У цей час можна зчитати OCR й CID для підтвердження діапазону робочої напруги, ємності карти або будь-якої іншої характеристики.

4.3.2. Протокол обміну даними

У транзакції з передачею даних після відповіді на команду може бути переданий/отриманий один або більше блоків даних. Блок даних передається як пакет даних, що складається з маркера (Token), блоку даних (Data Block) і контрольної суми (CRC). Формат пакета даних показаний на рис. 4.8. Існує три маркери даних. Один з них – маркер "Stop Tran", який означає кінець багатоблокового запису, використовується у вигляді одиночного байта без блоку даних та CRC.

Пакет даних

Маркер даних	Блок даних	CRC
1 байт	1- 2048 байт	2 байта

Маркер даних

1 1 1 1 1 1 1 0	Маркер даних для CMD17/18/24
1 1 1 1 1 1 0 0	Маркер даних для CMD25
1 1 1 1 1 1 0 1	Маркер зупинки CMD25

Відповідь

X	X	X	0	Стан	1

- 0 1 0 — Дані прийняті
1 0 1 — Дані відхилені через помилку контрольної суми
1 1 0 — Дані відхилені через помилку запису

Маркер помилки

0	0	0	Прапорці
			Помилка
			Помилка СС
			Помилка карти ЕСС
			Вихід з діапазона
			Карта заблокована

Рис. 4.8.

Одноблокове зчитування (рис. 4.9) ініціюється командою CMD17. Її аргумент задає комірку, з якої потрібно почати зчитування. Читання здійснюється по байтах. Адреса повинна бути байт-байт-орієнтованим, тобто якщо ви вказуєте адресу сектора, що задається декількома старшими бітами, вам необхідно доповнити його нулями для одержання повних чотирьох адресних байтів.

У відповідь на команду CMD17 карта передає ведучому контролеру пакет даних. Після виявлення правильного маркера даних ведучий контролер приймає наступний за ним блок даних і два байти CRC, які необхідно прийняти, навіть якщо CRC не використовується. По замовчуванню розмір блоку дорівнює 512 байт, але його можна змінити командою CMD16. Якщо під час операції зчитування відбулася яка-небудь помилка, замість пакета даних буде повернутий маркер помилки (рис. 4.8).



Рис. 4.9.

За допомогою команди багатоблокового зчитування CMD18 можна прочитати з карти послідовність із декількох блоків, починаючи із заданої адреси (рис. 4.10). Якщо перед цією командою за допомогою команди CMD23 (тільки для MMC) не було задане число переданих блоків, буде ініційоване необмежене багатоблокове зчитування, тобто операція зчитування буде тривати, поки ведучий контролер не перерве її командою CMD12.

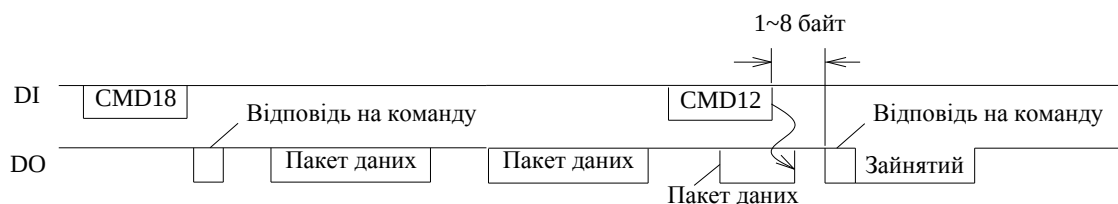


Рис. 4.10.

Одноблоковий запис демонструє часова діаграма, що представлена на рис. 4.11. Байт, що отримується відразу після передачі CMD12, є заповнюючим, і його не потрібно враховувати. Після цього байта приймається відповідь на команду.

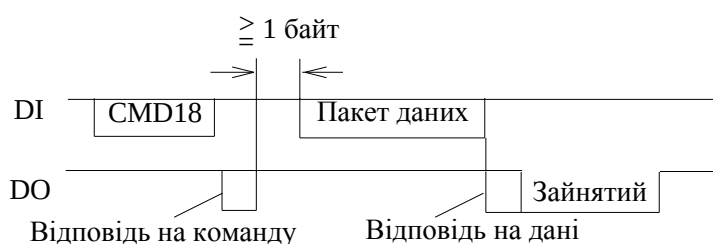


Рис. 4.11.

Після того як карта прийняла команду запису CMD24, ведучий контролер після деякого проміжку (один або більше байтів) передає в карту пакет даних. Формат пакета такий самий, як у команди блокового читання. Після передачі пакету, карта відразу ж видає відповідь на дані (Data Response), за яким передається прапор зайнятості. Більшість карт не можуть змінювати розмір блоку, що записується, він є фіксованим і становить 512 байт.

За правилами режиму SPI сигнал CS повинен перебувати в активному рівні протягом всієї транзакції, однак є виключення із цього правила. Коли карта зайнята, ведучий контролер може зняти сигнал CS, щоб звільнити шину SPI для якого-небудь іншого SPI-пристрою. Якщо ж знову вибрати карту в той час, коли вона зайнята виконанням внутрішнього процесу, карта знову встановить сигнал DO у низький рівень. Тому, щоб скоротити час очікування, краще виконувати перевірку на зайнятість безпосередньо перед видачею команди та пакета даних, а не очікувати звільнення карти після посилки команди. Крім того, внутрішній процес ініціюється через байт після відповіді даних, тобто необхідно видати 8 тактових імпульсів, щоб ініціювати внутрішню операцію запису. Стан сигналу CS під час цих восьми тактових імпульсів не враховується, тому можна сполучити цю ініціацію із процесом звільнення шини.

Багатоблоковий запис демонструє часова діаграма, представлена на рис. 4.12. За допомогою команди багатоблокового запису CMD25 можна записати послідовність із декількох блоків, починаючи із заданої адреси.

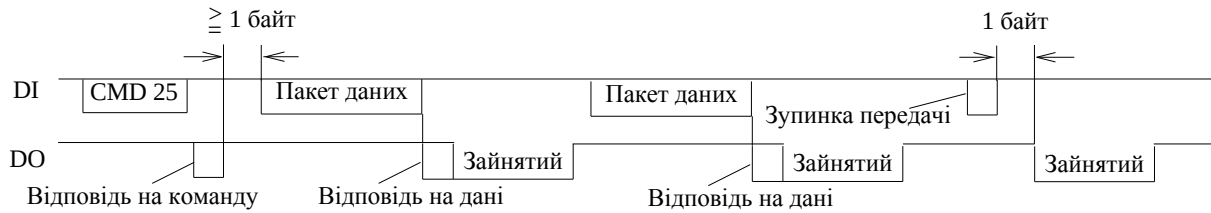


Рис. 4.12.

Якщо перед цією командою число переданих блоків не було задано за допомогою команди CMD23 (тільки для MMC) або ACMD23 (для SDC), транзакція буде ініційована як необмежений багато блоковий запис, тобто операція запису буде тривати, поки ведучий контролер не перерве її передачею маркера "Stop Tran". Що ж стосується SDC, то транзакція багатоблокового запису повинна перериватися маркером "Stop Tran" незалежно від того, чи є вона визначеною або необмеженою.

4.3.3. Приклад програми роботи з пам'яттю

На рис. 4.13 представлена принципова схема пристрою зчитування/запису карт пам'яті, створена в Proteus.

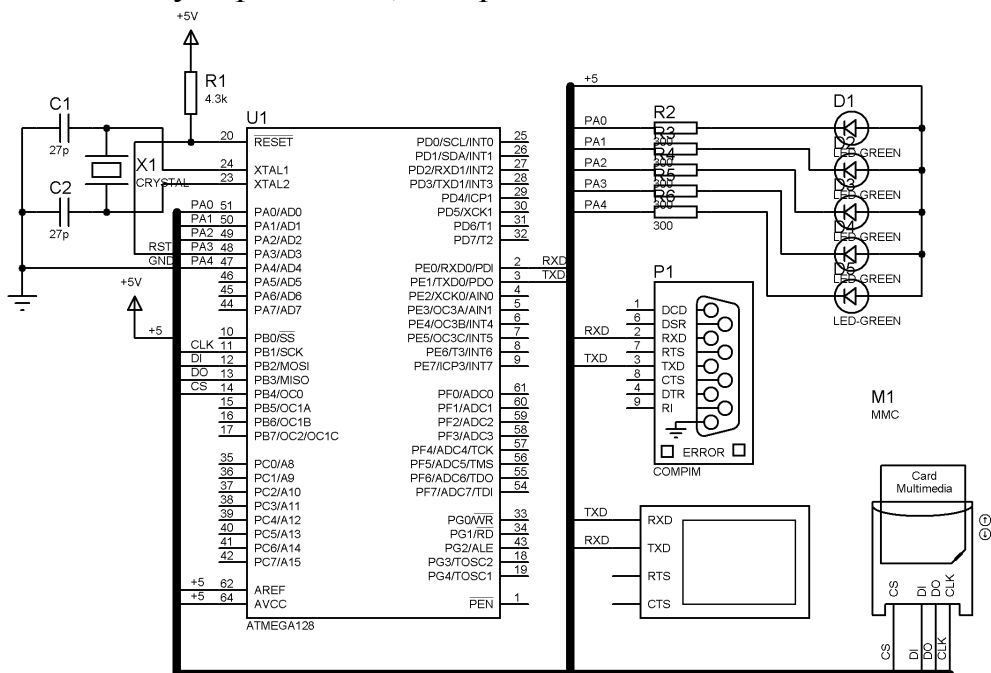


Рис. 4.13.

Приклад 4.1. Написати програму для мікроконтролера, що виконує запис 512 байт даних у заданий сектор карти пам'яті, а після цього зчитує ці дані й передає по UART. При кожному звертанні до карти пам'яті, по UART відправляється повідомлення, що інформує користувача про результат виконання дій. За допомогою світлодіодних індикаторів інформувати користувача про помилки звертання до карти пам'яті.

Текст програми для мікроконтролера АТМega128 представлений нижче.

```
#include <iom128v.h>           // підключення заголовочних файлів
#include <macros.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define SPIDDR DDRB           // присвоєння регістрам альтернативних імен
#define SPIPORT PORTB
#define SPIPIN PINB
#define CS 0x10
#define RST0() PORTA|=0x01
#define SET0() PORTA&=~0x01
#define RST1() PORTA|=0x02
#define SET1() PORTA&=~0x02
#define RST2() PORTA|=0x04
#define SET2() PORTA&=~0x04
#define RST3() PORTA|=0x08
#define SET3() PORTA&=~0x08
#define RST4() PORTA|=0x10
#define SET4() PORTA&=~0x10

#define DISABLE_MMC() PORTB|=CS
#define ENABLE_MMC() PORTB&=~CS

void delay(int ms);           // функція часової затримки
void init_spu(void);          // функція ініціалізації контролера
// функція передачі строки по UART
void uart_puts(char * s, int size);
// функція передачі байта даних по SPI
char sspi(char d);
// функція передачі команди в карту пам'яті
char MMC_Command(char bef, unsigned long addr, char bef);
char MMC_Init();              // функція ініціалізації карти пам'яті
// функція запису сектору
char MMC_WriteSector(unsigned long Sector, char * buff);
// функція зчитування сектору
char MMC_ReadSector(unsigned long Sector, char *buff);
char arr[512]; // масив з 512 елементів
// оголошення строкових констант для контролю операцій з картою
char err0[]="MMC Initialization error!";
char err1[]="Command 'writes a single block' (0x58) error!";
char err2[]="Invalid response! MMC write data error!";
char err3[]="Command 'read single block' (0x51) error!";
char msg0[]="Initialization...";
char msg1[]="Initialization OK!";
char msg2[]="Begin write data to MMC...";
char msg3[]="Write data OK!";
char msg4[]="Begin read data from MMC...";
char msg5[]="Read data OK!";

// функція часової затримки
```

```
void delay(int ms)
{
int c1, c2;                                // оголошення додаткових змінних
for(c1=0;c1<ms;c1++)                       // в циклі виконується порожня операція
    for(c2=0;c2<970;c2++) NOP();
}

// функція ініціалізації мікроконтролера
void init_cru(void)
{
    // налаштування UART
    UCSR0C = 0x06;
    UBRRL = 0x33;                          // швидкість передачі даних 9600 біт/с
    UCSR0B = 0x18;                          // дозвіл роботи приймача й передавача
    DDRB=0b00010111;                       // налаштування виводів порту В
    // налаштування SPI
    SPCR = (1 << SPE) | (1 << MSTR) | (1 << SPR1) | (1 << SPR0);
}

// функція передачі даних по UART
void uart_puts(char * s, int size)
{
    int i;                                  // додаткова змінна
    for(i=0;i<size-1;i++)
    {
        UDR0=s[i]; // передача даних
        // очікування завершення передачі даних
        while(!(UCSR0A&(1<<UDRE0)));
    }
    UDR0=13;                               // передача символу нового рядка
    while(!(UCSR0A&(1<<UDRE0)));
}

// функція передачі байта даних по SPI
char sspi(char d)
{
    char r=0;
    SPDR=d;                                // запис значення в регістр даних
    // очікування завершення процесу передачі
    while(!(SPSR&(1<<SPIF)));
    r=SPDR; // зчитування з регістра прийнятого байта
    return r;
}

// функція передачі команди в карту MMC
char MMC_Command(char bef, unsigned long addr, char bef)
{
    sspi(0xFF);                             // передача значення 0xFF
    sspi(bef);                               // передача коду команди
    sspi((char)(addr >> 24));               // передача адреси
    sspi((char)(addr >> 16));
    sspi((char)(addr >> 8));
}
```



```

sspi((char)addr);
sspi(bef);
sspi(0xFF);
// повертається останнє зчитане значення
return sspi(0xFF);
}

// функція ініціалізації карти MMC
char MMC_Init()
{
char i;
DISABLE_MMC();           // відключити MMC
// очікування 10*8=80 тактових імпульсів
for(i=0; i < 10; i++) sspi(0xFF);
ENABLE_MMC();           // активувати MMC

// відправити повідомлення по UART
uart_puts(msg0,sizeof(msg0));
if (MMC_Command(0x40,0,0x95)!=1)
{
SET0();
// відправити повідомлення по UART
uart_puts(err0,sizeof(err0));
return 1;                // виникла помилка при роботі з MMC
}
// якщо карта MMC відсутня то програма зациклюється
while (MMC_Command(0x41,0,0xFF)!=0);
// відправити повідомлення по UART
uart_puts(msg1,sizeof(msg1));
RST0();
return 0;
}
// функція запису сектора в пам'ять
char MMC_WriteSector(unsigned long Sector, char * buff)
{
int i;
ENABLE_MMC(); // активувати карту MMC
// передача команди CMD24
if (MMC_Command(0x58,(Sector<<9),0xFF) !=0)
{
SET1();
// відправити повідомлення по UART
uart_puts(err1,sizeof(err1));
DISABLE_MMC();           // деактивувати MMC
return 1; // вихід з функції
}
RST1();
// відправити повідомлення по UART
uart_puts(msg2,sizeof(msg2));
// формування пакетів даних у відповідності до протоколу
sspi(0xFF);
sspi(0xFF);

```

```
sspi(0xFE);
// передача 512 байт для запису
for (i=0; i<512; i++) sspi(buff[i]);
sspi(0xFF);
sspi(0xFF);
// перевірка коду стану
if((sspi(0xFF)& 0x1F)!=0x05)
{
    SET2();
    // відправити повідомлення по UART
    uart_puts(err2,sizeof(err2));
    DISABLE_MMC();           // деактивувати MMC
    return 2;                 // вихід з функції
}
RST2();
// очікування поки карта не стане вільною
while(sspi(0xFF) != 0xFF);
// відправити повідомлення по UART
uart_puts(msg3,sizeof(msg3));
DISABLE_MMC();              // деактивувати MMC
return 0;                    // вихід з функції
}
// функція зчитування сектора (512 байт) з карти MMC
char MMC_ReadSector(unsigned long Sector, char *buff)
{
    int i;                    // додаткова змінна
    ENABLE_MMC();             // активувати карту пам'яті
    // передача команди CMD17
    if(MMC_Command(0x51,(Sector<<9),0xFF)!=0)
    {
        SET3();
        // відправити повідомлення по UART
        uart_puts(err3,sizeof(err3));
        DISABLE_MMC();        // деактивувати карту MMC
        return 1;              // MMC: помилка зчитування
    }
    RST3();

    uart_puts(msg4,sizeof(msg4));
    // очікування значення 0xFE - кінець передачі
    while(sspi(0xFF)!=0xFE);
    // зчитування даних з карти пам'яті
    for(i=0; i<512; i++) buff[i]=sspi(0xFF);
    sspi(0xFF);
    sspi(0xFF);
    // відправити повідомлення по UART
    uart_puts(msg5,sizeof(msg5));
    DISABLE_MMC();            // деактивувати MMC
    return 0;                  // вихід з функції
}
// головна функція програми
void main()
```

```

{
int i;                                // оголошення додаткової змінної
// формування даних для запису
for(i=0;i<512;i++) arr[i]=(char)(i>>1);
init_cpu();                           // виклик функції ініціалізації
MMC_Init();                           // ініціалізація карти пам'яті
MMC_WriteSector(0,arr);               // запис даних у сектор 0
MMC_ReadSector(0,arr);               // зчитування даних з сектору 0
// передача даних по UART
for(i=0;i<512;i++)
{
    UDR0=arr[i]; //
    while(!(UCSR0A&(1<<UDRE0)));
}
while(1);                            // зациклення програми
}

```

Після компіляції програми й моделювання, за допомогою віртуального термінала (рис. 4.14) можна бачити результат всіх операцій звертання до пам'яті.

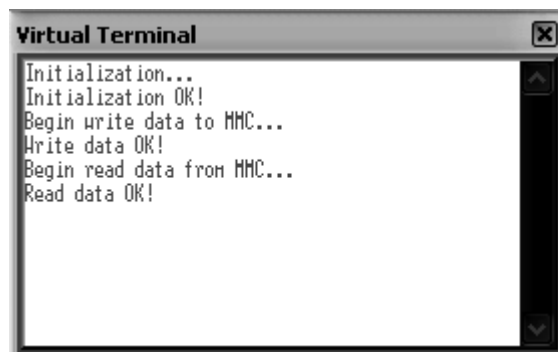


Рис. 4.14.

4.4. Організація обміну даними між мікроконтролерами

На сьогоднішній день при побудові автоматизованих систем управління виробничими процесами все частіше постають завдання, які не можуть бути ефективно вирішені системами з одним мікроконтролером або мікропроцесором, що зумовлює зацікавленість замовників у створенні багатопроцесорних систем з розподілом виконуваних функцій. Найпривабливішими для виробництва залишаються мікроконтролерні мережі типу “ведучий – ведений”. Більшість таких мереж мають відкритий протокол як на фізичному, так і на каналному рівні, відрізняються простотою його реалізації, і, що саме головне, дані протоколи легко адаптуються до різних застосувань за рахунок наявності в подібних системах властивостей нарощуваності та адаптованості.

Інтерфейс SPI часто використовується для обміну даними між мікроконтролерами. Структурна схема типового застосування представлена на рис. 4.15.

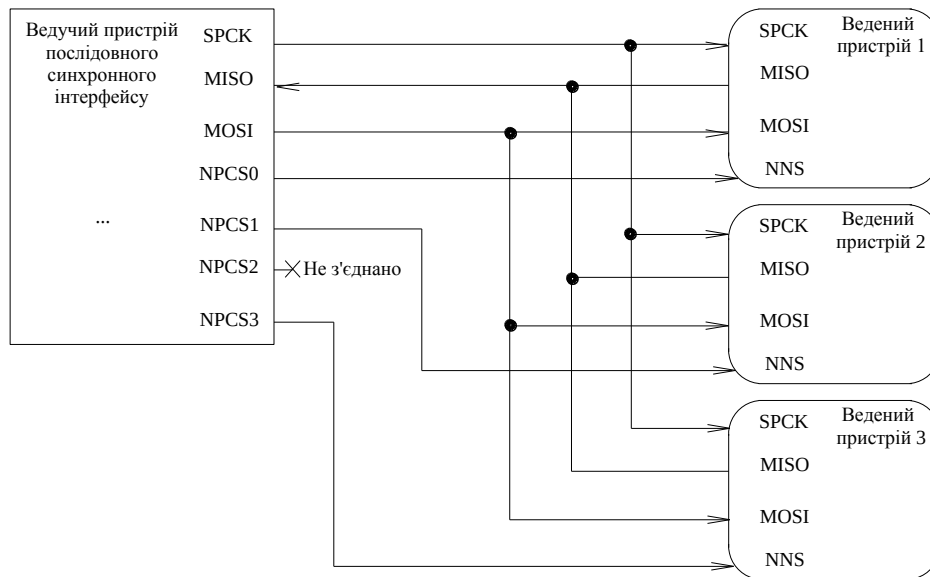


Рис. 4.15.

На рис. 4.16 представлена схема сполучення 2-х мікроконтролерів AT90S8535 по інтерфейсу SPI.

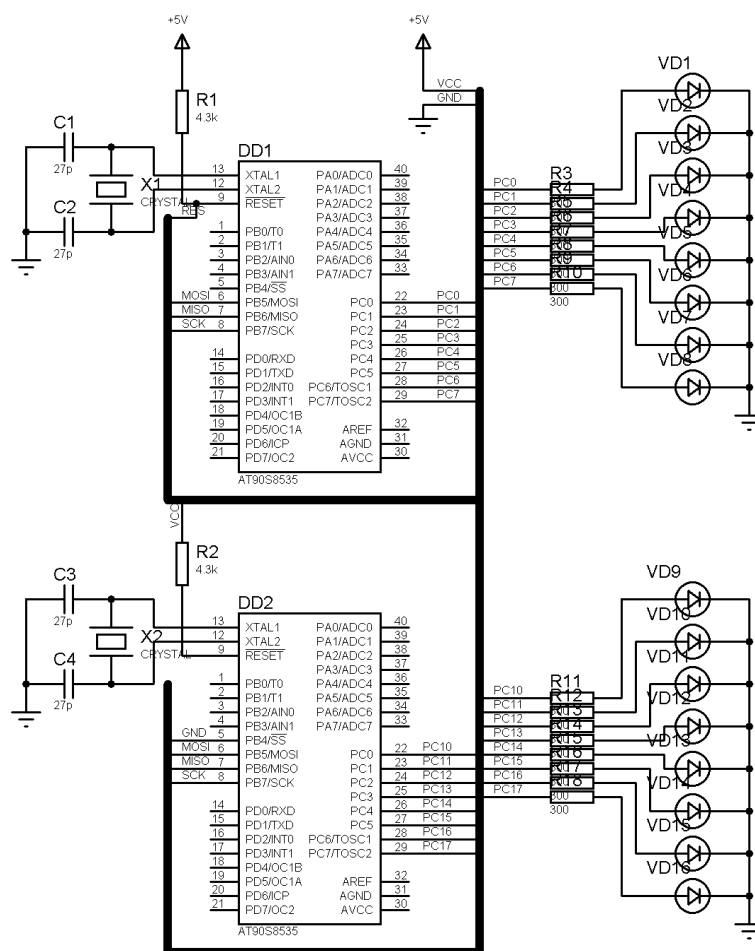


Рис. 4.16.

Приклад 4.2. Написати програми для ведучого і веденого мікроконтролерів AT90S8535, які реалізують наступний алгоритм: ведучий мікроконтролер (DD1) виконує інкремент змінної-лічильника, і кожні 200 мс передає значення цієї змінної по інтерфейсу SPI. Прийняте значення виводиться в порт C. Ведений мікроконтролер виводить у порт C прийняте по SPI значення. Прийняте число інвертується і передається назад ведучому мікроконтролеру.

Запускаємо середовище розробки ImageCraft і створюємо новий проект із назвою Master (див. розділ 2). У налаштуваннях проекту вибираємо контролер AT90S8535. Текст програми для мікроконтролера DD1, що працює в режимі ведучого, представлений нижче.

```
#include <io8535v.h>      // підключення заголовних файлів
#include <macros.h>
// функція ініціалізації мікроконтролера
void init_cpu()
{
    DDRA=0x00;           // порт A налаштований на вхід
    DDRB=0b10110000;     // лінії SS, MOSI, SCK налаштовані на вихід
    DDRC=0xFF;           // порт C налаштований на вихід
    DDRD=0x00;           // порт D налаштований на вхід
    SPCR=0x53;           // налаштування SPI: SPI включений у режимі
                        // ведучого, дільник тактової частоти 128
}

void delay_ms(int ms)    // функція затримки
{
    int c1, c2;           // додаткові змінні
    for(c1=0;c1<ms;c1++) // контролер виконує «порожню» операцію
    for(c2=0;c2<970;c2++) NOP();
}

void main()              // головна функція
{
    char counter=0;       // оголошення змінної-лічильника
    char rdata;           // додаткова змінна
    init_cpu();           // виклик функції ініціалізації контролера
    while(1)              // вічний цикл
    {
        SPDR=counter;     // передача значення змінної counter
        while(!(SPSR&0x80)); // очікування завершення передачі
        rdata=SPDR;       // зчитування з регістра даних SPI
        PORTC=rdata;      // вивід зчитаного значення в порт C
        counter++;        // інкремент значення змінної
        delay_ms(200);    // виклик функції затримки
    }
}
```

Після компіляції проекту буде створений файл прошивки мікроконтролера. Створюємо новий проект із назвою Slave. Текст

програми для мікроконтролера DD2, що працює в режимі веденого, представлений нижче.

```
#include <io8535v.h>      // підключення заголовних файлів
#include <macros.h>
// підпрограма обробки переривання
#pragma interrupt_handler spi_stc_isr:11
void spi_stc_isr(void)
{
    char data;              // оголошення додаткової змінної
    data=SPDR;              // зчитування прийнятого значення з регістра
                           // даних SPI
    PORTC=data;             // вивід даних у порт C
    SPDR=~data;             // запис у регістр даних інвертованого
                           // значення
}
// функція ініціалізації мікроконтролера
void init_cpu()
{
    DDRA=0;                 // порт A налаштований на вхід
    DDRB=0b01000000;        // вивід MISO налаштований на вихід
    DDRC=0xFF;              // порт C налаштований на вихід
    DDRD=0;                 // порт D налаштований на вхід
    SPCR=0xC3;              // налаштування SPI: SPI включений в режимі
                           // веденого, дільник тактової частоти 128
                           // дозволене переривання по прийому
    SEI();                  // дозвіл обробки переривань
}
void main()                // головна функція
{
    char c;                 // оголошення додаткової змінної
    init_cpu();             // виклик функції ініціалізації контролера
    while(1);               // вічний цикл
}
```

Питання до розділу 4

1. В яких режимах може працювати інтерфейс SPI?
2. Для інтерфейсу (SPI) є чотири комбінації фази і полярності сигналу SCK по відношенню до послідовної передачі даних. Поясніть, для чого це потрібно?
3. Яким чином визначається, ведучий чи ведений є пристрій, що підключений до шини SPI?
4. На якій швидкості можлива передача даних по інтерфейсу SPI? Яким чином виконується налаштування швидкості передачі?
5. Чи можливо реалізувати мережу обміну інформацією між декількома мікроконтролерами по інтерфейсу SPI? Поясніть чому.
6. Які регістри використовуються для налаштування SPI?
7. У чому полягає особливість запису і зчитування даних на карту пам'яті SD/MMC?

8. Яким чином можна визначити наявність помилки при зверненні до карти пам'яті (запис/зчитування даних)?
9. Яку послідовність команд необхідно сформулювати для запису/зчитування даних в/із карти пам'яті?
10. Які помилки можливі при роботі з картою пам'яті?
11. Опишіть принцип організації файлової системи на карті пам'яті SD/MMC.

Задачі до розділу 4

1. Розробити принципову схему сполучення двох мікроконтролерів по інтерфейсу SPI. На вхід АЦП першого мікроконтролера, що є головним, подається напруга від 0 до 5В. До другого мікроконтролера підключені чотири семисегментних індикатори. Написати програми для обох мікроконтролерів, що виконують наступну задачу: перший мікроконтролер зчитує дані з АЦП та передає їх другому мікроконтролеру, який відображає отримане значення на семисегментних індикаторах.
2. Розробити програму, що зчитує значення з АЦП із частотою дискретизації 100 Гц, і записує дані в карту пам'яті SD/MMC. При отриманні запиту від комп'ютера дані зчитуються з карти пам'яті і передаються в комп'ютер.
3. Розробити плеєр. Програма записує на карту пам'яті дані з WAV-файлу, а потім виконується програвання цього файлу (дані зчитуються з карти пам'яті і виводяться на ЦАП).
4. Записати в карту пам'яті дані про зображення (кілька кадрів) для графічного РКІ. Потім послідовно зчитувати ці дані і виводити на екран, створюючи анімацію.
5. Розробити мережу, що складається з 3-х мікроконтролерів. Обмін даними між мікроконтролерами відбувається по інтерфейсу SPI. До порту першого мікроконтролера, який в мережі є головний, підключені тактові кнопки. На двох інших мікроконтролерах, які в мережі є веденими, відображати стани кнопок – на першому МК молодшу тетраду, на другому МК – старшу тетраду.

5. ШИНА I2C

5.1. Послідовний порт TWI

Шина I2C широко використовується в побутовій електроніці, передачі даних і промисловій електроніці. Розроблена фірмою Philips проста двонапрявлена 2-провідна шина спочатку для ефективного керування та взаємодії різних блоків телевізорів, стала застосовуватися для зв'язку між собою однокристальних мікроконтролерів, РКІ-індикаторів, мікросхем пам'яті, аналого-цифрових і цифро-аналогових перетворювачів, мікросхем реального часу та ін.

Для здійснення процесу обміну інформацією по шині I2C використовуються всього два сигнали – лінія даних SDA та лінія синхронізації SCL. Для забезпечення реалізації двонапрявленості шини без застосування складних арбітрів шини вихідні каскади пристроїв, підключених до шини, мають відкритий стік або відкритий колектор для забезпечення функції монтажного "І". Як показано на рис. 5.1, обидві лінії шини підключені до живлення через резистори.

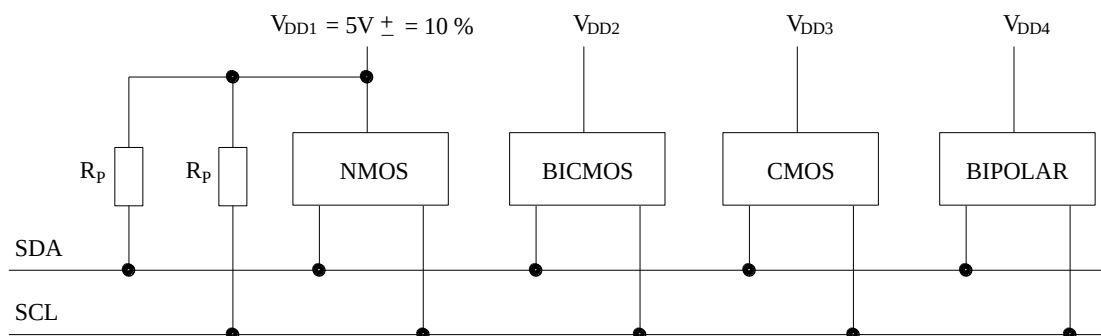


Рис. 5.1.

Проста двохпровідна послідовна шина I2C мінімізує кількість з'єднань між інтегральними схемами. Як результат – друковані плати стають більш простими та технологічними при виготовленні. Інтегрований I2C-протокол усуває необхідність у дешифраторах адреси та іншій зовнішній логіці узгодження. Максимальна припустима кількість мікросхем, приєднаних до однієї шини, обмежується максимальною ємністю шини 400 пФ.

Шина I2C має наступні переваги:

- Тільки дві лінії – послідовна лінія даних (SDA) і послідовна лінія синхронізації (SCL).

- Кожен елемент, з'єднаний із шиною, є програмно адресованим своєю унікальною адресою. При цьому взаємодія між ними може бути побудовано по простому принципу master/slave.

– Послідовна 8-розрядна передача даних може проводитися зі швидкістю від 0 до 100 кБіт/с у стандартному режимі та 400 кБіт/с у швидкому режимі. При розробці власних пристроїв можливе застосування і більш високих швидкостей.

– Низьке енергоспоживання, висока завадостійкість, широкий діапазон напруг живлення.

Всі абоненти шини діляться на два класи – «Master» та «Slave». Пристрій «Master» генерує тактовий сигнал (SCL), і тому є ведучим. Він може самостійно виходити на шину й адресуватися до будь-якого «Slave» – пристрою з метою передачі або прийому інформації. Всі «Slave» – пристрої «слухають» шину на предмет виявлення власної адреси, і, розпізнавши її, виконують вказані дії. Крім того, можливий режим «Multi-master» – режим, коли на шині встановлено кілька ведучих («Master») пристроїв, які або спільно розділяють загальні «Slave» – пристрої, або по чергово є ведучими пристроями, коли самі ініціюють обмін інформацією. Режим «Multi-master» вимагає арбітражу та розпізнавання конфліктів. Він складніше реалізується і тому рідше використовується в пристроях.

Двохпроводний послідовний інтерфейс TWI ідеально підходить для типових застосувань мікроконтролера.

Модуль TWI складається з декількох підмодулів (рис. 5.2.). Всі регістри доступні через шину даних мікроконтролера.

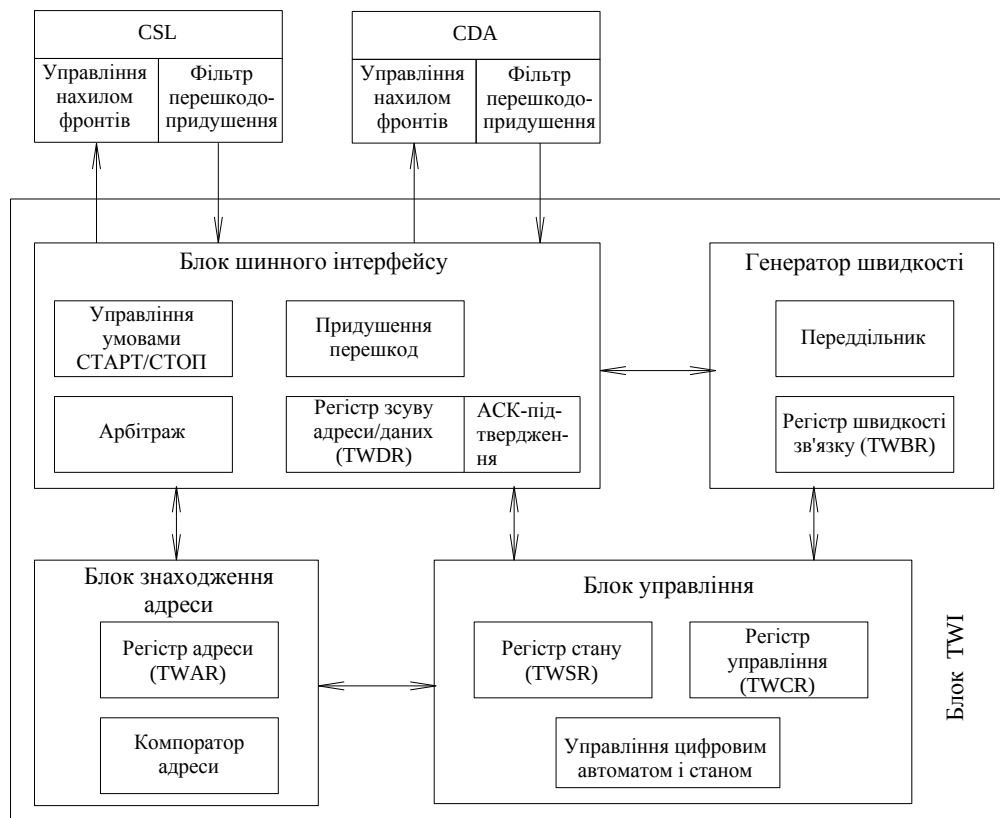


Рис. 5.2.

Виводи SCL та SDA зв'язують двохпровідний інтерфейс мікроконтролера з іншими мікроконтролерами в системі. Драйвери виходів містять обмежувач швидкості зміни фронтів для виконання вимог до TWI. Вхідні каскади містять блок захисту від завад, завдання якого складається в ігноруванні імпульсів тривалістю менше 50 нс. До кожної із цих ліній можна підключити внутрішній резистор.

Блок генератора швидкості зв'язку керує періодом імпульсів SCL у режимі ведучого пристрою. Період SCL задається регістром швидкості TWI (TWBR) і значенням біт керування дільником частоти у регістрі стану TWI (TWSR). У підлеглому режимі значення швидкості або установки дільника не впливають на роботу, але частота синхронізації підлеглого пристрою повинна бути мінімум в 16 разів вищою за частоту SCL. Частота SCL генерується у відповідності з наступним виразом:

$$f_{SCL} = \frac{f_{цпв}}{16 + 2 \cdot (TWBR) \cdot 4^{TWS}},$$

де WBR – значення регістра швидкості TWI; TWPS – значення біт дільника в регістрі стану TWI.

TWBR повинен бути не меншим 10, якщо TWI працює у ведучому режимі. Якщо TWBR менше 10, то ведучий може генерувати некоректний стан на лініях SDA та SCL.

Блок шинного інтерфейсу містить регістр зсуву адреси та даних (TWDR), контролер умов СТАРТ/СТОП і схему арбітражу. TWDR містить переданий байт адреси чи даних, або прийнятий байт адреси чи даних. Крім 8-розрядного регістра TWDR до складу блоку шинного інтерфейсу також входить регістр, що зберігає значення переданого або прийнятого біта. До даного регістру немає прямого доступу з боку програмного забезпечення. Однак під час прийому він може встановлюватися або скидатися шляхом маніпуляцій з регістром керування TWI (TWCR). У режимі передавача значення прийнятого біта підтвердження можна визначити за значенням регістра TWSR.

Контролер умов СТАРТ/СТОП відповідає за генерацію та виявлення умов СТАРТ, ПОВТОРНИЙ СТАРТ і СТОП.

Якщо TWI ініціював передачу як ведучий, то схема арбітражу безупинно контролює передачу, визначаючи можливість подальшої передачі. Якщо TWI губить арбітраж, то блок формує відповідний сигнал до блока керування, який виконує адекватні дії та генерує відповідний код стану.

Блок виявлення адреси перевіряє, чи дорівнює прийнята адреса значенню 7-розрядної адреси з регістра TWAR. Якщо встановлено біт дозволу виявлення загального виклику TWGCE у регістрі TWAR, то всі вхідні адресні біти будуть додатково порівнюватися з адресою загального виклику. При рівності адрес подається сигнал блоку керування, що дозволяє виконати йому необхідні дії.

Блок керування спостерігає за шиною TWI і генерує відгуки відповідно до установок регістра керування TWI (TWCR). Якщо на шині TWI виникає подія, яка вимагає уваги з боку програми, то встановлюється прапор переривання TWINT. Наступним тактом оновлюється вміст регістра статусу TWI – TWSR, у якому буде записаний код, що ідентифікує подію. Ця інформація зберігається в TWSR тільки тоді, коли встановлений прапорець переривання TWI. В інший час у регістрі TWSR утримується спеціальний код стану, що інформує про відсутність інформації про стан TWI. Поки встановлений прапор TWINT лінія SCL залишається в низькому стані. Цим забезпечується можливість завершити програмі всі завдання перед продовженням сеансу зв'язку. Прапор TWINT встановлюється в наступних ситуаціях:

- після передачі умови СТАРТ/ПОВТОРНИЙ_СТАРТ;
- після передачі ПІДЛЕГЛИЙ_АДР+ЧИТАННЯ/ЗАПИС;
- після передачі адресного байта;
- після втрати арбітражу;
- після того, як TWI адресований власною адресою або загальним викликом;
- після прийому байта даних;
- після прийому умови СТОП або ПОВТОРНИЙ_СТАРТ у режимі підлеглої адресації;
- після виникнення помилки через некоректну умову СТАРТ або СТОП.

5.2. Діаграми передачі даних

В початковий момент часу (у режимі очікування) лінії SDA і SCL перебувають у стані логічної одиниці. У режимі передачі (рис. 5.3) біт даних SDA синхронізується позитивним імпульсом SCL. Зміна інформації на лінії SDA виконується при нульовому стані лінії SCL. «Slave» – пристрій може утримувати лінію SCL у нульовому стані на час обробки чергового прийнятого байта, при цьому «Master» – пристрій повинен дочекатися звільнення лінії SCL, перш ніж продовжити передачу інформації.

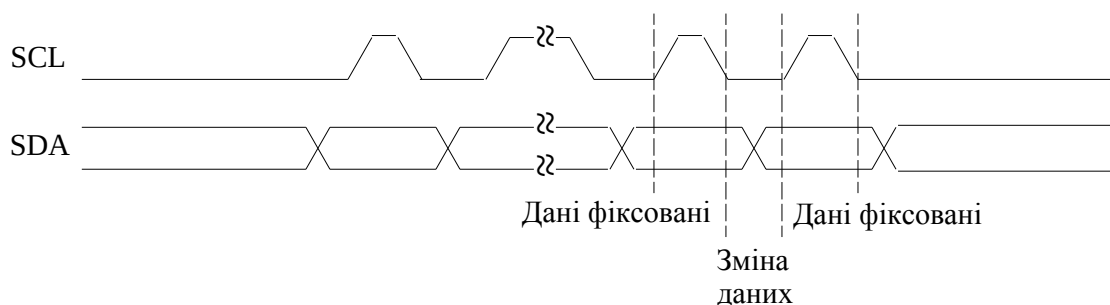


Рис. 5.3.

Для синхронізації пакетів шини I2C розрізняють дві умови – «Start» та «Stop», які обмежують початок і кінець інформаційного пакета (рис.

5.4). Для кодування цих умов використовується зміна стану лінії SDA при одиничному стані лінії SCL, що неприпустимо при передачі даних. «Start» – умова утворюється при зрізі сигналу SDA, коли лінія SCL перебуває в одиничному стані. «Stop» – умова утвориться фронтом сигналу на лінії SDA при одиничному стані лінії SCL.

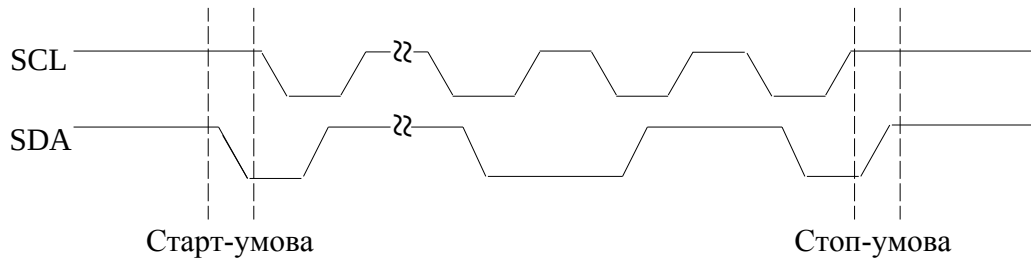


Рис. 5.4.

Стани «Старт» та «Стоп» завжди виробляються ведучим. Вважається, що шина зайнята після появи стану «Старт». Шина вважається звільненою через деякий час після появи умови «Стоп».

Передача даних починається при першому позитивному імпульсі на лінії SCL, яким синхронізується старший біт першого інформаційного байта. Кожен інформаційний байт містить 9 тактових періодів лінії SCL. У дев'ятому такті пристрій-одержувач видає підтвердження (#ACK) – негативний імпульс, що свідчить про прийом даних (рис. 5.5). Відсутність підтвердження інтерпретується як помилка.

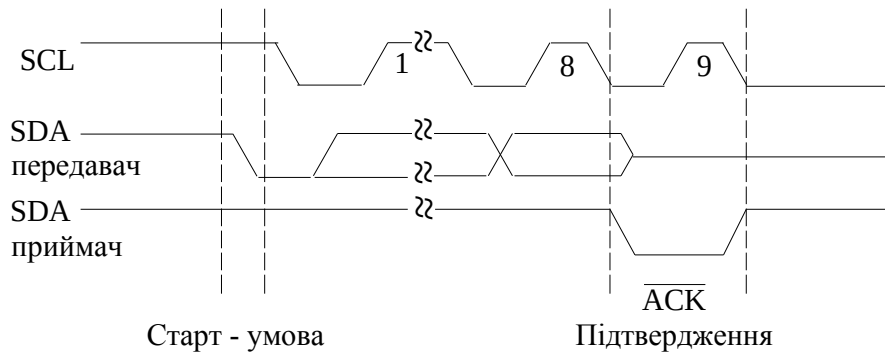


Рис. 5.5.

При прийомі останнього байта в серії, замість сигналу ACK майстер може виставити стан «Стоп», при цьому підлеглий передавач повинен звільнити лінію.

5.3. Адресація пристроїв

Кожен пристрій, що підключений до шини, може бути програмно адресований по унікальній адресі. Для вибору приймача повідомлення ведучий використовує унікальний адресний компонент у форматі посилки. При використанні однотипних пристроїв, інтегральні схеми часто мають додатковий селектор адреси, який може бути реалізований як у вигляді додаткових цифрових входів селектора адреси, так і у вигляді аналогового

входу. При цьому адреси таких однотипних пристроїв виявляються рознесені в адресному просторі пристроїв, підключених до шини. У звичайному режимі використовується 7-бітна адресація. Всі інтегральні мікросхеми, що підтримують роботу в стандарті шини I2C, мають набір фіксованих адрес, перелік яких зазначений виробником в описах контролерів.

Щоб почати операцію обміну пристрій «Master» видає на шину «Start» – умову, за яким передається байт із адресою веденого пристрою, що складається з 7-бітної адреси пристрою та одного біта – прапора операції «R/W», який визначає напрямок обміну даними, причому 0 означає передачу від ведучого до веденого, а 1 – зчитування читання з веденого пристрою (рис. 5.6). Всі біти по шині I2C передаються в порядку старший – молодший, тобто першим передається 7-й біт, останнім 0-й біт. Після того, як адреса передана, кожен пристрій у системі порівнює перші сім бітів після сигналу «Старт» зі своєю адресою. При збігу пристрій вважається вибраним як приймач або передавач, залежно від біта напрямку. Адреса веденого може складатися з фіксованої та програмованої частини.

За адресою можуть передаватися один або декілька інформаційних байтів (у напрямку, встановленому прапором «R/W»), біти яких синхронізуються сигналом SCL з «Master» – пристроєм.

При виконанні операції зчитування «Master» – абонент повинен супроводжувати прочитаний байт сигналом #ACK якщо необхідно прочитати наступний байт і не видавати сигнал #ACK якщо потрібно закінчити зчитування пакета (рис. 5.7, рис. 5.8). Логічна реалізація протоколів на шині I2C може бути довільною для кожної конкретної інтегральної схеми.

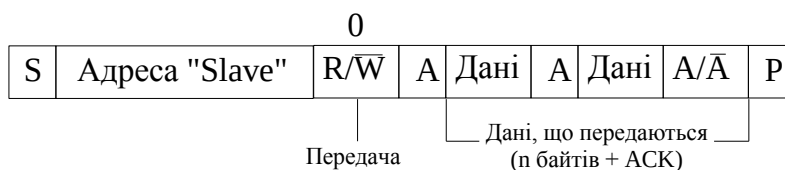


Рис. 5.6.

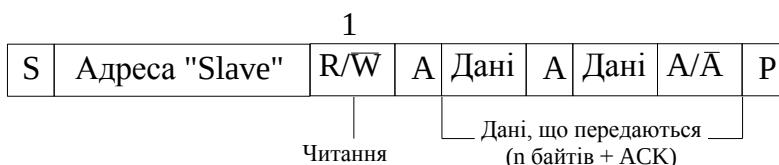


Рис. 5.7.

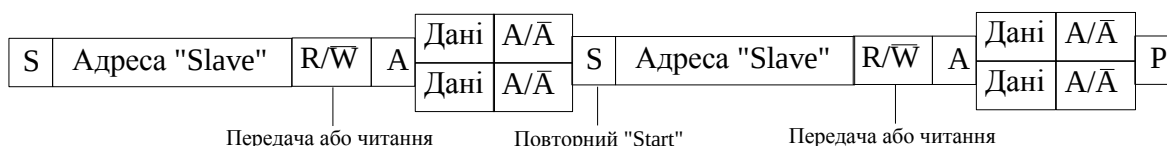


Рис. 5.8.

На представлених діаграмах «S» – умова «Start», «A» – біт підтвердження прийому #ACK, «P» – умова «Stop».

5.4. Опис регістрів TWI

Регістр швидкості зв'язку шини TWI – TWBR:

Розряд	7	6	5	4	3	2	1	0
	TWBR	TWBR	TWBR	TWBR	TWBR	TWBR	TWBR	TWBR
	7	6	5	4	3	2	1	0
Зчитування /запис	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.
Початкове значення	0	0	0	0	0	0	0	0

Розряди 7..0 – біти регістра швидкості зв'язку шини TWI.

TWBR задає коефіцієнт ділення частоти генератора швидкості зв'язку. Генератор частоти швидкості зв'язку – дільник частоти, який формує сигнал синхронізації SCL у режимі "Ведучий".

Регістр керування шиною TWI – TWCR:

Розряд	7	6	5	4	3	2	1	0
	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Зчитування /запис	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч.	Зч. /Зап.	Зч.	Зч. /Зап.
Початкове значення	0	0	0	0	0	0	0	0

Регістр TWCR призначений для керування роботою TWI. Він використовується для дозволу роботи TWI, для ініціації сеансу зв'язку шляхом генерації умови СТАРТ на шині, для генерації підтвердження прийому, для генерації умови СТОП. Він також сигналізує про спробу помилкового запису в регістр TWDR, коли доступ до нього був заборонений.

Розряд 7 – TWINT: Прапор переривання TWI. Даний біт встановлюється апаратно, якщо TWI завершує поточне завдання й очікує реакції програми. Якщо біт I в SREG і біт TWIE в TWCR встановлені, то мікроконтролер переходить на вектор переривання TWI. Лінія SCL залишається в низькому стані, поки встановлено прапор TWINT. Прапор TWINT скидається програмно шляхом запису в нього логічної 1. Також потрібно врахувати, що очищення даного прапора приводить до поновлення роботи TWI. Із цього випливає, що програмне скидання даного прапора необхідно виконати після завершення опитування регістрів TWAR, TWSR та TWDR.

Розряд 6 – TWEA: Біт дозволу підтвердження. Біт TWEA керує генерацією імпульсу підтвердження. Якщо в біт TWEA записана лог. 1, то імпульс підтвердження генерується на шині TWI, якщо виконується одна з наступних умов:

- прийнята власна адреса;
- прийнятий загальний виклик, коли встановлений біт TWGCE у регістрі TWAR;
- прийнятий байт даних у режимі ведучого приймача або підлеглого приймача.

Запис лог. 0 у біт TWEA дозволяє тимчасово відключитися від двохпровідної послідовної шини. Для поновлення розпізнавання адреси необхідно записати в даний біт лог. 1.

Розряд 5 – TWSTA: Біт умови СТАРТ. Програміст повинен установити даний біт при необхідності стати ведучим на двохпровідній послідовній шині. TWI апаратно перевіряє доступність шини та генерує умову СТАРТ, якщо шина вільна. Однак якщо шина зайнята, то TWI очікує появи умови СТОП, а потім генерує нову умову СТАРТ для перехоплення стану ведучого шини. TWSTA необхідно скидати програмно після передачі умови СТАРТ.

Розряд 4 – TWSTO: Біт умови СТОП. Установка біта TWSTO у режимі ведучого призводить до генерації умови СТОП на двохпровідній послідовній шині. Якщо на шині виконується умова СТОП, то біт TWSTO скидається автоматично. У підлеглому режимі установка біта TWSTO може використовуватися для виходу з умови помилки. У цьому випадку умова СТОП не генерується, але інтерфейс TWI повертається до режиму підлеглого і переводить лінії SCL та SDA у високоімпедансний стан.

Розряд 3 – TWWC: Прапор помилкового запису. Біт TWWC встановлюється при спробі запису в регістр даних TWDR, коли TWINT має низький рівень. Прапор скидається при запису даних в регістр TWDR, коли TWINT дорівнює 1.

Розряд 2 – TWEN: Біт дозволу роботи TWI. Біт TWEN дозволяє роботу TWI та активізує інтерфейс TWI. Якщо біт TWEN встановлений, то TWI бере на себе функції керування лініями вводу-виводу SCL і SDA. При цьому дозволяється робота обмежувачів швидкості зміни фронтів і фільтрів від перешкод. Якщо даний біт дорівнює нулю, то TWI відключаються і всі передачі припиняються незалежно від стану роботи.

Розряд 1 – Резервний біт. Даний біт є резервним і зчитується як 0.

Розряд 0 – TWIE: Дозвіл переривання TWI. Якщо в даний біт записана лог. 1 і встановлений біт I у регістрі SREG, то запит на переривання TWI буде генеруватися доти, поки встановлено прапор TWINT.

Регістр стану TWI – TWSR:

Розряд	7	6	5	4	3	2	1	0
	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0
Зчитування /запис	Зч.	Зч.	Зч.	Зч.	Зч.	Зч.	Зч. /Зап.	Зч. /Зап.
Початкове значення	1	1	1	1	1	0	0	0

Розряди 7..3 – TWS: Стан TWI. Дані 5 біт відображають стан логіки блоку TWI та двохпровідної послідовної шини. Зчитане значення з регістра TWSR містить 5-розрядний код стану та 2-розрядне значення, яке керує дільником.

Розряд 2 – Резервний біт. Даний біт є резервним і зчитується як 0.

Розряди 1..0 – TWPS: Біти дільника TWI. Дані біти відрізняються повним доступом (зчитування/запис) і дозволяють керувати дільником швидкості зв'язку.

Регістр дані шини TWI – TWDR:

Розряд	7	6	5	4	3	2	1	0
	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0
Зчитування /запис	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.
Початкове значення	1	1	1	1	1	1	1	1

У режимі передавача регістр TWDR містить наступний байт для передачі. У режимі приймача регістр TWDR містить останній прийнятий байт. Запис у регістр можлива тільки коли TWI не виконує процес зсуву даних. Такий стан настає, коли відбувається апаратна установка прапора переривання TWINT. Дані в регістрі TWDR залишаються стабільними поки встановлений біт TWINT. Під час зсуву послідовної передачі даних одночасно відбувається зсув для послідовного введення. TWDR завжди містить останній байт, що передавався по шині, крім ситуації відновлення нормальної роботи мікроконтролера по перериванню TWI. У цьому випадку стан TWDR є невизначеним. У випадку втрати арбітражу шини дані, передані від ведучого до підлеглого, не губляться. Керування бітом підтвердження відбувається автоматично під управлінням схеми TWI.

Розряди 7..0 – TWD: Регістр даних шини TWI. Дані 8 біт представляють байт даних, якому необхідно передати наступний, або останній прийнятий байт по двохпровідній послідовній шині.

Регістр адреси підлеглого пристрою шини TWI – TWAR:

Розряд	7	6	5	4	3	2	1	0
	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
Зчитування /запис	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.
Початкове значення	1	1	1	1	1	1	1	1

Якщо TWI налаштований на режим підлеглого передавача або приймача, то буде реагувати тільки на адресу, записану в цей регістр (в 7 старших розрядів TWAR). В інших режимах ведучого даний регістр не використовується. У багатомастерних системах регістр TWAR встановлюється в тому ведучому, до якого адресуються як до підлеглого.

Молодший розряд регістра TWAR використовується для дозволу виявлення адреси загального виклику (\$00). Спеціальний компаратор виконує порівняння адреси підлеглого пристрою (або адреси загального виклику) із прийнятою адресою. Якщо виявляється збіг, то генерується запит на переривання.

Розряди 7..1 – TWA: Регістр адреси підлеглого пристрою шини TWI. Дані сім біт складають адресу підлеглого TWI.

Розряд 0 – TWGCE: Біт дозволу виявлення загального виклику по шині TWI. Після установки даного біта дозволяється робота схеми виявлення загального виклику, переданого по двохпровідній послідовній шині.

5.5. Рекомендації з використання TWI

TWI орієнтований на передачу даних у форматі байт з керуванням по перериваннях. Переривання виникають після виявлення однієї з подій на шині, наприклад, прийом байта або передача умови СТАРТ. Керування TWI по перериваннях дозволяє звільнити програмне забезпечення на виконання інших завдань під час передачі байта даних. Установка прапора TWINT приводить до генерації запиту на переривання тільки в тому випадку, коли встановлений біт дозволу переривання TWIE у регістрі TWCR, а також дозволена робота переривань установкою біта в регістрі SREG. Якщо біт TWIE скинутий, то стан TWINT повинен відслідковуватися програмно для оцінки ситуації на шині TWI.

Після установки прапора TWINT інтерфейс TWI припиняє роботу і очікує реакції програми. У цьому випадку регістр статусу TWI (TWSR) містить значення, що показує поточний стан шини TWI. Виходячи із цього програма задає подальше поведіння шини TWI, маніпулюючи регістрами TWCR та TWDR.

На рис. 5.9. показаний приклад підключення до шини TWI. Передбачається, що ведучий бажає передати один байт даних підлеглому.

1. Першим кроком роботи TWI є передача умови СТАРТ. Це ініціюється шляхом запису специфічного значення в TWCR. Однак, необхідно стежити, щоб у записуваному в регістр значенні був встановлений біт TWINT. Запис лог. 1 в TWINT скидає цей прапор. TWI не почне роботу доти, поки буде встановлений прапор TWINT у регістрі TWCR. Відразу після скидання TWINT починається передача умови СТАРТ.

2. Після передачі умови СТАРТ встановлюється прапор TWINT у регістрі TWCR, а вміст TWSR оновлюється значенням коду стану, що говорить про успішну передачу умови СТАРТ.

3. У програмі необхідно виконати перевірку значення TWSR, щоб переконатися в тому, що умова СТАРТ була успішно передана. Якщо TWSR показує іншу ситуацію, то програма виконує особливі дії, наприклад, викликає процедуру обробки помилкових ситуацій. Якщо код стану має очікуване значення, то виконується завантаження умови ПІДЛЕГЛА_АДР + ЗАПИС в TWDR. Необхідно пам'ятати, що TWDR використовується для зберігання як адреси, так і даних. Після завантаження в TWDR бажаного значення ПІДЛЕГЛА_АДР + ЗАПИС у регістр TWCR повинне бути записане специфічне значення, що служить командою для передачі значення ПІДЛЕГЛА_АДР + ЗАПИС, що зберігається в TWDR. Яке саме значення необхідно записати буде сказано пізніше. Однак необхідно стежити, щоб у записуваному в регістр значенні був установлений біт TWINT. Запис лог. 1 в TWINT приводить до скидання цього прапора. TWI не почне роботу доти, поки встановлений біт TWINT у регістрі TWCR. Відразу після скидання прапора TWINT ініціюється передача адресного пакета.

4. Після передачі адресного пакета встановлюється прапор TWINT у регістрі TWCR, а вміст регістра TWSR оновлюється кодом стану, що говорить про успішність передачі адресного пакета. У коді стану також відображається, чи було підтвердження прийому адресного пакета з боку підлеглого, чи ні.

5. Виконується програмна перевірка значення TWSR, щоб переконатися в успішності передачі адресного пакета, а також що біт підтвердження має очікуване значення. Якщо TWSR показує іншу ситуацію, то при необхідності виконуються особливі дії, наприклад, викликається процедура обробки помилкових ситуацій. Якщо ж код стану має очікуване значення, то програма записує пакет даних в TWDR. Згодом у регістр TWCR записується специфічне значення, що служить командою для TWI і викликає апаратну передачу даних, записаних в TWDR. Запис лог. 1 в TWINT приводить до скидання цього прапора. TWI не почне роботу доти, поки буде встановлений біт TWINT у регістрі TWCR. Відразу після скидання TWINT починається передача пакета даних.

6. Після передачі пакета даних встановлюється прапор TWINT у регістрі TWCR, а вміст регістра TWSR оновлюється значенням коду стану, що сигналізує про успішну передачу пакета даних. У коді стану також відображається, чи було прийняте підтвердження від підлеглого, чи ні.

7. Виконується програмна перевірка значення в TWSR, щоб переконатися в успішності передачі пакета даних і у тому, що біт підтвердження має очікуване значення. Якщо TWSR сигналізує про іншу ситуацію, то програма виконує особливі дії, у тому числі викликає процедуру обробки переривання. Якщо код стану має очікуване значення, то виконується запис спеціального значення в TWCR, що служить командою для TWI та ініціює передачу умови СТОП. Під час запису повинна бути виконана установка біта TWINT. Запис лог. 1 в TWINT приводить до очищення цього прапора. TWI не почне роботу доти, поки

встановлено біт TWINT у регістрі TWCR. Відразу після скидання прапора TWINT ініціюється передача умови СТОП. Прапор TWINT не встановлюється по завершенню передачі умови СТОП.

Цей приклад показує принципи, покладені в основу будь-якої передачі через TWI.

5.6. Приклади пристроїв з інтерфейсом I2C

I2C – абоненти жорстко розділяються по класах: «Master» і «Slave» – пристрій. Той факт, що сигнал SCL завжди генерується «Master» – пристроєм означає, що «Master» – абонент може бути досить легко реалізований чисто програмними засобами, тому що всі зміни на шині будуть відбуватися тільки по сигналу SCL. Реалізація Slave» – пристрою вимагає апаратної підтримки. У типовій мікропроцесорній системі із застосуванням I2C – пристроїв дві лінії портів відведені для керування лініями SDA і SCL.

5.6.1. Цифровий термометр на базі DS1621

На рис. 5.10 представлена схема мікропроцесорної системи контролю температури. У схемі використовується датчик DS1621, з'єднаний з мікроконтролером AT90S8535 по шині I2C.

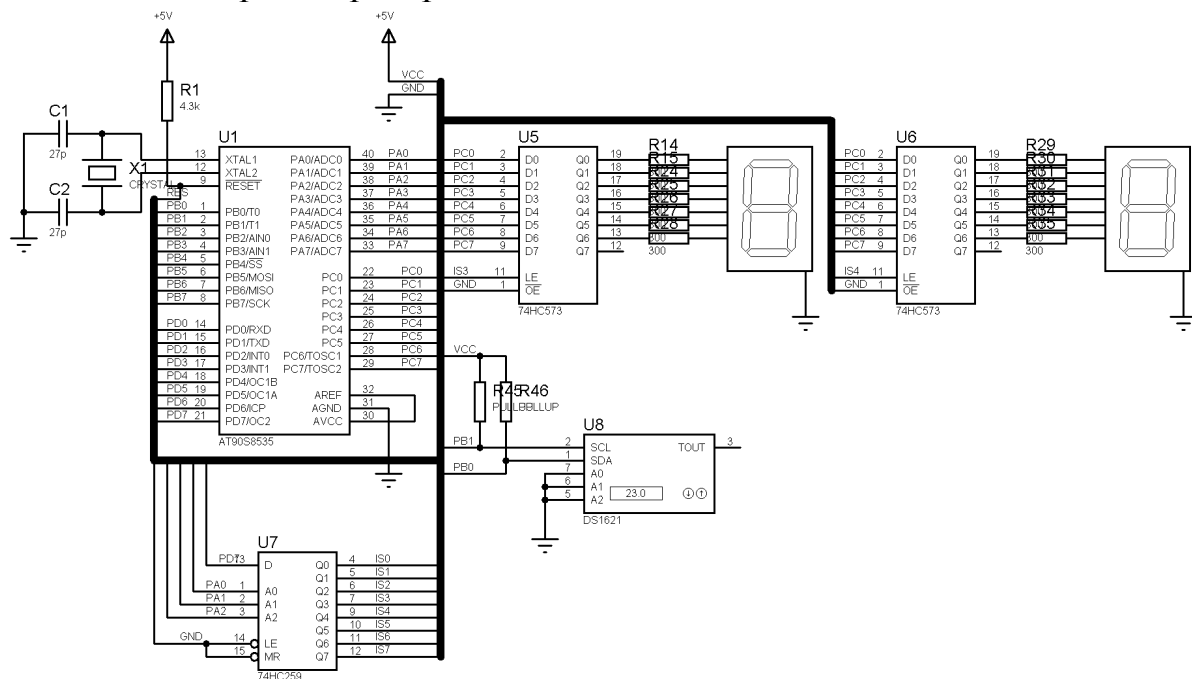


Рис. 5.10.

Мікросхема DS1621 – це цифровий термометр і термостат. Прилад є закінченим конвертером класу «температура – цифра» і не вимагає при використанні ніяких зовнішніх компонентів. Негативні значення температури представлені в коді з доповненням до 2-х. Прилад складається з п'яти основних компонентів:

– прецизійний датчик температури;

- аналого-цифровий перетворювач;
- електроніка двохпровідного інтерфейсу;
- регістр даних;
- компаратор термостата.

Ведучий шини може періодично зчитувати значення з температурного регістра, що містить результат попереднього перетворення.

Перший байт, переданий ведучим пристроєм після умови «Старт», містить адресу пристрою. Для DS1621 це значення 1001. Далі передаються 3 біти вибору пристрою A2, A1, A0. Якщо використовується тільки один пристрій на шині I2C, то значення цих біт може бути 0 за умови, що однойменні виводи мікросхеми заземлені. Останній біт – біт зчитування/запису – визначає напрямок передачі даних. Після цього команда, що визначає подальші дії по організації обміну даними. Набір команд для DS121 приводиться в табл. 5.1.

Табл. 5.1.

Команда	Опис	Код команди
Зчитування температури	Зчитує останнє значення температури з регістра температури	0xAA
Зчитування лічильника	Зчитує значення відліку, що залишається в лічильнику	0xA8
Зчитування TH	Зчитування збереженого значення межі високої температури з регістра TH, R/W=0	0xA1
Зчитування TL	Зчитування збереженого значення межі низької температури з регістра TL, R/W=0	0xA2
Запис TH	Запис значення межі високої, температури в регістр TH, R/W=1	0xA1
Запис TL	Запис значення межі низької температури в регістр TL, R/W=1	0xA2
Запис конфігурації	Запис даних конфігурації в регістр конфігурації, R/W=1	0xAC
Зчитування конфігурації	Зчитування даних з регістра конфігурації, R/W=0	0xAC
Почати перетворення	Початок перетворення температури, R/W=0	0xEE
Зупинити перетворення	Зупинка перетворення температури в безперервному режимі, R/W=0	0x22

Стан датчика DS1621, а також його режим роботи, визначається за допомогою регістра стану й керування:

DONE	THF	TLF	NVB	1	0	POL	1SHOT

DONE – біт завершення перетворення температури: «1» – перетворення закінчене, «0» – перетворення виконується.

THF – цей біт встановлюється якщо температура перевищує значення, встановлене в регістрі TH. Біт можна скинути шляхом запису 0 у цю позицію, або відключенням живлення датчика.

TLF – цей біт встановлюється якщо температура менше значення, встановленого в регістрі TL. Біт можна скинути шляхом запису 0 у цю позицію, або відключенням живлення датчика.

NVB – прапор зайнятості пам'яті: «1» – виконується запис у комірку пам'яті.

POL – біт полярності вихідного сигналу TOUT: «1» – активний високий рівень, «0» – активний низький рівень.

1SHOT – якщо значення біта «1», перетворення температури виконується 1 раз. Якщо значення біта «0» – перетворення температури виконується безупинно.

У табл. 5.2 представлена відповідність значення температури цифровому коду.

Табл. 5.2.

Температура, °C	Код (BIN)	Код (HEX)
+ 125	01111101 00000000	7B00h
+ 25	00011001 00000000	1900h
+ 0,5	00000001 00000000	0080h
0	00000000 00000000	0000h
– 0,5	11111111 10000000	FF80h
– 25	11100111 00000000	E700h
– 55	11001001 00000000	C900h

Приклад 5.1. Написати програму для мікроконтролера AT90S8535, що виконує зчитування температури з датчика DS1621 по інтерфейсу I2C і виводить значення температури на 2 семисегментних індикатори.

Текст програми для мікроконтролера, створеної з використанням компілятора ImageCraft, представлений нижче.

```
#include <io8535v.h> // підключення заголовних файлів
#include <macros.h>
#define SETSDA() PORTB|=0b00000001 // уст. сигналу SDA
#define SETSCL() PORTB|=0b00000010 // уст. сигналу SCL
#define RSTSDA() PORTB&=~0b00000001 // скидання сигналу SDA
#define RSTSCL() PORTB&=~0b00000010 // скидання сигналу SCL

// функція ініціалізації контролера
void init_cpu()
{
    DDRA=0x07; // 3 молодших біти порту A налаштовані на вихід
    DDRC=0xFF; // порт C налаштований на вихід
    DDRD=0x80; // налаштування виводу 7 порту D на вихід
    PORTD=0x80; // установка 1 на виводі 7 порту D
}
```

```

// функція затримки; параметр – затримка в мс
void delay_ms(int ms)
{
    int c1, c2;           // оголошення додаткових змінних
    for(c1=0;c1<ms;c1++)   // у циклі виконується «порожня»
        for(c2=0;c2<970;c2++) NOP();    // операція
}

// функція затримки; параметр - кількість тактів затримки
void delay_us(int st)
{
    int c1;               // оголошення додаткової змінної
    for(c1=0;c1<st;c1++) NOP();// у циклі виконується «порожня» операція
}

// функція перевірки «зайнятості» шини
char i2c_isbusy()
{
    char st;              // додаткова змінна
    DDRB=0x00;            // налаштування порту В на вхід
    PORTB=0x00;           // відключення резисторів, що підтягують
    if((PINB&0x03)!=0x03) st=1; // перевірка стану ліній
    else st=0;            // функція повертає 0 якщо шина вільна
    return st;            // інакше повертається значення 1
}

// функція формування сигналу початку передачі даних
void i2c_start()
{
    DDRB=0x03;            // налаштування 2-х біт порту В на вихід
    SETSDA();             // установка сигналу SDA
    SETSCL();             // установка сигналу SCL
    delay_us(2);          // затримка
    RSTSDA();             // скидання сигналу SDA
    delay_us(2);          // затримка
    RSTSCL();             // скидання сигналу SCL
    delay_us(2);          // затримка
}

// функція формування сигналу завершення обміну даними
void i2c_stop()
{
    DDRB=0x03;            // налаштування 2-х біт порту В на вихід
    RSTSDA();             // скидання сигналу SDA
    SETSCL();             // установка сигналу SCL
    delay_us(2);          // затримка
    SETSDA();             // установка сигналу SDA
    delay_us(2);          // затримка
}

// функція посилки 1 байта; параметр - переданий байт

```

unsigned char i2c_sendbyte(**char** d)

```
{
    unsigned char i=8,acknowledge=0; // додаткові змінні
    do {
        if((d & 0x80)!=0) SETSDA(); // установка сигналу SDA якщо
        // старший біт дорівнює 1
        else RSTSDA(); // інакше скидання сигналу SDA
        d =d << 1; // зсув переданого байта на 1 розряд ліворуч
        delay_us(2); // формування затримки
        // формування тактового імпульсу
        SETSCL(); // установка сигналу SCL
        delay_us(2); // затримка
        RSTSCL(); // скидання сигналу SCL
        delay_us(2); // затримка
    }
    while (--i); // цикл виконується 8 разів
    // формування тактового імпульсу для біта підтвердження
    DDRB=0x02; // переведення лінії SDA у третій стан
    RSTSDA(); // скидання сигналу SDA
    SETSCL(); // установка сигналу SCL
    delay_us(2); // затримка
    if ((PINB&0x01)==0) acknowledge=1; // якщо на лінії SCL низький
    // рівень, біт підтвердження прийнятий
    RSTSCL(); // скидання сигналу SCL
    delay_us(2); // затримка
    DDRB=0x03; // налаштування 2-х біт порту В на вихід
    return acknowledge; // повертається біт підтвердження
}
```

// функція зчитування одного байта

unsigned char i2c_readbyte(**unsigned char** acknowledge)

```
{
    unsigned char i=8,i2c_data=0; // додаткові змінні
    char d;
    DDRB=0x02; // переведення лінії SDA у третій стан
    do {
        i2c_data<<=1; // зсув на 1 розряд ліворуч
        SETSCL(); // установка сигналу SCL
        delay_us(2); // затримка
        d=PINB; // зчитування стану порту В
        d=d&0x01; // виділення 0 біта
        i2c_data|=d; // копіювання 0 біта в змінну
        RSTSCL(); // скидання сигналу SCL
        delay_us(2); // затримка
    }
    while (--i); // цикл виконується 8 разів
    if (acknowledge) // якщо необхідний біт підтвердження
    {
        DDRB=0x03; // налаштування 2-х біт на вихід
        RSTSDA(); // скидання сигналу SDA
    }
    // формування тактового імпульсу
}
```



```

SETSCCL();           // установка сигналу SCL
delay_us(2);         // затримка
RSTSCCL();           // скидання сигналу SCL
DDRB=0x02;           // переведення лінії SDA у третій стан
return i2c_data;     // функція повертає прийнятий байт
}

// головна функція
int main ()
{
    int c1=0,c2=0;    // оголошення додаткових змінних
    int t1, t2;
    char seg[10]={0b00111111, 0b00000110, 0b01011011, 0b01001111,
0b01100110, 0b01101101, 0b01111101, 0b00000111, 0b01111111,
0b01101111};        // дані з кодами символів для 7-сегм. індикатора
    init_cpu();        // виклик функції ініціалізації контролера
    while(1)           // вічний цикл
    {
        while (i2c_isbusy()!=0; // чекаємо поки лінія не звільниться
        DDRB=0x03;        // налаштування 2-х молодших біт порту В на вихід
        i2c_start();      // функції формування сигналу початку передачі
        i2c_sendbyte(0b10010000); // відправлення байта 0x90 (адреса)
        !i2c_sendbyte(0xEE); // відправлення команди 0xEE
        i2c_stop();       // формування сигналу закінчення передачі
        i2c_start();      // початок передачі даних
        i2c_sendbyte(0b10010000); // відправлення байта 0x90 (адреса)
        i2c_sendbyte(0xAA); // відправлення команди 0xAA
        i2c_stop();       // завершення передачі даних
        i2c_start();      // повторний старт
        i2c_sendbyte(0b10010001); // відправлення команди зчитування
        c1=i2c_readbyte(1); // зчитування старшого байта температури
        c2=i2c_readbyte(0); // зчитування молодшого байта температури
        i2c_stop();       // звільнення лінії I2C
        t1=c1/10;         // формування значення температури (десятки)
        t2=c1-t1*10;      // формування значення температури (одиниці)
        PORTA=4;          // включення лівого індикатора
        PORTC=seg[t2];     // вивід значення температури на індикатор
        PORTA=3;          // включення правого індикатора
        PORTC=seg[t1];     // вивід значення температури на індикатор
        delay_ms(1000);    // затримка 1 секунда
    }
}

```

На рис. 5.11 представлена діаграма роботи інтерфейсу I2C. Тактовий сигнал SCL формує мікроконтролер на виводі PB1. На лінії даних SDA (вивід PB0) встановлюються рівні лог.1 та лог.0 відповідно до переданого байта. Після 8 тактових імпульсів, на 9-му відбувається перевірка біта підтвердження, що формує підлеглий пристрій (датчик температури DS1620).

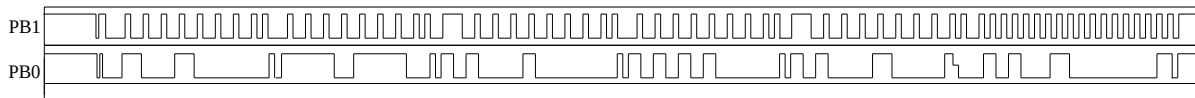


Рис. 5.11.

5.6.2. Електронний годинник на базі DS1307

Годинники реального часу (RTC – Real Time Clock) з послідовним інтерфейсом DS1307 – це двійково-десятковий годинник-календар, який містить 56 байт енергонезалежної статичної пам'яті. Адреси та дані передаються послідовно по двохпроводній двонапрямленій шині. Годинник-календар відраховує секунди, хвилини, години, день, дату, місяць і рік. Остання дата місяця автоматично коригується для місяців з кількістю днів менше 31, включаючи корекцію високосного року. Годинники працюють як в 24-годинному, так і в 12-годинному режимах з індикатором AM/PM. DS1307 має вбудовану схему спостереження за живленням, що виявляє перебої живлення і автоматично перемикається на живлення від батареї.

DS1307 на послідовній шині працює як підлеглий («Slave») пристрій. Доступ до нього виконується установкою умови START і передачею пристрою ідентифікаційного коду, за яким передається адреса регістра. До наступних за ним регістрів доступ здійснюється послідовно, поки не буде виконана умова STOP. Блок-схема на рис. 5.12 показує основні елементи RTC з послідовним інтерфейсом.

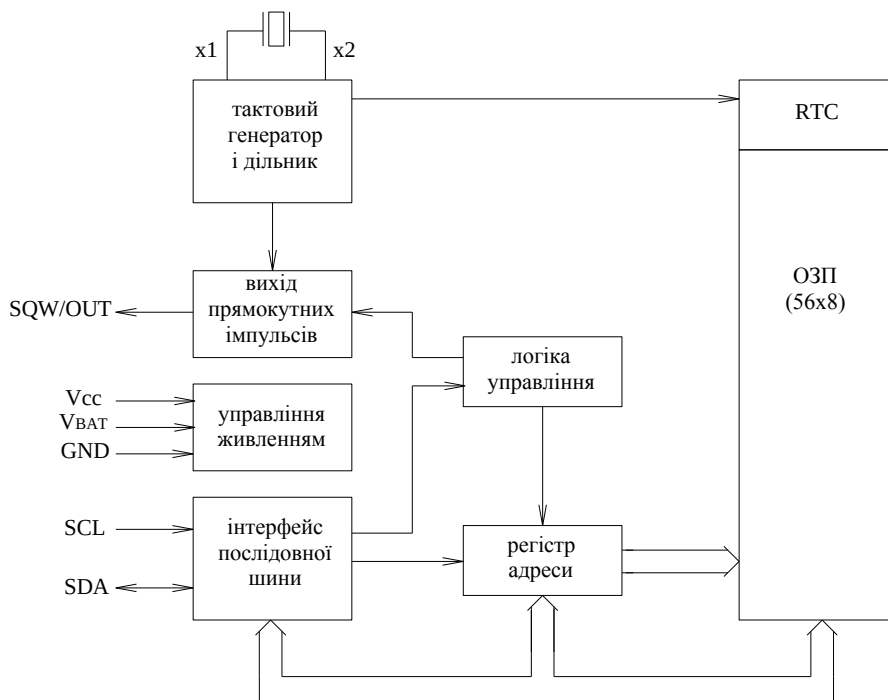


Рис. 5.12.

Типова схема підключення DS1307 представлена на рис. 5.13.

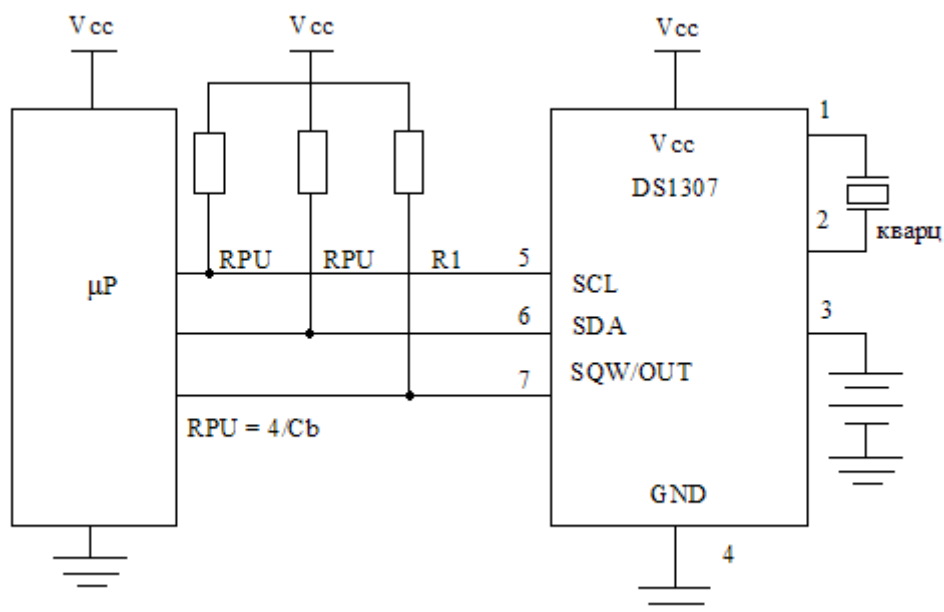


Рис. 5.13.

Карта адреси для RTC і регістрів ОЗП представлена на рис. 5.14. Регістри RTC розташовані в комірках пам'яті з адресами від 00h до 07h. Регістри ОЗП розташовані в комірках з адресами від 08h до 3Fh. У процесі багатобайтового доступу, коли вказівник адреси досягає 3Fh (кінець простору ОЗП), він переміщується на комірку 00h – початок простору RTC.

00H	СЕКУНДИ
	ХВИЛИНИ
	ГОДИНИ
	ДНІ ТИЖНЯ
	ДАТА
	МІСЯЦЬ
	РІК
07H	УПРАВЛІННЯ
08H	ОЗП
3FH	56 x 8

Рис. 5.14.

Регістри RTC показані на рис. 5.15. Час і календар встановлюються або ініціюються записом відповідних байтів регістра. Вміст регістрів годинника та календаря має двійково-десятковий формат. Біт 7 регістра 0 – це біт зупинки годинника (clock halt – CH). Коли цей біт встановлений в 1, тактовий генератор виключений. Коли скинутий в 0 – тактовий генератор включений.

DS1307 може працювати як в 12-годинному, так і в 24-годинному режимах. Біт 6 регістра годин відповідає за вибір 12- або 24-годинного

режиму. Коли він встановлений в 1, встановлюється 12-годинний режим. У цьому режимі біт 5 – це біт AM/PM, при цьому високий логічний рівень означає PM. В 24-годинному режимі біт 5 – це біт другого десятка годин (20 – 23 години).

З появою на двохпровідній шині умови START, значення поточного часу копіюється в другий набір регістрів. Інформація про час зчитується із цих допоміжних регістрів, і годинник може продовжувати працювати. Це усуває необхідність повторного зчитування вмісту регістрів DS1307 у випадку оновлення їхнього вмісту під час операції зчитування.

	Біт 7							Біт0	
00H	CH	ДЕСЯТКИ СЕКУНД			СЕКУНДИ				00 - 59
	0	ДЕСЯТКИ ХВИЛИН			ХВИЛИНИ				00 - 59
	0	12 24	ДРУГИЙ ДЕСЯТОК А/Р	ПЕРШИЙ ДЕСЯТОК ГОДИН	ГОДИНИ				01 - 12 00 - 23
	0	0	0	0	0	ДЕНЬ			1 - 7
	0	0	ДЕСЯТКИ ДАТИ		ДАТА				01 - 28/9 01 - 30 01 - 31
	0	0	0	ДЕСЯТОК МІСЯЦЯ	МІСЯЦЬ				01 - 12
	ДЕСЯТКИ РОКІВ				РІК				00 - 99
07H	OUT	0	0	SQWE	0	0	RS1	RS0	

Рис. 5.15.

Керуючий регістр використовується для керування роботою виводу SQW/OUT.

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
OUT	0	0	SQWE	0	0	RS1	RS0

OUT (Output control – керування виходом) – цей біт керує логічним рівнем на виводі SQW/OUT, коли вихід сигналу із прямокутними імпульсами відключений. Якщо SQWE = 0, то логічний рівень на виводі SQW/OUT дорівнює 1, якщо OUT = 1, і 0, якщо OUT = 0.

SQWE (Square Wave Enable – включення сигналу із прямокутними імпульсами) – коли цей біт встановлений в 1, включається генерація прямокутних імпульсів. Їхня частота залежить від значення бітів RS0 та RS1. (Якщо частота вихідного сигналу встановлена в 1 Гц, регістри годин оновлюються по задньому фронту цих імпульсів).

RS (Rate Select – вибір частоти) – ці біти керують частотою прямокутних імпульсів, коли біт SQWE=1. У табл. 5.3 наведені частоти, які можна вибрати за допомогою бітів RS.

Табл. 5.3.

RS1	RS0	Частота
0	0	1 Гц
0	1	4.096 кГц
1	0	8.192 кГц
1	1	32.768 кГц

Приклад 5.2. Розробити принципову схему годинника з виводом інформації на алфавітно-цифровий рідкокристалічний індикатор, з використанням мікросхеми DS1307 і мікроконтролера ATMEGA128. Написати програму для мікроконтролера, що виконує зчитування дати та часу по інтерфейсу I2C. Використати вбудований периферійний вузол TWI мікроконтролера.

Принципова схема годинника представлена на рис. 5.16.

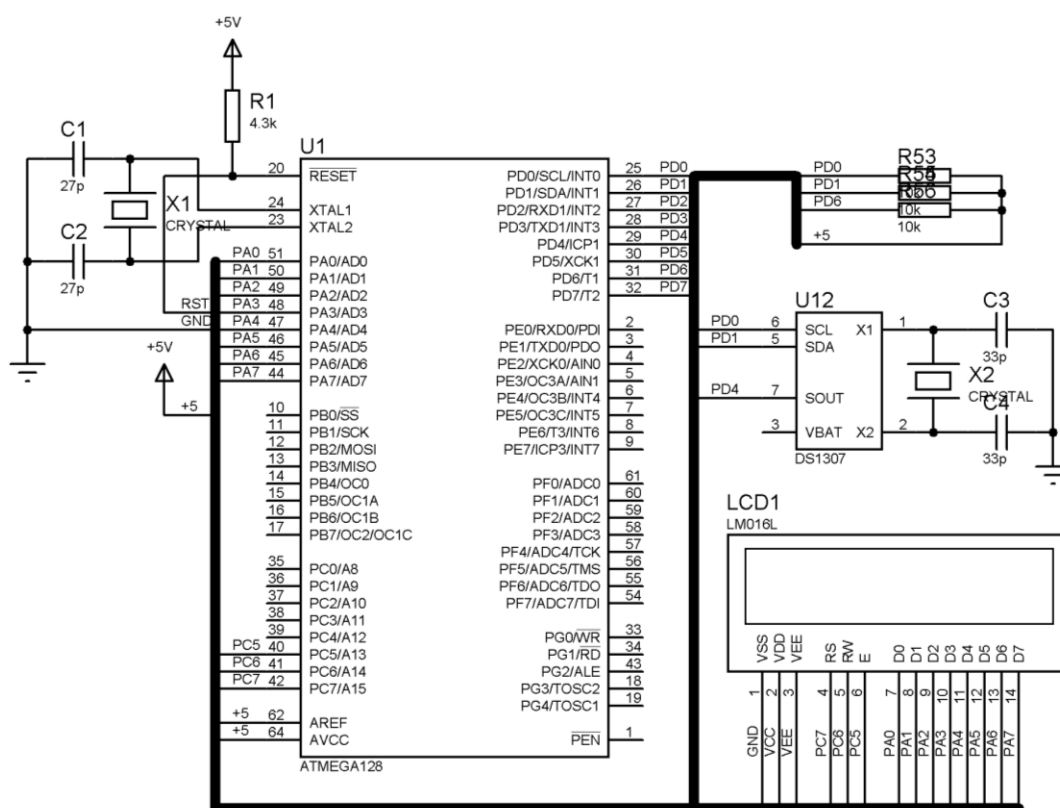


Рис. 5.16.

Текст програми для мікроконтролера, створеної з використанням компілятора ImageCraft, представлений нижче.

```
#include <iom128v.h>
#include <macros.h>
#define RS() PORTC|=0b10000000 // встановити сигнал команда/дані
#define NRS() PORTC&=~0b10000000 // скинути сигнал команда/дані
// встановити сигнал читання/запис (біт 6 порту A)
#define RW() PORTC|=0b01000000
```

```
// скинути сигнал читання/запис
#define NRW() PORTC&=~0b01000000
#define E() PORTC|=0b00100000 // встановити сигнал стробу
#define NE() PORTC&=~0b00100000 // скинути сигнал стробу
#define LCD PORTA // PKI модуль підключений до порту C
#define DDLCD DDRA // даємо назву регістру напрямку даних

void init_cpu() // функція ініціалізації мікроконтролера
{
    DDRC=0b11100000; // керування дешифратором шини даних
    DDRA=0xFF; // шина даних
    PORTC=0;
    PORTA=0xFF; // налаштування порту A на вихід
    PORTC=5;
    PORTA=0;
    NRS(); // скидання сигналу команда/дані
    NRW(); // скидання сигналу зчитування/запис
    NE(); // скидання тактового сигналу
}

void pause() // функція затримки
{
    int tmp; // додаткова змінна для використання в циклі
    // контролер виконує пусті операції
    for(tmp=0;tmp<450;tmp=tmp+1) NOP();}

void init_lcd() // функція ініціалізації PKI
{
    DDLCD=0xFF; // налаштування порту на вихід
    NRS(); // скидання сигналу команда/дані - передаються команди
    NRW(); // скидання сигналу запис/зчитування - запис
    LCD=0x38; // вивід команди 0x38 на порт контролера
    write(); // запис команди в модуль
    LCD=0x0C; // вивід команди 0x0C на порт контролера
    write(); // запис команди в модуль
    LCD=0x06; // вивід команди 0x06 на порт контролера
    write(); // запис команди в модуль
    LCD=0x02; // вивід команди 0x02 на порт контролера
    write(); // запис команди в модуль
    LCD=0x00; // вивід значення 0 на порт контролера
    DDLCD=0x00; // перевід порту в стан високого імпедансу
}

void write() // функція формування тактового сигналу
{
    E(); // установка тактового сигналу
    pause(); // часова затримка
    NE(); // скидання тактового сигналу
    pause(); // часова затримка
}
```

```

void delay_ms(int ms)                                // функція затримки
{
    int c1, c2;
    for(c1=0;c1<ms;c1++)
        for(c2=0;c2<970;c2++);
}

void fi2c_init(void)
{
    TWCR= 0x00;                                        // відключити TWI
    TWBR= 0x02;                                        // установка значення в регістрі дільника
    TWSR= 0x02;                                        // установка значення дільника частоти
    TWCR= 0x04;                                        // дозвіл роботи TWI
}

unsigned char fi2c_start(void)                      //функція формування start-умови
// повертається 1 якщо шина вільна, або 0 якщо шина зайнята
{
    // формування start-умови
    TWCR = (1<<TWEN)|(1<<TWINT)|(1<<TWSTA);
    // очікування установки прапора TWINT
    while (!(TWCR & (1<<TWINT)));
    // при відсутності помилок повертається значення 1
    if (((TWSR&0x08)==0x08) || ((TWSR&0x08)==0x10)) return 1;
    // якщо виникла помилка при формуванні start-умови, повертається 0
    return 0;
}

void fi2c_stop(void)                                // функція формування STOP-умови
{
    TWCR = (1<<TWEN)|(1<<TWINT)|(1<<TWSTO);
}

// функція зчитування байта з шини
unsigned char fi2c_read(unsigned char ack) // параметр функції:
// 0 - сигнал підтвердження ack не формується, 1 - формується
{
    unsigned char data;                                // додаткова змінна
    // якщо параметр переданий у функцію дорівнює 0
    if (ack == 0) TWCR = (1<<TWEN)|(1<<TWINT); // біт TWEA не встановлюється
    // інакше біт TWEA установлений
    else TWCR = (1<<TWEN)|(1<<TWINT)|(1<<TWEA);
    // очікування установки біта TWINT
    while (!(TWCR & (1<<TWINT)));
    data = TWDR;                                        // зчитування даних з регістра
    return data;                                        // функція повертає зчитане значення
}

// функція передачі байта
unsigned char fi2c_write(unsigned char data)
{
    TWDR = data;                                        // запис значення в регістр даних
}

```

```
TWCR &= 0xCF;           // біти формування умов start і stop скинуті
TWCR |= 0x84;           // установка біт TWINT і TWEN в 1
// очікування завершення передачі даних
while (!(TWCR & (1<<TWINT)));
// $40 SLA+R відправлений, $18 SLA+W відправлений, $28 дані
// відправлені й ACK отриманий
if (((TWSR&0xF8)==0x18) || ((TWSR&0xF8)==0x28) || ((TWSR&0xF8)
==0x40)) return 1;
// $20 SLA+W відправлений, $30 - дані відправлені, ACK не
// отриманий, $48 SLA+R відправлений
if (((TWSR&0xF8)==0x20) || ((TWSR&0xF8)==0x30) || ((TWSR&0xF8)
==0x48)) return 0;
return 0;
}
```

```
void main ()
{
    char h, m, s, y, mon, day, d, i;
    char txt[6]="Time: ";
    char txt1[6]="Date: ";
    init_cpu();    // виклик функції ініціалізації мікроконтролера
    init_lcd();
    while(1)      // вічний цикл
    {
        fi2c_init();           // ініціалізація TWI
        fi2c_start();          // формування start-умови
        fi2c_write(0b11010000); // відправлення адреси пристрою
        fi2c_write(0);         // установка вказівника адреси в комірку 0
        fi2c_stop();           // формування stop-умови
        delay_ms(1);           // пауза 1 мс
        fi2c_start();          // формування start-умови
        fi2c_write(0b11010001); // відправлення адреси пристрою
        s=fi2c_read(1);         // зчитування значення секунд
        m=fi2c_read(1);         // зчитування хвилин
        h=fi2c_read(1);         // зчитування годин
        day=fi2c_read(1);       // зчитування номера дня
        d=fi2c_read(1);         // зчитування дати
        mon=fi2c_read(1);       // зчитування номера місяця
        y=fi2c_read(0);         // зчитування року
        fi2c_stop();           // формування stop-умови
        NRS();                 // у РКІ-модуль передаються команди
        LCD=0x01;              // очищення екрана
        write();               // запис команди в РКІ-модуль
        RS();                  // установка сигналу передачі даних
        delay_ms(1);           // пауза 1 мс
        for(i=0;i<6;i++)       // у циклі по буквах передається текст
        {
            LCD=txt[i];        // вивід коду символу в порт
            write();            // запис даних у модуль
        }
        LCD=((h>4)&0x07)+48;    // вивід десятків годин
        write();               // запис даних у модуль
    }
}
```



```

LCD=(h&0x0F)+48;           // вивід одиниць годин
write();
LCD=': ';                   // вивід двокрапки на екран
write();
LCD=((m>>4)&0x07)+48;       // вивід десятків хвилин
write();
LCD=(m&0x0F)+48;           // вивід одиниць хвилин
write();
LCD=': ';                   // вивід двокрапки на екран
write();
LCD=((s>>4)&0x07)+48;       // вивід десятків секунд
write();
LCD=(s&0x0F)+48;           // вивід одиниць секунд
write();
NRS();                      // установка сигналу передачі команд
delay_ms(1);                // пауза 1 мс
LCD=0b10101000;            // команда установки курсору на нижній рядок
write();                    // запис команди в РКІ
RS();                       // установка сигналу передачі даних
delay_ms(1);                // пауза 1 мс
for(i=0;i<6;i++)            // в циклі по буквах виводиться текст
{
    LCD=txt1[i];             // вивід коду символу в порт
    write();                 // запис даних у модуль
}
LCD=((d>>4) &0x07)+48;       // вивід дати на екран РКІ
write();
LCD=(d&0x0F)+48;
write();
LCD='.';                    // вивід крапки на екран
write();
LCD=((mon>>4)&0x07)+48;       // вивід номера місяця
write();
LCD=(mon&0x0F)+48;
write();
LCD='.';                    // вивід крапки на екран
write();
LCD=((y>>4)&0x07)+48;         // вивід значення року
write();
LCD=(y&0x0F)+48;
write();
delay_ms(500);              // пауза 500 мс
}
}

```

Питання до розділу 5

1. Призначення та сфера застосування інтегральних пристроїв з послідовним інтерфейсом I2C.
2. Часові параметри сигналів інтерфейсу I2C.
3. Протокол обміну по інтерфейсу I2C.
4. Призначення зовнішніх адресних ліній на мікросхемах з I2C.
5. Особливості використання декількох пристроїв на одному інтерфейсі.
6. Методи адресації різних пристроїв з I2C.
7. Яким чином формуються умови СТАРТ та СТОП?
8. Які пристрої мають вбудований інтерфейс I2C?
9. У яких випадках передається умова «повторний старт»?
10. Чи завжди формується біт підтвердження Ack?

Задачі до розділу 5

1. Розробити програму контролю температури. Значення температури зчитується з датчика. Якщо температура перевищує заданий поріг, то в пам'ять записується значення часу, коли температура вийшла за задану межу. Управління датчиком температури, годинником реального часу і мікросхемою пам'яті здійснюється по шині I2C.
2. На семисегментні індикатори виводити значення часу, зчитане з мікросхеми RTC.
3. В мікросхему пам'яті необхідно записувати значення, зчитане з АЦП, з інтервалом 1 с. За запитом дані зчитуються з пам'яті і передаються в комп'ютер. Після цього пам'ять очищається.
4. Розробити схему пристрою та написати програму, яка зчитує значення температури по шині I2C з мікросхеми DS1621, і виводить її на семисегментні індикатори.
5. Написати програму, що працює з мікросхемою пам'яті. У комірку пам'яті записуються дані, потім зчитуються і виводяться на семисегментні індикатори. Реалізувати алгоритм двома способами: використовуючи вбудований інтерфейс TWI та інтерфейс, реалізований програмно.

6. ШИНА 1-WIRE

Однопровідний інтерфейс 1-Wire, розроблений наприкінці 90-х років фірмою Dallas Semiconductor Corp., регламентований розробниками для застосування в трьох основних сферах:

- прилади в спеціальних корпусах MicroCAN для рішення проблем ідентифікації, переносу або перетворення інформації (технологія iButton);
- програмування вбудованої пам'яті інтегральних компонентів;
- системи автоматизації (технологія мереж 1-Wire).

Переваги 1-Wire:

- просте й оригінальне рішення адресації абонентів;
- зручний протокол;
- проста структура лінії зв'язку;
- невелика кількість необхідних компонентів;
- легка зміна конфігурації мережі;
- значна довжина ліній зв'язку;
- невелика вартість всієї технології в цілому.

6.1. Загальні характеристики інтерфейсу

Мережа 1-Wire представляє собою інформаційну мережу, що використовує для здійснення цифрового зв'язку одну лінію даних та один загальний провід. Таким чином, для реалізації середовища обміну цієї мережі можуть бути застосовані доступні кабелі, що містять неекрановану кручену пару, і навіть звичайний телефонний провід. Такі кабелі при їхній прокладці не вимагають наявності якого-небудь спеціального устаткування, а обмеження максимальної довжини однопровідної лінії регламентовано розробниками на рівні 300 м. Основою архітектури мереж 1-Wire є топологія загальної шини, коли кожний із пристроїв підключений безпосередньо до єдиної магістралі, без яких-небудь каскадних з'єднань або розгалужень. При цьому в якості базової використовується структура мережі з одним ведучим, або майстром, і багатьма веденими. Хоча існує ряд специфічних прийомів організації роботи однопровідних систем у режимі мультимастера. Конфігурація будь-якої 1-Wire-мережі може довільно змінюватися в процесі її роботи, не створюючи перешкод подальшій експлуатації та працездатності всієї системи в цілому, якщо при цих змінах дотримуються основні принципи організації однопровідної шини. Ця можливість досягається завдяки присутності в протоколі 1-Wire-інтерфейсу спеціальної команди пошуку ведених пристроїв (пошук ПЗП), що дозволяє швидко визначити нових учасників інформаційного обміну. Стандартна швидкість відпрацьовування такої команди становить ~75 вузлів мережі в секунду.

Завдяки наявності в складі будь-якого пристрою з мережною версією 1-Wire-інтерфейсу унікальної індивідуальної адреси (відсутність збігу адрес для приладів, що випускаються компанією Dallas Semiconductor Corp, гарантується самою фірмою-виробником), така мережа має практично необмежений адресний простір. При цьому кожний з однопровідних приладів відразу готовий до використання в складі 1-Wire-мережі, без яких-небудь додаткових апаратно-програмних модифікацій. Однопровідні компоненти є самотактованими напівпровідниковими пристроями, в основі обміну інформацією між якими лежить керування зміною тривалості часових інтервалів імпульсних сигналів в однопровідному середовищі, та їх вимірювання. Передача сигналів, для 1-Wire-інтерфейсу, асинхронна й напівдуплексна, а вся інформація, що циркулює в мережі, сприймається абонентами або як команди, або як дані. Команди мережі генеруються майстром і забезпечують різні варіанти пошуку й адресації ведених пристроїв, визначають активність на лінії навіть без безпосередньої адресації окремих компонентів, керують обміном даними в мережі й т.д.

Стандартна швидкість роботи 1-Wire-мережі, яка становить 15,4 кБіт/с, була обрана, по-перше, з урахуванням забезпечення максимальної надійності передачі даних на великі відстані, і по-друге, з урахуванням швидкодії найбільш широко розповсюджених типів мікроконтролерів, які в основному повинні використовуватися при реалізації ведучих пристроїв однопровідної шини. Це значення швидкості обміну може бути зменшене до будь-якого можливого значення завдяки введенню примусової затримки між передачею в лінію окремих бітів даних (розтягуванню часових слотів протоколу), або збільшено за рахунок переходу на спеціальний прискорений режим обміну (швидкість Overdrive – до 125 кБіт/с), що допускається для окремих типів однопровідних компонентів на невеликій відстані, не перевантаженої іншими приладами лінії зв'язку.

Окремі види адаптерів, які дозволяють наділити будь-який персональний комп'ютер можливістю обслуговувати як майстер 1-Wire-мережі, випускаються самою фірмою Dallas Semiconductor Corp. До них відносяться адаптери для паралельного порту типу DS1410E, для COM-порту типу DS9097E та DS9097U, для USB-порту типу DS9490R. Ці прилади мають різні функціональні можливості й конструктивні особливості, що забезпечує розробнику максимальну зручність при конструюванні.

Часто як ведучий однопровідної шини виступає не комп'ютер, а найпростіший універсальний мікроконтролер. Для організації його сполучення з 1-Wire-мережею використовуються різні програмно-апаратні методи.

Чотирьохканальний однопровідний АЦП типу DS2450 та двоканальний однопровідний лічильник, сполучений з буферною пам'яттю, типу DS2423 дозволяють вирішувати завдання, пов'язані з оцифруванням аналогових та імпульсно-часових сигналів. Найбільш

незамінними при автоматизації є універсальні здвоєні транзисторні ключі типу DS2406P. На базі цих пристроїв можуть бути реалізовані вузли контролю логічних станів (рівнів) і схеми обслуговування датчиків "сухого контакту", а також різноманітні ключові схеми. Саме завдяки використанню цих компонентів здійснюється збір дискретної інформації з територіально розосереджених датчиків (моніторів дверей, контакторів положення арматури, будь-яких датчиків, що мають вихід ТАК/НІ – датчики положення, проходу, присутності, пожежної та охоронної сигналізації та ін.). Подібні ж прилади забезпечують керування перемиканням будь-яких видів силового устаткування, які мають два робочих стани: включене/виключене (нагрівачів, кондиціонерів, моторів, вентиляторів і т.д.). Значно розширює можливості однопровідних мереж по аналоговому керуванню розосередженим, у тому числі силовим, устаткуванням цифровий потенціометр DS2890 укомплектований мережним 1-Wire-інтерфейсом. Використовуючи цей прилад можна створювати найрізноманітніші системи керування, завдяки можливості плавної зміни аналогового регулюючого сигналу по 256 градаціям.

6.2. Блок-схема датчика температури DS18B20

Найбільш популярними компонентами, на базі яких реалізована найбільша кількість пристроїв, є цифрові термометри типу DS18S20. Розглянемо принципи роботи й програмування пристроїв з 1-Wire на прикладі датчика температури DS18B20 і мікроконтролера AT90S8535.

Широко розповсюджена мікросхема цифрового термометра DS18B20 забезпечує вимір температури в діапазоні $-55...+125^{\circ}\text{C}$. Точність перетворення температури може бути від 9 до 12 розрядів (налаштовується). DS18B20 допускає напругу живлення від +3 до +5,5 В. У режимі очікування споживаний струм близький до нуля (менше 1мкА), а під час перетворення температури він дорівнює приблизно 1 мА. Процес перетворення триває максимум 750 мс. Температурні датчики дозволяють не тільки здійснювати безпосередній моніторинг температури в режимі реального часу, але й завдяки наявності вбудованої енергонезалежної пам'яті температурних уставок, можуть забезпечувати пріоритетну сигналізацію в лінію про факт виходу контрольованого параметра за межі заданих значень.

Принцип дії такого цифрового датчика температури заснований на підрахунку кількості імпульсів, що виробляються генератором з низьким температурним коефіцієнтом у часовому інтервалі, який формується генератором з більшим температурним коефіцієнтом. Лічильник ініціюється значенням, що відповідає -55°C (мінімальній вимірюваній температурі). Якщо лічильник досягає нуля перед тим, як закінчується часовий інтервал (це означає, що температура більша -55°C), то регістр температури, що також ініційований значенням -55°C , інкрементується. Одночасно в лічильник завантажується нове значення, що задається схемою формування нахилу характеристики. Ця схема потрібна для

Проміжне ПЗП

Молодший байт температури	0
Старший байт температури	1
ТН / байт користувача 1	2
ТН / байт користувача 2	3
Зарезервовано	4
Зарезервовано	5
COUNT_REMEIN	6
COUNT_PER_C	7

Рис. 6.2.

Байти ТН та ТL представляють собою температурні пороги, з якими зрівнюються 8 біт кожного вимірюваного значення температури (молодший біт відкидається). За допомогою спеціальної команди можна організувати сигналізацію виходу температури за межі цих порогів. Якщо така функція не потрібна, байти ТН та ТL можна використати для зберігання будь-яких даних користувача.

Зчитування значення вимірної температури, а також передача команди початку перетворення та інших команд, виконується за допомогою 1-провідного інтерфейсу (1-Wire). На основі цього інтерфейсу фірма DALLAS створила мережу, що називається MicroLAN. Для роботи в цій мережі випускається цілий ряд пристроїв, таких як адресовані ключі, АЦП, термометри, годинники реального часу, цифрові потенціометри.

Протокол, що використовується 1-провідним інтерфейсом, досить простий. У будь-який момент часу на 1-провідній шині можна виділити пристрій-майстер, яким може бути мікропроцесор або комп'ютер, і підлеглий пристрій. Якщо на шині присутні тільки майстер і всього один підлеглий пристрій, можна опустити все те, що пов'язане з адресацією пристроїв. В результаті потрібно знати лише протокол передачі даних, які можуть бути командами або даними.

На рис. 6.3 показана апаратна конфігурація інтерфейсної частини DS18B20. У кожного однопровідного пристрою до шини підключений вхід приймача й вихід передавача з відкритим стоком. Відкритий стік дозволяє підключати до шини безліч пристроїв, забезпечуючи «монтажне або». Генератор струму 5 мкА забезпечує на вході 1-провідного пристрою низький логічний рівень, коли шина не підключена. Оскільки лінія тактового сигналу відсутня, обмін є асинхронним. Це означає, що в

процесі обміну потрібно досить точно витримувати необхідні часові співвідношення.

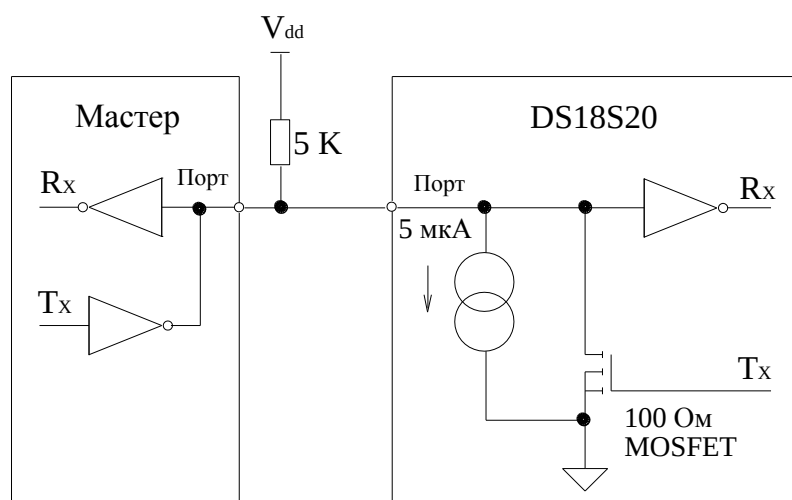


Рис. 6.3.

6.3. Протокол обміну даними

Однопровідна шина оперує з TTL-рівнями, тобто логічна одиниця представлена рівнем напруги близько 5 В, а логічний нуль – напругою поблизу 0 В. У вихідному стані на лінії присутній рівень логічної одиниці, що забезпечується резистором номіналом близько 5 кОм.

Ініціатором обміну по однопровідній шині завжди виступає майстер. Всі пересилання починаються із процесу ініціалізації. Ініціалізація виконується в наступній послідовності (рис. 6.4):

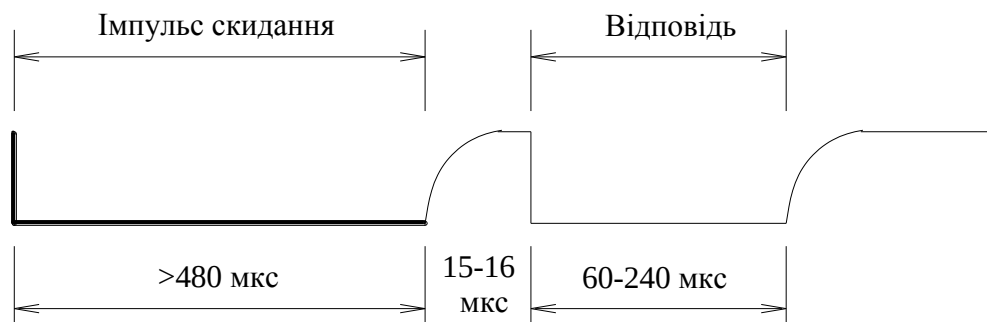


Рис. 6.4.

Майстер посилає імпульс скидання (reset pulse) – сигнал низького рівня тривалістю не менше 480 мкс. За імпульсом скидання відбувається відповідь підлеглого пристрою (presence pulse) – сигнал низького рівня тривалістю 60 – 240 мкс, що генерується через 15 – 60 мкс після завершення імпульсу скидання.

Відповідь підлеглого пристрою для головного пристрою означає, що на шині присутній термометр і він готовий до обміну даними. Після того, як майстер виявив відповідь, він може передати термометру одну з команд. Передача ведеться шляхом формування майстром спеціальних часових інтервалів (time slots). Кожен часовий інтервал служить для передачі

одного біта. Першим передається молодший біт. Інтервал починається імпульсом низького рівня, тривалість якого лежить у межах 1 – 15 мкс. Оскільки перехід з одиниці в нуль менш чутливий до ємності шини (він формується відкритим транзистором, у той час як перехід з нуля в одиницю формується резистором), саме цей перехід використовують однопровідні пристрої для синхронізації з майстром. У підлеглому пристрої запускається схема часової затримки, що визначає момент зчитування даних. Номінальне значення затримки дорівнює 30 мкс, однак воно може коливатися в межах 15 – 60 мкс. За імпульсом низького рівня видається черговий біт для передачі. Він повинен утримуватися майстром на шині протягом 60 – 120 мкс від початку інтервалу. Часовий інтервал завершується переводом шини в стан високого рівня на час не менший 1 мкс. Слід відзначити, що обмеження на цей час не накладається. Аналогічним чином формуються часові інтервали для всіх інших біт (рис. 6.5):

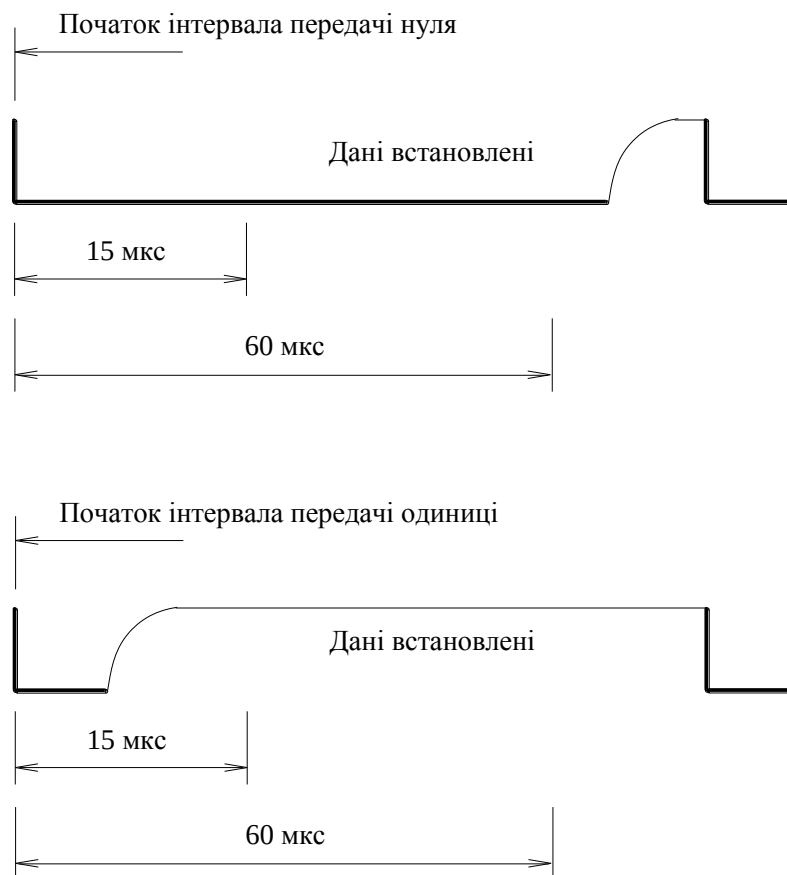


Рис. 6.5.

Першою командою, яку повинен передати майстер для DS18B20 після ініціалізації, є одна з команд функцій ПЗП. Усього DS18B20 має 5 команд функцій ПЗП:

- Read ROM [33h]. Ця команда дозволяє прочитати вміст ПЗП. У відповідь на цю команду DS18B20 передає 8-бітний код сімейства (10h), потім 48-бітний серійний номер, а потім 8-

бітне значення CRC для перевірки правильності прийнятої інформації.

- Match ROM [55h]. Ця команда дозволяє адресувати на шині конкретний термометр. Після цієї команди майстер повинен передати потрібний 64-бітний код, і тільки той термометр, що має такий код, буде «відгукуватися» до наступного імпульсу скидання.
- Skip ROM [CCh]. Ця команда дозволяє пропустити процедуру порівняння серійного номера, і тим самим заощадити час у системах, де на шині є всього один пристрій.
- Search ROM [F0h]. Ця досить складна у використанні команда дозволяє визначити серійні номери всіх термометрів, що присутні на шині.
- Alarm Search [ECh]. Ця команда аналогічна попередній, але «відгукуватися» будуть тільки ті термометри, у яких результат останнього виміру температури виходить за встановлені межі TH та TL.

Оскільки в нашому прикладі всього один пристрій, найбільш підходящою є функція Skip ROM. Крім неї ще може бути корисною функція Read ROM, що дозволяє ідентифікувати підключений на шину пристрій по його коду сімейства та серійному номеру.

При прийомі даних від підлеглого пристрою часові інтервали для прийнятих біт також формує майстер. Інтервал починається імпульсом низького рівня тривалістю 1 – 15 мкс. Потім майстер повинен звільнити шину, щоб дати можливість термометру вивести біт даних. По переходу з одиниці в нуль DS18B20 виводить на шину біт даних і запускає схему часової затримки, що визначає, як довго біт даних буде присутній на шині. Цей час лежить у межах 15 – 60 мкс. Для того щоб дані на шині, яка завжди має деяку ємність, гарантовано встановилися, потрібний деякий час. Тому момент зчитування даних майстром повинен відстояти якнайдалі, але не більше ніж на 15 мкс від початку часового інтервалу (рис. 6.6):

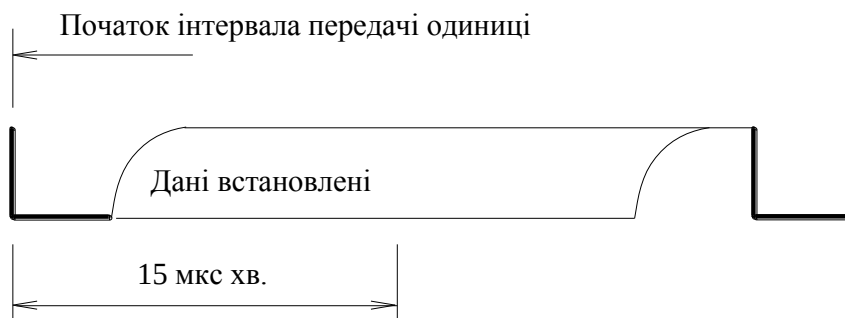


Рис. 6.6.

Прийом байта починається з молодшого біта. Спочатку передається байт коду сімейства. За кодом сімейства передається 6 байт серійного

номера, починаючи з молодшого. Потім передається байт контрольної суми (CRC). В обчисленні байта контрольної суми беруть участь перші 7 байт, або 56 переданих біт. Після прийому даних майстер повинен обчислити контрольну суму та порівняти значення з CRC. Якщо ці значення збігаються – прийом даних пройшов без помилок.

Після обробки однієї з команд функцій ПЗП, DS18B20 здатний сприймати ще кілька команд:

- Write Scratchpad [4Eh]. Ця команда дозволяє записати дані в проміжний ОЗП DS18B20.
- Read Scratchpad [BEh]. Ця команда дозволяє зчитувати дані із проміжного ОЗП.
- Copy Scratchpad [48h]. Ця команда копіює байти TH і TL із проміжного ОЗП в енергонезалежну пам'ять. Ця операція вимагає близько 10 мс.
- Convert T [44h]. Ця команда запускає процес перетворення температури.
- Recall E2 [B8h]. Ця команда діє зворотним чином відносно команди Copy Scratchpad, тобто вона дозволяє зчитувати байти TH і TL з енергонезалежної пам'яті в проміжний ОЗП. При включенні живлення ця команда виконується автоматично.
- Read Power Supply [B4h]. Ця команда дозволяє перевірити, чи використовує DS18B20 паразитне живлення. Справа в тому, що DS18B20 можна підключати всього за допомогою двох проводів. У цьому випадку для живлення використовується лінія даних.

При використанні DS18B20 тільки для виміру температури потрібні всього дві із цих команд: Convert T і Read Scratchpad. Послідовність дій при вимірі температури повинна бути наступною:

1. Посилаємо імпульс скидання і приймаємо відповідь термометра.
2. Посилаємо команду Skip ROM [CCh].
3. Посилаємо команду Convert T [44h].
4. Формуємо затримку мінімум 750 мс.
5. Посилаємо імпульс скидання і приймаємо відповідь термометра.
6. Посилаємо команду Skip ROM [CCh].
7. Посилаємо команду Read Scratchpad [BEh].
8. Читаємо дані із проміжного ОЗП (8 байт) і CRC.
9. Перевіряємо CRC, і якщо дані зчитані вірно, обчислюємо температуру.

6.4. Мікропроцесорна система контролю температури

На рис. 6.7 представлена схема підключення датчика температури DS18B20 до мікроконтролера AT90S8535. Мікроконтролер підключається до комп'ютера по інтерфейсу RS232 і передає значення температури. Для

узгодження рівнів напруг COM-порту комп'ютера і мікроконтролера використана мікросхема-перетворювач рівнів MAX232.

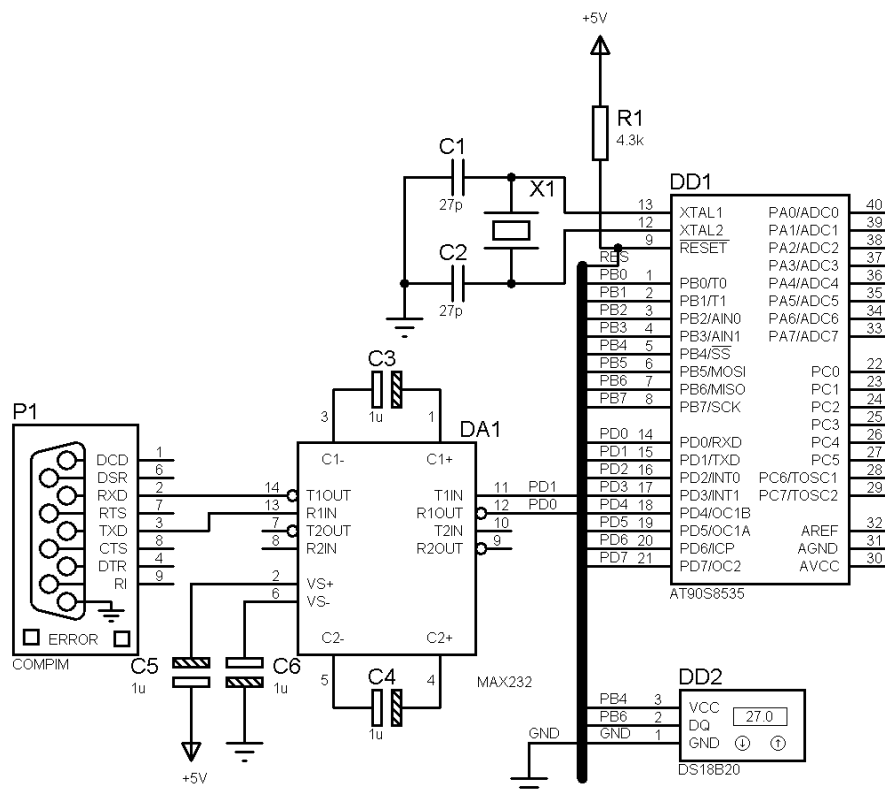


Рис. 6.7.

Приклад 6.1. Написати програму для мікроконтролера AT90S8535, що виконує зчитування температури з датчика DS18B21 по інтерфейсу 1-Wire і передає значення температури по COM-порту.

1. Запускаємо середовище розробки CodeVision і створюємо новий проект, вибравши з меню File команду New. У вікні, що з'явилося (рис. 6.8) відзначити позицію Project.

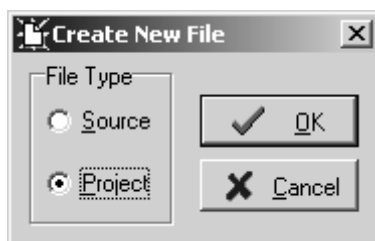


Рис. 6.8.

У вікні, що з'явилося, із пропозицією використання майстера проекту (CodeWizard), натиснути на кнопку «NO». Далі потрібно ввести назву проекту (наприклад TestWire).

2. У вікні налаштувань проекту встановити значення, як показано на рис. 6.9.

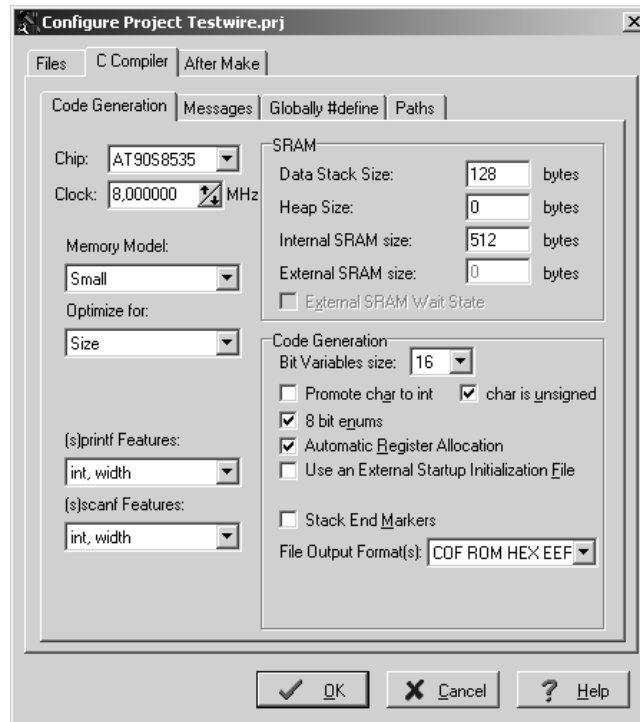


Рис. 6.9.

3. Додаємо в проект файл із вихідним кодом. Для цього в меню File вибрати команду New. Програма для мікроконтролера AT90S8535, створена в компіляторі CodeVision, представлена нижче.

```
#asm // вказівка виводів, до яких підключається датчик
.equ __w1_port=0x18
.equ __w1_bit=6 // лінія даних на виводі 6 порту В
#endasm
#include "1wire.h" // підключення заголовних файлів
#include "90s8535.h"
#include "delay.h"
#include "math.h"
unsigned char delay; // затримка опитування датчика
bit start_stop; // якщо біт =1, то почати перетворення
// оголошення прототипів функцій
void main(void); // основна функція програми
void init(void); // ініціалізація мікроконтролера
void temp(void); // опитування датчика температури
// змінює байт конфігурації датчика
void set_config(unsigned char config);
// підпрограма обробки переривання
// викликається по прийому символу з UART
interrupt [UART_RXC] void uart_rxc(void)
{
    unsigned char temp; // оголошення додаткової змінної
    temp = UDR; // зчитати прийнятий байт
    // якщо прийнято команду 0xBC, почати перетворення
    if(temp==0xBC) start_stop = 1; }
// функція опитування датчика температури
```

```
void temp()
{
    unsigned char buf[9];           // буфер зчитаних даних
    unsigned char i;                // додаткові змінні
    unsigned char k;
    if(w1_init())                   // виклик функції ініціалізації датчика
    {
        w1_write(0xCC);             // пропуск ПЗП
        w1_write(0x44);             // перетворення температури
        // очікування кінця перетворення температури
        delay_ms(delay);
        // цикл зчитування блоку пам'яті датчика
        for (k=0; k<3; k++)
        {
            w1_init();              // ініціалізація
            w1_write(0xCC);          // пропуск ПЗП
            w1_write(0xBE);          // зчитування блоку пам'яті
            for (i=0; i<9; i++)      // зчитати дев'ять байт
            {
                buf[i] = w1_read();  // запис зчитаного байта в буфер
            }
            // якщо помилка, то повторити цикл
            if(buf[8]==w1_dow_crc8(&buf[0], 8)) break;
        }
            // вийти, якщо датчик не відповідає
            if(buf[8]!=w1_dow_crc8(&buf[0], 8)) return;
            else                          // передача двох байт температури в комп'ютер
            {
                while(!USR.5);         // якщо біт UDRE == 1
                UDR = buf[0];          // передати байт
                while(!USR.5);
                UDR = buf[1];
            }
        }
    }
    start_stop=0;                   // скидання прапора дозволу перетворення
    return;
}

// функція установки байта конфігурації
void set_config(unsigned char config)
{
    while(!w1_init());              // виклик функції ініціалізації датчика
    // повторювати поки датчик не відповість

    switch(config)
    {
        case 0:                     // 9 біт
            config = 0b00011111;
            delay = 100;
            break;
        case 1:                     // 10 біт
            config = 0b00111111;
            delay = 188;
    }
}
```

```

        break;
    case 2:          // 11 біт
        config = 0b01011111;
        delay = 375;
        break;
    case 3:          // 12 біт
        config = 0b01111111;
        delay = 750;
        break;
    default:         // якщо помилка
        return;     // вихід з функції
}
// перепрошивання байта конфігурації датчика 9-12 розрядів
w1_write(0xCC);     // пропуск ПЗП
w1_write(0x4E);     // запис у блокнутому пам'яті

w1_write(0x00);     // запис TH
w1_write(0x00);     // запис TL
w1_write(config);   // запис байта конфігурації
}

// функція ініціалізації мікроконтролера
void init_cpu(void)
{
    DDRB = 0b00010100; // підключення DS18B20:6 - DQ, 4 - UCC
    PORTB = 0xFF;       // вивести одиницю на Port.4 - живлення
    // ініціалізація UART
    DDRD = 0b10000010; // налаштування виводів UART
    UCR = 0b10011000;  // дозвіл прийому/передачі
    UBRR = 51;          // швидкість обміну 9600 біт/с при 8 МГц
    // конфігурація за замовчуванням
    delay = 100;        // затримка
    start_stop = 0;     // прапор дозволу перетворення
    #asm("sei")         // дозволити переривання
}

void main(void)        // головна функція
{
    init_cpu();         // виклик функції ініціалізації мікроконтролера
    set_config(0);      // установка 9-бітного режиму перетворення T
    while(1)            // вічний цикл
    {
        if(start_stop!=0) temp(); // якщо біт дозволу
        // перетворення температури встановлений,
        // викликати функцію перетворення температури
    }
}

```

Зберігши файл на диск із назвою TestWire.c, проект можна компілювати. Для цього в меню Project вибрати команду Compile, або натиснути клавішу F9. У каталозі, де розташовані файли проекту, буде створений файл для програмування мікроконтролера TestWire.hex.

Питання до розділу 6

1. Які переваги використання інтерфейсу 1-Wire в мікропроцесорних системах?
2. Скільки провідників потрібно використовувати для з'єднання мікроконтролера з пристроєм 1-Wire?
3. Яка швидкість передачі даних по шині 1-Wire?
4. Опишіть апаратну конфігурацію інтерфейсної частини шини 1-Wire.
5. Які команди використовуються при організації обміну даними з датчиком температури DS18B20?
6. Яким чином відбувається ініціалізація обміну даними?
7. Скільки пристроїв з інтерфейсом 1-Wire можуть бути одночасно підключені до однієї лінії?
8. Яка інформація міститься в постійній пам'яті датчика температури DS18B20?
9. Яка топологія використовується при побудові мереж 1-Wire?
10. Як виконується підключення пристрою з інтерфейсом 1-Wire до мікроконтролера?

Задачі до розділу 6

1. Розробити мережу MicroLAN, яка складається з чотирьох датчиків температури DS18B20. Написати програму для мікроконтролера, задачею якого є зчитування значення температури з кожного датчика та вивід температури на алфавітно-цифровий РКІ.
2. Використовуючи мікросхему аналого-цифрового перетворювача DS2450 з інтерфейсом 1-Wire розробити чотирьохканальний вольтметр з виводом інформації на семисегментні індикатори. Номер каналу вибирається шляхом натискання тактової кнопки, що підключена до мікроконтролера. Документацію на мікросхему АЦП можна знайти в мережі Інтернет за адресою: <http://www.maxim-ic.com>.
3. Використовуючи таймер реального часу DS2415 з інтерфейсом 1-Wire, розробити принципову схему електронного годинника. Написати програму для мікроконтролера, що зчитує значення часу з мікросхеми і виводить його на екран алфавітно-цифрового РКІ. Документацію на мікросхему таймера реального часу можна знайти в мережі Інтернет за адресою: <http://www.maxim-ic.com>.
4. Розробити мікропроцесорну мережу, що складається з 2-х мікроконтролерів. Обмін даними між мікроконтролерами виконується з використанням інтерфейсу 1-Wire, який потрібно реалізувати програмно.
5. Розробити систему підтримання заданої температури в приміщенні. Якщо температура падає нижче заданого рівня, вмикається нагрівач, якщо перевищує заданий рівень, вмикається кондиціонер. Передбачити можливість налаштування нижньої та верхньої меж температури.

7. ШИНА USB

Шина USB (Universal Serial Bus) з'явилася на початку 1996 року. На той час персональні комп'ютери були оснащені великою кількістю різноманітних зовнішніх інтерфейсів, корисних і необхідних, але мали один недолік: всі вони потребували спеціального роз'єму, а також виділеного апаратного переривання.

Перша специфікація 1.0 була опублікована на початку 1996 року, тоді ж з'явилися перші пристрої що використовують даний протокол обміну даними. В 1998 році вийшла специфікація 1.1, що включала в себе ряд змін і доповнень. Швидкість передачі: Low speed – 1,5 Мбіт/с та Full speed – 12 Мбіт/с. Навесні 2000 року з'явилася специфікація 2.0, що відрізняється збільшенням максимальної пікової швидкості в 40 разів. Швидкість передачі High speed – 480 Мбіт/с.

7.1. Огляд методів реалізації передачі даних по USB

Існує 3 основних методи реалізації передачі даних по USB-протоколу. Перший метод полягає в реалізації програмного моста типу RS-232 → USB, з використанням для цього проміжного контролера. Проміжний контролер зберігає в собі програму перетворення пакетів RS-232 в USB. Необхідно лише подавати дані на UART проміжного контролера. Швидкість передачі транзакцій даного проміжного контролера відповідає мінімальній швидкості передачі по специфікації USB і становить 1.5 Мбіт/с. Однак загальна швидкість передачі даних у цьому випадку обмежена швидкістю протоколу RS-232. Даний метод зручний своєю простотою реалізації та низкою вартістю кінцевого продукту. Використання даного методу викликане тим, що на сьогоднішній день складно знайти персональний комп'ютер з вбудованим COM-портом, а різного роду датчики, що використовують RS-232, продовжують випускатися (тому що реалізація RS-232 у такому випадку більше виправдана, ніж USB). Існує ряд вже готових розробок моста RS-232 → USB, і так само розроблених до них драйверів і програм-терміналів. Дані розробки мають ATMEL, Microchip, Philips, FTDI та інші компанії.

Другий метод полягає в повністю програмній реалізації протоколу безпосередньо в самому контролері. У цьому випадку відпадає необхідність використання проміжного контролера. При такому методі ускладнюється розробка кінцевого програмного продукту, оскільки розробнику необхідно досконально вивчити протокол обміну даними USB. Також розробнику необхідно вирішити ряд питань, пов'язаних з реалізацією всього програмного циклу, оскільки в цьому випадку необхідно встигнути обробити вхідний сигнал за 8-10 тактів роботи самого контролера. За цей час необхідно синхронізуватися з комп'ютером, зчитати дані із проміжних буферів, обробити їх, очистити проміжні буфери та сформувати підтвердження прийнятих даних.

Програмна реалізація протоколу забирає основний час роботи та ресурси контролера, що обмежує виконання контролером інших операцій з обробки даних (зчитування даних з датчиків, їх обробка та підготовка до передачі по USB). Так само розробнику необхідно реалізувати драйвер для операційних систем і програму-термінал для обміну даними між комп'ютером та пристроєм. Швидкість передачі транзакцій так само становить 1,5 Мбіт/с, але швидкість передачі даних перебуває в межах 1,5 Мбіт/с, що приблизно в 10 разів швидше, ніж швидкість моста RS-232 → USB.

Третій метод є найбільш використовуваним і полягає в програмно-апаратній реалізації протоколу USB. У цьому випадку розробнику необхідно програмно описати протокол, що підготовлює дані, а передачу-прийом даних, синхронізацію, проміжні буфери, надати інтегрованим пристроям контролера. Швидкість передачі даних залежить від типу контролера і його властивостей. Найчастіше швидкість лежить у межах 1.5 Мбіт/с – 200 Мбіт/с. У деяких випадках швидкість обміну даними становить ~400 Мбіт/с.

У даному розділі буде розглянутий третій метод реалізації передачі даних по USB-протоколу з використанням мікроконтролера ATMEЛ типу AT90USB1286. Тип пристрою, що підключається – HID (Human Interface Device). Відпадає необхідність розробки драйвера для операційної системи, оскільки використовується стандартний драйвер Windows – hid.lib.

7.2. Структура USB на фізичному рівні

Фізичний інтерфейс USB складається із чотирьох провідників: 2 для живлення зовнішнього пристрою (VCC та GND) і 2 сигнальних провідника (DATA+ й DATA–). Через провідники живлення передається постійна напруга 5 В з навантажувальною здатністю максимум 500 мА. Мікроконтролер AVR живиться через виводи Vcc і GND. Сигнальні провідники називаються DATA+ й DATA– і управляють зв'язком між комп'ютером і пристроєм. Сигнали в цих провідниках є двонапрямленими. Рівні напруги – диференціальні: коли DATA+ має високий рівень, тоді DATA– перебуває на низькому рівні. Однак, є деякі випадки, коли DATA+ й DATA– мають однаковий рівень, наприклад, при команді EOP (кінець пакета). На рис. 7.1 представлена осцилограма сигналів при передачі даних низькошвидкісним драйвером.

У відповідності зі стандартом USB високий рівень на сигнальних провідниках повинен становити 3,0...3,6 В, при цьому, напруга живлення Vcc шини USB, що надходить від головного (комп'ютера) становить 4,4...5,25 В. Таким чином, якщо мікроконтролер живиться безпосередньо від шини USB, то лінії даних повинні пройти через каскад перетворення рівнів для компенсації рівнів диференціальної напруги. Іншим рішенням може бути використання стабілізатора напруги, що понизить напругу Vcc до рівня 3,3 В, при цьому мікроконтролер буде працювати із цією

зниженою напругою і, відповідно, генерувати знижені рівні напруг. В AT90USB1286 дані рівні реалізовані апаратно.

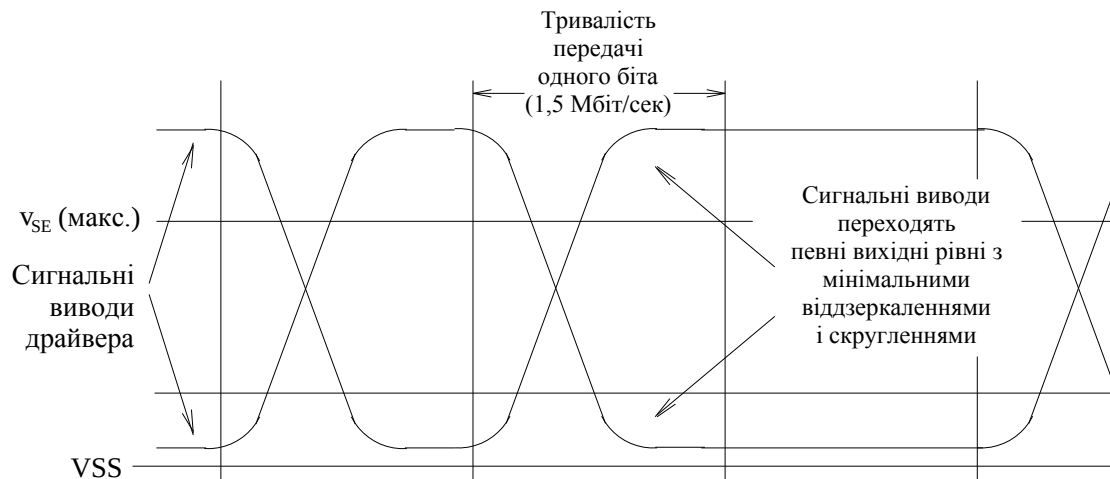


Рис. 7.1.

Дані по шині USB передаються пакетами, по декілька байт у кожному. На рис. 7.2 представлені осцилограми, що демонструють формування рівнів при передачі пакета даних. Розмір пакета визначається кожним пристроєм, але його граничний розмір обмежений. Для низькошвидкісних пристроїв максимальний розмір пакета дорівнює 8 байтам. Даний 8-байтний пакет разом з початковим і кінцевим полем повинні бути прийняті в буфер пристрою за одну USB-передачу.

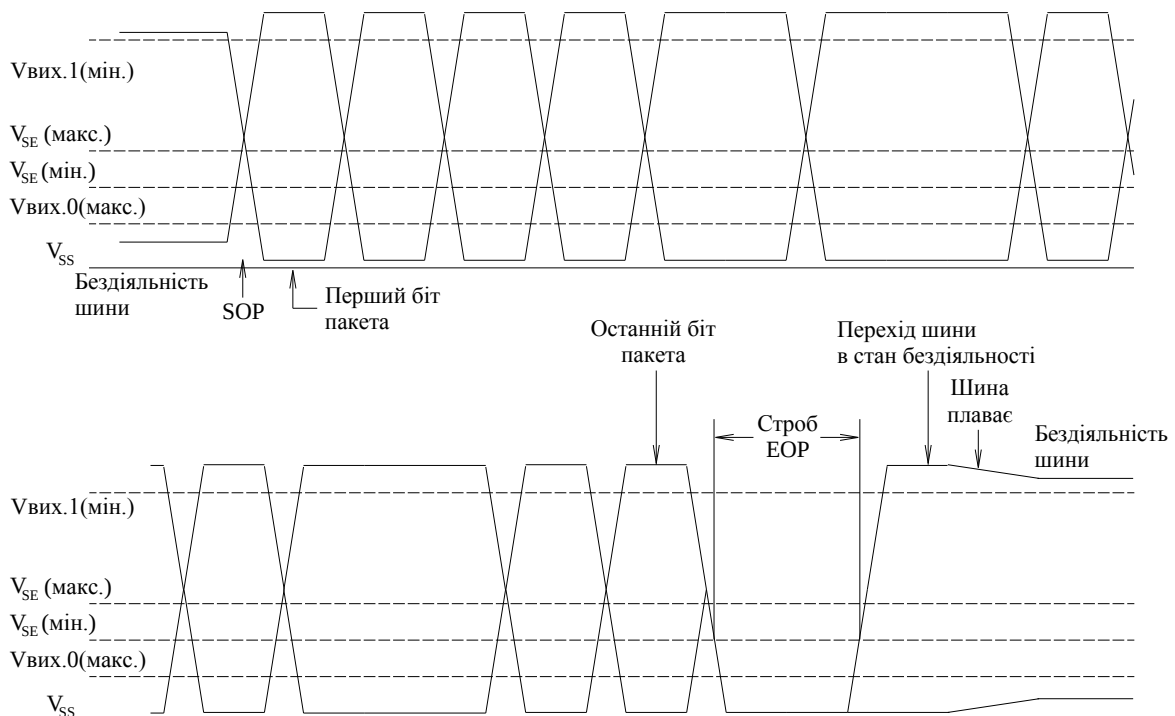


Рис. 7.2.

Після визначення нового пристрою головний починає зв'язок відповідно до фізичного протоколу USB. Протокол USB, на відміну від UART, заснований на синхронній передачі даних. Синхронізація передавача й приймача необхідна для здійснення зв'язку. Синхронізація виконується шляхом передачі невеликого заголовка "зразкова синхронізація", що передуює передачі даних (рис. 7.3). Даний заголовок представляє собою прямокутні імпульси (101010), за ними передаються два 0, а потім дані.

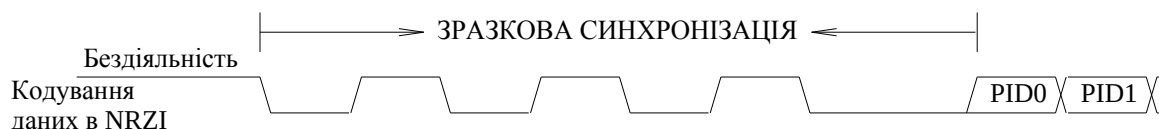


Рис. 7.3.

Зразкова синхронізація присутня в кожному пакеті. В апаратно-реалізованому USB-приймачі дана синхронізація гарантується цифровою системою ФАПЧ (фазове автопідстроювання частоти). У даній реалізації період перетворення даних повинен бути синхронізований зі зразковою синхронізацією, потім очікується два нулі, а потім починається процес прийому даних.

Прийом даних повинен задовольняти вимозі можливості засинхронізувати приймач і передавач у будь-який момент часу. Таким чином, не дозволяється передавати безперервний потік нулів або одиниць по лініях даних. Протокол USB гарантує синхронізацію за рахунок заповнення бітами. Це означає, що, після 6 безперервних одиниць або нулів на лініях даних, вставляється одна зміна (один біт). Сигнали по лініях USB передаються в коді NRZI (Non return to zero inverting – код без повернення до одиниці, інверсний). У коді NRZI кожен '0' представляється шляхом зсуву поточного рівня сигналу, а кожна '1' шляхом утримання поточного рівня (рис. 7.4). На рівні бітового заповнення це означає, що кожен нульовий біт вставляється в потік логічних даних після 6 безперервних логічних 1.

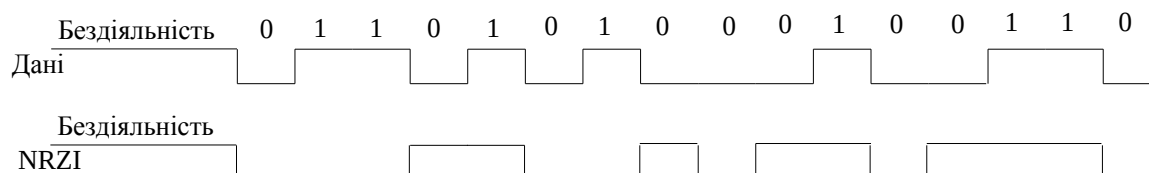


Рис. 7.4.

На рис. 7.5 представлена осцилограма, яка демонструє послідовність кодування даних.

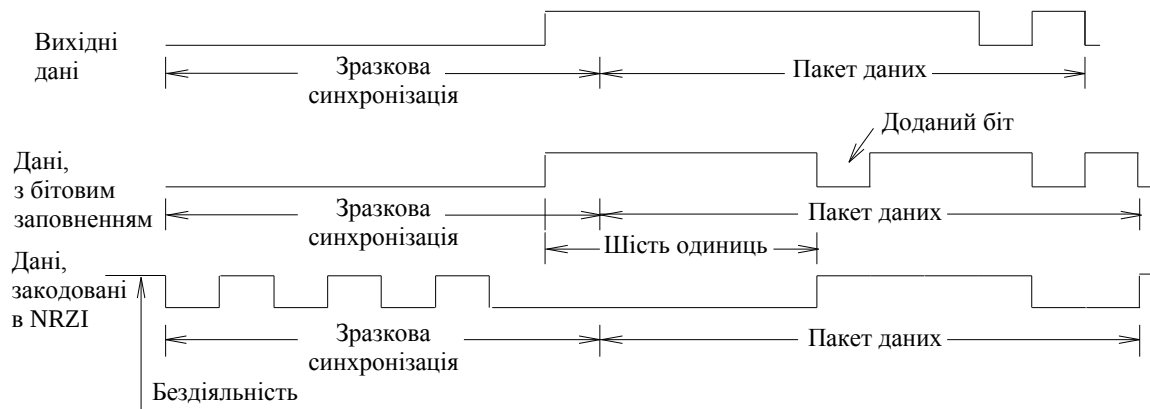


Рис. 7.5.

Повідомлення про закінчення передачі даних виконується за допомогою передачі сигналу "кінець пакета" (EOP). EOP передається шляхом установки низьких рівнів на обох лініях даних DATA+ й DATA-. EOP передається нетривалий час (мінімум два періоди швидкості даних). Це демонструє осцилограма на рис. 7.6. Після цього, виконується наступна транзакція.

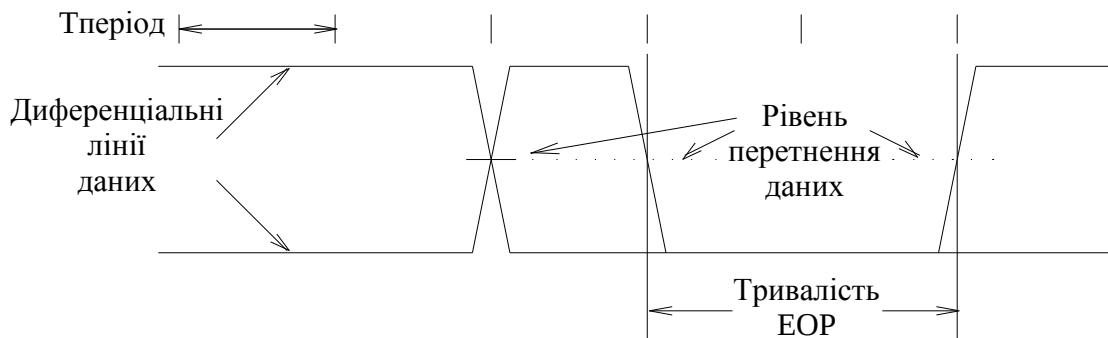


Рис. 7.6.

7.3. Протоколи та канали передачі даних

Будь-який обмін по шині USB ініціюється хостом. Контролер кожному 1 мс формує кадр, у який вкладаються всі необхідні передачі. Кожен кадр починається стартовим пакетом SOF (рис. 7.7). Потім ідуть заплановані передачі, і завершальний пакет EOF. У режимі високошвидкісної передачі кадр формується кожні 125 мкс. У кожному кадрі може бути виконано кілька транзакцій. Їх кількість залежить від швидкості передачі, довжини поля даних, затримок кабелю.

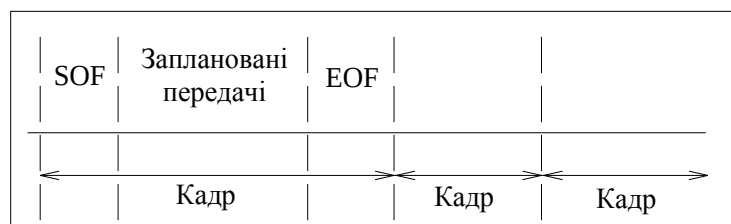


Рис. 7.7.

Частота формування кадрів може дещо варіюватися за допомогою спеціального регістра хоста, що дозволяє підлаштувати частоту для ізохронних передач. Структура кадрів показана на рис. 7.8.

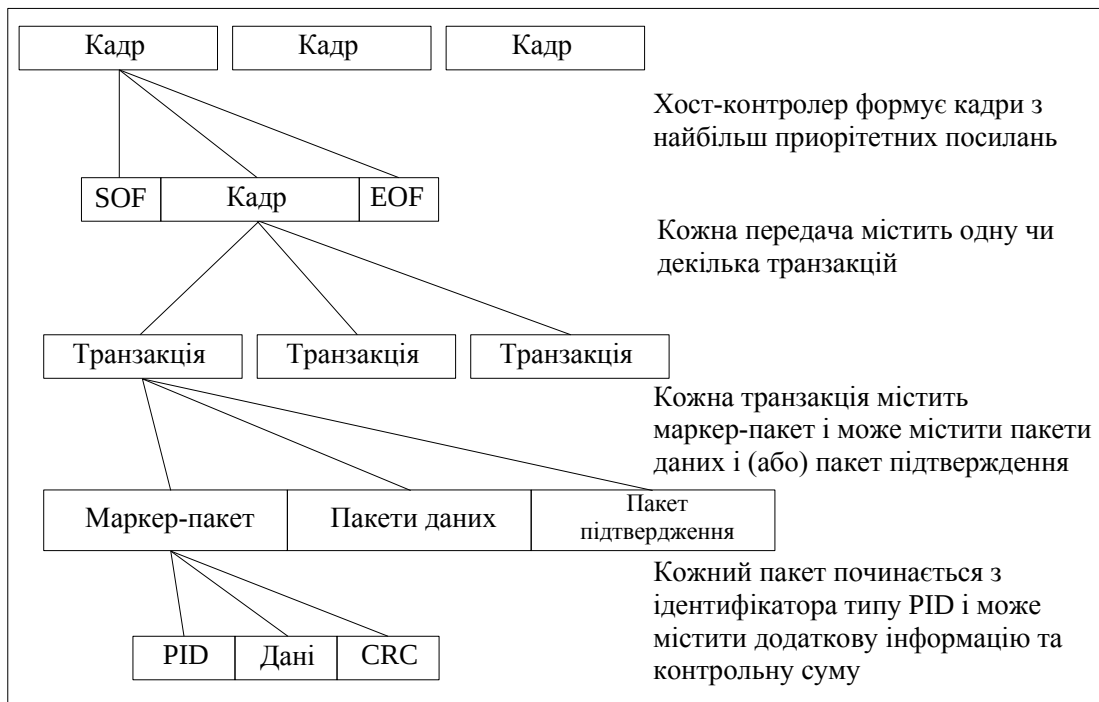


Рис. 7.8.

Кінцева точка (ENDPOINT) – це проміжний буфер, реалізований в USB-пристрої. Кінцева точка має свій ідентифікатор і представлена у вигляді області пам'яті або регістра контролера. В ENDPOINT відбувається завантаження-вивантаження даних шини USB. Кінцева точка має ряд параметрів:

- частота доступу до шини;
- припустима величина затримки;
- ширина смуги пропускання каналу;
- номер;
- спосіб обробки помилок;
- максимальний розмір пакета;
- тип передачі, що використовується;
- напрямок передачі.

Будь-який пристрій має дві нульові кінцеві точки, за допомогою яких відбувається обмін даними з пристроєм у момент визначення типу пристрою і його параметрів, і так само при ініціалізації та конфігуруванні пристрою. Крім нульових кінцевих точок USB – пристрій може мати ще 15 кінцевих точок. Обмін пакетами йде по нульовій кінцевій точці, потім USB – пристрій повинен сконфігурувати як мінімум 2 додаткові кінцеві точки. Одну для вхідних пакетів, іншу – для вихідних. Максимальне число кінцевих точок 16.

Нульові кінцеві точки доступні після того, як USB – пристрій підключений до шини, включений і одержав сигнал скидання шини. Всі інші точки після включення живлення або скидання перебувають у невизначеному стані.

Канал – це логічне з'єднання між кінцевою точкою USB – пристроєм та клієнтським програмним забезпеченням, а також між клієнтським програмним забезпеченням і хостом. На рис. 7.9 продемонстровано принцип використання каналів.

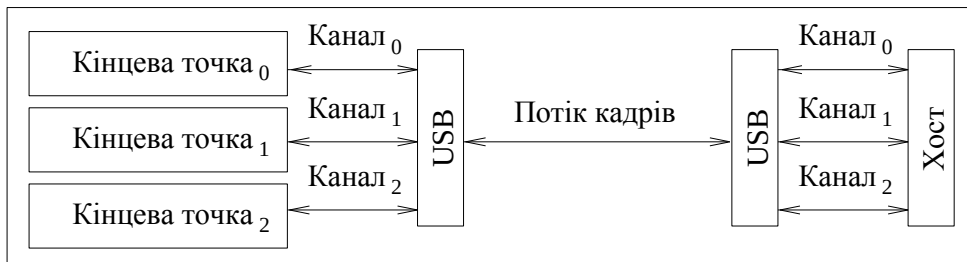


Рис. 7.9.

Існує два види каналів:

- потоковий канал;
- канал повідомлень.

Потоковий канал – це канал для передачі даних, структура яких визначається клієнтським програмним забезпеченням. Потік завжди однонапрямлений. Одна й та сама кінцева точка може використовуватися для обох типів пакетів (вхідних, вихідних). Використовуються особливі правила синхронізації (запити можуть виконуватися в різному порядку, запити одного каналу виконуються в порядку надходження, з появою помилки STALL потік зупиняється).

Канал повідомлень – це канал для передачі даних, структура яких визначається специфікацією USB. Потік передачі двонапрямлений. Для кожного типу передач (вхідних, вихідних) приділяється своя кінцева точка. Повідомлення строго синхронізовані. Забороняється одночасна обробка декількох запитів. У випадку виникнення помилки, передача повідомлення переривається хостом, після чого хост може почати передачу нового повідомлення.

Основними характеристиками такого каналу є:

- смуга пропускання каналу;
- тип передачі, що використовується каналом;
- характеристики кінцевої точки.

Смуга пропускання шини ділиться між всіма встановленими каналами. Виділена смуга закріплюється за каналом. Шина повинна забезпечувати обмін з такою швидкістю, щоб затримка даних у пристрої, що викликана буферизацією, не перевищувала декількох мілісекунд.

Так само існує канал обміну з нульовою кінцевою точкою, який називається основним каналом повідомлень. Він підтримує тільки керуючі передачі.

Набір клієнтських каналів, з якими працює драйвер USB – пристрою, називається інтерфейсом пристрою.

7.4. Пакети керуючих передач та передач по перериванню

Інформація з каналів передається у вигляді пакетів. Кожен пакет починається полем синхронізації, потім іде ідентифікатор пакета, дані (адреса, кінцева точка, біти даних), контрольна сума, закінчення пакета. Ідентифікатори пакета визначають призначення пакета, напрямок, додаткову інформацію. Ідентифікатори пакетів представлені в табл. 7.1.

Табл. 7.1.

Позначення	Код PID	Джерело	Опис
Ідентифікатори маркер-пакетів (Token Packet)			
OUT	0001b	Хост	Маркер транзакції виводу, передає адресу та номер кінцевої точки при передачі від хоста до функції
IN	1001b		Маркер транзакції виводу, передає адресу та номер кінцевої точки при передачі від функції до хосту
SOF	0101b		Маркер початку кадру, містить номер кадру
SETUP	1101b		Маркер транзакції керування: передає адресу та номер кінцевої точки при передачі команди від хоста до функції
Ідентифікатори пакетів даних (Data Packet)			
Data0	0011b	Хост, USB-пристрій	Пакети даних з парним та непарним PID, чергуються для точної ідентифікації підтверджень
Data1	1011b		
Data2	0111b		Додаткові типи пакетів даних, що використовуються в транзакціях з широкосмуговими ізохронними точками
MData	1111b		
Ідентифікатори пакетів підтвердження (Handshake)			
ACK	0010b	Хост, USB-пристрій	Підтвердження безпомилкового прийому пакета
NAK	1010b	USB-пристрій	Приймач не зміг передати дані. Може використовуватися для керування потоком даних. В транзакціях переривань є ознакою відсутності переривань, що не обслуговуються
STALL	1110b	USB-пристрій	Відбувся збій в кінцевій точці або запит не підтримується, потрібна участь хоста
NYET	0110b	USB-пристрій	Підтвердження безпомилкового приймання, але вказує на відсутність місця для приймання наступного пакету максимального розміру

Ідентифікатори спеціальних пакетів (Special Packet)			
PRE	1100b	Хост	Спеціальний маркер, що повідомляє, що наступний пакет буде передаватися в режимі LS (дозволяє трансляцію даних на низькошвидкісний порт хаба)
ERR		USB-пристрій, хаб	Сигналізація помилки з розщепленої транзакції
SPLIT (SS/CS)	1000b	Хост	Маркер розщепленої транзакції. В залежності від призначення позначається як SS (маркер запуску) та CS (маркер завершення), призначення визначається бітом SC в тілі маркера
PING	0100b	Хост	Пробний маркер високошвидкісного керування потоком
RESERV	0000b		Зарезервований PID

SETUP – є конфігураційним пакетом, який складається з 8 байт (рис. 7.10). Структурне представлення конфігураційного пакета:

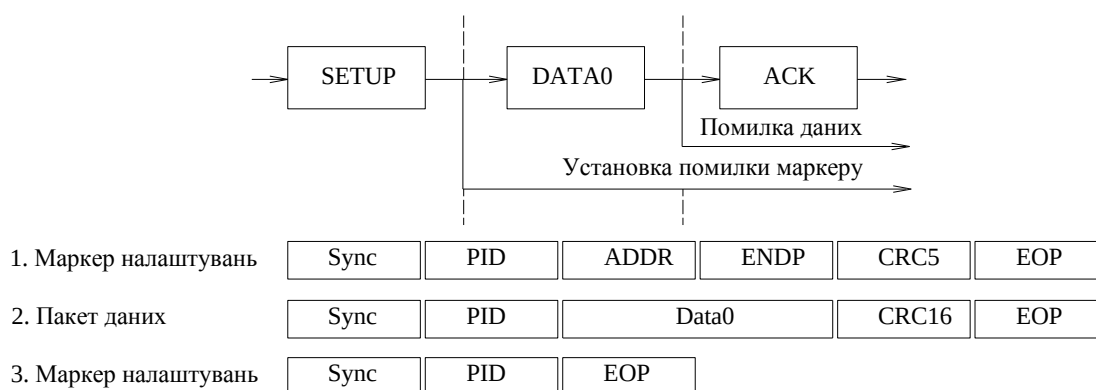


Рис. 7.10.

Як було відзначено вище, всі передачі ініціює хост. Перша передача Setup Token посилається хостом. У ній вказується ідентифікатор 11011101 (ідентифікатор є байтом, що двічі складається з комбінації PID-коду), адреса пристрою, кінцева точка, CRC5, завершення передачі EOF.

Пакет Data0 також посилає хост і несе в собі інформацію про вид запиту, напрямок даних, тип дескриптора і т. д.

Пакет Ack Handshake – передача, що підтверджує прийом SETUP – пакета пристроєм. Формується і передається пристроєм хосту.

IN – пакет забезпечує обмін даними від пристрою до хосту (рис. 7.11). Структурне представлення вхідного пакета:

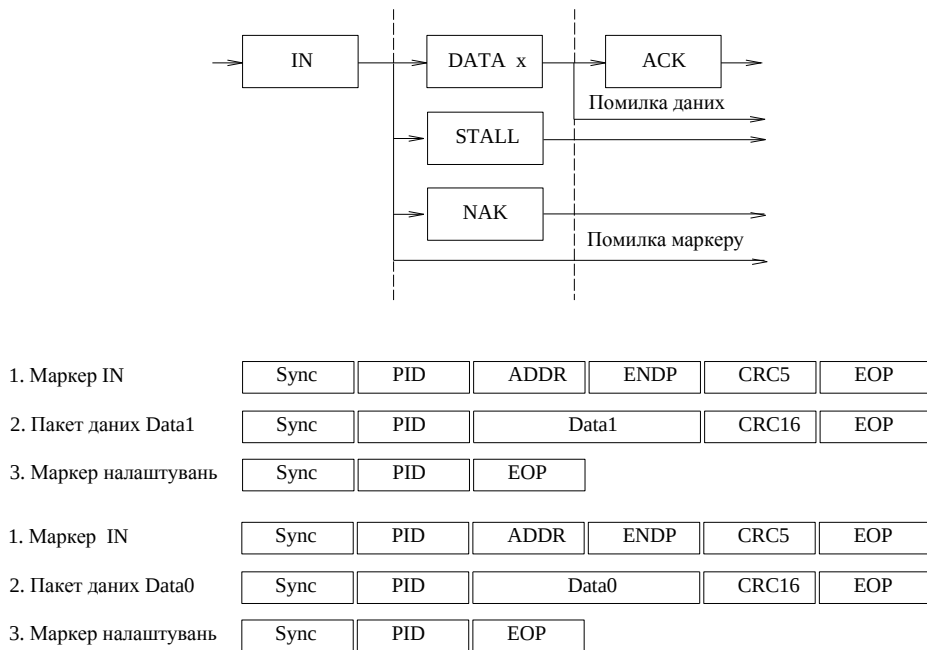


Рис. 7.11.

Однак необхідно чітко розрізняти напрямки передачі пакета й напрямки даних пакета.

Пакет Data1 передається пристроєм хосту і містить перші 8 байт дескриптора пристрою.

Ask Handshake – передача, що підтверджує прийом IN – пакета хостом. Потім іде аналогічна операція, у ході якої передаються байти, що залишилися, дескриптора пристрою (Data0). Якщо їх кількість менше 8, то байти, що залишилися, заповнюються нулями.

OUT – пакет забезпечує передачу даних від хоста до пристрою (рис. 7.12). Структурне представлення вихідного пакета:

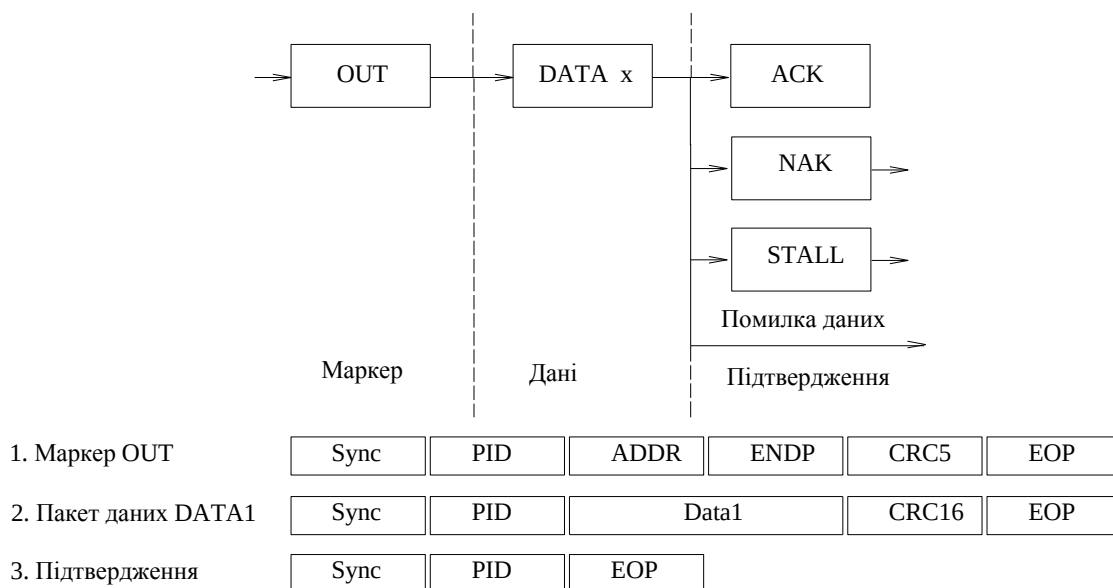


Рис. 7.12.

Починається даний пакет аналогічною передачею Out Token по напрямку від хоста до пристрою. Потім іде передача даних від хоста до пристрою Data1 Packet. При ініціалізації й конфігурації пристрою способом нульових кінцевих точок, передача даних від хоста до пристрою відсутня. Передається так званий нульовий рядок даних. Так само нульовий рядок даних може передаватися і в IN – пакеті.

7.5. Використання мікроконтролера AT90USB1286 для передачі даних

Для роботи USB контролера потрібна опорна частота $48 \text{ МГц} \pm 0,25\%$, що є виходом внутрішнього ФАПЧ. ФАПЧ генерує високу внутрішню частоту (48 МГц) для USB інтерфейсу, використовуючи зовнішню низьку частоту (генератор на кристалі або зовнішній генератор, підключений до входу XTAL1; тільки це джерело тактових сигналів відповідає вимогам USB по точності частоти й забезпечує нормальну роботу USB-контролера). На рис. 7.13 представлена структурна схема USB-контролера.

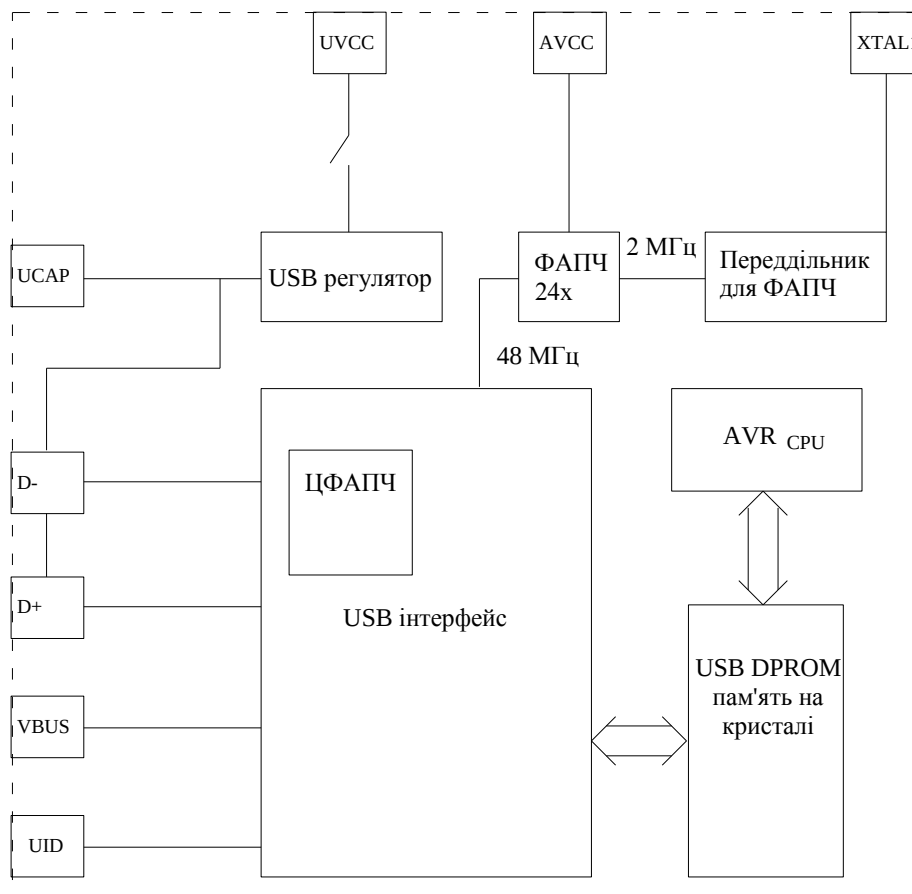


Рис. 7.13.

Тактовий сигнал частотою 48 МГц використовується для виділення 12 МГц (для Full Speed) тактового сигналу синхронізації бітів з отриманих диференціальних даних (або 1,5 МГц для Low Speed), і для передачі даних відповідно до обраної швидкості роботи USB-пристрою. Відновлення

тактового сигналу відбувається за допомогою цифрової системи ФАПЧ. Відповідно до електричної специфікації USB, виводи (D+ або D-) повинні бути підключені до напруги живлення в діапазоні від 3,0 до 3,6 В.

Даний контролер має цілий ряд робочих напруг і, відповідно, ряд режимів підключення. Для кожного рівня напруг обмежена частота роботи мікроконтролера. Режим підключення ліній живлення представлений на рис. 7.14.

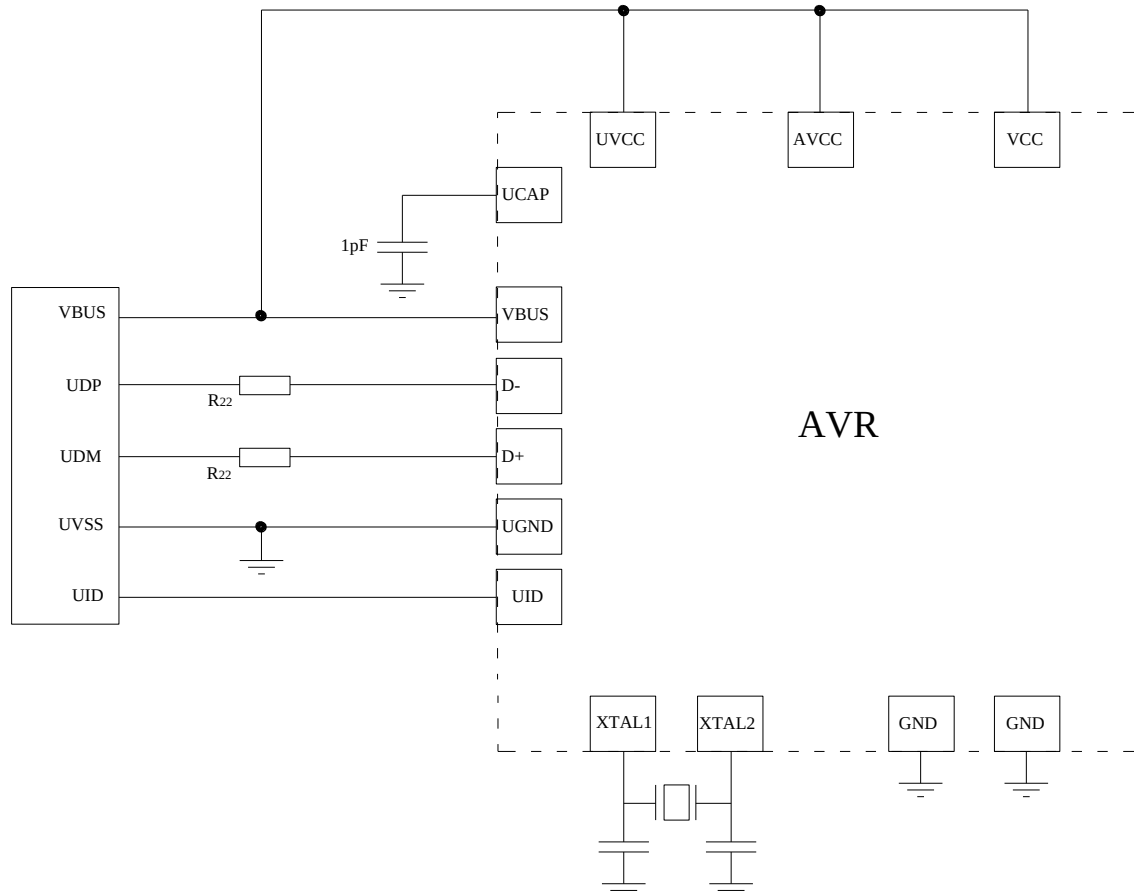


Рис. 7.14.

7.5.1. Структура мікроконтролера AT90USB1286

AT90USB1286/1287 має досить складну систему переривань. Для переривань USB-інтерфейсу реалізовано два вектори, що показані на рис. 7.15.

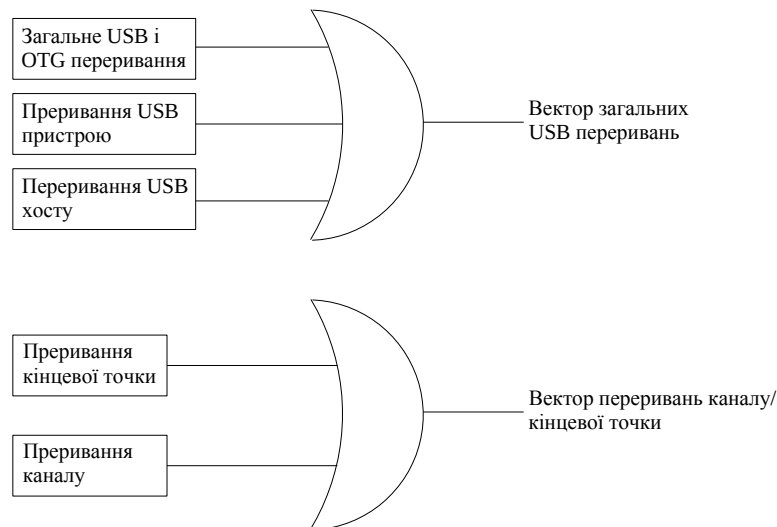


Рис. 7.15.

Існує два види переривань: функціональні (їх виникнення – це частина нормального функціонування) і виключення (помилки).

Функціональні переривання генеруються при наступних подіях:

- зміна стану виводу ID (підключення, відключення) (IDTI);
- підключення на лінії VBUS (підключення, відключення) (VBUSTI);
- визначення SRP (SRPI);
- зміна ролі (ROLEEXI).

Переривання-виключення виникають при наступних подіях:

- виявлено спадання напруги на Vbus (VBERRI);
- помилка при В-з'єднанні (BCERRI);
- HNP помилка (HNPERRI);
- помилка по тайм-ауту в режимі призупинки (suspend).

У режимі енергозбереження ядро процесора зупинено (зупинений тактуючий його сигнал). У режим idle можна увійти при працюючому USB контролері й при виключеному. Мікроконтролер "просипається" при будь-якому USB перериванні.

Коли USB інтерфейс налаштований для роботи в режимі пристрою, вибір швидкості (Full Speed або Low Speed) залежить від підтяжки до живлення на лініях UDP/UDM (рис. 7.16). Біт LSM у регістрі UDCON дозволяє вибрати внутрішню підтяжку на лініях даних UDM (режим Low Speed) або UDP (режим Full Speed).

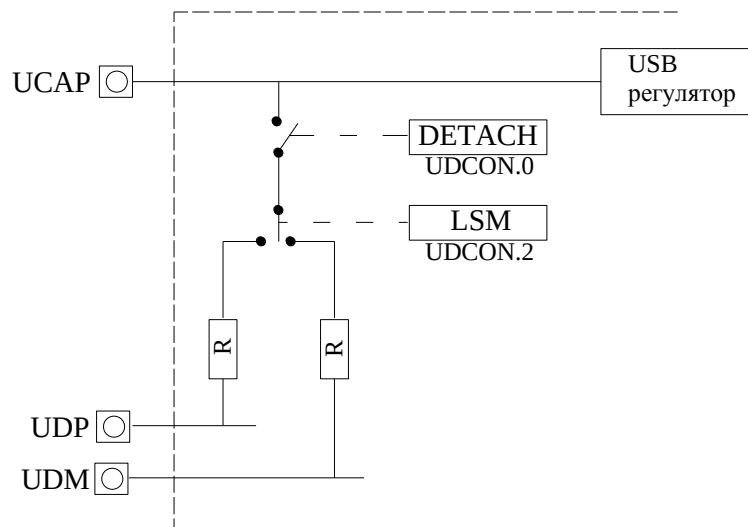


Рис. 7.16.

Коли USB інтерфейс налаштований для роботи в режимі хоста, активовані внутрішні резистори, що підтягують до землі лінії UDP та UDM (рис. 7.16), інтерфейс визначає швидкість підключеного до нього пристрою.

Контролер підтримує тільки наступний спосіб організації пам'яті (рис. 7.17). Розподіл пам'яті між кінцевими точками або каналами може здійснюватися тільки в порядку зростання (тобто починаючи з каналу/кінцевої точки 0, закінчуючи останнім каналом/кінцевою точкою). Тому програмне забезпечення повинне конфігурувати їх у такому ж порядку.



Рис. 7.17.

Залежно від режиму роботи USB програмне забезпечення повинне виконувати деякі з функцій:

1. Включення живлення USB інтерфейсу:
 - включення регулятора живлення виводів USB;

- конфігурація інтерфейсу ФАПЧ;
 - дозвіл ФАПЧ та очікування синхронізації ФАПЧ;
 - дозвіл USB інтерфейсу;
 - конфігурація USB інтерфейсу (швидкість, конфігурація кінцевих точок);
 - очікування з'єднання на лінії VBUS;
 - підключення USB пристроїв.
2. Вимикання живлення USB інтерфейсу:
- від'єднання USB інтерфейсу;
 - відключення USB інтерфейсу;
 - відключення ФАПЧ;
 - відключення регулятора напруги виводів USB.
3. Припинення USB інтерфейсу:
- скидання біта призупинення (suspend);
 - зупинка тактування USB;
 - відключення ФАПЧ;
 - контроль за наявністю дозволених переривань для виходу з режиму призупинення;
 - введення процесора в sleep режим.
4. Поновлення (resume) роботи USB інтерфейсу:
- включення ФАПЧ;
 - синхронізація ФАПЧ;
 - запуск тактування USB;
 - очищення інформації про поновлення.

На рис. 7.18 представлена структурна схема мікроконтролера.

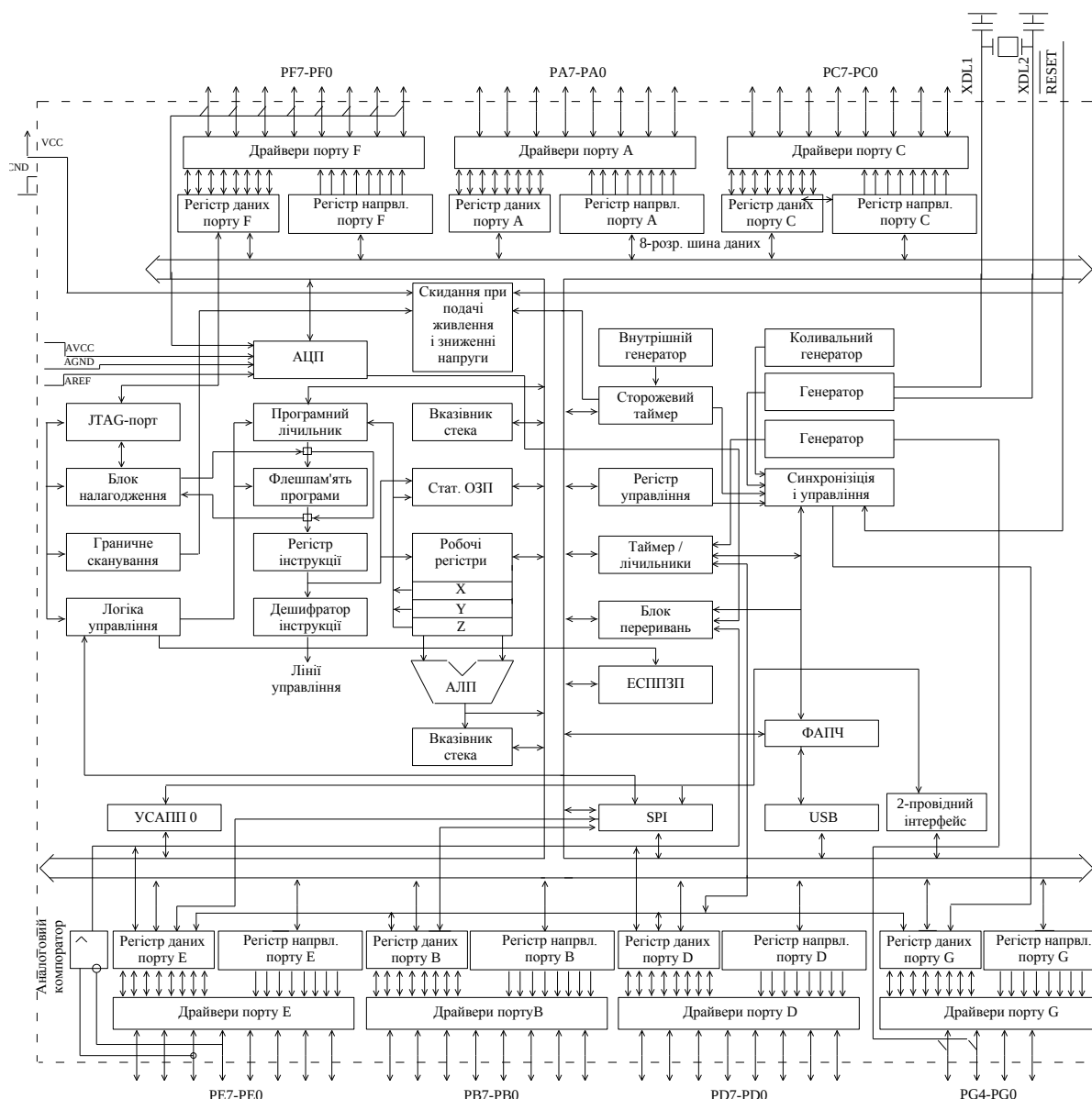


Рис. 7.18.

Опис регістрів та їх бітів наведено в документації на мікроконтролер.

7.5.2. Принципова схема пристрою

Принципова схема була розроблена в програмному пакеті Proteus 7.4 SP3, з урахуванням електромагнітної сумісності й наявності різних захистів і фільтрів. Схема пристрою (рис. 7.19) та програмне забезпечення розроблені випускником кафедри ТЕЕС НУК, магістром електроніки Пустоводовим С.А.

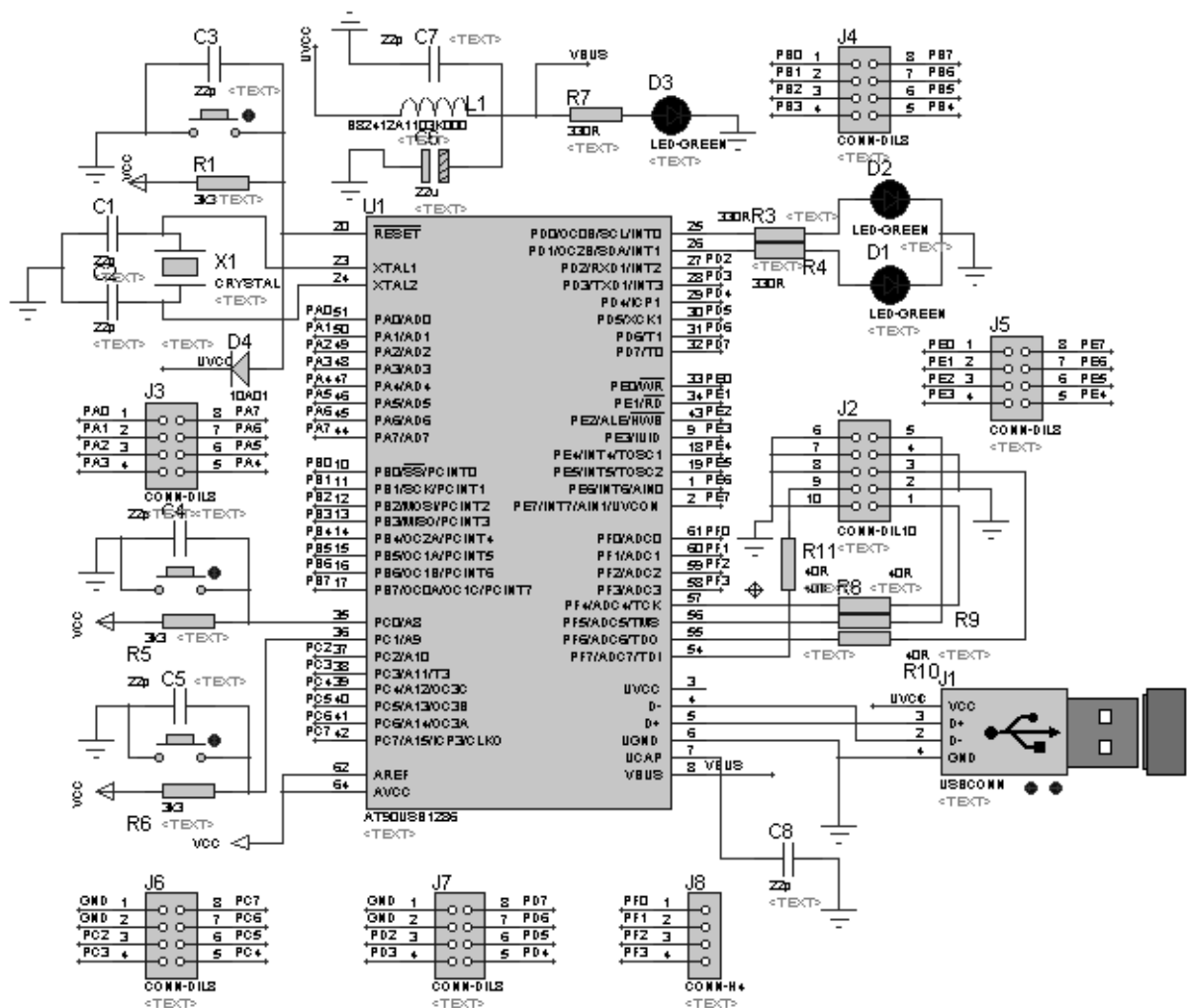


Рис. 7.19.

У наведеному прикладі реалізоване керування 2 світлодіодами (посилка даних) і зчитування стану 2 ключів (прийом даних) по протоколу USB. Використовується мікроконтролер фірми ATMEL AT90USB1286, що працює від зовнішнього кварцового резонатора 8 МГц.

Конденсатори C6 і C7 представляють фільтр, що обмежує пульсації живлення і високочастотні складові сигналу в момент комутації. Дросель L1, підключений послідовно, обмежує пульсації по струму (відповідно до специфікації AVR по електромагнітному захисту від перешкод). Передбачено скидання контролера натисканням кнопки SV3. Необхідно підключати діод VD1 до виводу RESET мікроконтролера (відповідно до специфікації AVR по електромагнітному захисту від перешкод).

До лінії D+ підключається резистор номіналом 15 кОм, для установки FS-швидкості передачі транзакцій при ініціалізації (інтегровано в мікроконтролер).

7.6. Програмна реалізація протоколу

При реалізації USB-протоколу використовуються чотири модулі, що підключаються до головного файлу. Кожен модуль складається з ряду бібліотек, відповідальних за формування транзакцій, прийом-передачу даних, вивантаження пакетів із проміжних буферів і т.д. На рис. 7.20 наведена структурна схема модулів і шляхи передачі даних між ними.

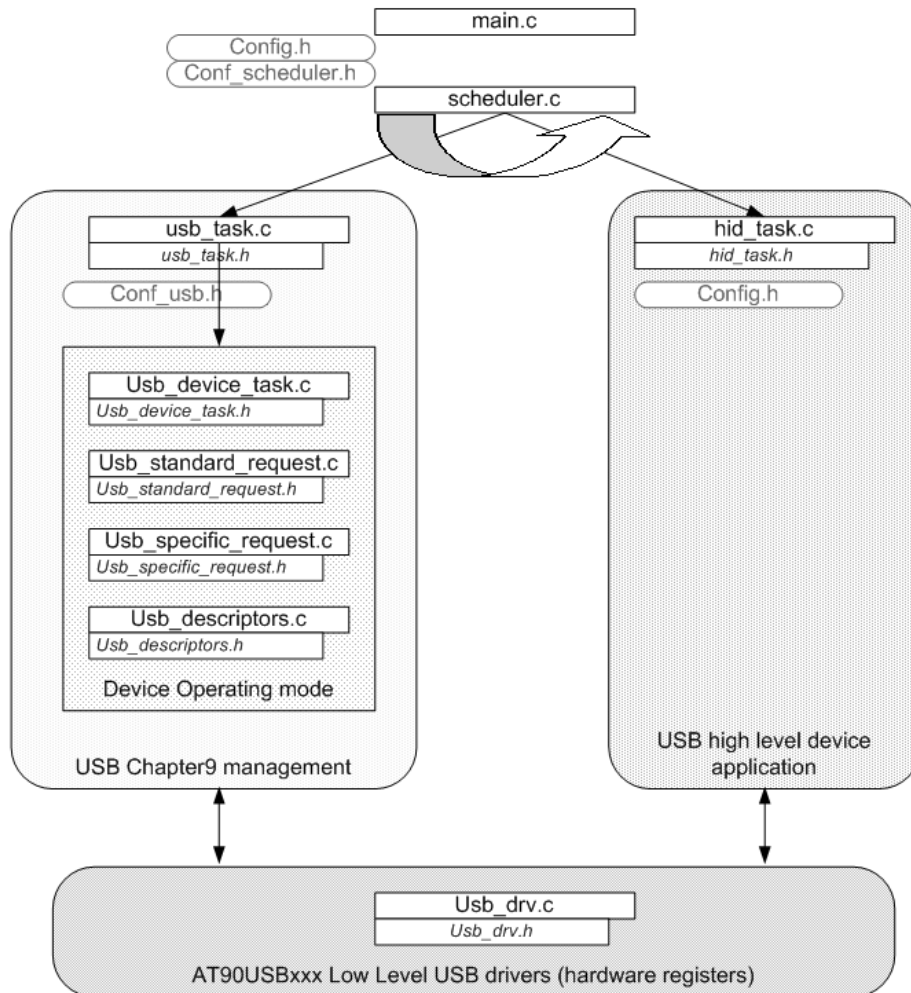


Рис. 7.20.

Перший модуль – модуль низькорівневої ініціалізації, що включає в себе драйвер проміжних буферів (кінцевих точок), драйвер ініціалізації регістрів контролера, драйвер HOST'a (реалізованого на контролері), драйвер регулятора, драйвер керуючих регістрів, драйвер конфігурації макросів для кінцевих точок, драйвер каналів передачі. Даний модуль прямо пов'язаний з регістрами і пристроями контролера.

Другий модуль складається із двох підмодулів. Перший підмодуль відповідає за реалізацію структур транзакцій, формування стандартних і спеціалізованих запитів, формування дескрипторів. Другий підмодуль відповідає за формування загальних подій і кадрів передачі залежно від типу передачі даних, формуючи тим самим необхідні дескриптори, реалізовані в першому підмодулі.

Третій модуль представляє ряд подій для формування HID-запитів. Цей модуль підключається до попереднього, і є модулем високого рівня, що може бути змінений розробником залежно від типу передачі. У ньому полягають оброблені дані, необхідні для передачі по USB-протоколу.

Другий і третій модуль так само пов'язані з першим модулем низькорівневої ініціалізації.

Четвертий модуль є планувальником між другим і третім модулем, відповідальний за трасування даних і підготовку їх для головної функції.

7.6.1. Модуль низькорівневої ініціалізації

Як було відзначено вище, перший модуль містить у собі ряд драйверів. Структура модуля представлена на рис. 7.21.

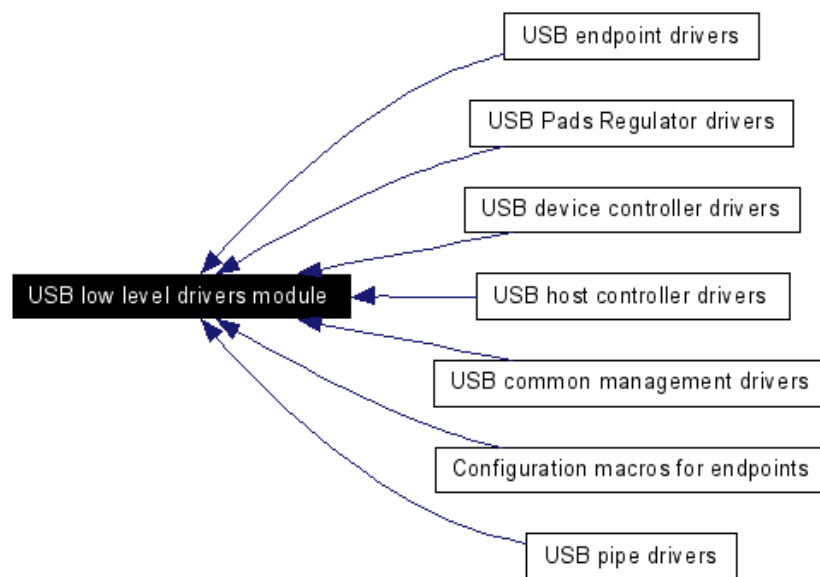


Рис. 7.21.

Перший модуль представлений бібліотекою *usb_drv*. Ця бібліотека зберігає в собі необхідні для подальшого конфігурування макроси:

– макроси кількості кінцевих точок:

```

#define MAX_EP_NB 7
#define EP_CONTROL 0
#define EP_1 1
#define EP_2 2
#define EP_3 3
#define EP_4 4
#define EP_5 5
#define EP_6 6
#define EP_7 7
  
```

– макроси кількості каналів передачі:

```

#define PIPE_CONTROL 0
#define PIPE_0 0
#define PIPE_1 1
#define PIPE_2 2
  
```

```
#define PIPE_3      3
#define PIPE_4      4
#define PIPE_5      5
#define PIPE_6      6
#define PIPE_7      7
```

– макроси кінцевих точок:

```
#define MSK_EP_DIR      0x7F
#define MSK_UADD        0x7F
#define MSK_EPTYPE      0x0
#define MSK_EPSIZE      0x70
#define MSK_EPBK        0x0C
#define MSK_DTSEQ       0x0C
#define MSK_NBUSYBK     0x03
#define MSK_CURRBK      0x03
#define MSK_DAT         0xFF // UEDATX
#define MSK_BYCTH       0x07 // UEBCHX
#define MSK_BYCTL       0xFF // UEBCLX
#define MSK_EPINT       0x7F // UEINT
#define MSK_HADDR       0xFF // UHADDR
```

– макроси каналів передач:

```
#define MSK_PNUM        0x07 // UPNUM
#define MSK_PRST        0x7F // UPRST
#define MSK_PTYPE       0x0   // UPCFG0X
#define MSK_PTOKEN      0x30
#define MSK_PEPNUM      0x0F
#define MSK_PSIZE       0x70 // UPCFG1X
#define MSK_PBK         0x0C
#define MSK_NBUSYBK     0x03
#define MSK_ERROR       0x1F
#define MSK_PTYPE       0x0   // UPCFG0X
#define MSK_PTOKEN      0x30
#define MSK_TOKEN_SETUP 0x30
#define MSK_TOKEN_IN    0x10
#define MSK_TOKEN_OUT   0x20
#define MSK_PEPNUM      0x0F
#define MSK_PSIZE       0x70 // UPCFG1X
#define MSK_PBK         0x0C
```

– макроси типів передач:

```
#define TYPE_CONTROL    0
#define TYPE_ISOCHRONOUS 1
#define TYPE_BULK       2
#define TYPE_INTERRUPT  3
```

– макроси максимального розміру:

```
#define SIZE_8          0
#define SIZE_16         1
#define SIZE_32         2
#define SIZE_64         3
#define SIZE_128        4
#define SIZE_256        5
#define SIZE_512        6
#define SIZE_1024       7
```

```

    – макроси напрямку передач:
#define DIRECTION_OUT 0
#define DIRECTION_IN 1
    – макроси кількості банків даних:
#define ONE_BANK 0
#define TWO_BANKS 1
    – макроси типів пакетів передач:
#define TOKEN_SETUP 0
#define TOKEN_IN 1
#define TOKEN_OUT 2
    – макроси станів
#define NYET_ENABLED 0
#define NYET_DISABLED 1

```

Наведені макроси є загальними для модулів протоколу. Далі йдуть макроси, необхідні для конфігурації кінцевих точок і каналів:

```

#define Usb_build_ep_config0(type, dir, nyet)((type<<6)|(nyet<<1)|(dir))
#define Usb_build_ep_config1(size, bank)((size<<4)|(bank<<2))
#define usb_configure_endpoint(num, type, dir, size, bank, nyet)
(Usb_select_endpoint(num), usb_config_ep(Usb_build_ep_config0(type, dir, nyet),
Usb_build_ep_config1(size, bank) ))
#define Host_build_pipe_config0(type, token, ep_num)((type<<6)|(token<<4)|(ep_num))
#define Host_build_pipe_config1(size, bank)((size<<4)|(bank<<2))
#define host_configure_pipe(num, type, token, ep_num, size, bank, freq)
( Host_select_pipe(num), Host_set_interrupt_frequency(freq),
host_config_pipe(Host_build_pipe_config0(type, token, ep_num),
Host_build_pipe_config1(size, bank)))

```

Повний перелік даних макросів наведений безпосередньо у файлі `usb_drv.h`. Так само в бібліотеці розміщуються оголошення функцій, реалізованих в `usb_drv.c`:

```

U8  usb_config_ep      (U8, U8);
U8  usb_select_endpoint_interrupt (void);
U16 usb_get_nb_byte_epw (void);
U8  usb_send_packet    (U8 , U8*, U8);
U8  usb_read_packet    (U8 , U8*, U8);
void usb_halt_endpoint (U8);
void usb_reset_endpoint (U8);
U8  usb_init_device    (void);
U8  host_config_pipe   (U8, U8);
U8  host_determine_pipe_size (U16);
void host_disable_all_pipe (void);

```

Функції, реалізовані в даній бібліотеці:

```

// функція, що конфігурує кінцеву точку заданого типу
U8 usb_config_ep(U8 config0, U8 config1)
{

```

```
    Usb_enable_endpoint();
    UECFG0X = config0;
    UECFG1X = (UECFG1X & (1<<ALLOC)) | config1;
    Usb_allocate_memory();
    return (Is_endpoint_configured());
}

// функція, що визначає стани кінцевих точок
U8 usb_select_endpoint_interrupt(void)
{
    U8 interrupt_flags;
    U8 ep_num;

    ep_num = 0;
    interrupt_flags = Usb_interrupt_flags();

    while(ep_num < 9)
    {
        if (interrupt_flags & 1)
        {
            return (ep_num);
        }
        else
        {
            ep_num++;
            interrupt_flags = interrupt_flags >> 1;
        }
    }
    return 0;
}

// функція, що перенаправляє дані в кінцеву точку для
// відправлення по USB
U8 usb_send_packet(U8 ep_num, U8* tbuf, U8 data_length)
{
    U8 remaining_length;

    remaining_length = data_length;
    Usb_select_endpoint(ep_num);
    while(Is_usb_write_enabled() && (0 != remaining_length))
    {
        Usb_write_byte(*tbuf);
        remaining_length--;
        tbuf++;
    }
    return remaining_length;
}

// функція, що перенаправляє дані з кінцевої точки
// в область пам'яті rbuf
U8 usb_read_packet(U8 ep_num, U8* rbuf, U8 data_length)
{

```

```

U8 remaining_length;

remaining_length = data_length;
Usb_select_endpoint(ep_num);

while(Is_usb_read_enabled() && (0 != remaining_length))
{
    *rbuf = Usb_read_byte();
    remaining_length--;
    rbuf++;
}
return remaining_length;
}

// функція, що відправляє STALL-підтвердження
void usb_halt_endpoint (U8 ep_num)
{
    Usb_select_endpoint(ep_num);
    Usb_enable_stall_handshake();
}

// функція, що ініціює пристрій і конфігурує
// нульову кінцеву точку
U8 usb_init_device (void)
{
    Usb_select_device();
    if(Is_usb_id_device())
    {
        Usb_select_endpoint(EP_CONTROL);
        if(!Is_usb_endpoint_enabled())
        {
            return usb_configure_endpoint(EP_CONTROL, \
                TYPE_CONTROL, \
                DIRECTION_OUT, \
                SIZE_64, \
                ONE_BANK, \
                NYET_DISABLED);
        }
    }
    return FALSE;
}

// функція, яка конфігурує канал передачі заданого типу
U8 host_config_pipe(U8 config0, U8 config1)
{
    Host_enable_pipe();
    UPCFG0X = config0;
    UPCFG1X = config1;
    Host_allocate_memory();
    return (Is_pipe_configured());
}

```

```
// функція, що повертає розмір каналу,  
// отриманий при нумерації пристрою  
U8 host_determine_pipe_size(U16 size)  
{  
    if(size <= 8 ) {return (SIZE_8 );}  
    else if(size <= 16 ) {return (SIZE_16 );}  
    else if(size <= 32 ) {return (SIZE_32 );}  
    else if(size <= 64 ) {return (SIZE_64 );}  
    else if(size <= 128 ) {return (SIZE_128 );}  
    else if(size <= 256 ) {return (SIZE_256 );}  
    else if(size <= 512 ) {return (SIZE_512 );}  
    else {return (SIZE_1024);}  
}  
  
// функція, що відключає всі канали передачі  
void host_disable_all_pipe(void)  
{  
    U8 i;  
    for (i=0;i<7;i++)  
    {  
        Host_reset_pipe(i);  
        Host_select_pipe(i);  
        Host_unallocate_memory();  
        Host_disable_pipe();  
    }  
}
```

Так само до першого модуля підключається бібліотека `config.h` та `conf_usb.h`. Бібліотека `config.h` містить у собі макроси для визначення швидкості передачі по UART, дільники для АЦП, підключення макросів для контролера (`io.h`, `mcu.h`), перевизначення типів даних, підключення визначників і макросів для компіляторів (`compiler.h`), підключення бібліотек для переривань (`interrupt.h`) і для роботи з пам'яттю (`pgmspace.h`), бібліотеку конфігурації планувальника (`conf_scheduler.h`). У даному прикладі використовується компілятор GCC.

Бібліотека `conf_usb.h` містить у собі можливі зовнішні конфігурації USB підключення.

Модуль низькорівневої ініціалізації є стандартним і універсальним для контролерів AT90USBxxx.

Структурне представлення бібліотек першого модуля, що підключаються, наведено на рис. 7.22.

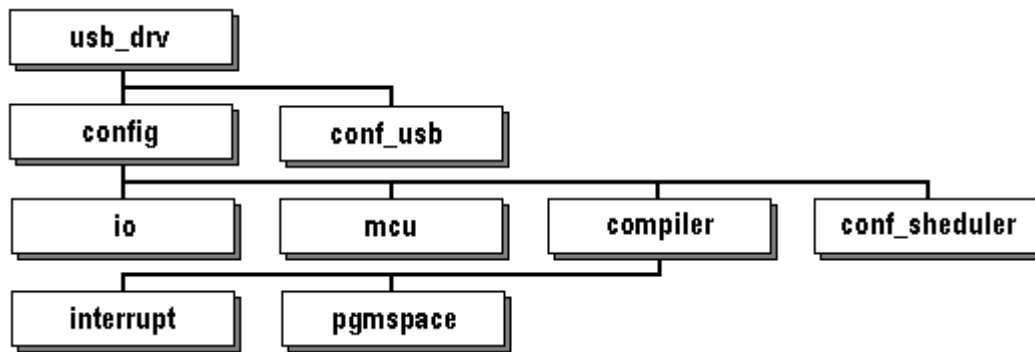


Рис. 7.22.

7.6.2. Модуль програмної реалізації USB

Другий модуль є центральним модулем програмної реалізації USB протоколу і по реалізації відповідає офіційній специфікації USB. Даний модуль найбільш об'ємний і містить у собі велику кількість бібліотек, відповідальних за реалізацію транзакцій, дескрипторів, пакетів, формування кадрів і т.д.

usb_task – перший підмодуль, що включає в себе три основних функції:

```

// функція, що налаштовує пристрій у залежності
// від режиму роботи
void usb_task_init(void)
{
    #if (USB_HOST_FEATURE == ENABLED)
        U8 delay;
    #endif

    #if (USE_USB_PADS_REGULATOR==ENABLE)    Usb_enable_regulator();
    #endif

    #if ((USB_DEVICE_FEATURE == ENABLED)&& (USB_HOST_FEATURE ==
ENABLED))
        Usb_enable_uid_pin();
        delay=PORTA;
        g_usb_mode=USB_MODE_UNDEFINED;
        if(Is_usb_id_device())
        {
            g_usb_mode=USB_MODE_DEVICE;
            usb_device_task_init();
        }
        else
        {
            g_usb_mode=USB_MODE_HOST;
            Usb_ack_id_transition();
            Usb_enable_id_interrupt();
            Enable_interrupt();
            usb_host_task_init();
        }
        g_old_usb_mode=g_usb_mode;
    
```

```
#elif ((USB_DEVICE_FEATURE == ENABLED)&& (USB_HOST_FEATURE ==  
DISABLE))  
    Usb_force_device_mode();  
    usb_device_task_init();  
  
#elif ((USB_DEVICE_FEATURE == DISABLE)&& (USB_HOST_FEATURE ==  
ENABLED))  
    Usb_force_host_mode();  
    usb_host_task_init();  
#elif ((USB_DEVICE_FEATURE == DISABLE)&& (USB_HOST_FEATURE ==  
DISABLE))  
    #error at least one of USB_DEVICE_FEATURE or USB_HOST_FEATURE should be  
    enabled  
#endif  
}  
// функція організації подій для зазначеного режиму передач  
void usb_task(void)  
{  
    // DUAL ROLE DEVICE USB MODE  
    #if ((USB_DEVICE_FEATURE == ENABLED)&& (USB_HOST_FEATURE ==  
ENABLED))  
  
    if(Is_usb_id_device())  
    { g_usb_mode=USB_MODE_DEVICE;}  
    else  
    { g_usb_mode=USB_MODE_HOST;}  
    #if ( ID_PIN_CHANGE_GENERATE_RESET == ENABLE)  
        // Hot ID transition generates wdt reset  
        if((g_old_usb_mode!=g_usb_mode))  
        {Wdt_change_16ms(); while(1);}  
    #endif  
    // Depending on current usb mode, launch the correct usb task  
  
    switch(g_usb_mode)  
    {  
        case USB_MODE_DEVICE:  
            usb_device_task();  
            break;  
        case USB_MODE_HOST:  
            usb_host_task();  
            break;  
        case USB_MODE_UNDEFINED:  
        default:  
            break;  
    }  
    // DEVICE ONLY USB MODE  
    #elif ((USB_DEVICE_FEATURE == ENABLED)&& (USB_HOST_FEATURE ==  
DISABLE))  
        usb_device_task();  
    // REDUCED HOST ONLY USB MODE
```

```

#elif ((USB_DEVICE_FEATURE == DISABLE)&& (USB_HOST_FEATURE ==
ENABLED))
    usb_host_task();
// ERROR, NO MODE ENABLED
#elif ((USB_DEVICE_FEATURE == DISABLE)&& (USB_HOST_FEATURE ==
DISABLE))
    #error at least one of USB_DEVICE_FEATURE or USB_HOST_FEATURE should be
enabled
    #error otherwise the usb task has nothing to do ...
#endif
}
// функція, що викликається при кожному перериванні;
// перевіряється стан пристрою, визначається переривання
// і подальші дії (залежно від режиму роботи)
#ifdef AVRGCC
ISR(SIG_USB_GEN)
#else
#pragma vector = USB_GENERAL_vect
__interrupt void usb_general_interrupt()
#endif
{
    // DEVICE events management
    #if (USB_DEVICE_FEATURE == ENABLED)
        // VBUS state detection
        if (Is_usb_vbus_transition() && Is_usb_vbus_interrupt_enabled())
        {
            Usb_ack_vbus_transition();
            if (Is_usb_vbus_high())
            {
                usb_connected = TRUE;
                Usb_vbus_on_action();
                Usb_send_event(EVT_USB_POWERED);
                Usb_enable_reset_interrupt();
                usb_start_device();
                Usb_attach();
            }
        }
        else
        {
            Usb_vbus_off_action();
            usb_connected = FALSE;
            Usb_send_event(EVT_USB_UNPOWERED);
        }
    }
    // Device start of frame received
    if (Is_usb_sof() && Is_sof_interrupt_enabled())
    {
        Usb_ack_sof();
        Usb_sof_action();
    }
    // Device Suspend event (no more USB activity detected)
    if (Is_usb_suspend() && Is_suspend_interrupt_enabled())
    {

```

```
    Usb_ack_suspend();
    Usb_enable_wake_up_interrupt();
    Usb_ack_wake_up(); // clear wake up to detect next event
    Usb_freeze_clock();
    Usb_send_event(EVT_USB_SUSPEND);
    Usb_suspend_action();
}
// Wake up event (USB activity detected): Used to resume
if (Is_usb_wake_up() && Is_swake_up_interrupt_enabled())
{
    Usb_unfreeze_clock();
    Usb_ack_wake_up();
    Usb_disable_wake_up_interrupt();
    Usb_wake_up_action();
    Usb_send_event(EVT_USB_WAKE_UP);
}
// Resume state bus detection
if (Is_usb_resume() && Is_resume_interrupt_enabled())
{
    Usb_disable_wake_up_interrupt();
    Usb_ack_resume();
    Usb_disable_resume_interrupt();
    Usb_resume_action();
    Usb_send_event(EVT_USB_RESUME);
}
// USB bus reset detection
if (Is_usb_reset() && Is_reset_interrupt_enabled())
{
    Usb_ack_reset();
    usb_init_device();
    Usb_reset_action();
    Usb_send_event(EVT_USB_RESET);
}
#endif // End DEVICE FEATURE MODE

// HOST events management
#if (USB_HOST_FEATURE == ENABLED)
// ID pin change detection
if(Is_usb_id_transition() && Is_usb_id_interrupt_enabled())
{
    Usb_ack_id_transition();
    if(Is_usb_id_device()) // Going to device mode
    {
        Usb_send_event(EVT_USB_DEVICE_FUNCTION);
    }
    else // Going to host mode
    {
        Usb_send_event(EVT_USB_HOST_FUNCTION);
    }
}
#if ( ID_PIN_CHANGE_GENERATE_RESET == ENABLE)
// Hot ID transition generates wdt reset
Wdt_change_16ms(); while(1);
```

```

    #endif
    Usb_id_transition_action();
}
// The device has been disconnected
if(Is_device_disconnection() && Is_host_device_disconnection_interrupt_enabled())
{
    Host_ack_device_disconnection();
    device_state=DEVICE_UNATTACHED;
    Usb_send_event(EVT_HOST_DISCONNECTION);
    Host_device_disconnection_action();
}
// Device connection
if(Is_device_connection() && Is_host_device_connection_interrupt_enabled())
{
    Host_ack_device_connection();
    host_disable_all_pipe();
    Host_device_connection_action();
}
// Host Start of frame has been sent
if (Is_host_sof() && Is_host_sof_interrupt_enabled())
{
    Host_ack_sof();
    Usb_send_event(EVT_HOST_SOF);
    private_sof_counter++;
    Host_sof_action();
}
// Host Wake-up has been received
if (Is_host_hwup() && Is_host_hwup_interrupt_enabled())
{
    Host_disable_hwup_interrupt();
    Pll_start_auto    Wait_pll_ready();
    Usb_unfreeze_clock(); // Enable clock on USB interface
    Host_ack_hwup();      // Clear HWUP interrupt flag
    Usb_send_event(EVT_HOST_HWUP); // Send software event
    Host_hwup_action();   // Map custom action
}
#endif // End HOST FEATURE MODE
}

```

Так само підключаються бібліотеки першого модуля (config.h, conf_usb.h, usb_drv.h), бібліотека сторожового таймера (wdt_drv.h), бібліотека ФАПЧ (pll_drv.h), бібліотека керування потужністю (power_drv.h), бібліотека дескрипторів (usb_descriptors.h) і бібліотеки другого підмодуля – події пристрою і хоста (usb_device_task.h, usb_host_task.h).

Другий підмодуль можна структурно об'єднати схемою, що представлена на рис. 7.23.

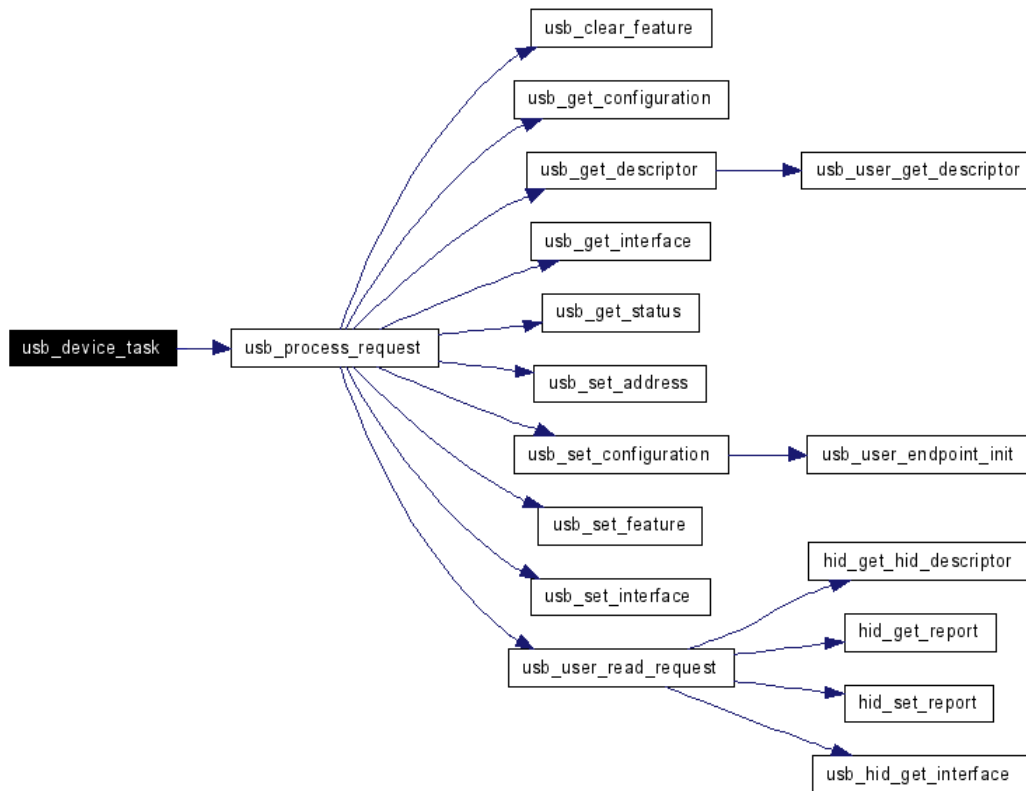


Рис. 7.23.

7.6.3. Функції ініціалізації USB-пристрою

// функція включення USB-пристрою

```
void usb_start_device (void)
```

```
{
```

```
    Pll_start_auto();           // включення ФАПЧ
```

```
    Wait_pll_ready();           // готовність ФАПЧ
```

```
    Usb_unfreeze_clock();       // скидання FRZCLK
```

```
    Usb_enable_suspend_interrupt(); // підключення suspend
```

```
    Usb_enable_reset_interrupt(); // підключення переривань скидання
```

```
    usb_init_device();           // конфігурування нульової кінцевої точки
```

```
    Usb_attach();               // сброс DETACH
```

```
    usb_connected = FALSE;      // прапор підключення USB
```

```
}
```

// функція ініціалізації подій пристрою

```
void usb_device_task_init(void)
```

```
{
```

```
    Enable_interrupt();          // підключення переривань
```

```
    Usb_disable();              // відключення USB
```

```
    Usb_enable();               // включення USB
```

```
    Usb_select_device();         // установка контролера в режим пристрою
```

```
    Usb_enable_vbus_interrupt(); // підключення переривань VBUS
```

```
    Enable_interrupt();          // підключення переривань
```

```
}
```

// функція, що описує події пристрою

```
void usb_device_task(void)
```

```

{
    // пристрій не підключений
    if (Is_usb_vbus_high() && usb_connected == FALSE)
    {
        // підключення пристрою
        usb_connected = TRUE;    // установка прапора підключення
        Usb_vbus_on_action();    // підключення подій VBUS
        // відправлення події EVT_USB_POWERED
        Usb_send_event(EVT_USB_POWERED);
        // підключення RESET переривання
        Usb_enable_reset_interrupt();
        Usb_attach();           // скидання DETACH
    }
    // перевірка скидання шини
    if (Is_usb_event(EVT_USB_RESET))
    {
        // підтвердження прийнятого RESET
        Usb_ack_event(EVT_USB_RESET);
        Usb_reset_endpoint(0);    // скидання кінцевої точки
        usb_configuration_nb = 0; // нульова конфігурація
    }

    // енумерация пристрою
    // вибір нульової кінцевої точки
    Usb_select_endpoint(EP_CONTROL);
    // перевірка дозволу на SETUP-пакет
    if (Is_usb_receive_setup())
    {
        usb_process_request();    // функція читання SETUP-пакета
    }
}

// функція читання SETUP-пакета
void usb_process_request(void)
{
    U8 bmRequest;
    bmRequestType = Usb_read_byte(); // тип запиту
    bmRequest = Usb_read_byte();    // запит
    switch (bmRequest)               // визначення запиту
    {
        case GET_DESCRIPTOR:
            if (0x80 == bmRequestType) { usb_get_descriptor(); }
            else { usb_user_read_request(bmRequestType, bmRequest); }
            break;

        case GET_CONFIGURATION:
            if (0x80 == bmRequestType) { usb_get_configuration(); }
            else { usb_user_read_request(bmRequestType, bmRequest); }
            break;

        case SET_ADDRESS:
            if (0x00 == bmRequestType) { usb_set_address(); }

```

```
else { usb_user_read_request(bmRequestType, bmRequest); }
break;

case SET_CONFIGURATION:
if (0x00 == bmRequestType) { usb_set_configuration(); }
else { usb_user_read_request(bmRequestType, bmRequest); }
break;

case CLEAR_FEATURE:
if (0x02 >= bmRequestType) { usb_clear_feature(); }
else { usb_user_read_request(bmRequestType, bmRequest); }
break;

case SET_FEATURE:
if (0x02 >= bmRequestType) { usb_set_feature(); }
else { usb_user_read_request(bmRequestType, bmRequest); }
break;

case GET_STATUS:
if ((0x7F < bmRequestType) & (0x82 >= bmRequestType))
{ usb_get_status(); }
else { usb_user_read_request(bmRequestType, bmRequest); }
break;

case GET_INTERFACE:
if (bmRequestType == 0x81) { usb_get_interface(); }
else { usb_user_read_request(bmRequestType, bmRequest); }
break;

case SET_INTERFACE:
if (bmRequestType == 0x01) {usb_set_interface();}
break;

case SET_DESCRIPTOR:
case SYNCH_FRAME:
default:
if(usb_user_read_request(bmRequestType, bmRequest) == FALSE)
{
    Usb_enable_stall_handshake();
    Usb_ack_receive_setup();
    return;
}
break;
}
}
```

По прийнятому запиту визначається подальше формування транзакцій і дескрипторів. HID-транзакції визначаються у функції *usb_user_read_request()*.

// функція зчитування заданих запитів
Bool usb_user_read_request(U8 type, U8 request)


```

{
U8 descriptor_type ;
U8 string_type ;

string_type = Usb_read_byte();
descriptor_type = Usb_read_byte();
switch(request)
{
    case GET_DESCRIPTOR:

        switch (descriptor_type)
        {
            case REPORT:
                hid_get_report();
                return TRUE;

            break;

            case HID:
                hid_get_hid_descriptor();
                return TRUE;

            break;
            default:
                return FALSE;

            break;
        }
        break;
    case SET_CONFIGURATION:
        switch (descriptor_type)
        {
            case SET_REPORT:
                hid_set_report();
                return TRUE;

            break;

            default:
                return FALSE;

            break;
        }
        break;
    case GET_INTERFACE:
        usb_hid_get_interface();
        return TRUE;
        break;
        default:
            return FALSE;

        break;
    }
    return FALSE;
}

```

Докладний опис функцій обробки запитів наведений в бібліотеках *usb_specific_request*, *usb_standart_request*. Нижче представлені структури транзакцій і дескрипторів:

// стандартний USB-запит (конфігураційний пакет)

```
struct USB_STANDART_REQUEST
{
    unsigned char    bmRequestType;    // тип запиту
    unsigned char    bRequest;         // код запиту
    unsigned short   wValue;           // параметр запиту
    unsigned short   wIndex;           // індекс або зсув
    unsigned short   wLength;          // число байт для передачі
};
```

// стандартний дескриптор пристрою

```
struct USB_DEVICE_DESCRIPTOR
{
    unsigned char    bLength;           // розмір дескриптора в байтах
    unsigned char    bDescriptorType;   // тип дескриптора
    unsigned short   bscUSB;            // бінарний код
    unsigned char    bDeviceClass;      // код класу USB
    unsigned char    bDeviceSubClass;   // код підкласу USB
    unsigned char    bDeviceProtocol;   // код протоколу USB
    unsigned char    bMaxPacketSize0;  // максимальний розмір
                                        // пакета для нульової кінцевої точки
    unsigned short   idVendor;          // ідентифікатор виготовлювача
    unsigned short   idProduct;         // ідентифікатор продукту
    unsigned short   bcdDevice;         // номер версії специфікації
    unsigned char    iManufacturer;     // індекс дескриптора
                                        // строка, що описує виготовлювача
    unsigned char    iProduct;          // індекс дескриптора
                                        // строка, що описує продукт
    unsigned char    iSerialNumber;     // індекс дескриптора
                                        // строка, що містить серійний номер
    unsigned char    bNumConfigurations; // кількість
                                        // можливих конфігурацій
};
```

// дескриптор конфігурації

```
struct USB_CONFIGURATION_DESCRIPTOR
{
    unsigned char    bLength;           // розмір дескриптора в байтах
    unsigned char    bDescriptorType;   // тип дескриптора
    unsigned short   wTotalLength;      // загальний обсяг даних
    unsigned char    bNumInterfaces;    // кількість інтерфейсів,
                                        // що підтримуються даною конфігурацією
    unsigned char    bConfigurationValue; // ідентифікатор
    // конфігурації, що використовується при виклику
    unsigned char    iConfiguration;    // індекс дескриптора
                                        // строка, що описує дану конфігурацію
    unsigned char    bmAttributes;      // характеристики
                                        // конфігурації
};
```

```

    unsigned char    MaxPower;           // код потужності,
                                         // що споживається USB-пристроєм від шини
};

// дескриптор інтерфейсу
struct USB_INTERFACE_DESCRIPTOR
{
    unsigned char    bLength;           // розмір дескриптора в байтах
    unsigned char    bDescriptorType;   // тип дескриптора
    unsigned char    bInterfaceNumber;  // номер інтерфейсу
    unsigned char    bAlternateSetting; // альтернативний
                                         // номер інтерфейсу
    unsigned char    bNumEndpoints;     // число кінцевих
    // точок для цього інтерфейсу, без нульової кінцевої точки
    unsigned char    bInterfaceClass;   // код класу інтерфейсу
    unsigned char    bInterfaceSubClass; // код підкласу
    unsigned char    bInterfaceProtocol; // код протоколу
    unsigned char    iInterface;        // індекс
                                         // дескриптора рядка, що описує інтерфейс
};

// дескриптор кінцевої точки
struct USB_ENDPOINT_DESCRIPTOR
{
    unsigned char    bLength;           // розмір дескриптора в байтах
    unsigned char    bDescriptorType;   // тип дескриптора
    unsigned char    bEndpointAddress;  // код адреси
                                         // кінцевої точки
    unsigned char    bmAttributes;      // атрибути кінцевої точки
    unsigned short   wMaxPacketSize;    // максимальний розмір
                                         // пакета для кінцевої точки
    unsigned char    bInterval;        // інтервал опитування кінцевої
    // точки при передачі даних (у мілісекундах)
};

// уточнюючий дескриптор пристрою
struct USB_DEVICE_QUALIFIER_DESCRIPTOR
{
    unsigned char    bLength;           // розмір дескриптора в байтах
    unsigned char    bDescriptorType;   // тип дескриптора
    unsigned short   bscUSB;           // номер версії специфікації USB
    unsigned char    bDeviceClass;      // код класу USB
    unsigned char    bDeviceSubClass;   // код підкласу USB
    unsigned char    bDeviceProtocol;   // код протоколу
    unsigned char    bMaxPacketSize0;   // максимальний розмір
                                         // пакета для нульової кінцевої точки
    unsigned char    bNumConfigurations; // кількість
                                         // додаткових конфігурацій пристрою
    unsigned char    bReserved;        // зарезервовано, дорівнює 0
};

// дескриптор мови

```

```
struct USB_LANGUAGE_ID_DESCRIPTOR
{
    unsigned char    bLength;           // розмір дескриптора в байтах
    unsigned char    bDescriptorType;   // тип дескриптора
    unsigned short    wlangid;          // ідентифікатор мови
};

// дескриптор розробника
struct USB_MANUFACTURER_STRING_DESCRIPTOR
{
    unsigned char    bLength;           // розмір дескриптора в байтах
    unsigned char    bDescriptorType;   // тип дескриптора
    unsigned short    wstring[USB_MN_LENGTH]; // характеристики
};

// дескриптор продукту
struct USB_PRODUCT_STRING_DESCRIPTOR
{
    unsigned char    bLength;           // розмір дескриптора в байтах
    unsigned char    bDescriptorType;   // тип дескриптора
    unsigned short    wstring[USB_PN_LENGTH]; // характеристики
};

// запит серійного номера
struct USB_SERIAL_NUMBER
{
    unsigned char    bLength;           // розмір дескриптора в байтах
    unsigned char    bDescriptorType;   // тип дескриптора
    unsigned short    wstring[USB_SN_LENGTH]; // характеристики
};

// HID-дескриптор
struct USB_HID_DESCRIPTOR
{
    unsigned char    bLength;           // розмір дескриптора в байтах
    unsigned char    bDescriptorType;   // тип дескриптора (HID)
    unsigned short    bscHID;           // бінарний код
    unsigned char    bCountryCode;      // код розташування пристрою
    unsigned char    bNumDescriptors;   // кількість дескрипторів
    unsigned char    bRDescriptorType;  // репорт типу дескриптора
    unsigned short    wDescriptorLength; // загальна довжина репорту
};
```

Ідентифікатори, визначники та символічні значення задаються в бібліотеці *usb_descriptors.h*. Нижче наведені ідентифікатори пристрою і визначники:

```
// USB Device descriptor
#define USB_SPECIFICATION    0x0200
#define DEVICE_CLASS         0
#define DEVICE_SUB_CLASS     0
```

```

#define DEVICE_PROTOCOL      0
#define EP_CONTROL_LENGTH 64
#define VENDOR_ID            0x03EB
#define PRODUCT_ID           0x1286
#define RELEASE_NUMBER       0x1000
#define MAN_INDEX            0x01
#define PROD_INDEX           0x02
#define SN_INDEX             0x03
#define NB_CONFIGURATION     1

    // HID generic CONFIGURATION
#define NB_INTERFACE      1
#define CONF_NB           1
#define CONF_INDEX        0
#define CONF_ATTRIBUTES   USB_CONFIG_BUSPOWERED
#define MAX_POWER         50

    // USB Interface descriptor
#define INTERFACE_NB      0
#define ALTERNATE         0
#define NB_ENDPOINT       2
#define INTERFACE_CLASS   0x03 // HID Class
#define INTERFACE_SUB_CLASS 0x00
#define INTERFACE_PROTOCOL 0x00
#define INTERFACE_INDEX   0

    // USB Endpoint 1 descriptor FS
#define ENDPOINT_NB_1      (EP_HID_IN | 0x80)
#define EP_ATTRIBUTES_1    0x03
#define EP_IN_LENGTH       8 // розмір репорту
#define EP_SIZE_1          EP_IN_LENGTH
#define EP_INTERVAL_1      20 //interrupt pooling from host

    // USB Endpoint 1 descriptor FS
#define ENDPOINT_NB_2      (EP_HID_OUT)
#define EP_ATTRIBUTES_2    0x03
#define EP_OUT_LENGTH      8 // розмір репорту
#define EP_SIZE_2          EP_OUT_LENGTH
#define EP_INTERVAL_2      20 //interrupt pooling from host

#define SIZE_OF_REPORT     0x35
#define LENGTH_OF_REPORT_IN 0x08
#define LENGTH_OF_REPORT_OUT 0x08

#define DEVICE_STATUS      0x00 // TBD
#define INTERFACE_STATUS   0x00 // TBD

#define LANG_ID            0x00

/* HID specific */
#define HID                0x21
#define REPORT             0x22

```

```
#define SET_REPORT          0x02
#define HID_DESCRIPTOR      0x21
#define HID_BDC             0x1001
#define HID_COUNTRY_CODE   0x00
#define HID_CLASS_DESC_NB  0x01
#define HID_DESCRIPTOR_TYPE 0x22
```

Нижче представлені символні значення, що визначають пристрій.
Всі символні значення передаються в unicode:

```
#define USB_MN_LENGTH      5
#define USB_MANUFACTURER_NAME \
{ Usb_unicode('A') \
, Usb_unicode('T') \
, Usb_unicode('M') \
, Usb_unicode('E') \
, Usb_unicode('L') \
}
```

```
#define USB_PN_LENGTH      18
#define USB_PRODUCT_NAME \
{ Usb_unicode('S') \
,Usb_unicode('T') \
,Usb_unicode('A') \
,Usb_unicode('N') \
,Usb_unicode('I') \
,Usb_unicode('S') \
,Usb_unicode('L') \
,Usb_unicode('A') \
,Usb_unicode('V') \
,Usb_unicode(' ') \
,Usb_unicode('H') \
,Usb_unicode('I') \
,Usb_unicode('D') \
,Usb_unicode(' ') \
,Usb_unicode('1') \
,Usb_unicode('2') \
,Usb_unicode('8') \
,Usb_unicode('6') \
}
```

```
#define USB_SN_LENGTH      0x05
#define USB_SERIAL_NUMBER \
{ Usb_unicode('1') \
,Usb_unicode('.') \
,Usb_unicode('0') \
,Usb_unicode('.') \
,Usb_unicode('0') \
}
```

```
#define LANGUAGE_ID        0x0409
```

Третій модуль містить у собі формування й зчитування репортів hid пристрою, обробку прийнятих даних й опитування ключів. Даний модуль підключається до попереднього. Опис функцій, реалізованих у бібліотеці hid:

```
void hid_task_init(void)                // ініціалізація
{
    hid_task();
}
void hid_task(void)                    // функція зчитування і відправлення
{
    // ініціалізація портів
    DDRD=0xff;
    DDRC=0x00;
    // зчитування даних
    if(Is_device_enumerated())          // перевірка енумерації пристрою
    {
        // установка кінцевої точки
        Usb_select_endpoint(EP_HID_OUT);
        if(Is_usb_receive_out())        // очікування репорту
        {
            first_byte=Usb_read_byte(); // зчитування першого байта
            // зчитування другого байта (стан світлодіодів)
            led1=Usb_read_byte();
            Usb_ack_receive_out();       // відправлення підтвердження
        }

        if(led1==7) PORTD|=0x01;        // активація LED1
        if(led1==8) PORTD&=~0x01;      // відключення LED1
        if(led1==9) PORTD|=0x02;        // активація LED2
        if(led1==10) PORTD&=~0x02;     // відключення LED2
        // опитування ключів
        if((PINC&0x01)==0x01) key1=0;
        if((PINC&0x02)==0x02) key2=0;
        if((PINC&0x01)!=0x01) key1=5;
        if((PINC&0x02)!=0x02) key2=5;
        // посилка даних
        // установка кінцевої точки
        Usb_select_endpoint(EP_HID_IN);
        if(Is_usb_in_ready()) // очікування репорту
        {
            // відправлення даних
            Usb_write_byte(key1);        // стан першого ключа
            Usb_write_byte(key2);        // стан другого ключа
            // відправлення інших даних
            Usb_write_byte(22);
            Usb_write_byte(45);
            Usb_write_byte(3);
            Usb_write_byte(2);
            Usb_write_byte(57);
            Usb_write_byte(112);
        }
    }
}
```

```
    Usb_ack_fifocon();                // очікування підтвердження
  }
}

void sof_action()                    // відлік кадрів (кожну 1 ms)
{
    cpt_sof++;                       // лічильник SOF
}
```

Четвертий модуль, як було зазначено раніше, є планувальником між другим і третім модулем, відповідальний за трасування даних і підключається до головної функції `main()`. Оголошення, що зіставляють функції планувальника з функціями подій, описаних раніше, представлені в бібліотеці *conf_scheduler.h*:

```
#define SCHEDULER_TYPE      SCHEDULER_FREE
#define Scheduler_task_1_init  usb_task_init
#define Scheduler_task_1      usb_task
#define Scheduler_task_2_init  hid_task_init
#define Scheduler_task_2      hid_task
```

Виклик заданих функцій планувальника описаний у бібліотеці *sheduler*:

```
void scheduler_init (void)
{
    #ifdef Scheduler_time_init
        Scheduler_time_init();
    #endif

    #ifdef TOKEN_MODE
        token = TOKEN_FREE;
    #endif

    #ifdef Scheduler_task_1_init
        Scheduler_task_1_init();
        Scheduler_call_next_init();
    #endif

    #ifdef Scheduler_task_2_init
        Scheduler_task_2_init();
        Scheduler_call_next_init();
    #endif
    ...
}

void scheduler_tasks (void)
{
    scheduler_empty_fct();

    for(;;)
```



```

{
    Scheduler_new_schedule();

    #ifdef Scheduler_task_1
        Scheduler_task_1();
        Scheduler_call_next_task();
    #endif

    #ifdef Scheduler_task_2
        Scheduler_task_2();
        Scheduler_call_next_task();
    #endif
    ...
}

```

Підключення функцій до main():

```

void main(void)
{
    #ifndef AVRGCC
        // відключення сторожового таймера
        Wdt_off();
    #else
        wdt_reset();
        Wdt_clear_flag();
        Wdt_change_enable();
        Wdt_stop();
    #endif
    Clear_prescaler();           // відключення дільника частоти
    scheduler();                 // підключення модуля USB
}

```

7.7. Моделювання пристрою в Proteus

По розробленій й описаній раніше принциповій схемі виконується налаштування моделі пристрою в Proteus. Вказується hex-файл програми прошивання, частота роботи мікроконтролера, параметри елементів, підключається USB-connector.

Для моделювання USB-передач необхідно встановити віртуальний USB-хост, що йде в програмному пакеті Proteus 7.4 SP3. Разом з ним встановлюються віртуальні драйвера, що дозволяють операційній системі звертатися до пристроїв через віртуальний USB-хост як до зовнішніх пристроїв. Всі інші етапи ініціалізації та конфігурації пристрою відбуваються як з фізичним пристроєм. Для моделювання пристрою задані наступні ідентифікатори – idVendor(0x03EB) і idProduct(0x1286). Після першого включення система визначає пристрій, задає йому адресу й підключає до диспетчера пристроїв як USB HID-сумісний пристрій (рис. 7.24 та рис. 7.25).

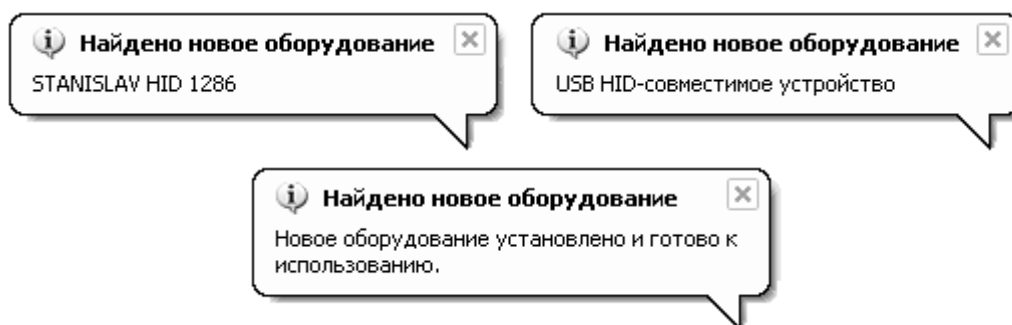


Рис. 7.24.

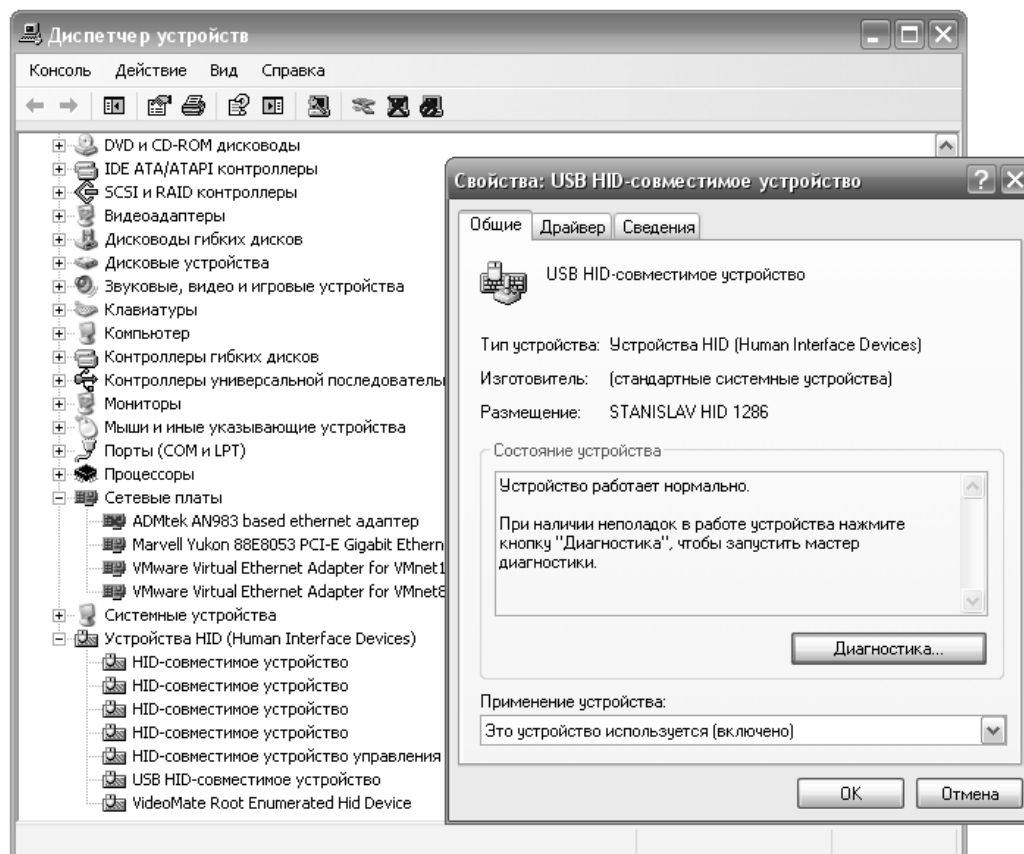


Рис. 7.25.

Так само в Proteus 7.4 SP3 є USB-аналізатор для роботи з USB-пристроями. USB-аналізатор дозволяє переглядати всі транзакції обміну між пристроєм і віртуальним хостом, переглянути наявність всіх пакетів та їхнього вмісту. USB-аналізатор так само надає опис змінних переданих структур, тип пакета, номера кінцевих точок, наявність підтверджень та ін.

7.8. Розробка програми-терміналу для HID-пристроїв

Програма USB_HID_terminal була написана на C++ і розроблена в оболонці Microsoft Visual Studio 2005. Програма використовує тип передачі даних – Interrupt transfers. Є можливість формування перших 8 байт репорту відправлення і візуалізація перших 8 байт прийнятого репорту. Програма USB_HID_terminal дозволяє керувати зовнішнім

пристроєм. Для прикладу обміну даними реалізоване керування 2 світлодіодами та опитування 2 ключів.

Для обміну даними між комп'ютером і зовнішнім пристроєм необхідно знати його ідентифікатори `idVendor` та `idProduct`. Для зменшення витрат системних ресурсів і прискорення пошуку підключених пристроїв можна вказати їхню кількість. За замовчуванням встановлене значення 10 пристроїв.

Так само програма вказує кількість прийнятих байт і номер пристрою в диспетчері пристроїв операційної системи. Програма використовує статичний метод опитування пристроїв. Визначення підключення, відправлення та прийому даних виконується шляхом натискання кнопок.

Зовнішній вигляд вікна програми представлений на рис. 7.26.

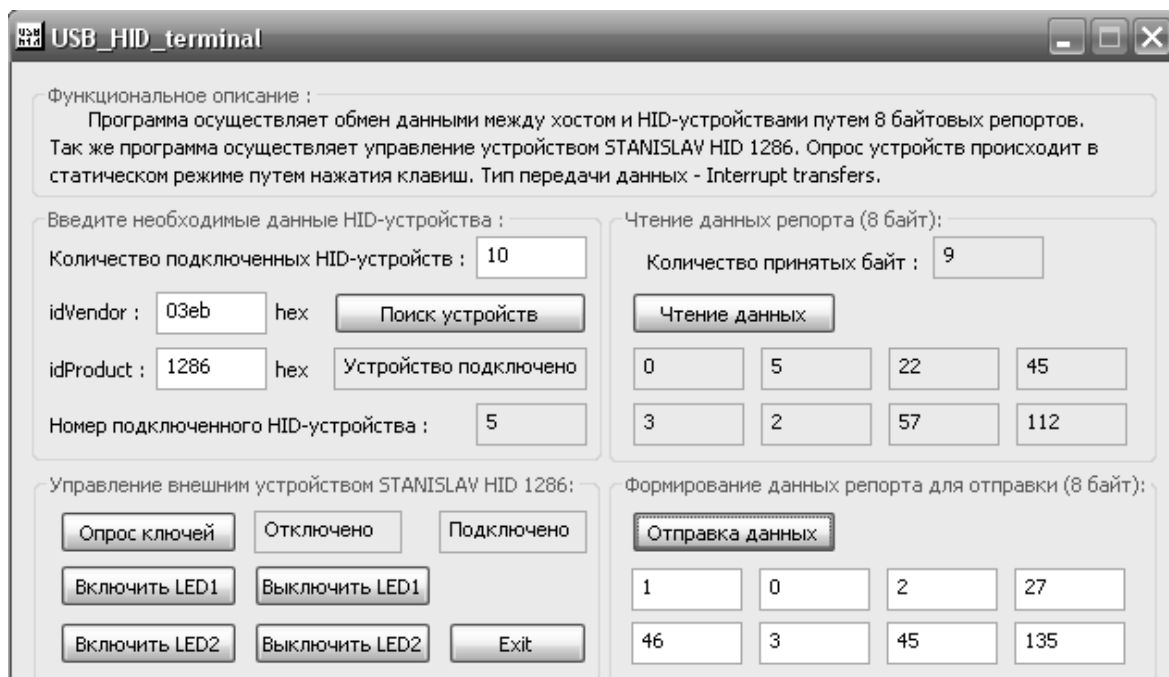


Рис. 7.26.

Програма використовує стандартний драйвер Windows – `hid.lib` для зв'язку із зовнішніми пристроями. Для роботи з HID-структурами використовується HIDAPI-функції, функції для роботи з файлами (`CreateFile`, `ReadFile`, `WriteFile`), функції для роботи зі структурами (`SetupDiEnumDeviceInterfaces`, `SetupDiGetDeviceInterfaceDetail` та ін.).

Структура програми:

- опитування підключених пристроїв;
- порівняння ідентифікаторів пристрою і введених ідентифікаторів;
- відкриття файлів для обміну даними;
- запис/зчитування файлів;
- закриття файлів.

Нижче наведені основні функції програми:

```
// змінні для роботи з USB-пристроями типу HID:
ULONG SIZE_DETAIL;
// масив вхідного репорту
char DATA_IN[256];
// масив вихідного репорту
char DATA_OUT[256];
// дескриптор пристрою
HANDLE
HANDLE_OF_DEVICE;
HANDLE hdevInfo;
// GUID пристрою
GUID HidGuid;
ULONG REQUIRED;
HIDP_CAPS CAPABILITIES;
DWORD
NUMBER_OF_BYTES_TO_READ;
PSP_DEVICE_INTERFACE_DETAIL_DATA DATA_DETAIL;
OVERLAPPED OVER_HID;
// дескриптор зчитування файлу
HANDLE READ;
// дескриптор запису у файл
HANDLE WRITE;
// idVendor зчитаний з hid драйвера
int vendorID;
// idProduct зчитаний з hid драйвера
int productID;
// розраховане значення idVendor (по введеним даним)
int set_vendorID;
// розраховане значення idProduct (по введеним даним)
int set_productID;
// масив введених даних (HEX)
char mas[4];
// масив розрахованих даних (двійково-десятковий)
char znach[4];
// змінна, що фіксує підключення пристрою
bool DEVICE_DETECT=false;
// змінна структури інтерфейсу
LONG
INTERFACE_DETAIL_DATA;
// змінна даних структури
SP_DEVICE_INTERFACE_DATA INTERFACE_DATA;
// інформація про пристрій (атрибути)
HIDD_ATTRIBUTES
DEVICE_ATTRIBUTE;
// номер пристрою
int DEVICE_NUMBER = 0;
// максимальна кількість підключених пристроїв
unsigned char max_number;
PHIDP_PREPARED_DATA PREPARED_DATA;
// функція опитування підключених пристроїв
bool CUSB_HID_terminalDlg::find_hid()
{
```

```

// початкова ініціалізація
HANDLE_OF_DEVICE=NULL;
DATA_DETAIL = NULL;
SIZE_DETAIL = NULL;
Hid_GetHidGuid(&HidGuid); // функція повертає GUID для HID
// одержання інформації HID пристроїв
hdevInfo=SetupDiGetClassDevs (
    &HidGuid,          // ідентифікатор GUID
    NULL,              // визначення імені пристрою по прапорах
    NULL,              // дескриптор вікна
    DIGCF_PRESENT|DIGCF_INTERFACEDevice); // визнач. класу
// визначення розміру структури інтерфейсу
INTERFACE_DATA.cbSize = sizeof(INTERFACE_DATA);
// одержання структури інтерфейса
INTERFACE_DETAIL_DATA=SetupDiEnumDeviceInterfaces (
    hdevInfo,          // дескриптор пристрою
    NULL,              // структура SP_DEVINFO_DATA
    &HidGuid,          // ідентифікатор GUID
    DEVICE_NUMBER,    // нумерація HID пристроїв
    &INTERFACE_DATA); // розмір структури інтерфейса
// функція визначає вміст інтерфейсу пристрою
INTERFACE_DETAIL_DATA=SetupDiGetDeviceInterfaceDetail (
    hdevInfo,          // дескриптор пристрою
    &INTERFACE_DATA, // розмір структури інтерфейса
    NULL,              // визначення детальної інформації
    NULL,              // розмір структури
    &SIZE_DETAIL,     // розмір буфера для детальної інформації
    NULL);
// виділення пам'яті для hdevInfo структури
DATA_DETAIL =
(PSP_DEVICE_INTERFACE_DETAIL_DATA)malloc(SIZE_DETAIL);
// установка розміру структури
DATA_DETAIL->cbSize = sizeof(SP_DEVICE_INTERFACE_DETAIL_DATA);
// функція визначає вміст інтерфейсу пристрою з урахуванням
// розміру структури
INTERFACE_DETAIL_DATA=SetupDiGetDeviceInterfaceDetail (
    hdevInfo,
    &INTERFACE_DATA,
    DATA_DETAIL,      // розмір структури
    SIZE_DETAIL,
    &REQUIRED,
    NULL);
// функція, що відкриває доступ до пристрою
HANDLE_OF_DEVICE=CreateFile (
    DATA_DETAIL->DevicePath,
    NULL, // режим доступу
    FILE_SHARE_READ|FILE_SHARE_WRITE, // спільний доступ
    (LPSECURITY_ATTRIBUTES)NULL,
    OPEN_EXISTING, // дія над пристроєм
    NULL,          // атрибути файлу
    NULL);         // дескриптор файлу шаблону
// відкриття файлів під конкретний пристрій    if(DEVICE_DETECT==true)

```

```
{
// визначення даних дескриптора пристрою
Hid_GetPreparedData
    (HANDLE_OF_DEVICE,
    &PREPARED_DATA);
// визначення властивостей
Hid_GetCaps
    (PREPARED_DATA,
    &CAPABILITIES);
// відкриття файлів
    Write_data();
    Read_data();
}
// установка кількості байт у структурі
DEVICE_ATTRIBUTE.Size = sizeof(DEVICE_ATTRIBUTE);
INTERFACE_DETAIL_DATA= Hid_GetAttributes (
    HANDLE_OF_DEVICE,
    &DEVICE_ATTRIBUTE);
vendorID=DEVICE_ATTRIBUTE.VendorID; // визначення idVendor
// визначення idProduct
productID=DEVICE_ATTRIBUTE.ProductID;
return false;
}
// функція перерахунку idVendor, idProduct з hex
// у двійково-десятковий
void CUSB_HID_terminalDlg::calculate_ID()
{
    unsigned char i=0;
    UpdateData(true);
    // зчитування введенного idVendor (hex)
    mas[0]=m_vendorID[0];
    mas[1]=m_vendorID[1];
    mas[2]=m_vendorID[2];
    mas[3]=m_vendorID[3];
    UpdateData(false);
    // перетворення
    for(i=0;i<=3;i++)
    {
        if((mas[i]>=48)&(mas[i]<=57)) znach[i]=mas[i]-48;
        if((mas[i]>=65)&(mas[i]<=90)) znach[i]=mas[i]-55;
        if((mas[i]>=97)&(mas[i]<=122)) znach[i]=mas[i]-87;
    }
    // запис перетвореного idVendor
    set_vendorID=znach[0]*4096+znach[1]*256+znach[2]*16+znach[3];
    // зчитування введенного idProduct
    UpdateData(true);
    mas[0]=m_productID[0];
    mas[1]=m_productID[1];
    mas[2]=m_productID[2];
    mas[3]=m_productID[3];
    UpdateData(false);
    // перетворення
```

```

    for(i=0;i<4;i++)
    {
        if((mas[i]>=48)&(mas[i]<=57)) znach[i]=mas[i]-48;
        if((mas[i]>=65)&(mas[i]<=90)) znach[i]=mas[i]-55;
        if((mas[i]>=97)&(mas[i]<=122)) znach[i]=mas[i]-87;
    }
    // запис перетвореного idProduct
    set_productID=znach[0]*4096+znach[1]*256+znach[2]*16+znach[3];
}
// функція визначення заданого пристрою
void CUSB_HID_terminalDlg::compare_hid_device()
{
    unsigned char number=0;
    UpdateData(true);
    max_number=m_max_number;    // установка кількості
    // підключених hid-пристроїв
    UpdateData(false);
    // цикл пошуку пристрою
    // підбір номера пристрою
    for(number=0;number<=max_number;number++)
    {
        DEVICE_NUMBER=number;    // установка номера пристрою
        find_hid();                // пошук пристроїв
        calculate_ID();            // розрахунок ідентифікаторів
        // якщо розраховані введені ідентифікатори то
        // ідентифікатори, зчитані із драйвера, збігаються
        if((set_vendorID==vendorID)&(set_productID==productID))
        {
            UpdateData(true);
            m_device_detect=_T("Пристрій підключений"); // стан
            m_number_device=number; // відображення заданого номера
            UpdateData(false);
            DEVICE_DETECT=true;    // визначення пристрою
            break;                // вихід з пошуку
        }
        else // в іншому випадку (пристрій не знайдений)
        {
            CloseHandle(HANDLE_OF_DEVICE); // закриваємо дескриптор
            UpdateData(true);
            m_device_detect=_T("Відключено"); // стан
            m_number_device=0;    // початковий номер пристрою
            UpdateData(false);
            DEVICE_DETECT=false;  // пошук пристрою
        }
    }
}
// "Пошук пристрою"
void CUSB_HID_terminalDlg::OnBnClickedButton1()
{
    // визначення пристрою
    compare_hid_device();
    // повторний пошук з відкриттям файлів

```

```
        find_hid();
        UpdateWindow();
    }
// "Вихід"
void CUSB_HID_terminalDlg::OnBnClickedOk()
{
    CloseHandle(HANDLE_OF_DEVICE); // закриття дескриптора
    CloseHandle(READ);             // закриття дескриптора зчитування
    CloseHandle(WRITE);            // закриття дескриптора запису
    OnOK(); // вихід
}
// "Зчитування даних"
void CUSB_HID_terminalDlg::OnBnClickedButton2()
{
    Read_bytes_from_device();      // функція зчитування байтів
    // відображення прийнятих даних
    UpdateData(true);
    if(DEVICE_DETECT==true)        // пристрій визначений
    {
        m_number_read_bytes=CAPABILITIES.InputReportByteLength;
        m_data_in=DATA_IN[1];
        m_data_in1=DATA_IN[2];
        m_data_in2=DATA_IN[3];
        m_data_in3=DATA_IN[4];
        m_data_in4=DATA_IN[5];
        m_data_in5=DATA_IN[6];
        m_data_in6=DATA_IN[7];
        m_data_in7=DATA_IN[8];
    }
    else                            // пристрій не визначений
    {
        MessageBox(_T("Пристрій не визначений"),
            _T("Info"),MB_ICONINFORMATION);
        m_number_read_bytes=0;
        m_data_in=0;
        m_data_in1=0;
        m_data_in2=0;
        m_data_in3=0;
        m_data_in4=0;
        m_data_in5=0;
        m_data_in6=0;
        m_data_in7=0;
    }
    UpdateData(false);
}
// функція відкриття файлу зчитування
void CUSB_HID_terminalDlg::Read_data()
{
    // відкриття дескриптора зчитування
    READ=CreateFile                // функція відкриття файлу
        (DATA_DETAIL->DevicePath,
        GENERIC_READ,              // зчитування
```



```

        // визначення прапорів (зчитування/запис)
        FILE_SHARE_READ|FILE_SHARE_WRITE,
        (LPSECURITY_ATTRIBUTES)NULL,
        OPEN_EXISTING,          // відкритий файл
        FILE_FLAG_OVERLAPPED,
        NULL);
    }
// функція зчитування байтів даних із пристрою
void CUSB_HID_terminalDlg::Read_bytes_from_device()
{
    // зчитування із пристрою ідентично зчитуванню з файлу
    DWORD    GET_DATA;          // пам'ять під структуру даних
    // якщо дескриптор відкритий і є коректним
    if (READ != INVALID_HANDLE_VALUE)
    {
        GET_DATA = ReadFile      // функція зчитування файлу
        (READ,                    // дескриптор зчитування
        DATA_IN,                 // масив прийнятих даних
        CAPABILITIES.InputReportByteLength,
        &NUMBER_OF_BYTES_TO_READ,
        (LPOVERLAPPED) &OVER_HID);
    }
}
// "Відправлення даних"
void CUSB_HID_terminalDlg::OnBnClickedButton3()
{
    if(DETECT==false)
    {
        MessageBox(_T("Пристрій не визначений"),_T("Info"),MB_ICONINFORMATION);
    }
    // формування даних репорту
    UpdateData(true);
    DATA_OUT[1]=m_data_out1;
    DATA_OUT[2]=m_data_out2;
    DATA_OUT[3]=m_data_out3;
    DATA_OUT[4]=m_data_out4;
    DATA_OUT[5]=m_data_out5;
    DATA_OUT[6]=m_data_out6;
    DATA_OUT[7]=m_data_out7;
    DATA_OUT[8]=m_data_out8;
    UpdateData(false);
    Write_bytes_to_device();      // запис даних у пристрій
}
// функція відкриття файлу запису
void CUSB_HID_terminalDlg::Write_data()
{
    WRITE=CreateFile(            // функція відкриття файлу
    DATA_DETAIL->DevicePath,
    GENERIC_WRITE,               // запис даних
    FILE_SHARE_READ|FILE_SHARE_WRITE,
    (LPSECURITY_ATTRIBUTES)NULL,

```

```
        OPEN_EXISTING,          // відкритий файл
        NULL,
        NULL);
}
// функція відправлення байтів даних
void CUSB_HID_terminalDlg::Write_bytes_to_device()
{
    DWORD    BYTES_TO_WRITE;
    ULONG     SET_DATA; // пам'ять під структуру файлу
    if (WRITE != INVALID_HANDLE_VALUE) // коректний дескриптор
    {
        SET_DATA = WriteFile // функція запису у файл
        (WRITE,              // дескриптор файлу
        DATA_OUT,           // дані репорту
        // розмір репорту
        CAPABILITIES.OutputReportByteLength,
        &BYTES_TO_WRITE,
        NULL);
    }
}
// функція опитування ключів
void CUSB_HID_terminalDlg::OnBnClickedButton8()
{
    m_data_in=0;
    m_data_in1=0;
    Read_bytes_from_device();          // функція читання байтів
    // відображення прийнятих даних
    UpdateData(true);
    if(DEVICE_DETECT==true)           // пристрій визначений
    {
        // зчитування даних
        m_data_in=DATA_IN[1];
        m_data_in1=DATA_IN[2];
        // визначення стану
        if(m_data_in==0) m_led1=_T("Відключено");
        if(m_data_in1==0) m_led2=_T("Відключено");
        if(m_data_in==5) m_led1=_T("Підключено");
        if(m_data_in1==5) m_led2=_T("Підключено");
    }
    else                               // пристрій не визначений
    {
        MessageBox(_T("Пристрій не
        визначений"),_T("Info"),MB_ICONINFORMATION);
        m_data_in=0;
        m_data_in1=0;
        m_led1=_T("Відключено");
        m_led2=_T("Відключено");
    }
    UpdateData(false);
}
// функція активації першого світлодіода
void CUSB_HID_terminalDlg::OnBnClickedButton4()
```

```

{
    if(DEVICE_DETECT==false)
    {
        MessageBox(_T("Пристрій не
визначений"),_T("Info"),MB_ICONINFORMATION);
    }
    UpdateData(true);
    DATA_OUT[2]=7;           // запис команди активації
    UpdateData(false);
    Write_bytes_to_device();  // передача команди
}
// функція відключення першого світлодіоду
void CUSB_HID_terminalDlg::OnBnClickedButton6()
{
    if(DEVICE_DETECT==false)
    {
        MessageBox(_T("Пристрій не
визначений"),_T("Info"),MB_ICONINFORMATION);
    }
    UpdateData(true);
    DATA_OUT[2]=8;           // запис команди відключення
    UpdateData(false);
    Write_bytes_to_device();  // передача команди
}
// функція активації другого світлодіода
void CUSB_HID_terminalDlg::OnBnClickedButton5()
{
    if(DEVICE_DETECT==false)
    {
        MessageBox(_T("Пристрій не
визначений"),_T("Info"),MB_ICONINFORMATION);
    }
    UpdateData(true);
    DATA_OUT[2]=9;           // запис команди активації
    UpdateData(false);
    Write_bytes_to_device();  // передача команди
}
// функція відключення другого світлодіода
void CUSB_HID_terminalDlg::OnBnClickedButton7()
{
    if(DEVICE_DETECT==false)
    {
        MessageBox(_T("Пристрій не
визначений"),_T("Info"),MB_ICONINFORMATION);
    }
    UpdateData(true);
    DATA_OUT[2]=10;          // запис команди відключення
    UpdateData(false);
    Write_bytes_to_device();  // передача команди
}

```

Питання до розділу 7

1. Які існують методи передачі даних по шині USB?
2. Опишіть фізичний інтерфейс шини USB.
3. Яким чином виконується синхронізація передавача й приймача?
4. Який код використовується при передачі даних по шині USB? Дайте характеристику цього коду.
5. Опишіть структуру кадрів даних, що передаються по шині USB.
6. Що таке кінцева точка? Яка максимальна кількість кінцевих точок?
7. Що таке канал передачі даних? Які види каналів існують?
8. У якому вигляді передається інформація з каналів?
9. Яка частота потрібна для роботи USB-контролера?
10. Яким чином відбувається вибір швидкості роботи інтерфейсу?
11. Які модулі використовуються при реалізації USB-протоколу?

Задачі до розділу 7

1. Розробити схему двоканального осцилографу. Зібрати модель в системі Proteus. Розробити програмне забезпечення для мікроконтролера, до функцій якого входить оцифровування аналогових сигналів з двох каналів АЦП та передача даних до комп'ютера.
2. Розробити інтерфейс користувача для спостереження за даними, що надходять по USB з осцилографа. Отримані дані повинні відображатися у графічному вигляді.
3. Розробити систему контролю температури в приміщенні. В якості термодатчика використати мікросхему DS1621 або DS18B20. Система повинна підключатися до комп'ютера по інтерфейсу USB. Розробити інтерфейс програми для комп'ютера для спостереження за температурою.
4. Розробити принципову схему пристрою, який виконує керування графічним рідкокристалічним індикатором. Написати програмне забезпечення для мікроконтролера та комп'ютера, до функцій якого входить передача графічної інформації (наприклад фотографії) з комп'ютера в мікроконтролер, та вивід її на екран графічного РКІ.

8. 32–РОЗРЯДНІ МІКРОКОНТРОЛЕРИ З АРХІТЕКТУРОЮ ARM

Мікропроцесорне ядро ARM7 було розроблене британськими інженерами в 1994 році, і представляє собою високопродуктивний 32–бітний RISC процесор з низьким енергоспоживанням. Малі розміри процесорного ядра дозволили використати велику кількість периферії і пам'яті в готових рішеннях на одному кристалі. Максимальна продуктивність в 130 MIPS дозволила використовувати ядро ARM7 в найрізноманітніших галузях електроніки і автоматизованих систем. Все частіше мікропроцесори ARM знаходять застосування в системах телекомунікацій, портативних пристроях, системах навігації, пристроях цифрової обробки сигналів, автомобільній електроніці, в різного роду терміналах, пристроях передачі даних і відображення інформації.

Наявність мінімального набору команд і їх фіксованої довжини призводить до збільшення об'єму виконавчих програм. Однак при сучасних тенденціях розвитку електронних систем даний фактор не відіграє головної ролі, так як сьогодні вже важко знайти мікропроцесор з менш ніж 32 кБ flash і 8 кБ SRAM пам'яті. Ядро має 32 регістри довжиною 32 біта, 16 з яких доступні одночасно. Останні 16 використовуються при зміні режимів роботи ядра. Регістр r15 (програмний лічильник), r14 (вказівник стека) і r13 (адреса повернення з поточної функції) є спеціалізованими регістрами і не можуть бути використані в якості регістрів загального призначення. Тобто розробнику одночасно доступні 13 регістрів, що дозволяє більш гнучко реалізувати поставлену задачу.

Процесор може працювати в 7 різних режимах: User, FIQ, IRQ, Supervisor, Abort, System, Undefined. 4 з цих режимів (FIQ, IRQ, Undefined, Abort) слугують для обробки операцій виключення — обробка швидкого переривання, обробка звичайного переривання, спроба виконати невідому інструкцію, спроба звернутись до неіснуючої області пам'яті (або за не вирівняною адресою). Режими Abort і Undefined дозволяють зробити емуляцію інструкцій співпроцесора і додати підтримку віртуальній пам'яті відповідно. Інші три режими служать для захисту операційної системи від прикладних програм. Всі режими (крім System) мають свої регістри r13 і r14. Таким чином, при перемиканні режимів немає необхідності зберігати значення вершини стека і адресу повернення.

Процесор підтримує два набори команд – ARM і Thumb. Основна їх відмінність – це співвідношення розміру і продуктивності. Команди ARM мають довжину 32 біта, займають більше місця, але час виконання таких вкрай малий. Команди Thumb мають довжину 16 біт, займають менше місця, але час виконання значно більший, ніж у ARM команд.

Всі приклади, що представлені в даному розділі, розроблені з використанням демоплати Etherus (ETH256), принципова схема якої представлена в додатку Б (<http://www.evodbg.com>).

8.1. Основні характеристики та архітектура ядра ARM7

На рис. 8.1 представлена структурна схема ядра ARM-мікроконтролера.

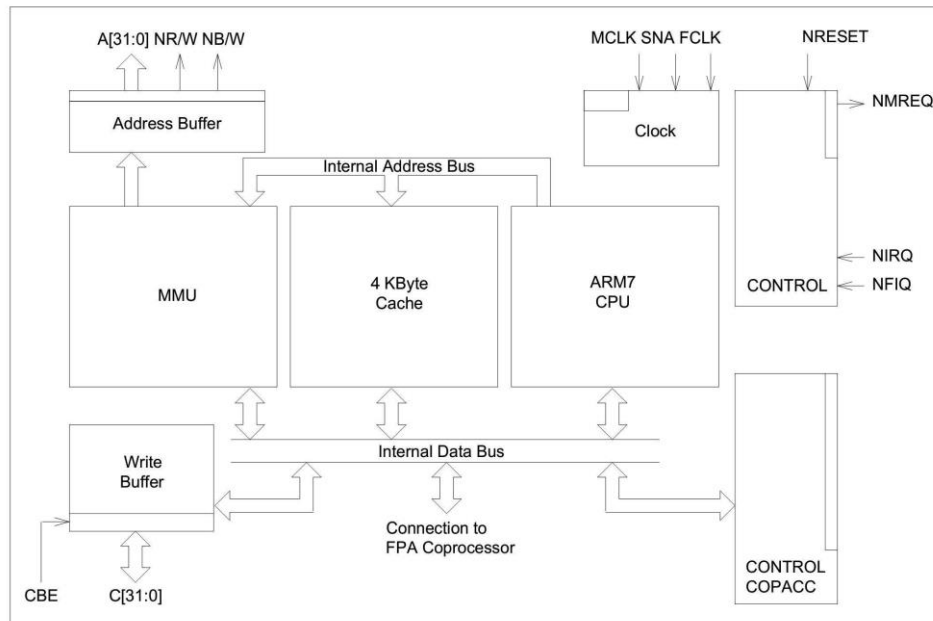


Рис. 8.1.

Відмінними особливостями мікроконтролерів AT91 на базі ядра ARM є:

- 32-розрядний RISC процесор (32-розрядні шини даних і адреси) з продуктивністю 17 MIPS (Million instructions per second) при тактовій частоті 25 МГц (пікова продуктивність 25 MIPS);
- 32-розрядна адресація – лінійний адресний простір в 4 ГБ – виключає потребу у сегментованій, розділеній на банки або оверлейній пам'яті;
- тридцять один 32-розрядний регістр загального призначення і шість регістрів стану;
- регістри адресу, запису і конвеєра;
- циклічний зсувний пристрій і перемножувач;
- трьохрівневий конвеєр (вибірка команди, її декодування і виконання);
- напруга живлення 3,3В і 5В;
- мале споживання 0,6 мА/МГц, при виготовленні по CMOS технології з топологічними нормами 0,8 мкм.;
- повністю статична робота, яка дозволяє додатково знижувати споживання за рахунок зменшення тактової частоти, що ідеально для критичних до споживання систем;
- швидкий відгук на переривання для систем реального масштабу часу;
- підтримка систем віртуальної пам'яті;
- проста, але потужна система команд;

– 32-розрядна система команд ядра ARM7 містить одинадцять базових типів команд:

– два типи використовують вбудований арифметико-логічний пристрій, циклічний зсувний пристрій і помножувач при операціях над даними в банку з 31 регістру, форматом по 32 розряди кожен;

– три класи команд керування пересуванням даних між пам'яттю і регістрами, один оптимізований на забезпечення гнучкості адресації, другий під швидке контекстне перемикання і третій під підкачування даних;

– три команди керують потоком і рівнем пріоритетів виконання;

– три типи призначені для керування зовнішніми співпроцесорами, що дозволяє розширити функціональні можливості системи команд за межами ядра.

Система команд ARM добре обробляється компіляторами мов високого рівня. На відміну від деяких RISC процесорів, процесор ARM7, при виникненні необхідності в деякому зменшенні об'єму кодів, допускає програмування і на асемблері.

8.2. Мікроконтролери сімейства AT90SAM7S

AT91SAM7S – це сімейство Atmel мікроконтролерів з вбудованою флеш-пам'яттю з малим числом зовнішніх виводів, в основу яких закладено 32-бітне RISC ядро ARM7TDMI. Відмінними особливостями цих контролерів є наявність вбудованої високошвидкісної флеш-пам'яті (FLASH) і статичної пам'яті (SRAM), великий набір периферійних вузлів і модулів, включаючи порт USB2.0 (крім AT91SAM7S32), і повний набір системних функцій, які скорочують число зовнішніх компонентів. Мікроконтролери цього сімейства є ідеальною заміною 8-бітним мікроконтролерам, застосування яких вже стає неможливим для реалізації більш складних задач, які потребують пам'ять більшого розміру і більш високої продуктивності.

Для мікроконтролерів цього сімейства існує можливість внутрішньосхемного перепрограмування вбудованої флеш-пам'яті через інтерфейс JTAG-ICE або через паралельний інтерфейс на зовнішньому програматорі. Вбудовані захисні біти і біт секретності прошивання захищають від випадкового перезапису флеш-пам'яті і дозволяють зберегти конфіденційність програми.

Системний контролер AT91SAM7S має в своєму складі контролер скидання, який керує послідовністю процесу вмикання і запуску процесора. Коректність цих дій контролюється вбудованим супервізором напруги живлення і сторожовим таймером, який тактується від незалежного внутрішнього RC-генератора.

Мікроконтролери сімейства AT91SAM7S є контролерами загального призначення. Вбудований в них USB порт дозволяє зробити на їх основі додатки, які потребують наявності зв'язку з комп'ютером або мобільним телефоном. Відмінна цінова політика і відповідність потребам за високим

ступенем інтеграції дозволяють займатися створенням дешевих пристроїв з масовим виготовленням.

8.2.1. Структурна схема

На рис. 8.2 представлена структурна схема мікроконтролера сімейства AT91SAM7S.

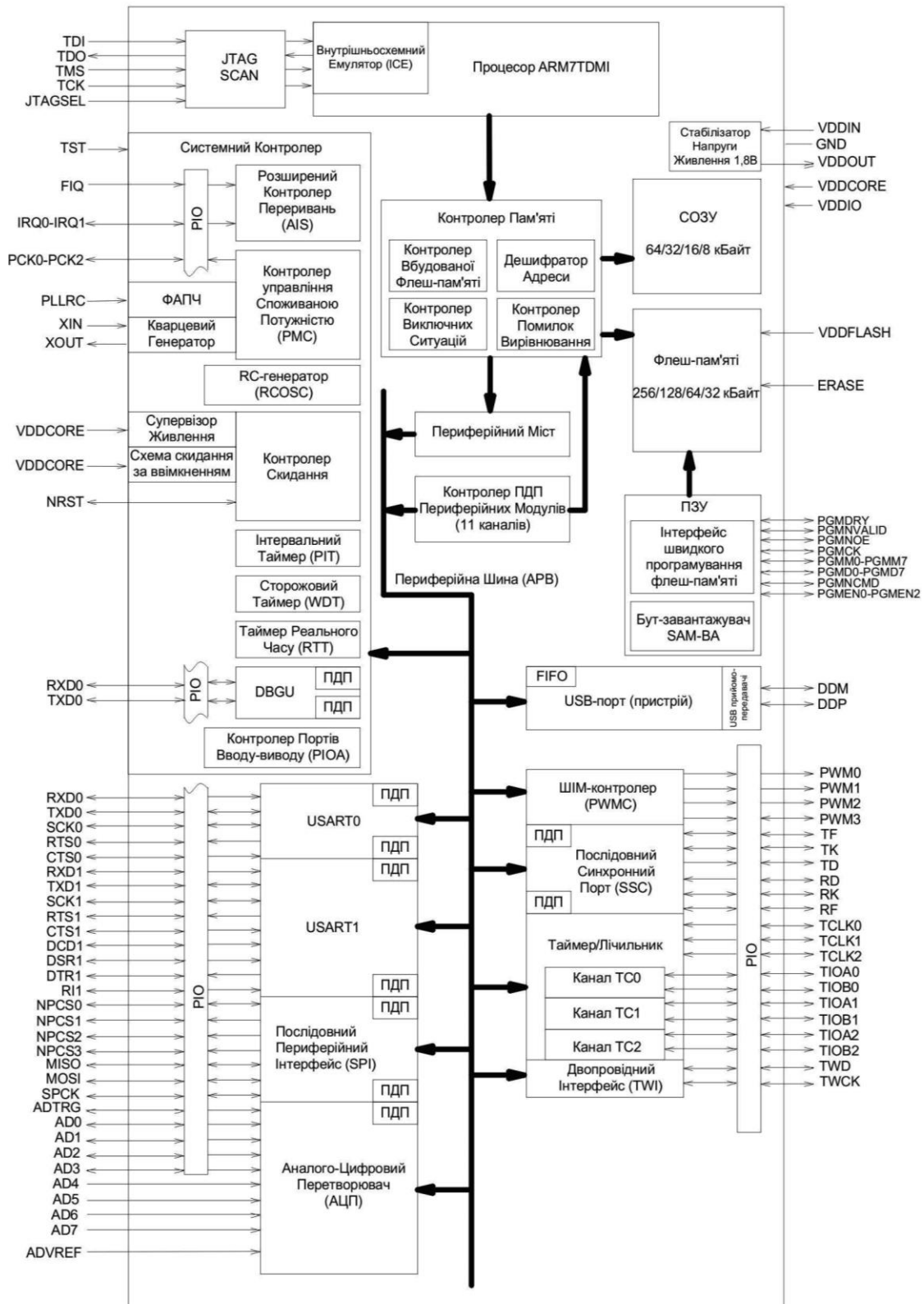


Рис. 8.2.

Коли розробник вмикає головний генератор (Main Oscillator), він повинен ініціалізувати його лічильник величиною, що відповідає часу запуску генератора. Цей час запуску залежить від частоти кварцу, підключеного до головного генератора. Коли біт MOSCEN і OSCOUNT записані в CKGR_MOR, щоб увімкнути головний генератор, біт MOSCS в PMC_SR (Status Register) очищується, і лічильник запускає відлік в зворотньому порядку від частоти повільного тактового генератора, поділеної на 8, починаючи від значення OSCOUNT. Так як значення OSCOUNT закодовано з 8 бітами, максимальний час запуску – приблизно 62 мс. Коли лічильник досягає 0, біт MOSCS встановлюється, що вказує на кінець стабілізації частоти головного генератора. Встановлення біта MOSCS в PMC_IMR може викликати переривання на процесор.

Є спосіб визначити частоту кварцу за допомогою Main Clock frequency counter. Коли біт MAINRDY в регістрі CKGR_MCFR (Main Clock Frequency Register) встановлений, поле MAINF регістру CKGR_MCFR показує, до якого числа лічильник встиг дорахувати за 16 періодів SLCK. Звичайно ж, значення тактової частоти визначається неточно через похибки частоти внутрішнього генератора RC.

Можна вимкнути кварц і подати зовнішню тактову частоту на ніжку XIN. Для того, щоб це працювало правильно, потрібно в регістрі CKGR_MOR встановити біт OSCBYPASS в 1, а біт MOSCEN скинути в 0.

Коли поле двигуна (DIV) встановлене у 0, вихід дільника і вихід PLL дають безперервний сигнал на рівні 0. Після скидання кожне поле DIV встановлюється в 0, таким чином відповідно вихідна тактова PLL встановлюється в 0. PLL дозволяє примножити вихідну частоту дільника. PLLCK (тактова частота, що виходить з PLL) має частоту, яка залежить від відповідної вихідної частоти сигналу (MAINCK) і параметрів DIV і MUL. Множник, що прикладається до вихідної частоти, рівний $(MUL + 1)/DIV$. Коли в MUL записано 0, PLL заборонено і споживана потужність зменшується. Увімкнути PLL можна, записавши в поле MUL величину, більшу від 0.

Незалежно від того, чи була PLL знову дозволена, чи були змінені її параметри, біт LOCK в регістрі PMC_SR автоматично очищується. Величини, записані в поле PLLCOUNT регістра CKGR_PLLR завантажуються в лічильник PLL. Потім лічильник PLL декрементується частотою SLCK, поки не досягає величини 0. В цей момент біт LOCK в PMC_SR встановлюється і може викликати переривання для процесора. Розробник повинен завантажити кількість циклів SLCK в поле PLLCOUNT, потрібну для перекриття часу перехідного процесу PLL. Початковий стан PLL і його цільової частоти може бути визначено, використовуючи спеціальний інструмент, який надає Atmel.

Power Management Controller (PMC). Потрібний для управління тактовими частотами як ядра процесора, так і його периферії, що впливає на енергоспоживання. Виробляє наступні тактові частоти:

– MCK Master Clock, основна частота, що подається на ті модулі процесора, які працюють постійно (не вимикаючись). Маже бути запрограмована на частоти від декількох сотень герц до максимальної тактової частоти пристрою. Обирається модулем PMC, при цьому в якості вихідних можуть братися SLCK, MAINCK, PLLCK.

– PCK Processor Clock – вимикається, коли процесор переходить в idle mode. Обирається модулем PMC.

– Peripheral Clocks тактова частота, що подається на вбудовану периферію (USART, SSC, SPI, TWI, TC, MCI, и т. п.). Зазвичай це MCK, але керується окремо. Обирається модулем PMC.

– UDPCCK USB Device Port Clock. Обирається модулем PMC.

– Programmable Clock Outputs, тактові сигнали, які можна виводити на ніжках PCKx. PMC (Power Management Controller) керує 4 сигналами, що програмуються на вихід ніжок PCKx. Кожен сигнал може бути незалежно запрограмований через регістри PMC_PCKx. Є також регістри PMC_SCER, PMC_SCDR, PMC_SCSR.

Вибір частоти MCK відбувається шляхом запису в поле CSS регістра PMC_MCKR необхідного значення. Далі обрана частота надходить на дільник, який встановлюється полем PRES того ж регістра PMC_MCKR. Кожен раз, коли в регістр PMC_MCKR записується нове значення, що визначає зміну MCK, в регістрі PMC_SR обнуляється біт MCKRDY, і далі читається як 0, поки частота MCK встановлюється. Після цього MCKRDY встановлюється і ця подія може викликати переривання процесора. Це корисно, коли відбувається перемикання з високої тактової частоти на більш низьку, щоб додаток міг коректно визначити момент реальної зміни тактової частоти.

Частота PCK (Processor Clock) може бути заборонена записом в регістр PMC_SCDR (System Clock Disable Register). Статус цієї частоти можна отримати в регістрі PMC_SCSR. PCK дозволяється при скиданні, і завжди автоматично дозволяється при будь-якому дозволеному перериванні (відбувається вихід процесору з режиму Idle). При забороні PCK перед входом в режим Idle поточна інструкція завершує своє виконання (перед тем, як такти зупиняться), однак це не відміняє передач даних між іншими майстрами на системній шині.

Внутрішня статична пам'ять – CO3Y (SRAM). До складу кожного мікроконтролера AT91SAM7X256/128 вбудована високошвидкісна статична пам'ять розміром 64/32 кБ відповідно. Одразу після скидання статична пам'ять доступна починаючи з адреси 0x0020 0000, але до виконання Remap SRAM (відображення статичної пам'яті на область розміщення флеш-пам'яті). Після виконання цієї команди статична пам'ять також стає доступною і за адресою 0x0000 0000.

Внутрішня перезаписувана пам'ять – ПЗП (ROM). До складу кожного мікроконтролера серії AT91SAM7S вбудована внутрішня енергозалежна одноразово програмована пам'ять – ПЗП (ROM), в якій розміщено інтерфейс для швидкого програмування флеш-пам'яті (FFPI) і

програма стартового бут-завантажувача SAM-BA, яка копіюється в два перших сектори флеш-пам'яті, і звідти ж виконується. По замовченню пам'ять ПЗП не відображена в область пам'яті мікроконтролера.

Контролер вбудованої флеш-пам'яті. Контролер вбудованої флеш-пам'яті (EFC) служить для обробки і виконання відповідних запитів з головних модулів мікроконтролера. Контролер дозволяє виконувати безпосереднє читання даних з флеш-пам'яті або збереження даних в буфері запису. Крім того, EFC має в своєму складі інтерфейс користувача, який на рівні з контролером пам'яті відображується на шині периферії (APB). Інтерфейс користувача дозволяє наступне:

- програмна зміна параметрів доступу до флеш-пам'яті (число тактів очікування, часові параметри і т.д.);
- виконання таких команд, як повне стирання, посторінкове стирання і програмування, установка/скидання NVM-бітів і т.д.;
- надання результату використання останньої команди, в тому числі помилок виконання;
- програмне налаштування переривань, що виникають одразу після успішного виконання останньої команди або при виникненні помилок в роботі флеш-пам'яті. Контролер вбудованої флеш-пам'яті також має в своєму складі подвійний 32-бітний буфер попередньої вибірки даних, який дозволяє оптимізувати операції 16-бітного доступу до флеш-пам'яті. Робота цього буфера особлива корисна під час роботи ядра процесора в режимі Thumb.

Контролер скидання. Контролер скидання оснований на схемі скидання при подачі живлення і супервізорі напруги живлення. Контролер скидання дозволяє зафіксувати джерело останнього скидання, що відбулося: скидання за ввімкненням живлення, програмне скидання, зовнішнє апаратне скидання, скидання від сторожового таймера, скидання від супервізора напруги живлення. На додаток до всього, контролер скидання керує дозволом і заборорою скидання від всіх внутрішніх джерел скидання і виводом NRST з відкритим стоком, формуючи на ньому імпульс низького логічного рівня визначеної тривалості.

8.2.2. Призначення виводів та живлення мікроконтролера

Кожен мікроконтролер з сімейства AT91SAM7S має шість типів виводів для підключення зовнішніх джерел живлення. Наявність вбудованого стабілізатора напруги дозволяє використовувати всього одне зовнішнє джерело для живлення мікроконтролера.

VDDIN: вхід вбудованого стабілізатора напруги і аналого-цифрового перетворювача. Допустимий діапазон напруг складає від 3,0В до 3,6В (номінал 3,3В).

VDDOUT: вихід вбудованого стабілізатора з вихідною напругою 1,8В.

VDDIO: живлення буферів портів вводу-виводу і вбудованих трансиверів USB порту. Реалізована підтримка двох рівнів живлення: від

3,0В до 3,6В (номінал 3,3В) і 1,65В...1,95В (номінал 1,8В). При напрузі живлення менше 3,0В робота USB трансиверів стає вже неможливою.

VDDFLASH: живлення вбудованої флеш-пам'яті. Наявність цього виводу необхідна для надійної роботи флеш-пам'яті. Допустимий діапазон напруг складає від 3,0В до 3,6В (номінал 3,3В).

VDDCORE: живлення ядра мікроконтролера і його внутрішніх логічних вузлів. Допустимий діапазон напруг складає від 1,65В до 1,95В (номінал 1,8В). Ці виводи можуть бути підключені до виводу VDDOUT з відповідним розв'язуючим конденсатором. Для нормальної роботи мікроконтролера необхідна наявність напруги на виводах VDDCORE.

VDDPLL: живлення внутрішнього тактового генератора і ФАПЧ (PLL). Цей вивід може бути підключений до виводу VDDOUT.

Для кожного з вказаних вище входів живлення існує один загальний вхід – виводи GND. Всі ці виводи необхідно з'єднувати з полігоном заземлення схеми в максимальній від нього близькості.

Кожен мікроконтролер з сімейства AT91SAM7S має вбудований стабілізатор напруги, який керується програмно через системний контролер. В звичайному режимі роботи стабілізатор споживає не більше 100 мкА і дозволяє живити навантаження з споживаним струмом до 100 мА.

Також стабілізатор напруги має можливість працювати в економічному режимі. В цьому режимі сам безпосередньо стабілізатор споживає не більше 25 мкА і дозволяє живити навантаження з споживаним струмом до 1 мА.

На виході стабілізатора напруги можливе виникнення пульсацій, що збільшує вірогідність збоїв в роботі тактового генератора. Щоб запобігти цьому в максимальній близькості від виводу VDDOUT потрібно розташувати керамічний конденсатор ємністю 470 пФ (або 1 нФ). Крім того, необхідна наявність танталового конденсатора X7R ємністю 2,2 мкФ (або 3,3 мкФ) між виводами VDDOUT і GND.

На вході стабілізатора напруги необхідно підключити між виводами VDDIN і GND керамічний конденсатор 0,1 мкФ і танталовий 4,7 мкФ в максимальній близькості від цих виводів з метою збільшення стабільності запуску мікроконтролера і зниження рівня провалів напруги на виході стабілізатора.

Внутрішній стабілізатор напруги підключається до цього джерела і формує на своєму виході напругу, яка подається на виводи VDDCORE і VDDPLL. На рис. 8.3 приведена умовна схема живлення при використанні в якості джерела USB порт комп'ютера.

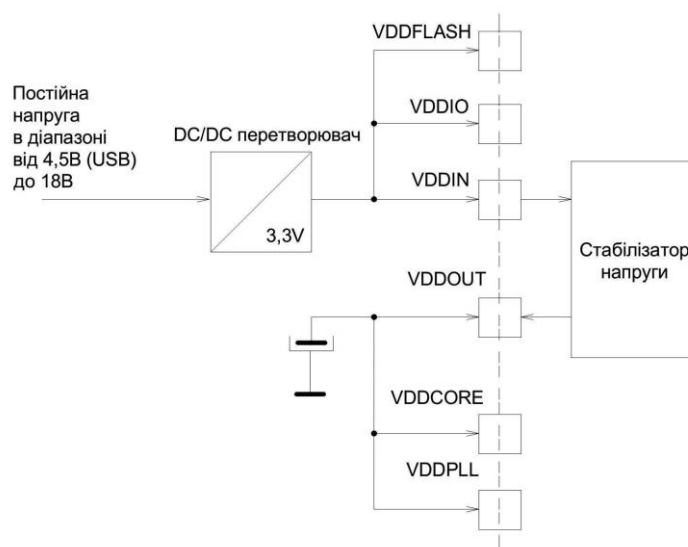


Рис. 8.3.

8.3. Робота з середовищем розробки Keil μ Vision

Середовище розробки для мікропроцесорів ARM7 і ARM9 Keil μ Vision включає компілятор Сі та відлагоджувач. Середовище підтримує більшість процесорів всіх виробників. В порівнянні з компілятором IAR, середовище Keil набагато легше налаштовується. Код, що створюється, більш оптимальний. Програмний пакет розробки Realview Microcontroller Development Kit (MDK ARM) компанії Keil об'єднує компілятор C/C++ ARM Realview та інтегроване середовище розробки Keil μ Vision. Цей продукт надає в розпорядження розробника багатофункціональне середовище розробки, оптимізоване для роботи з широкою лінійкою мікроконтролерів на базі ядра ARM. MDK ARM забезпечує підтримку пристроїв, що базуються на ARM7, ARM9 і Cortex M3 таких виробників, як Analog Devices, Atmel, Freescale, Luminary, OKI, NXP, Samsung, Sharp, Stmicroelectronics і TI.

Для створення нового проекту в меню Project потрібно вибрати пункт New μ Vision Project. Необхідно вказати шлях розташування і задати назву проекту. Потім обирається ядро AT91SAM7X256 і натискається кнопка OK (рис. 8.4).

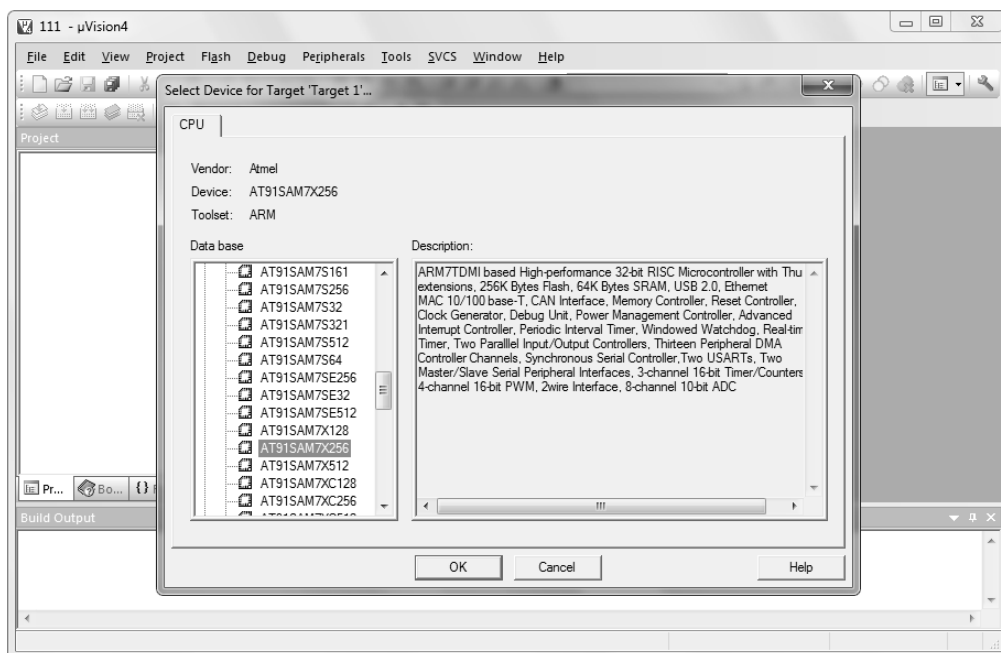


Рис. 8.4.

Компілятор запропонує автоматично створити файл низькорівневої ініціалізації Startup. У вікні Project створюємо нову папку Source Code. Потім створюємо новий файл – меню File/New, зберігаємо його з розширенням .с. Потім додаємо його в папку Source Code (рис. 8.5).

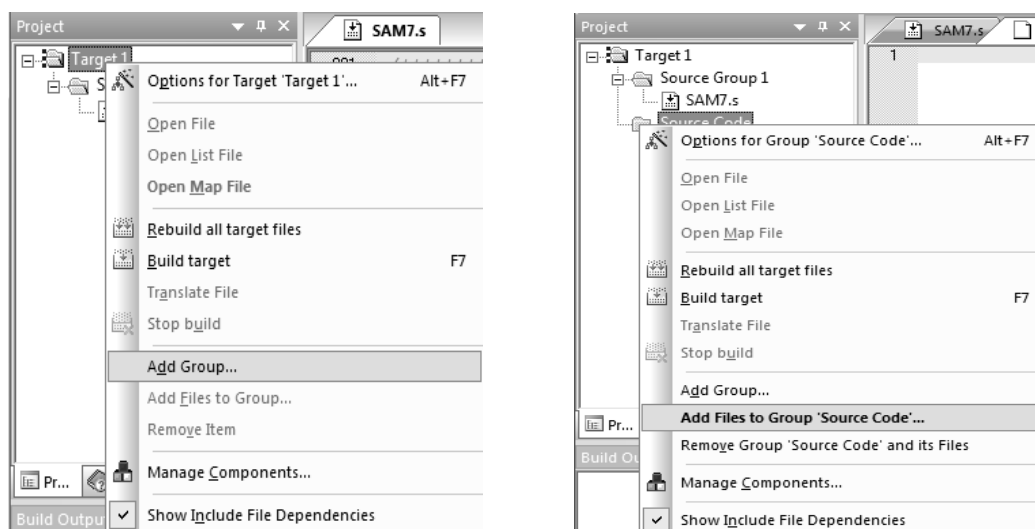


Рис. 8.5.

Потім підключаємо програматор JetLink8 і встановлюємо драйвери (рис. 8.6).

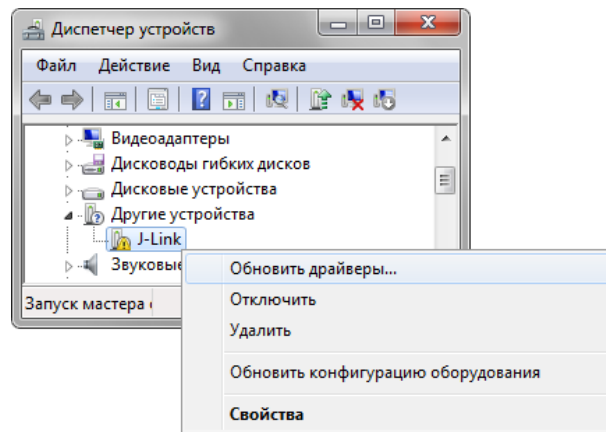


Рис. 8.6.

Вказуємо шлях розташування драйверів і підтверджуємо установку (рис. 8.7).

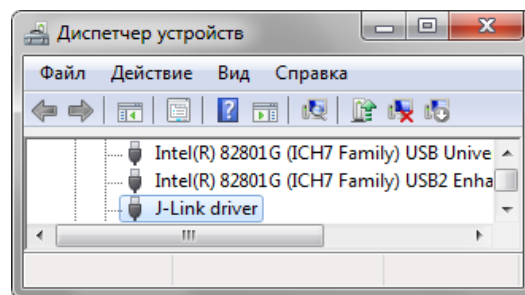


Рис. 8.7.

Після цього необхідно підключити програматор до середовища Keil. Для цього заходимо в меню Project/Options for Target (рис. 8.8 – рис. 8.10).

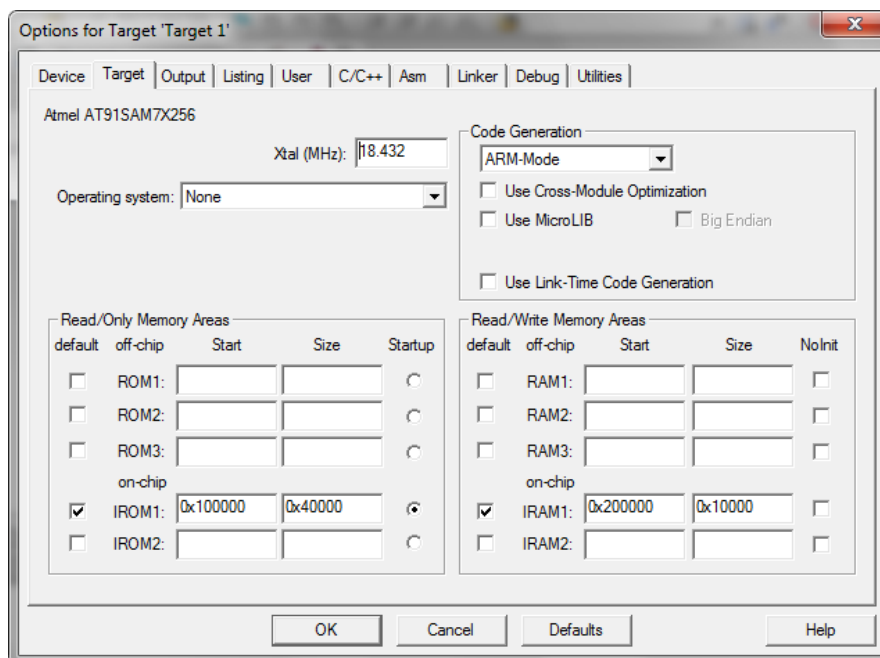


Рис. 8.8.

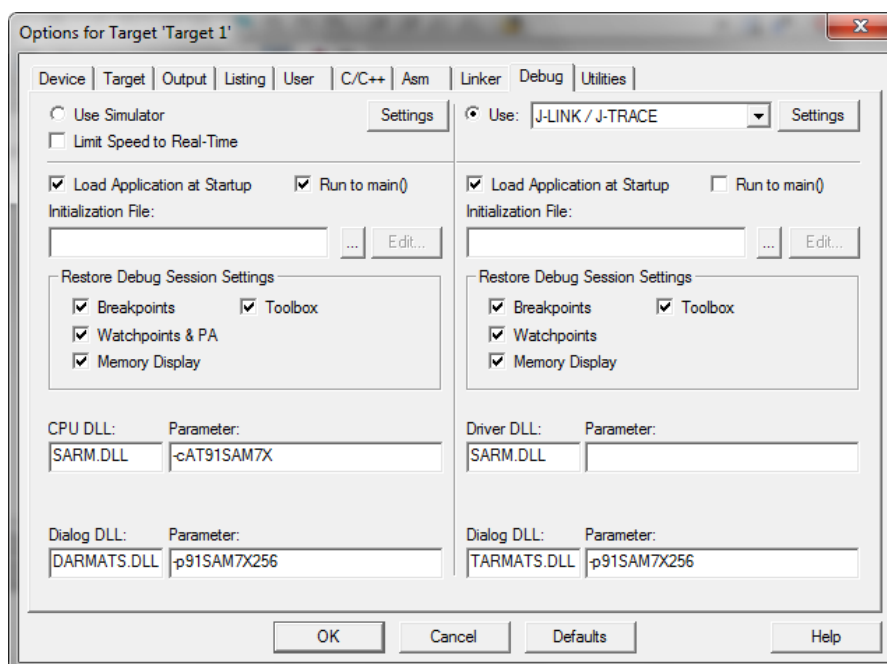


Рис. 8.9.

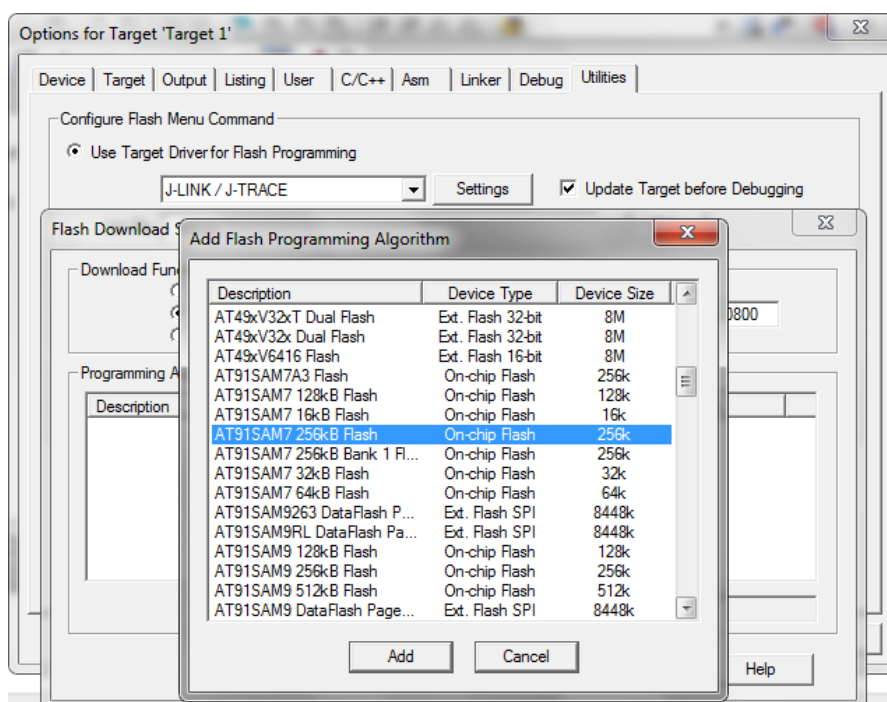


Рис. 8.10.

Вкладка Target: в списку Code Generation обираємо тип команд ARM–Mode. Якщо використовується операційна система реального часу (RTOS, uLinux и т.п.) то в списку Operating system необхідно обрати RTX Kernel. Вкладка Debug: В списку відлагоджувачів, що використовується, обираємо J–Link/J–Trace.

Вкладка Utilities: в списку програматорів обираємо J–Link/J–Trace. В налаштуваннях Settings необхідно додати алгоритм завантаження файлу прошивки в мікропроцесор – Add/AT91SAM7 256kB Flash/Add.

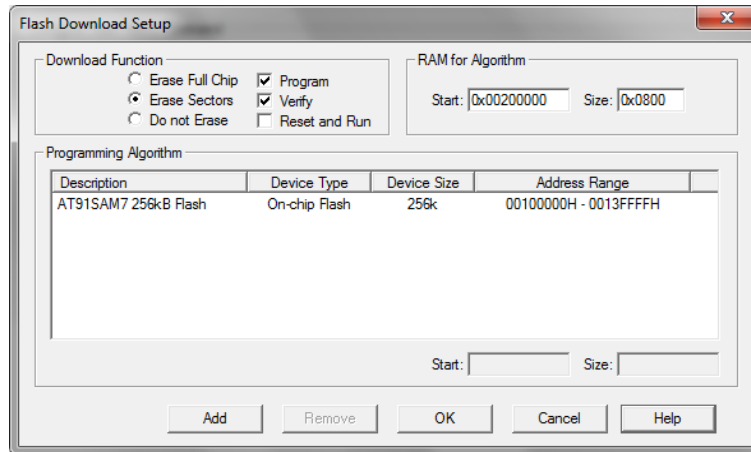


Рис. 8.11.

8.4. Периферійні модулі 32-розрядних мікроконтролерів

Для кожного периферійного модуля в адресному просторі процесора відведена область в пам'яті розміром по 16 кБ.

Особливістю мікроконтролерів сімейства AT91SAM7S є наявність контролера портів паралельного вводу-виводу (PIO), порти якого мультиплексовані з вбудованими периферійними модулями (табл. 8.1).

Контролер PIO керує 32 портами (21 порт в AT91SAM7S32). Кожен порт може бути програмно налаштований для роботи з одним із двох периферійних модулів, або працювати як звичайний порт цифрового вводу-виводу. Деякі з цих портів мають можливість роботи в якості аналогових входів вбудованого модуля АЦП.

Табл. 8.1.

Назва сигналу	Призначення	Тип	Адресний рівень	Примітки
Живлення				
VDDIN	Вхід внутрішнього стабілізатора напруги живлення модуля АЦП	Живлення		3,0В...3,6В
VDDOUT	Вихід стабілізації напруги	Живлення		1,85В (номінал)
VDDFLASH	Живлення флеш-пам'яті	Живлення		3,0В...3,6В
VDDIO	Живлення буферів портів вводу-виводу	Живлення		3,0В...3,6В або 1,65В...1,95В
VDDCORE	Живлення ядра	Живлення		1,65В...1,95В
VDDPLL	Живлення ФАПЧ (PLL)	Живлення		1,65В...1,95В
GND	Земля	Земля		
Тактові сигнали, генератори і ФАПЧ				
XIN	Вхід внутрішнього кварцового генератора	Вхід		

XOUT	Вихід внутрішнього кварцового генератора	Вихід		
PLLRC	Фільтр ФАПЧ (PLL)	Вхід		
PCKO-PCK2	Програмований вихід тактової частоти	Вихід		
ICE і JTAG				
TCK	Текстовий сигнал JTAG	Вхід		Підтягуючий резистор відсутній
TDI	Вхід послідовних даних JTAG	Вхід		Підтягуючий резистор відсутній
TDO	Вихід послідовних даних JTAG	Вихід		
TMS	Вибір режиму JTAG	Вхід		Підтягуючий резистор відсутній
JTAGSEL	Дозвіл роботи JTAG	Вхід		Підтягуючий резистор до GND
Флеш-пам'ять				
ERASE	Команда очистки деяких конфігураційних бітів з флеш-пам'яті	Вхід	Високий	Підтягуючий резистор до GND
Скидання/Тест				
NRST	Скидання мікроконтролера	Вхід/вихід	Низький	Вихід з відкритим стоком і підтягуючим резистором до VDDIO
TST	Ввімкнення режиму тестування	Вхід	Високий	Підтягуючий резистор до GND
Модуль внутрішньосхемної відладки				
DRXD	Прийом даних в режимі відладки	Вхід		
DTXD	Передача даних в режимі відладки	Вихід		
AIS				
IRQ0-IRQ1	Виходи зовнішніх переривань (IRQ)	Вхід		IRQ1 відсутній в AT91SAM7S32
FIQ	Вхід для виклику швидкого переривання (FIQ)	Вхід		
PIO				
PA0 – PA31	Паралельний порт вводу-виводу А	Вхід/вихід		Підтягуючі резистори до VDDIO за скиданням (PA0-PA20 в AT91SAM7S32)
USB порт				
DDM	USB дані-	Аналоговий		Відсутній в

				AT91SAM7S32
DDP	USB дані+	Аналоговий		Відсутній в AT91SAM7S32
USART				
SCKO-SCK1	Тактовий сигнал (в синхронному режимі)	Вхід/вихід		SCK1 відсутній в AT91SAM7S32
TXDO-TXD1	Дані, що передаються	Вхід/вихід		TDX1 відсутній в AT91SAM7S32
RXDO-RXD1	Дані, що приймаються	Вхід		RXD1 відсутній в AT91SAM7S32
RTSO-RTS1	Запит на передачу даних до модему	Вихід		RTS1 відсутній в AT91SAM7S32
CTSO-CTS1	Готовність модему до передачі даних	Вхід		CTS1 відсутній в AT91SAM7S32
DCD1	Контроль наявності несучої модему	Вхід		відсутній в AT91SAM7S32
DTR1	Готовність до прийому даних від модему	Вихід		відсутній в AT91SAM7S32
DSR1	Готовність модему до прийому даних	Вхід		відсутній в AT91SAM7S32
RI1	Індикатор виклику (індикатор дзвоника)	Вхід		відсутній в AT91SAM7S32
Контролер послідовного синхронного порту				
TD	Дані, що передаються	Вихід		
RD	Дані, що приймаються	Вхід		
TK	Тактовий сигнал для даних, що передаються	Вхід/вихід		
RK	Тактовий сигнал для даних, що приймаються	Вхід/вихід		
TF	Синхронізація кадру при передачі	Вхід/вихід		
RF	Синхронізація кадру при прийомі	Вхід/вихід		
Таймер/лічильник				
TCLKO-TCLK2	Зовнішні входи тактового сигналу	Вхід		TCLK1 і TCLK2 відсутній в AT91SAM7S32
TIOAO-TIOA2	Порт А вводу-виводу	Вхід/вихід		TIOA2 відсутній в AT91SAM7S32
TIOB0-TIOB2	Порт В вводу-виводу	Вхід/вихід		TIOB2 відсутній в AT91SAM7S32
ШІМ контролер				
PWMO-PWM3	Канали ШІМ	Вихід		

SPI				
MISO	Вхід ведучого і вихід веденого	Вхід/вихід		
MOSI	Вихід ведучого і вхід веденого	Вхід/вихід		
SPCK	Тактовий сигнал SPI	Вхід/вихід		
NPCSO	Вибір чипу SPI-CS0	Вхід/вихід	Низький	
NPCS1-NPCS3	Вибір чипу SPI-CS1- CS3	Вихід	Низький	
Двохпровідний інтерфейс				
TWD	Дані, що передаються/ дані, що приймаються	Аналоговий		
TWCK	Тактовий сигнал для даних, що передаються/ даних, що приймаються	Аналоговий		
Аналого-цифровий перетворювач				
AD0-AD3	Аналогові входи	Аналоговий		Підтягуючий резистор VDDIO. Ввімкнені як цифрові входи за скиданням.
AD4-AD7	Аналогові входи	Аналоговий		Аналогові входи
ADTRG	Запуск перетворення АЦП	Вхід		
ADVREF	Опорне джерело для АЦП	Аналоговий		
Інтерфейс швидкого програмування флеш-пам'яті				
PGMEN0-PGMEN2	Дозвіл програмування	Вхід		
PGMM0-PGMM3	Режим програмування	Вхід		
PSMD0-PGMD15	Дані для програмування	Аналоговий		PGMD0-PGMD7 тільки в AT91SAM7S32
PGMDY	Готовність до програмування	Вихід	Високий	
PGMNVALID	Напрямок передачі даних	Вихід	Низький	
PGMNOE	Дозвіл читання даних	Вхід	Низький	
PGMCK	Тактовий сигнал для читання/запису даних	Вхід		
PGMNCMD	Команда програмування	Вхід	Низький	

8.4.1. Контролери паралельного вводу-виводу (PIO)

Для мікроконтролерів з архітектурою ARM та мікроконтролерів AVR, PIC, MCS51 і подібних до них, можна провести аналогію між портами вводу/виводу PORTA, PORTB і контролерами PIOA, PIOB. Вони займаються однаковою роботою – виводять дані на виводи мікросхеми. Однак їх внутрішній устрій значно відрізняється. На рис. 8.12 представлена структура контролера PIO.

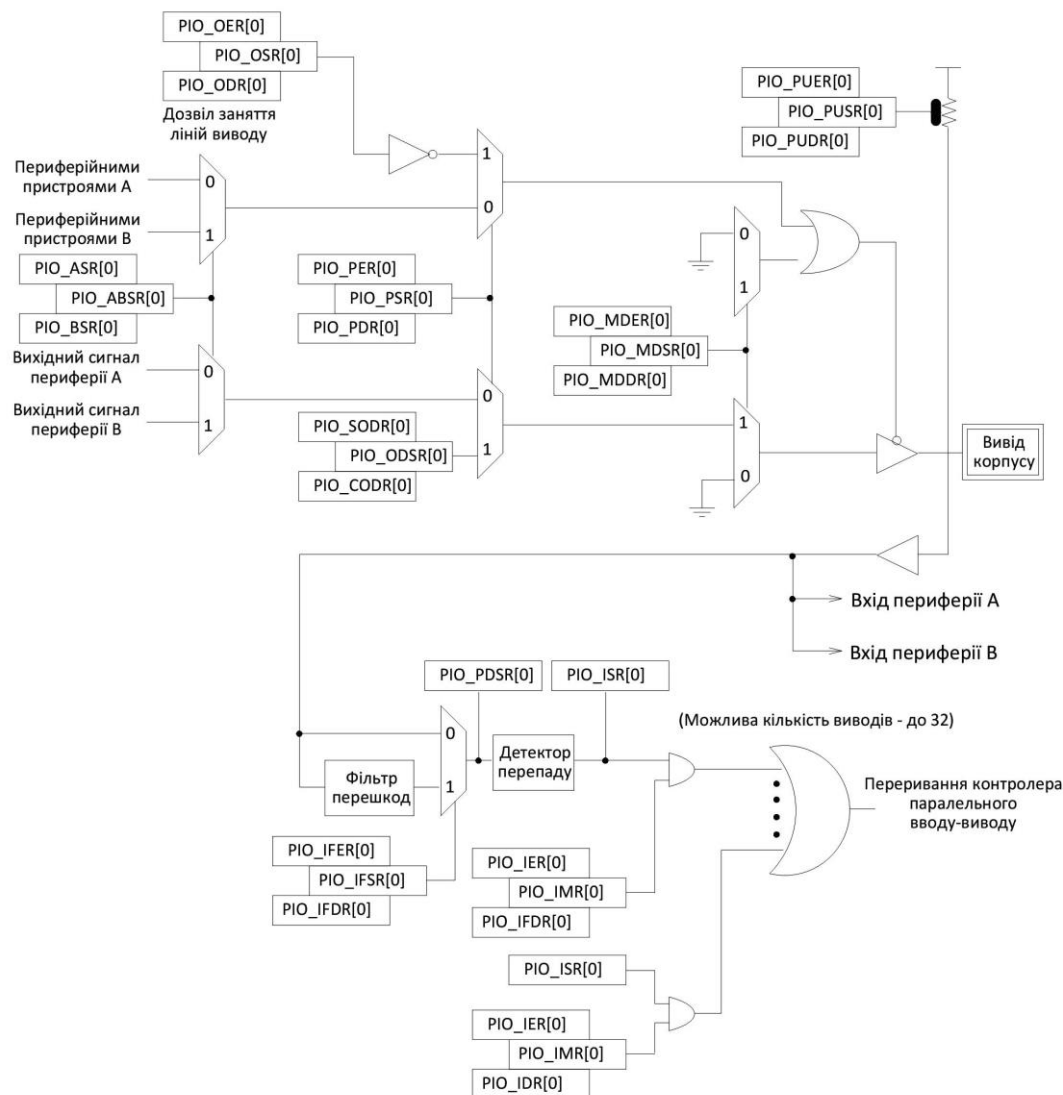


Рис. 8.12.

Кожна лінія вводу-виводу має вбудований підтягуючий резистор. Його номінал складає близько 10 кОм. У випадку, якщо лінія вводу-виводу об'єднана з двома або одною лініями периферійного пристрою, необхідно встановити режим роботи лінії (периферія або ввід/вивід). Для керування лініями вводу/виводу використовуються наступні регістри:

- PIO_OER – регістр дозволу введення;
- PIO_OWER – регістр завершення запису даних;
- PIO_SODR – регістр встановлення стану виведення;
- PIO_ODSR – регістр стану виводів;
- PIO_CODR – регістр скидання стану виведення;

PIO_PDSR – реєстр стану ніжки;
PIO_IFER – реєстр фільтрації перешкод;
PMC_PCER – реєстр дозволу тактування;
AT91C_BASE_PIOB – вказує на базову адресу PIOB (0xFFFFF600).

Приклад 8.1. Програма реалізує «хвилю, що біжить» на світлодіодах. Для роботи з 32-бітовими значеннями, особливо коли необхідно використовувати 1 або 2 виводи порту, зручним є запис вигляду (1<<32). Світлодіоди підключені катодами до виводів процесора.

```
#include <AT91SAM7X256.H> // підключення бібліотеки визначників МП
// визначники для світлодіодів
#define LED1 (1<<19)
#define LED2 (1<<20)
#define LED3 (1<<21)
#define LED4 (1<<22)

void delay(int cl) // функція затримки
{
    int i,i1=0;
    for(i=0;i<cl;i++)
    {
        for(i1=0;i1<1000;i1++) {}
    }
}

int main (void)
{
    AT91C_BASE_WDTC->WDTC_WDMR=0; // відключення сторожового таймера
    AT91C_BASE_PIOB->PIO_OER=LED1|LED2|LED3|LED4; // реєстр дозволу введення
    AT91C_BASE_PIOB->PIO_OWER=LED1|LED2|LED3|LED4; // реєстр дозволу запису
    вихідних даних (ODSR)
    while(1)
    {
        AT91C_BASE_PIOB->PIO_ODSR=~LED1; // управління LED4
        delay(1000); // затримка
        AT91C_BASE_PIOB->PIO_ODSR=~LED2; // управління LED3
        delay(1000); // затримка
        AT91C_BASE_PIOB->PIO_ODSR=~LED3; // управління LED2
        delay(1000); // затримка
        AT91C_BASE_PIOB->PIO_ODSR=~LED4; // управління LED1
        delay(1000); // затримка
    }
}
```

Для компіляції проекту натисніть F7. Для перекомпіляції всіх ресурсів проекту зайдіть в меню Project/Rebuild all target files. Для завантаження файлу прошивки зайдіть в меню Flash/Download. Після програмування мікроконтролера його необхідно перезавантажити (RESET).

Для виведення даних на лінію необхідно кожного разу записувати і скидати 2 регістри – PIO_SODR і PIO_CODR, що зручно для модулюючих сигналів (меандр). Однак при виведенні більшої кількості даних на лінії виводу можуть виникати короточасні імпульси, що збивають роботу підключених компонентів, через постійне перемикання між 2 регістрами. Для уникнення подібних імпульсів і збільшення пропускну здатності необхідно виводити дані безпосередньо через статус регістр ODSR (синхронне виведення). Для отримання доступу до регістру ODSR необхідно активізувати біти регістра дозволу запису виходу OWER. Під час ініціалізації процесора бажано відключити сторожовий таймер, так як неодноразово було помічено порушення в його роботі на ARM7TDMI від Atmel.

Приклад 8.2. Програма зчитує стан 3 кнопок і активує світлодіоди при їх натисненні. Кнопки підтягнуті до напруги 3,3 В через резистори номіналом 100 кОм. Детектування натискання відбувається за низьким рівнем сигналу.

```
#include <AT91SAM7X256.H> // підключення бібліотеки визначників МП
// визначники для світлодіодів
#define LED1 (1<<19)
#define LED2 (1<<20)
#define LED3 (1<<21)
#define LED4 (1<<22)
// визначники для кнопок
#define SW1 (1<<27)
#define SW2 (1<<28)
#define SW3 (1<<29)

int main (void)
{
    AT91C_BASE_WDTC->WDTC_WDMR=0; // відключення сторожового таймера
    AT91C_BASE_PIOB->PIO_OER=LED1|LED2|LED3|LED4; // регістр дозволу
    виведення
    AT91C_BASE_PIOB->PIO_OWER=LED1|LED2|LED3|LED4; // регістр завершення
    запису вихідних даних (ODSR)
    AT91C_BASE_PIOB->PIO_IFER=SW1|SW2|SW3; // регістр фільтрації вхідних
    перешкод (тремтіння контакту)
    AT91C_BASE_PIOB->PIO_ODSR=LED1|LED2|LED3|LED4; // загасити світлодіоди
    // дозвіл тактування PIO для виводів порту B
    AT91C_BASE_PMC->PMC_PCER = 1 << AT91C_ID_PIOB;
    while(1)
    {
        // при натисненні на кнопку, запалюється 1 світлодіод
        if((AT91C_BASE_PIOB->PIO_PDSR&SW1)==0)
            AT91C_BASE_PIOB->PIO_ODSR=~LED1;
        // при натисненні на кнопку, запалюється 2 світлодіод
        if((AT91C_BASE_PIOB->PIO_PDSR&SW2)==0)
            AT91C_BASE_PIOB->PIO_ODSR=~LED2;
        // при натисненні на кнопку, запалюється 3 світлодіод
        if((AT91C_BASE_PIOB->PIO_PDSR&SW3)==0)
```

```
        AT91C_BASE_PIOB->PIO_ODSR=~LED3;  
    }  
}
```

Для зчитування стану ніжки необхідно увімкнути тактування контролера PIO. При скиданні мікропроцесора тактування периферійних модулів вимкнене, тактування необхідно вмикати вручну для кожного окремо взятого модуля. Це зроблено для зниження споживаної потужності і підвищення перешкодостійкості кристала. Тактування дозволяється шляхом запису бітів в регістр дозволу тактування PMC_PCER. Регістр IEFER дозволяє фільтрувати тремтіння контакту при натисканні на кнопку, що зменшує кількість використання зовнішніх компонентів. Частота фільтрації складає половину періоду задаючої частоти, що цілком достатньо для зовнішніх механічних контактів.

8.4.2. Робота з контролером переривань (AIC)

Кожен вивід МК може бути сконфігурований як лінія вводу/виводу, або як лінія периферійного пристрою. Переривання може бути оброблено від будь якого виводу PIO МК. Однак всі вони об'єднані логічним OR і мають лише одне переривання в розширеному контролері переривань AIC. На розширеному контролері переривань варто зупинитись детальніше. Розширений контролер переривань AIC є 8-ми рівневим векторним контролером, з можливістю індивідуального маскування. Контролер переривань керує входами nFIQ (fast interrupt request – запит швидкого переривання) і входами nIRQ (standard interrupt request – запит стандартного переривання). Входи nFIQ і nIRQ є входами ARM ядра. Для стандартних задач використовуються IRQ переривання. FIQ переривання використовуються у випадку необхідності максимально швидкого виконання обробки переривань. На рис. 8.13 представлена структурна схема контролера переривань.

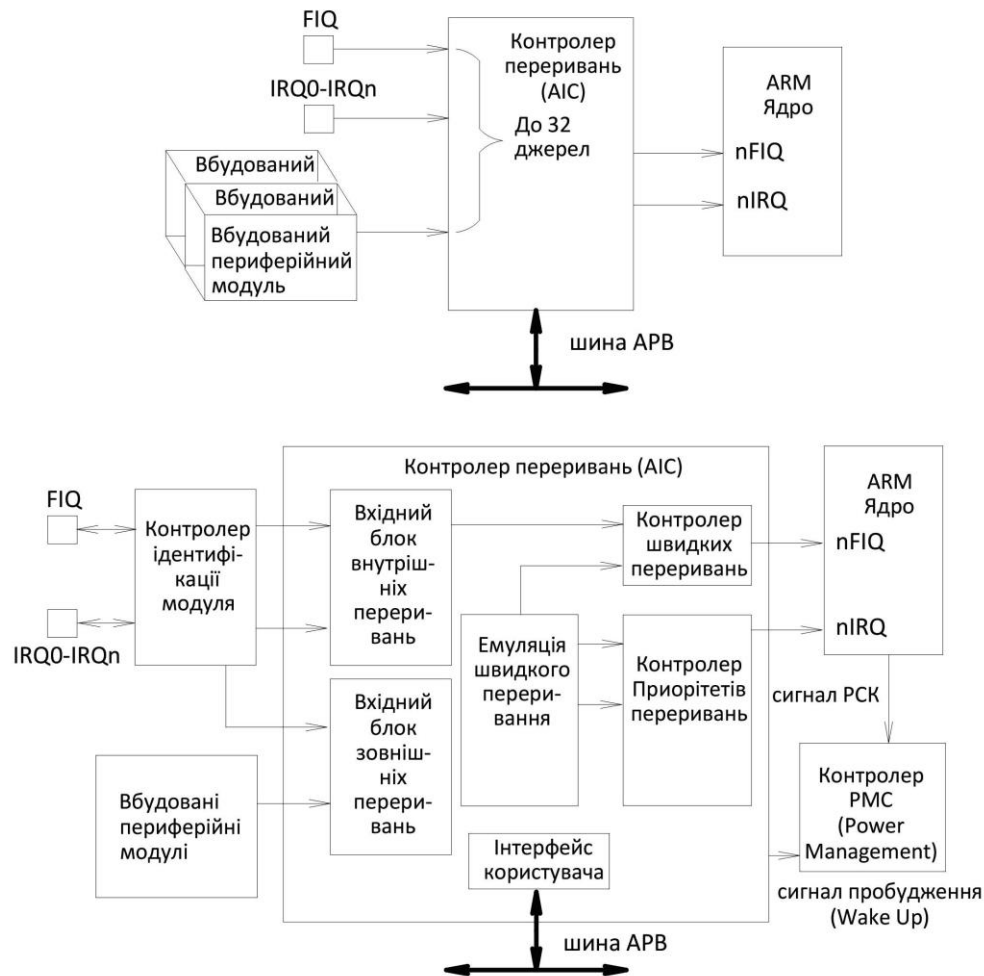


Рис. 8.13.

Внутрішні джерела переривань (рис. 8.14) можуть бути запрограмовані на спрацювання як за рівнем сигналу, так і за його фронтом. Для внутрішніх джерел переривань рекомендується використовувати спрацювання за рівнем сигналу. Обробник IRQ переривання повинен починатися з ключового слова `__irq`. При викликанні обробника переривання необхідно рахувати Status регістр використовуваного периферійного модуля, щоб скинути необхідні прапори переривань. По завершенні обробника переривання необхідно записати будь-яке значення в регістр завершення переривань EOICR. Частіше всього в регістр EOICR записують 0, так як сам факт запису даних сигналізує контролеру AIC про завершення переривання.

Восьмирівневий контролер пріоритетів дозволяє встановити пріоритет для кожного джерела переривання, що дозволяє використовувати систему вкладених переривань. Якщо на даний момент оброблюється низькопріоритетне переривання і надходить сигнал на обробку переривання з більш високим пріоритетом, то номер поточного переривання записується в апаратний стек, виконується запис до регістру EOICR (скидається поточне переривання), і виконується високопріоритетне переривання. Після чого зчитується адреса обробника

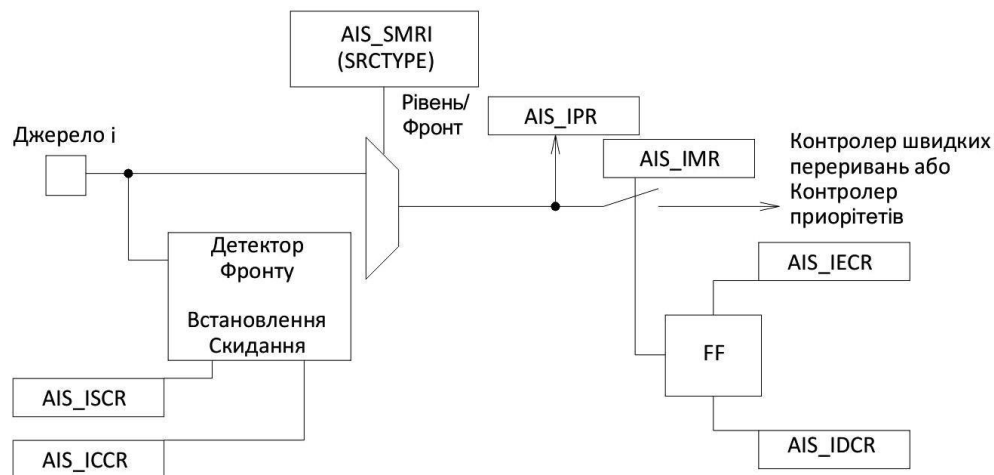


Рис. 8.14.

Приклад 8.3. Програма обробляє 2 переривання з ліній вводу–виводу РІО, веде відлік і виводить його на світлодіоди.

```
#include <AT91SAM7X256.H> // підключення бібліотеки визначників МП
volatile unsigned char incr=0; // змінна, що зберігає кількість відліків
// визначники для кнопок
#define SW1 (1 << 27) // кнопка 1
#define SW2 (1 << 28) // кнопка 2
// визначники для світлодіодів
#define LED1 (1<<19)
#define LED2 (1<<20)
#define LED3 (1<<21)
#define LED4 (1<<22)

__irq void PIO_interrupt(void) // обробник переривання
{
    volatile unsigned long pio_isr = AT91C_BASE_PIOB->PIO_ISR; // зчитування
PIO_ISR, визначення натисненої кнопки
    // відлік натискань
    if(pio_isr==SW1) incr++;
    if(pio_isr==SW2) incr--;
    // виведення на світлодіоди
    if(incr==1) AT91C_BASE_PIOB->PIO_ODSR=~LED1;
    if(incr==2) AT91C_BASE_PIOB->PIO_ODSR=~LED2;
    if(incr==3) AT91C_BASE_PIOB->PIO_ODSR=~LED3;
    if(incr==4) AT91C_BASE_PIOB->PIO_ODSR=~LED4;
    // обмеження
    if(incr>4) incr=0;
    if(incr<0) incr=0;
    AT91C_BASE_AIC->AIC_EOICR = 0; // завершення переривання
}
```

```

int main(void)
{
// ініціалізація
AT91C_BASE_WDTC->WDTC_WDMR=0;    // вимкнення сторожового таймера
AT91C_BASE_AIC->AIC_IDCR=0xFFFFFFFF; // вимкнення переривань
AT91C_BASE_PMC->PMC_PCER = 1 << AT91C_ID_PIOB; // ввімкнення тактування
порту B
// ініціалізація лінії виведення
AT91C_BASE_PIOA->PIO_ODR = SW1|SW2; // відключення виводу
AT91C_BASE_PIOA->PIO_PER = SW1|SW2; // дозвіл PIO
// виводи під світлодіоди
AT91C_BASE_PIOB->PIO_OER= LED1|LED2|LED3|LED4;
AT91C_BASE_PIOB->PIO_OWER= LED1|LED2|LED3|LED4;
AT91C_BASE_PIOB->PIO_ODSR= LED1|LED2|LED3|LED4;
AT91C_BASE_AIC->AIC_IDCR=SW1|SW2;    // вимкнення переривань
AT91C_BASE_PIOB->PIO_ISR;            // скидання PIO_ISR
// ініціалізація переривання
AT91C_BASE_AIC->AIC_SVR[AT91C_ID_PIOB] = (unsigned int)PIO_interrupt;
// адреса обробника
AT91C_BASE_AIC->AIC_SMR[AT91C_ID_PIOB] =
(AT91C_AIC_SRCTYPE_INT_HIGH_LEVEL|AT91C_AIC_PRIOR_LOWEST);
// високий рівень сигналу спрацьовування, низький пріоритет
AT91C_BASE_AIC->AIC_ICCR=SW1|SW2;    // очищення переривань
AT91C_BASE_PIOB->PIO_IER = SW1|SW2;    // дозвіл переривань PIO
AT91C_BASE_AIC->AIC_IECR = 1 << AT91C_ID_PIOB; // дозвіл переривань в AIC
while(1) {};
}

```

Регістри, що використовуються при роботі з контролером переривань:

- PIO_ODR – реєстр вимкнення ліній PIO в якості ліній виводу;
- PIO_PER – реєстр дозволу використання виводів;
- PIO_ISR – статус реєстр PIO;
- PIO_IER – реєстр дозволу переривань PIO;
- AIC_SMR – реєстр джерела переривання;
- AIC_SVR – реєстр, що повертає процесору адресу обробника переривання;
- AIC_IECR – реєстр дозволу переривань;
- AIC_IDCR – реєстр вимкнення переривань;
- AIC_ICCR – реєстр очищення переривань;
- AIC_EOICR – реєстр завершення переривань.

Для визначення джерела переривання PIO необхідно зчитати вміст Status-реєстра в змінну, потім порівнюємо визначення виводу, з якого надійшло переривання. Переривання PIO завжди спрацьовує за будь-якої зміни стану виводу. Можливо тільки програмно перевірити, чи прийшов потрібний фронт. Переривання PIO є внутрішніми. Для використання "реальних" зовнішніх переривань необхідно використовувати виводи IRQx кристалу. Змінні, що використовуються в обробниках переривань,

необхідно оголошувати з ключовим словом `volatile`. Це дозволить вимикати оптимізацію компілятора відносно даних змінних і уникнути невірної роботи з ними. Для отримання переривань від ліній вводу–виводу РІО також необхідно увімкнути тактування РІО.

Для ініціалізації переривань РІО необхідно виконати наступні операції:

- вимкнення сторожового таймера;
- ввімкнення тактування РІО;
- вимкнення ліній РІО в якості ліній виводу;
- дозвіл використання ліній для переривань;
- вимкнення переривань на час ініціалізації;
- скидання Status регістра;
- вказується адреса обробника переривання;
- вказується логічний рівень сигналу спрацювання і пріоритет переривання;
- очищуються обрані переривання;
- дозволяються переривання в РІО;
- дозволяються переривання в АІС.

8.4.3. Робота з аналого–цифровим перетворювачем (ADC)

АЦП виконаний за принципом послідовних наближень і має 10–розрядну дозволяючу здатність. Він також інтегрує 8–вхідний мультиплексор, який дозволяє виконати перетворення одного з 8 входів. Діапазон перетворення АЦП складає 0В...ADVREF.

АЦП підтримує два режими перетворення: 8–розрядний і 10–розрядний, а результат перетворення розміщується в загальному регістрі для всіх каналів, а також в окремому для кожного каналу регістрі. Передбачена можливість вибору джерел запуску перетворення: програмний запуск, зовнішній запуск за зростаючим фронтом на виводі ADTRG або внутрішнє перезавантаження вхідними сигналами таймера–лічильника. При роботі з АЦП використовуються наступні регістри:

ADC_SR – статус регістр

ADC_CR – регістр управління

ADC_CDRx – регістр каналу

ADC_MR – регістр режиму

ADC_CHER – регістр дозволу каналу

ADC_IER – регістр дозволу переривань

AT91C_ID_ADC – ідентифікатор, що вказує периферійний номер.

Структура АЦП представлена на рис. 8.15.

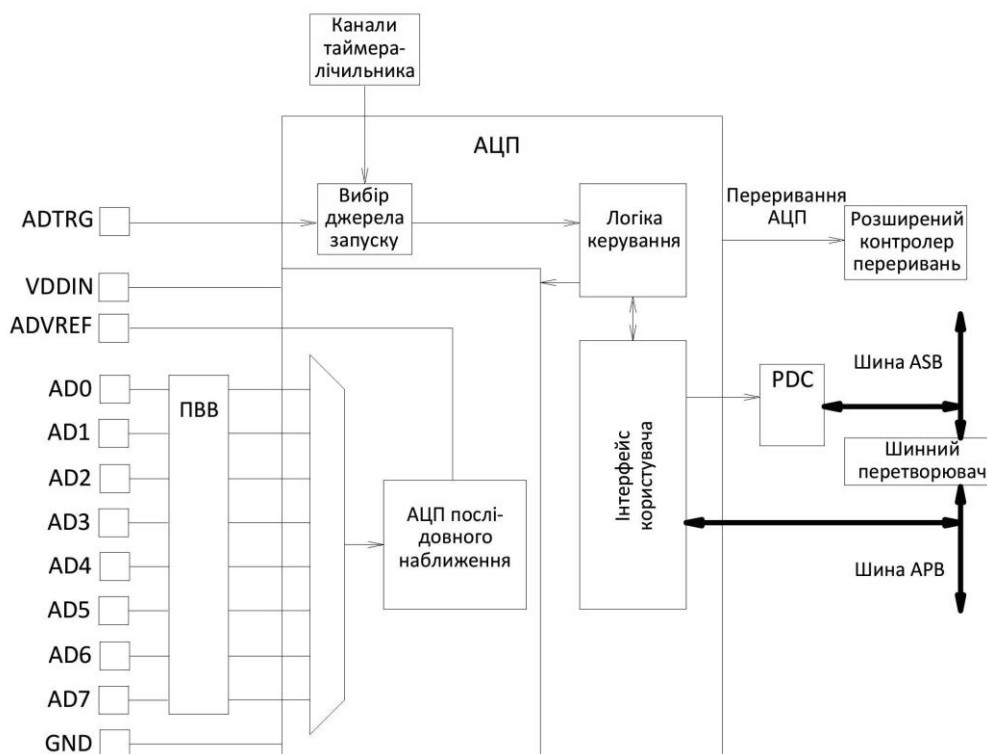


Рис. 8.15.

Приклад 8.4. Програма використовує АЦП для обробки зовнішнього сигналу. Частота опитування сигналу ~ 100 кГц.

```
#include <AT91SAM7X256.H>
// визначники каналу
#define ADC_CHN_0 0x01
#define ADC_CHN_1 0x02
#define ADC_CHN_2 0x04
#define ADC_CHN_3 0x08
#define ADC_CHN_4 0x10
#define ADC_CHN_5 0x20
#define ADC_CHN_6 0x40
#define ADC_CHN_7 0x80

AT91PS_ADC pADC = AT91C_BASE_ADC;
AT91PS_PMC pPMC = AT91C_BASE_PMC;

// визначники для регістру режиму TC_CMR
#define TC_CLKS_MCK 0x2 // дільник частоти 32
#define TC_CPCTRG 0x4000 // дозвіл перезавантаження при співпадінні з RC
// визначники для регістру управління TC_CCR
#define TC_CLKEN 0x1 // біт дозволу синхронізації лічильника
#define TC_CLKDIS 0x2 // біт вимкнення синхронізації лічильника
#define TC_SWTRG 0x4 // біт програмного перезавантаження

extern unsigned int adc_data=0; // змінна, що зберігає оцифроване значення

__irq void timer0 (void) // обробник переривання
```

```
{
    *AT91C_TC0_SR;          // зчитування Status регістру
    pADC->ADC_CR = 0x2;
    *AT91C_AIC_EOICR = 0; // дозвіл переривання
}

__irq void ADC_interrupt(void)    // обробник переривання
{
    pADC->ADC_SR;          // скидання Status Registr
    adc_data = pADC->ADC_CDR7; // зчитуємо значення
    *AT91C_AIC_EOICR=0;    // завершення переривання
}

void init_timer0(void)
{
    AT91C_BASE_WDTC->WDTC_WDMR=0;    // вимкнення сторожового таймера
    AT91C_BASE_AIC->AIC_IDCR=0xFFFFFFFF; // вимкнення переривань
    *AT91C_TC0_SR; // скидання статус регістра
    AT91C_BASE_PMC->PMC_PCER = (1<<AT91C_ID_TC0); // ввімкнення тактування
    таймера-лічильника
    AT91C_BASE_AIC->AIC_SMR[AT91C_ID_TC0] =
    AT91C_AIC_SRCTYPE_INT_HIGH_LEVEL| AT91C_AIC_PRIOR_LOWEST; // за
    високим рівнем, низький пріоритет
    AT91C_BASE_AIC->AIC_SVR[AT91C_ID_TC0] = (unsigned int) timer0; // адреса
    обробника переривання
    *AT91C_TC0_CMR = TC_CLKS_MCK|AT91C_TC_CPCTRG ; // дільник частоти 1024,
    перезавантаження за співпадінням RC
    *AT91C_TC0_RC = 0x000F; // значення RC для порівняння
    *AT91C_TC0_CCR = AT91C_TC_CLKEN|AT91C_TC_SWTRG ; // ввімкнення таймера,
    перезапуск таймера
    AT91C_BASE_AIC->AIC_ICCR = (1<<AT91C_ID_TC0) ; // очищення переривання TC0
    *AT91C_TC0_IER = AT91C_TC_CPCS ; // дозвіл переривання за співпадінням з RC
    AT91C_BASE_AIC->AIC_IECR = (1<<AT91C_ID_TC0) ; // дозвіл переривання TC0
}

void init_ADC(void) // ініціалізація АЦП
{
    pPMC->PMC_PCER = 1 << AT91C_ID_ADC; // встановлюємо тактування
    // скидання ADC
    pADC->ADC_CR = 0x1;
    pADC->ADC_CR = 0x0;          // встановити на максимум startup time і hold time
    pADC->ADC_MR = 0x0F1F0F1D; // дозволяємо всі потрібні канали
    pADC->ADC_CHER = ADC_CHN_7;
    AT91C_BASE_AIC->AIC_SMR[AT91C_ID_ADC]=
    AT91C_AIC_SRCTYPE_INT_HIGH_LEVEL|1;
    // пріоритет 1, переривання за рівнем
    AT91C_BASE_AIC->AIC_SVR[AT91C_ID_ADC]=(unsigned long)ADC_interrupt;
    // адреса обробника
    AT91C_BASE_AIC->AIC_IECR=(1<<AT91C_ID_ADC);
    //дозволяємо переривання в AIC
    pADC->ADC_IER= ADC_CHN_7; //дозволяємо переривання в IER
}
```

```

int main (void)
{
    init_timer0();
    init_ADC();
    while (1);    // обробляємо і передаємо оцифровані дані
}

```

Для забезпечення режиму неперервного перетворення в ARM7 Atmel, необхідно встановлювати прапорець START вручну. Для цього було використано таймер–лічильник, який в обробнику переривання скидає АЦП і встановлює прапор START для наступного перетворення. Для зменшення часу обробки перетворень використовувалось переривання АЦП. Також потрібно враховувати частоту синхронізації АЦП і обраний інтервал таймера–лічильника, його необхідно підбирати максимально близьким. Необхідно враховувати кількість тактів, потрібних для звернення до оцифрованих даних. Якщо опитування займає тривалий час, то на момент зчитування оцифрованих даних в регістрах можуть знаходитися викривлені значення. Для забезпечення збереження даних використовуються startup time і hold time, що забезпечують час обробки і утримання даних. Також необхідно ввімкнути тактування АЦП. На рис. 8.16 представлені часові затримки, що виникають при звертанні до АЦП.



Рис. 8.16.

8.4.4. Модуль широтно-імпульсної модуляції (PWM)

Модуль ШІМ (рис. 8.17) незалежно керує 4 каналами. Кожен з каналів генерує прямокутні імпульси на одному своєму виході. Параметри прямокутних імпульсів, такі як період, заповнення імпульсів і полярність конфігуруються через призначений для користувача інтерфейс. Кожен канал використовує один із сигналів синхронізації, що генеруються тактовим генератором (який саме сигнал обирається користувачем). Сигнали синхронізації, що формуються тактовим генератором, отримуються в результаті ділення сигналів головної синхронізації модуля ШІМ. Канали ШІМ можуть бути синхронізовані між собою, щоб генерувати імпульсні послідовності, що взаємно не перекриваються (рис. 8.18). Кожен канал має подвійну систему буферизації для запобігання

генерації непередбачуваних імпульсних послідовностей при програмній зміні режиму роботи.

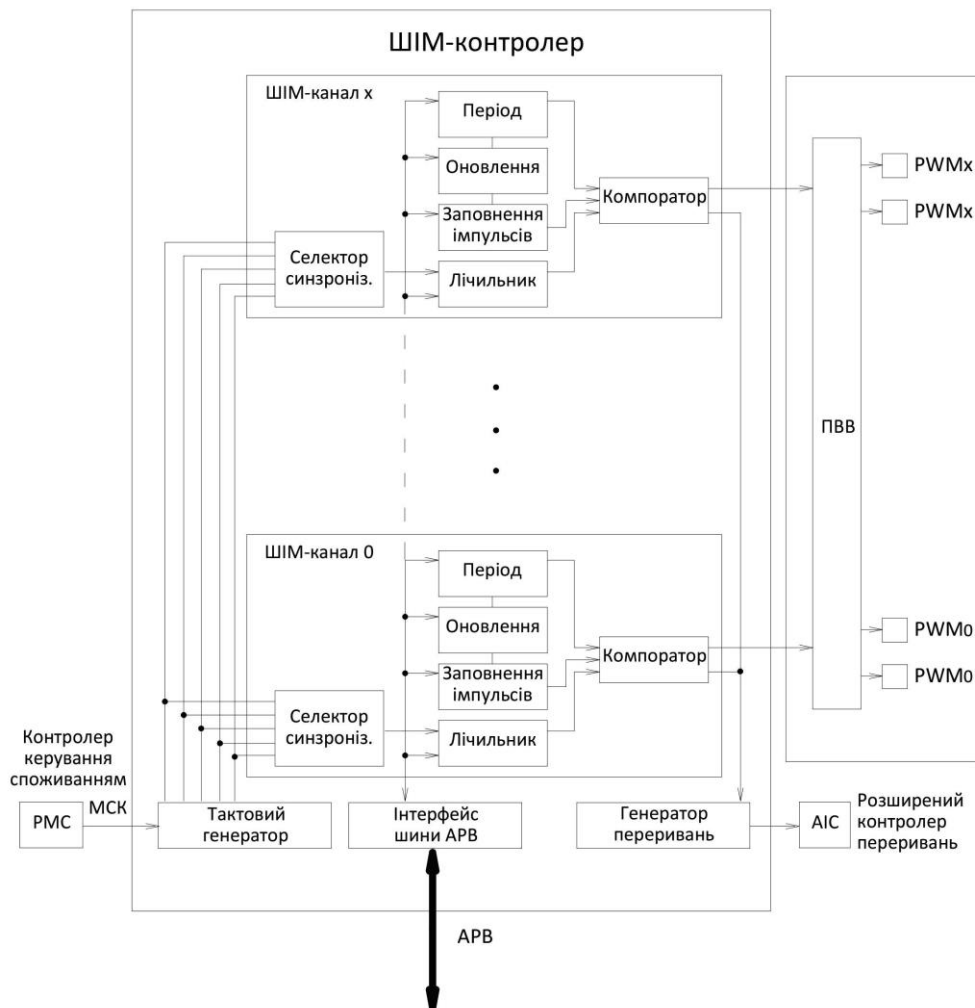


Рис. 8.17.

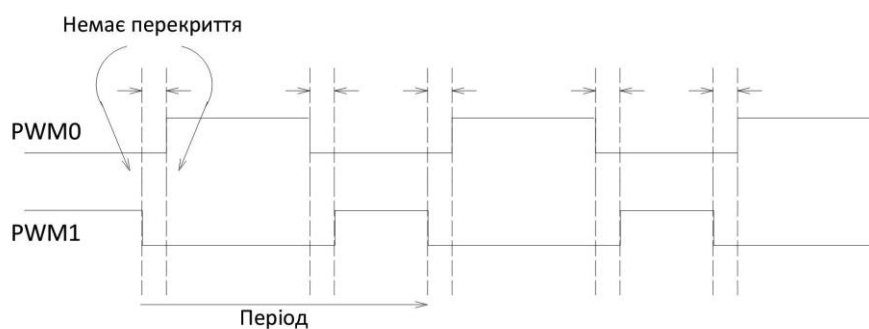


Рис. 8.18.

При роботі з модулем ШІМ використовуються наступні регістри:
 PPMC_ISR – регістр статусу переривання ШІМ;
 PPMC_SR – статус регістр;
 PPMC_MR – регістр режиму ШІМ;
 PPMC_CH0_CPRDR – регістр періоду каналу 0 (частота);
 PPMC_CH0_CRTYR – регістр робочого циклу каналу 0;

PWMC_ENA – регістр дозволу ШІМ;
 PWMC_IER – регістр дозволу переривання ШІМ;

Приклад 8.5. Програма використовує PWM контролер для формування зовнішнього управляючого ШІМ-сигналу.

```
#include <AT91SAM7X256.H>    // підключення бібліотек визначників МП
volatile unsigned char incr=0; // змінна, що зберігає кількість відліків
// визначники для світлодіодів
#define LED1 (1<<19)
#define LED2 (1<<20)
#define LED3 (1<<21)
#define LED4 (1<<22)
// визначники для PWM
#define CH0 1                // перший канал
#define DIVA 1               // коефіцієнт поділу CLKA
#define PREA (0xA<<8)       // коефіцієнт поділу тактової частоти МП (CLKA)
#define CPRE 0xA             // дільник частоти каналу
#define AT91C_PB_PWM0 (1<<27) // вивід PWM0

__irq void pwm0 (void)       // обробник переривання
{
    *AT91C_PWMC_ISR;        // зчитування Status регістра
    incr++; // відлік
    // виведення на світлодіоди
    if(incr==1) AT91C_BASE_PIOB->PIO_ODSR=~LED1;
    if(incr==2) AT91C_BASE_PIOB->PIO_ODSR=~LED2;
    if(incr==3) AT91C_BASE_PIOB->PIO_ODSR=~LED3;
    if(incr==4) {AT91C_BASE_PIOB->PIO_ODSR=~LED4;incr=0;}
    *AT91C_AIC_EOICR = 0; // завершення переривання
}

int main()
{
    // ініціалізація виводів під світлодіоди
    AT91C_BASE_PIOB->PIO_OER=LED1|LED2|LED3|LED4;
    AT91C_BASE_PIOB->PIO_OWER=LED1|LED2|LED3|LED4;
    AT91C_BASE_PIOB->PIO_ODSR=LED1|LED2|LED3|LED4;
    // ініціалізація виводу каналу ШІМ
    AT91C_BASE_PIOB->PIO_PDR = AT91C_PB_PWM0; // вимкнення виводу в PIO
    AT91C_BASE_PIOB->PIO_ASR = AT91C_PB_PWM0; // вимкнення виводу в периферії
    AT91C_BASE_WDTC->WDTC_WDMR=0; // вимкнення сторожового таймера
    AT91C_BASE_AIC->AIC_IDCR=0xFFFFFFFF; // вимкнення переривань
    *AT91C_PWMC_SR; // скидання статус регістра
    AT91C_BASE_PMC->PMC_PCER = (1<<AT91C_ID_PWM0); // ввімкнення тактування
    AT91C_BASE_AIC->AIC_SMR[AT91C_ID_PWM0] =
    AT91C_AIC_SRCTYPE_INT_HIGH_LEVEL| AT91C_AIC_PRIOR_LOWEST;
    // за високим рівнем, низький пріоритет
    AT91C_BASE_AIC->AIC_SVR[AT91C_ID_PWM0] = (unsigned int) pwm0;
    // адреса обробника переривання
    *AT91C_PWMC_MR = DIVA|PREA; // встановлення коефіцієнтів поділу частоти
```

```

*AT91C_PWMC_CH0_CMR = CPRE;    // встановлення дільника частоти каналу
*AT91C_PWMC_CH0_CPRDR = 0x000008FF;
// значення періоду пересування імпульсів каналу ШІМ
*AT91C_PWMC_CH0_CDTYR = 0x0000002F; // заповнення імпульсами каналу ШІМ
*AT91C_PWMC_ENA = CH0;          // дозвіл роботи каналу ШІМ
AT91C_BASE_AIC->AIC_ICCR = (1<<AT91C_ID_PWMC);
// очищення переривання ШІМ
*AT91C_PWMC_IER = CH0;          // дозвіл переривання каналу ШІМ
AT91C_BASE_AIC->AIC_IECR = (1<<AT91C_ID_PWMC); // дозвіл переривання ШІМ
while (1);
}

```

Для роботи ШІМ-модуля необхідно ввімкнути його тактування. Потім потрібно виставити коефіцієнт поділу частоти (відносно тактової частоти МП), встановити дільник частоти каналу, встановити частоту ШІМ-сигналу, встановити скважність сигналу і дозволити роботу каналу ШІМ. Для використання виводів МП в якості формуючих зовнішні сигнали ШІМ-модуля необхідно підключити задані виводи в периферії А. Часова діаграма, що представлена на рис. 8.19, демонструє властивості імпульсів, що генерує модуль ШІМ.

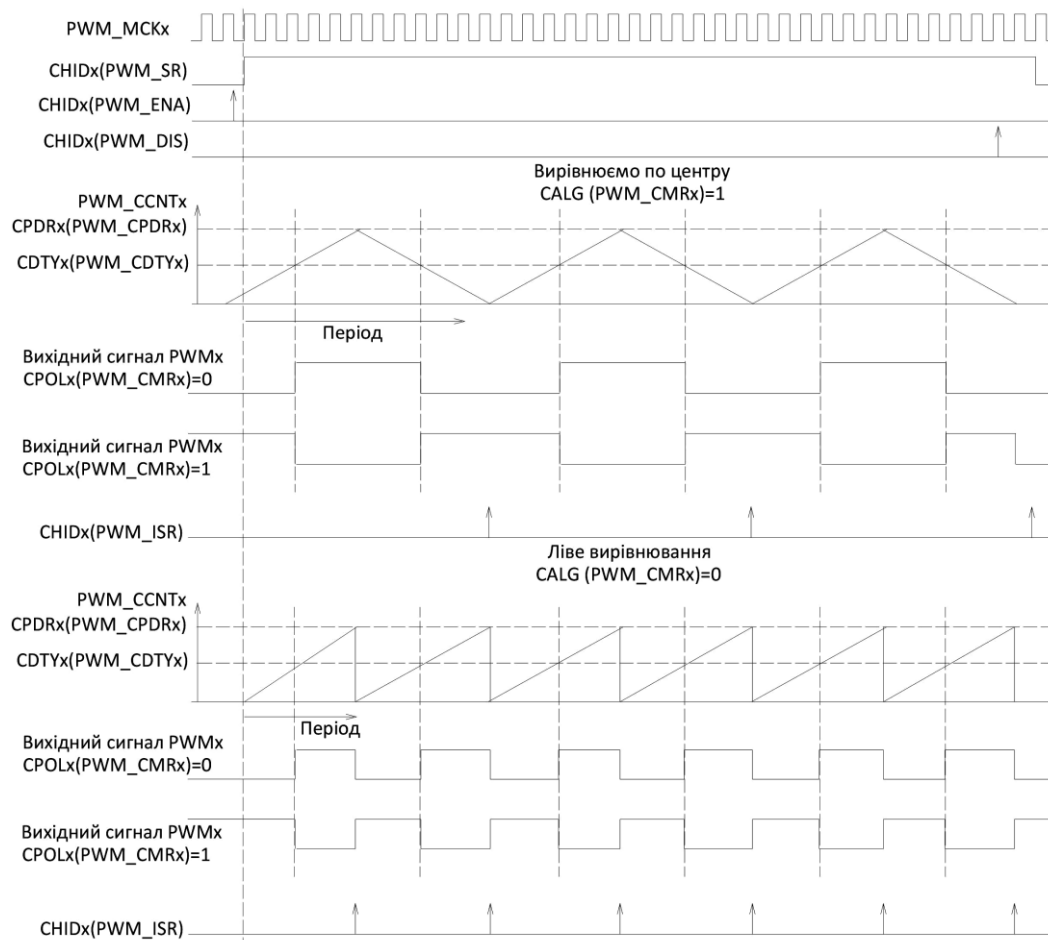


Рис. 8.19.

8.4.5. Таймери–лічильники (PIT, RTC, TCNT)

PIT – ПІТ, періодичний інтервальний таймер – використовується для операційних систем, так як має широкий часовий діапазон. Частота тактування ПІТ складає 1/16 від тактової частоти ядра (використовується передділник 16). Лічильник ПІТ має довжину в 20 біт. Відповідно, для отримання інтервалу 100ms необхідно розрахувати значення PIV.

$$48\text{МГц (ядро)}/16 = 3\text{МГц (PIO)}.$$

$$\text{PIV} = 0,1 \text{ (100мс)} \cdot 3\text{МГц (PIO)} = 300000 \text{ відліків} - 0\text{x}000493\text{E}0 \text{ (hex)}.$$

Регістри, що використовуються для налаштування періодичного інтервального таймера:

PITS – прапор переривання;

PICNT – лічильник переповнення PIV;

PIVR – регістр значення ПІТ (при читанні значення відбувається скидання PITS, PICNT);

PIIR – регістр поточного значення лічильника (при читанні скидання PITS, PICNT не відбувається);

PIMR – регістр режиму ПІТ (19–0 біт – значення PIV, що використовується для порівняння з відліком).

Приклад 8.6. Програма використовує Періодичний інтервальний таймер (PIT) для отримання часового інтервалу ~100ms.

```
#include <AT91SAM7X256.H>    // підключення бібліотеки визначників МП

volatile unsigned char incr=0;    // змінна, що зберігає кількість відліків
// визначники для світлодіодів
#define LED1 (1<<19)
#define LED2 (1<<20)
#define LED3 (1<<21)
#define LED4 (1<<22)

__irq void PIT_interrupt(void)    // обробник переривання
{
    incr++;                        // відлік переривань
    *AT91C_PITC_PIVR;             // скидання PICNT, скидання прапора переривання
    // PITS – читання регістру значення PIVR

    if(incr==1) AT91C_BASE_PIOB->PIO_ODSR=~LED1;
    if(incr==2) AT91C_BASE_PIOB->PIO_ODSR=~LED2;
    if(incr==3) AT91C_BASE_PIOB->PIO_ODSR=~LED3;
    if(incr==4){AT91C_BASE_PIOB->PIO_ODSR=~LED4; incr=0;}
    *AT91C_AIC_EOICR=0;           // завершення переривання
}

void PIT_init(void)               // ініціалізація ПІТ
{
    AT91C_BASE_AIC->AIC_IDCR=0xFFFFFFFF; // вимкнення переривань
    AT91C_BASE_AIC->AIC_SMR[AT91C_ID_SYS]=AT91C_AIC_SRCTYPE_INT_HIGH_LEVEL
|AT91C_AIC_PRIOR_LOWEST; // за високим рівнем, низький пріоритет
```

```
AT91C_BASE_AIC->AIC_SVR[AT91C_ID_SYS]=(unsigned long)PIT_interrupt; // адреса
обробника
AT91C_BASE_AIC->AIC_IECR=(1<<AT91C_ID_SYS); //дозволяємо переривання
*AT91C_PITC_PIMR= 0x000493E0; //встановлюємо значення порівняння PIV
*AT91C_PITC_PIMR|=AT91C_PITC_PITEN|
AT91C_PITC_PITIEN;          //дозволяємо переривання і запускаємо таймер
}

int main(void)
{
AT91C_BASE_WDTC->WDTC_WDMR=0;
PIT_init();
// ініціалізація виводів під світлодіоди
AT91C_BASE_PIOB->PIO_OER=LED1|LED2|LED3|LED4;
AT91C_BASE_PIOB->PIO_OWER=LED1|LED2|LED3|LED4;
AT91C_BASE_PIOB->PIO_ODSR=LED1|LED2|LED3|LED4;
while(1) {};
}
```

RTT – таймер реального часу, використовується для відліку секунд. При зчитуванні RTSR, прапори скидаються тільки через декілька циклів роботи процесора. Якщо обробник переривання має малий розмір (наприклад містить тільки інкремент секунд), то необхідно ввести невелику затримку (до 1мкс, декілька пустих операцій) перш ніж починати виконання основної програми, так як у деяких випадках можуть виникати збої основної програми (помічено лише для даної плати).

При частоті повільного тактового сигналу 32768 Гц значення лічильника дорівнює 0x00008000. Основні регістри таймера реального часу:

RTTC_RTISR – статус регістр RTT;
RTTC_RTMR – регістр режиму роботи RTT;
AT91C_RTTC_RTINCIEN – визначник дозволу переривання;
AT91C_RTTC_RTTRST – визначник перезапуску таймера.

Приклад 8.7. Програма використовує Таймер Реального часу (RTT) для отримання часового інтервалу 1с.

```
#include <AT91SAM7X256.H> // підключення бібліотеки визначників МП
volatile unsigned char incr=0; // змінна, що зберігає кількість відліків
// визначники для світлодіодів
#define LED1 (1<<19)
#define LED2 (1<<20)
#define LED3 (1<<21)
#define LED4 (1<<22)

__irq void RTT_interrupt(void) // обробник переривання
{
    *AT91C_RTTC_RTISR; // зчитування Status-регістру
    incr++;            // відлік
    // виведення на світлодіоди
}
```

```

    if(incr==1) AT91C_BASE_PIOB->PIO_ODSR=~LED1;
    if(incr==2) AT91C_BASE_PIOB->PIO_ODSR=~LED2;
    if(incr==3) AT91C_BASE_PIOB->PIO_ODSR=~LED3;
    if(incr==4) {AT91C_BASE_PIOB->PIO_ODSR=~LED4;incr=0;}
    *AT91C_AIC_EOICR = 0; // завершення переривання
}

int main(void)
{
    // ініціалізація
    AT91C_BASE_AIC->AIC_IDCR=0xFFFFFFFF; // вимкнення переривань
    AT91C_BASE_WDTC->WDTC_WDMR=0; // вимкнення сторожового таймера
    AT91C_BASE_AIC->
    AIC_SMR[AT91C_ID_SYS]=AT91C_AIC_SRCTYPE_INT_POSITIVE_EDGE
    |AT91C_AIC_PRIOR_LOWEST; // за фронтом, низький пріоритет
    // адреса обробника
    AT91C_BASE_AIC->AIC_SVR[AT91C_ID_SYS] = (unsigned int)RTT_interrupt;
    // дозвіл переривання AIC
    AT91C_BASE_AIC->AIC_IECR = (0x01 << AT91C_ID_SYS);
    *AT91C_RTTC_RTMR = 0x00008000 | AT91C_RTTC_RTTINCIEN |
    AT91C_RTTC_RTTRST; // встановлення значення лічильника, дозвіл переривання,
    перезапуск таймера
    // виведення на світлодіоди
    AT91C_BASE_PIOB->PIO_OER=LED1|LED2|LED3|LED4;
    AT91C_BASE_PIOB->PIO_OWER=LED1|LED2|LED3|LED4;
    AT91C_BASE_PIOB->PIO_ODSR=LED1|LED2|LED3|LED4;
    while(1);
}

```

Таймер-лічильник TCNT (рис. 8.20) має широку область застосування в електронних пристроях. Починаючи від отримання часових інтервалів, підрахунку подій, виміру частоти, генерації сигналів, і закінчуючи синхронізацією внутрішніх затримок пристрою, визначенням зсуву фаз і широтно-імпульсною модуляцією. Модуль таймера-лічильника має 3 незалежних один від одного 16-бітних канали. Кожен канал має три зовнішніх, п'ять внутрішніх джерел синхронізації і містить окреме джерело переривань в розширеному контролері переривань AIC. Таймер-лічильник може працювати в 2 режимах – режим захоплення і режим генерації, і має 2 глобальних регістри.

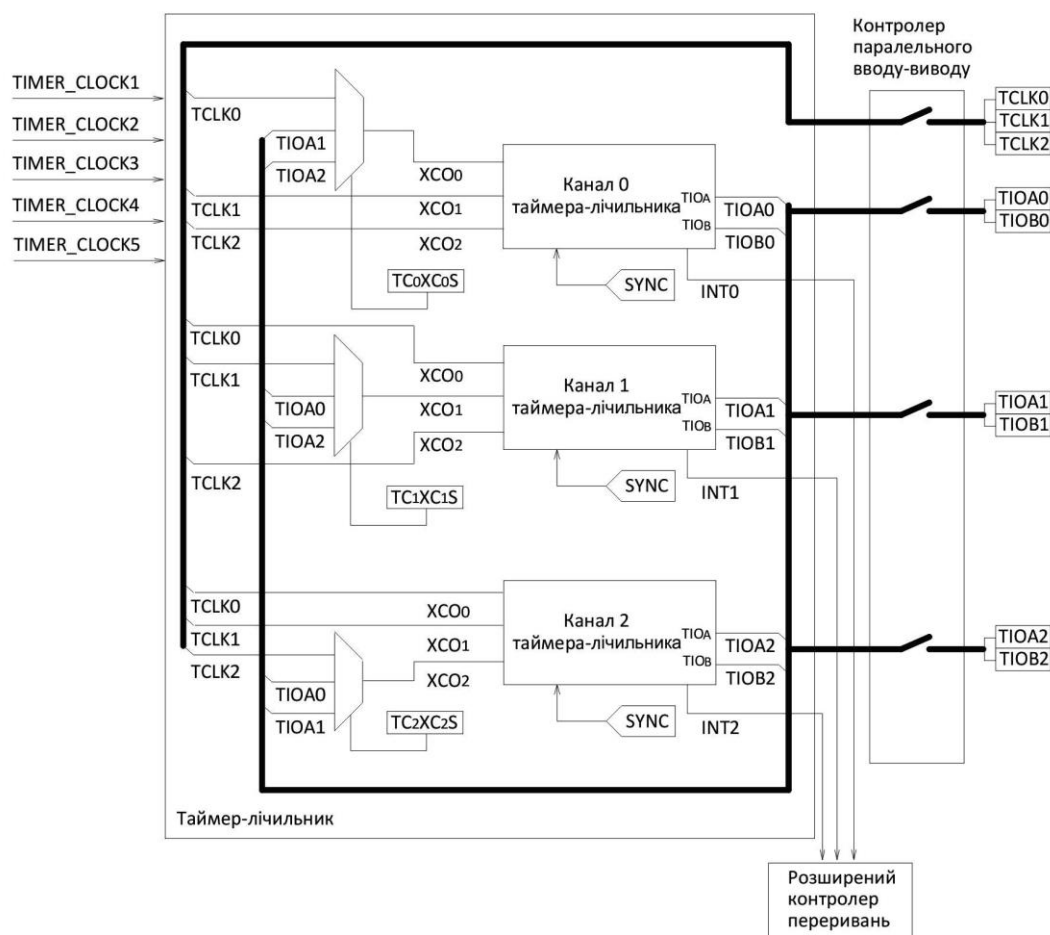


Рис. 8.20.

Приклад 8.8. Програма використовує таймер-лічильник в режимі захоплення для отримання часового інтервалу 1с.

```
#include <AT91SAM7X256.H> // підключення бібліотеки визначників МП
volatile unsigned char incr=0; // змінна, що зберігає кількість відліків
// визначники для світлодіодів
#define LED1 (1<<19)
#define LED2 (1<<20)
#define LED3 (1<<21)
#define LED4 (1<<22)
// визначники для регістру режиму TC_CMR
#define TC_CLKS_MCK 0x4 // дільник частоти 1024
#define TC_CPCTRG 0x4000 // дозвіл перезапуску при співпадінні з RC
// визначники для регістру управління TC_CCR
#define TC_CLKEN 0x1 // біт дозволу синхронізації лічильника
#define TC_CLKDIS 0x2 // біт вимкнення синхронізації лічильника
#define TC_SWTRG 0x4 // біт програмного перезавантаження

__irq void timer0 (void) // обробник переривання
{
    *AT91C_TC0_SR; // зчитування регістра стану
    incr++; // відлік
    // виведення на світлодіоди
```

```

    if(incr==1) AT91C_BASE_PIOB->PIO_ODSR=~LED1;
    if(incr==2) AT91C_BASE_PIOB->PIO_ODSR=~LED2;
    if(incr==3) AT91C_BASE_PIOB->PIO_ODSR=~LED3;
    if(incr==4) {AT91C_BASE_PIOB->PIO_ODSR=~LED4;incr=0;}
    *AT91C_AIC_EOICR = 0; // завершення переривання
}

int main()
{
    // ініціалізація виводів під світлодіоди
    AT91C_BASE_PIOB->PIO_OER=LED1|LED2|LED3|LED4;
    AT91C_BASE_PIOB->PIO_OWER=LED1|LED2|LED3|LED4;
    AT91C_BASE_PIOB->PIO_ODSR=LED1|LED2|LED3|LED4;
    // ініціалізація таймера в режимі захоплення
    AT91C_BASE_WDTC->WDTC_WDMR=0; // вимкнення сторожового таймера
    AT91C_BASE_AIC->AIC_IDCR=0xFFFFFFFF; // вимкнення переривань
    *AT91C_TC0_SR; // скидання статусу регістру
    // ввімкнення тактування таймера-лічильника
    AT91C_BASE_PMC->PMC_PCER = (1<<AT91C_ID_TC0);
    AT91C_BASE_AIC->AIC_SMR[AT91C_ID_TC0] =
    AT91C_AIC_SRCTYPE_INT_HIGH_LEVEL|
    AT91C_AIC_PRIOR_LOWEST; // за високим рівнем, низький пріоритет
    // адреса обробника переривання
    AT91C_BASE_AIC->AIC_SVR[AT91C_ID_TC0] = (unsigned int) timer0;
    // дільник частоти 1024, перезавантаження за співпадінням з RC
    *AT91C_TC0_CMR = TC_CLKS_MCK|AT91C_TC_CPCTRG ;
    *AT91C_TC0_RC = 0xB71B; // значення RC для порівняння
    // ввімкнення таймера, перезавантаження таймера
    *AT91C_TC0_CCR = AT91C_TC_CLKEN|AT91C_TC_SWTRG ;

    AT91C_BASE_AIC->AIC_ICCR = (1<<AT91C_ID_TC0) ; // очищення переривання TC0
    *AT91C_TC0_IER = AT91C_TC_CPCS ; // дозвіл переривання за співпадінням з RC
    AT91C_BASE_AIC->AIC_IECR = (1<<AT91C_ID_TC0) ; // дозвіл переривання TC0
    while (1);
}

```

Приклад, наведений вище, використовує режим захоплення таймера-лічильника. В цьому режимі можна вимірювати тривалість імпульсів, частоту, період, скважність і фазу сигналів, заведених на виводи ТІОА і ТІОВ, налаштовані на вхід. Для захоплення зовнішніх сигналів використовуються, відповідно, регістри RA і RB. В даному прикладі використовується регістр порівняння RC.

Для ввімкнення режиму захоплення необхідно встановити біт WAVE регістру режиму TC_CMR в 0 (початковий стан). Також необхідно ввімкнути тактування таймера-лічильника. Потім необхідно задати режим роботи таймера (перезавантаження таймера за співпадінням з RC) і задати частоту (дільник 1024). Встановлюємо значення регістру RC = 0xB71B (~1с). Після чого вмикаємо таймер в регістрі TC_CCR і перезавантажуємо його. Основні регістри таймера/лічильника:

TC0_CMR – регістр режиму таймера-лічильника;

TC0_RC – регістр порівняння;
TC0_CCR – регістр управління;
TC0_SR – статус регістр.

Приклад 8.9. Приклад використання таймера–лічильника в режимі генератора.

```
#include <AT91SAM7X256.H>    // підключення бібліотеки визначників МП
volatile unsigned char incr=0; // змінна, що зберігає кількість відліків
// визначники для світлодіодів
#define LED1 (1<<19)
#define LED2 (1<<20)
#define LED3 (1<<21)
#define LED4 (1<<22)
// визначники для регістру режиму TC_CMR
#define TC_CLKS_MCK 0x4      // дільник частоти 1024
// визначник для регістру управління TC_CCR
#define TC_CLKEN 0x1         // біт дозволу синхронізації лічильника
#define TC_CLKDIS 0x2        // біт вимкнення синхронізації лічильника
#define TC_SWTRG 0x4         // біт програмного перезавантаження
#define TC_WAVE (1<<15)      // біт режиму
#define TC_ACPA (3<<16)      // співпадіння з RA на TIOA
#define AT91C_PB0_TIOA0 (1<<23) // вивід TIOA
#define WAVSEL (3<<13)      // вибір способу генерації сигналу

__irq void timer0 (void)      // обробник переривання
{
    *AT91C_TC0_SR;           // зчитування регістра стану
    incr++;                  // відлік
    // виведення на світлодіоди
    if(incr==1) AT91C_BASE_PIOB->PIO_ODSR=~LED1;
    if(incr==2) AT91C_BASE_PIOB->PIO_ODSR=~LED2;
    if(incr==3) AT91C_BASE_PIOB->PIO_ODSR=~LED3;
    if(incr==4) {AT91C_BASE_PIOB->PIO_ODSR=~LED4;incr=0;}
    *AT91C_AIC_EOICR = 0; // завершення переривання
}

int main()
{
    // ініціалізація виводів під світлодіоди
    AT91C_BASE_PIOB->PIO_OER=LED1|LED2|LED3|LED4;
    AT91C_BASE_PIOB->PIO_OWER=LED1|LED2|LED3|LED4;
    AT91C_BASE_PIOB->PIO_ODSR=LED1|LED2|LED3|LED4;
    AT91C_BASE_PIOB->PIO_PDR = AT91C_PB0_TIOA0; // вимкнення виводу в PIO
    // вимкнення виводу в периферії A
    AT91C_BASE_PIOB->PIO_ASR = AT91C_PB0_TIOA0;
    // ініціалізація таймера в режимі захоплення
    AT91C_BASE_WDTC->WDTC_WDMR=0;    // вимкнення сторожового таймера
    AT91C_BASE_AIC->AIC_IDCR=0xFFFFFFFF; // вимкнення переривань
    *AT91C_TC0_SR; // скидання статус-регістра
    AT91C_BASE_PMC->PMC_PCER = (1<<AT91C_ID_TC0); // вмикання тактування
```



```

AT91C_BASE_AIC->AIC_SMR[AT91C_ID_TC0] =
AT91C_AIC_SRCTYPE_INT_HIGH_LEVEL|
AT91C_AIC_PRIOR_LOWEST           // за високим рівнем, низький пріоритет;
// адреса обробника переривань
AT91C_BASE_AIC->AIC_SVR[AT91C_ID_TC0] = (unsigned int) timer0;
// режим генерації, тривалість частоти 1024, співпадіння з RA
*AT91C_TC0_CMR = TC_WAVE|TC_CLKS_MCK|TC_ACPA|WAVSEL;
*AT91C_TC0_RA = 0x002F;           // значення RA для порівняння
*AT91C_TC0_RC = 0x006F;           // значення RC для порівняння
*AT91C_TC0_CCR = AT91C_TC_CLKEN|AT91C_TC_SWTRG; // ввімкнення таймера,
перезапуск таймера
AT91C_BASE_AIC->AIC_ICCR = (1<<AT91C_ID_TC0); // очистка переривання TC0
*AT91C_TC0_IER = AT91C_TC_CPAS; // дозвіл переривання за співпадінням з RA
AT91C_BASE_AIC->AIC_IECR = (1<<AT91C_ID_TC0); // дозвіл переривання TC0
while (1);
}

```

В цьому режимі таймер-лічильник генерує 1 або 2 зовнішніх управляючих ШІМ-сигнала з однаковою частотою і різною скважністю. Для ввімкнення режиму захоплення необхідно встановити біт WAVE регістра режиму TC_CMR в 1. Також необхідно увімкнути тактування таймера-лічильника. Потім необхідно задати один з чотирьох режимів роботи каналу, записавши необхідне значення в WAVSEL. В даному прикладі використовується четвертий режим, при якому значення TC_CV від 0 до RC. По досягненні значення рівного RC відбувається перемикавання на декрементування TC_CV аж до досягнення 0, а потім знову виконується інкрементування до RC і процес повторюється. На рис. 8. 21 представлена діаграма роботи таймера-лічильника.



Рис. 8.21.

Також можливий варіант складної ШІМ-генерації сигналу, шляхом інкрементування-декрементування лічильника після перезавантаження. Перезавантаження зовнішньою подією або програмне перезавантаження

може модифікувати TC_CV в будь-який момент часу. Якщо перезавантаження виникає під час інкрементування TC_CV, то це призводить до перемикання напрямку рахунку і TC_CV буде декрементуватися. Якщо перезавантаження виникає під час декрементування TC_CV, то відбудеться перемикання на інкрементування TC_CV. Співпадіння з RC може зупинити синхронізацію лічильника (CPCSTOP = 1) і/або вимкнути синхронізацію лічильника (CPC_DIS = 1). Принцип формування сигналів на виводах мікроконтролера демонструє часова діаграма, що представлена на рис. 8.22.



Рис. 8.22.

8.4.6. Обмін даними з використанням USART

Модуль внутрішньосхемного відлагодження працює як універсальний асинхронний приймач/передавач (рис. 8.23), і підтримує роботу тільки з 8-бітними пакетами даних (з урахуванням біта парності). Модуль має 2 незалежних каналу USART0 і USART1. Передача даних здійснюється без синхронізації, зовнішньої лінії синхронізації у модуля немає. Приймач/передавач модуля внутрішньосхемного відлагодження складається з незалежно працюючих приймача і передавача, і загального генератора тактової частоти передачі даних. Блокування за часом приймача і часовий захист передавача відсутні. За всіма іншими реалізованими функціями приймач/передавач модуля внутрішньосхемного відлагодження повністю сумісний стандартами.

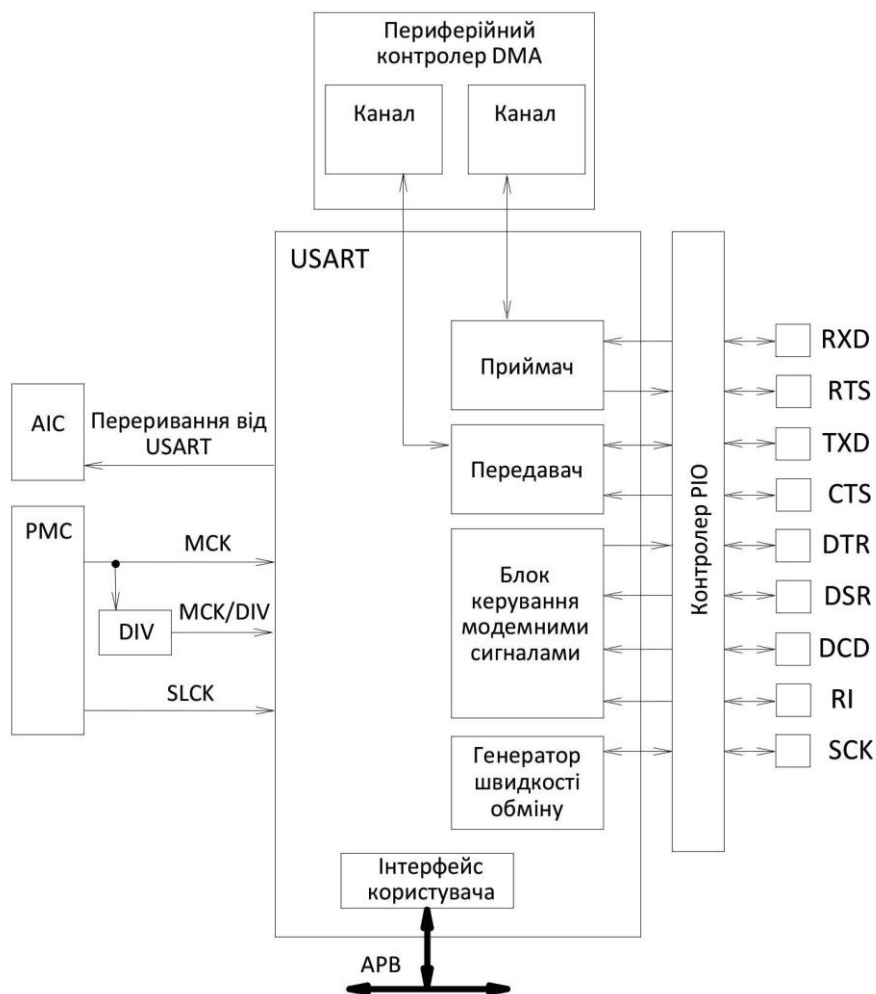


Рис. 8.23.

Для виводу даних по USART необхідно підключити відповідні виводи МП в периферії А. Потім потрібно відключити тактування модуля. Як в даному прикладі, так і в інших випадках рекомендовано вмикати тактування перед записом даних в управляючі регістри периферійних модулів. Після вмикання тактування приймач/передавач скидаються і вимикаються перед ініціалізацією управляючих регістрів. Встановлюється режим роботи, дільник швидкості і вмикається приймач/передавач. В обробнику переривання необхідно зчитувати статус регістр каналу, після обробки даних скинути приймач/передавач і закінчити переривання.

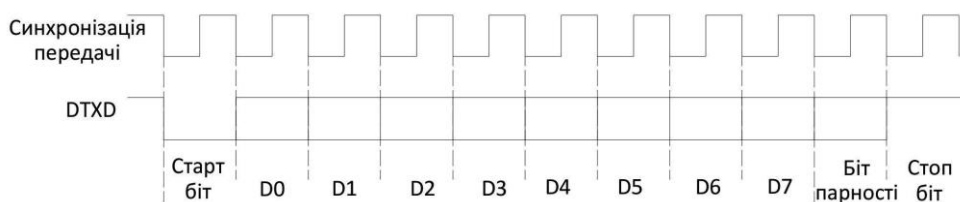


Рис. 8.24.

Модуль USART передає дані на вихід DTXT (рис. 8.24) зі швидкістю, що визначається тактовою частотою передачі даних. Формат передачі на лінію даних, що знаходяться в регістрі зсуву передавача, визначається установками регістра установки режиму роботи передавача (THR_MR). Передається один старт біт, що має рівень 0, потім 8 біт даних, послідовно від молодшого значущого до старшого, один додатковий біт парності, і стоп-біт, що має рівень 1 (рис. 8.25). Біти пакету даних послідовно зсовуються і видаються по лінії DTXD. Якщо передача біта дозволена, встановлюється рахунок суми біт даних на парність або на непарність, або передача фіксованих 1 або 0.

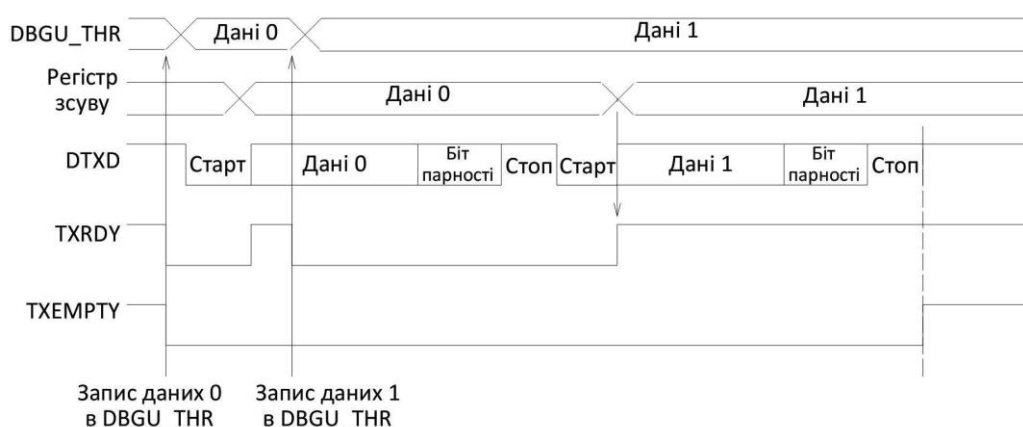


Рис. 8.25.

Рівень TXRDY підтримується високим доти, доки не буде записано другий пакет. Як тільки видача першого пакета по лінії DTXD буде завершена, і другий пакет буде передано в пустий регістр зсуву, TXRDY знову приймає високий рівень. Таким чином, TXRDY встановлюється в 1 при пустому, готовому до прийому пакета регістрі зберігання передавача.

При роботі з USART використовуються наступні регістри:

US0_CSR – статус регістр каналу;

US0_RHR – регістр значення приймача;

US_CR – регістр управління;

US_MR – регістр режиму роботи;

US_BRGR – регістр генератора тактової частоти приймача/передавача;

US_IER – регістр дозволу переривань.

Приклад 8.10. Програма використовує USART для передачі даних за послідовним портом в персональний комп'ютер.

```
#include <AT91SAM7X256.H>           // підключення бібліотеки визначників МП
#include <stdio.h>                   // стандартний заголовний файл вводу/виводу

#define AT91B_MAIN_OSC 18432000      // частота кварцового резонатора
#define AT91B_MCK ((AT91B_MAIN_OSC*73/14)/2) // вихідне значення ФАПЧ
#define BR 115200                    // швидкість передачі USART
#define BRD (AT91B_MCK/16/BR)        // дільник швидкості
```

```
AT91S_USART *pUSART = AT91C_BASE_US0; // глобальний вказівник USART0
volatile unsigned long RX_data=0; // змінна, що зберігає прийнятий байт
```

```
__irq void usart0 (void) // обробник переривання
{
    *AT91C_US0_CSR; // зчитування Status регістра каналу
    RX_data = *AT91C_US0_RHR; // зчитування прийнятого байта
    printf ("National University of Shipbuilding\n"); // виведення рядка
    pUSART->US_CR = AT91C_US_RSTRX; // скидання передавача
    pUSART->US_CR = AT91C_US_RXEN; // скидання приймача
    *AT91C_AIC_EOICR = 0; // завершення переривання
}
```

```
void init_serial (void) // ініціалізація USART0
{
    AT91C_BASE_WDTC->WDTC_WDMR=0; // вимкнення сторожового таймера
    AT91C_BASE_AIC->AIC_IDCR=0xFFFFFFFF; // вимкнення переривань
    // налаштування виводів МП
    *AT91C_PIOA_PDR = AT91C_PA0_RXD0 | // відключення виводу RX в PIO
    AT91C_PA1_TXD0; // відключення виводу TX в PIO
    *AT91C_PIOA_ASR = AT91C_PA0_RXD0 | // підключення виводу RX в периферії
    AT91C_PA1_TXD0; // підключення виводу TX в периферії А
    *AT91C_PMC_PCER = (1 << AT91C_ID_US0); // ввімкнення тактування USART0//
    регістр управління
    pUSART->US_CR = AT91C_US_RSTRX | // скидання приймача
    AT91C_US_RSTTX | // скидання передавача
    AT91C_US_RXDIS | // вимкнення приймача
    AT91C_US_TXDIS; // вимкнення передавача
    // ініціалізація переривання
    AT91C_BASE_AIC->AIC_SMR[AT91C_ID_US0] =
    AT91C_AIC_SRCTYPE_INT_HIGH_LEVEL| AT91C_AIC_PRIOR_LOWEST;
    // за високим рівнем, низький пріоритет
    AT91C_BASE_AIC->AIC_SVR[AT91C_ID_US0] = (unsigned int) usart0;
    // адреса обробника переривань
    // налаштування режиму роботи
    pUSART->US_MR = AT91C_US_USMODE_NORMAL | // нормальний режим
    AT91C_US_CLKS_CLOCK | // тактування MCK
    AT91C_US_CHRL_8_BITS | // 8 біт даних
    AT91C_US_PAR_NONE | // без біта парності
    AT91C_US_NBSTOP_1_BIT; // 1 стоповий біт

    pUSART->US_BRGR = BRD; // встановлення дільника швидкості
    AT91C_BASE_AIC->AIC_IECR=(1<<AT91C_ID_US0); // дозволяємо переривання
    pUSART->US_IER = AT91C_US_RXRDY; // дозволяємо переривання приймача
    pUSART->US_CR = AT91C_US_RXEN | // вмикаємо приймач
    AT91C_US_TXEN; // вмикаємо передавач
}
```

```
int sendchar (int ch) // функція запису байта в порт (використовуючи printf)
{
    if (ch == '\n') // якщо вказано перехід на наступну строку
```

```

{
    while (!(pUSART->US_CSR & AT91C_US_TXRDY)); // очікування очищення
прапора Tx
    pUSART->US_THR = '\r';                      // введення команди
}
while (!(pUSART->US_CSR & AT91C_US_TXRDY)); // очікування очищення прапора
return (pUSART->US_THR = ch);                  // введення байта даних
}

int main(void)
{
    init_serial();
    while(1);
}

```

Мікропроцесор очікує дані, що надходять з персонального комп'ютера і в момент прийняття будь-якого символу викликає обробник переривання, в якому виводиться в порт рядок «National University of Shipbuilding». Для виводу даних в порт використовувалась функція `printf()`, яка попередньо ініціалізована і перевизначена для роботи з USART. Для цього необхідно підключити групу системних викликів в проєкті System Calls/Retarget і описати функцію відправки `sendchar()`.

8.4.7. Використання двохпровідного інтерфейсу TWI для запису/зчитування даних

Двопровідний інтерфейс (TWI) об'єднує пристрої за допомогою унікальної двопровідної шини, що складається з одної лінії синхронізації і одної лінії даних, що дозволяє здійснити передачу у байт-орієнтованому форматі зі швидкістю до 400 кБіт в секунду. Інтерфейс може бути використаний для роботи з ЕСППЗП (EEPROM) марки Atmel, що має 2-провідну шину. TWI програмується як ведучий пристрій з постійною послідовною або побайтовою передачею даних. Налаштований генератор синхронізації передачі даних дозволяє налаштувати швидкість передачі даних під широкий діапазон тактових частот ядра. Структурна схема TWI представлена на рис. 8.26.

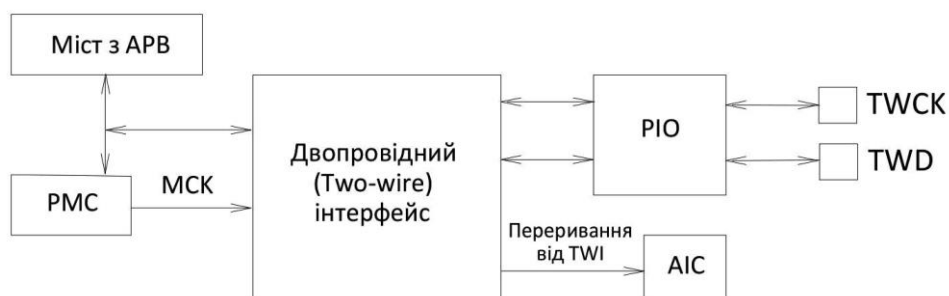


Рис. 8.26.

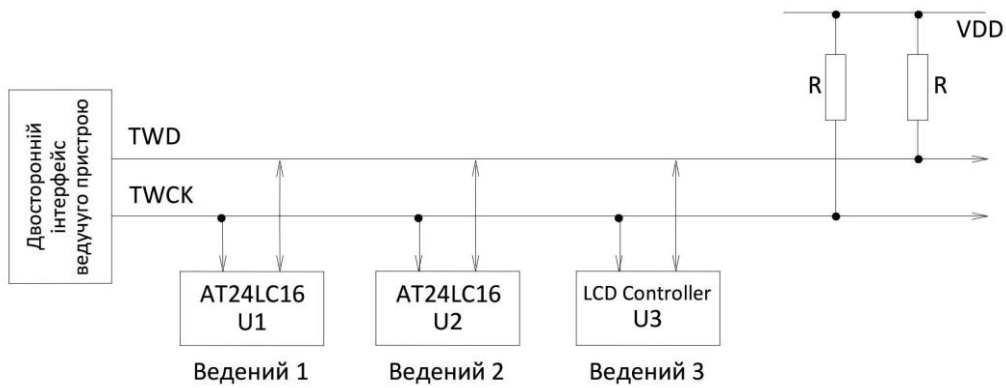


Рис. 8.27.

Кожна передача починається з умови Старт і закінчується умовою Стоп (рис. 8.28). Перепад високий – низький рівень на лінії TWI при високому рівні на TWCK визначається як умова Старт. Перепад низький – високий на лінії TWI при високому рівні на TWCK визначається як умова Стоп.

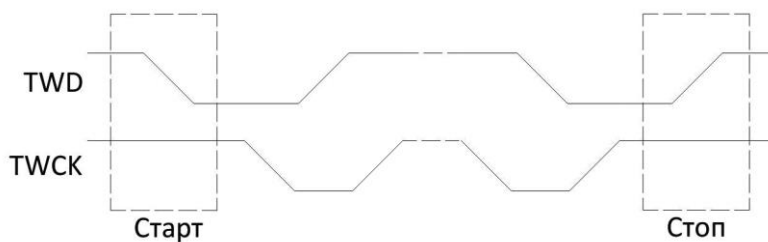


Рис. 8.28.

Після встановлення ведучим пристроєм умови Старт вона відсилає 7-бітну адресу веденого пристрою, задану в регістрі Режиму роботи (DADR в TWI_MMR). Наступний за передачею адреси біт визначає напрям передачі даних – прийом або передача. Рівність цього біта 0 показує, що буде здійснюватись запис даних (операція передавання). Якщо ж біт рівний 1, відбувається операція зчитування. Формат пакетів даних представлений на рис. 8.29.

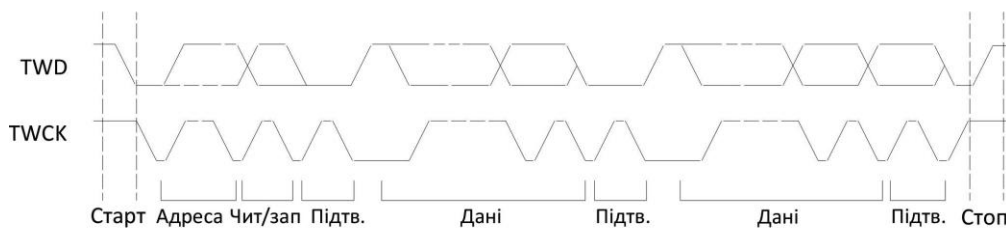


Рис. 8.29.

Для роботи двохпроводного інтерфейсу необхідне отримання підтвердження про прийняття кожного біта від веденого пристрою. Під час тактового імпульсу синхронізації, що співпадає з очікуванням підтвердження, ведучий пристрій звільняє управління лінією даних (на ній встановлюється високий рівень), з тим, щоб ведучий пристрій міг

встановити на ній низький рівень, видаючи таким чином рівень підтвердження. Ведучий пристрій опитує лінію даних під час цього імпульсу синхронізації, встановлюючи біт NAK в регістрі стану у випадку, якщо ведений пристрій не видав на лінію сигнал підтвердження. Встановлення цього біта, як і деяких інших біт стану, може генерувати переривання, якщо це дозволено відповідними установками в регістрі дозволу переривань інтерфейсу (TWI_IER).

Двохпровідний інтерфейс дозволяє здійснити передачу в різноманітних форматах (7-бітна адреса ведучого або 10-бітна адреса веденого). Три внутрішніх адресних біти можна конфігурувати за допомогою регістру режиму роботи ведучого пристрою (TWI_MMR). Якщо ведений пристрій підтримує тільки 7-бітну адресацію, IASDR повинен бути встановлений в 0. Для адреси веденого більш ніж 7 біт, користувач повинен налаштувати розмір адреси (IADRSZ) і встановити іншу адресу веденого в регістрі внутрішньої адреси. На рис. 8.30 представлений формат пакета для запису байта даних в ЕСППЗП Atmel AT24LC (EEPROM). Показано застосування внутрішньої адреси для доступу до пристрою.



Рис. 8.30.

При роботі з інтерфейсом TWI використовуються наступні регістри:

- TWI_SR – регістр стану;
- TWI_IADR – регістр внутрішньої адреси TWI;
- TWI_CWGR – регістр генератора синхросигналу;
- TWI_MMR – регістр режиму ведучого;
- TWI_CR – регістр управління;
- TWI_THR – регістр зберігання значення передачі;
- TWI_RHR – регістр зберігання значення прийому.

Приклад 8.11. Програма використовує двохпровідний інтерфейс TWI для запису/зчитування даних в енергозалежну мікросхему пам'яті типу EEPROM – AT24C16.

```
#include <AT91SAM7X256.H>
#include "lib_AT91SAM7X256.H"          // підключення додаткових функцій
#define DEV_ADR 0                      // адреса пристрою
#define EEPROM_DADR (DEV_ADR | 0x50) // адреса EEPROM
// визначники для світлодіодів
#define LED1 (1<<19)
```



```

#define LED2 (1<<20)
#define LED3 (1<<21)
#define LED4 (1<<22)

// функція ініціалізації TWI
void twi_init(void)
{
    // підключення виводів МП в периферії A
    AT91F_TWI_CfgPIO();
    // підключення тактування TWI модуля
    AT91F_TWI_CfgPMC();
    // ініціалізація TWI контролера, скидання TWI, встановлення режиму ведучого
    AT91F_TWI_Configure(AT91C_BASE_TWI);
    // встановлення синхроімпульса TWI
    // для перевірки використовуємо мінімальну швидкість обміну
    *AT91C_TWI_CWGR = (7<<16)|((255<<8)|(255));
    // трьохбайтова внутрішня адреса, запис ведучим пристроєм, адреса веденого пристрою
    *AT91C_TWI_MMR = 0x7F0200;
}

// функція відсилення байта
void i2cMasterSend( unsigned char dev_adr,      // адреса пристрою
                    unsigned short mem_adr,    // адреса комірки пам'яті
                    unsigned char data)        // дані
{
    unsigned int status;                       // змінна, що зберігає значення статус регістра
    // встановлення адреси комірки пам'яті
    *AT91C_TWI_IADR = mem_adr;
    // запис ведучим, встановлення адреси пристрою
    *AT91C_TWI_MMR &= ( 0xFF00EFFF | (dev_adr<<16) );
    // відсилення команди START, відсилення TWI дозволено, відсилення команди STOP
    *AT91C_TWI_CR = AT91C_TWI_START | AT91C_TWI_MSEN | AT91C_TWI_STOP;
    // запис даних до регістру відсилення
    *AT91C_TWI_THR = data;
    // зчитування статус регістра, очікування закінчення передачі
    status = *AT91C_TWI_SR;
    while ( !(status & AT91C_TWI_TXCOMP) ) {
        status = *AT91C_TWI_SR;
    }
}

// функція зчитування байта
unsigned char i2cMasterReceive( unsigned char dev_adr,      // адреса пристрою
                               unsigned short mem_adr)    // адреса комірки пам'яті
{
    unsigned int status; // змінна, що зберігає значення статус регістра
    // встановлення адреси комірки пам'яті
    *AT91C_TWI_IADR = mem_adr;
    // читання ведучим, встановлення адреси пристрою
    *AT91C_TWI_MMR |= ( 0x00001000 | (dev_adr<<16) );
    // зупинка пакета START, відсилення TWI дозволено, відсилення пакета STOP
    *AT91C_TWI_CR = AT91C_TWI_START | AT91C_TWI_MSEN | AT91C_TWI_STOP;

```

```
// зчитування статус регістра, очікування закінчення передачі
status = *AT91C_TWI_SR;
while ( !(status & AT91C_TWI_TXCOMP) ) {
status = *AT91C_TWI_SR;
}
// функція повертає прийняті дані
return (unsigned char)*AT91C_TWI_RHR; // зчитування регістра вхідних даних
}

void delay(int cl) // функція затримки
{
    int i,i1=0;
    for(i=0;i<cl;i++)
    {
        for(i1=0;i1<1000;i1++) {} // відлік
    }
}

int main(void)
{
    unsigned char data=0;          // змінна, що зберігає прийняте значення
    unsigned char i=2;             // змінна, що зберігає значення для відправлення
    unsigned short mem_addr;       // змінна, що зберігає адресу комірки пам'яті
    AT91C_BASE_WDTC->WDTC_WDMR=0; // вимкнення сторожового таймера
    AT91C_BASE_PIOB->PIO_OER=LED1|LED2|LED3|LED4; // регістр дозволу виводу
    AT91C_BASE_PIOB->PIO_OWER=LED1|LED2|LED3|LED4; // регістр дозволу запису
    AT91C_BASE_PIOB->PIO_ODSR=LED1|LED2|LED3|LED4; // загасити світлодіоди
    // ініціалізація TWI:
    twi_init();
    while(1)
    {
        for (mem_addr=0; mem_addr<1; mem_addr++) // запис блоку даних до EEPROM
        {
            i2cMasterSend(EEPROM_DADR, mem_addr, i);
        }
        delay(1000); // затримка
        for (mem_addr=0; mem_addr<1; mem_addr++) // зчитування блоку даних
        {
            data = i2cMasterReceive(EEPROM_DADR, mem_addr);
        }
        // якщо зчитане і відправлене значення співпадають – запалити світлодіод
        if(data==2) AT91C_BASE_PIOB->PIO_ODSR=~LED1;
        delay(1000); // затримка
    }
}
```

Для використання інтерфейсу TWI необхідно підключити виводи МК в периферії А, увімкнути тактування, ідентифікувати TWI контролер, скинути TWI, встановити режим ведучого, задати частоту обміну даними по TWI, встановити діапазон адрес ведених пристроїв. Для відсилання байта даних необхідно вказати адресу комірки пам'яті, обрати режим

відсилання ведучим, встановити адресу пристрою, відіслати пакет START, відіслати байт даних і очікувати кінця передачі. Для прийому байта необхідно вказати адресу комірки пам'яті, обрати режим читання ведучим, встановити адресу пристрою, відіслати пакет START, очікувати кінця передачі і повернути головній функції зчитане значення.

Для перевірки працездатності TWI-модуля використовувалась мінімальна швидкість передачі. Також помічено ряд ситуацій, що приводять апаратний модуль TWI в неробочий стан. А саме, робота з мікросхемою SAA7113, при використанні якої TWI модуль зависає. Також в умовах підвищених перешкод, при потраплянні перешкоди або випадковому замиканні на обох лініях передачі модуль висне. В подібних випадках краще використовувати програмну реалізацію TWI.

Питання до розділу 8

1. Які особливості архітектури мікроконтролерів AT91 на базі ядра ARM?
2. Від яких джерел тактових сигналів може працювати мікроконтролер?
3. Які функції виконує контролер живлення PMC?
4. Опишіть призначення виводів мікроконтролерів сімейства AT91SAM7S.
5. Які модулі входять до мікроконтролерів сімейства AT91SAM7S? Для чого використовується кожний з модулів?
6. Дайте характеристику середовища розробки Keil μ Vision.
7. Які периферійні модулі входять до складу мікроконтролерів сімейства AT91SAM7S?
8. Яка навантажувальна здатність виводів мікроконтролера?
9. В чому полягає особливість роботи контролера переривань (AIC)?
10. Які регістри використовуються при роботі з контролером переривань?
11. Опишіть послідовність операцій для ініціалізації переривань PIO.
12. Які таймери-лічильники має мікроконтролер? Для чого використовується кожний з таймерів-лічильників?
13. В яких режимах може працювати таймер-лічильник?
14. Опишіть формат передачі даних по USART.

Задачі до розділу 8

41. Реалізувати 5 світлових ефектів на світлодіодах.
42. Реалізувати лічильник з коефіцієнтом перерахунку $M = 12$. Значення лічильника виводити на світлодіоди.
43. Розробити реверсивний лічильник. При натисканні на одну кнопку лічильник інкрементується, при натисканні на іншу кнопку значення лічильника зменшується.
44. Написати програму, яка змінює яскравість світлодіодів: спочатку світлодіоди поступово загоряються, а потім поступово згасають.

45. Написати програму, яка після прийому значення з комп'ютера (від 0 до 9) встановлює відповідну яскравість світлодіодів.
46. Написати програму, яка виконує аналого-цифрове перетворення і відправляє отримане значення в комп'ютер.
47. Написати програму, яка працює як зворотний таймер, зменшує значення змінної кожен секунду на 1. Використовувати таймер-лічильник.
48. Розробити програму контролю температури. Значення температури зчитується з датчика, який має інтерфейс I2C (DS1621). Отримане значення передається в комп'ютер.
49. Написати програму, яка реалізує передачу/приймання даних по USB-інтерфейсу: опитування стану кнопок, керування світлодіодами, передачу даних, зчитаних з АЦП.
50. Написати програму, яка реалізує зчитування та запис даних на карту пам'яті SD/MMC.

9. РОЗРОБКА СПЕЦІАЛІЗОВАНИХ МІКРОПРОЦЕСОРНИХ МОДУЛІВ НА ПРОГРАМОВАНИХ ЛОГІЧНИХ ІНТЕГРАЛЬНИХ СХЕМАХ

При розробці цифрових пристроїв широке поширення знайшли програмовані логічні інтегральні схеми, які дуже вдало доповнили мікропроцесорні засоби. Це зручна елементна база, яка використовується в системах цифрової обробки сигналів, телекомунікації, для побудови різноманітних систем керування та ін. Існує можливість створювати на кристалі спеціалізовані процесорні ядра, що дозволяє вирішувати будь-які науково-практичні задачі. Однак схемна реалізація подібних систем – задача досить складна, а іноді і неможлива. Для вирішення подібних задач може бути використана мова опису апаратури (HDL), однією з яких є Verilog. Він вигідно відрізняється загальноалгоритмічною базою, гнучкістю, широким діапазоном охоплення систем та рівнів їх опису. Проектувальник може скласти функціональний HDL-опис проектованої системи, і використовуючи систему моделювання САПР, перевірити її роботу. Verilog є офіційно признаним стандартом опису цифрової апаратури.

В загальному випадку опис пристрою за допомогою Verilog представляє собою текст, що зберігається в одному або декількох файлах, які в сукупності складають уявлення розробника про проект. Тексти описів на більшості мов проектування цифрових пристроїв по складу синтаксичних конструкцій схожі із традиційними мовами програмування. Як правило, створюючи програму розробник абстрагується від конкретної фізичної реалізації, виділяючи насамперед алгоритм функціонування проектованого пристрою. На основі моделі може бути побудований фізичний пристрій.

Модель, що відображає об'єкт проектування у формі правил перетворення вхідних даних у вихідні, називається поведінковою. Структурна модель описує проект у вигляді сукупності модулів, кожен з яких реалізує певну частину завдання, і набору зв'язків між ними. Структурна модель є загальним способом створення складних проектів, опис яких доцільно виконати по ієрархічному принципу. В програмному модулі вищого рівня ієрархії присутні оголошення портів і внутрішніх зв'язків, а також декларації входжень модулів. Це ілюструє приклад асинхронного передавача. Як правило, першим етапом проектування є створення поведінкової моделі вищого рівня ієрархії. Часто може знадобитися кілька корекцій до одержання відповідності поведінкової моделі заданим вимогам. В процесі реалізації деякого проекту повна декомпозиція потрібна не завжди. На деякому етапі може виявитися, що сукупність операторів, яка відтворюється деяким модулем, присутня у бібліотеці, що поставляється із САПР. В такому випадку досить у відповідному структурному тілі послатися на цей бібліотечний модуль.

9.1. Основи Verilog. Структура модулів

Verilog HDL – це мова опису апаратури, яка використовується для опису цифрових систем, наприклад комп'ютерних модулів. Компілятори Verilog, поряд з VHDL-компіляторами, включені в більшість САПР інтегральних схем. Синтаксис мови Verilog нагадує синтаксис мови програмування C. У Verilog менша в порівнянні з VHDL кількість службових слів, що спрощує його вивчення. Verilog дозволяє ефективно виконувати опис і проводити моделювання та синтез цифрових схем завдяки вбудованим примітивам, засобам часового контролю, моделюванню затримок сигналів від входу до виходу. Крім того, через свої розширені можливості VHDL програє Verilog по ефективності, тобто для опису однієї і тієї ж конструкції на Verilog потрібно в кілька разів менше символів, чим в VHDL.

Говорячи про синтаксис мови Verilog слід зазначити, що він є контекстно-залежною мовою, тобто рядкові і прописні букви розрізняються. Базовою одиницею мови Verilog є проектний модуль, або просто модуль, який інтегрує як визначення інтерфейсу, так і правила функціонування блоку. Кожен модуль починається з ключового слова **module**, за яким розташована його назва та перелік портів. Опис модуля закінчується ключовим словом **endmodule**. Між цими дома ключовими словами розташоване описання інтерфейсу – оголошення портів, перелік параметрів. При описі портів вказуються їхні імена, розрядність, напрям. Кожен порт може мати один з трьох напрямів:

- вхідний – дані зчитуються модулем з його оточення (інших модулів) через вхідні порти. Неможливо записати дані в такий порт;
- вихідний – дані відсилаються модулем з його оточення через вихідні порти. Зчитування даних з такого порту неправильне;
- двонапрямлений – дані можуть бути як зчитані, так і записані в такий порт.

Далі розташоване тіло модуля – його внутрішній вміст. Тіло модуля може складатися з оголошення змінних, операторів безперервного або процедурного присвоєння, задач та функцій. Відмітимо, що в тілі модуля не може бути використана змінна, якщо вона не оголошена. Також всередині модуля може бути розташована директива компілятора **'include**, за допомогою якої відбувається підстановка текстового фрагменту з іншого файлу.

Концепції мови дозволяють описувати пристрої з використанням різних рівнів абстракції. Модулі можуть містити оператори включення інших модулів, що дозволяє створювати ієрархічні проекти. Текст програми може містити довільний набір проектних модулів, кожний з яких представлений назвою, описом портів і тілом модуля. Модуль не може містити в собі опис іншого модуля. Загальна структура модуля наведена нижче:

```
module «ім'я_модуля»(«список_портів_модуля»); // заголовок модуля
    // опис портів модуля;
    // реалізація модуля:
    // оголошення сигналів, шин;
    // підключення інших модулів;
    // функціональний опис пристрою;
    // підпрограми;
endmodule // кінець опису модуля
```

Ім'я модуля – ідентифікатор, що дозволяє використовувати модуль в інших модулях. Список портів модуля – перелік виводів, без вказівки їх напрямку і типу переданої інформації, за допомогою яких модуль підключається до інших частин схеми. Перелік портів розташовується в дужках після назви модуля. Опис портів модуля – це перше, що потрібно зробити при описанні модуля. Він представляє собою вказівку типу порту, його розрядності та назви. Після цього розташовується оголошення сигналів та шин. Ці сигнали визначаються в реалізації модуля – структурі модуля або операторів мови, які описують його поведінку. Розглянемо опис модуля, який реалізує прості логічні функції.

```
module primer(A1, A2, B1, B2); // заголовок модуля
input A1, A2; // опис портів модуля: вхідні порти
output B1, B2; // вихідні порти
wire w1=A1; // оголошення сигналів
wire w2=A2; // що підключені до кожного з входів
    nand(B2,w1,w2); // B2=~(A1&A2) – елемент 2І-НІ
    assign B1=A1 & A2; // функціональний опис: B1=A1&A2
endmodule
```

В описі модуля ідентифікатори A1 та A2 – вхідні сигнали, B1 та B2 – вихідні сигнали. За допомогою ключового слова **wire** створюються два додаткових ланцюги w1 і w2, які підключені до вхідних портів A1 та A2 відповідно. За допомогою цих ланцюгів підключається елемент **nand**, який реалізує логічну функцію 2І-НІ. Вихід логічного елемента підключений до виходу B2 описаного модуля. Сигнал на виході B1 представляє собою результат виконання логічної операції І над вхідними сигналами. Для цього був використаний оператор „&”. Більш детальний опис операторів приводиться нижче. За допомогою ключових слів **input** (вхідний) та **output** (вихідний) вказуються типи портів.

В Verilog використовуються складені імена – це послідовність імен, розділених крапками (з їхньою допомогою здійснюється доступ до елементів модулів, наприклад параметрів). У складних випадках, при великій кількості портів модуля і довільним порядком їх перерахунку в оголошенні портів, використовується ключовий принцип: список портів утворює пари з назви порту модуля і назви приєднаного до нього ланцюга:

```
dff dff1(.d (data), .q (q_out), .clk (clock), .clrn (clearn), .prn (presetn));
```

Основна форма представлення інформації з якою оперує Verilog – це сигнальні дані, що відображають стан змінних і ланцюгів в пристрої, який описується. Крім того, використовуються службові дані, що служать для задання конфігурації, керування моделюванням і компіляцією, а також відображення результатів у процесі моделювання.

Verilog підтримує наступні «стандартні» типи даних: ціле – **integer** (32-бітове зі знаком) і **real** – число із плаваючою крапкою. При розробці синтезованих моделей з перерахованих типів використовується тільки **integer**. Тип **real** не підтримується при синтезі.

При запису числових констант прописні і малі літери еквівалентні, наприклад:

8'b01101011	// 8-бітне число у двійковій системі
12'h5B	// 12-бітне число в шістнадцятковій системі
6'o43	// 6-бітне число у вісімковій системі

просте представлення про рівень сигналу дає двійковий (булевий) алфавіт, в якому сигнал може мати високий (TRUE, 1) або низький (FALSE, 0) рівні. Однак для практичного використання це занадто грубе представлення чисел. В Verilog використовується чотирьохзначний алфавіт з додатковими значеннями. Символ „x” використовується для задання невизначеного стану (наприклад, вихід невстановленого тригера позначається як 1'bx), символ „z” показує високоімпедансний стан (наприклад обрив лінії). При використанні в якості цифри замість „z” у числах можна використовувати символ „?”.

6'oz4	// запис числа z4 у восьмеричному форматі
8'h7x	// запис числа 7x у шістнадцятковій формі
12'dz	// запис числа в десятковій формі

В Verilog введено два основних типи для представлення сигналів: «ланцюги» і «регістри». Кожній групі відповідають специфічні способи збереження присвоєних значень, і в остаточному підсумку, різні схемні рішення для їхньої реалізації в мікросхемі. Найпоширеніші з них описуються ключовими словами **wire** та **reg** відповідно.

Розглянемо приклад опису тригера з входом дозволу:

```
module DFF(q, data, en, rst, clk); // заголовок програми
output q; // оголошення вихідного порту
input data, en, rst, clk; // оголошення вхідних портів
reg q; // вихід має регістровий тип
always @(posedge clk) // по фронту тактового сигналу
    if (rst==0) // якщо сигнал сбросу встановлений в 0
        q=1'b0; // на виході також встановлюється 0
    else if (en==1) // якщо сигнал дозволу встановлений в 1
        q=data; // на виході з'являється значення входу data
endmodule // кінець опису модуля
```


В приведеному описі вихід тригера *q* має регістровий тип. Це дозволило використовувати операцію присвоєння в процедурному блоці. На відміну від попередніх прикладів значення вихідного сигналу зберігається до наступного присвоєння, а не змінюється разом зі зміною вхідних сигналів. Операція присвоєння, а отже і установка на виході нового значення, відбувається по фронту сигналу *clk*. При цьому на вході дозволу *en* повинний бути сигнал лог.1. В протилежному випадку стан виходу залишається незмінним. Якщо в момент появи фронту тактового сигналу на вході *rst* був сигнал низького рівня, на виході *q* тригера встановлюється значення 0. Тобто тригер має інвертований вхід синхронного сбросу. Перевірка стану сигналів дозволу *en* та сбросу *rst* відбувається за допомогою оператора **if**.

В багатьох проектах потрібні багаторозрядні ланцюги, які називаються шинами, або векторами, по яким можуть одночасно передаватися різні значення сигналів. Хорошим прикладом є 32-розрядний мікропроцесор. По шині даних такого мікропроцесора в кожний момент часу передається 32 біти двійкової інформації. Всі біти впорядковані і розташовуються, як правило, від молодшого розряду праворуч до старшого розряду ліворуч. Оголошення векторів, які мають тип **reg** подібно до оголошення векторів з'єднань **wire**. Приклади оголошень векторів представлені нижче:

```
wire [7:0] input_signal;           // 8-розрядний ланцюг
reg [15:0] out_data;              // 16-розрядний регістр
out_data [7:0] = input_signal;      // часткове призначення сигналу
```

При оголошенні вектору його розрядність вказується за допомогою індексів, які розташовуються в квадратних дужках ('[', ']'). Індеси розділені за допомогою двокрапки (:), і розташовуються між вказівкою типу сигналу та його ідентифікатором.

Verilog має велику кількість вбудованих примітивів (**primitive**) – логічні елементи І, АБО та інші, а також дозволяє розробнику створювати власні – примітиви користувача (UDP – User Defined Primitives). Примітиви представляють з себе базові функціональні блоки, за допомогою яких виконується структурний опис системи.

Нижче приводиться приклад створення і використання примітиву користувача 2І-НІ, а також спосіб його використання в системі Quartus.

```
module udp(y1,y2,a,b,c);           // заголовок модуля
output y1, y2;                   // вихідні порти
input a,b,c;                     // вхідні сигнали
wire w1;                        // додатковий ланцюг
    xor(w1,a,b);                  // вихід примітиву – перший у списку
    and(y2,w1,c);                // логічний елемент І
    INE2( y1,a,b);               // підключення примітиву користувача
endmodule
```

```

// опис примітиву користувача
primitive INE2(Y,X1,X2);
output Y;           // єдиний вихід повинен бути першим в списку параметрів
input X1,X2;        // всі порти (до 10) повинні бути однобітовими
    table           // таблиця станів комбінаційної схеми
        0 ? : 1; // ? – байдуже значення аргументу
        ? 0 : 1;
        1 1 : 0;
    endtable        // кінець опису таблиці станів
endprimitive        // кінець опису модуля

```

Початок опису примітиву починається із зарезервованого слова **primitive**, за яким розташовується його назва та перелік портів. Опис примітиву повинен закінчуватися ключовим словом **endprimitive**. Перелік портів завжди починається з назви одного вихідного порту, за яким через кому перелічені всі вхідні порти. Вихідний порт примітиву послідовнісного типу повинен мати регістровий тип. Таблиця станів розташована між зарезервованими словами **table** та **endtable**. Якщо в таблиці визначені не всі можливі стани, в такому разі вихід перейде у невизначений стан. При описі примітивів послідовнісного типу в таблиці істинності повинні бути вказані як поточний стан виходу, так і наступний. В таблиці істинності спочатку вказуються стани вхідних сигналів в тому порядку, в якому вони перелічені в переліку портів. Далі через двокрапку вказується значення, яке встановиться на виході у відповідності до вхідних сигналів. В кінці кожного рядка повинна ставитися крапка з комою. Кількість пробілів при описанні таблиці істинності не має значення. Результати моделювання наведеного опису представлені на рис. 9.1.

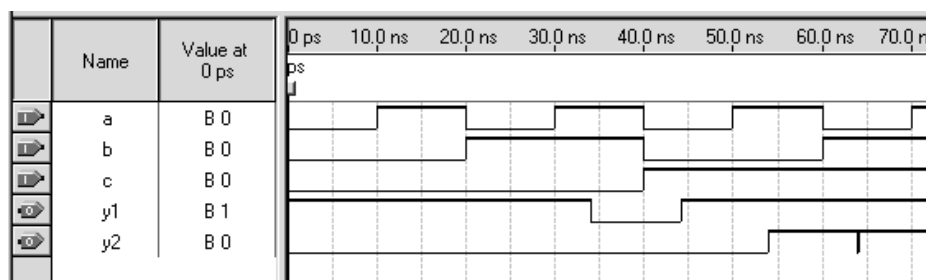


Рис. 9.1.

Як видно з діаграми, на виході y1 встановлюється сигнал високого рівня тоді, коли лише на одному з входів присутня лог.1. Описаний примітив представляє з себе логічний елемент 2І-НІ. Можливість використання великої кількості вхідних портів при описі примітивів дозволяє реалізовувати логічні функції, які представлені у вигляді таблиці станів.

В Verilog існують три типи операторів – з одним, двома та трьома операндами. Унарні оператори розташовуються ліворуч від операнду, бінарні – між двома операндами, а тернарний оператор розділяє три

операнди двома операторами. Для формування результату у виразі операнди поєднуються знаками операції. В якості операндів можуть використовуватися числа, імена змінних (ланцюгів, регістрів), окремі біти і групи бітів, виклики функцій, які повертають значення кожного з перерахованих типів.

Проектування систем обробки інформації неможливо без реалізації операцій добутку, суми, різниці, ділення. Нижче приводиться приклад використання арифметичних операторів.

```
module arithmetic(dataA, dataB, add, sub, mul);  
input [7:0] dataA, dataB;           // вхідні порти  
output [8:0] add;                   // вихідний порт суми  
output [8:0] sub;                   // вихідний порт різниці  
output [15:0] mul;                  // вихідний порт добутку  
assign add=dataA+dataB;             // призначення вихідним портам  
assign sub=dataA-dataB;             // результату виконання арифметичних  
assign mul=dataA*dataB;             // операцій  
endmodule
```

В Verilog розрізняють два види операторів – паралельні і послідовні. Послідовні оператори виконуються один за одним, в порядку запису. Паралельні оператори виконуються при будь-якій зміні сигналу, який використовується як аргумент. Послідовні оператори повинні бути в тілі складених операторів.

Оператор постійного повторення **always** перший раз виконується при початку моделювання, але потім повторюється після завершення вкладеного оператора. У зв'язку із циклічним характером цього оператора, він може мати сенс при наявності в його тілі конструкції, що передбачають його перехід у режим очікування. Найчастіше це очікування яких-небудь подій у системі. Елементарною формою події є зміна значення ланцюга або регістра, яка відслідковується оператором очікування події **@**.

```
reg data_out;  
always @(a or b or c)              // очікування зміни сигналів a, b та c  
if (b)                               // якщо b – істина, то data_out = a  
    data_out = a;  
else                                 // інакше data_out = c  
    data_out = c;
```

Припустимо, є наступний опис:

```
always @ (posedge CLK) a=b;         // по фронту CLK виконується  
always @ (posedge CLK) b=a;         // оператор присвоєння
```

Нехай a і b – регістри одиничної довжини, і до моменту позитивного фронту тактового сигналу CLK a = 0, b = 1. Яке значення будуть мати ці змінні після проходження фронту? Це не визначено в мові і залежить від

того, у якій послідовності операції присвоєння потрапляють у список. Тобто виходить конструкція, поводження якої непередбачене. Це означає, що обидві ці змінні приймуть значення 0 або 1. Конкретний результат залежить від симулятора і від послідовності, в якій будуть оброблені рядки вихідного файлу. «Блокуюче», або «блокове» присвоєння (=) забороняє виконання інших послідовних операцій доти, доки не буде виконана задана операція. Використання операції блокуючого присвоєння у подібній конструкції небажано і його варто уникати. Але в той же час, якщо в блоці потрібно провести послідовне виконання операторів, варто застосовувати даний тип присвоєння.

В якості активізуючої події може використовуватися фронт або спад сигналу. Конструкція

```
always @ (posedge CLK)           // початок процедурного блоку
begin
    a=0;                          // a=0
    b=a;                          // b=0
end
```

гарантує, що після фронту CLK обидві змінні – а та b – будуть мати значення 0. Один з фундаментальних принципів Verilog – будь-який об'єкт, якому присвоюється значення в процедурному блоці **always**, повинен мати регістровий (**reg**) тип.

Ознакою контролю події є символ @. У розглянутих раніше прикладах контроль подій використовувався в блоках **always**. Відмінність Verilog від VHDL в цьому випадку полягає в тому, що для опису фронтів і спадів сигналів використовуються не спеціалізовані атрибути сигналу, а спеціальна конструкція мови. Контроль подій використовується в процедурних блоках так само, як і часовий контроль. При цьому затримка виконання відбувається не на фіксований часовий інтервал, а доти, доки не відбудеться потрібна подія.

Події бувають наступних типів:

- @ (name) – зміна name, при цьому name може бути ланцюгом, регістром;
- @ (**posedge** A) або @ (**negedge** A) – фронт або спад сигналу A, при цьому A – однобітовий регістр або ланцюг;
- комбінація перерахованих подій із ключовим словом «**or**», наприклад @(**posedge** CLK **or** **posedge** SET); в цьому випадку варто розрізняти «**or**» з однойменною логічною операцією; у конструкції контролю подій «**or**» означає, що очікується будь-яка з перерахованих подій, а не певний результат логічної операції.

Таким чином список чутливості, що містить слово **posedge**, забезпечує виконання оператора тільки при зміні результату обчислення виразу, записаного в дужках, з нульового стану в одиничне. Аналогічно, ключове слово **negedge** задає чутливість тільки до зміни сигналу з

одиночного стану в нульовий. У префіксах з вказаними ключовими словами вирази повинні бути скалярними (розрядність 1 біт). Отже, коли відбувається виконання деякого виразу при виникненні події, то цей вираз є чутливим до цієї події. Verilog дозволяє визначати вирази, які будуть виконані при виникненні однієї з декількох подій. В такому разі назви сигналів повинні бути відокремлені словом **or**, а вирази знаходитися в блоці **begin** – **end**. Контроль подій дозволяє зупинити, або навпаки, відновити виконання нескінченного оператора **always**.

Оператор умови (**if**) і оператор варіанту (**case**) дозволяють вибрати один з можливих шляхів виконання алгоритму в залежності від поточних умов. У процесі моделювання вони відносяться до класу послідовних (процедурних) операторів і виконуються слідом за оператором, що передує їм у програмному блоці. При синтезі пристрою відтворюються блоки, що виконують всі описані альтернативи, з яких або ініціюється в кожний момент тільки один блок, або вихідні сигнали зчитуються тільки з одного із блоків. Конструкція **if** записується в такий спосіб:

```
if (<умова>
    <оператори>
else
    <оператори>
```

Для вибору з декількох варіантів можуть бути застосовані вкладені оператори **if**.

```
if (<умова>
    <оператори>
else if (<умова>
    <оператори>
else if (<умова>
    <оператори>
else
```

Тут <умова> – будь-який вираз мови, а <оператор> – оператор або група операторів, які розташовані між **begin** та **end**. Ключове слово **else** може бути відсутнім, але якщо є вкладені **if** (як у прикладі), то **else** відноситься до найближчого **if**. Якщо одержане у виразі <умова> значення не дорівнює 0 і не є невизначеною (x або z), то виконується гілка **if**, інакше – **else**. Варто пам'ятати, що так само, як у мові програмування C, операція порівняння записується як == (два знаки «=»), на відміну від операції присвоєння = (один знак).

Наступний приклад демонструє використання умовного оператора **if** для опису регістра з входом дозволу **en** та асинхронним сбросом **reset**, в який записуються дані по фронту тактового сигналу **clk**.

```

module reg_if(dout, clk, en, reset, din);    // заголовок модуля
output [3:0] dout;                        // вихідний 4-розрядний порт
input clk, en, reset;                     // вхідні сигнали
input [3:0] din;                          // вхід даних
reg [3:0] dout;                           // вихід має регістровий тип
always @(posedge clk or posedge reset)   // по фронту одного з сигналів
                                           // якщо сброс – на виході 0
if (reset) din=0;
else if (en) dout=din;                    // якщо встановлено дозвіл, на виході вхідні дані
else dout=0;                             // при невиконанні попередніх умов, на виході 0
endmodule

```

Діаграма на рис. 9.2 демонструє роботу модуля.

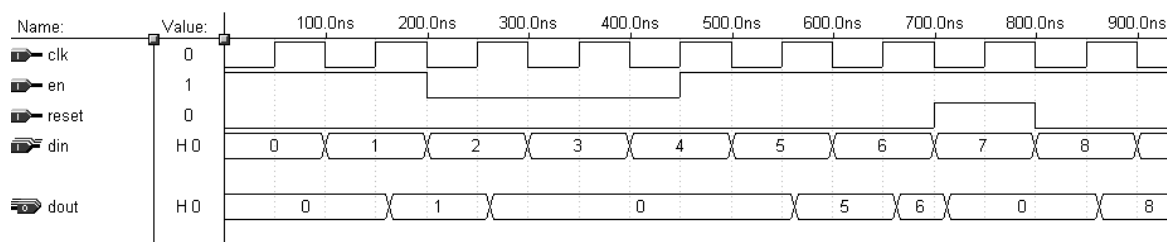


Рис. 9.2.

У мові Verilog визначені чотири форми операторів повторення: **forever**, **repeat**, **while**, **for**. У всіх операторах повторення вкладений оператор може бути простим або складовим, тобто містити сукупність послідовних операторів, які розташовані між ключовими словами **begin** та **end**. По синтаксису і по інтерпретації записаних дій, оператори повторення практично не відрізняються від таких самих операторів мови програмування C, за винятком оператора **forever**. Цей оператор повторюється нескінченно. Він починається з ключового слова **forever**, за яким розташований оператор, що буде постійно повторюватися. Це робить його подібним до блоку **always**. Якщо операторів декілька, вони розташовуються в блоці **begin** – **end**. В циклі **forever** обов'язково повинен бути присутній часовий контроль. В протилежному випадку симулятор постійно буде виконувати тільки цей оператор, забороняючи моделювання інших частин модуля. Найчастіше цикл **forever** використовується для генерування тактового сигналу.

Виконання циклу **for** контролюється трьома параметрами, які розташовуються в дужках після ключового слова **for**: початкове значення, умова для переривання циклу та алгоритм зміни лічильника циклу між ітераціями. Розглянемо приклад використання циклу **for**.

```

module forr (d, q);                      // початок опису модуля
input [2:0] d;                           // декларація вхідних портів та їх розрядності
output [1:0] q;                          // декларація вихідних портів та їх розрядності
integer num_bits;                        // оголошення додаткових змінних,
                                           // необхідних для роботи алгоритму
always @ (d)                             // початок блоку контролю подій

```

```

begin: block
integer i;           // оголошення змінної цілого типу
num_bits = 0;
for (i = 0; i < 3; i = i + 1) // початок оператора повторення
if (d[i] == 1)
    // перевірка на рівність 1 кожного біта вхідного масиву
    num_bits = num_bits + 1; // інкремент лічильника, якщо умова
                             // виконується
end                     // block
assign q = num_bits;
    // установка на виході підрахованого значення бітів
endmodule             // кінець опису модуля

```

Ця програма описує алгоритм підрахунку числа одиниць у вхідному слові. Вхідний масив *d* використовується як змінна в конструкції **always** для контролю появи події. Таким чином, зміна цієї змінної приводить до виконання блоку процедурного призначення. Якщо біт у вхідному масиві даних дорівнює 1, збільшується на 1 значення лічильника бітів, оскільки виконується умова. Цей алгоритм виконується доти, поки значення змінної „i” менше 3. Після цього відбувається установка значення кількості одиниць у вхідному слові на вихідному порту *q*.

Як і у традиційних мовах програмування, використання підпрограм дозволяє забезпечити структурування, отже, краще розуміння програм, а також заощаджує час проектувальника, дозволяючи однократно описувати однотипні фрагменти пристроїв і алгоритми їх функціонування. В Verilog розрізняють два типи підпрограм, що відрізняються по способу повернення результату: задачі (**task**) і функції (**function**). Задача повертає результат через зіставлення формальних і дійсних об'єктів, а функція повертає одне значення через назву підпрограми на місці її виклику у виразі.

В описі наступного модуля є функція Shift, яка виконує зсув 16-розрядного слова *data* на один розряд ліворуч (0) або праворуч (1), в залежності від значення сигналу *direction*, який є другим параметром, що передається в функцію. На виході *Rdata* з'явиться вхідний код, зсунутий на один розряд праворуч, а на виході *Ldata* – зсунутий на один розряд ліворуч.

```

module function2(data, Ldata, Rdata);    // заголовок модуля
parameter WIDTH=16;                    // розрядність вхідного порту
input [WIDTH-1:0] data;                  // вхідний порт
output [WIDTH-1:0] Ldata, Rdata;         // вихідні порти
reg [WIDTH-1:0] Ldata, Rdata;            // виходи мають регістровий тип
function [WIDTH-1:0] Shift;              // заголовок функції
    input [WIDTH-1:0] din;                 // перший параметр – вхідний код
    input direction;                      // другий параметр – напрям зсуву
if(direction==0) Shift=din<<1;          // в залежності від умови виконується
else if(direction==1) Shift=din>>1;      // зсув ліворуч або праворуч
endfunction                             // кінець опису функції

```

```
always @(data)           // щоразу при зміні вхідного коду
begin                     // виконується послідовний блок
    Ldata=Shift(data,0);   // виклик функції
    Rdata=Shift(data,1);
end                       // always
endmodule                // кінець опису модуля
```

Загальні для сценаріїв та функцій властивості полягають в наступному:

- задачі та функції можуть мати локальні змінні та регістри;
- ні задачі, ні функції не можуть містити локальних ланцюгів (**wire**);
- задачі та функції можуть містити лише поведінкові оператори і викликатися з інших поведінкових операторів (**always**, **initial**). Функції також можуть бути викликані з оператора безперервного присвоєння **assign**.

9.2. Арифметичні пристрої та схеми контролю

До арифметичних пристроїв відносяться суматори, компаратори, арифметично-логічні пристрої, а також схеми для математичної обробки інформації на їх основі. В практиці цифрової схемотехніки існує декілька типів арифметичних пристроїв, наприклад, паралельного, послідовного типу, пристрої із збереженням інформації та інші. В цьому розділі будуть розглядатись лише схеми комбінаційного типу.

9.2.1. Суматори

Суматором називається комбінаційний логічний пристрій, призначений для виконання операції арифметичного додавання чисел, представлених у вигляді двійкових кодів. Суматор є одним з основних вузлів арифметично-логічного пристрою. Класифікація суматорів може бути виконана по різних ознаках. За кількістю виводів розрізняють напівсуматори, однорозрядні суматори, багаторозрядні суматори. Напівсуматором називається пристрій, призначений для додавання двох однорозрядних кодів, який має два входи і два виходи та формуючий із вхідних сигналів сигнали суми і переносу в старший розряд. Багаторозрядним суматором називається пристрій, призначений для додавання двох багаторозрядних кодів, який формує на виході код суми і сигнал переносу у випадку, якщо результат додавання не може бути представлений кодом, розрядність якого збігається з розрядністю кодів доданків.

Розглянемо приклад суматора двох чотирьохрозрядних слів. Текст програми приводиться нижче.

```
module summator(sum,pout,a,b,pin);    // початок опису модуля
output [3:0] sum;                    // оголошення вихідних портів
output pout;
input [3:0] a;                        // оголошення вхідних портів
input [3:0] b;
input pin;                            // вхід переносу
```



```

reg [3:0] sum;
reg pout;
always @(a or b or pin)           // очікуємо зміни сигналів на вході
    begin
        sum = a + b + pin;           // виконуємо операцію додавання
        if (a+b<16) pout = 0;        // якщо сума менша за 16, то
                                    // значення переносу дорівнює 0
        else pout = 1;               // інакше встановлюємо сигнал переносу в 1
    end
endmodule                          // кінець опису модуля

```

В описаному суматорі входами є порти a, b – чотирьохрозрядні слова для виконання операції додавання, pin – вхід переносу. Вихідними є порти sum – сума чисел a та b, pout – вихід переносу. При реалізації віднімання даний вихід може служити знаковим розрядом результату. Часова діаграма роботи модуля суматора приводиться на рис. 9.3.

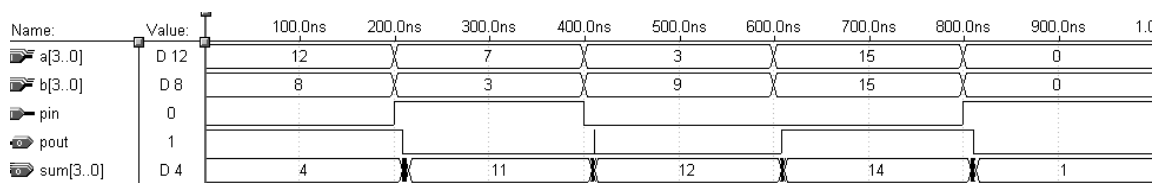


Рис. 9.3.

Проаналізуємо роботу модуля. Якщо на входах a та b присутні числа 12 і 8 відповідно, то при їх додаванні виходить число 20. Однак для представлення цього числа у двійковому коді потрібно 5 розрядів. Тому, на виході встановлюється значення 4 ($12+8-2^4$), і встановлюється біт переносу (pout). Якщо сума менша 15, то значення біту переносу дорівнює 0. В описі модуля відслідковується зміна кожного із сигналів на вході, і при зміні відбувається обчислення суми вхідних слів.

З таблиці бачимо нерівність значень затримок сигналів від різних входів на вихід. Це призведе до того, що при черговому підрахунку суми на виході буде "тремтіння" окремих бітів. Для запобігання цього явища рекомендується ставити на вихід суматора паралельний регістр, а в суматор додати вихід готовності результату.

Багаторозрядні суматори поділяються на послідовні та паралельні. У послідовних суматорах операція додавання виконується послідовно, розряд за розрядом, починаючи з молодшого. У паралельних суматорах всі розряди вхідних кодів додаються одночасно. Розрізняють комбінаційні суматори – пристрої, які не мають власної пам'яті, та накопичуючі суматори з власною внутрішньою пам'яттю, у якій зберігаються результати виконаної операції. При цьому кожний наступний доданок додається до того значення, що містилось у пристрої. По способу тактування розрізняють синхронні та асинхронні суматори. У синхронних суматорах час виконання операції арифметичного додавання двох кодів не залежить

від виду самих кодів і завжди залишається постійним. В асинхронних суматорах час виконання операції залежить від вигляду доданків. Тому по завершенню виконання операції додавання необхідно виробляти спеціальний сигнал завершення операції.

Verilog допускає велику різноманітність стилів опису об'єктів проекту. В попередньому прикладі був використаний поведінковий опис суматора. Однак існують інші способи функціонального опису об'єктів, більш близькі до традиційних мов програмування. Розглянемо приклад поведінкового опису n -розрядного суматора з послідовним переносом.

```
module adder(a,b,cin,sum,cout); // заголовок модуля
parameter n=10; // розрядність вхідних даних
input [n:1] a,b; // вхідні порти
input cin; // вхід попереднього переносу
output [n:1] sum; // вихід суми
output cout; // вихід переносу
integer i; // змінна для використання в циклі
reg [n:1] vsum; // додатковий регістр
reg carry; // змінна для зберігання біту переносу
always @(a or b or cin) // процедурний блок
begin
    carry=cin; // запам'ятовуємо попередній перенос
    for (i=1;i<=n;i=i+1) // виконується цикл n разів
        begin
            vsum[i]=(a[i]^b[i])^carry; // формування кожного біту суми
            carry=(a[i] & b[i])|(carry & (a[i]|b[i])); // формування переносу
        end
    end
    assign sum=vsum; // установка на виході значення суми
    assign cout=carry; // установка переносу
endmodule // кінець опису модуля
```

В наведеному описі використовується цикл, в тілі якого відбувається додавання по модулю 2 (операція ВИКЛ.АБО над бітами) окремих бітів кожного з вхідних слів a та b , з урахуванням біту переносу cin . За межами процедурного блоку **always** відбувається установка обчисленого значення суми на вихідному порту. Також формується біт переносу. Параметр n визначає розрядність вхідних слів.

Аналогічний суматор можна отримати при використанні оператора об'єднання. При цьому значно скорочується об'єм коду:

```
module adder(a,b,cin,sum,cout); // заголовок модуля суматора
parameter n=10; // параметр – розрядність вхідних даних
input [n:1] a,b; // вхідні дані
input cin; // вхід переносу
output [n:1] sum; // вихідний порт суми двох слів
output cout; // вихід переносу
assign {cout,sum}=a+b+cin; // визначення суми та значення біту переносу
endmodule // кінець опису модуля
```

9.2.2. Компаратори

Компаратори виконують порівняння двох чисел, представлених у вигляді двійкових кодів. Мікросхеми компараторів визначають не тільки рівність, але і нерівність двох чисел. Для цього мікросхема має три виходи: “ $A > B$ ”, “ $A < B$ ” і “ $A = B$ ”, на одному з яких в залежності від співвідношення величин $A = a_3a_2a_1a_0$ та $B = b_3b_2b_1b_0$ з'являється активний рівень сигналу. При описі компаратора на Verilog розробник може включати в свій модуль такі виводи, які йому необхідні (наприклад вихід, логічна одиниця на якому встановлюється якщо на вході значення слів розрізняються на 1, тобто майже рівні). Пристрої порівняння на рівність будуються на основі порозрядних операцій над однойменними розрядами обох слів. Побудувати багаторозрядний компаратор можна на основі суматора. У відповідності до законів арифметики, при $A = B$ на виходах всіх розрядів суматора s_0, s_1, s_2, s_3 буде 0 при $r_i + 1 = 1$. При $A > B$ значення 1 буде як на виході переносу $r_i + 1 = 1$, так і хоча б на одному виході s_i . Тому ознакою $A > B$ може бути функція $y = (r_i + 1) \cdot (\sum s_i)$. При $A > B$ результат ознаки переносу $r_i + 1 = 0$.

Компаратори широко використовуються в інформаційних системах для виділення необхідного слова в потоці цифрової інформації, для відмітки часу в часових пристроях, для виконання умовних переходів в обчислювальних пристроях. В пристроях автоматики компаратори використовуються для контролю виходу величин за припустимі межі, в приводах слідкуючих систем автоматики і т.д.

Розглянемо опис компаратора, який забезпечує порівняння двох 8-розрядних чисел по трьох ознаках.

```
module comparator(Agt, Alt, Aeq, A, B); // початок опису модуля
output Agt, Alt, Aeq; // оголошення вихідних портів
input [7:0] A; // оголошення вхідних портів
input [7:0] B;
reg Agt, Alt, Aeq; // вихідні порти мають тип регістр
    always @(A or B) // очікування зміни вхідних сигналів
    begin
        if (A > B) // якщо виконується умова A > B
        begin
            Agt = 1'b1; // установлюємо потрібні значення сигналів
            Alt = 1'b0; // на відповідних виходах
            Aeq = 1'b0;
        end
        else if (A < B) // якщо виконується умова A < B
        begin
            Agt = 1'b0;
            Alt = 1'b1; // установлюємо в 1 сигнал „менше”
            Aeq = 1'b0;
        end
        else // якщо A = B
        begin
            Agt = 1'b0;
```

```

Alt = 1'b0;
Aeq = 1'b1;           // установлюємо в 1 сигнал „рівні”
end
end
endmodule              // кінець опису модуля

```

Проаналізувавши програму можна бачити, що ми, як і при описі суматора, використали поведінковий опис пристрою. В даному прикладі демонструється використання операторів порівняння в конструкції **if**. Вихідні порти мають тип регістр, тому при призначенні їм відповідних значень не пишеться ключове слово **assign** (якщо його написати, то воно просто буде проігнороване компілятором). Діаграма на рис. 9.4 демонструє роботу приведенного опису компаратора.

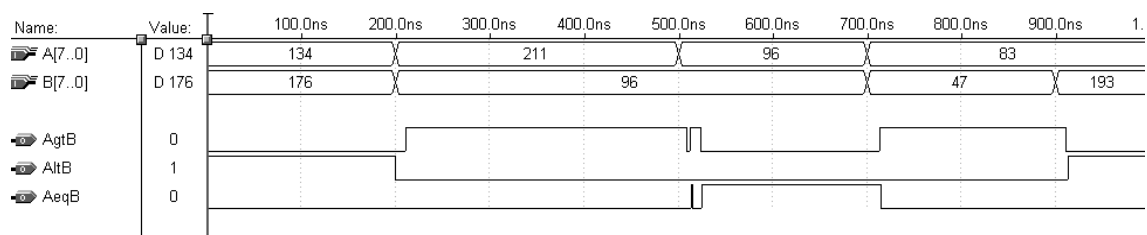


Рис. 9.4.

Використовуючи параметризовані модулі, розробник зможе вибирати розрядність суматора, а також необхідні ознаки порівняння. На приведеній часовій діаграмі в моменти зміни сигналів на входах компаратора, на виходах виникають паразитні імпульси. Один з шляхів вирішення цієї проблеми полягає у введенні синхронізуючого імпульсу.

9.2.3. Арифметично-логічні пристрої (АЛП)

Арифметично-логічні пристрої – це спеціалізовані мікросхеми, які виконують арифметичні та логічні операції у відповідності до двійкового коду, який подається на керуючі входи мікросхеми. Описуючи арифметично-логічний пристрій мовою Verilog, розробник має можливість описати тільки ті операції, які необхідні для побудови конкретної системи. Таким чином, не будуть використовуватися “дарма” ресурси програмованої мікросхеми. Це вигідно відрізняє текстовий опис модуля від використання готових елементів з бібліотеки, які мають значні можливості, які часом зовсім не використовуються.

Розглянемо невеликий приклад. Запропонований арифметично-логічний пристрій буде оперувати чотирьохрозрядними словами і мати три входи вибору операції. Пристрій виконує чотири логічні та чотири арифметичні операції. Опис модуля АЛП представлений нижче.

```

module alu(F, Pout, Sel, A, B);           // початок опису пристрою
output [3:0] F;                          // опис вихідних портів
output Pout;                             // вихід переносу
input [2:0] Sel;                          // опис вхідних портів

```

```

input [3:0] A;           // вхід A даних
input [3:0] B;           // вхід B даних
reg [3:0] F;             // вихідні порти мають тип "регістр"
reg Pout;
always @ (A or B or Sel) // стежимо за зміною вхідних сигналів
begin
  case (Sel)              // початок конструкції вибору
    4'b000 : F=~A;         // інверсія значення A
    4'b001 : F=A&B;        // операція побітового I
    4'b010 : F=~(A^B);     // інверсія
    4'b011 : F=&B;          // згортка по I
    4'b100 : {Pout,F}=A+B;  // сума
    4'b101 : {Pout,F}=A+(~B);
    4'b110 : {Pout,F}=(A^B)+(A-B);
    4'b111 : {Pout,F}=2*B;  // зсув вліво на один розряд
    default : F=0;
  endcase
end
endmodule                // кінець опису модуля

```

У конструкції case відбувається вибір однієї з альтернатив в залежності від стану сигналів Sel[2], Sel[1], Sel[0]. Розробник у кожній з альтернатив може описати той алгоритм, який йому потрібний. Тобто можна побудувати пристрій АЛП під конкретну задачу. У програмі також використаний оператор об'єднання розрядів вихідних портів. Це зроблено для зручності призначення цим портам сигналів. На рис. 9.5 приводиться часова діаграма роботи АЛП.

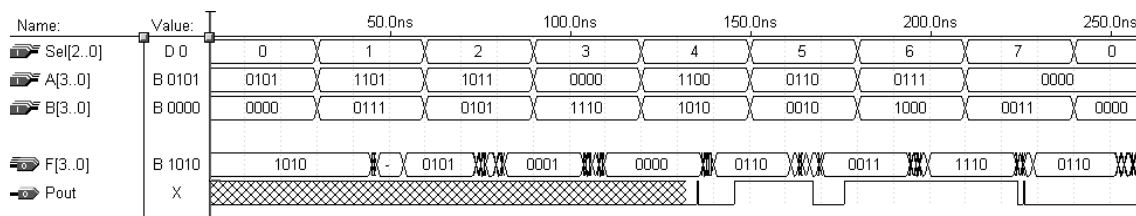


Рис. 9.5.

Проаналізувавши діаграму, можна переконатися в правильності роботи пристрою. Дійсно, коли на вході вибору Sel присутній 0, на виході встановлюється інвертоване значення A. Код 011 на вході Sel викликає виконання операції згортки по I над значенням B. Оскільки тільки при умові рівності 1 всіх розрядів числа B результатом операції згортки по I буде лог.1, то в цьому випадку на виході встановлюється значення 0. Значення Pout до першого присвоєння в програмі залишається невизначеним. Для того, щоб сигнал переносу був встановлений в 0 при виконанні операцій, які не впливають на стан цього виводу, перед початком конструкції case потрібно додати строчку наступного вигляду: Pout = 0;. Тип виконуваної операції відповідає коду на вході Sel.

9.2.4. Контроль парності

Незважаючи на високу надійність та перешкодостійкість цифрових систем передачі інформації, вірогідність появи помилок завжди існує. Тому всі високонадійні канали передачі інформації забезпечуються допоміжними схемами, які дають можливість впевнитись у відсутності помилок в прийнятих даних. В будь-якій структурі каналу передачі інформації з контролем помилок повинна бути надмірність каналу. Наприклад, якщо передається код $0101_2 = 5_{10}$ і на виході каналу з'являється $1101_2 = 13_{10}$, то в загальному плані помилку без спеціальних перевірок визначити неможливо. Але якщо ми знаємо, що інформація передається у двійково-десятковому коді, то одержаний результат хибний. Тобто, наявність шести надлишкових станів дає можливість виявити деякі помилки.

Простий спосіб визначення помилок у словах базується на припущенні, що найбільша вірогідність збою можлива тільки в одному біті, тобто в появі помилкової одиниці або нуля. Тому для визначення наявності такого збою використовують контроль парності або непарності одиниць у слові, що передається (parity check). В основі цього способу лежить операція знаходження суми по модулю 2 всіх двійкових розрядів контрольованого слова. При парній кількості одиниць вказана сума дорівнює 0, а при непарній – 1. Рівняння, за яким підраховується кількість одиниць, має вигляд:

$$Y = x_0 \oplus x_1 \oplus x_2 \oplus x_3 \oplus v = y \oplus v.$$

З аналізу рівняння витікає, що при парній кількості одиниць у слові $x_3x_2x_1x_0$ значення виходу $y=0$. При непарній – $y=1$. Опис модуля визначення парності має вигляд:

```
module parity(y, v, in);           // початок опису модуля
output y;                         // оголошення вихідних портів
input v;                          // оголошення вхідних портів
input [7:0] in;
reg y;
always @ (v or in)                // очікуємо зміни сигналів на вхідних портах
begin
    if(v)                         // якщо v має високий рівень
        y=(^in);                 // виконуємо операцію згортки по ВИКЛ.АБО
    else
        y=~(^in);               // інакше встановлюємо на виході інверсне значення
                                // результату операції згортки
    end
endmodule                        // кінець опису модуля
```

При зміні будь-якого сигналу на вході відбувається операція згортки вхідного байту по ВИКЛ.АБО. Значення „y” залежить від сигналу „v”, який задає режим контролю – контроль парності або контроль непарності. При контролі парності задається $v=0$. Якщо відбудеться контроль по

непарній кількості одиниць, то задається $v=1$. Вихід „у” називається контрольним бітом. При прийомі інформації одержане слово знову перевіряється на парність або непарність. Якщо при контролі парності в приймальному контрольному пристрої з'явилась 1, то це означає, що або в інформаційних даних, або в контрольному біті виникла помилка. Як бачимо, такий простий контроль не дає можливості виправити помилку, але він дає можливість визначитись з хибною інформацією, щоб потім або використати її, або відкоригувати.

В практиці контролю переважно використовують спосіб контролю по непарності. Пов'язано це з тим, що при появі слова з усіма нулями при контролі по парності цю ситуацію неможливо відрізнити від обриву лінії зв'язку.

Приведена схема не може визначити подвійної помилки і будь-якої кількості парних помилок. Пояснюється це лише малою надмірністю лінії зв'язку. Тому для більш глибокого контролю необхідно мати і більшу її надмірність.

При побудові мереж доводиться кодувати передані дані тим або іншим способом. Це обумовлено декількома причинами. Однією з них є присутність у переданій послідовності постійної складової. Тому елементи гальванічної розв'язки (наприклад, трансформатори) не зможуть передати довгу послідовність нулів або одиниць. Крім того, бажано забезпечити дані деякою надмірністю для виявлення помилок (більш ніж в одному розряді). Інакше кажучи, необхідно виконати логічне кодування. Тут варто підкреслити, що на етапі логічного кодування вже не формується форма сигналів. Тут просто борються з недоліками методів фізичного цифрового кодування – відсутність синхронізації, наявність постійної складової. Таким чином, спочатку за допомогою засобів логічного кодування формуються виправлені послідовності двійкових даних, які потім за допомогою методів фізичного кодування передаються по лініях зв'язку. Логічне кодування має на увазі заміну біт вхідної інформації новою послідовністю біт, що несе таку саму інформацію, але крім цього має додаткові властивості, наприклад можливість для приймальної сторони виявляти помилки в прийнятих даних. Супроводження кожного байта вихідної інформації одним бітом парності – це приклад способу логічного кодування який часто використовується при передачі даних за допомогою модемів. Іншим прикладом логічного кодування може служити шифрація даних, що забезпечує їх конфіденційність при передачі через загальні канали зв'язку. Одним з методів логічного кодування є застосування надлишкових кодів. Надлишкові коди засновані на розбивці вихідної послідовності біт на порції, які часто називають символами. Потім кожен вихідний символ замінюється на новий, який має більшу кількість біт. Один з прикладів надлишкового коду – логічний код 4В/5В. Логічний код 4В/5В замінює вихідні символи довжиною в 4 біти на символи довжиною в 5 біт. Оскільки результуючі символи містять надлишкові біти, то загальна кількість бітових комбінацій в них більша, ніж у вхідній. Таким чином,

п'ятибітова схема дає $32 (2^5)$ дворозрядних літерно-цифрових символів, які мають значення в десятковому коді від 0 до 31, в той час як вхідні дані можуть містити тільки чотири біти або $16 (2^4)$ символів. Тому в результуючому коді можна підібрати 16 таких комбінацій, які не містять великої кількості нулів, а інші вважати забороненими кодами (code violation). Очевидно, що в такому випадку довгі послідовності нулів перериваються. Зникає постійна складова, звужується спектр сигналу. Але цей метод знижує корисну пропускну здатність лінії, тому що надлишкові біти інформації не несуть, і тільки "забирають ефірний час". Надлишкові коди дозволяють приймачу розпізнавати помилкові біти. Якщо приймач приймає заборонений код, то це означає, що на лінії відбулося пошкодження даних. Комбінації, які мають більше трьох нулів (01–00001, 02–00010, 03–00011, 08–01000, 16–10000), інтерпретуються символом V і командою приймача VIOLATION – збій. Команда означає наявність помилки через високий рівень перешкод або збій передавача. Єдина комбінація з п'яти нулів (00–00000) ставиться до службових сигналів, означає символ Q і має статус QUIET – відсутність сигналу в лінії.

Таке кодування даних вирішує два завдання – синхронізації і поліпшення перешкодостійкості. Синхронізація відбувається за рахунок виключення послідовності більше трьох нулів, а висока перешкодостійкість досягається приймачем даних на п'ятибітовому інтервалі. Ціна за ці переваги при такому способі кодування даних – зниження швидкості передачі корисної інформації. Наприклад, в результаті додавання одного надлишкового біта на чотири інформаційних, ефективність використання смуги частот в протоколах з кодом MLT-3 і кодуванням даних 4B/5B зменшується відповідно на 25%. Схема кодування 4B/5B представлена в табл. 9.1.

Табл. 9.1.

Двійковий код 4В	Результуючий код 5В	Двійковий код 4В	Результуючий код 5В
0000	11110	1000	10010
0001	01001	1001	10011
0010	10100	1010	10110
0011	10101	1011	10111
0100	01010	1100	11010
0101	01011	1101	11011
0110	01110	1110	11100
0111	01111	1111	11101

Отже, відповідно до цієї таблиці формується код 4B/5B, потім передається по лінії за допомогою фізичного кодування по одному з методів потенційного кодування, чутливого тільки до довгих послідовностей нулів – наприклад, за допомогою цифрового коду NRZI. Символи коду 4B/5B довжиною 5 біт гарантують, що при будь-якому

їхньому сполученні на лінії не можуть зустрітися більше трьох нулів підряд. Спробуємо описати пристрій, який виконує логічне кодування даних. Опис модуля має вигляд:

```
module coder(y, v, in);           // початок опису модуля
output [4:0] y;                 // оголошення вихідних портів
input v;                        // оголошення вхідних портів
input [3:0] in;
reg [4:0] y;
always @ (y or v)              // очікуємо зміну сигналів на входах
    begin
        if(v)                    // якщо встановлено сигнал v
            case (in)            // вибираємо значення з декількох альтернатив
                4'b0000 : y=5'b11110;
                4'b0001 : y=5'b01001;
                4'b0010 : y=5'b10100;
                4'b0011 : y=5'b10101;
                4'b0100 : y=5'b01010;
                4'b0101 : y=5'b01011;
                4'b0110 : y=5'b01110;
                4'b0111 : y=5'b01111;
                4'b1000 : y=5'b10010;
                4'b1001 : y=5'b10011;
                4'b1010 : y=5'b10110;
                4'b1011 : y=5'b10111;
                4'b1100 : y=5'b11010;
                4'b1101 : y=5'b11011;
                4'b1110 : y=5'b11100;
                4'b1111 : y=5'b11101;
            endcase              // кінець конструкції вибору
            else y=0;            // встановлюємо на виході 0 якщо v=0
        end
    endmodule                   // кінець опису модуля
```

9.2.5. Перемножувачі та поділювачі

Ідеологія перемноження двох бінарних слів полягає у використанні операцій додавання і зсуву проміжної суми. Реальні чотирьохрозрядні перемножувачі використовують просту технологію, відповідно до якої для двох слів А та В, які необхідно перемножити, створюється таблиця істинності з вихідним словом С подвійної довжини. Кожен розряд c_i слова С виступає логічною функцією з логічними змінними слів А та В. Тому реалізація чотирьохрозрядного перемножувача є простою реалізацією 8-ми логічних функцій. При необхідності реалізувати перемножувач двох восьмирозрядних слів кожне з них розбивається на групи по 4 біти і з кожною з груп виконуються операції, як з однією змінною, за принципом знаходження проміжної суми з послідовним виконання операції зсуву. Розглянемо приклад. Необхідно розробити модуль підрахунку факторіала. Найпростіше було б використати згадану технологію побудови перемножувачів, тобто створити таблицю істинності і за допомогою

оператора вибору **case** реалізувати модуль. Але для демонстрації деяких можливостей мови вирішимо це завдання дещо інакше. Опис модуля підрахунку факторіалу має вигляд:

```
module factor(result, ready, in, clk, start); // початок опису модуля
output [7:0] result; // оголошення вихідної шини
output ready; // вихід готовності даних
input [2:0] in; // вхід даних
input clk, start; // входи тактового сигналу та запуску
reg [7:0] result; // вихід має регістровий тип
reg ready;
reg [2:0] temp; // додаткові змінні
reg [7:0] count;
always @ (posedge clk) begin
// очікування фронту тактового сигналу
    if (start==1) // якщо умова виконується
    begin
        temp=in; // присвоюємо тимчасовому регістру
                  // значення на вході
        count=8'b00000001; // установлюємо значення змінної
        ready=0; // обнуління сигналу готовності результату
    end
    else begin
        count=count*temp; // операція множення
        temp=temp-1; // зменшення значення тимчасового регістра
        if (temp<3'b010) // якщо значення регістра менше 2
        begin
            result=count; // установка результату на виході
            ready=1; // установка сигналу готовності
        end
    end
end
endmodule // кінець опису модуля
```

Модуль працює наступним чином. При появі фронту синхроімпульсу виконується перевірка входу start. При високому рівні сигналу на цьому вході в регістр temp записується значення того числа, факторіал якого необхідно визначити. На виході result зберігається попереднє значення. Сигнал готовності результату ready має низький рівень. Якщо на вході start присутній лог.0, то починається визначення факторіалу. Особливість роботи модуля полягає в тому, що для підрахунку результату знадобиться кількість тактів генератора, яка дорівнює значенню вхідного слова. Тобто для підрахунку значення факторіалу цифри 5 знадобиться п'ять тактів. Коли результат готовий, вихід ready встановлюється в рівень лог.1. Роботу модуля демонструє часова діаграма, що приводиться на рис. 9.6.

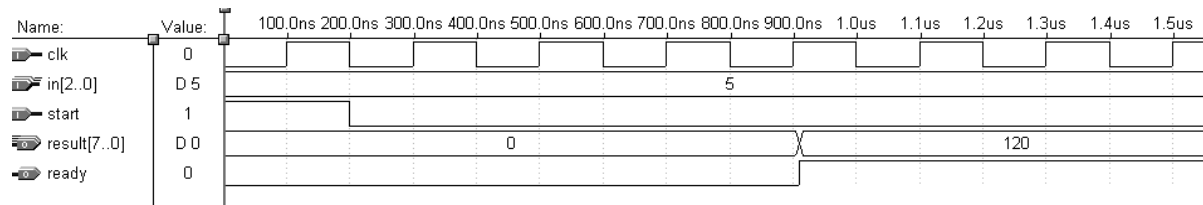


Рис. 9.6.

Спробуємо описати модуль, який підраховував би значення наступної функції:

$$y = a \cdot x + b \cdot x.$$

Опис модуля має наступний вигляд:

```

module func(y, x, r);           // початок опису модуля
output y;                       // оголошення вихідних портів
input x,r;                      // оголошення вхідних портів
wire [3:0] x;
reg [3:0] a;                    // оголошення додаткових змінних регістрового типу
reg [3:0] b;
reg [11:0] y;                   // розрядність вихідного порту дорівнює 12
    always @ (x or r)           // очікуємо зміни сигналів на вході
    begin
        a=4'd3;                 // присвоюємо змінній значення 3
        b=4'd7;                 // присвоюємо змінній значення 7
        if (r==1) y=a*x+b*x;    // якщо r=1 обчислюємо значення функції
    end
endmodule                      // кінець опису модуля

```

У конструкції контролю подій **always** відбувається стеження за зміною кожного з сигналів на вході. Коли відбувається зміна стану будь-якого з вказаних сигналів, на виході „y” встановлюється значення функції при умові, що на вході r присутня лог.1. Взагалі, змінні „a” та „b” можна в даному прикладі зробити змінними типу **integer**, оскільки вони використовуються лише в тілі модуля.

Цей приклад демонструє зручність опису подібних конструкцій, не вдаючись в особливості їх апаратної реалізації. Це завдання виконує компілятор. Для мікросхеми сімейства FLEX8000 EPF8282ALC84-2, на реалізацію описаної функції знадобилося 14% логічних комірок, а для реалізації модуля підрахунку факторіалу цей показник становить 37%.

У бібліотеці параметризованих модулів системи Quartus є перемножувач lpm_mult. Цей модуль зручно використовувати якщо відбувається проектування як за допомогою опису проекту на одній з HDL-мов, так і у Графічному редакторі. Використання параметризованого модуля в якості перемножувача двох п'ятирозрядних слів демонструє наступний приклад.

```
module multiplier4x4 (dataa, datab, result); // початок опису модуля
input [4:0] dataa; // вхідні шини даних
input [4:0] datab;
output [9:0] result; // результат множення
wire [9:0] sub_wire0; // додатковий ланцюг
wire [9:0] result = sub_wire0[9:0]; // з'єднання шини даних з ланцюгом
lpm_mult lpm_mult_component (.dataa (dataa), .datab (datab),
.result (sub_wire0));
// використовуємо параметризований модуль з бібліотеки
defparam // встановлення значень параметрів
    lpm_mult_component.lpm_widtha = 5,
    lpm_mult_component.lpm_widthb = 5,
    lpm_mult_component.lpm_widthp = 10,
    lpm_mult_component.lpm_widths = 10,
    lpm_mult_component.input_b_is_constant = "NO",
    lpm_mult_component.lpm_representation = "UNSIGNED",
    lpm_mult_component.use_eab = "OFF";
endmodule // кінець опису модуля
```

9.3. Оперативна пам'ять

Пам'ять, як витікає з назви, призначена для запам'ятовування, зберігання масивів інформації, цифрових таблиць, груп цифрових кодів. Кожний код зберігається в окремому елементі пам'яті, що називається коміркою пам'яті. Основна функція будь-якої пам'яті якраз і полягає у видачі цих кодів на виходи мікросхеми по зовнішньому запиту. Основний параметр пам'яті – це її об'єм, тобто кількість кодів, що можуть у ній зберігатися, а також розрядність цих кодів. Синтаксис опису пам'яті на Verilog має наступний вигляд:

```
reg [<розрядність елементів>] <ідентифікатор> [<кількість елементів>];
```

Принцип організації пам'яті записується наступним чином: спочатку записується розрядність коду, що зберігається в одній комірці, а потім записується кількість комірок. Нагадаємо, що приведена конструкція пам'яті підтримується компілятором Verilog тільки в САПР Quartus.

В залежності від способу занесення інформації в пам'ять та від способу зберігання інформації мікросхеми пам'яті розділяються на наступні основні типи:

- Постійна пам'ять. Інформація записується один раз на етапі програмування мікросхеми (заноситися користувачем) і не знищується при вимиканні живлення.
- Оперативна пам'ять, запис інформації в яку найбільш простий і може бути організований користувачем будь-яку кількість разів протягом всього періоду роботи мікросхеми. Інформація в пам'яті знищується при вимиканні живлення.

В загальному випадку будь-яка мікросхема пам'яті має наступні інформаційні виводи: адресні виводи, виводи даних, керуючі виводи. Для демонстрації прикладів з елементами пам'яті буде використана система

проектування Quartus. Розглянемо опис на мові Verilog модуля пам'яті на вісім напівбайтових слів. Текст опису модуля приводитися нижче.

```
module memor(out, in, addr, clk, enable);
output [3:0] out;           // визначаємо вихідний порт
input [3:0] in;             // вхідний інформаційний порт
input [2:0] addr;           // вхідний адресний порт
input clk;                  // тактовий сигнал
input enable;               // сигнал «запис / зчитування»
reg [3:0] temp[7:0];        // пам'ять на зберігання 8 слів
reg [3:0] out;              // вихідний порт має регістровий тип
always @ (posedge clk)      // по фронту тактового сигналу
begin
    if (enable)              // якщо встановлено сигнал запису
        begin
            case (addr)       // записуємо слово в зазначену комірку
            0: temp[0]=in;
            1: temp[1]=in;
            2: temp[2]=in;
            3: temp[3]=in;
            4: temp[4]=in;
            5: temp[5]=in;
            6: temp[6]=in;
            7: temp[7]=in;
            default: temp[4]=0;
            endcase
        end
    else                      // якщо встановлено сигнал читання
        begin
            case (addr)       // зчитуємо зазначену комірку
            0: out=temp[0];
            1: out=temp[1];
            2: out=temp[2];
            3: out=temp[3];
            4: out=temp[4];
            5: out=temp[5];
            6: out=temp[6];
            7: out=temp[7];
            default: out=0;
            endcase
        end
    end
end
endmodule                   // кінець опису модуля
```

В описанні модуля було використано конструкцію вибору **case**. Кожного разу по фронту сигналу **clk**, в залежності від коду на адресних входах, відбувається вибір комірки пам'яті. Рівнем сигналу на вході **enable** вибирається режим роботи мікросхеми (запис/зчитування). Оперативна пам'ять буває двох основних видів: з розділеними шинами вхідних і вихідних даних (в основному це однорозрядна пам'ять) і з двонапрямленою шиною вхідних і вихідних даних (це багаторозрядна пам'ять). Ми

розглянули опис модуля пам'яті з розділеними шинами вхідних та вихідних даних. Часова діаграма, отримана при моделюванні, приводиться на рис. 9.7.

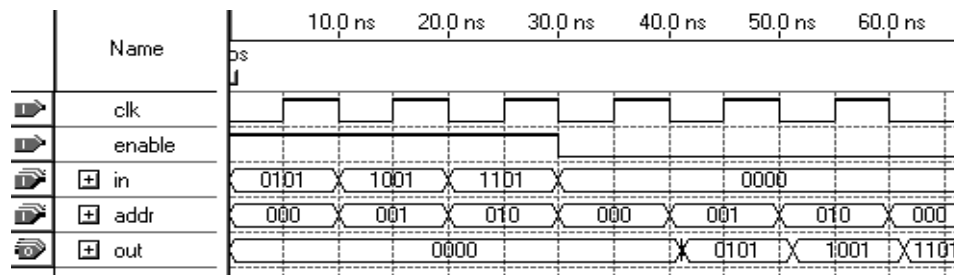


Рис. 9.7.

Це лише один із способів опису модуля пам'яті. Легко бачити, що даний спосіб має суттєвий недолік: необхідно описати всі можливі альтернативи в конструкції вибору і при збільшенні об'єму пам'яті пропорційно збільшується об'єм коду. Такий підхід зводить нанівець всю зручність мови Verilog. Розглянемо приклад, в якому доступ до комірки пам'яті відбувається іншим способом. Опис параметризованого модуля ОЗП, створеного у САПР Quartus, представлений нижче.

```

module mem (DataOut ,Addr, DataIn, CS, WE, OE); // заголовок модуля
parameter ADDRESS_SIZE = 4; // розрядність адресної шини
parameter WORD_SIZE = 8; // розрядність шини даних
output [WORD_SIZE-1:0] DataOut; // вихідна шина даних
input [ADDRESS_SIZE-1:0] Addr; // вхідна адресна шина
input [WORD_SIZE-1:0] DataIn; // вхідна шина даних
input CS, WE, OE; // входи керування
reg [WORD_SIZE-1:0] DataOut; // вихід має регістровий тип
reg [WORD_SIZE-1:0] Mem [ADDRESS_SIZE-1:0];
    always @(CS or WE or OE) // при зміні сигналів керування
    begin
        if (!CS && !WE) Mem[Addr] = DataIn; // запис даних в ОЗП
        else if (!CS && !OE) DataOut=Mem[Addr]; // зчитування даних
    end
endmodule // кінець опису модуля

```

Запис у комірку пам'яті, яка задається значенням на адресному вході Addr, відбувається при низькому рівні сигналів CS та WE (вибір мікросхеми та дозвіл запису). Зчитування даних відбувається при низькому рівні сигналів на входах CS та OE (вихід дозволений). Зміна станів зазначених сигналів відслідковується в конструкції **always**. Доступ до комірки пам'яті Mem відбувається по аналогії з доступом до елементу масиву у традиційних мовах програмування. Діаграма на рис. 9.8 демонструє роботу модуля ОЗП.

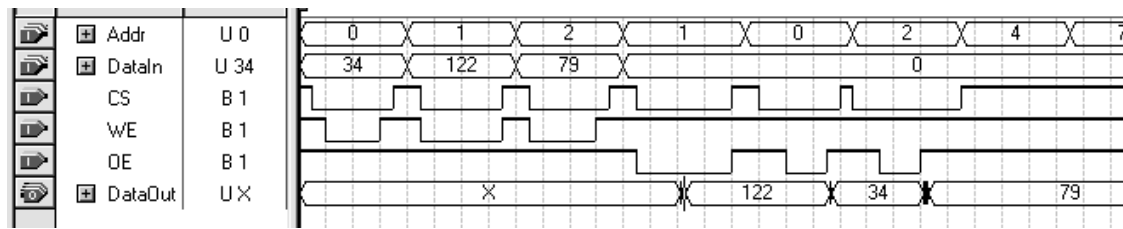


Рис. 9.8.

При роботі з пам'яттю є можливість доступу до визначеної комірки від деякого пристрою або від іншої частини схеми як з операцією запису, так і з операцією зчитування.

Розглянемо ще один приклад використання пам'яті. Наступний модуль представляє з себе детектор максимального (max) та мінімального (min) значень вхідного коду. Крім того, модуль має вихід, величина на якому представляє з себе усереднене значення останніх 9 відліків вхідного коду (din). Ці відліки зберігаються у пам'яті (memory). Опис модуля приводиться нижче.

```

module maxmin(max, min, avg, din, clk, en, reset); // заголовок модуля
parameter WIDTH=10; // розрядність даних
parameter DEPTH=10; // кількість комірок пам'яті
output [WIDTH-1:0] max, min, avg; // вихідні порти
input [WIDTH-1:0] din; // вхід даних
input clk, en, reset; // тактовий вхід, дозволу роботи та обнуління
reg [WIDTH-1:0] max, min, avg; // виходи мають регістровий тип
reg [WIDTH:0] tavg; // змінна для зберігання середнього значення
reg [WIDTH-1:0] memory [DEPTH-1:0];
// пам'ять на DEPTH-1 елементів
reg [WIDTH-1:0] tmax, tmin; // регістри для зберігання значень
integer i; // змінна для роботи циклу
always @(posedge clk or posedge reset) // перший процедурний блок
begin
  if (reset==1) // якщо виконується умова
  begin
    tmax=0; // обнуління змінної
    tmin=10'hFFF; // присвоєння максимального значення
    tavg=0; // середнє значення дорівнює 0
    for(i=DEPTH-1;i>=0;i=i-1) memory[i]=0; // очищення пам'яті
  end // if
  else if (en==1) // якщо встановлений сигнал дозволу
  begin
    tmax=(tmax>din)?tmax:din; // перевірка максимального значення
    tmin=(tmin<din)?tmin:din; // перевірка мінімального значення
    for(i=DEPTH-1;i>0;i=i-1) // в циклі відбувається зсув
    begin // даних на 1 комірку
      memory[i]=memory[i-1];
      tavg=(tavg+memory[i])>>1;
    end // for
  end
  memory[0]=din;

```

```
// нульовій комірці присвоюється значення на вході
end                                     // else if
end
always @(tmax or tmin or tavg)// другий процедурний блок
begin
max<=tmax;                             // установка на виходах максимального
min<=tmin;                             // мінімального
avg<=tavg;                             // та середнього значень
end
endmodule
```

Параметри WIDTH та DEPTH визначають розрядність вхідних даних і кількість комірок пам'яті. По сигналу сбросу reset відбувається обнуління внутрішніх змінних, а також сигналів на виходах модуля. На виході min встановлюється максимальне значення. По фронту тактового сигналу clk відбувається порівняння вхідних даних зі значеннями, які зберігаються у внутрішніх змінних. При виконанні умови в процедурному блоці, на виходах встановлюються максимальне, мінімальне та середнє значення. За допомогою циклу відбувається зсув вмісту комірок пам'яті. Значення на вході присвоюється комірці з адресою 0. Діаграма, що представлена на рис. 9.9, демонструє роботу детектора максимального та мінімального значень.

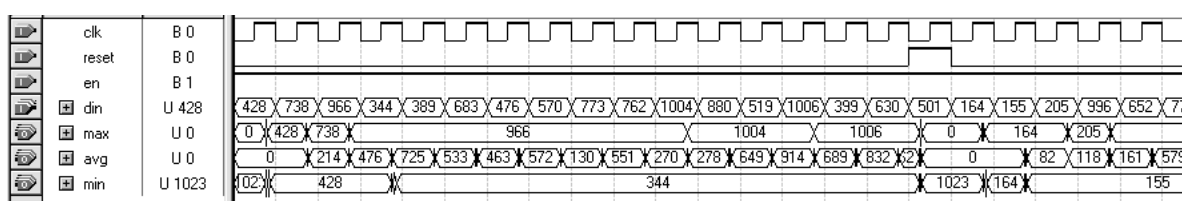


Рис. 9.9.

В залежності від того, в якому порядку може записуватися або зчитуватися інформація, існують два різновиди ОЗП:

- ОЗП з паралельним довільним доступ (це найбільш універсальна схема);
- ОЗП з послідовним доступом (це більш специфічна схема).

Паралельний (або довільний) доступ найбільш простий і не потребує ніяких додаткових елементів, оскільки саме на цей режим безпосередньо розраховані мікросхеми пам'яті. В цьому режимі можна записувати інформацію по будь-якій адресі ОЗП і зчитувати інформацію з будь-якої адреси ОЗП в довільному порядку. Однак паралельний доступ потребує формування досить складних послідовностей всіх вхідних сигналів пам'яті. Тобто для запису інформації необхідно сформувати код адреси комірки і лише потім подати дані, що супроводжуються керуючими сигналами запису інформації та вибору мікросхеми. Так само необхідно подавати повний код адреси комірки при операції зчитування. Цей режим найчастіше використовується в комп'ютерах та контролерах, де

найголовнішими факторами є універсальність і гнучкість використання пам'яті для будь-якої мети.

Послідовний доступ до пам'яті пропонує більш простий порядок звернення до пам'яті. В цьому випадку не потрібно задавати код адреси комірки, в яку відбувається запис або зчитування даних, оскільки адреса пам'яті формується схемою автоматично. Для запису інформації потрібно подати код даних, що записуються і сигнал запису. Для зчитування інформації потрібно подати сигнал зчитування і отримати дані. Автоматичне задання адреси при цьому відбувається внутрішніми лічильниками, що змінюють свій стан при кожному зверненні до пам'яті. Наприклад, десять послідовних циклів запису запишуть інформацію в десять послідовно розташованих комірок пам'яті. Недолік такого підходу зрозумілий: ми не маємо можливості записувати або зчитувати дані з комірок з довільними адресами в будь-якому порядку. Однак суттєво спрощується і прискорюється процедура обміну з пам'яттю.

9.3.1. Пам'ять типу LIFO

Можна виділити три основні типи оперативної пам'яті з послідовним доступом:

- пам'ять магазинного типу, стекового типу, що працює за принципом “останній увійшов – перший вийшов” (LIFO, Last In – First Out);
- пам'ять типу “перший увійшов – перший вийшов” (FIFO, First In – First Out);
- пам'ять для зберігання масивів даних.

Розглянемо опис модуля пам'яті типу LIFO у САПР Quartus. Текст програми має наступний вигляд:

```
module lifo_buf(out, full, empty, ind, clk, in, enable, push_pop);
output [3:0] out;           // вихідний порт
output full;               // сигнал переповнення
output empty;              // сигнал „буфер порожній”
output [3:0] ind;           // кількість елементів у буфері
input [3:0] in;             // вхідний порт
input enable;              // вхід дозволу
input clk;                 // тактовий сигнал
input push_pop;            // запис(зчитування) в (з) буфера
reg [3:0] temp[7:0];        // пам'ять для зберігання 8 напівбайтових слів
reg [3:0] out;              // вихідні порти мають регістровий тип
reg full;
reg empty;
reg [3:0] ind;
integer index;              // лічильник слів у буфері
always @(posedge clk)       // по фронту тактового сигналу
begin
    if (enable)              // якщо є присутнім сигнал дозволу
    begin
        if (push_pop==1)     // перевіряємо стан вхідного порту
            begin
```

```
        if (index==7) full=1;      // якщо буфер заповнений,  
//установлюємо високий рівень на виході переповнення  
        else      // інакше  
        begin  
            temp [index]=in;    // запис слова в пам'ять  
            index=index+1;      // збільшення значення індексу  
            ind=index;          // вивід на порт кількості слів в буфері  
            full=0;             // обнуління сигналу переповнення  
            empty=0;  
        end  
    end  
else  
begin  
    if (index>0)                // перевіряємо, чи є в буфері дані  
    begin                      // якщо є  
        out=temp[index]; // зчитуємо слово з пам'яті  
        index=index-1;    // зменшуємо індекс  
        ind=index;        // встановлюємо на вихідному  
                        // порту значення елементів буфера  
        empty=0;          // обнуління сигналу «буфер порожній»  
    end  
    else if (index==0)         // якщо буфер порожній  
    begin  
        empty=1;             // установка сигналу «буфер порожній»  
    end  
end  
end  
else                          // якщо сигнал дозволу має рівень 0  
begin  
    index=0;                  // обнуління значення індексу  
    out=0;                    // установка сигналу низького рівня на  
    full=0;                   // вихідних портах  
    ind=0;  
    empty=1;  
end  
end  
endmodule                     // кінець опису модуля
```

Як зазначалося вище, формування адреси в такому типі пам'яті відбувається внутрішнім лічильником. В описаному модулі цим лічильником є змінна `index`. Встановлення значення цієї змінної в 0 відбувається по фронту сигналу `clk` при умові того, що вхід `enable` має низький рівень. Взагалі, при низькому рівні сигналу на цьому вході, всі вихідні порти модуля, окрім сигналу `empty`, встановлюються в рівень логічного нуля. Якщо сигнал на вході `enable` має високий рівень, відбувається збільшення, або зменшення (в залежності від стану вхідного порту `push_por`) змінної `index`, а потім і звернення до визначеної комірки пам'яті. Якщо змінна `index` приймає значення 7, то вихідний порт `full` переходить в стан логічної одиниці, що сигналізує про те, що всі комірки пам'яті заповнені. Якщо змінна `index` дорівнює 0, то високий рівень

сигналу `empty` сигналізує про те, що буфер пустий. Часова діаграма, яка демонструє роботу модуля, приводиться на рис. 9.10.

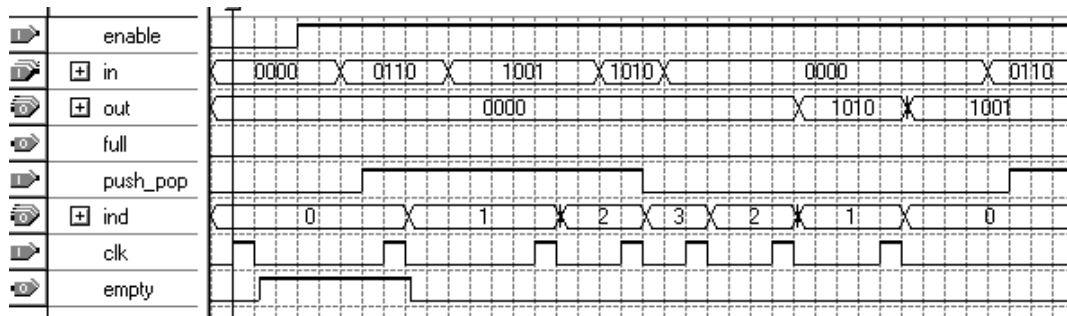


Рис. 9.10.

Як бачимо з часової діаграми, пам'ять LIFO видає дані у зворотній послідовності від тієї, у якій вони були записані. Пам'ять LIFO використовується в комп'ютерах (стек), де вона зберігає інформацію про параметри програм і підпрограм. Для пам'яті LIFO достатньо лише одного коду адреси.

Разом із системою проектування поставляється бібліотека параметризованих модулів. Від розробника вимагається лише визначити кількість елементів пам'яті та розрядність цих елементів. Це робиться шляхом перевизначення параметрів за допомогою службового слова **defparam**. Розглянемо опис модуля елемента пам'яті.

```
module ram256x8 (data, address, we, inclock, outclock, q);
input  [7:0] data;           // вхідна шина даних
input  [7:0] address;       // адресна шина
input  we, inclock, outclock; // керуючі сигнали
output [7:0] q;             // вихідна шина даних
lpm_ram_dq inst_1 (.q (q), .data (data), .address (address), .we (we),
.inclock (inclock), .outclock (outclock));
// використовуємо модуль пам'яті з бібліотеки
// параметризованих модулів
defparam inst_1.lpm_width = 8;
// визначаємо розрядність шини даних
defparam inst_1.lpm_widthad = 8;
// визначаємо розрядність адресної шини
endmodule                    // кінець опису модуля
```

Нічого нового з синтаксичних конструкцій в наведеному описі немає. Все дуже просто. Спочатку визначаємо тип та розрядність зовнішніх портів модуля, а потім підключаємо модуль пам'яті з бібліотеки параметризованих модулів. Далі встановлюємо розрядність елементів (8) та їх кількість (256) в описуваному модулі. В цьому прикладі параметри `lpm_width` та `lpm_widthad` використані для визначення розрядності вхідної шини даних, а також кількості слів даних, що будуть розміщені в пам'яті (розрядність адресної шини). Функції `lpm_ram_dq` потрібно передати всі

параметри, які вказані в її прототипі. У випадку, коли не вказана розрядність та кількість елементів у модулі пам'яті, система MAX+plus II автоматично встановлює їх. Ці дані беруться з include-файлу потрібної функції. У випадку, коли файл з описом на Verilog є головним файлом проекту, компілятор використовує дані, які вказані в глобальних параметрах проекту (Global Project Parameters).

9.3.2. Пам'ять типу FIFO

Пам'ять FIFO видає дані в тій самій послідовності, у якій вони були записані. Пам'ять FIFO можна порівняти з регістром зсуву, на виході якого дані з'являються в тій самій послідовності, у якій вони були записані в нього. Функціональна схема пам'яті типу LIFO простіша за структурою, ніж схема пам'яті FIFO, оскільки вона містить лише один лічильник і не потребує мультиплексування. Для обох типів пам'яті функціональні схеми досить складні, і в залежності від розрядності шини даних можуть суттєво змінюватися.

Опишемо модуль пам'яті типу FIFO. Опис на мові Verilog приводиться нижче.

```
module fifo_buf (data, wrreq, rdreq, clock, aclr, q, full, empty);  
input [7:0] data;           // вхідна 8-розрядна шина даних  
input wrreq;               // сигнал запису  
input rdreq;               // сигнал зчитування  
input clock;               // тактовий сигнал  
input aclr;                // сигнал обнуління  
output [7:0] q;            // вихідна шина  
output full;               // сигнал "буфер повний"  
output empty;              // сигнал "буфер пустий"  
wire sub_wire0;            // додаткові ланцюги для роботи модуля  
wire [7:0] sub_wire1;  
wire sub_wire2;  
wire empty = sub_wire0;  
wire [7:0] q = sub_wire1[7:0];  
wire full = sub_wire2;  
// використовуємо модуль з бібліотеки параметризованих модулів  
    SCFIFO      SCFIFO_component (.rdreq (rdreq), .aclr (aclr), .clock (clock),  
.wrreq (wrreq), .data (data), .empty (sub_wire0), .q (sub_wire1), .full (sub_wire2));  
    defparam  
        SCFIFO_component.LPM_WIDTH = 8,    // розрядність слів  
        SCFIFO_component.LPM_NUMWORDS = 256, // кількість слів  
        SCFIFO_component.LPM_SHOWAHEAD = "OFF",  
        SCFIFO_component.USE_EAB = "ON";  
endmodule                  // кінець опису модуля
```

Функціональна схема пам'яті типу FIFO містить в собі два лічильника – лічильник запису та лічильник зчитування, вихідні коди яких мультиплексуються за допомогою 2-канального мультиплексора. Запис даних відбувається по сигналу запису wrreq, зчитування – по сигналу

зчитування rdreq. Дані повинні виставлятися раніше появи сигналу запису, а закінчуватися після зняття сигналу на запис. Запис починається з нульової адреси пам'яті і відбувається послідовно по зростаючих адресах. Так само зчитування починається з нульової адреси пам'яті і відбувається по послідовно зростаючих адресах. Операції запису і зчитування можуть чергуватися. Часова діаграма, яка демонструє роботу модуля, приводиться на рис. 9.11.

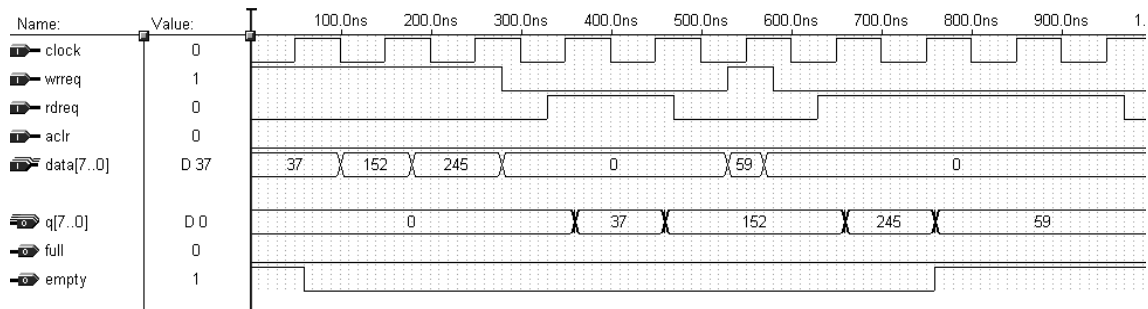


Рис. 9.11.

Важливе використання мікросхем оперативної пам'яті полягає в організації різноманітних інформаційних буферів, тобто буферної пам'яті для тимчасового зберігання даних, що передаються між двома пристроями або системами. Суть інформаційного буферу полягає в наступному: передаючий пристрій записує дані в буфер, а приймаючий пристрій зчитує дані з буферу. Зберігання масивів у пам'яті передбачає, що спочатку в пам'ять записується великий масив даних, а потім цей масив повністю зчитується. Таке проміжне зберігання дозволяє краще узгодити роботи пристроїв, які беруть участь в обміні даними, підвищують їх незалежність один від одного, узгоджують швидкості передачі та прийому даних. Це звичайний режим роботи процесора (швидкодіючого) з повільною периферією і називається режимом прямого доступу до пам'яті (ПДП), який забезпечується за допомогою контролерів і узгодженості асинхронного обміну.

Нехай, наприклад, в якості першого пристрою виступає комп'ютер, а в якості іншого – кабель локальної мережі. Комп'ютеру значно зручніше видавати дані зі швидкістю, що визначається його власною швидкодією, але в локальну мережу потрібно передавати дані з визначеною швидкістю, що задається стандартом на мережу. Крім того, комп'ютер, по можливості, не повинен відволікатися на спостереження за станом мережі. Тому буферна пам'ять в даному випадку необхідна. Так само вона потрібна при прийомі даних з локальної мережі в комп'ютер.

Головна відміна буферної пам'яті від пам'яті для тимчасового зберігання інформації полягає в тому, що до інформаційного буферу завжди мають доступ не один зовнішній пристрій, а два (і більше). Тому іноді суттєво ускладнюється як схема задання адреси мікросхеми пам'яті, так і схема розділення потоків даних.

9.4. Постійна пам'ять

Раніше були розглянуті приклади обчислення факторіала, множення двох чисел. Наприклад, для обчислення факторіала числа 5 в запропонованому варіанті модуля було потрібно 5 тактів генератора. Зі збільшенням числа пропорційно збільшується час, який затрачується на визначення значення факторіала. Крім того, істотний недолік полягає в тому, що затрачується велика кількість ресурсів мікросхеми. Тому для вирішення подібних завдань і не тільки вигідно використовувати постійний запам'ятовуючий пристрій. ПЗП може бути використаний для зберігання набору чисел, що відтворюють звук або мелодію, таблиці виправлень при лінеаризації, містити значення різних функцій. При цьому розрядність як адресної шини, так і внутрішніх комірок можна легко змінити і тим самим уникнути надлишкового використання ресурсів мікросхеми, як це може бути при використанні зовнішніх мікросхем пам'яті.

Система проектування Quartus підтримує бібліотеку LPM-функцій. Компанія Altera рекомендує в проектах використовувати для реалізації пам'яті саме цю бібліотеку.

Розглянемо приклад реалізації ПЗП, описаного на Verilog. Завданням описуваного модуля є зведення у квадрат чисел. Опис модуля має вигляд:

```
module rom_test(address, outclock, q);    // початок модуля
input  [3:0] address;                  // адресна шина
input  outclock;                       // тактовий сигнал
output [7:0] q;                        // вихідна шина даних
wire [7:0] sub_wire0;                 // 8-розрядний ланцюг
wire [7:0] q = sub_wire0[7:0];
    lpm_rom      lpm_rom_component (.outclock (outclock), .address (address),
    .q (sub_wire0));    // підключення модуля ПЗП з бібліотеки САПР
defparam                                // встановка значень параметрів
    lpm_rom_component.lpm_width = 8,
    lpm_rom_component.lpm_widthad = 4,
    lpm_rom_component.lpm_address_control = "UNREGISTERED",
    lpm_rom_component.lpm_outdata = "REGISTERED",
    lpm_rom_component.lpm_file = "D:/max2work/ram_for_sc.mif";
endmodule
```

В наведеному описі використаний модуль постійного запам'ятовуючого пристрою «lpm_rom» із синхронним зчитуванням даних. У довідковій системі можна знайти повну інформацію, необхідну для використання параметризованих модулів в своїх проектах. Для прикладу розглянемо фрагмент із довідкової системи, що описує порядок роботи з описаним модулем. Все, що необхідно знати розробнику для використання цього типу пам'яті – це її інтерфейс (прототип функції, сигнатуру), а також необхідна інформація про параметри. Нижче приводиться опис прототипу функції пам'яті.

```
FUNCTION lpm_rom (address[LPM_WIDTHAD-1..0], inclock, outclock, memenab)
WITH (LPM_WIDTH, LPM_WIDTHAD, LPM_NUMWORDS, LPM_FILE,
LPM_ADDRESS_CONTROL, LPM_OUTDATA)
RETURNS (q[LPM_WIDTH-1..0]);
```

Ця функція описана мовою AHDL, однак імена портів також можуть бути використані для мови Verilog HDL. Після назви функції, в дужках, міститься перелік вхідних сигналів. Після слова RETURNS зазначена назва та розрядність вихідного порту. Розрядності вхідних і вихідних портів вказані у вигляді параметрів. Назви параметрів, яким обов'язково необхідно присвоїти певне значення при роботі з модулем, перераховані в дужках після слова WITH. В табл. 9.2 приводиться опис кожного з параметрів.

Табл.9.2.

Назва параметра	Опис
LPM_WIDTH	Розрядність порту q[]
LPM_WIDTHAD	Розрядність шини адресу address[]. Параметр повинен дорівнювати $\log_2(\text{LPM\_NUMWORDS})$. Якщо розрядність шини менше, деякі з комірок пам'яті будуть недоступні. Якщо розрядність занадто велика, то при звертанні до незаповнених комірок значення на шині даних буде невизначеним.
LPM_NUMWORDS	Кількість слів, збережених у пам'яті. В загальному випадку, ця величина повинна бути дорівнює $2^{\text{LPM\_WIDTHAD}}$.
LPM_FILE	Шлях до файлу ініціалізації файлу (Memory Initialization File (.mif)).
LPM_ADDRESS_CONTROL	Може приймати значення "REGISTERED", "UNREGISTERED", і "UNUSED". Показує, чи зареєстрований порт адреси.
LPM_OUTDATA	Може приймати значення "REGISTERED", "UNREGISTERED", і "UNUSED".

	Показує, чи зареєстрований порт q[].

В розглянутому прикладі розрядність адресної шини дорівнює 4 (що дозволяє адресувати 16 комірок), розрядність збережених даних дорівнює 8. Розглянемо структуру файлу ініціалізації пам'яті. В коментарях приводиться опис кожного рядка файлу.

```
DEPTH=16;           // кількість збережених слів
WIDTH=8;            // розрядність збережених слів
ADDRESS_RADIX=DEC;  // адреса комірки записується в десятковій
                   // системі счислення
DATA_RADIX=DEC;     // дані записані в десятковій системі счислення.
CONTENT             // опис вмісту пам'яті
BEGIN              // початок
    0:0; 1:1;       // записані «адреса комірки»:«вміст комірки»;
    2:4; 3:9; 4:16; 5:25; 6:36;
    7:49; 8:64; 9:81; 10:100;
    11:121; 12:144; 13:169;
    14:196; 15:225;
END                // кінець опису
```

В даному файлі записана таблиця квадратів чотирьохрозрядних чисел. Після компіляції проекту і перевірки в Сигнальному редакторі, була отримана часова діаграма, представлена на рис. 9.12.

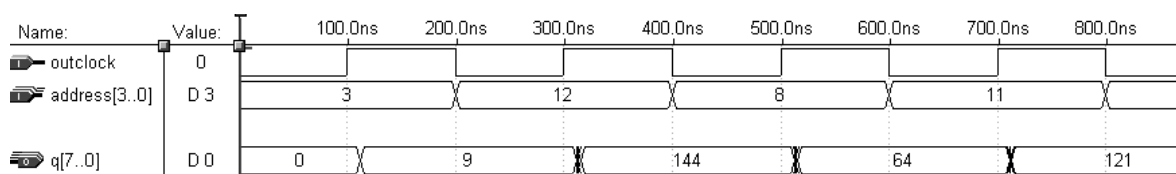


Рис. 9.12.

9.5. Розробка асинхронних приймача і передавача

Асинхронна передача даних широко використовується для обміну інформацією в мікропроцесорних системах. При такому способі передачі синхронізуючі події (початок передачі, закінчення передачі, помилки і т.п.), а також дані передаються по загальній лінії. Синхронізуюча інформація міститься в часових параметрах передачі, тому немає необхідності у використанні додаткових сигналів. В передавачі дані перетворюються з паралельного коду в послідовність бітів.

Передача слова даних при асинхронному зв'язку починається з переходу лінії з високого рівня в низький. Це ініціює схему приймача на прийом слова даних. В стані лог.0 лінія перебуває протягом періоду передачі одного біта. Перший біт, що передається від передавача до приймача завжди нульовий, і називається стартовим битом (Start bit). Потім послідовно передаються біти даних, починаючи з молодшого біта.

Останнім передається стоповий біт (Stop bit), який має високий рівень. В загальному випадку, посилка даних може містити довільну кількість біт даних (використовуються 5, 6, 7 або 8 біт), один стартовий біт. Після бітів даних може передаватися біт парності. Завершують передачу стопові біти. Формат посилки даних обов'язково узгоджується між приймачем і передавачем. Якщо слова передаються один за іншим, то мінімальний інтервал між словами даних становить тривалість передачі стопових біт.

Швидкість послідовної передачі вимірюється в бітах у секунду (б/с, b/s), або кілобитах у секунду (кб/с, kb/s). Для послідовного обміну даних прийнятий наступний ряд швидкостей: 50, 110, 150, 300, 600, 1200, 2400, 3600, 4800, 7200, 9600, 19200, 38400, 115200 б/с.

Наведені нижче приклади приймача і передавача не мають повний набір можливостей. При необхідності, можна буде змінити ці пристрої таким чином, щоб їх функції задовольняли конкретному завданню.

Для розробки асинхронного приймача виконаємо структурну декомпозицію проекту для того, щоб розділити реалізацію окремих модулів проекту. Кожен модуль може бути промодельований. Окремі модулі проекту поєднуються в єдину бібліотеку проекту. В бібліотеці перебувають символи кожного модуля, які далі можна використати або в графічному редакторі, або як окремі модулі у текстовому редакторі. В розглянутому нижче прикладі розробляється асинхронний приймач із використанням різних засобів опису (Графічний і Текстовий редактор), створений ряд файлів з описом окремих модулів, а також використані елементи із бібліотеки САПР. Проект створюється в системі MAX+plus II.

Схема розробленого асинхронного приймача в системі MAX+plus II приводиться на рис. 9.13.

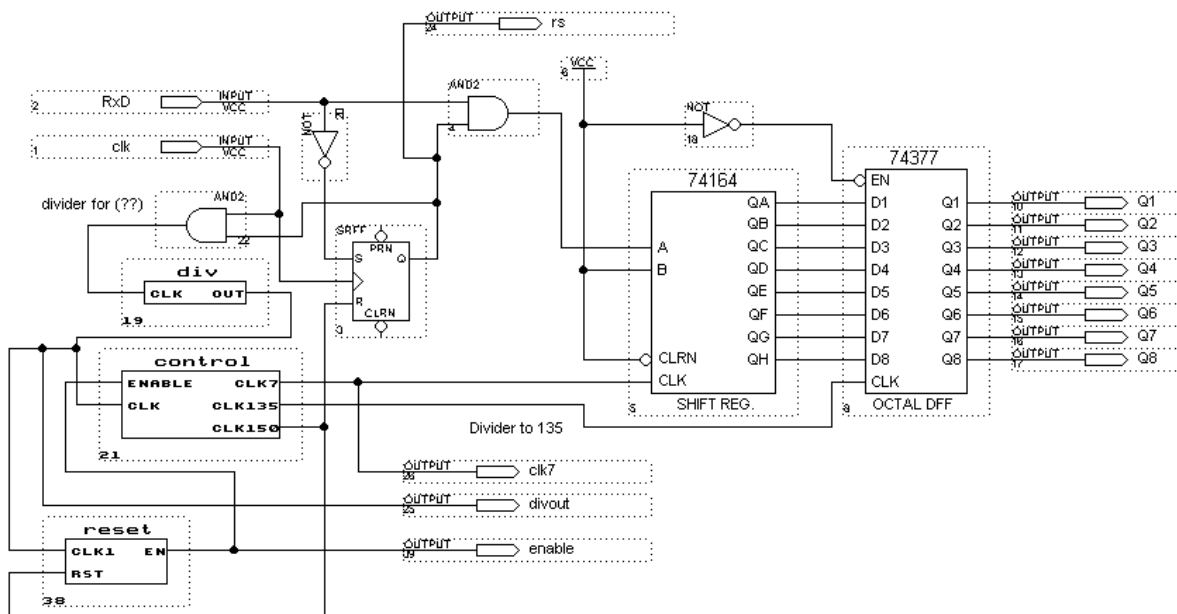


Рис. 9.13.

При відсутності даних, на лінії RxD встановлений сигнал високого рівня. В момент початку передачі чергового байту даних безпосередньо перед самими даними передається стартовий біт. Цей стартовий біт завжди має низький рівень. Таким чином, перехід сигналу на лінії з високого рівня в низький сигналізує про початок передачі даних. Якщо сигнал на лінії має високий рівень, на вході s тригера присутній сигнал низького рівня (це забезпечує інвертор). При переході сигналу на лінії зі стану з високим рівнем сигналу в стан з низьким рівнем, на вході тригера з'являється лог.1. На виході тригера встановлюється сигнал високого рівня, який дозволяє роботу тактового генератора і прийом даних шляхом переведу логічних елементів 2І в режим повторювача, і на вході А регістра зсуву будуть з'являтися біти, що відповідають прийнятому слову. Елемент Control, описаний мовою Verilog, виконує наступну функцію. На 7 такті сигналу генератора clk, на виході clk7 елемента з'являється імпульс, тривалістю 1 період тактового генератора. Цей імпульс з'являється в момент часу, що відповідає середині прийнятого біта. В цей момент відбувається зсув даних в регістрі 74164. Після того, як буде прийнятий 8 біт даних, на виході clk130 елемента control з'являється одиночний імпульс, що подається на вхід запису паралельного регістра 74377. Таким чином, на виході паралельного регістра з'являється прийнятий байт даних. Після цього з'являється імпульс на виході clk150, який переводить схему в режим очікування прийому чергового байта даних. Відбувається обнуління тригера. На його виході з'являється сигнал низького рівня. Цей сигнал блокує роботу регістра зсуву, забороняє тактовий сигнал. Модуль reset забезпечує обнуління внутрішніх змінних в елементі control. Шляхом зміни коефіцієнту ділення частоти в модулі div, приймач налаштовується на різні швидкості прийому даних.

Розглянемо опис використаних модулів. Програма, що описує дільник частоти div має вигляд:

```
module div(out,clk);           // початок опису модуля
output out;                   // оголошення вхідного сигналу
input clk;                     // оголошення вихідного сигналу
reg out;
reg [5:0] temp;                // тимчасовий регістр для підрахунку імпульсів генератора
always @(posedge clk)         // по фронту тактового сигналу
begin
    temp=temp+1;                // збільшуємо значення додаткового регістра
    if (temp==13)                // кожні 26 тактів генератора
        begin
            temp=0;              // обнуління значення змінної
            out=~out;            // інвертуємо значення на вихідному порту
        end
    end
end
endmodule                     // кінець опису модуля
```

Коефіцієнт ділення частоти змінюється шляхом встановлення іншого значення в дужках умовного оператора **if**. При великих значеннях дільника частоти можливо доведеться збільшити розрядність регістра `temp`.

Після того, як виконано опис модуля дільника частоти, з нього потрібно зробити символ. Для цього з меню **File** слід вибрати команду **Create Default Symbol**. Розглянемо опис модуля `control`:

```
module control(clk7,clk135,clk150,enable,clk); // початок модуля
output clk7, clk135, clk150;           // оголошення вихідних портів
input clk, enable;                     // оголошення вхідних портів
reg clk7, clk135, clk150;              // виходи мають реєстровий тип
reg [3:0] count15;                     // розрядність лічильника дорівнює 4
reg [7:0] count150;
always @ (posedge clk)                // по фронту тактового сигналу
begin
    if (enable)                         // якщо високий рівень на вході en
        begin
            clk7<=0;                    // обнуління значення внутрішніх змінних
            clk135<=0;
            clk150<=0;
            count15<=0;                 // на виходах встановлено 0
            count150<=0;
        end
    else
        begin
            count15<=count15+1; // збільшуємо лічильники на 1
            count150<=count150+1;
            if (count15==7)             // перевіряємо, якщо середина прийнятого біта
                begin
                    clk7=1;             // на виході одиночний імпульс
                end
            if (count15==8)
                begin
                    clk7<=0;            // на виході встановлено 0
                end
            if (count15==15)
                begin
                    count15<=0;
                    // для наступного біта починаємо рахунок заново
                end
            if (count150==135)          // якщо прийнято 8 бітів
                begin
                    clk135<=1;          // передаємо прийнятий байт на вихід
                end
                // сигналом запису в паралельний регістр
            if (count150==136)
                begin
                    clk135<=0;
                end
            if (count150==179)         // переводимо схему в режим очікування
                begin
                    clk150<=1;
                end
        end
end
```

```
        end
    if (count150==180)
        begin                                // обнуління значення
            clk150<=0;                        // всіх змінних
            count15<=0;                      // на виходах встановлено низький рівень
            count150<=0;
            clk7<=0;
            clk135<=0;
        end
    end
end
endmodule                                // кінець опису модуля
```

Умовний оператор **if** в залежності від заданих умов виконує визначені оператори. В описаному модулі змінна *count* служить для підрахунку імпульсів, що пройшли з моменту зміни даних на лінії. Саме за допомогою цієї змінної формується імпульс на середині інтервалу часу, впродовж якого на лінії втримується прийнятий біт даних. Фронт вихідного сигналу *clk7* забезпечує зсув даних у регістрі зсуву 74164 на один розряд. При цьому в регістр буде записаний черговий прийнятий біт. Після того, як прийняті 9 біт даних, на виході *clk135* формується короткий імпульс, по фронту якого відбувається запис прийнятого байта даних в паралельний регістр 74377. Імпульс на виході *clk150* переводить вихід тригера в низький логічний рівень, тим самим очікуючи наступного стартового біта на лінії. При цьому відбувається обнуління всіх змінних і установка сигналів на вихідних портах у низький стан. З цього опису також потрібно створити графічний символ.

Розглянемо опис модуля *reset*:

```
module reset(en,clk1,rst);                // початок модуля
output en;                               // оголошення вихідного порту
input clk1,rst;                           // вхідні порти
reg en;                                   // вихід має регістровий тип
reg flag;                                 // додаткова змінна для відстеження стану
always @ (clk1 or rst)                   // при зміні вхідних сигналів
begin
    if (!rst)                             // якщо умова виконується
begin
        en=1;                             // устанавлюємо на виходах 1
        flag=1;
    end
    if (flag==1)                          // якщо не було сигналу rst
        en=0;                             // устанавлюємо на виході нуль
    if (rst==1)                            // в протилежному випадку
        flag=0;                           // присвоєння змінній значення 0
end
endmodule                                // кінець опису модуля
```

В принципі, схема буде працювати і без модуля reset. Основним його призначенням є видача одиночного імпульсу при включенні схеми. Цей імпульс скине лічильники в модулі control, оскільки важко передбачити їх початкові значення безпосередньо після процедури програмування або конфігурування мікросхеми.

Після того, як створені всі модулі. А також набрана схема в Графічному редакторі, можна виконувати моделювання. Часова діаграма, що демонструє роботу асинхронного приймача, представлена на рис. 9.14.

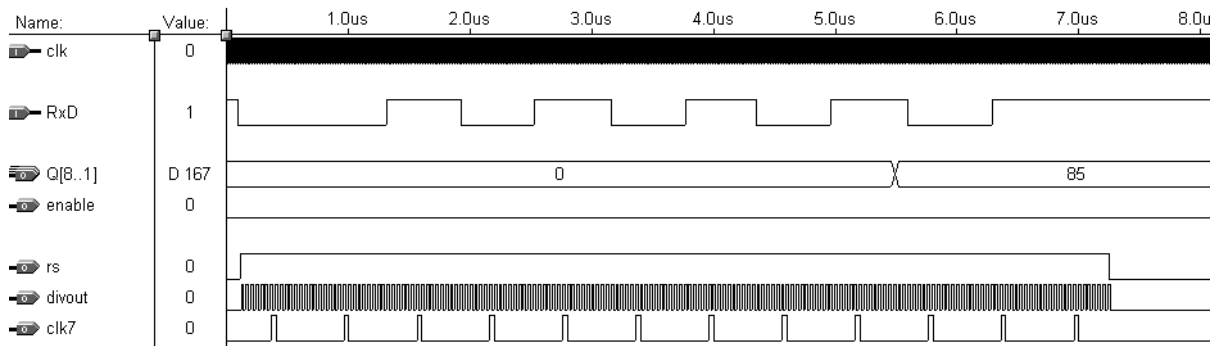


Рис. 9.14.

Як видно з діаграми, імпульси на виході clk7 з'являються на середині прийнятого біту. Після 9 біт даних (1 стартовий плюс 8 біт даних), на виході з'являється прийнятий байт даних. Розроблена схема не перевіряє наявності стопового біту, біту парності. Читачеві пропонується самостійно доробити схему так, щоб вона мала ці можливості. В попередньому розділі приводиться приклад опису модуля перевірки на парність.

В розглянутому прикладі був розроблений асинхронний приймач. При цьому ми використали не тільки текстовий опис модулів на Verilog, але і користувалися Графічним редактором системи MAX+plus II. Розробимо асинхронний передавач послідовного коду. Однак при цьому ми будемо користуватися винятково засобами Текстового редактора і можливостями мови Verilog. Схема передавача представлена на рис. 9.15. Передавач складається з наступних модулів: дільник частоти divider, регістр даних datareg, регістр зсуву trans, RS-тригер, логічні елементи І, АБО, НІ. Установивши дані на вході data[7:0] регістра даних datareg, подаємо одиницю на вхід load. Ці дані з'являються на виході регістра. Крім того, на виході тригера з'явиться високий рівень, що дозволить роботу регістра зсуву trans і забезпечить на вході clk модуля присутність тактового сигналу. Вихід регістра зсуву через елемент АБО з'єднаний з вихідним портом. Інший вхід елемента АБО з'єднаний через інвертор з виходом тригера. Це потрібно для того, щоб при очікуванні початку передачі байта даних тримати лінію у стані лог.1.

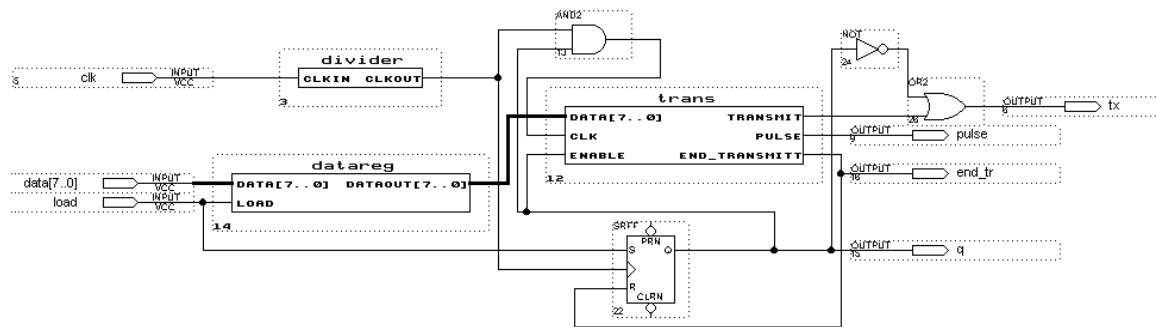


Рис. 9.15.

Структурний опис передавача на Verilog має вигляд:

```

module transmitter_rs(tx, end_tr, pulse, q, data, load, clock); // початок
output tx; // вихід
input [7:0] data; // вхідний порт
input load; // сигнал запису даних
output end_tr; // сигнал завершення передачі
output pulse; // імпульси кожні пів тактового інтервалу
output q;
reg pulse;
input clock; // вхід тактового сигналу
wire w2, w3, clk, clk1; // додаткові ланцюги для організації з'єднань
reg end_tr;
wire tr;
reg q;
reg [7:0] data_T;
    datareg dr(data_T, data, load); // регістр даних
    divider div1(clk, clock); // дільник частоти
    and(clk1, clk, q); // логічний елемент 2І
    trans tr1(tr, pulse, end_tr, data_T, clk1, q); // модуль trans
    or(tx, tr, ~q); // логічний елемент 2АБО
    trig_rs trig(q, clock, load, end_tr); // тригер
endmodule // кінець опису модуля

```

Даний модуль є головним модулем проекту. По суті, в ньому описані тільки зв'язки між іншими модулями і логічними елементами. Модуль також містить вхідні і вихідні порти, за допомогою яких буде відбуватися обмін даними. Схема на рис. 9.13 показує з'єднання між всіма модулями, які входять в проект.

Розглянемо опис RS-тригеру.

```
module trig_rs(q, clock, s, r);           // початок модуля
output q;                                // оголошення виходу
input clock;                             // вхідний тактовий сигнал
input s, r;                              // входи установки і сбросу
wire w1, w2;                            // додаткові ланцюги
srff sr1(s, r, clock, w1, w2, q);        // підключаємо модуль із бібліотеки
endmodule                               // кінець модуля
```

В описі тригера був використаний RS-тригер з бібліотеки системи MAX+plus II. Цей приклад показує, що поряд з описаними власними модулями можна використовувати також елементи з бібліотеки системи.

Опис дільника частоти приводиться нижче.

```
module divider(clkout, clk);           // початок опису модуля
output clkout;                       // вихідний порт
input clk;                           // вхідний порт
reg [5:0] count;                     // внутрішній лічильник
reg clkout;                          // вихід має тип регістр
always @(posedge clk)               // по фронту на вході clk
begin
    count=count+1;                   // збільшуємо значення лічильника
    if (count==13)                   // якщо виконується умова
        begin
            count=0;                 // обнуління лічильника
            clkout=~clkout;           // інвертуємо сигнал на виході
        end
    end
endmodule                           // кінець опису модуля
```

Як бачимо, опис цього модуля нічим не відрізняється від описаного раніше дільника. Наступний модуль представляє з себе найпростіший 8-розрядний паралельний регістр. По фронту сигналу load відбувається запис даних у регістр і установка їх на вихідному порту.

```
module datareg(dataout, data, load); // заголовок модуля
output [7:0] dataout;               // оголошення вихідного порту
input [7:0] data;                   // вхідний порт даних
input load;                         // вхід завантаження даних
reg [7:0] dataout;                  // вихід визначений як регістр
always @ (posedge load)             // по фронту сигналу завантаження
    dataout=data;                    // записуємо дані в регістр
endmodule                           // кінець опису модуля
```

Основним елементом розробленого передавача є модуль trans. Саме цей модуль формує на лінії біти переданого слова в потрібні моменти часу. Розглянемо його опис мовою Verilog.

```
module trans(transmit, pulse, end_transmitt, data, clk, enable);
output transmit;                    // визначення вихідних портів
output end_transmitt;               // сигнал завершення передачі
output pulse;
input [7:0] data;                   // вхідний порт даних
input clk;                          // вхід тактового сигналу
input enable;                       // вхід дозволу
reg transmit;                       // вихідні порти мають регістровий тип
reg pulse;
reg end_transmitt;
```

```
reg [7:0] temp2;           // оголошення додаткових змінних
reg [3:0] temp1;
always @ (posedge clk)    // по фронту тактового сигналу
begin
    if (enable)           // якщо встановлено сигнал дозволу передачі
    begin
        if (temp2==0) transmit=0;
        // сигнал завершення передачі має рівень 0
        temp2=temp2+1;    // збільшуємо на 1 значення змінних
        temp1=temp1+1;
        if (temp1==7) pulse=1;
        //на середині інтервалу формуємо імпульс
        if (temp1==9) pulse=0;
        if (temp1==15) pulse=1;    // при поси́лці наступного біту
        //даних також формуємо імпульс для демонстрації
        //моментів переходу на часовій діаграмі
        if (temp1==0) pulse=0;
        if (temp2==16) transmit=data[0];
        // в залежності від кількості тактів
        if (temp2==32) transmit=data[1];    // на лінії встановлюється
        if (temp2==48) transmit=data[2];    // відповідний біт даних
        if (temp2==64) transmit=data[3];
        if (temp2==80) transmit=data[4];
        if (temp2==96) transmit=data[5];
        if (temp2==112) transmit=data[6];
        if (temp2==128) transmit=data[7];
        if (temp2==144) transmit=1;
        if (temp2==159)
        begin
            end_transmitt=1;    // сигналізуємо про завершення передачі
            pulse=0;
        end
        if (temp2==160)
        begin
            end_transmitt=0;    // сигнал завершення передачі встановлено в 0
            temp2=0;
        end
        end
    else                  // якщо сигнал дозволу передачі не встановлений
    begin
        end_transmitt=0;
        temp2=0;          // обнуління додаткових змінних
        temp1=0;
    end
end
endmodule                // кінець опису модуля
```

Часова діаграма, отримана при моделюванні роботи схеми, представлена на рис. 9.16.

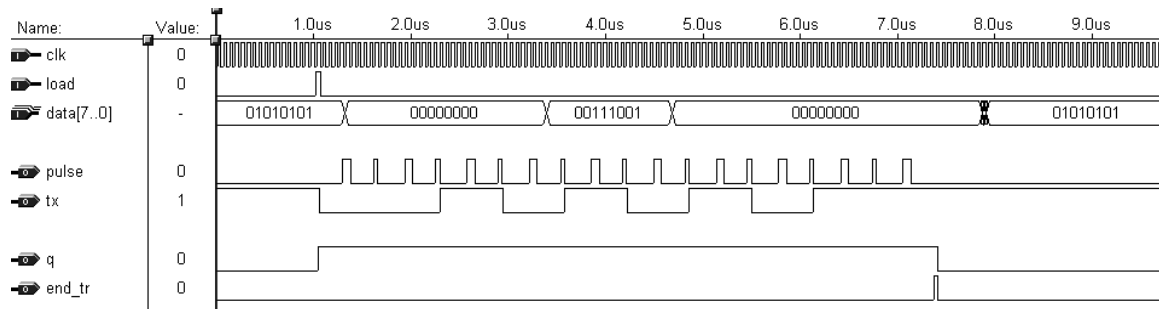


Рис. 9.16.

На часовій діаграмі широкий імпульс на виводі pulse відповідає середині інтервалу біта. В цей момент відбувається запис біта в регістр зсуву. Період вузького імпульсу дорівнює 16 періодам тактового генератора і демонструє момент зміни сигналу на лінії. Після завершення передачі на виході tnd_tr формується короткий імпульс.

9.6. CRC-алгоритми виявлення помилок

Методи виявлення помилок призначені для виявлення пошкодження повідомлень при їхній передачі по зашумленим каналам зв'язку. Для цього передавальний пристрій створює число, яке називається контрольною сумою. Це число представляє з себе функцію повідомлення і додає його до цього повідомлення. Приймач, використовуючи той же самий алгоритм, розраховує контрольну суму прийнятого повідомлення і порівнює її з переданим значенням. Основна ідея алгоритму CRC полягає в представленні повідомлення у вигляді двійкового числа, поділу його на інше фіксоване двійкове число і використанні залишку цієї операції в якості контрольної суми. Одержавши повідомлення, приймач може виконати аналогічну дію і порівняти отриманий залишок з "контрольною сумою" (переданим залишком).

Ключовим словом при роботі з CRC, є слово "поліном". Всі CRC алгоритми засновані на поліноміальних обчисленнях, і для будь-якого алгоритму CRC можна вказати, який поліном він використовує.

Замість представлення дільника, діленого (повідомлення), частки і залишку у вигляді позитивних цілих чисел, можна представити їх у вигляді поліномів із двійковими коефіцієнтами або у вигляді ряду бітів, кожен з яких є коефіцієнтом полінома. Наприклад, десяткове число 23 у шістнадцятковій системі счислення має вигляд 17, а у двійковій – 10111, що збігається з поліномом:

$$1 \cdot x^4 + 0 \cdot x^3 + 1 \cdot x^2 + 1 \cdot x^1 + 1 \cdot x^0,$$

або у спрощеному вигляді:

$$x^4 + x^2 + x^1 + x^0.$$

І повідомлення, і дільник можуть бути представлені у вигляді поліномів, з якими можна виконувати будь-які арифметичні дії. В алгоритмах CRC використовується поліноміальна арифметика по модулю 2 – це фактично двійкова арифметика по модулю 2 без урахування переносів. Додавання двох чисел в CRC арифметиці повністю аналогічно звичайній арифметичній дії, за винятком відсутності переносів з розряду в

розряд. Це означає, що кожна пара бітів визначає результат свого розряду поза залежністю від результатів інших пар.

Фактично, як операція додавання, так і операція віднімання в CRC арифметиці ідентичні операції "ВИКЛ.АБО", що дозволяє замінити 2 операції першого рівня (додавання і віднімання) однією дією. Щоб виконати обчислення CRC, нам необхідно вибрати дільник. Дільник називається генераторним поліномом і це ключове слово будь-якого CRC алгоритму. Ступінь полінома – це дійсна позиція старшого біта. Як правило, контрольна сума додається до повідомлення і разом з ним передається по каналах зв'язку. На іншому кінці каналу приймач може зробити одне з рівноцінних дій:

1. Виділити текст повідомлення, обчислити для нього контрольну суму (не забувши при цьому доповнити повідомлення W бітами), і зрівняти її з переданою.
2. Обчислити контрольну суму для всього переданого повідомлення (без додавання нулів), і подивитися, чи вийде в результаті нульовий залишок.

Як приклад розробимо модуль, який буде обчислювати контрольну суму для вхідного слова. Розрядність інформаційних даних дорівнює 16 бітам. Дільник представлений поліномом виду:

$$1 + x^1 + x^3 + x^4 + x^5 + x^7 + x^8.$$

Опис модуля має вигляд:

```
module CRC8_D16(din, dcrc, dout);           // модуль визначення контрольної суми
input [15:0] din;                           // вхідні дані
input [7:0] dcrc;                           // вхід CRC
output [7:0] dout;                          // вихідний порт
reg [7:0] dout;                             // вихід має регістровий тип
function [7:0] nextCRC8_D16;                // опис функції визначення CRC
input [15:0] Data;                          // вхідні дані
input [7:0] CRC;                            // значення CRC
reg [15:0] D;                               // внутрішня змінна
reg [7:0] C;
reg [7:0] NewCRC;                           // нове значення CRC
begin                                       // початок опису тіла функції
    D = Data;                             // присвоєння значення змінній
    C = CRC;
// визначення кожного біту контрольної суми операцією ВИКЛ.АБО
    NewCRC[0] = D[15] ^ D[7] ^ D[6] ^ D[5] ^ D[2] ^ D[1] ^ D[0] ^ C[7];
    NewCRC[1] = D[15] ^ D[8] ^ D[5] ^ D[3] ^ D[0] ^ C[0] ^ C[7];
    NewCRC[2] = D[9] ^ D[6] ^ D[4] ^ D[1] ^ C[1];
    NewCRC[3] = D[15] ^ D[10] ^ D[6] ^ D[1] ^ D[0] ^ C[2] ^ C[7];
    NewCRC[4] = D[15] ^ D[11] ^ D[6] ^ D[5] ^ D[0] ^ C[3] ^ C[7];
    NewCRC[5] = D[15] ^ D[12] ^ D[5] ^ D[2] ^ D[0] ^ C[4] ^ C[7];
    NewCRC[6] = D[13] ^ D[6] ^ D[3] ^ D[1] ^ C[5];
    NewCRC[7] = D[15] ^ D[14] ^ D[6] ^ D[5] ^ D[4] ^ D[1] ^ D[0] ^ C[6] ^
C[7];
    nextCRC8_D16 = NewCRC; // повернення результату роботи функції
```

```

end
endfunction                                // кінець опису функції
always @(din)                             // при зміні даних на вході
begin
dout=nextCRC8_D16(din, dcrc);             // виклик функції підрахунку CRC
end
endmodule                                // кінець опису модуля

```

Алгоритм обчислення значення CRC розташований у функції CRC8_D16. При зміні входних даних відбувається виклик цієї функції в процедурному блоці **always**, виконується обчислення нового значення CRC. Отже, при обчисленні CRC необхідно виконати наступні дії:

1. Вибрати ступінь полінома W і поліном G (ступеня W).
2. Додати до повідомлення W нульових бітів. Назвемо отриманий рядок M' .
3. Поділимо M' на G з використанням правил CRC арифметики. Отриманий залишок і буде контрольною сумою.

Часова діаграма, що демонструє роботу модуля, представлена на рис. 9.17.

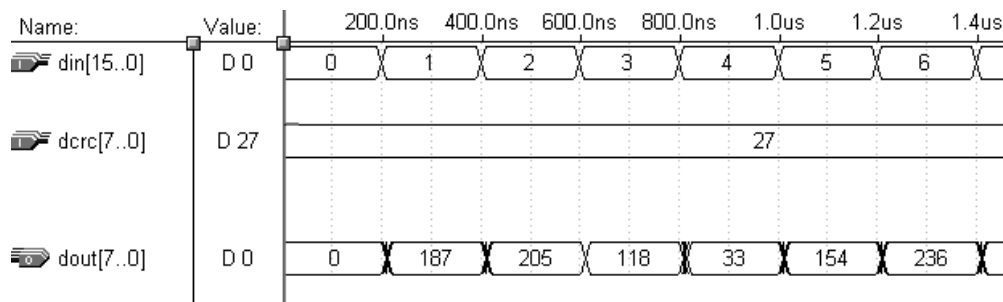


Рис. 9.17.

9.7. Проектування RISC-мікропроцесора

Мікропроцесор представляє собою пристрій, керування яким виконується за допомогою програми. Існують процесори з системою складних команд (CISC-процесор) та з обмеженою кількістю команд (RISC-процесор). Процесор з системою складних команд має складні блоки керування і як наслідок – порівняно невисока швидкодія і співвідношення продуктивність – апаратні затрати. Ідея RISC-процесора ґрунтується на тому, щоб замінити складні багатотактові команди на послідовність простих одноктактових. Одна команда повинна виконуватися за один такт. Виконання кожної команди проходить декілька послідовних стадій: вибірка команди, її дешифрація, вибірка операндів, операція з операндами, запис результатів. Якщо операнди знаходяться в регістровій пам'яті, то такі команди можуть виконуватися за один такт, не враховуючи вибірку та дешифрацію команд. При цьому результат, отриманий в даному такті, може бути використаний як операнд вже в наступному такті. Тому команди RISC-процесора поділяються на команди обробки даних в регістрах, команди пересилки даних і команди керування. Простий формат

команд, включаючи код операції, поля адрес, операндів, забезпечує не складну, а отже, швидко дешифрацію команд і адресів. За рахунок цього мінімізується як тривалість тактового інтервалу, так і апаратні затрати на схему керування. Всі команди в RISC-процесорах мають однакову довжину (розрядність) і мінімум форматів команд.

Під архітектурою процесора розуміють набір відомостей, які необхідні для розробки цього процесора і його використання. Загалом, ці відомості включають в себе структуру процесора, інформацію про систему команд, систему переривань, адресний простір. Також до цього можна віднести відомості про склад периферійних пристроїв, організацію кеш-пам'яті, механізми захисту доступу до ресурсів та інші. Архітектура запропонованого мікропроцесора включає в себе такі особливості, як простий формат команд, конвеєризацію, регістрову пам'ять. Команди процесора мають 16-розрядний формат і виконуються за один такт.

В процесорі система команд поділена на чотири типи: команди переходу, пересилки даних, арифметичні і побітові. В таблиці 9.3 приводиться перелік команд, їх кодування і опис. Поле COP означає код операції, в полях Rd, Rs представлені адреси регістрів загального призначення, поле Func кодує тип арифметичної операції або умову переходу. При виконанні команд переходу аналізуються біти регістра статусу і якщо умова переходу виконується, відбувається зміна значення лічильника команд, тобто перехід на задану комірку пам'яті програм. Для перевірки умови використовуються команди порівняння вмісту двох регістрів (CP) та порівняння регістра з константою (CPI). В результаті виконання зазначених операцій змінюються біти в регістрі статусу SREG. Команди пересилки даних виконують дії над вмістом регістрів (завантаження константи, копіювання вмісту одного регістра в інший). Арифметичні і побітові операції виконуються арифметично-логічним пристроєм. До таких команд відносяться функції логічного І, АБО, ВІКЛ.АБО, НІ, логічного і арифметичного зсуву праворуч на один розряд (SRL, SRA), логічного і арифметичного зсуву ліворуч на один розряд (SLL, SLA), додавання і віднімання (ADD, SUB), додавання і віднімання з урахуванням переносу (ADC, SBC).

Табл. 9.3.

COP [15..12]	[11..8]	[7..4]	[3..0]	Команда
0000	0000	0000	0000	NOP (пуста операція)
0001	Адреса порту		D	OUT portX, Rd (вивід значення в порт)
0010	Rd	Адреса порту		IN Rd, pinX (зчитування з порту)

9. Розробка спеціалізованих мікропроцесорних
модулів на програмованих логічних інтегральних схемах

0011	1000	Адреса комірки пам'яті програм (ПП)		BRGE Addr (перехід якщо більше)
	0100	Адреса комірки ПП		BRLO Addr (перехід якщо менше)
	0010	Адреса комірки ПП		BREQ Addr (перехід якщо дорівнює)
	0001	Адреса комірки ПП		BRNE Addr (перехід якщо не дорівнює)
	1010	Адреса комірки ПП		BRSN Addr (перехід якщо більше або дорівнює)
	0110	Адреса комірки ПП		BRLT Addr (перехід якщо менше або дорівнює)
0100	0000	Адреса комірки ПП		JMP Addr (абс. перехід)
	0001	Зсув		RJMP Addr (відн. перехід)
	0010	—		RETI (вихід з переривання)
	0011	Адреса комірки ПП		RCALL (виклик підпрограми)
	0100	—		RET (вихід з підпрограми)
0110	Rd	Rs	0000	MOV Rd, Rs (копіювання вмісту регістра)
0110	Rd	Rs	0001	CP Rd, Rs (порівняння регістрів)
1110	Rd	константа		CPI Rd, K (порівняння регістру і константи)
1010	Rd	константа		LDI Rd, K (завантаження константи в регістр)
0111	Rd	Rs	0000	AND Rd, Rs (логічне І)
	Rd	Rs	0001	XOR Rd, Rs (логічне ВИКЛ.АБО)

	Rd	Rs	0010	OR Rd, Rs (логічне АБО)
	Rd	–	0011	NOT Rd (інверсія)
	Rd	–	0100	LSR Rd (лог.зсув праворуч)
	Rd	–	0101	ASR Rd (арифм.зсув праворуч)
	Rd	Rs	0110	ADD Rd, Rs (додавання значень регістрів)
	Rd	Rs	0111	SUB Rd, Rs (операція віднімання)
	Rd	Rs	1000	ADC Rd, Rs (додавання з переносом)
	Rd	Rs	1001	SBC Rd, Rs (віднімання з переносом)
	Rd	–	1010	INC Rd (інкремент)
	Rd	–	1011	DEC Rd (декремент)
	Rd	–	1100	LSL Rd (лог.зсув ліворуч)
	Rd	–	1101	ASL Rd (арифм.зсув ліворуч)
	Rd	Rs	1110	MUL Rd, Rs (перемноження)
	Rd	–	1111	SWAP Rd (обмін тетрадами)

В залежності від типу команди, в окремих розрядах міститься різна інформація. Так, для арифметичних операцій в розрядах [11:8] та [7:4] команди розташовані адреси регістрів, над вмістом яких виконується операція. При виконанні арифметичних операцій результат розташовується в регістрі Rd. Якщо виконується операція множення, молодший та старший байти добутку розміщуються в регістрах R0 та R1. При виконанні команд з одним операндом значення бітів [7:4] не враховуються. Для команд роботи з константами (LDI та CPI) в бітах [11:8] міститься адрес регістра, а в молодших 8 бітах – значення константи. У командах переходу біти [11:8] визначають умову переходу, а в розрядах [7:0] міститься адреса комірки пам'яті, до якої буде виконаний перехід. Перехід відбувається якщо одна з умов переходу, яка передається разом з командою, співпадає зі значенням окремого біту (або декількох бітів) в регістрі-статусі. Перевірка виконання умови переходу відбувається

шляхом виконання операції лог. „I” полів [11:8] команди і значення в регістрі – статусі. Якщо результат не дорівнює 0, то значення лічильника команд приймає нове значення. В табл. 9.4 наводиться позначення бітів регістра статусу.

Табл. 9.4.

Номер біту	Умова установки
3	Більше
2	Менше
1	Дорівнює
0	Не дорівнює

Якщо в регістрі SREG встановлені декілька бітів, то це означає не строгу умову переходу. Наприклад значення коду 0101₂ означає перехід якщо більше або дорівнює.

Процесор має регістрову пам'ять яка складається з шістнадцяти 8-розрядних регістрів. Регістри розташовані в модулі CPUCORE. Це значно спростило доступ до них під час виконання арифметичних і побітових операцій. Регістри периферійних пристроїв займають частину адресного простору. В таблиці 9.5 приводяться адреси регістрів портів вводу/виводу та регістра керування перериваннями.

Табл. 9.5.

DDRA	8'h12
PORTA	8'h11
PINA	8'h13
DDRB	8'h15
PORTB	8'h14
PINB	8'h16
GIMSK	8'h17

Склад периферійних пристроїв залежить від використання процесора і в даному прикладі має лише порти вводу/виводу. Порт – це пристрій, за допомогою якого мікропроцесор з'єднується із зовнішніми пристроями.

Затримку в лініях зв'язку в ПЛІС можна оцінити, як пропорційну їх довжині і вона може бути більша, ніж сумарна затримка в логічних схемах. Якщо все керування зосередити в одному блоці керування, то його багаточисельні входи та виходи необхідно підключати до інших блоків мікропроцесора через довгі маршрути з великими затримками. В запропонованому варіанті процесора основну схему пристрою керування можна звести до комбінаційної схеми, яка дешифрує код команди. Тому для мінімізації затримок пропонується структура процесора з розподіленим керуванням. На рис. 9.18 представлена схема RISC-мікропроцесора. Модуль ProgCounter – це лічильник команд, PROM –

пам'ять програм, CPUCORE – центральний процесорний елемент, SREG – регістр статусу, PORTA та PORTB – двонапрямлені порти вводу/виводу.

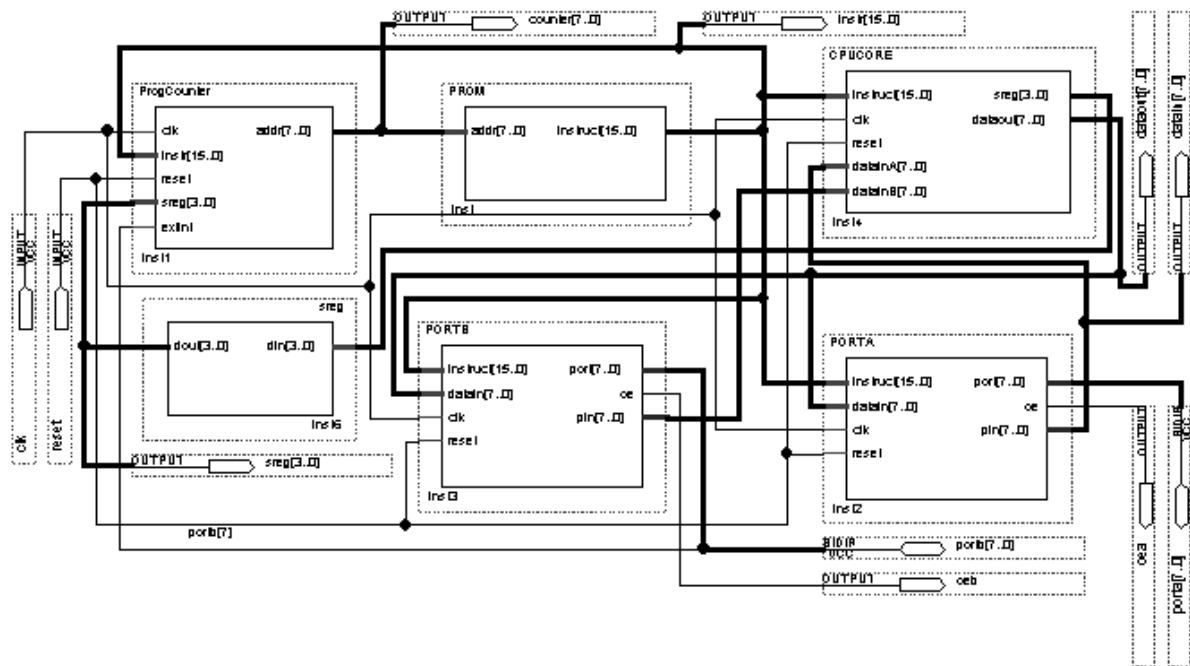


Рис. 9.18.

В кожному з блоків знаходиться локальний пристрій керування. Також завдяки простоті команд, їх дешифрація та керування виконанням нескладні. Центральним процесорним елементом (ЦПЕ) називається пристрій, який реалізує арифметичні і логічні операції, функції керування. В запропонованому варіанті процесора роль ЦПЕ виконує модуль CPUCORE. Налаштування портів вводу/виводу виконується шляхом запису значення 0 (вхід) або F_{16} (вихід) в регістр DDRA або DDRB.

Основні режими мікропроцесора – робота за програмою, яка записана в пам'яті, обробка переривань по командам від зовнішніх пристроїв. Для того, щоб мікропроцесор міг виконати команду, йому необхідно повідомити код операції, операнди або їх адреси (регістри загального призначення), а також адресу, куди потрібно помістити результат. Наприклад команда `ADD R0, R1` виконує додавання вмісту регістрів R0 та R1 і результат розташовується в регістрі R0. В мікропроцесорі передбачена обробка одного зовнішнього переривання. Його обробка відбувається при наявності лог.1 на 7 розряді порту В якщо нульовий біт в регістрі GIMSK має значення 1 (дозволена обробка переривань). При цьому поточне значення лічильника команд переміщується в стек і збільшується на одиницю значення вказівника стеку. При виході з обробника переривання значення лічильника команд береться зі стеку і продовжується виконання основного циклу програми. Опис модуля ProgCounter приводиться нижче.

module ProgCounter(addr, clk, instr, reset, sreg, extint);


```

output [7:0] addr;           // вихід лічильника команд
input clk;                   // вхід тактового сигналу
input [15:0] instr;          // вхідний порт команди
input reset;                 // вхід сбросу
input [3:0] sreg;             // вхід значення регістру статусу
input extint;                // вхід зовнішнього переривання
reg [7:0] addr;              // вихід лічильника має регістровий тип
reg [7:0] stack [2:0];       // стек
reg [2:0] sp;                // вказівник стеку
reg ien;                     // зберігається значення прапорця дозволу переривання
reg isint;                   // зберігається значення прапорця режиму роботи
always @(posedge clk) // по фронту тактового сигналу
    begin
        if(reset==1)           // якщо сброс
            begin
                addr=0;         // на виході модуля встановлюється 0
                ien=0;          // обнуління внутрішніх змінних
                sp=0;
                isint=0;
            end
        else if(ien==1 && extint==1 && isint==0) // перевірка режиму
            begin               // якщо виникло переривання
                stack[sp]=addr; // значення лічильника
                                // переміщується в стек
                sp=sp+1;        // інкремент вказівника стеку
                addr=8'h1;      // перехід на 1 комірку ПП
                isint=1;        // встановлення прапорця режиму
            end
        else
            case(instr[15:12]) // перевірка коду команди
                4'b0011:       // виконання команди переходу
                    if((instr[11:8]&sreg)!=0)
                        // перевірка регістра статусу
                        begin // якщо умова виконується
                            addr=instr[7:0]; // перехід на задану комірку
                        end
                    else
                        begin
                            addr=addr+1; // збільшення значення лічильника
                        end
                4'b0100:       // обробка команди переходу
                    if(instr[11:8]==0) addr=instr[7:0]; // JMP
                    else if(instr[11:8]==4'b0001) addr=addr+instr[7:0]; // RJMP
                    else if(instr[11:8]==4'b0010) // вихід з переривання
                        begin
                            isint=0; // зміна значення прапорця режиму
                            sp=sp-1; // зменшення значення вказівника стеку
                            addr=stack[sp]+1; // установка значення лічильника
                        end
                    else if(instr[11:8]==4'b0011) // RCALL
                        begin
                            stack[sp]=addr; // значення лічильника

```

```

// переміщується в стек
sp=sp+1; // інкремент вказівника голови стеку
addr=instr[7:0]; // установка адреси підпрограми
end
else if(instr[11:8]==4'b0100)
begin
    sp=sp-1; // декремент вказівника стеку
    addr=stack[sp]+1; // установка адреси
end
4'b0001:
// обробка команди запису даних в регістр GIMSK
begin
    if(instr[11:4]==8'h17) ien=instr[0]; // перевірка адреси
    addr=addr+1; // інкремент лічильника команд
end
default: // якщо жодна з альтернатив не співпала
    addr=addr+1; // інкремент лічильника команд
endcase // кінець конструкції вибору
end // кінець процедурного блоку
endmodule // кінець опису модуля

```

Основні функції описаного модуля полягають в наступному:

- збільшення значення лічильника команд по фронту тактового сигналу;
- дешифрація і виконання команд абсолютного і відносного переходів;
- дешифрація і виконання команди умовного переходу;
- перехід до обробника зовнішнього переривання із збереженням поточної адреси комірки у стеку;
- обробка команди виклику підпрограми.

Дешифрація команд відбувається за допомогою конструкції вибору **case**. При виконанні команди переходу аналізуються біти регістра статусу і якщо виконується умова переходу, лічильнику команд присвоюється адреса відповідної комірки. Оператор **if** використовується для перевірки стану вхідних сигналів. Якщо на вході **reset** присутня лог.1, відбувається обнуління всіх внутрішніх змінних модуля (лічильника команд, стеку). При появі лог.1 на вході **extint** відбувається перехід до обробника зовнішнього переривання. Перед цим значення лічильника команд переміщується в стек, функції якого виконує змінна **stack**. В змінній **ien** зберігається прапорець дозволу зовнішнього переривання. Якщо значення цієї змінної дорівнює 0, обробка переривання заборонена. За допомогою змінної **isint** визначається внутрішній стан мікропроцесора (чи відбувається обробка переривання).

Модуль **ProgCounter** має одну вихідну 8-розрядну шину, на якій з кожним тактовим імпульсом з'являється нове значення адреси комірки пам'яті програм. Лічильник команд виконує нескінченний цикл читання вмісту пам'яті. Розглянемо опис модуля пам'яті програм **PRAM**.

```

module PROM (address, q); // заголовок модуля

```

```

input [7:0] address;           // вхідний порт адреси комірки
output [15:0] q;              // вихідний порт
wire [15:0] sub_wire0;        // додаткова 16-розрядна шина
wire [15:0] q = sub_wire0[15:0]; // з'єднання виходу з шиною
lpm_rom lpm_rom_component (.address (address),.q (sub_wire0),
.inclock (), .memenab (), .outclock ());
// підключення модуля з бібліотеки
dfparam                                // визначення параметрів
    lpm_rom_component.intended_device_family = "APEX20KE",
    lpm_rom_component.lpm_width = 16,      // розрядність даних
    lpm_rom_component.lpm_widthad = 8,     // розрядність адресної
                                           // шини
    lpm_rom_component.lpm_address_control = "UNREGISTERED",
    lpm_rom_component.lpm_outdata = "UNREGISTERED",
    lpm_rom_component.lpm_file = "progmem2.mif", // назва файлу
                                           // ініціалізації пам'яті
    lpm_rom_component.lpm_hint= "ENABLE_RUNTIME_MOD=NO",
    lpm_rom_component.lpm_type = "LPM_ROM";
endmodule                        // кінець опису модуля

```

В якості елемента пам'яті був використаний параметризований модуль `lpm_rom` з бібліотеки САПР. Опис всіх параметрів модуля можна знайти в довідковій системі САПР Quartus або в 2 розділі даного посібника. Файл ініціалізації пам'яті містить коди команд, які будуть виконані процесором. Формат і вміст цього файлу буде розглянуто трохи пізніше. Модуль пам'яті програм має 8-розрядну адресну шину, тобто її ємність складає 256 слів. Розрядність команд процесора дорівнює 16.

Основна робота виконується центральним процесорним елементом (ядром процесора). На схемі рис. 9.18 цей модуль позначений як CPUCORE. Опис модуля CPUCORE представлений нижче.

```

module CPUCORE(instruct, clk, datainA, datainB, reset, sreg, dataout);
input [15:0] instruct;        // вхідна шина команд
input clk;                    // вхід тактового сигналу
input reset;                  // вхід обнуління
input [7:0] datainA;          // вхід даних з порту A
input [7:0] datainB;          // вхід даних з порту B
output [3:0] sreg;            // вихідний порт регістра статусу
output [7:0] dataout;         // вихідна шина даних
reg [3:0] sreg;               // обидва вихідних порти мають регістровий тип
reg [7:0] dataout;
reg [7:0] mem [15:0];         // регістри загального призначення
reg [7:0] Rd;                 // регістр - призначення
reg [7:0] Rs;                 // регістр - джерело
reg carry;                    // біт переносу
reg [3:0] i;                  // змінна для адресації пам'яті
always @(posedge clk)        // по фронту тактового сигналу виконується
begin                          // процедурний блок
    if(reset==1)                // якщо сброс
    begin

```

```
Rd=0;          // обнуління регістрів
Rs=0;
dataout=0;     // на шині даних встановлюється 0
for(i=0;i<15;i=i+1) mem[i]=0; // обнуління регістрової пам'яті
    sreg=0;     // в регістр статусу завантажується значення 0
end
case(instruct[15:12]) // перевірка коду операції
4'b0110:      // MOV
    if(instruct[3:0]==0) mem[instruct[11:8]]=mem[instruct[7:4]];
    else if(instruct[3:0]==4'b0001)
    begin      // CP
    if(mem[instruct[11:8]]>mem[instruct[7:4]])
        sreg=4'b1001; // установка значення SREG
        // відповідно до результату операції порівняння
    else if(mem[instruct[11:8]]<mem[instruct[7:4]])
        sreg=4'b0101;
    else if(mem[instruct[11:8]]==mem[instruct[7:4]])
        sreg=4'b0010;
    end
    4'b1110:    // CPI
    if(mem[instruct[11:8]]>instruct[7:0]) sreg=4'b1001;
    else if(mem[instruct[11:8]]<instruct[7:0]) sreg=4'b0101;
    else if(mem[instruct[11:8]]==instruct[7:0]) sreg=4'b0010;
4'b0001:      // OUT
    begin      // якщо вказані адреси портів вводу/виводу
    // на шині даних встановлюється значення регістра
    if(instruct[11:4]==8'h11 || instruct[11:4]==8'h14)
        dataout=mem[instruct[3:0]];
    end
4'b0010:      // IN
    begin      // перевіряється адреса порту
    // і в регістр записуються дані
    if(instruct[7:0]==8'h13) mem[instruct[11:8]]=datainA;
    if(instruct[7:0]==8'h16) mem[instruct[11:8]]=datainB;
    end
4'b1010:      // LDI
    begin      // завантаження константи в регістр
    mem[instruct[11:8]]=instruct[7:0];
    end
4'b0111:      // ALU
    begin
        Rd=mem[instruct[11:8]]; // запис значень в регістри
        Rs=mem[instruct[7:4]];
        case(instruct[3:0])      // дешифрація операції
        4'd0: Rd=Rd&Rs; // команда AND
        4'd1: Rd=Rd^Rs; // команда XOR
        4'd2: Rd=Rd|Rs; // команда OR
        4'd3: Rd=~Rd;   // команда NOT
        4'd4: Rd=Rd>>1; // команда LSR
        4'd5:          // команда ASR
        begin
            carry=Rd[7]; // запам'ятовуємо значення біту
```

```

Rd=Rd>>1;          // зсув на один розряд праворуч
Rd[7]=carry;        // завантаження біту в старший розряд
carry=0;
end
4'd6: {carry,Rd}=Rd+Rs; // команда ADD
4'd7: Rd=Rd-Rs;        // команда SUB
4'd8: {carry,Rd}=Rd+Rs+carry; // команда ADC
4'd9: Rd=Rd-Rs-carry;  // команда SBC
4'd10: {carry, Rd}=Rd+1; // команда INC
4'd11: Rd=Rd-1;        // команда DEC
4'd12: Rd=Rd<<1;      // команда LSL
4'd13:                // команда ASL
begin
carry=Rd[0];         // запам'ятовуємо значення біту
Rd=Rd<<1;            // зсув на один розряд ліворуч
Rd[0]=carry;         // завантаження біту в молодший розряд
carry=0;
end
4'd14: {mem[1],mem[0]}=Rd*Rs; // команда MUL
4'd15:                // команда SWAP
begin
Rs=Rd;
Rs=Rs>>4;           // зсув на 4 розряди праворуч
Rd=Rd<<4;           // зсув на 4 розряди ліворуч
Rd=Rd|Rs;           // операція «АБО»
end
endcase
mem[instruct[11:8]]=Rd; // запис в регістр нового значення
end
endcase                // кінець конструкції вибору
end                    // кінець процедурного блоку
endmodule              // кінець опису модуля

```

До основних функцій модуля CPUCORE відносяться:

- виконання арифметичних, побітових команд, а також команд пересилки даних;
- зміна значення в регістрі статусу SREG після виконання команд порівняння;
- установка даних на шині даних при виводі значення регістра в порт вводу/виводу;
- запис значення в потрібний регістр при зчитуванні даних з порту вводу/виводу.

За допомогою конструкції вибору **case** відбувається дешифрація коду операції, який знаходиться в розрядах [15:12] команди. В залежності від цього коду виконується відповідний блок. Всі операнди, або адреси регістрів, передаються в слові команди у відомих позиціях (табл. 9.3). При наявності високого рівня сигналу на вході reset відбувається обнуління регістрової пам'яті та додаткових змінних. В модуль поступають сигнали з портів вводу/виводу. Це дозволяє виконувати команду зчитування даних з

порту за один такт. Вихідна 8-розрядна шина даних з'єднана з входами кожного з портів. Значення коду на цій шині змінюється при появі команди OUT. Разом з командою передається адрес порту, тому значення коду встановлюється лише на визначеному порту і зберігається до наступної команди OUT.

В регістрі статусу SREG зберігається інформація про результат виконання операції порівняння. Значення в регістрі статусу аналізується при виконанні команд умовного переходу. Умова переходу передається в розрядах [11:8] команди. Опис модуля регістра статусу приводиться нижче.

```
module SREG(din, dout);           // заголовок модуля
input [3:0] din;                 // вхідний порт
output [3:0] dout;              // вихідний порт
reg[3:0] dout;                   // вихід має регістровий тип
always @(din)                   // при зміні значення на вході
    dout=din;                    // установка значення вхідного коду на виході
endmodule                       // кінець опису модуля
```

Для того, щоб можна було отримати результати виконання операцій, процесор має два порти: PORTA та PORTB. Кожен з портів може бути використаний як вхідний або вихідний. Це відбувається шляхом запису значення в регістри DDRA та DDRB відповідно. Якщо порт налаштований на вхід, то він переходить в стан високого імпедансу. Регістри DDRA та DDRB доступні тільки для запису даних, тобто їх значення не можуть бути зчитані. Опис модуля порту A приводиться нижче.

```
module PORTA(port, pin, instruct, datain, clk, oe, reset);
inout [7:0] port;                // двонапрявлена шина
output oe;                       // стан порту (1-вихід, 0-вхід )
inout [7:0] pin;                 // двонапрявлена шина для зчитування даних з порту
input [15:0] instruct;           // шина команд
input [7:0] datain;              // вхід даних
input clk;                       // тактових сигнал
input reset;                     // вхід обнуління
wire oe;
wire [7:0] wdata;                // внутрішня шина для з'єднання блоків
    ddra d1(.oe(oe), .instruct(instruct), .clk(clk), .reset(reset));
    dataregA p1(.porta(wdata), .din(datain), .instruct(instruct));
    bufferA b1(.oe(oe), .inp(wdata), .clk(clk), .outp(pin), .bidir(port));
endmodule                        // кінець опису модуля

module bufferA(oe, inp, clk, outp, bidir); // буфер з трьома станами
input oe;                         // вхід, що визначає напрям роботи порту
input clk;                        // вхід тактового сигналу
input [7:0] inp;                  // вхідний порт даних
inout [7:0] outp;                // вихідний порт (для зчитування значення з порту)
inout [7:0] bidir;               // двонапрявлена шина
reg [7:0] a;                    // додаткові регістри
```

```

reg [7:0] b;
assign bidir = (oe==1) ? a : 8'bZ; // перевірка сигналу oe: якщо 1 – на
// виході встановлюються дані, якщо 0 – високий імпеданс
assign outp = b;
always @ (posedge clk) // по фронту тактового сигналу
begin // виконується процедурний блок
    b <= bidir; // зчитування в змінну значення на виводах порту
    a <= inp; // запис в змінну вхідних даних
end // кінець процедурного блоку
endmodule // кінець опису модуля

module dataregA(porta, din, instruct); // модуль регістра даних порту
output [7:0] porta; // вихідна шина даних
input [7:0] din; // вхідна шина даних
input [15:0] instruct; // вхідна шина команд
reg [7:0] porta; // вихід має регістровий тип
always @ (instruct or din) // при зміні команди або даних
    begin // перевіряємо код команди та адресу порту
        if(instruct[15:4]==12'h111 || instruct[15:4]==12'h112) porta=din;
    end // кінець процедурного блоку
endmodule // кінець опису модуля

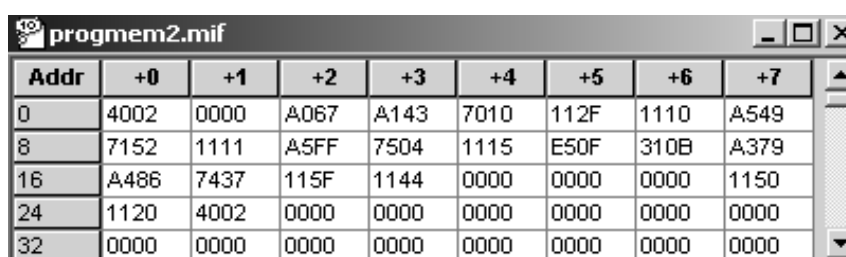
module ddra(oe, instruct, clk, reset);
// модуль регістра напрямку порту
parameter ADDRESS=12'h112; // адрес регістра напрямку DDRA
output oe; // сигнал режиму роботи порту
input [15:0] instruct; // вхідна шина команд
input clk; // вхід тактового сигналу
input reset; // вхід сбросу
reg oe;
reg [1:0] cnt; // лічильник для визначення стану порту
always @ (posedge clk) // по фронту тактового сигналу
    begin
        if(cnt==2) oe=1; // перевірка значення лічильника
        begin
            // якщо виконується умова, вихід переходить високий імпеданс
            if(instruct=={ADDRESS,4'h0} || reset==1)
                begin
                    oe=0;
                    cnt=0;
                end
            else if(instruct=={ADDRESS,4'hf} && cnt==0) cnt=1;
            else if(cnt==1) cnt=2; // зміна стану порту
        end
    end // кінець процедурного блоку
endmodule // кінець опису модуля

```

Опис модуля порту В аналогічний наведеному. Відмінність полягає лише в назві головного модуля (PORTB) та значенні параметра ADDRESS в модулі ddra. Для порту В значення цього параметру дорівнює 12'h115. Загалом, весь порт складається з чотирьох модулів: PORTA (структурний

опис порту), bufferA (буфер з трьома станами), dataregA (регістр даних) та ddra (регістр напрямку роботи порту). В модулі ddra відбувається перевірка адреси і якщо вона співпадає зі значенням параметру ADDRESS, відбувається установка сигналу режиму роботи порту oe. Якщо oe=1, порт працює як вихідний. В протилежному випадку виводи переходять в стан високого імпедансу і значення сигналів, які будуть подані на ці входи, можуть бути зчитані в регістри загального призначення. В модулі dataregA відбувається перевірка адреси порту і якщо вона співпадає із заданою код, який присутній на шині даних, записується в регістр і встановлюється на виводах порту. Модуль bufferA працює як повторювач, якщо значення сигналу oe дорівнює 1, або забезпечує високий імпеданс виходу якщо значення сигналу oe дорівнює 0.

Розглянемо приклади програм. На рис. 9.19 представлена карта пам'яті першої програми.



Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	4002	0000	A067	A143	7010	112F	1110	A549
8	7152	1111	A5FF	7504	1115	E50F	310B	A379
16	A486	7437	115F	1144	0000	0000	0000	1150
24	1120	4002	0000	0000	0000	0000	0000	0000
32	0000	0000	0000	0000	0000	0000	0000	0000

Рис. 9.19.

Для того, щоб створити такий файл, потрібно з меню File вибрати команду New. З'явиться діалогове вікно створення нового файлу. На закладці Other Files цього вікна потрібно вибрати пункт Memory Initialization File і натиснути кнопку ОК. Після цього з'явиться наступне діалогове вікно, в якому вказується кількість слів та їх розрядність. В табл. 9.6 представлені код та опис кожної з команд програми, а також її позначення.

Табл. 9.6.

Комірка пам'яті	Код команди	Позначення	Опис команди
00	4002	JMP Main	Перехід на мітку Main
01	0000	NOP	Холоста операція
Main:			
02	A067	LDI R0, 0x67	Завантаження константи в регістр
03	A143	LDI R1, 0x43	Завантаження константи в регістр
04	7010	AND R0, R1	Операція „І” над вмістом регістрів
05	112F	OUT DDRA, 0xF	Налаштування порту А на вихід
06	1110	OUT PORTA, R0	Вивід в порт А вмісту регістра R0
07	A549	LDI R5, 0x49	Завантаження константи в регістр
08	7152	OR R1, R5	Додавання значень двох регістрів
09	1111	OUT PORTA, R1	Вивід в порт А вмісту регістра R1

0A	A5FF	LDI R5, 0xFF	Завантаження константи в регістр
Loop:			
0B	7504	LSR R5	Логічний зсув праворуч
0C	1115	OUT PORTA, R5	Вивід в порт А вмісту регістра R5
0D	E50F	CPI R5, 0x0F	Порівняння регістру з константою
0E	310B	BRNE Loop	Перехід на мітку якщо не дорівнює
0F	A379	LDI R3, 0x79	Завантаження константи в регістр
10	A486	LDI R4, 0x86	Завантаження константи в регістр
11	7437	SUB R4, R3	Знаходження різниці
12	115F	OUT DDRB, 0xF	Налаштування порту В на вихід
13	1144	OUT PORTB, R4	Вивід в порт В вмісту регістра R4
14	0000	NOP	Пуста операція
15	0000	NOP	Пуста операція
16	0000	NOP	Пуста операція
17	1150	OUT DDRB, 0	Налаштування порту В на вхід
18	1120	OUT DDRA, 0	Налаштування порту А на вхід
19	4002	JMP Main	Перехід на мітку

Виконання програми починається з нульової комірки пам'яті, в якій знаходиться команда переходу на комірку з номером 2. В комірці 1 знаходиться адреса обробника переривання. Всі числа представлені в шістнадцятковому форматі. В даному прикладі переривання не виникає і тому значення в цій комірці 0. Далі виконуються команди завантаження констант в регістри R0 та R1. Над числами в цих регістрах виконується операція „І”, і результат цієї операції виводиться в порт А, який попередньо налаштовується на вихід. Отже, результатом операції „І” над числами 67 та 43 є значення 43: $1100111 \& 1000011 = 1000011$. Далі виконується завантаження числа 49 в регістр R5 і виконується команда „АБО” над вмістом регістрів R1 та R5. Результат розміщений в регістрі R1. Значення з цього регістру виводиться в порт А: $1000011 \mid 1001001 = 1001011$ (4B). Всі біти регістра R5 встановлюються в 1 і в циклі відбувається зсув вмісту регістра праворуч. Нове значення виводиться в порт А, після чого виконується команда порівняння значення регістру з константою 0x0F. Цикл продовжується доти, доки ці значення не співпадуть. Нескладно підрахувати, що цикл буде повторюватися чотири рази. Далі завантажуються константи в регістри R3 та R4, і над ними виконується операція віднімання. Результат виводиться в порт В, який попередньо налаштовується на вивід. Отже, результатом віднімання чисел 86 та 79 є значення 0D. Часова діаграма, отримана після моделювання роботи процесора представлена на рис. 9.20.

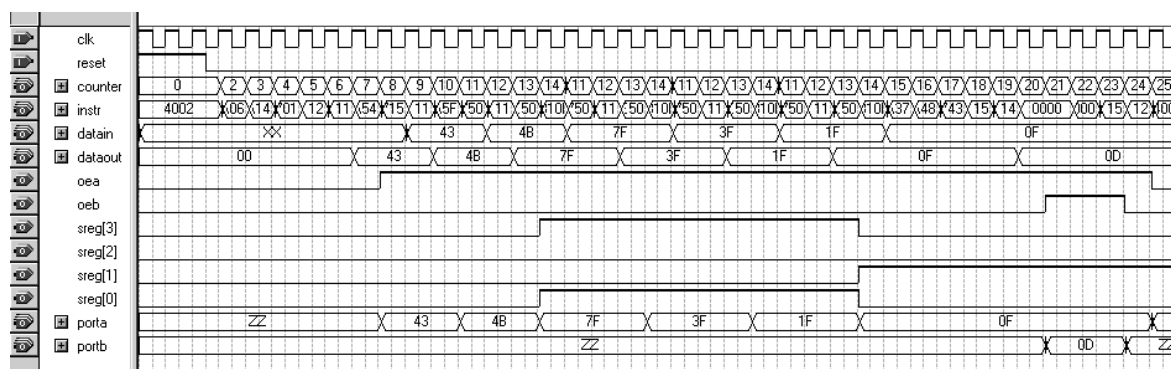


Рис. 9.20.

Аналіз діаграми підтверджує правильність виконання команд мікропроцесором. Для того, щоб прослідкувати послідовність виконання команд, на діаграмі відображені значення лічильника команд та код команди. Також на діаграмі показані сигнали oea та oeb, за допомогою яких визначається режим роботи кожного з портів. Значення регістра статусу змінюється після виконання команди порівняння. При рівності значення регістра R5 константі 0x0F встановлюється 1 біт в регістрі SREG, що є ознакою рівності. Наступний приклад програми демонструє обробку мікропроцесором зовнішнього переривання. Карта пам'яті програм представлена на рис. 9.21.

Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	4002	4017	A029	A145	7016	112F	1110	A549
8	2616	6460	7403	1114	7167	1171	1111	760B
16	E600	1116	310F	0000	0000	1120	4002	1170
24	A8FF	1118	0000	4200	0000	0000	0000	0000
32	0000	0000	0000	0000	0000	0000	0000	0000

Рис. 9.21.

В таблиці 9.7. представлені код та опис кожної з команд програми і її позначення.

Табл. 9.7.

Комірка пам'яті	Код команди	Позначення	Опис команди
00	4002	JMP Main	Перехід на мітку Main
01	4017	NOP	Холоста операція
Main:			
02	A029	LDI R0, 0x29	Завантаження константи в регістр
03	A145	LDI R1, 0x45	Завантаження константи в регістр
04	7016	ADD R0, R1	Додавання вмісту регістрів R0 та R1
05	112F	OUT DDRA, 0xF	Налаштування порту A на вихід
06	1110	OUT PORTA, R0	Вивід в порт A значення регістра R0
07	A549	LDI R5, 0x49	Завантаження константи в регістр
08	2616	IN R6, PINB	Запис в регістр значення на порту B
09	6460	MOV R4, R6	Копіювання вмісту регістрів

0A	7403	NOT R4	Інверсія коду в регістрі R4
0B	1114	OUT PORTA, R4	Вивід в порт значення регістра
0C	7167	SUB R1, R6	Знаходження різниці між регістрами
0D	1171	OUT GIMSK, 0x1	Дозвіл обробки зовнішнього переривання
0E	1111	OUT PORTA, R1	Вивід в порт А регістра R1
Loop:			
0F	760B	DEC R6	Декремент регістра R6
10	E600	CPI R6, 0	Порівняння вмісту регістра з 0
11	1116	OUT PORTA, R6	Вивід в порт А значення регістра
12	310F	BRNE Loop	Якщо не дорівнює, перехід на мітку
13	0000	NOP	Пуста операція
14	0000	NOP	—
15	1120	OUT DDRA, 0	Налаштування порту А на вхід
16	4002	JMP Main	Перехід на мітку
Ext_int:	Обробник зовнішнього переривання		
17	1170	OUT GIMSK, 0	Заборона обробки переривання
18	A8FF	LDI R8, 0xFF	Завантаження константи в регістр
19	1118	OUT PORTA, R8	Вивід в порт А значення регістра
1A	0000	NOP	Пуста операція
1B	4200	RETI	Вихід із переривання

Часова діаграма на рис. 9.22 демонструє роботу мікропроцесора по наведеному алгоритму.

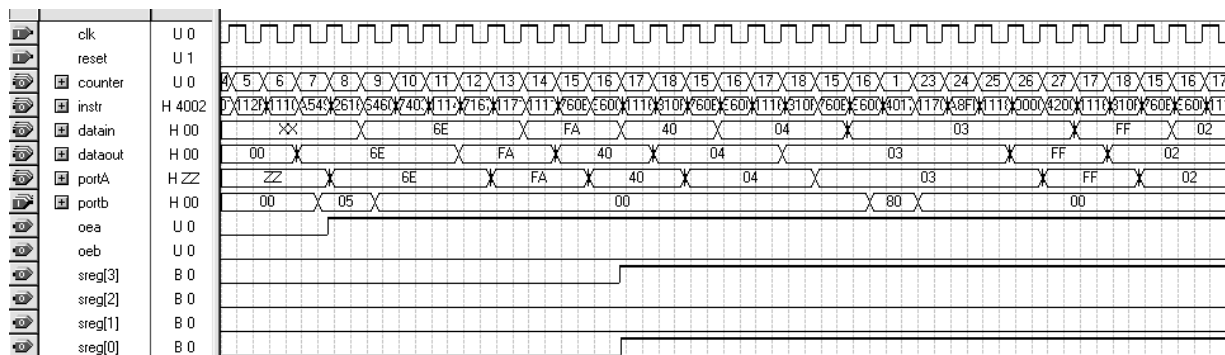


Рис. 9.22.

Результат додавання чисел 29 та 45 є значення 6Е. Результатом операції інверсії числа 5 є значення FA. Якщо з 45 відняти 5, буде отримане число 40. Саме в такій послідовності з'являються числа на порту А. На порту В встановлюється 5 в момент часу, коли лічильник програм приймає значення 7. В наступному такті це значення записується в регістр R6. Зовнішнє переривання виникає при появі значення лог.1 в старшому розряді на порту В. Це відбувається коли мікропроцесор виконує цикл. Лічильник програм переходить на першу комірку пам'яті, в якій мститься адреса обробника переривання (в даному прикладі це значення 17₁₆ або 23₁₀). В обробнику переривання відбувається заборона подальших переривань шляхом запису 0 в регістр GIMSK, а також вивід значення

0xFF в порт А. Після виходу з обробника переривання основна програма починає виконуватись з того ж місця, в якому виникло переривання.

При виконанні команди RCALL і виклику підпрограми стек запам'ятовує стан процесора, а при виконанні команди RETI – видає вектор стану, який знаходиться на вершині стеку. Вектор стану процесора в даному прикладі складається з адреси повернення, яка дорівнює поточному адресу команди. В традиційних RISC-процесорах при викликах підпрограм адрес повернення запам'ятовується в одному із загальних регістрів і потім зберігається в програмному стеку. Завдяки апаратно реалізованому стеку, обробка переривань виконується з високою швидкістю.

Проведемо часовий аналіз розробленого мікропроцесора. Зазначимо, що максимально припустима частота тактового сигналу визначається сімейством і типом мікросхеми, для якої проект був відкомпільований. На рис. 9.23 представлені результати часового аналізу для мікросхем APEX20KE EP20K60ETC144-1X (а) та FLEX10K EPF10K40RC208-3 (б). Як видно з рисунків, для мікросхеми FLEX10K максимально припустима частота, при якій мікропроцесор буде зберігати функціональні властивості, більш ніж втричі менша.

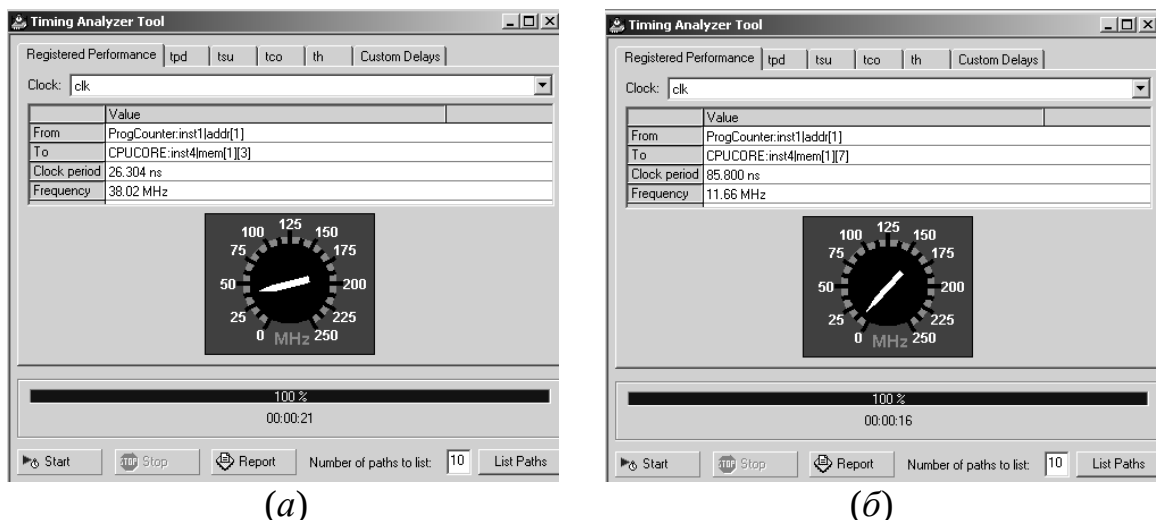


Рис. 9.23.

Для реалізації мікропроцесора знадобилось 2030 логічних комірок. Коректність результатів експериментів з RISC-мікропроцесором перевіряються шляхом моделювання. Процес відлагодження мікропроцесора можна звести до наступного:

- розробка або доробка окремого блоку процесора;
- тестування моделі блоку;
- синтез, розташування і трасировка блоку;
- тестування моделі блоку в складі структури процесора.

З самого початку роботи бажано досягти коректного тестування проекту. Тоді з кожним новим експериментом можна змінювати, доробляти частини

проекту, впевнюючись в їх працездатності. При невдачах можна повернути стан проекту назад і проаналізувати причини появи помилок.

Питання до розділу 9

1. Традиційні методи проектування цифрових пристроїв та їх недоліки.
2. Сучасні методи проектування цифрових пристроїв.
3. Методи проектування цифрових пристроїв на ПЛІС.
4. Структура опису пристрою на мові Verilog.
5. Опис інтерфейсу пристрою на мові Verilog.
6. Опис поведінки пристрою на мові Verilog.
7. Регістри та їх типи у мові Verilog.
8. Ланцюги (лінії зв'язку) та їх види у мові Verilog.
9. Чисельні типи даних у мові Verilog.
10. Масиви та побітова адресація у мові Verilog.
11. Умовні оператори у мові Verilog.
12. Оператори вибору (case) та їх види у мові Verilog.
13. Оператори циклу та їх види у мові Verilog.
14. Послідовні блоки у мові Verilog.
15. Паралельні блоки у мові Verilog.
16. Блок always у мові Verilog.
17. Блок initial у мові Verilog.
18. Блок assign у мові Verilog.
19. Рівні абстракції при проектуванні цифрових пристроїв

Задачі до розділу 9

1. Однією з задач аналізу сигналів є розкладання сигналу на елементарні складові. Для роботи з дискретними сигналами використовується дискретне перетворення Фур'є (ДПФ). Розробити модуль, який виконує ДПФ над фрагментом сигналу, який представлений 32 значеннями. Розрядність даних дорівнює 10. Перетворення виконується відповідно до виразу:

$$\dot{X}[n] = \sum_{k=0}^{N-1} x[k] \exp(-j \frac{2\pi nk}{N}),$$

де N – кількість вибірок, $x[k]$ – вхідні дискретні значення сигналів, n – номер гармоніки. Для перетворення комплексної експоненти виконати перетворення за формулою Ейлера.

2. Часто виникає задача визначення ступені схожості одного набору даних з іншим, детектування та ідентифікація сигналів в шумі, дослідження впливу входу системи на вихід та ін. Для вирішення цих задач використовується кореляційних аналіз. Розробити модуль, який визначає взаємну кореляцію R_{12} двох послідовностей даних $x_1[n]$ $x_2[n]$. Для визначення взаємної кореляції використовується наступний вираз:

$$R_{12} = \frac{1}{N} \sum_{n=0}^{N-1} x_1[n]x_2[n],$$

де N – кількість елементів у вхідних масивах даних.

3. В існуючий час все більша кількість інформації, що передається по лініях зв'язку, існує в цифровому вигляді. Кожному з можливих символів повідомлення ставиться у відповідність набір параметрів несучого коливання, при чому параметри несучого коливання змінюються раптово. Розробити модуль, який виконує:

- а) амплітудну маніпуляцію;
- б) частотну маніпуляцію;
- в) фазову маніпуляцію;
- г) квадратурну маніпуляцію.

Передачі підлягає множина з 8 чисел. Відліки несучого коливання можна тримати в модулі постійного запам'ятовуючого пристрою. Частота маніпульованого сигналу повинна бути не менше ніж на порядок вища за символну швидкість.

4. Для тестування багатьох приладів часто виникає потреба у генераторі складних сигналів. Розробити модуль генератора складних сигналів (не менше 10 типів). Модуль повинен мати входи вибору типу сигналу. Розрядність вихідних даних дорівнює 8. Форми сигналів вибрати самостійно.
5. Розробити модуль цифрового фільтру зі скінченною імпульсною характеристикою, структура якого представлена на рис. 9.24.

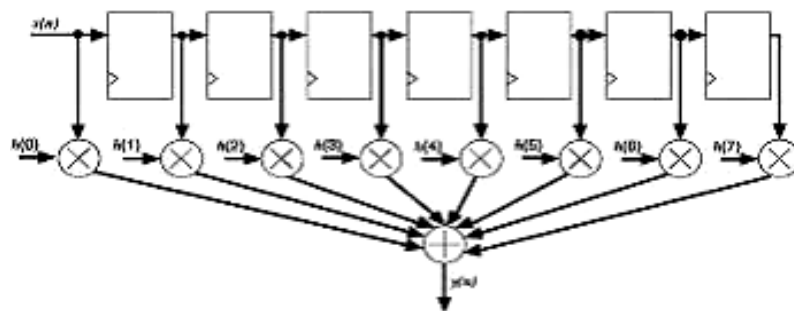


Рис. 9.24.

Розрядність вхідних даних дорівнює 8. Модуль повинен мати входи для задання коефіцієнтів фільтру.

6. Розробити модуль цифрового фільтру з нескінченною імпульсною характеристикою четвертого порядку. Рівняння, яке описує роботу фільтру, має вигляд:

$$y(n) = \sum_{i=0}^{\infty} a(i)x(n-i) - \sum_{i=1}^{\infty} b(i)y(n-i)$$

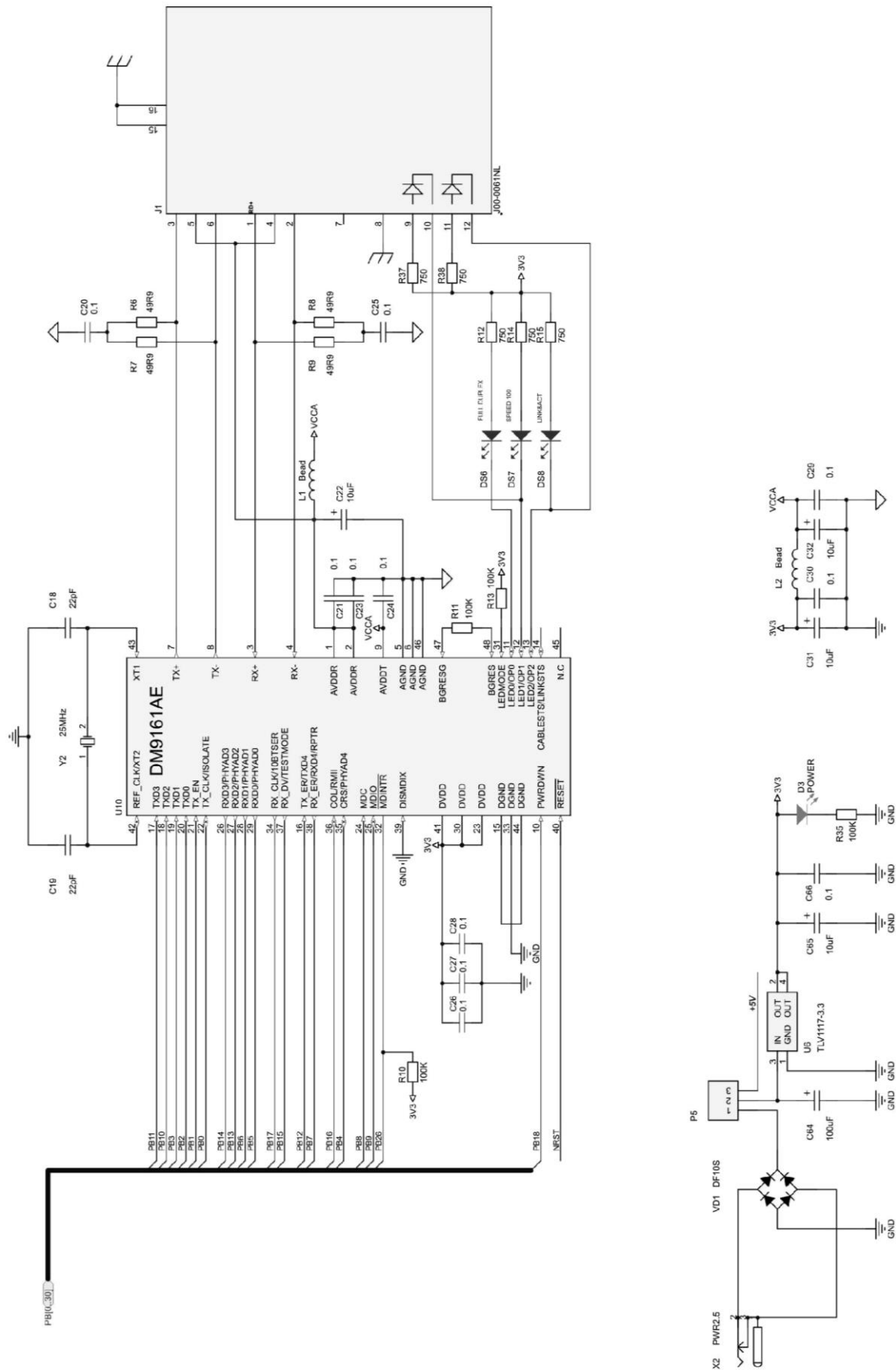
де $a(i)$ та $b(i)$ – коефіцієнти фільтру, які визначають його частотну характеристику, $x(n)$ – вхідні відліки, $y(n)$ – вихідні

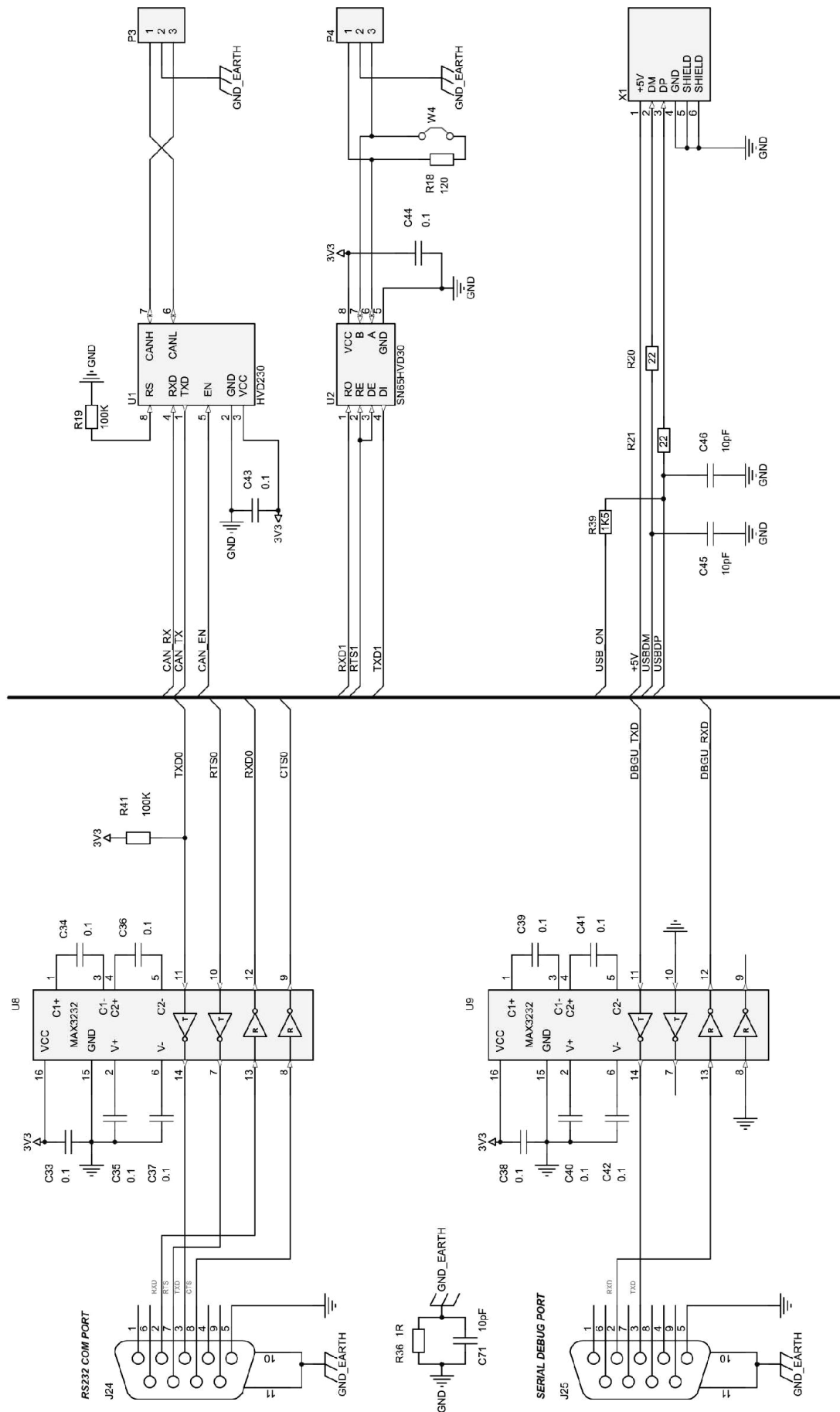
значення фільтру. Значення n визначає порядок фільтру. Розрядність вхідних даних дорівнює 8.

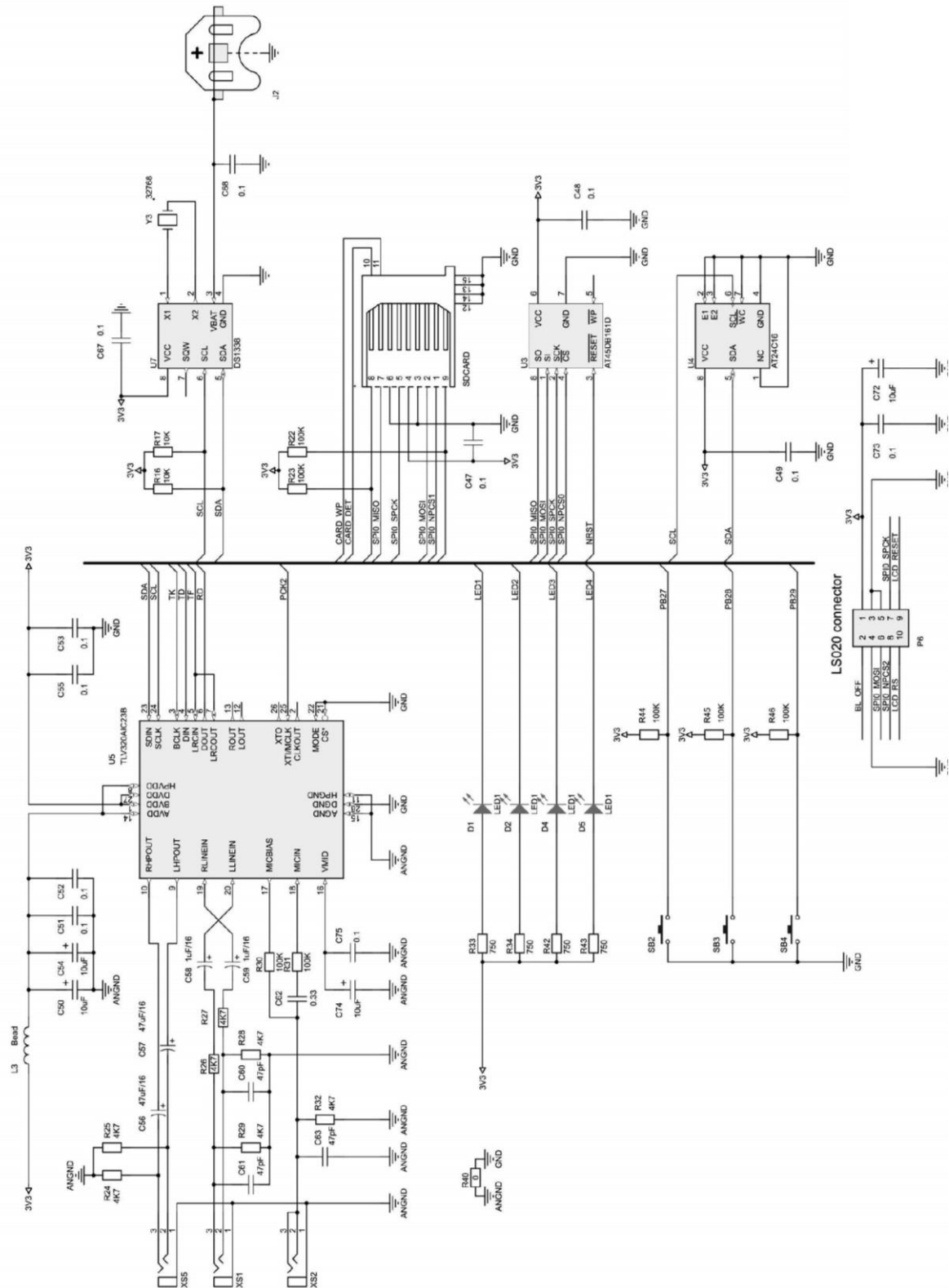
7. При цифровій обробці сигналів часто застосовуються різноманітні математичні функції. Наприклад, в простих алгоритмах компресії даних, деяких алгоритмах нелінійної обробки сигналів використовується функція логарифму. При моделюванні складних дискретних систем зустрічається функція кореню. Розробити модуль, який визначає логарифм по модулю 2 та корінь вхідних даних.
 8. Розробити модуль драйвера алфавітно-цифрового рідкокристалевого індикатора 16×2 , сумісного з системою команд контролера HD44780. Модуль повинен мати входи вибору строчки, позиції символу, вхід запису символу в задану позицію. Також в модулі повинні бути входи прокрутки та очистки екрану. Вхідні дані в модуль – ASCII - коди символів, що будуть виводитися на екран дисплею.
 9. Розробити модулі (шифратор та дешифратор) криптографічного захисту інформації для однобайтових даних. Модуль повинен мати вхід вибору алгоритму шифрування даних (4 варіанти). Дані завантажуються в модуль в паралельному форматі по фронту тактового сигналу, а передаються в послідовному форматі з паузою в три біти між переданими байтами.
 10. Розробити параметризований модуль для виконання арифметичних операцій додавання та віднімання. Модуль повинен мати два виходи. На першому виході дані з'являються в двійковому коді, на другому – в двійково-десятковому. Також повинен бути вихід переповнення.
 11. Розробити модуль двійкового множення на суматорах та логічних елементах I, що дозволяє множити чотирьохрозрядне число A на трьохрозрядне число B .
 12. Розробити модуль множення двох однобайтових слів використовуючи:
 - а) оператор множення „*”;
 - б) параметризований модуль з бібліотеки САПР.
- Проаналізувати часові затримки в кожному з реалізованих варіантів.
13. Спроектувати синхронний тактований автомат Мура, який виконував би перевірку лінії даних, що передаються, на парність. Автомат повинен мати два входи – вхід синхронізації C – і вхід даних D –, а також вхід тактового генератора, і один вихід y , високий рівень якого встановлює дев'ятий біт в 1, якщо кількість одиниць в однобайтовому слові непарна.
 14. Розробити модуль асоціативного запам'ятовуючого пристрою з загальним асоціативним накопичувачем, який має 4-бітову адресну шину комірок пам'яті (A), однобайтову шину даних (D), шину маски (MK), входи сигналів запису, зчитування та пошуку даних (WR , RD , FN), входи сигналів пошуку за адресою та даними (FA , FD), а також входи сигналів режиму запису та адресації (MO , MOA), сигнал стану (SA). Модуль виконує операції у відповідності до таблиці:

FA	FD	FN	MOA	MO	RD	WR	Виконувана операція
1	X	X	1	X	1	0	Запис за адресою
1	X	X	1	X	0	1	Зчитування за адресою
X	X	0	X	X	1	1	Пошук за ознакою
X	0	X	0	1	0	1	Зчитування за ознакою
X	0	X	0	1	1	0	Запис за ознакою









ПЕРЕЛІК ЛІТЕРАТУРИ

1. Ю.І.Якименко, Т.О.Терещенко, Є.І.Сокол, В.Я.Жуйков, Ю.С.Петергеря. Мікропроцесорна техніка / Під ред. Т. О. Терещенко. – К.: «Політехніка»; "Кондор", 2004 р. – 440 с.
2. Баранов В.Н. Применение микроконтроллеров AVR.Схемы, алгоритмы, программы / В.Н. Баранов. – М.: Издательский дом «Додэка-XXI», 2004. – 288 с.: ил.(серия «Мировая электроника»).
3. Евстифеев А.В. Микроконтроллеры AVR семейств Tiny и Mega фирмы "ATMEL" / А.В. Евстифеев. – М.: Издательский дом «Додэка-XXI», 2004. – 558 с.: ил.(серия «Мировая электроника»).
4. Голубцов М.С. Микроконтроллеры AVR: от простого к сложному / М.С. Голубцов. – М.: «Солон-пресс», 2003. – 288 с.: ил.
5. Гребнев В. В. Микроконтроллеры семейства AVR фирмы Atmel / В.В. Гребнев. – М.: ИП РадиоСофт, 2002. – 174 с.: ил.
6. Мортон Д. Микроконтроллеры AVR. Вводный курс / Д. Мортон. – М.: Издательский дом «Додэка-XXI», 2006. – 272 с.: ил.
7. Швец В. А., Шестакова В. В., Бурцева Н. В., Мелешко Т. В. Одноплатные микроконтроллеры. Проектирование и применение / В.А. Швец, В.В. Шестакова, Н.В. Бурцева и др. – К.: "МК-Пресс", 2005. – 304 с.: ил.
8. Трамперт В. AVR-RISC микроконтроллеры / В. Трамперт. – М.: МК-Пресс, 2006. – 484 с.: ил.
9. Рябенский В.М., Ушкаренко О.О. Verilog. Практика проектування цифрових пристроїв на ПЛІС. Навчальний посібник / В.М. Рябенский, О.О. Ушкаренко. – Миколаїв: Вид-во «Іліон», 2007. – 324 с.: ил.
10. Рябенский В.М., Ходаков В.Е., Ушкаренко А.О. Компьютерное управление внешними устройствами через стандартные интерфейсы. Учебное пособие / В.М. Рябенский, В.Е. Ходаков. А.О. Ушкаренко. – Херсон: Олди-плюс, 2008. – 380 с.: ил.
11. Соловьев В. В. Проектирование цифровых систем на основе программируемых логических интегральных схем / В.В. Соловьев. – М.: Горячая линия – Телеком, 2001. – 636 с.
12. Угрюмов Е. П. Цифровая схемотехника / Е.П. Угрюмов. – СПб.: БХВ-Петербург, 2000. – 528 с.
13. Поляков А.К. Языки VHDL и Verilog в проектировании цифровой аппаратуры. –М.: СОЛОН-Пресс, 2003. – 320 с.
14. www.gaw.ru/html.cgi/txt/doc/micros/arm/arh/index.htm
15. www.altera.com
16. www.atmel.com

ЗМІСТ

Вступ	3
Умовні скорочення	4
1. ЗАГАЛЬНІ ВІДОМОСТІ ПРО МІКРОПРОЦЕСОРИ	5
1.1. Основні поняття та визначення	5
1.2. Загальна структура мікропроцесорної системи	10
1.3. Мікропроцесорна система та її особливості	12
1.4. Режими роботи мікропроцесорної системи	16
1.4.1. Програмний обмін інформацією	16
1.4.2. Обмін по перериваннях	17
1.4.3. Прямий доступ до пам'яті	18
1.5. Загальна характеристика пристроїв пам'яті мікропроцесорної системи	20
1.6. Методи адресації до зовнішньої пам'яті	21
1.7. Обмін інформацією в мікропроцесорних системах	23
1.7.1. Архітектури мікропроцесорних систем	30
1.7.2. Типи мікропроцесорних систем	32
1.7.3. Функції процесора	33
1.7.4. Функції пам'яті	38
1.7.5. Функції пристроїв вводу/виводу	42
1.8. Організація однокристальних мікроконтролерів	44
1.9. Система команд процесора	46
1.10. Структура мікроконтролерів сімейства AVR	52
1.10.1. Основні характеристики процесора	55
1.10.2. Організація пам'яті	56
1.11. Характеристика периферійних пристроїв	62
Питання до розділу 1	65
2. ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ НА МОВІ CІ	66
2.1. Компілятор мови Cі ImageCraft	66
2.2. Синтаксис мови	72
2.2.1. Коментарії в програмі	72
2.2.2. Зарезервовані слова	73
2.2.3. Типи даних	73
2.2.4. Константи	73
2.2.5. Змінні	75
2.2.6. Організація SRAM–пам'яті	76
2.2.7. Математичні операції	80
2.2.8. Функції	80
2.3. Приклади програм	85
Питання до розділу 2	87
3. ПЕРИФЕРІЙНІ ПРИСТРОЇ МІКРОКОНТРОЛЕРА	88
3.1. Лабораторний стенд для вивчення мікроконтролерів	88
3.2. Паралельні порти вводу–виводу	91

3.2.1. Приклади програм виведення даних на індикатори різних типів	93
3.2.2. Введення даних в мікроконтролер	106
3.2.3. Керування алфавітно–цифровими РКІ	115
3.2.4. Керування графічними РКІ	125
3.3. Універсальний асинхронний приймач–передавач UART	137
3.3.1. Формати передачі даних	138
3.3.2. Ініціалізація та налаштування швидкості зв'язку	139
3.3.3. Передача та приймання даних	139
3.3.4. Багатопроцесорний режим зв'язку	145
3.3.5. Сполучення мікроконтролера з комп'ютером	149
3.4. Аналого–цифровий перетворювач	157
3.4.1. Регістри керування АЦП	160
3.4.2. Приклади програмування АЦП	162
3.5. Аналоговий компаратор	168
3.6. Таймери–лічильники загального призначення	171
3.6.1. Функціональні схеми 8– та 16–розрядних таймерів	171
3.6.2. Регістри керування таймерами	174
3.6.3. Приклади програм з таймером–лічильником	181
3.8. Обробка зовнішніх переривань	185
3.9. Сторожовий таймер	189
3.10. Налагодження програм в AVR–Studio	191
3.11. Апаратні засоби підтримки розробок для мікроконтролерів	197
Питання до розділу 3	203
Задачі до розділу 3	205
 4. ПОСЛІДОВНИЙ ІНТЕРФЕЙС SPI	 208
4.1. Блок–схема SPI	208
4.2. Конфігурація SPI	211
4.3. Мікросхеми пам'яті SD/MMC	212
4.3.1. Команди керування	214
4.3.2. Протокол обміну даними	215
4.3.3. Приклад програми роботи з пам'ятю	218
4.4. Організація обміну даними між мікроконтролерами	223
Питання до розділу 4	226
Задачі до розділу 4	227
 5. ШИНА I2C	 228
5.1. Послідовний порт TWI	228
5.2. Діаграми передачі даних	231
5.3. Адресація пристроїв	232
5.4. Опис регістрів TWI	234
5.5. Рекомендації по використанню TWI	237
5.6. Приклади пристроїв з інтерфейсом I2C	240

5.6.1. Цифровий термометр на базі DS1621	240
5.6.2. Електронний годинник на базі DS1307	246
Питання до розділу 5	254
Задачі до розділу 5	254
 6. ШИНА 1–WIRE	 255
6.1. Загальні характеристики інтерфейсу	255
6.2. Блок–схема датчика температури DS18B20	257
6.3. Протокол обміну даними	260
6.4. Мікропроцесорна система контролю температури	264
Питання до розділу 6	268
Задачі до розділу 6	268
 7. ШИНА USB	 269
7.1. Огляд методів реалізації передачі даних по USB	269
7.2. Структура USB на фізичному рівні	270
7.3. Протоколи та канали передачі даних	273
7.4. Пакети керуючих передач та передач по перериванню	276
7.5. Використання мікроконтролера AT90USB1286 для передачі даних	279
7.5.1. Структура контролера AT90USB1286	280
7.5.2. Принципова схема пристрою	284
7.6. Програмна реалізація протоколу	286
7.6.1. Модуль низькорівневої ініціалізації	287
7.6.2. Модуль програмної реалізації USB	293
7.6.3. Функції ініціалізації USB–пристрою	298
7.7. Моделювання пристрою в Proteus	309
7.8. Розробка програми–терміналу для HID–пристроїв	310
Питання до розділу 7	320
Задачі до розділу 7	320
 8. 32–РОЗРЯДНІ МІКРОКОНТРОЛЕРИ З АРХІТЕКТУРОЮ ARM	 321
8.1. Основні характеристики та архітектура ядра ARM7	322
8.2. Мікроконтролери сімейства AT90SAM7S	323
8.2.1. Структурна схема	324
8.2.2. Призначення виводів та живлення мікроконтролера	327
8.3. Робота з середовищем розробки Keil μVision	329
8.4. Периферійні модулі 32–розрядних мікроконтролерів	333
8.4.1. Контролери паралельного вводу–виводу (PIO)	337
8.4.2. Робота з контролером переривань (AIC)	340
8.4.3. Робота з аналого–цифровим перетворювачем (ADC)	344
8.4.4. Модуль широтно–імпульсної модуляції (PWM)	347
8.4.5. Таймери–лічильники (PIT, RTC, TCNT)	351
8.4.6. Обмін даними з використанням USART	358

8.4.7. Використання двохпровідного інтерфейсу TWI для запису/зчитування даних	362
Питання до розділу 8	367
Задачі до розділу 8	367
 9. РОЗРОБКА СПЕЦІАЛІЗОВАНИХ МІКРОПРОЦЕСОРНИХ МОДУЛІВ НА ПРОГРАМОВАНИХ ЛОГІЧНИХ ІНТЕГРАЛЬНИХ СХЕМАХ	 369
9.1. Основи Verilog. Структура модулів	370
9.2. Арифметичні пристрої та схеми контролю	380
9.2.1. Суматори	380
9.2.2. Компаратори	383
9.2.3. Арифметично–логічні пристрої (АЛП)	384
9.2.4. Контроль парності	386
9.2.5. Перемножувачі та подільовачі	389
9.3. Оперативна пам'ять	392
9.3.1. Пам'ять типу LIFO	397
9.3.2. Пам'ять типу FIFO	400
9.4. Постійна пам'ять	402
9.5. Розробка асинхронних приймача і передавача	404
9.6. CRC–алгоритми виявлення помилок	413
9.7. Проектування RISC–мікропроцесора	415
Питання до розділу 9	432
Задачі до розділу 9	433
 Додаток А. Принципова схема демоплати Etherus (ETH256)	 436
 ПЕРЕЛІК ЛІТЕРАТУРИ	 440

Навчальне видання

**РЯБЕНЬКИЙ Володимир Михайлович
УШКАРЕНКО Олександр Олегович
БУРЯК Володимир Святославович**

**СХЕМОТЕХНІКА
ЕЛЕКТРОННИХ ПРИСТРОЇВ
ТА СИСТЕМ**

Підручник

У ШЕСТИ ТОМАХ

ТОМ 3

МІКРОПРОЦЕСОРНА ТЕХНІКА

Рисунки – *Я. Дубовик*
Коректура та комп'ютерна верстка – *О. Матвєєва*

Формат 60x84¹/₁₆. Ум. друк. арк. 22,7. Тираж 100 пр. Зам. № 534-067.

В И Д А В Е Ц Ї

Товариство з обмеженою відповідальністю фірма «Ліон».
54001, м. Миколаїв, вул. Садова, 1.
Свідоцтво суб'єкта видавничої справи ДК № 1506 від 25.09.2003 р.

В И Г О Т О В Л Ю В А Ч

Поліграфічне підприємство СПД Румянцева Г.В.
54001, м. Миколаїв, вул. Садова, 1.
Свідоцтво суб'єкта видавничої справи МК № 11 від 26.01.2007 р.