# ВСТУП

Конструювання радіоелектронної схеми (РЕС) - складний процес, що не має поки суворої всеохоплюючої математичної бази та ведеться методом численних спроб, хоча рішення багатьох проблем конструювання засноване на використанні строгого математичного алгоритму, розрахунку теплових режимів, міцності, різних допусків. Тому незначні на перший погляд похибки або наближення, допущені на ранніх стадіях розробки РЕС, можуть стати причиною великих помилок у подальшій роботі. Широке використання радіоелектронної апаратури в різних галузях науки і техніки призводить до необхідності забезпечення високої надійності її роботи при різноманітних кліматичних і механічних впливах. Труднощі виконання даної вимоги пов'язана з різним призначенням РЕС, місцем її установки і умовами експлуатації. Про рівень надійності прийнято судити по здатності РЕС безвідмовно працювати протягом заданого часу в певних умовах. Тому для оцінки ступеня відповідності РЕС висунутим до неї вимогам здійснюють технічний контроль та випробування на всіх етапах конструювання і виробництва. Отримана при цьому інформація про якість роботи РЕС та про причини її відмов спільно з даними реальної експлуатації дозволяє прийняти своєчасні заходи щодо вдосконалення схеми і конструкції, а так само технології виробництва. Правильно організований технічний контроль і випробування сприяють значному підвищенню надійної роботи РЕС. У зв'язку з цим особливої ​​актуальності набувають радіотехнічні пристрої, призначені для випробування і контролю, за допомогою яких перевіряється працездатність та придатність до експлуатації різного устаткування і приладів.

 В останні роки з розвитком електроніки ускладнюється радіоапаратура. Характерний перехід від окремих радіоапаратів до складних комплексам і системам. Проте створення подібної апаратури ставить перед її розробниками різні проблеми: ваги, обсягів і габаритів; надійності і довговічності; виробництва і серійності; економіки. Для подолання цих проблем необхідно їх комплексне рішення на загальній технічній базі. Такою базою в даний час є комплексна механізація із застосуванням мікроелектроніки, впровадженням уніфікації та стандартизації. Розробка схеми конструкції і технологію стає єдиним процесом створення апаратури. Відпрацювання серійної здатності кожного виробу починається з моменту його зародження і продовжується в процесі його виробництва.

# 1 ПРИЗНАЧЕННЯ І ОБЛАСТЬ ЗАСТОСУВАННЯ ПЗ

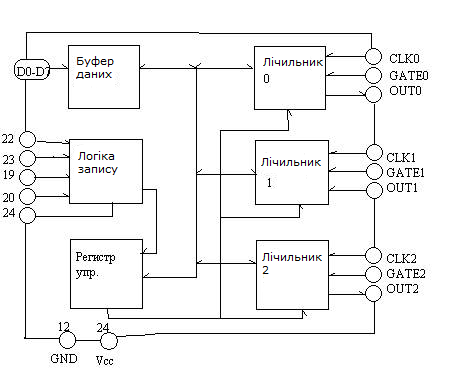
**1.1 Призначення системи**

В обчислювальних машинах використовують програмований системний таймер (ПСТ).

Його функції:

* Генерація переривань від системного годинника, що викликає інкремент лічильника системного часу в осередок з адресою 0000: 046Ch (лічильник тиків таймера).
* Генерація запитів генерацію пам'яті
* Генерація звукових сигналів

При ініціалізації BIOS встановлює свій обробник для переривання таймера. Цей обробник щораз збільшує на одиницю поточне значення 4-байтовой змінної - лічильника тиків таймера. Якщо цей лічильник переповнюється, тобто пройшло більше 24 годин з моменту запуску таймера, то в комірку з адресою 0000: 0470h заноситься одиниця. Інша дія, виконувана стандартним обробником переривань таймера - це контроль за роботою двигунів НГМД (аналог флопа). Якщо після останнього звернення до НГМД пройшло більше 2 секунд, обробник переривання вимикає двигун, осередок з адресою 0000: 0440h містить час, що залишився до вимкнення двигуна. Ще одна дія, яка виконує обробник переривання таймера - це виклик переривання INT1Ch. Типово по цьому перериванню нічого не виконується, однак програма може встановити власний обробник цього переривання для виконання будь-яких періодичних дій. В якості ПСТ раніше застосовувалася мікросхема Intel8254 роль якої на сучасних системних платах виконує чіпсет, зберігаючи повну програмну сумісність. ПСТ являє собою 3-канальні програмовані лічильники таймера з незалежними входами: CLK - вхід рахункових імпульсів, GATE - вхід дозволу рахунку і виходу OUT, що змінює стан по закінченню рахунку.



Кожен канал (0,1,2) містить регістри:

* стану каналу (8 розрядів)
* Керуючого слова RSW (8 розрядів)
* Буферний регістр OL (16 розрядів)
* Регістр лічильника CE (16 розрядів)
* Регістр констант перерахунку CR (16 розрядів)

Регістр лічильника CE працює в режимі віднімання. Його міст зменшується за умови, що на вхід GATE встановлений рівень логічної одиниці. Залежно від режиму роботи таймера, по досягненні лічильником CE нуля тим чи іншим чином змінюється вихідний сигнал OUT

Буферний регістр OL призначений для запам'ятовування поточного вмісту регістра лічильника СЕ без зупинки процесу рахунку. Після запам'ятовування буферний регістр доступний програмі для читання.

Регістр констант перерахунку CR може завантажуватися в регістр лічильника якщо потрібно в поточному режимі робота таймера.

Регістри стану каналу RS і керуючого слова RSW призначені для визначення поточного стану каналу і для завдання режиму роботи таймера.

Можливі шостій режимів роботи таймера:

* Режими 0,4 однократне виконання функцій
* Режими 1,5 робота з перезапуском
* Режими 2,3 робота з автозавантаженням

**1.2 Область застосування**

Таймер на PIC - контролері (англ. Timer - призначати час), призначений для автоматичного включення (викл.) Пристрою, після закінчення заданого проміжку часу, його відображення і сигналізації моменту включення (викл.) Пристрою.

Таймер на PIC - контролері (Таймер) забезпечує автоматичне включення і виключення різних електричних приладів, завдяки чому має широкий спектр застосування. Він здатний створити ефект присутності мешканців у квартирі або заміському будинку. Таймер в різних приміщеннях може запалювати або гасити світло, включать радіо або телевізор і т. Д. Він допоможе запрограмувати час дії побутових приладів (кавоварки, хлібопічки, пральної машини та інших). Дозволить обігріти дачу перед приїздом господарів: у потрібний момент включить обігрівач або електрокамін. Також здатний автоматизувати полив ділянки, парників, оранжерей (включити насоси кілька разів на день на певний час).

**2 ОГЛЯД ІСНУЮЧОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ. ПОСТАНОВКА ЗАДАЧІ**

**2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень**

Системні таймери - це таймери, до яких звертається операційна система для відстеження часу дня (дати) і обробки запланованих, що залежать від часу подій. З розвитком комп'ютерних технологій процесорні швидкості збільшилися на кілька порядків, а додатки стали набагато складнішими. Така еволюція висунула нові вимоги до архітектури таймерів, враховуючи роздільну здатність таймера і швидкість відгуку. Спроби задовольнити ці вимоги на сьогоднішніх системах змусили розробників пожертвувати якістю додатків і продуктивністю системи в цілому для подолання обмежень сьогоднішніх таймерних архітектур. Для задоволення потреб

додатків і підтримки складних додатків в майбутньому системні таймери повинні надати роздільну здатність як мінімум в 1 мілісекунду.

У персональних комп'ютерах на даний момент для різних завдань використовують чотири системних таймера, кожен з яких має власний набір атрибутів, що робить роботу з ними складною і заплутаною.

+8254 Programmable Interval Timer (PIT) був представлений в персональному комп'ютері IBM в 1981 році. Він має роздільну здатність в 1 мілісекунду і підтримує як періодичний, так і апериодический режим. Оскільки зчитування і запис на цей пристрій вимагають комунікації через порт вводу / виводу, використання його для операційної системи

дуже ресурсомісткі. Через це аперіодична функціональність даного таймерів використовується на практиці. З цієї ж причини даний таймер використовується тільки в періодичному режимі для здійснення періодичних переривань в однопроцесорних системах.

У 1984 р IBM ввели Перший таймер реального часу real-time clock (RTC) на додаток до 8254. Як і +8254, RTC має максимальний дозвіл в 1 мілісекунду і підтримує періодичний і апериодический режими. Як і у випадку з 8254, комунікація з пристроєм відбувається через порт вводу / виводу, і, відповідно, ця процедура ресурсомісткі. На додаток до цих обмежень RTC був розроблений для генерації переривань Advanced Configuration and Power Interface (ACPI) на системах, що підтримують ACPI, що знижує продуктивність системи. RTC використовується в періодичному режимі для здійснення переривань системного профілю на одно-процесорних системах і для переривань годин в мультипроцесорних

системах.

У Advanced Programmable Interrupt Controller (APIC) -Таймер є багато можливостей таймера рельной часу і він був розроблений для використання в синхронізації багатьох процесорів, але має погане дозвіл, а його чіп буває повільним. Оскільки багато системи не мають APIC-таймера, він не використовується за умовчанням.

ACPI-таймер, відомий як PM Clock, був доданий в архітектуру для надання надійних тимчасових відбитків, що залежать від процесорної швидкості. У зв'язку з цим таймер НЕ несе ніякої іншої функціонального навантаження.

Системи потребують апериодической функціональності і більш високої точності. Без цих двох пунктів системи будуть не в змозі задовольнити потреби майбутніх мультимедійних та чутливих до часу додатків. Новий таймер, який надасть таку функціональність, замінить таймер +8254, таймер реального часу, PM Clock і APIC- таймер і дозволить прибрати застарілі чіпи з архітектури.

Microsoft Windows XP використовує періодичні переривання для відстеження часу, спрацьовування таймерних об'єктів і зменшення квантів часу, виділених на роботу тредов. Коли Windows XP завантажується, за замовчуванням період переривання складає 10 мілісекунд, а іноді 15 на деяких системах. Це означає, що кожні 10 мілісекунд операційна система

отримує переривання від вбудованого в архітектуру таймера. Коли спрацьовує годинне переривання, Windows здійснює два основних дії:

оновлює лічильник тиків, якщо пройшов повний тик з попереднього оновлення, і перевіряє, чи не минув термін придатності запланованих таймерних об'єктів. Тік - це абстрактна одиниця часу, яку використовує Windows для відстеження часу дня і завершеності кванта треду. На більшості систем тик становить 10 мілісекунд, на деяких - 15. За замовчуванням годинне переривання відбувається кожен тик, але операційна система або додатки можуть змінювати період часових переривань, в той час, як період таймерного тіка змінюватися не може. Отже, тик завжди постійний, незалежно від періоду часових переривань. На додаток до відліку тиків під час годинних переривань система перевіряє, чи не минув термін таймерного об'єкта, і планує відкладений процедурний виклик (deferred procedure call), якщо такий знайдений. Таймерні об'єкти використовуються системою для відстеження термінів і оповіщення додатків про досягнення цих термінів. Наприклад, мультимедійне додаток, програє синхронізовані аудіо та відео, може запросити, щоб частина коду (наприклад, програвання звуку) була виконана через 233 мілісекунди. Windows відстежує цей запит запуском таймерного об'єкта. Згодом, коли генеруються годинні переривання, перевіряються таймерні об'єкти на досягнення потрібного часу. Очевидно, що при періоді часових переривань в 10 мілісекунд операційна система програє звук з 7-мілісекундним запізненням. Якщо операційна система перевіряє таймерні об'єкти в 230 мілісекунд і бачить, що жоден з них не закінчився, наступна перевірка відбувається в 240 мілісекунд, і тут виявляється запізнілий запит. В даному випадку відстань між вартовими перериваннями занадто велике і кінцевий користувач може помітити проблеми синхронізації аудіо / відео.

Очевидним недоліком такої імплементації є великий період часових переривань, який може вилитися у великі затримки. Для того щоб програми могли отримувати повідомлення на сьогоднішніх системах з більшою точністю, вони повинні запитувати менший період часових переривань. При розгляді попереднього прикладу мультимедійне додаток повинен запросити годинні переривання з періодичністю в одну мілісекунди для того, щоб код був виконаний точно вчасно.

**2.2** **Обґрунтування вибору засобів для побудови системи**

**2.3 Обгрунтування вибору мови програмування**

Асе́мблер - загальноприйнята назва [транслятора](https://uk.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BB%D1%8F%D1%82%D0%BE%D1%80) з [автокоду](https://uk.wikipedia.org/w/index.php?title=%D0%90%D0%B2%D1%82%D0%BE%D0%BA%D0%BE%D0%B4&action=edit&redlink=1). Асемблер переводить початкову [програму](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B0), написану на автокоді, в переміщувану програму на [мові машинній](https://uk.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0). Оскільки асемблер здійснює [трансляцію](https://uk.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BB%D1%8F%D1%86%D1%96%D1%8F) на мову [завантажувача](https://uk.wikipedia.org/w/index.php?title=%D0%97%D0%B0%D0%B2%D0%B0%D0%BD%D1%82%D0%B0%D0%B6%D1%83%D0%B2%D0%B0%D1%87&action=edit&redlink=1), при завантаженні програми необхідна налаштування умовних адрес, тобто адрес, значення яких залежать від розташування даної програми в пам'яті [ЦВМ](https://uk.wikipedia.org/w/index.php?title=%D0%A6%D0%92%D0%9C&action=edit&redlink=1) і від її зв'язків з іншими незалежно трансльованими програмами.

У простому випадку асемблер переводить одне речення початкової програми в один об'єкт (команду, [константу](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%81%D1%82%D0%B0%D0%BD%D1%82%D0%B0)) модуля завантаження (т. з. трансляція «один в один»). При цьому взаємне розташування об'єктів в модулі завантаження і, зрештою, в пам'яті машини визначається порядком пропозицій в початковій програмі на автокоді і повністю залежить від програміста. Асемблер виконує і допоміжні функції, такі, як підготовка до друку документів необхідної форми, реєстрація зв'язків даної програми з іншими програмами і т. д. Для цієї мети в автокодах передбачаються команди асемблера, які не породжують об'єктів в робочій програмі і призначені тільки для вказівки допоміжних дій асемблера.

Трансляція зазвичай вимагає двох переглядів початкової програми: при першому перегляді здійснюється [розподіл пам'яті](https://uk.wikipedia.org/w/index.php?title=%D0%A0%D0%BE%D0%B7%D0%BF%D0%BE%D0%B4%D1%96%D0%BB_%D0%BF%D0%B0%D0%BC%27%D1%8F%D1%82%D1%96&action=edit&redlink=1) і надання значень символічним іменам; при другому — формується робоча програма у вигляді модуля завантаження. В процесі трансляції асемблер проводить повний синтаксичний контроль початкової програми (див. [синтаксичний аналіз програм](https://uk.wikipedia.org/w/index.php?title=%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%87%D0%BD%D0%B8%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D1%96%D0%B7_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC&action=edit&redlink=1)), забезпечуючи при цьому достатньо точну діагностику помилок за місцем і характером.

Розширення можливостей автокодів досягається за рахунок використання [макрокоманд](https://uk.wikipedia.org/wiki/%D0%9C%D0%B0%D0%BA%D1%80%D0%BE%D0%BA%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D0%B0), що будуються за правилами, близькими до правил написання команд автокоду, але що описують складніші функції, для реалізації яких потрібна група звичайних команд. В цьому випадку перед трансляцією проводиться заміна макрокоманд макророзширеннями — послідовностями команд на базовій мові відповідно до макроозначень. У останніх задається прототип макрокоманди із структурою списку параметрів і процедура генерування макророзширення. Транслятор, що виконує функції макрогенератора і асемблера, називається [макроасемблером](https://uk.wikipedia.org/w/index.php?title=%D0%9C%D0%B0%D0%BA%D1%80%D0%BE%D0%B0%D1%81%D0%B5%D0%BC%D0%B1%D0%BB%D0%B5%D1%80&action=edit&redlink=1). При трансляції з мов високого рівня асемблер нерідко використовується для виконання завершальної фази трансляції.

Асемблер (assembler) — система програмування, яка включає [мову](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) асемблера та транслятор з цієї мови. Є мовою [програмування низького рівня](https://uk.wikipedia.org/w/index.php?title=%D0%9C%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%BD%D0%B8%D0%B7%D1%8C%D0%BA%D0%BE%D0%B3%D0%BE_%D1%80%D1%96%D0%B2%D0%BD%D1%8F&action=edit&redlink=1). Чим нижчий рівень мови програмування, тим ближча специфіка роботи програми до самого процесора, для якого вона й була написана. Вважається, що мови низького рівня складніші й потребують більш вузької спеціалізації програміста, оскільки програма написана на асемблері для одного типу процесорів виявиться не завжди придатною для роботи з іншими процесорами.

З іншого боку програми написані на асемблері компактні та швидкі, що теж є немаловажливим.

Поки існують [процесори](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D0%BE%D1%80), буде існувати й асемблер.

# 3 ОПИС ТА ОБГРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ ЩОДО РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Розробка структурної схеми пристрою

Структурна схема - це сукупність елементарних ланок об'єкта і зв'язків між ними, один з видів графічної моделі. Під елементарним ланкою розуміють частину об'єкта, системи управління і т. Д., Яка реалізує елементарну функцію.

Структурна схема призначена для відображення загальної структури пристрою, тобто його основних блоків, вузлів, частин та головних зв'язків між ними. Із структурної схеми повинно бути зрозуміло, навіщо потрібний даний пристрій і як він працює в основних режимах роботи, як взаємодіють його частини. Позначення елементів структурної схеми можуть обиратись довільно, хоча загальноприйнятих правил виконання схем слід дотримуватись.

Структурна схема для програмного забезпечення включає у себе 3 блоки:

* Блок графічних операцій – демонструє всі можливі послідовні дії в роботі програми, які виводяться на графічний пристрій.
* Блок вихідних даних – відповідає за виведення результату.
* Блок вхідних даних – відповідає за введення даних у матриці.

Кожен з блоків складається з менших блоків, кожен з яких представляє реальні структурні компоненти готового системного таймера. Напрям стрілок вказує на напрямок взаємодії компонентів між собою.

Спрощена структурна схема ПТ приведена на рис.3.1. До складу ВІС входять:

* буфер даних *(BD),* призначений для обміну даними і керуючими словами між МП і ПТ;
* схема керування читанням-записом *(RWCU),* що забезпечує виконання операцій введення-виводу інформації в ПТ;
* регістр керуючого слова *(RGR),* призначений для запису керуючих слів, що задають режими роботи лічильників;
* лічильник каналів (СТ0-СТ2).

Таймер містить три незалежних канали. У кожнім каналі є регістр слова управління, 16-ти розрядний програмований лічильник, що працює на вирахуванні в двійковому або двійково-десятковому коді, двобайтний буферний регістр, через який контакти регістра і команди управління зв'язані із системною шиною.

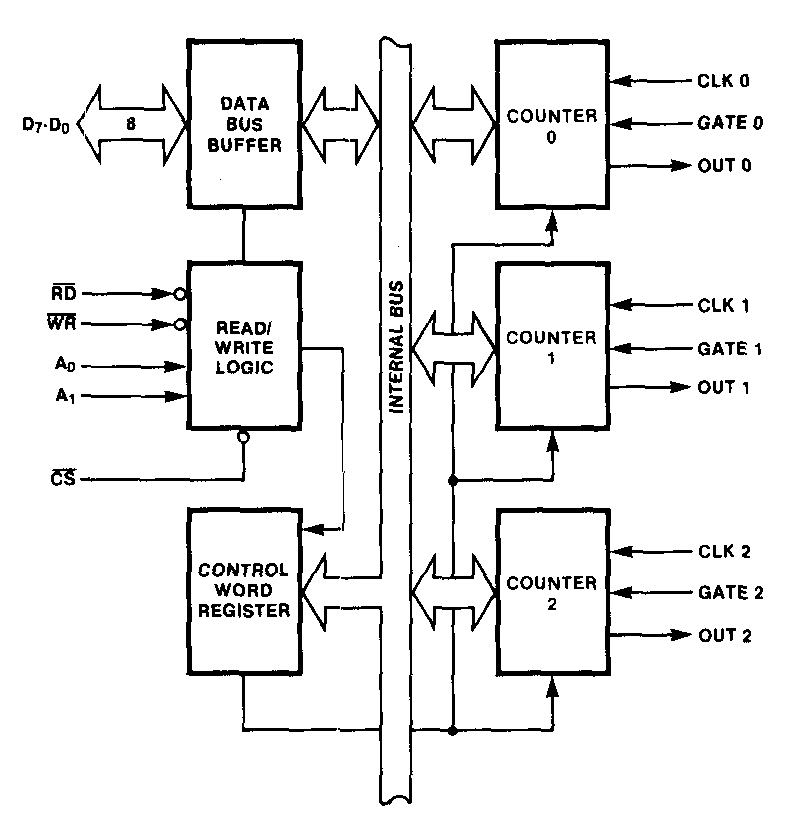


Рисунок 3.1-Спрощена структурна схема таймеру

*DBB -* буфер даних

*RWL -* схема керування читанням-записом

*CWR -* регістрке руючого слова

*СТ0-СТ2 –* лічильник каналів

Канали таймера підключаються до зовнішніх пристроїв за допомогою трьох ліній:

GATE - керуючий вхід;

CLK - вхід тактової частоти;

OUT - вихід таймера.

**3.2 Розробка функціональної схеми системи**

Функціональна схема – це схема яка пояснює певні процеси які відбуваються у функціональних ланках програмного забезпечення або у програмному забезпеченні в цілому. Функціональне коло – це лінія, канал, тракт певного призначення. Функціональна схема містить інформацію про способи реалізації пристроєм заданих функцій.

За такою схемою можна визначити, як здійснюються перетворення і які для цього необхідні функціональні елементи. Кожен функціональний елемент містить лице ті входи і виходи які необхідні для його коректної роботи. Дана схема розробляється на основі структурної схеми для кожного блоку, в результаті з окремих функціональних елементів складається загальні функціональна схема об’єкту.

На основі функціональної схеми та структурної схеми розробляється принципова схема.

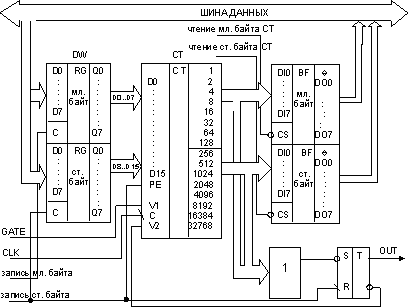
Також функціональна схеми можуть застосовуватися у програмуванні для візуалізації алгоритмів і спрощення обчислення їх складності, однак у цій сфері форма створення – довільна.

Функціональними схемами користуються для вивчення принципу роботи виробів (устаткування), а також при їх налагодженні, контролі чи ремонті. На такій схемі зображують всі функціональні частини виробу та основні зв’язки між ними. Функціональні частини на схемі зображують у вигляді умовних графічних позначень згідно з вимогами державних стандартів. Дозволяється окремі функціональні частини, на яких немає умовних графічних позначень, зображувати у вигляді прямокутників, а також розкривати до рівня принципових схем.

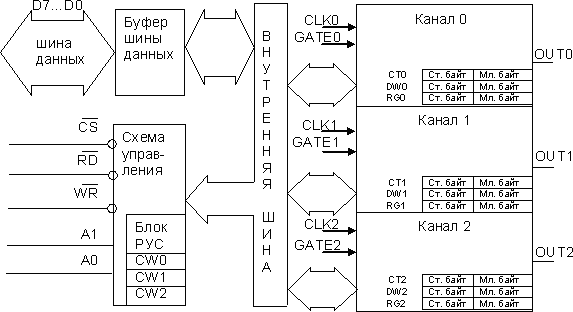
Дозволяється об’єднувати функціональні частини в функціональні групи, які виділяються на схемі штрих-пунктирними лініями. Кожній виділеній групі присвоюють назву або умовне позначення.

На схемі повинно бути вказано:

* Для кожної функціональної частини, зображеної прямокутником, її назва або умовне позначення, вписане у прямокутник.
* Для кожної функціональної частини або елемента, зображеного умовним графічним позначенням позиційне позначення.



3.2 Спрощена функціональна схема канала таймера в режимі 0



3.3 Загальна функціональна схема програмованого таймера

Таймер пов'язаний з мікроЕОМ виходами D7 ... D0 двобічної шини даних, а також входами:

http://drive.ispu.ru/elib/pikunov/5_files/image015.gif- «Вибір кристала»;

http://drive.ispu.ru/elib/pikunov/5_files/image016.gif- «Читання»;

http://drive.ispu.ru/elib/pikunov/5_files/image017.gif- «Запис»;

А1, А0 - двухразрядний адреса внутрішнього порту.

**3.3 Діаграми процесів**

Діаграма процесів це модель проектування, графічне представлення «потоків» даних в інформативній системі. Діаграма потоків даних також може використовуватись для візуалізації процесів обробки даних (структурне проектування).

Для розробника вважається звичним спочатку креслити діаграму процесів рівня контексту, завдяки чому буде показано взаємодію системи із зовнішніми модулями. Ця діаграма в подальшому підлягає уточненню шляхом деталізації процесів та потоків даних за метою показати розлого розроблювану систему.

Діаграми потоків даних містять чотири типи графічних елементів:

* Процеси – являють собою трансформацію даних в рамках описуваної системи;
* Сховища даних (репозиторії);
* Зовнішні по відношенню до системи сутності;
* Потоки даних між елементами трьох попередніх типів.



**4 ЕКСПЕРЕМЕНТАЛЬНЕ ПІДТВЕРДЖЕННЯ ВІРНОСТІ ПРОГРАМНИХ ТА ПРОЕКТНИХ РІШЕНЬ**

**4.1 Програмування таймера**

Програмування таймера досить гнучке, не має значення послiдовнiсть завантаження керуючих слів режиму окремих лiчильникiв, тобто не обов'язково повинно бути першим керуюче слово лiчильника 0, останнiм-2. Регiстр керуючого слова режиму кожного лiчильника має свою адресу i може завантажуватись незалежно.Однак завантаження змiсту лiчильника повинно вiдповiдати послiдовностi, запрограмованiй в керуючому словi режиму (-старший байт, молодший байт ).

При програмуванні таймера програміст вирішує наступні задачі:

1. Ініціалізація каналів інтервального таймера.

Здійснюється за допомогою

* + Запису керуючого слова CWR,
  + Запису константи перерахунку.

2.-Читання регістру стану інтервального таймера, або константи перерахунку RBC ( ReadBackCommand) здійснюється за допомогою

* запису керуючого байту RBC,
* читання регістру стану ініціалізації таймеру RST,
* читання константи перерахунку

а) з зупинкою таймера

б) без зупинки таймера.

Керуюче слово задає один з шести режимів роботи, тип ліку (двійковий чи двійково-десятковий), порядок завантаження і розмірність (один чи два байти) константи. Воно завантажується в регістри RSW каналів таймера, зберігається до слідуючого перепрограмування. Після цього завантажуються константи, які приймають значення:

* двійковий – 0-65536
* двійково-десятковий – 0-9999

Константи перерахунку завантажуються у таймер при А0, А1 ( 00, 01, 10 ).

Існує 2 варіанти завантаження: спочатку всі слова керування, а потім константи, чи слово управління і константу для кожного каналу послідовно.

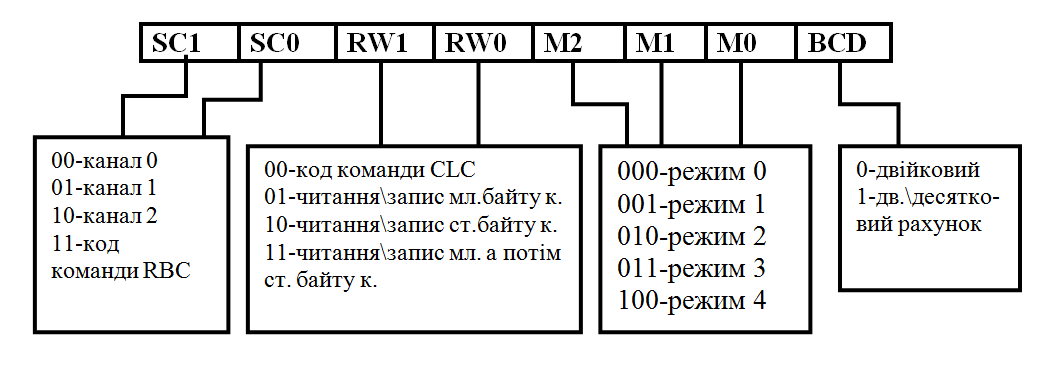


Рисунок 4.1 Формат керуючого слова

**4.2 Читання змісту лічильника**

Якщо потрiбно знати значення лiчби у процесi роботи (особливо це використовується при визначеннi кiлькостi подiй), то можливо використати два засоби читання, якi не пошкоджують iнформацiю:

1. Використовуючи команду вводу iз обраного лiчильника (READ), керуючи входами А0-А1 програмiст може вибирати лiчильник значення якого потрiбно знати.

При читанні значення лiчильника треба заборонити роботу лiчильника за допомогою команди GATE або шляхом заборони тактових iмпульсiв. При читаннi записiв лiчильника першим читається молодший байт LSB, другим- старший байт MSB. Слiд пам'ятати, що процедура читання повинна бути обов'язково доведена до кiнця. Якщо запрограмовано читати два байти, то вони повиннi бути прочитанi до слiдуючої команди завантаження.

2)Другий засiб - читання пiд час лiчби по коман іCLC (на льоту), без зупинки. Для реалiзацii цього засобу таймер має додатковий регiстр,звернення до якого здiйснюється за допомогою команди вивiд. При цьому для читання змiсту лiчильника "на льоту" в регiстр керуючого слова режиму завантажується спецiальний код, пiд дiєю якого змiст лiчильника без впливу на його роботу фiксується у додатковому регiстрi. Пiсля цього використовується звичайна операцiя ВВОД для читання змiсту регiстра.

Таймеру відповідають чотири порти введення/виводу з наступними адресами наприклад:

40h - канал 0;

41h - канал 1;

42h - канал 2;

43h - керуючий регістр.



Риунок 4.2 - Формат команди СLC

***Програмування таймера*** досить гнучке, і не має значення в якій послідовності, який канал програмувати. Операції програмування зводяться до виконання слідуючої послідовності команд:

1) запис у регістр керуючого слова для керування каналом (вибір формату двійкового або двійково-десяткового,

-вибір режиму;

-порядок завантаження const перерахунку;

-вибір каналу;

2) запис const перерахунку;

Наприклад:

MVI A, 00110000B ;формування керуючого слова лічильника 0

OUT PORT\_RUS ;запис 2-х байт конст., режим 0,

рахунок двійковий-десятковий

MVI A, DATA\_LOW ;завантаження молодшого байта кон. У лічильник 0

OUT PORT\_0 ; при А0,А1=00

MVI A, DATA\_HIGH ;завантаження старшого байта кон. У лічильник 0

OUT PORT\_0 ; при А0, А1=00

MVI A, 00000000B ;формування керуючого слова для читання

OUT PORT\_RUS ; на льоту слова стану і завантаження в таймер

IN PORT\_0 ;читання молодшого байта лічильника 0

IN PORT\_0 ;читання старшого байта лічильника 0

**Програмування таймера I8253\I8254 (КР 580ВИ53)**

**ASM-80**

Org 800h

Ldarej0 ; перший заданий режим

Out 43h ; до RUS

Lxih,c0 ; адреса константи

Mova,m ; зчитуванняконстанти

Out 40h ; до каналу 0

Inxh ;

Mova,m ; зчитування старшого байту конс.

Out 40h ; до каналу 0

##### Читання слова стану

Mvia,xx00xxxxb ; управляюче слово для читання слова стану

; на льоту (D7, D6-номер каналу якого треба

; прочитати слово стану, D5,D4-прикмета

; керуючого слова

out 43h

inxxh ; де хх –номер каналу ,який ми читаємо (мол.б)

inxxh ; старший байт константи

де у словістані - D0-тип рахунку, D1; D2; D3-код режиму, D4, D5- код послідовностізавантаженняконстанти , D6- даєінформаціючизавантажена константа , D7- стан виходу

hlt

Rej0: db 00110000b ;управляючі слова заданихрежимів реж.0,кан.0

Rej1: db 00110010b ; реж.1,кан.0

Rej2: db 01110100b ; реж.2, кан 1

Rej3: db 01110110b ; реж.3, кан. 1

Rej4: db 10111000b ; реж.4, кан. 2

Rej5: db 10111010b ; реж.4. кан.2

C0: dWxxh ; константа

end

**ASM-86**

Masm

Model small

. stack 256h

. data

Rej0: db 00110000b ;управляючісловазаданихрежимівреж.0,кан.0

Rej1: db 00110010b ; реж.1,кан.0

Rej2: db 01110100b ; реж.2, кан 1

Rej3: db 01110110b ; реж.3, кан. 1

Rej4: db 10111000b ; реж.4, кан. 2

Rej5: db 10111010b ; реж.4. кан.2

Cons :dwxxh ; константа

.code

main:

movax, @data ; налаштування на сегмент даних

mov ds, ax

movcx, 6

moval, rej0 ; канал 0, режим 0

out 43h, al

mov ax, cons

out 40h, al

out 40h, ah

moval, rej1 ; канал 0, режим 1

out 43h, al

mov ax, cons

out 40h, al

out 40h, ah

moval, rej2 ; канал 1, режим 2

out 43h, al

mov ax, cons

out 41h, al

out 41h, ah

moval, rej3 ; канал 1, режим 3

out 43h, al

mov ax, cons

out 41h, al

out 41h, ah

moval, rej4 ; канал 2, режим 4

out 43h, al

mov ax, cons

out 42h, al

out 42h, ah

moval, rej5 ; канал 2, режим 5

out 43h, al

mov ax, cons

out 42h, al

out 42h, ah

exst:

mov ax, 4c00h

int 21h

end main

**Ci**

#include <stdio.h>підключення бібліотеки

#include <dos.h>підключення бібліотеки

void main () головна підпрограма

{

unsignedc0=0xXXXX; // константа

unsignedcharrej0=0x30; // слово керування

// запис керуючого слова

outportb (0x43,rej0);

// запис константи

outportb (0x40,c0&ff); // запис молодшого байту константи

outportb (0x40,c0>>8); // запис старшого байту константи

**5 ІНСТРУКЦІЯ КОРИСТУВАЧА**

**6 ВИСНОВКИ**

**СПИСОК ЛІТЕРАТУРИ**

1.Новаченко І. В., Телець В.А .Мікросхеми для побутової радіоапаратури. Доповнення друге. Довідник.- М: Радіо і зв'язок, 1991-272 с. : Ил

2. Хвощ С. Т., Варлінскій Н. Н., Попов Е.О. Мікропроцесори і мікроЕОМ в системах автматіческого управління. Л .: Машинобудування, 1987. - 640 с .: ил.

3.Трейстер Р. Радіоаматорські схеми на ИС типу 555: Пер. з англ. -М .: Світ, 1988. -263 с. , Іл.

4. Шило. Л. В. Популярні цифрові мікросхеми .: Довідник, - М.: радіо і зв'язок, 1989. - 352 с.

5. Волков С. Генератори прямокутних імпульсів на МОП-елементах .: М .: Енергоіздат, 1986, 230 с.

6. Коломбет Е. А. Таймери. М .: Радіо і зв'язок, 1984. 126 с.

7. Бірюков С. А. Цифрові пристрої на МОП інтегральних мікросхемах. - М.: Радіо і зв'язок, 1990. - 128 с.