Практичне заняття 1.
Аналіз та синтез структур розподілених мікропроцесорних систем

1. Структура МПП у складі розподіленої мікропроцесорної системи.
2. Приклади побудови розподілених мікропроцесорних систем.
3. АСУ ТП розподілу, контролю якості та обліку електроенергії (з люб’язного дозволу ТОВ «ЕСП «Преобразователь»).
4. Задачі для розв’язання за темою.

1. Структура МПП у складі розподіленої мікропроцесорної системи

Мікропроцесорний пристрій (МПП) є **функціонально і конструктивно закінчений виріб**, що складається з декількох мікросхем, до складу яких входить **мікропроцесор МП (мікроконтролер МК)**; воно призначене до виконання певного набору функцій: **отримання, обробка, передача, перетворення інформації та управління**.

Стосовно РМПС МПП обов'язково містить у своєму складі або засоби передачі даних по каналах зв'язку, або інтерфейси для підключення таких засобів (наприклад, модемів, оптоволоконних ліній, супутникових терміналів та ін.).

Як правило, МПП у складі РМПС **має зберігати обмежену функціональність за відсутності з'єднань каналами зв'язку**.

Практично в будь-якій РМПС є періоди часу, коли зв'язок з центром та іншими МПП на деякий час переривається з тих чи інших причин.

З метою виконання вимоги доставки всіх введених у МПП даних без втрат зазвичай застосовується досить простий алгоритм: у відсутність зв'язку по всіх доступних каналах МПП запам'ятовує сформовані повідомлення і зберігає їх до появи зв'язку, а потім передає їх за призначенням.

Зазначимо, що цей алгоритм накладає на протокол передачі повідомлень **вимогу про наявність у кожному повідомленні часу формування**. Інакше центру неможливо буде дізнатися якому моменту часу яке повідомлення відповідає. Крім цього, очевидно, що всі МПУ, що входять до складу РМПС, мають бути синхронізовані за часом.

1. Структура МПП у складі розподіленої мікропроцесорної системи

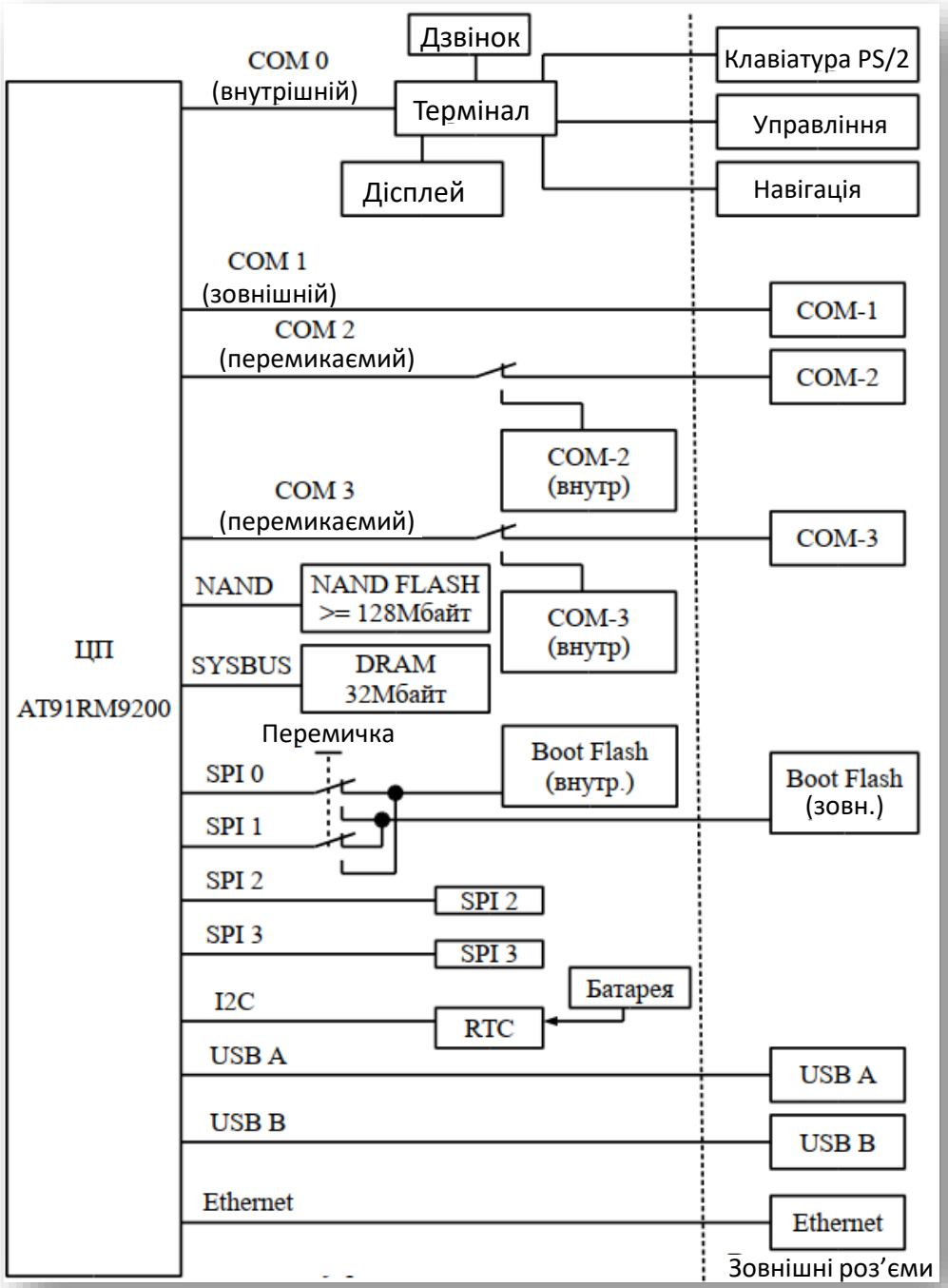
Загальні вимоги до МПП у складі РМПС:

- ✓ МПП має містити у своєму складі або **засоби передачі даних по каналах зв'язку**, **або інтерфейси** для підключення таких засобів;
- ✓ МПП **має забезпечувати обмежену функціональність без зв'язку** з іншими МПУ;
- ✓ всі МПП у складі РМПС **повинні мати годинник, синхронізований за часом**. При цьому годинник повинен коректно відраховувати час і за відсутності зв'язку з «центром» та іншими МПП даної системи;
- ✓ **інші вимоги до МПП, що накладаються завданнями,** що вирішуються.

Основні вимоги накладає цільове завдання (ЦЗ). Але, крім цього, можливі й інші вимоги до МПП, які не пов'язані з ЦЗ, але значно підвищують функціональні можливості МПП – наприклад, комфорт оператора під час роботи з МПП, зручність діагностування неполадок МПУ тощо.

Як приклад МПП, використовуваного у складі РМПС, розглянемо функціональний мікропроцесорний термінал. Мікропроцесорний термінал (рисунок 1) являє собою універсальний пристрій збору, обробки та передачі-прийому інформації.

Основою мікропроцесорного терміналу є МК (центральний процесор) AT91RM9200. Цей процесор складається з ядра ARM9 та вбудованих периферійних пристроїв.



1. Структура МПП у складі розподіленої мікропроцесорної системи

Пам'ять складається з трьох різних за функціональністю запам'ятовуючих пристроїв:

1) динамічного ОЗУ об'ємом 32 Мбайт, підключеного до системної шини процесора, призначеного для зберігання програмних компонентів та даних під час їх виконання;

2) NAND-FLASH ПЗУ об'ємом не менше 128 Мбайт, що використовується як «твердий диск» для зберігання програм та іншої інформації у вигляді файлів;

3) завантажувальної FLASH-ПЗУ (Boot Flash) об'ємом 8 Мбайт, призначеного для зберігання початкового завантажувача U-Boot та ядра ОС Linux.

Рисунок 1 – Приклад структури МПП у складі РМПС

1. Структура МПП у складі розподіленої мікропроцесорної системи

Інтерфейси USB A і B призначені для підключення різних зовнішніх периферійних пристроїв USB-Flash, USB-аудіокарт (гарнітур) і т.д.

Інтерфейс Ethernet призначений для підключення мікрпроцесорного терміналу до локальної мережі.

Інтерфейси RS-232 (COM-порти, послідовні порти). МПП має 4 вбудовані інтерфейси RS-232. Порт COM-0 є вбудованим і використовується для зв'язку центрального процесора з процесором вбудованого терміналу введення-виводу.

Порт COM-1 є зовнішнім. У робочому режимі цей порт використовується для підключення різних периферійних пристроїв з інтерфейсом RS-232. У режимі налагодження даний порт може використовуватися як системний термінал.

Порт COM-2 перемикається. Тобто роз'єм даного порту знаходиться всередині корпусу, але за допомогою перехідника може підключатися до роз'єму COM-2 на задній панелі МПП. Залежно від конкретного призначення терміналу порт використовується для підключення різних зовнішніх периферійних пристроїв з інтерфейсом RS-232, або для периферійних пристроїв, вбудованих в корпус мікрпроцесорного терміналу.

Порт COM-3 переключається і повністю аналогічний до функцій порту COM-2. Особливістю порту COM-3 і те, що він також служить для підключення вбудованого радиомодема.

2. Приклади побудови розподілених мікропроцесорних систем

Вбудований термінал введення-виведення є закінченим МПП, включеним до складу плати мікропроцесорного термігалу. Основою терміналу введення-виведення МПП є МК AT91SAM7S256, що базується на ядрі ARM7. Зв'язок між процесором терміналу введення-виведення та центральним процесором здійснюється за допомогою порту COM-0 центрального процесора.

ПЗ терміналу працює під керуванням ОС Linux. **Застосування ОС загального призначення дозволяє використовувати широкий набір програм, доступних у вигляді вихідного коду.**

Як приклади розглянемо побудову розподілених мікропроцесорних систем на основі мікропроцесорних терміналів на рисунку 1.

РМПС, розглянуті нижче, побудовані на базі трьох різних МПП – персонального комп'ютера (ПК), мікропроцесорного терміналу та плати розширення STK600 на базі мікроконтролера AVR ATMega2560.

При розширенні РМПС можуть бути доповнені необхідним обладнанням, наприклад, USB-аудіокартами, USB-відеокартами, датчиками тиску, температури, руху, зчитувачами ідентифікаційних карт тощо.

Програмування різних МПУ на різних рівнях абстракції: для ПК та мікрпроцесорних терміналів – програми виконуються під керуванням ОС Linux; для плати розширення STK600, яка не має ОС, програми пишуться без використання ОС.

2. Приклади побудови розподілених мікропроцесорних систем

Система збору даних (СЗД) забезпечує зчитування декількох датчиків та передачу інформації до центру збору даних.

Основне завдання СЗД – збір даних з МПП-джерел даних та передача зібраних даних до центру збору даних (ЦЗД). МПУ отримує дані від датчиків автоматично або дані вводяться оператором. Структурна схема СЗД представлена на рисунку 2.

Як центр системи збору даних використовується ПК під керуванням ОС Linux. Як МПП-джерела даних – мікропроцесорні термінали з підключеними до них модулями датчиків на базі МК AVR ATMega2560.

Дані передаються через локальна мережу, але може бути застосована мережа Internet.

Режим роботи та показання модулыв датчиків задаються за допомогою режимних кнопок, підключених до мікроконтролерів AVR ATMega2560.

Дані з модулів датчиків МД1-МДN передаються за інтерфейсом RS-232 на відповідні термінали МК1-МКN.

Мікропроцесорні термінали перетворюють дані від модулів датчиків у формати, необхідні для передачі їх у центр системи збору даних і передають через локальну мережу на персональний комп'ютер.

Управління мікропроцесорним терміналом здійснюється як за допомогою стандартної клавіатури PS/2, так і віддалено протоколом **ssh**, наприклад з ПК.

2. Приклади побудови розподілених мікропроцесорних систем

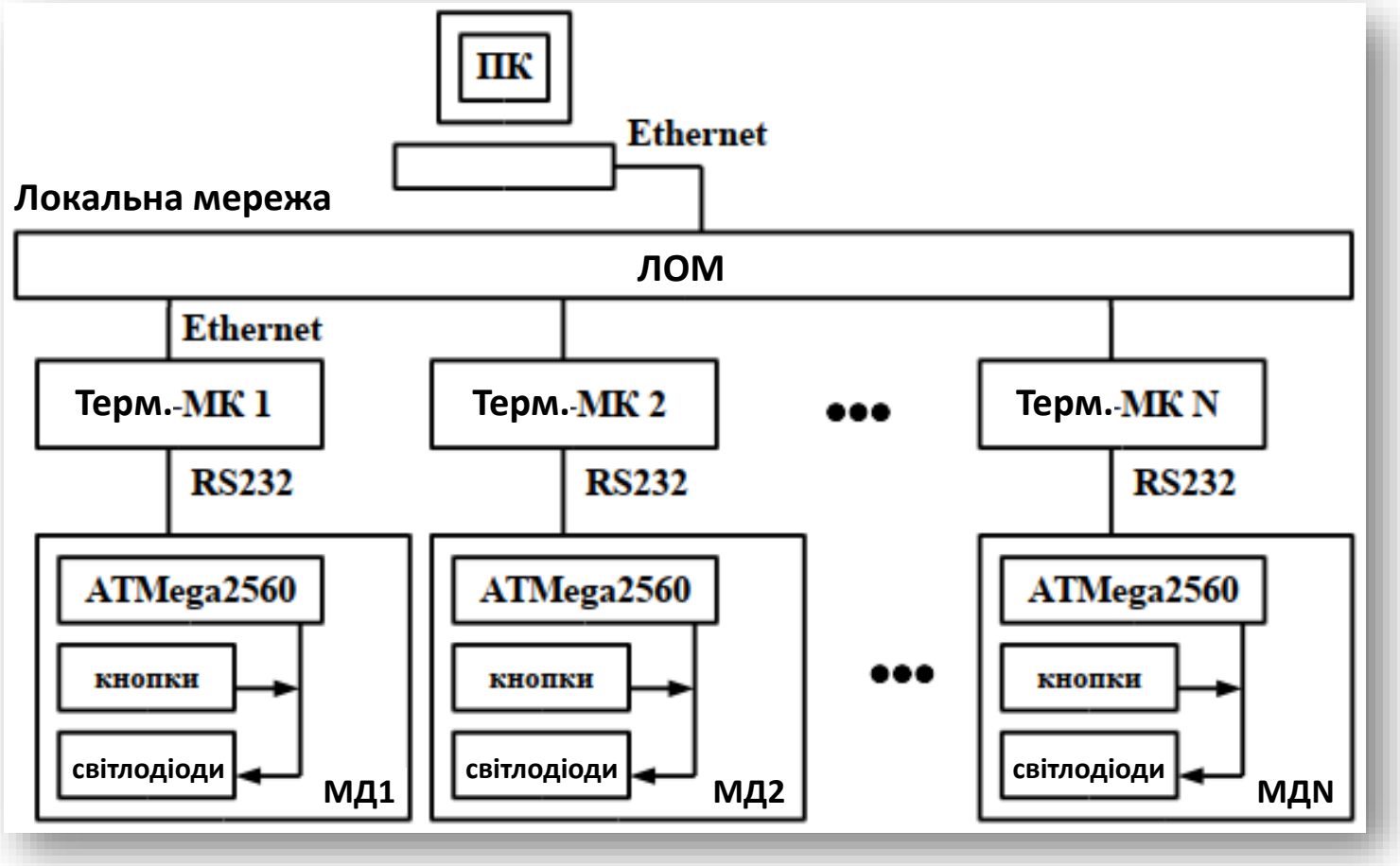


Рисунок 2 – Розподілена система збору даних

2. Приклади побудови розподілених мікропроцесорних систем

Процес формування даних та доставки їх у центр відбувається у два етапи:

1. Отримання МПП даних від датчика (оператора).
2. Передача даних від МПП до ЦЗД.

Алгоритми, застосовувані кожному етапі формування даних, і доставки їх у центр, залежить від конкретної завдання СЗД, наявних каналів зв'язку, протоколів та інших чинників.

Найпоширеніші алгоритми кожного етапу.

1. Отримання МПП даних від датчика (оператора):
   1) датчик віддає поточні дані на момент формування;
   2) датчик віддає поточні дані із заданими інтервалами часу;
   3) датчик віддає поточні дані на запит МПП.
2. Передача даних від МПП до ЦЗД:
   1) МПП передає до центру дані з власної ініціативи;
   2) МПП передає до центру дані на запит ЦЗД.

2. Приклади побудови розподілених мікропроцесорних систем

Система оповіщення об'єктів представлена на рисунку 3. Її основним завданням є доведення спеціальних повідомлень, які називають сигналами оповіщення, від центру до абонентів. У сигнал оповіщення можуть входити текстове повідомлення, звуковий сигнал (сирена), звукове (мовленнєве) повідомлення, світлова сигналізація.

Як центр системи оповіщення використовується ПК під керуванням ОС Linux. Як МПУ-абоненти – мікропроцесорні термінали з підключеними до них приладами світлозвукової сигналізації та модулів датчиків на базі МК AVR ATMega2560.

Мікропроцесорний термінал може бути доповнений USB-картою для відтворення звукових повідомлень, наприклад мовлення.

Дані між центром та абонентами передаються по локальній мережі, але може бути використана мережа Internet.

Крім основної функції – оповіщення об'єктів, система сповіщення, як правило, несе низку додаткових функцій, наприклад: телефонний зв'язок між об'єктом оповіщення та центром, збирання даних з будь-яких датчиків тощо.

Обмін даними між терміналами та приладами світлозвукової сигналізації та датчиків здійснюється за інтерфейсом RS-232.

2. Приклади побудови розподілених мікропроцесорних систем

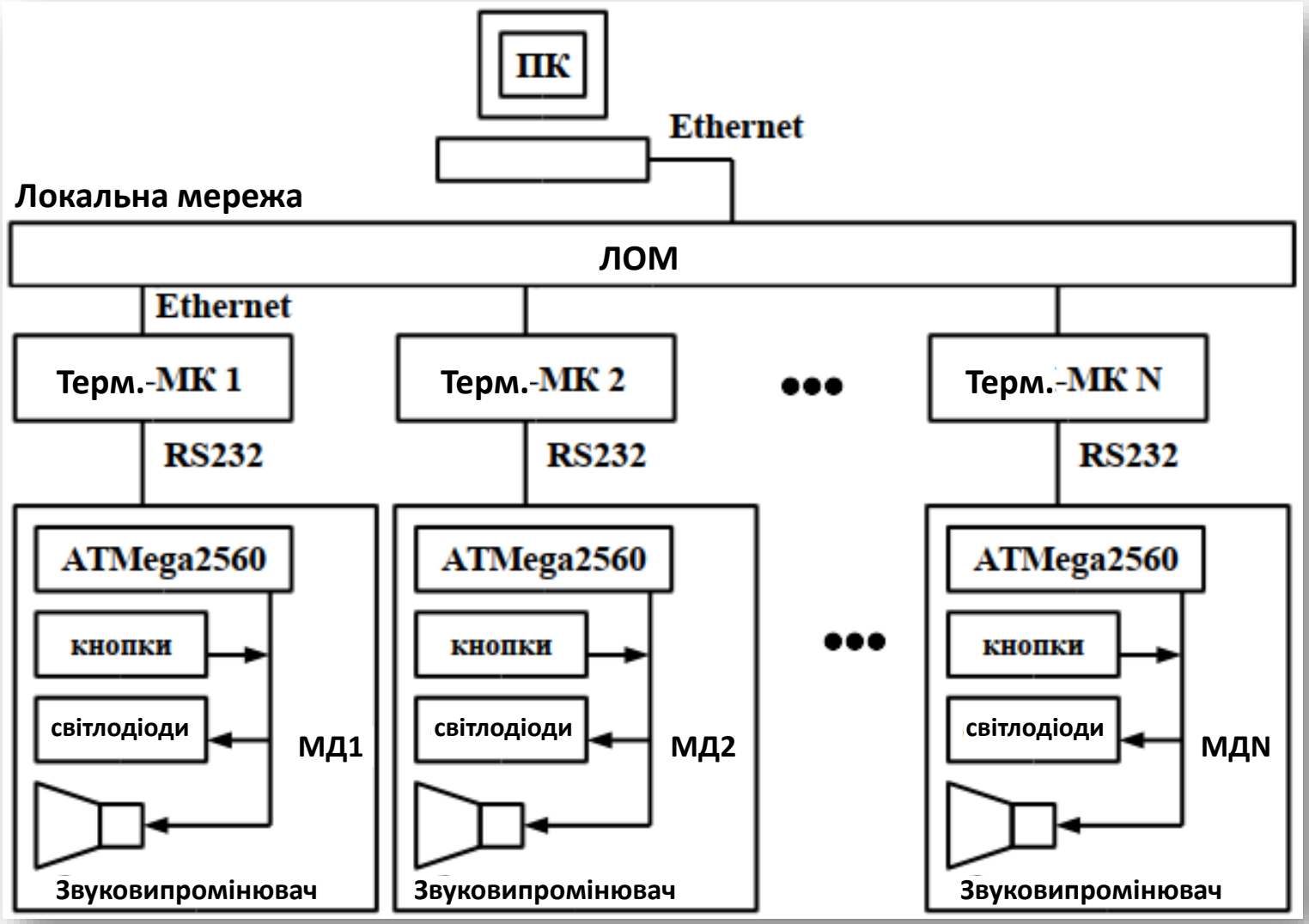
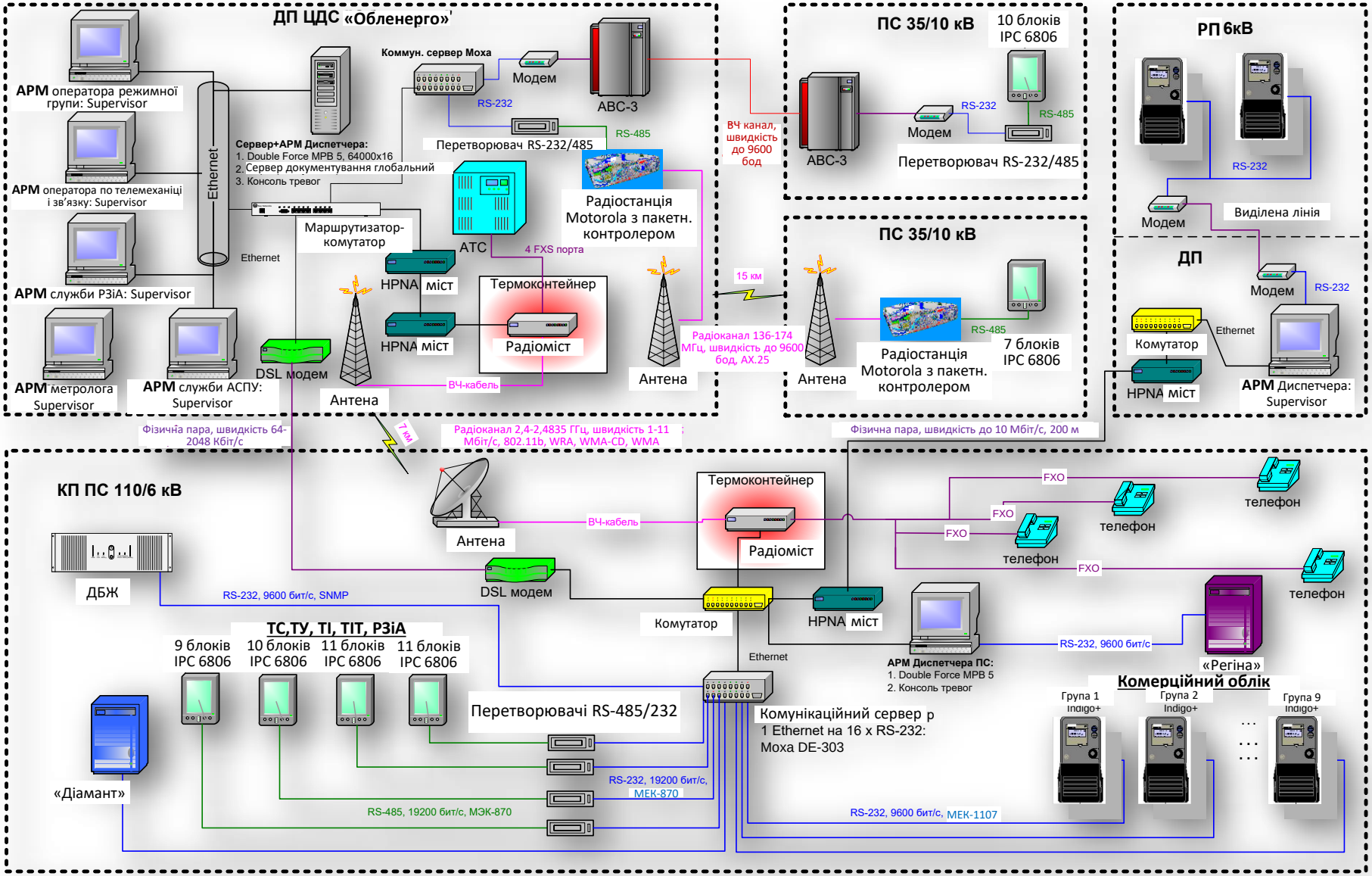


Рисунок 3 – Розподілена система оповіщення об’єктів

3. АСУ ТП розподілу, контролю якості та обліку електроенергії



4. Задачі для розв’язання за темою

Задача 1. За представленою структурою РМПС «АСУ ТП розподілу, контролю якості та обліку електроенергії» визначити:

1) цільове завдання РМПС, що розробляється;

2) основні завдання РМПС, які вирішуються;

3) основні технічні вимоги до РМПС;

4) кількість наявних рівнів ієрархії даної РМПС.

Надати перелік МПП (4-5), які входять до складу РМПС «АСУ ТП розподілу, контролю якості та обліку електроенергії», з формулюванням їх додаткових функцій, а також застосовані засоби комунікації і інтерфейси для наступних видів сполучення:

- ✓ між рівнями ієрархії;
- ✓ між МПП;
- ✓ між МПП і датчиками (абонентами).

Задача 2. Сформулювати мету, цільове завдання та основні задачі РМПС комерційного обліку енергоресурсів багатоквартирноного будинку. Розробити структуру даної РМПС комерційного обліку енергоресурсів багатоквартирноного будинку, визначити основні вимоги до основних блоків цієї РМПС та кількість рівнів ієрархії, надати перелік необхідних МПП у складі даної РМПС, засобів зв’язку між ними та типів інтерфейсу.

Практичне заняття 2.
Технологія розробки програмного забезпечення із застосуванням крос-засобів

1. Інтегроване середовище розробки та налагодження ПЗ для процесорів ARM Keil µVision IDE.
2. Основні етапи розробки та налагодження програмного забезпечення.
3. Вікно інтегрованого середовища µVision5.
4. Задачі для розв’язання за темою

1. Інтегроване середовище розробки та налагодження ПЗ для процесорів ARM Keil µVision IDE

Програмне забезпечення (ПЗ) мікропроцесорних систем (МПС) сьогодні розробляється **виключно на персональних комп'ютерах (ПК) з використанням крос-засобів**, тобто таких програмних продуктів, які встановлюються і працюють на універсальних комп'ютерах, створюючи виконавчі файли, призначені для завантаження і виконання в конкретному процесорі або мікроконтролері (МК) (цільовому пристрої), використовуваному розробником у своїй системі.

Сучасні крос-засоби є інтегрованими системами, які включають всі необхідні інструменти, об'єднані загальним графічним середовищем, що дозволяє практично повністю поєднати процес розробки ПЗ з процесом налагодження. При цьому користувачеві надаються кілька можливостей налагодження:

1) **у симуляторах** – програмно-логічних моделях цільового процесора, що працюють виключно на ПК і не потребують додаткової апаратури;

2) у створених виробниками МП та МК **оціночних платах**, підключених до ПК за універсальним інтерфейсом USB;

3) **у кінцевих пристроях**, апаратна частина яких створена розробниками конкретних додатків (контролерів) та мають необхідні інтерфейси для підключення відладчиків (зазвичай JTAG).

1. Інтегроване середовище розробки та налагодження ПЗ для процесорів ARM Keil µVision IDE

При цьому забезпечується **єдиний інтерфейс програми відладчика з будь-яким із трьох пристроїв**, в якому виконується програма, та **уніфікація функцій налагодження** – полегшується взаємодія спеціалістів-апаратників та програмістів при створенні будь-яких МПС.

Сучасні інтегровані середовища підтримують розробку як мінімум на Асемблері та одній з мов високого рівня (зазвичай C/C++).

З розвитком мікропроцесорної техніки відбулося збагачення мови Ассемблер засобами, які раніше застосовувалися виключно у мовах високого рівня.

Одне з найпоширеніших інтегрованих середовищ розробки та налагодження ПЗ для процесорів ARM (ARM Development Kit) Keil µVision IDE.

Причини вибору цього середовища наступні:

1) **Повний набір інструментів (спеціальних програм), інтегрованих у єдине графічне середовище** та рідкісних для створення та налагодження програмного забезпечення широкого класу мікропроцесорних пристроїв на базі процесорів фірми ARM, **у тому числі на базі процесорних ядер Cortex-M3/M4/M4F**.

2) **Наявність загальнодоступної версії (спрощеної)** з деякими обмеженнями функціональності (обсяг коду програми < 32 Кбайт), призначеної для роботи на будь-яких комп'ютерах під операційною системою Windows.

1. Інтегроване середовище розробки та налагодження ПЗ для процесорів ARM Keil µVision IDE

3) Ядро середовища містить **усі засоби розробки** (development tools) та **додаткові пакети програмного забезпечення** (software packs) у стандарті програмного інтерфейсу для МК ARM-Cortex (Cortex Microcontroller Software Interface standard – CMSIS), включаючи **проміжні бібліотеки для підтримки МК більшості фірм-виробників**, які можуть використовуватися як «цільові» пристрої (target devices), тобто пристрої, на яких планується прикладна розробка.

4) Фірма Keil давно спеціалізується на розробці ПЗ для процесорів ARM, а 2005 р. була куплена фірмою ARM і стала по суті її підрозділом. Це гарантія того, що в ПЗ фірми Keil **ретельно враховані всі апаратні та програмні можливості ARM-процесорів**.

5) Середовище містить **вбудований симулятор**, що максимально повно відображає особливості архітектури та системи команд процесорних ядер ARM, зручний як для новачків, так і для фахівців у галузі МПТ.

Для того, щоб завантажити відкрите середовище розробки через Інтернет, потрібно зайти на сайт фірми ARM. Фірма Keil є її підрозділом (http://www.keil.com/company/). Перейти до розділу продуктів MDK Microcontroller Development Kit (http://www2.keil.com/mdk5). Зареєструватись. Після реєстрації інсталяція продукту виконується автоматично.

1. Інтегроване середовище розробки та налагодження ПЗ для процесорів ARM Keil µVision IDE

Наприкінці інсталяції можуть встановлюватись бібліотеки для конкретного пристрою (мікроконтролера одного з відомих виробників). Ви можете вказати тип цього мікроконтролера або назву оцінної плати, яка використовується у вашій розробці. Їх можуть бути встановлені також приклади практичного використання.

Інтегроване середовище розробки Keil µVision призначене для створення та налагодження ПЗ мовою Асемблер фірми ARM, що має назву ARM ASM, а також мовами високого рівня C або C++, найбільш затребуваних прикладними програмістами.

Перевага використання мови Ассемблер фірми розробника процесорів очевидна: він максимально повно відбиває всі особливості архітектури процесорів ARM та їхнього набору команд.

Особливість будь-яких інтегрованих середовищ розробки – можливість створення програмного забезпечення з модулів, написаних як мовою високого рівня C/C++, і Асемблере. При цьому Асемблер використовується там, де потрібна гранична оптимізація коду за обсягом пам'яті та швидкодією.

Середовище µVision підтримує як написання коду програми двома мовами, а й налагодження програми у вихідних кодах (як у C/C++, і на Асемблері).

1. Інтегроване середовище розробки та налагодження ПЗ для процесорів ARM Keil µVision IDE

Інтегроване середовище розробки має у своєму складі **всі інструменти, необхідні для ефективної роботи програміста**:

1) **Текстовий редактор** з повними можливостями якісного введення та редагування тексту програми та додатковими можливостями колірного виділення синтаксичних конструкцій мов Асемблер та C/C++.

2) **Транслятор з мови Ассемблер**, що сприймає на вході файл на асемблері з розширенням «.s» або «.asm» і генерує вихідний файл в об'єктному коді з розширенням «.o» (або «.obj»), а також файл лістингу з розширенням «.lst», що містить результати трансляції, у тому числі, з повідомленнями про помилки.

3) **Компілятор з мови C/C++**, що сприймає на вході файл мовою C або C++ з розширенням «.c» або «.cpp» і генерує на виході файл в об'єктному коді «.o», що переміщується.

4) **Компонувальник або Лінкер** – програма, що об'єднує кілька вихідних файлів у об'єктному коді, що переміщується, в один файл у непереміщуваному об'єктному коді, в якому відносні адреси замінені абсолютними – всі взаємні посилання програмних модулів один на одного і на бібліотечні функції «дозволені» (у тому сенсі, що символічним іменам надано конкретні фізичні адреси).

1. Інтегроване середовище розробки та налагодження ПЗ для процесорів ARM Keil µVision IDE

5) **Бібліотекар** – програма, яка може об'єднати кілька відтрансльованих/ відкомпільованих та налагоджених програмних модулів у об'єктному, що переміщається, коді в один файл з розширенням «.lib» – спеціалізовану користувальницьку бібліотеку, для подальшого багаторазового використання, але вже без налагодження, як готового продукту.

6) **Симулятор** – програмно-логічна модель цільового процесора, реалізована на звичайному комп'ютері та повністю імітує архітектуру та систему команд цільового процесора. Симулятор не тільки імітує процесор, а й деякі з периферійних пристроїв – системну периферію. Він дозволяє налагоджувати програму виключно на комп'ютері, без необхідності підключення до нього цільової плати МПС, що розробляється.

7) **Завантажувач** – завантажує (розміщує) програму в абсолютному об'єктному коді на виконання або в пам'ять симулятора, або пам'ять реальної процесорної плати. Це може бути оціночна плата, створена виробником МК для оцінки можливості вирішення прикладного завдання на даному МК, або плата, розроблена самим користувачем, за умови, що в ній реалізовано один або кілька необхідних для налагодження інтерфейсів.

1. Інтегроване середовище розробки та налагодження ПЗ для процесорів ARM Keil µVision IDE

8) **Налагоджувач** – дозволяє виконувати код програми в режимі прогону, по кроках, з точками зупинки, при повному контролі поточного вмісту регістрів процесора та змінних у пам'яті. Дозволяє оцінити швидкодію програми в цілому та окремих її фрагментів (підпрограм).

9) **Утиліти перетворення формату** вихідного файлу в інший формат, наприклад, для завантаження коду в програматор зовнішніх постійних пристроїв (ПЗП) або зовнішньої флеш-пам'яті.

С/С++ найзручніша мова високого рівня, що створює максимально компактний і швидко виконуваний код, який може конкурувати з кодом, створеним на Асемблері. Він відрізняється універсальністю, зручністю доступу до пам'яті. та регістрів периферійних пристроїв. Саме тому він широко використовується при розробці ПЗ цифрових систем управління, що вбудовуються в обладнання.

2. Основні етапи розробки та налагодження програмного забезпечення

Для всіх інтегрованих середовищ розробки порядок створення та налагодження ПЗ приблизно однаковий і ілюструється на рисунку 1:

1) **Створюється новий проект**, для якого вибирається цільовий МП або МК, тобто кристал, на базі якого буде реалізована МПС користувача.

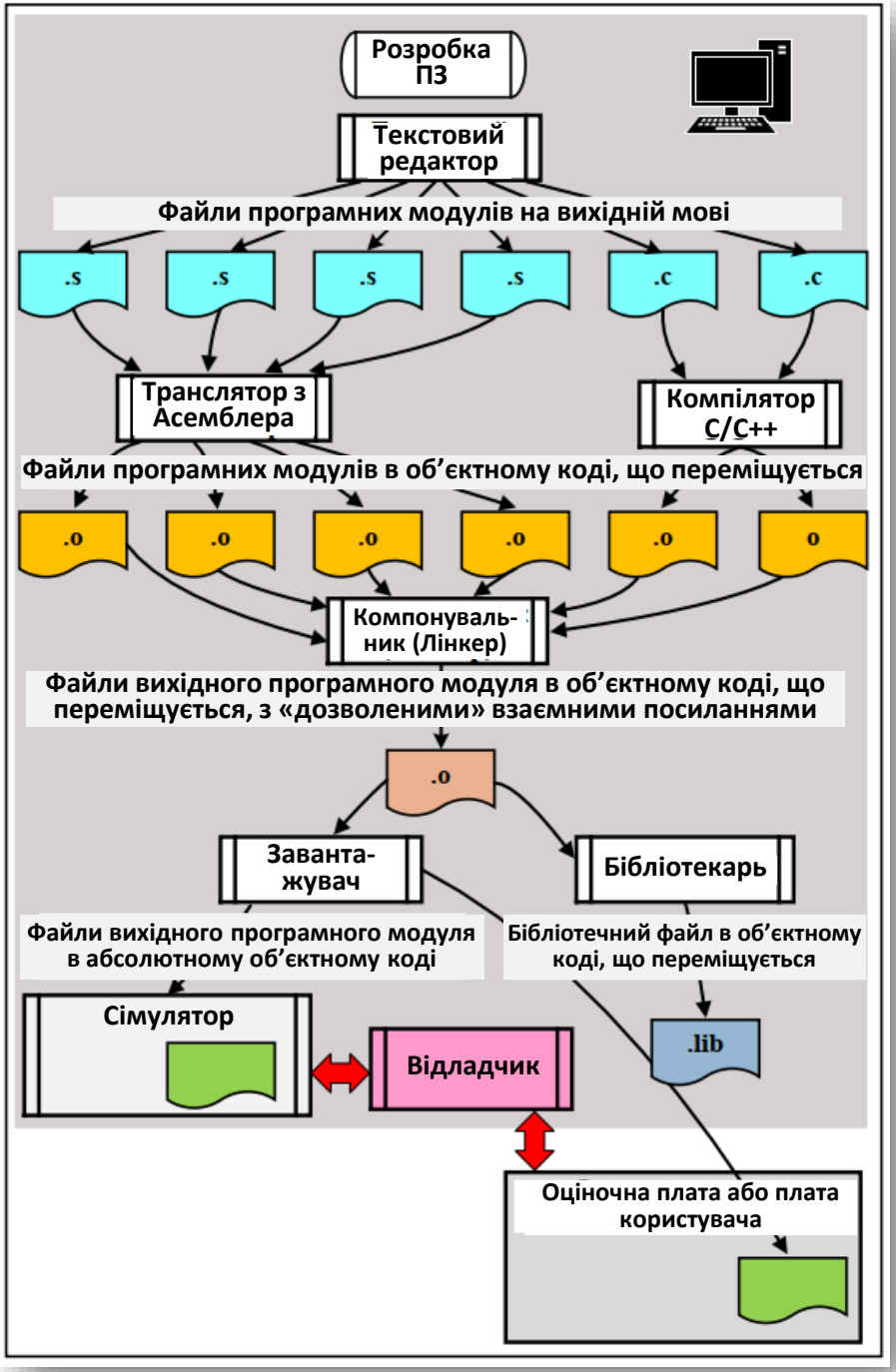
> *Файл проекту – це спеціальний файл, який містить інформацію про всі вихідні файли, написані користувачем/користувачами та підключені до проекту.*

Імена вихідних файлів та місця розташування запам'ятовуються у файлі проекту разом з додатковою інформацією про налаштування середовища для роботи з поточним проектом.

2) Для обраного цільового процесора необхідно **включити до проекту стартовий файл ініціалізації процесора** (фірма Keil люб'язно надає стартовий файл StartUp як допомогу розробникам, що роблять перші кроки).

> *Основне призначення цього файлу – ініціалізація процесора та вбудованих периферійних пристроїв.*

3) **Конфігурується інтегроване середовище** – для неї встановлюються необхідні параметри. Найчастіше для початку роботи достатньо «параметрів за замовчуванням».



2. Основні етапи розробки та налагодження програмного забезпечення

4) З використанням вбудованого текстового редактора пакета **створюються вихідні файли проекту або мовою Асемблер, або С/С++**.

Це може бути всього один файл, наприклад MyProg.s на Асемблері, або кілька, у тому числі мовою високого рівня С/С++. Як мінімум, до складу будь-якого проекту має входити стартовий файл StartUp.s та файл із програмою користувача.

Рисунок 1 – Порядок розробки ПЗ в інтегрованому середовищі

2. Основні етапи розробки та налагодження програмного забезпечення

5) **Кожен із вихідних файлів запускається на трансляцію**, якщо він написаний на Асемблері, або на компіляцію, якщо – на C/C++. Якщо у вихідних файлах будуть виявлені синтаксичні помилки, будуть надані відповідні сповіщення. Процес повторної трансляції або компіляції виконується для кожного окремого файлу, поки помилок у вихідних текстах програмних модулів не буде. В результаті створюється група файлів проекту в об'єктному коді, що переміщується.

6) **Виконується конфігурація пам'яті цільового пристрою**. Параметри інтегрованого середовища налаштовуються відповідно до реальних обсягів кодової пам'яті та пам'яті даних цільового мікропроцесора, щоб програма-компонувальник «знала», які ресурси вона має – куди можна розмістити код програми, куди – дані.

7) **Виконується компонування проекту**, тобто всі кодові секції вихідних файлів об'єднуються в одну секцію коду, яка розміщується за адресами доступної пам'яті програм. Усі ініціалізовані секції поєднуються в одну загальну секцію ініціалізованих даних, а секції неініціалізованих даних – в одну загальну секцію неініціалізованих даних. Вона розміщується по адресах доступної пам'яті даних.

2. Основні етапи розробки та налагодження програмного забезпечення

8) **Проект**, що дозволяє налагодження в симуляторі процесора, **завантажується в симулятор** і виконується під керуванням відладчика. У цьому випадку периферійні пристрої, які є у складі цільового МК, не повинні використовуватися. Так налагоджуються всі модулі, що виконують переважно обробку даних.

9) Повністю **налагоджені модулі** в об'єктному коді, що переміщується, **можуть бути об'єднані в користувальницьку бібліотеку** як ряд найбільш уживаних функцій. Надалі можна автоматично включати код будь-якої бібліотечної функції у вихідний файл проекту. Для цього достатньо лише підключити файл бібліотеки до проекту. Потрібна функція буде автоматично вилучена з бібліотеки компонувальником.

10) **Послідовна трансляція програмних модулів** дозволяє поступово створювати та налагоджувати проекти. Різні вихідні файли можуть створюватися та налагоджуватися різними програмістами. Тільки після повноцінного налагодження окремого програмного модуля програмістом він передається менеджеру всього проекту та підключається до проекту.

11) Тепер найвідповідальніший момент – **виконавчий код проекту може бути завантажений у реальну МПС** (оціночну плату чи плату розробленого контролера) та виконаний під керуванням відладчика.

2. Основні етапи розробки та налагодження програмного забезпечення

12) Будь-яка зупинка програми викличе зупинку в роботі та вбудованих у контролер периферійних пристроїв. Тому таке **налагодження** буде **повноцінним** налагодженням у реальному часі **тільки в режимі виконання програми повністю** (в режимі прогону).

13) Якщо в процесі налагодження програмного забезпечення у виробі виникли помилки, то виконується **процес редагування (модифікації)** окремих програмних модулів.

14) На останньому етапі **цільовий контролер вбудовується у виріб** (джерело живлення, перетворювач і т. д.) та його апаратна та програмна частини тестуються в умовах реальної експлуатації.

Подана на рисунку 1 схема розробки ПЗ може дещо відрізнятися для різних середовищ розробки. Так, компонувальник, а також програма-бібліотекар можуть отримувати на вході кілька файлів у об'єктному коді «.o», що переміщується, і генерувати в першому випадку відразу вихідний файл для завантаження в МК, а в другому випадку – вихідний бібліотечний файл.

3. Вікно інтегрованого середовища µVision5

Сучасне інтегроване середовище розробки програм IDE для МК, така як µVision, містить всі програмні модулі в одному пакеті, об'єднані зручним графічним середовищем, коли виклик потрібної функції виконується одним «клацанням» миші. Середовище містить і зручні засоби налаштування всіх програм (опцій), що входять до неї. Більш того, в одному середовищі можна розробляти ПЗ для МК різних виробників з різними периферійними пристроями, користуючись як Асемблером, так і мовою високого рівня С/С++, використовуючи функції з великої кількості спеціалізованих бібліотек, створених або розробниками процесорів, або самими розробниками або іншими користувачами, якщо вони зробили їх загальнодоступними.

Часто весь необхідний програміст комплекс програмних засобів (утиліт) називається одним словом toolchain (набір інструментів) або навіть compiler (компілятор) незважаючи на те, що він фактично складається з безлічі незалежних програм, об'єднаних графічним середовищем у єдину систему.

Загальний вигляд вікна інтегрованого середовища µVision5 представлений на рисунку 2.

Зауважимо, що можна задавати потрібні команди з розташованого у верхньому рядку меню, відкриваючи конкретні меню: **File** (Файл) – для роботи з файлами; **Edit** (Редактор) – для роботи з вихідними файлами у текстовому редакторі; **View** (Перегляд) – для виведення на екран вікна з потрібною в даний момент інформацією або для приховування цього вікна; **Project** (Проект) – до роботи з проектами;

3. Вікно інтегрованого середовища µVision5

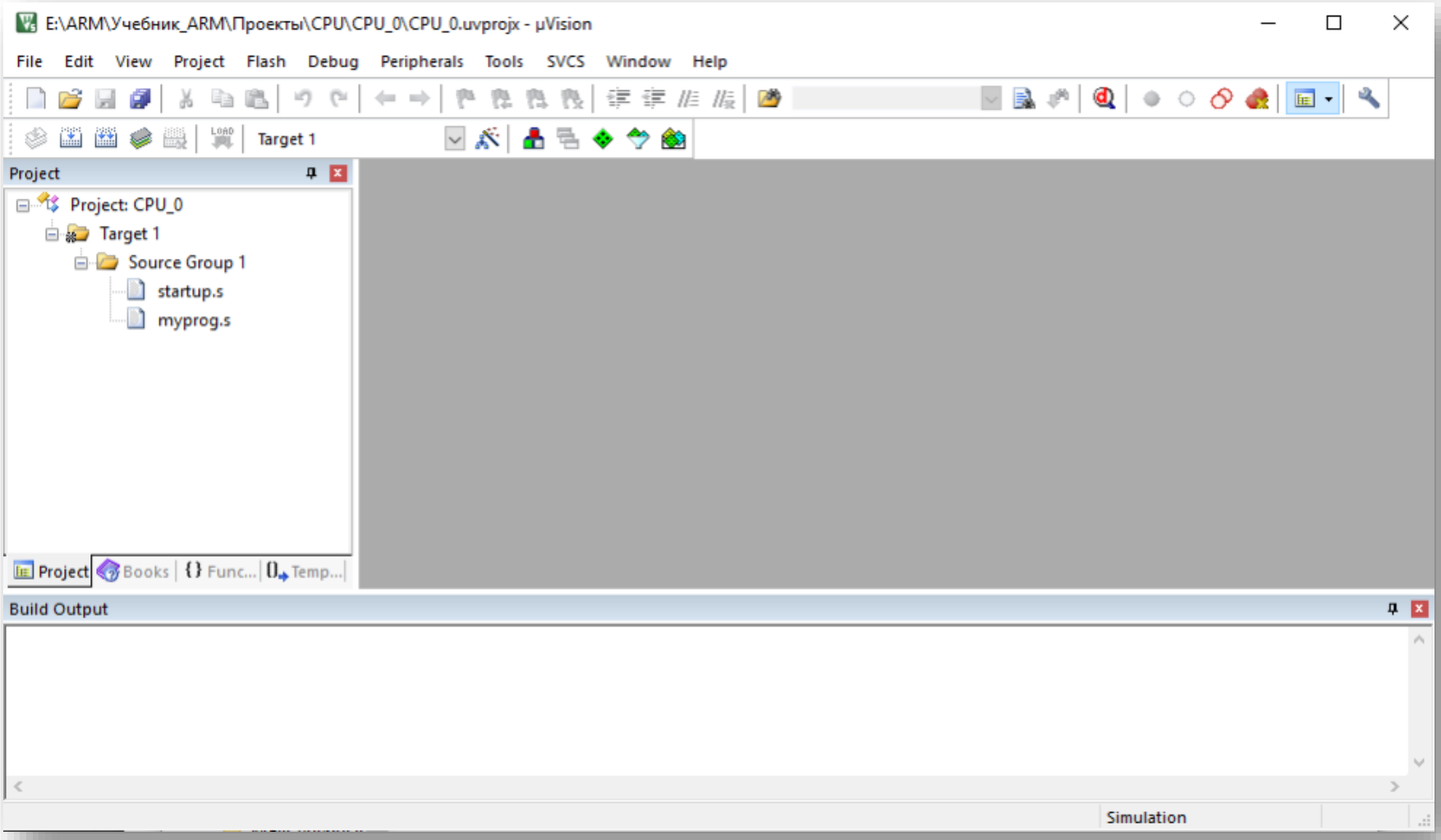


Рисунок 2 – Загальний вигляд вікна інтегрованого середовища µVision

3. Вікно інтегрованого середовища µVision5

Flash (Флеш) – для програмування флеш-пам'яті цільового пристрої; **Debug** (Налагоджувач) – для керування процесом налагодження програми, у тому числі в симуляторі; **Peripherals** (Периферія) – для роботи із вбудованою периферією; **Tools** (Кошти) – для роботи з додатковими інструментами пакета; **SVCS** (Software Version Control System) – доступ до системи контролю версій; **Window** (Вікна) – для управління вікнами середовища, їх переміщення, упорядкування та ін; **Help** (Допомога) – для отримання довідки щодо можливостей пакета, Асемблеру, системи команд та ін. При натисканні кнопкою миші відповідне меню розкривається, і з'являється можливість вибору потрібної команди.

Дві розташовані нижче панелі інструментів містять низку найбільш уживаних команд у вигляді піктограм. Так, у першій з них, напевно, знайомі піктограми «Створити файл», «Відкрити файл», «Зберегти», а також ряд піктограм, які є загальноприйнятими у всіх текстових редакторах «Вирізати», «Скопіювати», «Вставити» та ін. Деякі піктограми в цьому рядку широко використовуються при налагодженні, наприклад, для встановлення та зняття точок зупинки.

У другій панелі інструментів розташовані команди, які управляють процесом трансляції (компіляції) та складання файлів проекту.

На іншому полі можуть розташовуватися вікна з різною інформацією (програмами у вихідних кодах, вмістом необхідних при налагодженні областей пам'яті і і т.д.).

3. Вікно інтегрованого середовища µVision5

Поки можна побачити два з них – вікно проекту Project та вікно результату створення (побудови) вихідного файлу Build Output. В останнє вікно будуть виводитись результати трансляції файлів проекту та компонування. Тут же відображаються повідомлення про помилки та попередження.

У верхньому рядку завжди є ім'я файлу поточного проекту, яке має розширення «.uvprojx» із зазначенням повного шляху доступу до проекту. Цей файл має особливий формат. Він містить інформацію про всі програмні модулі проекту, цільовий процесор, опції (налаштування) середовища і так звані параметри оточення середовища. Це означає, що при повторному відкритті проекту для продовження роботи з ним Ви побачите на екрані ті самі вікна, які Ви відкривали раніше, в останньому сеансі роботи. Автоповернення до останньої конфігурації середовища, що використовується, істотно заощаджує час програміста на налаштування середовища при налагодженні програмного забезпечення.

У пакеті µVision використовуються загальноприйняті в комп'ютерах технології роботи з вікнами та командами, як у вигляді так званих меню, що випадають, так і у вигляді панелей інструментів з піктограмами найбільш важливих команд. Для виконання потрібної команди досить просто натиснути мишею на відповідній піктограмі. Команда буде негайно виконана.

4. Задачі для розв’язання за темою

Задача 1. Встановити Keil µVision IDE (ARM Development Kit) для ARM процесорів на персональний ком’пютер:

1) завантажити відкрите середовище розробки через Інтернет (сайт компанії ARM, її підрозділ фірма Keil, http://www.keil.com/company/);
2) перейти до розділу продуктів MDK Microcontroller Development Kit (http://www2.keil.com/mdk5);
3) зареєструватись.

Після реєстрації інсталяція продукту виконується автоматично.

Задача 2. Започаткувати новий проект в інтегральному середовищі Keil µVision IDE (ARM Development Kit) для ARM процесорів сімейства Cortex-M4 на персональний ком’пютер. Створити файл з програмою на Асемблер (рисунок 3) і подключити до започаткованого проекту.

Задача 3. Надати порівняльний аналіз інтегрованих середовищ Keil µVision IDE (version 5) і Microchip Studio (version 7) для створення ПЗ вбудованих електронних систем (технічні та комерційні вимоги для використання, функціональні можливості, повнота за зручність Toolchain, рівень підтримки існуючих сімейств МК та МП).

4. Задачі для розв’язання за темою

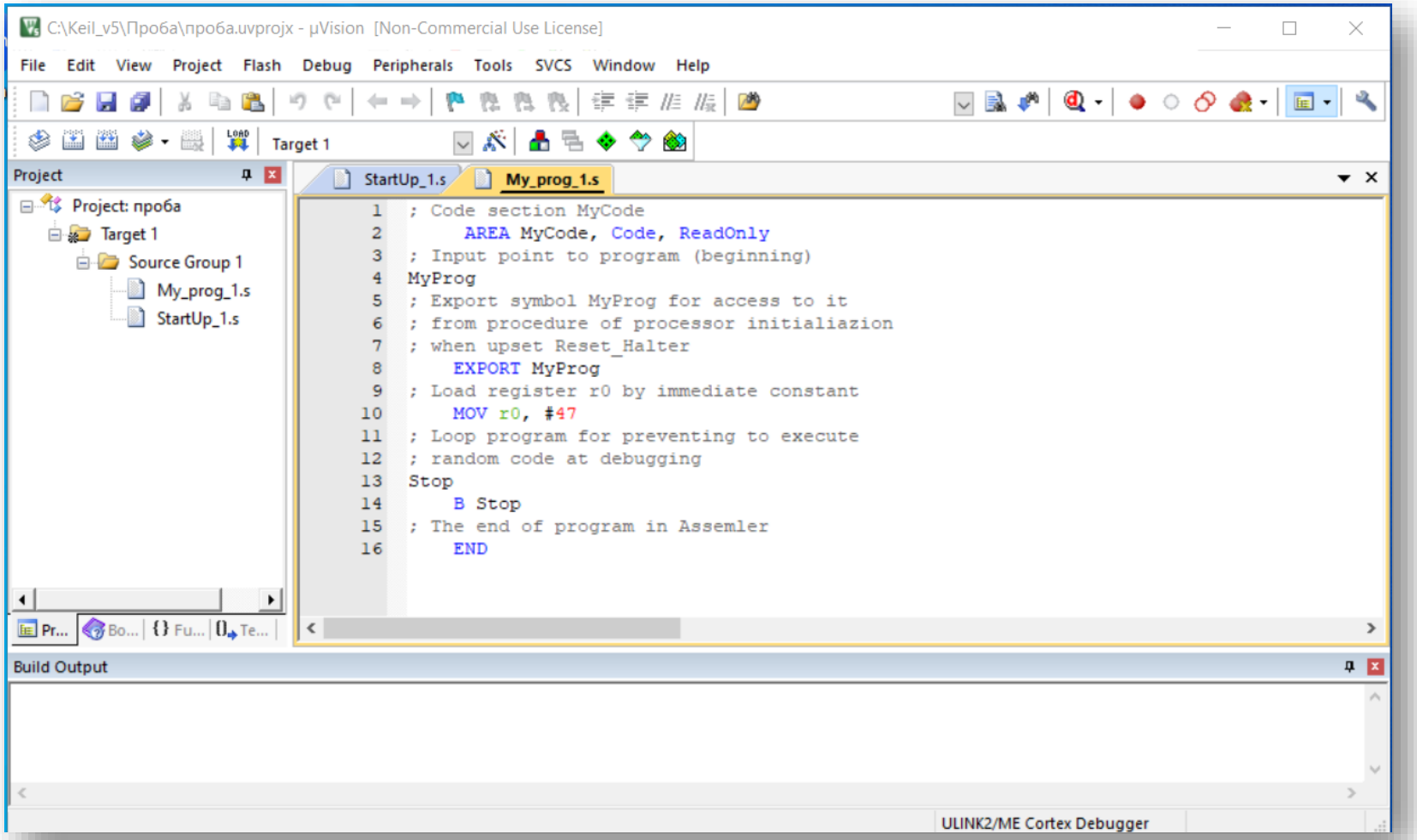


Рисунок 3 – Приклад програми на Асемблер до задачі 2

Практичне заняття 3.
Застосування УСАПП: мультипорт

1. Загальна характеристика УСАПП.
2. Підключення УСАПП до системної шини.
3. Організація обміну словами.
4. Схема мультипорту.
5. Задачі для розв’язання за темою.

1. Загальна характеристика УСАПП

Підсистема послідовного інтерфейсу – мультипорт з чотирьох послідовних портів введення/виведення 00h, 04h, 08h і 0Ch, налаштованих на режим передачі.

ІМС Intel 8251 виконана по nМОП-технології, живиться від джерела +5В і споживає струм 100 мА. Мікросхема являє **УСАПП послідовному зв'язку**, що виконує функції прийому й перетворення паралельних форматів слів даних у послідовні формати зі службовими символами для їхньої передачі по каналах зв'язку й послідовних форматів, прийнятих з каналів зв'язку слів даних, у паралельний формат для уведення в МП.

УСАПП може бути запрограмована на один з 5 режимів: **асинхронна передача, асинхронний прийом, синхронна передача, синхронний прийом із внутрішньою синхронізацією, синхронний прийом із зовнішньою синхронізацією**.

Швидкість обміну в синхронному режимі 0-64000 Кбіт/сек, в асинхронному режимі 0-19200 Кбіт/сек.

2. Підключення УСАПП до системної шини

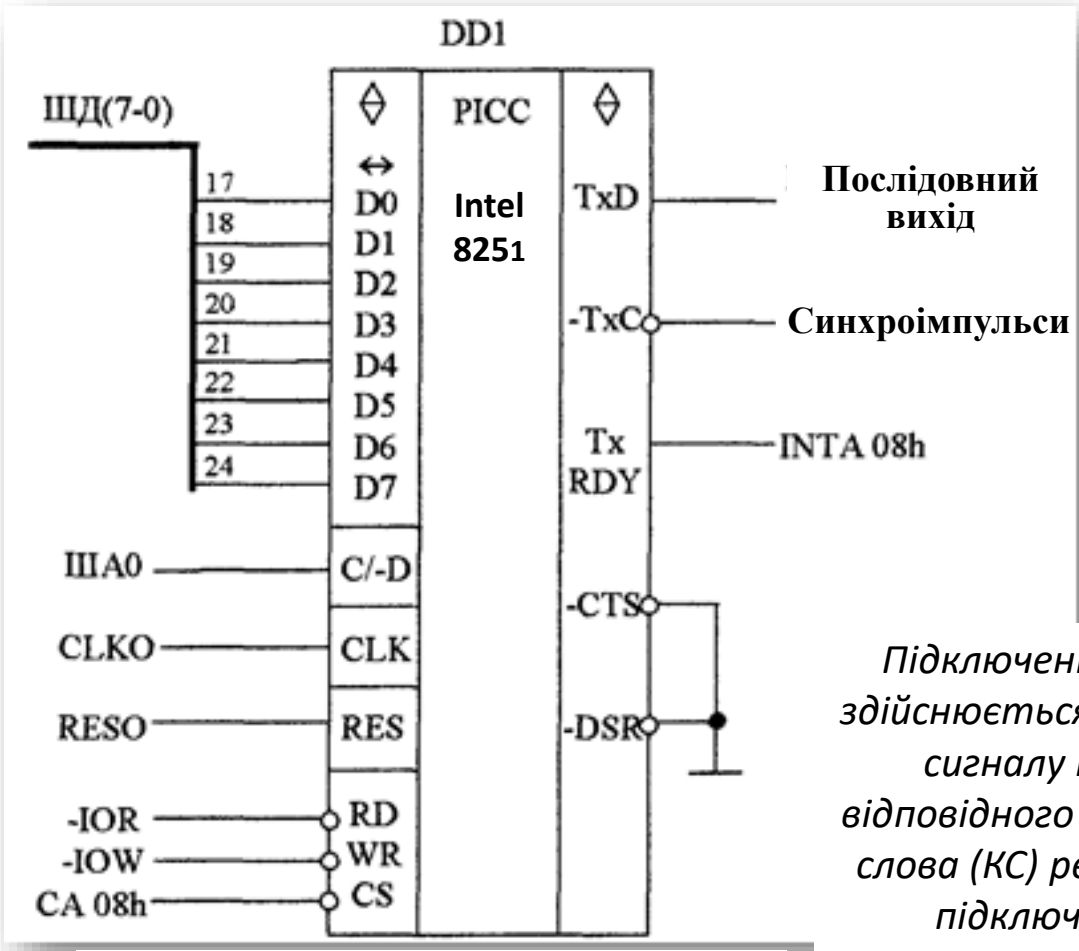


Рисунок 1 – Включення УСАПП в мультипорт в режимі асинхронної передачі

Включення УСАПП в мультипорт в режимі асинхронної передачі демонструється на рисунку 1. Зв'язок з системної ШД УСАПП виконує через двонаправлені виводи D0-D7 внутріш-нього буфера даних. Синхронізація роботи ВІС здійснюється зовнішніми синхроімпульсами, що надходять на вхід CLK з виходу CLKO МП. Для скидання УСАПП у початковий стан використовується сигнал МП RESO, що надходить на вхід RES.

Підключення ІС до системних шин мультипорта здійснюється селектором адреси подачею нульового сигналу на вхід -CS УСАПП при дешифруванні відповідного номера порту. Для виділення керуючего слова (КС) режиму й команди служить вхід C/-D, що підключається до ША0. Зв'язок з зовнішнім пристроєм здійснюється за допомогою виходів DD1 TxD і -TxC. На виході TxD формуються дані в послідовному форматі, а вхід -TxC синхронізує їхню видачу в лінію зв'язку. Сигнал переривання на виході TxRDY з'являється після завершення передачі чергового слова.

3. Організація обміну словами

При початковій ініціалізації мультипорту завдання асинхронного режиму передачі і його параметрів здійснюється КС режиму, що показаний на рисунку 2. Воно вводиться в УСАПП по команді OUT із зазначенням адреси послідовного порту і формуванням одиничного значення ША0. Відразу після уведення КС режиму в послідовний порт завантажується КС команди (рисунок 3). КС команди здійснює керування встановленим режимом обміну і може багаторазово задаватися в процесі обміну, управляючи різними його етапами. Вид КС команди при початковій ініціалізації УСАПП показаний на рисунку. Контроль стану УСАПП виконується шляхом зчитування слів стану ВІС, при цьому перевіряється розряд D0 слова стану: якщо D0 = 1, то є запит на обслуговування послідовного порту.

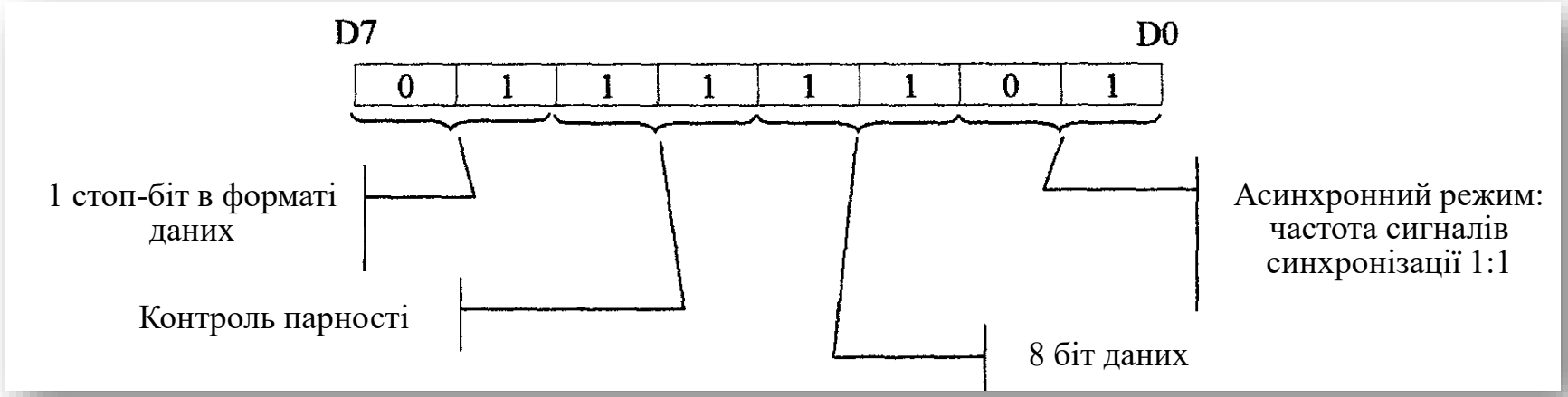


Рисунок 2 – Керуюче слово режиму асинхронного обміну для УСАПП

3. Організація обміну словами

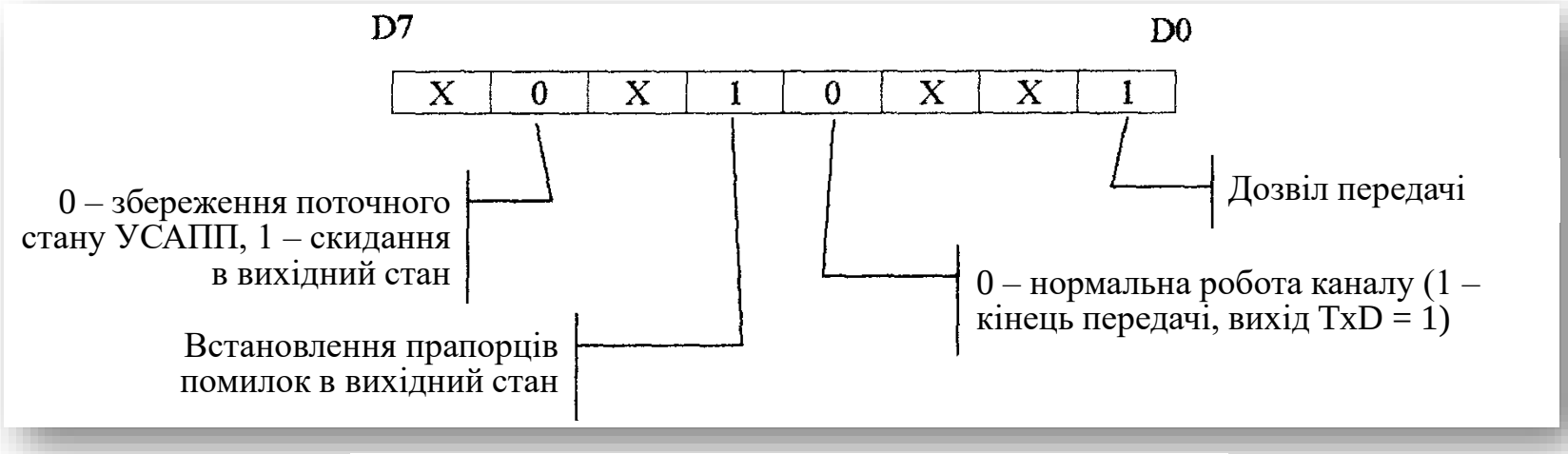


Рисунок 3 – Керуюче слово команди для УСАПП

У режимі асинхронної передачі після запису КС команди з встановленим D0 = 1 і -CTS = 0 на виході TxRDY з'являється напруга «логічної 1», сигналізуючи МП через вхід переривання про готовність передавача записати нові дані. Виявивши готовність передавача, МП записує слово даних у порт по сигналу -WR. При цьому на виході TxRDY установлюється нульове значення. Після запису даних у паралельному форматі в контролері відбувається автоматичне приєднання до кожної посилки старт-біта, контрольного біта і стоп-біта і їхня послідовна передача під керуванням сигналів на виході -TxC. Після передачі слова даних встановлюється одиничний стан сигналу готовності TxRDY, що з'являється з затримкою щодо середини стоп-бита на 16-20 періодів синхроімпульсів. Після запису КС команди виду 00001000 по закінченню посилки останнього слова даних передача припиняється і установлюється TxC = 1.

4. Схема мультипорту

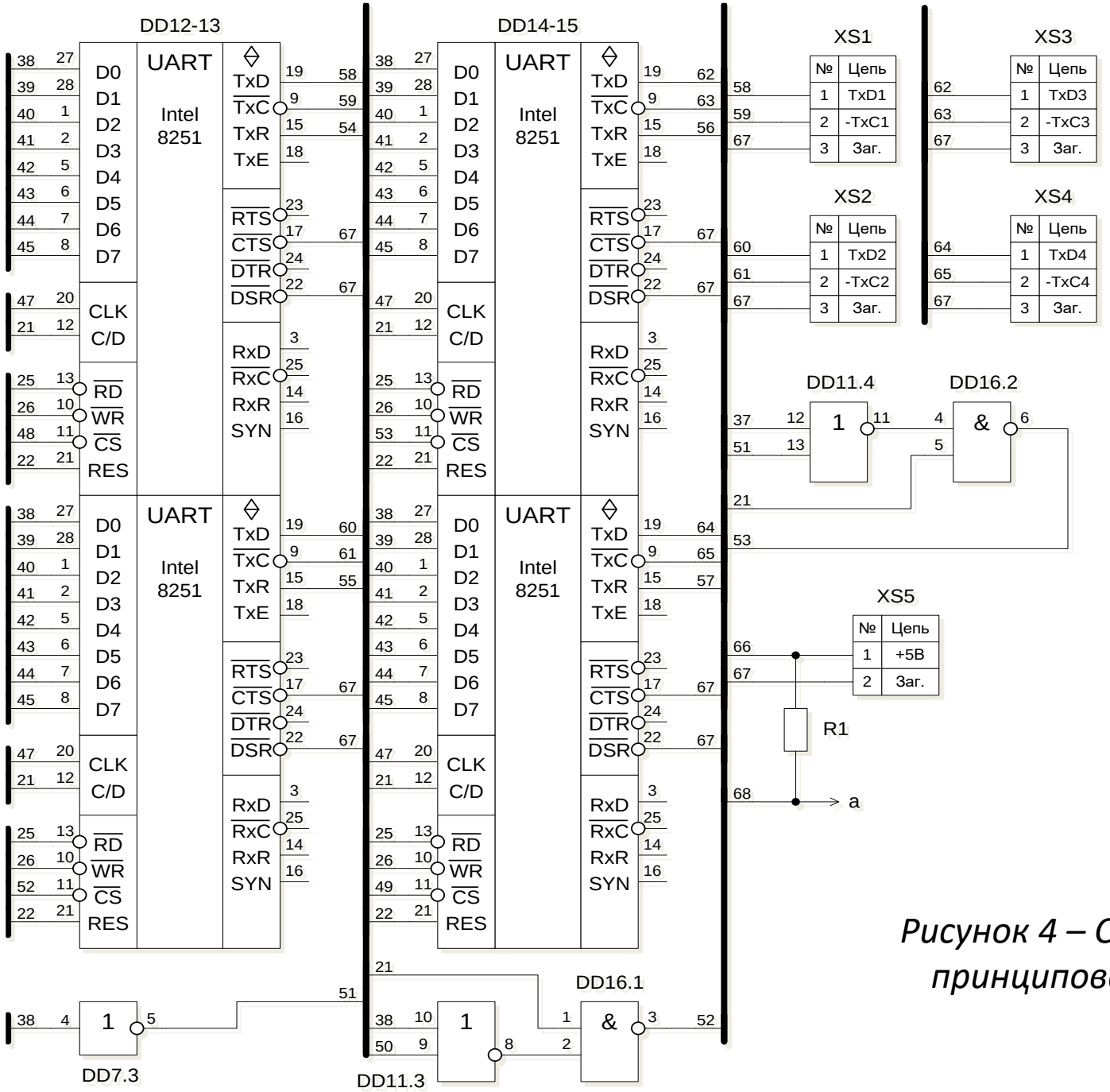


Рисунок 4 – Схема електрична принципова мультипорту

5. Задачі для розв’язання за темою

Задача 1. Побудувати блок-схему алгоритму функціонування мультипорту послідовного зв’язку (9-12 операцій), що представлений на рисунку 4.

Задача 2. Переналаштуйте адреси послідовних портів мультипорту, що представлений на рисунку 4, на адреси наступного виду 10h, 14h, 18h та 2Ch. Яким чином це можна було зробити без демонтажу і втручання в конструкцію мултипорту послідовного зв’язку?

Задача 3. Надати підпрограму ініціалізації та налаштування послідовних портів з адресами 00h, 04h, 08h і 0Ch мультипорту, що представлений на рисунку 4, на асинхронний режим, використовуючи мову Асемблер.

Задача 4. Визначити частоту синхроімпульсів, з якою необхідно подавати на контролери послідовного зв’язку зі схеми мультипорту (рисунок 4), для забезпечення швидкості передачі інформації по послідовному каналу 9600 біт/с.

Задача 5. Побудувати схему двопортового пристрою комутації паралельного зв’язку в режимі двостороннього обміну даними з адресами застосованих портів A0h і AAh на основі контролерів паралельного інтерфейсу Intel 8255 (без включення до схеми мультипорту блоку процесора).

Практичне заняття 4. Мультимікроконтролерні системи на основі послідовних портів

1. Послідовний порт МК сімейства MCS-51.
2. Мультимікроконтролерна система з асинхронним 8-бітовим режимом.
3. Мультимікроконтролерна система з асинхронним 9-бітовим режимом з фіксованою швидкістю передачі.
4. Задачі для розв’язання за темою.

1. Послідовний порт МК сімейства MCS-51

Через універсальний послідовний порт МК здійснюють прийом і передачу інформації, представленої в послідовному коді (молодшими бітами вперед).

Роботою послідовного порту управляють три регістра:

регістр керування /статусу приймача SCON; регістр управління потужністю PCON, біт SMOD; буферний регістр приймача SBUF.

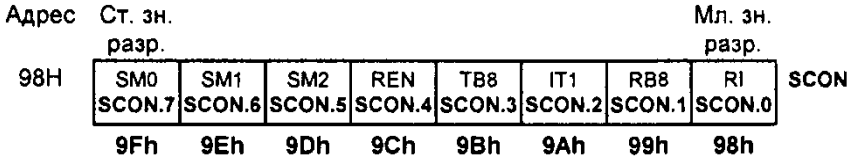
Послідовний порт може працювати в чотирьох різних режимах:

режим 0, синхронний; режим 1, асинхронний 8-бітовий обмін; режим 2, асинхронний 9-бітовий обмін з фіксованою швидкістю передачі; режим 3, асинхронний 9-бітовий режим.

Управління режимами роботи прийомопередавача здійснюється через спеціальний регістр SCON (рисунок 1). Він містить не тільки керуючі біти, що визначають режим роботи послідовного порту, але і дев'ятий біт прийнятих або переданих даних (RB8 і TB8) і біти переривання приймача/передавача (RI і TI).

У всіх 4-х режимах передача ініціалізується будь-якою командою, в якій буферний регістр SBUF зазначений як одержувач байту. Прийом в режимі 0 здійснюється за умови, що RI = 0 і REN = 1, в інших режимах - за умови, що REN = 1.
У біті TB8 програмно встановлюється значення 9-го біта даних, який буде переданий в режимі 2 або 3 в складі 9-бітового поля даних кадру передачі.

1. Послідовний порт МК сімейства MCS-51



Символ	Позиція	Ім'я й призначення
SM0	SCON.7	Біти керування режимом роботи приемопередавача. Установлюються/скидаються програмно див. примітка 1
SM1	SCON.6	
SM2	SCON.5	Біт керування режимом приемопередавача. Установлюється програмно для заборони прийому повідомлення, у якому дев'ятий біт має значення 0
REN	SCON.4	Біт дозволу прийому. Установлюється/скидається програмно для дозволу/заборони прийому послідовних даних
TB8	SCON. 3	Передача біта 8. Установлюється/скидається програмно для завдання дев'ятого переданого біта в режимі 9-бітового передавача
RB8	SCON.2	Прийом біта 8. Установлюється/скидається апаратно для фіксації дев'ятого прийнятого біта в режимі 9-бітового приймача
TI	SCON. 1	Прапор переривання передавача. Установлюється апаратно при закінченні передачі байта. Скидається програмно після обслуговування переривання
RI	SCON.0	Прапор переривання приймача. Установлюється апаратно при прийомі байта. Скидається програмно після обслуговування переривання

SM0	SM1	Режим роботи приемопередавача
0	0	регістр, Що Зрушує, розширення уведення/ виводу
0	1	8 бітовий приемопередавача, змінювана швидкість передачі
1	0	9 бітовий приемопередавача. Фіксована швидкість передачі
1	1	9 бітовий приемопередавача, змінювана швидкість передачі

У біті RB8 в цих режимах запам'ятовується 9-й біт даних, який приймається . У режимі 1 у біт RB8 заноситься значення стоп-біта. У режимі 0 біт RB8 не використовується.

Прапор переривання передавача TI встановлюється апаратно після передачі стоп-біта у всіх режимах. Підпрограма, яка обслуговує переривання або опитує прапор, повинна скидати біт TI. Прапор переривання приймача RI встановлюється апаратно після прийому 8-го біта даних в режимі 0 і в середині періоду прийому стоп-біта в режимах 1, 2 і 3. Підпрограма обслуговування переривання повинна скидати біт RI.

Рисунок 1 – Формат регістра SCON

2. Мультимікроконтролерна система з асинхронним 8-бітовим режимом

В асинхронному режимі інформація передається через вивід виходу передавача послідовного порту МК TxD, а приймається через вивід входу приймача RxD, т. е. в цьому режимі роботи *послідовний порт працює в дуплексному режимі*. *Передача і прийом інформації можуть вестися одночасно і незалежно один від одного*. Швидкість передачі в цьому режимі задається за допомогою таймера Т1.

Для отримання стандартної швидкості передачі зазвичай використовується кварцовий резонатор з частотою 11,0592 МГц.

Таблиця 1 – Відповідність завантажуваних констант стандартним швидкостям передачі

Частота прийому/ передачі	Частота резонатора МГц	Таймер/лічильник 1			
		SMOD	C/T	Режим (M1, M0)	Число, що перезаванта-жується
62,2 кбіт/с	12	1	0	1, 0	0FFH
19,2 кбіт/с	11,059	1	0	1, 0	0FDH
9,6 кбіт/с	11,059	0	0	1, 0	0FDH
4,8 кбіт/с	11,059	0	0	1, 0	0FAH
2,4 кбіт/с	11,059	0	0	1, 0	0F4H
1,2 кбіт/с	11,059	0	0	1, 0	0E8H
110 кбіт/с	6	0	0	1, 0	072H

Значення констант, які завантажують в таймер 1 для отримання стандартних швидкостей прийому /передачі при використанні кварцового резонатора 11,0592 МГц, наведені в таблиці 1.

2. Мультимікроконтролерна система з асинхронним 8-бітовим режимом

При роботі в асинхронному режимі два МК можуть обмінюватися інформацією між собою, використовуючи мінімум з'єднувальних проводів між блоками або окремими пристроями. Швидкості роботи передавального і приймаючого послідовних портів повинні бути однаковими (стандартні швидкості 1200, 2400 біт/с і т. д.).

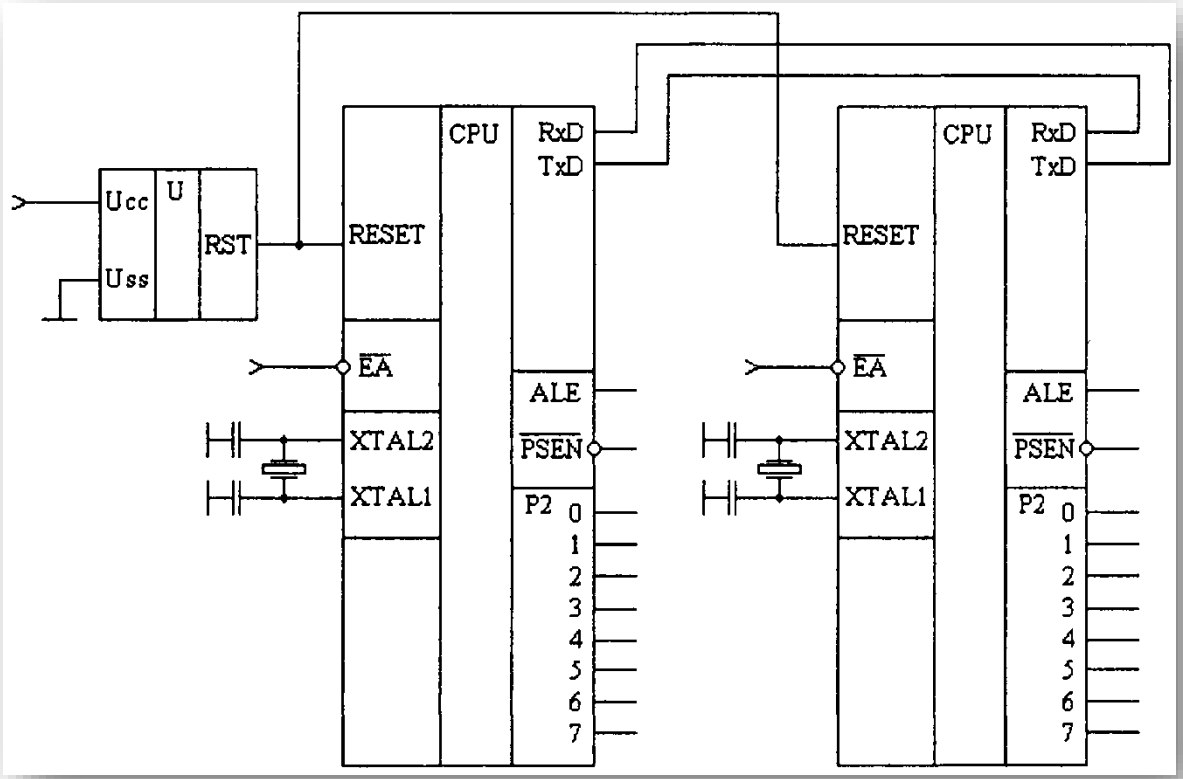


Рисунок 2 – Найпростіша багатопроцесорна система

В режимі 1 можливий обмін інформацією між двома МК. Таким чином, може бути побудована найпростіша багатопроцесорна система.

У режимі 1 для передачі байту через послідовний порт досить скопіювати його в буфер даних SBUF. Крім настройки регістра SCON, необхідно налаштувати таймер для завдання швидкості передачі інформації по послідовному порту. Прийом починається тільки після виявлення стартового біта.

2. Мультимікроконтролерна система з асинхронним 8-бітовим режимом

```
;*********************************************************************
;НАЛАШТУВАННЯ ПОСЛІДОВНОГО ПОРТУ
;*********************************************************************
;Налаштувати режим роботи послідовного порту  ----------------------
  MOV SCON,#01110000b ;настроить последовательный порт на режим 1
           ;||||||||
           ;|||||||+---Обнулити прапор приймача RI
           ;||||||+----Обнулити прапор передавача TI
           ;|||||+-----Обнулити дев'ятий біт приймача RB8
           ;||||+------Обнулити дев'ятий біт передавача TB8 TB8
           ;|||+-------Дозволити роботу приймача
           ;||+--------Перевіряти помилку кадра (прийом нульового біта
           ;||          на місці стоп-біта)
           ;++---------Включити асинхронний режим роботи


;Настроить режим работы таймера T1 ---------------------------
  ANL TMOD,#00001111b ;Підготвувати  таймер T1 до налаштування
                      ;(таймер T0 не чіпати!)
  ORL TMOD,#00100000b ;Перевести  таймер T1 в режим 2
           ;||||       ;(таймер T0 не чіпати!)
           ;||++------Перевести  таймер T1 в режим автозавантаження
           ;|+--------Робота від внутрішнього генератора
           ;+---------Заборонити управління таймером від виводу INT1

;Налаштувати таймер на генерацію 3-мікросекундного інтервалу часу  ----
  MOV TH0, #0fdh          ;Завантажити старший байт таймера
  MOV TL0, #0fdh          ;Завантажити молодший байт таймера
  SETB TR1                ;Включити таймер 1
```

Приклад програми, що дозволяє здійснити прийом інформації по послідовному порту в режимі асинхронного обміну

Продовження

```
;*********************************************************************
;РОБОТА З ПОСЛІДОВНИМ ПОРТОМ
;*********************************************************************
  JNB RI, $    ;Почекати закінчення прийому байта по послідовному порту
  MOV A, SBUF  ;і скопіювати його в акумулятор
```

3. Мультимікроконтролерна система з асинхронним 9-бітовим режимом з фіксованою швидкістю передачі

Послідовний порт працює на фіксованій швидкості передачі. Швидкість обміну визначається значенням біта SMOD і при частоті кварцового резонатора 12 МГц складає 375 Кбіт/с. У сучасних МК швидкість обміну може перевищувати 1 Мбіт/с.

Основна особливість цього режиму – передача 9-го інформаційного біта, який може бути використаний для контролю достовірності інформації, що передається. Для обчислення парності переданого байту можна скористатися апаратним обчислювачем, підключеним до акумулятора МК. Результат обчислення парності байту зберігається в біті парності P регістра PSW, звідки його можна скопіювати в 9-й інформаційний біт послідовного порту TB8, розташований в регістрі управління послідовним портом SCON.

Паралельні порти МК сімейства MCS-51 побудовані за схемою з відкритим стоком. Це дозволяє об'єднувати декілька виходів передавачів в одну шину. Таке виконання вихідних каскадів мікросхем полегшує побудову багатопроцесорних систем (рисунок 4). У багатопроцесорної системі один процесор повинен бути головним (Master), інші - підлеглими (slave). Природно, команди головного процесора повинні сприйматися підлеглими, тому вихід передавача головного процесора з'єднується з входами приймачів підлеглих.

Виходи ж передавачів підлеглих процесорів об'єднуються і підключаються до входу приймача головного процесора.

3. Мультимікроконтролерна система з асинхронним 9-бітовим режимом з фіксованою швидкістю передачі

Команди головного процесора можуть бути звернені до конкретного підлеглого процесору, тому до складу команд включається адреса підлеглого процесора. При роботі в шині необхідно вміти відрізняти адресну інформацію від даних. Це можна здійснити за допомогою 9-го біта. Зазвичай при передачі адреси в 9-й біт записують 1, а при передачі даних і команд - 0. Таким чином, МК, навіть підключившись до шини пізніше інших, легко може здійснити синхронізацію з багатопроцесорною шиною.

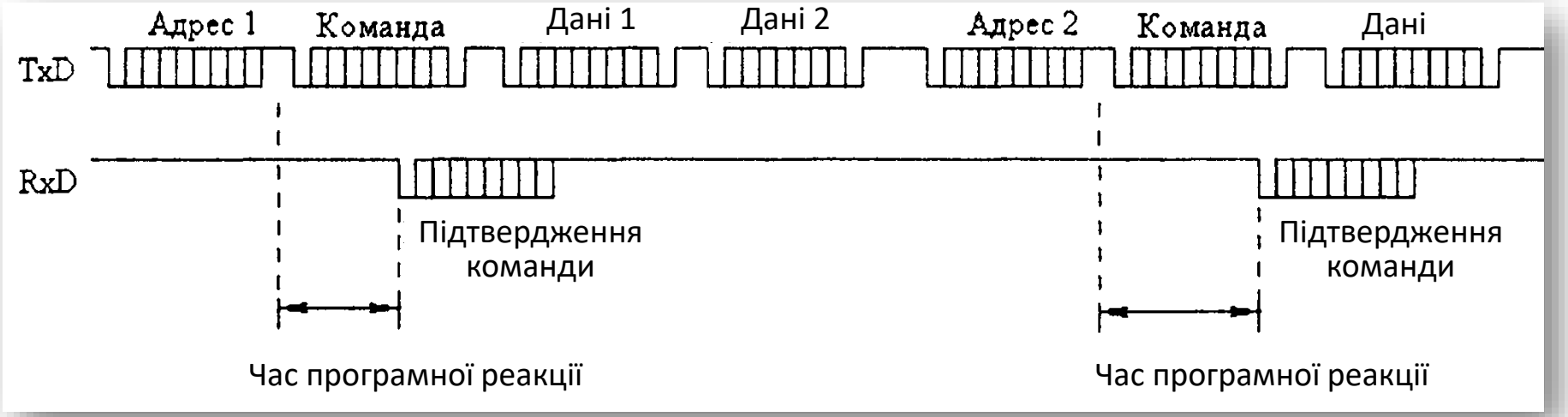


Рисунок 3 – Часова діаграма роботи багатопроцесорної шини

Налагодження та робота з портом в режимі 2 у звичайному порядку. Всі особливості обміну зосереджені на протокольному рівні

3. Мультимікроконтролерна система з асинхронним 9-бітовим режимом з фіксованою швидкістю передачі

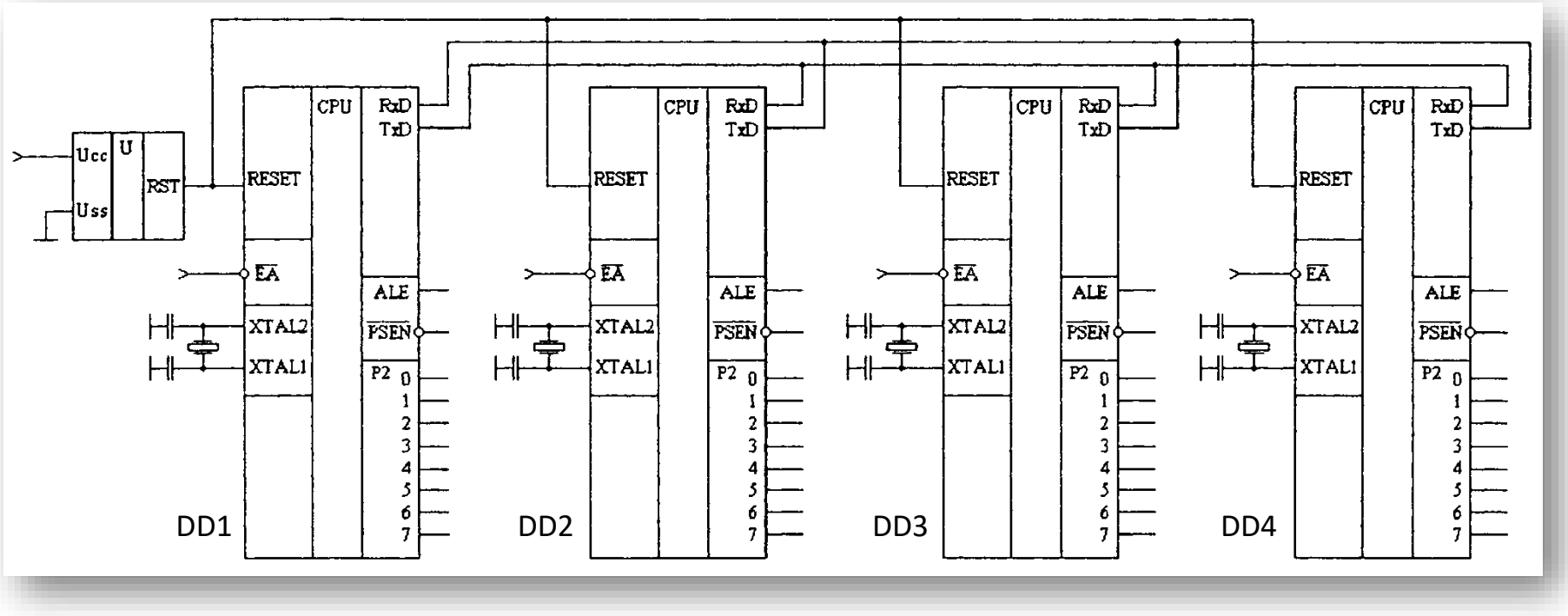


Рисунок 4 – Схема з'єднання декілька МК між собою через послідовні порти, які працюють в асинхронному режимі

4. Задачі для розв’язання за темою

Задача 1. Визначити значення константи, яке повинне бути завантажено в таймер 1 в асинхронному 8-бітовиму режимі, для забезпечення швидкості передачі інформації 2,4 Кбіт/с по послідовному порту мікроконтролера MCS-51 при частоті кварцового резонатора 8 МГц.

Задача 2. Написати програму мовою Асемблер, що дозволяє здійснити прийом інформації по послідовному порту в режимі 9-бітового асинхронного обміну по послідовному порту MCS-51 .

Задача 3. Створити протокол обміну даними між головним DD1 та підлеглими DD2-DD4 мікроконтролерами багатопроцесорної системи на рисунку 4 для зчитування:

- ✓ сигналів кодів датчиків, які під’єднані: до DD2 – 5 датчиків, DD3 – 4 датчика, DD4 – 8 датчиків;
- ✓ вмісту таймерів 0 підлеглих мікроконтролерів;
- ✓ середніх значень датчиків з кожного мікроконтролера за годину, добу і тиждень.

Представити алгоритм обміну даними між головним та підлеглими мікоконтролерами, вказуючі командні та адресні кодові комбінації, що прийняті для обміну даними.

Практичне заняття 5. Застосування перетворювачів інтерфейсу RS-232

1. Перетворювачі MAX318X, MAX3190, ADM3202, MAX1406.
2. Транзисторні перетворювачі сигналів інтерфейсу.
3. Схеми пристрою спряження на основі RS-232.
4. Задачі для розв’язання за темою.

1. Перетворювачі MAX318X, MAX3190, ADM3202, MAX1406

У мікроконтролерів обмін за інтерфейсом RS-232 здійснюється лініями TxD (передавач) і RxD (приймач). Рівні напруги цих лініях відповідають стандартним (цифровим) рівням напруги МК. Це означає, що рівень напруги логічної одиниці відповідає напрузі живлення мікроконтролера (3 або 5 В), рівень напруги логічного нуля – нульовій напрузі (або «землі»). Для поєднання зі стандартними рівнями напруги сигналів на лініях інтерфейсу RS-232 (приблизно рівними ±10 В) необхідно використовувати перетворювачі рівнів RS-232.

*Нові перетворювачі інтерфейсу RS-232 (MAX318X, MAX3190, ADM3202, MAX1406) мають перевагу – **висока швидкість обміну, малі габарити і споживання енергії**, а також **досить низька вартість** – перед тими, що використовувалися раніше.*

Крім того, перехід на 3-вольтове живлення, яке почало підтримуватися багатьма сучасними мікроконтролерами, дозволив по-новому підійти до використання стандартних перетворювачів інтерфейсу (наприклад, ADM231L).

1. Перетворювачі MAX318X, MAX3190, ADM3202, MAX1406

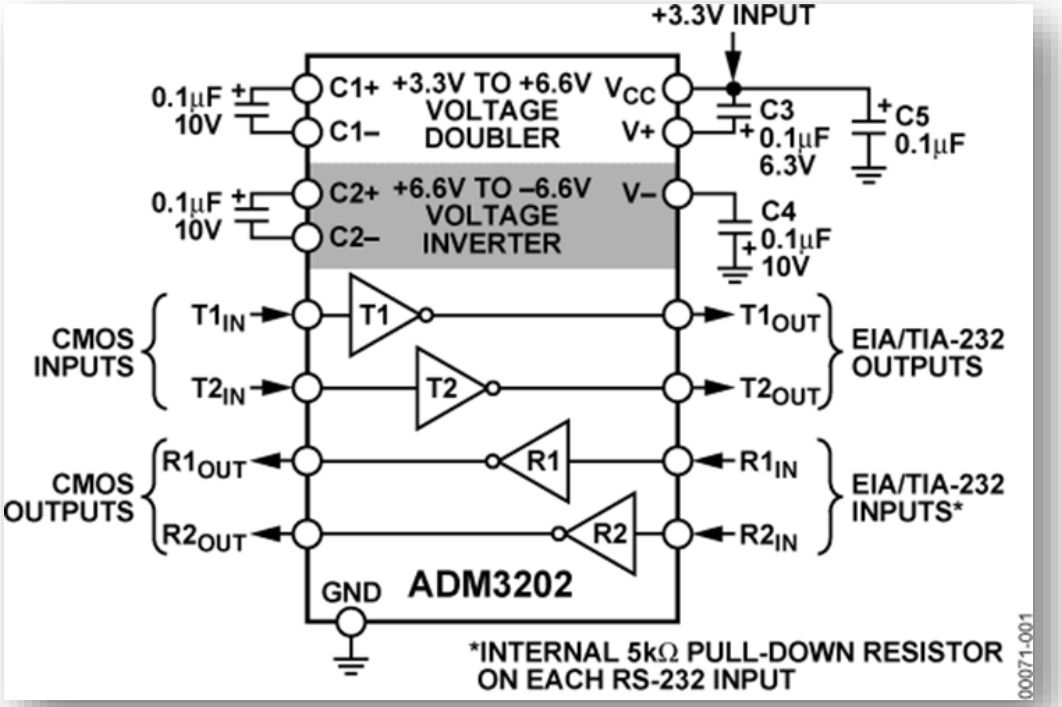


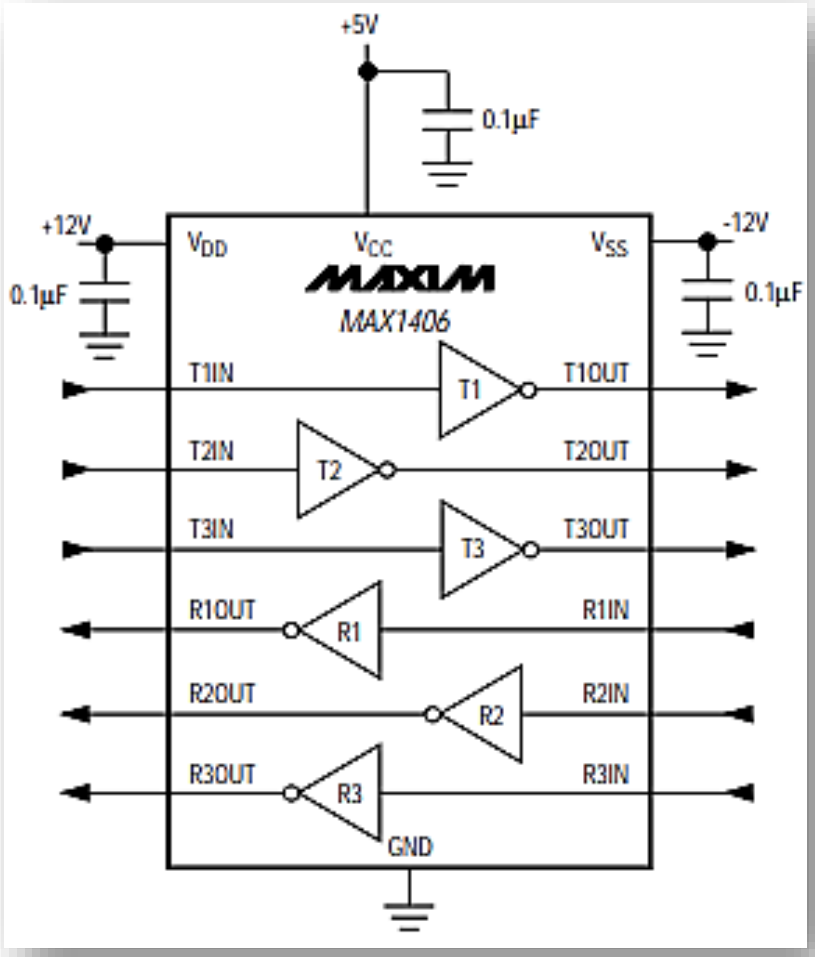
Рисунок 1 – Інтегральна схема Analog Devices ADM3202

Розмах сигналу драйвера (передавача) RS-232 становить ±10 В (при Vcc = +5 В) і ±6 В (при Vcc = +3 В) під час роботи на навантаження 5 ком. Сигнал дуже чистий, без паразитних спотворень. Крім того, ADM3202 відрізняється зниженим споживанням енергії (особливо під час живлення від +3 В).

Інтегральна схема Analog Devices ADM3202 (рисунок 1) відрізняється високою швидкістю роботи (до 460 кбод); ємності конденсаторів, необхідних роботи перетворювача, не перевищують 0,1 мкФ; мікросхема може працювати і за Vcc = +3 В, і за Vcc = +5 В.

Інтегральна схема Analog Devices ADM3202 відрізняється високою швидкістю роботи (до 460 кбод); ємності конденсаторів, необхідних роботи перетворювача, не перевищують 0,1 мкФ; мікросхема може працювати і за Vcc = +3 В, і за Vcc = +5 В.

1. Перетворювачі MAX318X, MAX3190, ADM3202, MAX1406

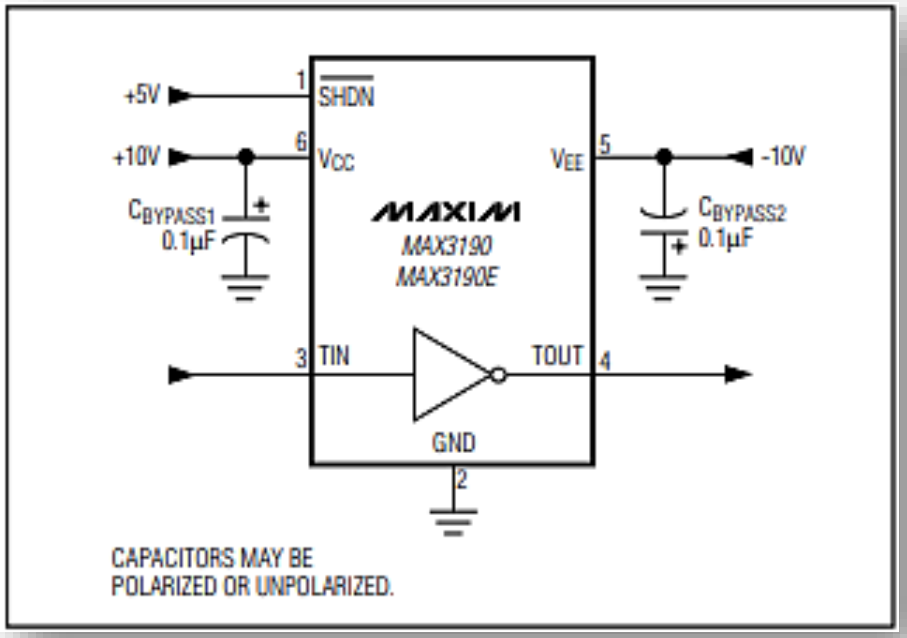


(VCC = +4.5V to +5.5V, VDD = +10.8V to +13.2V,
VSS = -10.8V to -13.2V,

Рисунок 2 – Інтегральна схема MAX1406

Інтегральна схема MAX1406 (рисунок 2) відрізняється високою швидкістю роботи (до 230 кбод), підвищеним розмахом сигналу драйвера (±11,5 В) при роботі на навантаження 5 кОм, незалежністю цього розмаху від напруги Vcc і високою чистотою сигналу. Мікросхема живиться від 3 джерел живлення (V+ = +12 В, V– = -12 В і Vcc: +3 ± +5 В), у зв'язку з чим вона не містить перетворювачів напруги (подвійників та інверторів напруги як, наприклад, ADM3202 та ADM231L) та не вимагає великої кількості конденсаторів для роботи. Особливістю мікросхеми є наявність трьох приймачів і трьох передавачів, що дозволяє сполучати з її допомогою комп'ютер з МК (який може працювати як в режимі програмування, так і в штатному режимі) без додаткових перетворювачів інтерфейсу RS-232. MAX1406 відрізняється зниженим споживанням енергії.

1. Перетворювачі MAX318X, MAX3190, ADM3202, MAX1406



- Small 6-Pin SOT23 Package
- ESD-Protected RS-232 Output (MAX3190E)
  - ±15kV per Human Body Model
  - ±8kV per IEC 1000-4-2 Contact Discharge
  - ±15kV per IEC 1000-4-2 Air-Gap Discharge
- 200µA Operating Supply Current
- Shutdown Reduces Supply Current to 0.4µA
- RS-232-Compliant Operation from ±7.5V to ±12V Supplies
- RS-232-Compatible Operations from ±6V to ±7.5V Supplies
- 460kbps Guaranteed Data Rate
- Three-State RS-232 Transmitter Output
- No External Components

Рисунок 3 – Інтегральна схема MAX3190

Мікросхема передавача RS-232 MAX3190 від MAXIM (рисунок 3) відрізняється підвищеною швидкістю роботи (до 460 кбод), має розмах сигналу драйвера до ±10 В при роботі на два входи приймачів (на навантаження в 2,5 кОм) при напругах живлення V+ = +12 В і V– = –12 В.

Крім того, вона споживає дуже мало енергії під час роботи і може бути переведена в режим наднизького споживання («сплячий» режим).

Особливістю мікросхеми є унікальний малий корпус SOT23-6 розміром 3×3 мм.

1. Перетворювачі MAX318X, MAX3190, ADM3202, MAX1406

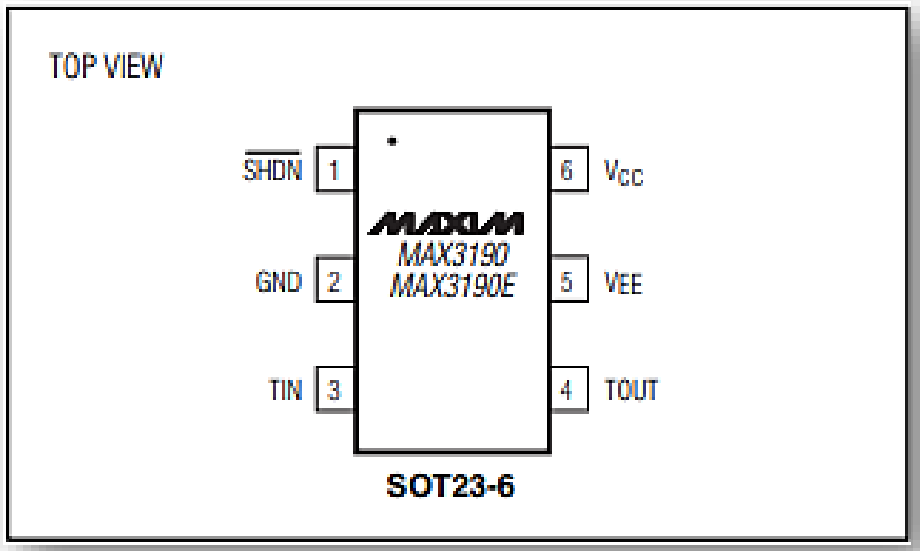
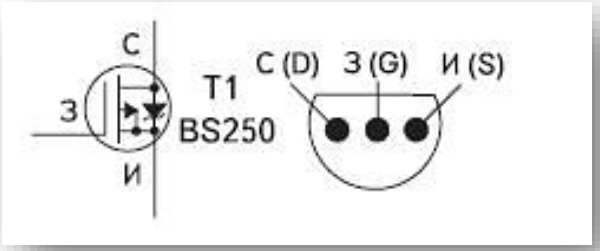
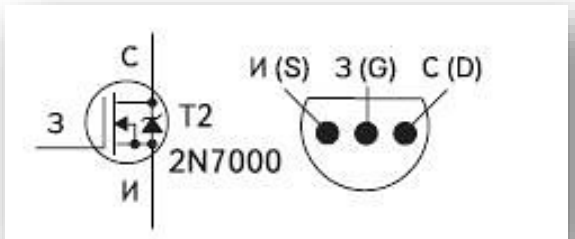


Рисунок 4 – Інтегральна схема MAX3190 (корпус)

Крім спеціалізованих ІС перетворювачів інтерфейсу RS-232, існують електронні компоненти, взагалі кажучи, що не є перетворювачами, але які можна використовувати як такі перетворювачі. Наприклад, це КМДП-транзистори прямої (BS250 від Vishay) та зворотної (2N7000 від Fairchild або Vishay) провідності з ізольованим затвором та КМДП-комутатор DG419 (від Vishay) на рисунку 5.





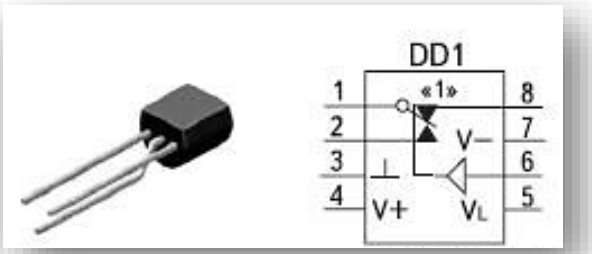


Рисунок 5 – КМДП-транзистори прямої (BS250) та зворотної (2N7000) провідності з ізольованим затвором та КМДП-комутатор DG419

2. Транзисторні перетворювачі сигналів інтерфейсу

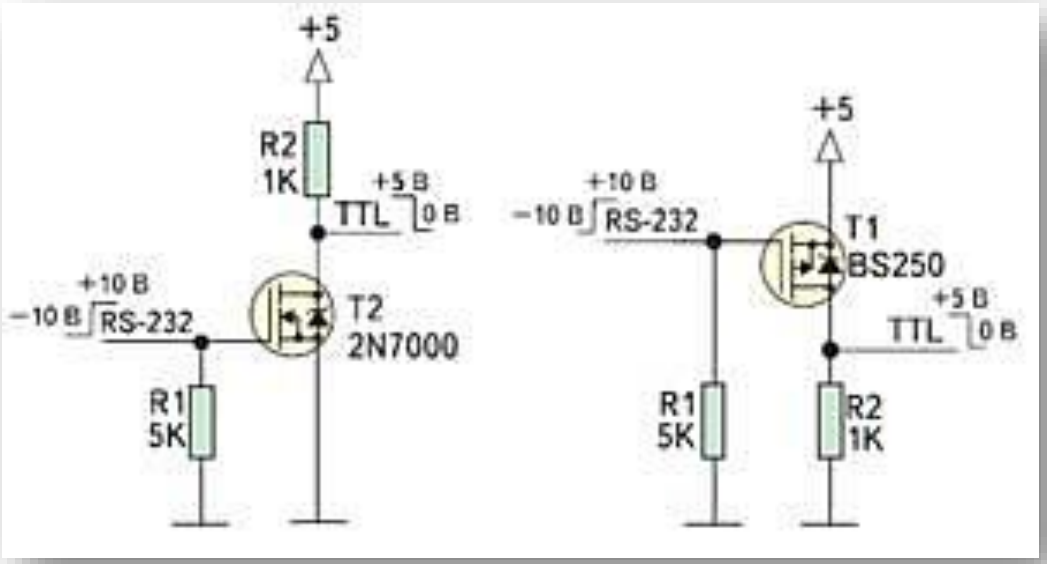


Рисунок 6 – Варіанти використання КМДП-транзисторів 2N7000 і BS250 як перетворювачі інтерфейсу RS-232 – інвертуючих приймачів RS-232

Транзистори випускаються у корпусі TO-92. Опір транзисторів у відкритому стані становить близько 10 Ом, максимальний струм стоку – трохи більше 200 мА, максимальна напруга «стік-витік» - не більше 50-60 В, час включення та вимкнення - близько 10 нс, потужність розсіювання - близько 0,5 Вт.

Максимальна напруга «затвор-витік» (Uзі_max) становить ±20 В. Остання властивість дозволяє підключати затвор транзистори безпосередньо до ліній RS-232 (нагадаю, що сигнал передавача RS-232 становить близько ±10 ± 12В), у зв'язку з чим транзистори можуть використовуватися як приймачі RS-232 (рисунки 6, 7).

2. Транзисторні перетворювачі сигналів інтерфейсу

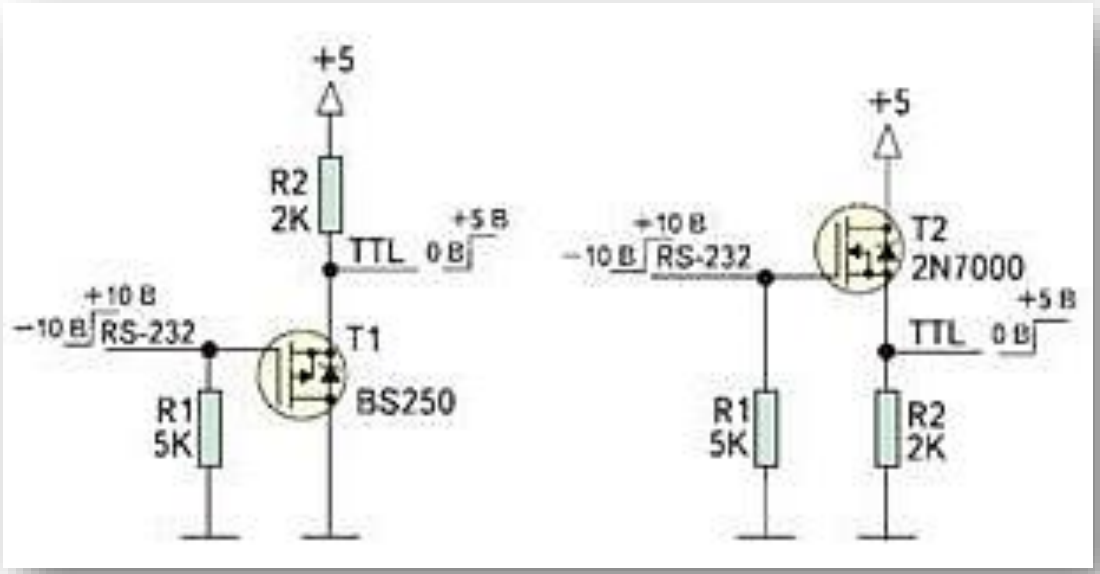


Рисунок 7 – Варіанти використання КМДП-транзисторів 2N7000 і BS250 як перетворювачі інтерфейсу RS-232 – неінвертуючих приймачів RS-232

Єдине, що необхідно передбачити, це резистор навантаження номіналом в 5 кОм, який слід підключити між затвором і загальним проводом («землею»), так як опір (ізольованого) затвора транзисторів становить сотні МОм, а стандарт RS-232 передбачає вхідний опір приймача в 5 ком.

3. Схеми пристрою спряження на основі RS-232

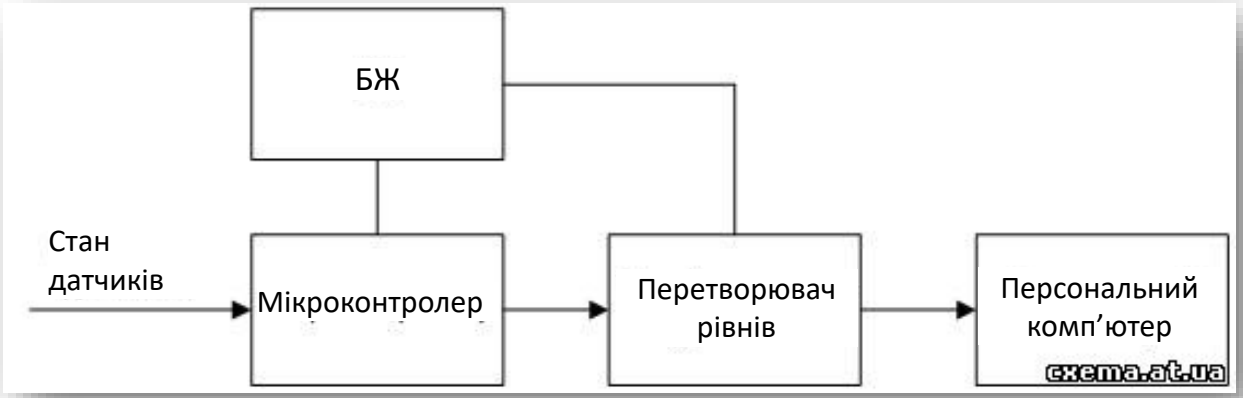


Рисунок 8 – Структурна схема пристрою спряження на основі інтерфейсу RS-232 (БЖ – блок живлення)

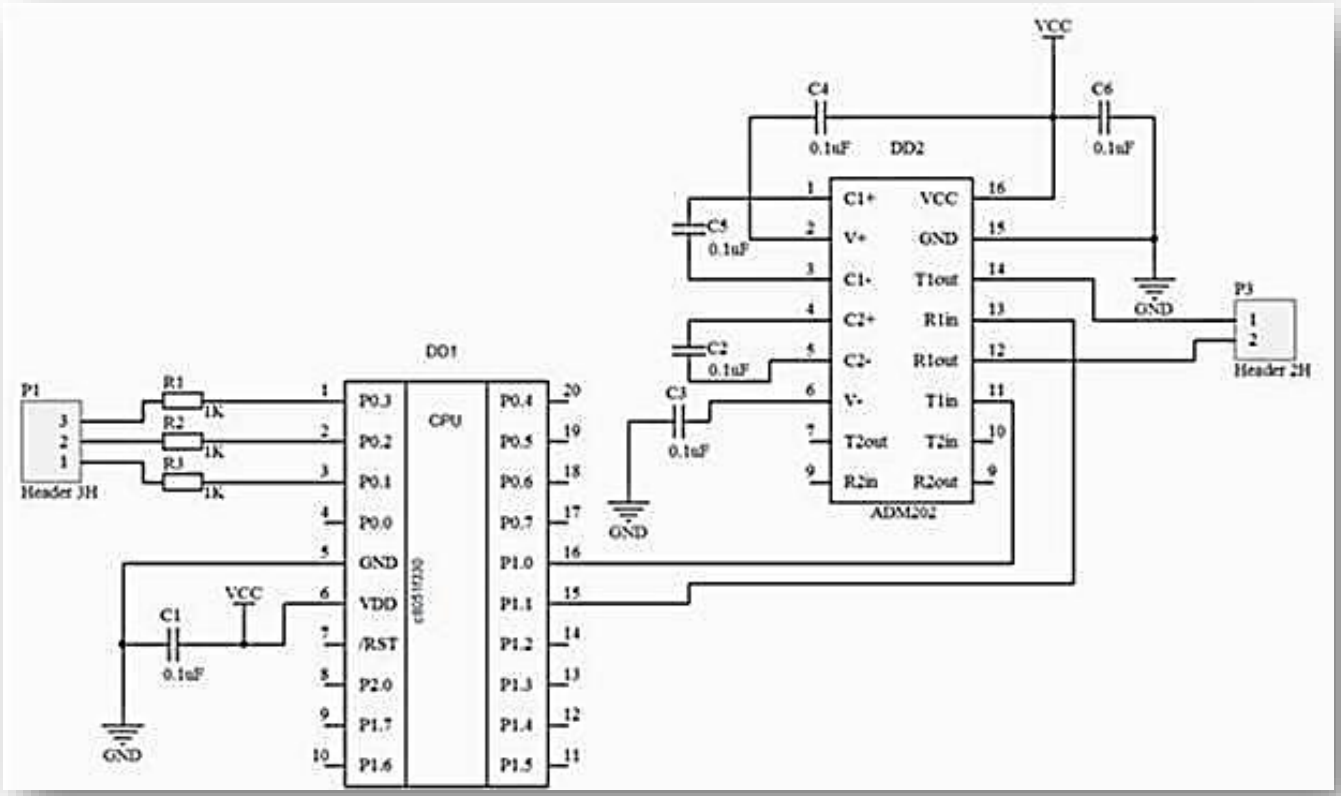


Рисунок 9 – Принципова схема пристрою спряження

4. Задачі для розв’язання за темою

Задача 1. Приєднайте (на рисунку 10 за допомогою стрілок, які відображують фізичну лінію та напрям передавання сигналів) зовнішній мікроконтролерний пристрій до персонального компютера згідно протоколу RS-232, використовуючи:

а) нуль-модемне з’єднання;

б) трьохпровідне з’єднання.

Рисунок 10 – З’єднання за інтерфейсом RS-232

Задача 2. За часовою діаграмою на інформаційній лінії RS-232 на рисунку 11 визначити, яка кодова комбінація передається. Яким чином зміниться часова діаграма, якщо буде використовуватися перевірка правильності передачі на парність? Відобразіть таку часову діаграму.

4. Задачі для розв’язання за темою

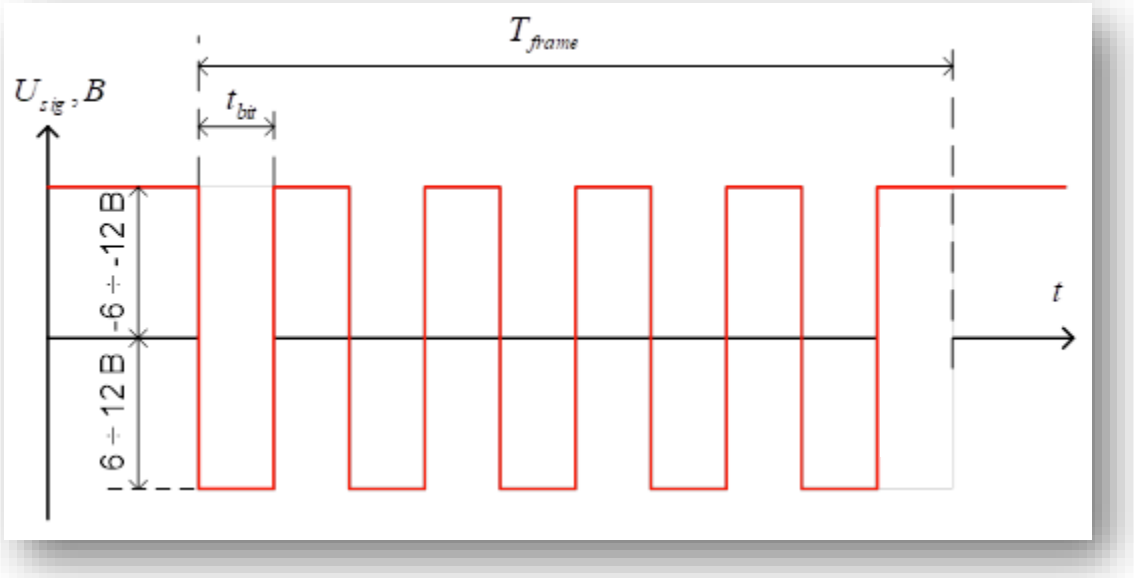


Рисунок 11 – Часова діаграма на інформаційній інтерфейсу RS-232

Задача 3. Відобразіть часову діаграму на інформаційній лінії RS-232 у випадку передачі двох кодових комбінацій 01001110 і 10100110. Додаткові налаштування інтерфейсу RS-232 – 2 стопових біта, перевірка на парність.

Задача 4. Продемонструйте, яким чином зміниться схема з’єднання за інтерфейсом RS-232 на рисунку 9, якщо перетворювач сигналів ADM202 замінити на перетворювач сигналів MAX3190.

Практичне заняття 6. Особливості з’єднання на основі інтерфейсу RS-485

1. Особливості роботи UART.
2. Апаратна реалізація інтерфейсу RS-485.
3. Інтегральні перетворювачі RS-485.
4. Узгодження та конфігурація лінії зв'язку RS-485.
5. Захисне зміщення в RS-485.
6. Задачі для розв’язання за темою.

1. Особливості роботи UART

UART можна розділити на приймач (Receiver) та передавач (Transmitter). До складу UART входять: тактовий генератор зв'язку (бодрейт-генератор), регістри, що управляють, статусні регістри, буфери і зсувні регістри приймача і передавача.

Бодрейт-генератор задає тактову частоту передавача-приймача для даної швидкості зв'язку.

Керівні регістри задають режим роботи послідовного порту та його переривань. У статусному регістрі встановлюються прапори з різних подій. У буфер приймача потрапляє прийнятий символ, буфер передавача поміщають символ, що передається.

Зсувний регістр передавача – це обойма, з якої в послідовний порт вистрілюються біти символу, що передається (кадр). Зсувний регістр приймача по біту накопичує біти, що приймаються з порту.

За різними подіями встановлюються прапори та генеруються переривання (завершення прийому/надсилання кадру, звільнення буфера, різні помилки).

UART – повнодуплексний інтерфейс, тобто приймач і передавач можуть працювати одночасно незалежно один від одного. За кожним із них закріплений порт – одна ніжка контролера. Порт приймача позначають RX, передавача – TX.

Послідовною установкою рівнів цих портах щодо загального дроту («землі») і передається інформація. За промовчанням передавач встановлює на лінії одиничний рівень.

1. Особливості роботи UART

Передача починається посилкою біта з нульовим рівнем (старт-біта), потім йдуть біти даних молодшим бітом уперед (низький рівень – «0», високий рівень – «1»), завершується посилка передачею одного або двох бітів з одиничним рівнем (стоп-бітів) (рисунок 1).

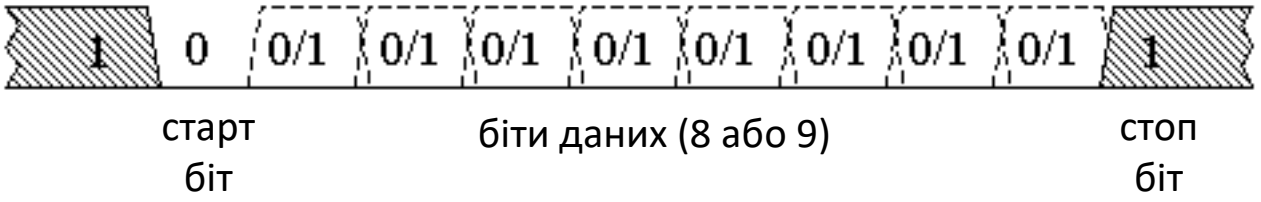


Рисунок 1 – Часова діаграма сигналів кадра посилання

Перед початком зв'язку між двома пристроями необхідно налаштувати їх передавачі-приймачі на однакову швидкість зв'язку та формат кадру.

Швидкість зв'язку або бодрейт (baudrate) вимірюється в бодах – число біт, що передаються в секунду (включаючи старт і стоп-біти). Задається ця швидкість в бодрейт-генераторі діленням системної частоти на коефіцієнт, що задається. Типовий діапазон швидкостей: 2400 … 115 200 бод.

Формат кадру визначає кількість стоп-бітів (1 або 2), число біт даних (8 або 9), а також призначення дев'ятого біта даних. Усе залежить від типу мікроконтролера.

1. Особливості роботи UART

Приймач та передавач тактуються, як правило, з 16-кратною частотою щодо бодрейту. Це необхідно для семплювання сигналу (рисунок 2). Приймач, упіймавши падаючий фронт старт-біта, відраховує кілька тактів і наступні три такти зчитує (семплює) порт RX. Це якраз середина старт-біту. Якщо більшість значень семплів – "0“, старт-біт вважається таким, що відбувся, інакше приймач приймає його за шум і чекає наступного фронту. Після успішного визначення старт-біта, приймач точно також семплює серединки бітів даних і по більшості семплів вважає біт "0" або "1", записуючи їх в зсувний регістр. Стоп-біти теж семплюються, і якщо рівень стоп-біта не "1" – UART визначає помилку кадру та встановлює відповідний прапор у регістрі, що управляє.

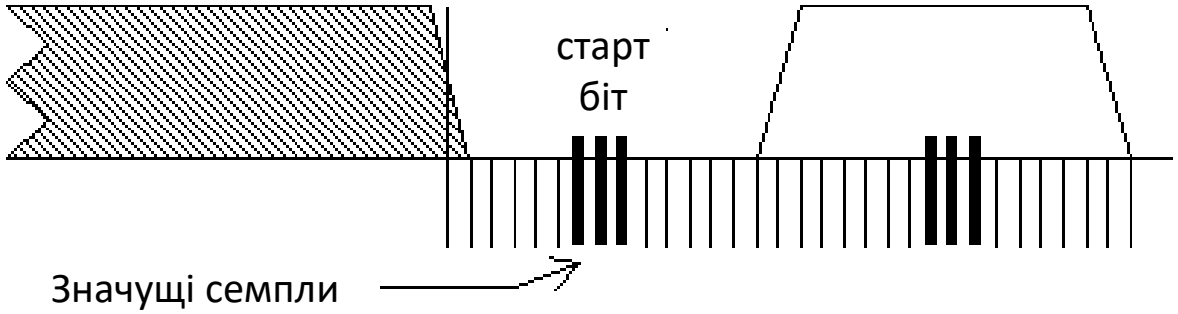


Рисунок 2 – Часова діаграма семплювання

Оскільки бодрейт встановлюється поділом системної частоти, при перенесенні програми на пристрій з іншим кварцовим резонатором необхідно змінити відповідні налаштування UART.

2. Апаратна реалізація інтерфейсу RS-485

Апаратна реалізація інтерфейсу RS-485 – мікросхеми передавачів-приймачів з диференціальними входами/виходами (до лінії) та цифровими портами (до портів UART контролера) (рисунок 3).

RS-485 – напівдуплексний інтерфейс. Прийом та передача йдуть по одній парі проводів із поділом за часом. У мережі може бути багато передавачів, оскільки вони можуть вимикатися в режимі прийому.

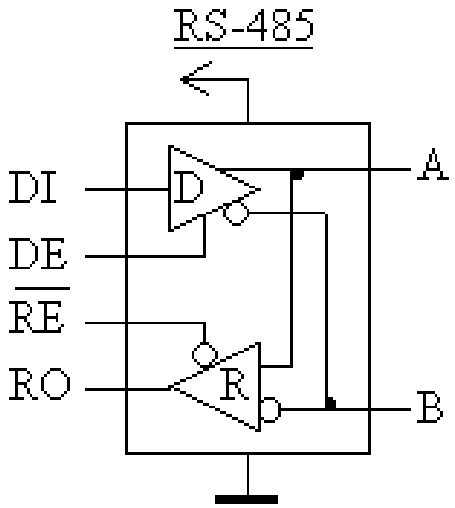


Рисунок 3 – Передавач-приймач RS-485

Цифровий вихід приймача (RO) підключається до порту приймача UART (RX). Цифровий вхід передавача (DI) до порту передавача UART (TX).

Оскільки на диференціальній стороні приймач і передавач з'єднані, під час прийому потрібно відключати передавач, а під час передачі – приймач. Для цього служать керуючі входи – дозвіл приймача (RE) та дозвіл передавача (DE).

Так як вхід RE інверсний, його можна з'єднати з DE і перемикати приймач і передавач одним сигналом з будь-якого порту мікроконтролера. При рівні "0" - робота на прийом, при "1" – передачу.

2. Апаратна реалізація інтерфейсу RS-485

Приймач, отримуючи на диференціальних входах (АВ) різницю потенціалів (U_{AB}) переводить їх у цифровий сигнал на виході RO (рисунок 4). Чутливість приймача може бути різною, але гарантований пороговий діапазон розпізнавання сигналу виробники ІС приймачів пишуть у документації.

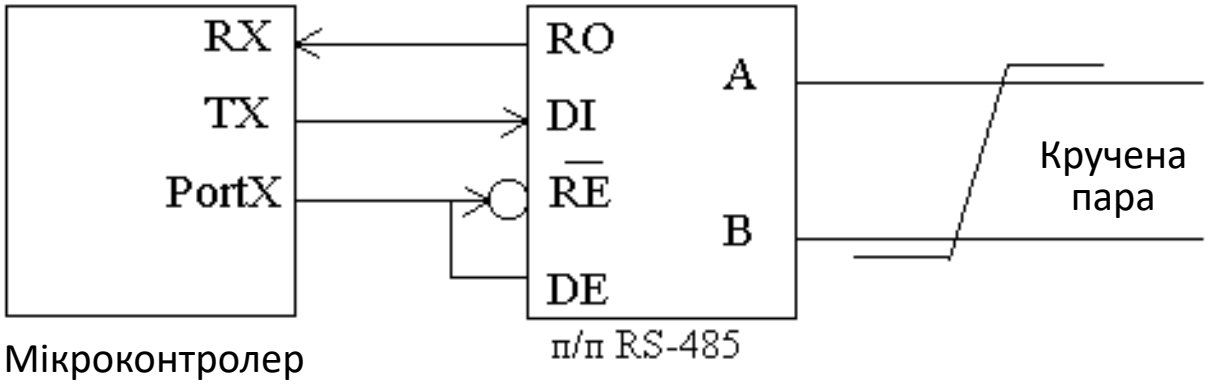


Рисунок 4 – Передавач-приймача RS-485 з мікроконтролером

Зазвичай пороги становлять ± 200 мВ. Тобто, коли U_{AB} > +200 мВ – приймач визначає "1", коли U_{AB} <-200 мВ – приймач визначає "0". Якщо різниця потенціалів у лінії настільки мала, що не виходить за граничні значення – правильне розпізнавання сигналу не гарантується. Крім того, в лінії можуть бути і не синфазні перешкоди, які спотворять такий слабкий сигнал.

2. Апаратна реалізація інтерфейсу RS-485

Всі пристрої підключаються до однієї кручений пари однаково: прямі виходи (A) до одного дроту, інверсні (B) – до іншого.

Вхідний опір приймача з боку лінії (R_{AB}) зазвичай становить 12 кОм.

Так як потужність передавача не безмежна, це створює обмеження кількості приймачів, підключених до лінії. Відповідно до специфікації RS-485 з урахуванням резисторів, що погоджують, передавач може вести до 32 приймачів.

Однак є ряд інтегральних схем із підвищеним вхідним опором, що дозволяє підключити до лінії значно більше 32 пристроїв.

Максимальна швидкість зв'язку специфікації RS-485 може досягати 10 Мбод/сек. Максимальна відстань - 1200 м.

Якщо необхідно організувати зв'язок на відстані більшій за 1200 м або підключити більше пристроїв, ніж допускає здатність навантаження передавача – застосовують спеціальні повторювачі (репітери).

3. Інтегральні перетворювачі RS-485

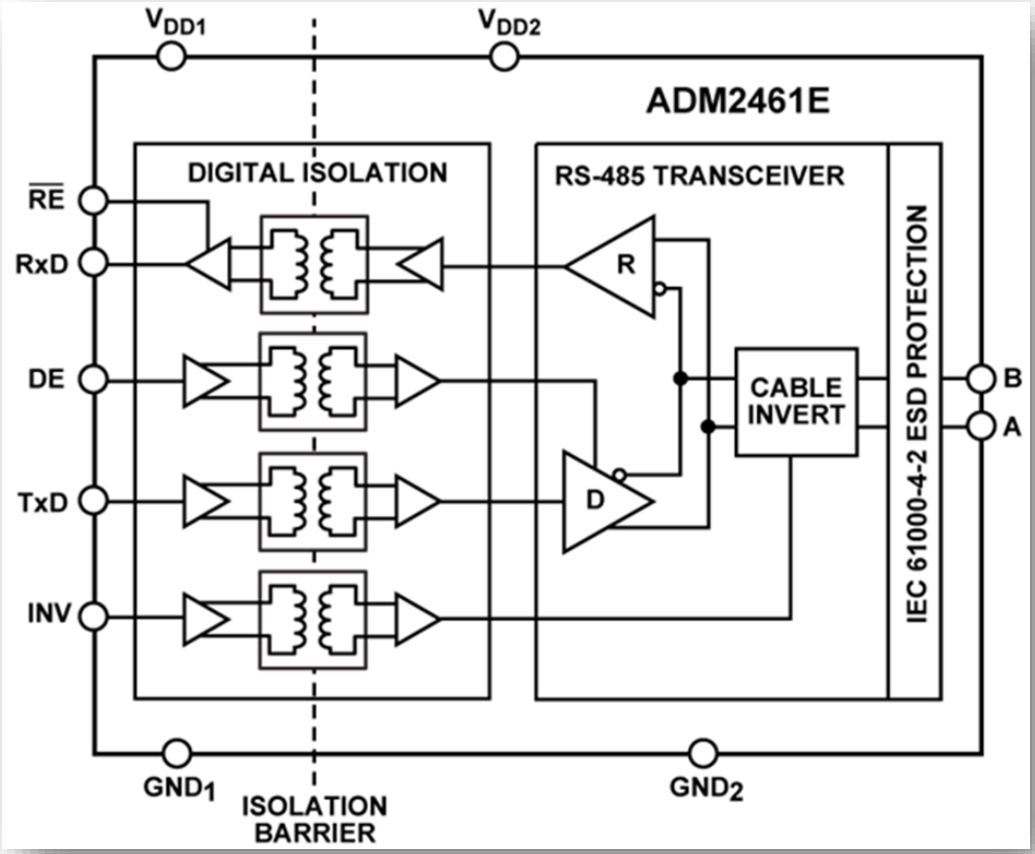


Рисунок 5 – Інтегральна мікросхема перетворювача RS-485 ADM2461E

ADM2461E. 500 kbps, 5.7 kV RMS, Signal Isolated Half Duplex **RS-485** Transceiver with ±15 kV IEC ESD

4. Узгодження та конфігурація лінії зв'язку RS-485

При великих відстанях між пристроями, пов'язаними по кручений парі і високих швидкостях передачі починають проявлятися так звані ефекти довгих ліній.

Причина цього – скінченість швидкості поширення електромагнітних хвиль у провідниках. Швидкість ця істотно менша за швидкість світла у вакуумі і становить трохи більше 200 мм/нс. Електричний сигнал має також властивість відбиватися від відкритих кінців лінії передачі та її відгалужень. Для коротких ліній та малих швидкостей передачі цей процес відбувається так швидко, що залишається непоміченим. Однак час реакції приймачів – десятки/сотні нс. У такому масштабі часу кілька десятків метрів електричний сигнал проходить не миттєво. І якщо відстань досить велика, фронт сигналу, що відбився в кінці лінії і повернувся назад, може спотворити поточний або наступний сигнал. У таких випадках необхідно якимось чином пригнічувати ефект відбиття.

Будь-яка лінія зв'язку має такий параметр, як хвильовий опір Zв. Воно залежить від характеристик кабелю, що використовується, але не від довжини.

Для зазвичай застосовуваних в якості ліній зв'язку кручених пар Zв = 120 Ом.

Виявляється, якщо на віддаленому кінці лінії, між провідниками кручений пари включити резистор з номіналом рівним хвильовому опору лінії (рисунок 6), то електромагнітна хвиля, яка дійшла до «глухого кута» поглинається на такому резисторі. Звідси його назви – резистор, що узгоджує, або «термінатор».

4. Узгодження та конфігурація лінії зв'язку RS-485

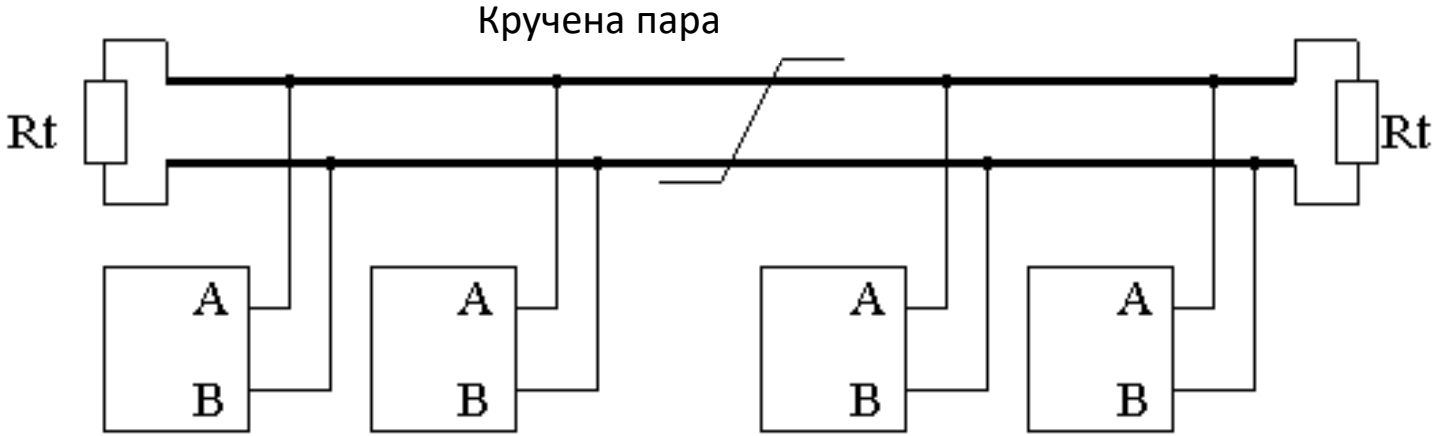


Рисунок 6 – Підключення передавачів-приймачів RS-485 до лінії зв’язку

Великий мінус узгодження на резисторах – підвищене споживання струму від передавача, адже до лінії включається низькоомне навантаження.

Тому рекомендується включати передавач тільки на час надсилання посилки. Однак такий спосіб має свої недоліки. Для коротких ліній (кілька десятків метрів) та низьких швидкостей (менше 38400 бод) погодження можна взагалі не робити.

Ефект відображення та необхідність правильного узгодження накладають обмеження на конфігурацію лінії зв'язку.

Лінія зв'язку повинна являти собою один кабель кручений пари.

4. Узгодження та конфігурація лінії зв'язку RS-485

До цього кабелю приєднуються всі приймачі та передавачі. Відстань від лінії до ІС інтерфейсу RS-485 має бути якомога коротшою, тому що довгі відгалуження вносять неузгодженість і викликають відображення.

В обидва найбільш віддалені кінці кабелю (Zв = 120 Ом) включають узгоджувальні резистори Rt по 120 Ом (0,25 Вт).

Якщо в системі тільки один передавач і він знаходиться в кінці лінії, достатньо одного узгоджуючого резистора на протилежному кінці лінії.

Як згадувалося, приймачі більшості мікросхем RS-485 мають пороговий діапазон розпізнавання сигналу на входах А-В - ±200мВ.

Якщо $|U_{AB}|$ менше порогового (близько 0), то на виході приймача RO можуть бути довільні логічні рівні через несинфазну перешкоду.

Таке може статися або при від'єднанні приймача від лінії, або за відсутності в лінії активних передавачів, коли ніхто не задає рівень. Щоб у цих ситуаціях уникнути видачі помилкових сигналів на приймач UART, необхідно на входах А-В гарантувати різницю потенціалів Uab > 200 мВ.

Це зміщення за відсутності вхідних сигналів забезпечує на виході приймача логічну «1», підтримуючи таким чином рівень стопового біта.

Досягти цього просто – прямий вхід (А) слід підтягнути до живлення, а інверсний (В) - до «землі» (рисунок 7).

5. Захисне зміщення в RS-485

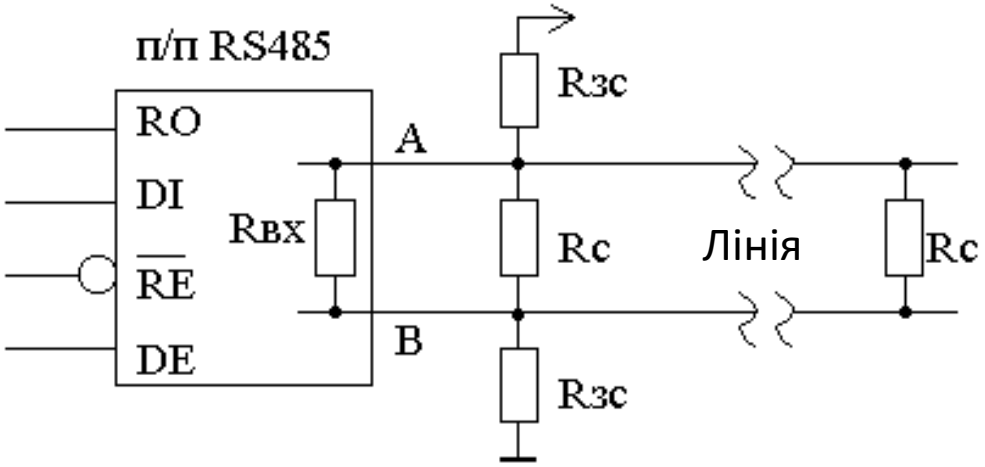


Рисунок 7 – Захист видачі помилкових сигналів в лінії зв'язку RS-485

Виходить дільник Rвх - вхідний опір приймача (зазвичай 12 кОм); Rс - узгоджувальні резистори (120 Ом); Rзс - резистори захисного зміщення. Величини опорів для резисторів захисного зміщення (Rзс) розраховується по дільнику.

Необхідно забезпечити U_{AB} > 200 мВ. Напруга живлення – 5В. Опір середнього плеча - 120Ом//120Ом//12КОм на кожен приймач - приблизно 57 Ом (для 10 приймачів). Таким чином, виходить приблизно по 650 Ом на кожен із двох Rзс. Для зміщення із запасом - опір Rзс має бути менше 650 Ом. Традиційно ставлять 560 Ом.

У розрахунку номіналу Rзс враховується навантаження. Якщо в лінії висить багато приймачів, то номінал Rзс повинен бути менше. У довгих лініях передачі необхідно також враховувати опір кручений пари, який може «з'їдати» частину різниці потенціалів, що зміщуєтьсі, для віддалених від місця підтяжки пристроїв. Для довгої лінії краще ставити два комплекти підтягуючих резисторів в обидва віддалені кінці поруч із термінаторами.

6. Задачі для розв’язання за темою

Задача 1. Вкажіть вірні топології мережі на основі інтерфейсу RS-485 (квадратиками позначені пристрої с інтерфейсом RS-485).

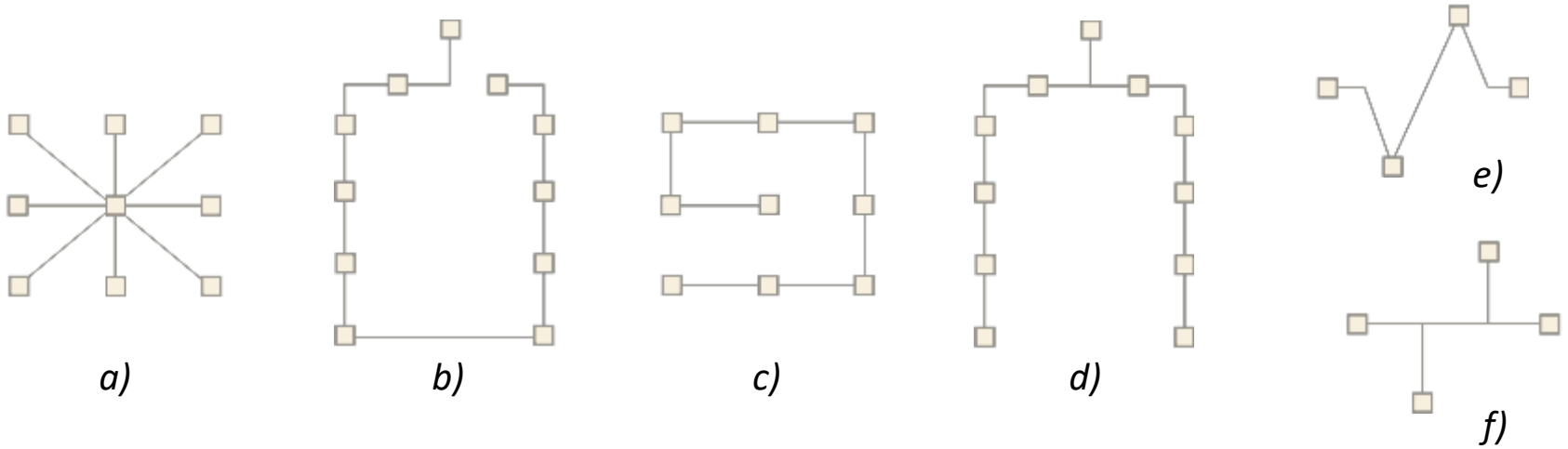


Рисунок 8 – Варіанти топології мережі

Задача 2. Наведіть результуючи часову діаграму на крученій парій інтерфейсу RS-485, якщо потенціал на лініях А та В виглядають так, як рисунку 9.

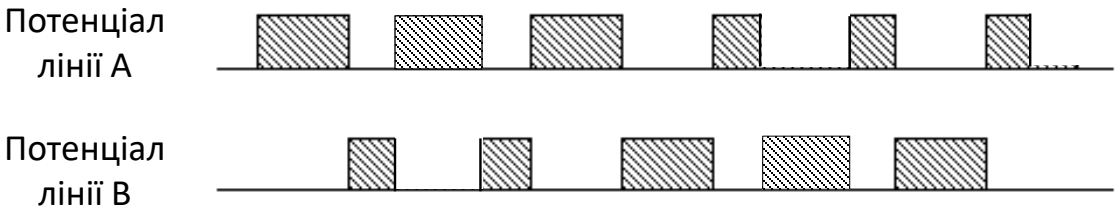


Рисунок 9 – Потенціали на лініях RS-485

6. Задачі для розв’язання за темою

Задача 3. Зробіть необхідні двохпровідні з’єднання пристроїв Master, Slave 1, Slave 2 і Slave 3 на основі інтерфейсу RS-485 (рисунок 10). Приведіть переваги і недоліки.

Рисунок 10 – Двохпровідне підключення RS-485

Задача 4. Зробіть необхідні чотирьохпровідні з’єднання пристроїв Master, Slave 1 і Slave 2 на основі інтерфейсу RS-485 (рисунок 11). Приведіть переваги і недоліки.

Рисунок 11 – Чотирьохпровідне підключення RS-485

Практичне заняття 7.
Аналіз режимів роботи SPI інтерфейсу

1. Протокол передачі SPI інтерфейсу.
2. Часові діаграми сигналів SPI інтерфейсу.
3. Осцилограми сигналів SPI інтерфейсу.
4. Задачі для розв’язання за темою.

1. Протокол передачі SPI інтерфейсу

Протокол передачі по інтерфейсу SPI ідентичний логіці роботи зсувного регістру, яка полягає у виконанні операції зсуву і, відповідно, побітного введення і виведення даних за певними фронтами сигналу синхронізації.

Установка даних при передачі і вибірка при прийомі завжди виконуються по протилежних фронтах синхронізації.

Це необхідно для гарантування вибірки даних після надійного їх встановлення. Якщо до цього врахувати, що в якості першого фронту в циклі передачі може виступати наростаючий або падаючий фронт, то

всього можливо чотири варіанти логіки роботи інтерфейсу SPI.

Ці варіанти отримали назву режимів SPI і описуються двома параметрами:

CPOL – вихідний рівень сигналу синхронізації (якщо CPOL = 0, то лінія синхронізації до початку циклу передачі і після його закінчення має низький рівень (тобто перший фронт наростаючий, а останній – падаючий), інакше, якщо CPOL = 1, - високий (тобто перший фронт падаючий, а останній - наростаючий));

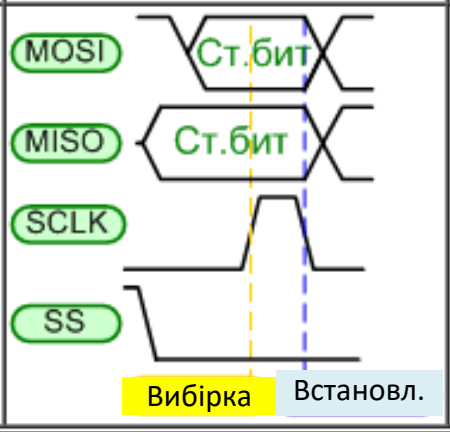
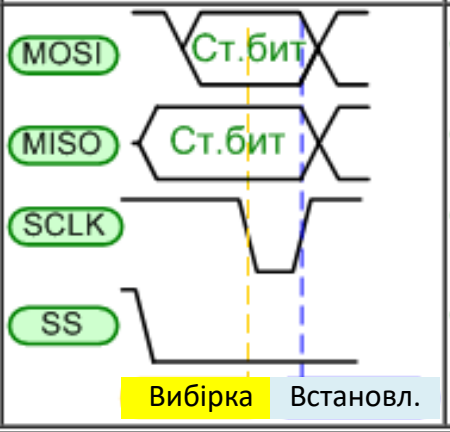
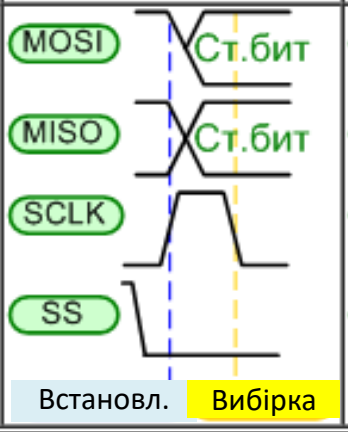
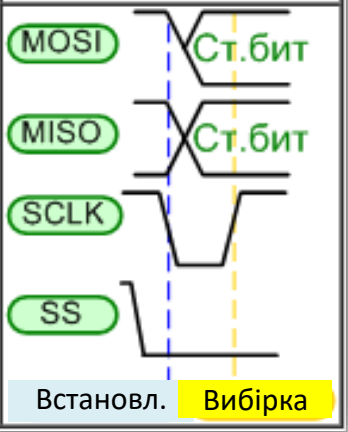
CPHA – фаза синхронізації; від цього параметра залежить, в якій послідовності виконується установка і вибірка даних (якщо CPHA = 0, то по передньому фронту в циклі синхронізації буде виконуватися вибірка даних, а потім, по задньому фронту, - установка даних; якщо ж CPHA = 1, то установка даних буде виконуватися по передньому фронту в циклі синхронізації, а вибірка – по задньому).

1. Протокол передачі SPI інтерфейсу

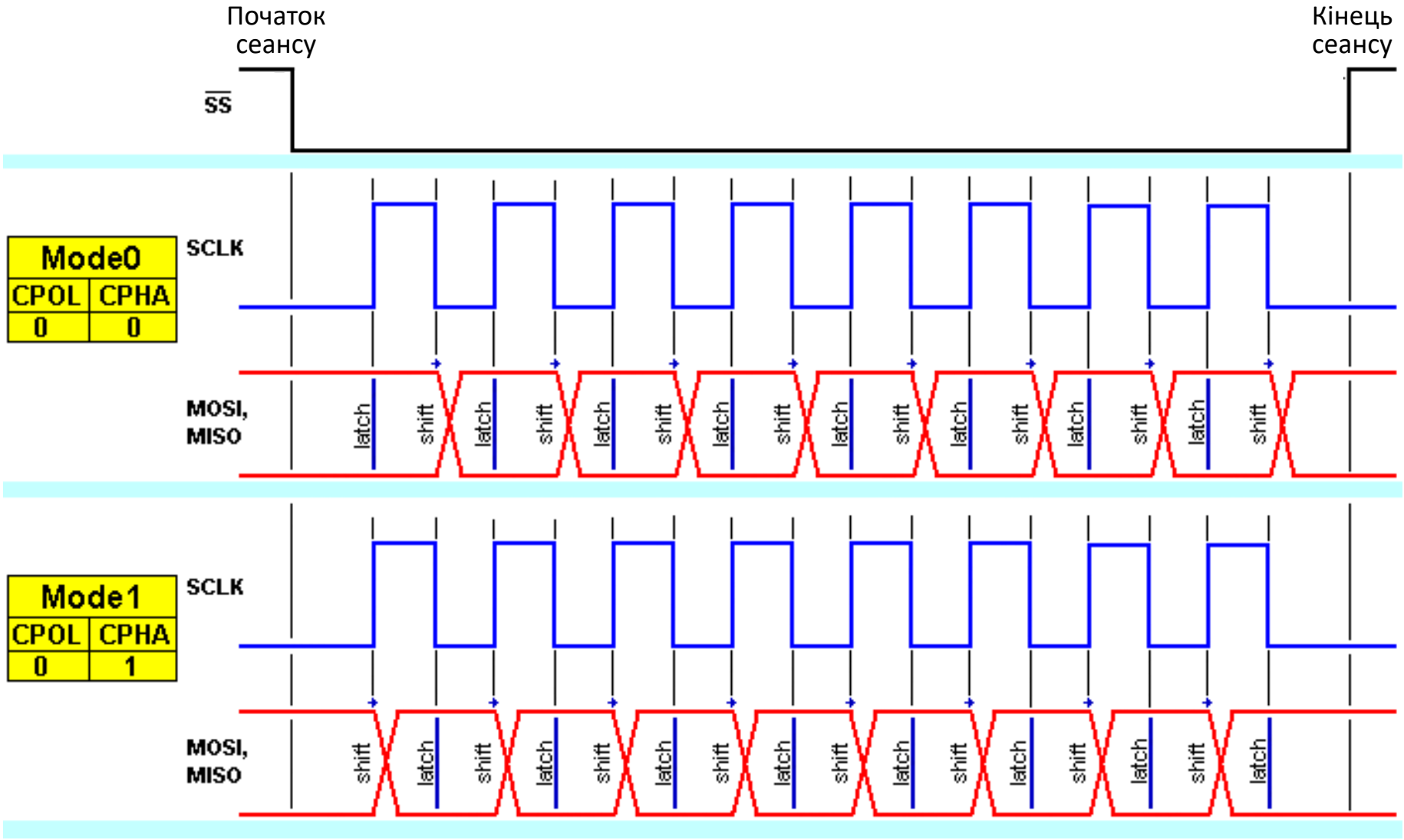
Таблиця 1 – Електричні сигнали шини SPI

Ведучий шини			Підлеглий шини		
Основне позначення	Альтернативне позначення	Опис	Основне позначення	Альтернативне позначення	Опис
MOSI	DO, SDO, DOUT	Вихід послідовної передачі даних	MOSI	DI, SDI, DIN	Вхід послідовного прийому даних
MISO	DI, SDI, DIN	Вхід послідовного прийому даних	MISO	DO, SDO, DOUT	Вихід послідовної передачі даних
SCLK	DCLOCK, CLK, SCK	Вихід синхронізації передачі даних	SCLK	DCLOCK, CLK, SCK	Вхід синхронізації прийому даних
SS	CS	Вихід вибору підлеглого (вибір мікросхеми)	SS	CS	Вхід вибору підлеглого (вибір мікросхеми)

Таблиця 2 – Режими SPI

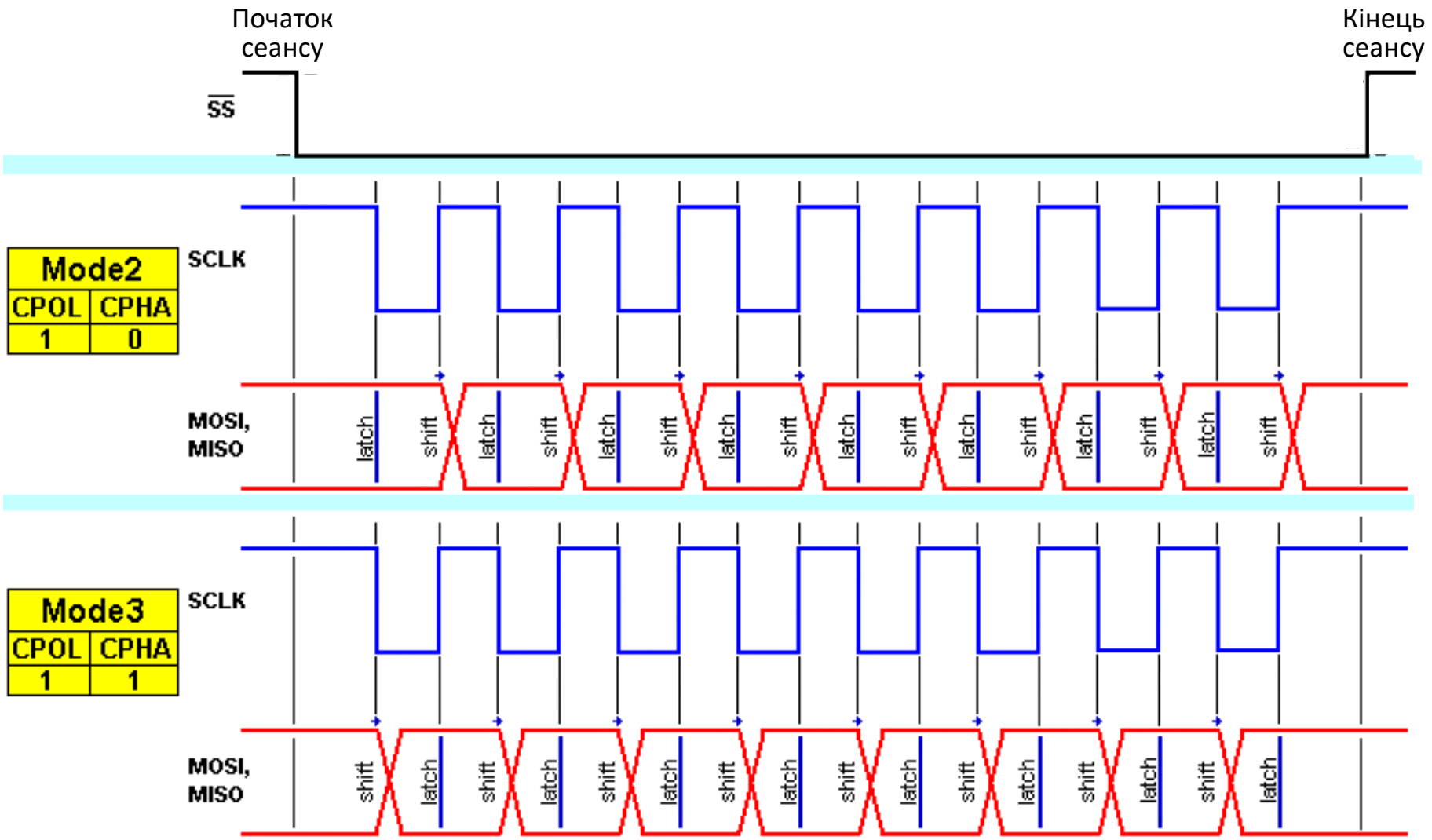
Режим SPI	0	1	2	3
CPOL	0	1	0	1
CPHA	0	0	1	1
Часова діаграма першого циклу синхронізації				

2. Часові діаграми сигналів SPI інтерфейсу



shift – зсув, latch – засувка, фіксація

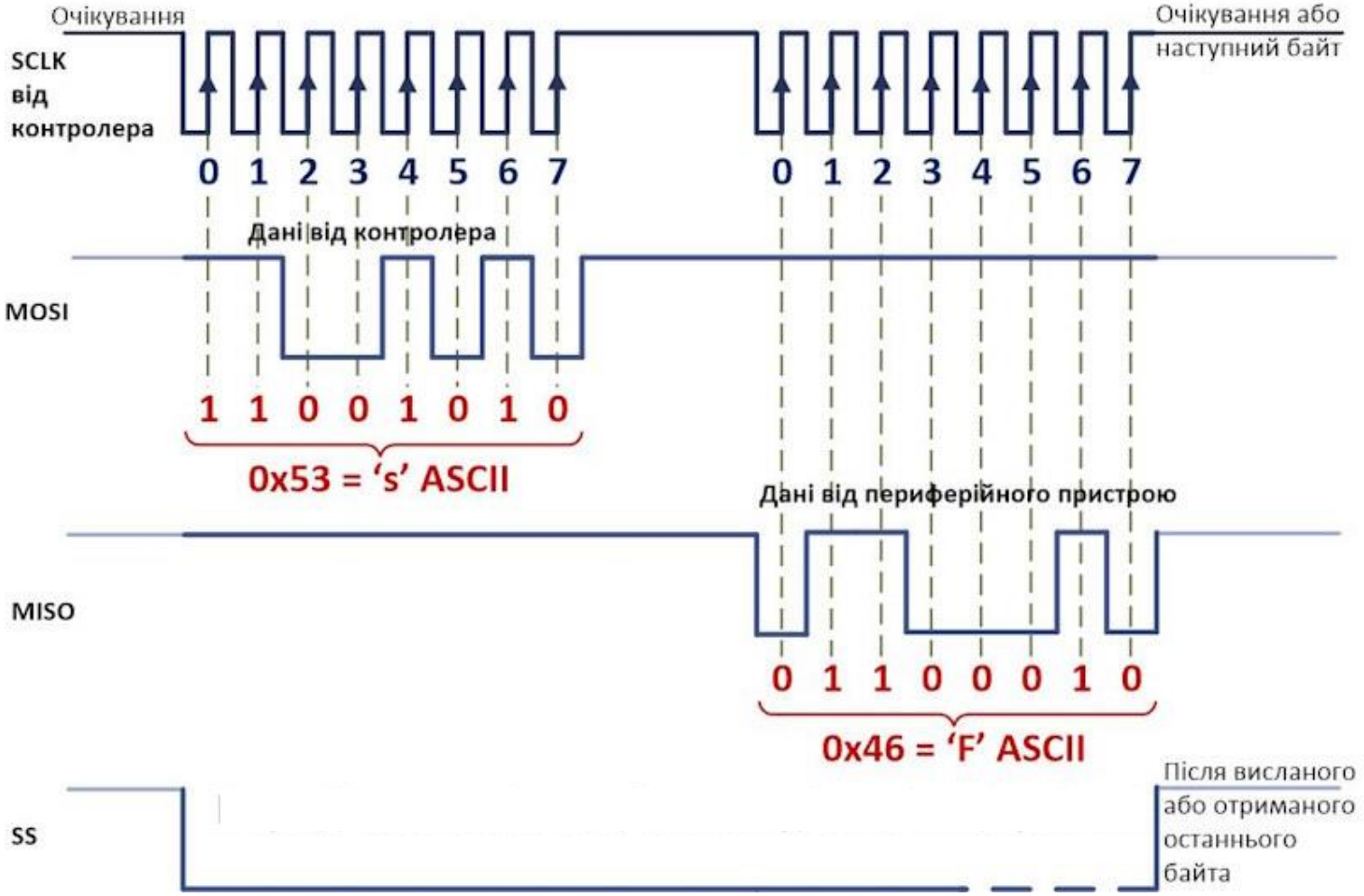
2. Часові діаграми сигналів SPI інтерфейсу



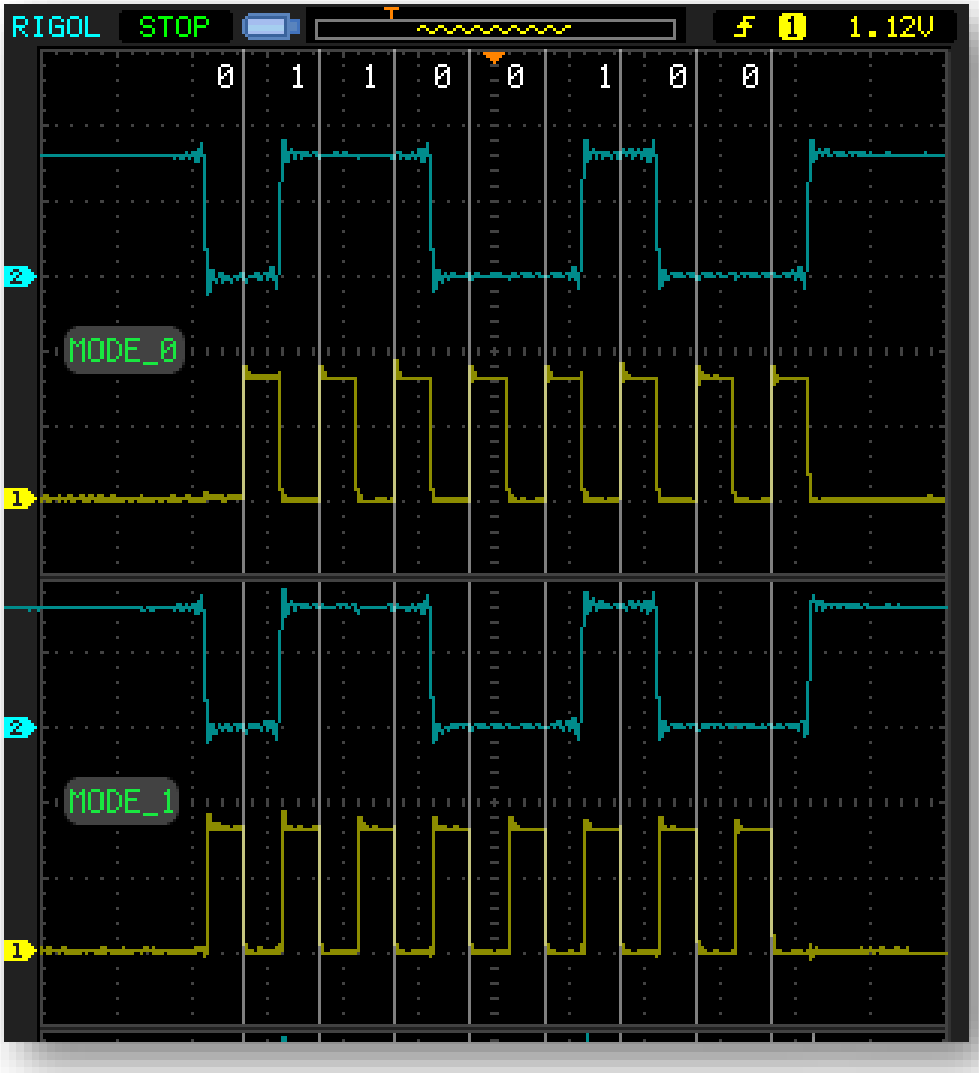
shift – зсув, latch – засувка, фіксація

2. Часові діаграми сигналів SPI інтерфейсу

У більшості випадків SPI працює шляхом передачі ведучим пристроєм команд, а периферійним пристроєм – відповідей. Нижче показана діаграма сигналів, де контролер висилає команду 0x53, а периферійний пристрій відсилає число 0x46 – відповідь на команду.

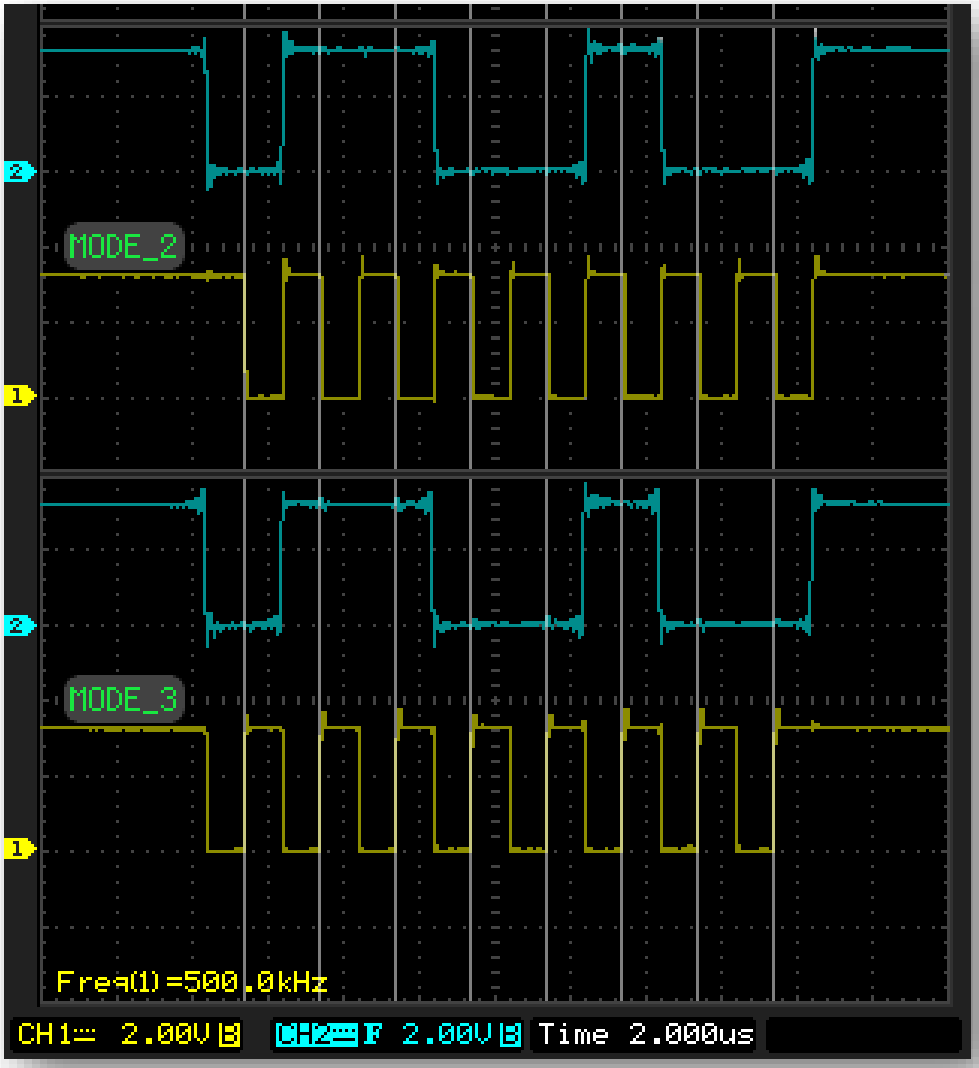


3. Осцилограми сигналів SPI інтерфейсу



Логічний рівень сигналу на шині тактування в неактивному стані (коли немає передачі даних) називають полярністю і позначають CPOL (тобто якщо за відсутності передачі на шині SCLK низький рівень, то CPOL=0, а якщо в цей час на шині SCLK високий рівень, то CPOL = 1).

3. Осцилограми сигналів SPI інтерфейсу



Порядок чергування зчитувань і зрушень називають фазою і позначають CPHA (якщо першим фронтом на SCLK відбувається зчитування, то CPHA=0, і якщо першому фронту на SCLK відбувається зсув, то CPHA=1). Залежно від поєднання значень CPOL і CPHA розрізняють 4 режими роботи інтерфейсу SPI, які і позначають Mode0, Mode1, Mode2 і Mode3.

3. Осцилограми сигналів SPI інтерфейсу

SS – це саме лінія управління сеансом обміну, а не просто лінія вибору slave.

Різниця тут у тому, що якщо вважати SS просто лінією вибору slave, то при підключенні master до одного єдиного slave виникає спокуса цією лінією не керувати, а жорстко закоротити її на загальний провід (типу, щоб slave завжди був обраний). Однак, логіка slave зазвичай така, що початок сеансу супроводжується різними підготовчими процедурами, такими як завантаження даних у вихідний зсувний регістр і скидання лічильника імпульсів, а виконувати якісь дії (відповідно до прийнятих SPI команд від master) slave починає тільки після завершення сеансу обміну. Крім того, може знадобитися кілька сеансів спілкування (наприклад, якщо в першому сеансі надсилаєте команди, а в наступному хочете отримати звіт про результат їх виконання).

Якщо жорстко притягнути лінію SS до загального дроту, то розпізнавання початку і кінця сеансу обміну (початок розпізнається по спаду на лінії SS, а кінець – по підйому) не відбудеться, відповідно весь обмін даними буде порушено . Тому важливість сигналу SS не варто недооцінювати.

Найбільш популярними є режими Mode0 і Mode3..

4. Задачі для розв’язання за темою

Задача 1. Відповідно до часової діаграми на рисунку 1 визначіть кодові послідовності, що передається та приймається пристроями SPI інтерфейсу. Для якого режиму обміну даними характерна приведена часова діаграма? В якому з чотирьох режимів працює SPI інтерфейс?

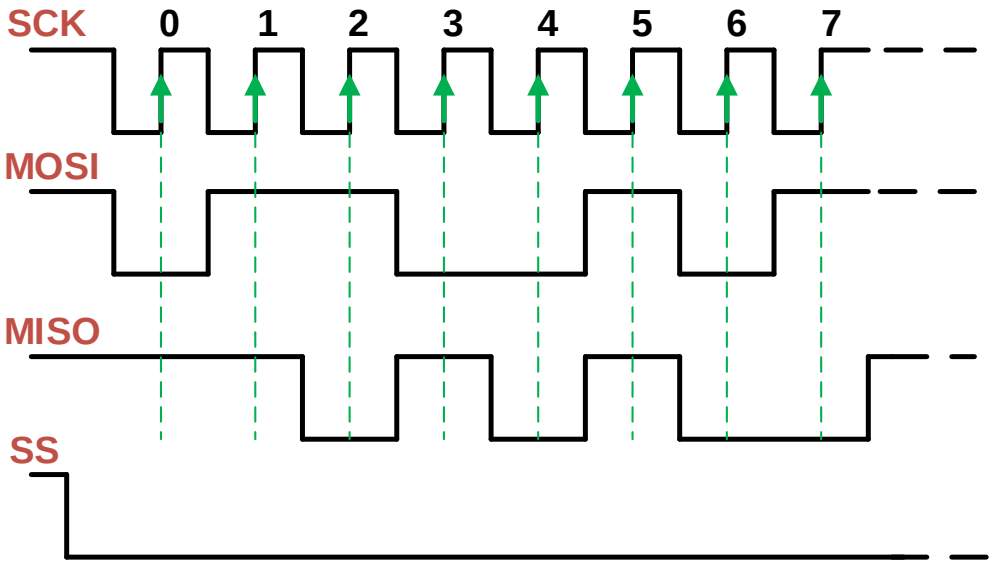


Рисунок 1 – Часова діаграма

Задача 2. Опишіть стисло достоїнства та недоліки SPI інтерфейсу.

4. Задачі для розв’язання за темою

Задача 3. Здійсніть з’єднання Master і Slave пристроїв за SPI інтерфейсом по незалежній схемі підключення (там, де це потрібно), вказуючи напрямки руху сигналів (рисунок 2).

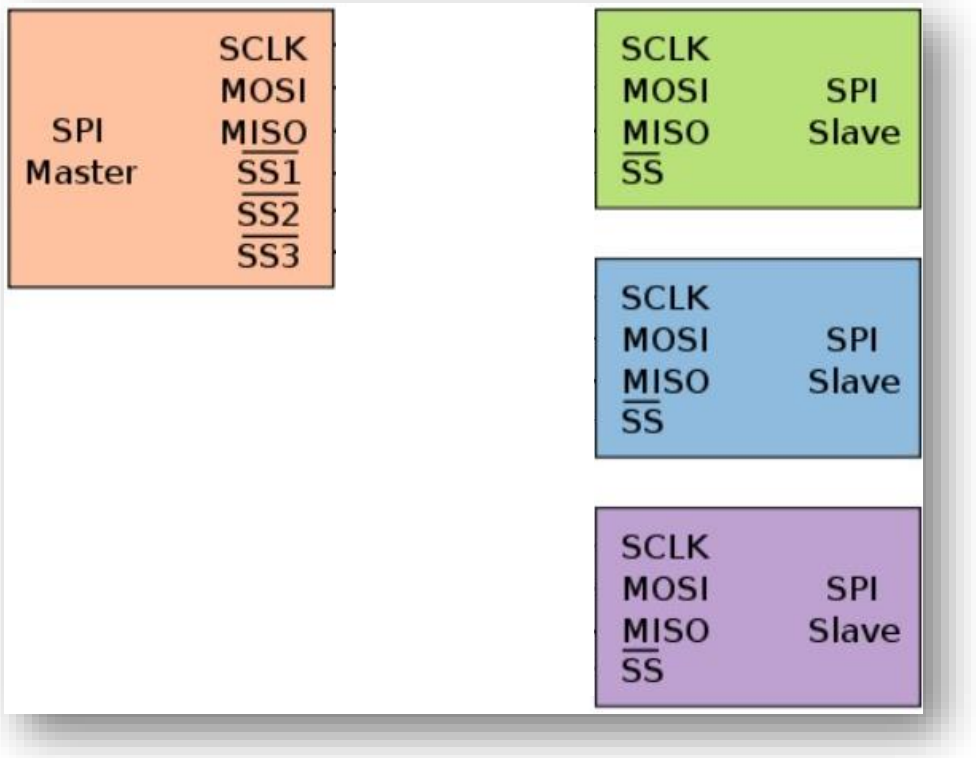
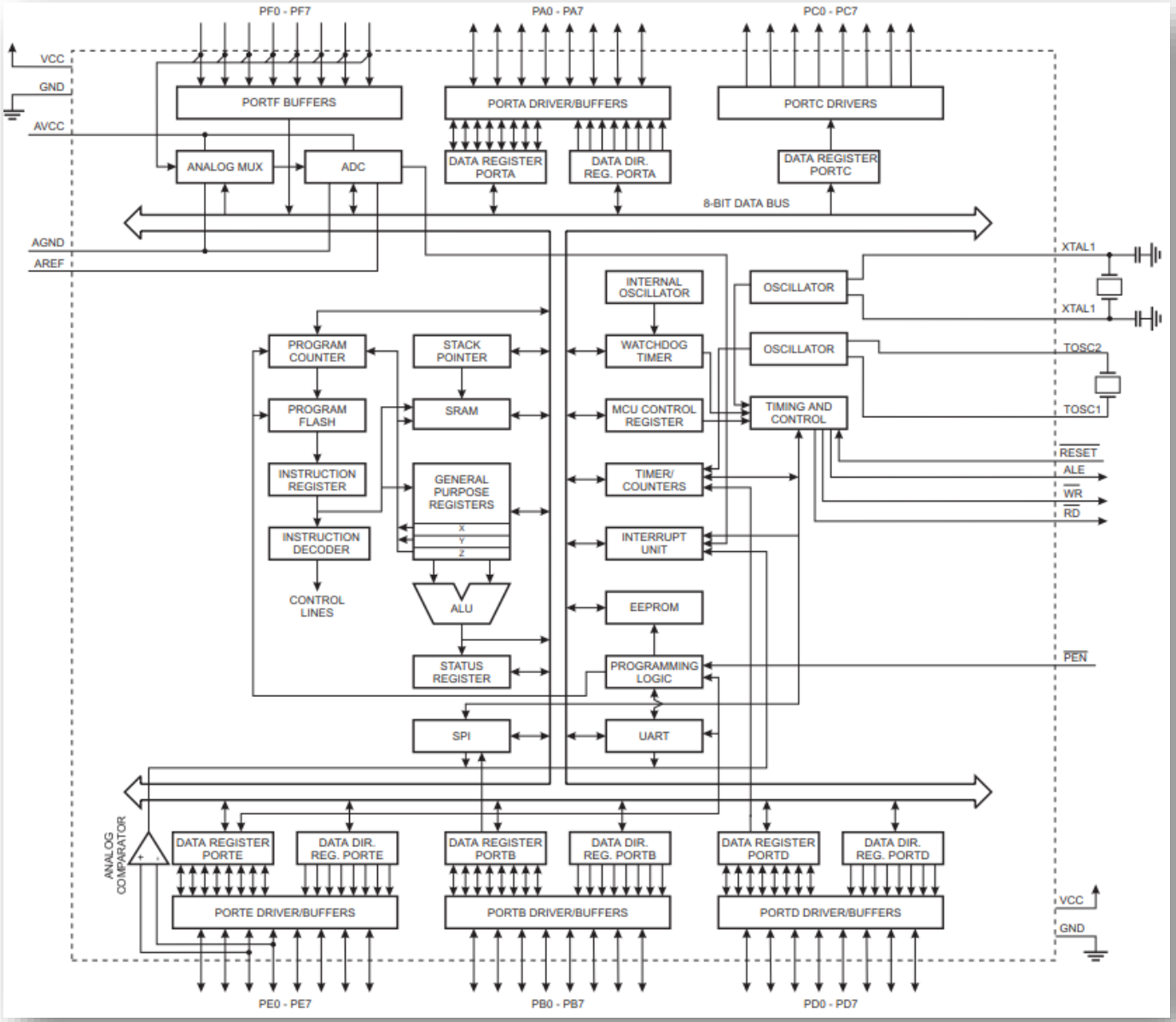


Рисунок 2 – З’єднання Master і Slave пристроїв по незалежній схемі підключення

Практичне заняття 8.
Практична реалізація SPI інтерфейсу на прикладі AVR ATmega 603/103

1. Загальна характеристика ATmega 603/103.
2. Основні характеристики SPI інтерфейсу.
3. З’єднання мікроконтролерів між собою.
4. Блок-схема SPI інтерфейсу.
5. Особливості функціонування SS виводів.
6. Формат регістру управління SPI.
7. Задачі для розв’язання за темою.

1. Загальна характеристика ATmega603/103



Блок-схема мікроконтролерів ATmega603/103

Тип прибо	Обсяг EEPROM	Обсяг SRAM
ATmega603	2 Кбайт	4 Кбайт
ATmega103	4 Кбайт	4 Кбайт

Тип прибора	Обсяг Flash пам’яті
ATmega603	64 Кбайт
ATmega103	128 Кбайт

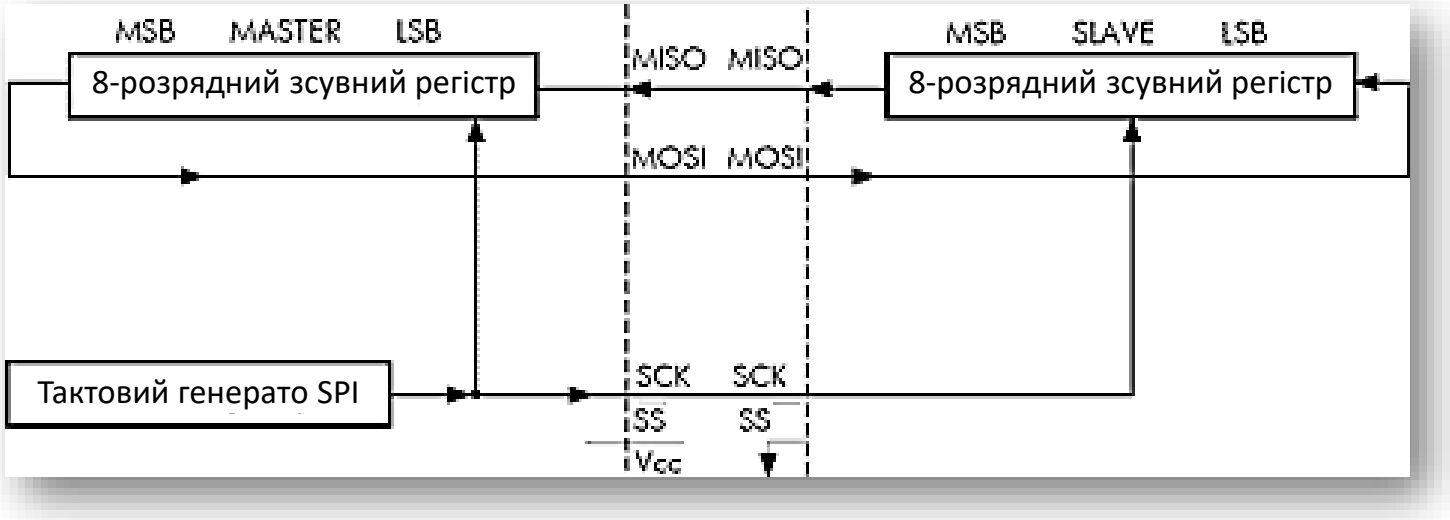
2. Основні характеристики SPI інтерфейсу

Послідовний периферійний інтерфейс (SPI) забезпечує високошвидкісний синхронний обмін даними між мікроконтролерами ATmega603/103 та периферійними пристроями або між декількома мікроконтролерами ATmega603/103.

Основні характеристики SPI інтерфейсу: *повнодуплексний 3-провідний синхронний обмін даними; режим роботи ведучий чи ведений; обмін даними з першими старшим або молодшим бітами; чотири програмовані швидкості обміну даними; прапор переривання після передачі; активація з режиму Idle (тільки в режимі керування).*

З'єднання між провідним і веденим CPU, що використовують інтерфейс SPI, показані на рисунку. Вивід PB1(SCK) є виходом тактового сигналу провідного мікроконтролера та входом тактового сигналу веденого. По запису провідним CPU даних в SPI регістр починає працювати тактовий генератор SPI і записані дані зсуваються через вивід виходу PB2 (MOSI) провідного МК на вивід входу PB2 (MOSI) веденого МК. Після зсуву одного байту тактовий генератор SPI зупиняється встановлюючи прапор закінчення передачі (SPIF). Якщо в регістрі SPCR буде встановлений біт дозволу переривання SPI (SPIE), буде запит переривання. Вхід вибору веденого PB0 (SS), для вибору індивідуального SPI пристрою як веденого, встановлюється на низький рівень. При установці високого рівня виводу PB0 (SS) порт SPI деактивується і вивід PB2 (MOSI) може бути використаний як вивід входу. Режим ведучий/відомий може бути встановлений і програмним способом встановлення або очищення біта MSTR в регістрі управління SPI.

3. З’єднання мікроконтролерів між собою

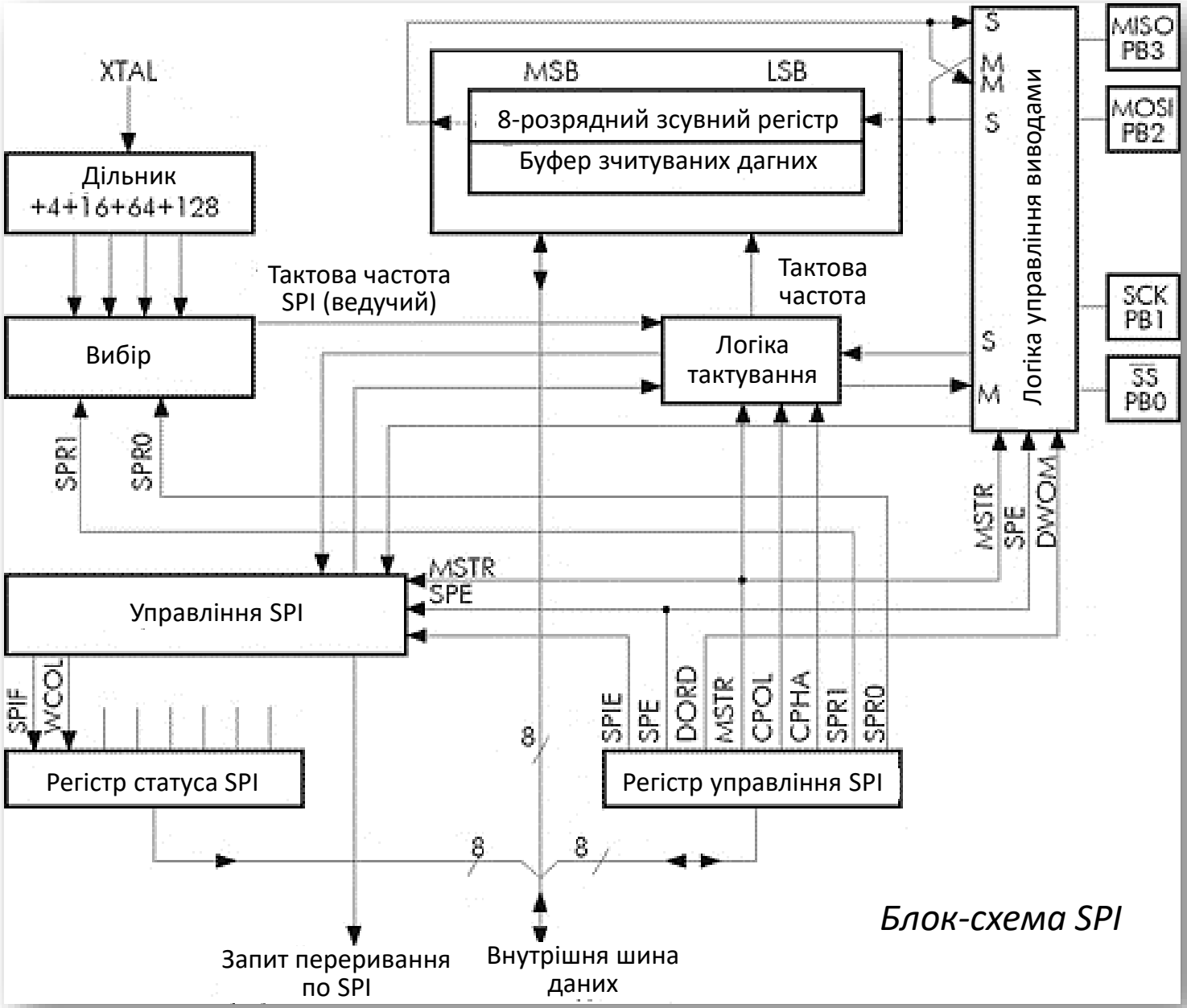


З'єднання між провідним і веденим CPU, що використовують інтерфейс SPI

Два зсувні регістри провідного та веденого МК можна розглядати як один рознесений 16-розрядний циклічний зсувний регістр. При зрушенні даних з ведучого мікроконтролера до веденого одночасно відбувається зсув даних з веденого МК до провідного, тобто протягом одного циклу зсуву відбувається обмін даними між провідним та веденим мікроконтролерами.

У системі організовано одиночне буферування передавальної сторони та подвійне буферування на приймальній стороні. Це означає те, що символи, що передаються, не можуть бути записані в регістр даних SPI перш, ніж буде повністю завершений цикл зсуву. З іншого боку, при прийомі даних символ, що приймається, повинен бути рахований з регістру даних SPI перш, ніж буде завершено прийом наступного символу, в іншому випадку попередній символ буде втрачений.

4. Блок-схема SPI інтерфейсу



Блок-схема SPI

5. Особливості функціонування SS виводів

У системі організовано одиночне буферування передавальної сторони та подвійне буферування на приймальній стороні. Це означає те, що символи, що передаються, не можуть бути записані в регістр даних SPI перш, ніж буде повністю завершений цикл зсуву. З іншого боку, при прийомі даних символ, що приймається, повинен бути вивантажений з регістру даних SPI перш, ніж буде завершено прийом наступного символу, в іншому випадку попередній символ буде втрачений.

Під час роботи SPI провідним (біт MSTR регістру SPCR встановлено), користувач має можливість встановити напрямок роботи виводу SS. Якщо вивід SS налаштовано як вихід, то вивід є вивідом загального призначення і він не активується системою SPI. Якщо ж вивід SS налаштований як вхід, то для забезпечення робота провідного SPI він повинен утримуватись на високому рівні. Якщо, в режимі ведучого, вивід SS є входом і зовнішньою периферійною схемою нього поданий низький рівень, то SPI сприймає його як звернення іншого провідного SPI себе як до веденого. Щоб уникнути конфліктної ситуації на шині, система SPI виконує такі дії:

1. Біт MSTR у регістрі SPCR очищається і SPI система стає веденою. Результатом цього є те, що MOSI та SCK виводи стають входами.

2. Встановлюється прапор SPIF регістру SPSR і, якщо дозволено переривання SPI, розпочнеться виконання підпрограми обробки переривання.

6. Формат регістру управління SPI

Таким чином, коли керований перериванням передавальний SPI використовується у провідному режимі, і існує ймовірність подачі на вивід SS керуючого сигналу низького рівня, переривання має завжди перевіряти, чи встановлено ще біт MSTR. Якщо біт MSTR був очищений вибором режиму веденого, він повинен бути встановлений користувачем.

Якщо ж SPI працює у режимі веденого, вивід SS постійно працює на вхід. Якщо на вивід SS поданий низький рівень, то SPI активується і MISO, якщо визначено користувачем, стає виходом. Решта виводів є входами. Якщо вивід SS утримується на високому рівні, всі виводи є входами , SPI пасивний, що означає, що він не буде отримувати вхідних даних.

Регістр управління SPI - SPCR - (Control Register)

Біти	7	6	5	4	3	2	1	0	
$0D ($2D)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Читання/Запис	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Початковий стан	0	0	0	0	0	0	0	0	

6. Формат регістру управління SPI

Bit 7 – SPIE: SPI Interrupt Enable – дозвіл переривання SPI. *Установка біта SPIE в стан 1 призводить до встановлення біта SPIF регістру SPSR і, при дозволі глобального переривання, виконання переривання SPI.*

Bit 6 – SPE: SPI Enable – дозвіл SPI. *Встановлення біта SPE у стан 1 дозволяє підключення SS, MOSI, MISO і SCK до виводів PB4, PB5, PB6 та PB7.*

Bit 5 – DORD: Data Order – порядок даних. *При встановленому стані 1 біті DORD передача слова даних відбувається LSB вперед. При очищеному біті DORD першим передається MSB слова даних.*

Bit 4 – MSTR: Master/Slave Select – вибір режиму ведучий/відомий. *При встановленому стані 1 біті MSTR SPI працює у провідному режимі і при очищеному біті у веденому режимі. Якщо SS налаштований як вхід і на нього поданий низький рівень при встановленому MSTR, то MSTR буде скинутий і буде встановлено біт SPIF у регістрі SPSR. Щоб знову дозволити провідний режим SPI, користувач має встановити MSTR.*

Bit 3 – CPOL: Clock Polarity – полярність тактового сигналу. *SCK в режимі очікування знаходиться на високому рівні при встановленому стані 1 біті CPOL і на низькому рівні при скинутому біті CPOL.*

Bit 2 – CPHA: Clock Phase – фаза тактового сигналу.

Bits 1,0 – SPR1, SPR0: SPI Clock Rate Select 1 and 0 – вибір частоти тактового сигналу, біти 1 та 0. *Ці два біти керують частотою тактового сигналу приладу, що працює у провідному режимі. У веденому режимі стану бітів впливу не надають (00 – $f_{SCK} = f_{CL}/4$, 01 – $f_{SCK} = f_{CL}/16$, 10 – $f_{SCK} = f_{CL}/64$, 11 – $f_{SCK} = f_{CL}/128$).*

7. Задачі для розв’язання за темою

Задача 1. Відповідно до часової діаграми на рисунку 1 визначіть кодові послідовності, що передається та приймається пристроями SPI інтерфейсу. Для якого режиму обміну даними характерна приведена часова діаграма? В якому з чотирьох режимів працює SPI інтерфейс?

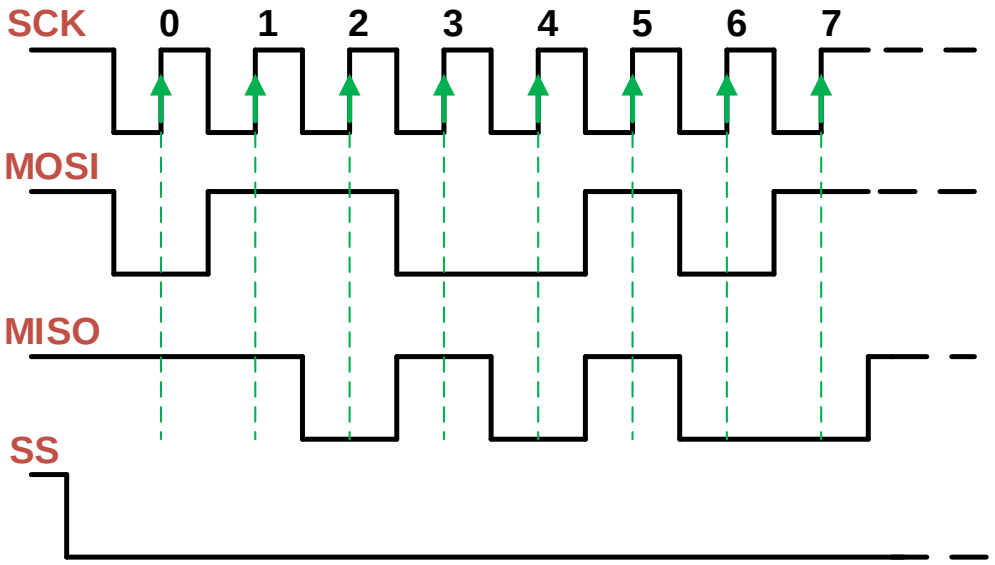


Рисунок 1 – Часова діаграма

Задача 2. Опишіть стисло достоїнства та недоліки SPI інтерфейсу.

4. Задачі для розв’язання за темою

Задача 3. Здійсніть з’єднання Master і Slave пристроїв за SPI інтерфейсом по незалежній схемі підключення (там, де це потрібно), вказуючи напрямки руху сигналів (рисунок 2).

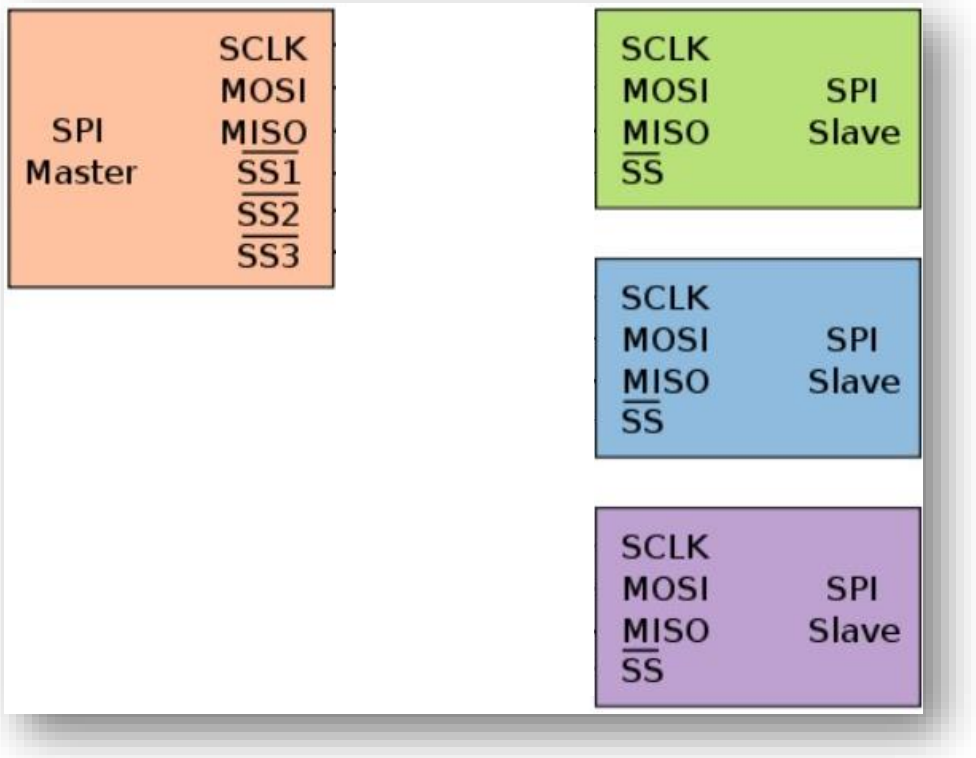
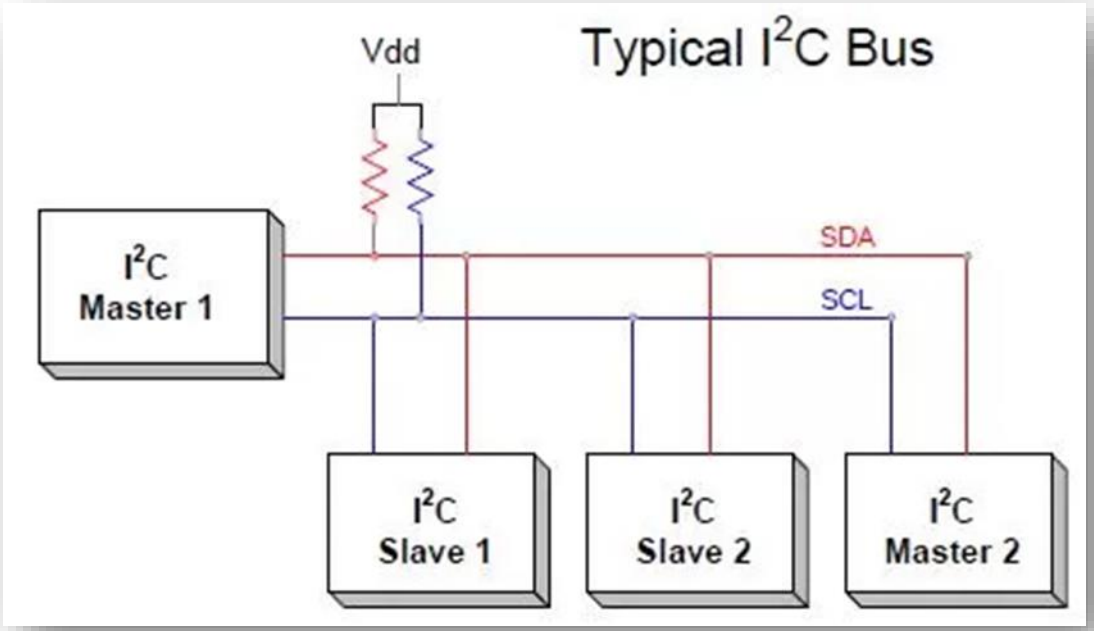


Рисунок 2 – З’єднання Master і Slave пристроїв по незалежній схемі підключення

Практичне заняття 9.
Аналіз часових діаграм на шині I^2C

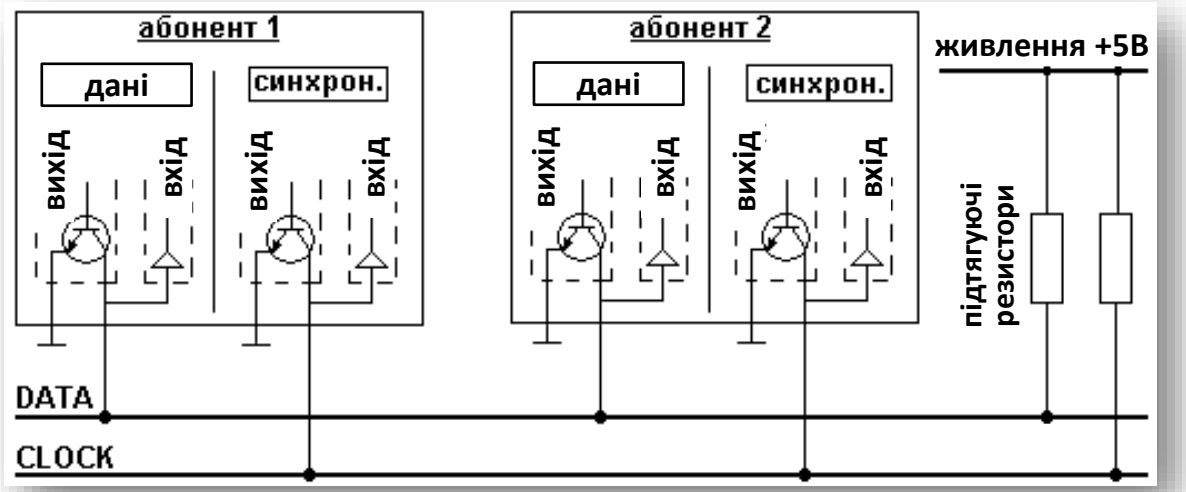
1. Схеми вихідних каскадів шини I^2C
2. Часові діаграми передачі по шині I^2C.
3. Часові діаграми приймання по шині I^2C.
4. Часові діаграми послідовності байтів по шині I^2C.
5. Види пакетів даних на шині I^2C.
6. Часова діаграма арбітражу по шині I^2C.
7. Задачі для розв’язання за темою.

1. Схеми вихідних каскадів шини I²C

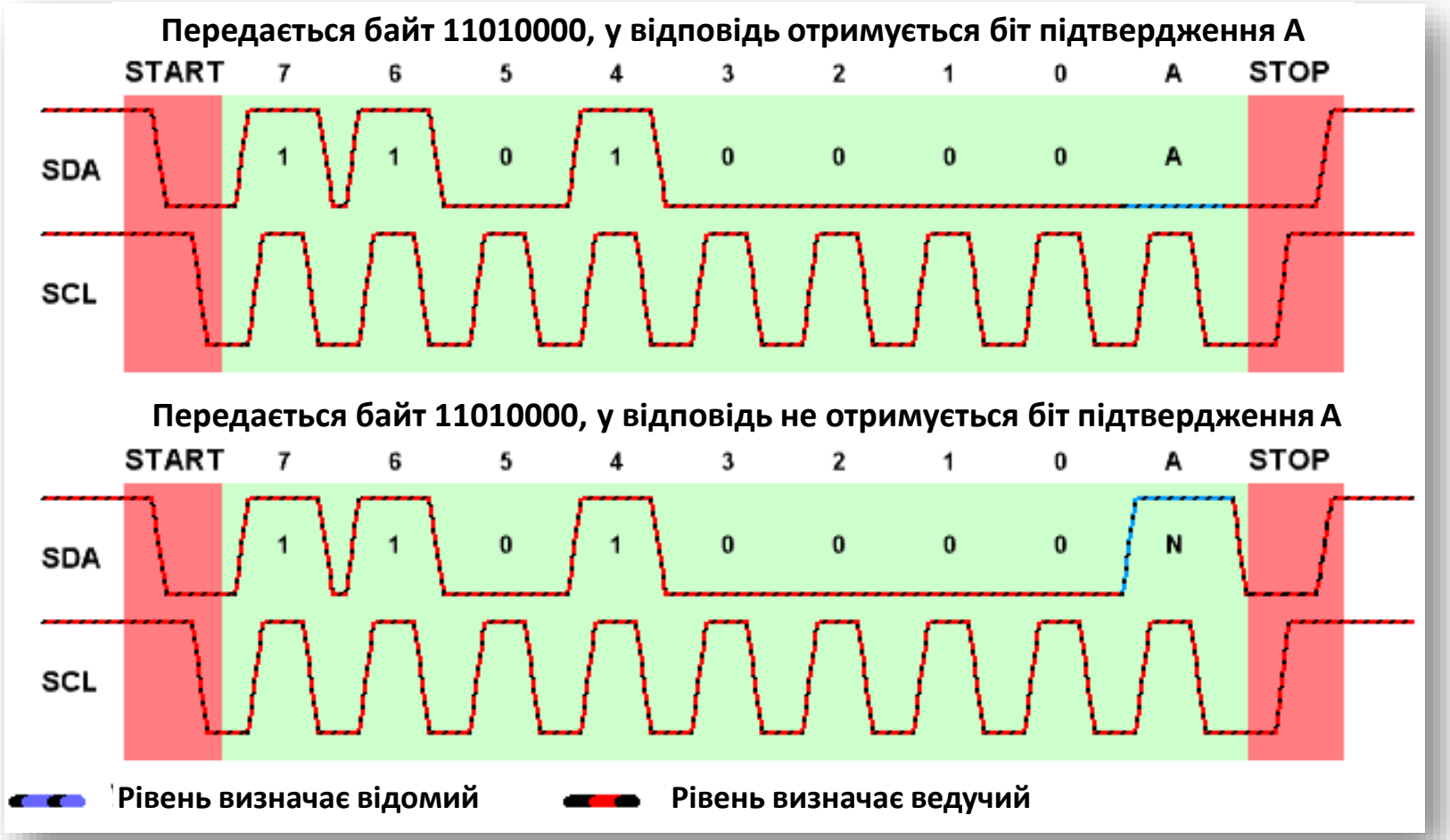


Підключення пристроїв до I²C шини

Схеми вихідних каскадів пристроїв, під'єднаних до I²C шини

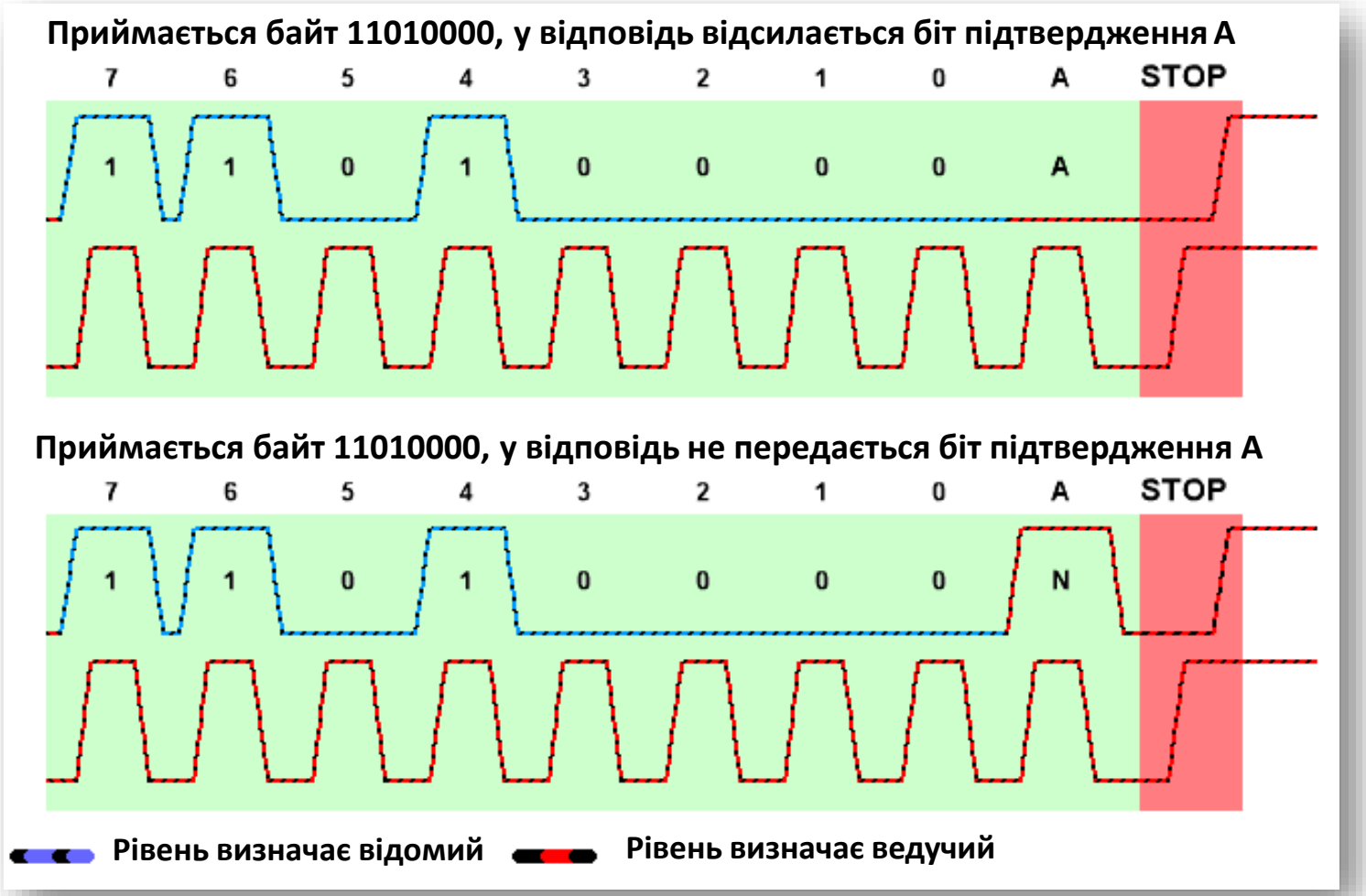


2. Часові діаграми передачі по шині I2C



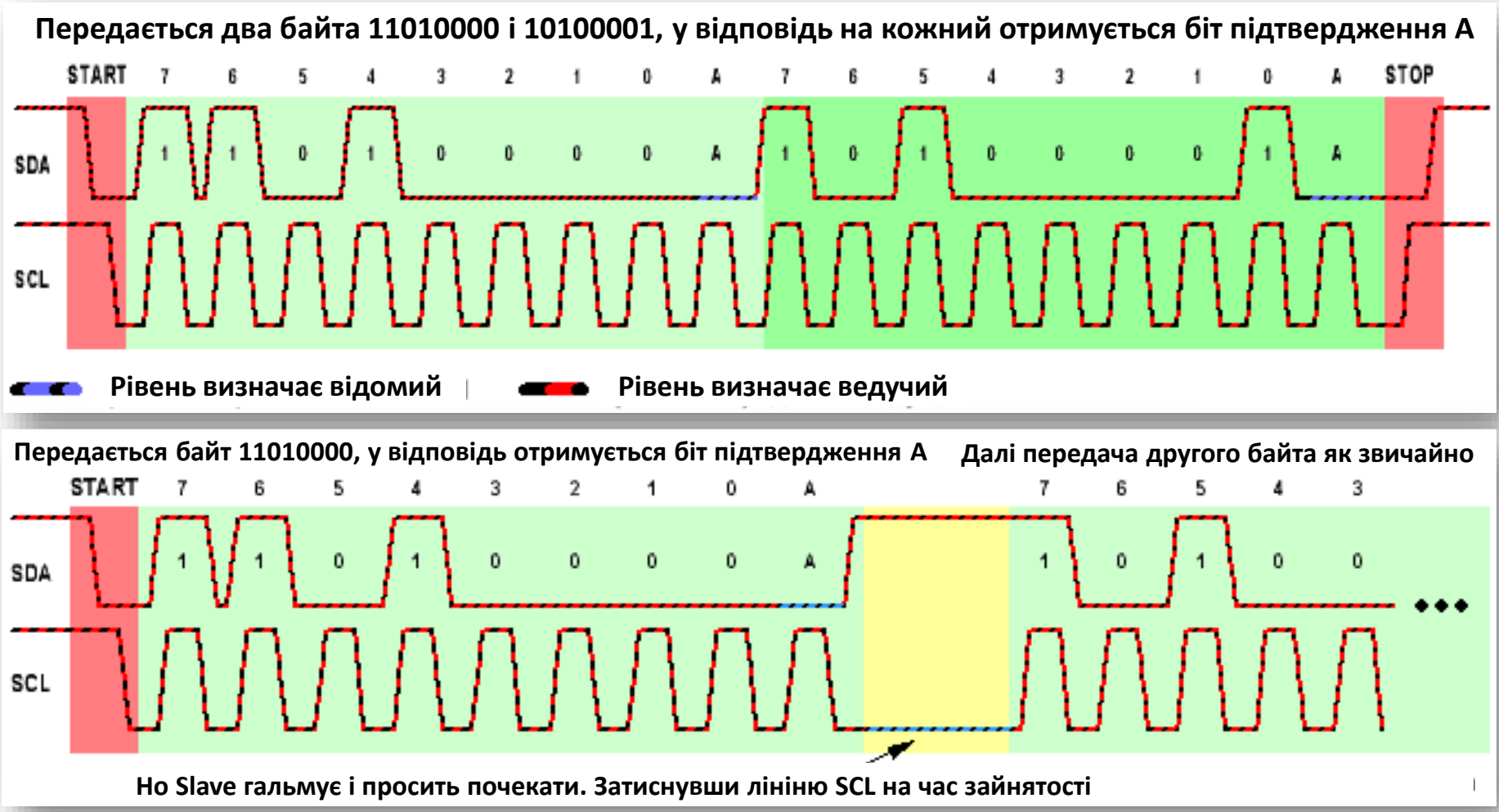
1. Початок передачі визначається Start послідовністю – провал SDA при високому рівні SCL.
2. При передачі від Master до Slave, ведучий генерує такти на SCL і видає біти на SDA, які ведений зчитує коли SCL стає 1.
3. При передачі інформації від Slave до Master, провідний генерує такти на SCL і дивиться, що там ведений творить з лінією SDA – зчитує дані. А ведений, коли SCL йде в 0, виставляє на SDA біт, який майстер зчитує, коли підніме SCL назад.
4. Закінчується STOP послідовністю, коли при високому рівні SCL лінія SDA переходить з 0 на 1.

3. Часові діаграми приймання по шині I²C



Зміна на шині даних у момент прийому даних може бути лише за низькому рівні SCL. Коли SCL вгорі, то йде читання. Якщо ж SDA змінюється за високим SCL, то це вже службові команди START або STOP.

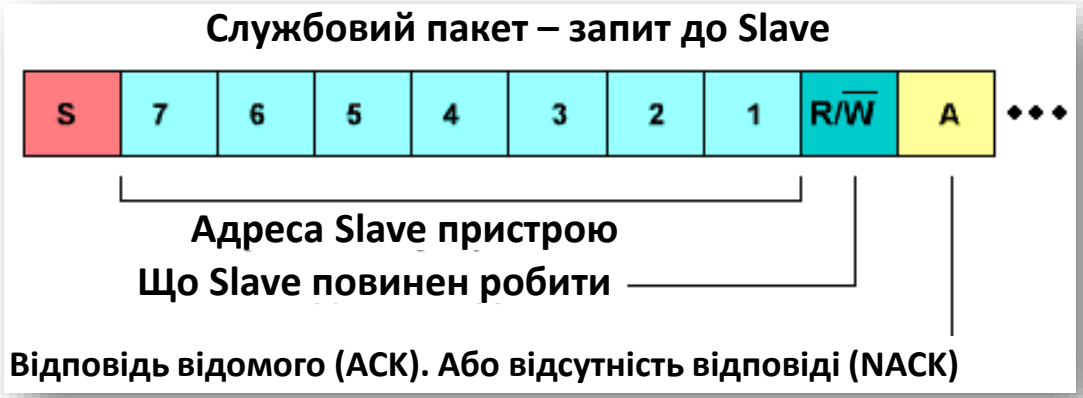
4. Часові діаграми послідовності байтів по шині I^2C



Якщо Slave повільний і не встигає (у EEPROM, наприклад, низька швидкість запису), він може насильно покласти лінію SCL в землю і не давати ведучому генерувати нові такти. Майстер повинен це зрозуміти і дати Slave прийняти байт. Так що не можна генерувати такти, при відпусканні SCL треба стежити, що лінія піднялася. Якщо не піднялася, то треба зупинитись і чекати доти, доки Slave її не відпустить. Потім продовжити з того самого місця.

5. Види пакетів даних на шині I²C

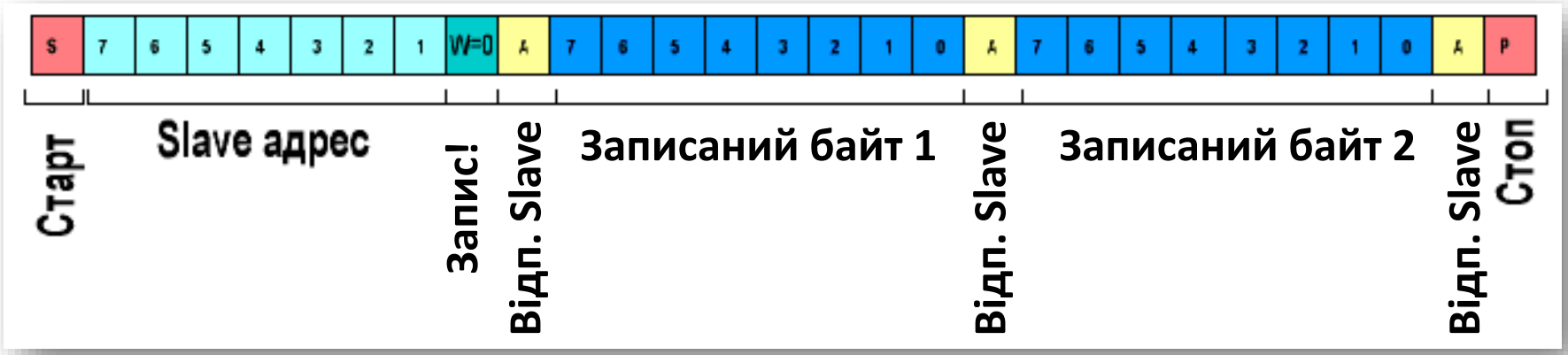
Дані надходять пакетами, кожен пакет складається з дев'яти біт: 8 даних та 1 біт підтвердження/не підтвердження прийому.
Перший пакет надсилається від ведучого до веденого – це фізична адреса пристрою і біт напрямку.



Сама адреса складається з семи бітів (чому до 127 пристроїв), а 8-й біт означає що робитиме Slave на наступному байті – приймати або передавати дані. 9-м бітом йде біт підтвердження ACK. Якщо Slave почув свою адресу та зчитав повністю, то на 9-му такті він притисне лінію SDA в 0, згенерувавши ACK – тобто – «Зрозумів!».

Майстер, помітивши це, розуміє, що все йде за планом, і можна продовжувати. Якщо Slave не виявився, проґавив адресу, неправильно прийняв байт, згорів або ще що з ним трапилося, то, відповідно, SDA на дев'ятому такті буде притиснути нікому і ACK не вийде. Буде NACK. Майстер припинить свої спроби до кращих часів.
Після адресного пакета йдуть пакети з даними у той чи інший бік, залежно від біту R/W у заголовному пакеті.

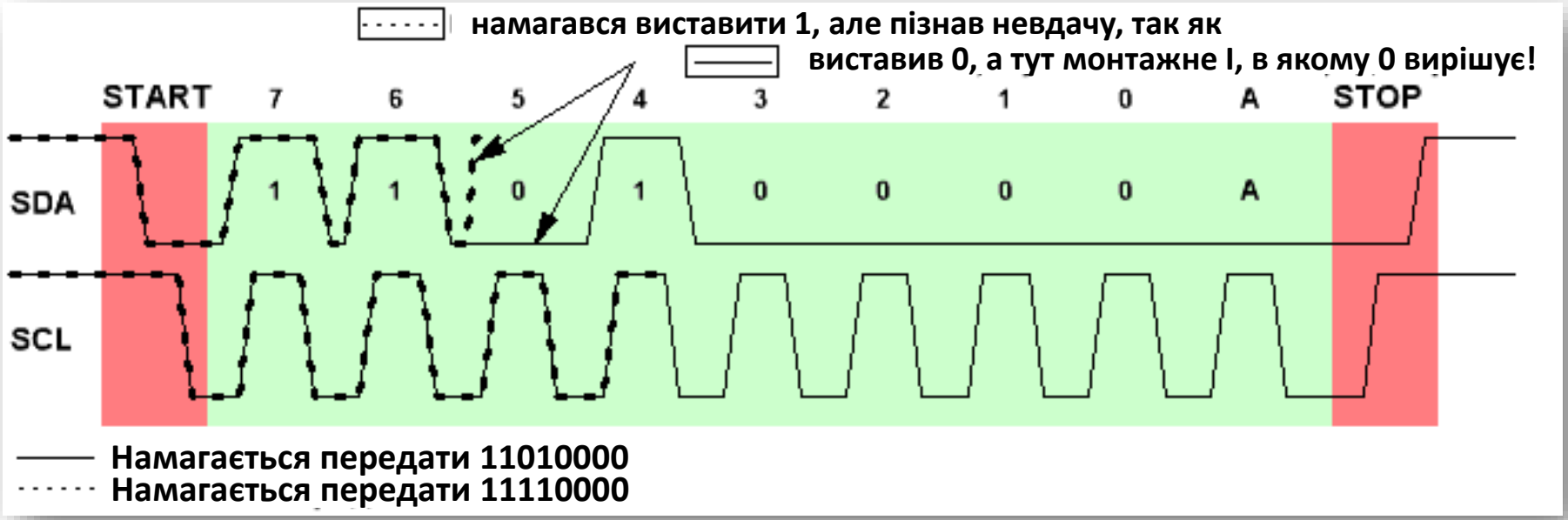
5. Види пакетів даних на шині I^2C



Читання практично теж саме, але один нюанс. При прийомі останнього байту треба дати відомому зрозуміти, що він більше не потрібен і відіслати NACK на останньому байті. Якщо надіслати ACK, то після стопу Master не відпустить лінію (R=1).

6. Часова діаграма арбітражу по шині I²C

Хто перший почав мовити – той поточний Master.



Неймовірне, але все ж таки можливе – два ведучі почали мовити одночасно. Тут допомагає властивість монтажного І, де проти нуля немає прийому. Обидва майстри біт за бітом грають у просту гру - ножик-камінь - 1 та 0 відповідно. Хто перший викине камінь проти ножа, той і перемагає арбітраж, продовжуючи мовити далі. Очевидно, що найважливіша адреса має починатися з нулів, щоб той, хто до нього намагався звертатися, завжди вигравав арбітраж. А сторона, що програла, змушена чекати поки шина не звільниться.

7. Задачі для розв’язання за темою

Задача 1. Відбувається передача даних від Master до Slave по шині I^2C (рисунок 1). Необхідно:

1) визначити за кожним номером часового інтервалу синхронізації, що здійснюється за цим проміжком часу на лінії даних;
2) відповісти, на яких часових інтервалах працює Master, а на яких Slave;
3) якого вигляду двійкова комбінація, яка передається першою;
4) які перші розряди за значенням наступної комбінації.

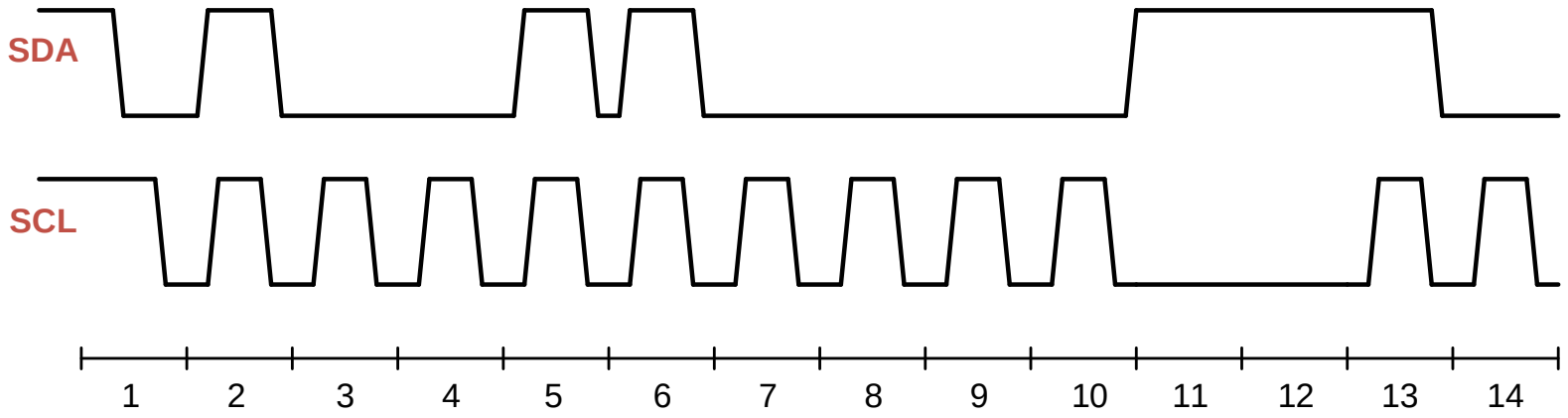


Рисунок 1 – Часова діаграма

Задача 2. Продемонструйте часову діаграму на шині I^2C у випадку приймання Master від Slave двійкової комбінації 00110110. Визначьте, за які часові інтервали відповідає Master, а за які Slave пристрій.

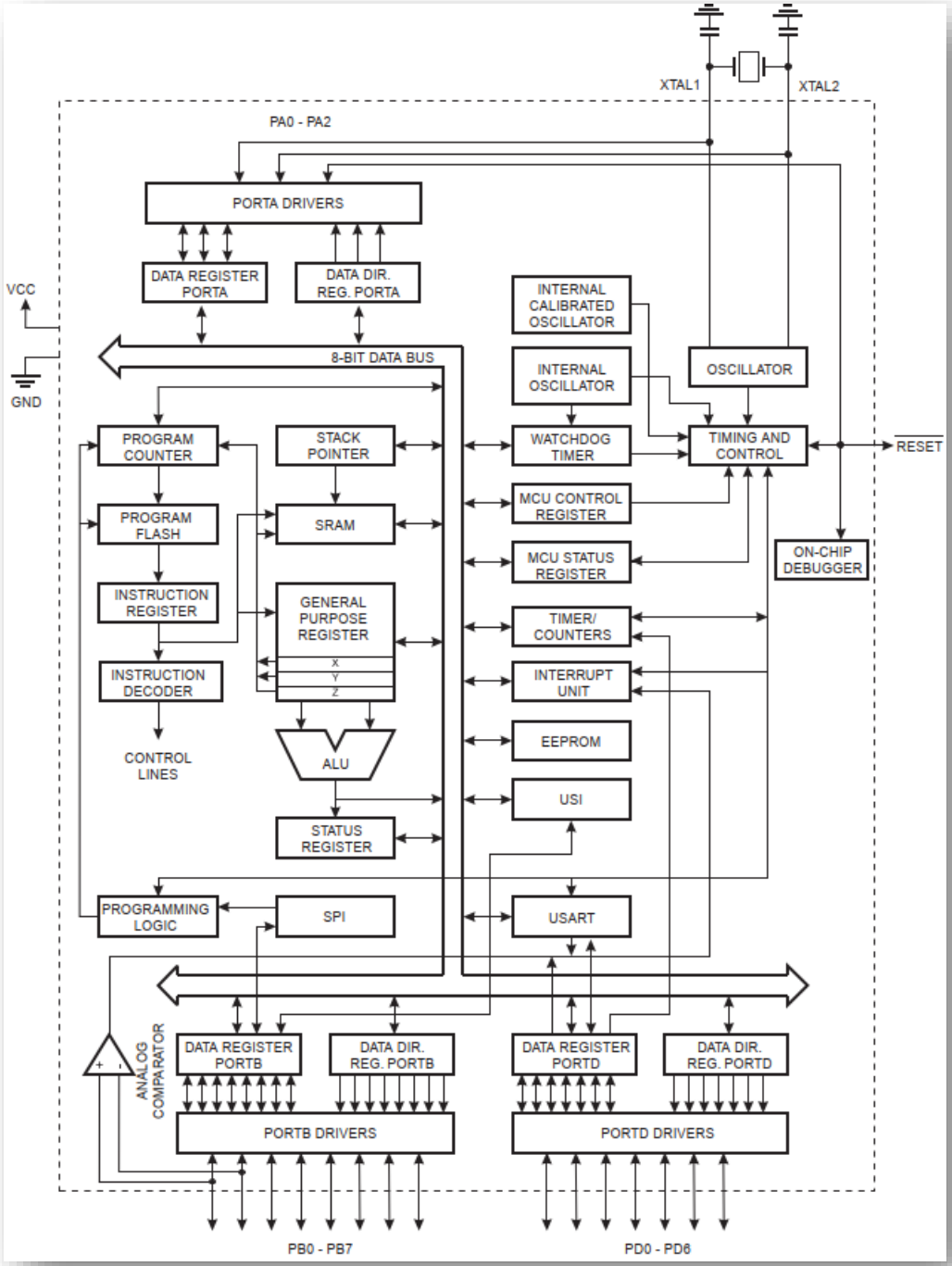
Практичне заняття 10. Розподілені мікроконтролерні системи на основі інтерфейсу USB (перша частина)

1. Практичний приклад реалізації інтерфейсу USB на мікроконтролері.
2. Задачі для розв’язання за темою.

1. Практичний приклад реалізації інтерфейсу USB на мікроконтролері

Використовується мікроконтролер ATtiny2313, який не має вбудованого апаратного USB, але ресурси якого цілком дозволяють реалізувати низькошвидкісну версію інтерфейсу USB програмно.

The ATtiny2313 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATtiny2313 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.



Блок-схема мікроконтролера ATtiny2313

1. Практичний приклад реалізації інтерфейсу USB на мікроконтролері

Використовується мікроконтролер ATtiny2313, який не має вбудованого апаратного USB, але ресурси якого цілком дозволяють реалізувати низькошвидкісну версію інтерфейсу USB програмно.

Схема достатньо проста: три світлодіоди і USB (ці діоди по USB керуються), навіть у такій простій конструкції доведеться врахувати деякі практичні нюанси:

1) знадобляться 2 піна МК для підключення інформаційних ліній інтерфейсу USB (D+/D-) та ще 3 піна для світлодіодів;

2) на лініях D+/D- використовуються рівні 0/3,3В, відповідно, якщо живлення МК відрізняється від напруги 3,3В, то потрібно узгоджувати рівні на цих лініях з рівнями на контактах МК. Живитиметься контролер прямо від роз'єму USB (Vcc=+5В), так що питання узгодження рівнів є актуальним. Тут треба враховувати, що з Datasheet, під час живлення +5В, мінімальне напруга, яка буде сприйматися МК як сигнал високого рівня, дорівнює 0.6*Vcc, тобто +3В, що дозволяє не робити двонаправлений перетворювач рівнів;

3) хаб визначає тип підключеного USB пристрою (Low Speed або Full/High Speed) за наявності резистора, що підтягує, 1,5 кОм на одній з інформаційних ліній. У нашому випадку потрібно буде підтягнути до живлення лінію D-.

1. Практичний приклад реалізації інтерфейсу USB на мікроконтролері

4) потрібно визначитися з кварцовим резонатором. Кварцовий резонатор підійде такий, щоб тривалість одного біта USB-передачі була кратна тривалості одного такту мікроконтролера (і бажано, щоб тактів в цей біт містилося побільше, інакше будуть труднощі з встиганням обробляти біти, що приймаються). Швидкість передачі на Low Speed дорівнює 1,5 Мбіт/с, причому мегабіти та кілобіти тут не справжні (по 1024 кілобайти та 1024 біти), а десяткові (по 1000 кілобіт та 1000 біт). Отже знадобиться кварцовий резонатор, частота якого без залишку ділиться на 1,5. Наприклад, якщо взяти «кварц» на 12 МГц (12/1,5 = 8) або 18 МГц (18/1,5 = 12), то передача одного біта даних буде займати, відповідно, 8 або 12 тактів МК.

Ресурси обмежені, тому бажано вміти максимально швидко визначати початок активності на шині USB.

Стан спокою для LS пристроїв відповідає тривалому стану J, який для LS пристроїв відповідає сигналу Diff0 (на D- – високий рівень сигналу, D+ - низький). Значить початок активності можна визначити або по переходу лінії D- до низького рівня (задній фронт на D-), або переходу лінії D+ до високого рівня (передній фронт на D+).

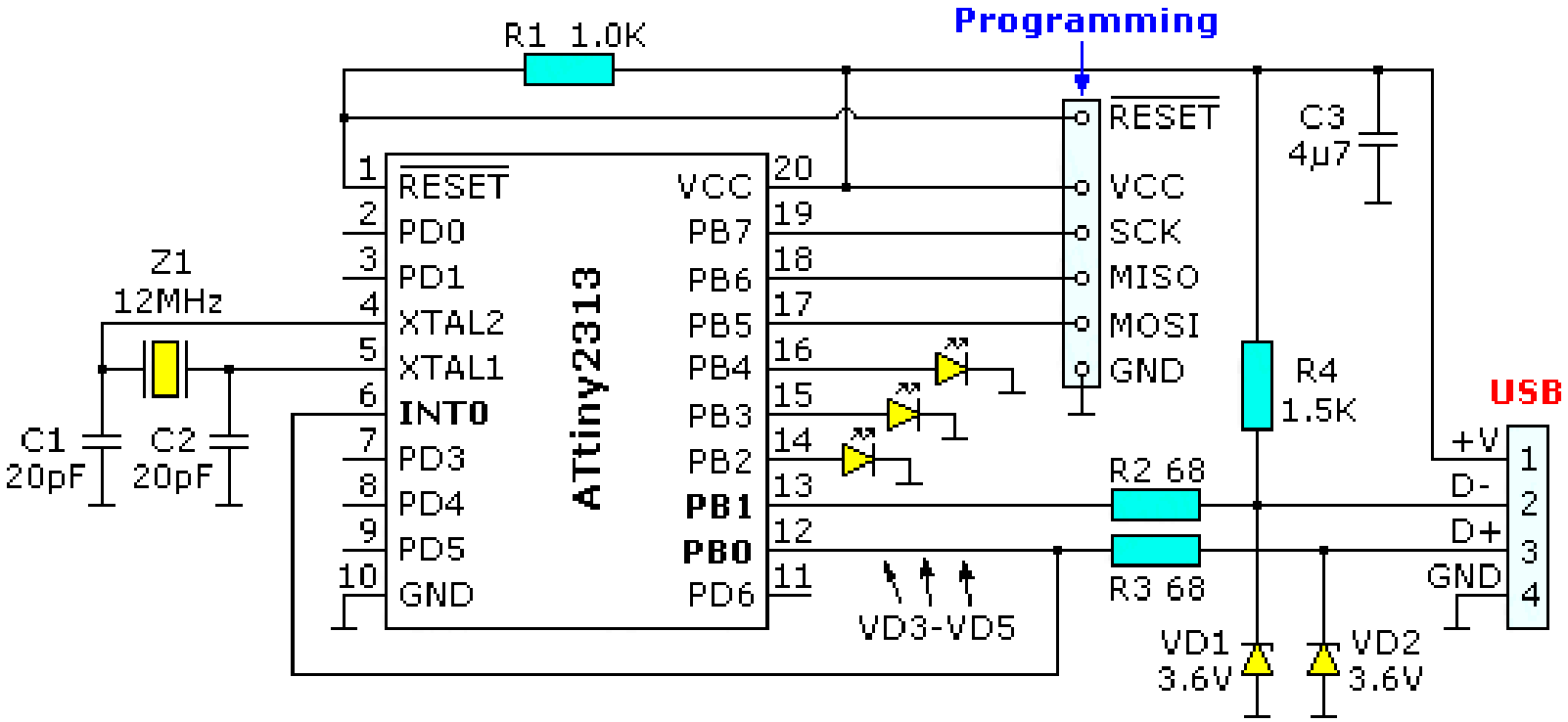
1. Практичний приклад реалізації інтерфейсу USB на мікроконтролері

Однак, якщо врахувати, що задній фронт на D- може говорити також і про перехід шини в стан SE0 (для LS-пристроїв перебування шини в цьому стані протягом 2 бітових інтервалів позначає кінець пакета, а тривале – disconnect), то найбільш переважним є варіант визначення початку активності переднього фронту на D+. А визначити цю активність якнайшвидше допоможе зовнішнє переривання INT0. Воно також, як і переривання Pin Change відбувається при зміні рівня на одному з контактів, проте, по-перше, INT0 прив'язане тільки до одного, строго певного піну контролера (не потрібно витрачати час, з'ясовуючи, з якого саме піну воно згенерувалося). По-друге, можемо вибрати, за яким фронтом (переднім або заднім це переривання генерувати), тобто для максимально швидкої реакції на початок активності потрібно завести лінію D+ на контакт INT0.

У другому варіанті зниження напругу живлення всього пристрою, замість того, щоб займатися рівнями напруги на окремих лініях. Зробити це можна як за допомогою інтегрального стабілізатора, типу LM1117-3.3, просто впаявши в ланцюг живлення послідовно пару-трійку звичайних діодів, оскільки струм у цій схемі зовсім невеликий.

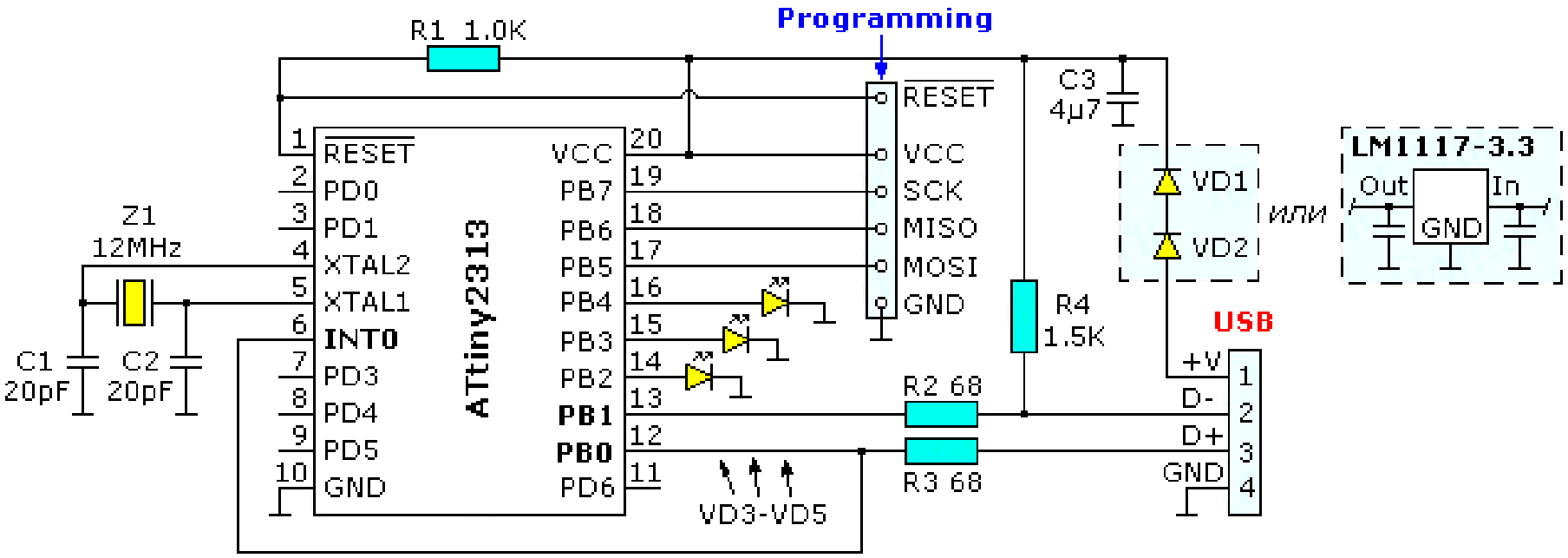
За даташитом, для деяких модифікацій ATtiny2313, частота 12 МГц перевищує максимально допустиму при живленні 3,3 В частоту. При цьому практика показує, що МК все одно нормально працює, але факт порушення заявлених характеристик в цьому випадку має бути.

1. Практичний приклад реалізації інтерфейсу USB на мікроконтролері



Варіант 1 схеми практичної реалізації

1. Практичний приклад реалізації інтерфейсу USB на мікроконтролері



Варіант 2 схеми практичної реалізації

1. Практичний приклад реалізації інтерфейсу USB на мікроконтролері

Технічні рішення:

1) інформаційні лінії будуть заводитися на контакти PB0 (D+) та PB1 (D-);

2) визначення початку передачі по передньому фронту на лінії D+ (у стані IDLE у нас на D+ нуль, а на D- одиниця), яку для максимально швидкої реакції додатково заведена на контакт INT0 (для якої можливо встановити переривання по передньому фронту);

3) тривалість одного інформаційного біта в нас становитиме 8 тактів контролера (кварцовий резонатор на 12 МГц);

5) пристрій керуватиме трьома світлодіодами, які підключені до контактів PB2, PB3, PB4.

Крім того, відомо, що:

1) всі пакети починаються з передачі синхропослідовності SYNC = 0b10000000,

2) всі дані, крім CRC, передаються молодшим бітом вперед (тобто SYNC передається у вигляді: 0b00000001),

3) дані передаються в NRZI-кодованому вигляді, тобто при передачі нуля стан шини (рівні на лініях D+/D-) змінюється протилежний, а передачі одиниці – залишається незмінним.

Продовження побудови розподіленої мікроконтролерної системи на основі USB інтерфейсі у другій частині практичного заняття

2. Задачі для розв’язання за темою

Задача 1. Яким чином огранізовується живлення мікроконтролера на першому та другому варіантах практичної реалізації схеми з інтерфейсом USB? Яка різниця в продемонстрованих підходах?

Задача 2. Як визначається початок передачі даних на шині USB у приведених прикладах реалізації мікроконтролерної схеми на основі USB? Якими функціональним та схемотехнічним рішенням вдається пришвидшити визначення активності на шині USB?

Задача 3. З передачі якої послідовності починається передачіа усіх пакетів на шині USB? Чому вона має саме такий вигляд та довжину?

Задача 4. Згідно схемотехнічній реалізації наведених варіантів практичної реалізації мікроконтролерного пристрою на основі інтерфейсу USB за яким інфтерфейсом відбувається програмування мікроконтролера і чому?

Практичне заняття 11. Розподілені мікроконтролерні системи на основі інтерфейсу USB (друга частина)

1. Програмна реалізація приймання біт пакетів по USB шині.

2. Задачі для розв’язання за темою.

1. Практичний приклад реалізації інтерфейсу USB на мікроконтролері

Початок побудови розподіленої мікроконтролерної системи на основі USB інтерфейсі у першій частині практичного заняття

Використовується мікроконтролер ATtiny2313, який не має вбудованого апаратного USB.

Схема достатньо проста: три світлодіоди і USB (ці світлодіоди по USB керуються).

Технічні рішення:

1) інформаційні лінії будуть заводитися на контакти PB0 (D+) та PB1 (D-);

2) визначення початку передачі по передньому фронту на лінії D+ (у стані IDLE у нас на D+ нуль, а на D- одиниця), яку для максимально швидкої реакції додатково заведена на контакт INT0 (для якої можливо встановити переривання по передньому фронту);

3) тривалість одного інформаційного біта в нас становитиме 8 тактів контролера (кварцовий резонатор на 12 МГц);

5) пристрій керуватиме трьома світлодіодами, які підключені до контактів PB2, PB3, PB4.

Крім того, відомо, що:

1) всі пакети починаються з передачі синхропослідовності SYNC = 0b10000000;

1. Практичний приклад реалізації інтерфейсу USB на мікроконтролері

2) всі дані, крім CRC, передаються молодшим бітом вперед (тобто SYNC передається у вигляді: 0b00000001);

3) дані передаються в NRZI-кодованому вигляді, тобто при передачі нуля стан шини (рівні на лініях D+/D-) змінюється протилежний, а передачі одиниці – залишається незмінним.

Виходячи з цього, а також початкового рівня лінії D+ в стані IDLE (нуль), робиться висновок про те, які дані на лінії D+ повинні бути на початку кожного пакета. Закодувавши SYNC NRZI щодо D+, отримуємо: 0b10101011.

За специфікацією деяка частина початкових біт синхропослідовності може загубитися (але скільки саме біт втрачено, не відомо), залишається лише один спосіб визначення початку пакет – перші дві поспіль одиниці на лінії D+ після виходу зі стану IDLE.

Таким чином, завдання визначення початку пакета зводиться до виявлення в перериванні INT0 перших двох одиниць, що послідовно передаються, на лінії D+. Наступний за цими двома одиницями біт – це вже перший інформаційний біт пакета. Плюс треба врахувати, що до того, як зустрінуться дві одиниці підряд – не повинно зустрічатися два поспіль нуля.

Причому для цього першого інформаційного біта попередній стан лінії D+ буде 1, тобто для нього 0 на лінії D+ відповідатиме передачі нульового біта (стан лінії змінюється на протилежне), а одиниця на D+ — передачі одиничного біта (стан лінії не змінюється).

1. Практичний приклад реалізації інтерфейсу USB на мікроконтролері

Тобто, ми можемо просто записувати через бітові інтервали стан лінії D+, а потім уже декодувати цю послідовність, виходячи з правил NRZI (усі дані у нас для цього є, - набір станів лінії та знання, що перший стан відповідає значенню біта, що передається).

Визначення кінця пакету можна за сигналом SE0 (протягом двох бітових інтервалів на обох інформаційних лініях нуль).

Крім того, тут же необхідно врахувати наступний факт: якщо досить довго нічого не відбувається і обидві лінії при цьому притягнуті до 0, значить в наявності DISCONNECT і USB інтерфейс потрібно перезавантажити (скинути адресу, поточний стан та інше).

Далі, потребує *обміркування* ще один момент: потреба приймати весь пакет повністю або при неправильних PID чи адреси відхилити залишок пакета.

Міркування: якщо виходити з переривання не дочекавшись кінця пакета, то відразу ж знову потрапимо в переривання, оскільки позитивні фронти будуть траплятися на D+ весь час, поки йдуть якісь дані. Але тепер потрапляння не в початок пакета, а в середину.

Більш того, що з даних, які тепер будуть отримуватися (з середини) може скластися якийсь пакет, який буде схожий на адресований мікроконтролер і навіть такий, у якого виявиться правильний CRC. І взагалі, навіщо ризикувати, якщо переривання в будь-якому випадку спрацьовуватиме і мікроконтролер все одно відволікатимуть.

1. Практичний приклад реалізації інтерфейсу USB на мікроконтролері

Тому доцільно у будь-якому випадку приймати всі пакети повністю, незалежно від того, чи можна відразу впізнати, що пакет призначений не мікроконтролеру чи ні, а потім вже вирішувати, що з цим робити.

Є ще одна проблема: прийняті дані складатимуться у буфер в оперативній пам'яті, але оскільки мікроконтролер «дрібний», то й оперативна пам'ять невеликого обсягу. Тоді формулюється обмеження, що корисні дані можуть бути розміром максимум 8 байт. Це означає, що, без урахування поля Sync, максимальний розмір призначеного нам пакета може бути 1(PID+CHECK)+8+2(CRC16) байт.

Скільки у такому повідомленні може бути бітових вставок? З огляду на те, що вставка робиться кожні 6 поспіль одиничних біт, отримуємо максимум 11*8/6=14 вставок. Це за умови, що абсолютно всі біти будуть одиницями (реально можна PID+CHECK виключити звідси). Тобто потрібен буфер для вхідних даних максимум на 13 байт (маркер-пакети та пакети підтвердження у будь-якому разі коротші). Якщо вхідних даних буде більше, то не записуватимуться зайві дані, розуміючи, що вони не для цього мікроконтролеру (просто чекання кінця пакета).

Наведений нижче код дозволяє зчитувати з шини USB всі low speed пакети (цілком або перші 13 байт), зберігати прийняті дані в SRAM за адресами 0x60-0x6C, а також визначати кінці пакетів, що приймаються (але не вирішені ще питання: як швидко відправити хосту підтвердження, як із «сирих» даних відновити інформацію, що передається, і що з цією інформацією робити далі).

2. Програмна реалізація приймання біт пакетів по USB шині

```
.device ATtiny2313
.include "tn2313def.inc"
.list
;---  визначаємо виводи портів  ---
.equ   InputPort   = PINB  ; звідси читаємо
.equ   OutputPort  = PORTB ; сюди пишемо
.equ   Direction   = DDRB  ; вибір напрямку
.equ   USBDPlus    = 0     ; PB0 - DATA+
.equ   USBDMinus   = 1     ; PB1 - DATA-
.equ   LED0        = 2
.equ   LED1        = 3
.equ   LED2        = 4
;---   допоможні константи   ---
.equ   USBPinMask = ~((1<<USBDMinus)|(1<<USBDPlus)) ; 0b11111100
;--------------------------------
;---   розподіл пам'яті   --------
.equ   MaxUSBBytes = 13          ; максимальний розмір буферу для «сирих» даних
.equ   StackTop    = RAMEND      ; вершини стеку
.equ   InputBuffer = RAMEND-127  ; початок буферу «сирих» вхідних даних USB (0)
;-------------------------------------------------------
.def   temp0       = r16 ; temporary register
.def   temp1       = r17 ; temporary register
.def   InputReg    = r18 ; вхідний регістр (сюди читаємо значення ліній)
.def   ShiftReg    = r19 ; зсувний регістр (сюди накопичуємо біти, які приймаються
.def   USBBufPtrXL = r26 ; XL - вказівник  на буфер USB
.def   USBBufPtrXH = r27 ; XH - вказівник  на буфер USB
;*******************************************************
;--  початок програмного коду
   .cseg
   .org 0
   rjmp  Init          ; перехід на початок програми (вектор скидання)
;--  далі йдуть вектори переривань
   rjmp  IRQ_INT0      ; зовнішне переривання INT0
   .org  WDTaddr+1     ; програма починається за таблицею векторів
;-------------------------------------------------------------
```

2. Програмна реалізація приймання біт пакетів по USB шині

```
;--      початок програми (ініціалізація портів і змінних)      ---
Init:
   ldi   temp0, StackTop
   out   SPL,temp0          ; ініціалізуємо стек

   ldi   temp0,(1&lt;&lt;LED0)+(1&lt;&lt;LED1)+(1&lt;&lt;LED2)
   out   Direction,temp0    ; лінії світлодіодів – виходи
   ldi   temp0,0b11111011
   out   PORTD,temp0        ; включаємо підтягування на PORTD, окрім PD2 (INT0)

   rcall USBReset           ; обнуління адрес, скидання станів і т.д.

   ldi   temp0,0x0F         ; INT0 – переривання по передньому фронту
   out   MCUCR,temp0
   ldi   temp0,1&lt;&lt;INT0      ; включаємо зовнішнє переривання INT0
   out   GIMSK,temp0
   sei                      ; дозволяємо немасковані переривання
;--- Основний цикл ---
General_loop:
   sbis  InputPort,USBDminus ; якщо D- = 0, то можливо це Disconnect,
   rjmp  CheckUSBReset       ; у випадку якого будемо робити Reset
   rjmp  General_loop        ; якщо D- = 1, то вважаємо, що це IDLE
;-- Перевіряємо, чи не трапився Disconnect ---
CheckUSBReset:
   ldi   temp0,255           ; будемо відраховувати 225 циклів
   ; якщо за цей час не зміниться стан D- (так і залишиться 0), то
   ; вважаємо, що відбувся Disconnect і необхідний Reset
WaitUSBReset:
   sbic  InputPort,USBDminus ; якщо D- все ще нуль – пропустити
   rjmp  General_loop
   dec   temp0               ; зменшуємо лічильник
   brne  WaitUSBReset        ; стрибаємо, якщо не нуль
   rcall USBReset
   rjmp  General_loop
;---  Кінець основного циклу  ---
;***********************************************************
```

2. Програмна реалізація приймання біт пакетів по USB шині

```
;************************************************************
;---  Скидання USB (обнуління адрес, скидання станів …)  ---
USBReset:
   ret
;************************************************************
;--- Зовнішнє переривання INT0 (позитивний фронт на D+) --
IRQ_INT0:
   ldi   temp0,2    ; готуємося відраховувати кількість однакових біт
   ldi   temp1,2
   ; визначаємо момент зміни біта
CheckDMOne:
   sbis  InputPort,USBDMinus
   rjmp  CheckDMOne
   ;--- тепер чекаємо одиниччний біт (в синхропослідовності це в
   ;--- обов'язковому порядку буде стан, коли D+ = 1)
   ;--- момент виявлення цього біта будемо застосовувати для синхронізації
   ;--- (від нього починаємо відлік бітових інтервалів, оскільки наступні частинки кода
   ;--- повинні бути строго виверені за кількістю тактів в відладчику)
CheckDPOne:
   sbis  InputPort,USBDPlus ; самий початок одиничного біта
   rjmp  CheckDPOne         ; 2-й такт
DetectSyncEnd:
   sbis  InputPort,USBDPlus ; 3,4-й такти (D+=1) або 3-й такт (D+=0)
   rjmp  TestBit0           ; -/-                    4,5-й такти (D+=0)
TestBit1:
   ldi   temp0,2            ; 5-й такт (скидаємо лічильник нулів)
   dec   temp1              ; 6-й такт
   nop                      ; 7-й такт (затримка, щоб отримати 8 тактів на біт)
   breq USBBeginPacket      ; 8-й такт при temp1>0 або 8-й і 1-й при temp1=0
   rjmp DetectSyncEnd       ; 1,2-й такти
TestBit0:
   ldi   temp1,2            ; 6-й такт (скидаємо лічильник одиниць)
   dec   temp0              ; 7-й такт
   nop                      ; 8-й такт
   brne  DetectSyncEnd      ; 1,2-й такти при temp0>0
   ;--- сюди потрапляємо, якщо два поспіль біта дорівнюють нулю
   ;--- вважаємо, що це був глюк і просто виходимо  ---
ExitFromIRQ:
   reti
```

2. Програмна реалізація приймання біт пакетів по USB шині

```
   ;--- Починаємо приймати інформаційні біти  ---
USBBeginPacket: ; сюди потрапляємо після 1-го такту першого інформаційного біта
   ;---  читаємо перший інформаційний біт байту  ---
   in    ShiftReg,InputPort ; і відразу пишемо його в зсувний регістр
   nop                      ; 3-й такт
USBLoopBegin:
   ldi   temp0,6                  ; лічильник бітів (для наступних 6 бітів)
   ldi   temp1,MaxUSBBytes        ; лічильник байтів
   ldi   USBBufPtrXL,InputBuffer ; 6-й такт
   nop                            ; 7-й такт
USBLoopByte:
   nop                      ; 8-й такт
   ;--- читаємо 2-7 інформаційні біти      ---
USBLoop27:
   in    InputReg,InputPort  ; читаємо значення бітів D+|D-       | 1-й такт
   cbr   InputReg,USBPinMask ; скидаємо усі біти порту, окрім значень D+|D-
   breq  EndPacket           ; якщо обидва нулі – кінець пакету    | 3-й такт, якщо не нуль
   ror   InputReg            ; витісняємо D+ в CF                  | 4-й такт
   rol   ShiftReg            ; і пишемо його в зсувний регістр     | 5-й такт
   dec   temp0               ; прочитали 7 бітів?                  | 6-й такт
   brne  USBLoop27           ; якщо ні – читаємо (7, 8-й такти)
   nop                       ; 8-й такт
   ;--- читаємо останній біт байту  ---
USBLoop8:
   in    InputReg,InputPort  ; 1-й такт
   cbr   InputReg,USBPinMask ; скидаємо усі біти порту, окрім значень D+|D-
   breq  EndPacket           ; якщо обидва нулі – кінець пакету    | 3-й такт, якщо не нуль
   ror   InputReg            ; витісняємо D+ в CF                  | 4-й такт
   rol   ShiftReg            ; і пишемо його в зсувний регістр     | 5-й такт
   ldi   temp0,7             ; налаштовуємо лічильник на прийом решти 7 бітів
   st    X+,ShiftReg         ; зберігаємо в буфер прийнятий байт| 7,8-й такт
   ;--- читаємо перший біт наступного байту---
USBLoop1:
   in    ShiftReg,InputPort  ; відразу пишемо його в зсувний регістр
   cbr   InputReg,USBPinMask ; скидаємо усі біти порту, окрім значень D+|D-
   breq  EndPacket           ; якщо обидва нулі – кінець пакету    | 3-й такт, якщо не нуль
   dec   temp0               ; 4-й такт
   dec   temp1               ; 5-й такт
   brne  USBLoopByte         ; 6,7-й такт, якщо не нуль            | 6-й такт, якщо нуль
```

2. Програмна реалізація приймання біт пакетів по USB шині

```
   ;---якщо буфер переповнений – решту бітів просто не записуємо,
   ;---(наш пакет по-любому менше, але кінця пакету необхідно дочекатися)
BufferOverrange:
   nop                        ; 7-й такт
   nop                        ; 8-й такт
   in    InputReg,InputPort   ; 1-й такт
   cbr   InputReg,USBPinMask  ; скидаємо усі біти порту, окрім значень D+|D-
   breq  EndPacket            ; якщо обидва нулі – кінець пакету     | 3-й такт, якщо не нуль
   nop                        ; 4-й такт
   rjmp  BufferOverrange      ; 5,6-й такт
   ;---завершили прийняття пакету і починаємо його аналізувати
   ;---потрапляємо сюди на 5-му такті першого або другого бітового
   ;---інтервалу SE0, тепер необхідно як можна скоріше зрозуміти чого
   ;---від нас потребується і відповісти (або не відповісти, якщо бажають щось не від нас)
EndPacket:
   cpi   USBBufPtrXL,InputBuffer+3 ; прийняли хоча б 3 байти?
   brcs  ExitFromIRQ               ; якщо ні – просто виходимо
   ;*****************************************
   ;!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   ;--- а ось якщо так – починаємо розбирати пакет
   reti
```

2. Задачі для розв’язання за темою

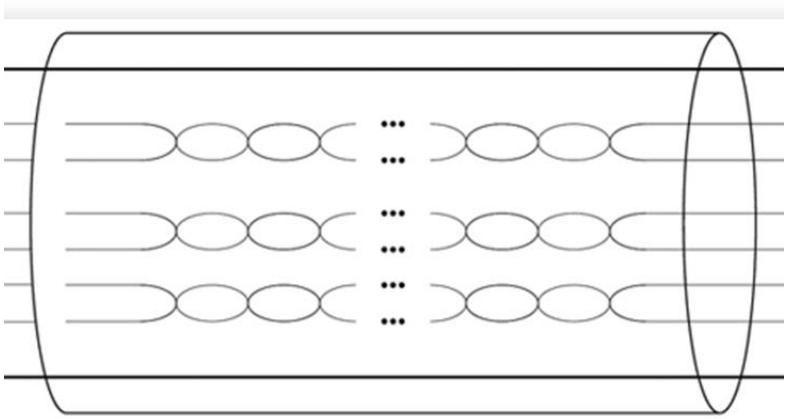
Задача 1. Поясніть, чому кодування щодо лінії D+ комбінації SYNC, яку містить початок інформаційного пакету інтерфейсу USB, дає кодову комбінацію виду 0b10101011.

Задача 2. Поясніть, чому в практичному прикладі реалізації мікроконтролерного пристрою на основі USB інтерфейсу визначення початку інформаційного пакету на шині інтерфейсу зводиться до виявлення в перериванні INT0 двох перших підряд двійкових одиниц.

Задача 3. Яким чином для прикладу реалізації мікроконтролерного пристрою на основі USB інтерфейсу визначається максимальна довжина інформаційного пакету? Чим пояснюється обмеження максимальної довжини даних 8 байт?

Задача 4. Проставьте у відповідність функціональні позначення сигналів для шини USB інтерфейсу на рисунку 1. До якої специфікації USB відноситься вид цієї шини (роз’єму)?

Рисунок 1 – Шина USB інтерфейсу



Практичне заняття 12.
Периферія мікроконтролера STM32F407V

1. Інтерфейс I^2C мікроконтролера STM32F407V.
2. Приклад використання SPI на платі STM32F4DISCOVERY.
3. Особливості підключення STM32 до COM-порту комп’ютера.
4. Задачі для розв’язання за темою.

1. Інтерфейс I^2C мікроконтролера STM32F407V

МК містить три блоки I^2C, що підтримують роботу в режимі Maste/Slave (ведучий або ведений), а також у режимі Multimaster (режим в якому на шині присутні кілька Master-пристроїв, які поділяють спільні ресурси Slave, або по черзі змінюють свій стан з Master на Slave і назад). У складі пристрою є модуль діагностики та виправлення пакетних помилок PEC.

Використовується 7-бітний і 10-бітний режим адресації. Підтримуються загальноприйняті для протоколу швидкості обміну даними до 100 кГц в простому режимі і 400 кГц в режимі швидкого обміну даними. Модулі можуть бути сконфігурірованни на розширені протоколи SMBus 2.0 і PMBus.

Для початку слід включити тактування модуля I^2C і налаштувати піни в режим альтернативної функції (*див. STM32F407XX Datasheet, PDF Режим доступу до ресурсу: http://www.alldatasheet.com/view.jsp?Searchword=Stm32f407xx, розділ Device Overview*). З нього видно, що I^2C знаходяться на шині APB1.

Наступний крок – включення та налаштування GPIO, обираємо режим альтернативної функції (I^2C відноситься до AF4), тип OpenDrain, а підтяжка повинна бути зовнішня. «Швидкість» пінов для 100 кГц можна вибрати Low (2 MHz), а для 400 кГц ST рекомендують вибирати Medium або Fast (від 10 MHz). І, далі, можна налаштувати I^2C.

До включення безпосередньо модуля I^2C слід в регістр CR2 записати поточне значення частоти тієї шини, до якої підключений модуль I^2C, в даному випадку це частота шини APB1. В рамках даташіта це значення називається PCLK.

1. Інтерфейс I2C мікроконтролера STM32F407V

Регістр CR2 керує перериваннями від даного модуля. Під цим розуміється то, що чи буде модуль I2C повідомляти в NVIC про те, що щось сталося, або ж просто поставить потрібні прапори в статусному регістрі. Варто зауважити, що в статусному регістрі завжди будуть ставитися подієві прапори, що логічно. В першу чергу цікаві переривання ITEVTEN і ITERREN, переривання подій і помилок відповідно. Можна обійтися цілком і тільки подіями, як найбільш загальним випадком:

```
I2C1 -> CR2 |= 48; // Peripheral frequency 24MHz
I2C1 -> CR2 |= I2C_CR2_ITEVTEN; // Enable events
```

Регістр CCR відповідає за тактування зовнішньої шини, тому в нього необхідно внести значення, яке розраховується за формулою PCLK/I2C_SPEED. Наприклад, для частоти шини 400 кГц, внутрішня шина APB1 тактується 48 МГц, відповідно в CCR запишемо значення, рівне $48 \times 10^6 / 4 \times 10^6 = 120$. Так само в даному регістрі необхідно вказати режим роботи Slow/Fast, це останній, 16-й біт:

```
I2C1 -> CCR &= ~I2C_CCR_CCR;
I2C1 -> CCR |= 120;
I2C1 -> CCR |= I2C_CCR_FS; // FastMode, 400 kHz
```

Регістр TRISE відповідає за фронти сигналів на SDA і SCL, сюди необхідно внести значення з невеликим запасом, яке розраховується так:

TRISE = RISE/tPCLK (tPCLK = 1/PCLK).

1. Інтерфейс I2C мікроконтролера STM32F407V

Константа RISE – це максимальний час наростання сигналу, по специфікації: 1000 нс для Slow Mode і 300 нс для Fast mode. tPCLK – це період, що обчислюється як 1/F. Оскільки обрано Fast Mode, то значення в TRISE буде наступне:

$3 \times 10^{-7} / 2{,}083 \times 10^{-8} = 14{,}4$, з невеликим запасом в більшу сторону, обираємо 15.

```
I2C1 -> TRISE = 24;
```

Після того, як дані дії будуть виконані, можна включати модуль і переривання в модулі NVIC (якщо потрібні).

```
I2C1 -> CR1 |= I2C_CR1_PE; // Enable I2C block
NVIC_EnableIRQ(I2C1_EV_IRQn);
NVIC_SetPriority(I2C1_EV_IRQn, 1);
```

Після настройки модуля, з ним можна працювати або в режимі чекання появи прапора в циклі while (і контролер буде зайнятий тільки цим, що в більшості випадків погано), або – за перериванням. В другому випадку необхідно додати в код обробник переривань подій від модуля I2C. Для модуля I2C1 функція називається I2C1_EV_IRQHandler. Тому в необхідний «.c» файл додаємо таку функцію:

```
void I2C1_EV_IRQHandler(void) {
}
```

На цьому ініціалізація закінчується.

1. Інтерфейс I2C мікроконтролера STM32F407V

Далі розглянемо передачу даних slave-пристрою.

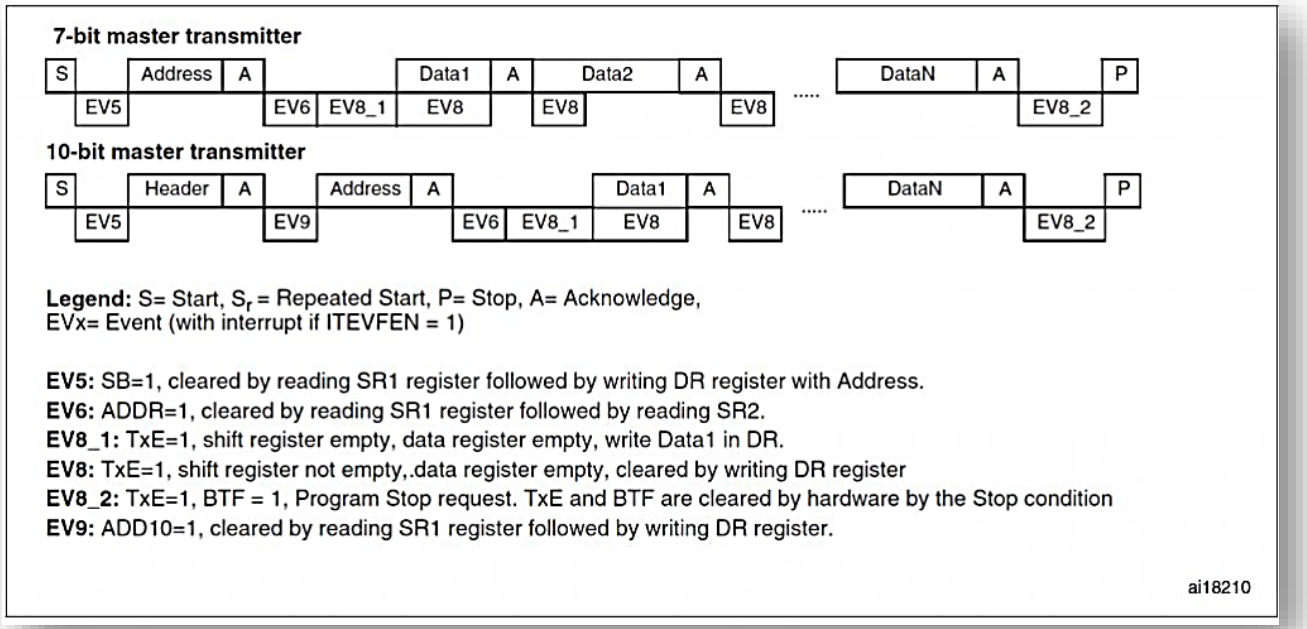


Рисунок 1 – Передача даних slave-пристрою

Наприклад, нехай необхідно відправити 1 байт даних пристрою і припинити передачу. Для цього будемо використовувати стани EV5, EV6 та EV8. Десь в коді програми визначена глобальна змінна. Початок передачі ініціює послідовність START:

I2C1 -> CR1 |= I2C_CR1_START;

Даний рядок можна поставити по ходу програми там, коли потрібно починати передачу. Наприклад, після того, як ініціалізували змінну data.

1. Інтерфейс I2C мікроконтролера STM32F407V

Далі вже буде код всередині функції-обробника переривань. Для початку необхідно в окремі змінні зберегти значення статусів:

```
volatile uint32_t sr1 = I2C1 -> SR1, sr2 = I2C1 -> SR2;
```

Після відправки стартової послідовності відбудеться переривання за подією EV5. В статусному регістрі повинен виставитися біт SB. Якщо даний біт виставлений, то необхідно відправити адресу з бітом режиму читання або запису. Для спрощення можна зробити так:

```
#define I2C_MODE_READ 1
#define I2C_MODE_WRITE 0
#define I2C_ADDRESS(addr, mode) ((addr<<1) | mode)
```

Тепер можна написати обробник стану EV5:

```
if( sr1 & I2C_SR1_SB ) {
 I2C1->DR = I2C_ADDRESS(0x14,I2C_MODE_READ); }
```

Коли адресу відправиться і slave-пристрій відповість послідовністю ACK, то станеться подія EV6 і одночасно EV8: встановиться прапор ADDR і TXE. Прапор ADDR означає, що адреса відправлена і прийнята slave-пристроєм, а TXE означає, що буфер вільний для внесення даних для подальшої передачі.

Прапор ADDR скинеться сам, як тільки буде прочитано SR1 і SR2, а прапор TXE опрацю-ємо окремим блоком коду. У обробнику TXE все, що потрібно – це передавати дані.

1. Інтерфейс I2C мікроконтролера STM32F407V

Так як передавати будемо тільки 1 байт, то відразу ж відправимо і послідовність STOP

```
if(sr1 & I2C_SR1_TXE) {
I2C1 -> DR = data;
I2C1 -> CR1 |= I2C_CR1_STOP; }
```

Таким чином, отримали наступну функцію-обробник:

```
#define I2C_MODE_READ 1
#define I2C_MODE_WRITE 0
#define I2C_ADDRESS(addr, mode) ((addr<<1) | mode) uint8_t iter;
uint8_t data[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
void I2C1_EV_IRQHandler(void) {
   volatile uint32_t sr1 = module ->SR1, sr2 = module ->SR2;
   if( sr1 & I2C_SR1_SB ) {
      module ->DR = I2C_ADDRESS(0x14,I2C_MODE_READ);
   }
   if(sr1 & I2C_SR1_TXE) {
      if( iter < 10 ) {
         I2C1->DR = data[iter++];
      } else {
         I2C1->CR1 |= I2C_CR1_STOP;
      }
   }
}
```

2. Приклад використання SPI на платі STM32F4DISCOVERY.

Приклад використання SPI на платі STM32F4DISCOVERY, показує як налаштувати SPI1 в режимі Master і відправити дані. У нескінченному циклі відбувається передача одного і того ж байту (0x93).

```
01.#include "stm32f4xx.h"
02.#include "stm32f4xx_gpio.h"
03.#include "stm32f4xx_rcc.h"
04.#include "stm32f4xx_spi.h"
05.
06.int main(void) {
07.GPIO_InitTypeDef GPIO_InitStructure;
08.SPI_InitTypeDef SPI_InitStructure;
09.
10.// Тактуваннямодуля SPI1 іпортуA
11.RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1, ENABLE);
12.RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
13.
14.// Налаштовуємо вивід SPI1 для роботи в режимі альтернативної функції
15.GPIO_PinAFConfig(GPIOA, GPIO_PinSource7, GPIO_AF_SPI1);
16.GPIO_PinAFConfig(GPIOA, GPIO_PinSource5, GPIO_AF_SPI1);
17.GPIO_PinAFConfig(GPIOA, GPIO_PinSource6, GPIO_AF_SPI1);
```

2. Приклад використання SPI на платі STM32F4DISCOVERY.

```
18.
19.GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
20.GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
21.GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
22.GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
23.GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7 | GPIO_Pin_6 | GPIO_Pin_5;
24.GPIO_Init(GPIOA, &GPIO_InitStructure);
25.
26.//Заповнюємо структуру с параметрами SPI модуля
27.SPI_InitStructure.SPI_Direction= SPI_Direction_2Lines_FullDuplex; //повний
дуплекс
28.SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b; // передаємо по 8 біт
29.SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low; // Полярність та
30.SPI_InitStructure.SPI_CPHA = SPI_CPHA_1Edge; // фаза тактовогосигналу
31.SPI_InitStructure.SPI_NSS = SPI_NSS_Soft; // Керувати станом сигналу
NSS програмно
32.SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_32; //
Передільник SCK
33.SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB; //
Першим відправляється старший біт
```

2. Приклад використання SPI на платі STM32F4DISCOVERY.

```
34.SPI_InitStructure.SPI_Mode = SPI_Mode_Master; // Режим - Master
35.SPI_Init(SPI1, &SPI_InitStructure); //Налаштовуємо SPI1
36.SPI_Cmd(SPI1, ENABLE); // Включаємомодуль SPI1....
37.
38.// Оскільки сигнал NSS контролюжться програмно, встановимо його в одиницю
39.// Якщо скинути його в нуль, то SPI модуль вирішить, що
40.// використовується мультимастерна топологія і його перевели в Slave.
41.SPI_NSSInternalSoftwareConfig(SPI1, SPI_NSSInternalSoft_Set);
42.while(1) {
43.SPI_I2S_SendData(SPI1, 0x93); //Передаембайт 0x93 через SPI1
44.while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_BSY) == SET)
45.}}
```

3. Особливості підключення STM32 до COM-порту комп’ютера

У комп'ютера інтерфейсу UART як такого немає, але є COM порт. Різниця між COM портом і UART тільки в рівнях напруг. В COM порті логічний нуль – це приблизно +12 вольт, а логічна одиниця приблизно – 12 вольт.

Якщо підключити мікроконтролер безпосередньо до виводів COM порту, то він може вийти з ладу. Для нього логічний нуль - це приблизно нуль вольт, а логічна одиниця – це приблизно повна напруга живлення. Для узгодження логічних рівнів використовують спеціальні мікросхеми, наприклад, max3232.

Контролери STM32 працюють від 3.3 вольта, і напруга логічної одиниці не повинна перевищувати напруги живлення. Таким чином, застосовуючи max232 потрібно знизити напругу виходу до 3,3 В. Або можна використовувати max3232. Схема включення показана на рисунку 2.

Розглянемо програмну реалізацію. Для того щоб відправити щось в UART використовують термінальні програми. Наприклад – стандартний HyperTerminal, або Bray's Terminal (рисунок 3).

Приклад. Передача на ПК рядка з вітанням“Hello:)”. На вивід PA9 в даному прикладі був підключений вхід (RxD) USB-UART перетворювача. Налаштування UART: швидкість 9600 біт/сек, 1 стоп біт, без перевірки парності.

3. Особливості підключення STM32 до COM-порту комп’ютера

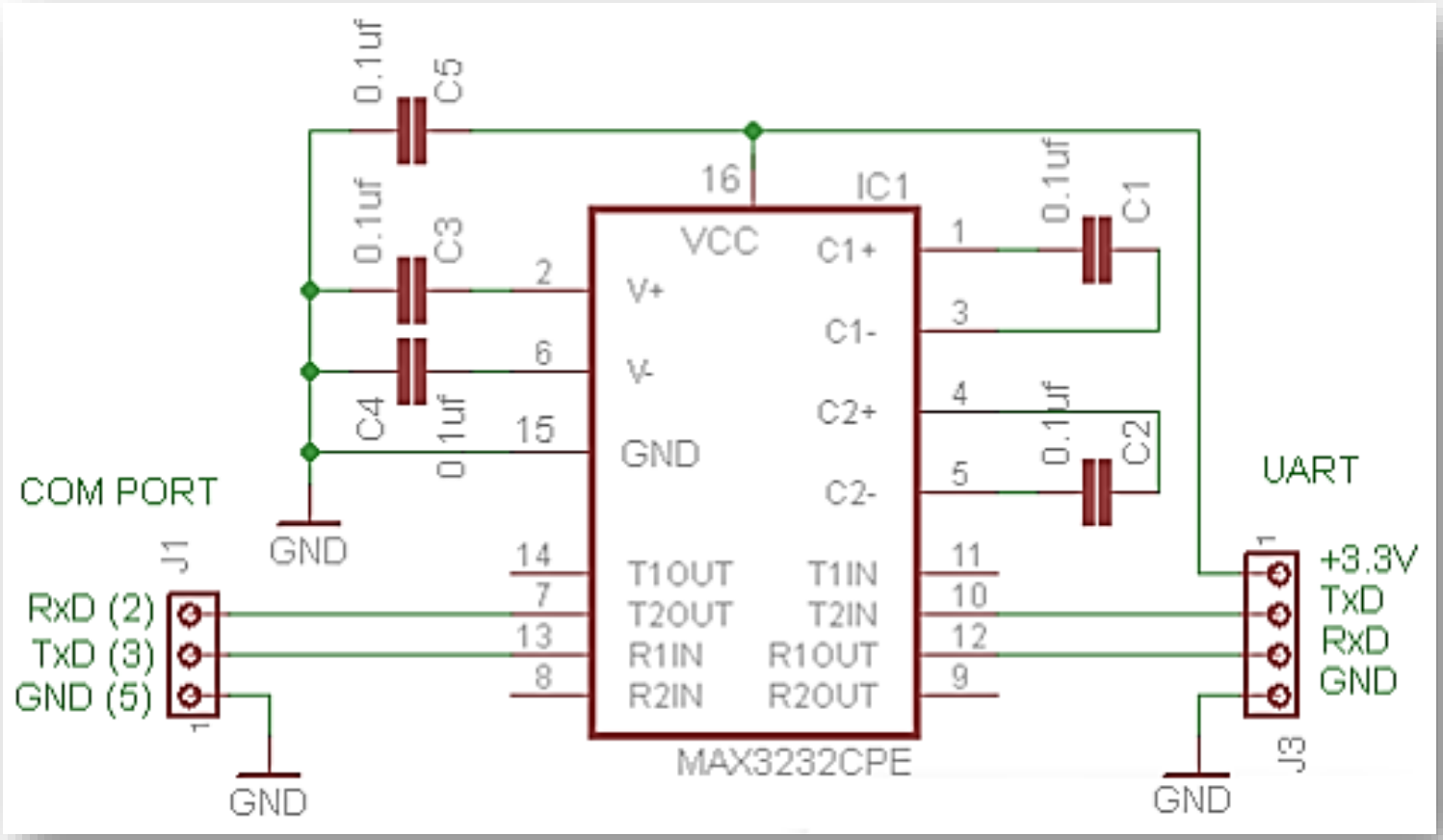


Рисунок 2 – Схема підключення мікросхеми max3232CPE

3. Особливості підключення STM32 до COM-порту комп’ютера

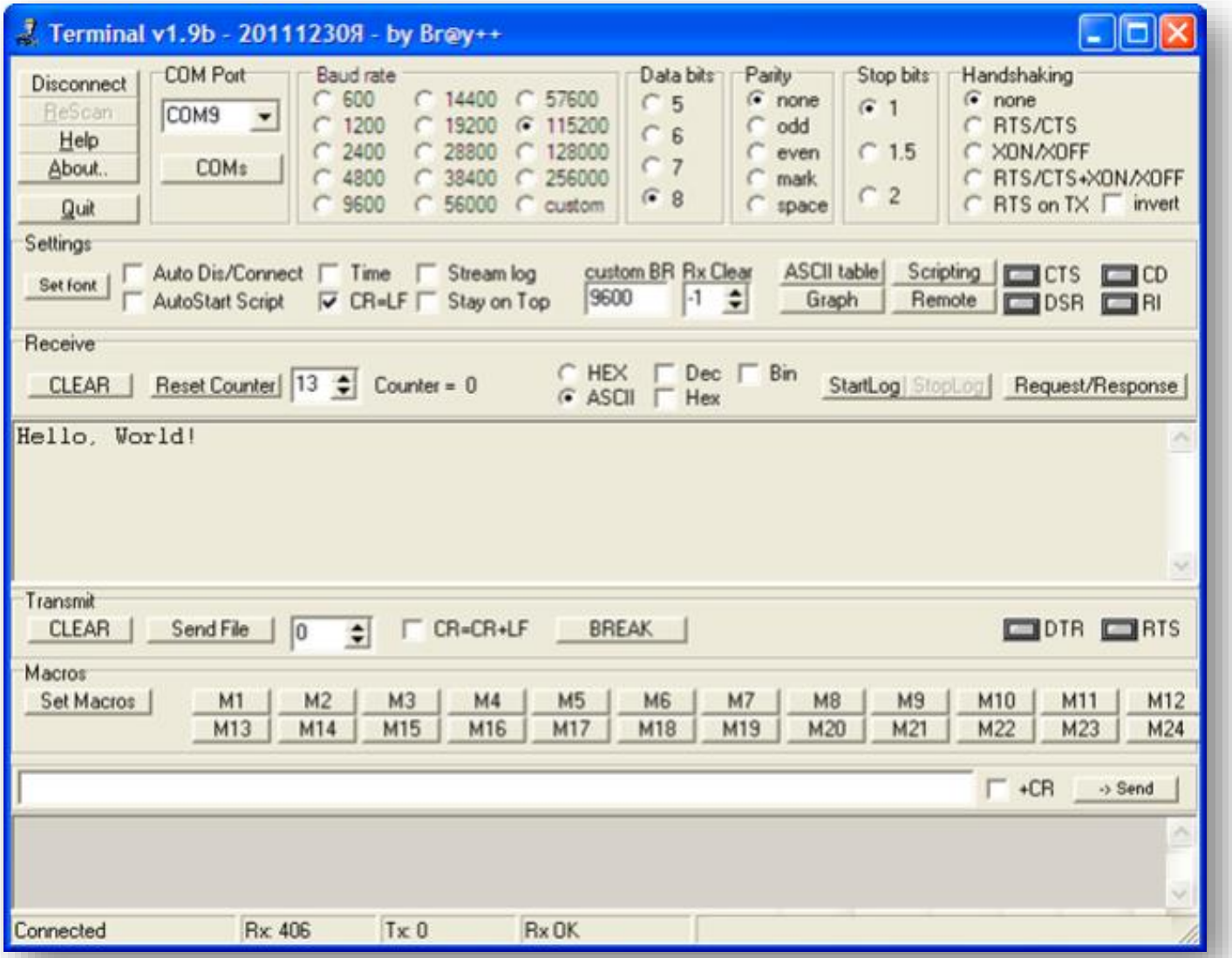


Рисунок 2 – Вікно програми Bray`s Terminal

3. Особливості підключення STM32 до COM-порту комп’ютера

```
01.#include "stm32f10x.h"
02.#include "stm32f10x_gpio.h"
03.#include "stm32f10x_rcc.h"
04.
05.//Функція призначена для формування невеликої затримки
06.void Delay(void) {
07.volatile uint32_t i;
08.for (i=0; i != 0x70000; i++);
09.}
10.
11.//Функція, що відправляє байт в UART
12.void send_to_uart(uint8_t data) {
13.while(!(USART1 -> SR & USART_SR_TC)); //Чекаємо поки біт TC
в регістрі SR станет 1
14.USART1 -> DR=data; //Відправляємо байт через UART
15.}
16.
17.int main(void) {
18.GPIO_InitTypeDef PORTA_init_struct;
19.// Включаємо тактування порта A та USART1
```

3. Особливості підключення STM32 до COM-порту комп’ютера

```
20.RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA |
RCC_APB2Periph_USART1, ENABLE);
21.// Налаштовуємо вивід TxD (PA9) як вихід push-pull з
альтернативною функцією
22.PORTA_init_struct.GPIO_Pin = GPIO_Pin_9;
23.PORTA_init_struct.GPIO_Speed = GPIO_Speed_50MHz;
24.PORTA_init_struct.GPIO_Mode = GPIO_Mode_AF_PP;
25.GPIO_Init(GPIOA, &PORTA_init_struct);
26.//Налаштовуємо UART
27.USART1->BRR=0x9c4; // BaudRate 9600
28.USART1->CR1 |= USART_CR1_UE; //Дозволяємо роботу USART1
29.USART1->CR1 |= USART_CR1_TE; //Включаемо передавач
30.while(1) {
31.//Відправляємочерез UART слово Hello
32.send_to_uart('H');
33.send_to_uart('e');
34.send_to_uart('l');
35.send_to_uart('l');
36.send_to_uart('o');
37.send_to_uart(' ');
```

3. Особливості підключення STM32 до COM-порту комп’ютера

```
38.send_to_uart(':');
39.send_to_uart(')');
40.send_to_uart('\n');
41.Delay(); //невеликазатримка
42.}}
```

Створюємо порожній проект, копіюємо код, компілюємо і прошиваємо. Вікно терміналу, після виконання програми прийме вигляд як на рисунку 4.

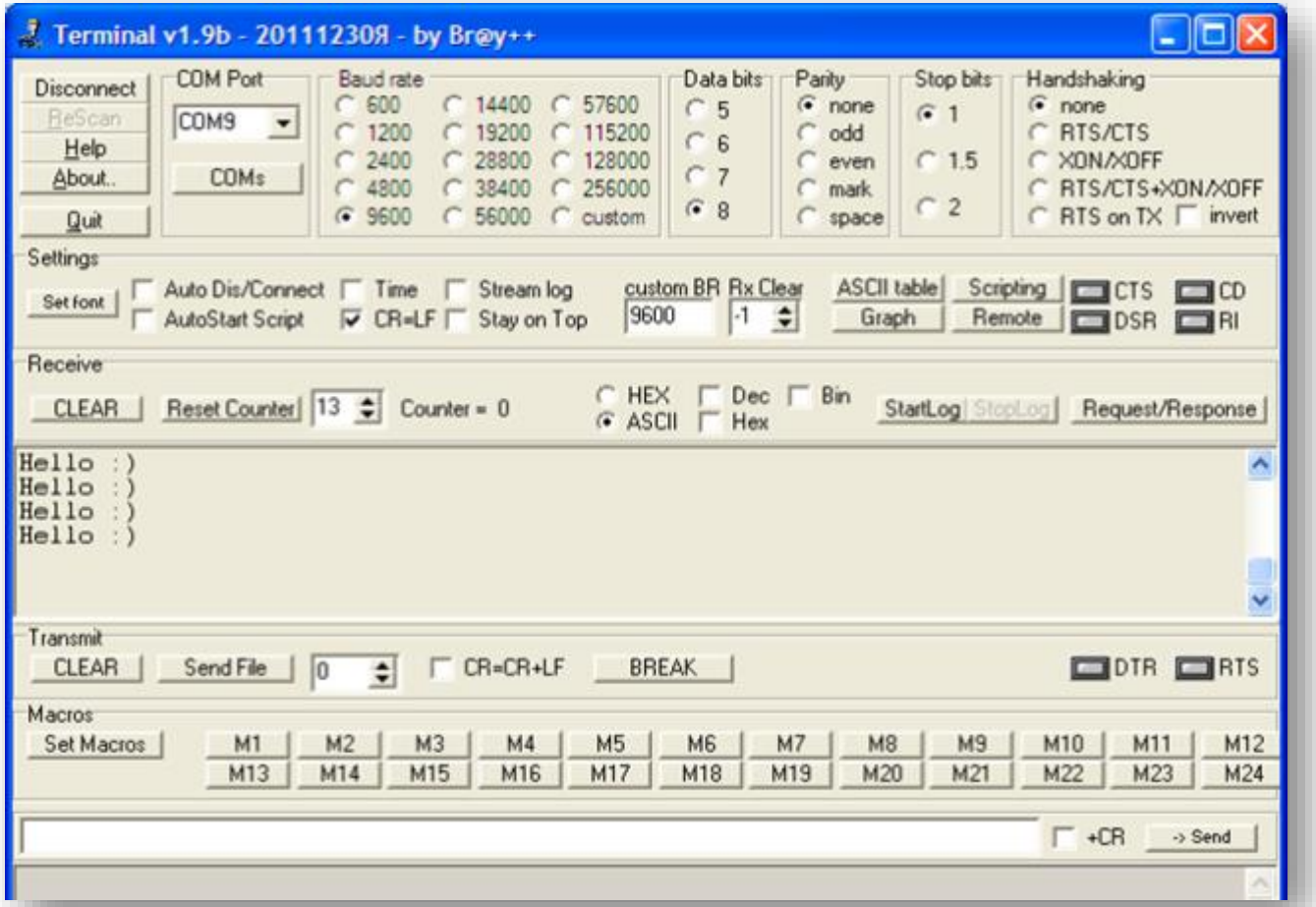


Рисунок 4 – Прийом інформації з мікросхеми у вікні Bray`sTerminal

4. Задачі для розв’язання за темою

Задача 1. Скільки внутрішніх блоків I^2C містить мікроконтролер STM32F407V і в яких режимах вони можуть працювати?

Задача 2. Наведіть стисло (тезисно) алгоритм ініціалізації та налаштування модуля I^2C, що міститься в мікроконтролері STM32F407V.

Задача 3. Яким чином формуються умова START (S) та умова STOP (P) I^2C інтерфейсу в мікроконтролері STM32F407V?

Задача 4. Які виводи мікроконтролера STM32F407V використовуються для організації обміну даними за SPI інтерфейсом?

Задача 5. Яким чином узгоджують рівні напруги сигналів COM-порту комп’ютера з рівнями напруги сигналів мікроконтролера STM32? Чому виникає така необхідність?