МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМ. В. ДАЛЯ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЕЛЕКТРОНІКИ
КАФЕДРА ПРОГРАМУВАННЯ ТА МАТЕМАТИКИ

До захисту допускається
В.о. завідувач кафедри
_____ Лифар В.О.
«____»_____ 2020 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

**до дипломної роботи**

**бакалавр**

**(освітньо-кваліфікаційний рівень)**

НА ТЕМУ:

# Комп’ютерна система управління відображенням рухомого рядка з послідовним інтерфейсом

Керівник роботи: _____ Марченко Д. М.
(підпис) (ініціали, прізвище)

Студент: _____ Кожем’якін В.Е.
(підпис) (ініціали, прізвище)

Група: СКС-16

Сєвєродонецьк 2020

ЛИСТ ПОГОДЖЕННЯ І ОЦІНЮВАННЯ

дипломної роботи студента гр. СКС-16 Кожем’якін В.Е.

Науковий керівник

Професор, д.т.н. _____ Марченко Д. М.

Оцінка наукового керівника: _____

Рецензент: _____

ПІБ, місто роботи, посада

Оцінка рецензента: _____

Кінцева оцінка за результатами захисту:

_____

Голова ЕК _____ Лифар В.О.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет Інформаційних технологій та електроніки

Кафедра Програмування та математики

Освітньо-кваліфікаційний рівень бакалавр

Напрям підготовки 6.050102 «Комп'ютерна інженерія»

(шифр і назва)

Спеціальність _____

(шифр і назва)

**ЗАТВЕРДЖУЮ:**

Завідувач кафедри ПМ

_____ В.О. Лифар

«_____» _____20____р.

**З А В Д А Н Н Я**

**НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ**

Кожем'якін Владислав Едуардович

(прізвище, ім'я, по батькові)

Тема роботи: **Комп'ютерна система управляння відображенням рухомого рядка з послідовним інтерфейсом**

керівник проекту (роботи) Марченко Д.М., доктор технічних наук, професор

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від " " _____2020 р № ____

Строк подання студентом роботи 07 червня 2020

Вихідні дані до роботи Аналіз варіантів побудови комп'ютерної системи. Розробка апаратно-програмного забезпечення.

Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Аналітичний огляд, розробка апаратного забезпечення, розробка програмного забезпечення. Висновки.

Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

_____

_____

Консультанти розділів проекту (роботи)

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |
| | | | |

Дата видачі завдання 22 березня 2020 року

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

**КАЛЕНДАРНИЙ ПЛАН**

| № з/п | Назва етапів дипломного проекту (роботи) | Строк виконання етапів проекту (роботи) | Примітка |
|---|---|---|---|
| 1 | Складання плану роботи | 22.03.20 - 24.03.20 | |
| 2 | Аналіз літератури | 24.03.20 – 29.03.20 | |
| 3 | Вивчення і підбирання матеріалу | 29.03.20 – 20.04.20 | |
| 4 | Написання розділів | 20.04.20 – 25.05.20 | |
| 5 | Оформлення пояснювальної записки | 25.05.20 – 28.05.20 | |
| 6 | Оформлення графічного матеріалу | 28.05.20 – 03.06.20 | |
| 7 | Підготовка доповіді і слайдів для презентації | 03.06.20 – 07.06.20 | |

**Студент** _____ _____
( підпис ) (прізвище та ініціали)

**Науковий керівник** _____ _____
( підпис ) (прізвище та ініціали)

# РЕФЕРАТ

Пояснювальна записка містить 82 сторінки, 16 рисунків, 1 таблицю, 4 джерела літератури.

Метою курсової роботи є розробка апаратно-програмного забезпечення для управління рухомим рядком з управління по послідовному інтерфейсу. Розроблена система передбачає наявність поля введення тексту та реалізацію відправки цього тексту на віртуальний COM-порт, який створюється в операційній системі драйвером перетворювача USB-to-UART FT232.

Програмне забезпечення було розроблено в середовищі програмування Borland C++ Builder на мові програмування C++.

**Ключові слова:** програмне забезпечення, C++, рухомий рядок, паралельний інтерфейс, UART, USB-UART

.

# ЗМІСТ

# ВСТУП

В сьогоднішній час тотальної інформатизації залишаються достатньо актуальними питання не тільки ефективної обробки інформації, а й належного, якісного і ефективного її представлення користувачеві.

Слід зазначити, що від якості і ефективності використаних засобів відображення інформації багато в чому залежить і ефективність її використання. Тобто, можна розробити достатньо ефективні методи і засоби обробки інформації, але все це може бути нівельовано неефективними засобами її представлення споживачу, що значно ускладнить та мінімізує ефективність її використання для прийняття управлінських задач.

На сьогодень існує дуже велика кількість різноманітних засобів відображення інформації, які базуються на різних фізичних принципах та спроможні вказувати дію на різні органи почуттів. При цьому, враховуючи той факт, що людини більше 86% інформації отримує за візуальним каналом зору, то засоби відображення візуальної інформації займають передову позицію та є найбільш використаними та ефективними.

Рухомий рядок є достатньо поширеним засобом представлення текстової інформації на обмеженому розмірі засобів відображення. При чому ця інформація може динамічно змінюватися, що спричиняє достатньо велику популярність різноманітним реалізаціям цього засобу відображення для будь-яких прикладних застосувань, починаючи від рекламних панелей, банерів і завершуючи панелями розкладу руху транспорту на вокзалах, виведення баржевих котировок, тощо.

На сьогодень, апаратне забезпечення засобів реалізації рухомого рідку є достатньо розробленими і основний акцент робиться на створенні удобного прикладного програмного забезпечення для управління цими засобами.

В цій дипломній роботі робиться акцент саме на цьому.

Прикладні програми призначені для вирішення функціональних завдань, що виконують обробку інформації різних предметних областей. Це найчисленніший клас програмних продуктів.

У свою чергу він умовно поділяється на дві групи:

- 1 група - програми для рішення окремих, самостійних завдань. Ці програми виконуються незалежно одна від одного і являють собою набір розрізнених, не пов'язаних між собою знань. Звичайно, і вони можуть бути досить складними і дуже необхідними завданнями.

- 2 група - системи програм для вирішення класів завдань з різних спеціалізованих галузей науки техніки і промисловості. Часто такі системи називають пакетами прикладних програм. Програми, що входять в пакет, виконуються не окремо один від одного, а спільно, в різних комбінаціях, залежно від конкретної розв'язуваної завдання.

Існує безліч прикладних програм для проектування та конструювання, що спрощують роботу інженеру у сфері електроніки та електротехніки (системи CAD, CAE, IDE та ін.).

Використання прикладного програмування в електроніці дозволяє розширити функціональність електронних пристроїв і систем, забезпечити зручний інтерфейс взаємодії користувача з пристроєм і спростити управління. Також кошти прикладного програмування дозволяють будувати масштабні, багаторівневі системи.

Дана робота присвячена розробці апаратно-програмному забезпеченню для управління індикацією "рухомий рядок". Взаємодія з апаратним забезпеченням та програмування рядка для відображення буде здійснюватися за допомогою послідовного інтерфейсу USB.

# 1 АНАЛІТИЧНИЙ ОГЛЯД

## 1.1 Загальні відомості про системи рухомих рядків

Світлова реклама рухомого рядка ефективна тільки коли правильно підібрані розміри табло. У випадках, коли розмір табло обмежений якими або зовнішніми факторами - вибирати не доводиться, необхідно уміщатися в задані рамки. Але коли місце дозволяє – необхідно серйозно поставитися до підбору табло. Хоча мова може йти не тільки про розмірі, але і кольорі світіння, способах управління і т.д. Найбільш дешевим кольором світіння традиційно є червоний. Найбільш популярним кроком між пікселями є крок 10 мм. Вибір висоти табло, або висоти символу, слід робити виходячи з максимально необхідної відстані читаності інформації на табло. Оціночно, це формула Lv/3 = Hs, де Lv - необхідна відстань читання інформації в метрах (гранична), 3 – коефіцієнт, Hs - необхідна для даної відстані висота символу в сантиметрах.

На рисунку 1.1 наведений приклад відображення символів 16 см і символом 32 см. Різниця сприйняття очевидна. Визначивши необхідну висоту символу і обравши один і розмір всього табло, необхідно вирішити, який із способів завантаження інформації в табло найбільш підходить. Найбільш популярний спосіб (так як найдешевший) - через USB флеш карту. У даного способу багато недоліків і використовувати його рекомендується тільки для табло встановлених низько. Для всіх інших випадків рекомендовано використовувати дротові або бездротові прямі підключення до комп'ютера.


Рисунок 1.1 – Співставлення відображення символів 16 см та 32 см

## 1.2 Сфери використання світлового рухомого рядка

Системи світлового рухомого рядка мають достатньо широке та різнопланове використання:

– для реклами товарів та інформування покупців про акції та знижки в торгових центрах, магазинах;

– в ресторанах і розважальних центрах в цілях залучення відвідувачів;

– інформування клієнтів в банках і офісах за допомогою що біжать рядків;

– у громадському транспорті та таксі;

– забезпечення спортивних змагань на стадіонах і у спортзалах;

– використання що біжать рядків у громадських та адміністративних установах;

– в аеропортах, на залізничних та автовокзалах;

– розміщення рекламної та соціальної інформації в місцях відпочинку на вулицях і площах.

## 1.3 Елементи світлового рухомого рядка

З точки зору крупновузлової збірки, рухомий рядок складається із світлодіодних плат (модулів), контролера, джерел живлення та корпусу. Плата в свою чергу побудована з керуючих мікросхем і світлодіодів (часто званих пікселями). Розглянемо кожен з цих компонентів докладніше.

### *1.3.1 Піксель*

Одиночний світлодіод або група світлодіодів являють собою піксель. Пікселі і формують зображення на рухомому рядку. В стандартних монохромних (один колір світіння) рядках частіше застосовують світлодіоди овальної форми. Це пов'язано з особливостями випромінювання світла діодом. Овальний діод випромінює в широкому вугіллі по горизонталі і в меншому по вертикалі. В повнокольоровому рухомому рядку будь-який його піксель містить сині, зелені та червоні світлодіоди. Якщо у колбі пікселя міститься відразу три кристали (червоний, синій і зелений або RGB) то така компоновка називається "три в одному".

### *1.3.2 Крок пікселів*

Крок пікселів – це відстань між центрами двох сусідніх пікселів. Чим менше крок, тим вища роздільна здатність табло при одному і тому ж розмірі. Роздільна здатність – кількість пікселів по горизонталі на кількість пікселів по вертикалі. Чим цифри більше, тим якісніше картинка на табло.

### *1.3.3 Світлодіодний моділь*

Світлодіоди, встановлені на плати та поміщені в пластиковий корпус, утворює модулі. В монохромних рядках частіше застосовують модулі розміром 320х160 мм. Світлодіоди в модулі захищені герметизуючим компаундом із зовнішньої сторони.

### *1.3.4 Контролер*

Для прийому файлів з даними для відображення і управління пікселями в рухомому рядку застосовується контролер. Контролери навіть одного виробника значно відрізняються один від одного за низкою характеристик. Перерахуємо найбільш суттєві:

– дозвіл (здатність підтримувати роботу певної кількості пікселів);

– спосіб завантаження інформації в контролер (флеш карта, через локальною мережею, сом порт, бездротової);

– кольоровість;

– швидкодія;

– пам 'ять;

– програмне забезпечення.

Всі ці характеристики напряму впливають на ціну контролера.

### *1.3.5 Корпус*

На відміну від відеоекранів, корпус рухомого рядку виконаний значно простіше. Як правило – це несущий двошаровий алюмінієвий профіль, силова частина (стійки для кріплення модулів) і пластикова або сталева задня панель.

## 1.4 Характеристики світлових рухомих рядків

### *1.4.1 Дозвіл рухомого рядка*

Для відображення простих текстів достатньо дозволу та 64x8 пікселів. Природно, що чим вище дане значення - тим краще (або більше) картинка, тим більше тексту можна вмістити у видимій області.

### *1.4.2 Яскравість*

Ця характеристика очевидна, але є однією з найважливіших. Від неї багато залежить ефективність самого пристрою, його інформативність . Для вулиць необхідний запас яскравості не менше 3000 Кд на квадратний метр.

### *1.4.3 Кут огляду*

Середні значення кута огляду постійно збільшуються. З кожним роком освоюються нові технології створення кристалів, що і зумовлює поліпшення характеристик вироблених світлодіодів. На теперішній час, для 2020 року кут огляду по горизонталі понад 120 градусів цілком типове явище. Деяких екземпляри наближаються до 160 градусів. Далі збільшити кут складно, так як ця цифра межує з нормальним сприйняттям оком і вимагає використання вже більш дорогих технологій відображення, яка IPS екрани, що значно збільшує вартість, а також ускладнює умови використання, особливо для виличного виконання.

З іншого боку, основна задача рухомого рядка – привернути увагу, а вже якщо людина зацікавиться інформацією він може самостійно обрати найбільш сприятливе положення та кут зору для найкомфортнішого

сприйняття інформації. В цьому випадку яскравість є більш важливою характеристикою ніж кут огляду.

### *1.4.4 Споживана потужність*

Споживання рухомого рядка залежить від її розміру і типу встановлених світлодіодних плат. В середньому, вуличне табло червоного світіння споживає 360 Вт на квадратний метр.

### *1.4.5 Вага*

Орієнтовна вага квадратного метра для рухомого рядка сучасного виконання складає близько 18 кг.

## 1.5 Постановка завдання для розробки

Метою дипломної роботи є проектування електронних комп'ютерної системи спеціального призначення, окремі вузли яких реалізовано на різних апаратно-програмних платформах. Там, сама апаратна платформа рухомого рядка представляє собою мікроконтролерний модуль, а програмне забезпечення для завантаження даних до рядка функціонуя під управлінням операційної системи Windows.

Завданням даної роботи є розробка комп'ютерної системи рухомого світлодіодного рядка, управління якою здійснюється по послідовному інтерфейсу. Також необхідно реалізувати обмін даними між обома апаратно-програмними платформами.

Пересилання даних на низькому рівні пристрою управління рухомим рядком здійснюється мікроконтролером Atmel AVR ATmega 8 з використанням приймача UART на мікросхемі перетворювача USB UART

FT232RL. У свою чергу, мікросхема FT232RL, при підключенні комп’ютерної системи рухомого рядка до комп'ютера через USB, за допомогою драйвера створює в комп'ютерній системі віртуальний COM-порт. Отже, передача даних на високому рівні відбувається так само, як і передача даних за допомогою COM-порту. Розглянемо особливості роботи з універсальним асинхронним приймачем-передавачем (UART) і роботу COM-порту.

### 1.6 Універсальний асинхронний приймач-передавач (UART)

Універсальний асинхронний приймач-передавач (англ. Universal Asynchronous RECEIVER Transmitter (UART)) – це елемент комп’ютерних систем, призначений для зв'язку з іншими периферійними пристроями та системами. Цей інтерфейс перетворює заданий набір даних у послідовний вигляд так, щоб було можливо передати їх у однодротовій цифровій лінії (або бездротовій лінії) іншому аналогічного устрою. При цьому інтервали часу між які передаються блоки даних не є постійними. Ці блоки виділяються за допомогою стартових і стопових бітів (без ліміту часу передачі даних). Метод перетворення добре стандартизований і широко застосовується в комп’ютерній техніці.

Передача даних в UART здійснюється по одному біту за рівні проміжки часу. Цей часовий проміжок визначається заданою швидкістю UART і для конкретного з'єднання вказується в бодах (бітах в секунду).

Крім власне інформаційного потоку, UART автоматично вставляє в потік синхронизируючі мітки, так звані, стартовий і стоповий біти. При прийомі ці зайві біти видаляються з потоку. Зазвичай стартовий і стоповий біти обрамляють один байт інформації (8 бітів, рисунок 1.2). Проте трапляються реалізації UART, які дозволяють передавати по 5, 6, 7, 8 або 9 біт. Обрамлені стартовим і стоповим біти є мінімальної посилкою. Деякі з

реалізацій UART дозволяють вставляти два стопових біта при передачі для зменшення ймовірності розсинхронізації приймача і передавача при щільному трафіку. Приймач ігнорує другий стоповий біт, сприймаючи його як коротку паузу на лінії.

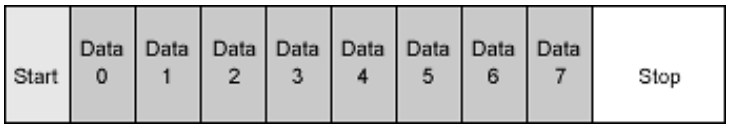


Рисунок 1.2 – Посилка байту даних, обрамленого стартовим і стоповим бітами

UART - повнодуплексний інтерфейс, тобто приймач і передавач можуть працювати одночасно, незалежно один від одного. За кожним з них закріплено порт – один вивід мікроконтролера. Порт приймача позначають RX, передавача – TX. Послідовною установкою рівнів на цих портах щодо загального дроту ("землі") і передається інформація.

Перед початком зв'язку між двома пристроями необхідно налаштувати їх приймачі на однакову швидкість передавання і формат кадру.

Для формування часових інтервалів, що передавач і приймач UART мають джерело точного часу (иактування). Точність цього джерела повинна бути такою, щоб сума похибок (приймача і передавача) установки інтервалу від початку стартового імпульсу до середини стопових бітів не перевищувала половини (а краще хоча б чверті) частоти вибірки інтервалу. Для 8 бітів посилки 0,5/9,5 = 5% (в реальності не більше 3%). Оскільки ця сума помилок приймача і передавача плюс можливі спотворення сигналу в лінії, то рекомендований допуск на точність тактування UART не більше 1,5%.

Оскільки синхронізуючі біти займають частину бітової швидкості, то результуюча пропускна здатність UART не дорівнює швидкості з'єднання.

Наприклад, для 8 бітових посилок синхронізуючі біти займають 20% потоку, що для фізичної швидкості 115200 бод дає бітову швидкість передавання даних 92160 біт/с або 11520 байт/с.

Існує загальноприйнятий ряд стандартних швидкостей: 300 бод, 600 бод, 1200 бод, 2400 бод, 4800 бод, 9600 бод, 19200 бод, 38400 бод, 57600 бод, 115200 бод, 230400 бод, 460800 бод, 921600 бод.

Приймач і передавач тактуються, як правило, з 16 кратною частотою щодо бодрейту. Приймач, піймавши спадаючий фронт стартового біту, відраховує кілька тактів і наступні три такту зчитує (семплірує) порт RX. Це якраз середина стартовго біту. Якщо більшість значень семплів дорівнює "0", стартовий біт вважається таким, що відбувся, інакше приймач приймає його за шум і чекає наступного спадаючого фронту. Після вдалого визначення стартового біту, приймач точно також семплірує серединки бітів даних і по більшості семплів вважає біт "0" або "1", записуючи їх до регістру зсуву. Стоп біти теж семплюються, і якщо рівень стоп біта не "1", UART визначає помилку кадру і встановлює відповідний прапор в керуючому регістрі (рисунок 1.3).

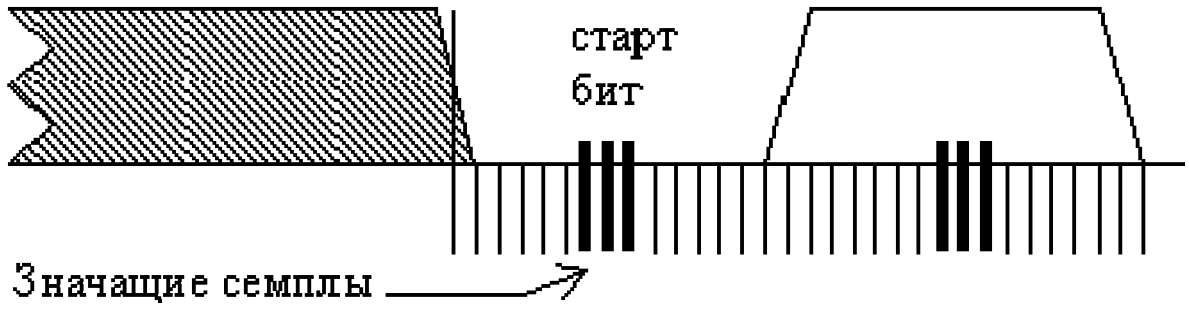


Рисунок 1.3 – Семплювання бітів

Багато реалізації UART мають можливість автоматично контролювати цілісність даних методом контролю бітової парності. Коли

ця функція включена, останній біт даних (біт парності) контролюється логікою UART і містить інформацію про парність кількості одиничних біт в посилці.

## 1.7 Послідовний COM-порт

### *1.7.1 Загальні відомості*

Послідовна лінія (COM-порт, англ. communications port) – двонаправлений послідовний інтерфейс.

Послідовним даний порт називається тому, що інформація через нього передається по одному біту, битий за бітом (на відміну від паралельного порту). Хоча деякі інші інтерфейси комп'ютера, такі як Ethernet, FireWire і USB, також використовують послідовний спосіб обміну, назва "послідовний порт" закріпилося за портом, таким, що має стандарт RS-232c.

Особливістю даного порту порівняно з іншими послідовними технологіями є факт відсутності будь-яких часових вимог між 2 байтами. Часові вимоги є тільки між бітами одного байту (включаючи старт, стоп і парність), величина, зворотна часової паузи між бітами одного байту, називається baud rate – швидкість передачі. Також в цій технології відсутнє поняття "пакет".

Інші послідовні технології, такі, як X.25, USB або Ethernet, мають поняття "пакет", і накладають жорсткі часові вимоги між усіма бітами одного пакету.

*1.7.2 Технічні характеристики COM-портів*

Серед основних характеристик послідовного інтерфейсу COM можна відзначити наступні.

– Тип роз'єму: DE9p (DB9P) або DB25P male, відповідна частина DE9s (DB9s) або DB25s femini (рисунки 1.4 і 1.5 – дев'яти і двадцятип'ятиконтактні роз'єми, відповідно);

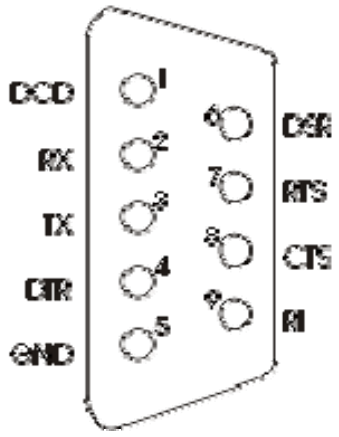


Рисунок 1.4 – Роз'єм DB-9S (DB-9P), DE-9S (DE-9P)

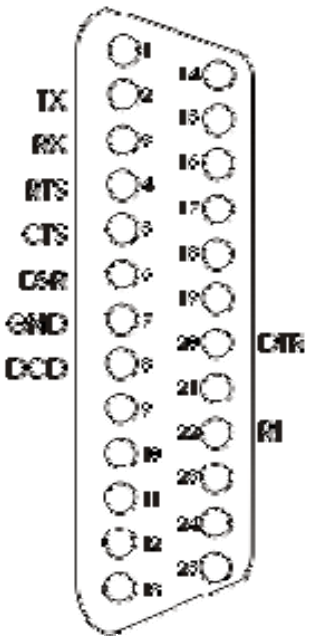


Рисунок 1.5 – Роз'єм DB-25S (DB-25P)

- Апаратна реалізація: мікросхеми UART Intel 8250/16450/16550;
- Рівень сигналу для TxD, RxD: 1 = -3...-12 В; 0=+3...+12 В (сигнали інвертовані);
- Рівень сигналів RTS, DTR, CTS, DSR, DCD, RI: 1 (True)= +3...+12 В ; 0 (False) = -3...-12 В;
- Зона несприйняття сигналів: -3...+3 В;
- Кілкість портів IBM XT: чотири COM1, COM2, COM3, COM4;
- Адреси в просторі вводу/виводу: COM1=3F8h, COM2=2F8h, COM3=3E8h, COM4=2E8h;
- Апаратні переривання: COM1, COM3= IRQ4 (IQ11) COM2, COM4= IRQ3 (IQ10);
- Функції BIOS: 14h (ініціалізація, запис, читання, опитування стану, настройка);
- Стандартна швидкість, біт/сек: 50, 75, 110, 150, 300, 600, 1200, 1800, 2000, 2400, 3600, 4800, 7200, 9600, 14400, 19200, 28800, 38400, 57600, 115200;
- Максимальна швидкість, біт/сек: 1 500 000;
- Кількість біт даних в переданому символі: 5,6,7,8;
- Довжина стопового бита: 1, 1.5, 2;
- Режими контрольного біту (Parity): N(None), E(Even), M(Mark), O(Odd), S(Space);
- Режими синхронізації обміну (Handshaking): 0-None, 1-XOnXoff, 2-RTS, 3-RTSXOnXoff;
- Канал передачі даних (інверсний) : TxD (3)-GND(5);
- Канал прийому даних (інверсній): RxD(2)-GND(5);
- Вихідні сервісні сигнали: RTS(7)-CND(5); DTR(6)-GND(5);
- Вхідні сервісні сигнали: CTS(8)-GND(5); DSR(6)-GND(5); DCD(1)-GND(5); RI(9)-GND(5);

– Відстань зв’язку: стандартна – 25ft (7.62м), максимальна (визначена багатьма факторами).

### *1.7.3 Порядок обміну по інтерфейсу RS-232C*

Порядок обміну по інтерфейсу RS-232 зведений в таблицю 1.1.

Таблиця 1.1 –Порядок обміну даними по інтерфейсу RS-232

| Найменування | Напрямок | Опис | Контакт (25 pin) | Контакт (9-pin) |
|---|---|---|---|---|
| DCD | IN | Carrie Detect (Визначення несучей) | 8 | 1 |
| RXD | IN | Receive Data (Отримані дані) | 3 | 2 |
| TXD | OUT | Transmit Data (передані данні) | 2 | 3 |
| DTR | OUT | Data Terminal Ready (Готовність терміналу) | 20 | 4 |
| GND | - | System Ground (Корпус системи) | 7 | 5 |
| DSR | IN | Data Set Ready (Готовність даних) | 6 | 6 |
| RTS | OUT | Request to Send (Запрос на відправку) | 4 | 7 |
| CTS | IN | Clear to Send (Готовність на прийняття) | 5 | 8 |
| RI | IN | Ring Indicator (Індикатор) | 22 | 9 |

### *1.7.4 Програмування в операційній системі Windows*

У системах Windows 2000 і вище, COM1 - COM4 не має стандартних адресів вводу-виводу і стандартних переривань. На відміну від MS-DOS, Windows автоматично розподіляє ресурси для COM портів. Також Windows не дає прямої можливості працювати з портами вводу-виводу, це можливо тільки при програмуванні на рівні ядра операційної системи. Але розробники Windows передбачили можливість роботи з комунікаційними портами через інтерфейс користувача Windows. У Windows до COM порту можна звернутися як до файлу (потоку). Перевага цього способу очевидні: немає необхідності думати про тип мікросхеми UART, про номери портів

вводу-виводу і про номери переривань. Операційна система непомітно для програміста працює з апаратною частиною комунікаційного порту.

Програмування в операційних системах Windows можна здійснювати за допомогою API функцій або за допомогою зовнішніх компонент ActiveX.

Розглянемо Докладніше програмування COM порту за допомогою API функцій Windows.

1. Для роботи з сом портом перше, що потрібно, це відкрити порт.

Для відкриття порту використовується функція CreateFile. Ця функція надається Win32 API. Її прототип виглядає так:

```
HANDLE CreateFile(
        LPCTSTR    lpFileName,
        DWORD      dwDesiredAccess,
        DWORD      dwShareMode,
        LPSECURITY_ATTRIBUTES
lpSecurityAttributes,
        DWORD      dwCreationDistribution,
        DWORD      dwFlagsAndAttributes,
        HANDLE     hTemplateFile
            );
```

Параметри функції ***CreateFile***:

- **lpFileName** - ім'я COM порту. Може приймати значення: "COM1", "COM2", "COM3", "COM4", "COM5", "COM6", "COM7", "COM8", "COM9", якщо більше однієї цифри, то у форматі «\\.\COM47»
- **dwDesiredAccess** - режим доступ до файлу. Це число в чотири байти, яке задає різні режими доступ до файлу. В поточному випадку для рішення поставленої задачі, цікавить тільки режим читання та запису, цей

режим задається числом: C0000000hex. В C можна замість цифри записати константу з ім'ям "GENERIC_READ | GENERIC_WRITE".

- **dwShareMode** - режим спільного доступу. COM порти комп'ютерної системи не підтримують спільний доступ, тільки одна програма може відкрити порт. Тому цей пункт повинен бути рівний 0 (режим заборонений).
- **lpSecurityAttributes** - атрибути захисту файлу. Для COM портів не використовується тому завжди дорівнює 0 ("NULL").
- **dwCreationDistribution** - управління режимом avtosozdaniya файла. Це chetyrekhbaytovoye число, яке для COM портів завжди повинно бути 00000003hex ("OPEN_EXISTING");
- **dwFlagsAndAttributes** - задає атрибути створеного файла. Це chetyrekhbaytovoye число, яке для COM портів завжди повинно бути 0 ("NULL");
- **hTemplateFile** - дескриптор файлу "шаблону" за якої створювався файл. Для com портів не використовується тому завжди рівний 0 ("NULL").

2. Після відкриття COM порту можна передавати і отримувати дані через цей COM порт.

Для передачі даних використовується API функція WriteFile з бібліотеки kernel32.

Для прийому даних використовується API функція ReadFile з бібліотеки kernel32.

3. Відкритий порт повинен бути закритий перед завершенням роботи програми. У Win32 закриття об'єкта за його дескриптором виконує функція CloseHandle:

```
BOOL CloseHandle (HANDLE hObject);
```

4. Параметри режиму роботи COM порту здійснюється за допомогою структур даних, які являють собою набір змінних різного типу. Структури завантажуються і читаються за допомогою API функцій.

# 2 ОПИС АПАРАТНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНОЇ СИСТЕМИ УПРАВЛІННЯ РУХОМИМ РІДКОМ

В цієї дипломної роботи розробляється комп'ютерна система управління рухомим рядком на соснові мікроконтролера. Принципова схема апаратного забезпечення комп'ютерної системи наведена на рисунку 2.1.

Розглянемо роботу пристрою.

Управління комп'ютерною системою здійснює мікроконтролер (МК) AVR Atmega8 фірми Atmel (DD1). Текст рухомого рядка відображається на світлодіодних матрицях, розмірністю 8х8 (DD4 і DD7).

Виводи мікроконтролерного порту С з'єднані зі входами дешифратора DD3. В якості дешифратора використовується мікросхема 74HC238 виробництва NXP Semiconductors. З мікроконтролера на ці видоди подається код рядка, який засвічується, а дешифратор у відповідності з цим кодом формує активний рівень (логічну "одиницю") на потрібному виході. Високий потенціал на затворі відповідного транзистора (VT1 - VT8) відкриває цей транзистор та разом з тим і лінія світлодіодної матриці замикається на загальний провід. В якості світлодіодних матриць (DD4 і DD7) обрані матриці RL m1588 і встановлені із загальним катодом у рядку. Отже, якщо подати логічний рівень "нуля" в рядку та рівень "одиниці" у стовпчику, то засвічується необхідний світлодіод. В один момент часу працює тільки один рядок, тобто на дешифратор приходить код послідовного рахунку від 000 до 111. На виходах дешифратора по черзі формується активний рівень. Так як активний рівень тільки на одному виході дешифратора, то відкривається тільки один транзистор і низький рівень формується тільки на виводі одного рядка світлодіодної матриці. В цей час інші виводи рядків світлодіодної матриці відключені.

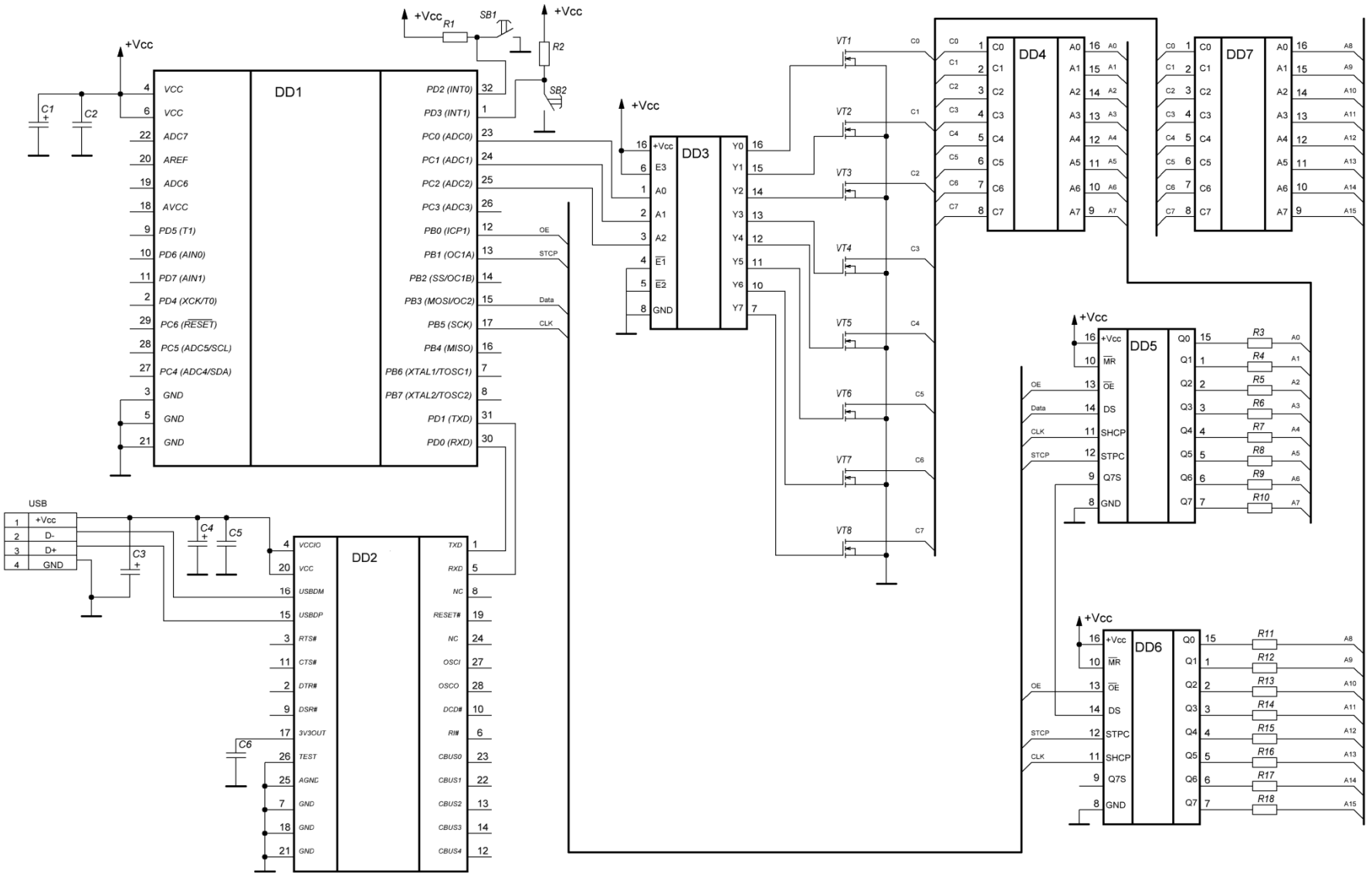


Рисунок 2.1 – Схема апаратного забезпечення комп’ютерної системи управління рухомим рядком

Формування логічних "одиниць" в стовпчиках світлодіодних матриць для засвічування потрібних світлодіодів відбувається за допомогою зсувних регістрів DD5 і DD6. В якості зсувних регістрів обрані мікросхеми 74HC595 фірми NXP Semiconductor. Порт В мікроконтролера керує завантаженням і виведенням даних з регістрів на світлодіодні матриці.

Вивід мікроконтролера PB3 (MOSI) з'єднується зі входом даних DS (serial data input) зсувного регістру і відправляє в регістр послідовність логічних "нулів" і "одиниць", тобто завантажує в регістри дані, які потрібно вивести на світлодіодні матриці.

Вивід PB5 (SCK) з'єднується зі входами SHCP (shift register clock input), служить для синхронізації послідовної відправки і завантаження даних в зсувні регістри.

Вивід мікроконтролера PB1 з'єднується зі входами регістрів STCP (storage register clock input) - тактовий вхід регістра, який видає комбінацію, що зберігається в ньому, на вихід (за умови, що на вхід OE (output enable) подано активний рівень сигналу).

Вивід мікроконтролера PB0 з'єднується зі входами регістрів OE, який при подачі на нього активного рівня сигналу (в даному випадку активний рівень - логічний "нуль") дозволяє включити виходи регістрів Q0 - Q7. Зсувні регістри з'єднані між собою послідовно з допомогою додаткового виходу зсувного регістру Q7S. Дані, що надходять через вхід DS, за допомогою виходу Q7S будуть пересуватися на другий зсувний регістр.

Для регулювання швидкості відображення рухомого рядку використовуються кнопки SB1 і SB2, що підключаються до виводів порту мікроконтролера INT1 і INT0 відповідно.

Мікросхема DD2 – перетворювач USB-UART. Перетворює інтерфейс USB в послідовний інтерфейс UART (The Universal Asynchronous serial Receiver and Transmitter). В якості такого перетворювача обрана

мікросхема FT232RL виробництва фірми FTDI Chip. Виводи RXD (Receiving Asynchronous Data, Input - дозвіл на отримання даних) та TXD (Transmit Asynchronous Data, Output - дозвіл на передачу даних) мікросхеми з'єднані з виводами UART мікроконтролера PD1 (TXD) і PD0 (RXD) відповідно.

# 3 КОНЦЕПТУАЛЬНА ДІАГРАМА ПРОЦЕСІВ В КОМП'ЮТЕРНІЙ СИСТЕМІ УПРАВЛІННЯ РУХОМИМ РЯДКОМ

Концептуальна діаграма комп'ютерної системи, що розробляється, представлена на рисунку 3.1.

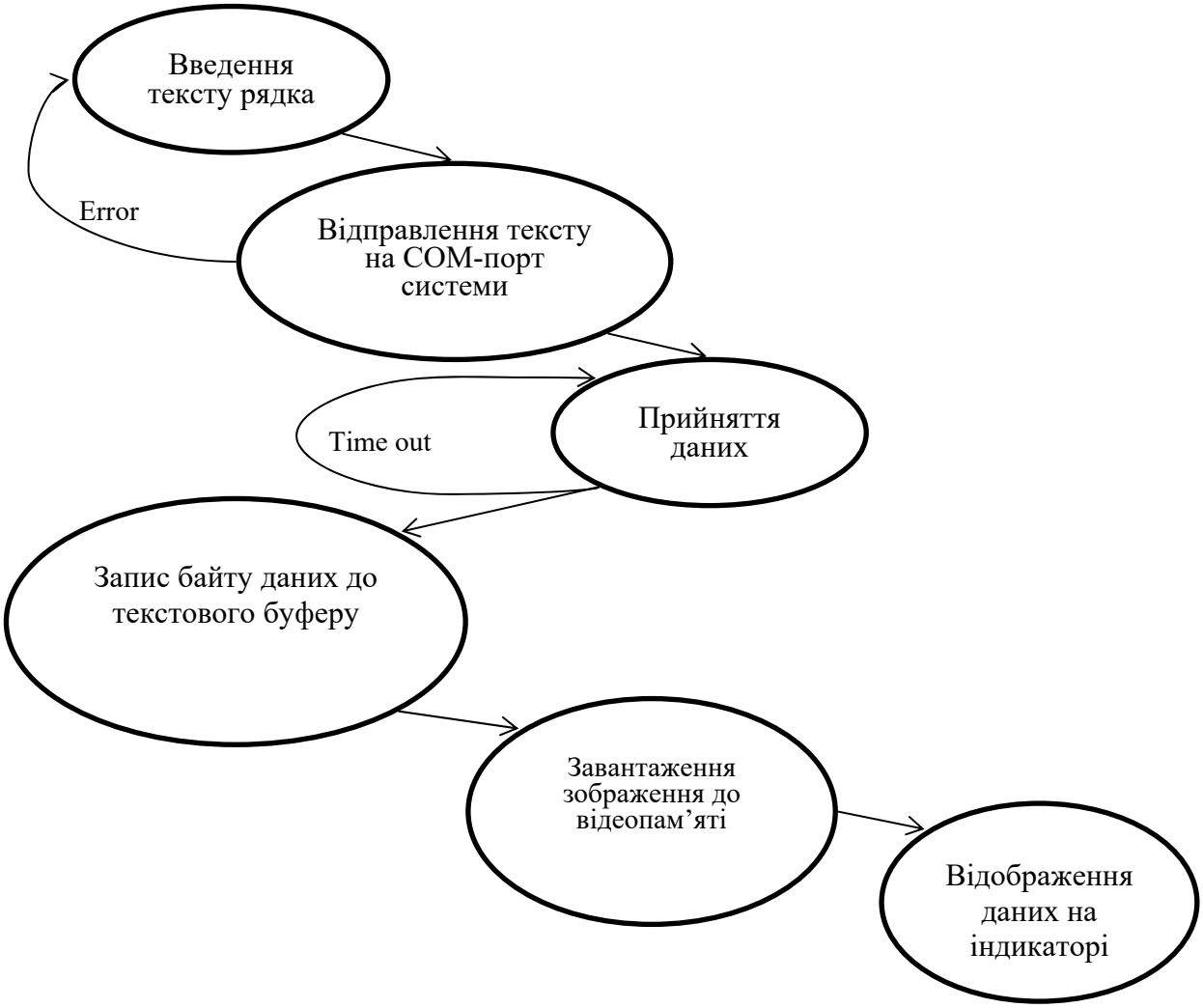


Рисунок 3.1 – Концептуальна діаграма процесів в комп'ютерній системі

В комп'ютерній системі, що розробляється, передбачена функція вводу та відправки необхідного тексту рухомого рядка на світлодіодну індикацію. Якщо відправка на COM-порт системи пройшла успішно,

триває робота на виведення тексту на світлодіодну індикацію. При цьому комп’ютерна система завжди знаходиться в режимі очікування. Якщо сталася помилка на COM-порті, то додаток попереджає про помилку користувача і про необхідність відправки текстового рядку заново.

Комп’ютерна система управління рухомим рядком здійснює виведення даних на світлодіодну індикацію в три етапи:

– запис байту даних в текстовий буфер;
– завантаження зображення символу у відеопам'ять;
– відображення даних на світлодіодних індикаторах.

# 4 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 4.1 Розробка структури алгоритму програмного забезпечення відправки даних для відображення на рухомому рядку

Структура основної нитки процесу алгоритму програмного забезпечення модулю відправки текстових даних для відображення представлена на рисунку 4.1.

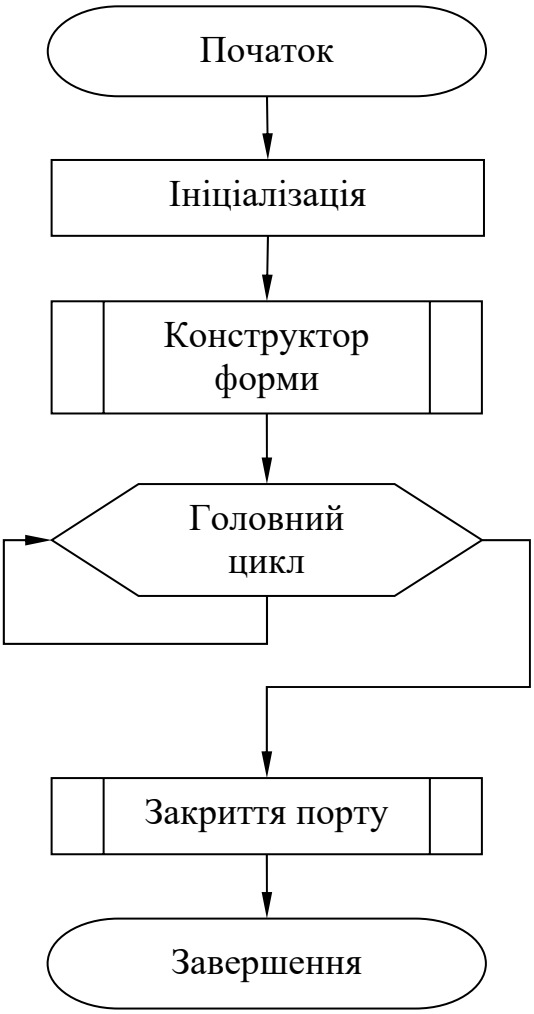


Рисунок 4.1 – Структура основної нитки процесу алгоритму програмного забезпечення модулю відправки текстових даних

Структура основної нитки процесу програмного забезпечення модулю відправки текстових даних на початковому етапі містить процес ініціалізації, який полягає у:

– створенні дескриптора порту, через який буде здійснюватися зв’язок та передача даних до комп’ютерної системи рухомого рядка;

– ініціалізація класу для настройки комунікаційного інтерфейсу порту;

– резервування змінної, що представлятиме кількість байт, що передаються;

– резервування змінної-буфера, яка буде містити строку тексту, що передається для відображення на рухомому рядку;

– резервування змінної-буфера, яка буде містити кількість символів у рядку, що надається для відображення на рухомому рядку;

– змінної-буфера, яка міститиме поточній байт, що передається по комунікаційному інтерфейсу.

Після процесу ініціалізації, в конструкторі основної форми проекту включаються наступні дії щодо настроювання комунікаційного інтерфейсу порту вводу-виводу. Докладніше цей процес представлений у структурі алгоритму підпрограми конструктора форми, що наведена на рисунку 4.2.

Після настроювання комунікаційного інтерфейсу порту вводу-виводу активується головний цикл програмного забезпечення в якому очікується виконання подій, що ініціює користувач засобами взаємодії з графічним інтерфейсом програмного забезпечення. Ці дії полягають у відправки текстових даних для відображення на рухомому рядку, очищення даних для пересилки та виклик довідки. В процесі завершення роботи програми здійснюється вихід з головного циклу та закриття комунікаційного інтерфейсу порту вводу-виводу та знищення дескриптора порту в системному реєстрі операційної системи.

Структура алгоритму підпрограми конструктора форми наведена на рисунку 4.2.

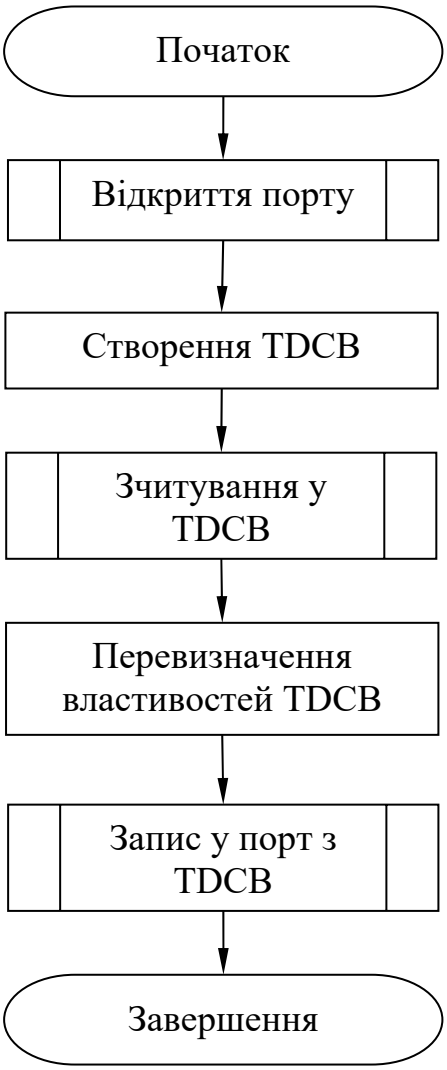


Рисунок 4.2 – Структура алгоритму підпрограми настроювання комунікаційного порту вводу виводу в конструкторі головної форми проекту

Представлена структура алгоритму включає наступні операції:

– відкриття комунікаційного інтерфейсу порту вводу виводу та отримання поточного його дескриптора для подальшого використання для адресування процесу обміну даними;

– створення екземпляру класу для отримання комунікаційних налаштувань відкритого порту вводу-виводу;

– зчитування поточних налаштувань комунікаційного інтерфейсу відкритого порту вводу виводу та зберігання їх у створеному спеціальному класі;

– перепризначення необхідних властивостей класів, о відповідають необхідним комунікаційним налаштуванням інтерфейсу для узгодженого процесу передавання даних між керуючою програмою та комп’ютерною системою рухомого рядка;

– запис модифікованого класу налаштувань до комунікаційного інтерфейсу відкритого порту вводу виводу та завершення налаштувань.

Надалі, в головному циклі програмного забезпечення здійснюється очікування ініціації користувачем одного з наступних програмних подій:

– натискання на кнопку довідки;

– натискання на кнопку очищення текстової строки, що передається;

– натискання на кнопку завершення роботи додатку;

– натискання на кнопку пересилки поточної текстової строки до комп’ютерної системи рухомого рядку.

Дії, що виконуються у випадку натискання кнопки довідки полягають у відображенні додаткової форми, на якій розміщується інформація стосовно того, як розроблений додаток працює.

Натискання на кнопку очищення даних ініціює видалення даних з буферу передачі, що дозволяє надалі заповнювати його новими даними.

Процес пересилки даних до комп’ютерної системи рухомого рядка представлений в структурі алгоритму на рисунку 4.3. Він базується на розкладанні строкової змінної, що задана у однобайтовому форматі ASCII, до послідовності байтів, що містять код кожного символу у послідовності.

Спочатку на основі даних, введених користувачем визначається значення змінної стокового типу, дині якої повинні бути побайтово

переслані по універсальному асинхронному послідовному приймачузпередавачу.

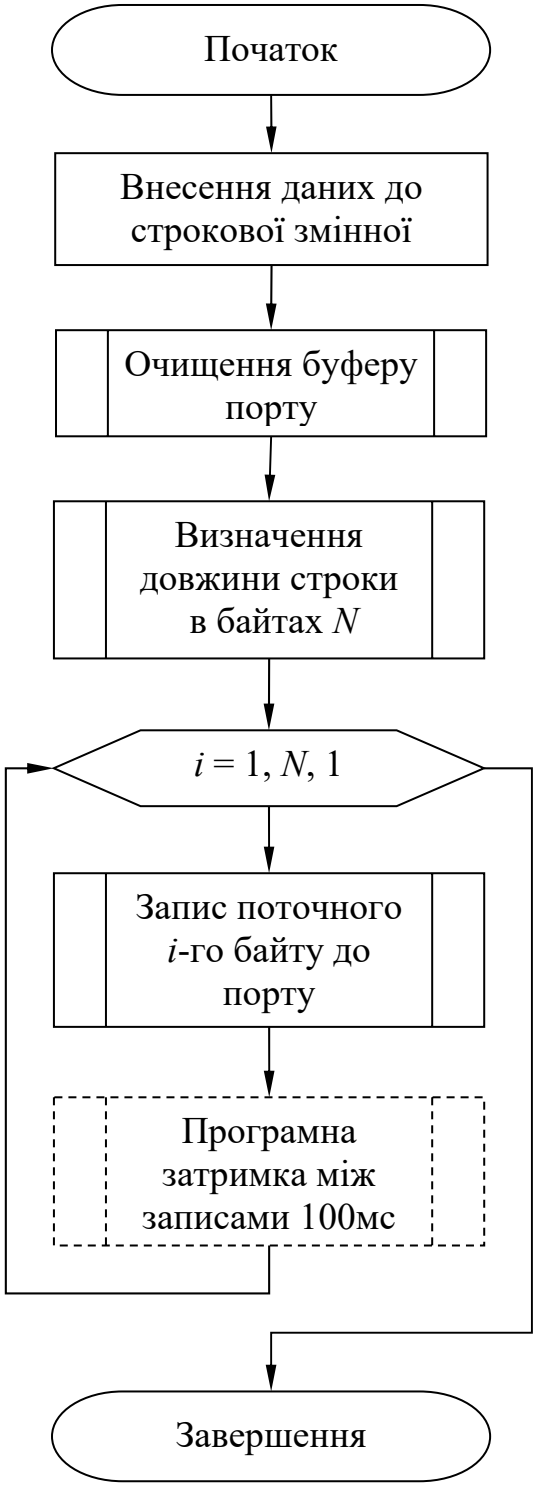


Рисунок 4.3 – Структура алгоритму підпрограми відправки текстових даних для відображення на рухомому рідку

Надалі, отримана строкова змінна, піддається обробки для визначення кількості символів, що містяться в неї. Враховуючи той факт, що в кодуванні ASCII кожен символ представляється одним байтом, то кількість символів буде відповідати і кількості байтів даних, що підлягають пересилці за послідовним портом вводу-виводу.

Наступним етапом здійснюється формування циклу на фіксовану кількість повторень, яка визначається кількістю байтів у рядку, що надсилається для відображення. Таким чином, в циклі ініціюється *i* викликів API функції запису даних до порту вводу виводу.

Таким чином, в результаті виконання циклу, здійснюється послідовна пересилка усієї послідовності символів у строковому рядку.

На етапі налагодження програмного забезпечення для наочності та перевірки процесу побайтового пересилання, до тіла циклу вводиться додаткова програмна затримка між етапами пересилання кожного байту. В релізній версії програмного забезпечення ця програмна затримка може бути видалена для прискорення передачі даних та зниження часу зайняття системного порту комп’ютерної системи прикладною програмою відправки даних на комп’ютерну систему рухомого рядка.

## 4.2. Розробка програмного забезпечення відправки даних для відображення на рухомому рядку

Відповідно до розробленого алгоритмоме роботи додатка і пристрою, здійснюємо розробку коду програмного забезпечення.

Програмне забезпечення для відправки текстових рядків на пристрій управління рухомим рядком реалізована на мові C++ в середовищі програмування Borland C++ Builder.

На рисунку 4.4 показаний вигляд графічного інтерфейсу головного вікна програмного забезпечення для відправки даних до комп’ютерної системи рухомого рядка.

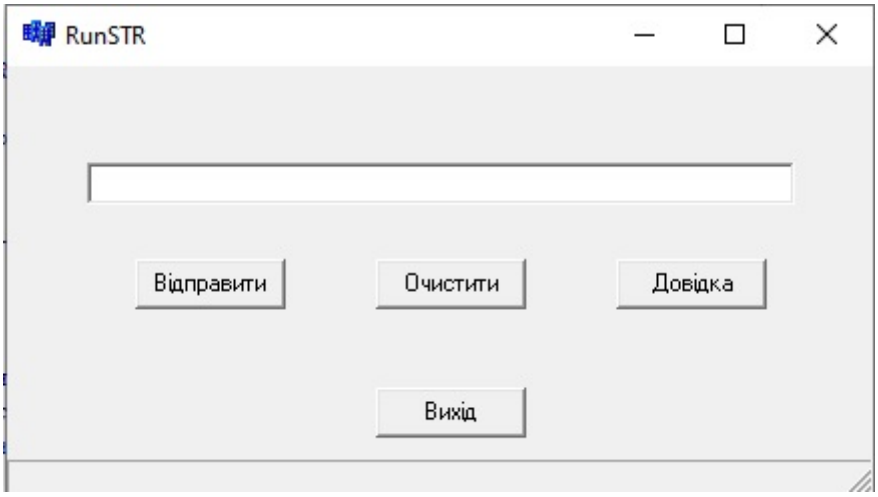


Рисунок 4.4 – Вигляд головного вікна додатка управління рухомим рядком

На формі «RunSTR» розташовані такі елементи:

– **Edit (TextRun)** – текстове поле для безпосереднього введення тексту рухомого рядка. Програмне забезпечення комп’ютерної системи управління рухомим рядком передбачає, що текст необхідно вводити великими літерами літерами: кирилицею і латиницею, також підтримуються цифри та деякі додаткові символи.

– **Button (Send)** – кнопка «Відправити». За натисканням цієї кнопки здійснюється відправка ASCII коду кожного символу текстового рядка на порт вводу-виводу.

– **Button (Clear)** – кнопка «Очистити». За натисканням цієї кнопки здійснюється очистка текстового поля TEdit1 Edit1.

– **Button (Exit)** – кнопка «Вихід». За натисканням цієї кнопки здійснюється завершення роботи додатку та закриття порту вводу виводу.

– **Button (?) –** кнопка, при натисканні якої з'являється вікно з довідкою за програмою. Вигляд вікна довідки представлений на рисунку 4.5.

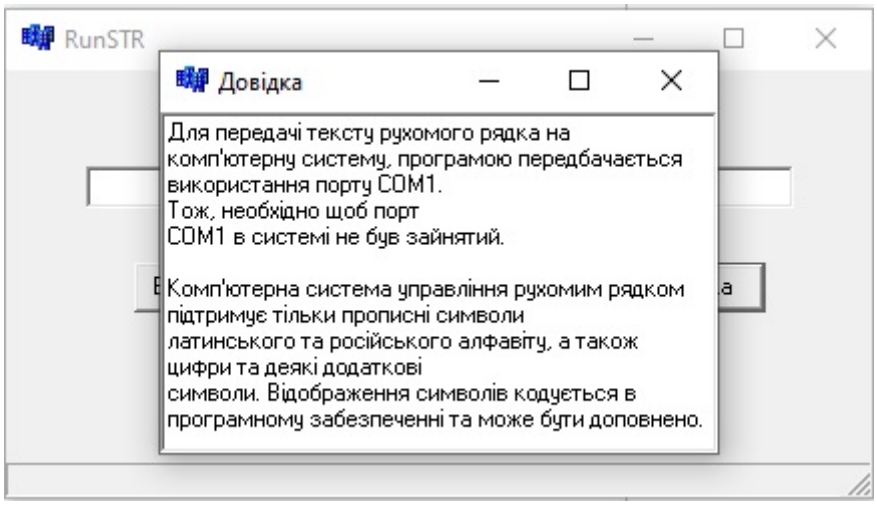


Рисунок 4.5 – Вигляд вікна довідки програми управління комп’ютерною системою відображення рухомого рядка

– **StatusBar**. Якщо відправка даних на порт вводу-виводу неможлива, то в рядку стану (внизу головної форми додатка) виводиться відповідна помилка. В цьому випадку необхідно звільнити порт, що використовується і перезапустити програму.

Текст програми, що реалізує даний додаток зі вказаним функціоналом приведений нижче.

```
//---------------------------------------------------------------------
#include <vcl.h>
#include <winbase.h>
#pragma hdrstop
#include "Unit1.h"
#include "Unit2.h"  // підключення другої форми
//---------------------------------------------------------------------
```

```
-------
#pragma package(smart_init)
#pragma resource "*.dfm"

HANDLE port;       // дескриптор порту

DCB dcb;           // структура ініціалізації порту

int n;

char LC;           // змінна, яка збкрігає поточний байт,
                   // що передається
int SL;            // змінна, що зберігає довжину строки

AnsiString Line;   // змінна, що зберігає текст строки

TForm1 *Form1;
//---------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
// Відкриття порту:
port=CreateFile("COM1",GENERIC_WRITE,0,NULL,OPEN_EXISTING,0,NULL);

// Виведення помилки, якщо не вдалося відкрити порт COM1:
 if(port == INVALID_HANDLE_VALUE)
  {
   Form1->StatusBar1->Panels->Items[0]->Text = "Не вдалося відкрити
порт!";
   return;
  }

GetCommState(port, &dcb); // зчитати структуру DCB з порту

dcb.BaudRate=CBR_4800;    // встановити швидкість передачі 4800 бод

SetCommState(port, &dcb); // записати структуру DCB в порт
```

```
}
//---------------------------------------------------------------------
void __fastcall TForm1::ExitClick(TObject *Sender)
{
Application -> Terminate();   //завершення роботи додатку по
                              // натисканню кнопки «Вихід»
}
//---------------------------------------------------------------------
void __fastcall TForm1::ClearClick(TObject *Sender)
{
TextRun -> Clear();       // Очищення текстового поля по натисканню
                          // кнопки «Очистити»
}
//---------------------------------------------------------------------
void __fastcall TForm1::SendClick(TObject *Sender)
{
// По натисканню кнопки «Відправити»:
Line = TextRun -> Text;  // Запис введеного тексту дод змінної Line
FlushFileBuffers(port);  // Очищення буфера порту
SL = Line.Length();      // Отримання довжини строки


for (int i=1; i<=SL; i++)  // Цикл побайтового пересилання строки
                           // символів на порт вводу-виводу
 {
  LC = Line[i];
  WriteFile(port,&LC,1,&DWORD(n),NULL);
  Sleep(100);              // Затримка 100 мс перед відправкою
                           // наступного байту
 }
}
//---------------------------------------------------------------------
void  __fastcall  TForm1::FormClose(TObject  *Sender,  TCloseAction
&Action)
 {
  CloseHandle(port);       // Закрити порт вводу-виводу при закритті
                           // головної форми Form1
```

```
 }
//---------------------------------------------------------------------
void __fastcall TForm1::HelpClick(TObject *Sender)
{
  HelpForm -> Show();  //Показати форму довідки по натисканню
                       // на кнопку «Довідка»
}
//---------------------------------------------------------------------
```

### 4.3 Розробка структури алгоритму програмного забезпечення комп'ютерної системи відображення рухомого рядку

Програмне забезпечення комп'ютерної системи відображення рухомого рядку, за умовчанням, забезпечує прослуховування вхідного порту вводу-виводу через який здійснюється з'єднання з програмним забезпеченням відправки текстових даних.

Після отримання кожного символу, здійснюється його порівняння з наявними у пам'яті і при співпадінні в пам'ять записується 8 байт даних, які описують коди засвічування матриці світлодіодів 8x8. Таким чином, в пам'яті формується інформація для відображення, яка представляє собою байтовий масив, де в суміжних комірках пам'яті послідовно розташовуються групи з 8 байтів, що описують кожний символ, що відображається. При чому, слід зазначити, що кожен байт описує один стовбець символу, що відображається.

Принцип розміщення даних у пам'яті представлений на рисунку 4.6. В якості ілюстрації узяте слово «HELLO», яке буде представлене в пам'яті 5 групами по 8 байт. Кожен байт представляє закодовану послідовність точок які, відповідають засвіченому чи незасвіченому світлодіоду в матриці.

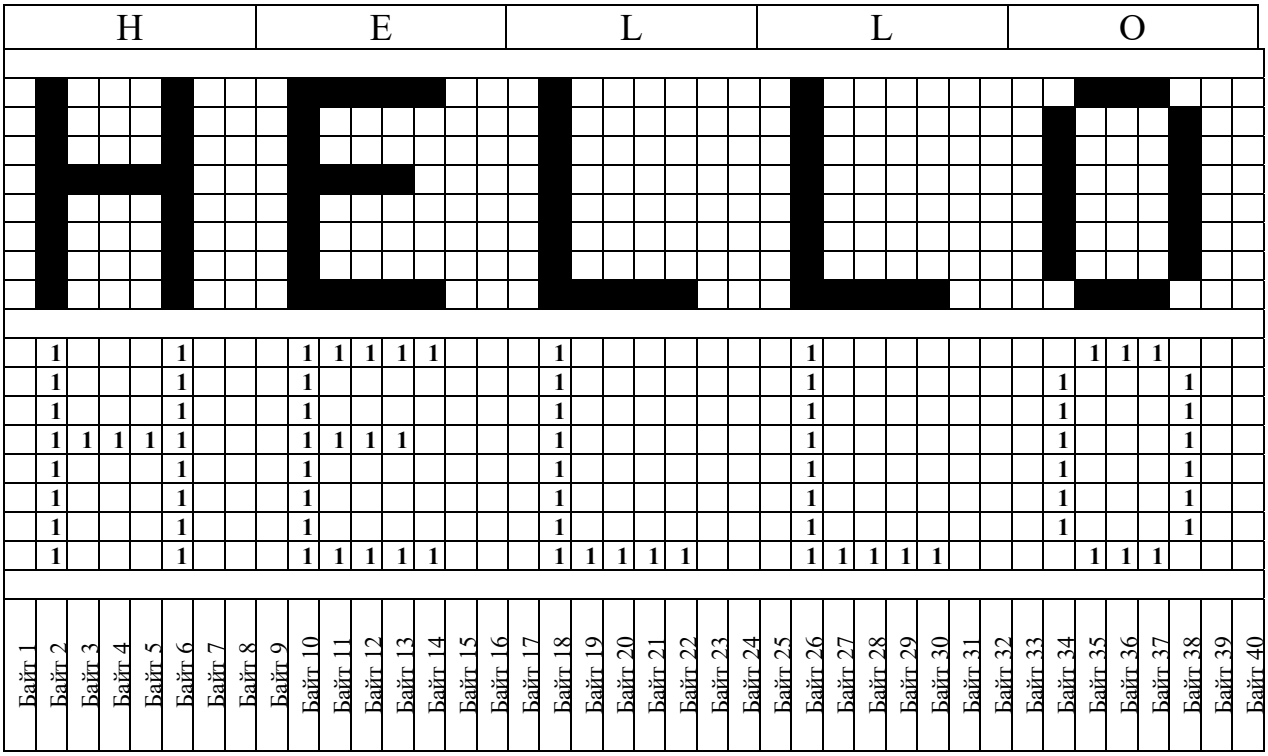


Рисунок 4.6 – Ілюстрація формування послідовності байтів для відображення в рухомому рядку

Так як відповідно до принципової схеми, що наведена на рисунку 2.1, управління здійснюється рядками, що поєднує катоди світлодіодів, то включений стан світлодіоду описується низьким рівнем сигналу на катоді. Так як логічні рівні з виводів мікроконтролера, відповідно до принципової схеми на рисунку 2.1, інвертуються, то це призводить до необхідності прорисовки символів значеннями «1», а неактивний стан світлодіоду – «0». Це і проілюстровано на рисунку 4.6. Таким чином, коли на виводі мікроконтролера формується сигнал логічної одиниці, це відповідає засвіченому стану відповідного світлодіоду.

Так, наприклад, в програмі, формування символу «+» буде виглядати наступним чином:

```
0b00000000
0b00000000
0b00010000
0b00010000
0b01111100
0b00010000
0b00010000
0b00000000
```

Якщо приглядатися до просторової послідовності представлених байт, то можна побачити відображення «+». Так як, відображення ведеться з боку молодших бітів, то символ «+» формується якби з ліва на право.

Структура алгоритму основної нитки процесів в програмному забезпеченні комп’ютерної системи відображення рухомого рядка наведена на рисунку 4.7а і 4.7б.

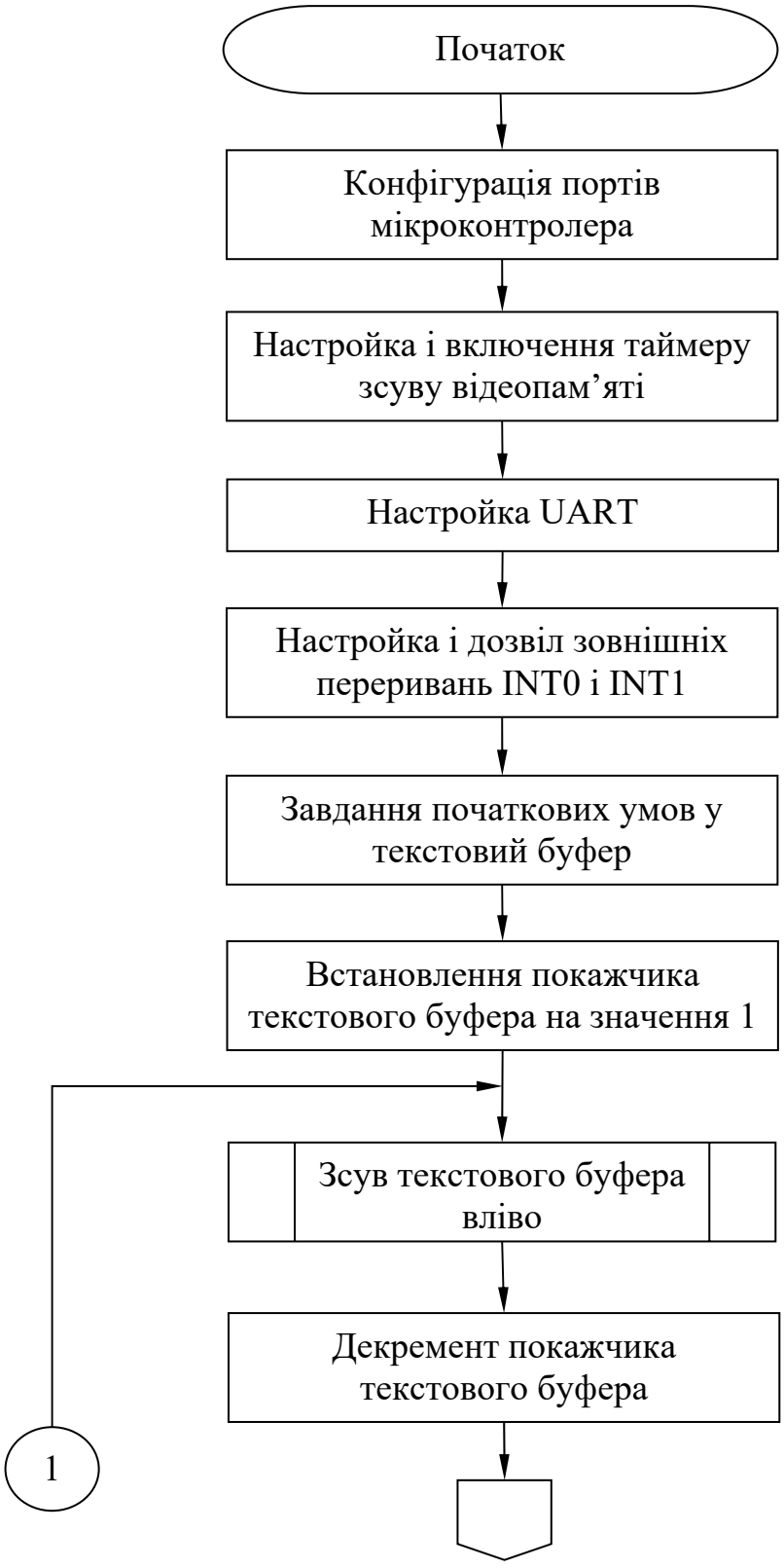


Рисунок 4.7а – Структура алгоритму програмного забезпечення комп’ютерної системи відображення рухомого рядка

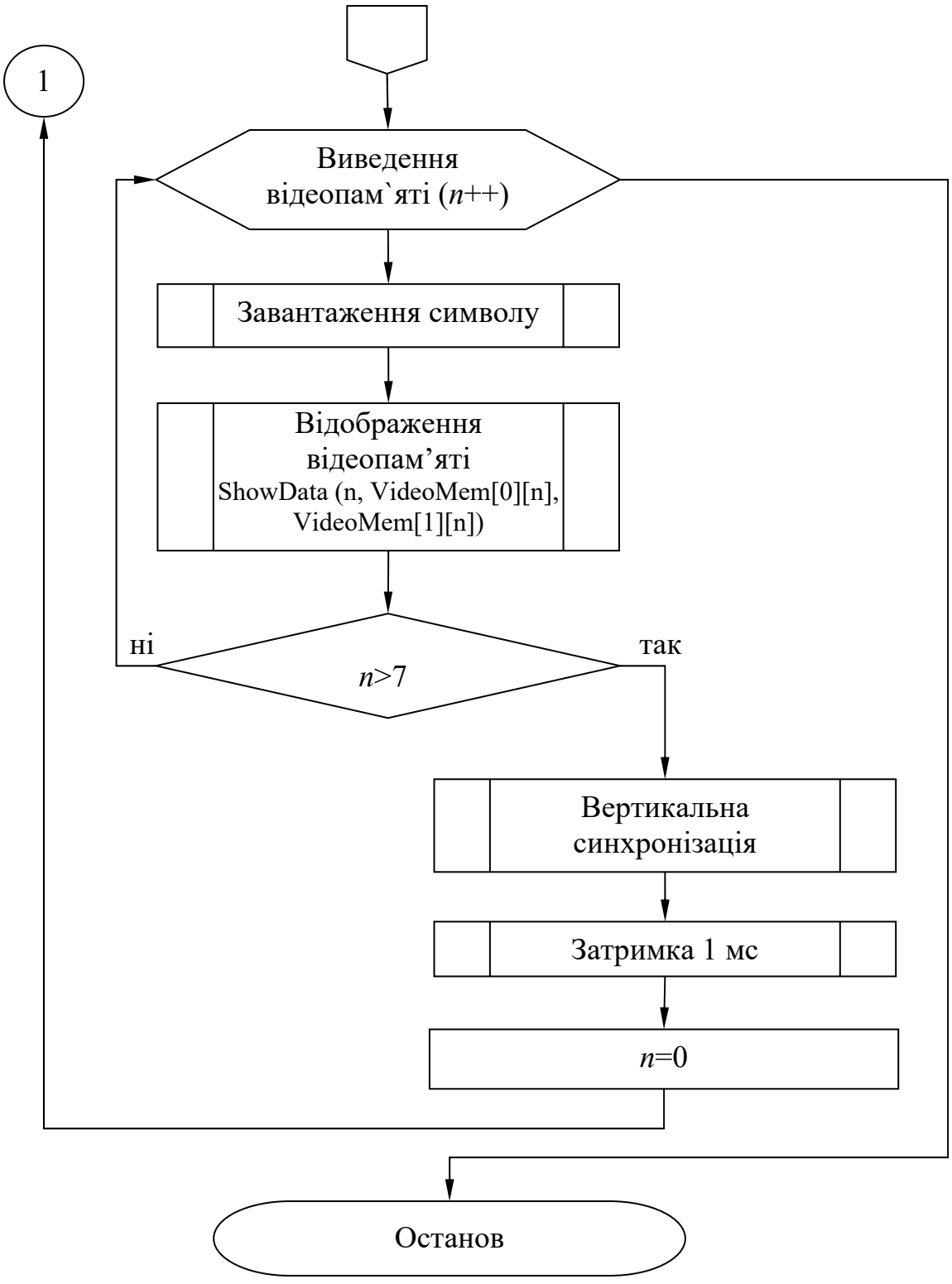


Рисунок 4.7а – Структура алгоритму програмного забезпечення комп’ютерної системи відображення рухомого рядка

## 4.3 Розробка програмного забезпечення комп'ютерної системи відображення рухомого рядка

Розробка програмного забезпечення, відповідно до раніше представленого алгоритму, здійснювалася на мові програмування С в середовищі CodeVision AVR.

Вихідний код реалізований двома модулями:

1) font.c – що представляє собою бібліотеку кодування символів, що можуть бути відображені на дисплеї. Цей вихідний код представляє собою масив бітових слів по 8 в кожній групі для опису символу, що може бути відображений, а також функції, що дозволяє визначити індекс символу в цьому масиві по значенню ASCII-коду, який передається до неї у якості параметра. В бібліотеці реалізована обмежена кількість символів в діапазоні ASCII-кодів від 32 до 95 (спеціальні символи і цифри) і від 192 до 223 – заголовні букви латиниці та кирилиці. Таким чином, цей модуль дозволяє визначити групу з 8 байтів для відображення символу, що переданий ASCII-кодом.

2) RunSTR.c – що є основним модулем програмного забезпечення комп'ютерної системи управління рухомим рядком. До цього модулю підключається модуль font.c.

Програмний код модулю font.c наведений далі у лістингу.

```
/*
 * Підключаємий модуль, в якому описаний масив,
 * що характеризує зображення символів.
 * В даній версії описані зображення 96 символів: спеціальних символів, цифр, заголовних латинських букв A-Z і заголовних букв кирилиці А-Я.
 * в коментарях кожного бітового елементу масиву представлений індекс символа, який повертається функцією
                                char GetSymbolIndex(char SymbolCode)
```

```
 *
 */

char FontArray[96][8] = {
    {               //0        -          пробіл
    0b00000000,
    0b00000000,
    0b00000000,
    0b00000000,
    0b00000000,
    0b00000000,
    0b00000000,
    0b00000000},
    {               //1        -          !
    0b00010000,
    0b00010000,
    0b00010000,
    0b00010000,
    0b00010000,
    0b00010000,
    0b00000000,
    0b00010000},
    {               //2        -          лапки
    0b01101100,
    0b01101100,
    0b00100100,
    0b00000000,
    0b00000000,
    0b00000000,
    0b00000000,
    0b00000000},
    {               //3        -          #
    0b00000000,
    0b00101000,
    0b01111100,
    0b00101000,
    0b00101000,
```

```
    0b01111100,
    0b00101000,
    0b00000000},
    {               //4        -          $
    0b00001000,
    0b00011100,
    0b00101100,
    0b00101000,
    0b00011000,
    0b00001100,
    0b00101100,
    0b00011000},
    {               //5        -          %
    0b00000000,
    0b00000100,
    0b01001000,
    0b00010000,
    0b00100100,
    0b01000000,
    0b00000000,
    0b00000000},

    {               //6        -          &
    0b00111000,
    0b01000100,
    0b00101000,
    0b00010000,
    0b00111000,
    0b01001100,
    0b01001000,
    0b00110100},
    {               //7        -          '
    0b00111000,
    0b00110000,
    0b00100000,
    0b00000000,
    0b00000000,
```

```
    0b00000000,
    0b00000000,
    0b00000000},
    {               //8        -          (
    0b00000100,
    0b00001000,
    0b00010000,
    0b00100000,
    0b00100000,
    0b00010000,
    0b00001000,
    0b00000100},
    {               //9        -          )
    0b01000000,
    0b00100000,
    0b00010000,
    0b00001000,
    0b00001000,
    0b00010000,
    0b00100000,
    0b01000000},
    {               //10       -          *
    0b00000000,
    0b01000100,
    0b00101000,
    0b01111100,
    0b00101000,
    0b01000100,
    0b00000000,
    0b00000000},
    {               //11       -          +
    0b00000000,
    0b00000000,
    0b00010000,
    0b00010000,
    0b01111100,
    0b00010000,
```

```
    0b00010000,
    0b00000000},
    {               //12       -          ,
    0b00000000,
    0b00000000,
    0b00000000,
    0b00000000,
    0b00000000,
    0b00111000,
    0b00011000,
    0b00010000},
    {               //13       -          -
    0b00000000,
    0b00000000,
    0b00000000,
    0b00000000,
    0b01111100,
    0b00000000,
    0b00000000,
    0b00000000},
    {               //14       -          .
    0b00000000,
    0b00000000,
    0b00000000,
    0b00000000,
    0b00000000,
    0b00000000,
    0b00011000,
    0b00011000},
    {               //15       -          /
    0b00000000,
    0b00000000,
    0b00000000,
    0b00000100,
    0b00001000,
    0b00010000,
    0b00100000,
```

```
    0b01000000},

    {               //16       -          0
    0b00111000,
    0b01000100,
    0b01001100,
    0b01010100,
    0b01100100,
    0b01000100,
    0b01000100,
    0b00111000},
    {               //17       -          1
    0b00001000,
    0b00011000,
    0b00101000,
    0b01001000,
    0b00001000,
    0b00001000,
    0b00001000,
    0b0011110},
    {               //18       -          2
    0b00111100,
    0b01000100,
    0b00000100,
    0b00111000,
    0b01000000,
    0b01000000,
    0b01000000,
    0b01111100},
    {               //19       -          3
    0b00111000,
    0b01000100,
    0b00000100,
    0b00111000,
    0b00000100,
    0b00000100,
    0b01000100,
```

```
    0b00111000},
    {               //20       -          4
    0b00000100,
    0b00001100,
    0b00010100,
    0b00100100,
    0b01111100,
    0b00000100,
    0b00000100,
    0b00000100},
    {               //21       -          5
    0b01111100,
    0b01000000,
    0b01000000,
    0b01111100,
    0b00000100,
    0b00000100,
    0b01000100,
    0b00111000},
    {               //22       -          6
    0b00111000,
    0b01000000,
    0b01000000,
    0b01111000,
    0b01000100,
    0b01000100,
    0b01000100,
    0b00111000},
    {               //23       -          7
    0b01111100,
    0b00000100,
    0b00000100,
    0b00000100,
    0b00001000,
    0b00010000,
    0b00100000,
    0b01000000},
```

```
    {               //24       -          8
    0b00111000,
    0b01000100,
    0b01000100,
    0b00111000,
    0b01000100,
    0b01000100,
    0b01000100,
    0b00111000},
    {               //25       -          9
    0b00111000,
    0b01000100,
    0b01000100,
    0b00111000,
    0b00000100,
    0b00000100,
    0b01000100,
    0b00111000},
    {               //26       -          :
    0b00000000,
    0b00011000,
    0b00011000,
    0b00000000,
    0b00000000,
    0b00011000,
    0b00011000,
    0b00000000},
    {               //27       -          ;
    0b00000000,
    0b00011000,
    0b00011000,
    0b00000000,
    0b00000000,
    0b00011000,
    0b00011000,
    0b00001000},
    {               //28       -          <
```

```
    0b00000000,
    0b00001000,
    0b00010000,
    0b00100000,
    0b01000000,
    0b00100000,
    0b00010000,
    0b00001000},
    {               //29       -          =
    0b00000000,
    0b00000000,
    0b00000000,
    0b01111100,
    0b00000000,
    0b01111100,
    0b00000000,
    0b00000000},
    {               //30       -          >
    0b00000000,
    0b01000000,
    0b00100000,
    0b00010000,
    0b00001000,
    0b00010000,
    0b00100000,
    0b01000000},
    {               //31       -          ?
    0b00111000,
    0b01000100,
    0b00000100,
    0b00001000,
    0b00010000,
    0b00010000,
    0b00000000,
    0b00010000},
    {               //32       -          @
    0b00111100,
```

```
    0b01000100,
    0b01011100,
    0b01010100,
    0b01010100,
    0b01011000,
    0b01000000,
    0b00111000},
    {               //33       -          A
    0b00111000,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01111100,
    0b01000100,
    0b01000100,
    0b01000100},
    {               //34       -          B
    0b01111000,
    0b01000100,
    0b01000100,
    0b01111000,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01111000},
    {               //35       -          C
    0b00111000,
    0b01000100,
    0b01000000,
    0b01000000,
    0b01000000,
    0b01000000,
    0b01000100,
    0b00111000},
    {               //36       -          D
    0b01111000,
    0b01000100,
```

```
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01111000},
    {               //37       -          E
    0b01111100,
    0b01000000,
    0b01000000,
    0b01111000,
    0b01000000,
    0b01000000,
    0b01000000,
    0b01111100},
    {               //38       -          F
    0b01111100,
    0b01000000,
    0b01000000,
    0b01111000,
    0b01000000,
    0b01000000,
    0b01000000,
    0b01000000},
    {               //39       -          G
    0b00111000,
    0b01000100,
    0b01000000,
    0b01011000,
    0b01010100,
    0b01000100,
    0b01000100,
    0b00111000},
    {               //40       -          H
    0b01000100,
    0b01000100,
    0b01000100,
```

```
    0b01111100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100},
    {               //41       -          I
    0b00111000,
    0b00010000,
    0b00010000,
    0b00010000,
    0b00010000,
    0b00010000,
    0b00010000,
    0b00111000},
    {               //42       -          J
    0b00001100,
    0b00000100,
    0b00000100,
    0b00000100,
    0b00000100,
    0b01000100,
    0b01000100,
    0b00111000},
    {               //43       -          K
    0b01000100,
    0b01001000,
    0b01010000,
    0b01100000,
    0b01010000,
    0b01001000,
    0b01000100,
    0b01000100},
    {               //44       -          L
    0b01000000,
    0b01000000,
    0b01000000,
    0b01000000,
```

```
    0b01000000,
    0b01000000,
    0b01000100,
    0b01111100},
    {               //45       -          M
    0b01000100,
    0b01101100,
    0b01010100,
    0b01010100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100},
    {               //46       -          N
    0b01000100,
    0b01000100,
    0b01100100,
    0b01010100,
    0b01001100,
    0b01000100,
    0b01000100,
    0b01000100},
    {               //47       -          O
    0b00111000,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b00111000},
    {               //48       -          P
    0b01111000,
    0b01000100,
    0b01000100,
    0b01111000,
    0b01000000,
```

```
    0b01000000,
    0b01000000,
    0b01000000},
    {               //49       -          Q
    0b00111000,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01010100,
    0b00111000,
    0b00000100},
    {               //50       -          R
    0b01111000,
    0b01000100,
    0b01000100,
    0b01111000,
    0b01010000,
    0b01001000,
    0b01000100,
    0b01000100},
    {               //51       -          S
    0b00111000,
    0b01000100,
    0b01000000,
    0b00110000,
    0b00011000,
    0b00000100,
    0b01000100,
    0b00111000},
    {               //52       -          T
    0b01111100,
    0b01010100,
    0b00010000,
    0b00010000,
    0b00010000,
    0b00010000,
```

```
    0b00010000,
    0b00010000},
    {               //53       -          U
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b00111000},
    {               //54       -          V
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b00101000,
    0b00010000},
    {               //55       -          W
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01010100,
    0b01010100,
    0b01101100,
    0b01000100},
    {               //56       -          X
    0b01000100,
    0b01000100,
    0b00101000,
    0b00010000,
    0b00101000,
    0b01000100,
    0b01000100,
```

```
    0b01000100},
    {               //57       -          Y
    0b01000100,
    0b01000100,
    0b01000100,
    0b00101000,
    0b00010000,
    0b00010000,
    0b00010000,
    0b00010000},
    {               //58       -          Z
    0b01111100,
    0b00000100,
    0b00001000,
    0b00010000,
    0b00100000,
    0b01000000,
    0b01000000,
    0b01111100},
    {               //59       -          [
    0b00111000,
    0b00100000,
    0b00100000,
    0b00100000,
    0b00100000,
    0b00100000,
    0b00100000,
    0b00111000},
    {               //60       -          '\'
    0b00000000,
    0b01000000,
    0b00100000,
    0b00010000,
    0b00001000,
    0b00000100,
    0b00000010,
    0b00000000},
```

```
{               //61       -          ]
0b00110000,
0b00010000,
0b00010000,
0b00010000,
0b00010000,
0b00010000,
0b00010000,
0b00110000},
{               //62       -          ^
0b00010000,
0b00101000,
0b01000100,
0b00000000,
0b00000000,
0b00000000,
0b00000000,
0b00000000},
{               //63       -          _
0b00000000,
0b00000000,
0b00000000,
0b00000000,
0b00000000,
0b00000000,
0b00000000,
0b01111100},
{               //64       -          A
0b00111000,
0b01000100,
0b01000100,
0b01000100,
0b01111100,
0b01000100,
0b01000100,
0b01000100},
```

```
    {               //65       -          Б
    0b01111100,
    0b01000000,
    0b01000000,
    0b01111000,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01111000},
    {               //66       -          В
    0b01111000,
    0b01000100,
    0b01000100,
    0b01111000,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01111000},
    {               //67       -          Г
    0b01111100,
    0b01000100,
    0b01000000,
    0b01000000,
    0b01000000,
    0b01000000,
    0b01000000,
    0b01000000},
    {               //68       -          Д
    0b00111000,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01111100,
    0b10000010},
    {               //69       -          Е
```

```
    0b01111100,
    0b01000000,
    0b01000000,
    0b01111000,
    0b01000000,
    0b01000000,
    0b01000000,
    0b01111100},
    {               //70       -          Ж
    0b01010100,
    0b01010100,
    0b01010100,
    0b00111000,
    0b00111000,
    0b01010100,
    0b01010100,
    0b01010100},
    {               //71       -          З
    0b00111000,
    0b01000100,
    0b00000100,
    0b00111000,
    0b00000100,
    0b00000100,
    0b01000100,
    0b00111000},
    {               //72       -          И
    0b01000100,
    0b01000100,
    0b01000100,
    0b01001100,
    0b01010100,
    0b01100100,
    0b01000100,
    0b01000100},
    {               //73       -          Й
    0b00010000,
```

```
    0b01000100,
    0b01000100,
    0b01001100,
    0b01010100,
    0b01100100,
    0b01000100,
    0b01000100},
    {               //74       -          К
    0b01000100,
    0b01001000,
    0b01010000,
    0b01100000,
    0b01010000,
    0b01001000,
    0b01000100,
    0b01000100},
    {               //75       -          Л
    0b00111100,
    0b00100100,
    0b00100100,
    0b00100100,
    0b00100100,
    0b00100100,
    0b00100100,
    0b01000100},
    {               //76       -          М
    0b01000100,
    0b01101100,
    0b01010100,
    0b01010100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100},
    {               //77       -          Н
    0b01000100,
    0b01000100,
```

```
    0b01000100,
    0b01111100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100},
    {               //78       -          O
    0b00111000,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b00111000},
    {               //79       -          П
    0b01111100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100},
    {               //80       -          P
    0b01111000,
    0b01000100,
    0b01000100,
    0b01111000,
    0b01000000,
    0b01000000,
    0b01000000,
    0b01000000},
    {               //81       -          C
    0b00111000,
    0b01000100,
    0b01000000,
```

```
    0b01000000,
    0b01000000,
    0b01000000,
    0b01000100,
    0b00111000},
    {               //82       -          T
    0b01111100,
    0b01010100,
    0b00010000,
    0b00010000,
    0b00010000,
    0b00010000,
    0b00010000,
    0b00010000},
    {               //83       -          У
    0b01000100,
    0b01000100,
    0b00100100,
    0b00011000,
    0b00001000,
    0b00010000,
    0b00100000,
    0b01000000},
    {               //84       -          Ф
    0b00010000,
    0b00111000,
    0b01010100,
    0b01010100,
    0b00111000,
    0b00010000,
    0b00010000,
    0b00010000},
    {               //85       -          X
    0b01000100,
    0b01000100,
    0b00101000,
    0b00010000,
```

```
    0b00101000,
    0b01000100,
    0b01000100,
    0b01000100},
    {               //86       -          Ц
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01000100,
    0b01111100,
    0b00000010},
    {               //87       -          Ч
    0b01000100,
    0b01000100,
    0b01000100,
    0b00111100,
    0b00000100,
    0b00000100,
    0b00000100,
    0b00000100},
    {               //88       -          Ш
    0b01010100,
    0b01010100,
    0b01010100,
    0b01010100,
    0b01010100,
    0b01010100,
    0b01010100,
    0b01111100},
    {               //89       -          Щ
    0b01010100,
    0b01010100,
    0b01010100,
    0b01010100,
    0b01010100,
```

```
    0b01010100,
    0b01111100,
    0b00000010},
    {               //90       -          Ъ
    0b01100000,
    0b00100000,
    0b00100000,
    0b00100000,
    0b00111000,
    0b00100100,
    0b00100100,
    0b00111000},
    {               //91       -          Ы
    0b01000010,
    0b01000010,
    0b01000010,
    0b01000010,
    0b01110010,
    0b01001010,
    0b01001010,
    0b01110010},
    {               //92       -          Ь
    0b00100000,
    0b00100000,
    0b00100000,
    0b00100000,
    0b00111000,
    0b00100100,
    0b00100100,
    0b00111000},
    {               //93       -          Э
    0b00111000,
    0b01000100,
    0b00000100,
    0b00011100,
    0b00000100,
    0b00000100,
```

```
    0b01000100,
    0b00111000},
    {               //94       -          Ю
    0b01001000,
    0b01010100,
    0b01010100,
    0b01110100,
    0b01010100,
    0b01010100,
    0b01010100,
    0b01001000},
    {               //95       -          Я
    0b00011100,
    0b00100100,
    0b00100100,
    0b00011100,
    0b00001100,
    0b00010100,
    0b00100100,
    0b00100100}
}   ;


/* реалізація функції, що повертає індекс зображення в масиві
FontArray того символу, ASCII-код якого переданий до SymbolCode */
char GetSymbolIndex(char SymbolCode)
{
    char i = 0;
    if ((SymbolCode >= 32) && (SymbolCode <= 95))
    {
        i = SymbolCode - 32;
    }
    else
    {
        if ((SymbolCode >= 192) && (SymbolCode <= 223))
        {
            i = SymbolCode - 128;
```

```
            }
      }

      return i;
 }
```

Представлений модуль, підключається до основного модулю проекту, який містить основний код програмного забезпечення, та представлений нижче.

```
/*
 * RunSTR.c
 *
 * Created: 11.04.2020 23:02:20
 *  Author: Администратор
 */

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/delay.h>
#include "Font.c"
#define high(x) ((x)>>8)
#define low(x) ((x)& 0xFF)

#define TEXTBUFSIZE        100        //розмір текстового буфера 100
#define FCLK    1000000               //тактова частота MCU

#define BAUDRATE      4800 //швидкість передачі даних від порту PC
#define UARTSPEED     12    //(FCLK / (16 * BAUDRATE) - 1) = 12



char VideoMem[3][8];       //відеопам'ть
char TextBuf[TEXTBUFSIZE]; //текстовий буфер (1 байт - 1 символ)
char TextPointer = 0;      /*  покажчик  на  "вершину"  текстового
буфера.  Число,  що  вказує  на  індекс  першого  доступного  елемента
масиву. */
char S = 0;           //кількість зсувів
```

```
int StrSpeed = 1800;

//---------------------------------------------------------------------
void ShiftTextBuf(void)    /* функція, що зсуває вміст текстового
буфера на одну позицію вліво, останній елемент дорівнюється до нуля.
{
      for (char l=0; l < (TEXTBUFSIZE - 1); l++)
      {
            TextBuf[l] = TextBuf[l + 1];
      }
      TextBuf[TEXTBUFSIZE - 1] = 0;
}

//---------------------------------------------------------------------
void UploadSymbol(char ascii_code)         /* функція, що завантажує
зображення символу з кодом ASCII в третій розряд відеопам'яті
{
      char k = 0;
      k = GetSymbolIndex(ascii_code); /* отримуємо індекс зображення
символу */
      VideoMem[2][0] = FontArray[k][0];
      VideoMem[2][1] = FontArray[k][1];
      VideoMem[2][2] = FontArray[k][2];
      VideoMem[2][3] = FontArray[k][3];
      VideoMem[2][4] = FontArray[k][4];
      VideoMem[2][5] = FontArray[k][5];
      VideoMem[2][6] = FontArray[k][6];
      VideoMem[2][7] = FontArray[k][7];
}

//---------------------------------------------------------------------
void ShowData(char NumStr, char Data1, char Data2)    /* функція, що
відображає два байти даних в рядку з номером NumStr (від 0 до 7) */
{
      char tmp;

      SPDR = Data2;                    //відправляємо другий байт
```

```
      while (!(SPSR & 0b10000000))    //очікуємо завершення передачі
      {}
      tmp = SPDR;

      SPDR = Data1;                   //відправляємо перший байт
      while (!(SPSR & 0b10000000))    //очікуємо завершення передачі
      {}
      tmp = SPDR;

      /*для виключення паразитного засвічування сусідніх строк виключаєм горіння індикатора перед зміною номера строки */
      PORTB = PORTB | 0b00000001;     /*       виводи       регістрів переключаються до Z-стану */

      //активуємо потрібну строку
      PORTC = NumStr;
      _delay_ms(4);
      /* після завантаження в регістри, примушуємо їх вивести завантажені данні */
      PORTB = PORTB | 0b00000010;
      PORTB = PORTB & 0b11111101;

      //після переключень знову даємо команду на засвічення
      PORTB = PORTB & 0b11111110;     /*    на    виводах    регістрів з'являються завантажені дані */
      _delay_ms(2);
}

//-------------------------------------------------------------------
ISR (TIMER1_COMPA_vect)    //переривання таймеру при співпадінні. Виконується кожні 0,5 сек. */
{
      //зсув відеопам'яті
      for (char i = 0; i < 8; i++)
      {
            /* зсув і-ї строки відеопам'яті, що сскладається з трьох 8-бітних розрядів */
```

```
            VideoMem[0][i] = VideoMem[0][i] << 1;
            VideoMem[0][i] = VideoMem[0][i] | (VideoMem[1][i] >> 7);

            VideoMem[1][i] = VideoMem[1][i] << 1;
            VideoMem[1][i] = VideoMem[1][i] | (VideoMem[2][i] >> 7);
            VideoMem[2][i] = VideoMem[2][i] << 1;
      }
      S++;                  //збільшуємо лічильник зсувів
      if (S > 7)            /* через кожні 8 зсувів завантажуємо нову
букву в 3 розряд відеопам'яті */
      {
            S = 0;
            /* аналіз текстового буфера – чи є елементи для
відображення? */
            if (TextBuf[0] != 0)
            {
                  UploadSymbol(TextBuf[0]);  /*            завантажуємо
зображення символу, що міститься в першому елементі буфера, до
третього разряду відеопам'яті */
                  ShiftTextBuf(); //зсув текстового буфера вліво
                  TextPointer--;
            }
      }
}

//---------------------------------------------------------------------
ISR (USART_RXC_vect)       /* переривання, що виникає при отриманні
байту від програми управління відображенням
{
      unsigned char Data;
      Data = UDR;     //зчитування отриманого байту до змінної Data

      //PORTD = Data;
      //if (Data == '1') {PORTD = ~PORTD;}

      if (TextPointer < (TEXTBUFSIZE - 1))       /*  перевіряємо,  чи
не повний текстовий буфер */
```

```
      {
            TextBuf[TextPointer] = Data;          /*    якщо    ні    -
записуємо отриманий байт до кінця текстового буфера
            TextPointer++;
      }
}

//---------------------------------------------------------------------
ISR (INT0_vect)
{
      StrSpeed = StrSpeed - 200;
      OCR1AH = high(StrSpeed);
      OCR1AL = low(StrSpeed);
}

//---------------------------------------------------------------------
ISR (INT1_vect)
{
      StrSpeed = StrSpeed + 200;
      OCR1AH = high(StrSpeed);
      OCR1AL = low(StrSpeed);
}

//---------------------------------------------------------------------
//------------------------ОСНОВНА ПРОГРАМА--------------------------

int main(void)
{
      char n = 0;
      //==========конфігурація мікропонтролера============
      DDRB = 0xFF;          //весь порт B буде працювати на вихід
      DDRC = 0xFF;          //весь порт C буде працювати на вихід
      //вимикаємо SPI
      SPCR = 0b01110000;
      /* настроюємо і вмикаємо таймер, що здійснює зсув відеопам'яті
      TCCR1B = 0b00001011;  /* настроюємо K=64 (частоту імпульсів) і
на скидання по співпадінню */
```

```
      OCR1AH = high(StrSpeed);        /* настроюємо на рахування до
розрахованого числа A=7812 имп. */
      OCR1AL = low(StrSpeed);
      TIMSK = 0b00010000;        /*    включаємо    переривання    при
співпадіння з числом A (7812) */

      //настройка UART
      UBRRH = high(UARTSPEED);
      UBRRL = low(UARTSPEED);
      UCSRB = 0b10010000;

      //настройка зовнішніх переривань INT0 і INT1
      GICR = 0b11000000;

      sei();     //глобальний дозвіл переривань

      //початкові умови
      //TextBuf[0]  =  'W';  TextBuf[1]  =  'E';  TextBuf[2]  =  'L';
TextBuf[3] = 'C';
      //TextBuf[4]  =  'O';  TextBuf[5]  =  'M';  TextBuf[6]  =  'E';
TextBuf[7] = ' '; TextBuf[8] = 'T'; TextBuf[9] = 'O';
      //TextBuf[10]  =  '  ';  TextBuf[11]  =  'H';  TextBuf[12]  =  'E';
TextBuf[13] = 'L'; TextBuf[14] = 'L';

      TextPointer = 1;

      UploadSymbol(TextBuf[0]);
      ShiftTextBuf();            //зсув текстового буфера вліво
      TextPointer--;

    while(1)              /* потік, що відповідає за виведення на
екран вмісту відеопам'яті VideoMem (двох символів з трьох)
    {
          ShowData(n, VideoMem[0][n], VideoMem[1][n]);
          n = n + 1;
          if (n > 7)
          {
```

```
                sei();                  //вертикальна синхронізація
                _delay_ms(1);
                n = 0;
                cli();
            }
    }
}
```

# 5 Моделювання комп’ютерної системи управління рухомим рядком

Моделювання комп’ютерної системи здійснювалося за допомогою середовища імітаційного моделювання Proteus. З цією метою, була розроблена модель комп’ютерної системи відображення рухомого рядка, яка представлена на рисунку 4.7.

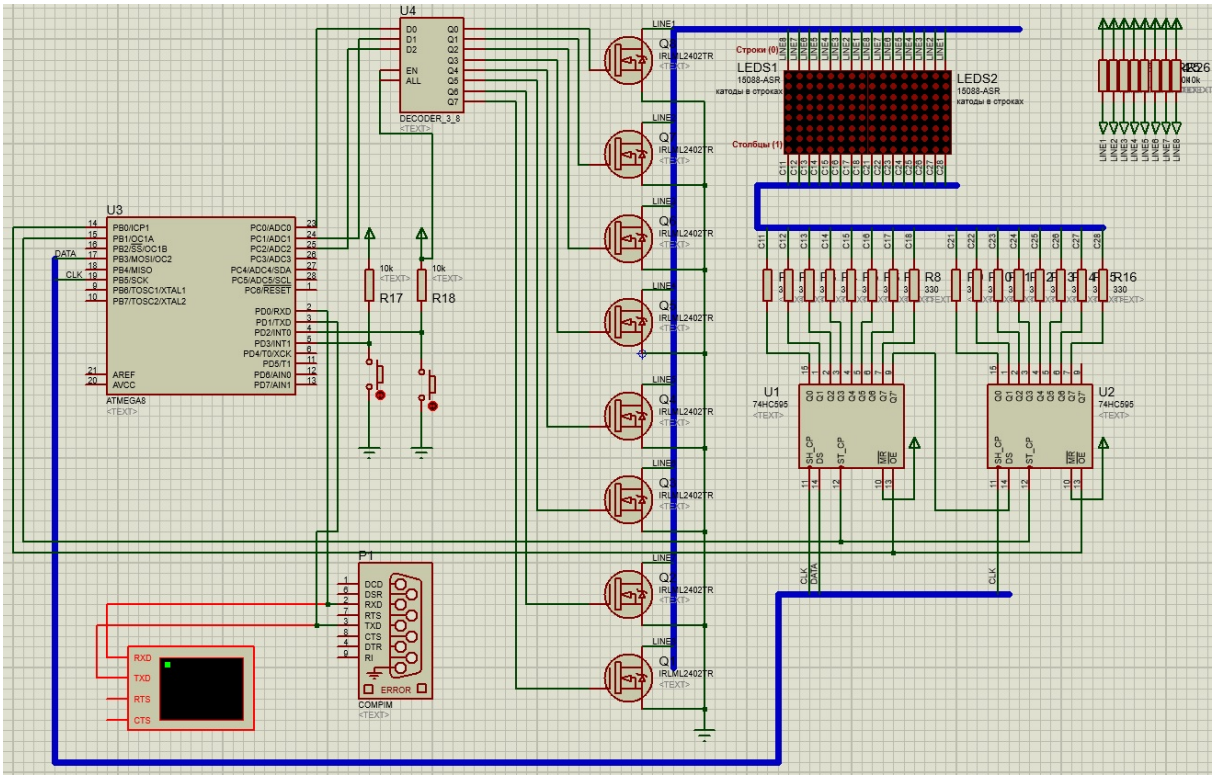


Рисунок 4.7 – Імітаційна модель комп’ютерної системи відображення рухомого рядка

В представленій моделі, використовується два матричних світлодіодних елемента для відображення рухомого рядка. Кількість таких елементів може збільшуватися, що буде вимагати використання додаткових секцій матричних світлодіодних індикаторів і відповідної кількості зсувних регістрів (в моделі для двох матричних індикаторів використовується два зсувних регістри U1 і U1. Додаткові лінії

мікроконтролера в цьому випадку не використовуються, так як для передачі даних використовується інтерфейс SPI.

Кнопки, підключені до входів зовнішніх переривань INT0 і INT1 дозволяють змінювати інтервал рахування таймеру та зменшувати і збільшувати частоту зсуву відеопам'яті (тобто змінювати швидкість зсуву рухомого рядка).

Для контроля даних, що передаються до системи з програмного забезпечення управління використовується Virtual Terminal.

Комунікація програмного забезпечення управління даними та комп'ютерної системи відображення здійснюється створенням на одному комп'ютері віртуальної пари з'єднаних COM портів. Для віртуалізації поєднаної пари COM портів використовується Virual Serial Ports Emulator. COM1 – використовується програмою передачі текстового рядка, а COM2 – комп'ютерною системою відображення рухомого рядка. Для зв'язку COM2 з моделлю використовується компонент COMPIM, який настроюється на узгоджену передачу на швидкості 4800 бод. На цю швидкість також настроюється програмне забезпечення передачі текстового рядка, мікроконтролер та Virtual Terminal.

Скріншот, що ілюструє коректну роботу усієї системи з елементами віртуалізації представлений скріншотом на рисунку 4.8.

На скріншоті представлений момент відображення частини строки, яка була переслана з додатка Windows RunSTR «!*ПРОВЕРКА РОБОТИ БЕГУЩЕЙ СТРОКИ*!», саме відображення фрагмента «ПРО» (кінець букви П, повністю буква Р і початок букви О).

Нажаль, використання елементів віртуалізації не дозволяють повністю здійснити моделювання в реальному часі, так як модель працює в якості додатка Windows. При цьому, нарощування кількості сегментів дисплея призводить до значного переускладнення моделі та появи значних затримок у віртуалізації відображення рухомого рядка. На реальному

апаратного забезпечення цього не буте спостерігатися. Фактично, використання двох сегментів вже призводить до затримок відображення та зриву вертикальної синхронізації. Таким чином, на моделі, повністю оцінити якість здійснення вертикальної синхронізації неможливо.

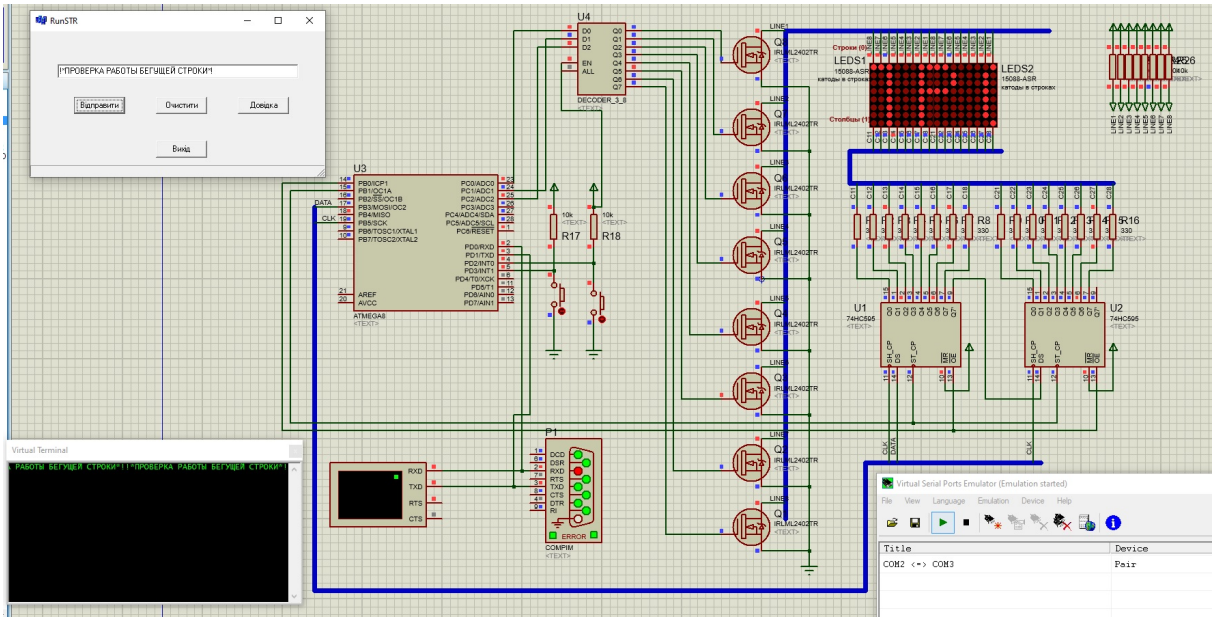


Рисунок 4.8 – Ілюстрація роботи комп’ютерної системи управління рухомим рядком з використанням елементів віртуалізації

# ВИСНОВКИ

В ході дипломної роботи була розроблена комп’ютерна система для управління рухомим рядком на світлодіодній індикації. Метою було розробка наступних елементів:

- апаратного забезпечення спеціалізованої комп’ютерної системи відображення рухомого рідка;
- програмного забезпечення спеціалізованої комп’ютерної системи відображення рухомого рідка;
- прикладного програмного забезпечення під платформу Windows для управління рухомим рядком та відправки даних для відображення.

Розроблена система дозволяє відправляти та відображати рухомим рядком текстові повідомлення довжиною до 100 символів, що обмежується в програмному забезпеченні мікроконтролера пристрою. Довжина рядка може бути збільшена, але з урахуванням наявної вільної оперативної пам’яті мікроконтролера. Обсяг для зберігання одного символу в оперативній пам’яті складає 1 байт. Обсяг зберігання одного образа символу у відеопам’яті складає 8 байт.

Слід зазначити, що для зменшення обсягу даних зберігання зображень символів було розроблено образи для прописні символи латиниці та кирилиці, цифр та деяких інших символів у кількості 95 штук. Тобто обсяг зберігання образів символів складає 95 х 8байт = .760 байт. За потребою до цього переліку можуть бути внесені будь які символи, але загальна їх кількість не повинна перевищувати 256, так як для кодування використовується один байт. Також, для створених символів було використано стандартне кодування ASCII.

# ПЕРЕЛІК ЛІТЕРАТУРИ

1. RS-485 для чайников. Описание RS-485 на русском. — [Электронный ресурс]: http://www.mayak-bit.narod.ru/rs485.html#gl1. — (Название с экрана).

2. Программирование COM порта ПК. — [Электронный ресурс]: http://www.softelectro.ru/rs232prog.html. — (Название с экрана).

3. А.Я. Архангельский «Программирование в C++ Builder 6». — Архангельский А.Я. — Бином. — 2002. — 1152 с.

4. Borland C++ Builder 6. Руководство разработчика. — Джаррод Холингворт, Боб Сворт, Марк Кэшмэн, Поль Густавсон. — Вильямс. — 2004.