МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

АВТОМАТИЗОВАНЕ ПРОЕКТУВАННЯ КОМП'ЮТЕРНИХ СИСТЕМ

Лабораторна робота № 2

Завдання 2: "Проста комунікаційна

схемаSW (клієнт) ↔ UART ↔ HW

(сервер)"

Виконав: ст. гр. КІ-404
Давида В.Р.

Прийняв:
Федак П.Р.

Львів 2024

Тема: SW <> HW (FEF).

Теоретичні відомості

UART (Universal Asynchronous Receiver/Transmitter) представляє собою стандартний протокол для обміну даними між пристроями через послідовний інтерфейс. Цей інтерфейс дозволяє пристроям передавати та приймати інформацію біт за бітом і знаходить широке застосування в мікроконтролерах, мікропроцесорах, сенсорах, модемах та інших пристроях.

Основні характеристики UART включають асинхронний режим, де дані передаються через два незалежні сигнали - TX (передача) та RX (прийом), структуру кадру, яка включає стартовий біт, біти даних, біти парності (опціонально) та стоповий біт. Ця структура дозволяє правильно ідентифікувати початок та кінець кожного байту.

Швидкість передачі (Baud Rate) грає ключову роль у визначенні того, скільки бітів передається за одну секунду, і ця швидкість повинна бути налаштована на обох пристроях для успішного обміну даними.

Парність (Parity) є опціональною функцією, яка дозволяє визначити четність чи непарність бітів даних для виявлення помилок передачі. Керівництво лінією (Flow Control) може використовуватися для управління потоком даних, особливо при великій швидкості передачі або різних швидкостях пристроїв.

Також іноді використовується ізоляція гальванічна для електричної ізоляції між пристроями та запобігання електричним помилкам.

Дистанція передачі за допомогою UART обмежена, хоча цей інтерфейс дозволяє передавати дані на значні відстані. Зазвичай це кілька метрів без спеціальних заходів.

Загалом, UART є простим та надійним інтерфейсом для обміну даними, і його широко використовують у вбудованих системах та промислових застосуваннях. Його важливі характеристики роблять його ефективним засобом для сполучення різних пристроїв у великому спектрі застосувань.

Порядок виконання лабораторної роботи:

№1 Створити просту схему зв'язку SW(клієнт) ↔ UART ↔ HW(сервер).

№2 Клієнт повинен відправити повідомлення на сервер. Сервер модифікувати повідомлення та надіслати його назад на адресу клієнта.

№3 Створити YML-файл з наступними можливостями:

a. побудова всіх двійкових файлів (створення скриптів в папці ci/ принеобхідності);
b. проводити тести;
c. створювати артефакти з двійковими файлами та звітами про випробування;

Хід виконання роботи

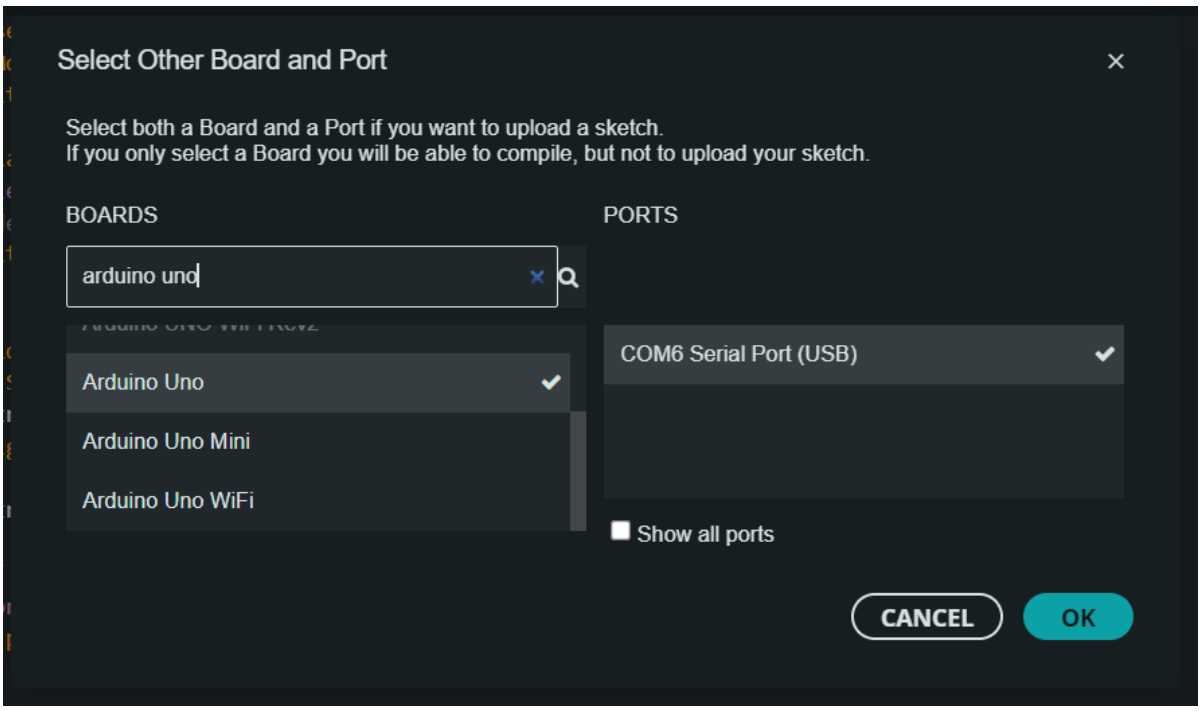

Рис. 1 Створення пов'язаних портів

Принцип роботи:

Результати відправлення повідомлення між сервером та клієнтом :

Код програми для серверної частини.

```
void setup() {
  pinMode(13, OUTPUT); //Initialization of foam for light bulb
  digitalWrite(13, HIGH);

  Serial.begin(9600);  //Configuring serial communication
  while (!Serial);     //Wait until the port becomes available (only for Leonardo
and similar boards)
  // delay(2000);
  digitalWrite(13, LOW);
}

void loop() {
  if (Serial.available() > 0) {
    String receivedMessage = Serial.readString();  //Read data from the serial port
    digitalWrite(13, HIGH);

    String parsedMessage = parseJSON(receivedMessage);

    //Convert all letters to uppercase
    for (int i = 0; i < parsedMessage.length(); i++) {
      parsedMessage[i] = toupper(parsedMessage[i]);
    }

    String jsonToSend = createJSON(parsedMessage);

    digitalWrite(13, LOW);
    //Send the modified message back
    Serial.print(jsonToSend);
  }
}

//JSON creation function
String createJSON(String message) {
  String json = "{\"message\":\"";
  json += message;
  json += "\"}";
  return json;
}

//JSON parsing function
String parseJSON(String json) {
  int start = json.indexOf("\"message\":\"") + 11;
  int end = json.indexOf("\"", start);

  if (start != -1 && end != -1) {
    return json.substring(start, end);
  }

  return ""; //If JSON is invalid
}
```

Код програми для серверної частини.

```
#include <windows.h>
#include <iostream>
#include <string>



using namespace std;


//JSON creation function
string createJSON(const string& message) {
    return "{\"message\":\"" + message + "\"}";
}

//JSON parsing function
string parseJSON(const string& json) {
    size_t start = json.find("\"message\":\"") + 11;
    size_t end = json.find("\"", start); //Find the end of the value

    if (start != string::npos && end != string::npos) {
        return json.substr(start, end - start);
    }

    return ""; //If JSON is invalid
}




int main() {

    /*Request COM port number from user*/
    int portNumber;
    cout << "Enter COM port number (e.g., 6 for COM6): ";
    cin >> portNumber;

    //Generating a port name
    string portName_s = "COM" + to_string(portNumber);
    const char* portName = portName_s.c_str();


    /*const char* portName = "COM6";*/

    //Opening COM port
    HANDLE hSerial = CreateFileA(portName, // portName.c_str()
        GENERIC_READ | GENERIC_WRITE,
        0,
        NULL,
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        NULL);

    if (hSerial == INVALID_HANDLE_VALUE) {
        cerr << "Error opening port" << endl;
        system("pause");
        return 1;
    }

    //Configure port parameters
    DCB dcbSerialParams = { 0 };
    dcbSerialParams.DCBlength = sizeof(dcbSerialParams);

    if (!GetCommState(hSerial, &dcbSerialParams)) {
        cerr << "Error getting port state" << endl;
        system("pause");
```

```
        return 1;
    }

    dcbSerialParams.BaudRate = CBR_9600;
    dcbSerialParams.ByteSize = 8;
    dcbSerialParams.StopBits = ONESTOPBIT;
    dcbSerialParams.Parity = NOPARITY;

    if (!SetCommState(hSerial, &dcbSerialParams)) {
        cerr << "Error setting port parameters" << endl;
        system("pause");
        return 1;
    }

    //Setting timeouts
    COMMTIMEOUTS timeouts = { 0 };
    timeouts.ReadIntervalTimeout = 50;
    timeouts.ReadTotalTimeoutConstant = 50;
    timeouts.ReadTotalTimeoutMultiplier = 10;

    if (!SetCommTimeouts(hSerial, &timeouts)) {
        cerr << "Error setting timeouts!" << endl;
        system("pause");
        return 1;
    }

    Sleep(2000);

    //Loop for endlessly sending messages
    while (true) {
        string message = "hello from pc";  //Message to send
        string messageXML = createJSON(message);
        DWORD bytesWritten;



        //Sending a message to Arduino
        if (!WriteFile(hSerial, messageXML.c_str(), messageXML.size(), &bytesWritten,
NULL)) {
            cerr << "Error writing to port" << endl;
            break;
        }
        cout << "Sent: " << message << " (" << messageXML << ")" << endl;

        //Reading the response from Arduino
        char buffer[128];
        DWORD bytesRead;

        if (ReadFile(hSerial, buffer, sizeof(buffer) - 1, &bytesRead, NULL)) {
            buffer[bytesRead] = '\0';
            if (bytesRead > 0) {
                cout << "Received from Arduino: " << parseJSON(buffer) << " (" <<
buffer << ")" << endl << endl;
            }
        }
        else {
            cerr << "Error reading from port" << endl;
            break;
        }

        //Delay before next request
        Sleep(1000);
    }

    //Closing COM port
    CloseHandle(hSerial);
    return 0;
}
```

Створений build.yaml для CI/CD :

```yaml
# This is a basic workflow to help you get started with Actions

name: C++ Client Build with MSBuild

# Controls when the workflow will run
on:
  # Triggers the workflow on push or pull request events but only for the "develop" branch
  push:
    branches: [ "develop" ]
  pull_request:
    branches: [ "develop" ]

  # Allows you to run this workflow manually from the Actions tab
  workflow_dispatch:

# A workflow run is made up of one or more jobs that can run sequentially or in parallel
jobs:
  # This workflow contains a single job called "build"
  build:
    # The type of runner that the job will run on
    runs-on: windows-latest

    # Steps represent a sequence of tasks that will be executed as part of the job
    steps:
      # Checks-out your repository under $GITHUB_WORKSPACE, so your job can access it
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Setup MSBuild
        uses: microsoft/setup-msbuild@v2

      - name: Build UART_Client_Lab_2 solution
        run: msbuild /p:Configuration=Release /p:Platform=x64 Lab2/win/UART_Client_Lab_2/UART_Client_Lab_2.sln

      - name: Upload build artifacts
        uses: actions/upload-artifact@v4
        with:
          name: build-output
```

Рис.2

В результаті завантажив роботу на GitHub :

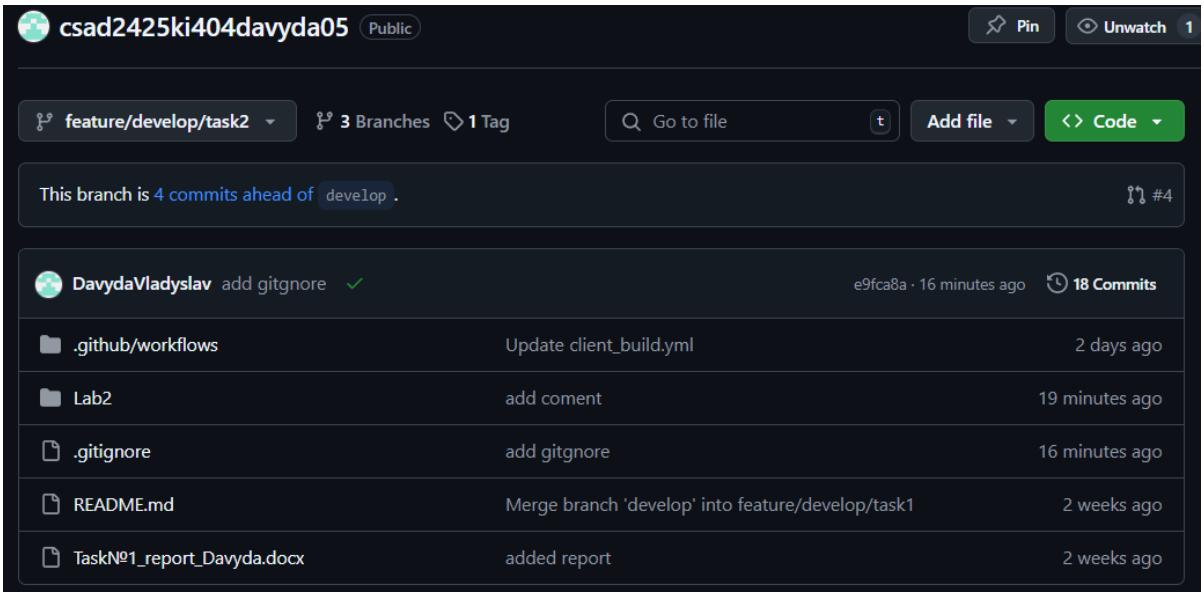

Рис.3

Висновок:

У процесі виконання цієї лабораторної роботи я ознайомився з основними принципами роботи комунікаційних інтерфейсів, зокрема UART (Universal Asynchronous Receiver-Transmitter). Я вивчив, як налаштовувати зв'язок між програмним забезпеченням та апаратним забезпеченням, використовуючи віртуальні порти для емуляції UART.

Під час роботи я реалізував просту архітектуру клієнт-сервер, у якій клієнт надсилав повідомлення на сервер, а сервер модифікував це повідомлення і повертав його назад. Це дало мені практичні навички в розробці та налагодженні програм, які використовують послідовний зв'язок.

Крім того, я дізнався про важливість тестування та CI/CD (безперервна інтеграція та безперервне розгортання) у розробці програмного забезпечення. Вивчаючи YAML-файли для автоматизації збірки та тестування, я отримав уявлення про те, як спростити процес розгортання програм і підвищити ефективність роботи команди.