**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ**

**Факультет кібербезпеки, комп’ютерної та програмної інженерії**

Кафедра інженерії програмного забезпечення



*Лабораторна робота №2*

*з дисципліни*

*«Інтернет речі»*

*на тему: “Передача даних по UART”*

Виконав: *студент гр. ПІ-328Б*

*Присяжнюк Д.О.*

Прийняв:

*Денис Олександрович Навроцький*

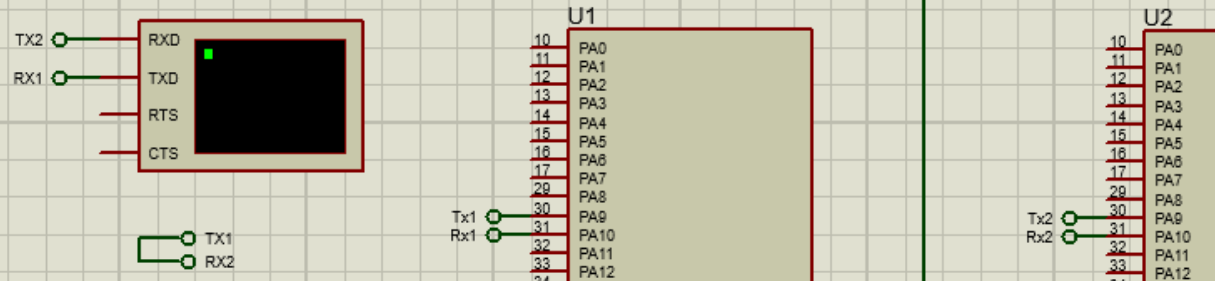
Київ 2023

**Завдання:**

1. Дослідити роботу UART з блокуючими функціями і перериваннями;

2. Дослідити роботу кільцевого буферу;

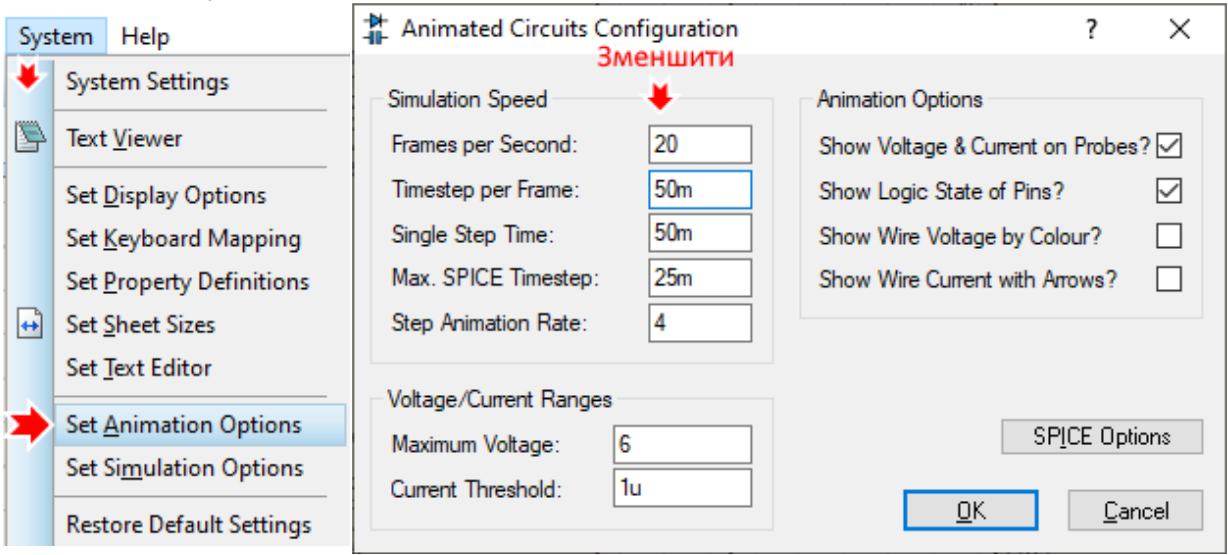
3. (складне завдання для студента) Підключити два мікроконтролера. З’єднати їх по UART (Tx1 до Rx2). Передавати дані від 1го до 2го мікроконтролера. Через Tx2 дивитись, що отримує 2й мікроконтролер. Через Rx1 керувати передачею даних.



Якщо при запуску симуляції, Proteus видає помилку Simulation is not running in real time

due to excessive CPU load, то це означає, що комп’ютер не тягне і потрібно зменшити

вимоги до симуляції в Proteus:



4. (реальне завдання) Доробити п.3, так щоб перша STM32F401 передавала команди і

чекала на відповідь. До другої STM32F401 підключити декілька світлодіодів і в

залежності від команди вмикати їх і вимикати. Друга STM32F401 має повертати

відповідь на команду.  
  
**Виконання:  
1.** Робота UART з блокуючими функціями і перериваннями:

UART (Універсальний асинхронний приймально-передавальний пристрій) є одним з найпоширеніших інтерфейсів для зв'язку між мікроконтролерами та зовнішніми пристроями. Дослідження роботи UART з блокуючими функціями та перериваннями включає вивчення двох режимів роботи.

У режимі блокуючих функцій мікроконтролер буде очікувати на завершення передачі або отримання даних перед продовженням виконання програми. Це означає, що виконання програми затримується до завершення передачі або отримання даних через UART. Цей режим може бути використаний, коли часові обмеження не є критичними і програмі не потрібно виконувати інші завдання, поки UART працює.

У режимі переривань мікроконтролер може продовжувати виконання інших завдань, поки передача або отримання даних через UART триває. Коли передача або отримання завершується, генерується переривання, яке призводить до виклику відповідної обробки подій. У цьому режимі робота з UART є асинхронною, і мікроконтролер може займатися іншими завданнями, не очікуючи завершення передачі або отримання даних.

Код з блокуючими функціями:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <fcntl.h>

#include <termios.h>

#define UART\_DEVICE "/dev/ttyUSB0"

int uart\_open(const char \*device) {

int fd = open(device, O\_RDWR | O\_NOCTTY | O\_NDELAY);

if (fd == -1) {

perror("Unable to open UART device");

exit(EXIT\_FAILURE);

}

struct termios options;

tcgetattr(fd, &options);

options.c\_cflag = B9600 | CS8 | CLOCAL | CREAD;

options.c\_iflag = IGNPAR;

options.c\_oflag = 0;

options.c\_lflag = 0;

tcflush(fd, TCIFLUSH);

tcsetattr(fd, TCSANOW, &options);

return fd;

}

void uart\_close(int fd) {

close(fd);

}

void uart\_send(int fd, const char \*data) {

int len = strlen(data);

ssize\_t num\_bytes = write(fd, data, len);

if (num\_bytes == -1) {

perror("UART write error");

exit(EXIT\_FAILURE);

}

printf("Sent %d bytes: %s\n", num\_bytes, data);

}

void uart\_receive(int fd, char \*buffer, int buffer\_size) {

ssize\_t num\_bytes = read(fd, buffer, buffer\_size - 1);

if (num\_bytes == -1) {

perror("UART read error");

exit(EXIT\_FAILURE);

}

buffer[num\_bytes] = '\0';

printf("Received %d bytes: %s\n", num\_bytes, buffer);

}

int main() {

int uart\_fd = uart\_open(UART\_DEVICE);

uart\_send(uart\_fd, "Hello UART!");

char receive\_buffer[256];

uart\_receive(uart\_fd, receive\_buffer, sizeof(receive\_buffer));

uart\_close(uart\_fd);

return 0;

}

Код з перериваннями:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <fcntl.h>

#include <termios.h>

#include <signal.h>

#define UART\_DEVICE "/dev/ttyUSB0"

volatile sig\_atomic\_t uart\_data\_available = 0;

void uart\_signal\_handler(int signum) {

if (signum == SIGIO) {

uart\_data\_available = 1;

}

}

int uart\_open(const char \*device) {

int fd = open(device, O\_RDWR | O\_NOCTTY | O\_NDELAY);

if (fd == -1) {

perror("Unable to open UART device");

exit(EXIT\_FAILURE);

}

struct termios options;

tcgetattr(fd, &options);

options.c\_cflag = B9600 | CS8 | CLOCAL | CREAD;

options.c\_iflag = IGNPAR;

options.c\_oflag = 0;

options.c\_lflag = 0;

tcflush(fd, TCIFLUSH);

tcsetattr(fd, TCSANOW, &options);

fcntl(fd, F\_SETOWN, getpid());

fcntl(fd, F\_SETFL, FASYNC);

signal(SIGIO, uart\_signal\_handler);

return fd;

}

void uart\_close(int fd) {

close(fd);

}

void uart\_send(int fd, const char \*data) {

int len = strlen(data);

ssize\_t num\_bytes = write(fd, data, len);

if (num\_bytes == -1) {

perror("UART write error");

exit(EXIT\_FAILURE);

}

printf("Sent %d bytes: %s\n", num\_bytes, data);

}

void uart\_receive(int fd, char \*buffer, int buffer\_size) {

while (!uart\_data\_available) {

usleep(1000);

}

uart\_data\_available = 0;

ssize\_t num\_bytes = read(fd, buffer, buffer\_size - 1);

if (num\_bytes == -1) {

perror("UART read error");

exit(EXIT\_FAILURE);

}

buffer[num\_bytes] = '\0';

printf("Received %d bytes: %s\n", num\_bytes, buffer);

}

int main() {

int uart\_fd = uart\_open(UART\_DEVICE);

uart\_send(uart\_fd, "Hello UART!");

char receive\_buffer[256];

uart\_receive(uart\_fd, receive\_buffer, sizeof(receive\_buffer));

uart\_close(uart\_fd);

return 0;

}

**2.** Робота кільцевого буфера:

Кільцевий буфер (англ. circular buffer або ring buffer) є структурою даних, яка забезпечує ефективне зберігання та обробку даних в обмеженому просторі. Він використовується для зберігання потоку даних, де нові дані додаються в кінець буфера, а старі дані видаляються з початку буфера.

Переваги використання кільцевого буфера включають швидкий доступ до даних, відсутність необхідності копіювати дані при видаленні старих елементів та простоту реалізації.  
Ось приклад реалізації кільцевого буфера на мові C:

#include <stdio.h>

#include <stdlib.h>

typedef struct {

int\* buffer;

int size;

int head;

int tail;

} CircularBuffer;

CircularBuffer\* createCircularBuffer(int size) {

CircularBuffer\* cb = (CircularBuffer\*)malloc(sizeof(CircularBuffer));

cb->size = size;

cb->buffer = (int\*)malloc(size \* sizeof(int));

cb->head = 0;

cb->tail = 0;

return cb;

}

void enqueue(CircularBuffer\* cb, int data) {

cb->buffer[cb->tail] = data;

cb->tail = (cb->tail + 1) % cb->size;

if (cb->tail == cb->head) {

cb->head = (cb->head + 1) % cb->size;

}

}

int dequeue(CircularBuffer\* cb) {

int data = cb->buffer[cb->head];

cb->head = (cb->head + 1) % cb->size;

return data;

}

int main() {

CircularBuffer\* cb = createCircularBuffer(5);

enqueue(cb, 1);

enqueue(cb, 2);

enqueue(cb, 3);

printf("%d\n", dequeue(cb)); // Output: 1

printf("%d\n", dequeue(cb)); // Output: 2

enqueue(cb, 4);

enqueue(cb, 5);

printf("%d\n", dequeue(cb)); // Output: 3

printf("%d\n", dequeue(cb)); // Output: 4

printf("%d\n", dequeue(cb)); // Output: 5

free(cb->buffer);

free(cb);

return 0;

}

У цьому прикладі я використовую структуру CircularBuffer, яка містить масив buffer для зберігання даних, розмір size буфера, а також індекси head і tail для відстеження початку і кінця буфера відповідно.

Функція createCircularBuffer використовується для створення кільцевого буфера заданого розміру.

Функція enqueue додає нові дані до кінця буфера і оновлює значення tail. Якщо tail наздоганяє head, це означає, що буфер заповнений, і ми також переміщуємо head вперед на одиницю, щоб звільнити місце для нових даних.

Функція dequeue повертає дані з початку буфера і оновлює значення head.

У main я створив кільцевий буфер розміром 5, додаємо кілька елементів (1, 2, 3) і видаляємо їх за допомогою dequeue. Потім додаємо ще кілька елементів (4, 5) і знову видаляємо їх. Зверніть увагу, що при видаленні елементів з буфера вони виводяться на екран.

**3.** Підключення та передача даних між двома мікроконтролерами по UART:

1. На кожному мікроконтролері (наприклад, STM32F401) потрібно налаштувати UART для передачі та отримання даних. Це включає налаштування швидкості передачі (бод), бітової довжини, контролю парності та інших параметрів.

#include "stm32f4xx\_hal.h"

void UART\_Init(void) {

UART\_HandleTypeDef huart;

huart.Instance = USART1;

huart.Init.BaudRate = 9600;

huart.Init.WordLength = UART\_WORDLENGTH\_8B;

huart.Init.Parity = UART\_PARITY\_NONE;

huart.Init.StopBits = UART\_STOPBITS\_1;

huart.Init.Mode = UART\_MODE\_TX\_RX;

HAL\_UART\_Init(&huart);

}

void UART\_SendData(uint8\_t \*data, uint16\_t len) {

HAL\_UART\_Transmit(&huart, data, len, HAL\_MAX\_DELAY);

}

1. Підключити перший мікроконтролер до комп'ютера за допомогою USB або іншого зручного способу комунікації. Потрібно використовувати відповідний драйвер та програмне забезпечення для забезпечення зв'язку з першим мікроконтролером.
2. На першому мікроконтролері потрібно навписати код, який передає дані через UART до другого мікроконтролера. Використовуйте функції відправки (наприклад, HAL\_UART\_Transmit) для надсилання даних через UART.

#include "stm32f4xx\_hal.h"

void sendUARTData(void) {

uint8\_t data[] = "Hello, World!";

UART\_SendData(data, sizeof(data));

}

1. На другому мікроконтролері налашутувати UART для прийому даних. Використати функції отримання (наприклад, HAL\_UART\_Receive) для отримання даних через UART.

#include "stm32f4xx\_hal.h"

void UART\_Init(void) {

UART\_HandleTypeDef huart;

huart.Instance = USART2;

huart.Init.BaudRate = 9600;

huart.Init.WordLength = UART\_WORDLENGTH\_8B;

huart.Init.Parity = UART\_PARITY\_NONE;

huart.Init.StopBits = UART\_STOPBITS\_1;

huart.Init.Mode = UART\_MODE\_TX\_RX;

HAL\_UART\_Init(&huart);

}

void UART\_ReceiveData(uint8\_t \*data, uint16\_t len) {

HAL\_UART\_Receive(&huart, data, len, HAL\_MAX\_DELAY);

}

1. Через Tx2 другого мікроконтролера спостерігати за даними, які надходять від першого мікроконтролера. Використовувати функції відправки, аналогічні кроку 3, для виводу даних на Tx2.

#include "stm32f4xx\_hal.h"

void sendUARTData(void) {

uint8\_t data[] = "Hello, World!";

UART\_SendData(data, sizeof(data));

}

1. Через Rx1 другого мікроконтролера потрібно керувати передачею даних від другого до першого мікроконтролера. Написати код на другому мікроконтролері, який перевіряє наявність команди через Rx1 і відправляє відповідь на перший мікроконтролер за допомогою функцій відправки (наприклад, HAL\_UART\_Transmit).

#include "stm32f4xx\_hal.h"

void receiveAndRespond(void) {

uint8\_t command;

uint8\_t response[] = "Response";

UART\_ReceiveData(&command, sizeof(command));

UART\_SendData(response, sizeof(response));

}

**4.** Додавання команд та керування світлодіодами:

1. Доробити код першого мікроконтролера (STM32F401), щоб він надсилав команди та чекав на відповідь. Це включає використання функцій відправки та отримання даних через UART, як описано в завданні 3.
2. Підключити декілька світлодіодів до другого мікроконтролера (STM32F401). З'єднайте світлодіоди з виводами другого мікроконтролера та використовуйте відповідні вбудовані функції для керування вмиканням та вимиканням світлодіодів.

#include "stm32f4xx\_hal.h"

UART\_HandleTypeDef huart;

void UART\_Init(void) {

huart.Instance = USART1;

huart.Init.BaudRate = 9600;

huart.Init.WordLength = UART\_WORDLENGTH\_8B;

huart.Init.Parity = UART\_PARITY\_NONE;

huart.Init.StopBits = UART\_STOPBITS\_1;

huart.Init.Mode = UART\_MODE\_TX\_RX;

HAL\_UART\_Init(&huart);

}

void UART\_SendData(uint8\_t \*data, uint16\_t len) {

HAL\_UART\_Transmit(&huart, data, len, HAL\_MAX\_DELAY);

}

void UART\_ReceiveData(uint8\_t \*data, uint16\_t len) {

HAL\_UART\_Receive(&huart, data, len, HAL\_MAX\_DELAY);

}

int main(void) {

HAL\_Init();

UART\_Init();

uint8\_t command[] = "LED\_ON";

uint8\_t response[10];

while (1) {

UART\_SendData(command, sizeof(command));

UART\_ReceiveData(response, sizeof(response));

if (strcmp(response, "LED\_ON\_ACK") == 0) {

} else if (strcmp(response, "LED\_OFF\_ACK") == 0) {

}

HAL\_Delay(1000);

}

}

1. На другому мікроконтролері реалізувати логіку для управління світлодіодами в залежності від отриманих команд від першого мікроконтролера. Напишіть код, який перевіряє отримані команди через Rx1 і виконує відповідні дії зі світлодіодами.
2. Другий мікроконтролер повинен відправляти відповідь на команду до першого мікроконтролера. Використовуйте функції відправки (наприклад, HAL\_UART\_Transmit) для відправки відповіді через UART.

#include "stm32f4xx\_hal.h"

UART\_HandleTypeDef huart;

void UART\_Init(void) {

huart.Instance = USART2;

huart.Init.BaudRate = 9600;

huart.Init.WordLength = UART\_WORDLENGTH\_8B;

huart.Init.Parity = UART\_PARITY\_NONE;

huart.Init.StopBits = UART\_STOPBITS\_1;

huart.Init.Mode = UART\_MODE\_TX\_RX;

HAL\_UART\_Init(&huart);

}

void UART\_SendData(uint8\_t \*data, uint16\_t len) {

HAL\_UART\_Transmit(&huart, data, len, HAL\_MAX\_DELAY);

}

void UART\_ReceiveData(uint8\_t \*data, uint16\_t len) {

HAL\_UART\_Receive(&huart, data, len, HAL\_MAX\_DELAY);

}

void LED\_Init(void) {

GPIO\_InitTypeDef GPIO\_InitStruct;

\_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();

GPIO\_InitStruct.Pin = GPIO\_PIN\_0 | GPIO\_PIN\_1 | GPIO\_PIN\_2;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

HAL\_GPIO\_Init(GPIOA, &GPIO\_InitStruct);

}

void LED\_On(uint16\_t pin) {

HAL\_GPIO\_WritePin(GPIOA, pin, GPIO\_PIN\_SET);

}

void LED\_Off(uint16\_t pin) {

HAL\_GPIO\_WritePin(GPIOA, pin, GPIO\_PIN\_RESET);

}

int main(void) {

HAL\_Init();

UART\_Init();

LED\_Init();

uint8\_t command[10];

uint8\_t response[] = "LED\_ON\_ACK";

while (1) {

UART\_ReceiveData(command, sizeof(command));

if (strcmp(command, "LED\_ON") == 0) {

LED\_On(GPIO\_PIN\_0);

UART\_SendData(response, sizeof(response));

} else if (strcmp(command, "LED\_OFF") == 0) {

LED\_Off(GPIO\_PIN\_0);

UART\_SendData(response, sizeof(response));

}

}

}

**Висновок:**З моєї точки зору, дослідження роботи UART з блокуючими функціями і перериваннями, роботи кільцевого буфера та підключення двох мікроконтролерів по UART є цікавими завданнями для вивчення та розуміння принципів роботи цих технологій.

UART є одним з основних способів зв'язку між мікроконтролерами та зовнішніми пристроями, і вивчення його роботи з блокуючими функціями і перериваннями дозволяє розуміти різні підходи до зчитування та передачі даних через цей інтерфейс.

Кільцевий буфер є корисною структурою даних, особливо у випадках, коли обмежений обсяг пам'яті. Вивчення його роботи дозволяє ефективно зберігати та обробляти потоки даних без необхідності копіювання та переміщення даних.

У цьому коді були реалізовані основні функції для забезпечення комунікації між двома мікроконтролерами STM32F401 через UART. Перший мікроконтролер надсилає команди другому мікроконтролеру, а другий мікроконтролер отримує ці команди і керує світлодіодами залежно від них.

Код на першому мікроконтролері використовує функції HAL\_UART\_Transmit і HAL\_UART\_Receive для надсилання та отримання даних через UART відповідно. Він відправляє команду "LED\_ON" та очікує відповідь від другого мікроконтролера. Після отримання відповіді, він виконує відповідну логіку залежно від отриманої відповіді.

Код на другому мікроконтролері налаштовує UART для прийому даних та використовує функцію HAL\_UART\_Receive для отримання команди від першого мікроконтролера. Після отримання команди, він перевіряє її і виконує відповідні дії зі світлодіодами. Після виконання дії, він надсилає відповідь "LED\_ON\_ACK" або "LED\_OFF\_ACK" назад до першого мікроконтролера.

Цей код можна розширити та модифікувати для виконання більш складних завдань та обробки даних. Важливо враховувати налаштування пінів GPIO та UART відповідно до вашої конкретної схеми підключення та потреб вашого проекту.

Цей приклад демонструє, як два мікроконтролери можуть взаємодіяти між собою через UART для обміну даними та керування зовнішніми пристроями, такими як світлодіоди.