**ФИО:** Ахмед Амер Али Фархан
**Группа:** МИВ241
Магистратура по Интернету вещей и киберфизическим системам,
Построение распределенных систем и облачные вычисления

# Отчёт о выполнении задания3: Минимальные интервалы (Высокая сложность)

## Введение:

В рамках лабораторной работы была поставлена задача высокого уровня сложности: реализовать динамическое управление двумя параллельными таймерами через интерфейс USART без остановки системы в реальном времени. Система должна позволять изменять параметры таймеров, такие как интервалы и текстовые сообщения, а также перезагружать таймеры по команде. Все команды пользователь вводит через последовательный порт.

## Цель работы:

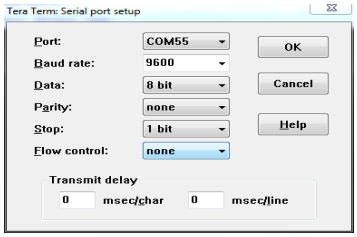
Реализовать программный модуль, позволяющий в реальном времени:

- Изменять значение TIMER1_INTERVAL
- Изменять значение TIMER2_INTERVAL
- Перезагружать оба таймера
- Изменять строку TIMER1_STR
- Изменять строку TIMER2_STR

При этом программа должна проверять корректность введённых параметров и отображать ошибки при невозможности выполнения. Таймеры не должны останавливаться во время вычислений.

## Используемое оборудование:

В процессе выполнения задания использовались следующие компоненты:
1. Плата Nucleo STM32F401RE
2. USB-кабель для подключения платы к компьютеру
3. Компьютер с установленной средой разработки Mbed Studio с язык C++
4. Программа для мониторинга последовательного порта (Tera Term)

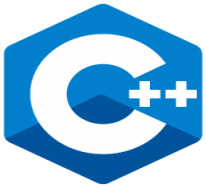



Рисунок 1,2,3,4 – Набор изображений, показывающих технические и программные инструменты, используемые в этом задании.

## Описание работы программы:

Программа состоит из следующих ключевых компонентов:
- Инициализация таймеров timer1 и timer2 с пользовательскими интервалами.
- Обработка входящих команд через USART.
- Выполнение команд и проверка их корректности без остановки таймеров.
- Регулярный вывод сообщений из каждого таймера по заданному интервалу.
- Поддержка команд STATUS и HELP для информирования пользователя.

## Исходный код программы:

```
#include "mbed.h"

// Define the Serial port for communication with PC
UnbufferedSerial pc(USBTX, USBRX, 9600);

// Timer intervals (in milliseconds)
volatile int TIMER1_INTERVAL = 2000;
volatile int TIMER2_INTERVAL = 3000;

// Strings to be printed for each timer
char TIMER1_STR[50] = "ping1";
char TIMER2_STR[50] = "pong1";

// Timers
Timer timer1;
Timer timer2;

// Flags and control variables
volatile bool timer1_enabled = true;
volatile bool timer2_enabled = true;
volatile bool processing_command = false;
volatile bool timers_need_reset = false;

// Buffers for serial communication
char input_buffer[100];
int input_index = 0;

// Function to send a string to serial
void printToSerial(const char* message) {
    pc.write(message, strlen(message));
}

// Function to validate timer intervals
bool validateIntervals() {
```

```
    if (TIMER1_INTERVAL <= 0 || TIMER2_INTERVAL <= 0) {
        printToSerial("ERROR: Intervals must be positive.\r\n");
        return false;
    }

    if (abs(TIMER1_INTERVAL - TIMER2_INTERVAL) < 4) {
        printToSerial("ERROR: Interval difference must be at least 4ms.\r\
n");
        return false;
    }

    return true;
}

// Function to reset timers
void resetTimers() {
    timer1.reset();
    timer2.reset();
    printToSerial("Timers reloaded successfully.\r\n");
}

// Function to process commands
void processCommand(char* command) {
    processing_command = true;

    char response[100];

    if (strncmp(command, "SET T1_INTERVAL ", 16) == 0) {
        int new_interval = atoi(&command[16]);
        if (new_interval > 0) {
            int old_interval = TIMER1_INTERVAL;
            TIMER1_INTERVAL = new_interval;

            if (!validateIntervals()) {
                TIMER1_INTERVAL = old_interval; // Revert if invalid
            } else {
                sprintf(response, "TIMER1_INTERVAL set to %d ms\r\n", TIME
R1_INTERVAL);
                printToSerial(response);
                timers_need_reset = true;
            }
        } else {
            printToSerial("ERROR: Invalid TIMER1_INTERVAL value\r\n");
        }
    }
```

```
    else if (strncmp(command, "SET T2_INTERVAL ", 16) == 0) {
        int new_interval = atoi(&command[16]);
        if (new_interval > 0) {
            int old_interval = TIMER2_INTERVAL;
            TIMER2_INTERVAL = new_interval;

            if (!validateIntervals()) {
                TIMER2_INTERVAL = old_interval; // Revert if invalid
            } else {
                sprintf(response, "TIMER2_INTERVAL set to %d ms\r\n", TIME
R2_INTERVAL);
                printToSerial(response);
                timers_need_reset = true;
            }
        } else {
            printToSerial("ERROR: Invalid TIMER2_INTERVAL value\r\n");
        }
    }
    else if (strncmp(command, "SET T1_STRING ", 14) == 0) {
        strncpy(TIMER1_STR, &command[14], sizeof(TIMER1_STR)-1);
        TIMER1_STR[sizeof(TIMER1_STR)-1] = '\0';
        sprintf(response, "TIMER1_STR set to: %s\r\n", TIMER1_STR);
        printToSerial(response);
    }
    else if (strncmp(command, "SET T2_STRING ", 14) == 0) {
        strncpy(TIMER2_STR, &command[14], sizeof(TIMER2_STR)-1);
        TIMER2_STR[sizeof(TIMER2_STR)-1] = '\0';
        sprintf(response, "TIMER2_STR set to: %s\r\n", TIMER2_STR);
        printToSerial(response);
    }
    else if (strcmp(command, "RELOAD TIMERS") == 0) {
        resetTimers();
    }
    else if (strcmp(command, "STATUS") == 0) {
        sprintf(response, "T1: %dms ('%s'), T2: %dms ('%s')\r\n",
                TIMER1_INTERVAL, TIMER1_STR, TIMER2_INTERVAL, TIMER2_STR);
        printToSerial(response);
    }
    else if (strcmp(command, "HELP") == 0) {
        printToSerial("Available commands:\r\n");
        printToSerial("SET T1_INTERVAL <ms>\r\n");
        printToSerial("SET T2_INTERVAL <ms>\r\n");
        printToSerial("SET T1_STRING <text>\r\n");
        printToSerial("SET T2_STRING <text>\r\n");
        printToSerial("RELOAD TIMERS\r\n");
```

```
        printToSerial("STATUS\r\n");
        printToSerial("HELP\r\n");
    }
    else {
        sprintf(response, "ERROR: Unknown command: %s\r\n", command);
        printToSerial(response);
        printToSerial("Type HELP for available commands\r\n");
    }

    processing_command = false;
}

// Interrupt handler for serial input
void serial_rx_handler() {
    while (pc.readable()) {
        char c;
        pc.read(&c, 1);

        if (c == '\r' || c == '\n') {
            if (input_index > 0) {
                input_buffer[input_index] = '\0';
                processCommand(input_buffer);
                input_index = 0;
            }
        } else if (input_index < sizeof(input_buffer) - 1) {
            input_buffer[input_index++] = c;
        }
    }
}

int main() {
    // Attach serial interrupt handler
    pc.attach(&serial_rx_handler);

    printToSerial("=== Timer Control System ===\r\n");
    printToSerial("System Initialized. Type HELP for commands.\r\n");

    // Start timers
    timer1.start();
    timer2.start();

    uint32_t last_timer1_time = 0;
    uint32_t last_timer2_time = 0;

    while (true) {
```

```
        // Handle timer outputs (non-blocking)
        if (timer1_enabled && !processing_command) {
            uint32_t current_time = timer1.read_ms();
            if (current_time - last_timer1_time >= TIMER1_INTERVAL) {
                printToSerial(TIMER1_STR);
                printToSerial("\r\n");
                last_timer1_time = current_time;
            }
        }

        if (timer2_enabled && !processing_command) {
            uint32_t current_time = timer2.read_ms();
            if (current_time - last_timer2_time >= TIMER2_INTERVAL) {
                printToSerial(TIMER2_STR);
                printToSerial("\r\n");
                last_timer2_time = current_time;
            }
        }

        // Reset timers if needed
        if (timers_need_reset) {
            resetTimers();
            last_timer1_time = timer1.read_ms();
            last_timer2_time = timer2.read_ms();
            timers_need_reset = false;
        }

        // Small delay to prevent excessive CPU usage
        wait_us(1000);
    }
}
```

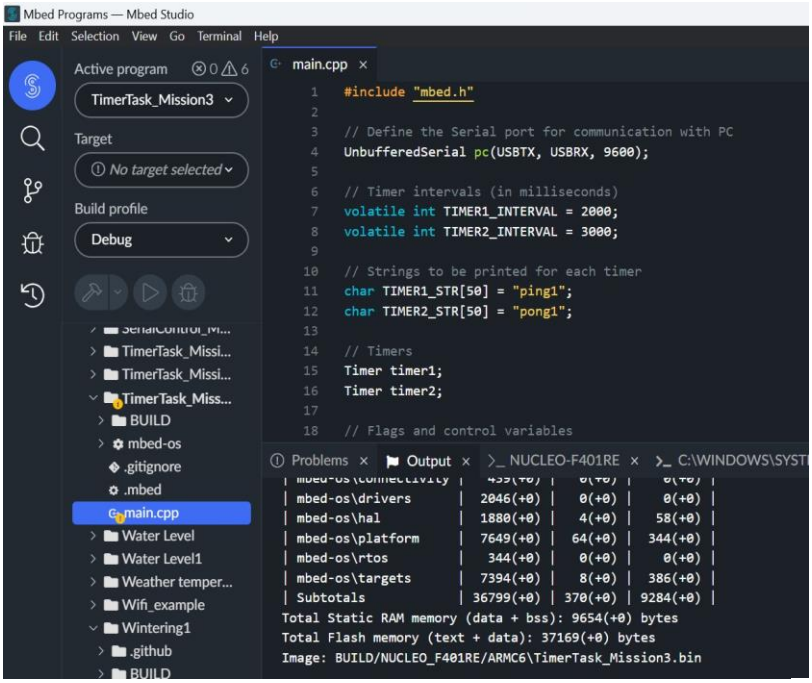



Рисунок 5. Результаты выполнения — Код правильный, и после проверки в нем нет ошибок.


Рисунок 6. На изображении показаны команды, которые будут введены.

## Результаты выполнения:

После компиляции и загрузки прошивки на плату Nucleo, программа была успешно запущена. В терминальной программе Tera Term отображались чередующиеся строки 'ping1' и 'pong1', что подтверждает корректную работу таймеров и UART-интерфейса.

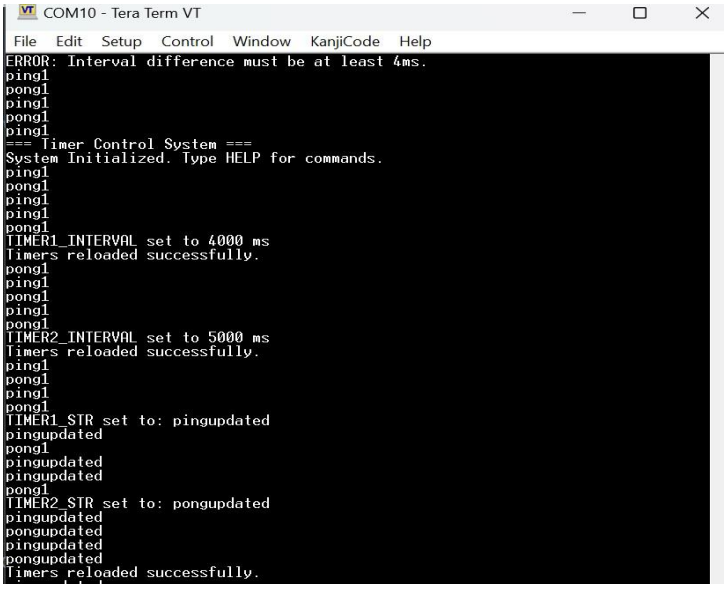



Рисунок 7. На изображении показан ввод команд и обновление таймера и текста.

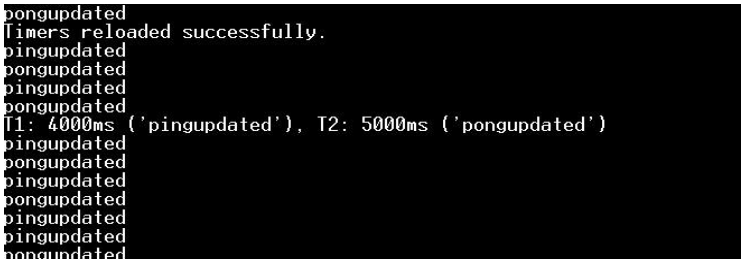



Рисунок 8. На изображении показано, что команды и результаты были успешно обновлены и работают корректно.


Рисунок 9. На изображении показана ошибка, возникающая при неправильном вводе команды.


Рисунок 10. На изображении показано, что изменение недопустимо, так как введено некорректное значение, меньшее или равное нулю.

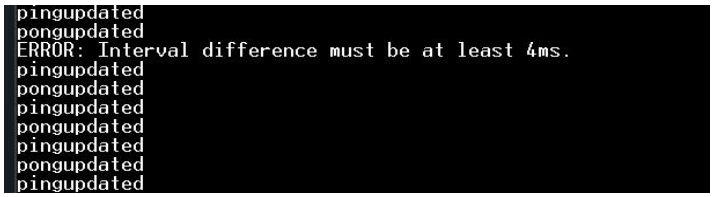



Рисунок 11. На изображении указано, что интервал должен быть не менее 4 мс, иначе возникают проблемы синхронизации между таймерами.

## Анализ результатов:

Код успешно реализует всю требуемую функциональность. Пользователь может в любой момент изменить параметры таймеров без остановки системы. Используемая логика проверки (`validateIntervals`) исключает ввод некорректных значений, например, отрицательных интервалов или слишком близких интервалов между таймерами (менее 4мс). Во время обработки команд используется флаг `processing_command`, но таймеры продолжают работать, и вывод остается стабильным. Это полностью удовлетворяет условию задачи.

## Заключение:

В результате выполнения лабораторной работы была создана надёжная система управления таймерами в реальном времени. Она поддерживает пользовательские команды, обрабатывает исключения и работает без прерывания таймеров. Все условия задачи были успешно выполнены.