Факультет программной инженерии и компьютерной техники

Лабораторная работа №1
«Последовательный интерфейс UART»
по дисциплине «Проектирование вычислительных систем»
Вариант – 1

Выполнили:
Студенты группы P3432
Чмурова М.В.
Комягин Д.А.

Преподаватель:
Пинкевич Василий Юрьевич

Санкт-Петербург
2025

Задание

Разработать и реализовать два варианта драйверов UART для стенда SDK-1.1M: с использованием и без использования прерываний. Драйверы, использующие прерывания, должны обеспечивать работу в «неблокирующем» режиме (возврат из функции происходит сразу же, без ожидания окончания приема/отправки), а также буферизацию данных для исключения случайной потери данных. В драйвере, не использующем прерывания, функция приема данных также должна быть «неблокирующей», то есть она не должна зависать до приема данных (которые могут никогда не поступить). При использовании режима «без прерываний» прерывания от соответствующего блока UART должны быть запрещены.

Написать с использованием разработанных драйверов программу, которая выполняет определенную вариантом задачу. Для всех вариантов должно быть реализовано два режима работы программы: с использованием и без использования прерываний. Каждый принимаемый стендом символ должен отсылаться обратно, чтобы он был выведен в консоли (так называемое «эхо»). Каждое новое сообщение от стенда должно выводиться с новой строки. Если вариант предусматривает работу с командами, то на каждую команду должен выводиться ответ, определенный в задании или «OK», если ответ не требуется. Если введена команда, которая не поддерживается, должно быть выведено сообщение об этом.

Скорость работы интерфейса UART должна соответствовать 57 600 бит/с

Доработать программу «светофор», добавив возможности отключения кнопки и задания величины тайм-аута (период, в течение которого горит красный).

Должны обрабатываться следующие команды, посылаемые через UART:

- ? – в ответ стенд должен прислать
 - состояние, которое отображается в данный момент на светодиодах: red, yellow, green, blinking green,
 - режим – mode 1 или mode 2 (см. 78 далее),
 - величину тайм-аута (сколько горит красный) – timeout …,
 - и задействованы ли прерывания – символ I (interrupt) или P (polling);
- set mode 1 или set mode 2 – установить режим работы светофора, когда обрабатываются или игнорируются нажатия кнопки;
- set timeout X – установить тайм-аут (X – длина периода в секундах);
- set interrupts on или set interrupts off – включить или выключить прерывания. Скорость обмена данными по UART – 57600 бит/с.

Описание структуры драйверов

В данном драйвере реализована поддержка двух режимов работы:

- **Режим прерываний** - неблокирующая работа с буферизацией данных
- **Режим опроса** - неблокирующая работа без буферизации

Структуры данных и буферизация:

```
static uint8_t rx_buf[UART_RX_BUF_SIZE];
static uint8_t tx_buf[UART_TX_BUF_SIZE];
static volatile uint16_t rx_head = 0, rx_tail = 0;
static volatile uint16_t tx_head = 0, tx_tail = 0;

static uint8_t rx_byte;
static uint8_t interrupts_enabled = 0;
```

Принцип работы кольцевого буфера:

- **head** указывает на позицию для записи следующего элемента
- **tail** указывает на позицию для чтения следующего элемента
- Буфер считается пустым, когда **head** == **tail**
- Буфер считается полным, когда (**head** + 1) % **size** == **tail**

Функции управления режимами работы:

```
uint8_t uart_get_interrupts(void) { return interrupts_enabled; }

void uart_set_interrupts(uint8_t enabled)
{
    interrupts_enabled = enabled;
    if (enabled)
    {
        rx_head = rx_tail = 0;
        tx_head = tx_tail = 0;

          HAL_NVIC_EnableIRQ(USART6_IRQn);
        HAL_UART_Receive_IT(&huart6, &rx_byte, 1);
    }
    else
    {
        HAL_UART_AbortReceive_IT(&huart6);
        HAL_UART_AbortTransmit_IT(&huart6);
        HAL_NVIC_DisableIRQ(USART6_IRQn);
    }
}
```

Данная функция выполняет переключение между режимами работы:

При включении прерываний:

- Сбрасывает указатели буферов
- Активирует прерывание
- Запускает асинхронный прием первого байта

При выключении прерываний:

- Останавливает все текущие операции приема и передачи
- Отключает прерывание

Функции передачи данных:

```
void uart_print_char(char ch)
{
    if (!interrupts_enabled)
    {
        HAL_UART_Transmit(&huart6, (uint8_t *)&ch, 1, UART_TIMEOUT);
        return;
    }
    uint16_t next = (tx_head + 1) % UART_TX_BUF_SIZE;
    if (!rb_full(tx_head, tx_tail, UART_RX_BUF_SIZE))
    {
        tx_buf[tx_head] = (uint8_t)ch;
        tx_head = next;
        if (__HAL_UART_GET_FLAG(&huart6, UART_FLAG_TXE))
            HAL_UART_Transmit_IT(&huart6, &tx_buf[tx_tail], 1);
    }
}
```

В режиме опроса: непосредственно передает байт через HAL_UART_Transmit с таймаутом

В режиме прерываний:

1. Проверяет наличие свободного места в буфере передачи
2. Добавляет данные в буфер
3. Если передатчик свободен (флаг TXE установлен), инициирует передачу

```
void uart_print_string(char *str) { while (*str)
uart_print_char(*str++); }
```

Последовательно передает каждый символ строки через `uart_print_char`

```
void uart_println_string(char *str)
{
     uart_print_string(str);
```

```
    uart_print_string("\r\n");
}
```
Дополняет базовую функцию передачи строки добавлением символов возврата каретки и перевода строк

Функции приема данных:
```
static int uart_read_char(char *ch)
{
    if (!interrupts_enabled)
    {
        if (HAL_UART_Receive(&huart6, (uint8_t *)ch, 1, 0) == HAL_OK)
          return 1;
        else
            return 0;
    }
    else
    {
     if (rb_empty(rx_head, rx_tail)) return 0;
        *ch = rx_buf[rx_tail];
        rx_tail = (rx_tail + 1) % UART_RX_BUF_SIZE;
        return 1;
    }
}
```
В режиме опроса: проверяет наличие данных без блокировки (таймаут = 0)

В режиме прерываний: извлекает данные из кольцевого буфера приема, если они доступны

Обработчики прерываний:
```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    if (huart->Instance != USART6) return;
    uint16_t next = (rx_head + 1) % UART_RX_BUF_SIZE;
    if (!rb_full(rx_head, rx_tail, UART_RX_BUF_SIZE))
    {
        rx_buf[rx_head] = rx_byte;
        rx_head = next;
    }
    HAL_UART_Receive_IT(&huart6, &rx_byte, 1);
}
```
Вызывается при завершении приема байта:

1. Проверяет источник прерывания
2. Добавляет принятый байт в кольцевой буфер приема
3. Перезапускает асинхронный прием для следующего байта

```
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
{
    if (huart->Instance != USART6) return;
    if (!rb_empty(tx_head, tx_tail))
    {
        tx_tail = (tx_tail + 1) % UART_TX_BUF_SIZE;
        if (!rb_empty(tx_head, tx_tail))
            HAL_UART_Transmit_IT(&huart6, &tx_buf[tx_tail], 1);
    }
}
```

Вызывается при завершении передачи байта:

1. Проверяет источник прерывания
2. Сдвигает указатель чтения буфера передачи
3. Если в буфере есть данные, инициирует передачу следующего байта

Обработка входящих команд:

```
void uart_receive_line_task(void) {
    char c;
    while (uart_read_char(&c)) {
        uart_print_char(c);
        switch (c) {
        case '\r':
        case '\n':
            if (s_line_len > 0) {
                s_line_buf[s_line_len] = '\0';
                uart_println_string("");
                process_command_line(s_line_buf);
                s_line_len = 0;
            }
            break;
        case '\b':
        case 0x7F:
            if (s_line_len > 0) s_line_len--;
            break;
        default:
            if (isprint((unsigned)c) && s_line_len < UART_MAX_LINE -
1)
                s_line_buf[s_line_len++] = c;
        }
    }
}
```

Функция обработки входящего потока данных реализует построчное чтение:

- **Эхо-вывод:** каждый принятый символ немедленно отправляется обратно
- **Обработка управляющих символов:**
    - \r и \n - завершение строки и вызов обработчика команд

    - \b и 0x7F - удаление последнего символа
- **Накопление строки:** сохраняет только печатные символы до достижения максимальной длины

Блок-схема прикладного алгоритма

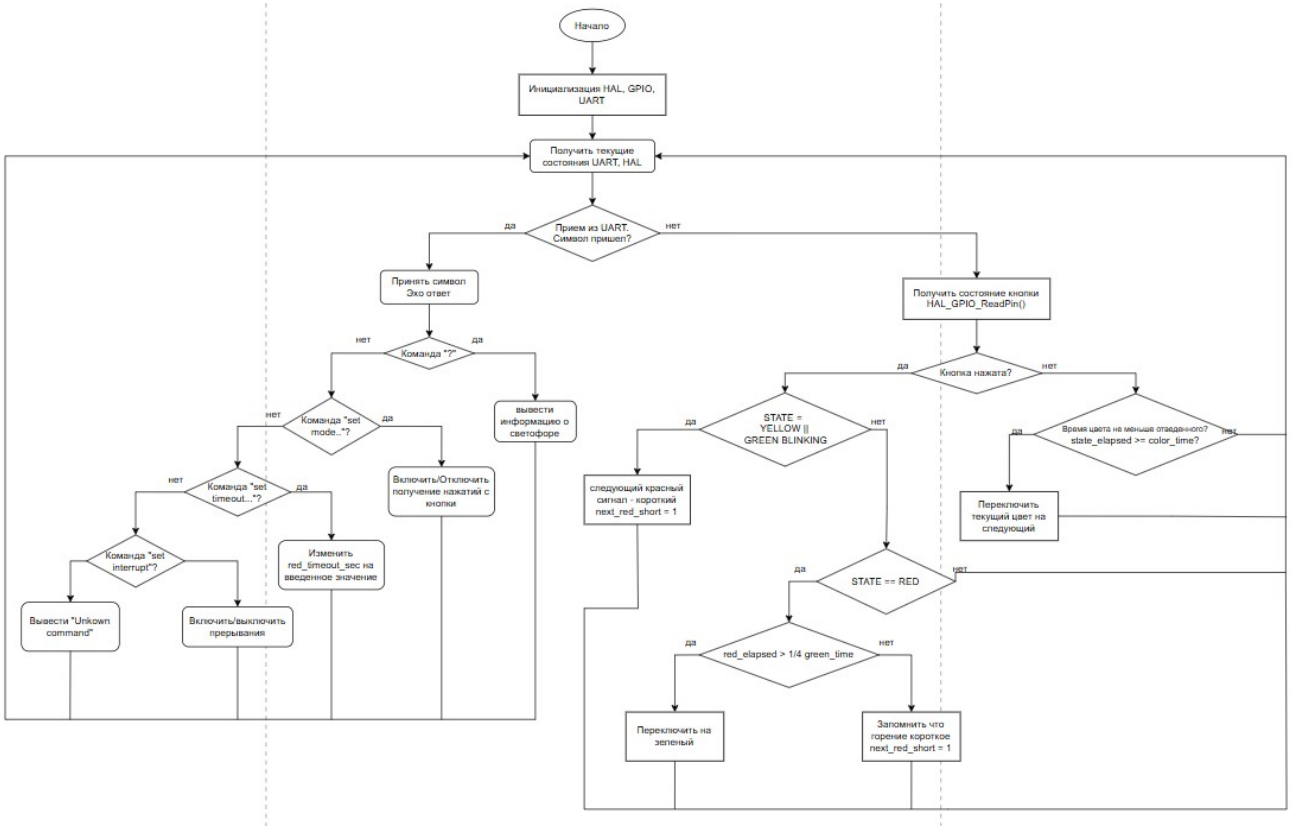

Рисунок 1: Блок-схема

Исходный код

```
#include "traffic_light_logic/command_process.h"
extern traffic_mode_t current_mode;
extern uint32_t red_timeout_sec;
extern traffic_light_state_t current_state;
void process_command_line(const char *line)
{
    char cmd[32];
    char arg1[32];
    char arg2[32];
    int count = sscanf(line, "%31s %31s %31s", cmd, arg1, arg2);
    if (count <= 0) {
        uart_println_string("Empty command");
        return;
    }
    command_type = CMD_UNKNOWN;
    if (strcmp(cmd, "?") == 0)
        command_type = CMD_STATUS;
    else if (strcmp(cmd, "set") == 0 && count >= 2)
    {
        if (strcmp(arg1, "mode") == 0) command_type = CMD_SET_MODE;
        else if (strcmp(arg1, "timeout") == 0) command_type =
CMD_SET_TIMEOUT;
        else if (strcmp(arg1, "interrupts") == 0) command_type =
CMD_SET_INTERRUPTS;
    }
    switch (command_type) {
    case CMD_STATUS:
    {
        char buffer[64];
        char *color = "";
        switch (current_state) {
              case STATE_RED:             color = "red"; break;
              case STATE_GREEN:           color = "green"; break;
              case STATE_GREEN_BLINKING:  color = "blinking green";
break;
              case STATE_YELLOW:          color = "yellow"; break;
              default:                     color = "unknown";
break;
          }
        // Color
        uart_print_string("State: ");
        uart_println_string((char *)color);

        // Mode
        uart_print_string("Mode: ");
        snprintf(buffer, sizeof(buffer), "%d", current_mode);
        uart_println_string(buffer);
```

```
        // Timeout
        uart_print_string("Timeout (sec): ");
        snprintf(buffer, sizeof(buffer), "%lu", (unsigned
long)red_timeout_sec / 1000);
        uart_println_string(buffer);

        // Interruption
        uart_print_string("Interruption: ");
        snprintf(buffer, sizeof(buffer), "%c",
uart_get_interrupts() ? 'I' : 'P');
        uart_println_string(buffer);

       break;
    }
    case CMD_SET_MODE:
    {
        if (count < 3) {
            uart_println_string("Usage: set mode 1|2");
            break;
        }
        int mode = atoi(arg2);
        if (mode == 1 || mode == 2) {
            current_mode = mode;
            uart_println_string("OK");
        } else {
            uart_println_string("ERROR: Invalid mode. Usage: set mode
1|2");
        }
        break;
    }
    case CMD_SET_TIMEOUT: {
        if (count < 3) {
            uart_println_string("Usage: set timeout X");
            break;
        }
        int t = atoi(arg2) * 1000;
        if (t > 0 && t > RED_TIME_SHORT_MS) {
            red_timeout_sec = (uint32_t)t;
            uart_println_string("OK");
        } else {
            uart_println_string("ERROR: Timeout must be more than
RED_TIME_SHORT");
        }
        break;
    }
    case CMD_SET_INTERRUPTS: {
        if (count < 3) {
            uart_println_string("Usage: set interrupts on|off");
```

```
            break;
        }
        if (strcmp(arg2, "on") == 0) {
            uart_set_interrupts(1);
            uart_println_string("Interrupts ON");
        } else if (strcmp(arg2, "off") == 0) {
            uart_set_interrupts(0);
            uart_println_string("Interrupts OFF");
        } else {
            uart_println_string("ERROR: Invalid parameter (use on|
off)");
        }
        break;
    }
    default:
        uart_println_string("Unknown command");
        break;
    }
}
```

Вывод

В ходе выполнения работы были успешно разработаны и реализованы два варианта драйверов для UART на стенде SDK-1.1M, а также доработана программа «светофор» с расширенным функционалом управления через UART-интерфейс.

Драйвер с использованием прерываний обеспечил неблокирующий режим работы с буферизацией данных, что исключило потерю символов при интенсивном обмене. Были реализованы кольцевые буферы для приёма и передачи, обработчики прерываний для их заполнения и опустошения, а также функции асинхронной отправки и чтения данных.

Драйвер без прерываний также работает в неблокирующем режиме, проверяя флаги состояния UART без ожидания данных. Это решение проще в реализации, но менее эффективно при высокой нагрузке, так как требует постоянного опроса регистров.