Университет ИТМО

Проектирование вычислительных систем

Лабораторная работа №2

Последовательный интерфейс UART

ФИО студентов:
Готовко Алексей Владимирович
Руденко Илья Александрович
Вариант: 5
Направление подготовки: 09.03.04 (СППО)
Учебная группа: P34101
ФИО преподавателя: Пинкевич Василий Юрьевич

Санкт-Петербург
2024г.

Содержание

1 Цели работы

1. Изучить протокол передачи данных по интерфейсу UART.
2. Получить базовые знания об организации системы прерываний в микроконтроллерах на примере микроконтроллера STM32.
3. Изучить устройство и принципы работы контроллера интерфейса UART, получить навыки организации обмена данными по UART в режимах опроса и прерываний.

2 Вариант задания

Доработать программу кодового замка. Теперь ввод кода должен происходить не с помощью кнопки стенда, а по UART. После ввода единственно верной последовательности из не более чем восьми латинских букв без учёта регистра и цифр должен загореться зелёный светодиод, обозначающий «открытие» замка. Светодиод горит некоторое время, потом гаснет, и система вновь переходит в «режим ввода». Каждый неправильно введённый элемент последовательности должен сопровождаться миганием красного светодиода и сбросом в «начало», каждый правильный – миганием жёлтого. После трёх неправильных вводов начинает мигать красный светодиод, и через некоторое время система вновь возвращается в «режим ввода». Если код не введен до конца за некоторое ограниченное время, происходит сброс в «начало». Должно быть предусмотрено изменение отпирающей последовательности, что производится следующей последовательностью действий:

- ввод символа «+»;
- ввод новой последовательности, который завершается либо по нажатию `enter`, либо по достижении восьми значений;
- стенд отправляет сообщение произвольного содержания, спрашивая, сделать ли последовательность активной, и запрашивает подтверждение, которое должно быть сделано вводом символа `y`;
- после ввода `y` введённая последовательность устанавливается как активная.

Включение/отключение прерываний должно осуществляться нажатием кнопки на стенде и сопровождаться отправкой в последовательный порт сообщения произвольного содержания, сообщающего, какой режим включен (с прерываниями или без прерываний).

Скорость обмена данными по UART – 9600 бит/с.

3 Описание программы

Опустим объяснение всей логики, реализованной и описанной в предыдущей лабораторной работе.

У программы есть три режима работы:

- `DEFAULT` – режим по умолчанию, когда происходит ввод пароля;
- `PASS_CHANGE` – режим ввода нового пароля;
- `CONFIRMATION` – режим подтверждения смены пароля.

В качестве буферов для асинхронного ввода и вывода в/из UART заведем два кольцевых буфера: `receive_buffer` и `transmit_buffer`. Также заведем две `bool` переменные, отвечающих за то, происходит ли в настоящий момент чтение или запись в UART: `receive_busy` и `transmit_busy`. Эти переменные будут выставляться в `true` при вызове асинхронных функций чтения/записи и в `false` после выполнения callback-функций.

Также реализуем функцию `process_char` проверки валидности введенного символа в зависимости от текущего режима работы.

Наконец, в функции `disable_interruptions` позаботимся о том, чтобы до перехода из режима прерываний в режим без прерываний все данные из `transmit_buffer` были переданы и не были потеряны безвозвратно.

На каждой итерации программы будем выполнять следующую последовательность действий:

1. проверяем, была ли нажата кнопка, и в случае нажатия меняем режим (с прерываниями или без);
2. если текущий режим `DEFAULT`, ввод уже начался и с момента последнего ввода прошло время таймаута, то сбрасываем текущий ввод и сигнализируем об этом через светодиоды;
3. пробуем считать символ из UART и записать его в `receive_buffer`;
4. в случае успеха смотрим на текущий режим работы и делаем следующее:

   режим `DEFAULT`:

   - если нажат "+" , переходим в режим `PASS_CHANGE`;
   - в ином случае сверяем введенный символ с правильным символом пароля и сигнализируем об успехе или ошибке через светодиоды;

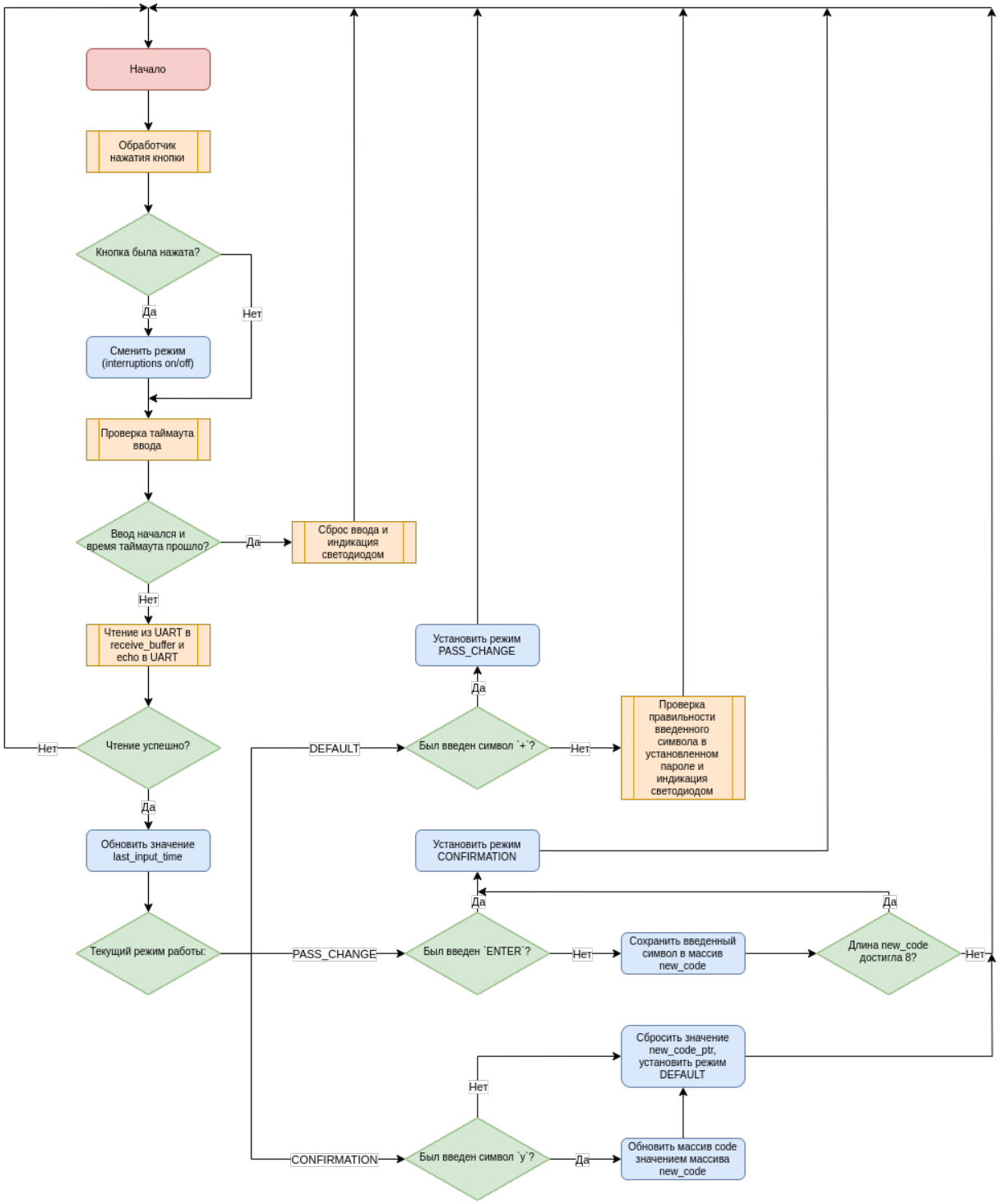
   режим `PASS_CHANGE`:

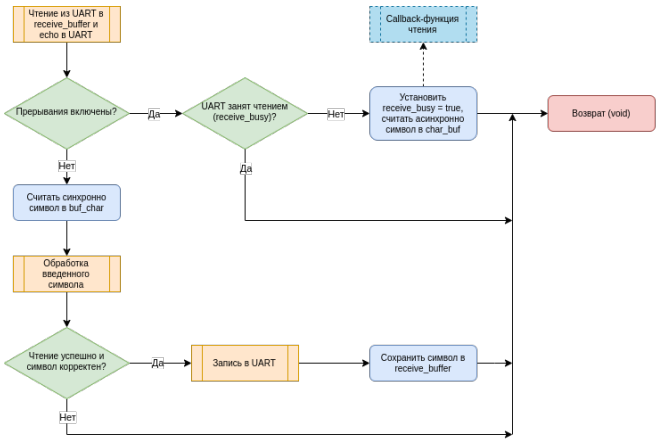
   - если введен `enter`, переходим в режим `CONFIRMATION`;
   - в ином случае сохраняем введенный символ в массив нового пароля `new_code` и при достижении количества символов в `new_code` восьми переходим в режим `CONFIRMATION`;

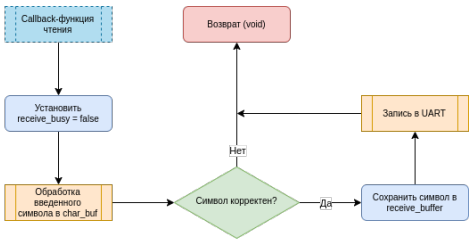
   режим `CONFIRMATION`:

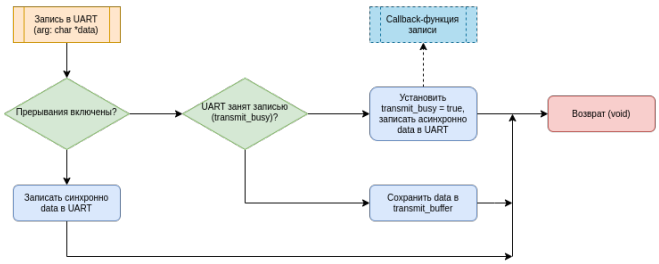
   - если введен символ `y`, перезаписываем массив пароля `code` данными из `new_code`;
   - сбрасываем в начальное состояние массив `new_code` и переходим в режим `DEFAULT`.

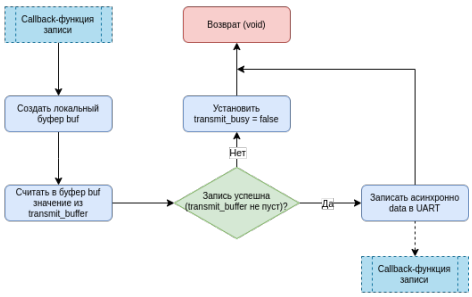
4 Блок-схема алгоритма

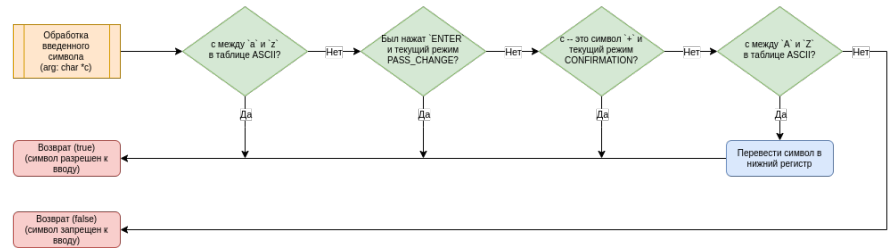



Чтение из UART в receive_buffer и echo в UART
Прерывания включены?
Да
UART занят чтением (receive_busy)?
Нет
Да
Нет
Считать синхронно символ в buf_char
Обработка введенного символа
Чтение успешно и символ корректен?
Да
Запись в UART
Сохранить символ в receive_buffer
Нет
Callback-функция чтения
Установить receive_busy = true, считать асинхронно символ в char_buf
Возврат (void)


Callback-функция чтения
Установить receive_busy = false
Обработка введенного символа в char_buf
Символ корректен?
Нет
Да
Сохранить символ в receive_buffer
Запись в UART
Возврат (void)


Запись в UART (arg: char *data)
Прерывания включены?
UART занят записью (transmit_busy)?
Записать синхронно data в UART
Сохранить data в transmit_buffer
Установить transmit_busy = true, записать асинхронно data в UART
Возврат (void)
Callback-функция записи


Callback-функция записи
Создать локальный буфер buf
Считать в буфер buf значение из transmit_buffer
Запись успешна (transmit_buffer не пуст)?
Нет
Да
Установить transmit_busy = false
Записать асинхронно data в UART
Callback-функция записи
Возврат (void)


Обработка введенного символа (arg: char *c)
с между `a` и `z` в таблице ASCII?
Нет
Был нажат `ENTER` и текущий режим PASS_CHANGE?
Нет
`c` -- это символ `+` и текущий режим CONFIRMATION?
Нет
с между `A` и `Z` в таблице ASCII?
Нет
Да
Да
Да
Да
Перевести символ в нижний регистр
Возврат (true) (символ разрешен к вводу)
Возврат (false) (символ запрещен к вводу)

5 Код программы

Вспомогательные определения:

```
#define BUTTON_PIN              GPIO_PIN_15
#define GREEN_LED_PIN           GPIO_PIN_13
#define YELLOW_LED_PIN          GPIO_PIN_14
#define RED_LED_PIN             GPIO_PIN_15
#define RESET_ALL_TIMEOUT_MS    1500
#define ENTER_ASCII             '\r'
#define ASCII_LATIN_CASE_DIFF   32
#define BUF_SIZE                1024

typedef enum {
    DEFAULT = 0,
    PASS_CHANGE,
    CONFIRMATION
} InputMode;

typedef struct {
    bool is_pressed;
    bool signaled;
    uint32_t press_start_time;
} ButtonState;

typedef struct RingBuffer {
    char data[BUF_SIZE];
    uint8_t head;
    uint8_t tail;
    bool is_empty;
} RingBuffer;
```

Интерфейс буфера:

```
void buf_init(RingBuffer *buf) {
    buf->head = 0;
    buf->tail = 0;
    buf->is_empty = true;
}

void buf_write(RingBuffer *buf, char *in) {
    uint32_t pmask = __get_PRIMASK();
    __disable_irq();

    uint64_t size = strlen(in);

    if (buf->head + size + 1 > BUF_SIZE)
        buf->head = 0;

    strcpy(&buf->data[buf->head], in);
    buf->head += size + 1;

    if (buf->head == BUF_SIZE)
        buf->head = 0;

```

```
    buf->is_empty = false;

    __set_PRIMASK(pmask);
}

bool buf_read(RingBuffer *buf, char *out) {
    uint32_t pmask = __get_PRIMASK();
    __disable_irq();

    if (buf->is_empty){
        __set_PRIMASK(pmask);
        return false;
    }

    uint64_t size = strlen(&buf->data[buf->tail]);

    strcpy(out, &buf->data[buf->tail]);
    buf->tail += size + 1;

    if (buf->tail == BUF_SIZE || buf->tail == '\0')
        buf->tail = 0;

    if (buf->tail == buf->head)
        buf->is_empty = true;

    __set_PRIMASK(pmask);
    return true;
}
```

Вспомогательные функции:

```
// here char *c is null-terminated char (char[2] = {'\0', '\0'})
bool process_char(char *c) {
    if ('A' <= c[0] && c[0] <= 'Z') {
        c[0] += ASCII_LATIN_CASE_DIFF;
        return true;
    }
    if ('a' <= c[0] && c[0] <= 'z') {
        return true;
    }
    if (c[0] == ENTER_ASCII && PASS_CHANGE == mode) {
        return true;
    }
    if (c[0] == '+' && DEFAULT == mode) {
        return true;
    }
    c[0] = '\0';
    return false;
}

bool update_button_state(void) {
    if (button_state.is_pressed) {
        button_state.is_pressed = read_button() == GPIO_PIN_RESET;
        if (button_state.signaled) return false;
        if ((HAL_GetTick() - button_state.press_start_time) > 20) {
```

```
            button_state.signaled = true;
            return true;
        }
        return false;
    }
    if (read_button() == GPIO_PIN_RESET) {
        button_state.press_start_time = HAL_GetTick();
        button_state.is_pressed = true;
        button_state.signaled = false;
    }
    return false;
}

void set_mode(InputMode new_mode) {
    uart_write_newline();
    mode = new_mode;
    switch (mode) {
        case DEFAULT:
            code_ptr = 0;
            break;
      case CONFIRMATION:
          if (new_code_ptr == 0) {
              set_mode(DEFAULT);
              break;
          }
          uart_write(confirmation_message);
          break;
      default:
          break;
    }
}

void enable_interruptions(void) {
    HAL_NVIC_EnableIRQ(USART6_IRQn);
    interruptions_enabled = true;
}

void disable_interruptions(void) {
    while (transmit_busy) ; // wait until everything from transmit_buffer is sent to the UART
    HAL_UART_AbortReceive(&huart6);
    HAL_NVIC_DisableIRQ(USART6_IRQn);
    interruptions_enabled = false;
}
```

Функции-драйверы:

```
GPIO_PinState read_button(void) {
    return HAL_GPIO_ReadPin(GPIOC, BUTTON_PIN);
}

void light_led(uint16_t led_pin, uint8_t blink_cnt, uint8_t time) {
    while (blink_cnt--) {
        HAL_GPIO_WritePin(GPIOD, led_pin, GPIO_PIN_SET);
        HAL_Delay(time);
        HAL_GPIO_WritePin(GPIOD, led_pin, GPIO_PIN_RESET);
```

```
        HAL_Delay(time);
    }
}

void uart_write(char *data) {
    uint16_t size = strlen(data);
    if (interruptions_enabled) {
        if (transmit_busy) {
            buf_write(&transmit_buffer, data);
        } else {
            transmit_busy = true;
            HAL_UART_Transmit_IT(&huart6, (uint8_t *) data, size);
        }
    } else HAL_UART_Transmit(&huart6, (uint8_t *) data, size, 100);
}

void uart_write_newline(void) {
    uart_write(newline);
}

void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart) {
    char buf[BUF_SIZE];
    if (buf_read(&transmit_buffer, buf))
        HAL_UART_Transmit_IT(&huart6, (uint8_t *) &buf, strlen(buf));
    else transmit_busy = false;
}

void uart_read_char(void) {
    if (interruptions_enabled) {
        if (receive_busy) return;
        receive_busy = true;
        HAL_UART_Receive_IT(&huart6, (uint8_t *) buf_char, sizeof(char));
        return;
    }
    HAL_StatusTypeDef status = HAL_UART_Receive(&huart6, (uint8_t *) buf_char, sizeof(char),
    ↪  100);
    if (HAL_OK == status && process_char(buf_char)) {
        uart_write(buf_char);
        buf_write(&receive_buffer, buf_char);
    }
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
    receive_busy = false;
    if (process_char(buf_char)) {
        buf_write(&receive_buffer, buf_char);
        uart_write(buf_char);
    }
}
```

Основной цикл программы:

```
disable_interruptions();
uart_write(interruptions_off_message);
uart_write_newline();
```

```
4
5   buf_init(&receive_buffer);
6   buf_init(&transmit_buffer);
7
8   char tmp[2];
9   while (1)
10  {
11      if (update_button_state()) {
12          if (interruptions_enabled) {
13              disable_interruptions();
14              uart_write(interruptions_off_message);
15              uart_write_newline();
16          } else {
17              enable_interruptions();
18              uart_write(interruptions_on_message);
19              uart_write_newline();
20          }
21      }
22
23      if (DEFAULT == mode && code_ptr != 0 && HAL_GetTick() - last_input_time >=
        ↪  RESET_ALL_TIMEOUT_MS) {
24          code_ptr = 0;
25          failed_attempts_cnt = 0;
26          uart_write_newline();
27          light_led(RED_LED_PIN | GREEN_LED_PIN, 5, 25);
28          last_input_time = HAL_GetTick();
29          continue;
30      }
31
32      uart_read_char();
33
34      if (!buf_read(&receive_buffer, tmp))
35          continue;
36
37      last_input_time = HAL_GetTick();
38      cur_char = tmp[0];
39
40      switch (mode) {
41          case DEFAULT:
42              if (cur_char == '+') {
43                set_mode(PASS_CHANGE);
44                continue;
45              }
46
47              if (code[code_ptr++] == cur_char) {
48                  light_led(YELLOW_LED_PIN, 1, 100);
49                  if (code_ptr == code_length) {
50                      uart_write_newline();
51                      light_led(GREEN_LED_PIN, 3, 50);
52                      code_ptr = 0;
53                      failed_attempts_cnt = 0;
54                  }
55              } else {
56                  uart_write_newline();
57                  light_led(RED_LED_PIN, 1, 100);
```

```
                code_ptr = 0;
                if (++failed_attempts_cnt == 3) {
                    failed_attempts_cnt = 0;
                    light_led(RED_LED_PIN, 3, 50);
                }
            }
            continue;

        case PASS_CHANGE:
            if (cur_char == ENTER_ASCII) {
                set_mode(CONFIRMATION);
                continue;
            }

            new_code[new_code_ptr++] = cur_char;
            if (new_code_ptr == 8) set_mode(CONFIRMATION);
                continue;

        case CONFIRMATION:
            if (cur_char == 'y') {
                code_length = new_code_ptr;
                for (uint8_t i = 0; i < code_length; i++)
                    code[i] = new_code[i];
                }
            new_code_ptr = 0;
            set_mode(DEFAULT);
            continue;
}
```

6 Вывод

В результате выполнения лабораторной работы был изучен протокол передачи данных по интерфейсу UART, получены базовые знания об организации системы прерываний в микроконтроллерах на примере микроконтроллера STM32, а также изучено устройство и принципы работы контроллера интерфейса UART и получены навыки организации обмена данными по UART в режимах опроса и прерываний.