Национальный исследовательский университет ИТМО
Факультет Программной Инженерии и Компьютерной Техники

Вариант №3
Лабораторная работа №2
«Последовательный интерфейс UART»
По дисциплине:
«Проектирование вычислительных систем»

Работу выполнили:
Никонова Наталья Игоревна
Сенина Мария Михайловна
Группа: P34102
Преподаватель:
Пинкевич Василий Юрьевич

Санкт-Петербург
2023

Цель 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.

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. 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.

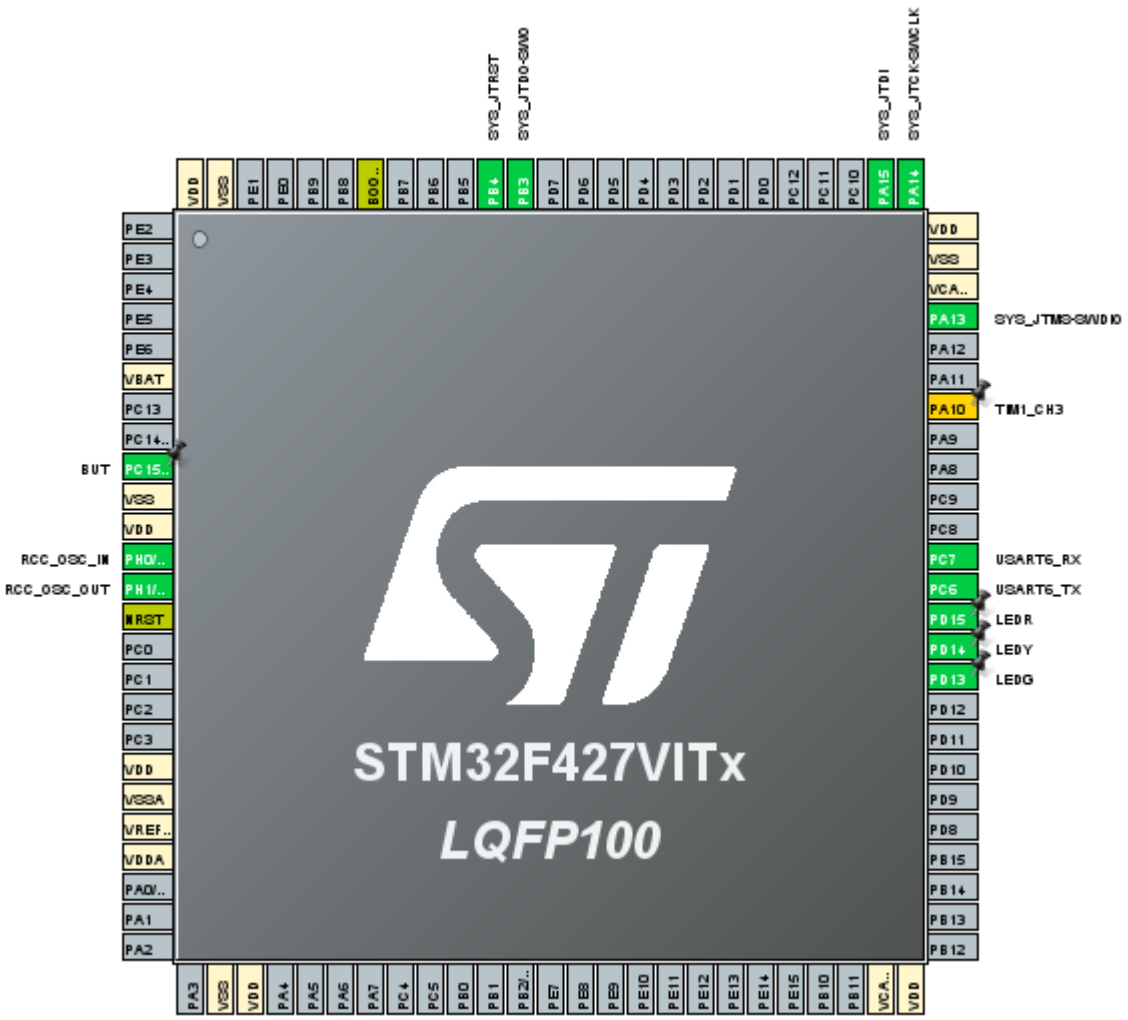
Доработать программу, посылающую сигналы азбуки Морзе для латинского алфавита. Последовательность точек и тире, полученных нажатием кнопки стенда, должна дешифроваться и отправляться через последовательный канал в виде символов (букв латинского алфавита).

Символы, получаемые стендом через последовательный канал, должны шифроваться и «проигрываться» с помощью светодиода. Если в момент «мигания» приходят новые символы, они добавляются в очередь. Должна быть возможность одновременной буферизации до восьми символов: они должны запоминаться стендом и последовательно проигрываться. Считывание нажатий кнопки и отправка дешифрованных функционировать одновременно с приёмом через последовательный канал и миганием светодиода.

Включение/выключение прерываний должно осуществляться при получении стендом символа «+».

Скорость обмена данными по UART: 115200 бит/с.

Используемые контакты



- PC15 – перехватывает нажатия кнопки (обозначен как BUT, настроен на GPIO_INPUT)
- PD13 – управляет зеленым светодиодом (обозначен как LEDG, настроен на GPIO_OUTPUT)
- PD14 - управляет желтым светодиодом (обозначен как LEDY, настроен на GPIO_OUTPUT)
- PD15 - управляет красным светодиодом (обозначен как LEDR, настроен на GPIO_OUTPUT)
- PA10 – получает прерывания от таймера (обозначен как TIM1_CH3)
- PC7 – получает прерывание RX о том, что uart завершил прием данных
- PC6 – получает прерывание TX о том, что uart завершил отправку данных
- PB3, PB4, PA13, PA14, PA15 – J-Tag для отладки

Описание алгоритма

Модуль ввода-вывода

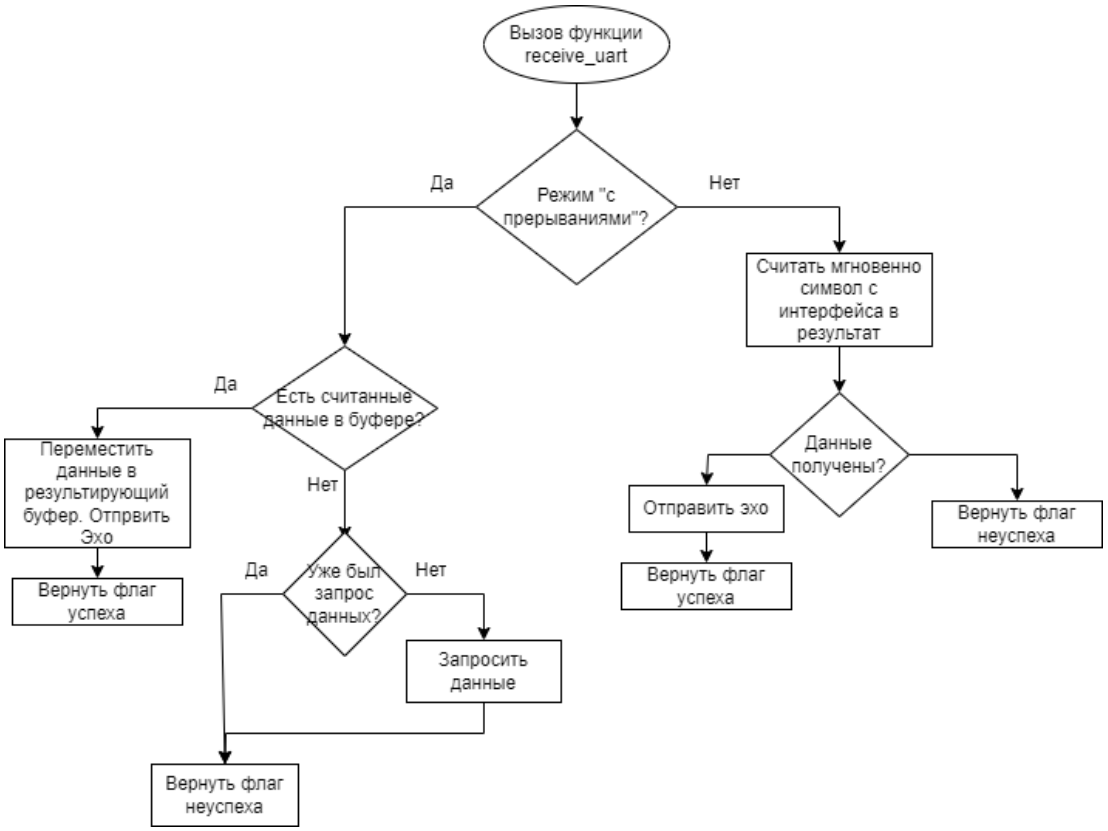
Данный модуль инкапсулирует в себе и буферизацию данных при отправке/считывании, и логику отправки по одному байту, и разных механизм работы в зависимости от режима с прерываниями или нет.

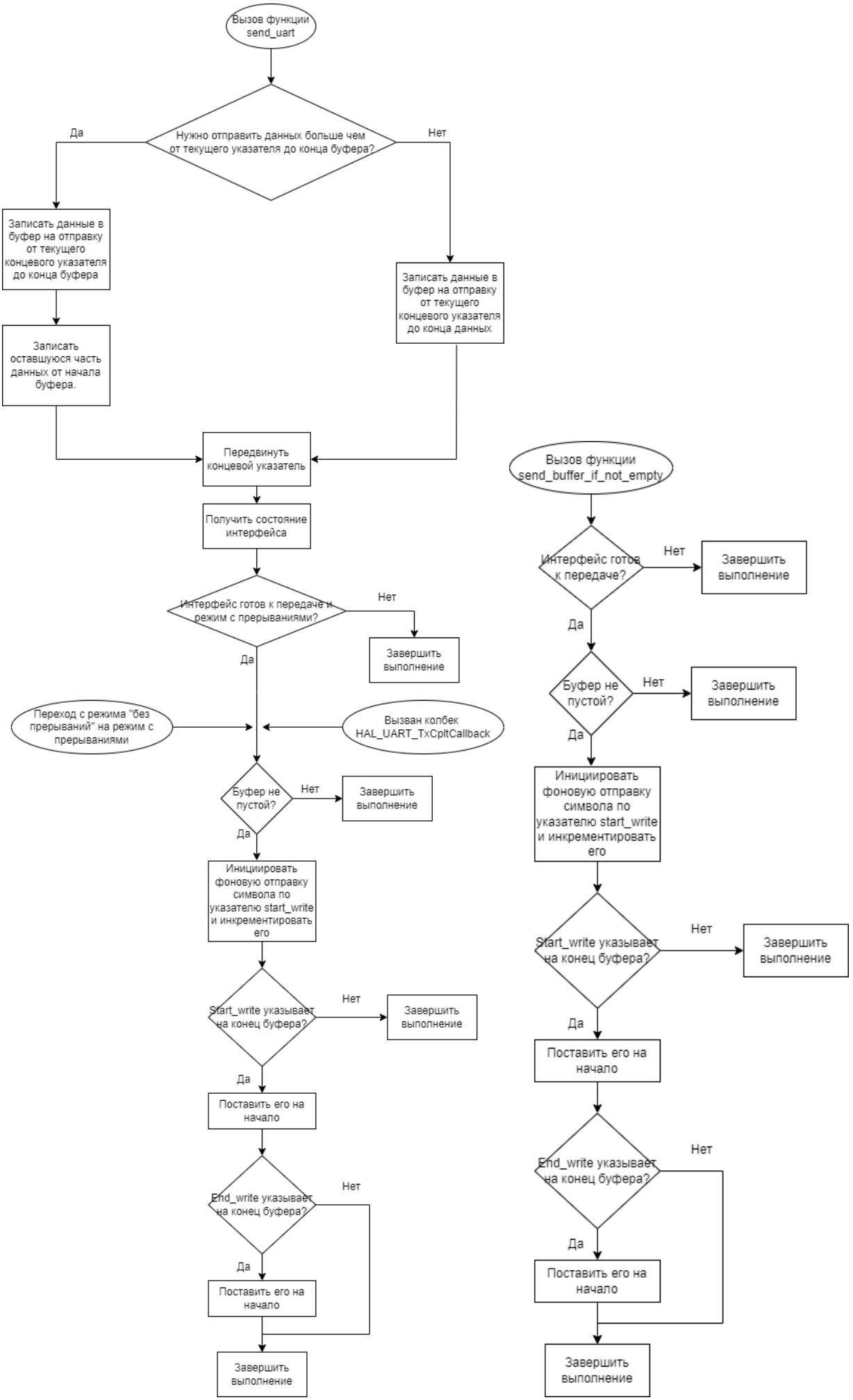
Для отправки у нас есть кольцевой буфер и два указателя - на символ, с которого надо начать следующую отправку и после последнего символа на отправку. Если указатели равны, то буфер пустой, ничего отправлять не надо.

При вызове функции send_uart данные на отправку записываются в этот буфер и проверяется состояние интерфейса. Если он готов к отправке и режим “с прерываниями”, то сразу же вызывается отправка первого символа. Такая же оптправка вызывается при срабатывании колбека, сообщающего что интерфейс закончил отправку данных. Также эту функцию нужно вызвать при переходе на режим “с прерываниями” если есть недосланные данные.

Для режима “без прерываний” есть отдельная функция send_buffer_if_not_empty, которая вызывается на каждой итерации цикла после полезной нагрузки, и блокирует поток на 100 мс, пытаясь отправить один символ, если буфер не пустой и интерфейс готов к передаче.

При получении данных алгоритмы с и без прерываний различаются с самого начала. Буфер, куда надо положить результат, функция принимает на входе, а возвращает флаг успеха. Без прерываний мы просто пробуем считать символ мгновенно, и возвращаем соответствующий результат. С прерываниями первый вызов функции всегда вернет неуспех, так как он только синициализирует ожидание приема данных. А так в нем проверяется, есть ли данные в внутреннем буфере, который заполняется колбеком, и если есть - то они возвращаются.




Вызов функции send_uart
Нужно отправить данных больше чем от текущего указателя до конца буфера?
Да
Нет
Записать данные в буфер на отправку от текущего концевого указателя до конца буфера
Записать оставшуюся часть данных от начала буфера.
Записать данные в буфер на отправку от текущего концевого указателя до конца данных
Передвинуть концевой указатель
Получить состояние интерфейса
Интерфейс готов к передаче и режим с прерываниями?
Нет
Завершить выполнение
Да
Переход с режима "без прерываний" на режим с прерываниями
Вызван колбек HAL_UART_TxCpltCallback
Буфер не пустой?
Нет
Завершить выполнение
Да
Инициировать фоновую отправку символа по указателю start_write и инкрементировать его
Start_write указывает на конец буфера?
Нет
Завершить выполнение
Да
Поставить его на начало
End_write указывает на конец буфера?
Нет
Да
Поставить его на начало
Завершить выполнение
Вызов функции send_buffer_if_not_empty
Интерфейс готов к передаче?
Нет
Завершить выполнение
Да
Буфер не пустой?
Нет
Завершить выполнение
Да
Инициировать фоновую отправку символа по указателю start_write и инкрементировать его
Start_write указывает на конец буфера?
Нет
Завершить выполнение
Да
Поставить его на начало
End_write указывает на конец буфера?
Нет
Да
Поставить его на начало
Завершить выполнение

```
uint8_t buffer_to_write[WRITE_BUFFER_SIZE] = {0}; // буфер на передачу
size_t start_write = 0; // номер символа с которого начинать передачу
size_t end_write = 0; // номер символа следом за последним символом, который надо передать

char buffer_to_read = '0';
bool is_read_buffer_full = false;
bool is_wait_read = false;

void send_buffer_if_not_empty_IT(UART_HandleTypeDef *huart){
	if (start_write != end_write) {
		HAL_UART_Transmit_IT(huart, (uint8_t*) (buffer_to_write + start_write), 1);
		start_write++;
		if (start_write == WRITE_BUFFER_SIZE) {
			start_write = 0;
			if (end_write == WRITE_BUFFER_SIZE) {
				end_write = 0;
			}
		}
	}
}

void send_buffer_if_not_empty(UART_HandleTypeDef *huart){
	char state = HAL_UART_GetState(huart);
	if (state != HAL_UART_STATE_BUSY_TX) {
		if (start_write != end_write) {
			HAL_UART_Transmit(huart, (uint8_t*) (buffer_to_write + start_write), 1, 100);
			start_write++;
			if (start_write == WRITE_BUFFER_SIZE) {
				start_write = 0;
				if (end_write == WRITE_BUFFER_SIZE) {
					end_write = 0;
				}
			}
		}
	}
}

void send_uart(UART_HandleTypeDef *huart, uint8_t* buffer, size_t buf_size, int has_irq) {
	// добавить данные в буффер
	char state = HAL_UART_GetState(huart);
	if (buf_size > WRITE_BUFFER_SIZE - end_write) {
		size_t first_size = WRITE_BUFFER_SIZE - end_write;
		if (first_size > 0) {
			memcpy(buffer_to_write + end_write, buffer, first_size);
		}
```

```
                memcpy(buffer_to_write, buffer, buf_size - first_size);
                end_write = buf_size - first_size;
        } else {
                memcpy(buffer_to_write + end_write, buffer, buf_size);
                end_write += buf_size;
        }

        if (state != HAL_UART_STATE_BUSY_TX) {
                // свободно, начать передачу
                if (has_irq == 1) send_buffer_if_not_empty_IT(huart);
        }
}

int receive_uart(UART_HandleTypeDef *huart, uint8_t* buffer, int has_irq){ //проверить, что таким образом я считала всё
        if (has_irq == 1) {
                if (!is_read_buffer_full) {
                        if (!is_wait_read){
                                HAL_UART_Receive_IT(huart, (uint8_t*) &buffer_to_read, 1);
                                is_wait_read = true;
                        }
                        return 0;
                } else {
                        memcpy(buffer, &buffer_to_read, 1);
                        is_read_buffer_full = false;
                        send_uart(huart, buffer_to_read, 1, has_irq);
                        return 1;
                }
        } else {
                bzero(buffer, 1);
                HAL_StatusTypeDef stat = HAL_UART_Receive(huart, (uint8_t*)buffer, 1, 0);
                switch (stat) {
                        case HAL_OK: {

                                send_uart(huart, (char*) buffer, 1, has_irq);
                                return 1;
                        }
                        case HAL_ERROR:
                        case HAL_BUSY:
                        case HAL_TIMEOUT:
                        break;
                }
                return 0;
        }
}
```

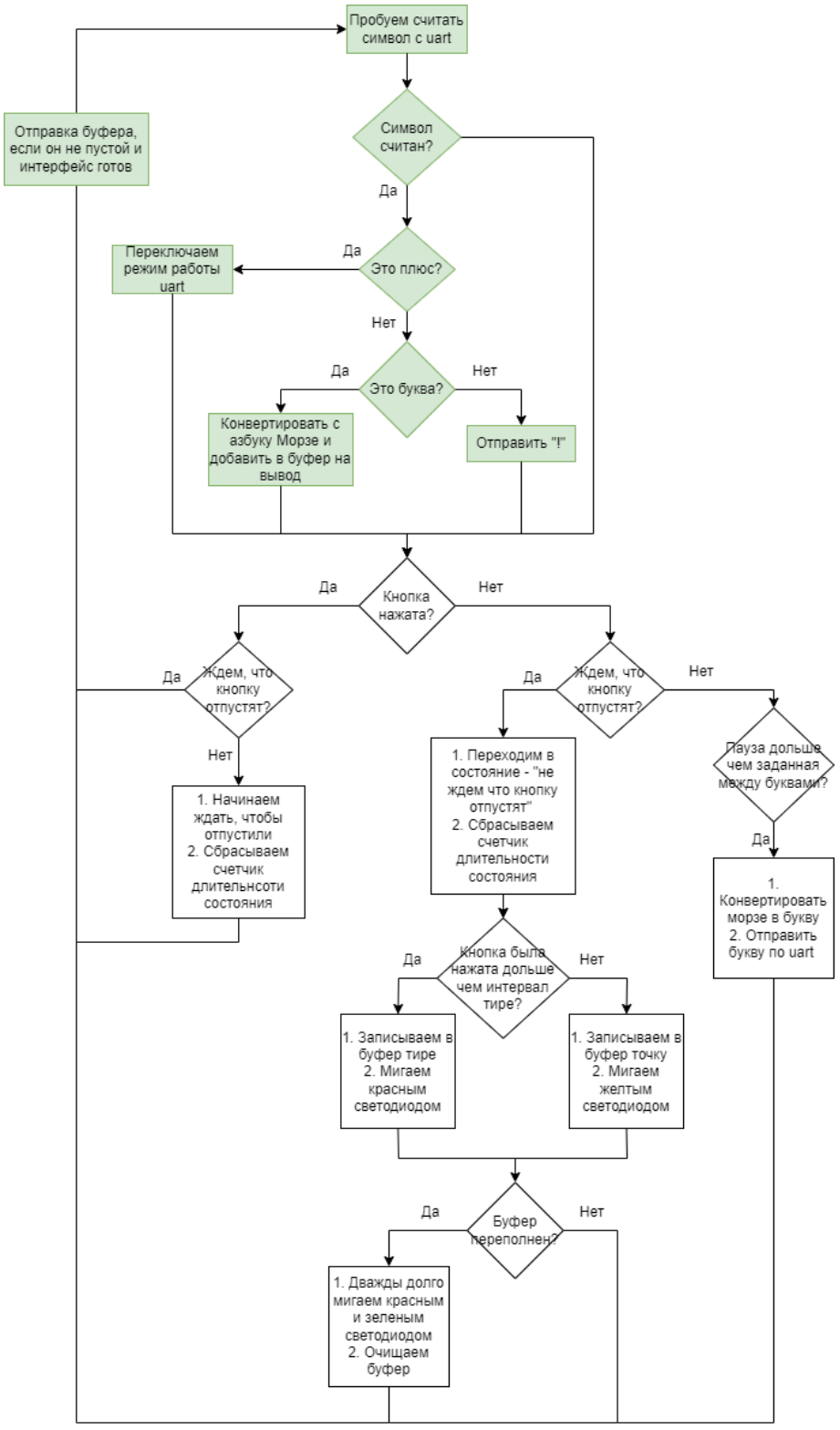
```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
  if (huart->Instance == USART6)
  {
    // USART6 завершил прием данных
          is_read_buffer_full = true;
          is_wait_read = false;
  }
}

void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart){
  if (huart->Instance == USART6)
  {
      // USART6 завершил отправку данных
          send_buffer_if_not_empty_IT(huart);
  }
}
```

Модуль основной логики

К тому, что было в предыдущей лабе, в начале мы добавили считывание символа и его эхо-отправку. Если это плюс, то переключаем режим, если это буква, конвертируем в морзе и добавляем в буфер на вывод светодиоду, иначе отправляем ! как сигнал о неизвестном символе. Далее чуть измененная логика из первой лабы. Изменили то, что мы считываем одну букву - и если больше 4 знаков ввели, то уже переполнение, и мы очищаем буффер на ввод. Далее вызываем функцию сброса буфера вывода в интерфейс - отправки, если интерфейс готов и буфер не пустой. В режиме с прерываниями эта функция никогда не сработает, потому что либо буфер будет пустой, либо интерфейс не будет готов к отправке - как только он будет готов, вызовется колбек, и данные отправятся из буфера с помощью него.

Также в обработчике прерываний есть функция, выводящая буфер с данными пришедшими с uart




```
while (1)
{
	int ret = receive_uart(&huart6, &str, has_irq);
	if (ret > 0) { //есть что сконвертировать в 0 и 1
		char* res = char_to_morze(str);
		if (res[0] == '+') {
			has_irq = (has_irq + 1) % 2;
```

```
                        if (has_irq == 1) {
                                send_buffer_if_not_empty_IT(&huart6);
                                NVIC_EnableIRQ(USART6_IRQn); // enable interrupts
                        } else NVIC_DisableIRQ(USART6_IRQn); // disable interrupts
                }else if(res[0] == '-') {
                        char i = '!';
                        send_uart(&huart6, &i , 1, has_irq);
                } else {
                        for (int i = 0; i < strlen(res); i++) {
                                int_w_buf[writing_ptr] = res[i] - '0';
                                writing_ptr++;
                        }
                }


}
if (0 == is_pressed && 0 == is_wait_unpressed) { // нопка не нажата
         if (buf_ptr != 0 && count_tick > BETWEEN_LETTERS_PERIOD_CT) {
                  uint8_t to_write = morze_to_str(buffer, buf_ptr);
                  buf_ptr = 0;
                  send_uart(&huart6, &to_write, 1, has_irq);
         }
} else if (0 == is_pressed && 1 == is_wait_unpressed) { //кнопку отпустили
        is_wait_unpressed = 0;
        if (count_tick > LONG_PERIOD_CT) { // вносим 1 или 0 в буфер на отправку
                turn_on_red_led();
                HAL_Delay(DEFAULT_DELAY);
                turn_off_red_led();
                buffer[buf_ptr] = 1;
        } else {
                turn_on_yellow_led();
                HAL_Delay(DEFAULT_DELAY);
                turn_off_yellow_led();
                buffer[buf_ptr] = 0;
        }
        count_tick = 0;
        buf_ptr++;
        if (BUFFER_SIZE == buf_ptr) {
                for (int i = 0; i < ERROR_COUNT; i++) { //если буфер переполнится
                        turn_on_red_led();
                        turn_on_green_led();
                        HAL_Delay(ERROR_DELAY);
                        turn_off_red_led();
                        turn_off_green_led();
                        HAL_Delay(ERROR_DELAY);
```

```
                        }
                        buf_ptr = 0;
                }
        } else if (1 == is_pressed && 0 == is_wait_unpressed) { //кнопку нажали -- ждём когда отпустят
                is_wait_unpressed = 1;
                count_tick = 0;
        }

        send_buffer_if_not_empty(&huart6);
  }

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
        if(htim->Instance==TIM1){


                int but = HAL_GPIO_ReadPin(BUT_GPIO_Port, BUT_Pin);
                but = !but;

                if(but == 1 && noisy == 0){
                        noisy = 1;
                } else if(but == 1 && noisy == 1) {
                        is_pressed = 1;
                        count_tick++;
                } else if(noisy == 1 && but == 0){
                        noisy = 0;
                } else if(but == 0 && noisy == 0){
                        is_pressed = 0;
                        count_tick++;
                }

    if(writing_ptr > 0){
                        if(is_green){ //надо выключить
                                if (dash_delay > 0){
                                        if (dash_delay < DASH_DELAY){ // ждём пока пауза не пройдёт
                                                dash_delay++;
                                        }else{ // пауза прошла -- меняем сигнал
                                                dash_delay = 0;
                                                turn_off_green_led();
                                                is_green = 0;
                                                current_write_ptr++;
                                        }
                                }else if(comma_delay > 0){
                                        if (comma_delay < COMMA_DELAY){ // ждём пока пауза не пройдёт
                                                comma_delay++;
```

```
                                        }else{ // пауза прошла -- меняем сигнал
                                                comma_delay = 0;
                                                turn_off_green_led();
                                                is_green = 0;
                                                current_write_ptr++;
                                        }
                                }
                        }else{ //надо включить
                                if (default_delay > DEFAULT_DELAY){ // ждём следующего момента выводить?
                                        // зажигаем зелёный цвет и включаем флаги ожидания
                                        default_delay = 0;
                                        turn_on_green_led();
                                        is_green = 1;
                                        if(int_w_buf[current_write_ptr] == 1){
                                                dash_delay = 1;
                                        }else{
                                                comma_delay = 1;
                                        }
                                }else{ // пока ещё ждём
                                        default_delay++;
                                }

                        }

                        if (current_write_ptr == writing_ptr){ // TODO: проверить, что прерывания не ломают
writing_ptr
                                current_write_ptr = 0;
                                writing_ptr = 0;
                        }
                }

        }

}
```

Вывод

В ходе выполнения этой лабораторной работы мы познакомились с последовательным интерфейсом UART и его работой в двух режимах. И поняли, что с ним лучше работать посимвольно, а не строками.