Национальный исследовательский университет ИТМО

Факультет Программной Инженерии и Компьютерной Техники

Вариант №3

Лабораторная работа №1

«Интерфейсы ввода-вывода общего назначения (GPIO)»

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. Получить базовые знания о принципах устройства программировании микроконтроллеров.
2. Изучить устройство интерфейсов ввода-вывода общего назначения (GPIO) в микроконтроллерах и приемы использования данных интерфейсов.

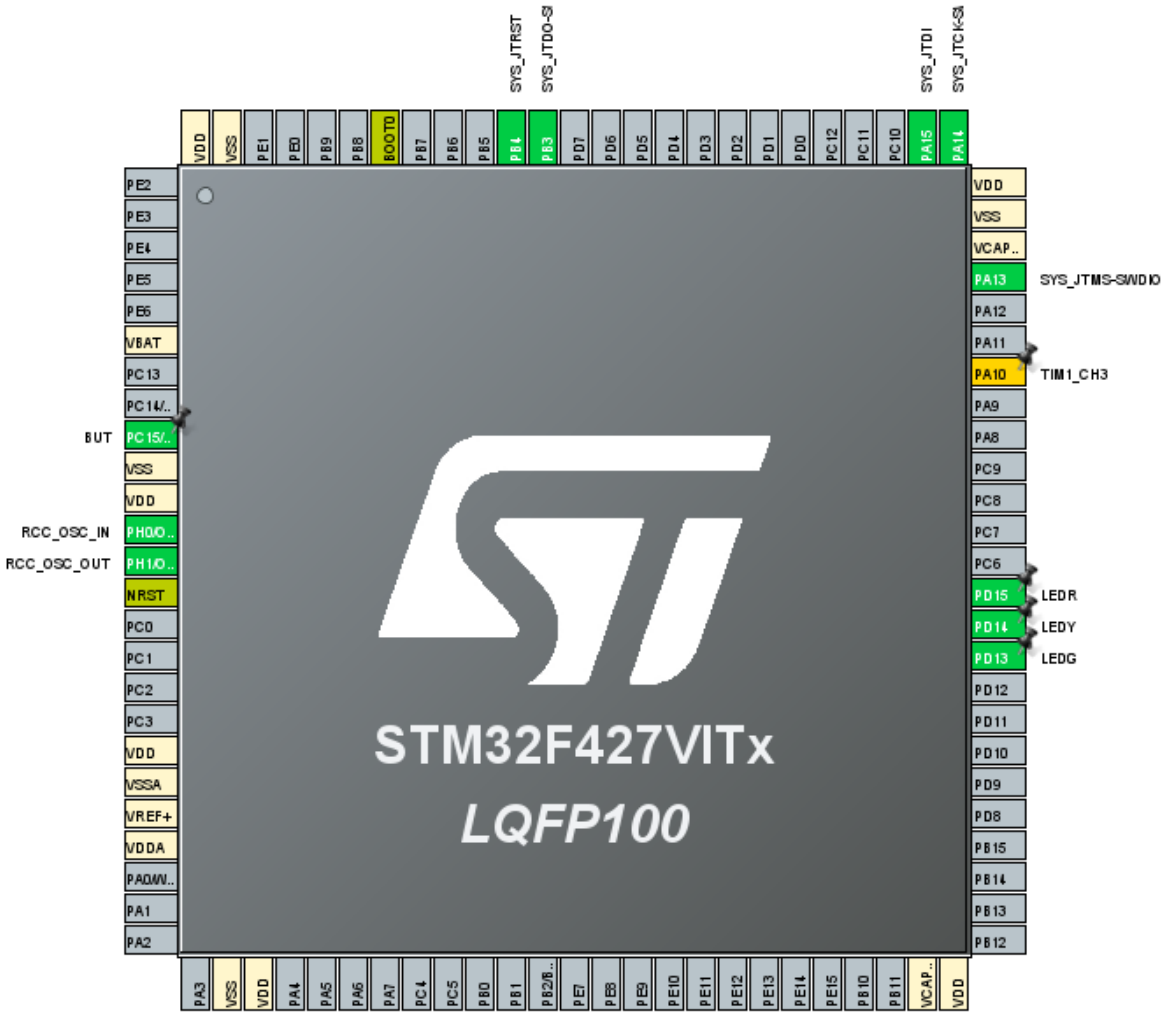
Задание

Разработать и реализовать драйверы управления светодиодными индикаторами и чтения состояния кнопки стенда SDK-1.1M (расположены на боковой панели стенда). Контакты подключения кнопки и светодиодов должны быть настроены в режиме GPIO. Функции и другие компоненты драйверов должны быть универсальными, т.е. пригодными для использования в любом из вариантов задания и не должны содержать прикладной логики программы. Функции драйверов должны быть неблокирующими, то есть не должны содержать ожиданий события (например, нажатия кнопки). Также, в драйверах не должно быть пауз с активным ожиданием функция HAL_Delay() и собственные варианты аналогичной реализации. Обработка нажатия кнопки в программе должна включать программную защиту от дребезга.

Написать программу с использованием разработанных драйверов в соответствии с вариантом задания.

Реализовать «передатчик» азбуки Морзе. Последовательность из нажатий кнопки (короткое – точка, длинное – тире) запоминается и после сигнала окончания ввода (долгая пауза) начинает «отправляться» при помощи зелѐного светодиода, последовательностью быстрых (точка) и долгих (тире) мерцаний. Во время ввода последовательности после каждого нажатия двухцветный светодиод должен коротким мерцанием индицировать, какой сигнал был введѐн (мигание жѐлтым – точка, мигание красным – тире).

Используемые контакты

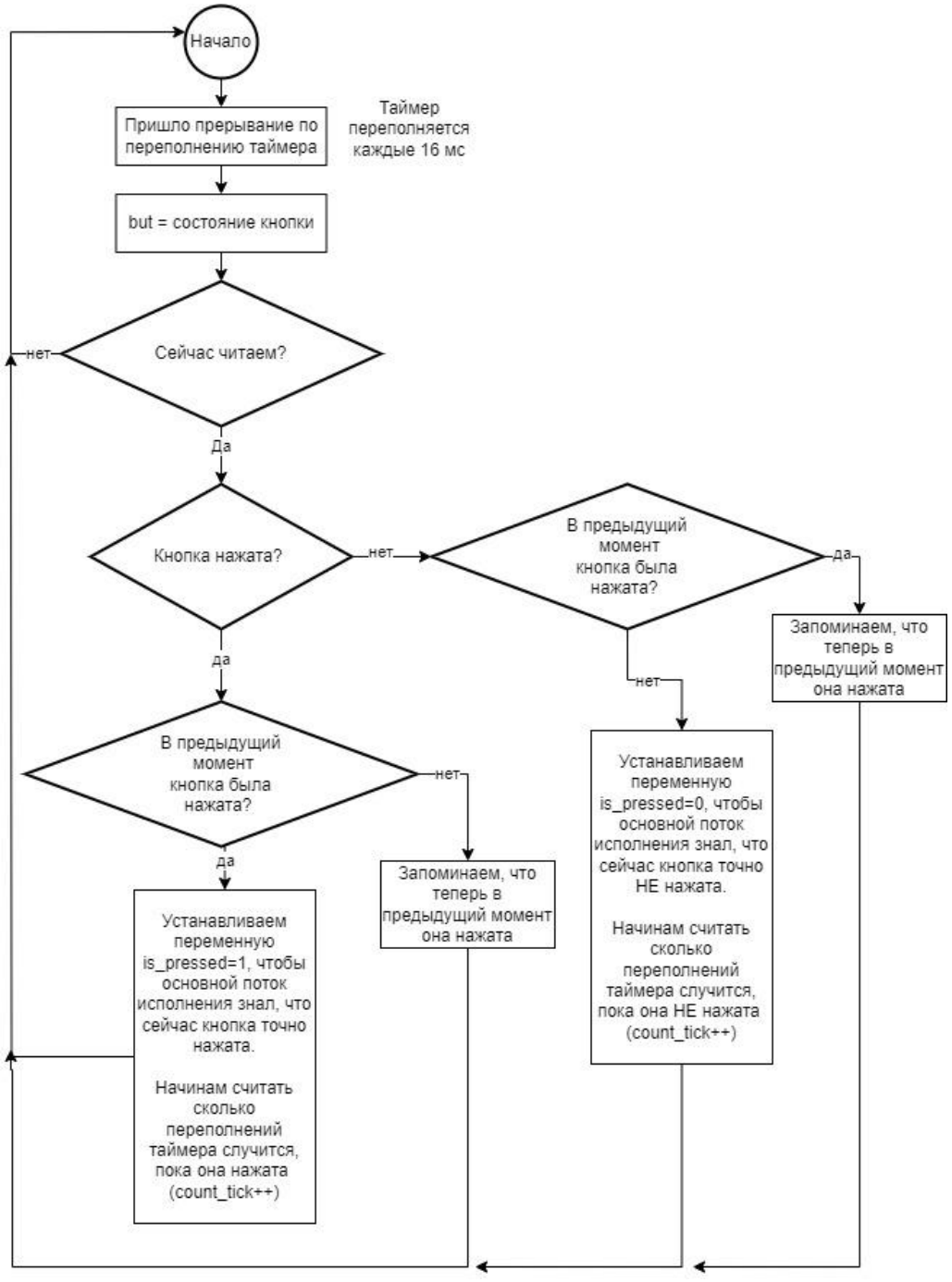


- PC15 – перехватывает нажатия кнопки (обозначен как BUT, настроен на GPIO_INPUT)
- PD13 – управляет зеленым светодиодом (обозначен как LEDG, настроен на GPIO_OUTPUT)
- PD14 - управляет желтым светодиодом (обозначен как LEDY, настроен на GPIO_OUTPUT)
- PD15 - управляет красным светодиодом (обозначен как LEDR, настроен на GPIO_OUTPUT)
- PA10 – получает прерывания от таймера (обозначен как TIM1_CH3)
- PB3, PB4, PA13, PA14, PA15 – J-Tag для отладки

Описание алгоритма

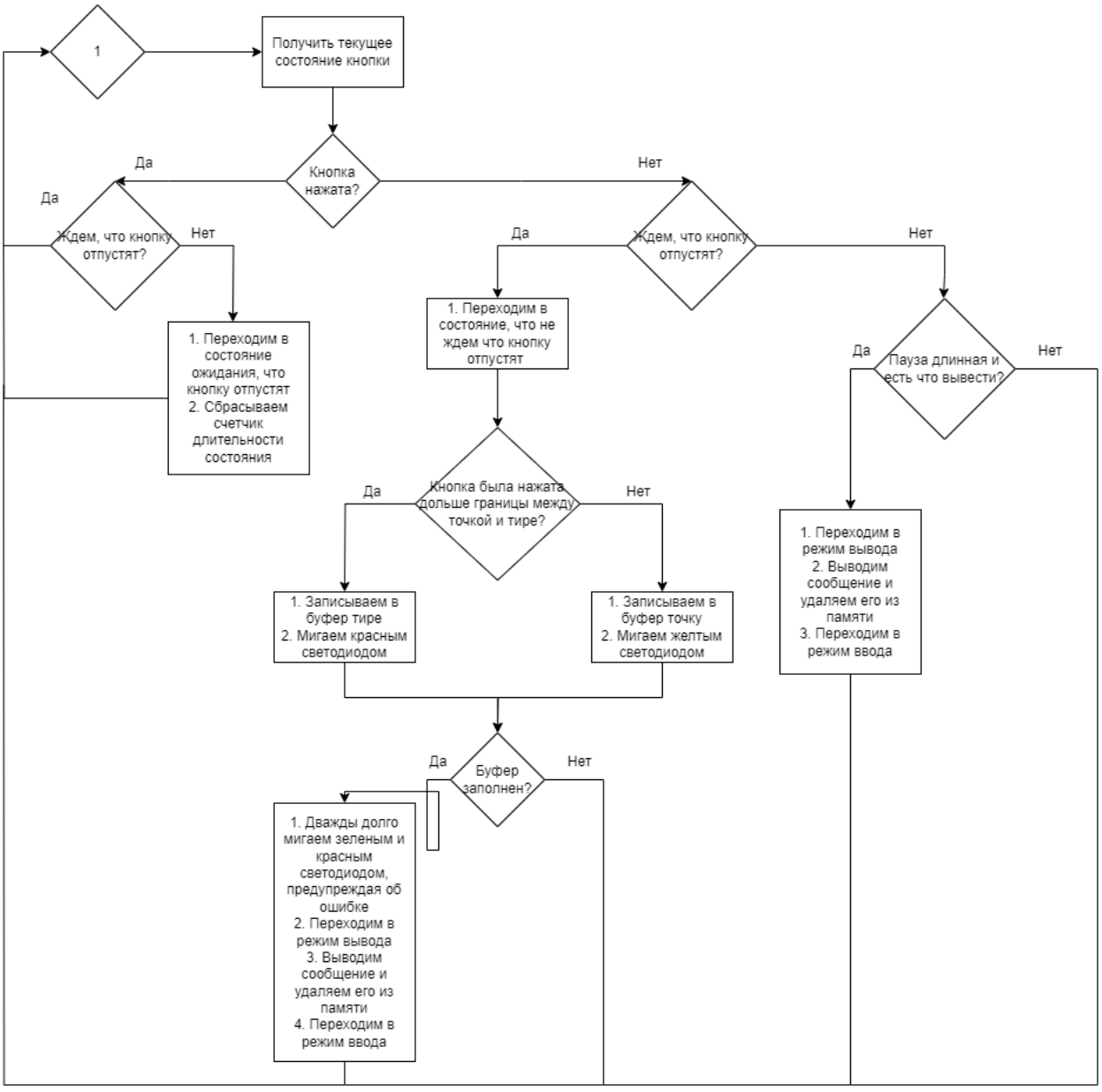
Блок-схема обработки прерывания таймера

Чтобы всё работало в неблокирующем режиме мы решили использовать встроенные таймеры. Мы выбрали 3 ий канал на первом таймере, под это зарезервирован ножку PA10. Этот канал мы настроили в режиме Output compare, no output. Таким образом на ножке ничего не отображается, но нам нужны были прерывания по этому таймеру. Поскольку на шине таймера частота была 16МГц мы настроили prescaler в 16000, а длину цикла таймера в 16. Включили прерывания по переполнению. Таким образом таймер будет срабатывать каждые 16 мс. А нам оно и надо, потому что если мы два цикла подряд зафиксируем, что кнопка нажата можно будет считать, что она точно была нажата, и мне зафиксировали не просто дребезг.



В обработчике прерываний нам нужно просто выставить правильное значение глобальных переменных. Действуя, как указано в блок схеме мы устанавливаем верное значение переменных is_pressed и count_tick. (Переменная count_tick обнуляется в основном потоке исполнения, потому что именно именно там у нас есть реальное понимание о том, когда кнопка становится отпущена или нажата).

Блок-схема основной логики



Исходный код

Драйвер вывода (светодиоды)

```
void turn_on_green_led() {
	HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_SET);
}

void turn_off_green_led() {
	HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_RESET);
}

void turn_on_yellow_led() {
	HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_SET);
```

```
}

void turn_off_yellow_led() {
	HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
}

void turn_on_red_led() {
	HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_SET);
}

void turn_off_red_led() {
	HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);
}
```

Драйвер ввода (кнопка, реализовано через опрос по прерыванию от таймера)

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
	if(htim->Instance==TIM1)
	{
		int but = HAL_GPIO_ReadPin(BUT_GPIO_Port, BUT_Pin);
		but = !but;

		if(is_reading == 1){
			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++;
			}
		}
	}

}
```

Основная логика

```
#define LONG_PERIOD_CT 125
#define BUFFER_SIZE 2048
#define DEFAULT_DELAY 200
#define ERROR_DELAY 600
#define ERROR_COUNT 2
#define COMMA_DELAY DEFAULT_DELAY
#define DASH_DELAY 1000



int is_pressed = 0; // нажата ли кнопка, устанавливается в таймере
int is_wait_unpressed = 0; // ожидаем ли отпускание кнопки
int is_reading = 1; // есть ли что в памяти чтобы читать (можно pointer сделать глобальным и проверять на
ноль
int count_tick = 0; // кол-во тиков таймера в текущем состоянии is_pressed, устанавливается в таймере
int noisy = 0; // доп проверка для дребезга



void send_message(int buffer[], int pointer) {
	for (int i = 0; i < pointer; i++) {
		turn_on_green_led();
		if (1 == buffer[i]) {
			HAL_Delay(DASH_DELAY);
		} else {
			HAL_Delay(COMMA_DELAY);
		}
		turn_off_green_led();
		HAL_Delay(DEFAULT_DELAY);
	}
}



/* USER CODE BEGIN WHILE */
```

```
int buffer[BUFFER_SIZE] = {0};
int pointer = 0;
HAL_TIM_Base_Start_IT((TIM_HandleTypeDef *)&htim1);
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
        printf("%i %i", is_pressed, is_wait_unpressed);
        if (0 == is_pressed && 0 == is_wait_unpressed) {
                if (count_tick > LONG_PERIOD_CT && 0 != pointer){
                        is_reading = 0;
                        send_message(buffer, pointer);
                        pointer = 0;
                        is_reading = 1;
                }
        } else if (0 == is_pressed && 1 == is_wait_unpressed) {
                is_wait_unpressed = 0;
                if (count_tick > LONG_PERIOD_CT) {
                        turn_on_red_led();
                        HAL_Delay(DEFAULT_DELAY);
                        turn_off_red_led();
                        buffer[pointer] = 1;
                } else {
                        turn_on_yellow_led();
                        HAL_Delay(DEFAULT_DELAY);
                        turn_off_yellow_led();
                        buffer[pointer] = 0;
                }
                count_tick = 0;
                pointer++;
                if (BUFFER_SIZE == pointer) {
                        for (int i = 0; i < ERROR_COUNT; i++) {
                                turn_on_red_led();
                                turn_on_green_led();
                                HAL_Delay(ERROR_DELAY);
                                turn_off_red_led();
                                turn_off_green_led();
                                HAL_Delay(ERROR_DELAY);
                        }
                        send_message(buffer, pointer);
                        pointer = 0;
                }
        } else if (1 == is_pressed && 0 == is_wait_unpressed) {
                is_wait_unpressed = 1;
                count_tick = 0;
        }
}
/* USER CODE END 3 */
```

Выводы

В ходе выполнения лабораторной работы мы вспомнили, как писать программы под микроконтроллеры семейства STM32, научились их дебажить и попробовали настроить таймер и реализовать логику неблокирующего чтения через опросы по прерыванию от него.