Факультет программной инженерии и компьютерной техники

Лабораторная работа №1

«Интерфейсы ввода-вывода общего назначения (GPIO)»

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 драйверы управления светодиодными индикаторами и чтения состояния кнопки стенда SDK-1.1M (расположены на боковой панели стенда). Обработка нажатия кнопки в программе должна включать программную защиту от дребезга. 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 с аналогичной функциональностью).

При необходимости параллельного выполнения разных процессов (например, опроса кнопки и ожидания перед переключением светодиодов) следует организовать псевдопараллельную работу процессов в стиле кооперативной многозадачности. Это возможно сделать без существенной потери точности соблюдения необходимых интервалов времени между действиями, поскольку действия выполняются очень быстро по сравнению с промежутками ожидания между ними. Каждый процесс (который может быть выражен всего в нескольких строках кода) должен использовать не активное ожидание (HAL_Delay()), а считывать текущее значение миллисекундного счетчика (HAL_GetTick()), проверяя, прошло ли необходимое количество времени для выполнения следующего действия. После проверки и (при необходимости) выполнения действия управление должно передаваться следующему процессу, чтобы он тоже мог провести такую проверку.

Контакты подключения кнопки и светодиодов должны быть настроены в режиме GPIO. Использование прерываний и дополнительных таймеров (кроме SysTick) не допускается.

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

Вариант 1:

Сымитировать работу светофора пешеходного перехода. Светофор циклически переключает цвета в следующем порядке (порядок условный, соответствие реальному светофору не соблюдается): красный, зелёный, зелёный мигающий, жёлтый, снова красный и т. д. По умолчанию период горения красного в четыре раза больше периода горения зеленого. Если во время горения зеленого мигающего, желтого или красного нажимается кнопка, светофор запоминает необходимость скорейшего переключения на зелёный. После нажатия кнопки последовательность переключения цветов не нарушается, но ближайший период горения красного должен быть сокращен до ¼ своего обычного периода. Если кнопка нажата во время горения красного, когда он уже горит более ¼ периода, то сразу происходит переключение на зеленый. Следующий красный должен снова гореть полную длительность, если только снова не будет нажата кнопка.

Описание организации программы и структуры драйверов

Светофор должен работать в постоянном режиме.

В момент нажатия кнопки длительность горения красного сигнала либо должна быть сокращена до ¼ времени горения зеленого сигнала, либо должно произойти мгновенное переключение на зеленый, если красный горел больше ¼ времени горения зеленого.

Для обозначения состояний светофора использовался *enum*, указанный в файле *traffic_light.h*:

```
typedef enum {
    STATE_RED,
    STATE_GREEN,
    STATE_GREEN_BLINKING,
    STATE_YELLOW
} traffic_light_state_t;
```

Для хранения информации о том, была ли нажата кнопка в момент горения красного, желтого или зеленого-мигающего использована переменная:

```
uint8_t next_red_short;
```

которая устанавливается в значение 0 или 1 в зависимости от наличия нажатия кнопки в определенный момент. Если значение кнопки равно единице, то время горения красного сигнала будет сокращено в четыре раза или же произойдет мгновенное переключение красного на зеленый.

Для драйвера *led_driver.h* использована следующая структура:

```
typedef struct {
    GPIO_TypeDef* port;
    uint16_t pin;
} led_t;
```

где port – порт, на котором находится светодиод, а pin – пин, к которому он подключен. Кроме того, этот драйвер содержит *enum*:

```
typedef enum {
    COLOR_RED,
    COLOR_YELLOW,
    COLOR_GREEN,
    COLOR_NONE
} led_color_t;
```

указывающий на цвет светодиода. Этот *enum* использован для установки требуемого цвета в файле *traffic_light.h* в функции:

```
void set_traffic_light(led_color_t color)
```

Драйвер кнопки driver_button.h содержит имеет следующую структуру:

```
typedef struct {
    GPIO_TypeDef* port;
    uint16_t pin;
    GPIO_PinState last_stable_state;
    GPIO_PinState last_raw_state;
    uint32_t last_change_time;
    uint8_t pressed_event_flag;
} button_t;
```

где

- port – порт, на котором находится светодиод,
- pin – пин, к которому он подключен,
- last_stable_state – последнее устойчивое состояние кнопки,
- las_raw_state – последнее полученное состояние кнопки,
- last_change_time – время, когда состояние кнопки менялось в последний раз,
- pressed_event_flag – флаг события нажатия на кнопку

Также, время подавления дребезга кнопки установлено в 40 мс (новое состояние считается «устойчивым» только если оно не менялось в течение 40 миллисекунд):

```
#define BUTTON_DEBOUNCE_MS   40u
```

Блок-схема прикладного алгоритма

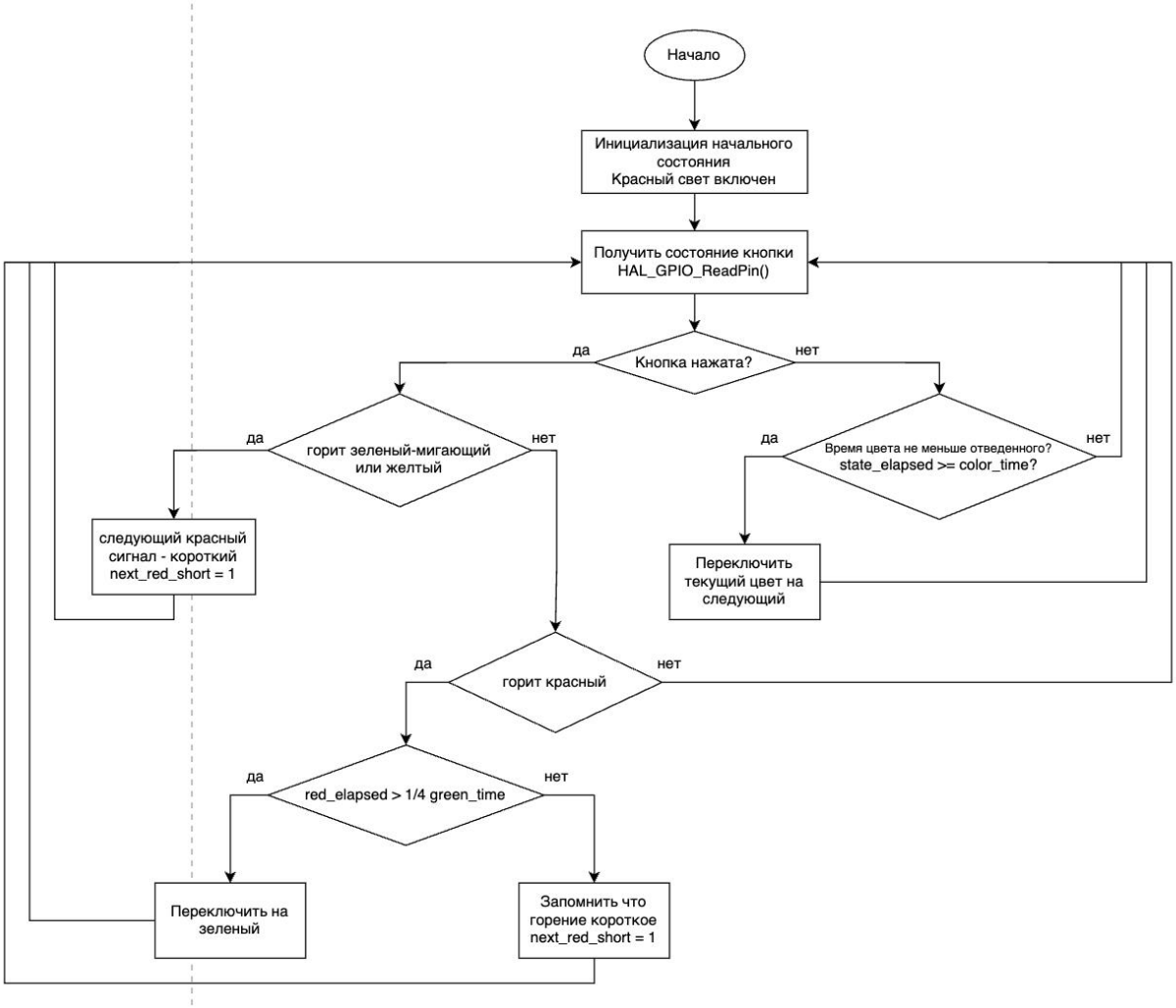

Рисунок 1. Блок-схема

Исходный код

В последующем разделе приведён разбор только исходных файлов с расширением *.c*. Заголовочные файлы *.h* не рассматриваются, поскольку описания используемых в них структур представлены выше.

led_driver.c:

```
#include "led_driver.h"

void led_init(led_t* led, GPIO_TypeDef* port, uint16_t pin) {
    led->port = port;
    led->pin = pin;
}

void led_on(led_t* led) {
    HAL_GPIO_WritePin(led->port, led->pin, GPIO_PIN_SET);
}

void led_off(led_t* led) {
    HAL_GPIO_WritePin(led->port, led->pin, GPIO_PIN_RESET);
}
```

button_driver.c:

```
#include "button_driver.h"

void button_init(button_t* btn, GPIO_TypeDef* port, uint16_t
pin) {
    btn->port = port;
    btn->pin = pin;
    btn->last_raw_state = HAL_GPIO_ReadPin(port, pin);
    btn->last_stable_state = btn->last_raw_state;
    btn->last_change_time = HAL_GetTick();
    btn->pressed_event_flag = 0;
}


void button_process(button_t* btn) {
     static uint32_t last_time = 0;
    GPIO_PinState raw = HAL_GPIO_ReadPin(btn->port, btn->pin);
    uint32_t now = HAL_GetTick();

    if (now - last_time < 10) return;
    last_time = now;

    if (raw != btn->last_raw_state) {
        btn->last_raw_state = raw;
        btn->last_change_time = now;
```

```
    } else {
        if (raw != btn->last_stable_state) {
            if ((now - btn->last_change_time) >=
BUTTON_DEBOUNCE_MS) {
                if (btn->last_stable_state == GPIO_PIN_SET &&
raw == GPIO_PIN_RESET) {
                    btn->pressed_event_flag = 1;
                }
                btn->last_stable_state = raw;
            }
        }
    }
}

uint8_t button_is_pressed(button_t* btn) {
    button_process(btn);

    if (btn->pressed_event_flag) {
        btn->pressed_event_flag = 0;
        return 1;
    }
    return 0;
}

GPIO_PinState button_get_state(button_t* btn) {
    button_process(btn);
    return btn->last_stable_state;
}
```

traffic_light.c:

```
#include "traffic_light.h"

#define GREEN_LED       GPIO_PIN_13
#define YELLOW_LED      GPIO_PIN_14
#define RED_LED         GPIO_PIN_15
#define BUTTON_PORT     GPIOC
#define LED_PORT        GPIOD

led_t green_led, yellow_led, red_led;
button_t pedestrian_button;
traffic_light_state_t current_state = STATE_RED;
uint32_t state_start_time = 0;
uint8_t next_red_short = 0;


void traffic_light_init(void) {
    led_init(&green_led, LED_PORT, GREEN_LED);
    led_init(&yellow_led, LED_PORT, YELLOW_LED);
    led_init(&red_led, LED_PORT, RED_LED);
    button_init(&pedestrian_button, BUTTON_PORT, GPIO_PIN_15);
```

```
    current_state = STATE_RED;
    state_start_time = HAL_GetTick();
    set_traffic_light(COLOR_RED);
}

void set_traffic_light(led_color_t color) {
    led_off(&red_led);
    led_off(&yellow_led);
    led_off(&green_led);

    switch(color) {
        case COLOR_RED:
            led_on(&red_led);
            break;
        case COLOR_YELLOW:
            led_on(&yellow_led);
            break;
        case COLOR_GREEN:
            led_on(&green_led);
            break;
        case COLOR_NONE:
            break;
    }
}

void traffic_light_handler(void) {
     static uint32_t last_time = 0;
     uint32_t current_time = HAL_GetTick();
     uint32_t state_elapsed = current_time - state_start_time;
     uint32_t red_time = next_red_short ? RED_TIME_SHORT_MS :
RED_TIME_FULL_MS;

     if (current_time - last_time < 10) return;
     last_time = current_time;

     switch (current_state) {

          case STATE_RED:
               set_traffic_light(COLOR_RED);

               if (state_elapsed >= red_time) {
                    current_state = STATE_GREEN;
                    state_start_time = current_time;
                    next_red_short = 0;
               }
               break;

          case STATE_GREEN:
               set_traffic_light(COLOR_GREEN);

               if (state_elapsed >= GREEN_TIME_MS) {
```

```
                    current_state = STATE_GREEN_BLINKING;
                    state_start_time = current_time;
                }
                break;

            case STATE_GREEN_BLINKING:
                if (state_elapsed % 1000 < 500) {
                    set_traffic_light(COLOR_NONE);
                } else {
                    set_traffic_light(COLOR_GREEN);
                }

                if (state_elapsed >= GREEN_BLINK_TIME_MS) {
                    current_state = STATE_YELLOW;
                    state_start_time = current_time;
                }
                break;

            case STATE_YELLOW:
                set_traffic_light(COLOR_YELLOW);

                if (state_elapsed >= YELLOW_TIME_MS) {
                    current_state = STATE_RED;
                    state_start_time = current_time;
                }
                break;
        }
}


void handle_button_press(void) {
    if (current_state == STATE_RED) {
        uint32_t red_elapsed = HAL_GetTick() - state_start_time;

        if (red_elapsed >= RED_TIME_SHORT_MS) {
            current_state = STATE_GREEN;
            state_start_time = HAL_GetTick();
            next_red_short = 0;
        } else {
            next_red_short = 1;
        }
    } else if (current_state == STATE_YELLOW || current_state ==
STATE_GREEN_BLINKING) {
        next_red_short = 1;
    }
}
```

main.c (без комментарией):

```
#include "main.h"
#include "traffic_light.h"
```

```
void SystemClock_Config(void);
static void MX_GPIO_Init(void);

int main(void)
{
  HAL_Init();
  traffic_light_init();
  SystemClock_Config();
  MX_GPIO_Init();

  while (1)
  {
       traffic_light_handler();
       button_process(&pedestrian_button);

       if (button_is_pressed(&pedestrian_button)) {
            handle_button_press();
       }
  }
}
```

С полной версией программы можно ознакомиться, перейдя на GitHub-репозиторий, расположенный по ссылке:

https://github.com/kkettch/design-of-computing-systems-semester-7

Вывод

В ходе выполнения данной лабораторной работы было выполнено написание кода в STM32CubeIDE на языке программирования C для разработки «светофора с кнопкой для пешеходов» на микроконтроллере STM32F427VIT6. Для этого были реализованы драйверы для управления логикой светодиодов и кнопкой. Были изучены интерфейсы ввода-вывода общего назначения (GPIO).