Федеральное государственное автономное образовательное учреждение высшего образования

«**Санкт-Петербургский национальный исследовательский**

**университет информационных технологий, механики и оптики»**

**Факультет ПИиКТ**

**Дисциплина: Проектирование вычислительных систем**

**Лабораторная работа № 1**

**Интерфейсы ввода/вывода общего**

**назначения (GPIO)**

Выполнили: Камышанская Ксения Васильевна, Гурин Евгений Иванович

Преподаватель: Пинкевич Василий Юрьевич

Группа: Р34122

Вариант: 2

Санкт-Петербург 2021г.

**Задание**

Разработать и реализовать драйверы управления светодиодными индикаторами и обработки нажатий кнопки стенда SDK-1.1М (индикаторы и кнопка расположены на боковой панели стенда). Написать программу с использованием разработанных драйверов в соответствии с вариантом задания.

**Задание варианта**  
Реализовать простой имитатор гирлянды с переключением режимов. Должно быть реализовано не менее четырех последовательностей переключения светодиодов, обязательно с разной частотой мигания. По нажатию кнопки происходит переключение на следующий режим. Если режим последний в списке, нажатие кнопки должно переключать на первый режим. При повторном выборе режима анимация на светодиодах должна запускаться с того места, на котором была прервана переключением на следующий режим.

**Описание организации программы**Для решения задачи создания имитации гирлянды, которая при повторном выборе режима анимации должна восстанавливаться с того же момента, на котором остановилась мы применили несколько структур.   
  
Во первых мы выделили структуру **состояния режима** – это конкретных набор параметров включения светодиодов и время, на которое этот режим должен быть включен.

**Graphical user interface, text

Description automatically generated with medium confidence**Во вторых была выделена структура **режим** – это указатель на массив из состояний и его длина  
Text

Description automatically generated with low confidence

В процессе работы одного режима состояния переключаются по кругу. В процессе работы программы в бесконечном цикле опрашивается кнопка. В каждой итерации цикла сохраняется состояние текущего режима в массиве сохраненных состояний режима. При нажатии кнопки проиходит переключение на следующий режим по кругу (после последнего идет самый первый). В процессе этого переключения восстанавливается время старта режима для правильного переключения его в следующее состояние. Далее применяется состояние активного режима (активация светодиодов).

**Блок-схема прикладного алгоритма**

Diagram

Description automatically generated

**Описание инструментария**Для написания кода использовалась STM32CubeIDE, часть кода для работы со стендом была сгенерирована и в main файле мы написала часть, которая отвечает за логику программы переключения режимов.

**Исходный код**

**#define** BUTTON\_PIN GPIO\_PIN\_15

**#define** RED\_PIN GPIO\_PIN\_15

**#define** YELLOW\_PIN GPIO\_PIN\_14

**#define** GREEN\_PIN GPIO\_PIN\_13

**#define** LOOP\_DELAY 100

**#define** BUTTON\_DELAY 5

**#define** BUTTON\_INACTIVE\_DELAY 2000

**struct** ModeState {

uint32\_t R;

uint32\_t Y;

uint32\_t G;

uint32\_t delay;

};

**struct** Mode {

**struct** ModeState \* states;

uint32\_t length;

};

**struct** State {

uint32\_t exceeded;

uint32\_t n\_mode\_state;

};

**struct** ModeState states\_1[] = {

{

.R = 0,

.Y = 1,

.G = 0,

.delay = 1000

},

{

.R = 1,

.Y = 0,

.G = 0,

.delay = 1000

}

};

**const** **struct** Mode mode\_1 = {

.states = states\_1,

.length = 2

};

**struct** ModeState states\_2[] = {

{

.R = 1,

.Y = 0,

.G = 1,

.delay = 1000

},

{

.R = 0,

.Y = 1,

.G = 1,

.delay = 1000

}

};

**const** **struct** Mode mode\_2 = {

.states = states\_2,

.length = 2

};

**struct** ModeState states\_3[] = {

{

.R = 0,

.Y = 0,

.G = 1,

.delay = 1000

},

{

.R = 0,

.Y = 1,

.G = 0,

.delay = 1000

},

{

.R = 0,

.Y = 0,

.G = 1,

.delay = 1000

},

{

.R = 1,

.Y = 0,

.G = 0,

.delay = 1000

},

};

**const** **struct** Mode mode\_3 = {

.states = states\_3,

.length = 4

};

**struct** ModeState states\_4[] = {

{

.R = 1,

.Y = 0,

.G = 1,

.delay = 500

},

{

.R = 0,

.Y = 0,

.G = 0,

.delay = 300

},

{

.R = 0,

.Y = 1,

.G = 1,

.delay = 500

},

{

.R = 0,

.Y = 0,

.G = 0,

.delay = 300

},

{

.R = 1,

.Y = 0,

.G = 0,

.delay = 500

},

{

.R = 0,

.Y = 0,

.G = 0,

.delay = 300

}

};

**const** **struct** Mode mode\_4 = {

.states = states\_4,

.length = 6

};

**#define** N\_MODES 4

**const** **struct** Mode MODES[] = { mode\_1, mode\_2, mode\_3, mode\_4 };

**struct** State saved\_states[] = { {0, 0}, {0, 0}, {0, 0}, {0, 0} };

uint32\_t exceeded\_time = 0;

uint32\_t mode\_start\_time = 0;

uint32\_t button\_click\_time = 0;

uint32\_t n\_mode = 0;

uint32\_t initialized = 0;

**void** **set\_light**(uint16\_t pin, **int** value) {

HAL\_GPIO\_WritePin(GPIOD, pin, value > 0 ? *GPIO\_PIN\_SET* : *GPIO\_PIN\_RESET*);

}

**void** **set\_red**(**int** value) {

set\_light(RED\_PIN, value);

}

**void** **set\_green**(**int** value) {

set\_light(GREEN\_PIN, value);

}

**void** **set\_yellow**(**int** value) {

set\_light(YELLOW\_PIN, value);

}

**void** **save\_state**() {

exceeded\_time = HAL\_GetTick();

saved\_states[n\_mode].exceeded = exceeded\_time - mode\_start\_time;

}

**void** **next\_mode**() {

n\_mode = (n\_mode + 1) % N\_MODES;

mode\_start\_time = exceeded\_time - saved\_states[n\_mode].exceeded;

}

**void** **apply\_state**() {

**struct** ModeState state = MODES[n\_mode].states[saved\_states[n\_mode].n\_mode\_state];

set\_red(state.R);

set\_green(state.G);

set\_yellow(state.Y);

}

**void** **apply\_mode**() {

uint32\_t n\_state = saved\_states[n\_mode].n\_mode\_state;

**if** (saved\_states[n\_mode].exceeded > MODES[n\_mode].states[n\_state].delay) {

saved\_states[n\_mode].n\_mode\_state = (saved\_states[n\_mode].n\_mode\_state + 1) % MODES[n\_mode].length;

saved\_states[n\_mode].exceeded = 0;

mode\_start\_time = exceeded\_time;

apply\_state();

}

}

**void** **apply\_button**() {

GPIO\_PinState button\_state = HAL\_GPIO\_ReadPin(GPIOC, BUTTON\_PIN);

**if** (button\_state == *GPIO\_PIN\_SET*) **return**;

HAL\_Delay(BUTTON\_DELAY);

button\_state = HAL\_GPIO\_ReadPin(GPIOC, BUTTON\_PIN);

**if** (button\_state == *GPIO\_PIN\_SET*) **return**;

**if** ((HAL\_GetTick() - button\_click\_time) < BUTTON\_INACTIVE\_DELAY) **return**;

button\_click\_time = HAL\_GetTick();

next\_mode();

}

**void** **prepare\_default\_states**() {

**if** (initialized > 0) **return**;

exceeded\_time = HAL\_GetTick();

mode\_start\_time = HAL\_GetTick();

apply\_state();

initialized = 1;

}

**int** **main**(**void**)

{

HAL\_Init();

prepare\_default\_states();

SystemClock\_Config();

MX\_GPIO\_Init();

**while** (1)

{ prepare\_default\_states();

save\_state();

apply\_button();

apply\_mode();

HAL\_Delay(LOOP\_DELAY);

}

}

**Вывод**

В процессе выполнения лабораторной работы мы придумали решение для проблемы сохранения состояния каждого режима при его переключении. Для этого при каждом изменении мы сохраняли состояние активного режима в массиве сохраненных состояний для каждого режима. Помимо этого столкнулись с проблемой одновременного опроса кнопки и работы гирлянды. Её мы решили с помощью получения активного времени и отсчета его с момента включения режима, при циклическом опросе кнопки – применении режима работы.