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: P34122

Вариант: 2

Санкт-Петербург 2021г.

Задание

Разработать и реализовать драйверы управления светодиодными индикаторами и обработки нажатий кнопки стенда SDK-1.1M (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.

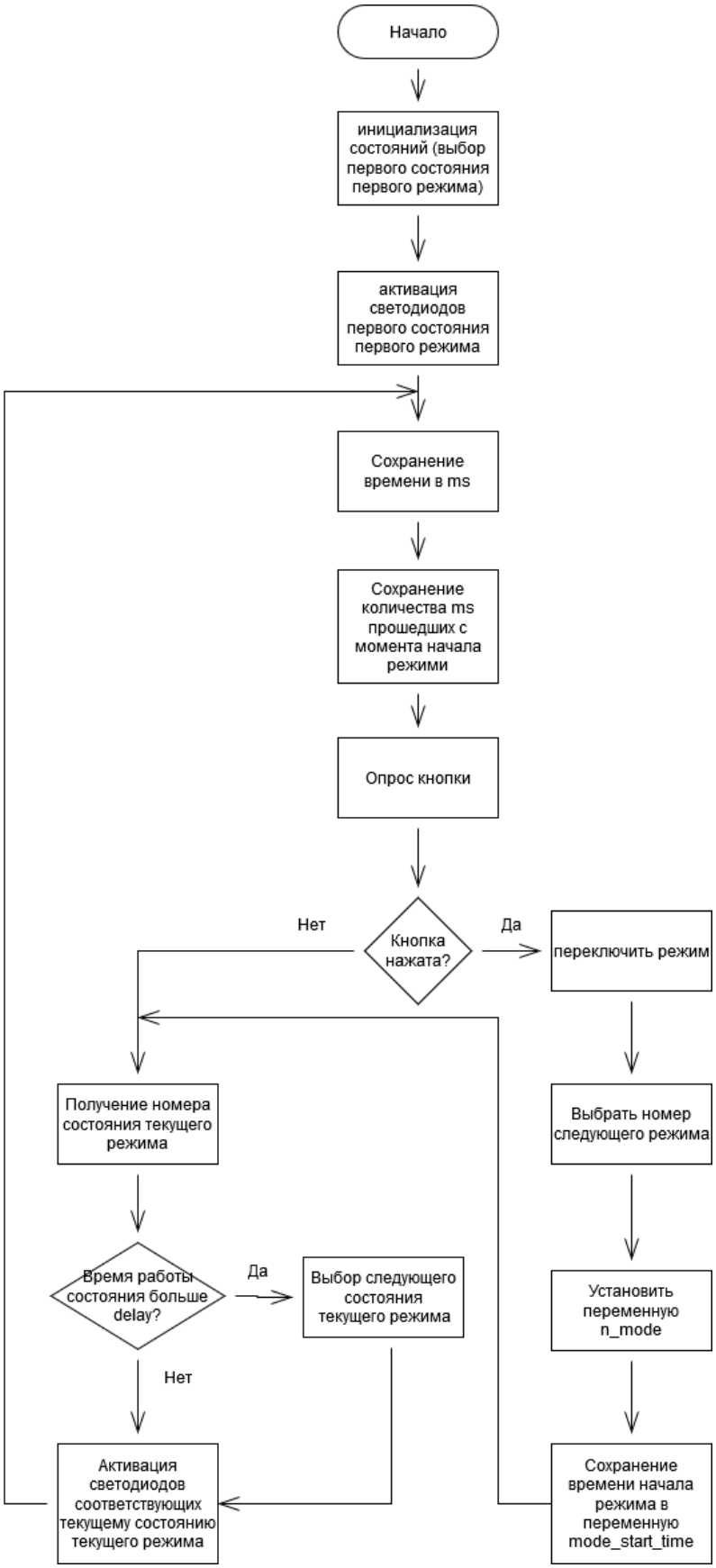
```
struct ModeState {
    int R;
    int Y;
    int G;
    int delay;
};
```

Во вторых была выделена структура **режим** – это указатель на массив из состояний и его длина

```
struct Mode {
    struct ModeState * states;
    int length;
};
```

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



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;
}
```

```c
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.