Университет ИТМО

Кафедра Вычислительной Техники

Лабораторная работа №1
по дисциплине “Встроенные системы”

Группа P3401

Комаров Егор Андреевич

Вариант 1

Оценка:_____

Принял: Ключев Аркадий Олегович

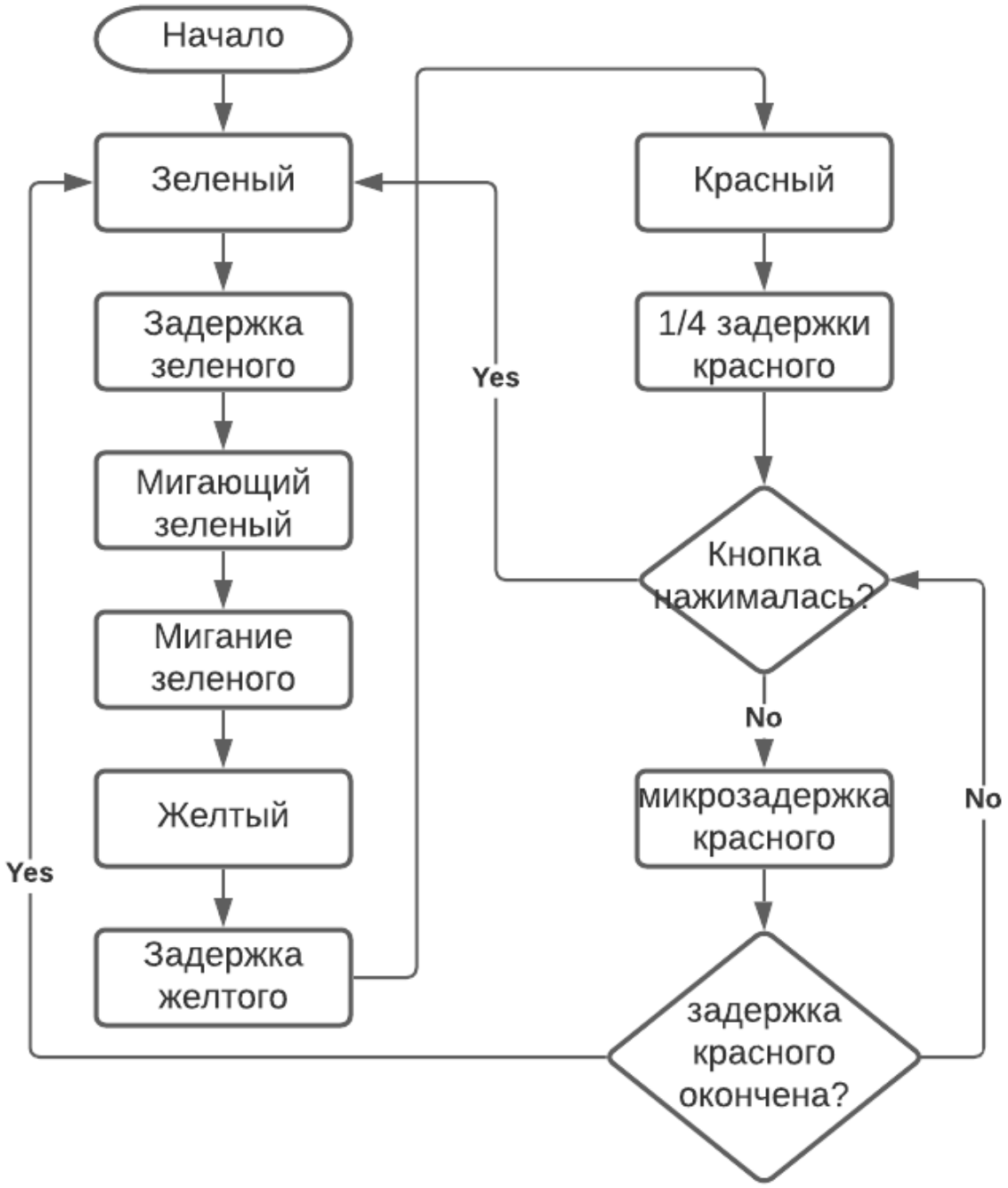
Санкт-Петербург, 2020

Задание

Разработать и реализовать драйверы GPIO (управление светодиодными индикаторами и обработка нажатий кнопки контроллера SDK-1.1M).

Сымитировать работу светофора пешеходного перехода. В режиме по умолчанию светофор переключает цвета в следующем режиме: зелёный-мигающий зелёный-жёлтый-красный-зелёный…, при этом период красного значительно больше. При нажатии кнопки происходит переключение с красного на зелёный, но два включения «зелёных» не могут идти сразу друг за другом – между ними должен быть период, больше или равный ¼ периода красного.

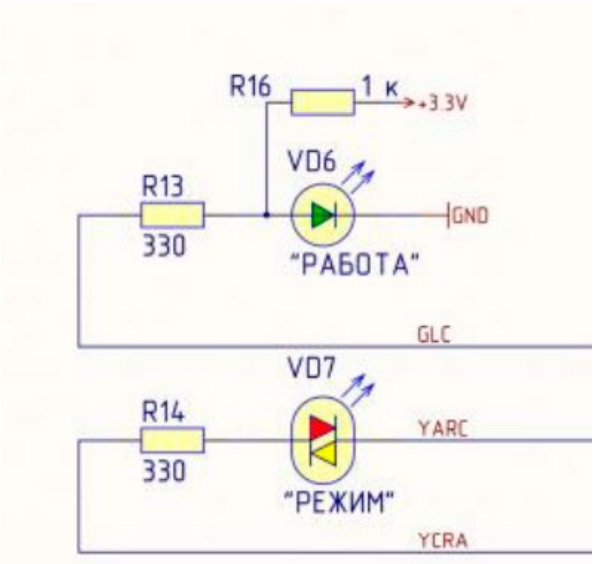
Блок-схема



Инструментарий

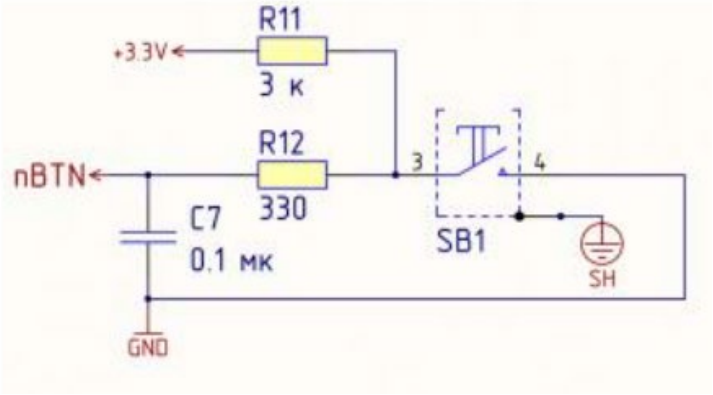
Аппаратные средства

Светодиоды



2 светодиода, которые абстрактно можно рассматривать как 3 из-за двухцветности второго, подключенные к контактам PD13 для зеленого, PD14 для желтого и PD15 для красного.

Кнопка



Одна кнопка, подключенная к PC15. В разомкнутом состоянии напряжение положительное.

Интерфейс GPIO

Представлен в виде набора контактов, к которым подключены описанные выше устройства

Программные средства

Управление светодиодами и считывание состояния кнопки

Использовались функции библиотеки HAL **HAL_GPIO_ReadPin**, **HAL_GPIO_WritePin** и **HAL_GPIO_TogglePin**. Они позволяют по id порта и конкретному пину считать его текущее состояние, либо его изменить. Всего 2 возможных значения - **GPIO_PIN_SET** и **GPIO_PIN_RESET**

Имитация задержки

Использовалась функция **HAL_Delay**, принимающая значение в миллисекундах и работающая на основе системного счетчика. А также **HAL_GetTick** для прямого доступа к счетчику с целью реализации собственной функции задержки

Инициализация GPIO

Для инициализации использовалась функция **HAL_GPIO_Init**, принимающая тип порта и структуру с параметрами, включающими используемые пины, их режим (input, output etc), режим активации (pull up/pull down/no pull) для input, и скорость переключения выходного сигнала

Исходный код

SDK

В связи с тем, что выполнение лабораторных работ проходило на удаленном устройстве с множеством ограничений по комплектации и крайне затрудненной отладкой, в первую очередь была написана небольшая библиотека, позволяющая упростить работу с базовыми сущностями, в особенности, специфичными для данного оборудования.

LED

На лабораторном стенде доступны всего 3 лампочки трех цветов на порте GPIOD с ограниченным функционалом (только включение/выключение), поэтому для удобства добавлена возможность оперировать только цветами и базовыми переключателями.

SDK/interface.h

```
// LED output
#define SDK_LED_GPIO GPIOD
#define SDK_LED_ON GPIO_PIN_SET
#define SDK_LED_OFF GPIO_PIN_RESET

#define SDK_LED1 LED1
#define SDK_LED2 LED2
#define SDK_LED3 LED3
#define SDK_LED_GREEN LED1
#define SDK_LED_YELLOW LED2
#define SDK_LED_RED LED3

#define SDK_LED_GREEN_PIN GPIO_PIN_13
#define SDK_LED_YELLOW_PIN GPIO_PIN_14
#define SDK_LED_RED_PIN GPIO_PIN_15

// led API
```

```
void SDK_LED_Set(uint16_t led, GPIO_PinState state);
void SDK_LED_Toggle(uint16_t led);
```

Каждое обращение к LED включает трассировку для возможности последующей визуализации при загрузке (если установлен глобальный define **SDK_REMOTE_MODE**)

SDK/sys.c

```
// led API
static uint16_t LED_MapLedToPin(uint16_t led)
{
	switch(led)
	{
	case SDK_LED_GREEN:
		return SDK_LED_GREEN_PIN;
	case SDK_LED_YELLOW:
		return SDK_LED_YELLOW_PIN;
	case SDK_LED_RED:
		return SDK_LED_RED_PIN;
	default:
		return 0;
	};
}
void SDK_LED_Set(uint16_t led, GPIO_PinState state)
{
	uint16_t pin = LED_MapLedToPin(led);

	HAL_GPIO_WritePin(SDK_LED_GPIO, pin, state);
#if SDK_REMOTE_MODE
	SDK_TRACE_Timestamp (led, HAL_GPIO_ReadPin(SDK_LED_GPIO, pin));
#endif
}
void SDK_LED_Toggle(uint16_t led)
{
	uint16_t pin = LED_MapLedToPin(led);

	HAL_GPIO_TogglePin(SDK_LED_GPIO, pin);
#if SDK_REMOTE_MODE
	SDK_TRACE_Timestamp (led, HAL_GPIO_ReadPin(SDK_LED_GPIO, pin));
#endif
}
```

Button

Стенд обладает лишь одной кнопкой, и всё, что нам нужно знать о ней – нажата ли она сейчас и не произошло ли только что отпускание (m_up) или нажатие (m_down). Так как последние два события представляют наибольший интерес, они были добавлены в финальную трассировку с особым идентификатором event типа - **P0** (**SDK_BTN**).

Для имитации кнопки на тестовом стенде была добавлена функция **SDK_BTN_SetDown**, представляющая абстрактное нажатие, например через прерывание по таймеру.

SDK/interface.h

```
// Button input
#define SDK_BTN_GPIO GPIOC
#define SDK_BTN P0
#define SDK_BTN_PIN GPIO_PIN_15
```

```
// button API
void SDK_BTN_ClearState();
void SDK_BTN_Update();
void SDK_BTN_SetDown();

bool SDK_BTN_IsPressed();
bool SDK_BTN_IsUp();
bool SDK_BTN_IsDown();
```

SDK/btn.c

```
/// TYPES ///
typedef struct BtnState
{
	bool m_pressed;
	bool m_down;
	bool m_up;
} BtnState;

/// STATIC ///
static BtnState s_btnState;

///  API  ///
void SDK_BTN_ClearState()
{
	s_btnState.m_pressed = false;
	s_btnState.m_up = false;
	s_btnState.m_down = false;
}
void SDK_BTN_Update()
{
	bool btn = HAL_GPIO_ReadPin(SDK_BTN_GPIO, SDK_BTN_PIN) == GPIO_PIN_RESET;
	s_btnState.m_up = s_btnState.m_pressed && !btn;
	s_btnState.m_down = !s_btnState.m_pressed && btn;
	s_btnState.m_pressed = btn;

#if SDK_REMOTE_MODE
	if(s_btnState.m_down || s_btnState.m_up)
	{
		SDK_TRACE_Timestamp(SDK_BTN, btn);
	}
#endif
}
bool SDK_BTN_IsPressed()
{
	return s_btnState.m_pressed;
}
bool SDK_BTN_IsUp()
{
	return s_btnState.m_up;
}
bool SDK_BTN_IsDown()
{
	return s_btnState.m_down;
}
void SDK_BTN_SetDown()
{
	s_btnState.m_pressed = true;
	s_btnState.m_up = false;
	s_btnState.m_down = true;
```

```
#if SDK_REMOTE_MODE
	SDK_TRACE_Timestamp(SDK_BTN, true);
#endif
}
```

Timer

В первую очередь необходимость в простейшем таймере в данной лабораторной работе вызвана потребностью в имитации нажатия на кнопку в определенный момент времени.

Реализованное API позволяет задать одно разовое или периодичное событие, вызывающее прерывание, без аргументов с заданными временными параметрами с помощью функции **SDK_TIM_SetInterrupt.**

Также для различных прикладных задач добавлены три вида задержек – обычная (**SDK_TIM_Delay**), прерываемая в указанных пределах (**SDK_TIM_InterruptDelay**), и задержка до выполнения какого-либо условия с таймаутом (**SDK_TIM_WaitEvent**)

SDK/interface.h

```
// timer
void SDK_TIM_Update();
void SDK_TIM_SetInterrupt(void(*callbackPtr)(), uint32_t delay, bool periodic);
void SDK_TIM_Delay(uint32_t delay);
void SDK_TIM_InterruptDelay(uint32_t minDelay, uint32_t maxDelay);
uint32_t SDK_TIM_WaitEvent(bool (*event)(), uint32_t timeout);
```

SDK/timer.c

```
/// TYPES ///
typedef struct Interrupt
{
	void (*m_callbackPtr)();
	uint32_t m_delay;
	uint32_t m_timer;
	bool m_periodic;
} Interrupt;

/// STATIC ///
static Interrupt s_interrupt;

/// API ///
void SDK_TIM_SetInterrupt(void(*callbackPtr)(), uint32_t delay, bool periodic)
{
	s_interrupt.m_callbackPtr = callbackPtr;
	s_interrupt.m_delay = delay;
	s_interrupt.m_timer = 0;
	s_interrupt.m_periodic = periodic;
}

void SDK_TIM_Update()
{
	if(s_interrupt.m_callbackPtr &&  ++s_interrupt.m_timer >= s_interrupt.m_delay)
	{
		s_interrupt.m_timer = 0;
		s_interrupt.m_callbackPtr();
		if(!s_interrupt.m_periodic)
		{
			s_interrupt.m_callbackPtr = 0;
```

```
		}
	}
}

void SDK_TIM_Delay(uint32_t delay)
{
	HAL_Delay(delay);
}

void SDK_TIM_InterruptDelay(uint32_t minDelay, uint32_t maxDelay)
{
	uint32_t passed = SDK_TIM_WaitEvent(SDK_BTN_IsDown, minDelay);
	if(passed)
	{
		SDK_TIM_Delay(minDelay - passed);
	}
	else
	{
		SDK_TIM_WaitEvent(SDK_BTN_IsDown, maxDelay - minDelay);
	}
}

uint32_t SDK_TIM_WaitEvent(bool (*event)(), uint32_t timeout)
{
	uint32_t tickstart = HAL_GetTick();

	// +1 to be sure that return positive if event() is true
	uint32_t wait = timeout + 1;
	uint32_t passed = 1;

	while( passed < wait)
	{
		if(event())
		{
			return passed;
		}
		passed = HAL_GetTick() - tickstart;
	}

	return 0;
}
```

System

Системный модуль отвечает за инициализацию и высвобождение различных частей приложения, а также их обновление при наличии необходимости каждую миллисекунду (через **SDK_SYS_Tick**)

SDK/interface.h

```
// system API
void SDK_SYS_Init();
void SDK_SYS_Shutdown();
void SDK_SYS_Tick(); // process every ms
```

SDK/sys.c

```
// system API
void SDK_SYS_Init()
{
```

```
#if SDK_REMOTE_MODE
	MX_TRACE_Init();
	SDK_TRACE_Start();
#endif
	SDK_BTN_ClearState();
}
void SDK_SYS_Shutdown()
{
#if SDK_REMOTE_MODE
	SDK_TRACE_Stop();
#endif
}
void SDK_SYS_Tick()
{
	SDK_BTN_Update();
	SDK_TIM_Update();
}
```

Функционал модуля дебага позволяет выводить текстовые и бинарные сообщения с их автоматической трассировкой

Debug

SDK/interface.h

```
// define 0 if local launch
#define SDK_REMOTE_MODE 1

// debug API
void SDK_DBG_Print(char * format, ...);
void SDK_DBG_Dump(uint32_t addr, uint16_t size);
```

SDK/sys.c

```
// debug API
void SDK_DBG_Print(char * format, ...)
{
#if SDK_REMOTE_MODE
	va_list args;
	va_start(args, format);

	SDK_TRACE_Timestamp(PRINT, 1);
	SDK_TRACE_VPrint(format, args);
	SDK_TRACE_Timestamp(PRINT, 0);
#endif
}
void SDK_DBG_Dump(uint32_t addr, uint16_t size)
{
#if SDK_REMOTE_MODE
	SDK_TRACE_Timestamp(DUMP, 1);
	SDK_TRACE_Dump(addr, size);
	SDK_TRACE_Timestamp(DUMP, 0);
#endif
}
```

Main

Для упрощения всё выполнение программы обернуто в аналогичную main функцию, позволяющую имитировать бесконечный цикл на тестовом стенде с задаваемым количеством итераций.

Требуется реализация функций **SDK_MAIN_PreLoop SDK_MAIN_PostLoop SDK_MAIN_LoopFunc** для каждого отдельного приложения.

SDK/interface.h

```
#define SDK_MAIN_LOOP_REPEATS 3

// main cycle wrapper
void SDK_MAIN_Wrapper();
void SDK_MAIN_Loop();

void SDK_MAIN_PreLoop();
void SDK_MAIN_PostLoop();
void SDK_MAIN_LoopFunc();
```

Переопределяемые функции идут в строгом порядке, причем **SDK_MAIN_LoopFunc** может выполняться множество раз или бесконечно в условиях, отличных от удаленного доступа.

SDK/main_wrapper.c

```
void SDK_MAIN_Wrapper()
{
	SDK_SYS_Init();

	// must be overriden by app
	SDK_MAIN_PreLoop();
	SDK_MAIN_Loop();
	SDK_MAIN_PostLoop();

	SDK_SYS_Shutdown();
}

void SDK_MAIN_Loop()
{
#if !SDK_REMOTE_MODE
	while(true)
#else
	for(int i = 0; i < SDK_MAIN_LOOP_REPEATS; i++)
#endif
	{
		SDK_MAIN_LoopFunc();
	}
}
```

Application

Semaphore

Весь функционал приложения из задания можно разделить на потенциально прерываемое включение света **SEM_Show** и мигание **SEM_Blink**. Обе функции для симуляции светофора комбинируются в **SEM_Cycle** с различными уникальными для каждого режима аргументами, варьируемые через несколько define в CONFIG.

App/semaphore.h

```h
/// CONFIG ///
#define SEM_BLINK_COUNT 2
#define SEM_BLINK_PERIOD 50
#define SEM_GREEN_PERIOD 200
#define SEM_YELLOW_PERIOD 50
#define SEM_RED_PERIOD 650
#define SEM_BTN_PERIOD 510

///  API  ///
void SEM_Init();
void SEM_Cycle();

void SEM_Show(uint16_t color, uint32_t delay, bool interruptible);
void SEM_Blink(uint16_t color, uint32_t delay, uint32_t count);
```

App/semaphore.c

```c
static uint16_t s_currentColor;

/// API ///
void SEM_Init()
{
	s_currentColor = SDK_LED_RED;
}

void SEM_Cycle()
{
	// green
	SEM_Show(SDK_LED_GREEN, SEM_GREEN_PERIOD, false);
	// blinking green
	SEM_Blink(SDK_LED_GREEN, SEM_BLINK_PERIOD, SEM_BLINK_COUNT);

	// yellow
	SEM_Show(SDK_LED_YELLOW, SEM_YELLOW_PERIOD, false);

	// red
	SEM_Show(SDK_LED_RED, SEM_RED_PERIOD, true);
}

void SEM_Show(uint16_t color, uint32_t delay, bool interruptible)
{
	// turn off previous color
	SDK_LED_Set(s_currentColor, SDK_LED_OFF);

	// update current state
	s_currentColor = color;

	// turn on color
	SDK_LED_Set(color, SDK_LED_ON);

	// perform delay
	if(interruptible)
	{
		SDK_TIM_InterruptDelay(delay / 4, delay);
	}
	else
	{
		SDK_TIM_Delay(delay);
	}
}
```

```
void SEM_Blink(uint16_t color, uint32_t delay, uint32_t count)
{
	// turn off previous color
	SDK_LED_Set(s_currentColor, SDK_LED_OFF);

	// update current state
	s_currentColor = color;

	// turn off-on some times
	for(int i = 0; i < count; i++)
	{
		SDK_LED_Set(color, SDK_LED_OFF);
		SDK_TIM_Delay(delay);
		SDK_LED_Set(color, SDK_LED_ON);
		SDK_TIM_Delay(delay);
	}
}
```

App cycle

Перед запуском цикла мы инициализируем светофор и запускаем отложенное прерывание, имитирующее нажатие кнопки в заданные моменты времени. В конце все цвета должны быть выключены.

App/application.c

```
///  API  ///
void SDK_MAIN_PreLoop()
{
	// init semaphore
	SEM_Init();

#if SDK_REMOTE_MODE
	// simulate button press
	SDK_TIM_SetInterrupt(&SDK_BTN_SetDown, SEM_BTN_PERIOD, true);
#endif
}
void SDK_MAIN_LoopFunc()
{
	SEM_Cycle();
}
void SDK_MAIN_PostLoop()
{
	// turn off all colors
	SDK_LED_Set(SDK_LED_GREEN, SDK_LED_OFF);
	SDK_LED_Set(SDK_LED_YELLOW, SDK_LED_OFF);
	SDK_LED_Set(SDK_LED_RED, SDK_LED_OFF);
}
```

Driver

GPIO

Настройка GPIO происходит довольно просто – 13 14 15 пины порта GPIOD, представляющие лампочки, обнуляются, переводятся в режим вывода и инициализируются.

Кнопка создается аналогичным образом с режимом на вход. Также выбран режим PULLUP исходя из схемы устройства.

gpio.c

```
void MX_GPIO_Init(void)
{

  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15, GPIO_PIN_RESET);

  /*Configure GPIO pin : PC15 */
  GPIO_InitStruct.Pin = GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pins : PD13 PD14 PD15 */
  GPIO_InitStruct.Pin = GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

}
```

main

Инициализация составных частей драйвера происходит в функции main

main.c

```
  HAL_Init();
  SystemClock_Config();

  MX_GPIO_Init();
  MX_IWDG_Init();

  SDK_MAIN_Wrapper();
```

Interuptions

Для ежемиллисекундного обновления некоторых пользовательских параметров программы немного модифицирован обработчик прерываний системного счетчика.

Stm32f4xx_it.c

```
void SysTick_Handler(void)
{
  HAL_IncTick();
  /* USER CODE BEGIN SysTick_IRQn 1 */
  SDK_SYS_Tick();
  /* USER CODE END SysTick_IRQn 1 */
}
```

Выводы

Основной проблемой, возникшей в ходе выполнения лабораторной работы, была невозможность инициировать нажатие на кнопку удаленно, что вынудило реализовать имитацию нажатия, функции задержки, поддерживающие его и примитивный таймер на основе системного для осуществления прерывания в требуемый момент времени.

Второй проблемой оказалась ограничение на количество отображаемых событий при трассировке в 32 штуки, что вынудило сократить количество трассируемых событий и сильно ограничить время и разнообразность тестов, что весьма досадно, так как не раскрывает возможностей платформы.

Также неочевидной оказалась среда разработки из множество схожих приложений и приведенные в пособии инструкции, частично конфликтовавшие с описанием шаблона проекта, что потребовало достаточно заметного времени для освоения платформы.

В целом в ходе выполнения стали понятны возможности и ограничения cLab, был получен опыт в области сборки и конфигурации проектов под платформу STM, частично была изучена работа системного счетчика и контроллера GPIO. Суммарно вышло довольно занимательно.

Полный исходный код ЛР можно найти на github https://github.com/Old-Fritz/EmbededSystems

Результат работы: