Университет ИТМО

Кафедра Вычислительной Техники

**Лабораторная работа №1**

**по дисциплине “Встроенные системы”**

Группа Р3401

Комаров Егор Андреевич

Вариант 1

Оценка:\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Принял: Ключев Аркадий Олегович

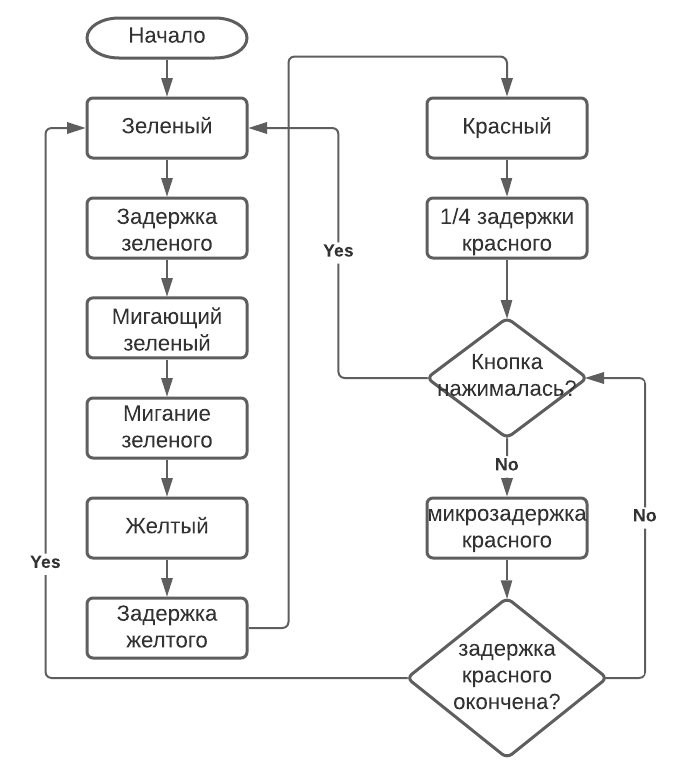
Санкт-Петербург, 2020

# Задание

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

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.

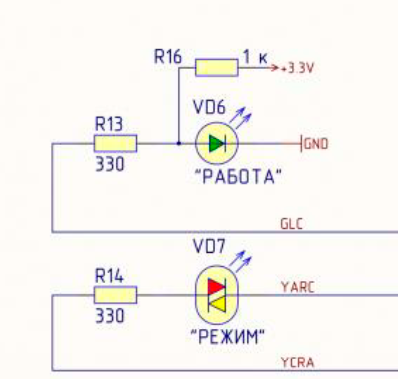
# Блок-схема



# Инструментарий

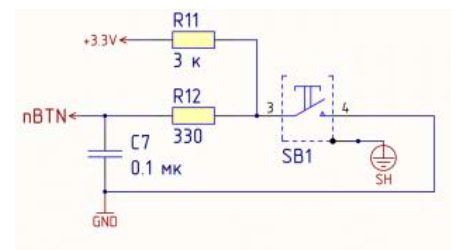
## Аппаратные средства

### 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

|  |
| --- |
| /// 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

|  |
| --- |
| **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:

