Университет ИТМО

Кафедра Вычислительной Техники

**Лабораторная работа №4**

**по дисциплине “Встроенные системы”**

Группа Р3401

Комаров Егор Андреевич

Оценка:\_\_\_\_\_\_\_\_\_\_\_\_\_\_

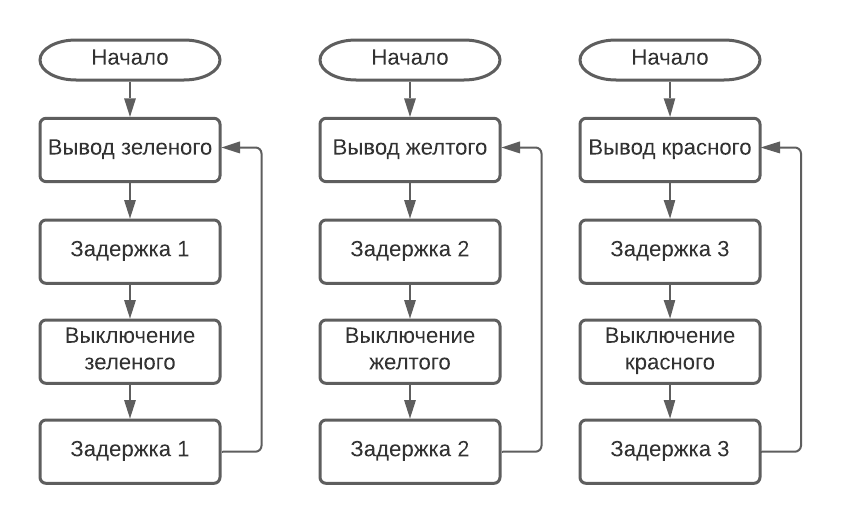
Принял: Ключев Аркадий Олегович

Санкт-Петербург, 2020

# Задание

Имея предоставленную систему cLab, установить на ней FreeRTOS и продемонстрировать её работоспособность.

# Блок схема



# Инструментарий

## FreeRTOS

### Система

**FreeRTOS** является многозадачной операционной системой реального времени для встраиваемых систем. Для **STM32** используется **CMSIS-RTOS API.**

### Создание потока

Задача в **FreeRTOS** представляет из себя функцию вида **void (func)(void const\*)** обычно с бесконечным циклом внутри. Для запуска потока используется **osThreadNew**, возвращающая handle для внешнего управления

### Инициализация ОС

Инициализация проходит через функции **osKernelInitialize** и сгенерированную конфигуратором **MX\_FREERTOS\_Init**, включающую инициализацию определенных ранее вспомогательных объектов. Наконец, для финального запуска используется **osKernelStart.**

### Конфигурация ОС

Конфигурацию можно произвести через **STM32CubeMX** в меню **MIddleware**, для чего требуется назначить один из таймеров (в ЛР используется **TIM6**) на генерацию системных тиков в **System Core->SYS**. Также можно сразу настроить задачи, очереди, семафоры, таймеры, мьютексы.

### Синхронизация

Для обмена данными используются очереди, для межзадачной синхронизации доступны семафоры и мьютексы.

# Исходный код

## SDK

В качестве основы была взят код из прошлой лабораторной работы, но максимально урезанный, так как предполагается, что с установкой **FreeRTOS** использование её инструментов будет более удобным и эффективным.

### System

При инициализации системы сразу ставим отложенный вызов **SDK\_SYS\_Shutdown** через 7 секунд (так как 8 секунд – максимально допустимое время на стенде). Вызов будет проходить через прерывание **SysClock**. После установки времени завершения запускается ОС.

#### SDK/sys.c

|  |
| --- |
| // system API  **void** **SDK\_SYS\_Init**()  {  **#if** SDK\_REMOTE\_MODE  MX\_TRACE\_Init();  SDK\_TRACE\_Start();  SDK\_TIM\_AddInterrupt(&SDK\_SYS\_Shutdown, 7000, false);  **#endif**  osKernelStart();  }  **void** **SDK\_SYS\_Tick**()  {  SDK\_TIM\_Update();  } |

### Main

Вся основная обертка сокращена до инициализации

#### SDK/main\_wrapper.h

|  |
| --- |
| /// API ///  **void** **SDK\_MAIN\_Wrapper**()  {  SDK\_SYS\_Init();  } |

## Application

### FreeRTOS

Для демонстрации работы **FreeRTOS** было создано 3 потока **blink0 blink1 blink2**, реализующие мигание светодиодов разного цвета через разные промежутки времени с использованием **osDelay.**

Запуск каждого потока состоит из определения его свойств, объявления функции потока и непосредственно создания потока с **osThreadNew**

#### freertos.c

|  |
| --- |
| /\* Definitions for blink01 \*/  osThreadId\_t blink01Handle;  **const** osThreadAttr\_t blink01\_attributes = {  .name = "blink01",  .priority = (osPriority\_t) *osPriorityNormal*,  .stack\_size = 128 \* 4  };  /\* Definitions for blink02 \*/  osThreadId\_t blink02Handle;  **const** osThreadAttr\_t blink02\_attributes = {  .name = "blink02",  .priority = (osPriority\_t) *osPriorityBelowNormal*,  .stack\_size = 128 \* 4  };  /\* Definitions for blink03 \*/  osThreadId\_t blink03Handle;  **const** osThreadAttr\_t blink03\_attributes = {  .name = "blink03",  .priority = (osPriority\_t) *osPriorityLow*,  .stack\_size = 128 \* 4  };  ... ... ...  **void** **StartBlink01**(**void** \*argument);  **void** **StartBlink02**(**void** \*argument);  **void** **StartBlink03**(**void** \*argument);  ... ... ...  /\* creation of blink01 \*/  blink01Handle = osThreadNew(StartBlink01, NULL, &blink01\_attributes);  /\* creation of blink02 \*/  blink02Handle = osThreadNew(StartBlink02, NULL, &blink02\_attributes);  blink03Handle = osThreadNew(StartBlink03, NULL, &blink03\_attributes);  ... ... ...  **void** **StartBlink01**(**void** \*argument)  {  **for**(;;)  {  SDK\_LED\_Toggle(SDK\_LED\_GREEN);  osDelay(700);  }  osThreadTerminate(NULL);  }  **void** **StartBlink02**(**void** \*argument)  {  **for**(;;)  {  SDK\_LED\_Toggle(SDK\_LED\_YELLOW);  osDelay(900);  }  osThreadTerminate(NULL);  }  **void** **StartBlink03**(**void** \*argument)  {  **for**(;;)  {  SDK\_LED\_Toggle(SDK\_LED\_RED);  osDelay(1200);  }  osThreadTerminate(NULL);  } |

## Driver

В функции main достаточно инициализировать ядро и определенные в **MX\_FREERTOS\_Init** потоки и позже запустить привычную обертку над main

Так как теперь используется иной таймер в качестве системного, его прерывания проходят через стандартный **HAL\_TIM\_PeriodElapsedCallback**, где имеется возможность добавить к этому событию пользовательские действия.

#### main.c

|  |
| --- |
| SystemClock\_Config();  MX\_GPIO\_Init();  MX\_IWDG\_Init();  osKernelInitialize();  MX\_FREERTOS\_Init();  SDK\_MAIN\_Wrapper();  **void** **HAL\_TIM\_PeriodElapsedCallback**(TIM\_HandleTypeDef \*htim)  {  **if** (htim->Instance == TIM6) {  HAL\_IncTick();  SDK\_SYS\_Tick();  }  } |

# Выводы

Единственной трудностью в установке **FreeRTOS** оказалась необходимость уместить всю работу между **SDK\_TRACE\_Start** и **SDK\_TRACE\_Stop** для возможности получения обратной связи от **cLab**. В связи с неочевидным порядком работа планировщика системы и условиями для корректной работы трассировщика было опробовано множество различных способов до нахождения рабочего, принцип которого заключается в вызове завершающей функции через прерывание системного таймера ближе к концу максимального срока работы с лабораторным стендом.

В целом использование **FreeRTOS** оказалось удобным, простым и достаточно увлекательным. Выполнение лабораторной работы позволило получить некоторые представления о работе операционных систем на маломощных устройствах и принципа синхронизации между задачами в ней.

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

#### Результат работы:

