Университет ИТМО

Кафедра Вычислительной Техники

Лабораторная работа №4

по дисциплине “Встроенные системы”

Группа P3401

Комаров Егор Андреевич

Оценка:_____

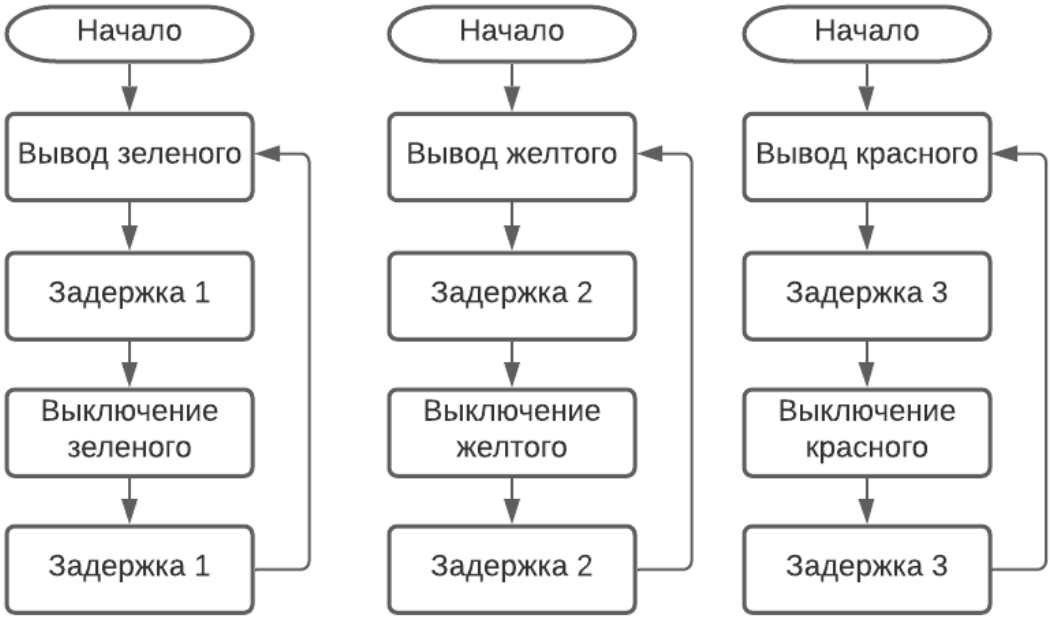
Принял: Ключев Аркадий Олегович

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: