Университет ИТМО

Кафедра Вычислительной Техники

**Лабораторная работа №3**

**по дисциплине “Встроенные системы”**

Группа Р3401

Комаров Егор Андреевич

Вариант 1

Оценка:\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Принял: Ключев Аркадий Олегович

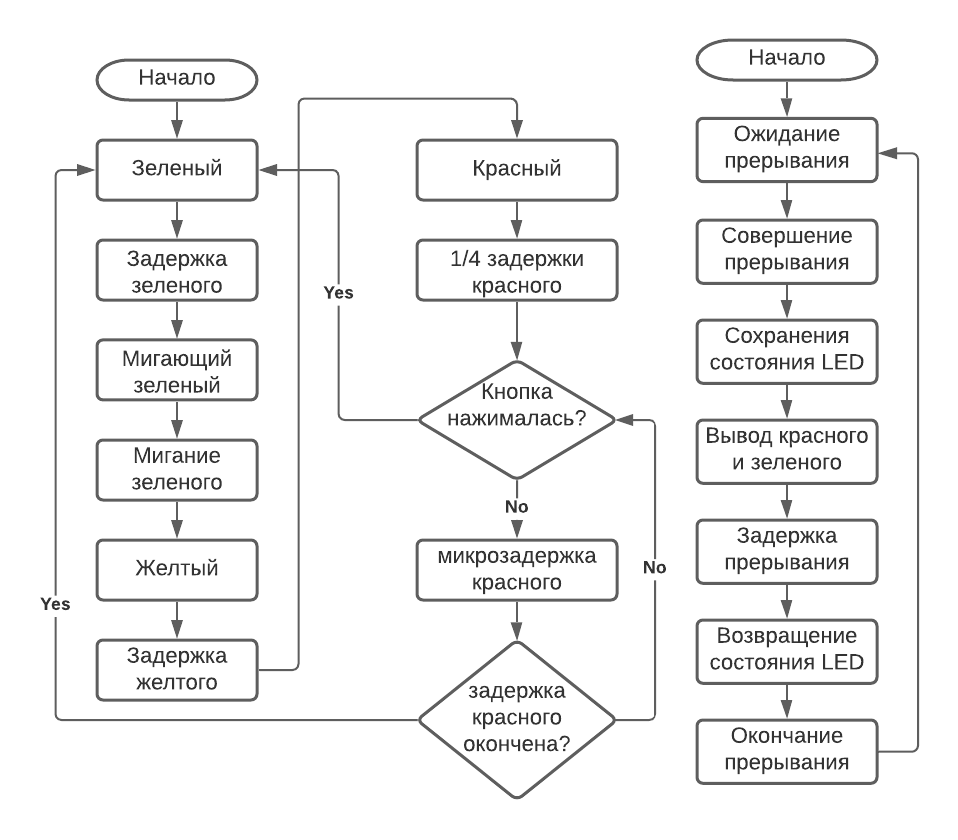
Санкт-Петербург, 2020

# Задание

# Реализовать обработчик прерываний, а также основную программу, которая будет прерываться (мигание светодиодов, вывод текста на дисплей и т.д)

|  |  |  |  |
| --- | --- | --- | --- |
| Основная программа | Обработчик прерывания | Источник прерывания | Длительность |
| Переключение светодиодов | Включается красный и зеленый светодиод | Таймер | 2с |

# Блок-схема



# Инструментарий

## Аппаратные средства

### Механизм прерываний

Встроенный обработчик, фиксирующий запросы на прерывание и исходя из приоритетов передающий управление по указанным в векторе данного прерывания данным.

### Таймер основного прерывания

Генерирует одно прерывание раз в 2 секунды для настраиваемого callback

Идентификатор – TIM1  
Prescaler – делитель основной частоты – 16800  
Period – 13500, т.е. время активации – через 1350 мс  
Группа приоритета – 1  
Включены глобальные прерывания  
Одноимпульсный

### Высокоприоритетный таймер

Обладает более высоким уровнем приоритета, чем другие используемые таймеры, за счет чего вызывает вложенные прерывания каждые 0.1 мс, на основе чего реализованы настраиваемые задержки в основном прерывании.

Идентификатор – TIM3  
Prescaler – делитель основной частоты – 4200  
Period – 1  
Группа приоритета – 0  
Включены глобальные прерывания

## Программные средства

### Старт таймеров

Для старта таймеров используется функция **HAL\_TIM\_Base\_Start\_IT** с переданным в качестве аргумента handle. Для предотвращения прерывания сразу на старте перед запуском очищается флаг прерывания **TIM\_SR\_UIF** через **\_\_HAL\_TIM\_CLEAR\_FLAG**

### Приоритет прерываний

Для задания приоритетности используются функции **HAL\_NVIC\_SetPriority** и **HAL\_NVIC\_SetPriorityGrouping**. Через первую задается группа и подгруппа для указанного тип прерывания. Через вторую – глобальное разбиение на группы и подгруппы. В качестве параметра может быть одно из пяти разбиений **NVIC\_PRIORITYGROUP\_X**, где **X** от **0** до **4** указывает на количество бит, отводимых под номер группы, при этом **4-X** – биты, отводимые на подгруппы.

Прерывания могут быть прерваны только прерываниями более приоритетной группы. Подгруппа определяет порядок одновременных (на соседних тактах процессора) прерываний.

### Инициализация таймеров

Для инициализации таймеров использовалась **HAL\_TIM\_Base\_Init**, принимающая структуру из основных свойств таймеров, описанных выше. Дополнительные параметры настраиваются отдельно через функции **HAL\_TIM\_ConfigClockSource,** **HAL\_TIM\_OnePulse\_Init**, **HAL\_TIMEx\_MasterConfigSynchronization** в зависимости от ситуации

Отдельно перед использованием запускается соответствующий CLK с помощью **\_\_HAL\_RCC\_TIMX\_CLK\_ENABLE**

### Прерывания таймеров

Наличием прерываний можно управлять динамически через функции **HAL\_NVIC\_DisableIRQ** и **HAL\_NVIC\_EnableIRQ** принимающие тип прерывания.

После осуществления прерывания выполнение переходит в общий **HAL\_TIM\_PeriodElapsedCallback** (для выбранного типа прерывания по переполнению периода)

# Исходный код

## SDK

В качестве основы была взят код из прошлой лабораторной работы.

### Interruptions

Для обработки разных типов прерываний было добавлено два callback – первый appCallback задается в контексте приложения, а второй необходим для обработки вложенных прерываний, на основе которых реализуется функция задержки, которую можно использовать в коде прерывания (**SDK\_INT\_Delay**).

При вызове обработке основного прерывания происходит его трассировка с уникальным ID типа события **P1** (**SDK\_INT**)

#### SDK/interface.h

|  |
| --- |
| **#define** SDK\_INT\_HANDLE\_APP htim1  **#define** SDK\_INT\_HANDLE\_PRIORITY htim3  **#define** SDK\_INT\_TYPE\_APP TIM1  **#define** SDK\_INT\_TYPE\_PROIORITY TIM3  **#define** SDK\_INT\_FREQ\_MS\_PRIORITY 10  **#define** SDK\_INT P1  // interrupts API  **void** **SDK\_INT\_Init**();  **void** **SDK\_INT\_AppCallback**();  **void** **SDK\_INT\_PriorityCallback**();  **void** **SDK\_INT\_Delay**(uint32\_t delay);  uint32\_t **SDK\_INT\_GetTicks**(); |

#### SDK/int.c

|  |
| --- |
| /// STATICS ///  \_\_IO uint32\_t s\_ticksHP = 0;  /// API ///  **void** **SDK\_INT\_Init**()  {  // timer for app defined interruptions  \_\_HAL\_TIM\_CLEAR\_FLAG(&SDK\_INT\_HANDLE\_APP, TIM\_SR\_UIF);  HAL\_TIM\_Base\_Start\_IT(&SDK\_INT\_HANDLE\_APP);  // timer for high frequency priority interruptions  \_\_HAL\_TIM\_CLEAR\_FLAG(&SDK\_INT\_HANDLE\_PRIORITY, TIM\_SR\_UIF);  HAL\_TIM\_Base\_Start\_IT(&SDK\_INT\_HANDLE\_PRIORITY);  }  **void** **HAL\_TIM\_PeriodElapsedCallback**(TIM\_HandleTypeDef \*htim)  {  **if**(htim->Instance == SDK\_INT\_TYPE\_APP)  {  **#if** SDK\_REMOTE\_MODE  SDK\_TRACE\_Timestamp(SDK\_INT, true);  **#endif**  SDK\_INT\_AppCallback();  **#if** SDK\_REMOTE\_MODE  SDK\_TRACE\_Timestamp(SDK\_INT, false);  **#endif**  }  **else** **if** (htim->Instance == SDK\_INT\_TYPE\_PROIORITY)  {  SDK\_INT\_PriorityCallback();  }  }  **void** **SDK\_INT\_PriorityCallback**()  {  s\_ticksHP += 1;  }  uint32\_t **SDK\_INT\_GetTicks**()  {  **return** s\_ticksHP;  }  **void** **SDK\_INT\_Delay**(uint32\_t delay)  {  uint32\_t tickstart = SDK\_INT\_GetTicks();  uint32\_t wait = delay \* SDK\_INT\_FREQ\_MS\_PRIORITY;  // guarantee minimum wait  wait += 1;  **while**((SDK\_INT\_GetTicks() - tickstart) < wait)  {  }  } |

## Application

### App cycle

Для того, чтобы не повлиять на выполнение основного кода, все затрагиваемые данные в виде состояния лампочек сохраняются и в конце прерывания восстанавливаются в прежнее состояние.

#### App/application.c

|  |
| --- |
| **void** **SDK\_MAIN\_PreLoop**()  {  // init semaphore  SEM\_Init();  }  **void** **SDK\_INT\_AppCallback**()  {  GPIO\_PinState stateGreen = SDK\_LED\_Read(SDK\_LED\_GREEN);  GPIO\_PinState stateYellow = SDK\_LED\_Read(SDK\_LED\_YELLOW);  GPIO\_PinState stateRed = SDK\_LED\_Read(SDK\_LED\_RED);  SDK\_LED\_Set(SDK\_LED\_GREEN, SDK\_LED\_ON);  SDK\_LED\_Set(SDK\_LED\_YELLOW, SDK\_LED\_OFF);  SDK\_LED\_Set(SDK\_LED\_RED, SDK\_LED\_ON);  SDK\_INT\_Delay(2000);  SDK\_LED\_Set(SDK\_LED\_GREEN, stateGreen);  SDK\_LED\_Set(SDK\_LED\_YELLOW, stateYellow);  SDK\_LED\_Set(SDK\_LED\_RED, stateRed);  } |

## Driver

### Main

Для обработки двух типов создается несколько групп приоритетов через функцию **HAL\_NVIC\_SetPriorityGrouping**

#### main.c

|  |
| --- |
| HAL\_Init();  SystemClock\_Config();  HAL\_NVIC\_SetPriorityGrouping(NVIC\_PRIORITYGROUP\_2);  MX\_GPIO\_Init();  MX\_IWDG\_Init();  MX\_USART2\_UART\_Init();  MX\_USART3\_UART\_Init();  MX\_TIM1\_Init();  MX\_TIM3\_Init();  SDK\_MAIN\_Wrapper(); |

### Timer

При инициализации помимо вызова стандортной INIT функции со структурой с параметрами в **HAL\_TIM\_Base\_MspInit** им назначается свой приоритет прерываний

#### tim.c

|  |
| --- |
| /\* TIM1 init function \*/  **void** **MX\_TIM1\_Init**(**void**)  {  TIM\_ClockConfigTypeDef sClockSourceConfig = {0};  TIM\_MasterConfigTypeDef sMasterConfig = {0};  htim1.Instance = TIM1;  htim1.Init.Prescaler = 16800;  htim1.Init.CounterMode = TIM\_COUNTERMODE\_UP;  htim1.Init.Period = 13500;  htim1.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1;  htim1.Init.RepetitionCounter = 0;  htim1.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_DISABLE;  **if** (HAL\_TIM\_Base\_Init(&htim1) != *HAL\_OK*)  {  Error\_Handler();  }  sClockSourceConfig.ClockSource = TIM\_CLOCKSOURCE\_INTERNAL;  **if** (HAL\_TIM\_ConfigClockSource(&htim1, &sClockSourceConfig) != *HAL\_OK*)  {  Error\_Handler();  }  **if** (HAL\_TIM\_OnePulse\_Init(&htim1, TIM\_OPMODE\_SINGLE) != *HAL\_OK*)  {  Error\_Handler();  }  sMasterConfig.MasterOutputTrigger = TIM\_TRGO\_RESET;  sMasterConfig.MasterSlaveMode = TIM\_MASTERSLAVEMODE\_DISABLE;  **if** (HAL\_TIMEx\_MasterConfigSynchronization(&htim1, &sMasterConfig) != *HAL\_OK*)  {  Error\_Handler();  }  }  /\* TIM3 init function \*/  **void** **MX\_TIM3\_Init**(**void**)  {  TIM\_ClockConfigTypeDef sClockSourceConfig = {0};  TIM\_MasterConfigTypeDef sMasterConfig = {0};  htim3.Instance = TIM3;  htim3.Init.Prescaler = 4200;  htim3.Init.CounterMode = TIM\_COUNTERMODE\_UP;  htim3.Init.Period = 1;  htim3.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1;  htim3.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_DISABLE;  **if** (HAL\_TIM\_Base\_Init(&htim3) != *HAL\_OK*)  {  Error\_Handler();  }  sClockSourceConfig.ClockSource = TIM\_CLOCKSOURCE\_INTERNAL;  **if** (HAL\_TIM\_ConfigClockSource(&htim3, &sClockSourceConfig) != *HAL\_OK*)  {  Error\_Handler();  }  sMasterConfig.MasterOutputTrigger = TIM\_TRGO\_RESET;  sMasterConfig.MasterSlaveMode = TIM\_MASTERSLAVEMODE\_DISABLE;  **if** (HAL\_TIMEx\_MasterConfigSynchronization(&htim3, &sMasterConfig) != *HAL\_OK*)  {  Error\_Handler();  }  }  **void** **HAL\_TIM\_Base\_MspInit**(TIM\_HandleTypeDef\* tim\_baseHandle)  {  **if**(tim\_baseHandle->Instance==TIM1)  {  /\* USER CODE BEGIN TIM1\_MspInit 0 \*/  /\* USER CODE END TIM1\_MspInit 0 \*/  /\* TIM1 clock enable \*/  \_\_HAL\_RCC\_TIM1\_CLK\_ENABLE();  /\* TIM1 interrupt Init \*/  HAL\_NVIC\_SetPriority(*TIM1\_UP\_TIM10\_IRQn*, 1, 0);  HAL\_NVIC\_EnableIRQ(*TIM1\_UP\_TIM10\_IRQn*);  /\* USER CODE BEGIN TIM1\_MspInit 1 \*/  /\* USER CODE END TIM1\_MspInit 1 \*/  }  **else** **if**(tim\_baseHandle->Instance==TIM3)  {  /\* USER CODE BEGIN TIM3\_MspInit 0 \*/  /\* USER CODE END TIM3\_MspInit 0 \*/  /\* TIM3 clock enable \*/  \_\_HAL\_RCC\_TIM3\_CLK\_ENABLE();  /\* TIM3 interrupt Init \*/  HAL\_NVIC\_SetPriority(*TIM3\_IRQn*, 0, 0);  HAL\_NVIC\_EnableIRQ(*TIM3\_IRQn*);  /\* USER CODE BEGIN TIM3\_MspInit 1 \*/  /\* USER CODE END TIM3\_MspInit 1 \*/  }  } |

# Выводы

Некоторые трудности в данной лабораторной работе вызвала реализация задержки в прерывании, что потребовало добавления еще одного таймера по функциям близкого к SysClock c более высоким приоритетом прерывания. Ещё одной незначительной проблемой оказалась разве что реализация счетчика тиков для этого таймера из-за необходимости объявления соответствующей переменной с модификатором volatile (\_\_IO в STM), что оказалось не совсем очевидным из-за отсутствия каких-либо средств отладки.

Выполнение лабораторной работы дало более четкое понимание принципа работы таймеров и организации прерываний, в том числе приоритетных и вложенных.

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

#### Результат работы:

