Университет ИТМО

Кафедра Вычислительной Техники

Лабораторная работа №3

по дисциплине “Встроенные системы”

Группа P3401

Комаров Егор Андреевич

Вариант 1

Оценка:_____

Принял: Ключев Аркадий Олегович

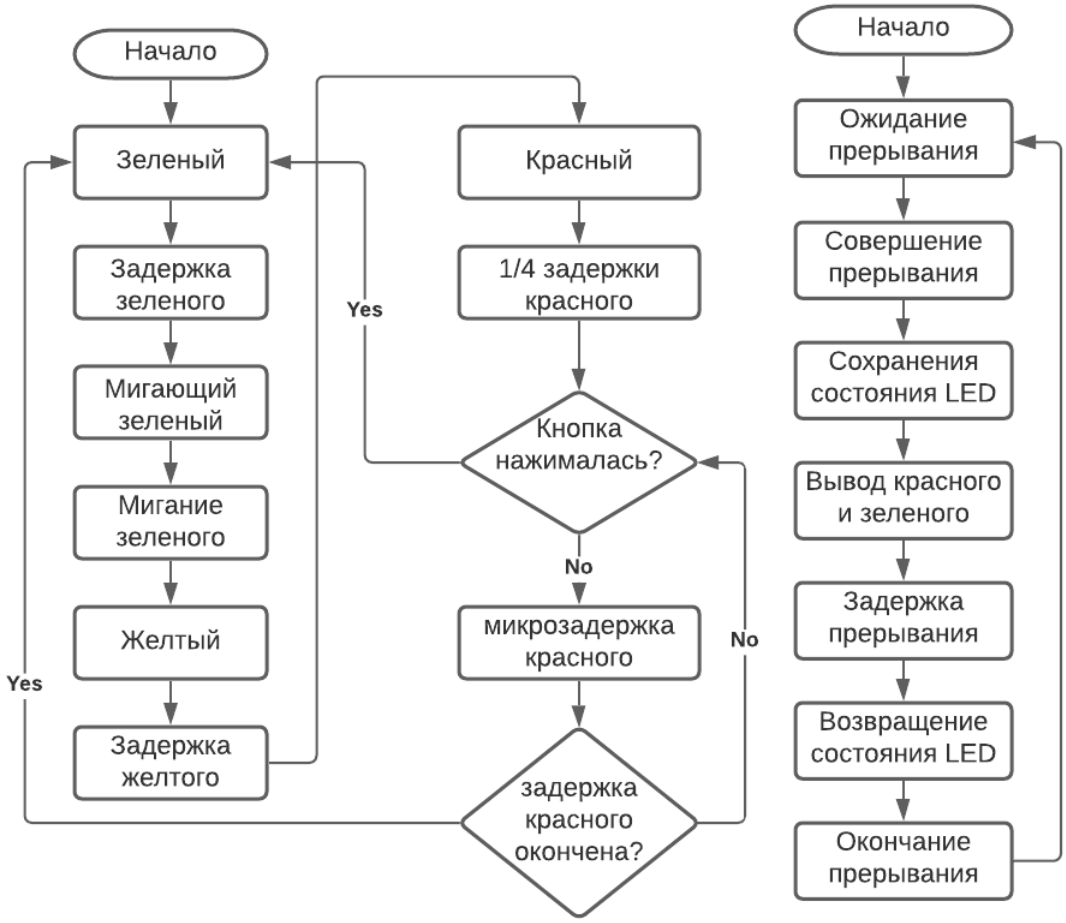
Санкт-Петербург, 2020

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 с 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: