НИИИС

МИКРОСХЕМЫ ИНТЕГРАЛЬНЫЕ
1914ВА018
Описание библиотек

Содержание

1. Описание библиотек микроконтроллера

1.1 Библиотека функций портов ввода-вывода GPIO

№	Функция	Описание
Инициализация и конфигурация		
1	GPIO_DeInit	Сброс регистров порта GPIOx к значениям по умолчанию
2	GPIO_Init	Инициализация порта GPIOx согласно заданным параметрам в GPIO_InitStruct
3	GPIO_StructInit	Присвоение членам GPIO_InitStruct значений по умолчанию
4	GPIO_ITSet	Настройка события для активации прерывания
5	GPIO_ITConfig	Независимая установка 16-ти маскированных прерываний.
Запись и чтение		
6	GPIO_ReadInputDataBit	Чтение входного значения вывода порта GPIOx
7	GPIO_ReadInputData	Чтение входных данных из порта GPIOx
8	GPIO_ReadOutputDataBit	Чтение выходного значения вывода порта GPIOx
9	GPIO_ReadOutputData	Чтение выходных данных из порта GPIOx
10	GPIO_SetBits	Независимая установка бит порта GPIOx в «1»
11	GPIO_SetHighBits	Независимая установка 8-ми старших бит порта GPIOx в «1»
12	GPIO_SetLowBits	Независимая установка 8-ми младших бит порта GPIOx в «1»
13	GPIO_Set	Установка всех бит порта GPIOx в «1»
14	GPIO_ResetBits	Независимый сброс бит порта GPIOx в «0»
15	GPIO_ResetHighBits	Независимый сброс 8-ми старших бит порта GPIOx в «0»
16	GPIO_ResetLowBits	Независимый сброс 8-ми младших бит порта GPIOx в «0»
17	GPIO_Reset	Сброс бит порта GPIOx в «0»
18	GPIO_SRBits	Независимая установка и сброс бит порта GPIOx
19	GPIO_SRHighBits	Независимая установка и сброс 8-ми старших бит порта GPIOx
20	GPIO_SRLowBits	Независимая установка и сброс 8-ми младших бит порта GPIOx
21	GPIO_WriteBit	Установка или сброс бита порта GPIOx
22	GPIO_Write	Запись данных в порт GPIOx
23	GPIO_ToggleBits	Независимая инверсия бит порта GPIOx
24	GPIO_ToggleHighBits	Независимая инверсия 8-ми старших бит порта GPIOx
25	GPIO_ToggleLowBits	Независимая инверсия 8-ми младших бит порта GPIOx
26	GPIO_GetITStatus	Чтение состояния прерывания по событию
27	GPIO_ClearITPendingBit	Сброс состояния прерывания по событию

1.1.1 Инициализация и конфигурация

Функция:

void GPIO_DeInit(GPIO_TypeDef* GPIOx)

Сброс регистров порта GPIOx к значениям по умолчанию (все выводы порта устанавливаются в режим входа).

Параметры:

GPIOx – где x выбранный порт (A..E);

Возвращаемые значения: нет.

Функция:

void GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct)

Инициализация порта GPIOx согласно заданным параметрам в *GPIO_InitStruct*.

Параметры:

GPIOx – где x выбранный порт (A..E);

GPIO_InitStruct – указатель на *GPIO_InitTypeDef* структуру которая содержит конфигурационную информацию для GPIO порта.

Возвращаемые значения: нет.

Функция:

void GPIO_StructInit(GPIO_InitTypeDef* GPIO_InitStruct)

Присвоение членам *GPIO_InitStruct* значений по умолчанию:

GPIO_Pin = GPIO_Pin_All;

GPIO_Mode = GPIO_Mode_IN

Параметры:

GPIO_InitStruct – указатель на *GPIO_InitTypeDef* структуру которая будет инициализирована.

Возвращаемые значения: нет.

Функция:

void GPIO_ITSet(GPIO_TypeDef* GPIOx, uint32_t GPIO_Pin, uint32_t GPIO_IT_TYPE, uint32_t GPIO_IT_POL);

Настройка события для активации прерывания.

Параметры:

GPIOx – где x выбранный порт (A..E);

GPIO_Pin – выбранный бит, этот параметр может быть любым GPIO_Pin_x, где x вывод порта [15:0].

GPIO_IT_TYPE - установка признака прерывания**.** Принимаемые значения:

GPIO_IT_Type_High - установить в качестве признака прерывания положительный или отрицательный перепад на внешних выводах порта

GPIO_IT_Type_Low - прерывание генерируется при НИЗКОМ или ВЫСОКОМ уровне на внешних выводах порта

GPIO_IT_POL - установка признака прерывания**.** Принимаемые значения:

GPIO_IT_Pol_High - прерывание генерируется при ВЫСОКОМ уровне или при положительном перепаде на внешних выводах порта
GPIO_IT_Pol_Low - прерывание генерируется при НИЗКОМ уровне или при отрицательном перепаде на внешних выводах порта

Возвращаемые значения: нет.

Функция:

void GPIO_ITConfig(GPIO_TypeDef* GPIOx, uint32_t GPIO_Pin, FunctionalState NewState);

Независимая установка 16-ти маскированных прерываний.

Параметры:

GPIOx – где x выбранный порт (A..E);

GPIO_Pin – выбранный бит, этот параметр может быть любым GPIO_Pin_x, где x вывод порта [15:0].

NewState – новое состояние работы порта. Принимаемые значения ENABLE и DISABLE.

Возвращаемые значения: нет.

1.1.2 Запись и чтение

Функция:

uint8_t GPIO_ReadInputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)

Чтение входного значения вывода порта GPIOx.

Параметры:

GPIOx – где x выбранный порт (A..E);

GPIO_Pin – выбранный бит, этот параметр может быть любым GPIO_Pin_x, где x вывод порта [15:0].

Возвращаемые значения: Состояние входа GPIO_Pin порта GPIOx.

Функция:

uint16_t GPIO_ReadInputData(GPIO_TypeDef* GPIOx)

Чтение входных данных порта GPIOx.

Параметры:

GPIOx – где x выбранный порт (A..E);

Возвращаемые значения: Значение входных данных порта GPIOx.

Функция:

uint8_t GPIO_ReadOutputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)

Чтение выходного значения вывода порта GPIOx.

Параметры:

GPIOx – где x выбранный порт (A..E);

GPIO_Pin – выбранный бит, этот параметр может быть любым GPIO_Pin_x, где x вывод порта [15:0].

Возвращаемые значения: Состояние выхода GPIO_Pin порта GPIOx.

Функция:

uint16_t GPIO_ReadOutputData(GPIO_TypeDef* GPIOx)

Чтение выходных данных порта GPIOx.

Параметры:

GPIOx – где x выбранный порт (A..E);

Возвращаемые значения: Значение выходных данных порта GPIOx.

Функция:

void GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)

Независимая установка бит порта GPIOx в «1» с помощью битовой маски.

Параметры:

GPIOx – где x выбранный порт (A..E);

GPIO_Pin – битовая маска, этот параметр может быть любой комбинацией GPIO_Pin_x, где x вывод порта [15:0].

Возвращаемые значения: нет.

Функция:

void GPIO_SetHighBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)

Независимая установка 8-ми старших бит порта GPIOx в «1» с помощью битовой маски.

Параметры:

GPIOx – где x выбранный порт (A..E);

GPIO_Pin – битовая маска, этот параметр может быть любой комбинацией GPIO_Pin_x, где x вывод порта [15:8].

Возвращаемые значения: нет.

Функция:

void GPIO_SetLowBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)

Независимая установка 8-ми младших бит порта GPIOx в «1» с помощью битовой маски.

Параметры:

GPIOx – где x выбранный порт (A..E);

GPIO_Pin – битовая маска, этот параметр может быть любой комбинацией GPIO_Pin_x, где x вывод порта [7:0].

Возвращаемые значения: нет.

Функция:

void GPIO_Set (GPIO_TypeDef* GPIOx)

Установка всех бит порта GPIOx в «1».

Параметры:

GPIOx – где x выбранный порт (A..E).

Возвращаемые значения: нет.

Функция:

void GPIO_ResetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)

Независимый сброс бит порта GPIOx в «0» с помощью битовой маски.

Параметры:

GPIOx – где x выбранный порт (A..E);

GPIO_Pin – битовая маска, этот параметр может быть любой комбинацией GPIO_Pin_x, где x вывод порта [15:0].

Возвращаемые значения: нет.

Функция:

void GPIO_ResetHighBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)

Независимый сброс 8-ми старших бит порта GPIOx в «0» с помощью битовой маски.

Параметры:

GPIOx – где x выбранный порт (A..E);

GPIO_Pin – битовая маска, этот параметр может быть любой комбинацией GPIO_Pin_x, где x вывод порта [15:8].

Возвращаемые значения: нет.

Функция:

void GPIO_ResetLowBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)

Независимый сброс 8-ми младших бит порта GPIOx в «0» с помощью битовой маски.

Параметры:

GPIOx – где x выбранный порт (A..E);

GPIO_Pin – битовая маска, этот параметр может быть любой комбинацией GPIO_Pin_x, где x вывод порта [7:0].

Возвращаемые значения: нет.

Функция:

void GPIO_Reset(GPIO_TypeDef* GPIOx)

Сброс всех бит порта GPIOx в «0».

Параметры:

GPIOx – где x выбранный порт (A..E).

Возвращаемые значения: нет.

Функция:

void GPIO_SRBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin_Set, uint16_t GPIO_Pin_Reset)

Независимая установка и сброс бит порта GPIOx с помощью битовых масок.

Параметры:

GPIOx – где x выбранный порт (A..E);

GPIO_Pin_Set – битовая маска для установки, этот параметр может быть любой комбинацией GPIO_Pin_x, где x вывод порта [15:0].

GPIO_Pin_Reset – битовая маска для сброса, этот параметр может быть любой комбинацией GPIO_Pin_x, где x вывод порта [15:0].

Возвращаемые значения: нет.

Функция:

void GPIO_SRHighBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin_Set, uint16_t GPIO_Pin_Reset)

Независимая установка и сброс 8-ми старших бит порта GPIOx в «0» с помощью битовой маски.

Параметры:

GPIOx – где x выбранный порт (A..E);

GPIO_Pin_Set – битовая маска для установки, этот параметр может быть любой комбинацией GPIO_Pin_x, где x вывод порта [15:8];

GPIO_Pin_Reset – битовая маска для сброса, этот параметр может быть любой комбинацией GPIO_Pin_x, где x вывод порта [15:8].

Возвращаемые значения: нет.

Функция:

void GPIO_SRLowBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin_Set, uint16_t GPIO_Pin_Reset)

Независимая установка и сброс 8-ми младших бит порта GPIOx в «0» с помощью битовой маски.

Параметры:

GPIOx – где x выбранный порт (A..E);

GPIO_Pin_Set – битовая маска для установки, этот параметр может быть любой комбинацией GPIO_Pin_x, где x вывод порта [7:0];

GPIO_Pin_Reset – битовая маска для сброса, этот параметр может быть любой комбинацией GPIO_Pin_x, где x вывод порта [7:0].

Возвращаемые значения: нет.

Функция:

void GPIO_WriteBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, BitAction BitVal)

Установка или сброс бита порта GPIOx.

Параметры:

GPIOx – где x выбранный порт (A..E);

GPIO_Pin – выбранный бит, этот параметр может быть любым GPIO_Pin_x, где x вывод порта [15:0];

Bit_Val – заданное значение для записи выбранного бита, этот параметр может быть одним из значений BitAction:

Bit_RESET – сброс бита порта;

Bit_SET – установка бита порта.

Возвращаемые значения: нет.

Функция:

void GPIO_Write(GPIO_TypeDef* GPIOx, uint16_t PortVal)

Запись данных в порт GPIOx.

Параметры:

GPIOx – где x выбранный порт (A..E);

PortVal – заданное значение для записи в порт.

Возвращаемые значения: нет.

Функция:

void GPIO_ToggleBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)

Независимая инверсия бит порта GPIOx с помощью битовой маски.

Параметры:

GPIOx – где x выбранный порт (A..E);

GPIO_Pin – битовая маска, этот параметр может быть любой комбинацией GPIO_Pin_x, где x вывод порта [15:0].

Возвращаемые значения: нет.

Функция:

void GPIO_ToggleHighBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)

Независимая инверсия 8-ми старших бит порта GPIOx.

Параметры:

GPIOx – где x выбранный порт (A..E);

GPIO_Pin – битовая маска, этот параметр может быть любой комбинацией GPIO_Pin_x, где x вывод порта [15:0].

Возвращаемые значения: нет.

Функция:

void GPIO_ToggleLowBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)

Независимая инверсия 8-ми младших бит порта GPIOx.

Параметры:

GPIOx – где x выбранный порт (A..E);

GPIO_Pin – битовая маска, этот параметр может быть любой комбинацией GPIO_Pin_x, где x вывод порта [15:0].

Возвращаемые значения: нет.

Функция:

ITStatus GPIO_GetITStatus(GPIO_TypeDef* GPIOx, uint32_t GPIO_Pin);

Чтение состояния прерывания по событию.

Параметры:

GPIOx – где x выбранный порт (A..E);

GPIO_Pin – битовая маска, этот параметр может быть любой комбинацией GPIO_Pin_x, где x вывод порта [15:0].

Возвращаемые значения: состояние прерывания.

Функция:

void GPIO_ClearITPendingBit(GPIO_TypeDef* GPIOx, uint32_t GPIO_Pin);

Сброс состояния прерывания по событию.

Параметры:

GPIOx – где x выбранный порт (A..E);

GPIO_Pin – битовая маска, этот параметр может быть любой комбинацией GPIO_Pin_x, где x вывод порта [15:0].

Возвращаемые значения: нет.

1.1.3 Применение библиотеки портов ввода-вывода

После сброса все выводы портов ввода-вывода находятся в состоянии входа. В микросхеме 5 16-битных портов ввода-вывода (**GPIOA, GPIOB … GPIOE**).

Порты настраиваются регистрами, которые инициализируются функцией ***GPIO_Init()***, в соответствии со значениями структуры ***GPIO_InitTypeDef***.

GPIO_Pin -> GPIO_Pin_0 … 15, GPIO_Pin_All

GPIO_Mode:

GPIO_Mode_IN	– режим входа;
GPIO_Mode_Out	- режим выхода.

После инициализации структуры возможно использование набора функций.

Пример_1

```
// Создаем переменную GPIO_Init_OUT с типом данных GPIO_InitTypeDef;
GPIO_InitTypeDef GPIO_Init_OUT;

// Описываем структуру GPIO_Init_OUT;
GPIO_Init_OUT.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_4 |
GPIO_Pin_5 | GPIO_Pin_6 | GPIO_Pin_7; // выбираем выводы порта;
GPIO_Init_OUT.GPIO_Mode = GPIO_Mode_OUT; // устанавливаем режим работы выбранных выводов как
выход;

//Инициализируем структуру GPIO_Init_OUT (название порта, указатель на структуру);
GPIO_Init(GPIOA, &GPIO_Init_OUT); // выводы 0-7 порта GPIOA переводятся в режим выхода;
```

Пример_2

```
GPIO_InitTypeDef GPIO_Init_IN;
GPIO_Init_IN.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_10 | GPIO_Pin_11 | GPIO_Pin_12 |
GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15; // выбираем выводы порта;
GPIO_Init_IN.GPIO_Mode = GPIO_Mode_IN; // устанавливаем режим работы выбранных выводов как вход;
GPIO_Init(GPIOA, &GPIO_Init_IN); // выводы 8-15 порта GPIOA переводятся в режим входа;
```

Пример_3

```
GPIO_InitTypeDef GPIO_Init_ABC;
```

```
GPIO_Init_ABC.GPIO_Pin = GPIO_Pin_All; // выбираем все выводы порта;
GPIO_Init_ABC.GPIO_Mode = GPIO_Mode_OUT;
GPIO_Init(GPIOB, &GPIO_Init_ABC); // все выводы порта GPIOB переводятся в режим выхода;
```

Пример_4

```
GPIO_Set(GPIOB); // установка всех выходов порта GPIOB в «1».
GPIO_Reset(GPIOB); // сброс всех выходов порта GPIOB в «0».
GPIO_SetBits(GPIOB, GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3); // установка выходов 0-3
в «1».
GPIO_SRBits(GPIOB, GPIO_Pin_8 | GPIO_Pin_10, GPIO_Pin_0 | GPIO_Pin_2); // установка выходов 8, 10
в «1» и сброс выходов 0, 2 в «0».
```

Пример_5 (BaseProject_GPIO);

```
#include "1914BA018.h"                  // Device header
#include "1914BA018_gpio.h"             // Keil::Drivers:GPIO

#define GPIOX_Handler GPIOE_Handler
#define GPIOX_IRQn    GPIOE_IRQn
#define GPIOX         GPIOE
#define GPIO_Pin_Key1 GPIO_Pin_10
#define GPIO_Pin_Key2 GPIO_Pin_11
#define GPIO_Pin_Key3 GPIO_Pin_12
#define GPIO_Pin_Led1 GPIO_Pin_0
#define GPIO_Pin_Led2 GPIO_Pin_1
#define GPIO_Pin_Led3 GPIO_Pin_2
#define GPIO_Pin_Led4 GPIO_Pin_3

void GPIOX_Handler(void) // обработчик прерывания
{
  if(GPIO_GetITStatus(GPIOX, GPIO_Pin_Key1) == SET){ //Проверка статуса прерывания
    GPIO_ToggleBits(GPIOX, GPIO_Pin_Led1); //Инвертирование состояния выхода соответствующего пина
    GPIO_ClearITPendingBit(GPIOX, GPIO_Pin_Key1); //Очистка статуса прерывания
  }
  if(GPIO_GetITStatus(GPIOX, GPIO_Pin_Key2) == SET){ //Проверка статуса прерывания
    GPIO_ToggleBits(GPIOX, GPIO_Pin_Led2); //Инвертирование состояния выхода соответствующего пина
    GPIO_ClearITPendingBit(GPIOX, GPIO_Pin_Key2); //Очистка статуса прерывания
  }
  if(GPIO_GetITStatus(GPIOX, GPIO_Pin_Key3) == SET){ //Проверка статуса прерывания
    GPIO_ToggleBits(GPIOX, GPIO_Pin_Led3); //Инвертирование состояния выхода соответствующего пина
    GPIO_ClearITPendingBit(GPIOX, GPIO_Pin_Key3); //Очистка статуса прерывания
  }
}

int main (void)
{
  NVIC_InitTypeDef NVIC_InitStructure;
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
  NVIC_InitStructure.NVIC_IRQChannel = GPIOX_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);

  GPIO_InitTypeDef GPIO_InitStructure;
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_Led1 | GPIO_Pin_Led2 | GPIO_Pin_Led3 | GPIO_Pin_Led4;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_Init(GPIOX, &GPIO_InitStructure);
  GPIO_ITSet(GPIOX, GPIO_Pin_Key1 | GPIO_Pin_Key2 | GPIO_Pin_Key3, GPIO_IT_Type_High,
GPIO_IT_Pol_High);
  GPIO_ITConfig(GPIOX, GPIO_Pin_Key1 | GPIO_Pin_Key2 | GPIO_Pin_Key3, ENABLE);
```

```
while(1)
  {
    for (int i=0;i<100000;i++);
    GPIO_SetBits(GPIOX, GPIO_Pin_Led4);
    for (int i=0;i<100000;i++);
    GPIO_ResetBits(GPIOX, GPIO_Pin_Led4);
  }
}
```

1.2 Библиотека функций последовательных портов UART

№	Функция	Описание
Инициализация и конфигурация		
1	UART_DeInit	Сброс регистров порта UARTx к значениям по умолчанию
2	UART_Init	Инициализация порта UARTx согласно заданным параметрам в UART_InitStruct
3	UART_StructInit	Присвоение членам UART_InitStruct значений по умолчанию
4	UART_Cmd	Изменение активности UARTx порта.
5	UART_ITConfig	Независимая установка 5-ти маскированных прерываний.
Запись и чтение		
6	UART_SendData	Загрузка в буфер передатчика 8 бит данных
7	UART_ReciveData	Чтение из буфера приемника 8 бит данных
8	UART_GetFlagStatus	Чтение состояния контрольного флага
9	UART_ClearFlag	Сброс состояния контрольного флага
10	UART_GetITStatus	Чтение состояния прерывания по событию
11	UART_ClearITPendingBit	Сброс состояния прерывания по событию

1.2.1 Инициализация и конфигурация

Функция:

void UART_DeInit(UART_TypeDef* UARTx)

Сброс регистров порта UARTx к значениям по умолчанию (Работа приемника и передатчика запрещена, все прерывания отключены).

Параметры:

UARTx – где x выбранный порт (1..3);

Возвращаемые значения: нет.

Функция:

void UART_Init(UART_TypeDef* UARTx, UART_InitTypeDef* UART_InitStruct)

Инициализация порта UARTx согласно заданным параметрам в *UART_InitStruct*.

Параметры:

UARTx – где x выбранный порт (1..3);

UART_InitStruct – указатель на *UART_InitTypeDef* структуру, которая содержит конфигурационную информацию для UART порта.

Возвращаемые значения: нет.

Функция:

void UART_StructInit(UART_InitTypeDef* UART_InitStruct)

Присвоение членам *UART_InitStruct* значений по умолчанию:

UART_BaudRate = 9600;

UART_Mode = UART_Mode_Rx | UART_Mode_Tx;

Параметры:

UART_InitStruct – указатель на *UART_InitTypeDef* структуру которая будет инициализирована.

Возвращаемые значения: нет.

Функция:

void UART_Cmd(UART_TypeDef* UARTx, FunctionalState NewState)

Изменение активности UART порта.

Параметры:

UARTx – где x выбранный порт (1..3);

NewState – новое состояние работы порта. Принимаемые значения ENABLE и DISABLE.

Возвращаемые значения: нет.

Функция:

void UART_ITConfig(UART_TypeDef* UARTx, uint32_t UART_IT, FunctionalState NewState)

Независимая установка 5-ти прерываний.

Параметры:

UARTx – где x выбранный порт (1..3);

UART_IT – Маска прерывания. Принимаемые значения маски состояния по событию:

UART_CTRL_TXI – «прерывание от передатчика»;
UART_CTRL_RXI – «прерывание от приемника»;
UART_CTRL_TXOVRI – «переполнение передающего буфера»;
UART_CTRL_RXOVRI – «переполнение приемного буфера»;
UART_CTRL_RXFI – «буфер приемника заполнен полностью»;

NewState – новое состояние маски прерывания. Принимаемые значения ENABLE и DISABLE;

Возвращаемые значения: нет.

1.2.2 Запись и чтение

Функция:

void UART_SendData(UART_TypeDef* UARTx, uint8_t Data)

Загрузка в буфер передатчика 8 бит данных.

Параметры:

UARTx – где x выбранный порт (1..3);

Data – 8 бит данных для загрузки в буфер передатчика.

Возвращаемые значения: нет.

Функция:

uint8_t UART_ReceiveData(UART_TypeDef* UARTx)

Чтение из буфера приемника 8 бит данных.

Параметры:

UARTx – где x выбранный порт (1..3);

Возвращаемые значения: 8 бит данных из буфера приемника.

Функция:

FlagStatus UART_GetFlagStatus(UART_TypeDef* UARTx, uint32_t UART_FLAG)

Чтение состояния контрольного флага порта UARTx.

Параметры:

UARTx – где x выбранный порт (1..3);

UART_FLAG – выбранный флаг, принимаемые значения:

UART_FLAG_TX_BUSY – передатчик занят;
UART_FLAG_RXE – буфер приемника пуст;
UART_FLAG_TXE – буфер передатчика пуст;
UART_FLAG_RXOVR – переполнение приемного буфера;
UART_FLAG_TXOVR – переполнение передающего буфера;
UART_FLAG_RXF – буфер приемника заполнен;
UART_FLAG_TXF – буфер передатчика заполнен;

Возвращаемые значения: Состояние флага UART_FLAG порта UARTx (SET или RESET).

Функция:

void UART_ClearFlag(UART_TypeDef* UARTx, uint32_t UART_FLAG)

Чтение выходных данных порта UARTx.

Параметры:

UARTx – где x выбранный порт (1..3);

UART_FLAG – выбранный флаг, принимаемые значения:

UART_FLAG_TX_BUSY – передатчик занят;
UART_FLAG_RXE – буфер приемника пуст;
UART_FLAG_TXE – буфер передатчика пуст;
UART_FLAG_RXOVR – переполнение приемного буфера;
UART_FLAG_TXOVR – переполнение передающего буфера;
UART_FLAG_RXF – буфер приемника заполнен;
UART_FLAG_TXF – буфер передатчика заполнен;

Возвращаемые значения: нет.

Функция:

ITStatus UART_GetITStatus(UART_TypeDef* UARTx, uint32_t UART_IT)

Чтение состояния прерывания по событию.

Параметры:

UARTx – где x выбранный порт (1..3);

UART_IT – выбранный флаг, принимаемые значения:

UART_IT_TXI – «прерывание от передатчика»

UART_IT_RXI – «прерывание от приемника»
UART_IT_TXOVRI – «переполнение передающего буфера»
UART_IT_RXOVRI – «переполнение приемного буфера»
UART_IT_RXFI – «приемный буфер заполнен полностью»

Возвращаемые значения: Состояние прерывания UART_IT порта UARTx (SET или RESET).

Функция:

void UART_ClearITPendingBit(UART_TypeDef* UARTx, uint32_t UART_IT)

Сброс состояния прерывания по событию.

Параметры:

UARTx – где x выбранный порт (1..3);

UART_IT – выбранный флаг, принимаемые значения:

UART_IT_TXI – «прерывание от передатчика»
UART_IT_RXI – «прерывание от приемника»
UART_IT_TXOVRI – «переполнение передающего буфера»
UART_IT_RXOVRI – «переполнение приемного буфера»
UART_IT_RXFI – «приемный буфер заполнен полностью»

Возвращаемые значения: нет.

1.2.3 Применение библиотеки последовательных портов UART

После сброса все порты UART переходят в состояние DISABLE и конфигурационные данные обнуляются. В микросхеме 3 UART интерфейса. У UART1 буфер приемника и буфер передатчика имеют ширину 8-бит, глубину 1 слово. У UART2-3 буфер приемника и буфер передатчика имеют ширину 8-бит, глубину 32 слова.

Порты настраиваются регистрами, которые инициализируются функцией ***UART_Init()***, в соответствии со значениями структуры ***UART_InitTypeDef***.

UART_BaudRate не более чем (SystemCoreClock/16)

UART_Mode:

UART_Mode_Rx – разрешение работы приемника;
UART_Mode_Tx – разрешение работы передатчика;

После инициализации структуры возможно использование набора функций.

Пример_1

```
// Создаем переменную UART_InitStructure с типом данных UART_InitTypeDef;
UART_InitTypeDef UART_InitStructure;

// Описываем структуру UART_InitStructure;
UART_InitStructure.UART_BaudRate = 9600; // выбираем скорость порта;
UART_InitStructure.UART_Mode = UART_Mode_Tx; // разрешаем работу передатчика;

//Инициализируем структуру UART_InitStructure (название порта, указатель на структуру);
UART_Init(UART1, &UART_InitStructure); //передатчик порта UART1 активирован и настроен на скорость 9600;
```

Пример_2

```
UART_InitTypeDef UART_InitStructure;
UART_InitStructure.UART_BaudRate = 9600; // выбираем скорость порта;
UART_InitStructure.UART_Mode = UART_Mode_Rx | UART_Mode_Tx; // разрешаем работу приемника и передатчика;
UART_Init(UART2, &UART_InitStructure); //приемник и передатчик порта UART2 активирован и настроен на скорость 9600;
```

Пример_3

```
UART_InitTypeDef UART_InitStructure;
UART_InitStructure.UART_BaudRate = 9600
UART_InitStructure.UART_Mode = UART_Mode_Tx;
UART_Init(UART1, &UART_InitStructure);
while(UART_GetFlagStatus(UART1,UART_FLAG_TXF)){};//пока буфер передатчика полон (не произошло чтение данных приемным устройством), ждать.
UART_SendData(UART1,'t'); //передача ASCII кода символа t в буфер передатчика
```

Пример_4

```
UART_InitTypeDef UART_InitStructure;
UART_InitStructure.UART_BaudRate = 9600
UART_InitStructure.UART_Mode = UART_Mode_Rx;
UART_Init(UART1, &UART_InitStructure);
while(UART_GetFlagStatus(UART1,UART_FLAG_RXE)){};//пока буфер приемника пуст (не произошла запись данных передающим устройством), ждать.
       uint8_t  tmp=UART_ReceiveData(UART1);  //чтение 8 бит данных из буфера приемника в переменную tmp
```

Пример_5

```
void UART1_Handler(void) //обработчик прерывания
{
  if(UART_GetITStatus(UART1,UART_IT_RXI) == SET){ //проверка статуса прерывания
    UART_SendChar(UART1,UART_GetChar(UART1));       //отправка в UART байт информации, который считали из буфера RX
    UART_ClearITPendingBit(UART1, UART_IT_RXI); //очистка флага состояния прерывания
  }
}

int main (void)
{
//Создаем переменную NVIC_InitStructure с типом данных NVIC_InitTypeDef для инициализации векторов прерывания(см. описание NVIC)
  NVIC_InitTypeDef NVIC_InitStructure;
```

```
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1); //устанавливаем приоритет
  NVIC_InitStructure.NVIC_IRQChannel = UART1_IRQn; //выбираем канал прерывания, который хотим
активировать
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE; //активируем прерывание
  NVIC_Init(&NVIC_InitStructure); //инициализируем структуру
  //настраиваем скорость работы
  UART_InitTypeDef UART_InitStructure;
  UART_InitStructure.UART_BaudRate = 9600
  UART_InitStructure.UART_Mode = UART_Mode_Rx | UART_Mode_Tx;
  UART_Init(UART1, &UART_InitStructure);
  UART_ITConfig(UART1, UART_CTRL_RXI, ENABLE);
}
```

1.3 Библиотека функций синхронных последовательных портов SPI

№	Функция	Описание
Инициализация и конфигурация		
1	SPI_DeInit	Сброс регистров порта SPIx к значениям по умолчанию
2	SPI_Init	Инициализация порта SPIx согласно заданным параметрам в SPI_InitStruct
3	SPI_StructInit	Присвоение членам SPI_InitStruct значений по умолчанию
4	SPI_Cmd	Изменение активности SPI порта.
5	SPI_DataSizeConfig	Изменение размера пакета в битах
6	SPI_CPHAConfig	Изменение фазы тактового сигнала
7	SPI_CPOLConfig	Изменение уровня активного сигнала SCK
8	SPI_SSOutputCmd	Выбор активной линии или их комбинация
9	SPI_ITConfig	Независимая установка 3-х маскированных прерываний.
Запись и чтение		
10	SPI_SendData	Загрузка в буфер передатчика данных
11	SPI_ReciveData	Чтение из буфера приемника данных
12	SPI_GetFlagStatus	Чтение состояния контрольного флага
13	SPI_GetITStatus	Чтение состояния прерывания по событию
14	SPI_ClearITPendingBit	Сброс состояния прерывания по событию
Дополнительные функции		
15	SPI_BaudRatePrescalerInit	Настройка тактовой частоты SPI

1.3.1 Инициализация и конфигурация

Функция:

void SPI_DeInit(SPI_TypeDef* SPIx)

Сброс регистров порта SPIx к значениям по умолчанию (Работа приемника и передатчика запрещена, все прерывания отключены).

Параметры:

SPIx – где x выбранный порт (1..2);

Возвращаемые значения: нет.

Функция:

void SPI_Init(SPI_TypeDef* SPIx, SPI_InitTypeDef* SPI_InitStruct)

Инициализация порта SPIx согласно заданным параметрам в *SPI_InitStruct*.

Параметры:

SPIx – где x выбранный порт (1..2);

SPI_InitStruct – указатель на *SPI_InitTypeDef* структуру которая содержит конфигурационную информацию для SPI порта.

Возвращаемые значения: нет.

Функция:

void SPI_StructInit(SPI_InitTypeDef* SPI_InitStruct)

Присвоение членам *SPI_InitStruct* значений по умолчанию:

SPI_Direction = SPI_Direction_2Lines_FullDuplex;

SPI_Mode = SPI_Mode_Slave;

SPI_DataSize = SPI_DataSize_8b;

SPI_CPOL = SPI_CPOL_Low;

SPI_CPHA = SPI_CPHA_1Edge;

SPI_NSS = SPI_NSS_Soft;

SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_128;

SPI_FirstBit = SPI_FirstBit_MSB;

Параметры:

SPI_InitStruct – указатель на *SPI_InitTypeDef* структуру которая будет инициализирована.

Возвращаемые значения: нет.

Функция:

void SPI_Cmd(SPI_TypeDef* SPIx, FunctionalState NewState)

Изменение активности SPI порта.

Параметры:

SPIx – где x выбранный порт (1..2);

NewState – новое состояние работы порта. Принимаемые значения ENABLE и DISABLE.

Возвращаемые значения: нет.

Функция:

void SPI_DataSizeConfig(SPI_TypeDef* SPIx, uint32_t SPI_DataSize)

Изменение размера пакета в битах.

Параметры:

SPIx – где x выбранный порт (1..2);

SPI_DataSize – новый размер пакета.

Принимаемые значения:

SPI_DATASIZE_4	4 бита
SPI_DATASIZE_5	5 бит
SPI_DATASIZE_6	6 бит
SPI_DATASIZE_7	7 бит
SPI_DATASIZE_8	8 бит
SPI_DATASIZE_9	9 бит
SPI_DATASIZE_10	10 бит
SPI_DATASIZE_11	11 бит
SPI_DATASIZE_12	12 бит
SPI_DATASIZE_13	13 бит

SPI_DATASIZE_14 14 бит
SPI_DATASIZE_15 15 бит
SPI_DATASIZE_16 16 бит

Возвращаемые значения: нет.

Функция:

void SPI_CPHAConfig(SPI_TypeDef* SPIx, uint32_t SPI_CPHA)

Изменение фазы тактового сигнала.

Параметры:

SPIx – где x выбранный порт (1..2);

SPI_CPHA – новая фаза тактового сигнала.

Принимаемые значения:

SPI_CPHA_1Edge – выборка данных осуществляется по фронту тактового сигнала;

SPI_CPHA_2Edge – выборка данных осуществляется по спаду тактового сигнала;

Возвращаемые значения: нет.

Функция:

void SPI_CPOLConfig(SPI_TypeDef* SPIx, uint32_t SPI_CPOL)

Изменение уровня активного сигнала SCK.

Параметры:

SPIx – где x выбранный порт (1..2);

SPI_CPOL – новый уровень активного сигнала SCK.

Принимаемые значения:

SPI_CPOL_Low – режим обратной полярности такового сигнала (управляющий 0);

SPI_CPOL_High – режим прямой полярности такового сигнала (управляющий 1);

Возвращаемые значения: нет.

Функция:

void SPI_SSOutputCmd(SPI_TypeDef* SPIx, uint32_t SPI_SSOutput, FunctionalState NewState)

Выбор активной линии или их комбинация.

Параметры:

SPIx – где x выбранный порт (1..2);

SPI_SSOutput – Выбор линии.

Принимаемые значения:

SPI_SSOutput_Set_0
SPI_SSOutput_Set_1
SPI_SSOutput_Set_2
SPI_SSOutput_Set_3

SPI_SSOutput_Set_All

NewState – новое состояние работы линии или их комбинаций. Принимаемые значения ENABLE и DISABLE.

Возвращаемые значения: нет.

Функция:

void SPI_ITConfig(SPI_TypeDef* SPIx, uint32_t SPI_IT, FunctionalState NewState)

Независимая установка 3-х маскированных прерываний.

Параметры:

SPIx – где x выбранный порт (1..2);

SPI_IT – Маска прерывания. Принимаемые значения маски состояния по событию:

SPI_IMSC_TXIM – «прерывание от передатчика»;

SPI_IMSC_RXIM – «прерывание от приемника»;

SPI_IMSC_RORIM – «переполнение приемного буфера»;

NewState – новое состояние маски прерывания. Принимаемые значения ENABLE и DISABLE;

Возвращаемые значения: нет.

1.3.2 Запись и чтение

Функция:

void SPI_SendData(SPI_TypeDef* SPIx, uint16_t Data)

Загрузка в буфер передатчика 16 бит данных.

Параметры:

SPIx – где x выбранный порт (1..2);

Data – 16 бит данных для загрузки в буфер передатчика.

Возвращаемые значения: нет.

Функция:

uint16_t SPI_ReceiveData(SPI_TypeDef* SPIx)

Чтение из буфера приемника 16 бит данных.

Параметры:

SPIx – где x выбранный порт (1..2);

Возвращаемые значения: 16 бит данных из буфера приемника.

Функция:

FlagStatus SPI_GetFlagStatus(SPI_TypeDef* SPIx, uint32_t SPI_FLAG)

Чтение состояния контрольного флага порта SPIx.

Параметры:

SPIx – где x выбранный порт (1..2);

SPI_FLAG – выбранный флаг, принимаемые значения:

SPI_FLAG_BUSY – интерфейс занят;

SPI_FLAG_RFF – буфер приемника заполнен;

SPI_FLAG_RNE – буфер приемника пуст;

SPI_FLAG_TNF – буфер передатчика не заполнен;

SPI_FLAG_TNE – буфер передатчика пуст;

Возвращаемые значения: Состояние флага SPI_FLAG порта SPIx (SET или RESET).

Функция:

ITStatus SPI_GetITStatus(SPI_TypeDef* SPIx, uint32_t SPI_IT)

Чтение состояния прерывания по событию.

Параметры:

SPIx – где x выбранный порт (1..2);

SPI_IT – выбранный флаг, принимаемые значения:

SPI_IMSC_TXIM – «прерывание от передатчика»;

SPI_IMSC_RXIM – «прерывание от приемника»;

SPI_IMSC_RORIM – «переполнение приемного буфера»;

Возвращаемые значения: Состояние прерывания SPI_IT порта SPIx (SET или RESET).

Функция:

void SPI_ClearITPendingBit(SPI_TypeDef* SPIx, uint32_t SPI_IT)

Сброс состояния прерывания по событию.

Параметры:

SPIx – где x выбранный порт (1..2);

SPI_IT – выбранный флаг, принимаемые значения:

SPI_IMSC_TXIM – «прерывание от передатчика»;

SPI_IMSC_RXIM – «прерывание от приемника»;

SPI_IMSC_RORIM – «переполнение приемного буфера»;

Возвращаемые значения: нет.

1.3.3 Дополнительные функции

Функция:

uint32_t SPI_BaudRatePrescalerInit(uint8_t HighBaudRate, uint8_t LowBaudRate)

Настройка тактовой частоты SPI

Параметры:

HighBaudRate – Длительность высокого уровня CLK в тактах системной частоты, принимаемые значения:

от 1 до 255;

LowBaudRate – Длительность низкого уровня CLK в тактах системной частоты, принимаемые значения:

от 1 до 255;

Возвращаемые значения: Значение, необходимое для записи в SPI_BaudRatePrescaler.

1.3.4 Применение библиотеки последовательных портов SPI

После сброса все порты SPI переходят в состояние DISABLE и конфигурационные данные обнуляются. В микросхеме 2 SPI интерфейса. У SPI буфер приемника и буфер передатчика имеют ширину от 4 до 16-бит, глубину 8 слов.

Порты настраиваются регистрами, которые инициализируются функцией ***SPI_Init()***, в соответствии со значениями структуры ***SPI_InitTypeDef***.

SPI_Direction:

SPI_Direction_2Lines_FullDuplex – Решим полнодуплексной передачи

SPI_Mode:

SPI_Mode_Master – SPI режим master;
SPI_Mode_Slave – SPI режим slave;

SPI_DataSize:

SPI_DATASIZE_4 - 4 бита
SPI_DATASIZE_5 - 5 бит
SPI_DATASIZE_6 - 6 бит
SPI_DATASIZE_7 - 7 бит
SPI_DATASIZE_8 - 8 бит
SPI_DATASIZE_9 - 9 бит
SPI_DATASIZE_10 - 10 бит
SPI_DATASIZE_11 - 11 бит
SPI_DATASIZE_12 - 12 бит
SPI_DATASIZE_13 - 13 бит
SPI_DATASIZE_14 - 14 бит
SPI_DATASIZE_15 - 15 бит
SPI_DATASIZE_16 - 16 бит

SPI_ CPOL:

SPI_CPOL_Low – режим обратной полярности такового сигнала (управляющий 0);
SPI_CPOL_High – режим прямой полярности такового сигнала (управляющий 1);

SPI_CPHA:

SPI_CPHA_1Edge – выборка данных осуществляется по фронту тактового сигнала;
SPI_CPHA_2Edge – выборка данных осуществляется по спаду тактового сигнала;

SPI_NSS:

SPI_NSS_Set – переключение сигнала между посылками слов активно**;**
SPI_NSS_Reset – переключение сигнала между посылками слов отключено**;**

SPI_BaudRatePrescaler:

SPI_BaudRatePrescaler_2
SPI_BaudRatePrescaler_4

SPI_BaudRatePrescaler_8
SPI_BaudRatePrescaler_16
SPI_BaudRatePrescaler_32
SPI_BaudRatePrescaler_64
SPI_BaudRatePrescaler_128
SPI_BaudRatePrescaler_256

SPI_FirstBit:

SPI_FirstBit_MSB – первым передается старший бит;

После инициализации структуры возможно использование набора функций.

Пример_1

```
// Создаем переменную SPI_InitStructure с типом данных SPI_InitTypeDef
SPI_InitTypeDef SPI_InitStructure;
// Описываем структуру SPI_InitStructure
SPI_InitStructure.SPI_Direction=SPI_Direction_2Lines_FullDuplex;//режим работы SPI
SPI_InitStructure.SPI_Mode=SPI_Mode_Master; //настройка SPI в режиме ведущего
SPI_InitStructure.SPI_DataSize=SPI_DataSize_8b;//размер пакета 8бит
SPI_InitStructure.SPI_CPOL=SPI_CPOL_High;//режим прямой полярности такового сигнала
SPI_InitStructure.SPI_CPHA=SPI_CPHA_2Edge;// выборка данных осуществляется по спаду тактового
сигнала
SPI_InitStructure.SPI_NSS=SPI_NSS_Reset;//переключение SS между посылками отключено
SPI_InitStructure.SPI_BaudRatePrescaler=SPI_BaudRatePrescaler_64;//установка делителя системной
частоты.

//Инициализируем структуру SPI_InitStructure (название порта, указатель на структуру);
SPI_Init(SPI1, &SPI_InitStructure);
SPI_Cmd(SPI1,ENABLE);//Разрешаем работу приемопередатчика
SPI_SSOutputCmd(SPI1, SPI_SSOutput_Set_0, ENABLE);

while(SPI_GetFlagStatus(SPI1,SPI_FLAG_BUSY)==SET){};//ожидание прекращения операций модулем
SPI_SendData(SPI1,0x55);//запись в передающий буфер 8 бит данных
while(SPI_GetFlagStatus(SPI1,SPI_FLAG_BUSY)==SET){}; //ожидание конца передачи
uint16_t tmp=SPI_ReceiveData(SPI1);//чтение из приемного буфера 8 бит данных в переменную tmp
```

Пример_2

```
void SPI1_Handler(void)
{
  if(SPI_GetITStatus(SPI1,SPI_INT_RXIS) == SET){ //проверка статуса прерывания
    uint8_t data= SPI_ReceiveData(SPI1);//Чтение данных из буфера приемника в переменную data
    SPI_ClearITPendingBit(SPI1, SPI_INT_RXIS); //очистка флага состояния прерывания
  }
}
int main (void)
{
  //Создаем переменную NVIC_InitStructure с типом данных NVIC_InitTypeDef для инициализации
векторов прерывания(см. описание NVIC)
  NVIC_InitTypeDef NVIC_InitStructure;
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1); //устанавливаем приоритет
  NVIC_InitStructure.NVIC_IRQChannel = SPI1_IRQn; //выбираем канал прерывания, который хотим
активировать
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
```

```
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE; //активируем прерывание
  NVIC_Init(&NVIC_InitStructure); //инициализируем структуру

  //настройка SPI1
  SPI_DeInit(SPI1);
  SPI_InitTypeDef SPI_InitStructure;
  SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
  SPI_InitStructure.SPI_DataSize = SPI_DataSize_16b;
  SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;
  SPI_InitStructure.SPI_CPHA = SPI_CPHA_2Edge;
  SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
  SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_128;
  SPI_InitStructure.SPI_Mode = SPI_Mode_Master;
  SPI_Init(SPI1, &SPI_InitStructure);
  SPI_ITConfig(SPI1,SPI_INT_RXIS,ENABLE); //Разрешение прерывания по событию SPI_INT_RXIS
  SPI_Cmd(SPI1, ENABLE);
  SPI_SSOutputCmd(SPI1, SPI_SSOutput_Set_0, ENABLE);
}
```

1.4 Библиотека таймеров

№	Функция	Описание
Инициализация и конфигурация		
1	TIM_DeInit	Сброс регистров TIMx к значениям по умолчанию
2	TIM_Init	Инициализация таймера TIMx согласно заданным параметрам в TIM_InitStruct
3	TIM_Cmd	Изменение активности таймера TIMx
4	TIM_ITConfig	Разрешение прерывания
Запись и чтение		
5	TIM_SetAutoreload	Установка периода перезапуска таймера
6	TIM_SetCounter	Установка текущего значения таймера
7	TIM_GetCounter	Чтение текущего значения таймера
8	TIM_GetITStatus	Чтение состояния прерывания
9	TIM_ClearITPendingBit	Сброс состояния прерывания

1.4.1 Инициализация и конфигурация

Функция:

void TIM_DeInit(TIM_TypeDef* TIMx)

Сброс регистров TIMx к значениям по умолчанию.

Параметры:

TIMx – где x выбранный таймер (1..3);

Возвращаемые значения: нет.

Функция:

void TIM_Init(TIM_TypeDef* TIMx, TIM_InitTypeDef* TIM_InitStruct)

Инициализация таймера TIMx согласно заданным параметрам в TIM_InitStruct**.**

Параметры:

TIMx – где x выбранный таймер (1..3);

TIM_InitStruct – указатель на *TIM_InitTypeDef* структуру которая содержит конфигурационную информацию для таймера.

Возвращаемые значения: нет.

Функция:

void TIM_Cmd(TIM_TypeDef* TIMx, FunctionalState NewState)

Изменение активности таймера.

Параметры:

TIMx – где x выбранный таймер (1..3);

NewState – новое состояние работы таймера. Принимаемые значения ENABLE и DISABLE.

Возвращаемые значения: нет.

Функция:

void TIM_ITConfig(TIM_TypeDef* TIMx, FunctionalState NewState)

Разрешение прерывания**.**

Параметры:

TIMx – где x выбранный таймер (1..3);

NewState – новое состояние разрешения прерывания. Принимаемые значения ENABLE и DISABLE.

Возвращаемые значения: нет.

1.4.2 Запись и чтение

Функция:

void TIM_SetAutoreload(TIM_TypeDef* TIMx, uint32_t Autoreload)

Установка периода перезапуска таймера.

Параметры:

TIMx – где x выбранный таймер (1..3);

Autoreload – период перезапуска таймера. Принимаемые значения: 0 – 4294967295;

Возвращаемые значения: нет.

Функция:

void TIM_SetCounter(TIM_TypeDef* TIMx, uint32_t Counter)

Установка текущего значения таймера.

Параметры:

TIMx – где x выбранный таймер (1..3);

Counter – новое текущее значение таймера. Принимаемые значения: 0 – 4294967295;

Возвращаемые значения: нет.

Функция:

uint32_t TIM_GetCounter(TIM_TypeDef* TIMx)

Чтение текущего значения таймера.

Параметры:

TIMx – где x выбранный таймер (1..3);

Возвращаемые значения: текущее значение таймера 0 – 4294967295.

Функция:

ITStatus TIM_GetITStatus(TIM_TypeDef* TIMx)

Чтение состояния прерывания.

Параметры:

TIMx – где x выбранный таймер (1..3);

Возвращаемые значения: Состояние прерывания таймера TIMx (SET или RESET).

Функция:

void TIM_ClearITPendingBit(TIM_TypeDef* TIMx)

Сброс состояния прерывания.

Параметры:

TIMx – где x выбранный таймер (1..3);

Возвращаемые значения: нет.

1.4.3 Применение библиотеки таймеров

После сброса все таймеры TIM переходят в состояние DISABLE и конфигурационные данные обнуляются. В микросхеме 3 TIM таймера.

Таймеры настраиваются регистрами, которые инициализируются функцией ***TIM_Init()***, в соответствии со значениями структуры ***TIM_InitTypeDef***.

TIM_ExtClk:

TIM_EXT_CLK_EN – разрешить тактирование от TMRx_EXTIN;

TIM_EXT_CLK_DIS – запретить тактирование от TMRx_EXTIN;

TIM_ExtEn:

TIM_EXT_EN – использовать перепад на входе TMRx_EXTIN в качестве сигнала разрешения для работы таймера;

TIM_EXT_DIS – не использовать вход TMRx_EXTIN как разрешающий вход;

После инициализации структуры возможно использование набора функций.

Пример_1

```
void TIM1_Handler(void)
{
  if(TIM_GetITStatus(TIM1) == SET){//Проверяем флаг состояния прерывания
    GPIO_ToggleBits(GPIOE, GPIO_Pin_2);//инвертируем состояние пина 2 порта GPIOE (см. Библиотеку портов GPIO)
    TIM_ClearITPendingBit(TIM1);//Очищаем флаг события прерывания
  }
}
int main (void)
{
  //Создаем переменную NVIC_InitStructure с типом данных NVIC_InitTypeDef для инициализации векторов прерывания(см. описание NVIC)
  NVIC_InitTypeDef NVIC_InitStructure;
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1); //устанавливаем приоритет
  NVIC_InitStructure.NVIC_IRQChannel = TIM1_IRQn; //выбираем канал прерывания, который хотим активировать
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE; //активируем прерывание
  NVIC_Init(&NVIC_InitStructure); //инициализируем структуру

  TIM_DeInit(TIM1) //Сброс всех настроек таймера TIM
  TIM_InitTypeDef TIM_InitStructure; //Создаем переменную TIM_InitStructure с типом данных TIM_InitTypeDef
  //Описываем структуру TIM_StructInit1
  TIM_StructInit1.TIM_ExtClk = TIM_EXT_CLK_DIS; //Выбираем тактирование от внутренней частоты
  TIM_StructInit1.TIM_ExtEn = TIM_EXT_DIS; //Отключаем использование TMRx_EXTIN в качестве сигнала разрешения работы таймера
```

```
  TIM_Init(SPI1, &TIM_InitStructure); //Инициализируем структуру TIM_InitStructure (название
порта, указатель на структуру)
  TIM_ITConfig(TIM1, ENABLE); //Разрешение прерывания
  TIM_SetAutoreload(TIM1,0x10000); //Устанавливаем период перезагрузки таймера на 65536 тактов
  TIM_Cmd(TIM1, ENABLE); //Разрешаем работу таймера
}
```

1.5 Библиотека контроллера прямого доступа к памяти DMA

№	Функция	Описание
Инициализация и конфигурация		
1	DMA_DeInit	Сброс регистров канала DMA к значениям по умолчанию
2	DMA_Master_Cmd	Изменение глобального разрешения работы контроллера DMA
3	DMA_Init	Инициализация канала DMA согласно заданным параметрам в DMA_InitStruct
4	DMA_GetMode	Чтение установленного подканала для выбранного канала DMA
5	DMA_SetMode	Установка подканала для выбранного канала DMA
6	DMA_GetPriority	Чтение приоритета выбранного канала DMA
7	DMA_SetPriority	Установка приоритета выбранному каналу DMA
8	DMA_Cmd	Изменение активности канала DMA
Запись и чтение		
9	DMA_SW	Программный запрос на обработку канала DMA
10	DMA_GetStatus	Чтение состояния контроллера DMA
11	DMA_GetMasterCmdStatus	Чтение состояния глобального разрешения работы контроллера DMA
12	DMA_GetCmdStatus	Чтение состояния активности выбранного канала DMA
13	DMA_GetErrStatus	Чтение флага ошибки выбранного канала DMA
14	DMA_ClearErrPendingBit	Сброс флага ошибки выбранного канала DMA

1.5.1 Инициализация и конфигурация

Функция:

void DMA_DeInit(DMA_Stream_TypeDef* DMA_Periph)

Сброс регистров субканала DMA_Periph к значениям по умолчанию.

Параметры:

DMA_Periph – выбранный субканал DMA_PRIMARY_STREAMx/ DMA_ALTERNATE_STREAMx, где x – 1..32;

Возвращаемые значения: нет.

Функция:

void DMA_Master_Cmd(FunctionalState NewState)

Изменение глобального разрешения работы контроллера DMA**.**

Параметры:

NewState – новое состояние работы таймера. Принимаемые значения ENABLE и DISABLE.

Возвращаемые значения: нет.

Функция:

void DMA_Init(DMA_Stream_TypeDef* DMA_Periph, DMA_InitTypeDef* DMA_InitStruct)

Инициализация канала DMA согласно заданным параметрам в DMA_InitStruct.

Параметры:

DMA_Periph – выбранный субканал DMA_PRIMARY_STREAMx/ DMA_ALTERNATE_STREAMx, где x – 1..32;

DMA_InitStruct – Указатель на *DMA_InitTypeDef* структуру, которая содержит конфигурационную информацию для субканала.

Возвращаемые значения: нет.

Функция:

uint8_t DMA_GetMode(uint32_t DMA_Stream)

Чтение установленного субканала для выбранного канала DMA.

Параметры:

DMA_Stream **–** выбранный канал DMA. Принимаемые значения: DMA_STREAMx, где x - номер канала 1..32;

Возвращаемые значения: установленный субканал. *DMA_Alternate* или *DMA_Primary*.

Функция:

void DMA_SetMode(uint32_t DMA_Stream, uint8_t Mode)

Установка подканала для выбранного канала DMA.

Параметры:

DMA_Stream **–** выбранный канал DMA. Принимаемые значения: DMA_STREAMx, где x - номер канала 1..32;

Mode **-** субканал. Принимаемые значения: *DMA_Alternate* или *DMA_Primary*.

Возвращаемые значения: нет.

Функция:

uint8_t DMA_GetPriority(uint32_t DMA_Stream)

Чтение приоритета выбранного канала DMA.

Параметры:

DMA_Stream **–** выбранный канал DMA. Принимаемые значения: DMA_STREAMx, где x - номер канала 1..32;

Возвращаемые значения: Приоритет: *DMA_Priority_Low* или *DMA_Priority_High*;

Функция:

void DMA_SetPriority(uint32_t DMA_Stream, uint8_t Priority)

Установка приоритета выбранному каналу DMA.

Параметры:

DMA_Stream **–** выбранный канал DMA. Принимаемые значения: DMA_STREAMx, где x - номер канала 1..32;

Priority **–** выбор приоритета. Принимаемые значения *DMA_Priority_Low* и *DMA_Priority_High;*

Возвращаемые значения: нет.

Функция:

void DMA_Cmd(uint32_t DMA_Stream, FunctionalState NewState)

Изменение активности канала DMA.

Параметры:

DMA_Stream **–** выбранный канал DMA. Принимаемые значения: DMA_STREAMx, где x - номер канала 1..32;

NewState **-** новое состояние работы таймера. Принимаемые значения ENABLE и DISABLE.

Возвращаемые значения: нет.

1.5.2 Запись и чтение

Функция:

void DMA_SW(uint32_t DMA_Stream)

Программный запрос на обработку канала DMA.

Параметры:

DMA_Stream **–** выбранный канал DMA. Принимаемые значения: DMA_STREAMx, где x - номер канала 1..32;

Возвращаемые значения: нет.

Функция:

uint8_t DMA_GetStatus(void)

Чтение состояния контроллера DMA.

Параметры: нет

Возвращаемые значения: состояние контроллера.

b0000 – в покое;
b0001 – чтение управляющих данных канала;
b0010 – чтение указателя конца данных источника;
b0011 – чтение указателя конца данных приемника;
b0100 – чтение данных источника;
b0101 – запись данных в приемник;
b0110 – ожидание запроса на выполнение DMA;
b0111 – запись управляющих данных канала;
b1000 – приостановлен;
b1001 – выполнен;
b1010 – режим работы с периферией «Исполнение с изменением конфигурации»;
b1011-b1111 – не определено

Функция:

FunctionalState DMA_GetMasterCmdStatus(void)

Чтение состояния глобального разрешения работы контроллера DMA.

Параметры: нет
Возвращаемые значения: ENABLE или DISABLE.

Функция:

FunctionalState DMA_GetCmdStatus(uint32_t DMA_Stream)

Чтение состояния активности выбранного канала DMA.

Параметры:

DMA_Stream **–** выбранный канал DMA. Принимаемые значения: DMA_STREAMx, где x - номер канала 1..32;

Возвращаемые значения: ENABLE или DISABLE.

Функция:

FlagStatus DMA_GetErrStatus(uint32_t DMA_Stream)

Чтение флага ошибки выбранного канала DMA.

Параметры:

DMA_Stream **–** выбранный канал DMA. Принимаемые значения: DMA_STREAMx, где x - номер канала 1..32;

Возвращаемые значения: SET или RESET.

Функция:

void DMA_ClearErrPendingBit(uint32_t DMA_Stream)

Сброс флага ошибки выбранного канала DMA.

Параметры:

DMA_Stream **–** выбранный канал DMA. Принимаемые значения: DMA_STREAMx, где x - номер канала 1..32;

Возвращаемые значения: нет.

1.5.3 Применение библиотеки ПДП

После сброса все каналы DMA переходят в состояние DISABLE и конфигурационные данные обнуляются. В микросхеме 32 канала ПДП, каждый из который имеет 2 конфигурационных субканала Primary и Alternate.

Субканалы настраиваются регистрами, которые инициализируются функцией ***DMA_Init()***, в соответствии со значениями структуры ***DMA_InitTypeDef***.

DMA_DstInc: Шаг инкремента адреса приемника

DMA_Inc_8b – байт;
DMA_Inc_16b – полуслово;
DMA_Inc_32b – слово;
DMA_Inc_No – нет инкремента. Адрес равен DMA_DestEndPointer;

DMA_SrcInc: Шаг инкремента адреса источника

DMA_Inc_8b – байт;
DMA_Inc_16b – полуслово;
DMA_Inc_32b – слово;

DMA_Inc_No – нет инкремента. Адрес равен DMA_SourceEndPointer;

DMA_Size: Размерность данных

DMA_Size_8b – байт;

DMA_Size_16b – полуслово;

DMA_Size_32b – слово;

DMA_Power:

DMA_Power_1 – Арбитраж производится после каждой передачи DMA;

DMA_Power_2 – Арбитраж производится после 2 передач DMA;

DMA_Power_4 – Арбитраж производится после 4 передач DMA;

DMA_Power_8 – Арбитраж производится после 8 передач DMA;

DMA_Power_16 – Арбитраж производится после 16 передач DMA;

DMA_Power_32 – Арбитраж производится после 32 передач DMA;

DMA_Power_64 – Арбитраж производится после 64 передач DMA;

DMA_Power_128 – Арбитраж производится после 128 передач DMA;

DMA_Power_256 – Арбитраж производится после 256 передач DMA;

DMA_Power_512 – Арбитраж производится после 512 передач DMA;

DMA_Power_1024 – Арбитраж производится после 1024 передач DMA. Это значит, что арбитраж не производится, т.к. максимальное количество передач DMA равно 1024;

DMA_Transmit: Общее число передач DMA **минус 1**

0-1023;

DMA_CycleCtrl:

DMA_DIR_Stop – Стоп. Означает, что структура управляющих данных является «неправильной»

DMA_DIR_Main – Основной. Контроллер должен получить новый запрос для выполнения цикла DMA, перед этим он должен выполнить процедуру арбитража.

DMA_DIR_Auto – Авто-запрос. Контроллер автоматически осуществляет запрос на обработку по соответствующему каналу в течение процедуры арбитража.

DMA_DIR_PingPong – Контроллер выполняет цикл DMA использую одну из структур управляющих данных. По окончании выполнения цикла DMA, контроллер выполняет следующий цикл DMA, используя другую структуру. Контроллер сигнализирует об окончании каждого цикла DMA, позволяя процессору перенастраивать неактивную структуру данных. Контроллер продолжает выполнять циклы DMA, до тех пор, пока он не прочитает «неправильную» структуру данных или пока процессор не изменит cycle_ctrl поле в состояние b001 или b 010;

DMA_DIR_MemREP - Режим работы с памятью «Исполнение с изменением конфигурации». При работе контроллера в данном режиме значение этого поля в первичной структуре управляющих данных должно быть b100;

DMA_DIR_MemREA - Режим работы с памятью «Исполнение с изменением конфигурации». При работе контроллера в данном режиме значение этого поля в альтернативной структуре управляющих данных должно быть b101;

DMA_DestEndPointer: Указатель последнего адреса данных приемника

DMA_SourceEndPointer: Указатель последнего адреса данных источника

После инициализации структуры возможно использование набора функций.

Пример_1;

```
int main (void)
{
  //Инициализируем два одномерных массива.
  uint32_t a[] = {2,3,4,5,6,7,8,9};
  uint32_t b[] = {0,0,0,0,0,0,0,0};

  DMA_Master_Cmd(ENABLE); //Разрешаем работу контроллера ПДП
  DMA_DeInit(DMA_PRIMARY_STREAM1); //Сбрасываем настройки первого основного канала
  DMA_InitTypeDef DMA_InitStructure; //Создаем структуру для настройки канала
  DMA_InitStructure.DMA_DstInc = DMA_Inc_32b; //Размерность инкремента данных приемника
  DMA_InitStructure.DMA_SrcInc = DMA_Inc_32b; //Размерность инкремента данных источника
  DMA_InitStructure.DMA_Size = DMA_Size_32b; //Размерность данных
  DMA_InitStructure.DMA_Power = 0x0; //Арбитраж после каждой передачи
  DMA_InitStructure.DMA_Transmit = 0x7; //Количество передач 8
  DMA_InitStructure.DMA_CycleCtrl = DMA_DIR_Auto; //Режим автозапроса
  DMA_InitStructure.DMA_SourceEndPointer = &a[7]; //Указатель конечного адреса источника
  DMA_InitStructure.DMA_DestEndPointer=&b[7]; //Указатель конечного адреса приемника
  DMA_Init(DMA_PRIMARY_STREAM1, &DMA_InitStructure); //Инициализация первого основного канала

  DMA_SetMode(DMA_STREAM1, DMA_Primary); //Выбор основных настроек канала для первого канала DMA
  DMA_SetPriority(DMA_STREAM1,DMA_Priority_Low); //Выбор низкого приоритета для первого канала
  DMA_Cmd(DMA_STREAM1,ENABLE); //Разрешение работы первого канала DMA

  DMA_SW(DMA_STREAM1); //Один запрос на передачу данных. Его достаточно для автоматического
режима
}
```

1.6 Библиотека функций интерфейса по ГОСТ 52070

№	Функция	Описание
Инициализация и конфигурация		
1	MIL_STD_1553_DeInit	Сброс регистров интерфейса к значениям по умолчанию
2	MIL_STD_1553_Init	Инициализация интерфейса согласно заданным параметрам в MIL_STD_1553_InitStruct
3	MIL_STD_1553_TimeInit	Присвоение членам MIL_STD_1553_InitStruct значений временных параметров по умолчанию
4	MIL_STD_1553_StructInit	Присвоение членам MIL_STD_1553_InitStruct значений по умолчанию
5	MIL_STD_1553_ITConfig	Настройка события для активации прерывания и разрешение установки бита MSG_OK для соответствующей команды управления
6	MIL_STD_1553_Transmitter_CMD	Выбор активного канала для передачи данных
Запись и чтение		
7	MIL_STD_1553_GetDebugDec	Чтение временных параметров последнего приема данных
8	MIL_STD_1553_GetCommandWord	Чтение командного слова (КШ, МШ)
9	MIL_STD_1553_SetCommandWord	Установка командного слова (КШ)
10	MIL_STD_1553_GetModeData	Чтение слова данных, принятое по команде управления (ОУ)
11	MIL_STD_1553_SetModeData	Запись слова данных, передаваемое по команде управления (ОУ)
12	MIL_STD_1553_GetMsg	Чтение кода команды или количества принятых/переданных сообщений
13	MIL_STD_1553_SetMsgType	Установка формата сообщения (КШ)
14	MIL_STD_1553_SetMsgInt	Количество принятых сообщений, после которого будет установлен MSG_OK
15	MIL_STD_1553_SetStatusWord	Установка статусного слова
16	MIL_STD_1553_GetStatusWord	Чтение статусного слова
17	MIL_STD_1553_GetStatus	Чтение статусного регистра
18	MIL_STD_1553_GetFlagStatus	Чтение состояния флага выбранного события
19	MIL_STD_1553_GetMsgSubAddr	Чтение подадреса в принятом сообщении
20	MIL_STD_1553_ClearStatus	Очистка статусного регистра
21	MIL_STD_1553_ClearFlagStatus	Очистка состояния флага выбранного

		события
22	MIL_STD_1553_StartTransmision	Функция старта передачи
23	MIL_STD_1553_ReceiveDataFromBuffer	Чтение данных из буфера приемника
24	MIL_STD_1553_WiteDataToSendBuffer	Запись данных в буфер передатчика

1.6.1 Инициализация и конфигурация

Функция:

void MIL_STD_1553_DeInit(MIL_STD_1553_TypeDef * MIL_STD_1553x)

Сброс регистров интерфейса к значениям по умолчанию.

Параметры:

MIL_STD_1553x – где x выбранный интерфейс (1..2);

Возвращаемые значения: нет.

Функция:

void MIL_STD_1553_Init(MIL_STD_1553_TypeDef * MIL_STD_1553x, MIL_STD_1553_InitTypeDef * MIL_STD_1553_InitStruct)

Инициализация интерфейса согласно заданным параметрам в MIL_STD_1553_InitStruct.

Параметры:

MIL_STD_1553x – где x выбранный интерфейс (1..2);

MIL_STD_1553_InitStruct – указатель на *MIL_STD_1553_InitTypeDef* структуру которая содержит конфигурационную информацию.

Возвращаемые значения: нет.

Функция:

void MIL_STD_1553_TimeInit(MIL_STD_1553_InitTypeDef * MIL_STD_1553_InitStruct)

Присвоение членам MIL_STD_1553_InitStruct значений временных параметров по умолчанию**.**

MIL_STD_1553_HalfBit = SystemCoreClock/2000000L;
MIL_STD_1553_Pause = 10 * (SystemCoreClock/2000000L);
MIL_STD_1553_NoGap = 2 * (SystemCoreClock/2000000L);
MIL_STD_1553_NoWord = 24 * (SystemCoreClock/2000000L);
MIL_STD_1553_JtrNbMax = (SystemCoreClock/2000000L)/2;
MIL_STD_1553_JtrNbMin = (SystemCoreClock/2000000L)/2;
MIL_STD_1553_Jtr1bMax = (SystemCoreClock/2000000L)/2;
MIL_STD_1553_Jtr1bMin = (SystemCoreClock/2000000L)/2;
MIL_STD_1553_JtrNsMax = 2 * (SystemCoreClock/2000000L);
MIL_STD_1553_JtrNsMin = 2 * (SystemCoreClock/2000000L);
MIL_STD_1553_Jtr1sMax = 2 * (SystemCoreClock/2000000L);
MIL_STD_1553_Jtr1sMin = 2 * (SystemCoreClock/2000000L);

Параметры:

MIL_STD_1553_InitStruct – указатель на *MIL_STD_1553_InitTypeDef* структуру.

Возвращаемые значения: нет.

Функция:

void MIL_STD_1553_StructInit(MIL_STD_1553_InitTypeDef * MIL_STD_1553_InitStruct)

Присвоение членам MIL_STD_1553_InitStruct значений по умолчанию**.**

MIL_STD_1553_Mode = MIL_STD_1553_ModeBusController;

MIL_STD_1553_ADR = 0x1;

MIL_STD_1553_OutPolar = MIL_STD_1553_OutPolar_Low;

MIL_STD_1553_EnPolar = MIL_STD_1553_EnPolar_Low;

MIL_STD_1553_TimeInit(MIL_STD_1553_InitStruct);

А так-же значений временных параметров по умолчанию**. (см.** ***MIL_STD_1553_TimeInit*****)**

Параметры:

MIL_STD_1553_InitStruct – указатель на *MIL_STD_1553_InitTypeDef* структуру.

Возвращаемые значения: нет.

Функция:

void MIL_STD_1553_ITConfig(MIL_STD_1553_TypeDef * MIL_STD_1553x, uint32_t MIL_STD_1553_IT, FunctionalState NewState)

Настройка события для активации прерывания и разрешение установки бита MSG_OK для соответствующей команды управления**.**

Параметры:

MIL_STD_1553x – где x выбранный интерфейс (1..2);

MIL_STD_1553_IT – команда управления или маска прерывания.

Принимаемые значения:

Разрешение установки бита MSG_OK для соответствующей команды управления:

MIL_STD_1553_INTERRUPT_EVTMC_DYNBC «принять управление интерфейсом»

MIL_STD_1553_INTERRUPT_EVTMC_SYNC «синхронизация»

MIL_STD_1553_INTERRUPT_EVTMC_SENDSW «передать ответное слово»

MIL_STD_1553_INTERRUPT_EVTMC_INITST «начать самоконтроль ОУ»

MIL_STD_1553_INTERRUPT_EVTMC_ENCON «разблокировать передатчик»

MIL_STD_1553_INTERRUPT_EVTMC_ENCOFF «блокировать передатчик»

MIL_STD_1553_INTERRUPT_EVTMC_MASKON «блокировать признак неисправности ОУ»

MIL_STD_1553_INTERRUPT_EVTMC_OVMSKOFF «разблокировать признак неисправности ОУ»

MIL_STD_1553_INTERRUPT_EVTMC_RESET «установить ОУ в исходное состояние»

MIL_STD_1553_INTERRUPT_EVTMC_SENDVEC «передать векторное слово»

MIL_STD_1553_INTERRUPT_EVTMC_SENDCW «передать последнюю команду»

MIL_STD_1553_INTERRUPT_EVTMC_SYNCDW «синхронизация с СД»

MIL_STD_1553_INTERRUPT_EVTMC_SENDBITW «передать слово встроенной системы контроля»

MIL_STD_1553_INTERRUPT_EVTMC_SENCON «разблокировать i-ый передатчик»

MIL_STD_1553_INTERRUPT_EVTMC_SENCOFF «блокировать i-ый передатчик»

Маска прерывания:

MIL_STD_1553_INTERRUPT_ERR_NOWORD «нет ответа»

MIL_STD_1553_INTERRUPT_ERR_NOGAP «нет паузы»

MIL_STD_1553_INTERRUPT_ERR_SYNC «ошибка типа слова»

MIL_STD_1553_INTERRUPT_ERR_PAR «ошибка бита четности»

MIL_STD_1553_INTERRUPT_ERR_M2 «ошибка Манчестер II кодирования»

MIL_STD_1553_INTERRUPT_MSG_OK «для режимов КШ и ОУ "сообщение выполнено успешно". Также в режиме ОУ выбирается для каких команд управления этот бит будет устанавливаться. Для режима МШ обозначает "принята группа сообщений"»

NewState – новое состояние. Принимаемые значения ENABLE и DISABLE.

Возвращаемые значения: нет.

Функция:

void MIL_STD_1553_Transmitter_CMD(MIL_STD_1553_TypeDef * MIL_STD_1553x, uint32_t TRANSMITTERx)

Выбор активного канала для передачи данных**.**

Параметры:

MIL_STD_1553x – где x выбранный интерфейс (1..2);

TRANSMITTERx – где x выбранный канал.

Принимаемые значения:

MIL_STD_1553_TRANSMITTER_MAIN - основной канал

MIL_STD_1553_TRANSMITTER_RESERVE – резервный канал

Возвращаемые значения: нет.

1.6.2 Запись и чтение

Функция:

void MIL_STD_1553_GetDebugDec(MIL_STD_1553_TypeDef * MIL_STD_1553x, MIL_STD_1553_DebugTime* MIL_STD_1553_DebugTimeStruct)

Чтение временных параметров последнего приема данных.

Параметры:

MIL_STD_1553x – где x выбранный порт (1..2);

MIL_STD_1553_DebugTimeStruct – указатель на структуру, в поля которой будет записана отладочная информация.

//dbgdec_a. Времена относятся к приемному тракту на основной шине интерфейса.

MIL_STD_1553_MainTrNs - Время центрального перепада в окне приема NS.

MIL_STD_1553_MainTrNb - Время центрального перепада в окне приема NB.

MIL_STD_1553_MainTr1b - Время центрального перепада в окне приема 1B.

MIL_STD_1553_MainTr1s - Время центрального перепада в окне приема 1S.

//dbgdec_b. Времена относятся к приемному тракту на основной шине интерфейса.

MIL_STD_1553_MainTrNsAny - Время перепада, который не попал окно приема NS.

MIL_STD_1553_MainTrNbAny - Время перепада, который не попал окно приема NB.

MIL_STD_1553_MainTr1bAny - Время перепада, который не попал окно приема 1B.

MIL_STD_1553_MainTr1sAny - Время перепада, который не попал окно приема 1S.

//dbgdec_c. Времена относятся к приемному тракту на резервной шине интерфейса.

MIL_STD_1553_ReserveTrNs - Время центрального перепада в окне приема NS.

MIL_STD_1553_ReserveTrNb - Время центрального перепада в окне приема NB.

MIL_STD_1553_ReserveTr1b - Время центрального перепада в окне приема 1B.

MIL_STD_1553_ReserveTr1s - Время центрального перепада в окне приема 1S.

//dbgdec_d. Времена относятся к приемному тракту на резервной шине интерфейса.

MIL_STD_1553_ReserveTrNsAny - Время перепада, который не попал окно приема NS.

MIL_STD_1553_ReserveTrNbAny - Время перепада, который не попал окно приема NB.

MIL_STD_1553_ReserveTr1bAny - Время перепада, который не попал окно приема 1B.

MIL_STD_1553_ReserveTr1sAny - Время перепада, который не попал окно приема 1S.

Возвращаемые значения: нет.

Функция:

uint32_t MIL_STD_1553_GetCommandWord(MIL_STD_1553_TypeDef * MIL_STD_1553x, uint32_t COMMAND_WORDx)

Чтение командного слова.

Параметры:

MIL_STD_1553x – где x выбранный порт (1..2);

COMMAND_WORDx – где x – выбранное командное слово (1..2).

Возвращаемые значения: командное слово. Выравнивание по правой границе.

Функция:

void MIL_STD_1553_SetCommandWord(MIL_STD_1553_TypeDef * MIL_STD_1553x, uint32_t COMMAND_WORDx, MIL_STD_1553_CommandWordTypeDef * CommandWord)

Установка командного слова.

Параметры:

MIL_STD_1553x – где x выбранный порт (1..2);

COMMAND_WORDx – где x – выбранное командное слово (1..2).

CommandWord – *указатель на структуру, содержащую командное слово.*

Возвращаемые значения: нет.

Функция:

uint32_t MIL_STD_1553_GetModeData(MIL_STD_1553_TypeDef * MIL_STD_1553x, uint32_t SubAddr, uint32_t Command)

Слово данных, принятое по команде управления.

Параметры:

MIL_STD_1553x – где x выбранный порт (1..2);

SubAddr – подадрес.

Принимаемые значения:

MIL_STD_1553_CWSUBADR0 – слово данных, принятое по команде управления в 0-й подадрес

MIL_STD_1553_CWSUBADR1 – слово данных, принятое по команде управления в 31-й подадрес

Command – команда управления.

Принимаемые значения:

MIL_STD_1553_CWCOMMANDSYNCW - синхронизация с СД

MIL_STD_1553_CWCOMMANDVECW - передать векторное слово

MIL_STD_1553_CWCOMMANDBITW - передать BCK слово

MIL_STD_1553_CWCOMMANDTRONW - разблокировать i-ый передатчик

MIL_STD_1553_CWCOMMANDTROFFW - блокировать i-ый передатчик

Возвращаемые значения: слово данных, принятое по команде управления.

Функция:

uint32_t MIL_STD_1553_SetModeData(MIL_STD_1553_TypeDef * MIL_STD_1553x, uint32_t SubAddr, uint32_t Command, uint32_t Message)

Слово данных, принятое по команде управления.

Параметры:

MIL_STD_1553x – где x выбранный порт (1..2);

SubAddr – подадрес.

Принимаемые значения:

MIL_STD_1553_CWSUBADR0 – слово данных, передаваемое по команде управления из 0-й подадреса

MIL_STD_1553_CWSUBADR1 – слово данных, передаваемое по команде управления из 31-й подадреса

Command – команда управления.

Принимаемые значения:

MIL_STD_1553_CWCOMMANDSYNCW - синхронизация с СД

MIL_STD_1553_CWCOMMANDVECW - передать векторное слово

MIL_STD_1553_CWCOMMANDBITW - передать BCK слово

MIL_STD_1553_CWCOMMANDTRONW - разблокировать i-ый передатчик

MIL_STD_1553_CWCOMMANDTROFFW - блокировать i-ый передатчик

***Message* –** слово данных

Возвращаемые значения: нет.

Функция:

uint32_t MIL_STD_1553_GetMsg(MIL_STD_1553_TypeDef * MIL_STD_1553x)

Чтение формата сообщения.

Параметры:

MIL_STD_1553x – где x выбранный порт (1..2);

Возвращаемые значения: формат сообщения.

Функция:

void MIL_STD_1553_SetMsgType(MIL_STD_1553_TypeDef * MIL_STD_1553x,uint32_t Message)

Установка формата сообщения.

Параметры:

MIL_STD_1553x – где x выбранный порт (1..2);

Message – формат сообщения.

Принимаемые значения:

0 – передача данных от КШ к ОУ.

1 – передача данных от ОУ к КШ.

2 – передача данных от ОУ к ОУ.

3 – передача команды управления.

4 – передача команды управления и прием слова данных от ОУ.

5 – передача команды управления со словом данных оконечному устройству.

6 – передача данных (в групповом сообщении) от КШ к ОУ.

7 – передача данных (в групповом сообщении) от ОУ к ОУ.

8 – передача групповой команды управления.

9 – передача групповой команды управления со словом данных.

Возвращаемые значения: нет.

Функция:

void MIL_STD_1553_SetMsgInt(MIL_STD_1553_TypeDef * MIL_STD_1553x,uint32_t value)

Количество принятых сообщений, после которого будет установлен MSG_OK.

Параметры:

MIL_STD_1553x – где x выбранный порт (1..2);

value – количество сообщений.

Принимаемые значения: 1..32;

Возвращаемые значения: нет.

Функция:

void MIL_STD_1553_SetStatusWord(MIL_STD_1553_TypeDef * MIL_STD_1553x, MIL_STD_1553_StatusWordTypeDef * StatusWord)

Установка статусного слова.

Параметры:

MIL_STD_1553x – где x выбранный порт (1..2);

StatusWord – указатель на структуру, содержащую статусное слово.

Возвращаемые значения: нет.

Функция:

uint32_t MIL_STD_1553_GetStatusWord(MIL_STD_1553_TypeDef * MIL_STD_1553x, uint32_t STATUS_WORDx)

Чтение статусного слова.

Параметры:

MIL_STD_1553x – где x выбранный порт (1..2);

STATUS_WORDx – выбор статусного слова.

Принимаемые значения:

MIL_STD_1553_STATUS_WORD1 – первое статусное слово

MIL_STD_1553_STATUS_WORD2 – второе статусное слово

Возвращаемые значения: нет.

Функция:

uint32_t MIL_STD_1553_GetStatus(MIL_STD_1553_TypeDef * MIL_STD_1553x)

Чтение статусного регистра.

Параметры:

MIL_STD_1553x – где x выбранный порт (1..2);

Возвращаемые значения: нет.

Функция:

FlagStatus MIL_STD_1553_GetFlagStatus(MIL_STD_1553_TypeDef * MIL_STD_1553x, uint32_t MIL_STD_1553_FLAG)

Чтение состояния флага выбранного события.

Параметры:

MIL_STD_1553x – где x выбранный порт (1..2);

MIL_STD_1553_FLAG – выбранный флаг состояния.

Принимаемые значения:

MIL_STD_1553_STATUS_RT_BUS

MIL_STD_1553_STATUS_RT_TR

MIL_STD_1553_STATUS_RT_BROAD

MIL_STD_1553_STATUS_RT_MC

MIL_STD_1553_STATUS_ERR_NOWORD

MIL_STD_1553_STATUS_ERR_NOGAP

MIL_STD_1553_STATUS_ERR_SYNC

MIL_STD_1553_STATUS_ERR_PAR

MIL_STD_1553_STATUS_ERR_M2
MIL_STD_1553_STATUS_MSG_OK

Возвращаемые значения: Состояние флага MIL_STD_1553_FLAG порта MIL_STD_1553x (SET или RESET).

Функция:

uint32_t MIL_STD_1553_GetMsgSubAddr(MIL_STD_1553_TypeDef * MIL_STD_1553x)

Чтение подадреса в принятом сообщении.

Параметры:

MIL_STD_1553x – где x выбранный порт (1..2);

Возвращаемые значения: подадрес в принятом сообщении.

Функция:

void MIL_STD_1553_ClearStatus(MIL_STD_1553_TypeDef * MIL_STD_1553x)

Очистка статусного регистра

Параметры:

MIL_STD_1553x – где x выбранный порт (1..2);

Возвращаемые значения: нет.

Функция:

void MIL_STD_1553_ClearFlagStatus(MIL_STD_1553_TypeDef * MIL_STD_1553x, *uint32_t MIL_STD_1553_FLAG*)

Очистка статусного регистра

Параметры:

MIL_STD_1553x – где x выбранный порт (1..2);

MIL_STD_1553_FLAG – выбранный флаг состояния.

Принимаемые значения:

MIL_STD_1553_STATUS_RT_BUS
MIL_STD_1553_STATUS_RT_TR
MIL_STD_1553_STATUS_RT_BROAD
MIL_STD_1553_STATUS_RT_MC
MIL_STD_1553_STATUS_ERR_NOWORD
MIL_STD_1553_STATUS_ERR_NOGAP
MIL_STD_1553_STATUS_ERR_SYNC
MIL_STD_1553_STATUS_ERR_PAR
MIL_STD_1553_STATUS_ERR_M2
MIL_STD_1553_STATUS_MSG_OK

Возвращаемые значения: нет.

Функция:

void MIL_STD_1553_StartTransmision(MIL_STD_1553_TypeDef * MIL_STD_1553x)

Функция старта передачи.

Параметры:

MIL_STD_1553x – где x выбранный порт (1..2);

Возвращаемые значения: нет.

Функция:

void MIL_STD_1553_ReceiveDataFromBuffer(MIL_STD_1553_TypeDef * MIL_STD_1553x, uint32_t Subaddress, uint32_t NumberDataWords, uint32_t * ptr_Dest)

Чтение данных из буфера приемника.

Параметры:

MIL_STD_1553x – где x выбранный порт (1..2);

Subaddress – подадрес. Для КШ необходимо указать 0x1.

NumberDataWords – количество слов данных.

ptr_Dest – указатель на массив, в который будут записаны слова данных из буфера приемника.

Возвращаемые значения: нет.

Функция:

void MIL_STD_1553_WriteDataToSendBuffer(MIL_STD_1553_TypeDef * MIL_STD_1553x, uint32_t Subaddress, uint32_t NumberDataWords, uint32_t * ptr_Src)

Запись данных в буфер передатчика.

Параметры:

MIL_STD_1553x – где x выбранный порт (1..2);

Subaddress – подадрес. Для КШ необходимо указать 0x1.

NumberDataWords – количество слов данных.

ptr_Src – указатель на массив, из которого будут считаны слова данных и записаны в буфер приемника.

Возвращаемые значения: нет.

1.6.3 Применение библиотеки интерфейса по ГОСТ 52070

После сброса все конфигурационные данные интерфейсов обнуляются. В микросхеме 2 MIL_STD_1553 интерфейса. Порты настраиваются регистрами, которые инициализируются функцией ***MIL_STD_1553_Init()***, в соответствии со значениями структуры ***MIL_STD_1553_InitTypeDef***.

MIL_STD_1553_Mode: режим работы контроллера

MIL_STD_1553_ModeTerminal – режим ОУ

MIL_STD_1553_ModeMonitor – режим МШ

MIL_STD_1553_ModeBusController – режим КШ

MIL_STD_1553_ADR: адрес устройства (0-31)

MIL_STD_1553_OutPolar: состояние прямого и инверсного выходов, когда передача отсутствует

MIL_STD_1553_EnPolar: состояние выхода включения передатчика, когда передача отсутствует.
MIL_STD_1553_Pause: пауза перед передачей ответного слова в режиме ОУ и пауза перед началом передачи нового сообщения в режиме КШ. Стандартный диапазон от 4×HALFBIT до 20×HALFBIT, что соответствует временному интервалу 4-12 мкс по ГОСТ Р 52070.
MIL_STD_1553_HalfBit: длительность половины бита на шине интерфейса (измеряется в тактах системной частоты)
MIL_STD_1553_NoGap: минимальная пауза приема ответного слова и приема нового сообщения. Рекомендуемое значение 2×HALFBIT.
MIL_STD_1553_NoWord: максимальная пауза приема ответного слова или слова данных. Стандартное значение должно быть не менее 24×HALFBIT, что соответствует временному интервалу не менее 14 мкс по ГОСТ Р 52070.
MIL_STD_1553_JtrNbMax: определяет правую границу окна приема NB. Рекомендуемое значение HALFBIT/2.
MIL_STD_1553_JtrNbMin: определяет левую границу окна приема NB. Рекомендуемое значение HALFBIT/2.
MIL_STD_1553_Jtr1bMax: определяет правую границу окна приема 1B. Рекомендуемое значение HALFBIT/2.
MIL_STD_1553_Jtr1bMin: определяет левую границу окна приема 1B. Рекомендуемое значение HALFBIT/2.
MIL_STD_1553_JtrNsMax: определяет правую границу окна приема NS. Рекомендуемое значение 2×HALFBIT.
MIL_STD_1553_JtrNsMin: определяет левую границу окна приема NS. Рекомендуемое значение 2×HALFBIT.
MIL_STD_1553_Jtr1sMax: определяет правую границу окна приема 1S. Рекомендуемое значение 2×HALFBIT.
MIL_STD_1553_Jtr1sMin: определяет левую границу окна приема 1S. Рекомендуемое значение 2×HALFBIT.

После инициализации структуры возможно использование набора функций.

Для автоподстройки работы интерфейса можно использовать возвращаемые параметры временных задержек входного сигнала. Для их хранения используется структура ***MIL_STD_1553_DebugTime***:

Для основного канала:
MIL_STD_1553_MainTrNs: Время центрального перепада в окне приема NS.
MIL_STD_1553_MainTrNb: Время центрального перепада в окне приема NB.
MIL_STD_1553_MainTr1b: Время центрального перепада в окне приема 1B.
MIL_STD_1553_MainTr1s: Время центрального перепада в окне приема 1S
MIL_STD_1553_MainTrNsAny: Время перепада, который не попал окно приема NS.
MIL_STD_1553_MainTrNbAny: Время перепада, который не попал окно приема NB.
MIL_STD_1553_MainTr1bAny: Время перепада, который не попал окно приема 1B.
MIL_STD_1553_MainTr1sAny: Время перепада, который не попал окно приема 1S.

Для резервного канала:
MIL_STD_1553_ReserveTrNs: Время центрального перепада в окне приема NS.
MIL_STD_1553_ReserveTrNb: Время центрального перепада в окне приема NB
MIL_STD_1553_ReserveTr1b: Время центрального перепада в окне приема 1B.
MIL_STD_1553_ReserveTr1s: Время центрального перепада в окне приема 1S
MIL_STD_1553_ReserveTrNsAny: Время перепада, который не попал окно приема NS
MIL_STD_1553_ReserveTrNbAny: Время перепада, который не попал окно приема NB.
MIL_STD_1553_ReserveTr1bAny: Время перепада, который не попал окно приема 1B.
MIL_STD_1553_ReserveTr1sAny: Время перепада, который не попал окно приема 1S.

Объединение ***MIL_STD_1553_StatusWordTypeDef:***
uint32_t StatusWord: Полное статусное слово.
MIL_STD_1553_StatusWordBitFields Fields: Статусное слово разбитое на битовые поля

Структура ***MIL_STD_1553_StatusWordBitFields:***
FaultTDBit: Неисправность ОУ (принимаемые значения SET или RESET)
AdoptionControlInterfaceBit: Принято управление интерфейсом (принимаемые значения SET или RESET)
AbonentFaultBit: Неисправность абонента (принимаемые значения SET или RESET)
BusyBit: Абонент занят (принимаемые значения SET или RESET)
GroupCommandBit: Принята групповая команда (принимаемые значения SET или RESET)
Reserved: 3 неиспользуемых бита в ОС
ServiceRequestBit: Запрос на обслуживание (принимаемые значения SET или RESET)
TransferReplyWordBit: Передача ОС (принимаемые значения SET или RESET)
ErrorBit: Ошибка в сообщении (принимаемые значения SET или RESET)
TerminalDeviceAddress: Адрес ОУ (принимаемые значения 0x00 - 0x1F)

Объединение ***MIL_STD_1553_DataFieldsTypeDef:***
uint32_t NumberDataWords: Число СД.
uint32_t Cmd: Команда управления.

Объединение ***MIL_STD_1553_CommandWordTypeDef***
uint32_t CommandWord: Полное командное слово
MIL_STD_1553_CommandWordBitFields Fields: Командное слово разбитое на битовые поля

Структура ***MIL_STD_1553_CommandWordBitFields:***
MIL_STD_1553_DataFieldsTypeDef Data: Указатель на структуру, содержащую число СД или код команды.
uint32_t Subaddress: Подадрес. (подадрес 0b00001 – 0b11110, команда управления 0b00000 или 0b11111)
uint32_t ReadWriteBit: Разряд признака «Прием/передача» (принимаемые значения SET или RESET. RESET – прием в ОУ, SET – передача от ОУ)

uint32_t TerminalDeviceAddress: Адрес ОУ (принимаемые значения 0b0000-0b11111)

Формат 1 (Передача четырех СД от КШ к ОУ):

Для контроллера шины:

```
MIL_STD_1553_DeInit(MIL_STD_15531);
MIL_STD_1553_InitTypeDef MIL_STD_1553_InitStruct1;  //Создаем новую структуру
MIL_STD_1553_TimeInit(&MIL_STD_1553_InitStruct1); //Инициализируем временные параметры по-
умолчанию
MIL_STD_1553_InitStruct1.MIL_STD_1553_ADR = 0x1; //Устанавливаем адрес
MIL_STD_1553_InitStruct1.MIL_STD_1553_EnPolar = MIL_STD_1553_EnPolar_Low; // состояние выхода
включения передатчика, когда передача отсутствует.
MIL_STD_1553_InitStruct1.MIL_STD_1553_OutPolar = MIL_STD_1553_OutPolar_Low; // состояние выхода
включения передатчика, когда передача отсутствует в 0
MIL_STD_1553_InitStruct1.MIL_STD_1553_Mode = MIL_STD_1553_ModeBusController; // режим работы КШ
MIL_STD_1553_Init(MIL_STD_15531, &MIL_STD_1553_InitStruct1);

MIL_STD_1553_CommandWordTypeDef firstCommandWord;
firstCommandWord.Fields.ReadWriteBit = RESET;
firstCommandWord.Fields.TerminalDeviceAddress = 0x2;
MIL_STD_1553_DataFieldsTypeDef Data;
Data.NumberDataWords = 4;
firstCommandWord.Fields.Data = Data;
firstCommandWord.Fields.Subaddress = 0x1;
MIL_STD_1553_SetCommandWord(MIL_STD_15531,MIL_STD_1553_COMMAND_WORD1,&firstCommandWord);

uint32_t send_ptr[4] = {1,2,3,4};
MIL_STD_1553_WriteDataToSendBuffer(MIL_STD_15531, 0x1, 0x4, send_ptr);
MIL_STD_1553_SetMsgType(MIL_STD_15531,0x0);
MIL_STD_1553_Transmitter_CMD(MIL_STD_15531, MIL_STD_1553_TRANSMITTER_MAIN);

MIL_STD_1553_StartTransmision(MIL_STD_15531);
```

Для оконечного устройства:

```
uint32_t res_ptr[4] ;

void MIL_STD_15532_Handler(void)
{
  MIL_STD_1553_ReceiveDataFromBuffer(MIL_STD_15532, 0x1, 0x4, res_ptr);
  MIL_STD_1553_ClearStatus(MIL_STD_15532);
}

int main (void)
{
  NVIC_InitTypeDef NVIC_InitStructure1;
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
  NVIC_InitStructure1.NVIC_IRQChannel = MIL_STD_15532_IRQn;
  NVIC_InitStructure1.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure1.NVIC_IRQChannelSubPriority = 0;
  NVIC_InitStructure1.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure1);

  MIL_STD_1553_InitTypeDef MIL_STD_1553_InitStruct2;
  MIL_STD_1553_DeInit(MIL_STD_15532);
  MIL_STD_1553_TimeInit(&MIL_STD_1553_InitStruct2);
  MIL_STD_1553_InitStruct2.MIL_STD_1553_ADR = 0x2;
  MIL_STD_1553_InitStruct2.MIL_STD_1553_EnPolar =MIL_STD_1553_EnPolar_Low;
  MIL_STD_1553_InitStruct2.MIL_STD_1553_OutPolar = MIL_STD_1553_OutPolar_Low;
  MIL_STD_1553_InitStruct2.MIL_STD_1553_Mode = MIL_STD_1553_ModeTerminal;
  MIL_STD_1553_Init(MIL_STD_15532, &MIL_STD_1553_InitStruct2);
```

```
  MIL_STD_1553_ITConfig(MIL_STD_15532, MIL_STD_1553_INTERRUPT_MSG_OK, ENABLE);
}
```

Формат 2 (Передача четырех СД от ОУ к КШ):

Для контроллера шины:

```
uint32_t res_ptr[4];

MIL_STD_1553_DeInit(MIL_STD_15531);
MIL_STD_1553_InitTypeDef MIL_STD_1553_InitStruct1;  //Создаем новую структуру
MIL_STD_1553_TimeInit(&MIL_STD_1553_InitStruct1); //Инициализируем временные параметры по-
умолчанию
MIL_STD_1553_InitStruct1.MIL_STD_1553_ADR = 0x1;
MIL_STD_1553_InitStruct1.MIL_STD_1553_EnPolar = MIL_STD_1553_EnPolar_Low;
MIL_STD_1553_InitStruct1.MIL_STD_1553_OutPolar = MIL_STD_1553_OutPolar_Low;
MIL_STD_1553_InitStruct1.MIL_STD_1553_Mode = MIL_STD_1553_ModeBusController;
MIL_STD_1553_Init(MIL_STD_15531, &MIL_STD_1553_InitStruct1);

MIL_STD_1553_CommandWordTypeDef firstCommandWord;
firstCommandWord.Fields.ReadWriteBit = SET;
firstCommandWord.Fields.TerminalDeviceAddress = 0x2;
firstCommandWord.Fields.Data = 0x4;
firstCommandWord.Fields.Subaddress = 0x1;
MIL_STD_1553_SetCommandWord(MIL_STD_15531,MIL_STD_1553_COMMAND_WORD1,&firstCommandWord);

MIL_STD_1553_SetMsgType(MIL_STD_15531,0x1);
MIL_STD_1553_Transmitter_CMD(MIL_STD_15531, MIL_STD_1553_TRANSMITTER_MAIN);

MIL_STD_1553_StartTransmision(MIL_STD_15531);
while (MIL_STD_1553_GetFlagStatus(MIL_STD_15531, MIL_STD_1553_INTERRUPT_MSG_OK) != SET){}
MIL_STD_1553_ReceiveDataFromBuffer(MIL_STD_15531, 0x1, 0x4, res_ptr);
```

Для оконечного устройства:

```
uint32_t send_ptr[4] = {1,2,3,4};

void MIL_STD_15532_Handler(void)
{
  MIL_STD_1553_ClearStatus(MIL_STD_15532);
}

int main (void)
{
  NVIC_InitTypeDef NVIC_InitStructure1;
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
  NVIC_InitStructure4.NVIC_IRQChannel = MIL_STD_15532_IRQn;
  NVIC_InitStructure4.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure4.NVIC_IRQChannelSubPriority = 0;
  NVIC_InitStructure4.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure1);

  MIL_STD_1553_InitTypeDef MIL_STD_1553_InitStruct2;
  MIL_STD_1553_DeInit(MIL_STD_15532);
  MIL_STD_1553_TimeInit(&MIL_STD_1553_InitStruct2);
  MIL_STD_1553_InitStruct2.MIL_STD_1553_ADR = 0x2;
  MIL_STD_1553_InitStruct2.MIL_STD_1553_EnPolar =MIL_STD_1553_EnPolar_Low;
  MIL_STD_1553_InitStruct2.MIL_STD_1553_OutPolar = MIL_STD_1553_OutPolar_Low;
  MIL_STD_1553_InitStruct2.MIL_STD_1553_Mode = MIL_STD_1553_ModeTerminal;
  MIL_STD_1553_Init(MIL_STD_15532, &MIL_STD_1553_InitStruct2);

  MIL_STD_1553_WriteDataToSendBuffer(MIL_STD_15531, 0x1, 0x4, send_ptr);
  MIL_STD_1553_ITConfig(MIL_STD_15531, MIL_STD_1553_INTERRUPT_MSG_OK, ENABLE);
```

```
  while(1){}
}
```

Формат 3 (Передача четырех СД от одного ОУ к другому ОУ):

Для контроллера шины

```
MIL_STD_1553_DeInit(MIL_STD_15531);
MIL_STD_1553_InitTypeDef MIL_STD_1553_InitStruct1;  //Создаем новую структуру
MIL_STD_1553_TimeInit(&MIL_STD_1553_InitStruct1); //Инициализируем временные параметры по-
умолчанию
MIL_STD_1553_InitStruct1.MIL_STD_1553_ADR = 0x1;
MIL_STD_1553_InitStruct1.MIL_STD_1553_EnPolar = MIL_STD_1553_EnPolar_Low;
MIL_STD_1553_InitStruct1.MIL_STD_1553_OutPolar = MIL_STD_1553_OutPolar_Low;
MIL_STD_1553_InitStruct1.MIL_STD_1553_Mode = MIL_STD_1553_ModeBusController;
MIL_STD_1553_Init(MIL_STD_15531, &MIL_STD_1553_InitStruct1);

MIL_STD_1553_CommandWordTypeDef firstCommandWord;
firstCommandWord.Fields.ReadWriteBit = RESET;
firstCommandWord.Fields.TerminalDeviceAddress = 0x1;
firstCommandWord.Fields.Data = 0x4;
firstCommandWord.Fields.Subaddress = 0x1;
MIL_STD_1553_SetCommandWord(MIL_STD_15531,MIL_STD_1553_COMMAND_WORD1,&firstCommandWord);

MIL_STD_1553_CommandWordTypeDef secondCommandWord;
secondCommandWord.Fields.ReadWriteBit = SET;
secondCommandWord.Fields.TerminalDeviceAddress = 0x2;
secondCommandWord.Fields.Data = 0x4;
secondCommandWord.Fields.Subaddress = 0x1;
MIL_STD_1553_SetCommandWord(MIL_STD_15531,MIL_STD_1553_COMMAND_WORD2,& secondCommandWord);

MIL_STD_1553_SetMsgType(MIL_STD_15531,0x3);
MIL_STD_1553_Transmitter_CMD(MIL_STD_15531, MIL_STD_1553_TRANSMITTER_MAIN);

MIL_STD_1553_StartTransmision(MIL_STD_15531);
```

Для первого оконечного устройства

```
uint32_t send_ptr[4] ;
void MIL_STD_15532_Handler(void)
{
  MIL_STD_1553_ReceiveDataFromBuffer(MIL_STD_15532, 0x1, 0x4, send_ptr);
  MIL_STD_1553_ClearStatus(MIL_STD_15532);
}
int main (void)
{
  NVIC_InitTypeDef NVIC_InitStructure1;
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
  NVIC_InitStructure1.NVIC_IRQChannel = MIL_STD_15532_IRQn;
  NVIC_InitStructure1.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure1.NVIC_IRQChannelSubPriority = 0;
  NVIC_InitStructure1.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure1);

  MIL_STD_1553_DeInit(MIL_STD_15532);
  MIL_STD_1553_InitTypeDef MIL_STD_1553_InitStruct1;
  MIL_STD_1553_TimeInit(&MIL_STD_1553_InitStruct1);
  MIL_STD_1553_InitStruct1.MIL_STD_1553_ADR = 0x1;
  MIL_STD_1553_InitStruct1.MIL_STD_1553_EnPolar = MIL_STD_1553_EnPolar_Low;
  MIL_STD_1553_InitStruct1.MIL_STD_1553_OutPolar = MIL_STD_1553_OutPolar_Low;
  MIL_STD_1553_InitStruct1.MIL_STD_1553_Mode = MIL_STD_1553_ModeTerminal;
  MIL_STD_1553_Init(MIL_STD_15532, &MIL_STD_1553_InitStruct1);
  MIL_STD_1553_ITConfig(MIL_STD_15532, MIL_STD_1553_INTERRUPT_MSG_OK, ENABLE);
  while(1){}
}
```

Для второго оконечного устройства

```
int main (void)
{
  MIL_STD_1553_InitTypeDef MIL_STD_1553_InitStruct2;
  MIL_STD_1553_DeInit(MIL_STD_15532);
  MIL_STD_1553_TimeInit(&MIL_STD_1553_InitStruct2);
  MIL_STD_1553_InitStruct2.MIL_STD_1553_ADR = 0x2;
  MIL_STD_1553_InitStruct2.MIL_STD_1553_EnPolar = MIL_STD_1553_EnPolar_Low;
  MIL_STD_1553_InitStruct2.MIL_STD_1553_OutPolar = MIL_STD_1553_OutPolar_Low;
  MIL_STD_1553_InitStruct2.MIL_STD_1553_Mode = MIL_STD_1553_ModeTerminal;
  MIL_STD_1553_Init(MIL_STD_15532, &MIL_STD_1553_InitStruct2);
  MIL_STD_1553_ITConfig(MIL_STD_15532, MIL_STD_1553_INTERRUPT_MSG_OK, ENABLE);
  uint32_t res_ptr[4] = {1,2,3,4};
  MIL_STD_1553_WriteDataToSendBuffer(MIL_STD_15532, 0x1, 0x4, res_ptr);
  while(1){}
}
```

Формат 4 (Передача КУ):

Для контроллера шины

```
MIL_STD_1553_DeInit(MIL_STD_15531);
MIL_STD_1553_InitTypeDef MIL_STD_1553_InitStruct1;  //Создаем новую структуру
MIL_STD_1553_TimeInit(&MIL_STD_1553_InitStruct1); //Инициализируем временные параметры по-
умолчанию
MIL_STD_1553_InitStruct1.MIL_STD_1553_ADR = 0x1;
MIL_STD_1553_InitStruct1.MIL_STD_1553_EnPolar = MIL_STD_1553_EnPolar_Low;
MIL_STD_1553_InitStruct1.MIL_STD_1553_OutPolar = MIL_STD_1553_OutPolar_Low;
MIL_STD_1553_InitStruct1.MIL_STD_1553_Mode = MIL_STD_1553_ModeBusController;
MIL_STD_1553_Init(MIL_STD_15531, &MIL_STD_1553_InitStruct1);

MIL_STD_1553_CommandWordTypeDef firstCommandWord;
firstCommandWord.Fields.ReadWriteBit = RESET;
firstCommandWord.Fields.TerminalDeviceAddress = 0x2;
firstCommandWord.Fields.Data = 0x1;
firstCommandWord.Fields.Subaddress = 0x0;
MIL_STD_1553_SetCommandWord(MIL_STD_15531,MIL_STD_1553_COMMAND_WORD1,&firstCommandWord);

MIL_STD_1553_SetMsgType(MIL_STD_15531,0x4);
MIL_STD_1553_Transmitter_CMD(MIL_STD_15531, MIL_STD_1553_TRANSMITTER_MAIN);

MIL_STD_1553_StartTransmision(MIL_STD_15531);
```

Для оконечного устройства

```
void MIL_STD_15532_Handler(void)
{
  MIL_STD_1553_ClearStatus(MIL_STD_15532);
}

int main (void)
{
  NVIC_InitTypeDef NVIC_InitStructure1;
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
  NVIC_InitStructure4.NVIC_IRQChannel = MIL_STD_15532_IRQn;
  NVIC_InitStructure4.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure4.NVIC_IRQChannelSubPriority = 0;
  NVIC_InitStructure4.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure1);

  MIL_STD_1553_InitTypeDef MIL_STD_1553_InitStruct2;
  MIL_STD_1553_DeInit(MIL_STD_15532);
```

```
  MIL_STD_1553_TimeInit(&MIL_STD_1553_InitStruct2);
  MIL_STD_1553_InitStruct2.MIL_STD_1553_ADR = 0x2;
  MIL_STD_1553_InitStruct2.MIL_STD_1553_EnPolar =MIL_STD_1553_EnPolar_Low;
  MIL_STD_1553_InitStruct2.MIL_STD_1553_OutPolar = MIL_STD_1553_OutPolar_Low;
  MIL_STD_1553_InitStruct2.MIL_STD_1553_Mode = MIL_STD_1553_ModeTerminal;
  MIL_STD_1553_Init(MIL_STD_15532, &MIL_STD_1553_InitStruct2);
  MIL_STD_1553_ITConfig(MIL_STD_15532, MIL_STD_1553_INTERRUPT_EVTMC_SYNC, ENABLE);

  while(1){}
}
```

Формат 5 (Передача КУ и прием СД от ОУ):

Для контроллера шины

```
MIL_STD_1553_DeInit(MIL_STD_15531);
MIL_STD_1553_InitTypeDef MIL_STD_1553_InitStruct1;  //Создаем новую структуру
MIL_STD_1553_TimeInit(&MIL_STD_1553_InitStruct1); //Инициализируем временные параметры по-
умолчанию
MIL_STD_1553_InitStruct1.MIL_STD_1553_ADR = 0x1;
MIL_STD_1553_InitStruct1.MIL_STD_1553_EnPolar = MIL_STD_1553_EnPolar_Low;
MIL_STD_1553_InitStruct1.MIL_STD_1553_OutPolar = MIL_STD_1553_OutPolar_Low;
MIL_STD_1553_InitStruct1.MIL_STD_1553_Mode = MIL_STD_1553_ModeBusController;
MIL_STD_1553_Init(MIL_STD_15531, &MIL_STD_1553_InitStruct1);

MIL_STD_1553_CommandWordTypeDef firstCommandWord;
firstCommandWord.Fields.ReadWriteBit = SET;
firstCommandWord.Fields.TerminalDeviceAddress = 0x2;
firstCommandWord.Fields.Data = 16;
firstCommandWord.Fields.Subaddress = MIL_STD_1553_CWSUBADR0;
MIL_STD_1553_SetCommandWord(MIL_STD_15531,MIL_STD_1553_COMMAND_WORD1,&firstCommandWord);

MIL_STD_1553_SetMsgType(MIL_STD_15531,0x5);
MIL_STD_1553_Transmitter_CMD(MIL_STD_15531, MIL_STD_1553_TRANSMITTER_MAIN);

MIL_STD_1553_StartTransmision(MIL_STD_15531);
while (MIL_STD_1553_GetFlagStatus(MIL_STD_15531, MIL_STD_1553_INTERRUPT_MSG_OK) != SET){}
MIL_STD_1553_ReceiveDataFromBuffer(MIL_STD_15531, 0x1, 0x1, res_ptr);
```

Для оконечного устройства

```
MIL_STD_1553_InitTypeDef MIL_STD_1553_InitStruct1;
MIL_STD_1553_DeInit(MIL_STD_15532);
MIL_STD_1553_TimeInit(&MIL_STD_1553_InitStruct1);
MIL_STD_1553_InitStruct1.MIL_STD_1553_ADR = 0x2;
MIL_STD_1553_InitStruct1.MIL_STD_1553_EnPolar = MIL_STD_1553_EnPolar_Low;
MIL_STD_1553_InitStruct1.MIL_STD_1553_OutPolar = MIL_STD_1553_OutPolar_Low;
MIL_STD_1553_InitStruct1.MIL_STD_1553_Mode = MIL_STD_1553_ModeTerminal;
MIL_STD_1553_Init(MIL_STD_15532, &MIL_STD_1553_InitStruct1);

MIL_STD_1553_SetModeData(MIL_STD_15532, MIL_STD_1553_CWSUBADR0, MIL_STD_1553_CWCOMMANDVECW,
0x55);
```

Формат 6 (Передача КУ со СД):

Для контроллера шины

```
MIL_STD_1553_DeInit(MIL_STD_15531);
MIL_STD_1553_InitTypeDef MIL_STD_1553_InitStruct1;  //Создаем новую структуру
MIL_STD_1553_TimeInit(&MIL_STD_1553_InitStruct1); //Инициализируем временные параметры по-
умолчанию
MIL_STD_1553_InitStruct1.MIL_STD_1553_ADR = 0x1;
MIL_STD_1553_InitStruct1.MIL_STD_1553_EnPolar = MIL_STD_1553_EnPolar_Low;
MIL_STD_1553_InitStruct1.MIL_STD_1553_OutPolar = MIL_STD_1553_OutPolar_Low;
MIL_STD_1553_InitStruct1.MIL_STD_1553_Mode = MIL_STD_1553_ModeBusController;
MIL_STD_1553_Init(MIL_STD_15531, &MIL_STD_1553_InitStruct1);
```

```
MIL_STD_1553_CommandWordTypeDef firstCommandWord;
firstCommandWord.Fields.ReadWriteBit = RESET;
firstCommandWord.Fields.TerminalDeviceAddress = 0x2;
firstCommandWord.Fields.Data = 17;
firstCommandWord.Fields.Subaddress = MIL_STD_1553_CWSUBADR0;
MIL_STD_1553_SetCommandWord(MIL_STD_15531,MIL_STD_1553_COMMAND_WORD1,&firstCommandWord);

uint32_t send_ptr[1] = 55;

MIL_STD_1553_SetMsgType(MIL_STD_15531,0x5);
MIL_STD_1553_Transmitter_CMD(MIL_STD_15531, MIL_STD_1553_TRANSMITTER_MAIN);
MIL_STD_1553_WriteDataToSendBuffer(MIL_STD_15531, 0x1, 0x1, send_ptr);
MIL_STD_1553_StartTransmision(MIL_STD_15531);
```

Для оконечного устройства

```
uint32_t res_ptr[1] = {0};

void MIL_STD_15532_Handler(void)
{
  res_ptr[0] = MIL_STD_1553_GetModeData(MIL_STD_15532, MIL_STD_1553_CWSUBADR0,
MIL_STD_1553_CWCOMMANDSYNCW);
  MIL_STD_1553_ClearStatus(MIL_STD_15532);
}

MIL_STD_1553_InitTypeDef MIL_STD_1553_InitStruct1;
MIL_STD_1553_DeInit(MIL_STD_15532);
MIL_STD_1553_TimeInit(&MIL_STD_1553_InitStruct1);
MIL_STD_1553_InitStruct1.MIL_STD_1553_ADR = 0x2;
MIL_STD_1553_InitStruct1.MIL_STD_1553_EnPolar = MIL_STD_1553_EnPolar_Low;
MIL_STD_1553_InitStruct1.MIL_STD_1553_OutPolar = MIL_STD_1553_OutPolar_Low;
MIL_STD_1553_InitStruct1.MIL_STD_1553_Mode = MIL_STD_1553_ModeTerminal;
MIL_STD_1553_Init(MIL_STD_15532, &MIL_STD_1553_InitStruct1);

MIL_STD_1553_ITConfig(MIL_STD_15532, MIL_STD_1553_INTERRUPT_MSG_OK, ENABLE);
MIL_STD_1553_ITConfig(MIL_STD_15532, MIL_STD_1553_INTERRUPT_EVTMC_SYNCDW, ENABLE);
while(1) {}
```

1.7 Библиотека конфигурации тактирования PLL

№	Функция	Описание
Инициализация и конфигурация		
1	RCC_DeInit	Переход на тактирование от кварцевого резонатора
2	RCC_Init	Инициализация PLL согласно заданным параметрам в RCC_InitStruct
3	RCC_StructInit	Присвоение членам RCC_InitStruct значений для работы на выбранной частоте

1.7.1 Инициализация и конфигурация

Функция:

void RCC_DeInit(void)

Переход на тактирование от кварцевого резонатора**.** Сброс множителя и делителя частоты. Величина таймаута сигнала Ready не изменяется.

Параметры: нет

Возвращаемые значения: нет.

Функция:

void RCC_Init(RCC_InitTypeDef* RCC)

Инициализация блока PLL согласно заданным параметрам в *RCC_InitStruct*.

Параметры:

RCC_InitStruct – указатель на *RCC_InitTypeDef* структуру которая содержит конфигурационную информацию для блока PLL.

Возвращаемые значения: нет.

Функция:

void RCC_StructInit(RCC_InitTypeDef* PLL, uint32_t freq)

Присвоение членам *RCC_InitStruct* значений, соответствующих выбранной частоте freq при соответственно выбранном кварцевом генераторе.

Выбор кварцевого генератора осуществляется раскомментированием соответствующей строки в файле 1914BA018_rcc.h:

```
//#define QUARZ_12Mhz
//#define QUARZ_14Mhz
#define QUARZ_16Mhz
//#define QUARZ_18Mhz
```

Для возможности выбора частоты выше 60МГц, но не более 72МГц, необходимо раскомментировать строку:

```
#define OVERCLOCK
```

Стабильность работы микроконтроллера, в данном случае, не гарантируется.

Параметры:

freq – частота тактирования процессора с учетом настройки блока PLL.

Принимаемые значения, в соответствии с выбранным кварцевым генератором, см. в файле 1914BA018_rcc.h

Возвращаемые значения: нет.

1.7.2 Применение библиотеки конфигурации тактирования PLL

После сброса, выполняется переход на тактирование от кварцевого резонатора**.** Происходит сброс множителя и делителя частоты. Величина таймаута сигнала Ready 0xFF.

Блок PLL настраивается значениями, которые инициализируются функцией ***RCC_Init()***, в соответствии со значениями структуры ***RCC_InitTypeDef***.

PLL_Mul:

2 - 32;

PLL_Div:

1 - 32;

PLL_Count:

0 - 256;

После инициализации структуры возможно использование набора функций.

Пример_1

```
// Создаем переменную RCC_InitStruct  с типом данных RCC_InitTypeDef;
RCC_InitTypeDef RCC_InitStruct;
//Присваиваем членам RCC_InitStruct значений, соответствующих 60MHz
RCC_StructInit(&RCC_InitStruct, RCC_FREQ_60Mhz);
//Инициализируем структуру RCC_InitStruct
RCC_Init(&RCC_InitStruct);
```

Пример_2

```
RCC_InitTypeDef RCC_InitStruct;
//Устанавливаем делитель PLL в структуре RCC_InitStruct
RCC_InitStruct.PLL_Div=2;
//Устанавливаем множитель PLL в структуре RCC_InitStruct
RCC_InitStruct.PLL_Mul=3;
//Устанавливаем таймаут для сигнала Ready в структуре RCC_InitStruct
RCC_InitStruct.PLL_Count=3;
RCC_Init(&RCC_InitStruct);
```

1.8 Библиотека векторов прерываний NVIC

№	Функция	Описание
Инициализация и конфигурация		
1	NVIC_PriorityGroupConfig	Задание группы/подгруппы приоритетов
2	NVIC_Init	Инициализация вектора прерывания, согласно заданным параметрам в NVIC_InitStruct
3	NVIC_SetVectorTable	Изменение адреса векторов
4	SysTick_CLKSourceConfig	Выбор делителя тактирования таймера SysTick

1.8.1 Инициализация и конфигурация

Функция:

void NVIC_PriorityGroupConfig(uint32_t NVIC_PriorityGroup)

Задание группы/подгруппы приоритетов

Параметры:

NVIC_PriorityGroup – выбор группы приоритетов.

Принимаемые значения:

NVIC_PriorityGroup_0;
NVIC_PriorityGroup_1;
NVIC_PriorityGroup_2;
NVIC_PriorityGroup_3;
NVIC_PriorityGroup_4;

Возвращаемые значения: нет.

Функция:

void NVIC_Init(NVIC_InitTypeDef* NVIC_InitStruct);

Инициализация вектора прерывания, согласно заданным параметрам в NVIC_InitStruct**.**

Параметры:

NVIC_InitStruct – указатель на *NVIC_InitTypeDef* структуру которая содержит конфигурационную информацию для вектора прерывания.

Возвращаемые значения: нет.

Функция:

void NVIC_SetVectorTable(uint32_t NVIC_VectTab, uint32_t Offset)

Изменение активности таймера.

Параметры:

NVIC_VectTab – адрес. Принимаемые значения:

NVIC_VectTab_RAM;
NVIC_VectTab_EXT_FLASH;
NVIC_VectTab_FLASH;

Offset – смещение. Принимаемые значения: 0x0 - 0x000FFFFF;

Возвращаемые значения: нет.

Функция:

void SysTick_CLKSourceConfig(uint32_t SysTick_CLKSource)

Выбор делителя тактирования таймера SysTick**.**

Параметры:

SysTick_CLKSource. Принимаемые значения

SysTick_CLKSource_HCLK_Div2;

SysTick_CLKSource_HCLK;

Возвращаемые значения: нет.

1.8.2 Применение библиотеки векторов прерываний

После сброса все внешние прерывания запрещаются и конфигурационные данные обнуляются.

Векторы прерываний настраиваются регистрами, которые инициализируются функцией ***NVIC_Init()***, в соответствии со значениями структуры ***NVIC_InitTypeDef***.

NVIC_IRQChannel – Выбор канала прерывания;

NVIC_IRQChannelPreemptionPriority – выбор группы приоритета;

NVIC_IRQChannelSubPriority – выбор группы суб-приоритета;

NVIC_IRQChannelCmd – новое состояние канала прерывания. Принимаемые значения:

ENABLE или DISABLE;

После инициализации структуры возможно использование набора функций.

Пример_1

```
void TIM1_Handler(void)
{
  if(TIM_GetITStatus(TIM1) == SET){/
    GPIO_ToggleBits(GPIOE, GPIO_Pin_2);//инвертируем состояние пина 2 порта GPIOE (см. Библиотеку портов GPIO)
    TIM_ClearITPendingBit(TIM1);
  }
}
int main (void)
{
//Создаем переменную NVIC_InitStructure с типом данных NVIC_InitTypeDef для инициализации векторов прерывания(см. описание NVIC)
  NVIC_InitTypeDef NVIC_InitStructure;
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1); //устанавливаем группу приоритета
  NVIC_InitStructure.NVIC_IRQChannel = TIM1_IRQn; //выбираем канал прерывания, который хотим активировать
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0; //выбираем приоритет
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0; //выбираем суб-приоритет
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE; //активируем прерывание
  NVIC_Init(&NVIC_InitStructure); //инициализируем структуру

  //настройка TIM1
  //Сброс всех настроек таймера TIM
  TIM_DeInit(TIM1)
  //Создаем переменную TIM_InitStructure с типом данных TIM_InitTypeDef
```

```
  TIM_InitTypeDef TIM_InitStructure;
  //Описываем структуру TIM_StructInit1
  TIM_StructInit1.TIM_ExtClk = TIM_EXT_CLK_DIS; //Выбираем тактирование от внутренней частоты
  TIM_StructInit1.TIM_ExtEn = TIM_EXT_DIS; //Отключаем использование TMRx_EXTIN в качестве
сигнала разрешения работы таймера
  //Инициализируем структуру TIM_InitStructure (название порта, указатель на структуру)
  TIM_Init(SPI1, &TIM_InitStructure);
  //Разрешение прерывания
  TIM_ITConfig(TIM1, ENABLE);
  //Устанавливаем период перезагрузки таймера на 65536 тактов
  TIM_SetAutoreload(TIM1,0x10000);
  //Разрешаем работу таймера
  TIM_Cmd(TIM1, ENABLE);
}
```

Пример_2

```
uint32_t msTicks=0; //Инициализируем глобальную переменную для счетчика
void SysTick_Handler(void){ //Инициализируем обработчик прерывания
  msTicks++; //Инкрементируем значение переменной
}

int main (void)
{
  SysTick_Config(10000); //Инициализируем таймер (значение Reload = 10000, см. библиотеку TIM)
  SysTick_CLKSourceConfig(SysTick_CLKSource_HCLK_Div2); //Выбираем делитель системной частоты
}
```

1.9 Библиотека таймера WATCHDOG

№	Функция	Описание
Инициализация и конфигурация		
1	IWDG_WriteAccessCmd	Разрешение записи в регистры WDG
2	IWDG_ResetConfig	Разрешение установки сигнала сброса
3	IWDG_ITConfig	Разрешение работы таймера и генерации прерывания
Запись и чтение		
4	IWDG_SetReload	Установка периода перезапуска таймера
5	IWDG_GetCounter	Чтение текущего значения таймера
6	IWDG_GetFlagStatus	Чтение текущего состояния прерывания
7	IWDG_ClearITPendingBit	Сброс прерывания и перезапуск таймера

1.9.1 Инициализация и конфигурация

Функция:

void IWDG_WriteAccessCmd(uint32_t IWDG_WriteAccess);

Разрешение записи в регистры WDG.

Параметры:

WriteAccess – новое состояние разрешения записи в регистры WDT. Принимаемые значения IWDG_WriteAccess_Enable и IWDG_WriteAccess_Disable;

Возвращаемые значения: нет.

Функция:

void IWDG_ResetConfig(FunctionalState NewState);

Разрешение установки сигнала сброса.

Параметры:

NewState – новое состояние разрешения записи в регистры WDT. Принимаемые значения ENABLE и DISABLE;

Возвращаемые значения: нет.

Функция:

void IWDG_ITConfig(FunctionalState NewState);

Разрешение установки сигнала сброса.

Параметры:

NewState – новое состояние разрешения записи в регистры WDT. Принимаемые значения ENABLE и DISABLE;

Возвращаемые значения: нет.

1.9.2 Запись и чтение

Функция:

void IWDG_SetReload(uint32_t Reload);

Установка периода перезапуска таймера.

Параметры:

Reload – период перезапуска таймера. Принимаемые значения: 0 – 4294967295;

Возвращаемые значения: нет.

Функция:

uint32_t IWDG_GetCounter(void);

Чтение текущего значения таймера.

Параметры: нет;

Возвращаемые значения: текущее значение таймера 0 – 4294967295.

Функция:

FlagStatus IWDG_GetFlagStatus(void);

Чтение текущего состояния прерывания.

Параметры: нет**;**

Возвращаемые значения: Состояние прерывания таймера(SET или RESET)..

Функция:

void IWDG_ClearITPendingBit(void);

Сброс прерывания и перезапуск таймера.

Параметры: нет.

Возвращаемые значения: нет.

1.9.3 Применение библиотеки WATCHDOG

После сброса запись в регистры WDT разрешена. Прерывание, счет и разрешение генерации RST отключены.

Пример_1

```
void NMI_Handler(void) //Переопределяем обработчик немаскированного прерывания
{
  if(IWDG_GetFlagStatus() == Set) //Проверяем флаг состояния прерывания по событию
  {
    IWDG_ClearITPendingBit(); //Очистка состояния прерывания и перезапуск таймера
  }
}

int main (void)
{
  IWDG_WriteAccessCmd(IWDG_WriteAccess_Enable); //Разрешение модификации регистров WatchDog
  IWDG_SetReload(0xFFFFF); //Устанавливаем период перезапуска таймера
  IWDG_ResetConfig(ENABLE); //Разрешаем генерацию сигнала Reset
  IWDG_ITConfig(ENABLE); //Разрешаем прерывание и запускаем счетчик
}
```

2. Описание библиотек отладочной платы

2.1 Библиотека функций «Serial»

№	Функция	Описание
Запись и чтение		
12	UART_SendChar	Передача 8 бит данных в буфер передатчика, если он не заполнен
13	UART_GetChar	Чтение 8 бит данных из буфера приемника
14	UART_Print	Последовательная запись массива данных в буфер передатчика с проверкой от переполнения
15	UART_Println	Последовательная передача массива данных с дополнительными кодами 0xA и 0xD (перенос строки)
16	UART_GetInt	Последовательное чтение из буфера приемника целого числа
17	UART_GetString	Последовательное чтение из буфера приемника массива данных
18	UART_IntPrint	Последовательная запись в буфер передатчика целого числа (преобразованного в ASCII)
19	UART_BinPrint	Последовательная запись в буфер передатчика двоичного числа (преобразованного в ASCII)
20	UART_HexPrint	Последовательная запись в буфер передатчика шестнадцатиричного числа (преобразованного в ASCII)

2.1.1 Запись и чтение

Функция:

void UART_SendChar(UART_TypeDef* UARTx, uint8_t data)

Передача 8 бит данных в буфер передатчика, если он не заполнен.

Параметры:

UARTx – где x выбранный порт (1..3);

data **–** 8 бит данных

Возвращаемые значения: нет.

Функция:

uint8_t UART_GetChar(UART_TypeDef* UARTx)

Чтение 8 бит данных из буфера приемника.

Параметры:

UARTx – где x выбранный порт (1..3);

Возвращаемые значения: 8 бит данных.

Функция:

void UART_Print(UART_TypeDef* UARTx, char *Text)

Последовательная запись массива данных в буфер передатчика с проверкой от переполнения.

Параметры:

UARTx – где x выбранный порт (1..3);

****Text –*** ссылка на начало текстового массива

Возвращаемые значения: нет.

Функция:

void UART_Println(UART_TypeDef* UARTx, char *Text)

Последовательная запись массива данных в буфер передатчика с дополнительными кодами 0xA и 0xD (перенос строки)

Параметры:

UARTx – где x выбранный порт (1..3);

****Text –*** ссылка на начало текстового массива

Возвращаемые значения: нет.

Функция:

long UART_GetInt(UART_TypeDef* UARTx)

Последовательное чтение из буфера приемника целого числа.

Параметры:

UARTx – где x выбранный порт (1..3);

Возвращаемые значения: целое положительное число типа long.

Функция:

void UART_GetString(UART_TypeDef* UARTx, unsigned char* s)

Последовательное чтение из буфера приемника массива данных.

Параметры:

UARTx – где x выбранный порт (1..3);

****s –*** ссылка на начало массива.

Возвращаемые значения: нет.

Функция:

void UART_IntPrint (UART_TypeDef* UARTx, unsigned long c)

Последовательная запись в буфер передатчика целого числа (преобразованного в ASCII)

Параметры:

UARTx – где x выбранный порт (1..3);

c – целое положительное число типа long.

Возвращаемые значения: нет.

Функция:

void UART_BinPrint (UART_TypeDef* UARTx, char *Text, uint32_t value, int num)

Последовательная запись в буфер передатчика двоичного числа (преобразованного в ASCII)

Параметры:

UARTx – где x выбранный порт (1..3);

****Text*** – ссылка на начало текстового массива

c – целое положительное число.

num – длина числа (для форматированного вывода необходимой битности)

Возвращаемые значения: нет.

Функция:

void UART_HexPrint (UART_TypeDef* UARTx, char *Text, uint32_t value, int num)

Последовательная запись в буфер передатчика шестнадцатиричного числа (преобразованного в ASCII)

Параметры:

UARTx – где x выбранный порт (1..3);

****Text*** – ссылка на начало текстового массива

c – целое положительное число.

num – длина числа (для форматированного вывода необходимой битности)

Возвращаемые значения: нет.

2.1.2 Применение библиотеки функций серийного порта

Перед использованием функций данной библиотеки, необходимо сконфигурировать необходимый порт UART.

После инициализации структуры возможно использование набора функций.

Пример_1

```
int main(void){
	UART_InitTypeDef UART_InitStructure;
	UART_InitStructure.UART_BaudRate = 9600;
	UART_InitStructure.UART_Mode = UART_Mode_Rx | UART_Mode_Tx;
	UART_Init(UART1, &UART_InitStructure);

	UART_Println(UART1,"hello world");
	UART_HexPrint(UART1,"85 in hex = 0x",85,2);
}
```

2.2 Библиотека функций «Buttons»

№	Функция	Описание
Инициализация и конфигурация		
1	Buttons_Initialize	Инициализация настроек gpio для использования кнопок на плате
2	Buttons_Uninitialize	Сброс настроек gpio
Запись и чтение		
3	Buttons_GetState	Чтение статуса нажатия кнопок на плате

2.2.1 Инициализация и конфигурация

Функция:

int32_t Buttons_Initialize (void);

Инициализация настроек gpio для использования кнопок на плате.

Параметры: нет.

Возвращаемые значения: нет.

Функция:

int32_t Buttons_Uninitialize (void);

Сброс настроек gpio.

Параметры: нет.

Возвращаемые значения: нет.

2.2.2 Запись и чтение

Функция:

uint8_t Buttons_GetState (uint32_t Button_user_pin)

Чтение статуса кнопок на плате.

Параметры:

Button_user_pin – интересуемая кнопка. Принимаемые значения:

GPIO_BUTTON_2 – USW2;

GPIO_BUTTON_3 – USW3;

Возвращаемые значения: нет.

2.2.3 Применение библиотеки функций «Buttons»

Пример_1

```
int main(void){
	Buttons_Initialize();
	if (Buttons_GetState(GPIO_BUTTON_2) == SET) {};
}
```

2.3 Библиотека функций «LCD»

№	Функция	Описание
Инициализация и конфигурация		
1	LCD_Init	Инициализация LCD
Запись и чтение		
2	LCD_Goto	Перенос курсора на соответстующий адрес
3	LCD_ClrDisp	Команда очистки экрана дисплея
4	LCD_RetHome	Перенос курсора в положение (0,0)
5	LCD_SendChar	Передача и вывод символа на позиции курсора дисплея
6	LCD_CreateChar	Запись в память пользовательского символа
7	LCD_Print	Вывод строки символов
8	LCD_Println	Вывод строки символов с переходом на новую.
9	LCD_BinPrint	Вывод двоичного числа (адаптированного под ASCII)
10	LCD_IntPrint	Вывод целого положительного числа (адаптированного под ASCII)

2.3.1 Инициализация и конфигурация

Функция:

void LCD_Init (uint8_t line, uint8_t address);

Инициализация LCD.

Параметры:

line – количество строк дисплея

address – количество столбцов дисплея

Принимаемые значения: ***LCD_Init(2, 16) или LCD_Init(4, 20).***

Возвращаемые значения: нет.

2.3.2 Запись и чтение

Функция:

void LCD_Goto (uint8_t line, uint8_t address);

Перенос курсора на соответстующий адрес.

Параметры:

line – строка. Принимаемые значения: 0..3.

address – столбец. Принимаемые значения: 0..19.

Возвращаемые значения: нет.

Функция:

void LCD_ClrDisp (void);

Команда очистки экрана дисплея.

Параметры: нет.

Возвращаемые значения: нет.

Функция:

void LCD_RetHome (void);

Перенос курсора в положение (0,0).

Параметры: нет.

Возвращаемые значения: нет.

Функция:

void LCD_SendChar (uint8_t ByteToSend);

Передача и вывод символа на позиции курсора дисплея.

Параметры:

ByteToSend – 8 бит данных.

Возвращаемые значения: нет.

Функция:

void LCD_CreateChar (uint8_t value[], int location);

Запись в память пользовательского символа.

Параметры:

value[] – массив из 8 байт данных.

location – порядковый номер символа.

Возвращаемые значения: нет.

Функция:

void LCD_Print (char *Text);

Вывод строки символов.

Параметры:

****Text*** – ссылка на начало текстового массива.

Возвращаемые значения: нет.

Функция:

void LCD_Println (char *Text);

Вывод строки символов с переходом на новую.

Параметры:

****Text*** – ссылка на начало текстового массива.

Возвращаемые значения: нет.

Функция:

void LCD_BinPrint (uint32_t value, int num);

Вывод двоичного числа (адаптированного под ASCII).

Параметры:

value – целое положительное число.

num – длина числа (для форматированного вывода необходимой битности).

Возвращаемые значения: нет.

Функция:

void LCD_IntPrint (unsigned long c);

Вывод целого положительного числа (адаптированного под ASCII).

Параметры:

c – целое положительное число.

Возвращаемые значения: нет.

2.3.3 Применение библиотеки функций «Buttons»

Пример_1

```
int main(void){
	LCD_Init(4,20);
	LCD_ClrDisp();

	uint8_t a[5][8]={
		{0x11,0x11,0x13,0x15,0x19,0x11,0x11,0x00},
		{0x1E,0x01,0x01,0x0F,0x01,0x01,0x1E,0x00},
		{0x0E,0x15,0x15,0x15,0x0E,0x04,0x04,0x00},
		{0x0F,0x11,0x11,0x0F,0x05,0x09,0x11,0x00},
		{0x12,0x12,0x12,0x12,0x12,0x12,0x1F,0x01}
	};

	LCD_CreateChar(a[0],0);
	LCD_CreateChar(a[1],1);
	LCD_CreateChar(a[2],2);
	LCD_CreateChar(a[3],3);
	LCD_CreateChar(a[4],4);

	//delay_ms(2000);
	LCD_Goto(0,4);
	LCD_Print("P");
	LCD_SendChar(0x2);
	LCD_SendChar(0x3);
	LCD_SendChar(0x4);
	LCD_Print(" BH");
	LCD_SendChar(0x0);
	LCD_SendChar(0x0);
	LCD_SendChar(0x1);
	LCD_SendChar(0x2);
	LCD_Goto(2,2);
	LCD_Print("EVALUATION BOARD");
	LCD_Goto(3,4);
	LCD_Print("EB-1914BA018");
}
```

2.4 Библиотека функций «LED»

№	Функция	Описание
Инициализация и конфигурация		
1	LED_Uninitialize	Сброс настроек GPIO для сетодиодов на плате
2	LED_Initialize	Инициализация настроек для светодиодов на плате
Запись и чтение		
3	LED_On	Включить выбранный светодиод
4	LED_Off	Выключить выбранный светодиод

2.4.1 Инициализация и конфигурация

Функция:

int32_t LED_Uninitialize (void);

Сброс настроек GPIO для сетодиодов на плате.

Параметры: нет.

Возвращаемые значения: нет.

Функция:

int32_t LED_Initialize (void);

Инициализация настроек для светодиодов на плате.

Параметры: нет.

Возвращаемые значения: нет.

2.4.2 Запись и чтение

Функция:

int32_t LED_On (uint32_t num);

Включить выбранный светодиод.

Параметры:

num – интересуемый светодиод. Принимаемые значения:

GPIO_LED_X, где X – выбранный светодиод (1..4);

Возвращаемые значения: нет.

Функция:

int32_t LED_Off (uint32_t num);

Выключить выбранный светодиод.

Параметры:

num – интересуемый светодиод. Принимаемые значения:

GPIO_LED_X, где X – выбранный светодиод (1..4);

Возвращаемые значения: нет.

2.4.3 Применение библиотеки функций «LED»

Пример_1

```
int main(void){
	LED_Initialize();
```

```
        LED_On(GPIO_LED_1);
}
```

2.5 Библиотека функций «Flash»

№	Функция	Описание
Инициализация и конфигурация		
1	SPI_FlashInitialize	Инициализация SPI интерфейса для работы с flash
2	SPI_FlashUninitialize	Сброс настроек инициализации SPI
Запись и чтение		
3	SPI_REMS	Чтение Electronic Manufacturer ID & Device ID
4	SPI_WriteEnable	Снятие защиты от записи
5	SPI_WriteDisable	Установка защиты от записи
6	SPI_SectorErase	Очистка сектора памяти
7	SPI_BlockErase	Очистка блока памяти
8	SPI_ChipErase	Полная очистка чипа памяти
9	SPI_DeepPowerDown	Перевод микросхемы памяти в режим низкого потребления
10	SPI_ReleaseFromDPD	Вывод микросхемы памяти из режима низкого потребления
11	SPI_ReadStatus	Чтение статусного регистра микросхемы памяти
12	SPI_Read	Чтение 8 бит данных
13	SPI_Write	Запись 8 бит данных

2.5.1 Инициализация и конфигурация

Функция:

void SPI_FlashInitialize (SPI_TypeDef* SPIx);

Инициализация SPI интерфейса для работы с flash на выборке CS0.

Параметры:

SPIx – где x выбранный порт (1..2);

Возвращаемые значения: нет.

Функция:

void SPI_FlashUninitialize (SPI_TypeDef* SPIx);

Сброс настроек инициализации SPI.

Параметры:

SPIx – где x выбранный порт (1..2);

Возвращаемые значения: нет.

2.5.2 Запись и чтение

Функция:

uint8_t SPI_REMS (SPI_TypeDef* SPIx, uint8_t value);

Чтение Electronic Manufacturer ID & Device ID.

Параметры:

SPIx – где x выбранный порт (1..2);

value – интересуемые данные. Принимаемые значения:

0x00 – Manufacturer ID;

0x01 – Device ID;

Возвращаемые значения: значение ID.

Функция:

void SPI_WriteEnable (SPI_TypeDef* SPIx);

Снятие защиты от записи. Защита устанавливается после каждого цикла записи.

Параметры:

SPIx – где x выбранный порт (1..2);

Возвращаемые значения: нет.

Функция:

void SPI_WriteDisable (SPI_TypeDef* SPIx);

Установка защиты от записи. Защита устанавливается после каждого цикла записи.

Параметры:

SPIx – где x выбранный порт (1..2);

Возвращаемые значения: нет.

Функция:

void SPI_SectorErase (SPI_TypeDef* SPIx, uint32_t address);

Очистка сектора памяти.

Параметры:

SPIx – где x выбранный порт (1..2);

address – адрес, который лежит в необходимом секторе памяти;

Возвращаемые значения: нет.

Функция:

void SPI_BlockErase (SPI_TypeDef* SPIx, uint32_t address);

Очистка блока памяти.

Параметры:

SPIx – где x выбранный порт (1..2);

address – адрес, который лежит в необходимом блоке памяти;

Возвращаемые значения: нет.

Функция:

void SPI_ChipErase (SPI_TypeDef* SPIx);

Полная очистка микросхемы памяти.

Параметры:

SPIx – где x выбранный порт (1..2);

Возвращаемые значения: нет.

Функция:

void SPI_DeepPowerDown (SPI_TypeDef* SPIx);

Перевод микросхемы памяти в режим низкого потребления.

Параметры:

SPIx – где x выбранный порт (1..2);

Возвращаемые значения: нет.

Функция:

void SPI_ReleaseFromDPD (SPI_TypeDef* SPIx);

Вывод микросхемы памяти из режима низкого потребления.

Параметры:

SPIx – где x выбранный порт (1..2);

Возвращаемые значения: нет.

Функция:

uint8_t SPI_ReadStatus (SPI_TypeDef* SPIx);

Чтение статусного регистра микросхемы памяти.

Параметры:

SPIx – где x выбранный порт (1..2);

Возвращаемые значения: 8 бит данных – регистр статуса.

Бит 1	бит 0
1 = запись разрешена 0 = запись запрещена	1 = микросхема занята операцией записи 0 = операция записи не производится

Функция:

uint8_t SPI_Read (SPI_TypeDef* SPIx, uint32_t address);

Чтение 8 бит данных.

Параметры:

SPIx – где x выбранный порт (1..2);

address – адрес;

Возвращаемые значения: 8 бит данных.

Функция:

void SPI_Write (SPI_TypeDef* SPIx, uint32_t address, uint8_t value);

Чтение 8 бит данных.

Параметры:

SPIx – где x выбранный порт (1..2);

address – адрес;

value – 8 бит данных;

Возвращаемые значения: нет.

2.5.3 Применение библиотеки функций «LED»

Пример_1

```
int main(void){
        SPI_FlashInitialize(SPI1);
        SPI_WriteEnable(SPI1);
```

```
        SPI_BlockErase(SPI1,0x0);
        SPI_WriteEnable(SPI1);
        SPI_Write(SPI1,0x0,0x55);
}
```