

1. Список параметров и API функций библиотеки FIL

Раздел содержит список макросов, предоставляемых библиотекой, Раздел содержит полный перечень параметров и функций, какую роль они выполняют,

1.1 Стандартизация названий

По мере пополнения функционала библиотеки для облегченного восприятия команд и быстрого доступа к ним было принято решение - обеспечить стандартизацию названий и разграничения команд по их направленности.

Группы параметров легко выделяются из программного кода, они имеют приставку `__config`, которая сразу дает понять пользователю о принадлежности макроса к группе настроек. Далее идут ключевые слова или подгруппа, основные и критические параметры, в большинстве содержат название *USE*. Их использование сопровождается добавлением чего-либо, поэтому они и называются критическими. Ключевым словом *CALC* обозначаются параметры, включение которых приведет к расширению и добавлению одной и более функций отладки, расчета. Их использование может значительно облегчить отладку разработанного кода.

Некоторые параметры (опциональные) предназначены лишь для детальной настройки конфигурации. Не получив их определения, библиотека продолжит свою работу. Тем не менее, их можно использовать для настройки под специальные условия без низкоуровневого вмешательства через регистровые переменные.

Функции библиотеки FIL выполняют роль не только обертки низкоуровневой части, но и могут выполнять до 15 действий за один вызов. Простые функции можно использовать для установки/очистки ключевых регистров микроконтроллера, инициализацию интерфейсов и прочей



периферии. Библиотека лишь предоставляет набор инструментов для отладки и инициализации, всю основную работу по написанию логики работы программного обеспечения выполняет программист.

1.2 Перечень макросов конфигурации периферии

`__configUSE_RCC`

```
#define __configUSE_RCC (0)
```

Описание: Критически необходимый параметр. Используется линкером для предоставления пользователю доступа к сектору тактирования библиотеки, файл RCC.h

Значение:

0 – доступ для пользователя заблокирован, инструменты нельзя использовать в пользовательском пространстве;

1 – предоставление доступа для пользователя. Можно использовать базовый комплект функций библиотеки для контроллера RCC.

`__configUSE_GPIO`

```
#define __configUSE_GPIO (0)
```

Описание: Критически необходимый параметр. Используется линкером для предоставления пользователю доступа к сектору настройки портов библиотеки, файл GPIO.h

Значение:

0 – доступ для пользователя заблокирован, инструменты нельзя использовать в пользовательском пространстве;

1 – предоставление доступа для пользователя. Можно использовать базовый комплект функций библиотеки для контроллера GPIO.

Примечания: параметр необходим для создания карты портов, потребует включения для вариантов конфигурации, изложенных в разделе 2.

`__configUSE_TIM`

```
#define __configUSE_TIM (0)
```

Описание: Критически необходимый параметр. Используется линкером для предоставления пользователю доступа к сектору настройки таймеров, файл TIM.h



Значение:

0 – доступ для пользователя заблокирован, инструменты нельзя использовать в пользовательском пространстве;

1 – предоставление доступа для пользователя. Можно использовать базовый комплект функций библиотеки для настройки и подключения таймеров.

`__configUSE_USART`

```
#define __configUSE_USART (0)
```

Описание: Критически необходимый параметр. Используется линкером для предоставления пользователю доступа к сектору настройки интерфейса USART, файл USART.h

Значение:

0 – доступ для пользователя заблокирован, инструменты нельзя использовать в пользовательском пространстве;

1 – предоставление доступа для пользователя. Можно использовать базовый комплект функций библиотеки для настройки и подключения USART интерфейса для передачи и приема данных.

`__configUSE_DMA`

```
#define __configUSE_DMA (0)
```

Описание: Критически необходимый параметр. Используется линкером для предоставления пользователю доступа к сектору настройки контроллера прямого доступа к памяти, файл DMA.h

Значение:

0 – доступ для пользователя заблокирован, инструменты нельзя использовать в пользовательском пространстве;

1 – предоставление доступа для пользователя. Можно использовать базовый комплект функций библиотеки для настройки и контроллера прямого доступа DMA.

`__configUSE_I2C`

```
#define __configUSE_I2C (0)
```

Описание: Критически необходимый параметр. Используется линкером для предоставления пользователю доступа к сектору настройки интерфейса I2C, файл I2C.h



Значение:

0 – доступ для пользователя заблокирован, инструменты нельзя использовать в пользовательском пространстве;

1 – предоставление доступа для пользователя. Можно использовать базовый комплект функций библиотеки для настройки и подключения I2C для передачи данных.

`__configUSE_ADC`

```
#define __configUSE_ADC (0)
```

Описание: Критически необходимый параметр. Используется линкером для предоставления пользователю доступа к сектору настройки АЦП, файл ADC.h

Значение:

0 – доступ для пользователя заблокирован, инструменты нельзя использовать в пользовательском пространстве;

1 – предоставление доступа для пользователя. Можно использовать базовый комплект функций библиотеки для настройки и аналого-цифрового преобразователя.

`__configUSE_EXTI`

```
#define __configUSE_EXTI (0)
```

Описание: Критически необходимый параметр. Используется линкером для предоставления пользователю доступа к сектору настройки внешний прерываний, файл EXTI.h

Значение:

0 – доступ для пользователя заблокирован, инструменты нельзя использовать в пользовательском пространстве;

1 – предоставление доступа для пользователя. Можно использовать базовый комплект функций библиотеки для настройки и внешних прерываний.

`__configUSE_RTC`

```
#define __configUSE_RTC (0)
```

Описание: Критически необходимый параметр. Используется линкером для предоставления пользователю доступа к сектору настройки таймера реального времени, файл RTC.h



Значение:

0 – доступ для пользователя заблокирован, инструменты нельзя использовать в пользовательском пространстве;

1 – предоставление доступа для пользователя. Можно использовать базовый комплект функций библиотеки для настройки и использования таймера реального времени RTC.

`__configCALC_RCC`

```
#define __configCALC_RCC (0)
```

Описание: Опциональный параметр. Используется библиотекой FIL для предоставления доступа к расширенному функционалу контроллера RCC, файл RCC.h

Значение:

0 – доступ для пользователя заблокирован, расширенный набор функций не будет добавлен;

1 – предоставление доступа для пользователя. Можно использовать специальный комплект функций библиотеки для расчетов и отладки модуля RCC.

Примечания: параметр может быть задан в случае активации макроса __configUSE_RCC. В противном случае игнорируется и ничего не добавит.

`__configCALC_TIM`

```
#define __configCALC_TIM (0)
```

Описание: Опциональный параметр. Используется библиотекой FIL для предоставления доступа к расширенному функционалу таймеров, файл TIM.h

Значение:

0 – доступ для пользователя заблокирован, расширенный набор функций не будет добавлен;

1 – предоставление доступа для пользователя. Можно использовать специальный комплект функций библиотеки для расчетов и отладки модуля TIM.

Примечания: параметр может быть задан в случае активации макроса __configUSE_TIM. В противном случае игнорируется и ничего не добавит.



`__configCALC_USART`

```
#define __configCALC_USART (0)
```

Описание: Опциональный параметр. Используется библиотекой FIL для предоставления доступа к расширенному функционалу интерфейса USART, файл USART.h

Значение:

0 – доступ для пользователя заблокирован, расширенный набор функций не будет добавлен;

1 – предоставление доступа для пользователя. Можно использовать специальный комплект функций библиотеки для расчетов и отладки модуля USART.

Примечания: параметр может быть задан в случае активации макроса __configUSE_USART. В противном случае игнорируется и ничего не добавит.

```
#define __configCALC_I2C_SCANNING (1)
```

Описание: Опциональный параметр. Добавляет расширенный набор функций отладки интерфейса I2C.

Значение:

0 – Доступ к расширенному набору не будет предоставлен.

1 – Предоставление доступа к расширенному набору функций I2C.

`__configCONVERT_Volts`

```
#define __configCONVERT_Volts (0)
```

Описание: Опциональный параметр. Изменяет выходной параметр данных, преобразуя данные с АЦП в напряжение.

Значение:

0 – выходные данные будут в дискретном формате

1 – выходные данные будут преобразованы в вольты

Примечание: Не рекомендуется проводить прием данных в прерывании по АЦП, уменьшает время реакции системы. Смотри описание функций AnalogReadInjected и AnalogReadRegular.



`__configUSE_Battery_Charging`

```
#define __configUSE_Battery_Charging (0)
```

Описание: Опциональный параметр. Используется для подключения измерения напряжения на батарее платы.

Значение:

0 – Измерение батареи не будет добавлено в очередь обработки АЦП.

1 - Измерение батареи будет добавлено в очередь обработки АЦП.

`__configUSE_Temperature_Sensor`

```
#define __configUSE_Temperature_Sensor (0)
```

Описание: Опциональный параметр. Используется для подключения измерения температуры через встроенный термодатчик.

Значение:

0 – Измерение термодатчика не будет добавлено в очередь обработки АЦП.

1 - Измерение термодатчика будет добавлено в очередь обработки АЦП.

Примечание: Включение параметра `__configUSE_Battery_Charging` имеет высший приоритет, при его включении параметр термодатчика будет игнорирован.

`__configADC_InterruptRequest`

```
#define __configADC_InterruptRequest (1)
```

Описание: Опциональный параметр. Включает прерывание АЦП.

Значение:

0 – После инициализации прерывание АЦП не будет запущено.

1 – Запуск прерывания после инициализации АЦП.

`__configADC_DMAResultRequest`

```
#define __configADC_DMAResultRequest (1)
```

Описание: Опциональный параметр. Выполняет подготовительные действия для работы с контроллером DMA.



Значение:

0 – Инициализация АЦП в обычном режиме.

1 – Запуск совместной работы с контроллером DMA.

__configADC_Divider

```
#define __configADC_Divider (3)
```

Описание: Опциональный параметр. Задаёт величину делителя шины АЦП.

Значение:

0 – Частоты шины АЦП делится на 2 от шины APB2.

1 – Частоты шины АЦП делится на 4 от шины APB2.

2 – Частоты шины АЦП делится на 6 от шины APB2.

3 – Частоты шины АЦП делится на 8 от шины APB2.

__configADC_RESOLUTION

```
#define __configADC_RESOLUTION (12)
```

Описание: Опциональный параметр. Задаёт величину разрешения АЦП.

Значение:

6 – Разрядности АЦП будет настроена на 6 бит.

8 – Разрядности АЦП будет настроена на 8 бит.

10 – Разрядности АЦП будет настроена на 10 бит.

12 – Разрядности АЦП будет настроена на 12 бит.

__configADC_CYCLES

```
#define __configADC_CYCLES (ADC_480_CYCLES)
```

Описание: Опциональный параметр. Указывается величина количества циклов обработки каждого канала АЦП.

Значение:

ADC_3_CYCLES – 3 цикла обработки.

ADC_15_CYCLES – 15 циклов обработки.

ADC_28_CYCLES – 28 циклов обработки.

ADC_56_CYCLES – 56 циклов обработки.



ADC_84_CYCLES – 84 цикла обработки.

ADC_112_CYCLES – 112 циклов обработки.

ADC_144_CYCLES – 144 цикла обработки.

ADC_480_CYCLES – 480 циклов обработки.

__configI2C_FindListSize

```
#define __configI2C_FindListSize    (5)
```

Описание: Опциональный параметр. Указывается максимальное количество устройств, адреса которых можно запоминать в процессе выполнения функции *I2C_FindDevices()*.

Значение: Целое число устройств. Записывается в структуру *I2CStatus*.

__configI2C_TIMEOUT

```
#define __configI2C_TIMEOUT    (20000)
```

Описание: Опциональный параметр. Задаёт величину таймаута – времени ожидания соединения и ключевых флагов шины I2C.

Значение: Значение количества итераций ожидания, указывать значение больше от 5000.

Board_Config

Описание: Главная функция применения настроек. Стандартизированное макроопределение применения настроек. Используется готовый вариант из конфигураций либо написанный пользователем лично.

InitPeriph

Описание: Функция настройки режима работы портов микроконтроллера. Стандартизированное макроопределение применения настроек. Используется готовый вариант из конфигураций либо написанный пользователем лично.

1.3 Перечень API функций

В этом подразделе представлен полный список функций, предоставляемых библиотекой FIL, которые могут быть использованы в пользовательских проектах.



3.3.1 Функции RCC (Reset Clock Configuration)

SetDmax

```
#define SetDMA1 (RCC->AHB1ENR |= ((uint32_t)(1 << 21)))
```

Описание: Функция включает тактирование для выбранного модуля DMA.

Результат: Устанавливает значение 1 в регистре DMAEN.

Примечание: Функция открывает доступ к редактированию регистров модуля DMA.

SetGPIOx

```
#define SetGPIOA (RCC->AHB1ENR |= ((uint32_t)(1)))
```

Описание: Функция включает тактирование для выбранной группы GPIO портов.

Результат: Устанавливает значение 1 в регистре GPIOxEN.

Примечание: Функция открывает доступ к редактированию регистров для включенной группы портов GPIO.

SetTIMx

```
#define SetTIM1 (RCC->APB2ENR |= ((uint32_t)(1)))
```

Описание: Функция включает тактирование для модуля таймера.

Результат: Устанавливает значение 1 в регистре TIMxEN.

Примечание: Функция открывает доступ к редактированию регистров для включенного таймера.

SetUSARTx

```
#define SetUSART1 (RCC->APB2ENR |= ((uint32_t)(1 << 4)))
```

Описание: Функция включает тактирование для модуля USART.

Результат: Устанавливает значение 1 в регистре USARTxEN.

SetI2Cx

```
#define SetI2C1 (RCC->APB1ENR |= ((uint32_t)(1 << 21)))
```

Описание: Функция включает тактирование для модуля I2C.

Результат: Устанавливает значение 1 в регистре I2CxEN.



SetADCx

```
#define SetADC1      (RCC->APB2ENR |= ((uint32_t)(1 << 8)))
```

Описание: Включает тактирование модуля ADC.

Результат: Устанавливает значение 1 в регистре ADCxEN.

SetDACx

```
#define SetDAC      (RCC->APB1ENR |= ((uint32_t)(1 << 29)))
```

Описание: Включает тактирование модуля DAC.

Результат: Устанавливает значение 1 в регистре DACxEN.

SetCANx

```
#define SetCAN1      (RCC->APB1ENR |= ((uint32_t)(1 << 25)))
```

Описание: Включает тактирование модуля CAN.

Результат: Устанавливает значение 1 в регистре CANxEN.

SetPWR

```
#define SetPWR      (RCC->APB1ENR |= ((uint32_t)(1 << 28)))
```

Описание: Включает тактирование модуля PWR.

Результат: Устанавливает значение 1 в регистре PWREN.

SetSPIx

```
#define SetSPI2      (RCC->APB1ENR |= ((uint32_t)(1 << 14)))
```

Описание: Включает тактирование модуля SPI.

Результат: Устанавливает значение 1 в регистре SPIxEN.

SetSDIO

```
#define SetSDIO      (RCC->APB2ENR |= ((uint32_t)(1 << 11)))
```

Описание: Включает тактирование модуля SDIO.

Результат: Устанавливает значение 1 в регистре SDIOEN.

SetSYSCFG

```
#define SetSYSCFG    (RCC->APB2ENR |= ((uint32_t)(1 << 14)))
```

Описание: Включает тактирование модуля SYSCFG.

Результат: Устанавливает значение 1 в регистре SYSCFGEN.



CalcRCCClocks(void)

```
void CalcRCCClocks(void);
```

Описание: Отладочная функция. Позволяет вычислить настроенные частоты основных шин микроконтроллера и отобразить их в структуре Clocks.

Результат: Структура Clocks заполняется значениями частот в герцах.

Name	Value
Clocks	
PLLVC0	336000000
CurrentSYSTICK	84000000
CurrentAHB	84000000
CurrentAPB1	42000000
CurrentAPB2	84000000

3.3.2 Функции GPIO (General Purpose Input/Output)

GPIOPinID(PIN_PORT, PIN)

```
/*!  
* @brief GPIOPinID(PIN_PORT,PIN) - set the custom define with necessary pin address  
* @value PIN_PORT - pins group  
* @value PIN - pin in selected group  
*/  
#define GPIOPinID(PIN_PORT, PIN) ( (PIN_PORT<<4)|PIN )
```

Описание: Базовая функция. Используется в файле-карте портов для создания уникального идентификатора порта. В последствии используется при настройке режима работы портов микроконтроллера.

Результат: Идентификатор порта. Содержит информации о группе и порядковому номеру.

GPIOPinMode(pin, mode)

```
/*!  
* @brief GPIOPinMode(pin,mode) - calculate mode for initialization MODER  
* @value pin - number of pin  
* @value mode - value of MODER for selected pin  
*/  
#define GPIOPinMode(pin, mode) ((uint32_t) ( mode) <<((pin&0xF)<<1))
```

Описание: Базовое макроопределение. Используется в расчетах инициализации портов. Позволяет получить идентификатор режима работы порта.

Результат: Идентификатор режима порта. Содержит информации о режиме работы порта.



РЕСУРСНЫЙ ЦЕНТР
РОБОТОТЕХНИКИ
ДГТУ



GPIOPinType(pin, type)

```
/*!
 * @brief GPIOPinType(pin,type) - calculate type for initialization OTyPER
 * @value pin - number of pin
 * @value type - value of OTyPER for selected pin
 */
#define GPIOPinType(pin, type) ((uint32_t) ( type) <<((pin&0xF)))
```

Описание: Базовое макроопределение. Используется в расчетах инициализации портов. Позволяет получить идентификатор типа подключения порта. Необходим в функции инициализации.

Результат: Идентификатор типа подключения порта. Содержит информацию о типе подключения.

GPIOPinSpeed(pin, speed)

```
/*!
 * @brief GPIOPinSpeed(pin,speed) - calculate speed for initialization OSPEEDR
 * @value pin - number of pin
 * @value speed - value of OSPEEDR for selected pin
 */
#define GPIOPinSpeed(pin, speed) ((uint32_t) ( speed) <<((pin&0xF)<<1))
```

Описание: Базовое макроопределение. Используется в расчетах инициализации портов. Позволяет получить идентификатор скорости работы порта. Необходим в функции инициализации.

Результат: Идентификатор скорости работы порта. Содержит информацию о типе подключения.

GPIOPinPull(pin, pull)

```
/*!
 * @brief GPIOPinPull(pin,pull) - calculate pu/pd for initialization PUPDR
 * @value pin - number of pin
 * @value pull - value of PUPDR for selected pin
 */
#define GPIOPinPull(pin, pull) ((uint32_t) ( pull) <<((pin&0xF)<<1))
```

Описание: Базовое макроопределение. Используется в расчетах инициализации портов. Позволяет получить идентификатор типа подключения порта. Необходим в функции инициализации.

Результат: Идентификатор типа подключения порта. Содержит информацию о типе подключения.

GPIOPinAF(pin, af)



```

/*!
 * @brief GPIOPinAF(pin,af) - calculate mode for initialization AFR
 * @value pin - number of pin
 * @value af - value of AFR for selected pin
 */
#define GPIOPinAF(pin,af) ((uint32_t) (af) <<((pin&0x7)<<2))

```

Описание: Базовое макроопределение. Используется в расчетах инициализации портов. Позволяет получить идентификатор типа подключения порта. Необходим в функция инициализации.

Результат: Идентификатор типа подключения порта. Содержит информацию о типе подключения.

GPIOClearMode(pin)

```

/*!
 * @brief GPIOClearMode(pin) - clear MODER on selected pin
 * @value pin
 */
#define GPIOClearMode(pin) (*((uint32_t *) (GPIO_base(pin)))) &= ~GPIOPinMode(pin, 0x3)

```

Описание: Базовое макроопределение. Используется в расчетах инициализации портов. Позволяет получить идентификатор типа подключения порта. Необходим в функция инициализации.

Результат: Идентификатор типа подключения порта. Содержит информацию о типе подключения.

GPIOClearType(pin)

```

/*!
 * @brief GPIOClearType(pin) - clear OTYPER on selected pin
 * @value pin
 */
#define GPIOClearType(pin) (*((uint32_t *) (GPIO_base(pin)+ 0x04)) &= ~GPIOPinType(pin, 0x1)

```

Описание: Базовое макроопределение. Используется в расчетах инициализации портов. Позволяет получить идентификатор типа подключения порта. Необходим в функция инициализации.

Результат: Идентификатор типа подключения порта. Содержит информацию о типе подключения.

GPIOClearSpeed(pin)

```

/*!
 * @brief GPIOClearSpeed(pin) - clear OSPEEDR on selected pin
 * @value pin
 */
#define GPIOClearSpeed(pin) (*((uint32_t *) (GPIO_base(pin) + 0x08)) &= ~GPIOPinSpeed(pin, 0x3))

```



Описание: Базовое макроопределение. Используется в расчетах инициализации портов. Позволяет получить идентификатор типа подключения порта. Необходим в функция инициализации.

Результат: Идентификатор типа подключения порта. Содержит информацию о типе подключения.

GPIOClearPull(pin)

```
/*!
 * @brief GPIOClearPULL(pin) - clear PUPDR on selected pin
 * @value pin
 */
#define GPIOClearPull(pin) (*((uint32_t *) (GPIO_base(pin) + 0x0C)) &= ~GPIOPinPull(pin, 0x3))
```

Описание: Базовое макроопределение. Используется в расчетах инициализации портов. Позволяет получить идентификатор типа подключения порта. Необходим в функция инициализации.

Результат: Идентификатор типа подключения порта. Содержит информацию о типе подключения.

GPIOClearConf(pin)

```
/*!
 * @brief GPIOClearConf(pin) - clear all registers on selected pin
 * @value pin
 */
#define GPIOClearConf(pin) {\
    GPIOClearMode(pin); \
    GPIOClearType(pin); \
    GPIOClearSpeed(pin); \
    GPIOClearPull(pin); }
```

Описание: Базовое макроопределение. Используется в расчетах инициализации портов. Позволяет получить идентификатор типа подключения порта. Необходим в функция инициализации.

Результат: Идентификатор типа подключения порта. Содержит информацию о типе подключения.

GPIOSetMode(pin, mode)

```
/*!
 * @brief GPIOSetMode(pin, mode) - set MODER on selected pin
 * @value pin
 * @value mode - set the mode in register MODER
 */
#define GPIOSetMode(pin,mode) (*((uint32_t *) (GPIO_base(pin))) |= GPIOPinMode(pin, mode))
```

Описание: Базовое макроопределение. Используется в расчетах инициализации портов. Позволяет получить идентификатор типа подключения порта. Необходим в функция инициализации.



Результат: Идентификатор типа подключения порта. Содержит информацию о типе подключения.

GPIOSetType(pin, type)

```
/*!
 * @brief GPIOSetType(pin, type) - set OYPER on selected pin
 * @value pin
 * @value type - set the mode in register OYPER
 */
#define GPIOSetType(pin,type) (*((uint32_t *) (GPIO_base(pin)+ 0x04)) |= GPIOPinType(pin, type))
```

Описание: Базовое макроопределение. Используется в расчетах инициализации портов. Позволяет получить идентификатор типа подключения порта. Необходим в функция инициализации.

Результат: Идентификатор типа подключения порта. Содержит информацию о типе подключения.

GPIOSetSpeed(pin, speed)

```
/*!
 * @brief GPIOSetSpeed(pin, speed) - set OSPEEDR on selected pin
 * @value pin
 * @value speed - set the mode in register OSPEEDR
 */
#define GPIOSetSpeed(pin,speed)((*(uint32_t *) (GPIO_base(pin)+ 0x08)) |= GPIOPinSpeed(pin, speed))
```

Описание: Базовое макроопределение. Используется в расчетах инициализации портов. Позволяет получить идентификатор типа подключения порта. Необходим в функция инициализации.

Результат: Идентификатор типа подключения порта. Содержит информацию о типе подключения.

GPIOSetPull(pin, pull)

```
/*!
 * @brief GPIOSetPull(pin, pull) - set PUPDR on selected pin
 * @value pin
 * @value pull - set the mode in register PUPDR
 */
#define GPIOSetPull(pin,pull) (*((uint32_t *) (GPIO_base(pin)+ 0x0C)) |= GPIOPinPull(pin, pull))
```

Описание: Базовое макроопределение. Используется в расчетах инициализации портов. Позволяет получить идентификатор типа подключения порта. Необходим в функция инициализации.

Результат: Идентификатор типа подключения порта. Содержит информацию о типе подключения.



GPIOSetConf(pin, mode, type, speed, pull)

```
/*!
 * @brief GPIOConfPin(pin, mode, type, speed, pull) - set properties on selected pin
 * @value pin
 * @value mode - set the mode in register MODER
 * @value type - set type of working pin in register OTYPER
 * @value speed - set speed in register OSPEEDR
 * @value pull - set the pull in register PUPDR
 */
#define GPIOSetConf(pin, mode, type, speed, pull) {\
    GPIOSetMode(pin, mode);\
    GPIOSetType(pin, type);\
    GPIOSetSpeed(pin, speed);\
    GPIOSetPull(pin, pull);\
}
```

Описание: Базовое макроопределение. Используется в расчетах инициализации портов. Позволяет получить идентификатор типа подключения порта. Необходим в функции инициализации.

Результат: Идентификатор типа подключения порта. Содержит информацию о типе подключения.

GPIOConfMode(pin, mode)

```
/*!
 * @brief GPIOConfMode(pin, mode) - clear and set MODER on selected pin
 * @value pin
 * @value mode - set the mode in register MODER
 */
#define GPIOConfMode(pin, mode) {*((uint32_t *) (GPIO_base(pin)+0x00)) = \
    (((*(uint32_t *) (GPIO_base(pin)+0x00))&(~GPIOPinMode(pin, 0x3)))|GPIOPinMode(pin, mode));}
```

Описание: Базовое макроопределение. Используется в расчетах инициализации портов. Позволяет получить идентификатор типа подключения порта. Необходим в функции инициализации.

Результат: Идентификатор типа подключения порта. Содержит информацию о типе подключения.

GPIOConfType(pin, type)

```
/*!
 * @brief GPIOConfType(pin, type) - clear and set OTYPER on selected pin
 * @value pin
 * @value type - set type of working pin in register OTYPER
 */
#define GPIOConfType(pin, type) {*((uint32_t *) (GPIO_base(pin)+0x04)) = \
    (((*(uint32_t *) (GPIO_base(pin)+0x04))&(~GPIOPinType(pin, 0x1)))|GPIOPinType(pin, type));}
```

Описание: Базовое макроопределение. Используется в расчетах инициализации портов. Позволяет получить идентификатор типа подключения порта. Необходим в функции инициализации.



Результат: Идентификатор типа подключения порта. Содержит информацию о типе подключения.

GPIOConfSpeed(pin, speed)

```
/*!
 * @brief GPIOConfSpeed(pin, speed) - clear and set OSPEEDR on selected pin
 * @value pin
 * @value speed - set speed in register OSPEEDR
 */
#define GPIOConfSpeed(pin, speed) {*((uint32_t *) (GPIO_base(pin)+0x08)) = \
    (((uint32_t *) (GPIO_base(pin)+0x08))&(~GPIOPinSpeed(pin, 0x3)))|GPIOPinSpeed(pin, speed));}
```

Описание: Базовое макроопределение. Используется в расчетах инициализации портов. Позволяет получить идентификатор типа подключения порта. Необходим в функция инициализации.

Результат: Идентификатор типа подключения порта. Содержит информацию о типе подключения.

GPIOConfPull(pin, pull)

```
/*!
 * @brief GPIOConfPull(pin, pull) - clear and set PUPDR on selected pin
 * @value pin
 * @value pull - set the pull in register PUPDR
 */
#define GPIOConfPull(pin, pull) {*((uint32_t *) (GPIO_base(pin)+0x0C)) = \
    (((uint32_t *) (GPIO_base(pin)+0x0C))&(~GPIOPinPull(pin, 0x3)))|GPIOPinPull(pin, pull);}
```

Описание: Базовое макроопределение. Используется в расчетах инициализации портов. Позволяет получить идентификатор типа подключения порта. Необходим в функция инициализации.

Результат: Идентификатор типа подключения порта. Содержит информацию о типе подключения.

GPIOConfPin(pin, mode, type, speed, pull)

```
/*!
 * @brief GPIOConfPin(pin, mode, type, speed, pull) - clear and set properties on selected pin
 * @value pin
 * @value mode - set the mode in register MODER
 * @value type - set type of working pin in register OTYPER
 * @value speed - set speed in register OSPEEDR
 * @value pull - set the pull in register PUPDR
 */
#define GPIOConfPin(pin, mode, type, speed, pull) {\
    GPIOConfMode(pin, mode)           \
    GPIOConfType(pin, type)           \
    GPIOConfSpeed(pin, speed)         \
    GPIOConfPull(pin, pull)           \
}
```

Описание: Базовое макроопределение. Используется в расчетах инициализации портов. Позволяет получить идентификатор типа подключения порта. Необходим в функция инициализации.



Результат: Идентификатор типа подключения порта. Содержит информацию о типе подключения.

GPIOConfAF(pin, af)

```
/*!
 * @brief GPIOConfAF(pin, af) - clear and set alternate function for necessary mode of pin
 * @value pin
 * @value af - set alternate function with necessary mode
 */
#define GPIOConfAF(pin, af) {*((uint32_t *) (GPIO_base(pin)+0x20+((pin&0x08)>>1))) = \
    (((uint32_t *) (GPIO_base(pin)+0x20+((pin&0x08)>>1)))&(~GPIOPinAF(pin,0xF)))|GPIOPinAF(pin,af));}
```

Описание: Базовое макроопределение. Используется в расчетах инициализации портов. Позволяет получить идентификатор типа подключения порта. Необходим в функция инициализации.

Результат: Идентификатор типа подключения порта. Содержит информацию о типе подключения.

SetPin(pin)

```
/*!
 * @brief SetPin(pin) - set pin to logic high potential (>= 1.08 V)
 * @value pin
 */
#define SetPin(pin) (*((uint32_t *) (GPIO_base(pin) + 0x18)) = ((uint32_t)1<<(((uint32_t)pin&0x000000f)))
```

Описание: Базовое макроопределение. Используется в расчетах инициализации портов. Позволяет получить идентификатор типа подключения порта. Необходим в функция инициализации.

Результат: Идентификатор типа подключения порта. Содержит информацию о типе подключения.

ResetPin(pin)

```
/*!
 * @brief ResetPin(pin) - reset pin to logic low potential(< 1.08 V)
 * @value pin
 */
#define ResetPin(pin) (*((uint32_t *) (GPIO_base(pin) + 0x18)) = (uint32_t)1<<(((uint32_t)pin&0xf)+0x00000010))
```

Описание: Базовое макроопределение. Используется в расчетах инициализации портов. Позволяет получить идентификатор типа подключения порта. Необходим в функция инициализации.

Результат: Идентификатор типа подключения порта. Содержит информацию о типе подключения.



TooglePin(pin)

```
/*!
 * @brief TooglePin(pin) - toggle pin to logic low/high potential
 * @value pin
 */
#define TooglePin(pin) { \
    if(PinVal(pin)) ResetPin(pin); \
    else SetPin(pin); \
}
```

Описание: Макроопределение, которое устанавливает противоположный логический уровень на указанном выходе.

Результат: -

PinVal(pin)

```
/*!
 * @brief PinVal(pin) - check the input status of selected pin(logical type of value)
 * @value pin
 */
#define PinVal(pin) (((uint32_t *)(GPIO_base(pin) + 0x10))&((uint32_t)1<<(pin&0xf)))
```

Описание: Базовое макроопределение. Используется в расчетах инициализации портов. Позволяет получить идентификатор типа подключения порта. Необходим в функция инициализации.

Результат: Идентификатор типа подключения порта. Содержит информацию о типе подключения.

PinOut(pin)

```
/*!
 * @brief PinOut(pin) - check the output status of selected pin(logical type of value)
 * @value pin
 */
#define PinOut(pin) (((uint32_t *)(GPIO_base(pin) + 0x14))&((uint32_t)1<<(pin&0xf)))
```

Описание: Базовое макроопределение. Используется в расчетах инициализации портов. Позволяет получить идентификатор типа подключения порта. Необходим в функция инициализации.

Результат: Идентификатор типа подключения порта. Содержит информацию о типе подключения.



3.3.3 Функции TIM (Timer Interface Module)

TimPWMConfigure(TIMx, PSC, ARR, ch1, ch2, ch3, ch4)

```
/*!
 * @brief TimPWMConfigure(TIM, prescaler, autoreset, ch1, ch2, ch3, ch4) - configuration timer in PWM mode
 * @arg TIM - number of timer
 * @arg prescaler - value for increase the input frequency
 * @arg autoreset - value for set the period of timer
 * @arg ch<n> - activate/deactivate the channels of timer
 */
#define TimPWMConfigure(TIM, PSC, ARR, ch1, ch2, ch3, ch4) {\
    ConfChannelsTim(TIM, 6, 6, ch1, ch2, ch3, ch4);\
    ConfTimPSC(TIM, PSC);\
    ConfTimARR(TIM, ARR);\
    TimStart(TIM);\
    SetTimMainOutput(TIM);\
    ResetTimCCR1(TIM);\
    ResetTimCCR2(TIM);\
}
```

Описание: Функция инициализации таймера. Настраивает таймер на генерацию импульсов ШИМ с заданной периодичностью.

Входные параметры: *TIMx* – Номер таймера.

PSC – Значение предделителя таймера.

ARR – Значение периода таймера.

chx – Включение каналов. Для включения задать аргумент равным 1, 0 не включать канал в работу.

Результат: Таймер будет генерировать импульсы с заданной частотой. Для изменения скважности необходимо обращаться к регистрам CCRx или функции SetVoltage.

TimPIDConfigure(TIMx, PSC, ARR)

```
/*!
 * @brief TimPIDConfigure(TIM, PSC, ARR) - configuration timer in calculating mode
 * (P-regulation, PI - regulation, PID - regulation and other regulations)
 * @arg TIM - number of timer
 * @arg PSC - Prescaler value timer
 * @arg ARR - Autoreset value timer
 */
#define TimPIDConfigure(TIM, PSC, ARR) {\
    ResetTimCNT(TIM);\
    ConfTimPSC(TIM, PSC);\
    ConfTimARR(TIM, ARR);\
    SetTimUpdateInterrupt(TIM);\
    TimStart(TIM);\
}
```

Описание: Функция инициализации таймера. Настраивает таймер на генерацию прерываний с заданной периодичностью. Можно использовать для расчетов.

Входные параметры: *TIMx* – Номер таймера.

PSC – Значение предделителя таймера.



ARR – Значение периода таймера.

Результат: Таймер будет генерировать прерывания с заданной частотой. Используется обработчик прерываний для наполнения последовательностью действий.

TimPWMConfigureAutomatic(TIMx, FREQUENCY, ch1, ch2, ch3, ch4)

```
/*!
 * @brief TimPWMConfigureAutomatic(TIM,FREQUENCY,ch1,ch2,ch3,ch4) - configuration timer in PWM mode (auto-mode)
 * @arg TIM - number of timer
 * @arg FREQUENCY - target frequency
 * @arg ch<n> - activate/deactivate the channels of timer
 */
#define TimPWMConfigureAutomatic(TIM,FREQUENCY,ch1,ch2,ch3,ch4) {\
    ConfChannelsTim(TIM,6,ch1,ch2,ch3,ch4);\
    CalcTimFrequency(TIM,FREQUENCY);\
    TimStart(TIM);\
    SetTimMainOutput(TIM);\
    ResetTimCCR1(TIM);\
    ResetTimCCR2(TIM);\
}
```

Описание: Функция инициализации таймера. Настраивает таймер на генерацию импульсов ШИМ с заданной периодичностью. В качестве входного аргумента используется частота, автоматический режим.

Входные параметры: *TIMx* – Номер таймера.

FREQUENCY – необходимая частота в герцах.

chx – Включение каналов. Для включения задать аргумент равным 1, 0 не включать канал в работу.

Результат: Таймер будет генерировать импульсы с заданной частотой. Для изменения скважности необходимо обращаться к регистрам CCRx или функции SetVoltage. Упрощенная функция, не требует вычисления пользователем предделителя и периода.

Примечание: требуется включение параметра __configCALC_TIM.

TimPIDConfigureAutomatic(TIMx, FREQUENCY)

```
/*!
 * @brief TimPIDConfigureAutomatic(TIM,frequency) - configuration timer in calculating mode (auto-mode)
 * (P-regulation, PI - regulation, PID - regulation and other regulations)
 * @arg TIM - number of timer
 * @arg FREQUENCY - target frequency for calculating
 */
#define TimPIDConfigureAutomatic(TIM,FREQUENCY) {\
    ResetTimCNT(TIM);\
    CalcTimFrequency(TIM,FREQUENCY);\
    SetTimUpdateInterrupt(TIM);\
    TimStart(TIM);\
}
```

Описание: Функция инициализации таймера. Настраивает таймер на генерацию прерываний с заданной периодичностью. Можно использовать для расчетов. Требуется в качестве входного аргумента частоту, автоматический режим.

Входные параметры: *TIMx* – Номер таймера.



РЕСУРСНЫЙ ЦЕНТР
РОБОТОТЕХНИКИ
ДГТУ



FREQUENCY – требуемая частота таймера в герцах.

Результат: Таймер будет генерировать прерывания с заданной частотой. Используется обработчик прерываний для наполнения последовательностью действий. Не требует вычислений предделителя и периода.

Примечание: требуется включение параметра __configCALC_TIM.

TimEncoderConfigure(TIMx)

```
/*!
 * @brief TimEncoderConfigure(TIM) - configuration timer in encoder mode
 * @arg TIM - number of timer
 */
#define TimEncoderConfigure(TIM) {\
    ResetTimPSC(TIM);           \
    ConfTimARR(TIM,0xFFFFFFFF); \
    TimStart(TIM);              \
    ConfTimSMS(TIM,0x3);        \
    ConfTimEtf(TIM,0xF);        \
}
```

Описание: Функция инициализации таймера. Настраивает таймер на работу в режиме энкодера. В этом режиме таймер принимает импульсы от углового энкодера, накапливается в счетчике таймера.

Входные параметры: *TIMx* – Номер таймера.

Результат: В регистре CNT таймера хранится значение счетчика импульсов. Используется в расчетах.

TimPWMInputCaptureConfigure(TIM, PSC, ARR)

```
/*!
 * @brief TimPWMInputCaptureConfigure(TIM) - configuration timer for check length of PWM
 * @arg TIM - number of timer
 */
#define TimPWMInputCaptureConfigure(TIM,PSC,ARR) {\
    ConfTimPSC(TIM,PSC);           \
    ConfTimARR(TIM,ARR);           \
    ConfTimCaptureSelection1(TIM,0x1,0x2,1); \
    ConfTimCaptureSelection2(TIM,0x1,0x2,1); \
    ConfTimCapturePolarity(TIM,0x0,0x0,0x0,0x0,1,1); \
    ConfTimCaptureComplementaryPolarity(TIM,0x0,0x0,0x0,0x0,1,1); \
    ConfTimTriggerSelection(TIM,0x5); \
    ConfTimSMS(TIM,0x4);           \
    ConfTimCapture(TIM,0x1,0x1,0x1,0x1,1,1,1,1); \
    TimStart(TIM);                 \
}
```

Описание: Функция инициализации таймера. Настраивает таймер на работу в режиме измерения прямоугольных импульсов.

Входные параметры: *TIMx* – Номер таймера.



PSC – Значение предделителя таймера.

ARR – Значение периода таймера.

Результат: В регистрах CCR1 и CCR2 будут храниться значения периода и длины импульса. Можно использовать для измерений и расчетов.

TimPWMCentralAlignedModeConfigure(TIM, PSC, ARR, ch1, ch2, polarity1, ch3, ch4, polarity2)

```
/*!
 * @brief TimPWMCentralAlignedModeConfigure(TIM, prescaler, autoreset, ch1, ch2,
 *                                           polarity1, ch3, ch4, polarity2) - configuration timer with central aligned mode
 *
 * @arg TIM - number of timer
 * @arg prescaler - value for increase the input frequency
 * @arg autoreset - value for set the period of timer
 * @arg polarity<n> - set polarity level for PWM output
 * @arg ch<n> - activate/deactivate the channels of timer
 */
#define TimPWMCentralAlignedModeConfigure(TIM, PRESCALER, AUTORESET, ch1, ch2, polarity1, ch3, ch4, polarity2) {\
    ConfChannelsTim(TIM, polarity1, polarity2, ch1, ch2, ch3, ch4); \
    ConfTimPSC(TIM, PRESCALER); \
    ConfTimARR(TIM, AUTORESET); \
    TimStart(TIM); \
    ResetTimCCR1(TIM); \
    ResetTimCCR2(TIM); \
}
```

Описание: Функция инициализации таймера. Настраивает таймер на режим генерации импульсов с центрированием.

Входные параметры: *TIMx* – Номер таймера.

PSC – Значение предделителя таймера.

ARR – Значение периода таймера.

chx – Включение каналов. Для включения задать аргумент равным 1, 0 не включать канал в работу.

polarityX – полярность половины импульса. Если 0 то импульс положительной полярности, 1 – отрицательной.

Результат: Таймер будет генерировать импульсы с заданной частотой. Для изменения скважности необходимо обращаться к регистрам CCRx или функции SetVoltage.

CalcTimClockSource(TIMx)

```
/*!
 * @brief CalcTimClockSource(TIM_TypeDef *TIMx) - Calculating Timer Clock Source
 *
 * @arg TIMx - number of timer
 */
void CalcTimClockSource(TIM_TypeDef *TIMx);
```



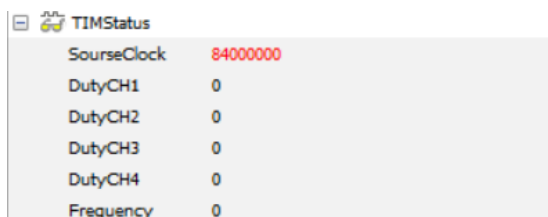
РЕСУРСНЫЙ ЦЕНТР
РОБОТОТЕХНИКИ
ДГТУ



Описание: Базовая и отладочная функция. Позволяет вычислить используемый таймером ресурс.

Входные параметры: *TIMx* – номер таймера.

Результат: Структура TIMStatus будет заполнена частотой выбранного таймера. Значение используется в функциях, работающих в автоматическом режиме.



SourceClock	84000000
DutyCH1	0
DutyCH2	0
DutyCH3	0
DutyCH4	0
Frequency	0

Примечание: требуется включение параметра __configCALC_TIM.

CalcTimStatus(TIMx)

```
/*!
 * @brief CalcTimStatus(TIM_TypeDef *TIMx) - Calculating Timer Status
 * @arg TIMx - number of timer
 */
void CalcTimStatus(TIM_TypeDef *TIMx);
```

Описание: Отладочная функция. Позволяет получить текущее состояние таймера – используемых ресурсов, частоту и скважность каналов.

Входные параметры: *TIMx* – номер таймера.

Результат: Структура TIMStatus будет заполнена параметрами частоты таймера, ресурса и скважностью каналов.

Примечание: требуется включение параметра __configCALC_TIM.

CalcTimFrequency(TIMx, freq)

```
/*!
 * @brief CalcTimFrequency(TIM_TypeDef *TIMx, uint16_t freq) - Calculating Timer frequency
 * @arg TIMx - number of timer
 * @arg freq - necessary frequency
 */
void CalcTimFrequency(TIM_TypeDef *TIMx, uint16_t freq);
```

Описание: Базовая функция. Вычисляет коэффициенты для настройки таймера в автоматическом режиме.

Входные параметры: *TIMx* – номер таймера.

freq – требуемая частота работы таймера.



РЕСУРСНЫЙ ЦЕНТР
РОБОТОТЕХНИКИ
ДГТУ



Результат: Коэффициенты таймера будут выставлены на основе требуемой частоты.

Примечание: требуется включение параметра `__configCALC_TIM`.

SetVoltage(Duty)

```
bool SetVoltage(float Duty);
```

Описание: Опциональная функция. Позволяет управлять драйвером двигателя постоянного тока по требуемому значению скважности.

Входные аргументы: *Duty* – требуемая скважность.

Результат: Вращение двигателя. Коэффициент может быть отрицательным, но в диапазоне $-1.0 \leq Duty \leq 1.0$.

Примечание: требуется включение параметра `__configCALC_TIM`.

ServoInit(Servo, servoType, TIMx, ms)

```
void ServoInit(Servomotor* Servo, char servoType, TIM_TypeDef *TIMx, uint16_t ms);
```

Описание: Опциональная функция. Инициализирует серводвигатель перед его использованием.

Входные аргументы: *Servo* – указатель на структуру, в которой будут храниться настройки для серводвигателя.

servoType – тип серводвигателя. Может быть равен PDI6225MG_300 или MG996R (модели серводвигателей).

Результат: После выполнения серводвигатель готов к работе.

Примечание: требуется включение параметра `__configCALC_TIM`.

ServoSetRange(Servo, min_angle, max_angle)

```
void ServoSetRange(Servomotor* Servo, uint16_t min_angle, uint16_t max_angle);
```

Описание: Опциональная функция. Устанавливает ограничение на вращения серводвигателем.

Входные аргументы: *Servo* – указатель на структуру с настройками.

min_angle – минимальное пороговое значения угла серводвигателя в градусах.

max_angle – максимальное пороговое значение угла серводвигателя в градусах.



Результат: Настройки будут применены и изменены в структуре.

Примечание: требуется включение параметра __configCALC_TIM.

SetServoAngle(Servo, angle)

```
void SetServoAngle(Servomotor* Servo, uint16_t angle);
```

Описание: Опциональная функция. Устанавливает ограничение на вращения серводвигателем.

Входные аргументы: *Servo* – указатель на структуру с настройками.

angle – значение угла, на которое нужно повернуть серводвигатель в градусах

Результат: Серводвигатель начнет вращение на указанное количество градусов.

Примечание: требуется включение параметра __configCALC_TIM.

delay_ms(ticks)

Описание: Опциональная функция. Позволяет установить задержку в миллисекундах.

Входные аргументы: *ticks* – значение задержки в миллисекундах.

Результат: Программа прекратит выполнение программы пока не пройдет определенное количество времени

Время выполнения: зависимость от значения. Использовать в конструкции while.

```
void delay_ms(uint32_t ticks)
{
    if(ticks == 0) return;
    startTick = globalTime;
    while((globalTime - startTick) < ticks);
}
```

Примечание: требуется включение параметра __configCALC_TIM.

StartMeas(void)

```
uint32_t StartMeas(void);
```

Описание: Отладочная функция. Объявляет старт для начала

3.3.4 Функции ADC (Analog-Digital Converter)



РЕСУРСНЫЙ ЦЕНТР
РОБОТОТЕХНИКИ
ДГТУ



ADCAAllScanConfigure(ADC, LENGTH, CH1, CH2, CH3, CH4, CH5, CH6, CH7, CH8, CH9, CH10, CH11, CH12, CH13, CH14, CH15, CH16)

```
/*!  
*  @brief  ADCAAllScanConfigure(ADC, LENGTH,  
*          CH1, CH2, CH3, CH4, CH5, CH6, CH7, CH8, CH9, CH10, CH11, CH12, CH13, CH14, CH15, CH16)  
*          - Configuration all channels ADC (Scan mode)  
*          @arg ADC - Number of ADC  
*          @arg LENGTH - Regular channel sequence length  
*          @arg CHx - Connect ADC_IN pins to conversion queue  
*          @arg NUMBER - NUMBER of ADC_IN active pins  
*/  
#define ADCAAllScanConfigure(ADC, LENGTH,  
    CH1, CH2, CH3, CH4, CH5, CH6, CH7, CH8, CH9, CH10, CH11, CH12, CH13, CH14, CH15, CH16){\  
    SetADCLength(ADC, LENGTH); \  
    ConfADCSeq1(ADC, CH1, CH2, CH3, CH4); \  
    ConfADCSeq2(ADC, CH5, CH6, CH7, CH8, CH9, CH10); \  
    ConfADCSeq3(ADC, CH11, CH12, CH13, CH14, CH15, CH16); \  
    SetADCScan(ADC); \  
    SetADCDDS(ADC); \  
    SetADCDMA(ADC); \  
    SetADCCont(ADC); \  
    SetADCAdon(ADC); \  
    ADCStartRegular(ADC); \  
}
```

Описание: Функция инициализации АЦП. Позволяет обрабатывать несколько регулярных каналов.

Входные параметры: *ADC* – номер используемого АЦП.

LENGTH – Количество необходимых каналов для обработки АЦП. Задается целым числом от 0 до 16

CHx – слот для указания канала обработки. Указывается номер *ADC_IN* входа микроконтроллера. Различается в зависимости от серии контроллера.

Результат: Инициализирует АЦП в режиме сканирования указанных регулярных каналов. Получение данных реализуется через функцию *ConnectADCTODMA*.

Примечание: Исходя из документации на микроконтроллеры *ST*, считывание нескольких регулярных каналов возможно только через преобразование *DMA*. Является основной при значении параметра *__configADC_Mode = 2*.

ADCSimpleConfigure(ADC)




```

/*!
 * @brief ADCSimpleConfigure(ADC) - Configuration ADC simple
 * @arg ADC - Number of ADC
 *
 */
#define ADCSimpleConfigure(ADC) { \
    SetADCADon(ADC); \
    SetADCCont(ADC); \
    SetADCScan(ADC); \
    (__configADC_InterruptRequest == 1) ? SetADCRegularInterrupt(ADC) : \
    ResetADCRegularInterrupt(ADC); \
    (__configADC_InterruptRequest == 1) ? SetADCInjectedInterrupt(ADC) : \
    ResetADCInjectedInterrupt(ADC); \
    ADCStartRegular(ADC); \
    while(!(ADC->SR & ADC_SR_EOC)) {} \
    if(CheckADCJStart(ADC) == 1) ADCStartInjected(ADC); \
    while(!(ADC->SR & ADC_SR_EOC)) {} \
}

```

Описание: Функция инициализации АЦП в режиме сканирования без добавления каналов обработки. Можно добавлять один регулярный и четыре инжектированных канала.

Входные параметры: *ADC* – номер используемого АЦП.

Результат: Инициализирует работу АЦП в простом режиме. Для получения данных рекомендуется обратиться к регистрам DR и JDRx.

Примечание: Может быть использована в режиме ручного добавления. Является основной в режимах __configADC_Mode = 1,3,4.

ADCAddRegularChannel(ADC, CHANNEL, CYCLES, RCH)

```

/*!
 * @brief ADCAddRegularChannel(ADC,CHANNEL,CYCLES) - Adding new channel to conversions ADC
 * @arg ADC - Number of ADC
 * @arg CHANNEL - Number of target channel
 * @arg CYCLES - Sample time selection
 *
 */
#define ADCAddRegularChannel(ADC,CHANNEL,CYCLES,RCH) { \
    ADC->SQR1 &= ~(0xF << 20); \
    ADC->SQR1 |= CH << 20; \
    *(&ADC->SQR3 - (CH / 6)) |= CHANNEL << ((CH * 5) % 30); \
    *(&ADC->SMPR2 - ((CH / 10)*0x4)) |= CYCLES << ((CH * 3) % 30); \
    RCH++; \
}

```

Описание: Функция добавляет регулярный канал для обработки.

Входные параметры: *ADC* – номер используемого АЦП.

CHANNEL – Необходимый канал для обработки.

CYCLES – Количество циклов преобразования АЦП.

RCH – Технический параметр



Результат: В регистры SQR SMPR добавляются данные для обработки нового регулярного канала АЦП.

Примечание: RCH необходимо объявить static uint8_t RCH при использовании вне функции ADC_Init. Количество циклов – необязательный параметр, при установлении глобального параметра __configADC_CYCLES значение аргумента можно приравнять.

ADCAddSingleChannel(ADC, CHANNEL, CYCLES)

```
/*!
 * @brief AddADCSingleChannel(ADC,CHANNEL,CYCLES) - Adding single channel to conversions ADC
 * @arg ADC - Number of ADC
 * @arg CHANNEL - Number of target channel
 * @arg CYCLES - Sample time selection
 * @attention This function delete old channel from ADC list before writing new target channel
 */
#define ADCAddSingleChannel(ADC,CHANNEL,CYCLES) {\
    ADC->CR2 &= ~ADC_CR2_ADON;\
    ADC->SQR3 &= ~(0x1F);\
    ADC->SMPR2 &= ~(0x7);\
    ADC->SQR3 |= CHANNEL;\
    ADC->SMPR2 |= CYCLES;\
    ADC->CR1 &= ~ADC_CR1_SCAN;\
    ADC->CR2 |= (ADC_CR2_ADON | ADC_CR2_CONT);}
```

Описание: Функция инициализации АЦП. Можно добавить один регулярный канал для обработки.

Входные параметры: ADC – номер используемого АЦП.

CHANNEL – регулярный канал, необходимый для обработки.

CYCLES – количество циклов преобразования для выбранного канала.

Результат: Добавляет один регулярный канал для обработки АЦП.

Примечание: Количество циклов – необязательный параметр, при установлении глобального параметра __configADC_CYCLES значение аргумента можно приравнять. Является основной функцией инициализации при режиме __configADC_Mode = 1, однако, может быть использована и в других режимах кроме 2.

ADCAddInjectedChannel(ADC, CHANNEL, CYCLES, JCH)



```

/*!
 * @brief ADCAddInjectedChannel(ADC,CHANNEL,CYCLES) - Adding new injected channel to conversions ADC
 * @arg ADC - Number of ADC
 * @arg CHANNEL - Number of target injected channel
 * @arg CYCLES - Sample time selection
 */
#define ADCAddInjectedChannel(ADCx,CHANNEL,CYCLES, JCH)
ADCx->JSQR &= ~(0xF << 20);
ADCx->JSQR |= (JCH << 20);
ADCx->JSQR |= (CHANNEL << (((JCH + 3) * 5) % 30));
*(&ADCx->SMPR2 - ((JCH / 10)*0x4)) |= (CYCLES << (JCH * 3) % 30);
if(JCH == 3) JCH = 0;
JCH++;
}

```

Описание: Добавляет инжектированный канал для обработки АЦП.

Входные параметры: *ADC* – номер используемого АЦП.

CHANNEL – регулярный канал, необходимый для обработки.

CYCLES – количество циклов преобразования для выбранного канала.

JCH – Технический параметр

Результат: Добавляет инжектированный канал обработки АЦП.

Примечание: Количество циклов – необязательный параметр, при установлении глобального параметра `__configADC_CYCLES` значение аргумента можно приравнять. *JCH* необходимо объявить `static uint8_t JCH` при использовании вне функции `ADC_Init`.

ADCAddInjectedGroup(ADC, NUMBER, J1, J2, J3, J4)

```

/*!
 * @brief ADCAddInjectedGroups(ADC,NUMBER,J1,J2,J3,J4) -
 * @arg ADC - Number of ADC
 * @arg NUMBER -
 * @arg Jx -
 */
#define ADCAddInjectedGroup(ADC,NUMBER,J1,J2,J3,J4)
ADC->JSQR &= ~(0x3FFFFFFF);
ADC->JSQR |= (NUMBER << 20);
ADC->JSQR |= ((J1) | (J2 << 5) | (J3 << 10) | (J4 << 15));
SetADCContInjected(ADC);
}

```

Описание: Добавляет группу инжектированных каналов АЦП.

Входные параметры: *ADC* – номер используемого АЦП.

NUMBER – количество добавляемых инжектированных каналов.

Jx – слот для указания канала обработки. Указывается номер *ADC_IN* входа микроконтроллера. Различается в зависимости от серии контроллера.

Результат: Добавляет один или несколько инжектированных каналов.



ADC_Multiplexer_Get(ADCx)

```
/*!
 * @brief ADC_Multiplexer_Get(ADC_TypeDef *ADCx) - Calculating ADC Data from multiplexor
 * @arg ADCx - number of ADC
 * @arg isConvert - optional parameter to convert data to Volts
 */
void ADC_Multiplexer_Get(ADC_TypeDef *ADCx, bool isConvert);
```

Описание: Функция для считывания данных с мультиплексора/мультиплексоров.

Входные параметры: *ADC* – номер используемого АЦП.

Результат: Считывает данные с мультиплексора и помещает их в структуру ADCStatus.

ADCStatus		ADCStatus	
ADCSOURCE	"ADC1"	ADCSOURCE	"ADC1"
BATTERY	0	BATTERY	0
TEMPERATURE	0	TEMPERATURE	0
Multiplexer1		Multiplexer1	
[0]	616	[0]	0.77746582
[1]	546	[1]	0.808081031
[2]	582	[2]	0.829028308
[3]	624	[3]	0.831445336
[4]	593	[4]	0.828222632
[5]	594	[5]	0.793579102
[6]	585	[6]	0.787133813
[7]	588	[7]	0.815332055
Multiplexer2		Multiplexer2	
adc_data		adc_data	

Примечание: При режиме работы АЦП __configADC_Mode = 4 можно считывать данные с двух мультиплексоров или одного мультиплексора с 4 дополнительными каналами.

ADC_Simple_Get(ADCx)

```
/*!
 * @brief ADC_Simple_Get(ADC_TypeDef *ADCx) - Calculating ADC Data from multiplexor
 * @arg TIMx - number of timer
 */
void ADC_Simple_Get(ADC_TypeDef *ADCx);
```

Описание: Считывает данные с одного регулярного канала.

Входные параметры: *ADC* – номер используемого АЦП.



Результат: Считывает данные и помещает в структуру ADCStatus в параметр adc_data.

ADC_Init(ADCx)

```
/*!
 * @brief ADC_Init(ADC_TypeDef *ADCx) - main initialization function for ADC
 * @arg ADCx - number of ADC
 */
void ADC_Init(ADC_TypeDef *ADCx);
```

Описание: Функция инициализации АЦП. При вызове выполняет сценарий инициализации в зависимости от параметра *__configADC_Mode*.

Входные параметры: *ADC* – номер используемого АЦП.

Результат: Инициализирует АЦП и выставляет необходимый режим работы.

AnalogReadRegular(void)

```
/*!
 * @brief AnalogReadRegular(void) - parsing data from regular channel
 *
 */
void AnalogReadRegular(void);
```

Описание: Функция считывает данные с регулярного канала АЦП.

Результат: Данные помещаются в структуру ADCStatus, параметр adc_data.

AnalogReadInjected(ADCx)

```
/*!
 * @brief AnalogReadInjected(void) - parsing data from injected channels
 * @brief ADCx - number of ADC
 */
void AnalogReadInjected(ADC_TypeDef *ADCx);
```

Описание: Функция считывает данные с инжектированных каналов АЦП.

Входные параметры: *ADC* – номер используемого АЦП.

Результат: Данные помещаются в структуру ADCStatus, параметр adc_data или MultiplexerX.



3.3.5 Функции USART (Universal Asynchronous Receiver-Transmitter)

USARTReceiverConfigure(USART, BAUD, RXInterrupt)

```
/*!
 * @brief USARTReceiverConfigure(USART,BAUD,RXInterrupt) - receiver config
 * @arg USART - number of UART/USART interface
 * @arg BAUD - speed of transmission in bauds
 * @arg RXInterrupt - interrupt when RxE = 1
 */
#define USARTReceiverConfigure(USART,BAUD,RXInterrupt){\
    ConfUSARTBaud(USART,BAUD);\
    SetUSARTReceiver(USART);\
    if(RXInterrupt == 1) SetUSARTRXInterrupt(USART);\
    USARTStart(USART);\
}
```

Описание: Функция инициализации интерфейса USART в режиме приемника данных.

Входные параметры: *USARTx* – номер используемого USART.

BAUD – при значении параметра `__configCALC_USART = 1` скорость в бодах, иначе значение, записываемое в регистр BRR.

RXInterrupt - включение прерывания по приему данных. Для включения задать равным 1.

Результат: Инициализирует работу USART в режиме приемника данных.

USARTTransmitterConfigure(USART, BAUD, TXInterrupt)

```
/*!
 * @brief USARTTransmitterConfigure(USART,BAUD,TXInterrupt) - transmitter config
 * @arg USART - number of UART/USART interface
 * @arg BAUD - speed of transmission in bauds
 * @arg TXInterrupt - interrupt when TxE = 1
 */
#define USARTTransmitterConfigure(USART,BAUD,TXInterrupt){\
    ConfUSARTBaud(USART,BAUD);\
    SetUSARTTransmitter(USART);\
    if(TXInterrupt == 1) SetUSARTTXInterrupt(USART);\
    USARTStart(USART);\
}
```

Описание: Функция инициализации интерфейса USART в режиме отправки данных.

Входные параметры: *USARTx* – номер используемого USART.



BAUD – при значении параметра `__configCALC_USART = 1` скорость в бодах, иначе значение, записываемое в регистр BRR.

TXInterrupt - включение прерывания по отправке данных. Для включения задать равным 1.

Результат: инициализирует USART в режиме отправки данных.

USARTBothConfigure(USART,BAUD, TXInterrupt, RXInterrupt)

```
/*!
 * @brief USARTBothConfigure(USART,BAUD,TXInterrupt, RXInterrupt) - both config
 * @arg USART - number of UART/USART interface
 * @arg BAUD - speed of transmission in bauds
 * @arg RXInterrupt - interrupt when RxE = 1
 * @arg TXInterrupt - interrupt when TxE = 1
 */
#define USARTBothConfigure(USART,BAUD,TXInterrupt, RXInterrupt){\
    ConfUSARTBaud(USART,BAUD);\
    SetUSARTTransmitter(USART);\
    SetUSARTReceiver(USART);\
    if(RXInterrupt == 1) SetUSARTRXInterrupt(USART);\
    if(TXInterrupt == 1) SetUSARTTXInterrupt(USART);\
    USARTStart(USART);\
}
```

Описание: Функция инициализации интерфейса USART в режиме приемника данных и отправки.

Входные параметры: *USARTx* – номер используемого USART.

BAUD – при значении параметра `__configCALC_USART = 1` скорость в бодах, иначе значение, записываемое в регистр BRR.

TXInterrupt - включение прерывания по отправке данных. Для включения задать равным 1.

RXInterrupt - включение прерывания по приему данных. Для включения задать равным 1.

Результат: возвращает число `uint16_t baudrate`, которое необходимо записывать в регистр BRR.

CalcUSARTBaudrate(USARTx, BaudRate)



```

/*!
 * @brief CalcUSARTBaudrate(USART_TypeDef *USARTx, uint32_t BaudRate) - automatic calculating bauds
 * @arg USARTx - number of UART/USART
 * @arg BaudRate - necessary baudrate
 * @return uint16_t baudrate
 */
uint16_t CalcUSARTBaudrate(USART_TypeDef *USARTx, uint32_t BaudRate);

```

Описание: Функция автоматического расчета бод для интерфейса USART.

Входные параметры: *USARTx* – номер используемого USART.

BaudRate – требуемая скорость в бодах.

Результат: возвращает число `uint16_t baudrate`, которое необходимо записывать в регист BRR.

3.3.6 Функции I2C (Inter Integrated Circuit)

I2CSimpleConfigure(I2C, SPEED)

```

/*!
 * @brief I2CSimpleConfigure(I2C,SPEED) - configuration I2C without adding addresses
 * @arg I2C - number of I2C interface
 * @arg SPEED - speed mode interface (SLOW or FAST)
 * @list I2C_Slow - slow mode I2C
 *       I2C_Fast - fast mode I2C
 */
#define I2CSimpleConfigure(I2C, SPEED) {\
    SetI2CPeriphDisable(I2C);           \
    ConfI2CFreq(I2C,I2CDefaultFREQ);    \
    ConfI2CCCR(I2C, SPEED);              \
    if(SPEED == I2C_Slow) ConfI2CTrise(I2C,RiseTimeDefaultSM); \
    if(SPEED == I2C_Fast) ConfI2CTrise(I2C,RiseTimeDefaultFM); \
    SetI2CPeriphEnable(I2C);            \
}

```

Описание: Функция инициализации I2C в обычном режиме. Совпадает с функцией *I2CMasterConfigure*

Входные параметры: *I2C* – номер используемого I2C.

SPEED – скорость работы шины I2C. Может равняться значениям *I2C_Fast* или *I2C_Slow*.

Результат: инициализирует шину I2C в режиме мастера.

I2CMasterConfigure(I2C, SPEED, ADDRESS)




```

/*!
 * @brief I2CMasterConfigure(I2C,SPEED, ADDRESS) - configuration I2C in master mode
 * @arg I2C - number of I2C interface
 * @arg SPEED - speed mode interface (SLOW or FAST)
 * @list I2C_Slow - slow mode I2C
 *        I2C_Fast - fast mode I2C
 * @arg ADDRESS - Own address on bus I2C
 */
#define I2CMasterConfigure(I2C, SPEED, ADDRESS) {\
    ConfI2CFreq(I2C,I2CDefaultFREQ);\
    ConfI2CCCR(I2C, SPEED);\
    SetI2CPeriphDisable(I2C);\
    if(SPEED == I2C_Slow) { SetI2CMasterModeSlow(I2C);\
                           ConfI2CTrise(I2C,RiseTimeDefaultSM);\}\
    if(SPEED == I2C_Fast) {ConfI2CTrise(I2C,RiseTimeDefaultFM);\
                           SetI2CMasterModeFast(I2C);\}\
    SetI2CPeriphEnable(I2C);\
    ConfI2CAddress(I2C, ADDRESS);\
}

```

Описание: Функция инициализации I2C в обычном режиме. Совпадает с функцией *I2CSimpleConfigure*.

Входные параметры: *I2C* – номер используемого I2C.

SPEED – скорость работы шины I2C. Может равняться значениям *I2C_Fast* или *I2C_Slow*.

ADDRESS – адрес устройства, работающего в режиме мастера.

Результат: инициализирует шину I2C в режиме мастера.

I2CSlaveConfigure(I2C, SPEED, ADDRESS)

Функция в разработке.

I2C_ClearAllStats(I2Cx)

```

/*!
 * @brief I2C_ClearAllStats(I2C_TypeDef* I2Cx) - Clear all active flags and parameters
 * @arg I2Cx - number of target I2C
 * @note [FIL:I2C] Frequency and rise time current bus will not be cleared
 */
void I2C_ClearAllStats(I2C_TypeDef* I2Cx);

```

Описание: Функция очистки всех флагов статуса текущей шины I2C.

Входные параметры: *I2Cx* – номер используемого I2C.

Результат: Очищает все регистры статусов SR1 и SR2 шины I2C.

I2C_SingleSend(I2Cx, Byte, Write)



```

/*!
 * @brief I2C_SingleSend(I2C_TypeDef* I2Cx, uint8_t Byte, bool IsWrite) - Single Write on Bus IIC
 * @arg I2Cx - number of target I2C
 * @arg Byte - Sending value
 * @arg IsWrite - necessary Write/Read bytes on next transfer:
 *               0 - Reads next transfer
 *               1 - Write by slave on next transfer
 */
bool I2C_SingleSend(I2C_TypeDef* I2Cx, uint8_t Byte, bool IsWrite);

```

Описание: Функция отправки одного байта устройству по шине I2C.

Входные параметры: *I2Cx* – номер используемого I2C.

Byte – байт данных.

Write – направление передачи.

Результат: Возвращает true если байт был успешно отправлен, false если возникли проблемы с отправкой.

I2C_MultipleSend(I2Cx, bufBytes)

```

/*!
 * @brief I2C_MultipleSend(I2C_TypeDef* I2Cx, uint8_t *bufBytes) - Multiple Write on Bus IIC
 * @arg I2Cx - number of target I2C
 * @arg bufBytes - Sending buffer of values
 */
uint16_t I2C_MultipleSend(I2C_TypeDef* I2Cx, uint8_t *bufBytes);

```

Описание: Функция отправки нескольких байт устройству по шине I2C.

Входные параметры: *I2Cx* – номер используемого I2C.

bufBytes – буфер данных, который будет отправлен.

Результат: Возвращает количество успешно отправленных байт информации.

I2C_SingleRead(I2Cx)

```

/*!
 * @brief I2C_SingleRead(I2C_TypeDef* I2Cx) - Single read data from I2C bus
 * @arg I2Cx - number of I2C
 */
uint8_t I2C_SingleRead(I2C_TypeDef* I2Cx);

```

Описание: Функция чтения одного байта по шине I2C.

Входные параметры: *I2Cx* – номер используемого I2C.



Результат: Возвращает true если байт был успешно отправлен, false если возникли проблемы с отправкой.

I2C_MultipleRead(I2Cx, bufBytes)

```
/*!  
 * @brief I2C_MultipleRead(I2C_TypeDef* I2Cx, uint8_t *bufBytes) - Multiple Read on Bus IIC  
 * @arg I2Cx - number of target I2C  
 * @arg bufBytes - Sending buffer of values  
 * @return success received bytes  
 */  
uint16_t I2C_MultipleRead(I2C_TypeDef* I2Cx, uint8_t *bufBytes);
```

Описание: Функция чтения нескольких байт с шины I2C.

Входные параметры: *I2Cx* – номер используемого I2C.

bufBytes – буфер для хранения данных.

Результат: Возвращает количество байт, успешно записанных в буфер.

I2CFindDevices(I2Cx)

Описание: Функция сканирования шины I2C на наличие готовых к общению устройств.

Входные параметры: *I2Cx* – номер используемого I2C.

Результат: Записывает в структуру I2CStatus адреса найденных устройств.

Время выполнения: для 5 устройств не более 700 мс.

3.3.7 Функции DMA (Data Memory Access)

ConnectADCTODMA(EVENT, PRIORITY, MEMORY, TXINTERRUPT)



```

/*!
 * @brief DMAConnectTOADC(PRIORITY, MEMORY, TXINTERRUPT) -> Connect DMA to ADC
 * @arg PRIORITY - Priority of target channel
 * @arg MEMORY - Buffer address for storing values
 * @arg TXINTERRUPT - Interrupt selection mode: 0.5 - half transferred interrupt
 *                                     1 - transferred complete interrupt
 *                                     2 - transferred error interrupt
 *
 * @version STM32F401xx
 */
#define ConnectADCTODMA(PRIORITY, MEMORY, TXINTERRUPT)
{
    SetDMAChannel(DMA2,ADC1Req);
    SetDMAPAR(DMA2,ADC1Req,(&ADC1->DR));
    SetDMAMEMORY0(DMA2,ADC1Req,(&MEMORY));
    ClearDMADir(DMA2,ADC1Req);
    SetDMANDTR(DMA2,ADC1Req,ADC1_NUMB);
    ClearDMAPinc(DMA2,ADC1Req);
    SetDMAMinc(DMA2,ADC1Req);
    SetDMACirc(DMA2,ADC1Req);
    SetDMApsize(DMA2,ADC1Req,HALFWORD);
    SetDMAmsize(DMA2,ADC1Req,HALFWORD);
    SetDMAPriority(DMA2,ADC1Req,PRIORITY);
    if(TXINTERRUPT == 1) SetDMATransferIT(DMA2,ADC1Req);
    if(TXINTERRUPT == 0.5) SetDMAHFTTransferIT(DMA2,ADC1Req);
    if(TXINTERRUPT == 2) SetDMATransferErrIT(DMA2,ADC1Req);
    DMAStart(DMA2,ADC1Req);
}

```

Описание: Опциональная функция. Позволяет включить обработку каналов АЦП через контроллер прямого доступа к памяти. Автоматически обновляет информацию с АЦП и добавляет в стандартный буфер файла DMA_FIFOBuffers.h.

Входные параметры: *PRIORITY* – Приоритет обработки канала DMA. Может принимать значения – LOW_P, MEDIUM_P, HIGH_P, VeryHigh_P.

MEMORY – Буфер для хранения результатов преобразований. Выбирается из стандартных в файле DMA_FIFOBuffers.h.

TXINTERRUPT – Подключение прерывания по преобразованию:

0 – прерывание не будет включено.

0.5 – прерывание по преобразованию половины каналов АЦП.

1 – прерывание по завершению преобразования каналов АЦП.

2 – прерывание по ошибке преобразования каналов АЦП.

Результат: Массив ADCx_Data автоматически заполняется данными с АЦП.

ConnectUSARTTODMA(DMA, EVENT, PRIORITY, MEMORY, TXINTERRUPT)



РЕСУРСНЫЙ ЦЕНТР
РОБОТОТЕХНИКИ
ДГТУ



```

/*!
 * @brief DMAConnectUSART(EVENT, PRIORITY, MEMORY, TXINTERRUPT) -> Connect DMA to UART/USART
 * @arg DMA - Number of DMA
 * @arg EVENT - UART/USART Event
 * @arg PRIORITY - Priority of target channel
 * @arg MEMORY - Buffer address for storing values
 * @arg TXINTERRUPT - Interrupt selection mode: 0.5 - half transferred interrupt
 *                                     1 - transferred complete interrupt
 *                                     2 - transferred error interrupt
 *
 * @version STM32F40_41xxx
 */
#define ConnectUSARTDMA(DMA,EVENT, PRIORITY, MEMORY, TXINTERRUPT) {\
    SetDMAChannel(DMA,EVENT);\
    if(EVENT == USART1RXReq || EVENT == USART1TXReq) SetDMAPAR(DMA,EVENT,(&USART1->DR));\
    if(EVENT == USART2RXReq || EVENT == USART2TXReq) SetDMAPAR(DMA,EVENT,(&USART2->DR));\
    if(EVENT == USART3RXReq || EVENT == USART3TXReq) SetDMAPAR(DMA,EVENT,(&USART3->DR));\
    if(EVENT == UART4RXReq || EVENT == UART4TXReq) SetDMAPAR(DMA,EVENT,(&UART4->DR));\
    if(EVENT == UART5RXReq || EVENT == UART5TXReq) SetDMAPAR(DMA,EVENT,(&UART5->DR));\
    if(EVENT == USART6RXReq || EVENT == USART6TXReq) SetDMAPAR(DMA,EVENT,(&USART6->DR));\
    SetDMAMEMORY0(DMA,EVENT,(&MEMORY));\
    if(EVENT == USART1RXReq || EVENT == USART2RXReq || EVENT == USART3RXReq ||\
        EVENT == UART4RXReq || EVENT == UART5RXReq || EVENT == USART6RXReq) ClearDMADir(DMA,EVENT);\
    else SetDMADir(DMA,EVENT,MemoryToPeripheral);\
    SetDMANDTR(DMA,EVENT,sizeof(MEMORY));\
    ClearDMAPinc(DMA,EVENT);\
    SetDMAMinc(DMA,EVENT);\
    SetDMACirc(DMA,EVENT);\
    SetDMAPsize(DMA,EVENT,BYTE);\
    SetDMAMsize(DMA,EVENT,BYTE);\
    SetDMAPriority(DMA,EVENT,PRIORITY);\
    if(TXINTERRUPT == 1) SetDMATransferIT(DMA,EVENT);\
    if(TXINTERRUPT == 0.5) SetDMAHFTTransferIT(DMA,EVENT);\
    if(TXINTERRUPT == 2) SetDMATransferErrIT(DMA,EVENT);\
    DMAStart(DMA,EVENT);\
}

```

Описание: Опциональная функция. Позволяет включить обработку модуля USART через контроллер прямого доступа к памяти. Автоматически обновляет информацию с USART и добавляет в стандартный буфер файла DMA_FIFOBuffers.h.

Входные параметры: *DMA* – Номер контроллера DMA.

EVENT – Событие, необходимое для преобразования. Для модуля USART значение может быть следующее: USARTxRXReq или USARTxTXReq.

PRIORITY – Приоритет обработки канала DMA. Может принимать значения – LOW_P, MEDIUM_P, HIGH_P, VeryHigh_P.

MEMORY – Буфер для хранения результатов преобразований. Выбирается из стандартных в файле DMA_FIFOBuffers.h.

TXINTERRUPT – Подключение прерывания по преобразованию:

0 – прерывание не будет включено.

0.5 – прерывание по преобразованию половины данных USART.

1 – прерывание по завершению преобразования данных USART.

2 – прерывание по ошибке преобразования данных USART.

Результат: Массив USART_Data автоматически заполняется данными.

3.3.8 Функции для работы с регуляторами

Раздел находится в разработке



РЕСУРСНЫЙ ЦЕНТР
РОБОТОТЕХНИКИ
ДГТУ



3.3.9 Функции для работы с матрицами

@добавить функции. раздел готов

3.3.10 Функции EXTI (External Interrupts)

Раздел находится в разработке

3.3.11 Функции RTC (Real-Time Clock)

@добавить функции. раздел готов

3.3.12 Функции SPI (Serial Peripheral Interface)

Раздел находится в разработке

3.3.13 Функции DAC (Digital-Analog Converter)

Раздел находится в разработке

3.3.14 Функции CAN (Controller Area Network)

Раздел находится в разработке

