1. **MDR1636\_write\_byte**

**Резюме**

**static** uint32\_t **MDR1636\_write\_byte**(uint32\_t address, uint32\_t data)- функция записи байта в Flash память.

**Описание**

Данная функция позволяет осуществить запись одного байта по любому адресу Flash памяти. Первым аргументом передается адрес, по которому требуется осуществить запись, вторым аргументом передается непосредственно передаваемый байт. Функция возвращает статус выполнения операции. Если возвращаемое значение равно 0, операция записи байта в память выполнена успешно. Другое значение будет свидетельствовать об ошибке при записи.

**Пример**

uint32\_t op\_result = *OpSuccess*;

uint32\_t exit = 0;

uint32\_t temp32u;

uint32\_t i;

**while**(!exit)

{

// Unlock bypass

op\_result = MDR1636\_write\_byte(MDR1636\_BUS\_ADDR\_555, MDR1636\_BUS\_CODE\_AA);

**if** (op\_result != *OpSuccess*) **break**;

op\_result = MDR1636\_write\_byte(MDR1636\_BUS\_ADDR\_2AA, MDR1636\_BUS\_CODE\_55);

**if** (op\_result != *OpSuccess*) **break**;

op\_result = MDR1636\_write\_byte(MDR1636\_BUS\_ADDR\_555, MDR1636\_BUS\_CODE\_20);

**if** (op\_result != *OpSuccess*) **break**;

}

1. **MDR1636\_read\_word**

**Резюме**

**static** uint32\_t **MDR1636\_read\_word**(uint32\_t address, uint32\_t \*data)- функция чтения 32-битного слова из Flash памяти.

**Описание**

Данная функция позволяет осуществить чтение одного 32-битного слова по любому адресу Flash памяти. В качестве первого аргумента передается адрес младшего байта слова. Вторым аргументом передается указатель на адрес, по которому требуется сохранить прочитанное слово. Функция возвращает статус выполнения операции: 0 – операция выполнена успешно, 1 – ошибка.

**Пример**

uint32\_t op\_result = *OpSuccess*;

uint32\_t dev\_status = *FS\_Error*;

uint32\_t w0, w1;

uint32\_t exit = 0;

// Read four bytes

op\_result = MDR1636\_read\_word(0x00, &w0);

**if** (op\_result != *OpSuccess*) **break**;

1. **MDR1636\_init**

**Резюме**

uint32\_t **MDR1636\_init**(flash\_drv\_config\_t \*drv\_config, **void** \*hw\_config)- функция инициализации драйвера работы процессора с памятью «Миландр «1636РР4У»

**Описание**

Данная функция инициализирует драйвер для осуществления обмена данными между процессором 1967ВН28 и Flash памятью 1636РР4У. Перед использованием остальных функций драйвера необходимо всегда инициализировать драйвер данной функцией.

**Пример**

**int** **main**(){

uint32\_t \*ptr;

uint32\_t \*ptr\_buff;

uint32\_t i = 0;

uint32\_t result;

**static** flash\_drv\_config\_t driver\_config;

ptr\_src = (uint32\_t \*)&input\_data;

driver\_config.page\_size = //value

driver\_config.sector\_count = //value

driver\_config.sector\_size = //value

result = MDR1636\_init(&driver\_config, 0);

result = MDR1636\_reset();

result = MDR1636\_erase\_sector(0);

result = MDR1636\_write\_data(0x0, ptr, 64);

}

1. **MDR1636\_reset**

**Резюме**

uint32\_t **MDR1636\_reset**(**void**)- функция включение режима Reset

**Описание**

Данная функция передает командную последовательность для входа в режим Reset. Функция не имеет входных аргументов и возвращает статус выполнения операции. 0- операция выполнена успешно, 1 – ошибка. **Пример**

**int** **main**(){

**static** flash\_drv\_config\_t driver\_config;

result = MDR1636\_init(&driver\_config, 0);

result = MDR1636\_reset();

**return** 0;

}

1. **MDR1636\_reset\_UnlockBypass**

**Резюме**

uint32\_t **MDR1636\_reset\_UnlockBypass**(**void**)- функция сброса режима UnlockBypass

**Описание**

Данная функция передает командную последовательность для выполнения выхода из режима Unlock Bypass. Функция не имеет входных аргументов и возвращает статус выполнения операции. 0- операция выполнена успешно, 1 – ошибка.

**Пример**

**int** **main**(){

**static** flash\_drv\_config\_t driver\_config;

result = MDR1636\_init(&driver\_config, 0);

result = MDR1636\_reset\_UnlockBypass();

**return** 0;

}

1. **MDR1636\_get\_status**

**Резюме**

uint32\_t **MDR1636\_get\_status**(uint32\_t \*status) – функция чтения статуса Flash памяти

**Описание**

Данная функция получает текущий статус Flash памяти. В качестве аргумента передается указатель на переменную, в которую нужно сохранить значение статуса. Функция возвращает статус выполнения операции: 0 – операция выполнена успешно, 1 – ошибка.

**Пример**

uint32\_t op\_result;

uint32\_t dev\_status;

**do** {

op\_result = MDR1636\_get\_status(&dev\_status);

} **while** ((op\_result == *OpSuccess*) && (dev\_status == *FS\_Busy*));

1. **MDR1636\_wait\_device**

**Резюме**

uint32\_t **MDR1636\_wait\_device**(**void**)- функция ожидания выполнения операции

**Описание**

Данная функция содержит цикл ожидания завершения выполнения операции Flash памятью. Пока операция не завершится, выход из функции не будет выполнен. Функция считывает статус выполнения операции и после её завершения цикл ожидания прекращается. Функция возвращает статус выполнения: 0 – функция выполнена корректно, 1 – ошибка. Входных аргументов функция не имеет.

**Пример**

uint32\_t op\_result;

op\_result = MDR1636\_start\_erase\_sector(sa);

**if** (op\_result == *OpSuccess*)

{

op\_result = MDR1636\_wait\_device();

}

1. **MDR1636\_read\_info**

**Резюме**

uint32\_t **MDR1636\_read\_info**(uint32\_t param, uint32\_t \*data)- функция чтения информации об устройстве

**Описание**

Функция считывает и возвращает служебную информацию из FLASH. Используется для считывания ID кода микросхемы. Команда может вызываться многократно во время работы с FLASH. Функция имеет два входных аргумента: param – идентификатор считываемого параметра, \*data – указатель на переменную, через которую возвращается результат чтения. Функция возвращает статус выполнения операции: 0 – чтение информации выполнено успешно, 1 – ошибка.

**Пример**

**#define** MDR1636PP4\_SECTOR\_SIZE 0x40000

**#define** MDR1636PP4\_SECTOR\_COUNT 8

**#define** MDR1636PP4\_VENDOR\_ID 0x01

**#define** MDR1636PP4\_DEVICE\_ID 0xC8

**int** **main**(){

uint32\_t id\_info;

**static** flash\_drv\_config\_t driver\_config;

result = MDR1636\_init(&driver\_config, 0);

result = MDR1636\_read\_info(MDR1636PP4\_DEVICE\_ID, &id\_info);

}

1. **MDR1636\_get\_sector\_protection**

**Резюме**

uint32\_t **MDR1636\_get\_sector\_protection**(uint32\_t sa, uint32\_t \*value)- функция чтения статуса защиты сектора.

**Описание**

Функция позволяет получить текущий статус защиты одного сектора. Первым аргументом передается начальный адрес нужного сектора, вторым аргументом – указатель на переменную, в которую требуется сохранить прочитанный статус. Функция возвращает статус выполнения операции: 0 – операция выполнена успешно, 1 – ошибка. Если сохраненное значение статуса по указателю \*value равно 0, значит защита не установлена. Если 1 – сектор защищен.

**Пример**

**int** **main**(){

uint32\_t status;

**static** flash\_drv\_config\_t driver\_config;

result = MDR1636\_init(&driver\_config, 0);

result = MDR1636\_get\_sector\_protection(0x0, &status);

**return** 0;

}

1. **MDR1636\_get\_chip\_protection**

**Резюме**

uint32\_t **MDR1636\_get\_chip\_protection**(uint32\_t \*value)- функция получения статуса защиты устройства (всех секторов).

**Описание**

Функция считывает статус защиты всех секторов устройства. Аргументом функции является указатель на переменную, в которую необходимо сохранить полученный статус. В результате вызова функции в переменную по указателю \*value будет сохранен один из трех статусов:

**enum** FlashProtectionCodes {

*FP\_Unprotected*,

*FP\_Protected*,

*FP\_ProtectedPart*

};

*FP\_Unprotected* – сектора устройства не защищены,

*FP\_Protected –* все сектора устройства защищены,

*FP\_ProtectedPart –* защищена только часть секторов.

Перед использованием данной функции необходимо инициализировать драйвер функцией uint32\_t **MDR1636\_init**(flash\_drv\_config\_t \*drv\_config, **void** \*hw\_config). Функция возвращает статус выполнения операции: 0 – операция выполнена успешно, 1 – ошибка.

**Пример**

**int** **main**(){

uint32\_t status;

uint32\_t result;

**static** flash\_drv\_config\_t driver\_config;

driver\_config.sector\_count = 2;

driver\_config.sector\_size = 1024;

result = MDR1636\_init(&driver\_config, 0);

result = MDR1636\_get\_chip\_protection(&status);

**return** 0;

}

1. **MDR1636\_start\_erase\_chip**

**Резюме**

uint32\_t **MDR1636\_start\_erase\_chip**(**void**)- функция старта операции стирания устройства Flash.

**Описание**

Данная функция отправляет на устройство командную последовательность, запускающую процесс стирания всех секторов, не дожидаясь его завершения. Функция не имеет аргументов и возвращает статус выполнения: 0 – успешно, 1 – ошибка.

**Пример**

**int** **main**(){

uint32\_t status;

uint32\_t result;

**static** flash\_drv\_config\_t driver\_config;

result = MDR1636\_init(&driver\_config, 0);

result = MDR1636\_start\_erase\_chip();

**return** 0;

}

1. **MDR1636\_start\_erase\_sector**

**Резюме**

uint32\_t **MDR1636\_start\_erase\_sector**(uint32\_t sa)- функция старта стирания одного сектора.

**Описание**

Данная функция запускает операцию стирания одного сектора Flash памяти, не дожидаясь её завершения. Функция имеет один аргумент, через который передается адрес сектора, который необходимо стереть. Функция возвращает статус выполнения операции: 0 – успешно, 1 – ошибка.

**Пример**

**int** **main**(){

uint32\_t status;

uint32\_t result;

**static** flash\_drv\_config\_t driver\_config;

result = MDR1636\_init(&driver\_config, 0);

result = MDR1636\_start\_erase\_sector(0);

**return** 0;

}

1. **MDR1636\_erase\_chip**

**Резюме**

uint32\_t **MDR1636\_erase\_chip**(**void**)- функция старта операции стирания устройства Flash.

**Описание**

Даная функция отправляет на устройство командную последовательность, запускающую процесс стирания всех секторов, при этом дожидаясь его завершения. Функция не имеет аргументов и возвращает статус выполнения: 0 – успешно, 1 – ошибка.

**Пример**

**int** **main**(){

uint32\_t status;

uint32\_t result;

**static** flash\_drv\_config\_t driver\_config;

result = MDR1636\_init(&driver\_config, 0);

result = MDR1636\_erase\_chip();

**return** 0;

}

1. **MDR1636\_erase\_sector**

**Резюме**

uint32\_t **MDR1636\_erase\_sector**(uint32\_t sa)- функция старта стирания одного сектора.

**Описание**

Данная функция запускает операцию стирания одного сектора Flash памяти, при этом дожидаясь её завершения. Функция имеет один аргумент, через который передается адрес сектора, который необходимо стереть. Функция возвращает статус выполнения операции: 0 – успешно, 1 – ошибка.

**Пример**

**int** **main**(){

uint32\_t status;

uint32\_t result;

**static** flash\_drv\_config\_t driver\_config;

result = MDR1636\_init(&driver\_config, 0);

result = MDR1636\_erase\_sector(0);

**return** 0;

}

1. **MDR1636\_write\_data**

**Резюме**

uint32\_t **MDR1636\_write\_data**(uint32\_t address, uint32\_t \*data, uint32\_t count)- функция записи массива данных во Flash память.

**Описание**

Данная функция осуществляет запись массива данных в память. Имеет 3 входных аргумента:

Address – адрес памяти (байтовый), в который необходимо записать данные;

\*data – указатель на массив данных;

Count – количество байт, которое требуется записать в память.

Упаковывать данные заранее не требуется, функция самостоятельно разобьет 32-битные данные на байты и запишет в память. Функция возвращает статус выполнения операции: 0 – успешно, 1 – ошибка. Функция блокируется, пока все данные не будут записаны из буфера в микросхему FLASH.

**Пример**

**int** **main**(){

uint32\_t input\_data[N] = {};

uint32\_t \*ptr\_src;

flash\_drv\_config\_t driver\_config;

ptr\_src = (uint32\_t \*)&input\_data;

result = MDR1636\_init(&driver\_config, 0);

result = MDR1636\_reset();

result = MDR1636\_erase\_chip();

result = MDR1636\_write\_data(addr, ptr\_src, 4\*count\_data);

**return** 0;

}

1. **MDR1636\_read\_data**

**Резюме**

uint32\_t **MDR1636\_read\_data**(uint32\_t address, uint32\_t \*data, uint32\_t count)- функция чтения данных из Flash.

**Описание**

Функция считывает данные из Flash. Блокируется, пока все данные не будут прочитаны из памяти. Содержит 3 входных аргумента:

Address - адрес памяти (байтовый), откуда требуется считать данные;

\*data – указатель на массив, в который требуется поместить прочитанные данные;

Count – количество байт для чтения.

Функция возвращает статус выполнения операции: 0 – успешно, 1 – ошибка.

**Пример**

**int** **main**(){

uint32\_t input\_data[N] = {};

uint32\_t result\_data[N] = {};

uint32\_t \*ptr\_rslt;

uint32\_t \*ptr\_src;

flash\_drv\_config\_t driver\_config;

ptr\_src = (uint32\_t \*)&input\_data;

ptr\_rslt = (uint32\_t \*)&result\_data;

result = MDR1636\_erase\_chip();

result = MDR1636\_write\_data(addr, ptr\_src, 4\*count\_data);

result = MDR1636\_read\_data(addr, ptr\_rslt, 4\*count\_data);

**return** 0;

}