

УТВЕРЖДАЮ
Генеральный директор
ООО «НОВОМАР»
_____ Т.В. Буга
«___» _____ 2020 г.

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ
«ДРАЙВЕР MIL1553UD» И «БИБЛИОТЕКА ВЗАИМОДЕЙСТВИЯ MIL1553UD»

Модулей
“PCIE-1553UDx”
“ХМС-1553UDx”
“СРСIS-1553UDx”
“mPCIE-1553UDx”
(ОС WINDOWS)

Руководство программиста

ЛИСТ УТВЕРЖДЕНИЯ

XXX-ЛУ

От

«___» _____ 2020 г.

2020

Из	Под	Дат

Литера

Инва. № подл.	Подп. и
Взам. инв. №	Подп. и
Инва. № дубл.	Подп. и

Утвержден
ЛУ

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ
«ДРАЙВЕР MIL1553UD» И «БИБЛИОТЕКА ВЗАИМОДЕЙСТВИЯ MIL1553UD»

Модулей
“PCIe-1553UDx”
“ХМС-1553UDx”
“СРСIS-1553UDx”
“mPCIe-1553UDx”
(ОС WINDOWS)

Руководство программиста

XXX
Листов 52

2020

Инва. № подл.	Подп. и
Взам. инв. №	Инва. № дубл.
Подп. и	Подп. и

Из	Под	Дат
----	-----	-----

Литера

АННОТАЦИЯ

В книге описываются технологические принципы, использованные в программном обеспечении «ДРАЙВЕР MIL1553UD» И «БИБЛИОТЕКА ВЗАИМОДЕЙСТВИЯ MIL1553UD» (ОС WINDOWS), для работы модулей PCie-1553UDx», «ХМС-1553UDx», «СРСIS-1553UDx», «mPCie-1553UDx» в сети МКИО ГОСТ Р 52070-2003. В частности, рассмотрены функциональное назначение и область применения, условия выполнения.

<i>Из</i>	<i>Под</i>	<i>Дат</i>

СОДЕРЖАНИЕ

1	НАЗНАЧЕНИЕ ПРОГРАММЫ	6
1.1	ДРАЙВЕР MIL1553UD	6
1.2	ЗАГОЛОВОЧНЫЕ ФАЙЛЫ MIL1553UD	6
2	УСЛОВИЯ ВЫПОЛНЕНИЯ ПРОГРАММЫ	7
2.1	УСТАНОВКА ДРАЙВЕРА MIL1553UD	7
3	ХАРАКТЕРИСТИКА ПРОГРАММЫ	8
3.1	ДРАЙВЕР MIL1553UD	8
3.1.1	Настройка в режим КШ	8
3.1.2	Настройка в режим ОУ	8
3.2	ЗАГОЛОВОЧНЫЕ ФАЙЛЫ MIL1553UD	10
4	ОБРАЩЕНИЕ К ПРОГРАММЕ.....	11
4.1	ДРАЙВЕР MIL1553UD	11
4.1.1	Запись в регистр - IOCTL_WRITE_REG	11
4.1.2	Запись в регистр заданных бит - IOCTL_WRITE_REG_BIT_MASK	12
4.1.3	Чтение из регистра - IOCTL_READ_REG	13
4.1.4	Чтение подробной информации о плате и драйвере - IOCTL_VERSION.....	14
4.1.5	Чтение версии драйвера - IOCTL_VERSION_DRIVER.....	15
4.1.6	Сброс указателя dma канала - IOCTL_CLEAR_DMA.....	16
4.1.7	Включить dma устройства - IOCTL_ENABLE_DMA	17
4.1.8	Выключить dma устройства - IOCTL_DISABLE_DMA	18
4.1.9	Получить количество 128 байтных блоков из ДМА - IOCTL_GET_NBLOCK_RAW_DMA	19
4.1.10	Считать заданное количество 128 байтных из ДМА - IOCTL_READ_BLOCKS_RAW_DMA.....	20
4.1.11	Управление прерываниями INTERRUPT_MASK - IOCTL_MANAGE_INTERRUPT	21
4.1.12	Управление прерываниями по передаче из подадреса - IOCTL_MAN_IRQ_SUBADDRESS_TR.....	22
4.1.13	Управление прерываниями по приёму в подадрес - IOCTL_MAN_IRQ_SUBADDRESS_RCV	23
4.1.14	Выбор режима работы канала - IOCTL_SWITCH_MODE	24

<i>Из</i>	<i>Под</i>	<i>Дат</i>

4.1.15	Выбор шины канала - IOCTL_SWITCH_BUS	25
4.1.16	Управление работой канала - IOCTL_DEV_CHANNEL_WORK	26
4.1.17	Установить режим ОУ - IOCTL_SET_RT_TR_MODE	27
4.1.18	Установить готовность буфера ОУ - IOCTL_SET_RT_TR_BUF_READY	28
4.1.19	Управление разрешением буферов - IOCTL_RT_BUF_MAN_EN	29
4.1.20	Установить адрес ОУ - IOCTL_SET_ADDRESS	30
4.1.21	Установить таймауты ОУ - IOCTL_SET_TIMER_RTBM_CONF_REG_PCI	31
4.1.22	Установить таймауты КШ - IOCTL_SET_TIMER_BC_CONF_REG_PCI	32
4.1.23	Записать подадрес на отправку - IOCTL_WR_BLOCK_BUF_SUBADDR	33
4.1.24	Прочитать подадрес на отправку - IOCTL_RD_BLOCK_BUF_SUBADR	34
4.1.25	Запись блока в BC RAM - IOCTL_WRITE_BC_RAM	35
4.1.26	Чтение блока из BC RAM - IOCTL_WRITE_BC_RAM	36
4.1.27	Получить кол-во блоков прерываний - IOCTL_GET_COUNT_INTERRUPT_BLOCKS	37
4.1.28	Считать блоки прерываний - IOCTL_READ_INTERRUPT_BLOCKS 38	
4.1.29	Считать входной подадрес - IOCTL_GET_RCV_SUBADR_DATA ..	39
4.2	ВЗАИМОДЕЙСТВИЕ С ДРАЙВЕРОМ MIL1553UD	40
4.3	Пример вызова IOCTL	41
5	Библиотека mil1553ud	42
5.1	Назначение программы	42
5.2	Подключение библиотеки к разрабатываемому проекту	42
5.3	Список доступных функций	43
5.4	Описание функций	44
5.4.1	mil1553UdEnumerateDevices	44
5.4.2	mil1553UdFreeDeviceList	44
5.4.3	mil1553UdOpen	44
5.4.4	mil1553UdReadReg	45
5.4.5	mil1553UdWriteReg	45

<i>Из</i>	<i>Под</i>	<i>Дат</i>

5.4.6	mil1553UdGetCountDmaBlocksReadyRead	46
5.4.7	mil1553UdGetCountInterruptBlocksReadyRead	46
5.4.8	mil1533udReadDmaBlocks	47
5.4.9	mil1533udReadInterruptBlocks	48
5.4.10	mil1533udReadInterruptBlocks	48
5.4.11	mil1553UdDmaBlockIsSuccessTransaction.....	49
5.4.12	mil1553UdDmaBlockGetTypeTransaction	49
5.4.13	mil1553UdDmaBlockGetFreeTimer	49
5.4.14	mil1553UdDmaBlockGetCountServiceWords16.....	49
5.4.15	mil1553UdDmaBlockGetCountDataWords16	49
5.4.16	mil1553UdDmaBlockTdModeSubaddress	50
5.4.17	mil1553UdDmaBlockGetActiveBus	50
5.4.18	mil1553UdDmaBlockGetPtrDataPart.....	50
5.4.19	mil1553UdNewBcRamBlock	51
5.4.20	mil1553UdFreeBcRamBlock	51
5.4.21	mil1553UdBcProgram	52
СПИСОК СОКРАЩЕНИЙ.....		53

<i>Из</i>	<i>Под</i>	<i>Дат</i>

1 НАЗНАЧЕНИЕ ПРОГРАММЫ

1.1 ДРАЙВЕР MIL1553UD

Программное обеспечение «ДРАЙВЕР MIL1553UD» (далее – драйвер) обеспечивает возможность управления PCI-устройствами “PCIe-1553UDx”, “ХМС-1553UDx”, “СРСIS-1553UDx”, “mPCIe-1553UDx” (далее MIL1553UD).

MIL1553UD (1-4-х канальный контроллер интерфейса МКИО – ГОСТ Р 52070-2003).

Драйвер обеспечивает выполнение следующих основных задач:

- определение и инициализация устройства на шине PCI;
- инициализация символьных устройств каналов (1-4) для обеспечения взаимодействия из юзерспейс пространства;
- реализация команд управления каналами в режимах КШ, ОУ, МШ, МША.

1.2 ЗАГОЛОВОЧНЫЕ ФАЙЛЫ MIL1553UD

Описывают используемые драйвером коды запросов ЮСТЛ и структуры данных.

<i>Из</i>	<i>Под</i>	<i>Дат</i>

2 УСЛОВИЯ ВЫПОЛНЕНИЯ ПРОГРАММЫ

2.1 УСТАНОВКА ДРАЙВЕРА MIL1553UD

Драйвер является модулем ядра и предназначен для функционирования в ОС семейства Windows - 7, 8, 8.1, 10, поддерживаются архитектуры x86 и amd64.

Установка и удаление драйвера производятся через диспетчер устройств ОС Windows.

<i>Из</i>	<i>Под</i>	<i>Дат</i>

3 ХАРАКТЕРИСТИКА ПРОГРАММЫ

3.1 ДРАЙВЕР MIL1553UD

Драйвер является модулем ядра ОС Windows, разработан на языке С.

Находящиеся под управлением драйвера устройства доступны в системе по абсолютному пути \Device\mil1553ud_n, где n - порядковый номер устройства начиная с 0.

Драйвер реализует интерфейс класса GUID_DEVINTERFACE_MIL1553 (3bd2b180-d211-4d88-8f46-b73845cf38fb), что позволяет обнаружить все устройства в системе с помощью функции SetupDiGetClassDevs (см. документацию setupapi).

Взаимодействие с драйвером происходит посредством запросов ioctl.

3.1.1 Настройка в режим КШ

Настройка канала в режим КШ осуществляется с помощью следующей последовательности IOCTL-команд:

```
IOCTL_SWITCH_MODE // установить режим КШ
IOCTL_SWITCH_BUS // установить шину (А или Б)
IOCTL_BC_RAM // составляем и записываем микропрограмму
IOCTL_ENABLE_DMA // включаем ДМА
IOCTL_DEV_CHANNEL_WORK // разрешаем работу канала
IOCTL_WRITE_REG_BIT_MASK // устанавливаем бит BCSTRT
...
// разбор принятых данных
IOCTL_GET_NBLOCK_RAW_DMA
IOCTL_READ_BLOCKS_RAW_DMA
```

3.1.2 Настройка в режим ОУ

Настройка канала в режим ОУ осуществляется с помощью следующей последовательности IOCTL-команд:

```
IOCTL_SWITCH_MODE // установить режим ОУ
IOCTL_SET_ADDRESS // установить адрес ОУ (1-30)
IOCTL_SWITCH_BUS // установить шину (А или Б)
IOCTL_RT_BUF_MAN_EN // задать маску разрешённых поадресов на приём
```

Из	Под	Дат

IOCTL_RT_BUF_MAN_EN // задать маску разрешённых подадресов на передачу

IOCTL_SET_RT_TR_MODE // выбрать режим работы буферов

IOCTL_SET_RT_BUF_READY // задать готовность буферов подаресов к отправке

IOCTL_ENABLE_DMA // включаем ДМА

IOCTL_DEV_CHANNEL_WORK // разрешаем работу канала

...

// разбор принятых данных

IOCTL_GET_NBLOCK_RAW_DMA

IOCTL_READ_BLOCKS_RAW_DMA

<i>Из</i>	<i>Под</i>	<i>Дат</i>

3.2 ЗАГОЛОВОЧНЫЕ ФАЙЛЫ MIL1553UD

Заголовочные файлы описывают используемые драйвером коды запросов IOCTL и структуры данных и предназначены для разработки на языках C и C++:

- regs.h
- info.h;
- ioctl_def.h;

Для использования заголовочных файлов в проекте необходимо включить вышеназванные файлы в состав разрабатываемого проекта.

<i>Из</i>	<i>Под</i>	<i>Дат</i>

4 ОБРАЩЕНИЕ К ПРОГРАММЕ

4.1 ДРАЙВЕР MIL1553UD

Взаимодействие с каналами происходит посредством ioctl-команд, описание команд и типы данных представлены в файле «ioctl_def.h».

4.1.1 Запись в регистр - IOCTL_WRITE_REG

Позволяет записать значение в заданный регистр управления.

Описание параметров команды приведено на рисунке 1.

```
#define IOCTL_WRITE_REG CTL_CODE(FILE_DEVICE_CONTROLLER, 0,  
METHOD_BUFFERED, FILE_ANY_ACCESS)  
  
//входные данные  
typedef struct {  
    // адрес регистра (смещение в соотв. со спецификацией)  
    UINT32 daddr;  
    // значение регистра  
    UINT32 data;  
    //номер канала устройства  
    UINT8 channel;  
} SADDR_DATA;
```

Рисунок 1 – Листинг команды

Из	Под	Дат

4.1.2 Запись в регистр заданных бит - IOCTL_WRITE_REG_BIT_MASK

Позволяет изменить отдельные биты в заданном регистре управления.

Описание параметров команды приведено на рисунке 2.

```
#define IOCTL_WRITE_REG_BIT_MASK CTL_CODE(FILE_DEVICE_CONTROLLER,  
1, METHOD_BUFFERED, FILE_ANY_ACCESS)  
  
//входные данные  
  
typedef struct {  
    // адрес регистра (смещение в соотв. со спецификацией)  
    UINT32 daddr;  
    // значение регистра  
    UINT32 data;  
    // brief битовая маска  
    // 1 - бит записывается из data, 0 - бит остаётся неизменным  
    UINT32 mask;  
    //номер канала устройства  
    UINT8 channel;  
} SADDR_DATA_BIT_MASK;
```

Рисунок 2 – Листинг команды

<i>Из</i>	<i>Под</i>	<i>Дат</i>

4.1.3 Чтение из регистра - IOCTL_READ_REG

Позволяет прочитать значение из заданного регистра управления.

Описание параметров команды приведено на рисунке 3.

```
#define IOCTL_READ_REG CTL_CODE(FILE_DEVICE_CONTROLLER, 2,  
METHOD_BUFFERED, FILE_ANY_ACCESS)  
  
//входные данные  
typedef struct {  
    // адрес регистра (смещение в соотв. со спецификацией)  
    UINT32 daddr;  
    // неприменимо  
    UINT32 data;  
    //номер канала устройства  
    UINT8 channel;  
} SADDR_DATA;  
  
//выходные данные  
typedef struct {  
    // адрес регистра (смещение в соотв. со спецификацией)  
    UINT32 daddr;  
    // значение регистра  
    UINT32 data;  
    //номер канала устройства  
    UINT8 channel;  
} SADDR_DATA;
```

Рисунок 3 – Листинг команды

<i>Из</i>	<i>Под</i>	<i>Дат</i>

4.1.4 Чтение подробной информации о плате и драйвере - IOCTL_VERSION

Позволяет считать подробную информацию о pci-плате.

Описание параметров команды приведено на рисунке 4.

```
#define IOCTL_VERSION
    CTL_CODE(FILE_DEVICE_CONTROLLER, 31, METHOD_BUFFERED,
FILE_ANY_ACCESS)

//выходные данные
typedef struct {
    // идентификатор устройства
    UINT32 device_id;
    // вендор устройства
    UINT32 vendor_id;
    // тип устройства (кол-во каналов)
    UINT32 type;
    // ревизия устройства
    char revision;
    // неприменимо
    char dev_name[MAXIMUM_FILENAME_LENGTH];
    // неприменимо
    int minor;
    // номер прерывания
    int irq;
    // размер ДМА буфера
    UINT32 size_dma;
    // неприменимо
    UINT32 pciBars;
} VERSION;
```

Рисунок 4 – Листинг команды

<i>Из</i>	<i>Под</i>	<i>Дат</i>

4.1.5 Чтение версии драйвера - IOCTL_VERSION_DRIVER

Позволяет считать версию и дату драйвера.

Описание параметров команды приведено на рисунке 5.

```
#define IOCTL_VERSION_DRIVER
    CTL_CODE(FILE_DEVICE_CONTROLLER, 40, METHOD_BUFFERED,
FILE_ANY_ACCESS)

//выходные данные
/// \remark информация о дате и версии
/// \brief старший номер версии
#define DRIVER_MAJOR_VER                2
/// \brief младший номер версии
#define DRIVER_MINOR_VER                0
/// \brief день создания
#define DRIVER_DATE_DAY                 16
/// \brief месяц создания
#define DRIVER_DATE_MONTH               06
/// \brief год создания
#define DRIVER_DATE_YEAR                19
/// \brief закодированная дата и версия
/// 31..28 - major_ver; 27..24 - minor_ver; 23..16 - day; 15..8 -
month; 7..0 - year;
#define DRIVER_DATE_N_VERSION           ((DRIVER_MAJOR_VER & 0xF) <<
28) | ((DRIVER_MINOR_VER & 0xF) << 24) | ((DRIVER_DATE_DAY & 0xFF)
<< 16) | ((DRIVER_DATE_MONTH & 0xFF) << 8) | (DRIVER_DATE_YEAR &
0xFF)
```

Рисунок 5 – Листинг команды

<i>Из</i>	<i>Под</i>	<i>Дат</i>

4.1.6 Сброс указателя dma канала - IOCTL_CLEAR_DMA

Позволяет обнулить DMA_INDEX и программные указатели чтения/записи.

Описание параметров команды приведено на рисунке 6.

```
#define IOCTL_CLEAR_DMA CTL_CODE(FILE_DEVICE_CONTROLLER, 5,  
METHOD_BUFFERED, FILE_ANY_ACCESS) //сброс буфера  
  
//входные данные  
typedef struct _MODE_DATA{  
    UINT8 channel; //номер канала  
    UINT8 mode;    //неприменимо  
} MODE_DATA, *PMODE_DATA;
```

Рисунок 6 – Листинг команды

<i>Из</i>	<i>Под</i>	<i>Дат</i>

4.1.7 Включить dma устройства - IOCTL_ENABLE_DMA

Разрешает работу DMA.

Описание параметров команды приведено на рисунке 7.

```
#define IOCTL_ENABLE_DMA CTL_CODE(FILE_DEVICE_CONTROLLER, 8,  
METHOD_BUFFERED, FILE_ANY_ACCESS) //вкл DMA  
  
//входные данные  
typedef struct _MODE_DATA{  
    UINT8 channel; //номер канала  
    UINT8 mode;    //неприменимо  
} MODE_DATA, *PMODE_DATA;
```

Рисунок 7 – Листинг команды

<i>Из</i>	<i>Под</i>	<i>Дат</i>

4.1.8 Выключить dma устройства - IOCTL_DISABLE_DMA

Запрещает работу DMA.

Описание параметров команды приведено на рисунке 8.

```
#define IOCTL_DISABLE_DMA CTL_CODE(FILE_DEVICE_CONTROLLER, 9,  
METHOD_BUFFERED, FILE_ANY_ACCESS) //выкл DMA  
  
//входные данные  
typedef struct _MODE_DATA{  
    UINT8 channel; //номер канала  
    UINT8 mode; //неприменимо  
} MODE_DATA, *PMODE_DATA;
```

Рисунок 8 – Листинг команды

<i>Из</i>	<i>Под</i>	<i>Дат</i>

4.1.9 Получить количество 128 байтных блоков из ДМА - IOCTL_GET_NBLOCK_RAW_DMA

Позволяет получить количество готовых для чтения блоков из буфера ДМА.

Описание параметров команды приведено на рисунке 9.

```
#define IOCTL_GET_NBLOCKS_RAW_DMA    CTL_CODE(FILE_DEVICE_CONTROLLER,  
7, METHOD_BUFFERED, FILE_ANY_ACCESS)  
  
//входные данные  
typedef struct _DMA_COUNT_REQUEST{  
    UINT8 channel; //канал  
    UINT32 count;  //неприменимо  
} DMA_COUNT_REQUEST, *PDMA_COUNT_REQUEST;  
  
//выходные данные  
typedef struct _DMA_COUNT_REQUEST{  
    UINT8 channel; //канал  
    UINT32 count;  //количество блоков  
} DMA_COUNT_REQUEST, *PDMA_COUNT_REQUEST;
```

Рисунок 9 – Листинг команды

<i>Из</i>	<i>Под</i>	<i>Дат</i>

4.1.10 Читать заданное количество 128 байтных из ДМА -
IOCTL_READ_BLOCKS_RAW_DMA

Позволяет получить данные (блоки) из буфера ДМА.

Описание параметров команды приведено на рисунке 10.

```
#define IOCTL_READ_BLOCKS_RAW_DMA
    CTL_CODE(FILE_DEVICE_CONTROLLER, 6, METHOD_BUFFERED,
FILE_ANY_ACCESS)

#define DMA_RAW_BLOCK_SIZE          128
#define DMA_SUPERBLOCK_SIZE        8192 * DMA_RAW_BLOCK_SIZE

//входные данные
/// \brief считываемый блок дма, кратный 128 байтам
typedef struct {
    /// \brief количество 128 байтных блоков
    UINT32 countBlocks;
    /// неприменимо
    UINT8 data[DMA_SUPERBLOCK_SIZE];
    //канал
    UINT8 channel;
} DMA_READ_BLOCK;

//выходные данные
/// \brief считываемый блок дма, кратный 128 байтам
typedef struct {
    /// \brief количество 128 байтных блоков
    UINT32 countBlocks;
    /// \brief данные блоков в сыром виде
    UINT8 data[DMA_SUPERBLOCK_SIZE];
    //канал
    UINT8 channel;
} DMA_READ_BLOCK;
```

Рисунок 10 – Листинг команды

<i>Из</i>	<i>Под</i>	<i>Дат</i>

4.1.11 Управление прерываниями INTERRUPT_MASK -
IOCTL_MANAGE_INTERRUPT

Позволяет управлять прерываниями.

Описание параметров команды приведено на рисунке 11.

```
#define IOCTL_MANAGE_INTERRUPT      CTL_CODE(FILE_DEVICE_CONTROLLER,  
16, METHOD_BUFFERED, FILE_ANY_ACCESS)  
  
/// \brief управление маскированием прерываний поканально  
///входные данные  
typedef struct {  
    /// \brief состояние прерывания от контроллера flash  
    UINT8 int_flash;  
    /// \brief состояние прерывания от КШ  
    UINT8 int_bc;  
    /// \brief состояние прерывания от ОУ при приёме данных  
    UINT8 int_rt_ren;  
    /// \brief состояние прерывания от ОУ при отправке данных  
    UINT8 int_rt_ten;  
    /// \brief состояние прерывания при заполнении 1/18  
    UINT8 int_qdat;  
    /// \brief состояние прерывания при заполнении 1/2  
    UINT8 int_hdat;  
    /// \brief состояние прерывания счётчика данных контроллера MII  
    UINT8 int_data_cnt_en;  
    /// \brief состояние прерывания интервального таймера MII  
    UINT8 int_timeout_itven;  
    /// \brief состояние прерывания абсолютного таймера MII  
    UINT8 int_timeout_absen;  
    ///канал  
    UINT8 channel;  
} INTERRUPT_MAN;  
  
/// \remark константы для управления прерываниями  
/// 0 - не меняется значение, 1 - включить прерывание, 2 - выключить  
прерывание  
/// \brief не изменять бит прерывания  
#define INTERRUPT_MAN_NO_CHANGE      0  
/// \brief включить бит прерывания  
#define INTERRUPT_MAN_ON              1  
/// \brief выключить бит прерывания  
#define INTERRUPT_MAN_OFF            2
```

Рисунок 11 – Листинг команды

Из	Под	Дат

4.1.12 Управление прерываниями по передаче из подадреса -
IOCTL_MAN_IRQ_SUBADDRESS_TR

Позволяет управлять прерываниями по передаче из подадреса.

Описание параметров команды приведено на рисунке 12.

```
#define IOCTL_MAN_IRQ_SUBADDRESS_TR
    CTL_CODE(FILE_DEVICE_CONTROLLER, 38, METHOD_BUFFERED,
FILE_ANY_ACCESS)

//входные данные
typedef struct {
    /// \brief состояние прерывания (вкл - 1 / выкл - 2)
    UINT8 rt_int;
    /// \brief маскирование прерываний по подадресам 1й бит - 1й
подадрес и т.д.
    UINT32 subaddress_mask;
    //канал
    UINT8 channel;
} INTERRUPT_SUBADDRESS;

/// \brief включить бит прерывания
#define INTERRUPT_MAN_ON 1
/// \brief выключить бит прерывания
#define INTERRUPT_MAN_OFF 2
```

Рисунок 12 – Листинг команды

Из	Под	Дат

4.1.13 Управление прерываниями по приёму в поадрес -
IOCTL_MAN_IRQ_SUBADDRESS_RCV

Позволяет управлять прерываниями по приёму в поадрес.

Описание параметров команды приведено на рисунке 13.

```
#define IOCTL_MAN_IRQ_SUBADDRESS_RCV
    CTL_CODE(FILE_DEVICE_CONTROLLER, 39, METHOD_BUFFERED,
FILE_ANY_ACCESS)

//входные данные
typedef struct {
    /// \brief состояние прерывания (вкл - 1 / выкл - 2)
    UINT8 rt_int;
    /// \brief маскирование прерываний по поадресам 1й бит - 1й
поадрес и т.д.
    UINT32 subaddress_mask;
    ///канал
    UINT8 channel;
} INTERRUPT_SUBADDRESS;

/// \brief включить бит прерывания
#define INTERRUPT_MAN_ON 1
/// \brief выключить бит прерывания
#define INTERRUPT_MAN_OFF 2
```

Рисунок 13 – Листинг команды

<i>Из</i>	<i>Под</i>	<i>Дат</i>

4.1.14 Выбор режима работы канала - IOCTL_SWITCH_MODE

Позволяет выбрать режим работы канала.

Описание параметров команды приведено на рисунке 14.

```
#define IOCTL_SWITCH_MODE
    CTL_CODE(FILE_DEVICE_CONTROLLER, 4, METHOD_BUFFERED,
FILE_ANY_ACCESS)

//входные данные
typedef struct _MODE_DATA{
    UINT8 channel; //номер канала
    UINT8 mode;    //режим работы канала
} MODE_DATA, *PMODE_DATA;

/// \remark константы режимов работы устройства
/// \brief адресуемый монитор шины
#define MIL_MODE_MONITOR_ADRR        0x0
/// \brief контроллер шины
#define MIL_MODE_BUS_CONTR           0x1
/// \brief оконечное устройство
#define MIL_MODE_TERMINAL_DEV        0x2
/// \brief неадресуемый монитор шины
#define MIL_MODE_MONITOR              0x3
```

Рисунок 14 – Листинг команды

<i>Из</i>	<i>Под</i>	<i>Дат</i>

4.1.15 Выбор шины канала - IOCTL_SWITCH_BUS

Позволяет выбрать шину канала.

Описание параметров команды приведено на рисунке 15.

```
#define IOCTL_SWITCH_BUS
    CTL_CODE(FILE_DEVICE_CONTROLLER, 33, METHOD_BUFFERED,
FILE_ANY_ACCESS)

//входные данные
typedef struct _MODE_DATA{
    UINT8 channel; //номер канала
    UINT8 mode;    //шина канала
} MODE_DATA, *PMODE_DATA;

/// \remark константы выбора шины канала
/// \brief включить шину А
#define MIL_BUS_A_EN                0x1
/// \brief включить шину В
#define MIL_BUS_B_EN                0x2
```

Рисунок 15 – Листинг команды

<i>Из</i>	<i>Под</i>	<i>Дат</i>

4.1.16 Управление работой канала - IOCTL_DEV_CHANNEL_WORK

Позволяет управлять работой канала.

Описание параметров команды приведено на рисунке 16.

```
#define IOCTL_DEV_CHANNEL_WORK
    CTL_CODE(FILE_DEVICE_CONTROLLER, 34, METHOD_BUFFERED,
FILE_ANY_ACCESS)

//входные данные
typedef struct _MODE_DATA{
    UINT8 channel; //номер канала
    UINT8 mode;    //константа включения/выключения канала
} MODE_DATA, *PMODE_DATA;

/// \remark константы вкл/выкл канала
/// \brief включить канал в работу
#define MIL_DEV_CHANNEL_ON          0x1
/// \brief выключить канал
#define MIL_DEV_CHANNEL_OFF        0x0
```

Рисунок 16 – Листинг команды

<i>Из</i>	<i>Под</i>	<i>Дат</i>

4.1.17 Установить режим ОУ - IOCTL_SET_RT_TR_MODE

Позволяет установить режим ОУ.

Описание параметров команды приведено на рисунке 17.

```
#define IOCTL_SET_RT_TR_MODE CTL_CODE(FILE_DEVICE_CONTROLLER, 35,  
METHOD_BUFFERED, FILE_ANY_ACCESS)  
  
//входные данные  
typedef struct {  
    // подадрес  
    UINT8 subaddress;  
    // режим  
    UINT32 mode;  
    //канал  
    UINT8 channel;  
} RT_TR_MODE;  
  
/// \remark константы для режима ОУ  
/// \brief режим - программный  
#define MIL_RT_MODE_PROG 0x0  
/// \brief режим - аппаратный
```

Рисунок 17 – Листинг команды

<i>Из</i>	<i>Под</i>	<i>Дат</i>

4.1.18 Установить готовность буфера ОУ -
IOCTL_SET_RT_TR_BUF_READY

Позволяет установить готовность буфера ОУ.

Описание параметров команды приведено на рисунке 18.

```
#define IOCTL_SET_RT_TR_BUF_READY CTL_CODE(FILE_DEVICE_CONTROLLER,  
37, METHOD_BUFFERED, FILE_ANY_ACCESS)  
  
//входные данные  
typedef struct {  
    // подадрес 1 - 30  
    UINT8 subaddress;  
    // команда упр. буфером  
    UINT32 cmd_buf;  
    //канал  
    UINT8 channel;  
} RT_TR_MODE_BUF;  
  
/// \remark команды управления буферами cmd_buf  
/// \brief буфер данных RTF_BUF0 и RTF_BUF1 - выкл  
#define MIL_RT_BUF_0_OFF_1_OFF 0x0  
/// \brief буфер данных RTF_BUF0 - выкл и RTF_BUF1 - вкл  
#define MIL_RT_BUF_0_OFF_1_ON 0x1  
/// \brief буфер данных RTF_BUF0 - вкл и RTF_BUF1 - выкл  
#define MIL_RT_BUF_0_ON_1_OFF 0x2  
/// \brief буфер данных RTF_BUF0 - вкл и RTF_BUF1 - вкл  
#define MIL_RT_BUF_0_ON_1_ON 0x3
```

Рисунок 18 – Листинг команды

<i>Из</i>	<i>Под</i>	<i>Дат</i>

4.1.19 Управление разрешением буферов - IOCTL_RT_BUF_MAN_EN

Позволяет осуществлять управление разрешением буферов.

Описание параметров команды приведено на рисунке 19.

```
#define IOCTL_RT_BUF_MAN_EN CTL_CODE(FILE_DEVICE_CONTROLLER, 36,  
METHOD_BUFFERED, FILE_ANY_ACCESS)  
  
//входные данные  
typedef struct {  
    // направление (приём/передача)  
    UINT32 direction;  
    // маскирование (выбор) по подадресам 1й бит - 1й подадрес и  
    т.д.  
    /// 1 - бит изменяем, 0 - бит не изменяем  
    UINT32 subaddress_mask;  
    // маскирование (выбор) действий по подадресам 1й бит - 1й  
    подадрес и т.д.  
    // 1 - вкл, 0 - выкл  
    UINT32 action_mask;  
    //канал  
    UINT8 channel;  
} RT_BUF_MAN_EN;  
  
/// \remark направление передачи direction  
/// \brief направление - передача  
#define MIL_RT_BUF_TRANSMIT 0x0  
/// \brief направление - передача  
#define MIL_RT_BUF_RECEIVE 0x1
```

Рисунок 19 – Листинг команды

<i>Из</i>	<i>Под</i>	<i>Дат</i>

4.1.20 Установить адрес ОУ - IOCTL_SET_ADDRESS

Позволяет установить адрес ОУ.

Описание параметров команды приведено на рисунке 20.

```
#define IOCTL_SET_ADDRESS      CTL_CODE(FILE_DEVICE_CONTROLLER, 19,  
METHOD_BUFFERED, FILE_ANY_ACCESS)  
  
//входные данные  
typedef struct _ADDRESS_REQ{  
    UINT8 channel;  
    UINT8 address;  
} ADDRESS_REQ, *PADDRESS_REQ;
```

Рисунок 20 – Листинг команды

<i>Из</i>	<i>Под</i>	<i>Дат</i>

4.1.21 Установить таймауты ОУ -
IOCTL_SET_TIMER_RTBM_CONF_REG_PCI

Позволяет установить таймауты ОУ.

Описание параметров команды приведено на рисунке 21.

```
#define IOCTL_SET_TIMER_RTBM_CONF_REG_PCI
    CTL_CODE(FILE_DEVICE_CONTROLLER, 20, METHOD_BUFFERED,
FILE_ANY_ACCESS)

//входные данные
typedef struct {
    UINT32 tck; // значение RT_TCK
    UINT32 trck; // значение RT_TRCK
    UINT32 rcvck; // значение RT_RCVCK
    UINT8 channel; // канал
} TIMER_TIMEOUT;

/// \remark константы для timer timeout RCVCK
#define MIL_T_RCVCK_17MKS 0x0
#define MIL_T_RCVCK_60MKS 0x1
#define MIL_T_RCVCK_85MKS 0x2
#define MIL_T_RCVCK_110MKS 0x3
/// \remark константы для timer timeout TRCK
#define MIL_T_TRCK_6MKS 0x0
#define MIL_T_TRCK_8MKS 0x1
#define MIL_T_TRCK_11MKS 0x2
#define MIL_T_TRCK_13MKS 0x3
#define MIL_T_TRCK_18MKS 0x4
#define MIL_T_TRCK_61MKS 0x5
#define MIL_T_TRCK_86MKS 0x6
#define MIL_T_TRCK_111MKS 0x7
/// \remark константы для timer timeout TCK
#define MIL_T_TCK_OFF 0x0
#define MIL_T_TCK_1MKS 0x1
#define MIL_T_TCK_2MKS 0x2
#define MIL_T_TCK_4MKS 0x3
#define MIL_T_TCK_8MKS 0x4
#define MIL_T_TCK_16MKS 0x5
#define MIL_T_TCK_32MKS 0x6
#define MIL_T_TCK_64MKS 0x7
```

Рисунок 21 – Листинг команды

<i>Из</i>	<i>Под</i>	<i>Дат</i>

4.1.22 Установить таймауты КИШ - IOCTL_SET_TIMER_BC_CONF_REG_PCI

Позволяет установить таймауты КИШ.

Описание параметров команды приведено на рисунке 22.

```
#define IOCTL_SET_TIMER_BC_CONF_REG_PCI
    CTL_CODE(FILE_DEVICE_CONTROLLER, 32, METHOD_BUFFERED,
FILE_ANY_ACCESS)

//входные данные
typedef struct {
    UINT32 tck; // значение RT_TCK
    UINT32 trck; // значение RT_TRCK
    UINT32 rcvck; // значение RT_RCVCK
    UINT8 channel; // канал
} TIMER_TIMEOUT;

/// \remark константы для timer timeout RCVCK
#define MIL_T_RCVCK_17MKS          0x0
#define MIL_T_RCVCK_60MKS          0x1
#define MIL_T_RCVCK_85MKS          0x2
#define MIL_T_RCVCK_110MKS         0x3
/// \remark константы для timer timeout TRCK
#define MIL_T_TRCK_6MKS             0x0
#define MIL_T_TRCK_8MKS             0x1
#define MIL_T_TRCK_11MKS            0x2
#define MIL_T_TRCK_13MKS            0x3
#define MIL_T_TRCK_18MKS            0x4
#define MIL_T_TRCK_61MKS            0x5
#define MIL_T_TRCK_86MKS            0x6
#define MIL_T_TRCK_111MKS           0x7
/// \remark константы для timer timeout TCK
#define MIL_T_TCK_OFF               0x0
#define MIL_T_TCK_1MKS              0x1
#define MIL_T_TCK_2MKS              0x2
#define MIL_T_TCK_4MKS              0x3
#define MIL_T_TCK_8MKS              0x4
#define MIL_T_TCK_16MKS             0x5
#define MIL_T_TCK_32MKS             0x6
#define MIL_T_TCK_64MKS             0x7
```

Рисунок 22 – Листинг команды

<i>Из</i>	<i>Под</i>	<i>Дат</i>

4.1.23 Записать подадрес на отправку -
IOCTL_WR_BLOCK_BUF_SUBADDR

Позволяет записать данные в подадрес на отправку.

Описание параметров команды приведено на рисунке 23.

```
#define IOCTL_WR_BLOCK_BUF_SUBADDR CTL_CODE(FILE_DEVICE_CONTROLLER,  
27, METHOD_BUFFERED, FILE_ANY_ACCESS)  
  
//входные данные  
typedef struct {  
    UINT8 num_buf;           // номер буфера  
    UINT8 subaddress;       // подадрес (1-30)  
    UINT16 data_words[32];  // подадрес (32 слова данных)  
    UINT8 channel;          // канал  
} RT_TR_BUF_SUBADDR;  
  
/// \remark номер буфера num_buf  
/// \brief отправной буфер данных RTF_BUF0  
#define MIL_RT_BUF0          0  
/// \brief отправной буфер данных RTF_BUF1  
#define MIL_RT_BUF1          1
```

Рисунок 23 – Листинг команды

<i>Из</i>	<i>Под</i>	<i>Дат</i>

4.1.24 Прочитать подадрес на отправку -
IOCTL_RD_BLOCK_BUF_SUBADR

Позволяет прочитать данные из подадреса на отправку.

Описание параметров команды приведено на рисунке 24.

```
#define IOCTL_RD_BLOCK_BUF_SUBADDR CTL_CODE(FILE_DEVICE_CONTROLLER,  
28, METHOD_BUFFERED, FILE_ANY_ACCESS)  
  
//входные данные  
typedef struct {  
    UINT8 num_buf;           // номер буфера  
    UINT8 subaddress;       // подадрес (1-30)  
    UINT16 data_words[32];  // неприменимо  
    UINT8 channel;         // канал  
} RT_TR_BUF_SUBADDR;  
  
//выходные данные  
typedef struct {  
    UINT8 num_buf;           // номер буфера  
    UINT8 subaddress;       // подадрес (1-30)  
    UINT16 data_words[32];  // подадрес (32 слова данных)  
    UINT8 channel;         // канал  
} RT_TR_BUF_SUBADDR;  
  
/// \remark номер буфера num_buf  
/// \brief отправной буфер данных RTF_BUF0  
#define MIL_RT_BUF0          0  
/// \brief отправной буфер данных RTF_BUF1  
#define MIL_RT_BUF1          1
```

Рисунок 24 – Листинг команды

<i>Из</i>	<i>Под</i>	<i>Дат</i>

4.1.25 Запись блока в BC RAM - IOCTL_WRITE_BC_RAM

Позволяет записать блок данных для записи в BC_RAM (область микропрограммы для КШ).

Описание параметров команды приведено на рисунке 25.

```
#define IOCTL_WRITE_BC_RAM CTL_CODE(FILE_DEVICE_CONTROLLER, 29,  
METHOD_BUFFERED, FILE_ANY_ACCESS)  
  
//входные данные  
typedef struct {  
    /// \brief тип инструкций  
    /// INSTR, OPERATION, DATA  
    UINT32 type;  
    /// \brief смещение от начала буфера в 32-х разрядных словах  
    /// INSTR - max 4095, OPERATION - max 4095, DATA - max 8191  
    UINT32 shift;  
    // размер поля dwords  
    UINT32 length;  
    // данные  
    UINT32 dwords[BC_RAM_MAX];  
    // канал  
    UINT8 channel;  
} BC_RAM_BLOCK; /// \remark константы для команды записи/чтения BC  
RAM  
  
#define MIL_BC_RAM_TYPE_INSTRUCTION 0x0  
#define MIL_BC_RAM_TYPE_OPERATION 0x1  
#define MIL_BC_RAM_TYPE_DATA 0x2
```

Рисунок 25 – Листинг команды

Из	Под	Дат

4.1.26 Чтение блока из BC RAM - IOCTL_WRITE_BC_RAM

Позволяет прочитать блок данных из BC_RAM (область микропрограммы для КШ).

Описание параметров команды приведено на рисунке 26.

```
#define IOCTL_READ_BC_RAM      CTL_CODE(FILE_DEVICE_CONTROLLER, 30,
METHOD_BUFFERED, FILE_ANY_ACCESS)

//входные данные
typedef struct {
    // неприменимо
    UINT32 type;
    /// \brief смещение от начала буфера в 32-х разрядных словах
    /// INSTR - max 4095, OPERATION - max 4095, DATA - max 8191
    UINT32 shift;
    // размер поля dwords
    UINT32 length;
    // неприменимо
    UINT32 dwords[BC_RAM_MAX];
    // канал
    UINT8 channel;
} BC_RAM_BLOCK;

//выходные данные
typedef struct {
    /// \brief тип инструкций
    /// INSTR, OPERATION, DATA
    UINT32 type;
    /// \brief смещение от начала буфера в 32-х разрядных словах
    /// INSTR - max 4095, OPERATION - max 4095, DATA - max 8191
    UINT32 shift;
    // размер поля dwords
    UINT32 length;
    // данные
    UINT32 dwords[BC_RAM_MAX];
    // канал
    UINT8 channel;
} BC_RAM_BLOCK;

/// \remark константы для команды записи/чтения BC RAM

#define MIL_BC_RAM_TYPE_INSTRUCTION      0x0
#define MIL_BC_RAM_TYPE_OPERATION        0x1
#define MIL_BC_RAM_TYPE_DATA             0x2
```

Рисунок 26 – Листинг команды

<i>Из</i>	<i>Под</i>	<i>Дат</i>

4.1.27 Получить кол-во блоков прерываний - IOCTL_GET_COUNT_INTERRUPT_BLOCKS

Позволяет получить кол-во блоков с информацией о возникших прерываниях из кольцевого буфера.

Описание параметров команды приведено на рисунке 27.

```
#define IOCTL_GET_COUNT_INTERRUPT_BLOCKS
    CTL_CODE(FILE_DEVICE_CONTROLLER, 50, METHOD_BUFFERED,
FILE_ANY_ACCESS)

//входные данные
UINT8 //номер канала
//выходные данные
UINT32 //количество блоков
```

Рисунок 27 – Листинг команды

<i>Из</i>	<i>Под</i>	<i>Дат</i>

4.1.28 Считать блоки прерываний - IOCTL_READ_INTERRUPT_BLOCKS

Позволяет считать заданное кол-во блоков из кольцевого буфера.

Описание параметров команды приведено на рисунке 28.

```
#define IOCTL_READ_INTERRUPT_BLOCKS CTL_CODE(FILE_DEVICE_CONTROLLER, 51,  
METHOD_BUFFERED, FILE_ANY_ACCESS)  
  
struct block_info {  
    UINT32    free_timer_value; // значение таймера в момент  
прерывания  
    UINT32    interrupt_ch; // маска прерываний  
    UINT32    hw_stat_reg1; // регистр HW_STAT_REG1  
    UINT32    hw_stat_reg2; // регистр HW_STAT_REG2  
};  
  
#define INTERRUPT_BLOCKS_MAX 2048  
  
// входные данные  
// блок данных для чтения накопленных прерываний  
typedef struct {  
    UINT8      channel;  
    UINT32     count_blocks;  
    // неприменимо  
    struct block_info blocks[INTERRUPT_BLOCKS_MAX];  
} INTERRUPT_BLOCK_BUFFER;  
  
// выходные данные  
// блок данных для чтения накопленных прерываний  
typedef struct {  
    // кол-во блоков в ответе  
    UINT32     count_blocks;  
    // массив блоков  
    struct block_info blocks[INTERRUPT_BLOCKS_MAX];  
} INTERRUPT_BLOCK_BUFFER;  
  
// биты маски прерываний  
#define BLINF_INT_HDAT 0  
#define BLINF_INT_QDAT 1  
#define BLINF_RT_INT_SADDR 2  
#define BLINF_RT_INT_MC_ERR 3  
#define BLINF_INT_BC 4  
#define BLINF_INT_FLASH 5  
#define BLINT_INT_DATA_CNT_EN 6  
#define BLINT_TIMEOUT_ITVEN 7  
#define BLINT_TIMEOUT_ABSEN 8
```

Рисунок 28 – Листинг команды

<i>Из</i>	<i>Под</i>	<i>Дат</i>

4.1.29 Считать входной подадрес - IOCTL_GET_RCV_SUBADR_DATA

Позволяет считать данные из входного подареса.

Описание параметров команды приведено на рисунке 29.

```
#define IOCTL_GET_RCV_SUBADDR_DATA
    CTL_CODE(FILE_DEVICE_CONTROLLER, 52, METHOD_BUFFERED,
FILE_ANY_ACCESS)

//входные данные
typedef struct {
    /// \brief подадрес (1-30)
    UINT8 sa;
    //неприменимо
    UINT16 words[32];
    // канал
    UINT8 channel;
} SUBA_DATA_BLOCK;

//выходные данные
typedef struct {
    // подадрес (1-30)
    UINT8 sa;
    // слова данных
    UINT16 words[32];
    // канал
    UINT8 channel;
} SUBA_DATA_BLOCK;
```

Рисунок 29 – Листинг команды

<i>Из</i>	<i>Под</i>	<i>Дат</i>

4.2 ВЗАИМОДЕЙСТВИЕ С ДРАЙВЕРОМ MIL1553UD

Взаимодействие с драйвером производится с помощью IOCTL вызовов. Определения кодов IOCTL вызовов находятся в файле ioctl_def.h.

Для работы необходимо сперва получить HANDLE устройства с помощью CreateFile:

```
HANDLE getDevice() {
    HANDLE hDevice = NULL;

    hDevice = CreateFile(L"\\\\\\?\\GLOBALROOT\\Device\\mil1553ud_0",
        GENERIC_READ | GENERIC_WRITE,
        FILE_SHARE_READ | FILE_SHARE_WRITE,
        NULL,
        OPEN_EXISTING,
        0,
        NULL);
    if (hDevice == INVALID_HANDLE_VALUE) {
        printf("Error: x%X\\n", GetLastError());
    }
    return hDevice;
}
```

Рисунок 30 – листинг получения HANDLE устройства

Из	Под	Дат

4.3 Пример вызова IOCTL

После успешного открытия устройства следует осуществить IOCTL вызов VERSION для запроса информации об устройстве. Запрос вернёт коды производителя и устройства, ревизию и количество доступных каналов:

```
VERSION v;  
HANDLE hDevice = getDevice();  
DWORD ioctl = IOCTL_VERSION, rsize;  
  
DeviceIoControl(dev, ioctl, NULL, 0, &v, sizeof(VERSION),  
&rsize, NULL);  
  
printf("VEN x%X, DEV x%X rev.%d\tchannels: %d\n", v.vendor_id,  
v.device_id, v.revision, v.type);
```

Рисунок 31 – запрос версии устройства

Из	Под	Дат

5 Библиотека mil1553ud

5.1 Назначение программы

Программное обеспечение «Библиотека взаимодействия MIL1553ud» (далее – библиотека) обеспечивает вспомогательный сервисный функционал при взаимодействии с PCI-устройством MIL1553UD.

Библиотека обеспечивает выполнение следующих основных задач:

- поиск присутствующих в системе устройств
- реализация сервисных функций поканально

5.2 Подключение библиотеки к разрабатываемому проекту

Для получения доступа к функциям библиотеки в разрабатываемый проект необходимо подключить заголовочный файл "*libmil1553ud.h*". Далее, в настройках сборщика указать путь к файлу библиотеки "*libmil1553ud.lib*"

<i>Из</i>	<i>Под</i>	<i>Дат</i>

5.3 Список доступных функций

Название вызова	Краткое описание
mil1553UdEnumerateDevices	Получить список устройств
mil1553UdFreeDeviceList	Освободить память списка устройств
mil1553UdOpen	Открыть устройство
mil1553UdReadReg	Прочитать регистр
mil1553UdWriteReg	Записать регистр
mil1553UdGetCountDmaBlocksReadyRead	Считать количество доступных блоков DMA
mil1553UdGetCountInterruptBlocksReadyRead	Считать количество доступных блоков прерываний
mil1533udReadInterruptBlocks	Считать блоки прерываний
mil1533udReadDmaBlocks	Считать блоки DMA
mil1533udFreeBlockInfo	Освободить память блоков прерываний
mil1553UdDmaBlockIsSuccessTransaction	Разбор DMA - успешность транзакции
mil1553UdDmaBlockGetTypeTransaction	Разбор DMA - тип транзакции
mil1553UdDmaBlockGetFreeTimer	Разбор DMA - значение таймера free
mil1553UdDmaBlockGetCountServiceWords16	Разбор DMA - количество служебных слов
mil1553UdDmaBlockGetCountDataWords16	Разбор DMA - количество слов данных
mil1553UdDmaBlockTdModeSubaddress	Разбор DMA - значение подадреса
mil1553UdDmaBlockGetActiveBus	Разбор DMA - используемая шина
mil1553UdDmaBlockGetPtrDataPart	Разбор DMA - получить указатель на слова данных
mil1553UdNewBcRamBlock	Создать структуру BC_RAM_BLOCK нужного размера
mil1553UdFreeBcRamBlock	Освободить память структуры BC_RAM_BLOCK
mil1553UdBcProgram	Запуск программы КШ

<i>Из</i>	<i>Под</i>	<i>Дат</i>

5.4 Описание функций

5.4.1 mil1553UdEnumerateDevices

Назначение: получение списка присутствующих в системе совместимых устройств

Аргументы: НЕТ

Результат: указатель на структуру Mil1553UdDeviceList.

После вызова функции необходимо очистить указатель на список устройств с помощью mil1553UdFreeDeviceList.

Пример вызова: см. рис.б.1

5.4.2 mil1553UdFreeDeviceList

Назначение: освобождение памяти, занятой структурой списка устройств.

Аргументы: указатель на структуру Mil1553UdDeviceList.

Результат: НЕТ

Пример вызова: см. рис.б.1

```
//Получение списка устройств
Mil1553UdDeviceList* devlist = mil1553UdEnumerateDevices();
printf("Количество устройств: %d\n\n", devlist->count);
for(int i=0;i<devlist->count;i++){
    printf("Коды: dev_x%X, ven_x%X, каналов: %d\n",
           devlist->device[i].deviceId, devlist->device[i].vendorId,
           devlist->device[i].channelCount);
    printf("Путь: %ls\n", devlist->device[i].devicePath);
    printf("Местоположение: %ls\n\n", devlist->device[i].deviceLocation);
}
mil1553UdFreeDeviceList(devlist);
```

Рисунок б.1

5.4.3 mil1553UdOpen

Назначение: открытие символического устройства

Аргументы:

1. WCHAR*devicePath - путь к устройству

Результат: HANDLE хэндл устройства или INVALID_HANDLE_VALUE

Пример вызова: см. рис.б.2

```
//Получение хэндла устройства
Mil1553UdDeviceList* devlist = mil1553UdEnumerateDevices();
printf("Количество устройств: %d\n\n", devlist->count);
if(devlist->count > 0){
    HANDLE hDevice = mil1553UdOpen(devlist->device[0].devicePath);
    CloseHandle(hDevice);
}
mil1553UdFreeDeviceList(devlist);
```

Рисунок б.2

Из	Под	Дат

5.4.4 mil1553UdReadReg

Назначение: чтение регистра устройства

Аргументы:

1. HANDLE dev - хэндл устройства
2. UINT32 address - адрес регистра
3. UINT8 channel - номер канала (отсчёт с нуля)

Результат: UINT32 - содержимое регистра

Пример вызова: см. рис.б.3

5.4.5 mil1553UdWriteReg

Назначение: запись регистра устройства

Аргументы:

1. HANDLE dev - хэндл устройства
2. UINT32 address - адрес регистра
3. UINT8 channel - номер канала (отсчёт с нуля)
4. UINT32 data - содержимое регистра

Результат: НЕТ

Пример вызова: см. рис.б.3

```
//чтение и запись регистров
HANDLE hDevice = mil1553UdOpen(devicePath);
UINT32 reg;
reg = mil1553UdReadReg(hDevice, BC_CONF_REG_PCI_CH1, 0);
printf("x%X\n", reg);
mil1553UdWriteReg(hDevice, BC_CONF_REG_PCI_CH1, 0, 1 << 15);
CloseHandle(hDevice);
```

Рисунок б.3

Из	Под	Дат

5.4.6 mil1553UdGetCountDmaBlocksReadyRead

Назначение: получение количества доступных для чтения блоков DMA

Аргументы:

1. HANDLE dev - хэндл устройства
2. UINT8 channel - номер канала (отсчёт с нуля)

Результат: UINT32 - количество блоков

Пример вызова: см. рис.б.4

5.4.7 mil1553UdGetCountInterruptBlocksReadyRead

Назначение: получение количества доступных для чтения блоков прерываний

Аргументы:

1. HANDLE dev - хэндл устройства
2. UINT8 channel - номер канала (отсчёт с нуля)

Результат: UINT32 - количество блоков

Пример вызова: см. рис.б.4

```
//получение количества блоков DMA и прерываний
HANDLE hDevice = mil1553UdOpen(devicePath);
UINT32 countDMABlocks, countInterrupts;
countDMABlocks = mil1553UdGetCountDmaBlocksReadyRead(hDevice, 0);
countInterrupts = mil1553UdGetCountInterruptBlocksReadyRead(hDevice, 0);
printf("DMA: %d, Interrupts: %d\n");
CloseHandle(hDevice);
```

Рисунок б.4

<i>Из</i>	<i>Под</i>	<i>Дат</i>

5.4.8 mil1533udReadDmaBlocks

Назначение: чтение блоков DMA

Аргументы:

1. HANDLE dev - хэндл устройства
2. UINT8 channel - номер канала (отсчёт с нуля)
3. UINT32 count - максимальное количество блоков
4. UINT8** dmaBuffer - указатель на неинициализированную область

памяти

Результат: UINT32 - количество считанных блоков

После вызова функции необходимо освободить dmaBuffer.

Пример вызова: см. рис.б.5

```
//Чтение блоков DMA
HANDLE hDevice = mil1533UdOpen(devicePath);
UINT32 countDMABlocks, count;
UINT8* dmaBuffer;
countDMABlocks = mil1533UdGetCountDmaBlocksReadyRead(hDevice, 0);
count = mil1533udReadDmaBlocks(hDevice, 0, countDMABlocks, &dmaBuffer);
for(int i=0;i<count;i++){
    //пример реализации processDMABlock - далее, см. рис. 7
    processDMABlock(dmaBuffer + (i * DMA_RAW_BLOCK_SIZE), MIL1533UD_BC_DMA_BLOCK);
}
free(dmaBuffer);
CloseHandle(hDevice);
```

Рисунок б.5

Из	Под	Дат

5.4.9 mil1533udReadInterruptBlocks

Назначение: чтение блоков прерываний

Аргументы:

1. HANDLE dev - хэндл устройства
2. UINT8 channel - номер канала (отсчёт с нуля)
3. UINT32 count - максимальное количество блоков
4. BLOCK_INFO** pblock_info - указатель на неинициализированную

область памяти

Результат: UINT32 - количество считанных блоков

После вызова функции необходимо освободить block_info с помощью вызова mil1533ud_freeBlockInfo.

Пример вызова: см. рис.б.6

5.4.10 mil1533udReadInterruptBlocks

Назначение: освобождение памяти, занятой блоком прерываний

Аргументы:

1. BLOCK_INFO* pblock_info - указатель на область памяти, занятую блоком прерываний

Результат: НЕТ

Пример вызова: см. рис.б.6

```

//Чтение блоков прерываний
HANDLE hDevice = mil1533UdOpen(devicePath);
UINT32 countInterrupts, count;
UINT8* dmaBuffer;
countInterrupts = mil1533UdGetCountInterruptBlocksReadyRead (hDevice, 0);
if(countInterrupts){
    count = mil1533udReadInterruptBlocks(dev, channel, countInterrupts, &pblock_info);
    for(int i=0;i<count;i++){
        //вывод значения free таймера и содержимого регистра прерываний канала
        printf("%u: x%X\n", pblock_info[i].free_timer_value,
pblock_info[i].interrupt_ch);
    }
    free(pblock_info);
}
CloseHandle(hDevice);

```

Рисунок б.6

Из	Под	Дат

5.4.11 mil1553UdDmaBlockIsSuccessTransaction

Назначение: Обработка блока DMA - проверка успешности транзакции

Аргументы:

1. UINT32 typeDmaBlock - тип блока DMA
2. UINT8* dmaBlock - указатель на первый байт блока DMA

Результат: UINT32 - MIL1553UD_TRUE или MIL1553UD_FALSE

Пример вызова: см. рис.б.7

5.4.12 mil1553UdDmaBlockGetTypeTransaction

Назначение: Обработка блока DMA - получение типа транзакции

Аргументы:

1. UINT32 typeDmaBlock - тип блока DMA
2. UINT8* dmaBlock - указатель на первый байт блока DMA

Результат: UINT32 - тип транзакции

Пример вызова: см. рис.б.7

5.4.13 mil1553UdDmaBlockGetFreeTimer

Назначение: Обработка блока DMA - получение значения таймера free

Аргументы:

1. UINT32 typeDmaBlock - тип блока DMA
2. UINT8* dmaBlock - указатель на первый байт блока DMA

Результат: UINT32 - значение таймера

Пример вызова: см. рис.б.7

5.4.14 mil1553UdDmaBlockGetCountServiceWords16

Назначение: Обработка блока DMA - получение количества служебных слов

Аргументы:

1. UINT32 typeDmaBlock - тип блока DMA
2. UINT8* dmaBlock - указатель на первый байт блока DMA

Результат: UINT32 - количество служебных слов

Пример вызова: см. рис.б.7

5.4.15 mil1553UdDmaBlockGetCountDataWords16

Назначение: Обработка блока DMA - получение количества слов данных

Аргументы:

1. UINT32 typeDmaBlock - тип блока DMA
2. UINT8* dmaBlock - указатель на первый байт блока DMA

Результат: UINT32 - количество слов данных

Пример вызова: см. рис.б.7

Из	Под	Дат

5.4.16 mil1553UdDmaBlockTdModeSubaddress

Назначение: Обработка блока DMA - получение подадреса

Аргументы:

1. UINT32 typeDmaBlock - тип блока DMA
2. UINT8* dmaBlock - указатель на первый байт блока DMA

Результат: UINT32 - подадрес

Пример вызова: см. рис.б.7

5.4.17 mil1553UdDmaBlockGetActiveBus

Назначение: Обработка блока DMA - получение активной шины (А или В)

Аргументы:

1. UINT32 typeDmaBlock - тип блока DMA
2. UINT8* dmaBlock - указатель на первый байт блока DMA

Результат: UINT32 - MIL1553_BUS_V или MIL1553_BUS_A

Пример вызова: см. рис.б.7

5.4.18 mil1553UdDmaBlockGetPtrDataPart

Назначение: Обработка блока DMA - получение указателя на слова данных

Аргументы:

1. UINT32 typeDmaBlock - тип блока DMA
2. UINT8* dmaBlock - указатель на первый байт блока DMA

Результат: UINT8* - указатель на слова данных

Пример вызова: см. рис.б.7

```
//Обработка блока DMA
void processDMABlockBC(UINT8* block){
    UINT32 i;
    UINT32 type = MIL1553UD_BC_DMA_BLOCK;
    UINT8* dataPart;

    UINT32 success = mil1553UdDmaBlockIsSuccessTransaction(type, block);
    UINT32 trtype = mil1553UdDmaBlockGetTypeTransaction(type, block);
    UINT32 freetimer = mil1553UdDmaBlockGetFreeTimer(type, block);
    UINT32 countServiceWords16 = mil1553UdDmaBlockGetCountServiceWords16(type, block);
    UINT32 countDataWords16 = mil1553UdDmaBlockGetCountDataWords16(type, block);
    UINT32 subaddr = mil1553UdDmaBlockTdModeSubaddress(block);
    UINT32 activeBus = mil1553UdDmaBlockGetActiveBus(type, block);

    printf("\nTX type: %d %s\nfreeTimer %u\tserviceWords: %d\tdataWords:
%d\tsubaddress: %d\tbus: %s\n", trtype, success ? "ok" : "error", freetimer,
countServiceWords16, countDataWords16, subaddr, (activeBus == MIL1553_BUS_A) ? "A" : "B");

    if(countDataWords16 > 0){
        dataPart = mil1553UdDmaBlockGetPtrDataPart(block);
        printf(" data:");
        for(i=0;i<countDataWords16*2;i+=2){
            printf(" x%X", dataPart[i+1] << 8 | dataPart[i]);
        }
        printf("\n");
    }
}
```

Рисунок б.7

Из	Под	Дат

5.4.19 mil1553UdNewBcRamBlock

Назначение: выделение памяти под блок памяти КШ

Аргументы:

1. UINT32 type - тип блока,
MIL_BC_RAM_TYPE_INSTRUCTION, MIL_BC_RAM_TYPE_OPERATION, MIL_BC_RAM_TYPE_DATA
2. UINT32 sizeInDwords - количество слов данных
3. UINT32 shiftInDwords - смещение относительно начала

Результат: BC_RAM_BLOCK* - указатель на блок памяти КШ

Пример вызова: см. рис.б.8

5.4.20 mil1553UdFreeBcRamBlock

Назначение: освобождение памяти под блок памяти КШ

Аргументы:

1. BC_RAM_BLOCK* - указатель на блок памяти КШ

Результат: НЕТ

Пример вызова: см. рис.б.8

```
//Запись блока памяти КШ
BC_RAM_BLOCK *instruction = mil1553UdNewBcRamBlock(MIL_BC_RAM_TYPE_INSTRUCTION, 6, 0);
instruction->channel = 0;
instruction->dwords[0] = INSTRUCTION(XEQ, ALWAYS, 0);
instruction->dwords[1] = INSTRUCTION(XEQ, ALWAYS, 2);
instruction->dwords[2] = INSTRUCTION(IRQ, 1, 0);
instruction->dwords[3] = INSTRUCTION(XEQ, ALWAYS, 0);
instruction->dwords[4] = INSTRUCTION(XEQ, ALWAYS, 2);
instruction->dwords[5] = INSTRUCTION(HLT, ALWAYS, 0);

DeviceIoControl(hDevice, IOCTL_WRITE_BC_RAM, instruction, sizeof(BC_RAM_BLOCK), NULL, 0,
&size, NULL);

mil1553UdFreeBcRamBlock(instruction);
```

Рисунок б.8

<i>Из</i>	<i>Под</i>	<i>Дат</i>

5.4.21 mil1553UdBcProgram

Назначение: запуск программы КШ

Аргументы:

1. HANDLE hDev - хэндл устройства
2. UINT8 chnum - номер канала
3. UINT32 action - действие

Результат: BOOL - результат вызова DeviceIoControl

Пример вызова: см. рис.б.9

```
//Запуск программы КШ  
mil1553UdBcProgram(hDevice, 0, BC_PROGRAM_START);
```

Рисунок б.9

<i>Из</i>	<i>Под</i>	<i>Дат</i>

СПИСОК СОКРАЩЕНИЙ

ПО – программное обеспечение;

МКИО – интерфейс по ГОСТ 52070;

КШ – контроллер шины;

ОУ – оконечное устройство;

МШ – монитор шины;

МША – монитор шины адресный;

<i>Из</i>	<i>Под</i>	<i>Дат</i>

