Белорусский Государственный Университет
Информатики и Радиоэлектроники
Кафедра ЭВМ

Отчет по лабораторной работе № 6

Тема:
«Высокоуровневые библиотеки
SD-карта. Прямой доступ к памяти»
Вариант 12

Выполнили:

ст. гр. 950503
Зарубо Д. Ю
Ященко В.П

Проверил:
Шеменков В.В

Минск 2022

1 Цели работы

В ходе выполнения лабораторной работы необходимо изучить принципы организации прямого доступа к памяти на базе микроконтроллера MSP430F5529 и работы с SD-картой на основе экспериментальной платы MSP-EXP430F5529. Получить навыки комплексного использования периферийных устройств микроконтроллера MSP430F5529 и устройств экспериментальной платы MSP-EXP430F5529.

2 Исходные данные к работе

Для выполнения лабораторной работы используется плата MSP-EXP430F5529 с использованием среды разработки Code Composer Studio. В процессе выполнения работы требуется написать программу, организующую обмен с FLASH-памятью, SD-картой с применением прямого доступа к памяти и с использованием высокоуровневых библиотек

3 Теоретические сведения

3.1 Высокоуровневые библиотеки

Примеры программ для микроконтроллера MSP430F5529 написаны с использованием API для управления компонентами микроконтроллера и экспериментальной платы. Заголовочные файлы API находятся по *TI / msp430 / src /*.*. Структура библиотеки следующая:

- */CTS* – библиотека для поддержки функций сенсорной клавиатуры;
  - */structure.h* – описание используемых библиотекой структур данных;
  - */CTS_HAL.h* – функции ядра библиотеки, поддержка методов измерения RO, fRO, RC, установка прерываний таймеров;
  - */CTS_Layer.h* – слой API, содержит функции отслеживания базового уровня сенсора, определения нажатия каждого сенсора и т.д.;
- */F5xx_F6xx_Core_Lib* – библиотека ядра;
  - */HAL_UCS.h* – функции работы с унифицированной системой тактирования — выбор источников сигнала MCLK, SMCLK, ACLK, установка делителя, настройки генераторов XT1, XT2, режим блока FLL;
  - */HAL_PMM.h* – функции работы с менеджером питания;
  - */HAL_FLASH.h* – библиотека для работы с FLASH-памятью;
- */FatFs* – стек файловой системы FAT для поддержки SD-карты;
- */MSP-EXP430F5529_HAL* — библиотека для поддержки основных устройств экспериментальной платы;

- */HAL_Wheel.h* – работа с потенциометром;
- */HAL_SDCard.h* – работа с SD-картой памяти;
- */HAL_Dogs102x6.h* – работа с ЖКИ экраном, включая простейшие графические функции;
- */HAL_Cma3000.h* – работа с акселерометром;
- */HAL_Buttons.h* – работа с кнопками;
- */HAL_Board.h* – работа со светодиодами;
- */HAL_AppUart.h* – работа с USCI в режиме UART;

- */USB* – стек USB для экспериментальной платы;
- */UserExperienceDemo* — пример приложения с использованием высокоуровневых библиотек. Именно это приложение использовалось в лабораторной работе №1 для знакомства с комплектом.

3.2 SD-карта памяти

MMC/SD или SD-карта памяти представляет собой функционально и конструктивно законченный модуль, который содержит в своем составе собственно память и управляющий микроконтроллер. Обмен данными возможен по двум протоколам: MMC и SPI. Протокол MMC обеспечивает большую скорость и возможность параллельного включения нескольких карт и является основным. Тем не менее для многих платформ удобнее использовать протокол SPI, который поддерживается микроконтроллером на аппаратном уровне.

Схема подключения SD-карты памяти приведена на рисунке 6.1. Как видно из рисунка, карта подключается по интерфейсу SPI, причем на одном и том же канале, что и ЖКИ. Соответственно, при программировании необходимо учитывать этот момент: на шине SPI не должны одновременно присутствовать два устройства, которым разрешена работа.

Соответствие выводов разъема SD-карты и микроконтроллера приведены в таблице 3.1.

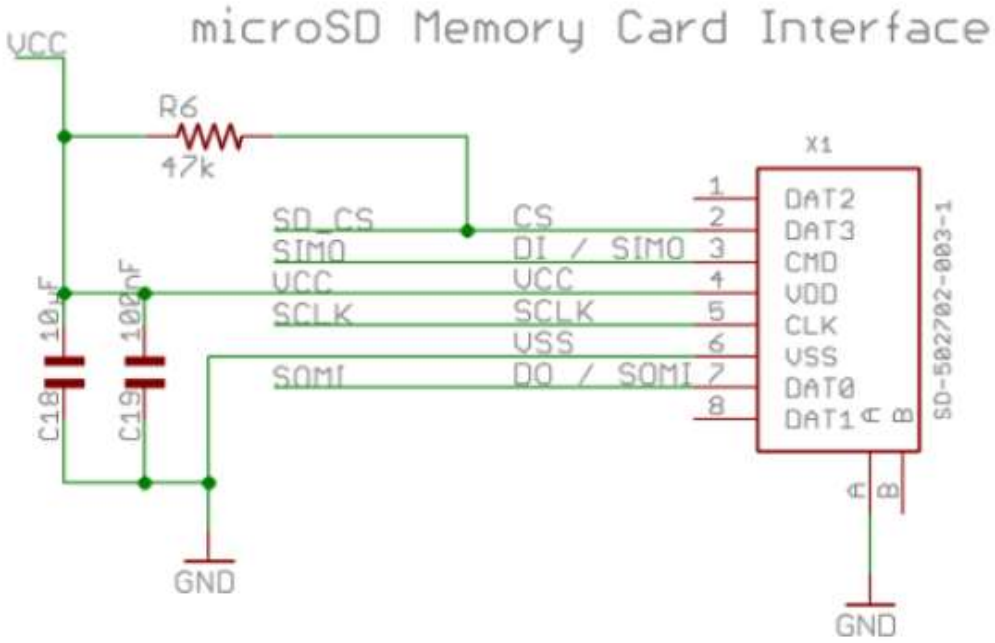

Рис. 3.1 Схема подключения разъема SD-карты памяти

Заголовочный файл *HAL_SDCard.h* определяет следующие функции:
void SDCard_init(void) — подключение линий микроконтроллера и инициализация интерфейса SPI в режиме 3-проводной, Master, MSB, 8-бит, активный уровень CLK – низкий, источник тактирования SMCLK, частота тактирования 397 КГц (при инициализации должна быть менее 400 КГц).
void SDCard_fastMode(void) — устанавливает тактовую частоту 12,5 МГц для быстрого обмена.
*void SDCard_readFrame(uint8_t *pBuffer, uint16_t size)* — чтение данных из памяти, 1 параметр — указатель на буфер приема, 2 параметр — количество байт.
*void SDCard_sendFrame(uint8_t *pBuffer, uint16_t size)* — запись данных в память, 1 параметр — указатель на буфер с данными, 2 параметр — количество байт.
void SDCard_setCSHigh(void) — установка сигнала выбора устройства в 1.
void SDCard_setCSLow(void) — сброс сигнала выбора устройства в 0.

Таблица 3.1. Соответствие выводов разъема SD-карты памяти

Выводы SD-разъема	**Обозначение линии на схеме**	**Назначение**	**Вывод MSP430F5529**	**Требуемый режим**
DAT3	SD_CS	Разрешение устройства	P3.7/TB0OUTH	P3.7

Выводы SD-разъема	Обозначение линии на схеме	Назначение	Вывод MSP430F5529	Требуемый режим
CMD	SIMO	SIMO данные (запись в память)	P4.1/ PM_UCB1SIMO/ PM_UCB1SDA	PM_UCB1SIMO
CLK	SCLK	Синхросигнал	P4.3/ PM_UCB1CLK/ PM_UCA1STE	PM_UCB1CLK
DAT0	SOMI	SOMI данные (чтение из памяти)	P4.2/ PM_UCB1SOMI/ PM_UCB1SCL	PM_UCB1SOMI

Однако эти функции всего лишь подключают SPI интерфейс, и этого недостаточно для работы с MMC/SD картой. Для корректной связи с ней необходимо поддерживать протокол обмена, установленный для MMC/SD. Кратко рассмотрим его особенности. Более подробно работа с SD-картой памяти рассмотрена в [28, 29].

MMC/SD карта принимает от микроконтроллера ряд команд, на которые она выдаёт либо ответы определённого типа, либо блоки данных. Ответ – R1, R2 или R3 – может состоять из одного, двух или пяти байтов. Формат обмена представлен на рис. 3.2.

Блоки данных могут быть различной длины и состоят из стартового байта (0xFE), собственно данных (их длина 1...N байт, где N определяется размером физического сектора, в большинстве случаев – 512 байт) и двух байт контрольной суммы. Контрольная сумма является опциональной в SPI интерфейсе и, как правило, не используется для упрощения процедуры обмена. Значения двух байт контрольной суммы можно игнорировать, но сами эти байты должны обязательно передаваться/приниматься для соблюдения протокола обмена.

Перед передачей команды или после этого микроконтроллер должен выдавать не менее 8 тактовых импульсов по линии CLK, т.е. просто передавать «лишний» байт 0xFF. При чтении блока данных после передачи соответствующей команды микроконтроллер принимает байты 0xFF до тех пор, пока не встретится байт 0xFE (стартовый байт блока данных). Любой иной байт (отличный от 0xFF), полученный в этот момент, будет означать ошибку.

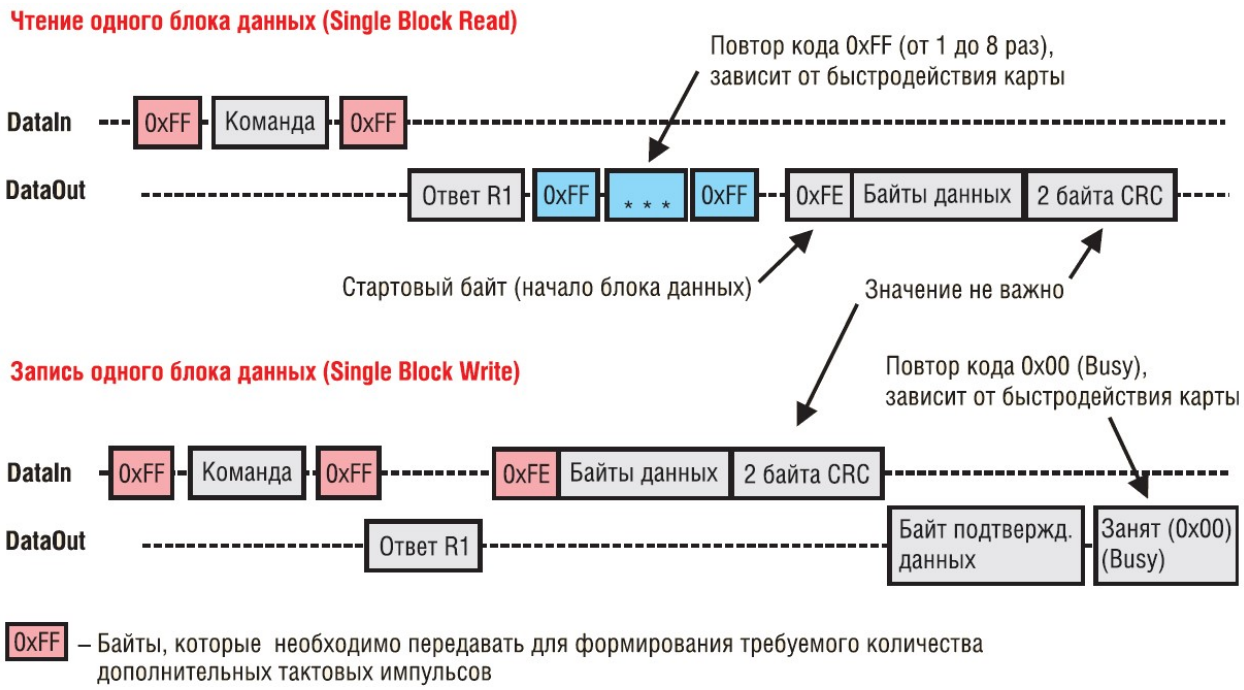

Рис. 3.2 Формат обмена с MMC/SD-картой памяти

Все команды, воспринимаемые MMC/SD картой, имеют длину 6 байт. Индекс команды (порядковый номер) находится в битах 0..5 первого байта команды, биты 7 и 6 всегда содержат 0 и 1 соответственно. Следующие 4 байта содержат аргумент команды, например, 32-битный адрес первого байта данных. Последний байт команды сдержит в битах 1..7 контрольную сумму, бит 0 всегда равен 1.

MMC/SD карта после приёма команды выдаёт ответ, содержащий один, два или пять байт. Первым передаётся старший байт. Ответ формата R1 содержит один байт. Структура ответа R1:

- бит 7 – всегда 0;
- бит 6 — ошибка параметра команды;
- бит 5 — ошибка адреса;
- бит 4 — ошибка стирания;
- бит 3 — ошибка контрольной суммы CRC;
- бит 2 — неверная команда;
- бит 1 — прервана команда стирания;
- бит 0 — режим простоя, выполняется инициализация.

Ответ R2 состоит из двух байт, причём первый байт ответа идентичен структуре ответа R1. Структура второго байта в ответе R2:

- бит 7 — выход за пределы / ошибка перезаписи;
- бит 6 — ошибка параметра при стирании;

- бит 5 — попытка записи в защищенную от записи область;
- бит 4 — ошибка коррекции;
- бит 3 — внутренняя ошибка;
- бит 2 — общая / неизвестная ошибка;
- бит 1 — попытка стирания защищенного от записи сектора / ошибка блокирования/разблокирования;
- бит 0 — карта заблокирована.

Ответ R3 состоит из 5 байт. Первый байт идентичен ответу R1, остальные 4 байта представляют собой содержимое регистра OCR.
При записи данных в MMC/SD карту после получения блока данных карта отвечает байтом подтверждения данных. Бит 4 подтверждения всегда равен 0, бит 0 — 1. В битах 1..3 указывается статус операции, успешной записи соответствует значение 010.
Команды записи и чтения сопровождаются пересылкой блоков данных. Каждый блок данных начинается со стартового байта. Следующий за ним байт – это фактические данные. Завершаются фактические данные двумя байтами контрольной суммы (16 бит CRC). Так как в режиме SPI контрольную сумму можно не вычислять, значения этих двух байтов не имеют значения, но сами байты контрольной суммы обязательны. Если операция чтения данных завершилась неудачно и карта не может предоставить запрашиваемые данные, то она будет посылать байт ошибки данных.

После подачи напряжения питания MMC/SD карта находится в режиме MMC, а не в режиме SPI. Для перевода карты в режим SPI и инициализации карты необходимо выполнить определённую последовательность действий:

- не выбирая устройство (сигнал CS = 1) послать 80 импульсов по линии CLK (передать 10 байт 0xFF);
- выбрать MMC/SDC карту (CS = 0);
- послать команду CMD0 (сброс): 0x40, 0, 0, 0, 0, 0x95 (в этой команде контрольная сумма должна иметь реальное значение (0x95), т.к. данная команда посылается в тот момент, когда MMC/SD карта находится в режиме MMC, а не SPI);
- дождаться правильного ответа 0x01;
- в цикле посылать команду CMD1 (инициализация) и ждать, когда будет получен ответ 0x00 (этот ответ означает, что карта инициализирована в режиме SPI и готова принимать команды). Для SD-карт в случае отклонения команды CMD1 рекомендуется использовать команду ACMD41.

В таблице приведён ряд команд для MMC/SD карты в режиме SPI:

Таблица 3.2. Команды MMC/SD карты памяти

Команда	Код	Аргу-мент	Ответ	Данные	Аббревиатура	Описание
CMD0	40h	Нет(0)	R1	Нет	GO_IDLE_STATE	Программный сброс
CMD1	41h	Нет(0)	R1	Нет	SEND_OP_COND	Запуск процесса инициализации
ACMD41	69h	*	R1	Нет	APP_SEND_OP_COND	Только для карт SDC. Запуск процесса инициализации
CMD8	48h	**	R7	Нет	SEND_IF_COND	Только для карт SDC v2. Проверка диапазона напряжения питания
CMD9	49h	Нет(0)	R1	Да	SEND_CSD	Чтение регистра CSD
CMD10	4Ah	Нет(0)	R1	Да	SEND_CID	Чтение регистра CID
CMD12	4Ch	Нет(0)	R1b	Нет	STOP_TRANSMISSION	Остановка чтения данных
CMD16	50h	Длина блока [31:0]	R1	Нет	SET_BLOCKLEN	Установка размера блока чтения записи
CMD17	51h	Адрес [31:0]	R1	Да	READ_SINGLE_BLOCK	Чтение блока
CMD18	52h	Адрес [31:0]	R1	Да	READ_MULTIPLE_BLOCK	Чтение нескольких блоков
CMD23	57h	Число блоков [15:0]	R1	Нет	SET_BLOCK_COUNT	Только для MMC. Указание количества блоков для передачи со следующей командой многоблочного чтения/записи
ACMD23	57h	Число блоков [22:0]	R1	Нет	SET_WR_BLOCK_ERASE_COUNT	Только для SD. Указание количества блоков для предварительного стирания для последующей команды многоблочной записи
CMD24	58h	Адрес [31:0]	R1	Да	WRITE_BLOCK	Запись блока
CMD25	59h	Адрес [31:0]	R1	Да	WRITE_MULTIPLE_BLOCK	Запись нескольких блоков
CMD55	77h	Нет(0)	R1	Нет	APP_CMD	Начало команды ACMD
CMD58	7Ah	Нет(0)	R3	Нет	READ_OCR	Чтение OCR

ACMD означает последовательность двух команд CMD55 + CMD;
* - Бит 30 - HCS, остальные в 0
** - Биты 31..12 = 0, биты 11..8 — напряжение питания, биты 7..0 — 0xAA

После простоя более 5 мс карта памяти переходит в энергосберегающий режим, и способна принимать только команды CMD0, CMD1 и CMD58. Поэтому процесс инициализации (CMD1) необходимо практически каждый раз

повторять при чтении/записи блока данных или делать проверку состояния карты.

Основные регистры контроллера карты, которые доступны по SPI протоколу:

- CID (Card identification data): содержит данные, по которым можно идентифицировать карту памяти (серийный номер, ID производителя, дату изготовления и т.д.);
- CSD (Card-specific data): содержит всевозможную информацию о карте памяти (от размера сектора карты памяти до потребления в режиме чтения/записи);
- OCR (Operation Conditions Register): содержит напряжения питания карты памяти, тип питания карты памяти, статус процесса инициализации карты.

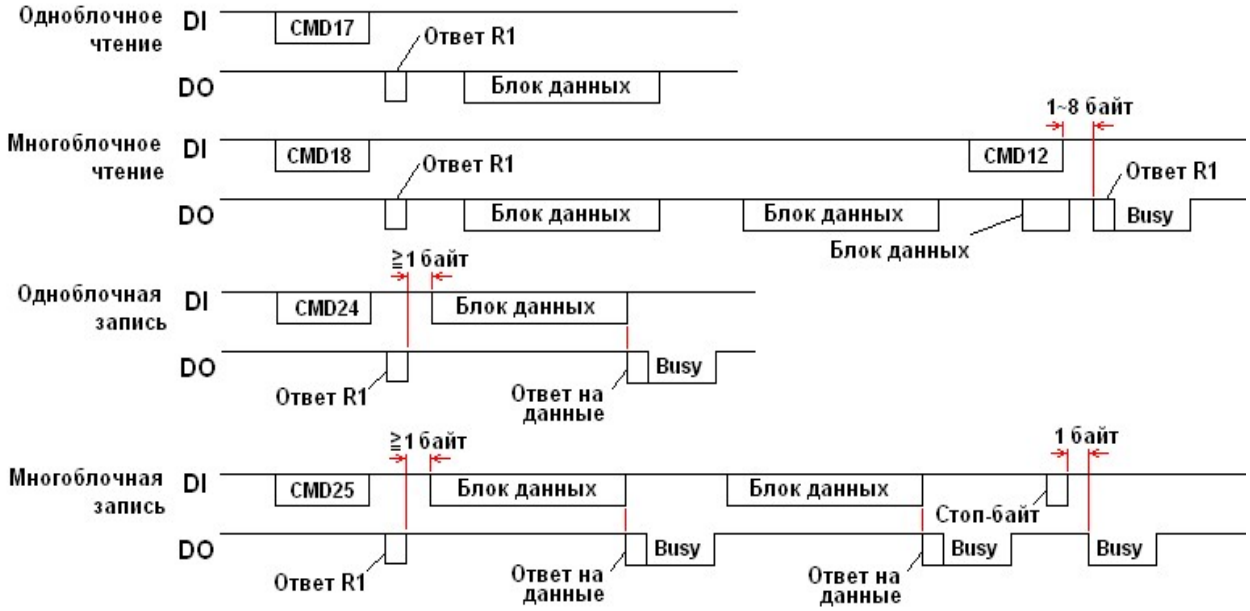

Рис. 3.3 Схема одноблочного и многоблочного чтения/записи

Схема передачи в режимах одноблочной и многоблочной чтения/записи приведена на рисунке 3.3. Одноблочное чтение инициируется командой CMD17. Ее аргумент задает адрес начала чтения. Чтение осуществляется побайтно. В ответ на команду CMD17 карта выдает ведущему контроллеру пакет данных. После обнаружения правильного маркера данных ведущий контроллер принимает следующий за ним блок данных и два байта CRC, которые необходимо принять, даже если CRC не используется. По умолчанию размер блока 512 байтов, но его можно изменить командой CMD16. Если во время операции чтения произошла какая-нибудь ошибка, вместо пакета данных будет возвращен маркер ошибки.

С помощью команды многоблочного чтения CMD18 можно прочитать последовательность нескольких блоков, начиная с заданного адреса. Если перед

этой командой с помощью команды CMD23 (только для MMC) не было задано число передаваемых блоков, будет инициировано неограниченное многоблочное чтение, то есть операция чтения будет продолжаться, пока ведущий контроллер не прервёт ее командой CMD12. Байт, получаемый сразу же после передачи CMD12, является наполняющим, его не нужно учитывать. После этого байта следует ответ на команду.

После того, как карта приняла команду записи CMD24, ведущий контроллер после байтового промежутка (один или более байтов) передает пакет данных. Формат пакета такой же, как и у команды блочного чтения. После передачи пакета карта сразу же выдает ответ на данные, за которым следует флаг занятости. Большинство карт не могут менять размер записываемого блока, он является фиксированным и составляет 512 байтов. По правилам режима SPI сигнал CS должен находится в активном уровне в течение всей транзакции, однако есть исключение из этого правила. Когда карта занята, ведущий контроллер может снять сигнал CS, чтобы освободить шину SPI для какого-нибудь другого SPI-устройства. Если же снова выбрать карту в то время, когда она занята выполнением внутреннего процесса, карта снова установит сигнал DO в низкий уровень. Поэтому, чтобы сократить время ожидания, лучше выполнять проверку на занятость непосредственно перед выдачей команды и пакета данных, а не ожидать освобождения карты после посылки команды. Кроме того, внутренний процесс инициируется спустя байт после ответа данных, т.е. необходимо выдать 8 тактовых импульсов, чтобы инициировать внутреннюю операцию записи. Состояние сигнала CS во время этих восьми тактовых импульсов не учитывается, поэтому можно совместить эту инициацию с процессом освобождения шины.

С помощью команды многоблочной записи CMD25 можно записать последовательность из нескольких блоков, начиная с заданного адреса. Если перед этой командой число передаваемых блоков не было задано с помощью команды CMD23 (только для MMC) или ACMD23 (для SD), транзакция будет инициирована как неограниченная многоблочная запись, то есть операция записи будет продолжаться, пока ведущий контроллер не прервет ее передачей маркера остановки передачи (стоп-байт, FDh). Флаг занятости появится байт спустя после стоп-байта. Что же касается SD, то транзакция многоблочной записи должна прерываться стоп-байтом независимо от того, является ли она предопределенной или неограниченной.

Функции пользовательского API, необходимые для работы с MMC/SD картой находятся в заголовочном файле */FatFs/mmc.h.* На пользовательском уровне карта памяти представляется диском. Функции этого слоя:

DSTATUS disk_status (BYTE drv) — получение состояния диска. Передается номер диска (0).

DSTATUS disk_initialize (BYTE drv) — инициализация диска. Параметры и результат аналогичны предыдущей функции.

*DRESULT disk_read (BYTE drv, BYTE *buff, DWORD sector, BYTE count)* – чтение данных с диска. Параметры: номер диска, указатель на буфер для размещения данных, начальный номер сектора для чтения (LBA), количество секторов.
*DRESULT disk_write (BYTE drv, const BYTE *buff, DWORD sector, BYTE count)* — запись данных на диск. Параметры: номер диска, указатель на буфер с данными для записи, начальный номер сектора для записи (LBA), количество секторов.
*DRESULT disk_ioctl (BYTE drv, BYTE ctrl, void *buff)* — команда управления. Параметры: номер диска, код команды, указатель на буфер для приема/передачи данных команды управления.
uint8_t detectCard(void) — обнаружение карты и попытка подключения если карта не обнаружена. Возвращает 1, если карта готова к работе, 0 — карта не обнаружена.

Этого уже достаточно, чтобы обмениваться неформатированными данными. Физически память MMC/SD карты разбита на сектора по 512 байт, карта имеет, как правило, файловую систему FAT16. Поэтому для полноценной поддержки обмена файлами, которые потом будут видимы при использовании SD-карты на других устройствах, необходимо еще и поддержать файловую систему FAT. Структуру MBR, FAT и формат каталога в этой файловой системе рассматривать не будем. Заголовочный файл */FatFs/ff.h* содержит необходимые функции:
FRESULT f_mount (BYTE, FATFS)* — подключение/отключение логического диска.
FRESULT f_open (FIL, const TCHAR*, BYTE)* — открытие или создание файла.
FRESULT f_read (FIL, void*, UINT, UINT*)* — чтение данных из файла.
FRESULT f_lseek (FIL, DWORD)* — перемещение файлового указателя.
FRESULT f_close (FIL)* — закрытие открытого файла.
FRESULT f_opendir (DIRS, const TCHAR*)* — открытие существующего каталога.
FRESULT f_readdir (DIRS, FILINFO*)* — чтение эелементов каталога.
FRESULT f_stat (const TCHAR, FILINFO*)* — получение состояния файла.
FRESULT f_write (FIL, const void*, UINT, UINT*)* — запись данных в файл.
FRESULT f_getfree (const TCHAR, DWORD*, FATFS**)* — получение количества свободных кластеров на диске.
FRESULT f_truncate (FIL)* — усечение файла.
FRESULT f_sync (FIL)* — очистка кеша для записанных данных.
FRESULT f_unlink (const TCHAR)* — удаление существующего файла или каталога.
FRESULT f_mkdir (const TCHAR)* — создание нового каталога.
FRESULT f_chmod (const TCHAR, BYTE, BYTE)* — изменение атрибутов файла или каталога.
FRESULT f_utime (const TCHAR, const FILINFO*)* — изменение времени файла или каталога.

FRESULT f_rename (const TCHAR, const TCHAR*)* — переименование или перемещение файла или каталога.
FRESULT f_forward (FIL, UINT(*)(const BYTE*,UINT), UINT, UINT*)* — помещение данных в поток.
FRESULT f_mkfs (BYTE, BYTE, UINT) — создание файловой системы на диске.
FRESULT f_chdrive (BYTE) — смена текущего диска.
FRESULT f_chdir (const TCHAR)* — смена текущего каталога.
FRESULT f_getcwd (TCHAR, UINT)* — получение текущего каталога.
int f_putc (TCHAR, FIL)* — запись символа в файл.
int f_puts (const TCHAR, FIL*)* — запись строки в файл.
int f_printf (FIL, const TCHAR*, ...)* — запись форматированной строки в файл.
TCHAR f_gets (TCHAR*, int, FIL*)* — стение строки из файла.
Кроме того, приведены следующие макроопределения:
#define *f_eof(fp)* (((fp)->fptr == (fp)->fsize) ? 1 : 0)
#define *f_error(fp)* (((fp)->flag & FA__ERROR) ? 1 : 0)
#define *f_tell(fp)* ((fp)->fptr)
#define *f_size(fp)* ((fp)->fsize)

3.3 Прямой доступ к памяти

Прежде, чем рассматривать прямой доступ к памяти, изучим, как организована работа с ОЗУ. ОЗУ (RAM) микроконтроллера MSP430F5529 разделена на 4 сектора по 2 Кб. Сектор 0 содержит адреса 002400h – 002BFFh, сектор 1 — 002C00h – 0033FFh, сектор 2 — 003400h - 003BFFh, сектор 3 — 003C00h – 0043FFh. Кроме того, имеется 2 Кб сектор USB RAM (сектор 7), который может использоваться как обычное ОЗУ, если не используется USB. Каждый из секторов может быть отключен битом RCRSyOFF регистра RCCTL0. Чтение из отключенного сектора всегда дает 0. Стек располагается в ОЗУ, поэтому нельзя отключать сектор, содержащий стек, если используются прерывания или LPM режим. В LPM режиме процессор микроконтроллера отключен, поэтому память находится в режиме удержания для уменьшения тока утечки. Адрес регистра RCCTL0 – 6900h. Доступ к этому регистру защищен ключом. Перед изменением RCCTL0_L (младшего байта, содержащего флаги отключения секторов, номер бита соответствует номеру сектора), в регистр RCCTL0_H необходимо записать код 5Ah. При чтении старший байт содержит 69h.

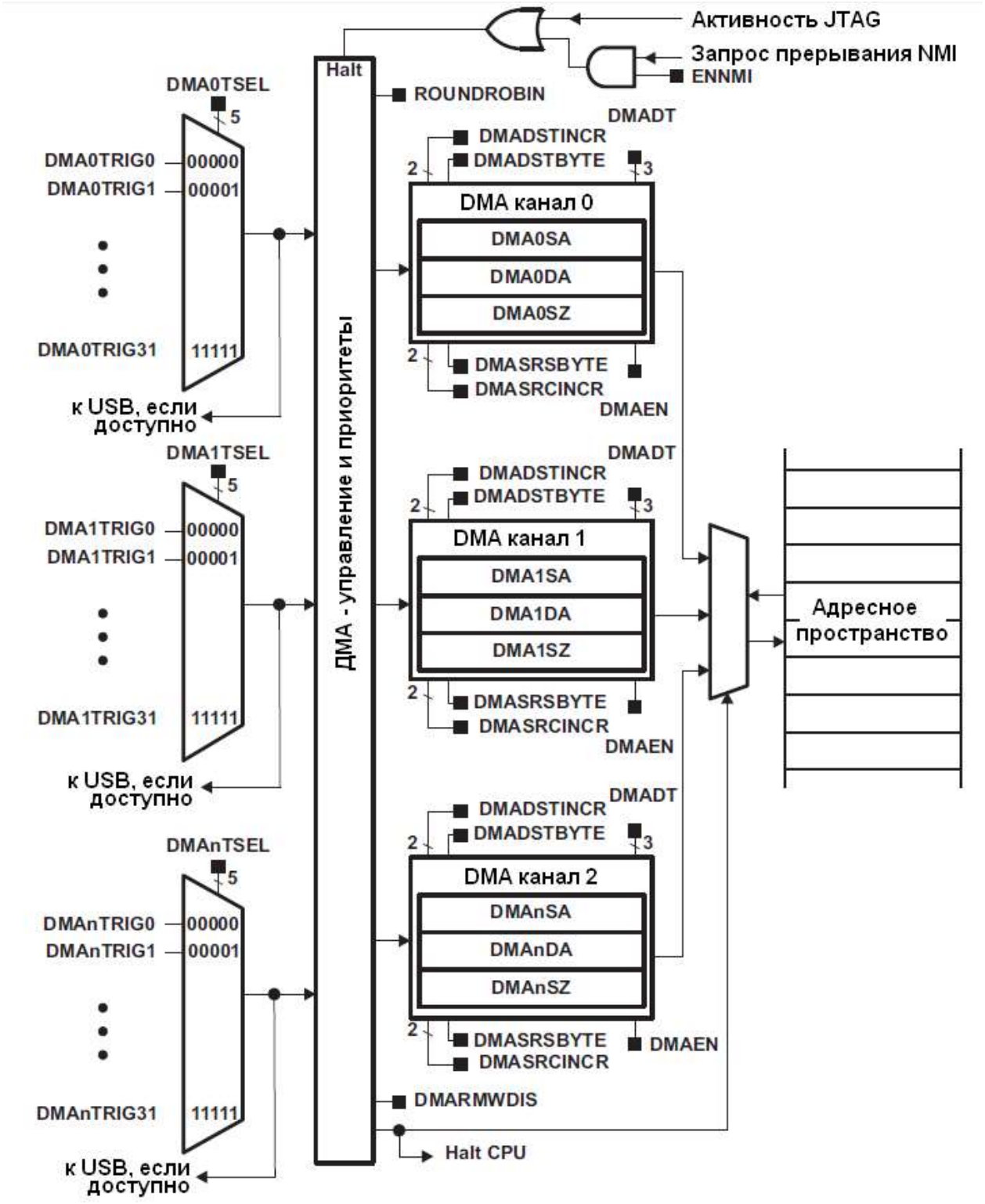

Рис. 3.4 Структура контроллера DMA

Контроллер прямого доступа к памяти (DMA) выполняет пересылку данных между адресами без участия центрального процессора. В

микроконтроллере MSP430F5529 контроллер DMA содержит 3 канала. Использование DMA может увеличить производительность периферии, а также снизить ток потребления, поскольку центральный процессор может оставаться в LPM режиме. Характеристики DMA-контроллера:

- три независимых канала;
- программируемые приоритеты каналов;
- требуется всего 2 MCLK такта на пересылку;
- возможность пересылки байт, слов или смешанные;
- размер блока данных до 65 К байт или слов;
- программируемый выбор триггеров передачи;
- пересылки по перепаду сигнала триггера или по уровню;
- 4 режима адресации;
- 3 режима пересылки: одиночные, блочные и многоблочные.

Схема контроллера представлена на рис. 6.4.

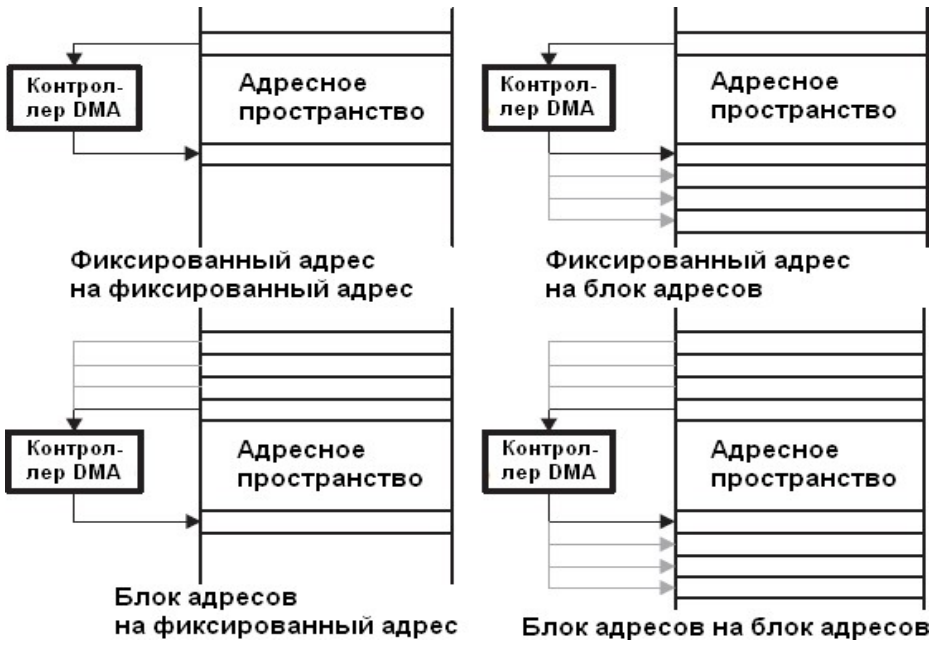

Рис. 3.5 Режимы адресации данных в DMA

Доступны следующие режимы адресации: фиксированный адрес на фиксированный адрес, фиксированный адрес на блок адресов, блок адресов на фиксированный адрес, блок адресов на блок адресов (рис. 3.5). Биты DMASRCINCR и DMADSTINCR выбирают, будут ли адреса источника и приемника, соответственно, инкрементироваться, декрементироваться или оставаться без изменений. Пересылки возможны байт в байт, байт в слово (старший байт результата обнуляется), слово в байт (пересылается младший байт источника) и слово в слово.

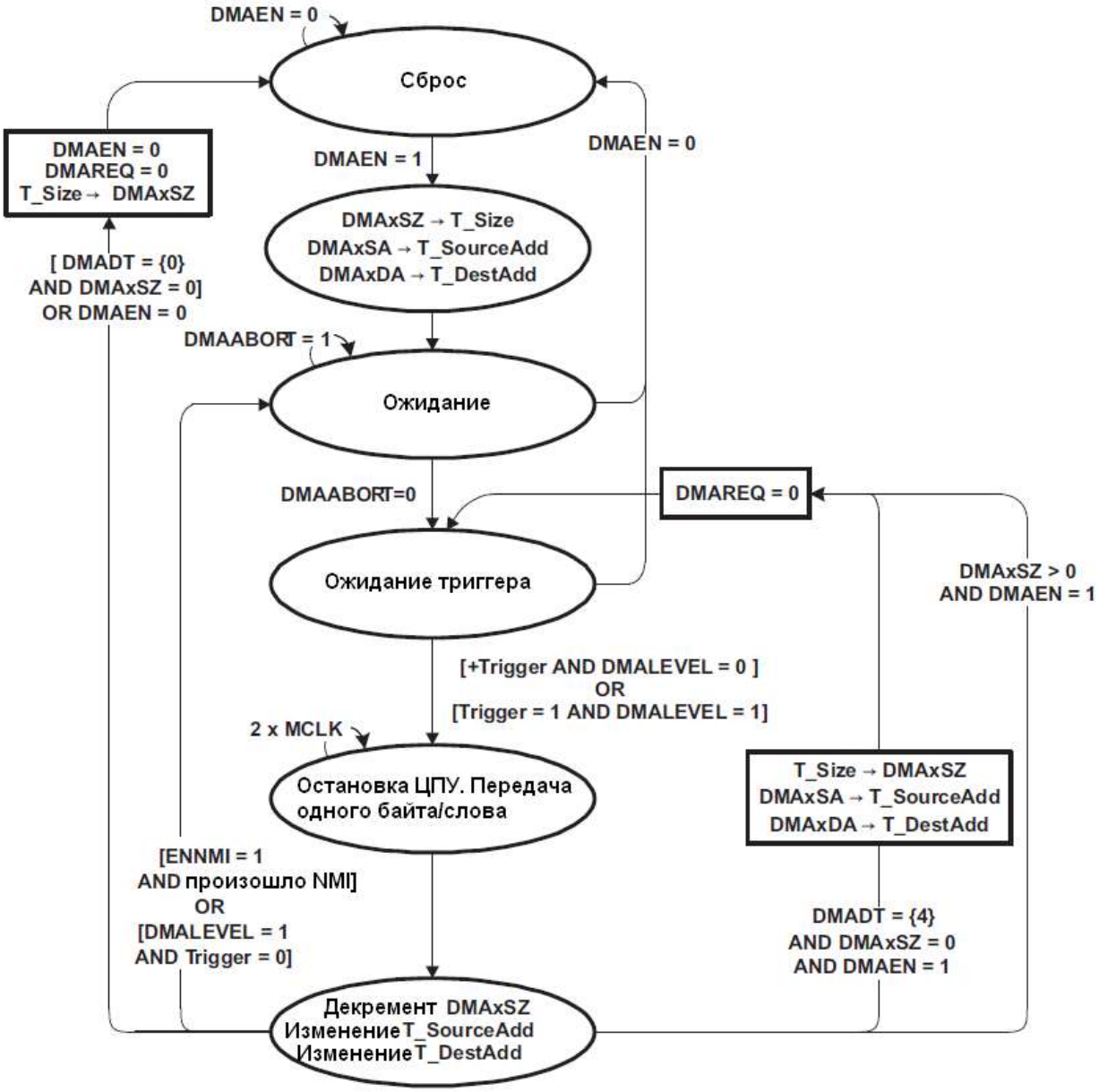

Рис. 3.6 Диаграмма состояний в режиме одиночных пересылок

Биты DMADT задают 6 режимов пересылки, программируемые отдельно для каждого из каналов:

000 — одиночная пересылка;

001 — блочная пересылка;

010, 011 — импульсная блочная пересылка;

100 — повторяющаяся одиночная пересылка;

101 — повторяющаяся блочная пересылка;

110, 111 — повторяющаяся импульсная блочная пересылка.

В режиме одиночных пересылок текущие значения адреса источника, адреса назначения и количества пересылок копируются во временные регистры, которые изменяются после каждой пересылки. Каждая пересылка требует срабатывания триггера, DMAEN автоматически сбрасывается, когда сделано DMAxSZ пересылок. Когда DMAxSZ становится равен 0, он повторно копируется, и устанавливается соответствующий флаг DMAIFG. В случае повторяющихся одиночных пересылок, DMAEN остается активным, и при каждом новом срабатывании триггера происходит пересылка. Для примера на рис. 6.6 приведена диаграмма состояний режима одиночных пересылок. В остальных режимах диаграмма незначительно отличается.

В случае блочной пересылки по срабатыванию одного триггера пересылается блок данных, в процессе пересылки игнорируются любые другие сработавшие триггеры. DMAxSZ определяет размер блока. На протяжении всего обмена процессор остановлен. DMAEN автоматически сбрасывается после передачи блока. В случае повторяющихся блочных пересылок DMAEN остается активным, и по окончании пересылки новое срабатывание триггера вызывает новую пересылку блока. В остальном режим подобен одиночным пересылкам. Пересылка занимает 2 x MCLK x DMAxSZ тактов.

В импульсно-блочных пересылках после каждых 4 пересылок байт либо слов, на 2 такта MCLK включается процессор. В результате он занимает 20% времени. В остальном режим подобен блочному. Существенное отличие в повторяющемся импульсно-блочном режиме: так как DMAEN остается активным, то после пересылки блока новый блок начинает пересылку следующего, и для этого не требуется срабатывание нового триггера. Такая пересылка может быть остановлена сбросом DMAEN бита либо прерыванием NMI при установленном ENNMI.

Для режима DMALEVEL = 1 триггер определен по уровню сигнала. Для правильной работы этого режима источником сигнала триггера должен быть выбран внешний источник DMAE0. Пересылка активна все время, пока сигнал триггера остается высоким и DMAEN = 1. В этом режиме сигнал триггера должен оставаться высоким на все время пересылки. Если он станет низким для блочной или импульсно-блочной пересылки, контроллер DMA будет остановлен в текущем состоянии и продолжит работу при возврате сигнала в высокий уровень. В это время простоя могут быть изменены регистры DMA. Режим рекомендуется использовать для режима пересылок, когда DMAEN автоматически сбрасывается (DMADT = 0 .. 3).

Если триггеры разных каналов срабатывают одновременно, на выполнение ставится пересылка того канала, у которого максимальный приоритет. По умолчанию приоритет каналов DMA0 – DMA1 – DMA2. Если установлен бит ROUNDROBIN, то после завершения пересылки каналу назначается минимальный приоритет (т.е. при постоянном одновременном срабатывании

триггеров обслуживание каналов будет циклическим). Если срабатывает триггер канала с большим приоритетом, текущая пересылка не прерывается.

Для каждого канала выбор сигнала источника для триггера выполняется битами DMAxTSEL, при этом обязательно DMAEN должен быть равен 0. Перечень источников одинаков для каждого из каналов и приведен в таблице:

Таблица 3.3. Источники сигналов триггера для DMA-каналов

№	Источник	Описание и определение флагов в msp430f5529.h
0	DMAREQ	Запрос DMA (программный запуск). Триггер срабатывает при установке бита. Сигнал DMAREQ автоматически сбрасывается после начала пересылки. Определение: DMA0TSEL__DMA_REQ
1	TA0CCR0 CCIFG	Запуск по каналам таймеров. Триггер срабатывает при установке бита. Соответствующий сигнал CCIFG автоматически сбрасывается после начала пересылки. Если установлен соответствующий бит CCIE, выбранный флаг CCIFG не запускает пересылку DMA. Макроопределения: DMA0TSEL__TA0CCR0 .. DMA0TSEL__TB0CCR2
2	TA0CCR2 CCIFG	
3	TA1CCR0 CCIFG	
4	TA1CCR2 CCIFG	
5	TA2CCR0 CCIFG	
6	TA2CCR2 CCIFG	
7	TB0CCR0 CCIFG	
8	TB0CCR2 CCIFG	
16	UCA0RXIFG	Запуск по каналам USCI. Триггер срабатывает при приеме (RX)/готовности к передаче (TX) данных по соответствующему каналу USCI. Сигнал RXIFG/TXIFG автоматически сбрасывается после начала пересылки. Если установлен соответствующий бит RXIE/TXIE, выбранный флаг RXIFG/TXIFG не запускает пересылку DMA. Макроопределения: DMA0TSEL__USCIA0RX .. DMA0TSEL__USCIB1TX
17	UCA0TXIFG	
18	UCB0RXIFG	
19	UCB0TXIFG	
20	UCA1RXIFG	
21	UCA1TXIFG	
22	UCB1RXIFG	
23	UCB1TXIFG	
24	ADC12IFGx	Запуск по АЦП. Триггер срабатывает при установке бита (завершении одно-канального преобразования АЦП или завершении последнего преобразования в последовательности). Программная установка бита не запускает триггер. Все ADC12IFG флаги автоматически сбрасываются, когда к соответствующему ADC12MEMx обратился контроллер DMA. Макроопределение: DMA0TSEL__ADC12IFG
27	USB FNRXD	USB триггер. Макроопределение: DMA0TSEL__USB_FNRXD
28	USB ready	USB триггер. Макроопределение: DMA0TSEL__USB_READY
29	MPY ready	Запуск по умножителю. Триггер срабатывает, когда умножитель готов для нового операнда. Макроопределение: DMA0TSEL__MPY

№	Источник	Описание и определение флагов в msp430f5529.h
30	DMAxIFG	DMA2IFG – для канала 0, DMA0IFG – для канала 1, DMA1IFG – для канала 2. Триггер срабатывает при установке бита. Сигнал DMAxIFG не сбрасывается автоматически. Макроопределения: DMA0TSEL__DMA2IFG, DMA1TSEL__DMA0IFG, DMA2TSEL__DMA1IFG
31	DMAE0	Пересылка по внешнему сигналу триггера. Макроопределение: DMA0TSEL__DMAE0

Контроллер DMA требует 1-2 тактов MCLK для синхронизации перед каждым обменом, потом 2 такта MCLK на пересылку байта либо слова и 1 такт ожидания после пересылки. Таким образом, пересылка займет 4-5 тактов. В случае, если источник MCLK выключен, контроллер DMA временно включает MCLK, генерируемую DCOCLK, для выполнения пересылки. В этом случае дополнительно потребуется еще 5 мкс для запуска DCOCLK.

Пересылки DMA не прерываются системными прерываниями, прерывания ожидают завершения пересылки. Только прерывание NMI может прервать пересылку, если установлен бит ENNMI. Выполнение обработчиков прерываний приостанавливается для DMA пересылки. Чтобы этого не происходило, на время выполнения обработчика прерываний следует отключать DMA контроллер.

Каждый канал DMA имеет собственный флаг DMAIFG. Флаг устанавливается, когда соответствующий DMAxSZ становится равным нулю. Если при этом установлены флаги DMAIE и GIE, возникает запрос на прерывание.

Состав регистров контроллера DMA и назначение отдельных полей приведены в таблицах 3.4 и 3.5. Другие флаги для выбора источников триггеров указаны в табл. 3.3.

Таблица 3.4. Регистры контроллера DMA

Регистр	Адрес	Назначение
DMACTL0	0500h	Общий регистр управления DMA
DMACTL1	0502h	Общий регистр управления DMA
DMACTL4	0508h	Общий регистр управления DMA
DMAIV	050Eh	Вектор прерываний DMA
DMA0CTL	0510h	Управление каналом 0 DMA
DMA0SA	0512h	Адрес источника канала 0 DMA
DMA0DA	0516h	Адрес приемника канала 0 DMA
DMA0SZ	051Ah	Размер пересылки канала 0 DMA
DMA1CTL - DMA1SZ	0520h - 052Ah	Аналогичные регистры канала 1 DMA
DMA2CTL-DMA2SZ	0530h - 053Ah	Аналогичные регистры канала 2 DMA

Таблица 3.5 Поля регистров контроллера DMA

Регистр	Биты	Поле	Назначение	Определение флагов в msp430f5529.h
DMACTL0	8-12	DMA1TSEL	Выбор источника триггера канала 1 DMA	DMA1TSEL_0.. DMA1TSEL_31
	0-4	DMA0TSEL	Выбор источника триггера канала 1 DMA	DMA0TSEL_0.. DMA0TSEL_31
DM	4-0	DMA2TSEL	Выбор источника триггера канала 1 DMA	DMA2TSEL_0.. DMA2TSEL_31
DMACTL4	2	DMARMWDIS	Запрет прерывания цикла операции процессора на шине (чтение/изменение/ запись).Пересылка DMA ожидает завершения операции процессора. Если бит не установлен, DMA пересылка может прерывать операцию процессора.	DMARMWDIS
	1	ROUNDROBIN	Установка циклического приоритета каналов	ROUNDROBIN
	0	ENNMI	Разрешение прерывания DMAпересылки посредством NMI	ENNMI
DMAxCTL	12-14	DMADT	Режим пересылки	DMADT_0 .. DMADT_7
	10-11	DMADSTINCR	Инкремент адреса назначения после пересылки (при пересылке слов адрес изменяется на 2): 00,01 — без изменений, 10 — декремент, 11 - инкремент	DMADSTINCR_0.. DMADSTINCR_3
	8-9	DMASRCINCR	Инкремент адреса источника после пересылки (при пересылке слов адрес изменяется на 2): 00,01 — без изменений, 10 — декремент, 11 - инкремент	DMASRCINCR_0.. DMASRCINCR_3
	7	DMADSTBYTE	Размер данных приемника: 0 - слово, 1 - байт	DMASRCBYTE, DMADSTBYTE, DMASWDW, DMASBDW, DMASWDB, DMASBDB
	6	DMASRCBYTE	Размер данных источника: 0 - слово, 1 - байт	

Регистр	Биты	Поле	Назначение	Определение флагов в msp430f5529.h
	5	DMALEVEL	Режим срабатывания триггера: 0 — по переднему фронту, 1 — по высокому уровню	DMALEVEL
	4	DMAEN	Разрешение DMA (=1)	DMAEN
	3	DMAIFG	Флаг прерывания DMA	DMAIFG
	2	DMAIE	Разрешение прерывания DMA	DMAIE
	1	DMAABORT	Флаг, устанавливается, если пересылка DMA была прервана NMI	DMAABORT
	0	DMAREQ	Запрос DMA. Программно-управляемый за-пуск пересылки. Сбрасывается автоматически	DMAREQ
DMAxSA	0-19	DMAxSA	Адрес источника. Обращение к регистру требует расширенных операций. Использо-вание операций для слов очищает регистр	DMA0SA.. DMA2SA
DMAxDA	0-19	DMAxDA	Адрес назначения. Обращение к регистру требует расширенных операций. Использование операций для слов очищает регистр	DMA0DA.. DMA2DA
DMA	0-15	DMAxSZ	Количество передаваемых данных (байт или слов)	DMA0SZ.. DMA2SZ
DMAIV	0-15	DMAIV	Вектор прерываний	DMAIV_NONE, DMAIV_DMA0IFG.. DMAIV_DMA2IFG

4 ВЫПОЛНЕНИЕ РАБОТЫ

4.1 Описание программы

В бесконечном цикле программа ожидает в режиме LPM0 прерывания DMA, которое заносит значения циклов из счетчика TA2 в буфер. DMA

работает в режиме одиночной пересылки. После прерывания микроконтроллер выходит из режима LPM0, проверяет состояния кнопок PAD2 и кнопки S1. Если была нажата кнопка S1 значения буфера заносятся в файл «buffer.bin».

При нажатия сенсорной кнопки PAD2 значения буфера считываются из файла и выводятся на экран. Вывод измеряемых значений прекращается. При повторном нажатии на экран режим отображения заканчивается и экран очищается. После этого снова запускается измерение значений датчика температуры, подключенного ко внутреннему АЦП, и DMA.

Для взаимодействия с экраном используется библиотека «HAL_Dogs102x6», для работы с файловой системой «FatFS». Для измерения значений датчика температуры используется стандартный HAL АЦП.

4.2 Листинг кода

```
#include <msp430.h>
#include "HAL_Dogs102x6.h"
#include "HAL_Cma3000.h"
#include "ff.h"
#include "structure.h"
#include "CTS_Layer.h"
#include <stdlib.h>
#include <math.h>
/* Функция для взаимодействия с GPIO */
#define GPIO_DIR_INPUT(...) GPIO_DIR_INPUT_SUB(__VA_ARGS__)
#define GPIO_DIR_INPUT_SUB(port, pin) (P##port##DIR &= ~(1 << (pin)))
#define GPIO_PULLUP(...) GPIO_PULLUP_SUB(__VA_ARGS__)
#define GPIO_PULLUP_SUB(port, pin) P##port##REN |= (1 << (pin)); \
P##port##OUT |= (1 << (pin))
#define GPIO_READ_PIN(...) GPIO_READ_PIN_SUB(__VA_ARGS__)
#define GPIO_READ_PIN_SUB(port, pin) ((P##port##IN & (1 << (pin))) ? 1 : 0)
#define GPIO_WRITE_PIN(...) GPIO_WRITE_PIN_SUB(__VA_ARGS__)
#define GPIO_WRITE_PIN_SUB(port, pin, value) (P##port##OUT = (P##port##OUT & ~(1 <<
(pin))) | (value << (pin)))
//описание свойств PAD2
const struct Element PAD2 =
{ //CB0
.inputBits = CBIMSEL_1,
.maxResponse = 250,
.threshold = 125
};
//структура описания PAD2 для библиотеки CTS_Layer.h
const struct Sensor Sensor1 =
{
.halDefinition = RO_COMPB_TA1_TA0,
.numElements = 1,
.baseOffset = 1,
.cbpdBits = 0x0001, //CB1 ????
.arrayPtr[0] = &PAD2,
.cboutTAxDirRegister = (uint8_t*)&P1DIR,
.cboutTAxSelRegister = (uint8_t*)&P1SEL,
.cboutTAxBits = BIT6, // P1.6
// информация таймера
.measGateSource = TIMER_ACLK,
.sourceScale = TIMER_SOURCE_DIV_0,
/* 50 ACLK/1 циклов or 50*1/32Khz = 1.5ms */
.accumulationCycles = 50
```

```
};
//кнопка S1
#define S1_PORT 1
#define S1_PIN 7
// данные и позиции для рисования дополнительной информации
#define DRAW_TEXT_ROW 7
#define LINE_Y 45
// имя файла для хранения буфера
#define FILE_NAME "buffer.bin"
//буфер
#define BUFFER_SIZE 80
#define BUFFER_COUNT (BUFFER_SIZE / 2)
volatile uint16_t buffer[BUFFER_COUNT];
volatile uint8_t index = 0;
//отслеживание состояния нажатия кнопок и файла
uint8_t first_press_PAD = 0;
uint8_t no_press_PAD = 0;
uint8_t first_press_S1 = 0;
uint8_t no_press_S1 = 0;
uint8_t file_draw = 0;
UINT bw = 0;
//значение оси Z акселерометра
uint16_t termo_sensor_value = 0;
//получаем данные термодатчика
void read_termo_sensor()
{
	while (ADC12CTL0 & ADC12BUSY);
	ADC12CTL0 |= ADC12ENC | ADC12SC;
	int val_TERM = ADC12MEM0;
	termo_sensor_value = (uint16_t)(val_TERM / 17.7);
}



void setupADC()
{
	// Turn on ADC and enable multiple conversions
	ADC12CTL0 = ADC12SHT0_2 | ADC12ON | ADC12MSC | ADC12REFON | ADC12REF2_5V;
	// Sampling timer, single sequence
	ADC12CTL1 |= ADC12SHP | ADC12CONSEQ_1;
	//ADC12CTL1 |= ADC12SSEL0 | ADC12SSEL1; //adc clock select (SMCLK)
	ADC12CTL2 &= ~ADC12SR; //set sample rate buffer to 200ksps
	// 12-bit conversion
	ADC12CTL2 |= ADC12RES_2;
	// Enable ADC interrupt on MEM1
	ADC12IE |= ADC12IE1;
	// A10 - termal
	ADC12MCTL0 |= ADC12INCH_10 | SELREF_0 | ADC12EOS;
}



// вычисления значения для рисования
uint16_t get_draw_value(uint8_t index)
{
	uint16_t draw_value = (uint16_t)((float)45 * (float)buffer[index] / 180);
	return draw_value;
}
void draw()
{
	uint16_t val = buffer[index];
```

```
	char str[40] = "";
	int8_t i = 1;
	do {
		str[i++] = (char)(val % 10 + '0');
		val = val / 10;
	} while (val > 0 && index < 40);
	uint8_t j = 0;
	for (i; i >= j; i--) {
		char temp = str[i];
		str[i] = str[j];
		str[j] = temp;
		j++;
	}
	str[0] = 'T';
	Dogs102x6_clearRow(6);
	Dogs102x6_stringDraw(DRAW_TEXT_ROW - 1, 0, str, DOGS102x6_DRAW_NORMAL);
}
void draw_from_file(FIL file)
{
	no_press_PAD = 1;
	first_press_PAD = 0;
	file_draw ^= 1;
	if (file_draw)
	{
		f_open(&file, FILE_NAME, FA_READ);
		f_read(&file, buffer, BUFFER_SIZE, &bw);
		char str[40] = "Loaded from file";
		Dogs102x6_clearScreen();
		Dogs102x6_horizontalLineDraw(0, DOGS102x6_X_SIZE - 1, LINE_Y,
DOGS102x6_DRAW_NORMAL);
		Dogs102x6_stringDraw(DRAW_TEXT_ROW, 0, str, DOGS102x6_DRAW_NORMAL);
		f_close(&file);
		uint16_t i = 0;
		for (i = 0; i < BUFFER_COUNT; i++)
		{
			uint16_t draw_value = get_draw_value(i);
			Dogs102x6_pixelDraw(i * 2, 45 - draw_value, DOGS102x6_DRAW_NORMAL);
			Dogs102x6_pixelDraw(i * 2 + 1, 45 - draw_value,
DOGS102x6_DRAW_NORMAL);
		}
	}
	else {
		Dogs102x6_clearScreen();
		Dogs102x6_horizontalLineDraw(0, DOGS102x6_X_SIZE - 1, LINE_Y,
DOGS102x6_DRAW_NORMAL);
	}
}
void write_to_file(FIL file)
{
	if (first_press_S1 == 0)
		first_press_S1 = 1;
	else if (first_press_S1 == 1)
	{
		no_press_S1 = 1;
		first_press_S1 = 0;
		f_open(&file, FILE_NAME, FA_WRITE | FA_CREATE_ALWAYS);
		f_write(&file, buffer, BUFFER_SIZE, &bw);
		f_close(&file);
		char str[40] = "Saved to file";
		Dogs102x6_clearScreen();
```

```
            Dogs102x6_horizontalLineDraw(0, DOGS102x6_X_SIZE - 1, LINE_Y,
DOGS102x6_DRAW_NORMAL);
            Dogs102x6_stringDraw(DRAW_TEXT_ROW, 0, str, DOGS102x6_DRAW_NORMAL);
      }
}
uint16_t main(void)
{
      //остановка сторожевого таймера
      WDTCTL = WDTPW + WDTHOLD;
      //инициализация S1
      GPIO_DIR_INPUT(S1_PORT, S1_PIN);
      GPIO_PULLUP(S1_PORT, S1_PIN);
      //настройка АЦП
      setupADC();
      //инициализация экрана
      Dogs102x6_init();
      Dogs102x6_backlightInit();
      Dogs102x6_setBacklight(255);
      Dogs102x6_clearScreen();
      Dogs102x6_horizontalLineDraw(0, DOGS102x6_X_SIZE - 1, LINE_Y,
DOGS102x6_DRAW_NORMAL);
      // Инициализация базовых значения для сенсорных кнопок
      TI_CAPT_Init_Baseline(&Sensor1);
      TI_CAPT_Update_Baseline(&Sensor1, 5);
      //инициализация акселерометра
      Cma3000_init();
      FATFS fs; //файловая система
      FIL file; //файл для буфера
      // монтирование диска
      FRESULT res = f_mount(0, &fs);
      if (res == FR_NO_FILESYSTEM) {
            f_mkfs(0, 0, 512);
      }
      //инициализация DMA
      DMACTL0 = DMA0TSEL_5; // пересылка по TA2CCR0.IFG
      // одиночная пересылка, включение DMA, разрешение прерываний
      // нет инкремента dst, src, размер данных 16 бит.
      DMA0CTL = DMADT_0 + DMAEN + DMAIE;
      // Размер = 1
      DMA0SZ = 1;
      // DMA0SA - источник (ось Z акселерометра)
      __data16_write_addr((unsigned short)&DMA0SA, (unsigned
long)&termo_sensor_value);
      // DMA0DA - назначение (элемент буфера)
      __data16_write_addr((unsigned short)&DMA0DA, (unsigned long)&buffer[index]);
      TA2CCR0 = 200;
      TA2CTL = TIMER_ACLK + TIMER_SOURCE_DIV_0;
      TA2CTL |= (TACLR + MC__UP);
      while (1)
      {
            //вход в режим LPM0 с разрешением прерываний
            __bis_SR_register(LPM0_bits + GIE);
            //рисуем значение на экран
            if (file_draw == 0)
            {
                  draw();
            }
            read_termo_sensor();
            //считываем значения буфера из файл по нажатию PAD2 и выводим на экран
            struct Element* keypressed = 0;
            keypressed = (struct Element*)TI_CAPT_Buttons(&Sensor1);
```

```
            if (keypressed == 0)
            {
                  no_press_PAD = 0;
            }
            if (keypressed && no_press_PAD == 0)
            {
                  if (first_press_PAD == 0)
                  {
                        first_press_PAD = 1;
                  }
                  else if (first_press_PAD == 1)
                  {
                        draw_from_file(file);
                  }
            }
            //заносим значения буфера в файл по нажатию S1
            uint8_t value_S1 = !GPIO_READ_PIN(S1_PORT, S1_PIN);
            if (value_S1 == 0)
            {
                  no_press_S1 = 0;
            }
            if (value_S1 && no_press_S1 == 0)
            {
                  write_to_file(file);
            }
            index++;
            if (index == BUFFER_COUNT)
            {
                  index = 0;
            }
            //обновляем адрес назначения DMA
            __data16_write_addr((unsigned short)&DMA0DA, (unsigned
long)&buffer[index]);
            //запуск DMA
            DMA0CTL |= DMAEN;
            //запуск Таймера 2
            TA2CTL |= (TACLR + MC__UP);
      }
}
#pragma vector=DMA_VECTOR
__interrupt void DMA_ISR(void)
{
      switch (__even_in_range(DMAIV, 16))
      {
            //прерывание DMA0IFG
      case 2:
            TA2CTL &= ~MC__UP; //остановка таймера
            _bic_SR_register_on_exit(LPM0_bits); // выход из LPM0
            break;
      default: break;
      }
}
```

ВЫВОД

В ходе лабораторной работы были изучены принципы организации прямого доступа к памяти на базе микроконтроллера MSP430F5529 и работы

с SD-картой на основе экспериментальной платы MSP-EXP430F5529 и получены навыки комплексного использования периферийных устройств микроконтроллера MSP430F5529 и устройств экспериментальной платы MSP-EXP430F5529. В результате выполнения работы была написана программа, делающая измерения с значений датчика температуры, подключенного к АЦП, с использованием прямого доступа к памяти и выводящая значения, сохраненные в файл, на ЖКИ в виде графика.