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: Проверил:

Шеменков В.В

ст. гр. 950503

Зарубо Д. Ю

Ященко В.П

Минск 2022

# 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.

# 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

# Теоретические сведения

* 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;
  + */СTS\_HAL.h* – функции ядра библиотеки, поддержка методов измерения RO, fRO, RC, установка прерываний таймеров;
  + */СTS\_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**

ММС/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: ММС и SPI. Протокол ММС 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.

Ра

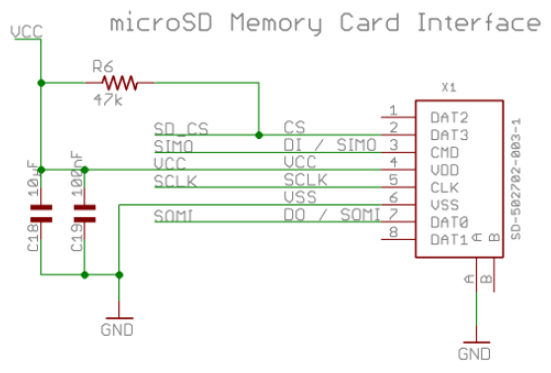


Рис. 3.1 Схема подключения разъема SD-карты памяти

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 |
| CMD | SIMO | SIMO данные (запись в память) | P4.1/ PM\_UСB1SIMO/ PM\_UСB1SDA | PM\_UСB1SIMO |
| CLK | SCLK | Синхросигнал | P4.3/ PM\_UСB1CLK/ PM\_UСA1STE | PM\_UСB1CLK |
| DAT0 | SOMI | SOMI данные (чтение из памяти) | P4.2/ PM\_UСB1SOMI/ PM\_UСB1SCL | PM\_UСB1SOMI |

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].

ММС/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...Nбайт, 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 0хFE (стартовый байт блока данных). Любой иной байт (отличный от 0хFF), полученный в этот момент, будет означать ошибку.

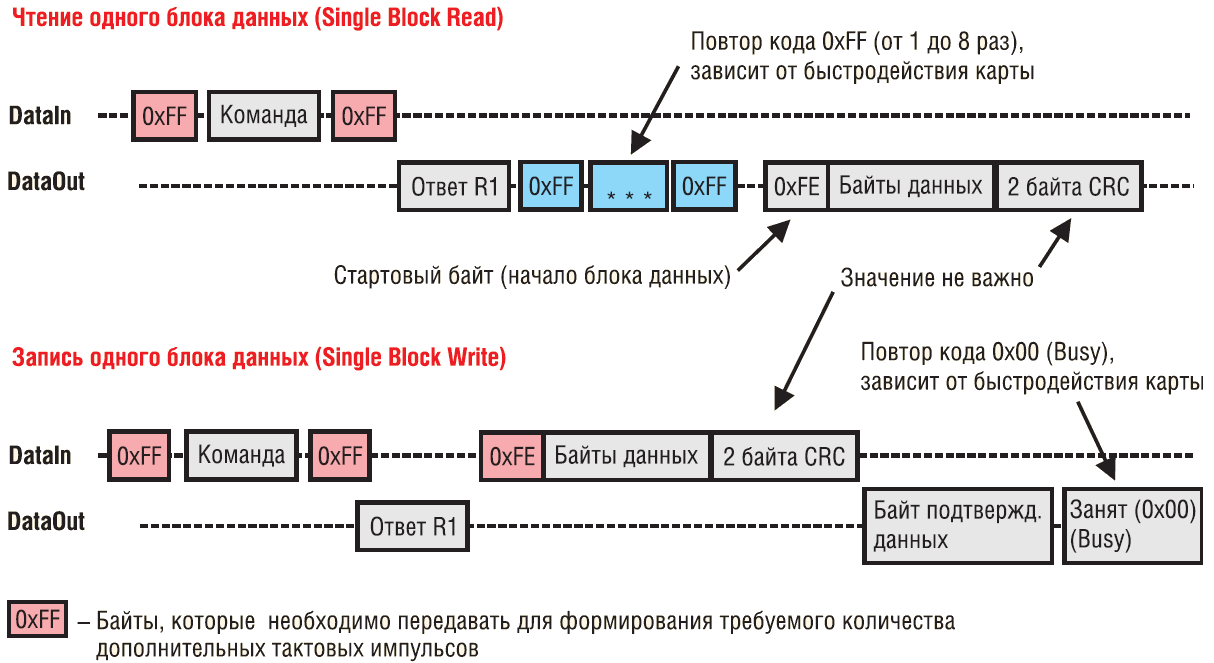


Рис. 3.2 Формат обмена с MMC/SD-картой памяти

Все команды, воспринимаемые ММС/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.

ММС/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 ММС/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 ММС/SD карта находится в режиме ММС, 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 ММС/SDC карту (CS = 0);
* послать команду CMD0 (сброс): 0х40, 0, 0, 0, 0, 0х95 (в этой команде контрольная сумма должна иметь реальное значение (0х95), т.к. данная команда посылается в тот момент, когда ММС/SD карта находится в режиме ММС, а не SPI);
* дождаться правильного ответа 0x01;
* в цикле посылать команду CMD1 (инициализация) и ждать, когда будет получен ответ 0х00 (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 ММС/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.

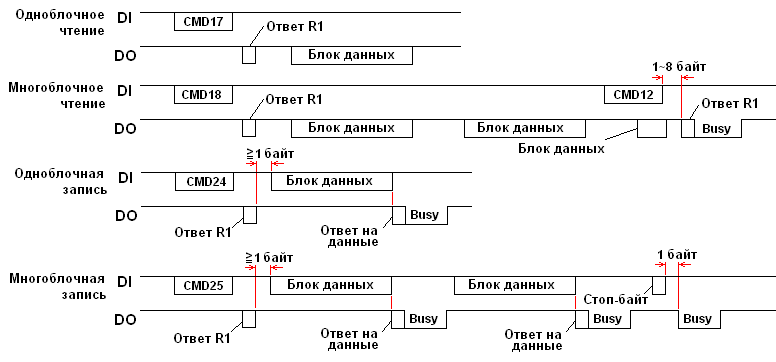


Рис. 3.3 Схема одноблочного и многоблочного чтения/записи

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 ММС/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 – 002ВFFh, сектор 1 — 002С00h – 0033FFh, сектор 2 — 003400h - 003BFFh, сектор 3 — 003С00h – 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.

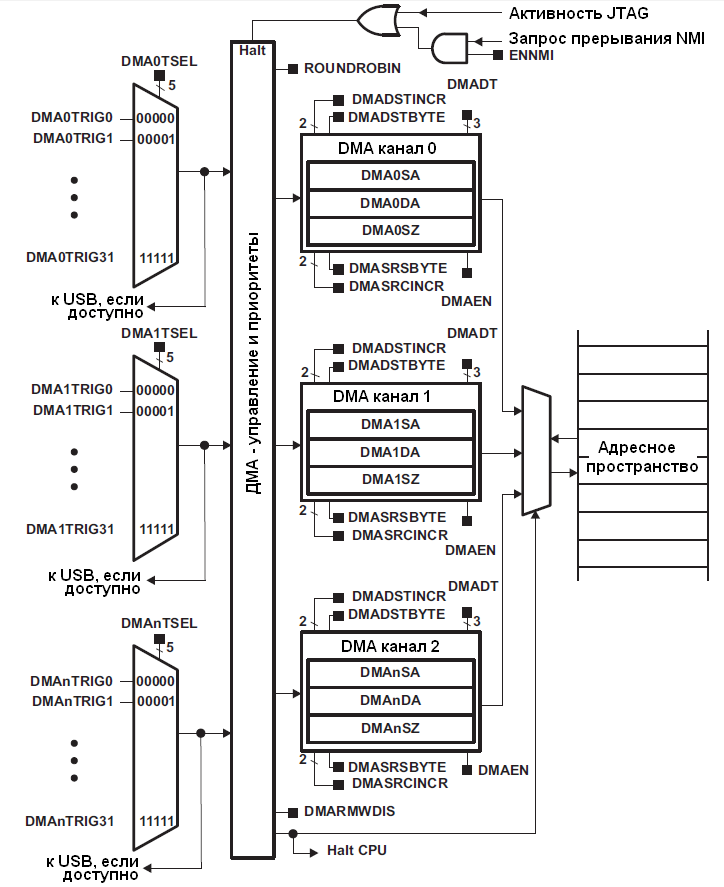


Рис. 3.4 Структура контроллера DMA

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.

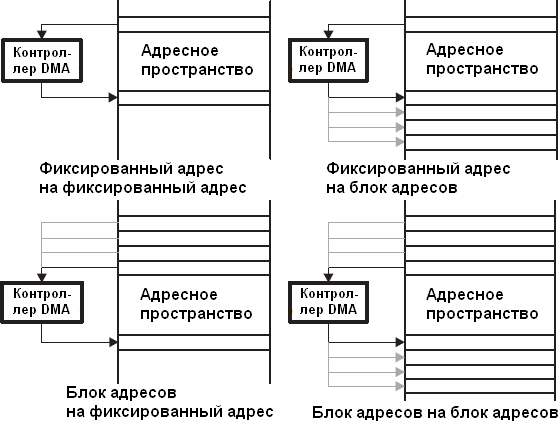


Рис. 3.5 Режимы адресации данных в DMA

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.

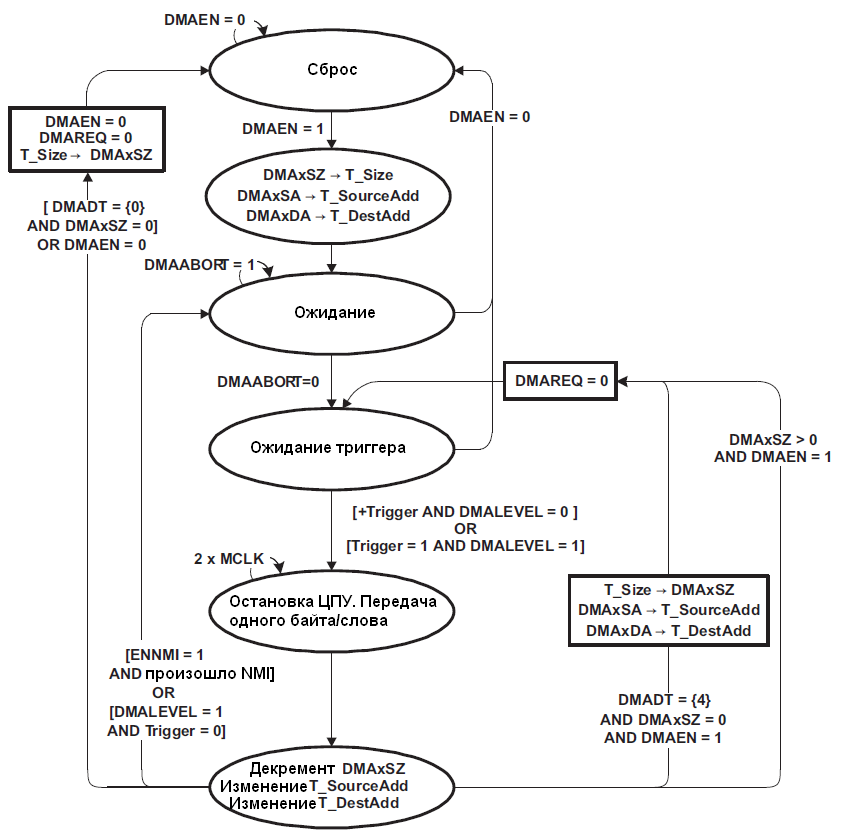


Рис. 3.6 Диаграмма состояний в режиме одиночных пересылок

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 х 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 |
| 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 |
| DMACTL1 | 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 |
| 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 |
| DMAxSZ | 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 LPMO 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.