БГУИР

Кафедра ЭВМ

Отчет по лабораторной работе № 6
Тема: «Высокоуровневые библиотеки SD-карта. Прямой доступ к памяти»
Вариант №8

Выполнил:
студент группы 150503 Ходосевич М.А.

Проверил:
ассистент каф. ЭВМ _____ Шеменков В.В.

Минск
2024

1. Постановка задачи

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.

В соответствии с вариантом написать программу организующую обмен с FLASH-памятью, SD-картой с применением прямого доступа к памяти и с использованием высокоуровневых библиотек. Измеряемый сигнал – PAD2, кнопка для чтения – ось акселерометра X, частота снятия данных – 100, размер буфера – 50, режим отображения – динамически с каждым прочитанным значением, сохранение в файл – по кнопке S2.

2. Теоретические сведения

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.

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 2.1. 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.

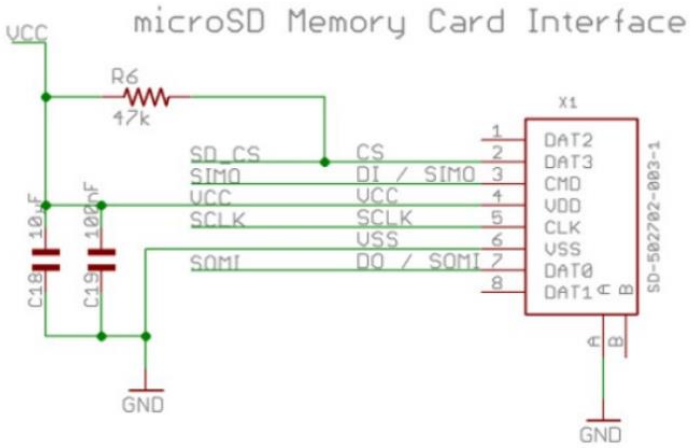

Рисунок 2.1 – Схема подключения разъема SD-карты памяти

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.

Соответствие выводов разъема SD-карты и микроконтроллера приведены в таблице на рисунке 2.2.

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_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

Рисунок 2.2 - Таблица соответствия выводов разъема SD-карты памяти

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.

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.

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.

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.

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.

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.

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.

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. 6.5). 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.

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.

3. Выполнение работы

Программа заносит получаемые с заданной частотой 100 от PAD5 данные в память, используя прямой доступ к памяти. Данные отображаются в виде графика на экране в зависимости от заданного режима отображения и сохраняются в файл на SD-карту. При нажатии указанной кнопки сохраненные данные считываются из файла и выводятся на экран в виде графика, при ее повторном нажатии происходит возврат в режим измерения.

Данные считываются с PAD5 и на экран выводится график, при нажатии на кнопку S2 данные записываются в файл и сохраняются на SD-карту. При нажатии на вторую кнопку данные начинают считываться из файла и выводятся на экран, при повторном нажатии снова производится измерения сигнала на кнопке.

4. Листинг программы

```
#include <msp430.h>
#include "HAL_Dogs102x6.h"
#include "ff.h"
#include "structure.h"
#include "CTS_Layer.h"
#include <stdlib.h>
/* Функция для взаимодействия с GPIO */
#define GPIO_DIR_OUTPUT(...) GPIO_DIR_OUTPUT_SUB(__VA_ARGS__)
#define GPIO_DIR_OUTPUT_SUB(port, pin) (P##port##DIR |= (1 << (pin)))
#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_PULLDOWN(...) GPIO_PULLDOWN_SUB(__VA_ARGS__)
#define GPIO_PULLDOWN_SUB(port, pin) P##port##REN |= (1 << (pin)); \
P##port##OUT &= ~(1 << (pin))
#define GPIO_NOPULL(...) GPIO_NOPULL_SUB(__VA_ARGS__)
#define GPIO_NOPULL_SUB(port, pin) (P##port##REN &= ~(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)))
#define GPIO_TOGGLE_PIN(...) GPIO_TOGGLE_PIN_SUB(__VA_ARGS__)
#define GPIO_TOGGLE_PIN_SUB(port, pin) (P##port##OUT ^= (1 << (pin)))
#define GPIO_TRIG_EDGE_FALLING(...) GPIO_TRIG_EDGE_FALLING_SUB(__VA_ARGS__)
#define GPIO_TRIG_EDGE_FALLING_SUB(port, pin) (P##port##IES |= (1 << (pin)))
#define GPIO_TRIG_EDGE_RISING(...) GPIO_TRIG_EDGE_RISING_SUB(__VA_ARGS__)
#define GPIO_TRIG_EDGE_RISING_SUB(port, pin) (P##port##IES &= ~(1 << (pin)))
#define GPIO_INTERRUPT_ENABLE(...) GPIO_INTERRUPT_ENABLE_SUB(__VA_ARGS__)
#define GPIO_INTERRUPT_ENABLE_SUB(port, pin) P##port##IFG &= ~(1 << (pin)); \
P##port##IE |= (1 << (pin))
#define GPIO_INTERRUPT_DISABLE(...) GPIO_INTERRUPT_DISABLE_SUB(__VA_ARGS__)
#define GPIO_INTERRUPT_DISABLE_SUB(port, pin) (P##port##IE &= ~(1 << (pin)))
#define GPIO_PERIPHERAL(...) GPIO_PERIPHERAL_SUB(__VA_ARGS__)
#define GPIO_PERIPHERAL_SUB(port, pin) (P##port##SEL |= (1 << (pin)))
#define GPIO_CLEAR_IT_FLAG(...) GPIO_CLEAR_IT_FLAG_SUB(__VA_ARGS__)
#define GPIO_CLEAR_IT_FLAG_SUB(port, pin) (P##port##IFG &= ~(1 << (pin)))
/*****************************************************************/
// Описание свойств PAD1
const struct Element PAD1 = { //CB0
            .inputBits = CBIMSEL_0, .maxResponse = 250, .threshold = 125 };
// Описание свойств PAD5
const struct Element PAD5 = { //CB4
            .inputBits = CBIMSEL_4, .maxResponse = 1900, .threshold = 475 };
// Структура описания PAD1 для библиотеки CTS_Layer.h
const struct Sensor Sensor1 = { .halDefinition = RO_COMPB_TA1_TA0,
            .numElements = 1, .baseOffset = 0,
            .cbpdBits = 0x0001, //CB0
            .arrayPtr[0] = &PAD1, .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 };
// Структура описания PAD5 для библиотеки CTS_Layer.h
const struct Sensor Sensor5 = { .halDefinition = RO_COMPB_TA1_TA0,
            .numElements = 1, .baseOffset = 4,
            .cbpdBits = 0x0010, //CB4
            .arrayPtr[0] = &PAD5, .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 272*1/32Khz = 8.5ms */
            .accumulationCycles = 272 };
/*************************************************/
#define LED_ON 1
#define LED_OFF 0
/* Порты и пины используемых выводов */
#define LED1_PORT 1
#define LED1_PIN 0
#define BUTT2_PORT 2
#define BUTT2_PIN 2
// Данные и позиции для рисования дополнительной информации
#define DRAW_TEXT_ROW 7
#define INFO_TEXT "PRESSED UNPRESSED"
#define LINE_Y 53
// Имя файла для хранения буфера
#define FILE_NAME "buffer.bin"
```

```
// Буфер
#define BUFFER_SIZE 50
#define BUFFER_COUNT (BUFFER_SIZE / 2)
volatile uint16_t buffer[BUFFER_COUNT];
volatile uint8_t index = 0;
// Кастомизированная функция для измерения PAD5 из CTS_HAL
void TI_CTS_RO_COMPB_TA1_TA0_HAL_CUSTOM(const struct Sensor *group,
            uint8_t index) {
// Конфигурируем пин CBOUT.
      *(group->cboutTAxDirRegister) |= (group->cboutTAxBits);
      *(group->cboutTAxSelRegister) |= (group->cboutTAxBits);
// Источник опорного напряжения Vcc,
// Vcc*(0x18+1)/32 for CBOUT = 1 and Vcc*((0x04+1)/32 for CBOUT = 0 (Такие
значения используются в CTS_HAL)
      CBCTL2 = CBRS_1 + CBREF14 + CBREF13 + CBREF02;
// Отключает входной буфер пинов
      CBCTL3 |= (group->cbpdBits);
// TimerA2 используется для задания времени измерения
      TA2CCR0 = group->accumulationCycles;
      TA2CTL = group->measGateSource + group->sourceScale;
// Включаем компаратор
      CBCTL1 = CBON;
// Включаем вход определенный вход компаратора
      CBCTL0 = CBIMEN + (group->arrayPtr[index])->inputBits;
// Таймер A1 используется для измерения релаксационных циклов сенсора,
которые
// Источник TACLK, непрерывный счет.
      TA1CTL = TASSEL__TACLK + MC__CONTINUOUS + TACLR;
// Сбрасываем флаг прерывания
      TA1CTL &= ~TAIFG;
// Запуск Таймера 2
      TA2CTL |= (TACLR + MC__UP);
}
// Вычисления значения для рисования
uint16_t Get_Draw_Value(uint8_t index) {
      uint16_t data_range = PAD5.maxResponse - PAD5.threshold;
      int16_t value = buffer[index] - PAD5.threshold;
      if (value < 0) {
            value = 0;
      }
      uint16_t draw_value = (uint16_t) ((float) DOGS102x6_X_SIZE * (float)
value
                  / (float) data_range);
      return draw_value;
}
uint16_t main(void) {
// Остановка сторожевого таймера
      WDTCTL = WDTPW + WDTHOLD;
      FATFS fs;
      FIL file;
      GPIO_DIR_OUTPUT(LED1_PORT, LED1_PIN);
      GPIO_WRITE_PIN(LED1_PORT, LED1_PIN, LED_OFF);
      GPIO_DIR_INPUT(BUTT2_PORT, BUTT2_PIN);
      GPIO_PULLUP(BUTT2_PORT, BUTT2_PIN);
// Инициализация экрана
      Dogs102x6_init();
      Dogs102x6_backlightInit();
      Dogs102x6_setBacklight(255);
      Dogs102x6_clearScreen();
      Dogs102x6_horizontalLineDraw(0, DOGS102x6_X_SIZE - 1, LINE_Y,
                  DOGS102x6_DRAW_NORMAL);
// Dogs102x6_stringDraw(DRAW_TEXT_ROW, 0, INFO_TEXT, DOGS102x6_DRAW_NORMAL);
// Инициализация базовых значения для сенсорных кнопок
```

```
      TI_CAPT_Init_Baseline(&Sensor1);
      TI_CAPT_Update_Baseline(&Sensor1, 5);
      TI_CAPT_Init_Baseline(&Sensor5);
      TI_CAPT_Update_Baseline(&Sensor5, 5);
// Монтирование диска
      FRESULT res = f_mount(0, &fs);
      if (res == FR_NO_FILESYSTEM) {
            f_mkfs(0, 0, 512);
      }
// Работа DMA по прерыванию TA2CCR0 CCIFG(из даташита)
      DMACTL0 = DMA0TSEL_5;
// Одиночная пересылка, включение DMA, разрешение прерываний
// нет инкремента dst, src, размер данных 16 бит.
      DMA0CTL = DMADT_0 + DMAEN + DMAIE;
// Размер = 1
      DMA0SZ = 1;
// Источник - значение счетчика TA1.
      __data16_write_addr((unsigned short) &DMA0SA, (unsigned long) &TA1R);
// Назначение - в элемент массива.
      __data16_write_addr((unsigned short) &DMA0DA,
                  (unsigned long) &buffer[index]);
      TI_CTS_RO_COMPB_TA1_TA0_HAL_CUSTOM(&Sensor5, 0);
      uint8_t first_press_PAD = 0;
      uint8_t no_press_PAD = 0;
      uint8_t first_press_S2 = 0;
      uint8_t no_press_S2 = 0;
      uint8_t file_draw = 0;
      UINT bw = 0;
      while (1) {
// Вход в режим LPMO с разрешением прерываний
            __bis_SR_register(LPM0_bits+GIE);
// Рисуем значение на экран
            if (file_draw == 0) {
                  uint16_t draw_value = Get_Draw_Value(index);
                  uint8_t x_pos = index * 2;
                  if (x_pos % 16 == 0) {
                        if (x_pos == 48) {
                              Dogs102x6_clearRow(x_pos / 8);
                              Dogs102x6_horizontalLineDraw(0,
DOGS102x6_X_SIZE - 1,
                                                LINE_Y, DOGS102x6_DRAW_NORMAL);
                        } else {
                              Dogs102x6_clearRow(x_pos / 8);
                              Dogs102x6_clearRow((x_pos / 8 + 1) % 8);
                        }
                  }
                  Dogs102x6_pixelDraw(draw_value, x_pos,
DOGS102x6_DRAW_NORMAL);
                  Dogs102x6_pixelDraw(draw_value, x_pos + 1,
DOGS102x6_DRAW_NORMAL);
                  uint8_t string[128];
                  uint16_t val = buffer[index];
                  uint8_t i = 0;
                  while (val > 0) {
                        string[i++] = val % 10 + '0';
                        val /= 10;
                  }
                  string[i] = '\0';
                  i--;
                  uint8_t j = 0;
                  for (i; i >= j; i--) {
                        uint8_t temp = string[i];
                        string[i] = string[j];
```

```
                        string[j] = temp;
                        j++;
                  }
// itoa(10, string, 127);
                  Dogs102x6_stringDraw(DRAW_TEXT_ROW, 0, string,
                              DOGS102x6_DRAW_NORMAL);
            }
// Считываем значения буфера из файл по нажатию PAD1 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) {
                        GPIO_WRITE_PIN(LED1_PORT, LED1_PIN, LED_ON);
                        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);
                              Dogs102x6_clearScreen();
                              Dogs102x6_horizontalLineDraw(0,
DOGS102x6_X_SIZE - 1,
                                          LINE_Y, DOGS102x6_DRAW_NORMAL);
                              Dogs102x6_stringDraw(DRAW_TEXT_ROW, 0,
INFO_TEXT,
                                          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(draw_value, 2 * i,
                                                DOGS102x6_DRAW_NORMAL);
                                    Dogs102x6_pixelDraw(draw_value, 2 * i +
1,
                                                DOGS102x6_DRAW_NORMAL);
                              }
                        } else {
                              Dogs102x6_clearScreen();
                              Dogs102x6_horizontalLineDraw(0,
DOGS102x6_X_SIZE - 1,
                                          LINE_Y, DOGS102x6_DRAW_NORMAL);
                              Dogs102x6_stringDraw(DRAW_TEXT_ROW, 0,
INFO_TEXT,
                                          DOGS102x6_DRAW_NORMAL);
                        }
                        GPIO_WRITE_PIN(LED1_PORT, LED1_PIN, LED_OFF);
                  }
            }
// Заносим значения буфера в файл по нажатию S2
            uint8_t value_S2 = !GPIO_READ_PIN(BUTT2_PORT, BUTT2_PIN);
            if (value_S2 == 0) {
                  no_press_S2 = 0;
            }
            if (value_S2 && no_press_S2 == 0) {
                  if (first_press_S2 == 0) {
                        first_press_S2 = 1;
                  } else if (first_press_S2 == 1) {
                        no_press_S2 = 1;
```

```
                        first_press_S2 = 0;
                        GPIO_WRITE_PIN(LED1_PORT, LED1_PIN, LED_ON);
                        f_open(&file, FILE_NAME, FA_WRITE |
FA_CREATE_ALWAYS);
                        f_write(&file, buffer, BUFFER_SIZE, &bw);
                        f_close(&file);
                        GPIO_WRITE_PIN(LED1_PORT, LED1_PIN, LED_OFF);
                  }
            }
            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;
// Таймер A1 используется для измерения релаксационных циклов сенсора,
которые подключены к TACLK.
// Источник TACLK, непрерывный счет.
            TA1CTL = TASSEL__TACLK + MC__CONTINUOUS + TACLR;
// Сбрасываем флаг прерывания
            TA1CTL &= ~TAIFG;
// 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:
// Остановка таймеров
            TA1CTL &= ~MC__CONTINUOUS;
            TA2CTL &= ~MC__UP;
// Выход из LPM0
            _bic_SR_register_on_exit(LPM0_bits);
            break;
      default:
            break;
      }
}
```

5. Заключение

В ходе лабораторной работы были изученыпринципы организации прямого доступа к памяти на базе микроконтроллера MSP430F5529 и работы с SD-картой на основе экспериментальной платы MSP-EXP430F5529. Были получены навыки комплексного использования периферийных устройств микроконтроллера MSP430F5529 и устройств экспериментальной платы MSP-EXP430F5529. Удалось написать программу в соответствии с выданным вариантом. Программа имеет 2 режима: режим чтения из файла и измерения сигнала на кнопке. По нажатию на кнопку также происходит запись в файл.