Программа для PIC-микроконтроллера и карты памяти MMC с FAT16

Поиск и чтение файла (даже если он состоит из нескольких фрагментов).

Здесь представлен пример программы, который находит файл на отформатированной под FAT16 карте памяти MMC 64Мб.
Наша предыдущая программа могла обращаться к любому участку памяти MMC карты.
Для того чтобы найти файл на отформатированном носителе, нужны: *позиция начала файла,* и, если файл состоит из множества фрагментов, нужно знать п*озицию каждого фрагмента*. Для этого программа должна находить начальные сектора файловой структуры FAT, а также перебирать байты данных и находить нужные.

Следующие функции выполняют вышеперечисленные действия:

Имя функции	Вход	Выход	Описание
`void **Byte_Read**(void)`	`AX, AL, BytN`	`ByteLes`	Считывает 2 байта из определённой позиции в секторе и сохраняет их в переменную ByteLes. Старший (H) и младший (L) байты адреса сектора указываются в AX,AL, позиция - в переменной BytN (0..512)
`void **Find**(const char *word)`	`String, Verz`	`ByteLes`	Ищет 11-байтовую строку - имя файла + расширение (8+3). Сохраняет в переменную ByteLes номер первого кластера файла. Перед вызовом функции необходимо записать начальный адрес области каталога файлов в переменную Verz.
`void **Read**(char H, uns16 L)`	`H, L`	-	Считывает данные с сектора и отправляет их в компьютер. Перед вызовом нужно указать адрес сектора.

Пошаговое описание программы:

Шаг	Описание
1	Инициализация последовательного интерфейса
2	Инициализация SPI-интерфейса и MMC карты
3	2-х байтовое значение в нулевом секторе, позиция 1C6h - это начало записи MBR1 (Master Boot Recorder - главная загрузочная запись). Таблица раздела (1BEh) 1C6h-1BEh=8 - на этом месте в записи раздела находится значение смещения между сектором MBR и первым сектором раздела.
4	В AL передаётся 2-х байтовое значение, теперь AL содержит начальный адрес загрузочной записи раздела.
5	2-х байтовое значение, прочитанное из сектора = начальный адрес загрузочной записи раздела (смещение Eh). Это количество зарезервированных секторов.
6	Начальный адрес загрузочной записи раздела суммируется с количеством зарезервированных секторов и сохраняется в переменную "FAT" - это начальный адрес таблицы FAT.
7	2-х байтовое значение, прочитанное из сектора = начальный адрес загрузочной записи раздела (смещение 16h) Это количество секторов на FAT. Так как у нас есть 2 таблицы FAT, полученное значение умножаем на 2, чтобы узнать общее количество секторов, занимаемых FAT.
8	Расчёт начального адреса записей каталога: начальный адрес FAT-таблицы складывают с количествомвом секторов, которые заняты FAT. Результат сохраняется в переменной "Verz".
9	Далее рассчитывается количество секторов, которые резервируются для записей корневого каталога. Для этого используется формула: (количество записей каталога) * 32 / 512. Из нескольких источников я узнал, что количество записей каталога всегда резервируется для 512 записей. Итак, количество секторов, которые резервируются для записей корневого каталога: 512*32/512=32
10	Рассчитывается адрес первого сектора области данных (Cluster 0 - нулевой кластер). Начальный адрес записей каталога суммируется с количествомвом секторов записей каталога (32). Результат записывается в переменную "Dat". Таким образом находятся все начальные сектора структуры FAT и записываются в переменные "FAT", "Verz", "Dat".
11	Запускается подпрограмма Suche("TEST TXT"). Она ищет первый кластер файла Test.txt. Имя файла всегда должно быть 8 символов в длину. Если оно короче, оно дополняется пробелами. Точка между именем файла и расширением в

	записи каталога не отображается. Искомая строка должна содержать 11 ПРОПИСНЫХ символов. Найденный номер первого кластера записывается в переменную "ByteLes".
12	Здесь начинается цикл, который закончится, когда номер кластера примет значение 0XFFFF. Это означает, что файл не имеет следующих кластеров (конец файла). В цикле выполняется следующее:
13	При первом прохождении переменная "Cluster" получает номер первого кластера файла. При следующих прохождениях переменная "Cluster" получает из "ByteLes" номер следующего кластера, который устанавливается в конце цикла.
14	Номер кластера корректируется, так как две первых позиции в FAT зарезервированы, а в области данных данные начинаются с кластера 0. Таким образом, настоящий номер кластера = номер кластера - 2.
15	Настоящий номер кластера передаётся в переменную "AL".
16	Рассчитывается абсолютный физический адрес сектора: Для этого номер кластера нужно умножить на 2 (так как 1 кластер = 2 сектора). Это происходит сдвигом позиции влево, а именно через флаг переноса. Если при этом происходит переполнение "AL" (если файл на MMC находится в области > 32 Мбайт), перенос попадает в "AH".
17	Полученный номер сектора складывается с адресом первого сектора области данных (кластер 0). Суммирование происходит в режиме ассемблера, так как результат – это 24-битное число, а компилятор cc5x может работать только с 16-битными числами. Таким образом мы получаем абсолютный физический адрес сектора в "AH", "AL", где AH – 8-битная переменная, содержит старшую часть адреса сектора, а AL – 16-битная переменная, содержит младшую часть адреса сектора.
18	Функцией Lesen(AH,AL) читаем из первого сектора кластера файла.
19	Физический адрес сектора увеличиваем на 1 с соблюдением переноса в "AH"
20	Функцией Lesen(AH,AL) читаем из второго сектора кластера файла.
21	Из таблицы FAT считываем следующий номер кластера: Рассчитывается сектор, в котором находится запись FAT. Номер кластера делится на 100h (256), так как в секторе 200h (512) байтов, а размер записи FAT составляет 2 байта (в секторе умещается 100h (256) записей). Результат сохраняется во временную переменную.
22	В AL содержится фактический адрес сектора искомой записи FAT, который складывается из начального адреса FAT и рассчитанного значения во временной переменной.
23	Чтобы получить номер записи FAT в вычисленном секторе, мы должны вычесть из номера кластера количество записей FAT в предыдущих секторах. Так как размер записи FAT составляет 2 байта, то позиция байта = (номер записи FAT в вычисленном секторе)*2.
24	Теперь, когда у нас есть адрес сектора и позиция в секторе, с помощью функции Byte_lessen мы можем считать запись FAT.
25	Теперь "ByteLes" содержит номер следующего кластера файла. Программа продолжается с шага 12.

Ниже приведён код программы на СИ (компилятор CC5x).

```
//*************************************************************************
// Чтение файла из 64 Мб MMC-памяти под FAT16
// Последовательный интерфейс (9600,n,8,1) на 20MHz
// Компилятор CC5x

#include <D:\cc5\16F877.H>
#pragma config |= 0b.11.111101.11.00.10
#pragma bit CS @ PORTC.2                        // Выход для выбора микросхемы CS

uns16 AL, BytN,ByteLes,Claster,FAT,Verz,Dat;    // Объявляем переменные
char AH ;

#pragma origin 100

//*************************************************************************
void InitUSART()
{
    BRGH=1;
    SPBRG=129;      // (9600 бод при входной тактовой частоте 20 МГц)
    SPEN = 1;       // Set_Serial_Pins;
    SYNC = 0;       // Set_Async_Mode;
    TX9 = 0;        // Set_8bit_Tx;
    RX9 = 0;        // Set_8bit_Rx;
    CREN = 1;       // Enable_Rx;
    TXEN = 1;       // Enable_Tx;
    RCIE=0;         // Отключить прерывание Rx
}
```

```
//*****************************************************************************
void SerString(const char *str)   //Передача последовательной строки
{
    char ps;
    ps = *str;                    // Запомнить указатель на начало строки в ps
    while(ps>0)                   // Проверка, не достигнут ли конец строки
    {
        str++;                            // Увеличение указателя для доступа к следующем символу
        if (ps== 0) break;        // Проверка, не достигнут ли конец строки
        while(!TXIF);             // Проверка, пуст ли регистр TXREG
        TXREG =ps ;               // Передача байта данных
        ps = *str;                // Запомнить содержимое указателя в ps
    }
}

//*****************************************************************************
char SPI(char d)           // Передача по интерфейсу SPI
{
    SSPBUF=d;
    while (!BF);          // Ожидание, пока идёт передача
    return SSPBUF;        // Одновременный приём
}

//*****************************************************************************
char Command(char befF,uns16 AdrH,uns16 AdrL,char befH )
{                                 // Передача команды в MMC
    char a;
    for (a=0;a<9;a++)
    {
        Carry=0;
        AdrL = rl(AdrL);
        AdrH = rl(AdrH);
    }
    SPI(0xFF);
    SPI(befF);
    SPI(AdrH.high8);
    SPI(AdrH.low8);
    SPI(AdrL.high8);
    SPI(AdrL.low8);
    SPI(befH);
    SPI(0xFF);
    return SPI(0xFF);             // Возвращаем ответ на команду
}

//*****************************************************************************
bit MMC_Init()
{
    SMP=0;      // Вход действителен в середине синхроимпульса
    CKE=0;      // CKE=1;   // Фронт передачи (Fordere Flanke) (передний)
    CKP=1;      // CKP=0;   // Низкий уровень - пассивное состояние
    SSPM0=1;
    SSPEN=1;       // Включить SPI
    CS=1;          // Отключить MMC
    char i;        // Переменная
    for(i=0; i < 10; i++)SPI(0xFF);
    CS=0;
    if (Command(0x40,0,0,0x95) !=1)goto Fehler ;
    st:
    if (Command(0x41,0,0,0xFF) !=0) goto st ;
    return 1;
    Fehler:
    return 0;
}

//*****************************************************************************
void Byte_lesen (void)
{
    bit zw;
    char a;
    int16 i;
    zw=0;
    Command(0x51,AH,AL,0xFF);
    while(SPI(0xFF) != 0xFE);            // Ожидание 0xFE - начала каждой передачи данных
    for(i=0; i < 512; i++)
    {
        a=SPI(0xFF);
        if (zw)
        {
            ByteLes.high8=a;
            zw=0;
```

```
        }
        if (i==BytN)
        {
            ByteLes.low8=a;
            zw=1;
        }

    }
    SPI(0xFF);                          // В конце 2 незначимых байта
    SPI(0xFF);
}

//*****************************************************************************
void Suche(const char *wort)
{
    uns16 i,L,N;
    bit gef;
    char a,m,b;
    gef=0;
    m=0;
    N=Verz+32;                          // Конец области каталога
    for (L=Verz;L<N;L++)
    {
        Command(0x51,0,L,0xFF);
        while(SPI(0xFF) != 0xFE);
        for(i=0; i < 512; i++)
        {
            a=SPI(0xFF) ;
            b=wort[m];
            if(a==b)
            {
                m++;
                if(!wort[m])
                {
                    SerString("Gefunden ");
                    m=0;
                    BytN=i;
                    AL=L;
                    AH=0;
                    gef=1;
                }
                goto fu;
            }

            m=0;
            fu:
        }
        SPI(0xFF);                      // В конце 2 незначимых байта
        SPI(0xFF);
        if(gef) goto raus;
    }
    raus:
    SerString("Beendet ");
    BytN+=16;
    Byte_lesen();
}

//*****************************************************************************
void Lesen(char H,uns16 L)
{
    uns16 i;

    // 512-байтное чтение
    if (Command(0x51,H,L,0xFF) !=0) SerString("Lese_resp_Fehler");  // "Ошибка"
    while(SPI(0xFF) != 0xFE);           // Ожидание 0xFE - начала каждой передачи данных
    for(i=0; i < 512; i++)
    {
        while(!TXIF);                   // Проверка, пуст ли регистр TXREG
        TXREG =SPI(0xFF);               // Передача байта данных
    }
    SPI(0xFF);                          // В конце 2 незначимых байта
    SPI(0xFF);
}

//*****************************************************************************

void main(void)
{
    uns16 temp;
    INTCON=0;                   // Отключить прерывания
```

```
    ADCON1=0x6;                 // Все выходы PortA - цифровые
    TRISC=0b.1101.0011;         // sck rc3-0, sdo rc5-0, CS rc2-0.
    TRISB=0b.0000.0010;         // RB2>TX, RB1<RX
    InitUSART();                // 1 шаг. Инициализация последовательной передачи данных
    //SerString("Ich bin ON ");  // Стартовое сообщение
    MMC_Init();                 // 2 шаг

    // Определение начала каждой области
    AL=0;                       // 3 шаг
    AH=0;
    BytN=0x1C6;
    Byte_lesen();
    AL=ByteLes;                 // 4 шаг
    BytN=0xE;                   // 5 шаг
    Byte_lesen();
    FAT=ByteLes+AL;             // 6 шаг
    BytN=0x16;                  // 7 шаг
    Byte_lesen();
    ByteLes=ByteLes*2;
    Verz=FAT+ByteLes;           // 8 шаг
    Dat=Verz+32;                // 9, 10 шаги


    Suche("TEST    TXT");       // 11 шаг

    while(ByteLes != 0xFFFF)    // 12, 25 шаги
    {
        Claster=ByteLes;        // 13 шаг
        Claster=Claster-2;      // 14 шаг
        AL=Claster;             // 15 шаг
        Carry=0;                // 16  шаг Здесь учитываются границы переменной (флаг переноса)
        AL = rl(AL);            // Умножение на 2 с переносом
        AH = rl(AH);

        #asm                    // 17 шаг
        MOVF  Dat+1,W
        ADDWF AL+1,1
        MOVF  Dat,W
        ADDWF AL,1
        BTFSS 0x03,Carry
        GOTO  keinCarry
        BCF   0x03,Carry
        INCF  AL+1,1
        BTFSC 0x03,Carry
        INCF  AH,1
        keinCarry:
        #endasm

        Lesen(AH,AL);           // 18 шаг Первый блок кластеров
        Carry=0;                // 19 шаг
        AL++;
        if(Carry)AH++;          // Здесь учитываются границы переменной (флаг переноса)
        Lesen(AH,AL);           // 20 шаг Второй блок кластеров

        AL=FAT;                 // 21 шаг следующий кластер
        temp=ByteLes/0x100;
        AL=AL+temp;             // 22 шаг
        temp=temp*0x100;        // 23 шаг
        BytN=ByteLes-temp;
        BytN=BytN*2;
        Byte_lesen();           // 24 шаг

    }

//*****************************************************************************
```