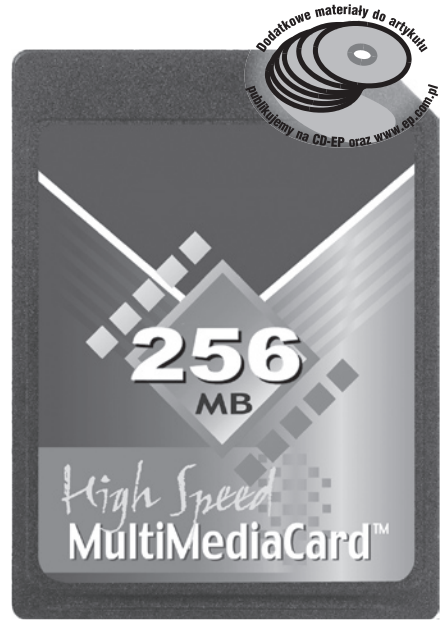
Prosty system plików dla kart MMC/SD



Duże pojemności, niska cena oraz możliwość komunikacji z wykorzystaniem interfejsu SPI powodują, że konstruktorzy systemów mikroprocesorowych coraz częściej interesują się stosowaniem kart pamięci Flash typu MMC i SD do zapamiętywania dużych ilości informacji.

Stosowanie kart MMC i SD wymaga zaimplementowania procedur inicjalizacji i transferu danych. Po pokonaniu tej bariery staniemy przed koniecznością zorganizowania dużej ilości informacji zapisywanej i odczytywanej z karty. W klasycznych pamięciach dołączanych do systemów mikroprocesorowych dostęp do danych (odczytywanie lub zapisywanie) określony był przez ich adresy w pamięci. Adres mógł być wystawiany na liniach równoległej magistrali adresowej mikrokontrolera lub był przesyłany magistralą szeregową I^2C, SPI, itp. Pierwszy sposób adresowania był stosowany między innymi w mikroprocesorach i mikrokontrolerach Intela: 8080, 8085, 8051. Dostęp do danych był szybki, ale magistrala adresowa (i danych) zajmowała większość linii mikrokontrolera. W wielu nowych mikrokontrolerach nie ma możliwości stosowania magistrali równoległej (przykładowo nie stosuje jej Microchip, STM) i kiedy potrzebna jest zewnętrzna pamięć, to stosowane są pamięci z interfejsem szeregowym. Jedne z najpopularniejszych to pamięci typu 24C04 z interfejsem I^2C. Są proste w implementacji, ale mają małą pojemność i ich stosowanie ogranicza się do zapisywania małej ilości danych (na przykład konfiguracyjnych). W przypadku tego typu pamięci musi być wysyłany adres komórki, pod który dana ma być zapisana lub z której ma być odczytana.

Oprócz pamięci z równoległą magistralą adresową stosowane są pamięci o innej filozofii adresowania i przechowywania danych. Wywodzą się z nośników magnetycznych: dyskietek magnetycznych i twardych dysków. Nośniki magnetyczne od początku powstania charakteryzowały się dużymi pojemnościami (z zachowaniem skali rozwoju techniki). Klasyczne zaadresowanie danych na dysku twardym o pojemności 100 MB wymagałoby szerokich magistral, ponadto dostęp do dużej porcji danych (na przykład 1 MB) bajt po bajcie wymagałby wielu cykli dostępu i w rezultacie byłby niezbyt szybki. Wymyślono inny sposób organizacji danych na nośnikach o dużej pojemności. Są one zapisywane w porcjach nazywanych sektorami. Nie można zapisać porcji mniejszej niż sektor lub jego wielokrotność. Również odczytywanie danych odbywa się całymi sektorami, co nie zawsze jest regułą. Dane na nośniku są adresowane przez podanie numeru całego sektora, a nie pojedynczego bajtu lub wielobajtowego słowa. Nawet, jeżeli chcemy zapisać 1 bajt, to trzeba zapisać cały sektor. Podobnie jest z odczytem. Nie stanowi to jednak żadnego problemu, bo do zapisywania pojedynczych bajtów stosowane są inne pamięci (na przykład wspomniana już 24C04), a na nośnikach o dużych pojemnościach zapisuje się duże ilości danych.

W pamięciach Flash MMC i SD podobnie jak w nośnikach magnetycznych dane są zapisywane sektorami o długości 512 bajtów i najczęściej odczytywane blokami o takiej samej długości. W czasie odczytywania możliwość ustawienia długości bloku od 1 bajta do 512 bajtów, ale jeżeli dane są również zapisywane, to ze względów praktycznych blok danych do odczytu ma długość równą wielkości sektora, czyli 512 bajtów.

Załóżmy, że długość bloku jest równa długości sektora, czyli 512 bajtów. Zapisanie danych w pamięci karty polega na wysłaniu 512 bajtów. Tutaj musimy postawić sobie pytanie, jak zaadresować zapisywany sektor? Adresowanie jest proste, ale konieczne jest wyjaśnienie kilku kwestii. Komenda zapisu danych wymaga podania w argumencie adresu fizycznego początku sektora. Sektor zerowy zaczyna się od fizycznego adresu 0, a kończy na fizycznym adresie 511, sektor pierwszy zaczyna się od adresu 512, a kończy na adresie 1023 i tak dalej. Mimo iż karta wymaga adresu fizycznego, to może on być tylko wielokrotnością długości sektora. Gdybyśmy próbowali zapisać sektor pierwszy od fizycznego adresu na przykład 514, to dane się nie zapiszą i karta zasygnalizuje w rejestrze STATUS błąd ADDRESS ERROR.

Adresowanie danych na karcie można uprościć używając zamiast adresów fizycznych, logicznych numerów sektorów o długości 512 bajtów. Sektory są ponumerowane od zera do wartości maksymalnej określonej przez pojemność karty (można ją odczytać z rejestru CID). Fizyczny adres początku sektora niezbędny jako argument komendy zapisu lub odczytu danych jest wyliczany przez pomnożenie logicznego numeru sektora przez 512 (długość sektora). Używając ponumerowanych sektorów operujemy na blokach danych o długości 512 bajtów. Kiedy decydujemy się stosować karty o pojemności 256, czy 512 MB zakładamy, że potrzebny będzie dostęp do informacji o pojemności wielu sektorów.

Przykładem zastosowania kart MMC/SD może być akwizycja danych z systemów pomiarowych. Jedną z możliwości może być też zapisywanie na karcie próbkowanych informacji dźwiękowych automatycznej sekretarki. Dźwięk przy ograniczeniu szerokości pasma do 3,5 kHz będzie próbkowany z częstotliwością 8 kHz. Każda próbka jest zapisywana na 2 bajtach, czyli sekunda nagrania wymaga zapisania 8000 próbek po 2 bajty – razem 16000 bajtów. Komunikat 2 minutowy 2*60*16000=1920000 bajtów – będą one zapisane na 3750 sektorach. Gdybyśmy chcieli wykorzystać kartę 256 MB, to nagralibyśmy na nią ponad 120 takich komunikatów o długości 3750 sektorów. Stosowanie karty MMC/SD byłoby w pełni uzasadnione. W pamięci sekretarki może być umieszczonych wiele komunikatów o różnej długości. Automatyczna sekretarka musi mieć możliwość odczytania komunikatów, wykasowania komunikatów ze środka listy z pozostawieniem początkowych i końcowych, itp. Żeby mieć możliwość manipulowania nagraniami trzeba stworzyć system zarządzania nagraniami. Ogólniej mówiąc przy wykorzystaniu kart o dużej pojemności będzie potrzebny system organizowania bloków sektorów, tak by można je było zidentyfikować, a potem odtwarzać, kasować i dogrywać nowe w wolnej przestrzeni pamięci.

Tab. 1. Przykładowa tablica z zapisanymi początkami i końcami plików

Numer sektora początku pliku 1
Numer sektora końca pliku 1
Numer sektora początku pliku 2
Numer sektora końca pliku 2
Numer sektora początku pliku 3
Numer sektora końca pliku 3
Numer sektora początku pliku 4
Numer sektora końca pliku 4
Numer sektora początku pliku 4
Numer sektora końca pliku 4
Numer sektora początku pliku 5
Numer sektora końca pliku 5
...

Przykład organizacji danych w automatycznej sekretarce

Nazwijmy blok sektorów reprezentujących jedno nagranie plikiem. Plik można zidentyfikować w pamięci karty MMC/SD przez podanie numeru logicznego pierwszego sektora (początku pliku) i numeru logicznego ostatniego sektora (końca pliku). Jeżeli zapiszemy w tablicy początki i końce wszystkich plików zapisanych w karcie, to na podstawie tych informacji można manipulować informacjami pliku i samym systemem plików.

W naszym przykładzie z sekretarką na podstawie numerów początku i końca sektorów nagrania, (czyli pliku) można określić jego długość w sektorach. Znając długość w sektorach i częstotliwość próbkowania można wyliczyć czas nagrania w sekundach. Dla częstotliwości próbkowania 8 kHz, 1 sektor to 0,32 sekundy. Przykładową tablicę przechowująca dane o plikach zapisanych na karcie przedstawiono w **tab. 1**.

Karta o pojemności 256 MB ma nie mniej niż 490000 sektorów, a karta o pojemności 512 MB nie mniej niż 990000 sektorów pamięci przeznaczonej do zapisania przez użytkownika. Określenie numeru sektora zajmuje 24 bity, co najmniej taką daną należy do tego celu zarezerwować. Ponieważ przykładowe procedury organizujące pamięć karty będą napisane w języku C, to dla wygody będziemy stosować liczby 32–bitowe najczęściej definiowane typem long lub unsigned long.

Każdy numer sektora z tab. 1 będzie zapisany w 4–bajtowej liczbie, a cała tablica początków i końców plików zostanie umieszczona w sektorze zerowym karty MMC/SD. 32–bitową liczbę można zapisać w 8–bitowej pamięci na różne sposoby. Przyjmiemy, że najstarszy bajt zostanie zapisany pod najniższym adresem.

ADRES	BAJT	PLIK
0x000	0x00	PLIK #1 początek 0x000001
0x001	0x00	
0x002	0x00	
0x003	0x01	
0x004	0x00	PLIK #1 koniec 0x00003C15
0x005	0x00	
0x006	0x3C	
0x007	0x15	
0x008	0x00	PLIK #2 początek 0x00003C16
0x009	0x00	
0x00a	0x3C	
0x00b	0x16	
0x00c	0x00	PLIK #2 koniec 0x000AB476
0x00d	0x0A	
0x00e	0xB4	
0x00f	0x76	
0x010	0x00	PLIK #3 początek
0x011	0x0A	
0x012	0xB4	
0x013	0x77	

Rys. 1. Sposób zapisania identyfikatorów początku i końca plików 8 bitowej pamięci Flash sektora zerowego

Pliki w tablicy są identyfikowane kolejnością zapisania: plik pierwszy jest zapisany jako pierwszy, plik drugi jest zapisany jako drugi i tak dalej. Numery sektorów kolejnych plików są zapisywane w tablicy bez żadnych przerw. Numer początku sektora pliku jest zwiększonym o jeden numerem końca pliku poprzedniego. Plik pierwszy zaczyna się od sektora numer 1. Na **rys. 1** został pokazany przykład zapisania numerów sektorów dla pliku pierwszego i drugiego. Plik pierwszy zaczyna się od sektora o numerze 1, a kończy na sektorze o numerze 0x00003C15. Plik drugi zaczyna się od sektora o numerze 0x00003C16, a kończy na sektorze o numerze 0x000AB476.

W 512 bajtowym sektorze zerowym można zapisać 128 liczb 32–bitowych. Pozwala to na zorganizowanie identyfikacji 64 plików (każdy plik to 2 liczby: sektor początku i sektor końca) w pamięci karty MMC/SD. Oprócz adresów logicznych początków i końców plików potrzebne będzie zapisanie innych zmiennych. Ograniczymy więc liczbę plików do 63, a miejsce przeznaczone w sektorze zerowym karty na plik 64 będzie zarezerwowane na numer ostatnio zapisanego pliku.

Na **list. 1** zostały pokazane procedury zapisania numeru sektora początku i końca pliku w 512–bajto-

List. 1. Zapisanie numeru początku i numeru końca sektora pliku

```
unsigned char buf_sector[512];   //bufor RAM o długości sektora.

char WriteIdFile(char file, unsigned long start_sect, unsigned long stop_sect)
{
int index;     //indeks bufora buf_sector;
  if(file>62)
  return(-1);//powrót z błędem numeru pliku
  index=file*8;//początek obszaru zapisywania numeru sektora początku pliku
  buf_sector[index++]=start_sect>>24;//najstarszy bajt numeru początku
             buf_sector[index++]=start_sect>>16;
  buf_sector[index++]=start_sect>>8;
  buf_sector[index++]=start_sect;//najmłodszy bajt numeru początku

  buf_sector[index++]=stop_sect>>24;//najstarszy bajt numeru końca
             buf_sector[index++]=stop_sect>>16;
  buf_sector[index++]=stop_sect>>8;
  buf_sector[index]=stop_sect;//najmłodszy bajt numeru końca
  return(0);//powrót bez błędu
}
```

List. 2. Funkcja odczytywania numerów sektorów pliku i umieszczania ich w zmiennych *start_sect* i *stop_sect*.

```
char ReadIdFile(char file)
{
int index;
  if(file>62)
  return(-1);//powrót z błędem numeru pliku
  index=file*8;
  start_sect=buf_sector[index++];//najstarszy bajt numeru sektora początku
  start_sect=start_sect<<8;
  start_sect=start_sect|buf_sector[index++];
  start_sect=start_sect<<8;
  start_sect=start_sect|buf_sector[index++];
  start_sect=start_sect<<8;
  start_sect=start_sect|buf_sector[index++];//najmłodszy bajt numeru sektora
początku

  stop_sect=buf_sector[index++];//najstarszy bajt numeru sektora końca
  stop_sect=stop_sect<<8;
  stop_sect=stop_sect|buf_sector[index++];
  stop_sect=stop_sect<<8;
  stop_sect=stop_sect|buf_sector[index++];
  stop_sect=stop_sect<<8;
  stop_sect=stop_sect|buf_sector[index];//najmłodszy bajt numeru sektora końca

  return (0);
}
```

List. 3. Funkcja obliczenia długości pliku w sektorach

```
unsigned long GetSizeFile(char file)
{
  ReadIdFile(file);//odczytaj początek i koniec pliku
  return((stop_sect-start_sect)+1);
}
```

List. 4. Funkcja odczytująca numeru sektora końca ostatniego pliku

```
unsigned long GetEndFile(void)
{unsigned long end_file;
 int index;
  index=buf_sector[504]*8;//pod lokacją 504 jest zapisany numer ostatniego pliku
  end_file=buf_sector[index++];//poczatek pliku o numerze 63
  end_file=end_file<<8;
  end_file=end_file|buf_sector[index++];
  end_file=end_file<<8;
  end_file=end_file|buf_sector[index++];
  end_file=end_file<<8;
  end_file=end_file|buf_sector[index];
  return(end_file);
}
```

wym buforze RAM zapisanym wartością odczytaną z sektora zerowego karty. Dla wygody adresowania bufora zmienna *file* określająca numer pliku numeruje pliki od zera do 63. Dla pliku #1 pokazanego na rys. 1 *file=0*, dla pliku #2 *file=1* i tak dalej.

Bufor *buf_sector[]* po zmodyfikowaniu jest zapisywany do sektora zerowego karty. Na **list. 2** pokazano funkcję odczytywania numerów sektorów pliku i umieszczania ich w zmiennych *start_sect* i *stop_sect*.

Funkcja *GetSizeFile()* oblicza i zwraca długość pliku (w sektorach) o numerze określonym w argumencie *file* (**list. 3**).

Jeżeli chcemy dopisać do zapisanych już plików kolejny plik, to trzeba odczytać numer ostatnio zapisanego pliku. Numer ten, jak pamiętamy, jest zapisany w przestrzeni pamięci zarezerwowanej dla ostatniego 64 pliku w komórce o adresie 504 sektora zerowego. Znając numer ostatnio zapisanego pliku można odczytać z tablicy identyfikatorów (rys. 1) numer sektora ostatnio zapisanego pliku. Po doczytaniu numeru sektora ostatnio zapisanego pliku jest on zwiększany o jeden i staje się numerem początkowym sektora nowo zapisywanego pliku.

Odczytanie numeru sektora ostatnio zapisanego pliku pozwala na szybkie wyliczenie pojemności pamięci zajętej przez dane. Jest też wykorzystywany w procedurach porządkowania zawartości pamięci. Na **list. 4** pokazana jest funkcja odczytująca numeru końca ostatniego pliku.

Funkcja zapisania pliku *WriteFile* (**list. 5**) zaczyna się od odczytania sektora zerowego (*SectorRead*) karty i wpisania jego zawartości do bufora *buf_sector* umieszczonego w pamięci RAM mikrokontrolera. Po sprawdzeniu, czy teraz zapisywany plik nie ma numeru większego niż dopuszczalny, odczytywany jest numer sektora końca ostatnio zapisanego pliku. Ten numer po inkrementacji i przepisaniu do zmiennej *start_sect* jest numerem początku zapisywanego pliku. Po zakończeniu zapisywania zawartości pliku trzeba uaktualnić tablicę identyfikatorów plików. Ponieważ do zapisywania pliku był używany bufor *buf_sector*, to trzeba ponownie odczytać zawartość sektora zerowego karty. Pod adresem 504 zostanie zapisany inkrementowany numer pliku, a funkcja *WriteIdFile* zapisze początek i koniec pliku w tablicy identyfikatorów. Zmodyfikowany bufor *buf_sector* zostanie zapisany w sektorze zerowym karty.

Odczytywanie pliku również rozpoczyna się od odczytania sektora zerowego karty. Numer odczytywanego pliku jest umieszczony w argumencie funkcji i nie może być większy niż numer ostatnio zapisanego pliku umieszczony pod adresem 504 sektora zerowego karty. Początek i koniec pliku o numerze umieszczonym w argumencie funkcji jest odczytywany przez funkcję *ReadIdFile*. Kolejno odczytywane sektory mogą być na przykład przesyłane łączem szeregowym do komputera, przesyłane do kodeka audio itp. Proces odczytywania kończy się, gdy bieżący numer sektora jest równy numerowi sektora końca pliku.

Dopisywanie nowych plików do zapisanych wcześniej jest operacją dość prostą. Na podstawie numeru ostatnio nagranego pliku określamy miejsce, od którego można nagrywać. Jednak przychodzi taki moment, w którym trzeba będzie porządkować nagrane pliki. Porządkowanie będzie polegało na kasowaniu niepotrzebnych plików. W przykładzie z automatyczną sekretarką konieczne będzie skasowanie nagranych wiadomości po odsłuchaniu. Skasowanie wszystkich informacji w pokazywanym tu rozwiązaniu jest banalnie proste.

Wystarczy wyzerować zmienną zapisaną pod adresem 504 sektora zerowego karty określającą numer ostatnio nagranego pliku. Procedura odczytywania plików sprawdza tą wartość i jeżeli jest zerowa to nie odczytuje zawartości pliku. Wyzerowanie numeru ostatnio nagranego pliku można porównać do formatowania nośnika. Wszystkie wcześniej zapisane sektory pozostają na karcie (do momentu zapisania nowych plików), ale system plików ich nie widzi. W czasie nagrywania nowego pliku zacznie się on nagrywać od pierwszego sektora kasując wszystkie wcześniej nagrane sektory.

Proste jest również skasowanie ostatnio nagranego pliku. Wystarczy dekrementować numer ostatnio nagranego pliku. Sektory tego pliku zostają na karcie, ale system plików już go nie widzi.

Wszystko się komplikuje, jeżeli chcemy usunąć część plików ze środka listy pozostawiając te, które zostały nagrane jako ostatnie. Na **rys. 2** został pokazany przykładowy system czterech plików.

Załóżmy, że nie jest już nam potrzebny plik #2, a pozostałe chcemy zachować. Najprościej byłoby zmodyfikować tablicę identyfikatorów tak, by po pliku #2 pozostało wolne miejsce na karcie. Inaczej mówiąc plik #2 pozostałby na karcie, ale nie byłby widziany przez system plików. Takie rozwiązanie mimo dużej prostoty powoduje, że w prostym systemie plików miejsce zajęte przez #2 nie będzie wykorzystane do momentu wykasowania wszystkich plików poza plikiem #1. Jeżeli pliki będą duże i operacja kasowania będzie wykonywana często, to może zabraknąć miejsca na nowe pliki. Zastosujemy inne rozwiązanie pozbawione tych wad.

Kasowanie pliku #2 będzie polegało na przepisaniu plików #3 i #4 tak, by dalej tworzyły jeden blok sektorów i plik #3 zaczynał

0x00000001	PLIK #1
0x000001DC	
0x000001DD	PLIK #2
0x00000234	
0x00000235	PLIK#3
0x00000760	
0x00000761	PLIK#4
0x00002345	

Rys. 2. System plików zapisanych na karcie (zapisano 4 pliki)

List. 5. Funkcja zapisująca plik

```
char WriteFile(void)
{
char file;
unsigned long start_sect, sector;
  SectorRead(0,buf_sector);//odczytanie sektora zerowego karty
                    //i zapisanie bufora buf_sector
  file=buf_sector[504];
  ++file;
  if(file==64)
  return(-1);//powrót z błędem - nie da się już zapisać pliku
  start_sect=GetEndFile();//odczytanie sektora końca ostatniego pliku
  ++start_sect;//wyliczenie sektora początku zapisywanego pliku
  sector=start_sect;//bieżący licznik zapisywanych sektorów

  while (.....){
  ......
  .......
  WriteSector(++sector, buf_sector);//zapisywanie sektorów pliku

  ........
  ........
  }
  //koniec zapisywania sektorów pliku
  ReadSector(0,buf_sector);//odświeżenie bufora sektora zerowego
  buf_sector[504]=file;//numer teraz zapisanego pliku
WriteIdFile(file,start_sect,sector);//po zakończeniu zapisywania pliku
            //zapisywane są identyfikatory początku i końca WriteSector(0,buf_sec-
tor);//zapisanie zmodyfikowanego sektora zerowego
}
```

List. 6. Funkcja odczytująca plik

```
char ReadFile(char file)
{
unsigned int sector;
  SectorRead(0,buf_sector);//odczytanie sektora zerowego karty i zapisanie bufora
buf_sector
  if(buf_sector[504]==0)
  return(0);//nie ma plików do odtwarzania
  if(file>buf_sector[504])
  return(-1);//powrót z błędem
  if(buf_sector[504]==0)
  return(0);//nie ma plików do odtwarzania
  ReadIdFile(file);//odczytanie identyfikatorów pliku
  while(start_sect!=stop_sect+1)
  {SectorRead(start_sect++,buf_sector);//odczytywanie sektorów pliku
   .......
        .......
   .......
  }
}
```

się od tego samego sektora, od jakiego zaczynał się plik #2 przed skasowaniem. Obrazowo można to porównać do wieży z klocków, z której wyjęto ze środka jeden klocek. Pozostałe klocki przesuwają się na miejsce wyjętego i dalej tworzą wieżę.

Do przepisania bloku sektorów potrzebne będą:

- numer sektora końca ostatnio nagranego pliku,
- numer sektora początku obszaru do kasowania, czyli numer sektora początku kasowanego pliku,
- numer sektora końca obszaru do kasowania, czyli numer sektora końca kasowanego pliku.

Na podstawie numeru sektora końca ostatnio nagranego pliku i numeru końca kasowanego pliku wyliczana jest liczba sektorów do przepisania – w naszym przykładzie z rys. 2 jest to łączna liczba sektorów plików #3 i #4. Ten blok sektorów jest przepisywany tak, by zaczynał się od początku kasowanego pliku. Przepisywanie bloku sektorów może być operacją trwającą długo, bo może wymagać zapisania dużej liczby sektorów. Samo przepisanie nie kończy procesu kasowania pliku. Trzeba jeszcze skorygować tablicę identyfikatorów plików tak, by „trafiały" w nowo przepisane sektory.

Na **list. 7** została pokazana funkcja kasowania pojedynczego pliku z listy. Po odczytaniu zerowego sektora karty sprawdzane jest, czy kasowany plik nie jest ostatnim na liście. Jeżeli tak, to zmniejszana jest tylko zmienna określająca liczbę plików na karcie i funkcja kończy działanie.

Kasowanie pliku ze środka listy plików rozpoczyna się od odczytania sektora końca ostatnio nagranego pliku oraz sektorów początku i końca kasowanego pliku. Na podstawie tych informacji wyliczana jest liczba sektorów do przepisania i skasowania.

Blok przepisywanych sektorów ma długość równą sumie długości plików nagranych po kasowanym pliku

0x00000001	PLIK #1
0x000001DC	
0x000001DD	PLIK #2 (poprzednio #3)
0x00000708	
0x00000709	PLIK#3 (poprzednio #4)
0x000022ED	
.....	Identyfikatory niewidziane przez system plików
.....	

Rys. 3. System plików zapisanych na karcie po skasowaniu pliku #2

List. 7. Kasowanie pojedynczego pliku

```
char DeleteFile(char file)
{
unsigned long max_sector,start_sec_del,stop_sec_del,sec_to_del,sec_to_rem;

unsigned char max_file;
  SectorRead(0,buf_sector);//odczytanie sektora zerowego karty
                   //i zapisanie bufora buf_sector
  max_file=buf_sector[504];//ostatnia nagrana ścieżka
  if(file>max_file)
   return(-1);//powrót z błędem
  if(file==max_file) //kasowany jest ostatni plik na liście plików
  {--max_file;
                buf_sector[504]=max_file;
                WriteSector(0,buf_sector);
                return;}
     ReadIdFile(max_file);//wyliczenie identyfikatorów ostatniego pliku
  max_sector=stop_sect;//ostatni nagrany sektor na karcie (w całym systemie plików)

  ReadIdFile(file);//wyliczenie identyfikatorów kasowanego pliku

  start_sec_del=start_sect;//numer sektora początku obszaru do skasowania
  stop_sec_del=stop_sect;//numer sektora końca obszaru do kasowania

  sec_to_del=GetSizeFile(file);//ilość sektorów do skasowania - wartość
                       //potrzebna do korekty identyfikatorów

  sec_to_rem=max_sector-stop_sec_del;//liczba sektorów do przepisania
     ++stop_sec_del;//ustaw się na początek następnej ścieżki
  //przepisz wszystkie sektory od stop_sec_del do max_sector
  //w dół do pozycji start_sec_del
  do{
  SectorRead(stop_sec_del++,buf_sector);//odczytaj sektor
  SectorWrite(start_sec_del++,buf_sector);//zapisz sektor
     --sec_to_rem;
      }while(sec_to_rem!=0);
  //wszystkie sektory przepisane
  //korekta początków i końców ścieżek
  do{
  ++file;
  ReadIdFile(file);//identyfikatory następnej ścieżki
  start_sect=start_sect-sec_to_del;
  stop_sect=stop_sect-sec_to_del;
  --file;
  WriteIdFile(file,start_sect,stop_sect);//zapisz do korygowanej ścieżki
  ++file;//ustaw się ponownie na następną.

  }while(file!=max_file);
  --max_file;
  buf_sector[504]=max_file;
  WriteSector(0,buf_sector);
}
```

i jest zapisywany od sektora, w którym zaczynał się kasowany plik. Po przepisaniu miejsce zajmowane przez kasowany plik jest zajmowane przez pliki nagrane po nim.

Żeby można było poprawnie zidentyfikować przepisane pliki trzeba skorygować ich identyfikatory początku i końca. W tym celu od każdego identyfikatora trzeba odjąć liczbę określającą długość kasowanego pliku. Ostatnią czynnością jest zmniejszenie o jeden liczby nagranych plików na karcie.

Po skasowaniu pliku #2 (przykład z rys. 2) tablica identyfikatorów będzie wyglądać tak, jak to zostało pokazane na **rys. 3**.

Pokazany tu system organizacji plików pozwala na zorganizowanie nagrywanych informacji. Jego zaletą jest prostota i możliwość zaimplementowania na dowolnym mikrokontrolerze wyposażonym w pamięć RAM o pojemności pozwalającej zmieścić 512–bajtowy bufor danych i pozostałe zmienne programu.

Tomasz Jabłoński, EP
tomasz.jablonski@ep.com.pl