Czytnik kart SD na ARM-ie, część 2

AVT-5134

Pamięci masowe w postaci kart SD/MMC znalazły już stałe miejsce zarówno w urządzeniach profesjonalnych, jak i amatorskich, stąd coraz większe zainteresowanie nimi naszych Czytelników. W artykule opisujemy budowę czytnika takich kart. Po zapoznaniu się z teorią przedstawioną w pierwszej części, w drugiej zostaną opisane konkretne rozwiązania zastosowane w czytniku kart SD.

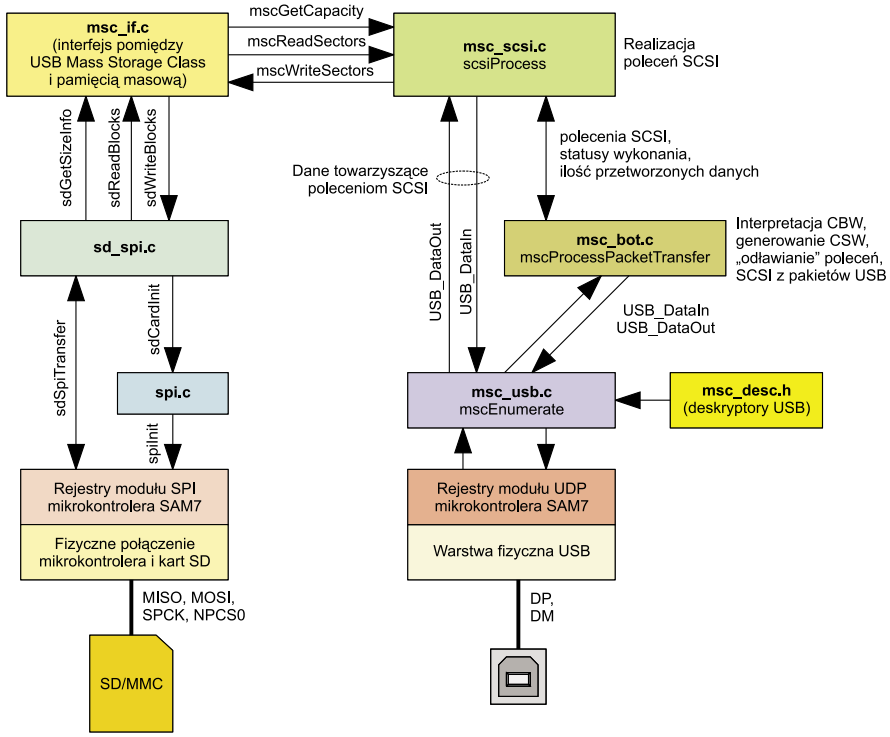
Rekomendacje:

czytnik pozwoli wygodnie przesyłać dane pomiędzy kartą pamięciową SD a komputerem.

Skoro znamy już zarys teorii, warto spojrzeć na **rys. 4** ilustrujący, jak powyższe rozważania zostały zrealizowane w praktyce w czytniku kart. Na rysunku tym naniesione są najważniejsze nazwy plików oraz funkcji biorących udział w przetwarzaniu danych pomiędzy interfejsem USB i kartą SD.

Blokiem, który wymaga szczególnego komentarza jest plik *msc_if.c*. Plik ten składa się z samych funkcji *inline*, więc nie powinien wprowadzić dodatkowych narzutów czasowych. Zastosowany został po to, aby jak najbardziej ułatwić dostosowanie biblioteki *USB Mass Storage* do posiadanej pamięci masowej. Interfejs tworzony w pliku *msc_if.c* jest najprostszym z możliwych i jednocześnie uniwersalnym interfejsem do pamięci masowych. Należy wspomnieć, że odmianą tego interfejsu posługują się biblioteki implementacji systemu plików FatFs oraz EFSL.

W funkcjach znajdujących się w pliku *msc_if.c* jeden z parametrów niewykorzystywanych w czytniku kart to `lun`. Jest to parametr jednobajtowy (w czytniku kart ma zawsze wartość 0), reprezentujący numer



Rys. 4. Graficzna interpretacja działania czytnika kart SD

PODSTAWOWE PARAMETRY

- Płytka o wymiarach 85x44 mm
- Zasilanie: przez port USB lub z zasilacza zewnętrznego
- Obsługiwane karty pamięci: SD i MMC
- Interfejs: USB 2.0, Full Speed
- Klasa urządzenia: Mass Storage Class (pamięć masowa USB)
- Rzeczywista szybkość zapisu danych na kartę: około 220 kB/s
- Rzeczywista szybkość odczytu danych z karty: około 300 kB/s
- Brak konieczności instalacji sterowników dla systemów Windows XP i 2000

jednostki logicznej LUN (*Logical Unit Number*), do której aktualnie odnosi się host. Zerowa wartość parametru `lun` bierze się stąd, że podczas inicjalizacji funkcja *mscInit* (z pliku *msc_ui.c*) została wywołana z parametrem 0. Jeśli dla przykładu funkcję tę wywołamy z parametrem 2, w funkcjach zawartych w *msc_if.c* (czyli w interfejsie pamięci masowej) będą pojawiać się parametry `lun` o wartościach 0, 1 i 2. Zależnie od parametru `lun` będziemy wtedy mogli wykonać działania dla odpowiedniej pamięci masowej, np. dla kilku kart SD lub dla karty SD, pamięci DataFlash i dysku twardego, obsługiwanych za pomocą jednego czytnika. Każda z tych pamięci będzie jednostką logiczną i przypiszemy jej jedną z wartości parametru `lun`.

Co z systemem plików?

W dyskusjach na temat *Mass Storage* Class, jakie toczą się na forach internetowych, wiele osób twierdzi, iż do poprawnej pracy urządzenia potrzebna jest implementacja systemu plików (np. często wykorzystywany w urządzeniach przenośnych FAT). Otóż system plików warto zaimplementować w czytniku jedynie wtedy, gdy chcemy, aby czytnik miał możliwość autonomicznej modyfikacji lub odczytu zawartości karty SD (na poziomie plików, a nie sektorów) bez udziału hosta USB. Urządzenie *Mass Storage Class* samo w sobie nie ma jakiejkolwiek kontroli nad treścią zapisywaną lub odczytywaną z niego, a operacje zapisu/odczytu prowadzone są sektorami (dysk) lub blokami (karta). Host podaje adres bloku (czyli jego numer LBA) z dostępnej puli wszystkich bloków oraz dane, jakie ma do niego zapisać lub z niego odczytać. Urządzenie *Mass Storage* ma jedynie odczytać lub zapisać blok zgodnie z tym, co mówi host, i to host zajmuje się organizacją bloków i danych w nich zawartych tak, aby na dysku/karcie dane były zapisywane w określonym systemie plików.

Dostęp do karty

Najważniejszą zasadą, jakiej należy bezwzględnie przestrzegać podczas pisania oprogramowania wykorzystującego *Mass Storage Class* i bibliotekę systemu plików jest zapewnienie wyłącznego dostępu do karty tylko hostowi USB (biblioteka *Mass Storage*), albo funkcjom modyfikującym zawartość karty (biblioteki systemu plików). Modyfikacja zawartości karty wtedy, gdy host ma do niej dostęp będzie miała nieprzewidywalne skutki, nawet jeśli host w danym momencie nie prowadzi żadnych operacji na zawartości karty.

Budując urządzenie dołączane do portu USB, ale mogące także pracować z własnym zasilaniem będziemy musieli wykryć, czy host USB został do niego podłączony, czy też nie. Sprawdzenie, czy host został dołączony można przeprowadzić, odczytując napięcie sygnału V_DETECT. Z kolei napięcie V_DETECT najprościej odczytać sprawdzając stan logiczny wyprowadzenia PA18 mikrokontrolera.

Jeśli dysponuje się w swoim projekcie systemem operacyjnym, do kontroli dostępu do karty SD doskonale nadaje się mechanizm semaforów wbudowany w system. Dzięki niemu po podłączeniu urządzenia do hosta USB i „przejęcia" przez niego semafora, urządzenie może być nadal funkcjonalne z wyłączeniem jedynie procedur modyfikacji karty SD. Semafor zostanie zwolniony po wykryciu odłączenia urządzenia od hosta – wtedy przejęcie semafora będzie możliwe przez bibliotekę systemu plików.

Najrozsądniejszym sposobem implementacji dostępu do karty SD we własnym programie nie jest modyfikacja danych na poziomie bloków, lecz zastosowanie odpowiedniej biblioteki systemu plików. Przykładowo, jako popularne biblioteki można wymienić: EFSL (*Embedded Filesystems Library*) lub FatFs – obie służą do implementacji systemu plików FAT. Stosując bibliotekę systemu plików dokonujemy operacji na plikach, a nie na blokach.

Przykład: Jeśli chcemy zbudować rejestrator temperatury zapisujący dane w pliku tekstowym, implementujemy w urządzeniu przedstawioną tutaj bibliotekę *Mass Storage Class* i bibliotekę obsługi systemu plików, powiedzmy EFSL. W pętli głównej programu sprawdzamy, czy na wyprowadzeniu odpowiadającym V_DETECT jest napięcie, czyli czy host USB jest podłączony. Jeśli sygnał V_DETECT będzie w stanie niskim (host odłączony, urządzenie pracuje autonomicznie), możemy wykonywać pomiary temperatury i zapisywać dane do plików za pomocą EFSL, jeśli natomiast okaże się, że V_DETECT jest w stanie wysokim, to znaczy, że musimy:

a) zakończyć prowadzone operacje na plikach (zamknąć pliki),
b) włączyć podciągnięcie linii DP interfejsu USB do +3,3 V, gdy tylko będziemy gotowi do rozpoczęcia transmisji USB (w czytniku do tego celu służy sygnał DP_PULLUP),
c) zacząć wywoływać w pętli funkcję *mscUIEnumerate* aż do czasu, gdy sygnał V_DETECT nie przejdzie w stan niski (czyli cały czas, gdy urządzenie będzie dołączone do hosta).

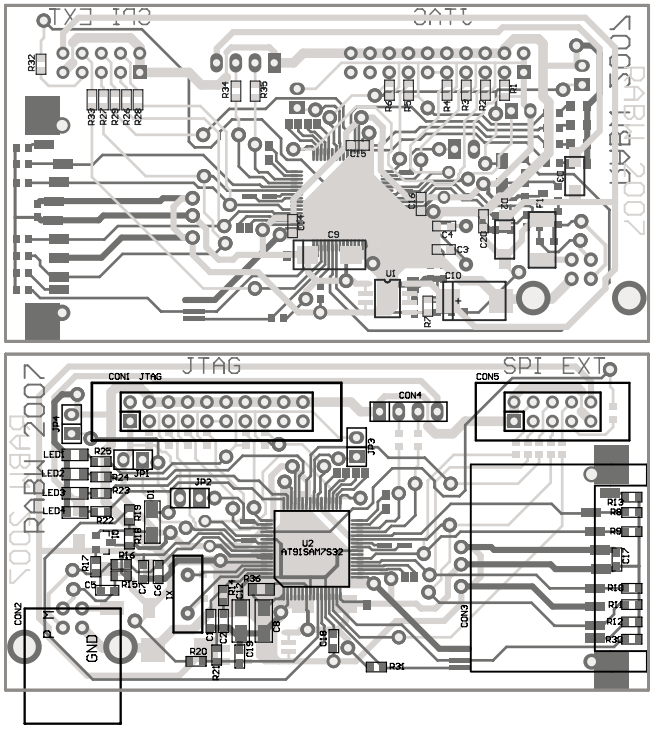
W tym czasie (gdy urządzenie jest podłączone do hosta) nie wolno wykonywać innych dostępów do karty niż za pomocą *mscUIEnumerate*. Po odłączeniu hosta USB, w urządzeniu należy ponownie zainicjalizować system plików, ponieważ host mógł zmodyfikować dotychczas używane w urządzeniu pliki.

Rozbudowa urządzenia

Zgodnie z tym, co zostało napisane wcześniej, czytnik kart jest jedynie prezentacją wykorzystania klasy magazynującej we własnym projekcie. Sam czytnik kart o takiej funkcjonalności jak w niniejszym artykule ma bardzo niewielką szybkość (dużo szybszy czytnik wyprodukowany w Chinach można kupić za kilka złotych), jednak wzbogacenie własnego urządzenia o możliwość dostępu do pamięci masowej za pomocą USB może zaowocować znaczącym wzrostem jego funkcjonalności.

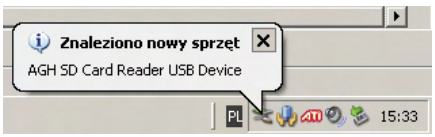
Pisząc oprogramowanie dla własnego urządzenia wykorzystującego pamięć masową, zaistnieje potrzeba dołączenia do niego, oprócz opisanej tutaj biblioteki *Mass Storage Device Class*, także biblioteki obsługującej konkretny system plików – najczęściej będzie to biblioteka obsługująca którąś z odmian systemu plików FAT.

Obsługa wszystkich interfejsów napisana jest bez wykorzystania jakichkolwiek przerwań. Można powiedzieć, że takie postępowanie marnuje możliwości mikrokontrolera. Jest to prawda, lecz tutaj szybkość pracy nie jest krytyczna, choć i tak jest



Rys. 5. Schemat montażowy płytki czytnika

spora jak na USB *Full Speed*. Zrezygnowanie z wykorzystania przerwań przede wszystkim upraszcza kod i ułatwia przenoszenie biblioteki do innych aplikacji. Przykładowo projekt *Mass Storage* został przeze mnie pomyślnie przetestowany, gdy pracował wraz z systemem czasu rzeczywistego FreeRTOS [12]. Jedyne, co wystarczyło zrobić, to uruchomić *task*, który – podobnie jak w niniejszej wersji projektu – zawierał wywołanie funkcji *mscUIEnumerate* i niewielkie opóźnienie (dla bezpieczeństwa można wyłączyć scheduler systemu operacyjnego na czas wykonywania funkcji *mscUIEnumerate*). Każdy, kto próbował zaimplementować synchronizację przerwań i tasków w systemie FreeRTOS wie, ile kosztuje stabilne uruchomienie przerwań i przekazanie odpowiednich parametrów do synchronizacji (np. handlerów do kontroli semaforów, czy tasków) pomiędzy trybami Thumb (system) i ARM (przerwania). Ważne jest, aby pętla z wywołaniem *mscUIEnumerate* wykonywała się dość szybko (jeśli program będzie za długo zwlekał z kolejnymi transferami danych, host



Rys. 6. „Dymek" ukazujący się podczas instalacji czytnika

zinterpretuje taką sytuację jako błąd transmisji danych).

Montaż i uruchomienie

Schemat montażowy płytki drukowanej czytnika przedstawiono na **rys. 5**. Montaż może być trudny ze względu na niewielki rozstaw wyprowadzeń mikrokontrolera AT91SAM7S64. Nie polecam wykonywania płytki drukowanej w warunkach domowych bez doświadczenia praktycznego i przestrzegania reżimu technologicznego. Dobrym rozwiązaniem może być zastosowanie płytki ewaluacyjnej z mikrokontrolerem AT91SAM7S64/128/256 możliwej do kupienia w popularnych sklepach internetowych. W przypadku zastosowania zestawu ewaluacyjnego należy odpowiednio:

- skorygować definicję wyprowadzenia sterującego podciągnięciem linii DP interfejsu USB,
- sprawdzić wyprowadzenie mikrokontrolera, do którego podłączony jest sygnał V_DETECT,
- skorygować ewentualne przyłączenie diod LED,
- zmienić przypisanie wyprowadzeń kontrolerów PIO i SPI dla karty SD.

Plik wynikowy dostarczony jest w wersji binarnej pod nazwą *SD_CARD_READER.bin*.

Zalecane jest programowanie mikrokontrolera SAM7 przy pomocy interfejsu JTAG. Jeśli Czytelnik zamierza uruchomić układ w miarę szybko i bezstresowo polecam zakup odpowiedniego interfejsu JTAG dla ARM7TDMI. Schematy interfejsu JTAG są także ogólnodostępne w Internecie. Ściągając pakiet Win ARM [1] otrzymujemy schematy kilku różnych wersji tanich JTAG–ów podłączanych do portu LPT komputera PC. Są to klony tzw. Wigglera – wtyczki komercyjnej, jednak rozpracowanej przez zwolenników darmowych narzędzi. Jako software do programowania pamięci Flash mikrokontrolera najłatwiej użyć programu H–Flasher będącego modułem programu H–JTAG dostępnego za darmo.

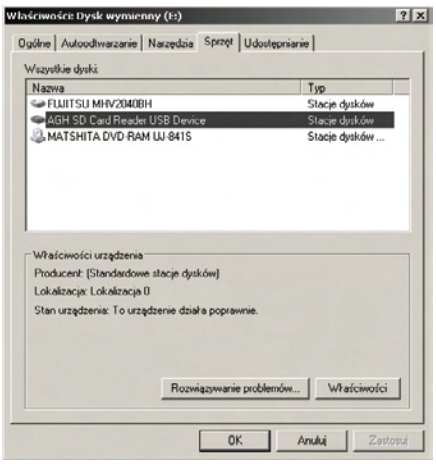
Programowanie przy pomocy tzw. SAM Boot Assistant (SAM–BA) jest bardzo kłopotliwą czynnością, szczególnie, że w układach SAM7S należy podciągać odpowiednie wyprowadzenia (PA0...2) do szyn zasilania. Ta metoda programowania była już opisywana w EP [14]. Płytka posiada złącze USB i DBGU, więc przy odrobinie cierpliwości można odpowiednio wysterować wyprowadzenia PA0...2 oraz zaprogramować układ przez SAM–BA.

Po odpowiednim zaprogramowaniu mikrokontrolera, przy odłączonym zasilaniu, można włożyć kartę SD do gniazda, a następnie podłączyć urządzenie do portu USB. W tym momencie powinna rozpocząć się pierwsza enumeracja układu i po pewnym czasie powinien pojawić się dysk wymienny w oknie „Mój komputer" o wielkości rzeczywistej pojemności karty SD (jeśli karta jest sformatowana). Przykładowy „dymek" podczas instalacji został przedstawiony na **rys. 6**, a widok okienka właściwości dysku na **rys. 7**.

Uwagi końcowe

Układ był testowany na komputerach PC z systemami Windows 2000 oraz XP – w każdym przypadku działał poprawnie bez instalacji jakichkolwiek dodatkowych sterowników. Uruchomienie układu na komputerze z systemem Windows 98SE nie powiodło się z powodu braku sterowników ogólnej klasy magazynującej (*Generic Mass Storage Device Class*).

Czytnik poprawnie działał z kartami:



Rys. 7. Okno właściwości dysku wymiennego

– SD 128 MB oraz 1 GB firmy Kingston (wersja standardowa, bez hi–speed),
– SD 256 MB bliżej nieokreślonej firmy (na karcie znajduje się jednak oficjalne logo SD),
– kartą oznaczoną jako „Secure Digital" firmy Hama, lecz bez oficjalnego logo SD, o pojemności 256 MB,
– SD 16 MB firmy Canon,
– MMC 16 MB firmy Canon.

Na koniec chciałbym jeszcze wytłumaczyć się z rozbieżności w organizacji kodu i nazwach funkcji w tym opracowaniu oraz w artykule [13]. Biorą się one stąd, że zanim bieżący artykuł został opublikowany napisałem nową wersję oprogramowania obsługującego klasę magazynującą USB. Niniejszy artykuł przepisałem tak, aby dotyczył już nowej wersji, która zapewne ułatwi czytanie i analizę kodu.

Opiekunami naukowymi projektu są: dr inż. Cezary Worek i mgr inż. Łukasz Krzak. Projekt powstał w ramach praktyk wakacyjnych kierowanych przez dr inż. Cezarego Worka w Katedrze Elektroniki Akademii Górniczo–Hutniczej w Krakowie. Specjalne podziękowania kieruję do Pana mgr inż. Łukasza Krzaka za pomoc przy realizacji transmisji USB.

Robert Brzoza–Woch