Czytnik kart SD na ARM-ie, część 1

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. W pierwszej części artykułu zajmiemy się wyjaśnieniem ogólnej zasady jego działania i rozszyfrujemy kilka związanych z tym pojęć.

Rekomendacje:

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

Dodatkowe materiały do artykułu publikujemy na CD-EP oraz www.ep.com.pl

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

W urządzeniach typu *embedded* dostrzega się tendencje podobne do tych, które były typowe dla komputerów klasy PC: stosowanie coraz szybszych procesorów, powiększanie pamięci operacyjnej oraz nieulotnej. Do niedawna rozwiązaniem zadowalającym wielu konstruktorów było użycie w układzie mikroprocesorowym pamięci EEPROM o pojemności kilkudziesięciu do kilkuset kilobajtów. Obecnie chcąc skonstruować np. rejestrator mający możliwości zgromadzenia dużych ilości danych, naturalnym rozwiązaniem wydaje się zastosowanie w nim popularnych kart pamięci masowej. Z kilku powodów, takich jak niska cena, łatwy i szybki interfejs (SPI) oraz dostępność, optymalnym rozwiązaniem wydają się być karty *Secure Digital* (SD) lub ich „poprzedniczki" – *Multi Media Card (MMC)*.

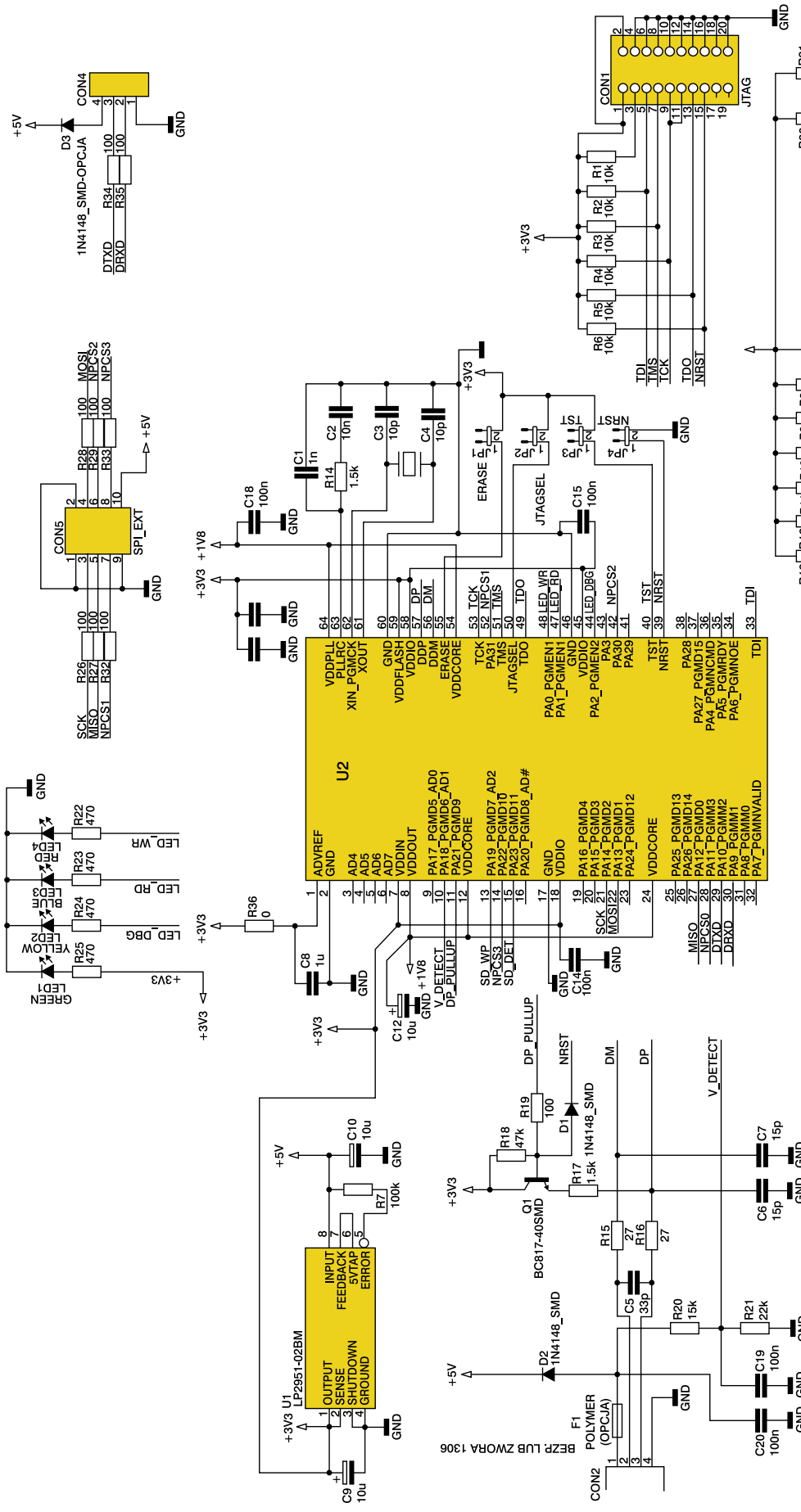
Możliwości urządzenia

Opisywane urządzenie umożliwia dostęp do zawartości zewnętrznych kart SD oraz MMC za pomocą portu USB. Dołączona do czytnika karta pamięci widziana jest przez system operacyjny jako dysk wymienny (w systemie Windows dysk pojawiający się w oknie „Mój komputer"). Dozwolone jest zatem formatowanie karty z poziomu systemu operacyjnego oraz wszystkie operacje na plikach zawartych na karcie SD/MMC: ich kopiowanie, przenoszenie czy usuwanie. Funkcjonalność tę należy traktować jednak tylko jako przykład realizacji dostępu do pamięci masowej poprzez USB. Głównym celem artykułu jest opisanie działania urządzenia klasy magazynującej w taki sposób, aby Czytelnik mógł wykorzystać kod programu czytnika kart we własnym projekcie opartym na układach serii AT91SAM7 firmy ATMEL. Lektura artykułu powinna także ułatwić implementację klasy magazynującej na innym mikrokontrolerze lub też dołączenie do czytnika innego rodzaju pamięci masowej.

Rozwiązania sprzętowe

Ze względu na stosunkowo prostą implementację oraz duże zasoby sprzętowe zastosowano w czytniku popularny mikrokontroler z rdzeniem ARM7TDMI typu AT91SAM7S64 firmy ATMEL. Kompletny schemat elektryczny płytki ewaluacyjnej czytnika zamieszczono na **rys. 1**.

Do komunikacji z komputerem PC za pomocą interfejsu USB służy złącze CON2. Aby możliwe było użycie tego interfejsu w mikrokontrolerach SAM7 należy dołączyć specjalny układ pośredniczący składający się z kilku elementów dyskretnych. Kondensatory C5...C7, rezystory R15 oraz R16 tworzą układ kondycjonowania sygnału USB – zapewniają dopasowanie



Rys. 1. Schemat elektryczny płytki ewaluacyjnej czytnika

impedancji oraz tłumią zakłócenia wysokich częstotliwości. Elementy wokół tranzystora Q1 (R17...R19) tworzą układ podciągania linii DP interfejsu USB przez rezystor o wartości 1,5 kΩ. Dioda D1 ma za zadanie wyłączenie podciągnięcia linii DP, podczas zerowania układu nóżką NRST mikrokontrolera. Tranzystor Q1 pracuje jako wtórnik emiterowy sterowany linią PA21, toteż załączanie podciągnięcia jest kontrolowane programowo przy pomocy wyprowadzenia PA21 układu SAM7 i do normalnej pracy musi być włączone (PA21 w stanie wysokim). Gdyby układ czytnika miał być zasilany zewnętrznie, należy zapewnić odpowiednie sterowanie tej linii wg specyfikacji USB 2.0 [2]. W najprostszym ujęciu trzeba po prostu włączać podciągnięcie linii DP tylko wtedy, gdy jest możliwa do zrealizowania transmisja USB.

Rozwiązanie wspomnianego kondycjonowania sygnału za pomocą elementów C5...C7, R15, R16

jest zaczerpnięte ze starej wersji noty katalogowej mikrokontrolerów AT91SAM7S64/128/256. W jej nowszych wersjach kondensatora C5 (pomiędzy liniami DM i DP interfejsu USB) nie montuje się, natomiast zamiast kondensatorów C6 i C7 o niewielkiej pojemności 15 pF stosuje się rezystory o dużej wartości rezystancji (330 kΩ), co według producenta jedynie zapobiega „wiszącym wejściom" i związanymi z tym ewentualnym problemami ze zwiększeniem poboru prądu. Przykładowo sytuacja taka może mieć miejsce, gdy host USB jest odłączony. wtedy bowiem sygnały DDM i DDP mikrokontrolera pracują jako wejścia cyfrowe.

Złącze CON4 posiada wyprowadzenia sygnałów DTXD i DRXD jednostki DBGU (*Debug Unit*) mikrokontrolera SAM7. W podstawowej wersji urządzenia (praca jako czytnik kart SD) złącze to nie jest wykorzystywane, lecz dla osób, które mają zamiar modyfikować bibliotekę *Mass Storage* może ono być bardzo przydatne do debugowania kodu, np. przy użyciu terminala. Dioda D3 jest głównie elementem zabezpieczającym przed nieprawidłową polaryzacją napięcia. Jeśli mamy zamiar zasilać płytkę czytnika kart ze złącza CON4, diodę D3 montujemy tak jak na schemacie (anodą do złącza). Zapewniamy wtedy zasilanie płytki z portu USB lub ze złącza CON4. Jeśli natomiast chcemy dołączyć do złącza CON4 układ translatora napięć (np. MAX3232), diodę D3 montujemy katodą w stronę wyprowadzenia 4 złącza CON3 lub zastępujemy zworą.

Złącze kart SD (CON3) jest podłączone do interfejsu SPI mikrokontrolera. Karty SD mają możliwość komunikacji przy pomocy interfejsu typowego dla nich (interfejs 4–bitowy) lub przy pomocy standardowej odmiany interfejsu SPI, którą można ustawić w rejestrach konfiguracyjnych mikrokontrolera SAM7. Oczywiście możliwość wykorzystania układów peryferyjnych mikrokontrolera przemawia za użyciem w projekcie interfejsu SPI, który w tym przypadku jest bardziej wydajny. Dodatkowym atutem SPI jest to, iż bardzo dobre biblioteki systemu plików FAT: EFSL (*Embedded Filesystems Library*) [9] oraz FatFs [10] także posługują się tak skonfigurowanym interfejsem karty SD. Ma to duże znaczenie przy rozszerzaniu funkcjonalności za pomocą własnych modułów lub bibliotek.

Dodatkowe rozszerzenie możliwości czytnika jest możliwe przy pomocy złącza CON5 SPI_EXT, będącego wyprowadzeniem zasilania 5 V (m.in. z USB), linii danych SPI: MOSI, MISO, SPCK oraz sygnałów aktywacji układów podrzędnych (*slave*): NPCS1, NPCS2, NPCS3. Złącze to także nie jest wykorzystywane w najbardziej podstawowej wersji urządzenia. Linie SPI są współdzielone z kartą SD, natomiast linie NPCS1...3 mogą być skonfigurowane w dowolny sposób (kontrola programowa lub przez moduł SPI w mikrokontrolerze).

Zworka JP1 ERASE służy do kasowania zawartości pamięci programu mikrokontrolera. Zworka JP2 JTAGSEL jest używana przy debugowaniu przez interfejs JTAG w trybie *EmbeddedICE* – do normalnej pracy pozostawiamy ją otwartą. Zworka JP3 TST może być używana, jeśli programuje się mikrokontroler przy pomocy SAM–BA [8]. Zworka JP4 (NRST) to sygnał zerowania mikrokontrolera. Będzie ona działała w czasie normalnej pracy jedynie wtedy, gdy sprzętowy sygnał NRST jest włączony programowo (w dostarczonej Czytelnikom aplikacji czytnika jest on aktywowany na początku programu).

Źródło sygnału zegarowego dla mikrokontrolera zapewnia rezonator kwarcowy X1 o częstotliwości pracy 18,432 MHz oraz wbudowany w mikrokontroler układ PLL, dla którego filtr stanowią elementy C1, C2, R14. Dzięki rezonatorowi i włączonej pętli fazowej PLL, rdzeń mikrokontrolera jest taktowany sygnałem o częstotliwości 48,054857 MHz. Wszystkie zastosowane wartości elementów oraz uzyskana częstotliwość taktowania to wartości typowe, powszechnie spotykane w urządzeniach i na płytkach ewaluacyjnych z mikrokontrolerami SAM7.

Do zasilania czytnika został wykorzystany interfejs USB. Obok gniazda USB znajduje się miejsce na bezpiecznik polimerowy (nie montowany w podstawowej wersji – bezpiecznik nie jest konieczny i można zastąpić go zworą, rezystorem 0 Ω w obudowie 1206 lub koralikiem ferrytowym). Mikrokontroler AT91SAM7S64 jest zasilany napięciem 3,3 V uzyskanym z wyjścia taniego stabilizatora low–drop LP2951. Napięcie zasilania rdzenia procesora wewnątrz mikrokontrolera (VDDCORE) oraz układu PLL (VDDPLL) wynosi 1,8 V i jest dostarczone z wewnętrznego stabilizatora znajdującego się w mikrokontrolerze SAM7.

Do wizualizacji stanu układu służą 4 diody LED o różnych kolorach świecenia. Dioda zielona

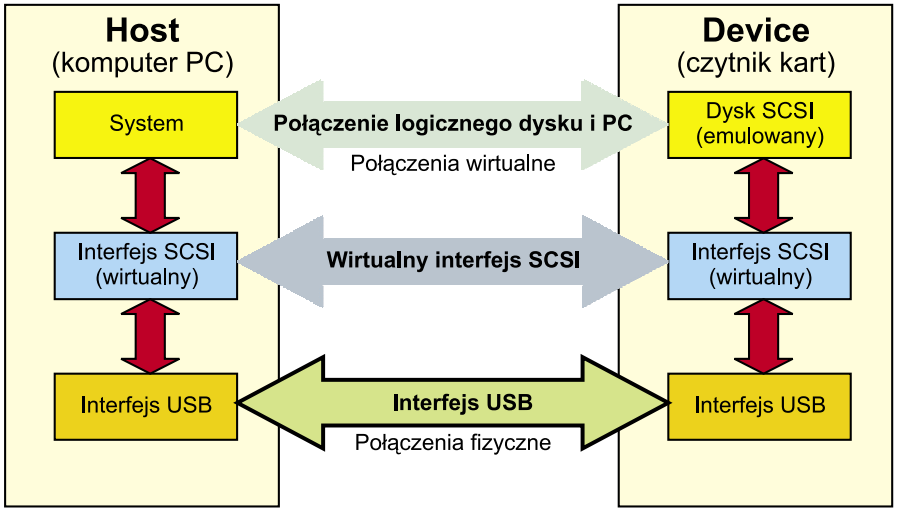
WYKAZ ELEMENTÓW

Rezystory
R1...R6, R8...R13, R30, R31: 10 kΩ (0603)
R7: 100 kΩ (0603)
R14, R17: 1,5 kΩ (0603)
R15, R16: 27: Ω (0603)
R18: 47 kΩ (0603)
R19, R26...R29, R32...R35: 100 Ω (0603)
R20: 15 kΩ (0603)
R21: 22 kΩ (0603)
R22...R25: 470 Ω (0603)
R36: 0 Ω (lub zworka) (0805)

Kondensatory
C1: 1 nF (0603)
C2: 10 nF (0603)
C3, C4: 10 pF (0603)
C5: 33 pF (0603)
C6, C7: 15 pF (0603)
C8: 1 μF (0805) monolit
C9, C10, C12: 10 μF/16 V tantalowy, rozmiar A
C14...C20: 100 nF (0603)

Półprzewodniki
U1: LP2951 (SO-8)
U2: AT91SAM7S64
D1: 1N4148
D2: S1D lub podobna
D3: S1D, podobna lub zwora
LED1: dioda LED zielona (0805)
LED2: dioda LED żółta (0805)
LED3:: dioda LED niebieska (0805)
LED4 : dioda LED czerwona (0805)
Q1: BC817

Inne
CON1: goldpin 2x10
CON2: gmniazdo USB-B
CON3: złącze kart SD
CON4: goldpin 1x4
CON5: goldpin 2x5
F1: zwora lub bezpiecznik
JP1...JP4: goldpin 2x1
X1: rezonator kwarcowy 18,432 MHz



Rys. 2. Schemat blokowy tzw. modelu warstwowego transmisji

(LED1) oznacza obecność napięcia zasilania. Dioda żółta (LED2) zmienia swój stan z każdym przejściem pętli głównej programu. Dioda niebieska (LED3) oznacza odczyt bloku z karty SD, natomiast dioda czerwona (LED4) zapis bloku do karty SD. Dzięki takiej sygnalizacji można orientować się jakie czynności w danym momencie wykonuje czytnik. Jeśli podczas startu czytnika nie umieścimy w nim karty SD/MMC lub wystąpi błąd podczas inicjalizacji karty, włączą się jednocześnie diody: niebieska i czerwona, natomiast żółta pozostanie wyłączona.

USB – zarys działania

Przewijające się w tekście sformułowanie „klasa magazynująca" oznacza typ (klasę) urządzenia zdefiniowaną w standardzie USB (*Mass Storage Class*). Niskopoziomowa wymiana danych oraz przesyłanie pakietów (czyli odpowiednio przygotowanych do wysłania lub odbioru ciągów danych) między urządzeniami USB różnych klas wygląda podobnie – różnice występują jedynie w treści danych.

Zaraz po włączeniu wtyczki USB do czytnika kart zostaje przeprowadzony tzw. proces enumeracji. W czasie enumeracji host (w tym przypadku komputer PC) wysyła rozmaite polecenia i pytania do urządzenia, czyli czytnika kart. Dzięki tym pytaniom urządzenie ma szansę „przedstawić się", czyli m.in. określić w jakiej klasie pracuje. Host zbiera także informacje o rozmiarach buforów danych w urządzeniu (tzw. *Endpointy*), a nawet o tym, ile prądu potrzebuje urządzenie. Informacje o urządzeniu są przekazywane przez tzw. deskryptory, czyli ciągi bajtów o odpowiednich wartościach. W deskryptorze zawarte są także informacje o tym, w jaki sposób należy się komunikować z urządzeniem oraz jaki to rodzaj urządzenia. Gdy host zbierze wszystkie potrzebne informacje o urządzeniu, rozpoczyna właściwą wymianę danych z urządzeniem magazynującym.

Przedstawiony tutaj czytnik pracuje w klasie *Mass Storage Class* – mówimy więc, że klasą jego interfejsu – *Interface Class* – jest *Mass Storage Class*. Podklasa jego interfejsu (*Interface Subclass*) to *SCSI Transparent*, a protokołem interfejsu (*Interface Protocol*) w tym przypadku jest *Bulk–Only Transport*. Poniżej zostanie przedstawione znaczenie tych nazw.

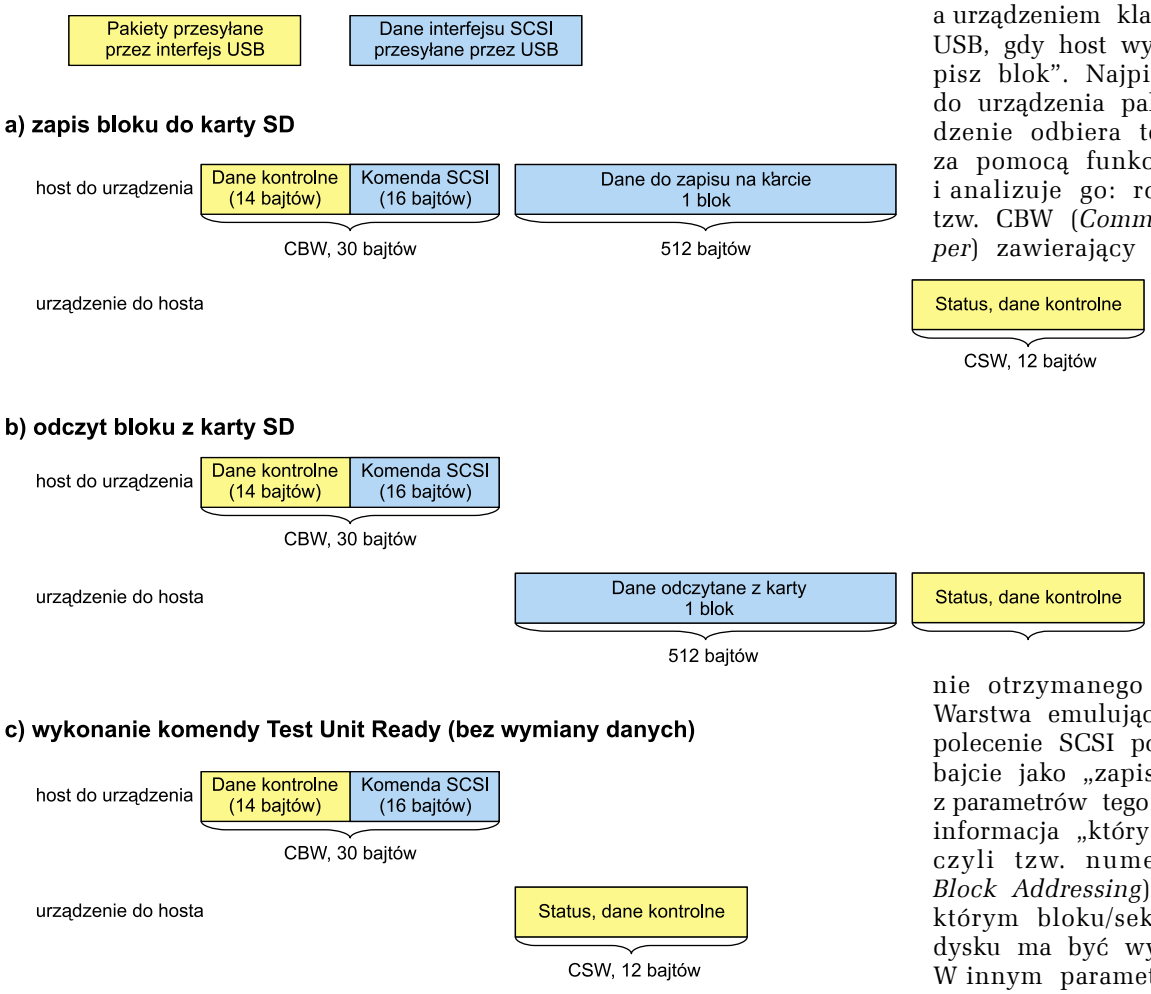
Mass Storage Class oznacza, iż urządzenie będzie widziane w systemie jako tzw. pamięć masowa (przykładem pamięci masowej jest dysk twardy). Jest to oczywiste.

Protokół *Bulk–Only Transport* (BOT) [3][4] oznacza sposób wymiany informacji z hostem, m.in. ze względu na przydział czasu. BOT jest rodzajem transmisji USB o takim priorytecie, że transmisja jest zoptymalizowana pod kątem dużych ilości danych, a nie ze względu na szybką reakcję na małe ilości danych. BOT nie będzie więc dobry np. do myszki lub klawiatury pracującej w standardzie USB, ponieważ urządzenia te przesyłają niewiele danych na jednostkę czasu, czas ich dostarczenia jest natomiast krytyczny. W przypadku pamięci masowej opóźnienia pojedynczych pakietów nie są tak istotne (nie są zauważalne dla użytkownika i są częściowo redukowane przez zastosowanie odpowiednich buforów). Dlatego BOT w przypadku pamięci masowej jest optymalnym rodzajem komunikacji.

Wspomniane zostało, że czytnik kart jest dla hosta skonfigurowany jako urządzenie z interfejsem *SCSI–Transparent*. Oznacza to, że przez USB będą przesyłane pakiety SCSI – oczywiście odpowiednio przygotowane do transmisji USB. Możemy sobie wyobrazić taką sytuację: system operacyjny „widzi" interfejs SCSI i wpisuje do niego dane. Dane te są następnie tłumaczone na odpowiednio przygotowane pakiety USB i przez USB wysyłane do urządzenia. Urządzenie otrzymując taki pakiet USB „wyciąga" z niego ramkę danych SCSI, jaką nadał host, a następnie wykonuje to, co mówi aktualnie komenda SCSI.

Umieszczanie ramek danych jednego rodzaju (tutaj SCSI) w ramkach danych innego rodzaju (tutaj USB) można uznać za realizację tzw. modelu warstwowego transmisji. Schemat blokowy takiego modelu przedstawiono na **rys. 2**. Dzięki wprowadzeniu podziału na warstwy, elementy przedstawione na schemacie blokowym komunikują się między sobą przez połączenia wirtualne. Na pewnym poziomie abstrakcji można przyjąć, że host (komputer PC) wpisuje dane prosto do wirtualnego „dysku" przez interfejs SCSI nie bacząc na fakt, że pomiędzy komputerem a „dyskiem" dane muszą przejść przez wiele warstw. Oczywiście w naszym przypadku „wirtualny dysk" tworzymy (emulujemy) programowo w czytniku kart.

Wspomniany tutaj dysk jest przyłączony wirtualnym „kablem" do hosta (górna strzałka na rys. 2). Dzięki podziałowi na warstwy komunikacji, host może się odnosić do tego dysku jakby znajdował się on wewnątrz komputera. Jest tak nawet pomimo tego, że informacje przechodzą najpierw przez kilka warstw „w dół", łącze USB i znowu kilka warstw „w górę", a nie bezpośrednio przez np. tasiemkę ATA/ATAPI czy SATA, tak jak do



Rys. 3. Graficzne wyjaśnienie warstwowej transmisji danych pomiędzy hostem a urządzeniem klasy magazynującej USB a) po wydaniu przez hosta polecenia „zapisz blok", b) po wydaniu przez hosta polecenia „odczytaj blok"; c) po wydaniu przez hosta polecenia „*Test Unit Ready*"

„zwykłego" dysku wewnątrz komputera.

Co to jest SCSI

SCSI (*Small Computer System Interface*) [5][6][7] jest interfejsem, w którym dane są przekazywane w ściśle zdefiniowanych pakietach. Podobnie jak w przypadku USB, istnieje wiele różnych specyfikacji SCSI dla różnych rodzajów urządzeń. Popularnymi urządzeniami komunikującymi się tą metodą są pamięci masowe. Czytelnicy mogą zwrócić uwagę, że np. napędy CD–ROM mają interfejs ATAPI (choć już coraz częściej SATA). Fakt. Są to jednak interfejsy bardzo zbliżone do siebie i translacja danych pomiędzy nimi jest stosunkowo prosta.

W tym momencie nasuwa się pytanie, dlaczego operujemy standardem SCSI, skoro karta SD wykorzystuje w projekcie interfejs SPI oraz specyficzny dla niej zestaw komend? Po pierwsze: typy wymiany danych zastosowane w projekcie czytnika kart (połączenie *BOT* i *SCSI–Transparent*) są jednymi z najłatwiejszych do implementacji; po drugie: nie ma wielu innych standardów do wyboru; po trzecie: transmisja taka wydaje się być optymalna dla urządzeń pamięci masowej z powodu tego, że zestaw komend SCSI „obowiązujący" przy realizacji urządzenia *USB Mass Storage Class* jest tak dobrany, aby interfejs pamięci masowej był jednocześnie funkcjonalny i nie za bardzo rozbudowany.

Jak realizowane są polecenia SCSI?

Wspomniana warstwowa transmisja danych jest realizowana tak, jak zostało to przedstawione na **rys. 3a, 3b** i **3c**.

Na rys. 3a przedstawiono schemat komunikacji pomiędzy hostem a urządzeniem klasy magazynującej USB, gdy host wyda polecenie „zapisz blok". Najpierw host wysyła do urządzenia pakiet USB, a urządzenie odbiera ten pakiet (u nas za pomocą funkcji *USB_DataOut*) i analizuje go: rozpoznaje w nim tzw. CBW (*Command Block Wrapper*) zawierający 14 bajtów sterujących oraz 16 bajtów stanowiących komendę SCSI. Teraz następuje przekierowanie polecenia SCSI „wyłowionego" z ostatnich 16 bajtów pakietu do warstwy emulującej dysk, gdzie nastąpi rozpoznanie otrzymanego polecenia SCSI. Warstwa emulująca dysk rozpozna polecenie SCSI po pierwszym jego bajcie jako „zapisz blok". Jednym z parametrów tego polecenia będzie informacja „który sektor zapisać", czyli tzw. numer LBA (*Logical Block Addressing*) oznaczający, na którym bloku/sektorze na karcie/dysku ma być wykonana operacja. W innym parametrze zawarta jest informacja o liczbie bloków, które należy zapisać (dla uproszczenia jednak w naszych rozważaniach przyjmujemy, że tylko 1). Typowo, jeden blok ma rozmiar 512 bajtów. W tym momencie urządzenie wie, że następne dane jakie nadejdą, będą właściwymi danymi, czyli muszą zostać zapisane na kartę do bloku wskazanego adresem LBA. Urządzenie zatem wykona serię odczytów danych nadchodzących z hosta USB i zapisze te dane do medium (czyli do określonego bloku karty SD). Odczytów powinno być tyle, ile pakietów danych wyśle host, co jednocześnie jest zgodne z ilością danych, jaka została zdeklarowana przez hosta w parametrach komendy SCSI.

Po odebraniu danych z hosta i zapisie tych danych na karcie SD, czytnik kart wyśle przez port USB do hosta pakiet stanowiący „podsumowanie" transmisji. Podsumowaniem tym będzie pakiet nazywany CSW, czyli *Command Status Wrapper*. W pakiecie CSW urządzenie da znać hostowi, czy otrzymało tyle danych ile należa-

ło oraz zgłosi ewentualne błędy (np. poinformuje hosta, czy nie wystąpił błąd w czasie zapisu lub czy urządzenie wymaga specjalnej uwagi hosta – tzw. status *CHECK CONDITION*).

Analogicznie, na rys. 3b widzimy, co stanie się, gdy host wyśle do naszego czytnika kart pakiet zawierający polecenie „odczytaj blok". Po otrzymaniu tego pakietu i rozpoznaniu w nim komendy SCSI oraz jej parametrów, czytnik kart odczyta blok o adresie danym w parametrach komendy SCSI (podobnie jak w poprzednim przypadku) oraz wyśle odczytane dane do hosta. Po zakończeniu wysyłania danych podsumuje transmisję wysyłając do hosta pakiet USB z danymi CSW.

Na podobnej zasadzie jak odczyt danych z karty pamięci, host może zażądać np. informacji o pojemności nośnika, czyli o tym, jaki rozmiar ma blok danych (u nas blok ma rozmiar 512 bajtów) oraz ile bloków danych zawiera nośnik. W takim przypadku czytnik kart, zamiast wysłać do hosta dane odczytane z bloku na karcie pamięci, wyśle specjalną ramkę zdefiniowaną w standardzie SCSI, zawierającą odpowiedź (tzw. *Response*) na zgłoszone przez hosta żądanie. Taka sytuacja będzie miała miejsce m.in. w przypadku rozpoznawania nowopodłączonego dysku wymiennego do hosta. Na rys. 3c przedstawiono sytuację, gdy transmitowane są jedynie dane sterujące, czyli tylko pakiet CBW z komendą SCSI i pakiet podsumowujący CSW. Przykładem komendy SCSI działającej na tej zasadzie jest *Test Unit Ready* – komenda ta jest wysyłana mniej więcej co sekundę do urządzenia magazynującego. Jeśli urządzenie wykryje, że np. ktoś wyjął kartę z gniazda lub inne medium przechowujące dane, powinno zgłosić ten fakt ustawiając status *CHECK CONDITION* w odsyłanym pakiecie CSW po otrzymaniu komendy *Test Unit Ready*. Później host powinien dowiedzieć się co spowodowało ustawienie statusu *CHECK CONDITION*. Pomiędzy wysłaniem pakietu z CBW zawierającego komendę *Test Unit Ready* a odesłaniem przez czytnik pakietu z CSW, nie ma transmisji jakichkolwiek danych. W przypadku przedstawionego tutaj czytnika kart, możliwość dynamicznego wykrywania czy karta została wyjęta, czy włożona nie została zaimplementowana – odpowiedź na *Test Unit Ready* jest zawsze „bez zmian" czyli status *GOOD*.

Dzięki powyższym informacjom możemy teraz nie zajmować się samym interfejsem USB i jego pakietami, lecz skoncentrować się wyłączenie na poleceniach SCSI.

W artykule *Opis implementacji klasy magazynującej USB na przykładzie mikrokontrolera AT91SAM7S64* [13] znajdują się najważniejsze informacje dotyczące szczegółów zawartości pól w ramkach SCSI oraz kiedy jaką ramkę danych wysłać. Lektura obu artykułów powinna znacząco ułatwić poszukiwanie najważniejszych informacji w kodzie źródłowym i dokumentacji.

Rozpoznawanie dysku

Wróćmy do procesu rozpoznawania nowego urządzenia SCSI. Host i jego system operacyjny muszą dowiedzieć się, jaki nowy dysk został do niego przyłączony. Zbieranie informacji o nowym urządzeniu SCSI rozpoczyna się od wysłania przez hosta komendy *Inquiry* (zapytanie) oznaczającej dla urządzenia podrzędnego SCSI (jest nim nasz emulowany dysk twardy stworzony z karty SD) żądanie wysłania odpowiedniego ciągu bajtów danych z informacjami o podstawowych parametrach naszego wirtualnego napędu (tzw. *Inquiry Response*). W standardach SCSI zdefiniowane są bajty tego ciągu.

Odpowiedź na ramkę *Inquiry* jest zawarta w tablicy stałych *scsiInquiryResponse* w pliku *msc_scsi.c*. W niej też możemy zmienić (oprócz bajtów definiujących m.in., że urządzenie jest urządzeniem z medium wymienialnym) nazwę producenta, nazwę urządzenia oraz jego wersję. Te dwie pierwsze nazwy wyświetlane są w „dymkach" systemu Windows po pierwszej enumeracji urządzenia na danym porcie USB oraz będą one nazwą producenta i dysku w systemie (podmienianie tych nazw jest bardzo proste, a przyjemność z patrzenia na własne urządzenie USB w systemie duża). Należy pamiętać, aby, modyfikując zawartość tablicy *scsiInquiryResponse*, zmieścić się w danej liczbie pól (znaków) – nie polecam dodawania lub przesuwania w niej bajtów, ponieważ mogą z tego wyniknąć niepożądane konsekwencje, np. błędna interpretacja pól w ramce *Inquiry Response* prowadząca do całkowitego zwieszenia się czytnika kart lub nawet chwilowego zawieszenia się systemu operacyjnego.

System operacyjny hosta żąda także rozmiarów dysku. Konkretnie chodzi o rozmiar sektora oraz liczbę sektorów na dysku – na tej podstawie host może obliczyć wielkość przyłączonego dysku. W pliku *sd_spi.c* znajduje się specjalna funkcja *sdGetSizeInfo* obliczająca te wartości na podstawie zawartości odczytanych rejestrów karty SD. Odczytane wartości są następnie odpowiednio układane do ramki odpowiedzi na żądanie SCSI, a ramka ta jest przesyłana przez opisane „wirtualne" połączenie SCSI do hosta.

Robert Brzoza–Woch

Literatura

[1] WinARM – darmowy pakiet programów do pracy z mikrokontrolerami ARM dostępny na stronie [11]
[2] Specyfikacja Universal Serial Bus, Rev. 2.0, www.usb.org
[3] Specyfikacja Mass Storage Class Bulk–Only Transport, Rev. 1.0, www.usb.org
[4] Specyfikacja Mass Storage Class Specification Overview, Rev. 1.2, www.usb.org
[5] SPC–2 – SCSI Primary Commands – 2, Project T10/1236–D, www.t10.org
[6] SPC–3 – SCSI Primary Commands – 3, Project T10/1416–D, www.t10.org
[7] SBC–2 – SCSI Block Commands – 2, Project T10/1417–D, www.t10.org
[8] Datasheet mikrokontrolera AT91SAM7S32/321/64/128/256, Preliminary, wersja 6175E, 4.04.2006,
[9] EFSL – Embedded Filesystems Library, www.efsl.be
[10] FatFs – biblioteka systemu plików FAT, http://elm–chan.org/fsw/ff/00index_e.html
[11] http://www.siwawi.arubi.uni–kl.de/avr_projects/arm_projects/
[12] FreeRTOS, www.freertos.org
[13] Robert Brzoza–Woch – Opis implementacji klasy magazynującej USB na przykładzie mikrokontrolera AT91SAM7S64, EP nr 6 i 7/2007
[14] Lucjan Bryndza – ARM–y w praktyce, Programowanie ISP mikrokontrolerów LPC2000 i AT91SAM7S, część 2, EP nr 1/2006