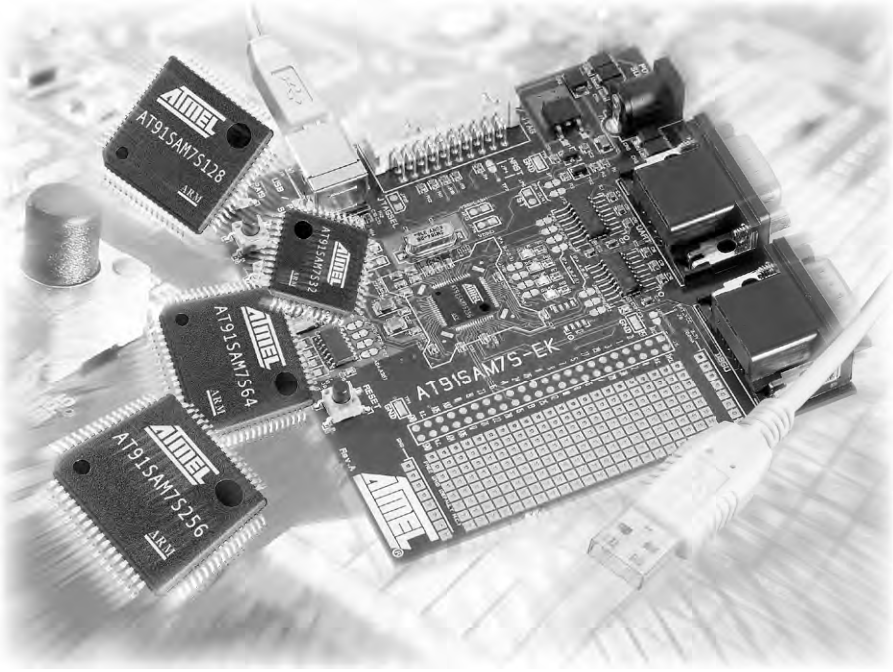
Opis implementacji klasy magazynującej USB na przykładzie mikrokontrolera AT91SAM7S64, część 2

Specyfikacja interfejsu USB jest dość trudna do zrozumienia, stąd wynika olbrzymia popularność układów firmy FTDI, które sprowadzają obsługę tego portu do poziomu transmisji przez znany „na wylot" RS232. Chcąc jednak wykorzystać wszystkie możliwości, jakie udostępnia nam USB trzeba sięgnąć po inne rozwiązania.



Deskryptory USB i Mass Storage – Configuration Descriptor, Interface Descriptor, Endpoint Descriptor

Kolejnym deskryptorem po Device Descriptor, o jaki host pyta urządzenie, jest deskryptor konfiguracji oraz „podległe mu" deskryptory interfejsu i endpointów. Zestawienia i omówienia pól deskryptora konfiguracji, interfejsu i endpointów (z przykładowymi wartościami dla czytnika kart SD) przedstawiono w **tab. 2...4**.

Należy zwrócić uwagę, że host wysyła zapytania o deskryptor konfiguracyjny i jemu podległe jednocześnie. Istnieje także ściśle ustalona kolejność wysyłania tych deskryptorów (z tego też względu w tej części opisane są one razem). Na **list. 3** przedstawiono wszystkie te trzy deskryptory w jednej tablicy stałych, pomimo że tablica nazywa się USB_ConfigDescriptor. Powód umieszczenia ich w jednej tablicy jest taki, że wysłane muszą być zawsze w takiej samej kolejności, a także deskryptory te są ściśle powiązane ze sobą, jak zostało to wspomniane trochę wcześniej.

Tab. 2. Deskryptor konfiguracji

Bajt	Nazwa	Opis	Wartość (dla czytnika kart SD)
0	bLength	Długość deskryptora w bajtach	0x09
1	bDescriptorType	Numer rodzaju deskryptora	0x02
2	wTotalLength	Całkowita długość deskryptora (Config + Interface + EP1...EPn) (L)	0x20
3	wTotalLength	Całkowita długość deskryptora (H)	0x00
4	bNumInterfaces	Liczba interfejsów w tej konfiguracji	0x01
5	bConfigurationValue	Wartość „uaktywniająca" daną konfigurację – numer tej konfiguracji	0x00
6	iConfiguration	Indeks do tekstowego deskryptora danej konfiguracji	0x00
7	bmAttributes	Charakterystyka konfiguracji zasilania i wzbudzania	0xC0
8	bMaxPower	Prąd zużywany przez urządzenie [bMaxPower · 2mA]	0x00

Dwubajtowe pole wTotalLength musi zawierać sumę długości (liczbę bajtów) deskryptora konfiguracji, interfejsu oraz deskryptorów endpointów podległych zawartemu tutaj interfejsowi. Zgodnie z tym, co zostało wspomniane wyżej, możemy ustawić w Configuration Descriptor ile różnych interfejsów ma obsługiwać nasze urządzenie oraz w polu bConfigurationValue możemy zdefiniować wartość przypisaną tej konfiguracji, gdyby było więcej możliwych konfiguracji w urządzeniu. Pola bmAttributes i bMaxPower służą m.in. do powiadamiania hosta o możliwości zasilania urządzenia i zapotrzebowaniu na prąd pobierany przez urządzenie. Nie są one szczególnie istotne dla poprawnej enumeracji, lecz eleganckim rozwiązaniem w ostatecznej wersji urządzenia jest odpowiednie skonfigurowanie tych pól.

W deskryptorze interfejsu ponownie występują pola bInterfaceClass, bInterfaceSubClass, bInterfaceProtocol tak, jak w deskryptorze urządze-

List. 3. Zawartość deskryptorów Configuration Descriptor, Interface Descriptor i Endpoint Descriptor

```
const char USB_ConfigDescriptor[] =
{
    0x09,   //size of this description
    0x02,   //bDescriptorType: config decriptor type
    0x20,   //wTotalLength (L)
    0x00,   //wTotalLength (H)
    0x01,   //bNumInterfaces
    0x01,   //bConfigurationValue
    0x00,   //iConfiguration: index of config string
    0xC0,   //bmAttributes: Configuration characteristics - bit7-reserved,
bit6-self powered
    0x00,    //MaxPower [*2mA]

//interface descriptor
    0x09,   //bLength: size of this descriptor
    0x04,   //bDescriptorType
    0x00,   //bInterfaceNumber
    0x00,   //bAlternateSetting
    0x02,   //bNumEndpoints: Number of endpoints excluding endpoint 0
    0x08,   //bInterfaceClass: Here 0x08 Mass Storage Class
    0x06,   //bInterfaceSubClass: Here: SCSI Transparent Command Set
    0x50,   //bInterfaceProtocol: Here: BULK-ONLY TRANSPORT
    0x00,   //iInterface: Index to string descriptor of this interface

//BULK-IN Endpoint descriptor (EP1)
// ( DEVICE -> HOST )
    0x07,   //bLength: size of this descriptor
    0x05,   //bDescriptorType: Here: Endpoint descriptor type
    0x81,   //Endpoint 1, direction IN
    0x02,   //bmAttributes: This is a Bulk Endpoint
    0x40,   //wMaxPacketSize: Here it is 64
    0x00,   //wMaxPAcketSize higher byte
    0x00,    //bInterval: does not apply to bulk endpoints

//BULK-OUT Endpoint descriptor (EP2)
// ( HOST -> DEVICE )
    0x07,   //bLength: size of this descriptor
    0x05,   //bDescriptorType: Here: Endpoint descriptor type
    0x02,   //bEndpointAddress: Here EP2
    0x02,   //bmAttributes: This is a Bulk Endpoint
    0x40,   //wMaxPacketSize: Here it's 64
    0x00,   //wMaxPAcketSize higher byte
    0x00,   //bInterval: does not apply to bulk endpoints

};
```

nia (Device Descriptor). W Interface Descriptor definiujemy już konkretne wartości pól, w których zostały wpisane zerowe wartości w Device Descriptor. Dzięki definicji klasy na poziomie interfejsu, możemy dopisać do urządzenia możliwość pracy w dodatkowych klasach. Podawanie klasy i interfejsu urządzenia w Interface Descriptor jest także praktyką stosowaną przez producentów urządzeń dostępnych w handlu.

Sądzę, że deskryptory endpointów zostały wytłumaczone na tyle jasno w tab. 4, że nie jest potrzebny dalszy komentarz (wiele bitów nie jest używanych przy pracy endpointu w trybie bulk – szczegóły także w tab. 4). Pozostałe deskryptory, takie jak Language Descriptor, w przypadku najprostszej implementacji MSC nie grają roli, a wyświetlana w systemie nazwa urządzenia jest określana na podstawie odpowiedzi na komendę Inquiry (SCSI). Odpowiedź ta jest omówiona przy okazji objaśniania innych komend SCSI.

Tab. 3. Deskryptor interfejsu – Interface Descriptor

Bajt	Nazwa	Opis	Wartość (dla czytnika kart SD)
0	bLength	Długość deskryptora w bajtach	0x09
1	bDescriptorType	Numer rodzaju deskryptora	0x04
2	bInterfaceNumber	Numer tego interfejsu	0x00
3	bAlternateSetting	Pole zmiany interfejsu w czasie pracy (związane z bInterfaceNumber i obsługą kilku interfejsów – tutaj nie wykorzystywane)	0x00
4	bNumEndpoints	Liczba endpointów używanych przez ten interfejs	0x02
5	bInterfaceClass	Klasa interfejsu – tutaj Mass Storage Class	0x08
6	bInterfaceSubClass	Podklasa interfejsu – tutaj SCSI Transparent Command Set	0x06
7	bInterfaceProtocol	Protokół interfejsu – tutaj Bulk Only Transport (BOT)	0x50
8	iInterface	Indeks do tekstowego deskryptora tego interfejsu	0x00

Urządzenie USB jest rozpoznane – co dzieje się dalej?

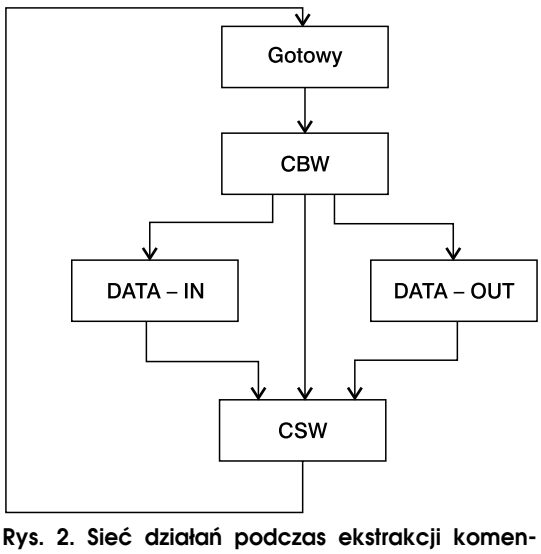
Po pomyślnym przejściu przez proces enumeracji, system komunikuje się już z urządzeniem pracującym w konkretnej klasie, więc jego żądania i pytania są typowe dla danej klasy. W końcowej części funkcji USBRequestProcessor znajduje się obsługa poleceń specyficznych dla MSC (ciągle jeszcze z endpointu EP0) – wywołanie funkcji USB_MSC_RequestProcessor. Tak więc otrzymaliśmy już jakieś dane specyficzne dla klasy magazynującej. W tym momencie zamykamy już temat „czystej", ogólnej specyfikacji USB 2.0, a zajmujemy się tematem Mass Storage Class (dokładny wykaz literatury podano na końcu artykułu). Typowe dla MSC pierwsze polecenie po enumeracji to Get Max LUN. W MSC zdefiniowane są jedynie 2 polecenia specyficzne dla tej klasy. Pierwsze zostało wspomniane i oznacza ile jednostek logicznych (Logical Unit Number) znajduje się w urządzeniu, a drugie to Bulk–Only Mass Storage Reset. Przykładowo: standardowy, dalekowschodni, uniwersalny czytnik kart, który po zainstalowaniu w systemie zgłasza 4 dyski wymienialne, powinien odpowiedzieć na Get Max LUN liczbą 3 (dysk 0, 1, 2 i 3). Można spróbować skompilować kod programu wpisując (w pliku *usb_mass_storage.c*) w odpowiedzi na to polecenie inną liczbę, powiedzmy 2, a wtedy powinny pojawić się w „Mój komputer" 3 dyski wymienialne (oczywiście zaraz potem program zachowa się nieprzewidywalnie, ponieważ aby obsłużyć te „dyski" trzeba je także odpowiednio oprogramować). Właściwie polecenie Get Max LUN jest jedynym poleceniem wydawanym przez system Windows, dlatego w funkcji USB_MSC_RequestProcessor brak jest obsługi zerowania (Bulk–Only Mass Storage Reset).

Dane z endpointu EP2, czyli na styku USB i SCSI

Po żądaniu Get Max LUN i poprawnej odpowiedzi na to żądanie, host powinien – w przypadku czytnika kart – rozpocząć wysyłanie danych do endpointu EP2, czyli Bulk–Out (przypominam nomenklaturę USB: Out oznacza od hosta do urządzenia). Otrzymaliśmy dane, które następnie odczytaliśmy

z jednego z banków endpointu EP2 i co dalej? Obsługą danych, które urządzenie odczytuje z EP2 zajmuje się wstępnie funkcja USB_ProcessProtocol. Te dane, to... komenda SCSI. Oczywiście komenda ta jest „opakowana" przez dodatkowe bajty kontrolujące transmisję. Na razie proponuję Czytelnikowi, aby przyjął standard SCSI jako „czarną skrzynkę" – zostanie on omówiony później, teraz powinno wystarczyć, że do urządzenia SCSI wysyłamy komendę (ramkę) i możemy otrzymać od niego odpowiedź także w tym standardzie (i komenda i odpowiedź to ciągi bajtów zdefiniowane w specyfikacji SCSI). Nie wszystkie komendy SCSI wymagają odpowiedzi. Po ewentualnej odpowiedzi urządzenie musi – można to tak nazwać – „podsumować" transmisję.

Specyfikacja USB Mass Storage Class Bulk–Only Transport zawiera informacje, jak należy „wyłowić" ramkę SCSI z ciągu bajtów znajdującego się w buforze danych nadchodzących z USB. Tę rolę pełni funkcja USB_ProcessProtocol. Czytając specyfikację MSC BOT można się przestraszyć ilości rożnych możliwości i algorytmu postępowania z poprawnymi lub błędnymi danymi nadchodzącymi od hosta. Na szczęście błędne dane rzadko się zdarzają (lub wcale), dlatego obsługa działającego czytnika kart wymaga do poprawnej pracy najprostszych algorytmów. Oczywiście, jeśli ktoś chce, aby jego urządzenie spełniało wszystkie wymagania specyfikacji, należy zaimplementować wszystkie algorytmy.



Rys. 2. Sieć działań podczas ekstrakcji komendy SCSI z nadchodzącego pakietu i tworzenia podsumowania transmisji

Tab. 4. Deskryptor Endpointu – EndpointDescriptor

Bajt	Nazwa	Opis	Wartość (dla czytnika kart SD) EP1/EP2
0	bLength	Długość deskryptora w bajtach	0x07
1	bDescriptorType	Numer rodzaju deskryptora	0x05
2	bEndpointAddress	„Adres" tego endpointu: Bity 3...0 – numer edpointu (tutaj 1 i 2) Bity 6...4 – 0 Bit 7 – kierunek (1–IN, 0–OUT)	0x81/0x02
3	bmAttributes	Atrybuty endpointu: Bity 1...0 – typ transferu – tutaj 10 Bulk Bity 5...2 – tutaj 0 (dla Bulk)	0x02
4	wMaxPacketSize	Maksymalna ilość danych na endpoint – bierzemy wartość z noty katalogowej AT91SAM7S64 (L)	0x40
5	wMaxPacketSize	jw. (H)	0x00
6	bInterval	Odstęp czasu pomiędzy odczytami stanu endpointu przez hosta – tutaj nie używane	0x00

Zasadę ekstrakcji komendy SCSI z nadchodzącego pakietu i tworzenie wspomnianego podsumowania transmisji przedstawiono na **rys. 2**. Jest to rysunek zaczerpnięty ze specyfikacji Mass Storage Class i pokazuje schematycznie etapy wymiany informacji. Bajty znajdujące się w buforze danych nadchodzących z portu USB (w programie jest to USB_EP2_Buffer) zawierają informacje opisane w dokumentacji jako CBW, czyli Command Block Wrapper (dosł. opakowanie bloku komendy), natomiast CSW to Command Status Wrapper, czyli opakowanie statusu komendy i jest to blok danych, który odsyłamy przez USB do hosta po wykonaniu polecenia SCSI – wspomniane wcześniej „podsumowanie" i kontrola transmisji.

Struktura CBW jest przedstawiona w **tab. 5**. 4–bajtowe pole dCBWSignature to liczba, która powinna być równa 0x43425355 (wartość podana w specyfikacji USB MSC) – należy to sprawdzić. W razie, gdyby urządzenie wykryło błędną wartość tej liczby, należy wysłać tzw. Stall do portu USB. Stan Stall oznacza, że urządzenie wymaga interwencji hosta. Następne pole pakietu CBW to dCBWTag. Tę wartość należy odesłać niezmienioną do hosta w strukturze CSW, lecz jej poprawności urządzenie nie sprawdza. Pole dCBWDataTransferLength mówi o tym, ile bajtów ma zostać przesłanych pomiędzy polami CBW i CSW – mowa tutaj o „użytecznych" bajtach, np. danych z karty SD w ramce SCSI, a kierunek ich przesyłania jest opisany przez bit 7 bajtu bmCBWFlags (bit kierunku, reszta bitów tego bajtu jest albo zarezerwowana, albo „obsolete" czyli przestarzała). Jeśli dCBWDataTransferLength wynosi 0, to wtedy bezpośrednio po CBW odsyłamy CSW i ignorujemy bit kierunku. Pole bCBWLUN oznacza urządzenie logiczne, z którym w tym pakiecie wymienia dane host. Jak wspomniałem wcześniej, jedno urządzenie może posiadać np. kilka dysków twardych lub kart pamięci wyświetlanych jako osobne dyski w oknie „Mój komputer". W przypadku czytnika kart pole to powinno wynosić zawsze 0. Przedostatnim polem w CBW jest bCBWCBLength i oznacza ono długość rozpoczynającej się po nim ramki SCSI. Od bajtu 15 CBW do bajtu (15+ bCBWCBLength) bajt po bajcie możemy odczytać polecenie SCSI wydane przez hosta.

Po wykonaniu polecenia SCSI (przypominam, że w jego skład może wchodzić wymiana danych z hostem), urządzenie odsyła do hosta pakiet CSW opisany w **tab. 6**. W tym pakiecie należy w polu dCSWSignature umieścić (jak zosta-

Tab. 5. Struktura CBW

Bajt \ Bit	7	6	5	4	3	2	1	0
0...3	dCBWSignature							
4...7	dCBWTag							
8...11	dCBWDataTransferLength							
12	bmCBWFlags							
13	Reserved (0)						bCBWLUN	
14	Reserved (0)					bCBWCBLength		
15...30	CBWCB							

Tab. 6. Struktura CSW

Bit / Bajt	7	6	5	4	3	2	1	0
0...3	dCSWSignature							
4...7	dCSWTag							
8...11	dCSWDataResidue							
12	bCSWStatus							

ło to wcześniej nadmienione) liczbę 0x53425355. W polu dCSWTag należy umieścić wartość odczytaną z dCBWTag. Pole dCSWDataResidue oznacza różnicę pomiędzy wartością dCBWDataTransferLength a rzeczywistą ilością wysłanych do hosta danych w przypadku transmisji data–in lub ilością danych od hosta odebranych w przypadku transmisji data–out.

Ostatnie pole CSW (bCSWStatus) oznacza status urządzenia po transmisji danych. Wartość zerowa tego pola wskazuje, że transmisja danych zakończyła się sukcesem. Warunki wpisania wartości 0x01 i 0x02 do tego pola są opisane wspomnianymi wcześniej złożonymi algorytmami detekcji błędów. Algorytmy nie zostały zaimplementowane w czytniku kart w pełni, lecz jedynie w bardzo okrojonej wersji. Na szczęście błędy transmisji w czasie normalnej pracy czytnika kart nie zdarzały się nigdy. Nie rozpoznanie komendy ma miejsce w czasie każdego uruchomienia urządzenia, gdy host żąda wysłania danych opisanych w innej specyfikacji SCSI niż wszystkie inne komendy obsługiwane przez czytnik, lecz bez których czytnik może poprawnie działać.

Na tym etapie omawiania programu sterującego czytnikiem kart możemy już zapomnieć o standardzie USB. Jest tak dlatego, że dysponujemy w tym momencie ramką polecenia SCSI, a host (z „wirtualnym" masterem SCSI) jest w stanie odebrać odpowiedź SCSI wygenerowaną przez urządzenie. Tak więc w tym momencie możemy przejść do innego poziomu abstrakcji – do standardu SCSI, ponieważ od hosta do czytnika kart prowadzi już wirtualny „kabel" SCSI.

Tab. 7. Komenda Inquiry

Bit / Bajt	7	6	5	4	3	2	1	0	Wartość
0	Operation Code								0x12
1							Obsolete	EVPD	
2	Page code								
3	Allocation Length								
4									
5	Control								

Interfejs SCSI

Interfejs SCSI jest podobny do interfejsu ATA/ATAPI – wymiana danych przebiega na zasadzie: żądanie układu nadrzędnego (master, tutaj host), wykonanie polecenia i ewentualna odpowiedź układu podrzędnego (np. czytnika kart). Oczywiście odpowiedź urządzenia nie zawsze występuje, czasem host żąda jedynie wykonania pewnych poleceń bez odsyłania odpowiedzi SCSI. Wszystkie komendy SCSI nie będą tutaj opisywane – jedynie najważniejsze zostaną przybliżone skrótowo Czytelnikowi dla umożliwienia analizy programu sterującego. Zainteresowanych tematem odsyłam do specyfikacji SCSI SPC–2, SPC–3 oraz SBC–2. Przebrnięcie przez te specyfikacje nie jest trudne, lecz dość czasochłonne, ponieważ są one bardzo uniwersalne i przez to obszerne – zawierają polecenia dla różnego typu urządzeń. Są jednak przykładem porządku i logiki w tworzeniu dokumentacji, co ułatwia ich czytanie. Przedstawione tutaj komendy SCSI mogą wydawać się opisane

Tab. 8. Ramka odpowiedzi na komendę Inquiry (pierwsze 35 bajtów – tyle należy odesłać)

Bit / Bajt	7	6	5	4	3	2	1	0	Wartości istotnych parametrów
0	Peripheral Qualifier			Peripheral Device Type					0x00
1	RMB	Reserved							0x80
2	Version								0x04
3	Obsolete	Obsolete	NormACA	HiSup	Response Data Format				0x02
4	Additional Length								0x20
5	SCCS	ACC	TPGS		3PC	Reserved		Protect	0x00
6	BQue	EncServ	VS	MultiP	MChngr	Obsolete	Obsolete	ADDR16	0x00
7	Obsolete	Obsolete	Wbus16	Sync	Linked	Obsolete	CmdQue	VS	0x00
8	T10 Vendor Identification								Znak ASCII
...	T10 Vendor Identification								Znak ASCII
15	T10 Vendor Identification								Znaki ASCII
16	Product Identification								Znak ASCII
...	Product Identification								Znaki ASCII
31	Product Identification								Znak ASCII
32	Product Revision Level								Znak ASCII
...	Product Revision Level								Znaki ASCII
35	Product Revision Level								Znak ASCII

Parametrów innych niż kod operacji (Operation Code) można nie brać pod uwagę.

Objaśnienia najważniejszych pól:
Peripheral Device Type = 0x00, wg standardu SCSI urządzenie dostępu bezpośredniego (*Direct–access device*), na przykład: dysk z nośnikiem magnetycznym. Obowiązującą dokumentacją w tym przypadku (oprócz SCSI SPC–2, jest dokumentacja SBC – SCSI Block Commands).
RMB – Removable – nośnik wyjmowalny, jeśli ten bit jest ustawiony.
Version = 0x04 – urządzenie zgodne ze standardem SPC (SCSI Primary Commands)
Response Data Format = 0x02. Ciekawy przypadek (normalny w przypadku wielu modyfikacji standardu) – wartości większe od 0x02 są zastrzeżone (*reserved*), a wartości mniejsze od 0x02 są przestarzałe (*obsolete*).
Additional Length = 0x20 = 32dec. Długość parametrów w bajtach, wg specyfikacji należy podać długość n – 4, czyli 36 – 4 = 32.
T10 Vendor Identification – nazwa dostawcy sprzętu przyznana przez organizację T10 (oczywiście w zastosowaniach amatorskich można wpisać dowolne znaki ASCII – największa przyjemność z modyfikacji)
Product Identification – nazwa identyfikująca produkt – uwagi j.w. – wpisujemy znaki ASCII wyświetlane później w dymkach systemu Windows – opis modyfikacji w głównej części artykułu.
Product Revision Level – wersja produktu – oczywiście można wpisać tutaj coś innego niż same spacje... Uwagi jak dla T10 Vendor Identification.

List. 4. Lista komend SCSI wydanych przez hosta w czasie inicjalizacji czytnika kart SD

```
SCSI Inquiry
SCSI Unknown Command =  0x23
SCSI Read Capacity
SCSI Read
SCSI Mode Sense
SCSI Mode Sense
SCSI Read Capacity
SCSI Read Capacity
SCSI Read
SCSI Read
SCSI Read Capacity
SCSI Read Capacity
SCSI Read
SCSI Test Unit Ready
SCSI Read Capacity
SCSI Read Capacity
SCSI Read Capacity
SCSI Read
SCSI Read
SCSI Read
SCSI Mode Sense
SCSI Test Unit Ready
SCSI Test Unit Ready
SCSI Test Unit Ready
SCSI Test Unit Ready
SCSI Test Unit Ready
SCSI Test Unit Ready
SCSI Test Unit Ready
SCSI Test Unit Ready
SCSI Test Unit Ready
SCSI Test Unit Ready
...
```

Tab. 9. Komenda SCSI Read (10)

Bit / Bajt	7	6	5	4	3	2	1	0	Wartości istotnych parametrów
0	Operation Code								0x28
1	Rdprotect			DPO	FUA	Reserved	FUA_INV	Obsolete	
2	(MSB) Logical Block Address								Adres LBA
3	Logical Block Address								Adres LBA
4	Logical Block Address								Adres LBA
5	Logical Block Address (LSB)								Adres LBA
6	Reserved				Group Number				
7	(MSB) Transfer Length								Ilość bloków
8	Transfer Length (LSB)								Ilość bloków
9	Control								

Operation Code – kod operacji – jak w przypadku Inquiry, po tym polu rozpoznajemy, jaką komendę wysłał do naszego urządzenia host.
Logical Block Address – adres bloku (w naszym przypadku) na karcie SD wg bardzo prostego systemu LBA, czyli: każdy blok posiada swój własny, kolejny numer.
Transfer Length – długość transferu, ile bloków chce odczytać z/zapisać do nośnika host.
WProtect (Write) i **Rdprotect** (Read) – ochrona przed zapisem/odczytem. W specyfikacji SBC–3 wyszczególnione są wszystkie możliwe konfiguracje tego pola.
Pola inne niż (Operation Code, Logical Block Address i Transfer Length) nie są wykorzystywane w czytniku – zainteresowanych odsyłam do dokumentacji SCSI SBC–3. Liczba 10 przy tytule tabeli oznacza długość komendy w bajtach (jest kilka komend Read/Write: (6), (10), (12), (16), (32) różniących się liczbą parametrów i możliwą długością pól, np. Transfer Length, lecz wykorzystywana jest wyłącznie przedstawiona tutaj Read/Write (10).

lakonicznie, jednak celem artykułu jest przede wszystkim przedstawienie ich działania, a nie wnikanie w szczegóły ich konfiguracji.

Na list. 4 przedstawiono listę komend SCSI wydanych przez hosta w czasie inicjalizacji czytnika kart SD (debugging tekstowy przeprowadzony na terminalu). Poniżej zamieszczam znaczenia kluczowych komend i odpowiedzi na nie.

Inquiry. Struktura komendy Inquiry została przedstawiona w **tab. 7**, a pierwsze (i dla nas najważniejsze) bajty odpowiedzi na nią w **tab. 8**. Host, wysyłając komendę Inquiry, żąda od urządzenia SCSI, aby przedstawiło mu swoje parametry. W pliku *scsi_processing.c* znajduje się odpowiedź na to polecenie – tablica stałych z parametrami urządzenia. Opis edycji pól „tekstowych" odpowiedzi na Inquiry został opisany w głównej części artykułu o czytniku kart SD. Najbardziej zagmatwane wydają się być pola początkowe tej odpowiedzi. Zainteresowanych znaczeniami najważniejszych z nich odsyłam do kodu programu. W tym przypadku ważnym bitem jest bit 7 (RMB) bajtu 1 oznaczający, że urządzenie posiada wymienny nośnik. Wartości pozostałych, początkowych bajtów można przyjąć za „słuszne", sprawdzone i działające, i najlepiej ich nie zmieniać.

Read Capacity. Ta komenda żąda od urządzenia, aby zwróciło liczbę bloków oraz rozmiar bloku. Liczbę bloków i rozmiar bloku odczytujemy z rejestrów CSD kary SD. Praktycznie wszystkie karty SD mają rozmiar bloku wynoszący 512 bajtów. Liczba bloków oraz wielkość bloku na karcie SD jest obliczana w funkcji sdGetSizeInfo w pliku *sd_spi.c*. Dane odczytane z tej funkcji są następnie układane do odpowiedzi na żądanie Read Capacity. W tym miejscu warto wyjaśnić różnicę pomiędzy blokiem (Block), a sektorem (Sector) na karcie SD. Otóż blok jest jednostką zapisu i odczytu danych. Może być programowalny lub mieć ustaloną wielkość. Sektor natomiast może być zestawem kilku bloków i procedura kasowania zawartości karty (Erase) odbywa się sektorami. Bardzo często spotyka się, że te dwa pojęcia są stosowane wymiennie, jednak najbardziej poprawne w tym przypadku wydaje się użycie słowa „blok".

Dane na karcie SD są ułożone w blokach po 512 bajtów i każdy blok ma swój własny adres. Ten adres z punktu widzenia SCSI może być interpretowany jako LBA – Logical Block Address. Jest to bardzo wygodna forma adresowania nośników pamięci masowej z powodu prostoty jej stosowania – stanie się to lepiej widoczne przy omawianiu zapisu i odczytu danych z karty.

Test Unit Ready, Request Sense i flagi SENSE KEY, ASC, ASCQ. Komenda Test Unit Ready jest bardzo często wysyłana do urządzenia SCSI. Nakazuje ona urządzeniu SCSI ustawienie odpowiednich flag: SENSE KEY, ASC i ASCQ na wartości zdefiniowane w standardzie SCSI i odpowiednie dla stanu urządzenia. Jeśli urządzenie ma wyjmowane medium (tutaj może to być karta SD), należy poinformować hosta odpowiednim ustawieniem tych flag o tym, iż medium zostało wyjęte lub zmienione. Host odczytuje wartości flag SENSE KEY, ASC i ASCQ przy pomocy komendy Request Sense. Przygotowanie odpowiedzi na nią odbywa się w funkcji SCSI_ProcessRequestSense w pliku *scsi_processing.c*.

Read i Write. Znaczenie tych komend jest chyba najbardziej oczywiste. Niskopoziomowa obsługa karty SD zawiera funkcje sdReadBlock i sdWriteBlock. Funkcje te realizują odpowiednio odczyt i zapis sektora (poprawnie: bloku) z i do karty SD i są wywoływane, gdy urządzenie otrzyma odpowiednie polecenie zapisu lub odczytu. Sama obsługa tych komend SCSI odbywa

Tab. 10. Komenda SCSI Write (10)

Bajt \ Bit	7	6	5	4	3	2	1	0	Wartości istotnych parametrów
0	Operation Code								0x2A
1	WProtect			DPO	FUA	Reserved	FUA_NV	Obsolete	
2	(MSB) Logical Block Address								Adres LBA
3	Logical Block Address								Adres LBA
4	Logical Block Address								Adres LBA
5	Logical Block Address (LSB)								Adres LBA
6	Reserved								Group Number
7	(MSB) Transfer Length								Ilość bloków
8	Transfer Length (LSB)								Ilość bloków
9	Control								

Operation Code – kod operacji – jak w przypadku Inquiry, po tym polu rozpoznajemy, jaką komendę wysłał do naszego urządzenia host.
Logical Block Address – adres bloku (w naszym przypadku) na karcie SD wg bardzo prostego systemu LBA, czyli: każdy blok posiada swój własny, kolejny numer.
Transfer Length – długość transferu, ile bloków chce odczytać z/zapisać do nośnika host.
WProtect (Write) i **Rdprotect** (Read) – ochrona przed zapisem/odczytem. W specyfikacji SBC–3 wyszczególnione są wszystkie możliwe konfiguracje tego pola.
Pola inne niż (Operation Code, Logical Block Address i Transfer Length) nie są wykorzystywane w czytniku – zainteresowanych odsyłam do dokumentacji SCSI SBC–3. Liczba 10 przy tytule tabeli oznacza długość komendy w bajtach (jest kilka komend Read/Write: (6), (10), (12), (16), (32) różniących się liczbą parametrów i możliwą długością pól, np. Transfer Length, lecz wykorzystywana jest wyłącznie przedstawiona tutaj Read/Write (10).

się trochę inaczej niż pozostałych. Dla zmniejszenia wielkości bufora danych SCSI zastosowałem bezpośredni zapis i odczyt do buforów endpointów EP1 i EP2 interfejsu USB, łamiąc zasadę rozgraniczenia tych dwóch interfejsów na dwie różne warstwy. Ilość danych odczytanych lub zapisanych na kartę SD zależy od wartości otrzymanej w parametrach komendy SCSI Read i Write. Struktury tych komend zostały przedstawione w **tab. 9** (Read) i **tab. 10** (Write). Czytnik kart do ustawienia adresu bloku na karcie SD używa bezpośrednio wartości Logical Block Address tych komend. Liczba bloków, jaka ma być odczytana lub zapisana, znajduje się w polach Transfer Length i wartość ta jest wykorzystywana do określenia ile razy należy wywołać funkcje sdReadBlock i sdWriteBlock. Czytelnik lubiący eksperymentować może porównać liczbę bajtów, jakiej żąda host USB, z tym ile bloków należy odczytać (oczywiście liczymy liczbę bloków pomnożoną przez rozmiar bloku) – w moim przypadku liczby te zawsze były zgodne. Odczytać te wartości można oczywiście przy pomocy prostych funkcji dla portu DBGU i terminala.

Interfejs kart SD

Mikroprocesor w czytniku kart komunikuje się z kartą SD za pomocą sprzętowego interfejsu SPI. Funkcje obsługi karty SD zostały napisane w oparciu o opracowanie Martina Thomasa służące do implementacji najniższej warstwy komunikacji dla biblioteki systemu plików FAT: Embedded Filesystems Library. Jest to opracowanie w pełni działające i raczej nie pozostawiające dużych możliwości modyfikacji – może poza implementacją operacji zapisu i odczytu bloków karty SD z użyciem bezpośredniego dostępu do pamięci, tzw. DMA (*Direct Memory Access*) przez kontroler PDC (*Peripheral DMA Controller*) mikrokontrolera AT91SAM7S64. Przy okazji należy wspomnieć, iż DMA w mikrokontrolerach SAM7S jest łatwy w implementacji i dość skuteczny.

Z punktu widzenia obsługi karty SD najważniejsze i docelowe są funkcje:
- inicjalizacja karty SD,
- odczyt pojemności karty SD na podstawie jej rejestrów CSD,
- zapis bufora do bloku na karcie SD o zadanym adresie,
- odczytu bloku z karty SD o zadanym adresie do bufora.

Pozostałe funkcje zawarte w pliku *sd_spi.c* służą jako funkcje pomocnicze (narzędzia) do poprawnego działania wymienionych wyżej funkcji wyższego poziomu.

Co dalej?

Biblioteka obsługi kart SD jest bardzo łatwa do przeniesienia na inne platformy. Dodatkowo, cała może zostać „podmieniona" obsługą innego urządzenia magazynującego. Nic nie stoi na przeszkodzie, aby zaimplementować komunikację czytnika kart np. z dyskiem twardym przez interfejs równoległy ATA/ATAPI. Takie próby mam zamiar przeprowadzić już wkrótce, ponieważ tłumaczenie komend interfejsu SCSI na ATAPI nie powinno sprawić większych trudności. Odpadnie wtedy na przykład konieczność programowej „emulacji" dysku lub napędu CD przez generowanie flag interfejsu SCSI, ponieważ będzie je można odczytać bezpośrednio z urządzenia ATA/ATAPI. Wspomniana tutaj programowa emulacja dysku zostanie dość obrazowo przedstawiona w opisie czytnika kart SD.

Robert Brzoza–Woch