VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

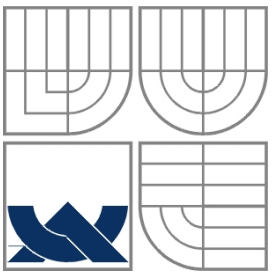
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

PRIPOJENIE PAMÄŤOVEJ KARTY SD K MIKROKONTROLÉRU

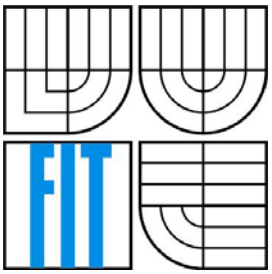
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. PAVEL LAURINC
AUTHOR

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

PŘIPOJENÍ PAMĚŤOVÉ KARTY SD K MIKROKONTROLÉRU

CONNECTING SD MEMORY CARD TO MICROCONTROLLER

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PAVEL LAURINC

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAROSLAV ŠKARVADA

BRNO 2007

Zadání diplomové práce/3351/2006/xlauri00

Vysoké učení technické v Brně – Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2006/2007

Zadáni diplomové práce

Řešitel: **Laurinc Pavel, Bc**.
Obor: Počítačové systémy a sítě
Téma: **Připojení paměťové karty SD k mikrokontroléru**
Kategorie: Počítačová architektura
Pokyny:

1. Seznamte se s architekturou paměťových karet SD (Secure Digital).
2. Prostudujte komunikační protokoly používané těmito kartami. Seznamte se s možnostmi karet v oblasti CPRM (Content Protection for Recordable Media).
3. Zvolte vhodný mikrokontrolér a navrhněte rozhraní pro připojení SD karet k tomuto mikrokontroléru.
4. Navržené rozhraní prakticky realizujte.
5. Navrhněte a implementujte knihovnu podprogramů pro komunikaci s SD kartou přes zvolené rozhraní. Knihovna musí umožnit detekci karty, inicializaci karty a čtení/zápis/vymazání dat v nešifrované paměťové oblasti karty.
6. Funkčnost knihovny demonstrujte na vhodném příkladu.

Literatura:

- dle pokynů vedoucího

Při obhajobě semestrální části diplomového projektu je požadováno:

- Splnění prvních 3 bodů zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné správy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Škarvada Jaroslav, Ing.**, UPSY FIT VUT
Datum zadání: 28. února 2006
Dátum odevzdání: 22. května 2007

_____

doc. Ing. Zdeněk Kotásek, CSc.

vedoucí ústavu

Licenční smlouva

Licenční smlouva je uložena v archivu Fakulty informačních technologií Vysokého učení technického v Brně.

Abstrakt

Autor se zaobírá paměťovými kartami Secure Digital (SD) a 8-bitovými mikrokontrolérmi z rady Atmel AVR. Popisuje jejich architekturu, vlastnosti a technologie použité v těchto zařízeních. Dále vysvětluje princip komunikačních protokolů, pomocí kterých probíhá komunikace s ostatními zařízeními. Rozebírá funkcionalitu souborového systému FAT. Popisuje návrh a implementaci rozhraní pro připojení SD karet k mikrokontroléru. Dále taktéž vysvětluje SW řešení projektu a podává objektivní pohled na využití a srovnaní implementovaných režimů komunikace s kartou.

Klíčová slova

Paměťová karta, SD, MMC, mikrokontrolér, Atmel, AVR, ATmega128, SPI, CPRM, systém souborů, FAT, fragmentace, JTAG.

Abstract

Author concerns with SD memory cards and microcontroller Atmel ATmega128. He describes their architecture, features, properties and technology used in devices. He is mentioning principle of communication protocols used by SD card, through that cards can communicate with other connected devices. He analyzes the functionality of FAT file system. He describes the design and implementation of interfaces to connect the SD cards to microcontroller. He explains software solutions of this project and gives impartial view to usage and comparisons of implemented modes of communication witch card.

Keywords

Memory card, SD, MMC, microcontroller, Atmel, AVR, ATmega128, SPI, CPRM, file system, FAT, fragmentation, JTAG.

Citace

Bc. Laurinc Pavel: Pripojenie pamäťovej karty SD k mikrokontroléru. Brno, 2007, diplomová práce, FIT VUT v Brně.

Pripojenie pamäťovej karty SD k mikrokontroléru

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Jaroslava Škarvadu.
Další informace mi poskytl Ing. Karel Radovský.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jméno Příjmení
Datum

Poděkování

Za ochotu a pomoc pri riešení diplomového projektu by som chcel prejaviť vďaku môjmu vedúcemu Ing. Jaroslavovi Škarvadovi, za cenné rady Ing. Karlovi Radkovskému a Bc. Tomášovi Filipovi.

Obsah

1 Úvod

Pamäťové karty Secure Digital (SD) sú jedným z typov tohto druhu pamäťového média ponúkaných na dnešnom trhu. Významnú konkurenciu predstavujú karty Memory Stick, Compact Flash a xD Picture card. Využitie pamäťových kariet je všestranné, najmä ako médium pre ukladanie digitálnych dát zvukového a vizuálneho charakteru. S veľkou obľubou sa stretávajú v digitálnych kamerách, fotoaparátoch, prenosných osobných prehrávačoch, mobilných telefónoch, hracích konzolách (Nintendo, GP2X, Sony PlayStation Portable), prenosných počítačoch atď.

Všetky spomínané dátové karty používajú spoločný typ internej pamäte, tzv. flash memory. Je to stála pamäť (energeticky nezávislá), ktorá môže byť elektrickým prúdom vymazaná a preprogramovaná. Tieto operácie sa vykonávajú nad blokom dát, čo prináša výhodu oproti niektorým typom pamätí, ktoré môžu zmazať iba celú oblasť (napr. EEPROM). Budúcnosť SD kariet, resp. ich rozhrania, vidí konzorcium, tvorené spoločnosťami SanDisk, Panasonic a Toshiba aj v inej oblasti. Jedná sa o periférne zariadenia ako televízne tunery, kamery, skenery a pod. Podrobnejší popis SD kariet, rozdelenia, ich architektúry a komunikačných protokolov bude uvedený v kapitole 2.1.

Pre realizáciu projektu som zvolil 8-bitový mikrokontrolér ATmega128L z rady AVR od firmy Atmel. Jeho univerzálnosť umožní pripojenie SD karty v oboch režimoch (SD a SPI). Špecifikáciu a súhrn dôležitých informácií uvediem v kapitole 2.2.

Väčšina pamäťových kariet používa pre uloženie dát súborový systém FAT. Z tohto dôvodu budem implementovať podporu pre čítanie, zápis a mazanie súborov a adresárov v novšej verzii FAT32, ale aj v staršej FAT16. Obzvlášť sa zameriam na prehľadný a jednoduchý prístup k súborovému systému s ohľadom na efektívne využitie vyrovnávacej pamäte. Princíp funkčnosti FAT vysvetlím v kapitole 2.3.

Diplomový projekt je postavený na základe semestrálneho projektu. Zaoberal sa naštudovaním a teoretickým spracovaním získaných znalostí v oblasti SD kariet a mikrokontroléru Atmega128 (kapitoly 2.2 a 2.3). Jeho súčasťou bolo aj vypracovanie prvotného návrhu pre pripojenie karty. Podrobná analýza, návrh a implementácia projektu sú súčasťou diplomovej práce a nájdeme ich v kapitole 3.

Zhodnotením prác na projekte sa zaoberá 4. kapitola. V nej sa zameriam na dosiahnuté výsledky a funkčnosť hardwarového rozhrania SPI a softwarových implementácií rozhrania SPI a SD.

V záverečnej kapitole zhrniem získané poznatky a poukážem na ďalší smer práce.

2 Teoretický úvod

Cieľom projektu je vytvoriť plnohodnotné rozhranie pre komunikáciu mikrokontroléru s pamäťovou kartou SD. Vyžitie nájde najmä v oblasti vstavaných systémov s potrebou ukladania získaných dát a jednoduchý prístup k nim. Momentálne sú bežné dostupné čítačky kariet pre osobné počítače. Projekty, ktoré sa zaoberajú tematikou pripojenia SD kariet k mikrokontroléru je niekoľko (napr. URL: http://cc5x.de/MMC/MMC_FAT.htm). Ale majú väčšinou obmedzenia, ako je využitie iba rozhrania SPI a implementácia súborového systému len pre čítanie a nie je k ním dostupná takmer žiadna dokumentácia. Náplňou tejto práce bude odstrániť spomínané nedostatky a pokúsiť sa o zhodnotenie, ktoré by pomohlo pri výbere technológií v ďalších projektoch.

V tejto kapitole uvediem teoreticky spracované detaily o SD kartách, mikrokontréru ATmega128 a súborovom systéme FAT.

2.1 Secure Digital

2.1.1 Úvod

SD karty boli prvýkrát predstavené v roku 1999. Na vývoji spoločne pracovali spoločnosti Matsushita Electric Industrial Co. (Panasonic), SanDisk a Toshiba. Cieľom bolo vytvoriť pamäťovú kartu schopnú konkurovať Memory Stick kartám od firmy Sony. Vyhovuje štandardu SDMI (Secure Digital Music Initiative), ktorý predstavuje vysokú ochranu proti nelegálnemu kopírovaniu hudby. SD karty sú založené na starších pamäťových kartách MMC (MultiMedia Card). Oproti predchodcovi majú radu výhod. Vodivé kontakty sú zapustené do povrchu puzdra, čím sú chránené pred dotykmi prstov. Obal je tvarovaný tak, aby nedošlo k opačnému vloženiu karty do adaptéru. SD karty obsahujú tiež mechanický prepínač, zaisťujúci ochranu proti prepísaniu dát na karte. Zvýšila sa aj prenosová rýchlosť (nové verzie SD kariet prinášajú ďalší nárast rýchlosti) a pribudlo už spomínané zabezpečenie proti porušovaniu autorských práv pri nelegálnom kopírovaní. Spätná kompatibilita je iba jednosmerná – staršie MMC karty sa môžu použiť v novších SD adaptéroch [2].

Prenosová rýchlosť SD kariet sa dnes pohybuje v rozmedzí 100 – 200Mbit/s v 4 bitovom móde a 25Mbit/s v 1 bitovom SPI (Serial Peripheral Interface) móde. Prvé karty sa vyrábali v kapacitách 32 a 64MB (maximálne 2GB). Nový formát SD 2.0, tzv. SDHC (Secure Digital High Capacity), definuje kapacity vyššie ako 2GB (maximálne 32GB), ktorý nie je priamo podporovaný staršími zariadeniami. Výhodou formátu SD je mnohonásobne vyššie využite ako u konkurencie. Rozhranie SDIO umožňuje pripojiť do SD adaptérov aj iné zariadenia ako je pamäťové médium: kamera, modem, skener, GPS prijímač, digitálny TV tuner, Bluetooth®, 802.11b atď.

2.1.2 Rozdelenie SD kariet

SD karty sa rozdeľujú do niekoľkých skupín (tried) podľa vlastností a funkcií, ktorými disponujú.

Podľa typu pamäte:

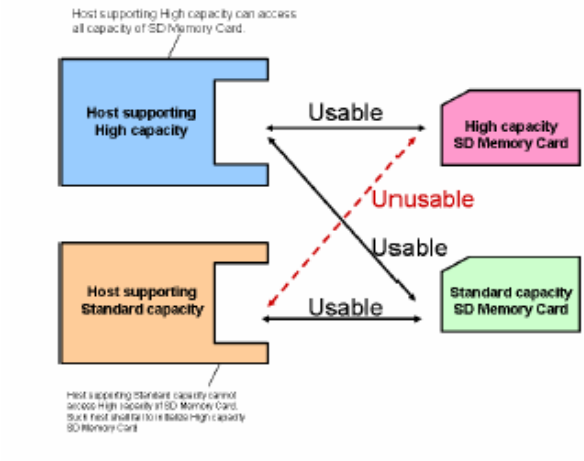
- Read/Write (RW) – používajú sa ako pamäťové média pre ukladanie osobných dát.
- Read Only Memory (ROM) – médium pre distribúciu softwaru, hudby, videa atď.

Podľa operačného napätia:

- High Voltage – rozsah 2.7V – 3.6V
- Dual Voltage – podporuje High Voltage a Low Voltage (T.B.D) – definícia až ďalšej verzii špecifikácie.

Podľa kapacity:

- Standard Capacity – do 2GB vrátane
- High Capacity (SDHC) – pamäť väčšia ako 2GB, ale maximálne 32GB (prichádza v špecifikácii SD 2.00)



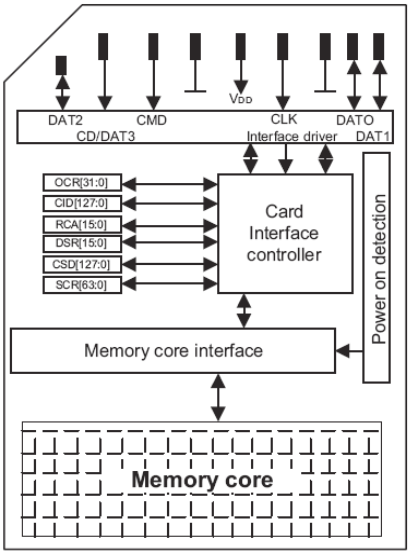
Obr 2-1 Kompatibilita adaptér – karta (Prevzaté z [3])

Podľa prenosových rýchlostí:

- Class 0 – staršie karty, nešpecifikujú rýchlosť
- Class 2 – garantovaná minimálna rýchlosť 2MB/s v najlepšom fragmentovom stave
- Class 4 – garantovaná minimálna rýchlosť 4MB/s v najlepšom fragmentovom stave
- Class 6 – garantovaná minimálna rýchlosť 6MB/s v najlepšom fragmentovom stave

2.1.3 Architektúra

Pamäťové karty SD sú pripojené k adaptéru deviatimi neizolovanými kontaktmi, tak ako je naznačené na Obr 2-2.



Obr 2-2 Architektúra SD (Prevzaté z [6])

Označenie jednotlivých pinov pre SD aj SPI mód ukazuje Tabuľka 2-1.

Tabuľka 2-1 Označenie pinov (Prevzaté z [4])

Pin No.	Name	Type[1]	Description
SD Mode			
1	CD/DAT3[2]	I/O[3], PP	Card detect/Data line [Bit 3]
2	CMD	I/O, PP	Command/Response
3	V_{SS1}	S	Supply voltage ground
4	V_{DD}	S	Supply voltage
5	CLK	I	Clock
6	V_{SS2}	S	Supply voltage ground
7	DAT0	I/O, PP	Data line [Bit 0]
8	DAT1	I/O, PP	Data line [Bit 1]
9	DAT2	I/O, PP	Data line [Bit 2]
SPI Mode			
1	CS	I	Chip Select (active low)
2	DataIn	I	Host-to-card Commands and Data
3	V_{SS1}	S	Supply voltage ground
4	V_{DD}	S	Supply voltage
5	CLK	I	Clock
6	V_{SS2}	S	Supply voltage ground
7	DataOut	O	Card-to-host Data and Status
8	RSV[4]	---	Reserved
9	RSV[5]	---	Reserved

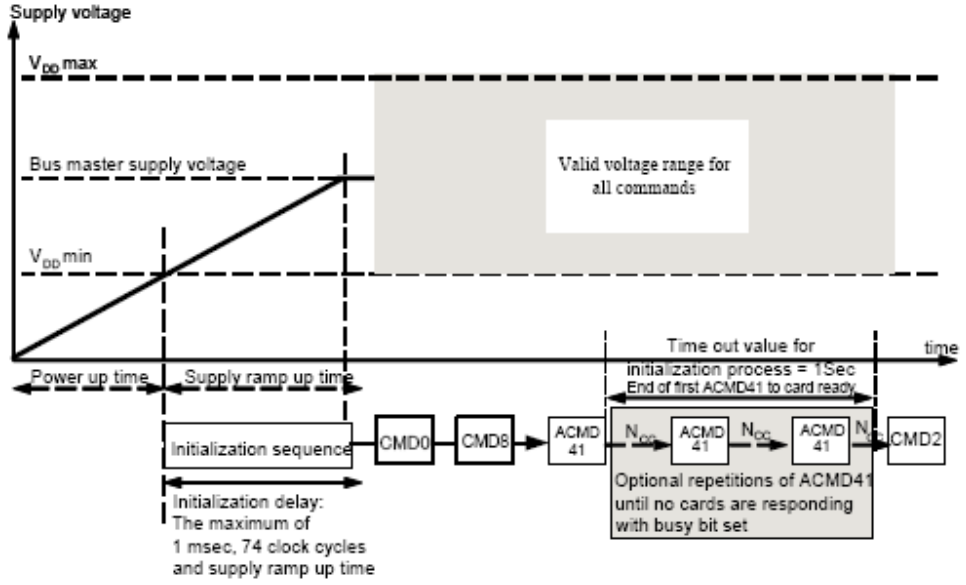
Karty obsahujú tzv. informačné registre, detailnejší popis v Tabuľke 2-2.

Tabuľka 2-2 Registre karty SD

Názov registru	Šírka v bitoch	Popis
CID	128	Card Identification Register. Obsahuje informácie potrebné pre identifikáciu karty.
OCR	32	Operation Conditions Register. Popisuje rozsah podporovaných napätí a stavový bit napájania.
CSD	128	Card Specific Data Register. Sprístupňuje informácie o prístupe k obsahu karty. Niektoré oblasti registru sú programovateľné príkazom PROGRAM_CSD (CMD27).
SCR	64	SD Card Configuration Register. Obsahuje informácie o špeciálnych funkciách danej karty. Tieto informácie ukladá výrobca.
RCA	16	Relative Card Address. Udáva adresu karty v SD režime.
DSR	16	Driver Stage Register. Použitie je voliteľné, pre zvýšenie rýchlosti zbernice.

SD karty podporujú dva typy prenosových režimov SD a SPI. Prv zmieňovaný je rýchlejší, pretože používa na prenos až 4 bity, naopak u SPI módu je používaný iba jeden bit pre prenos dát. Obidva režimy budú popísané v nasledujúcich kapitolách.

Výber protokolu a operačného napätia sa prevádza po pripojení karty k host adaptéru, prebieha tzv. identifikačný režim. Karta sa po pripojení nachádza v idle stave. Prijatie ako prvého príkazu CMD0 prepne kartu do SPI režimu. Nastavenie operačného napätia ukazuje diagram na Obr 2-3. Cieľom je vybrať napätie, ktoré je podporované zo strany host adaptéru aj karty. Obyčajne sa preferujú nižšie napätia kvôli menšej spotrebe energie. Verzia 2.0 SD špecifikácie definuje nový príkaz SEND_IF_COND (CMD8), verifikujúci operačné podmienky. Ako parameter príkazu sa predáva podporované napätie zo strany adaptéru. Ak je karta schopná pracovať s ponúkaným napätím, ako odpoveď pošle dané napätie, inak neodpovie a zostáva v idle stave.



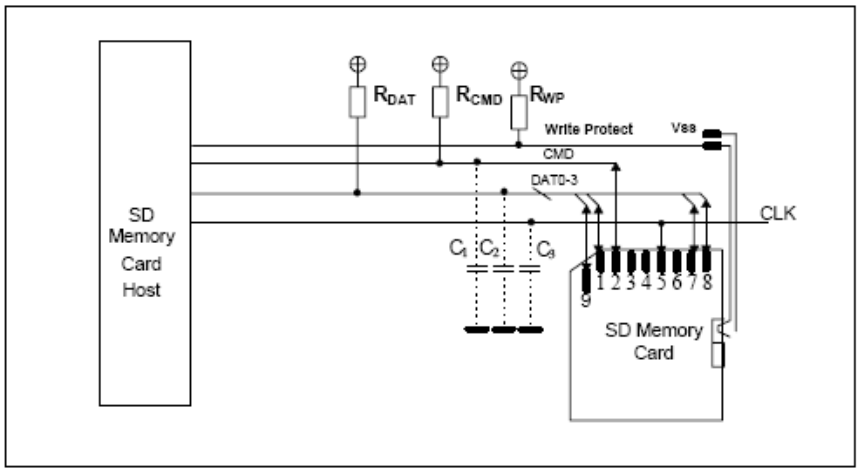
Obr 2-3 Identifikačný režim - nastavenie operačného napätia (Prevzaté z [3])

Ďalším príkazom je SD_SEND_OP_COND (ACMD41), ktorý umožní host adaptéru zistiť rozsah operačných napätí karty (ako parameter príkazu sa posielajú napätia podporované hostom). Ak karta nepodporuje napätie z daného rozsahu, prejde do neaktívneho (inactive) stavu. Inak odpovie v podobe poslaní dát z OCR registra. Pre karty SD verzie 2.0 je nevyhnutné pred poslaním ACMD41, najprv odoslať CMD8, aby bolo možné použiť nové funkcie (nízke operačné napätie a vysoko-kapacitné karty). Host adaptér nemôže zmeniť operačné napätie počas identifikácie. Pre zmenu napätia musí kartu najprv vypnúť a opäť zapnúť už s nastaveným novým operačným napätím.

2.1.4 SD režim

Topológia zbernice SD

SD karty majú 6 komunikačných kanálov a 3 napájacie linky (viď Obr 2-4). SD mód zbernice umožňuje dynamické prideľovanie dátových liniek. Po pripojení karty k host adaptéru je využitý pre prenos dát iba jeden bit DAT0. Až po inicializácii je možné využiť plnú prenosovú rýchlosť realizovanú v štyroch paralelných linkách DAT0-3. Schému pripojenia karty k adaptéru je znázornené na Obr 2-4.



Obr 2-4 SD režim - pripojenie karty k adaptéru (Prevzaté z [4])

Rezistory R_{CMD} a R_{DAT} slúžia k ochrane zbernice príkazových (CMD) a komunikačných (DAT) liniek bez pripojenej karty, príp. všetky ovládacie prvky sú v stave vysokej impedancie. Rezistor R_{WP} slúži pre prepínač ochrany proti zápisu (Write Protect switch).

K jednému adaptéru môžu byť zároveň pripojené viaceré karty (pri podpore zo strany adaptéru aj rôzne typy SD, MMC). Detekcia kariet je uskutočňovaná osobitne a pridelí každej karte logickú adresu. Dáta do kariet sú stále posielané individuálne, pre zjednodušenie sa používa po detekcii posielanie príkazov do všetkých kariet (rozlišovanie na základe adresy, posielanej v príkazovom pakete).

SD protokol

Komunikácia po zbernici SD prebieha posielaním príkazov, odpovedí a dát ako postupnosť bitov, začínajúcich štartovacím a končiacich stop bitom. Pred samotným prenosom dát medzi kartou a hosťujúcim adaptérom (host adaptér – master, riadi celú komunikáciu), je nutná identifikácia karty. SD host adaptér sa nachádza buď v stave identifikácie, alebo v stave prenosu dát (viď Obr 3-8).

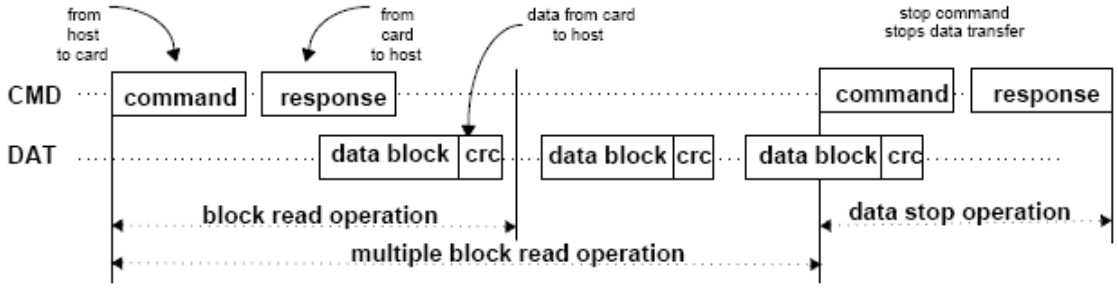
Identifikácia karty prebieha po resete, kedy master hľadá na zbernici pripojené nové karty (viď kap 2.3). V tomto stave ostávajú karty až do prijatia príkazu SET_RCA (CMD3). Stav prenosu dát nastaví adaptér po identifikácii všetkých kariet.

Tabuľka 2-3 Stav karty a režim host adaptéru

Stav karty	Operačný režim host adaptéru
Neaktívny	Neaktívny
Nečinný (idle), pripravený (Ready), identifikácia	Identifikačný režim
Pohotovostný (Standby), Prenos(Transfer), Posielanie dát(Send Data), prijímanie dát (Receive data), programovanie, odpojenie	Režim prenosu dát

Každú operáciu zahajuje príkaz poslaný konkrétnej karte, príp. všetkým pripojeným kartám. Odpoveď je posielaná následne po prijatí príkazu kartou, príp. kartami (synchrónne). Prenos príkazov aj odpovedí sa prevádza po sériovej CMD linke.

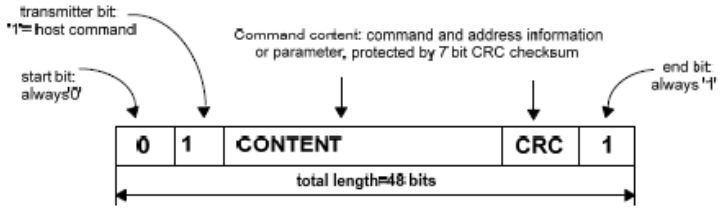
Dáta sú posielané po dátových linkách z host adaptéru do karty a naopak v blokoch dát. Dátové bloky sú vždy ošetrené CRC bitmi. Existujú dva typy operácií: blokové a mnoho blokové (multiple block), ktoré sú vhodné pre rýchly prenos dát. Typ prenosu vyberá host adaptér.



Obr 2-5 Multibloková operácia READ (Prevzaté z [3])

Pri zápise sa používa po poslaní dát čakací signál BUSY (na linke DAT0) na signalizáciu dokončenia operácie zápisu. Nezáleží pri tom na počte použitých dátových liniek.

Formát príkazového rámca znázorňuje Obr 2-6. Vždy začína štartovacím bitom s hodnotou 0 a končí bitom 1. Dĺžka rámcu je konštantná 48 bitov a každý rámec je chránený siedmimi CRC bitmi.

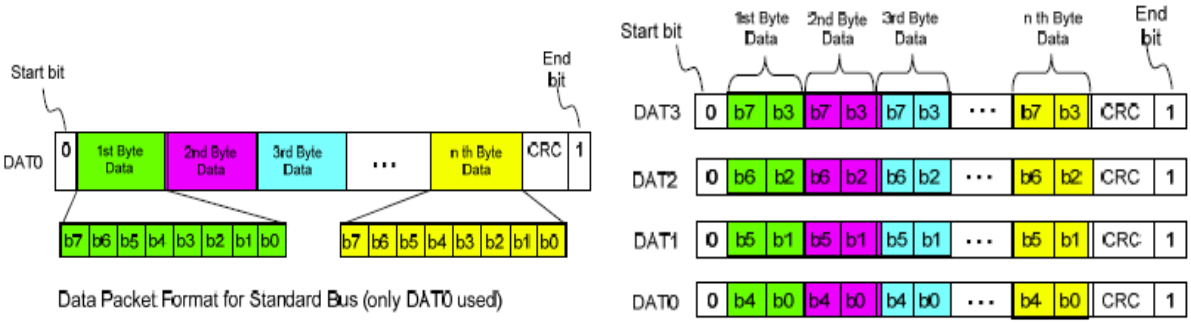


Obr 2-6 Formát príkazového rámca (Prevzaté z [3])

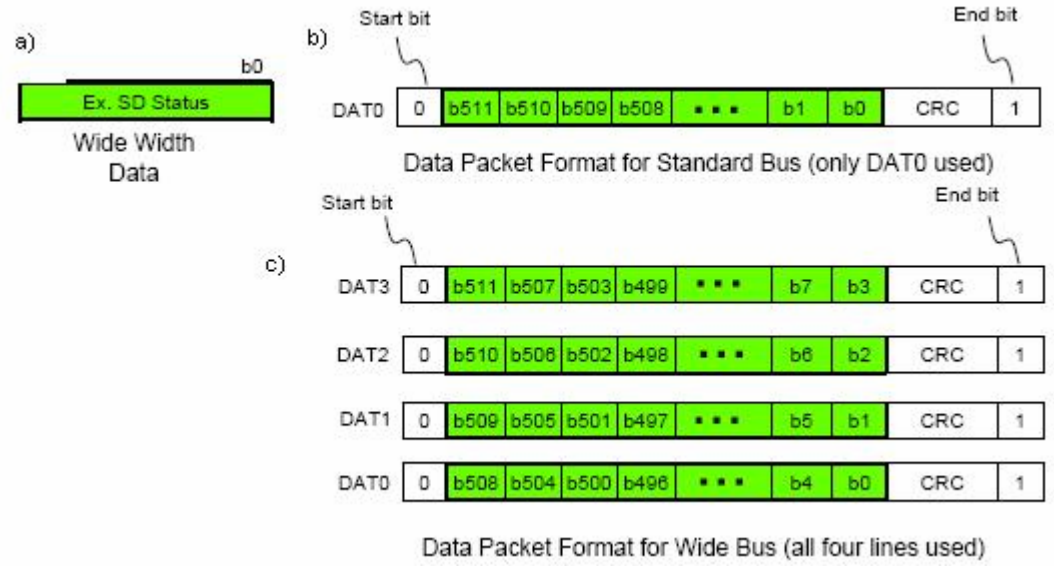
Formát rámca s odpoveďou má dve variácie: s dĺžkou 48 alebo 136 bitov.

Pre prenos dát sú dva typy paketov:

- Bežné dáta (Usual Data) – 8 bitov: najprv sa posiela menej významový bajt, ale v bajte sa posiela ako prvý viac významový bit. Orientácie bajtov a bitov vysvetľuje Obr 2-7a pre 1 bitový a Obr 2-7b pre 4 bitový prenos.
- Wide Width Data (SD Memory Register) – pre väčší počet bajtov, posielajú sa od najviac významového bajtu (aj bitu). Formát paketu je naznačený na Obr 2-8.



Obr 2-7 Formát paketu pre Usual Data: a) iba DAT0, b) DAT0-3 (Prevzaté z [3])



Obr 2-8 Formát paketu pre Wide Width Data (Prevzaté z [3])

2.1.5 SPI režim

SPI režim je druhým komunikačným protokolom (podmnožinou SD protokolu) podporovaným zo strany pamäťových kariet SD a niektorých mikrokontrolérov (Motorola, Atmel atď.). Režim sa vyberá v prvom príkaze po zapnutí a nemôže byť zmenený bez vypnutia. Samostatný SPI štandard nedefinuje kompletný protokol pre prenos dát, ale iba jeho fyzickú vrstvu. Preto implementácia v SD kartách používa ako základ protokol SD, aj s jeho príkazmi. Výhodou je jednoduchší návrh host adaptéru pre pripojenie karty, nevýhodou pomalšia prenosová rýchlosť (iba jedna dátová linka a osobitný Chip Select (CS) signál pre každú kartu).

Topológia SPI zbernice

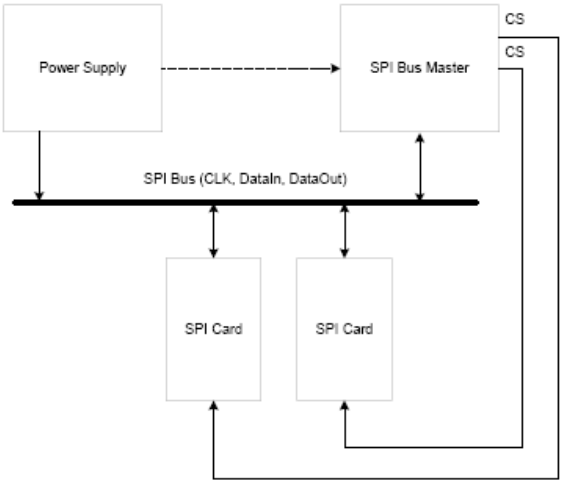
Rozhranie SPI používa pre komunikáciu štyri signály:

- CS (host to card Chip Select signal) – signál pre výber čipu (karty)
- CLK (host to card CLocK signal) – hodinový signál
- DataIn (host to card Data signal) – dátový signál z adaptéru do karty
- DataOut (card to host Data signal) – dátový signál z karty do adaptéru

Rozhranie SPI implementované v SD kartách používa prenos po bajtoch. Všetky rámce sú násobkom 8 bitových bajtov zarovnané na bajt. Adresovanie kariet prebieha na rozdiel od SD režimu hardwarovým signálom CS. Karta musí byť takto vybratá počas trvania každého príkazu, odpovedi a prenosu dát, výnimkou je len programovanie karty.

Ďalším rozdielom oproti SD módu sú jednosmerné linky DataIn a DataOut, ktoré nahradzujú obojsmerné CMD a DAT linky v SD režime. Táto skutočnosť zabraňuje vykonávaniu príkazov počas prenosu dát – nie je možný sekvenčný prenos.

Pripojenie SD kariet k zbernici SPI znázorňuje Obr 2-9.



Obr 2-9 Zbernica SPI: adaptér - SD karty (Prevzaté z [4])

SPI protokol

Komunikácia v protokole SPI sa skladá z príkazov, odpovedí a dátových rámcov. Celú komunikáciu kontroluje host adaptér (master) a každú transakciu začína nastavením signálu CS (na logickú 0). Vybratá karta vždy odpovedá na príkaz, pri problémoch s čítaním je to chybová odpoveď. Oproti SD režimu, ktorý to zistí až vypršaním času na odpoveď, je to výhoda. Na každý dátový blok poslaný počas operácie zápisu, sa reaguje dátovým odpovedajúcim rámcom.

Pri štandardnej kapacite karty je najnižšou možnou dátovou jednotkou jeden bajt a najväčšou jeden pamäťový blok karty. Parciálne blokové operácie čítania a zápisu sú v tomto režime povolené/nepovolené v závislosti na hodnote v CSD registre.

Vysokokapacitné SD karty (SDHC) majú danú pevnú dĺžku bloku na 512 bajtov. Veľkosť bloku je síce možné zmeniť príkazom CMD16, ale nepoužíva sa pre dátové prenosy (iba pre CMD42). To znamená, že parciálne blokové operácie (čítanie, zápis) nie sú povolené.

Výber režimu a inicializácia

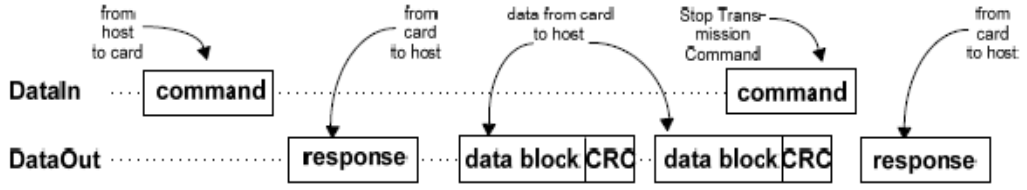
Po pripojení napájania ku karte, sa automaticky nastaví režim SD. Zmena režimu na SPI sa vykoná v prípade, že CS signál je nastavený (na logickú nulu) počas prijatia reset príkazu CMD0. Vtedy odpovie s SPI R1 odpoveďou. Ak karta rozozná, že sa má zachovať SD mód, neodpovedá na príkaz. Zmena režimu z SPI na SD je možná iba po vypnutí a následnom zapnutí (power cycle).
Na začiatku sa vyberie SPI mód. Potom nasleduje niekoľko rozhodovacích krokov, ktorými sa zistí o akú verziu karty sa jedná (príkaz CMD8), aké napätie podporuje (CMD58 – READ OCR) a či sa jedná o veľkokapacitnú pamäťovú kartu (ACMD41 a CMD58 – Card Capacity Status).

Prenos dát – čítanie, zápis a zmazanie

Každý príkaz je chránený CRC bitmi. Implicitne je CRC mód vypnutý (CRC OFF), v tom prípade sú CRC bity bezvýznamné. CRC mód sa zapína a vypína príkazom CRC_ON_OFF (CMD59) zo strany host adaptéru. Príkaz CMD8 má CRC verifikáciu stále zapnutú. V prípade detekcii chyby, karta pošle CRC chybu v R1 rámci.

SPI režim umožňuje čítanie po jednom, alebo viacerých blokoch. Veľkosť bloku sa nastavuje príkazom SET_BLOCKLEN (CMD16) pri štandardnej kapacite karty. U veľkokapacitných kariet je dĺžka bloku fixne nastavená na 512 bajtov, bez ohľadu na CMD16. Platný blok dát chráni 16 bitový CRC kód, generovaný podľa štandardu CCITT polynómom $x^{16} + x^{12} + x^5 + 1$.

V prípade, keď karta nemôže získať požadované dáta, pošle chybový dátový rámec.

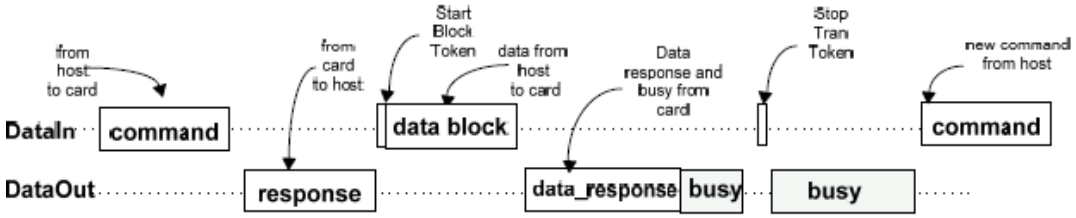


Obr 2-10 Multibloková operácia READ (Prevzaté z [3])

Na Obr 2-10 je schéma ukazujúca následnosť paketov pri multiblokovom čítaní z karty. Ukončenie prenosu dát sa vykoná príkazom CMD12.

Zápis v režime SPI je obdobná operácia ako čítanie. Každý dátový blok začína ‚Start Block' bajtom. Po prijatí dát od host adaptéru, karta odpovie data-response paketom. Počas programovania karta posiela po zbernici busy pakety. Po ukončení programovania host môže požiadať o výsledok zápisu pomocou príkazu SEND_STATUS (CMD13). Používa sa to pre prípad, že k chybe došlo až pri fyzickom zápise dát do pamäte karty (napr. ochrana proti zápisu, alebo adresa mimo platný rozsah). Pri multiblokovom zápise dát do karty sa ukončenie posielania dát vykonáva paketom ‚Stop Tran' namiesto ‚Start Block' bajtu na začiatku ďalšieho bloku. Ak dôjde počas zápisu k chybe, posiela sa oznámenie o chybe v data-response pakete. Vtedy je vhodné využiť príkaz SEND_NUM_WR_BLOCKS (ACMD22) na získanie počtu správne zapísaných blokov.

Karta počas programovania, keď posiela pakety BUSY, nepreruší programovanie ani reset CS signálu. Prerušenie programovania (čakajúceho aj aktívneho) sa docieli až resetovaním karty príkazom CMD0. Je to nebezpečné, kvôli možnému zničeniu dátovej štruktúry. Host adaptér by mal preto predchádzať resetu karty počas programovania.



Obr 2-11 Multibloková operácia WRITE (Prevzaté z [3])

Pri mazaní alebo menení bitov chránených proti zápisu z pred deklarovaného zoznamu sektorov, ostáva karta v stave BUSY a linku DataOut drží na úrovni logickej nuly.

Čítanie obsahu informačných CID a CSD registrov (viď Obr 2-4) sa v SPI režime vykonáva jednoduchou blokovou operáciou. Karta odpovedá štandardným ‚response' paketom a blokom dát.

Indikácia chýb je na rozdiel od SD režimu lepšia, pretože vždy je poslaná odpoveď na príkaz (akceptácia alebo zamietnutie príkazu). Zamietnutie príkazu môže nastať v týchto prípadoch:

- Je poslaný, keď karta vykonáva operáciu READ (výnimkou je CMD12 – ukončenie prenosu).
- Je poslaný, keď karta je zaneprázdnená – stav BUSY.
- Karta je zablokovaná a príkaz je inej triedy ako Class 0 alebo 7.
- Príkaz je nepodporovaný.

- CRC kontrola zlyhala.
- Obsahuje nepovolený operátor.
- Príkaz je mimo sekvenciu príkazov počas mazania.

Príkazy v SPI režime

Všetky príkazy v SPI režime majú nasledovný formát: dĺžku 6 bajtov a prvý sa posiela najvýznamnejší bajt.

		Byte 1		Byte 2-5		Byte 6	
7	6	5	0	31	0	7	0
0	1	Command		Command Argument		CRC	1

Obr 2-12 Formát paketu príkazu (Prevzaté z [3])

Zabezpečovací 7 bitový CRC kód sa vypočíta:

$$G(x) = x^7 + x^3 + 1$$
$$M(x) = (startbit) * x^{39} + (hostbit) * x^{38} + ... + (lastbit) * x^0$$
$$CRC[6..0] = \left(M(x) * x^7\right) \bmod G(x)$$

Príkazy v SPI režime sú rozdelené to tried (viď Obr 2-13), každá trieda podporuje určitú skupinu funkcií. Tieto sú rozdielne pre SD a SPI režim.

CCC		0	1	6	8	10	12	13	16	17	18	24	25	27	28	29	30	32	33	34	35	36	37	38	42	50	52	53	55	56	57	58	59
0	Basic	+	+		+	+	+	+																								+	+
1	NS																																
2	Block read								+	+	+																						
3	NS																																
4	Block write								+			+	+	+																			
5	Erase																	+	+					+									
6	Write-protect														+	+	+																
7	Lock card								+																+								
8	App-specific																												+	+			
9	I/O mode																										+	+					
10	Switch			+																+	+	+	+		+					+		
11	R																																

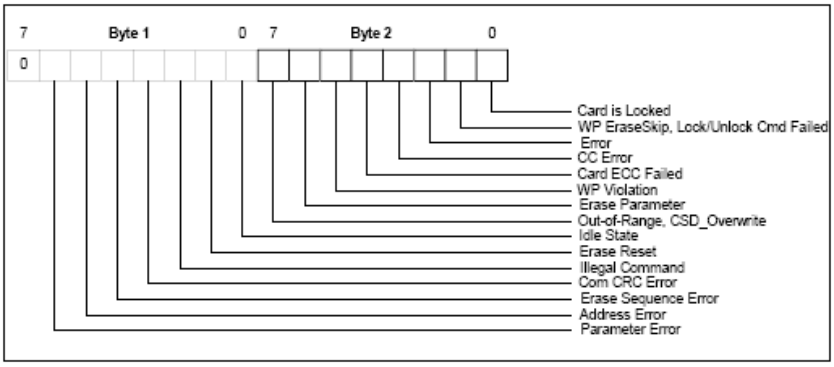
Key: NS = Not supported in SPI mode.
R = Reserved

Obr 2-13 Triedy príkazov v SPI (Prevzaté z [4])

Detailný popis jednotlivých príkazov nájdeme v špecifikácii SD [3]. Binárne kódy príkazov symbolicky vychádzajú z poradového čísla príkazu, napr. pre CMD0 to je ‚000 000', alebo pre CMD39 ‚100111'.

Existuje niekoľko typov odpovedí: R1, R1b, R2, R3, R4, R5 a data-response. Formát R1 je posielaný vždy okrem príkazu SEND_STATUS, prvý MSB bit je 0, ostatné indikujú výskyt určitej chyby (viď Obr 2-14 Byte 1). R1b rámec je rozšírený R1 o BUSY signál na ľubovoľnom počte bajtov, indikujúci zaneprázdnenie karty (logická 0) alebo pripravenosť na prijatie ďalšieho príkazu (logická 1).

Rámec R2 je dlhý 2 bajty posiela stav ako odpoveď na príkaz SEND_STATUS. Prvý bajt je zhodný s rámcom R1, bližší popis na Obr 2-14.



Obr 2-14 Formát R2 (Prevzaté z [4])

R3 rámec odpovedá na príkaz READ_OCR. Je dlhý 5 bajtov a prvý bajt je R1, ostatné bajty predstavujú obsah registra OCR.

R4 a R5 odpovede sú rezervované pre SDIO zariadenia.

Data-response je potvrdzujúci paket oznamujúci prijatie dát na zápis. Jeho dĺžka je jeden bajt a formát znázorňuje Obr 2-15.

7	6				0
x	x	x	0	Status	1

Obr 2-15 Data-response (Prevzaté z [4])

Stavové bity predstavujú:

- 010 – dáta akceptované
- 101 – dáta zamietnuté pre CRC chybu
- 110 – dáta zamietnuté kvôli chybe pri zápise

Pri výskyte chyby v multiblokovom zápise, môže host adaptér prerušiť prenos dát príkazom CMD12. Potom je možné zistiť chyby v zápise príkazom CMD13 (SEND_STATUS), alebo počet správne zapísaných bajtov (ACMD22).

2.1.6 Zabezpečenie dát proti nelegálnemu rozširovaniu v SD kartách

Content Protection for Recordable Media (CPRM) je technológiou pre zabezpečenie dát, chránených autorskými právami. Technológia bola navrhnutá spoločnosťou 4C (The digital contents copyright protection technology licensing organiztion of IBM, Intel, Matsushita a Toshiba) [5].

Pre ochranu použili technológiu (rozšírenú z DVD) „key revocation“ – odmietnutie kľúča, ktorá je zabudovaná do SD kariet. Riadiace obvody v karte umožňujú čítanie a zápis v zabezpečenej zóne až po autentizácii pripojeného zariadenia. Kopírovanie z počítača do pamäte SD karty

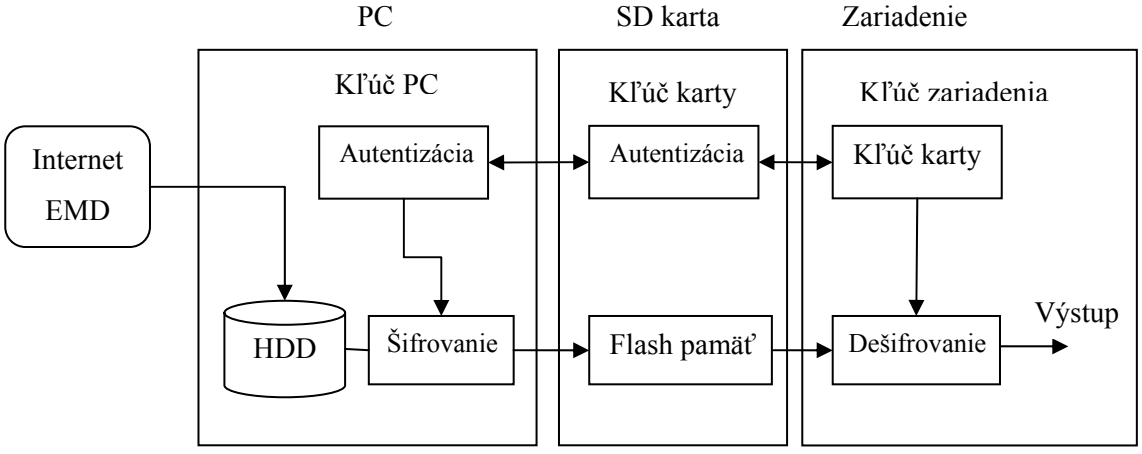
obmedzené na 3 kópie tak, ako to vyplýva zo špecifikácie SDMI (Secure Digital Music Initiative – všetky SD audio produkty ju akceptujú).

Ochrana proti porušovaniu autorských práv neoprávneným kopírovaním v SD kartách zahŕňa tieto funkcie:

- Prístup do zabezpečenej oblasti musí byť povolený až po autentizácii medzi zariadeniami
- Zaistenie vygenerovania náhodného čísla pri každej autentizácii a výmene dôverných informácií

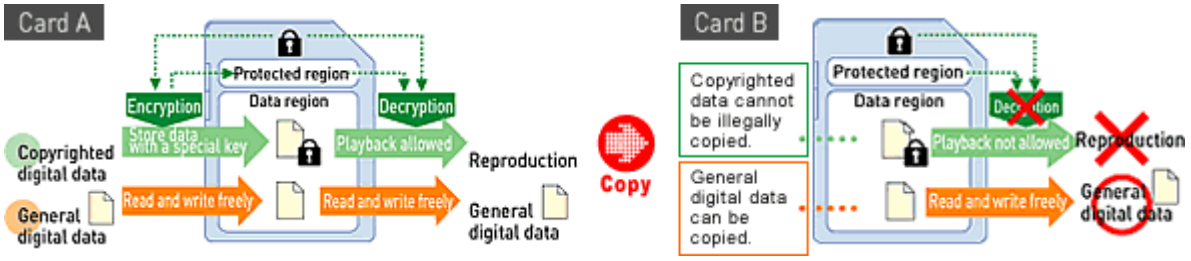
Nasledovný príklad ukazuje postup akým sa dáta uložia v chránenej oblasti pamäťovej karty a potom prečítajú v hudobnom prehrávači:

1. Hudobná skladba získaná z internetu (Electronic Music Distribution – EMD) sa uloží do počítača už v kryptovanej podobe (sieťové kryptovanie).
2. Pred akoukoľvek operáciou s SD kartou v chránenej oblasti je nutná autentizácia oboch strán. Najprv počítač legitimuje kartu a karta overí dôveryhodnosť počítačového programu. Po skončení autentizácie sú dáta kryptované v súlade s unikátnym kľúčom SD karty a kľúčom prideleným obsahu hudobných dát.
3. Súčasne sa kryptované dáta uložia v pamäti karty.
4. Pri prehrávaní hudby z šifrovanej oblasti SD karty je tiež nutná autentizácia oboch zariadení, podobne ako pri ukladaní dát na kartu. Hudobný prehrávač preverí kartu a naopak. Až potom získa prehrávač potrebný kľúč pre dešifrovanie dát.
5. Prehrávač číta kryptovaný obsah a pomocou získaného kľúča ho rozkóduje a prehrá.



Obr 2-16 Príklad použitia šifrovanej oblasti: PC – SD karta – Prehrávač

Okrem oblasti v karte, kde je uložený kľúč, karta obsahuje aj chránenú oblasť, ku ktorej sa dá pristupovať až po autentizácii. Hlavné zašifrované dáta sú síce voľne kopírovateľné, ale potrebné informácie pre dešifrovanie sú uložené práve v chránenej oblasti. Kvôli tomu nie je možné kopírovať napr. hudobné skladby z jednej karty na druhú. Dáta sú síce skopírované, ale chýba kód pre dešifrovanie (viď Obr 2-17).



Obr 2-17 Kopírovanie chránených dát na inú kartu (Prevzaté z [1]).

Pri prelomení ochrany (tzv. hack) karty a odhalení prístupu do zabezpečenej oblasti, systém „key revocation“ urobí kľúč zariadenia, ktoré sa pokúsilo o neoprávnený prístup

2.2 Atmel ATmega128

Mikrokontrolér rady ATmega128 od výrobcu Atmel je 8 bitový s CMOS technológiou, založený na AVR zdokonalenej RISC architektúre. Keďže dokáže vykonávať zložité inštrukcie v dobe jedného hodinového cyklu, dosahuje vysokej priepustnosti až 1 MIPS na jeden MHz. Mikrokontrolér má nízky príkon a vývojárom umožňuje reguláciu spotreby znížením výkonu.

Špeciálne AVR jadro disponuje bohatou inštrukčnou súpravou s 32 pracovnými registrami. Každý z nich je priamo pripojený na aritmeticko-logickú jednotku (ALU), ktorá zvládne prístup k dvom nezávislým registrom pri vykonávaní jednej inštrukcie v jednom hodinovom cykle. Takáto architektúra je efektívnejšia a môže prevýšiť výkon CISC architektúry až desaťnásobne.

Mikrokontrolér ATmega128 má 128KB internej flash pamäte so schopnosťou súčasného čítania počas zápisu. Taktiež dovoľuje dnes veľmi využívanú funkciu programovania v už zapojenom obvode, tzv. In-System Programming. Ďalšími parametrami ATmega128 sú: 4KB EEPROM a 4KB SRAM pamäte, 53 I/O liniek, 32 registrov, počítadlo reálneho času (RTC), SPI sériový port, IEEE atď. Umožňuje aj testovanie programu a programovanie za behu, t.j. v zapojenom obvode (On-Chip Debugging and programming). Programovo je možné vybrať až 6 rôznych úsporných režimov:

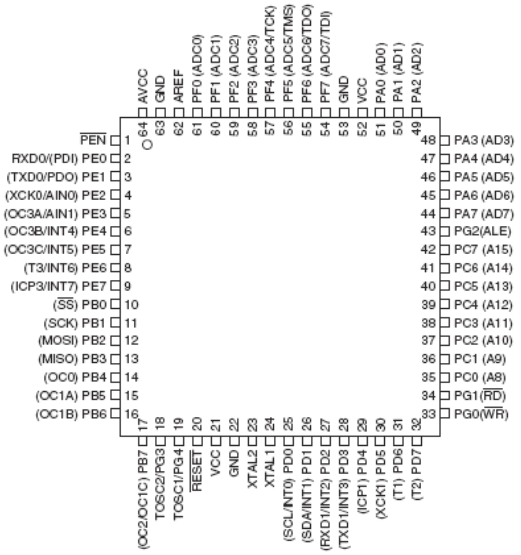
- Idle – nečinný, zastaví procesor – povolí SRAM, časovačom/počítadlám, SPI portu a systému pre správu prerušení.
- Power-down – v stave nízkeho príkonu sa síce zastaví oscilátor, ale hodnoty v registroch sa nestratia. Zmenu stavu vykoná iba ďalšie prerušenie alebo hardwarový reset.
- Power-safe – v úspornom režime asynchrónny časovač pokračuje, ostatná časť systému je pozastavená.
- ADC Noise Reduction – zastaví procesor, všetky I/O moduly (okrem asynchrónneho časovača a ADC) kvôli minimalizovaniu šumu počas ADC konverzií.

- Standby – pohotovostný režim, oscilátor beží, ostatné súčasti sú vypnuté, ale pripravené na rýchly štart.
- Extended Standby – rozšírený pohotovostný režim, oscilátor aj asynchrónny časovač beží.

Mikrokontrolér Atmel ATmega128 používa vlastnú technológiu permanentnej pamäte s vysokou hustotou dát. Funkcia ISP umožňuje preprogramovanie mikrokontroléru vo výslednom systéme cez SPI rozhranie, bežným programátorom permanentných pamätí alebo pomocou boot programu spustenom na AVR jadre. Boot program môže použiť ľubovoľné rozhranie pre nahratie aplikácii do internej flash pamäte.

2.2.1 Architektúra – popis vývodov

V tejto podkapitole vysvetlím význam jednotlivých vývodov mikrokontroléru podľa Obr 2-18 v Tabuľke 2-4.



Obr 2-18 Vývody mikrokontroléru ATmega128 (prevzaté z [7])

Tabuľka 2-4 Význam pinov ATmega128

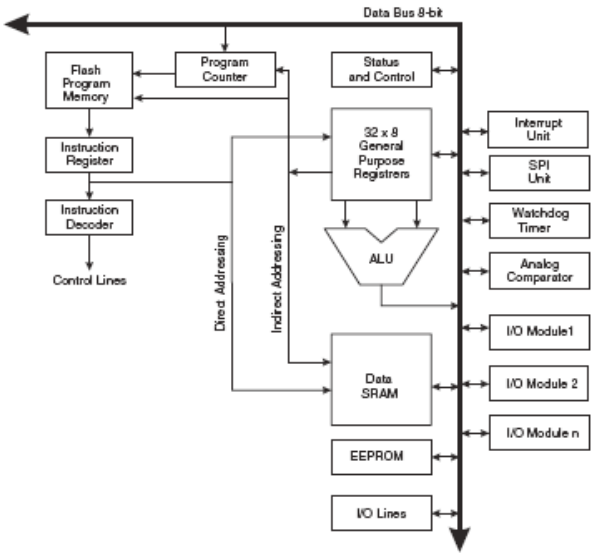
Názov	Popis
VCC	Napájacie napätie
GND	Zem
Port A (PA7-0)	8 bitový obojsmerný I/O port s internými pull-up rezistormi (zapojených voči zemi)
Port B (PB7-0)	8 bitový obojsmerný I/O port s internými pull-up rezistormi
Port C (PC7-0)	8 bitový obojsmerný I/O port s internými pull-up rezistormi
Port D (PD7-0)	8 bitový obojsmerný I/O port s internými pull-up rezistormi
Port E (PE7-0)	8 bitový obojsmerný I/O port s internými pull-up rezistormi
Port F (PF7-0)	Slúži ako analógový vstup pre A/D konvertor, ako 8 bitový obojsmerný I/O port s internými pull-up rezistormi, aj pre funkcie JTAG rozhrania
Port G (PG4-0)	5 bitový obojsmerný I/O port s internými pull-up rezistormi, slúži pre rôzne špecifické funkcie

RESET	Logická nula na tomto vstupe po dobu minimálnej dĺžky pulzu spôsobí reset. Kratšie pulzy nemusia zaručiť reset.
XTAL1	Vstup na invertujúci oscilátor a vstup pre obvod interného hodinového signálu.
XTAL2	Výstup z invertujúceho oscilátora.
AVCC	Napájacie napätie pre port F a A/D prevodník.
AREF	Analógový referenčný vývod pre A/D prevodník
PEN	Povoľuje programovanie v SPI sériovom programovacom režime. Interne je nastavený na logickú 1. Privedením logickej 0 na tento pin počas resetu, zariadenie prejde do programovacieho režimu.

2.2.2 Jadro procesoru AVR

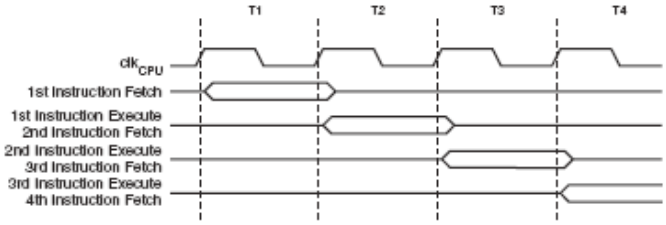
AVR je rada RISC mikrokontrolérov od fimry Atmel. Internú štruktúru navrhli študenti Alf-Egil Bogen a Vegard Wollan na Nórskom technologickom inštitúte (Norwegian Institute of Technology). V dnešnej dobe už existujú 32 bitové AVR mikrokontroléry, ktoré dokážu spracovávať aj zvuk a video pomocou nových prídavných funkcií.

Pre zvýšenie výkonu a paralelizmu AVR používa Harvard architektúru. Vyznačuje sa osobitným pamäťovým systémom s vlastnými zbernicami pre program a dáta. Inštrukcie programu sú vykonávané s jednoduchým zreťazením (Single Level Pipelining). Zatiaľ čo sa jedna inštrukcia spracúva, druhá sa rozpracúva tak, ako to zobrazuje Obr 2-20.



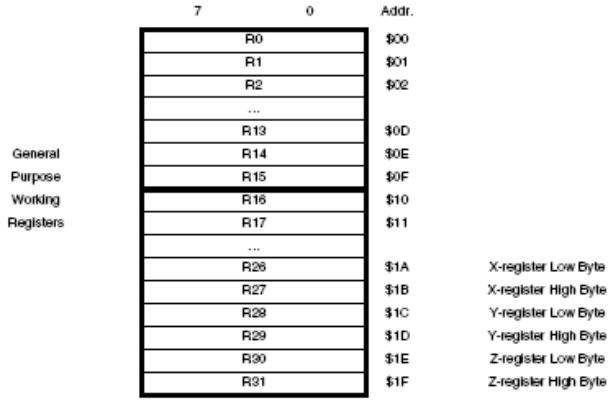
Obr 2-19 Architektúra AVR procesora (Prevzaté z [7]).

Programovú pamäť tvorí flash pamäť podporujúca In-System Reprogramming (ISP).



Obr 2-20 Paralelné spracovanie inštrukcií (Prevzaté z [7]).

Registre sú 8-bitové a rýchlo-prístupové, každý z nich je prístupný v osobitnom čase hodinového cyklu. Preto aritmeticko-logická jednotka zvládne vykonanie jednej operácie za jeden takt. Šesť z týchto registrov je pripravených pre využitie ako tri 16-bitové registre pre priame adresovanie v dátovom adresnom priestore. Jeden z nich je využiteľný aj ako adresný ukazovateľ pre vyhľadávaciu tabuľku programovej pamäte. Spomínané tri 16-bitové registre sa nazývajú tiež X-register, Y-register a Z-register.



Obr 2-21 Registre AVR procesoru (Prevzaté z [7]).

ALU podporuje aritmetické a logické operácie medzi registrami alebo medzi konštantou a registrom. Po aritmetických operáciách sa obnoví obsah ‚status' registra, podávajúceho informácie o výsledku operácie.

Beh programu môže byť ovplyvnený podmienenými a nepodmienenými skokmi (jump) a volaniami (call) s adresovaním celého adresného priestoru. Veľa inštrukcií má jednoduchý 16-bitový formát a na každej adrese v programovej pamäti je uložená 16 alebo 32-bitová inštrukcia.

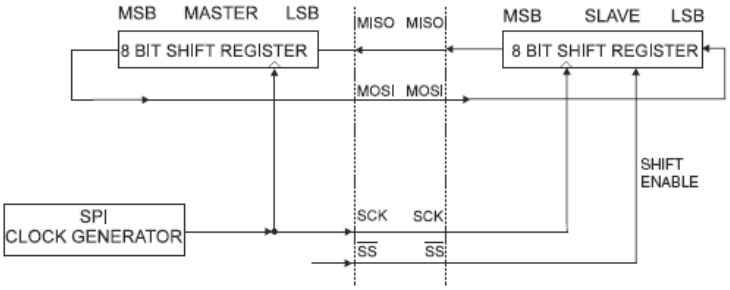
Kvôli už spomínanému preprogramovaniu aplikácie je nutné rozdelenie programovej pamäte na dva priestory: zavádzací (boot) a aplikačný. Obidve sekcie majú pridelené blokovacie bity pre zabezpečenie ochrany proti prepisu, alebo čítaniu. Inštrukcie, ktoré zapisujú do aplikačnej oblasti programovej pamäte musia byť uložené v boot časti programovej pamäte.

Pri prerušeniach a skokoch do podprogramov, sa návratová adresa inštrukčného ukazovateľa (Program Counter – PC) ukladá štandardne na zásobník. Zásobník je namapovaný do dátovej pamäte SRAM a obmedzený iba veľkosťou SRAM. Modul systému prerušení má kontrolné registre v I/O priestore. Pre získanie adresy obslužného podprogramu prerušenia využíva tabuľku vektorov prerušení. Priorita prerušení plynie z indexu v tabuľke: čím nižší index, tým vyššia priorita. Vstupno-výstupný pamäťový systém (I/O) obsahuje 64 adries priamo prístupných. ATmega128 prináša aj rozšírený I/O priestor v pamäti SRAM.

2.2.3 Serial Peripheral Interface – SPI

SPI rozhranie umožňuje vysoko-rýchlostné synchrónne dátové prenosy medzi mikrokontrolérom a periférnym zariadením, alebo medzi dvoma mikrokontrolérmi. Významné funkcie ATmega128

v SPI režime sú: trojlinkový synchrónny dátový prenos, riadenia masterom alebo slaveom, 7 programovateľných prenosových rýchlostí, dvojnásobná rýchlosť v Master SPI režime.



Obr 2-22 Prepojenie Master - Slave cez SPI rozhranie (Prevzaté z [7]).

Na Obr 2-22 je zapojenie master a slave zariadení cez SPI rozhranie. Obidve zaradenia majú 8-bitový posuvný register, v ktorom pripravia dáta pre odoslanie. Komunikáciu riadi master generovaním hodinového signálu (SCK). Dáta z registrov sú postupne posielané opačnej strane, až kým sa nepošle celý bajt. Po poslaní paketu master synchronizuje slave uvedením signálu SS (Slave Select) na logickú 1.

V konfigurácii mikrokontroléru ako master, musí najprv program zaručiť riadenie signálu SS. Potom sa zapíše bajt dát do SPDR (SPI Data Register) a pošle slave zariadeniu. Po poslaní bajtu sa zastaví generátor hodinového taktu a nastaví sa príznak ukončenia transakcie (SPIF) v SPI Status Registri. Ak je nastavený SPIE (SPI Interrupt Enable) bit v SPCR (SPI Control Register), bolo vyvolané prerušenie. Master potom môže pokračovať posielaním ďalšieho bajtu, alebo oznámiť koniec paketu – nastaví SS.

Ak je konfigurovaný ako slave zariadenie, musí počkať kým sa SS nezmení na logickú 0. Následne sa môžu poslať dáta z SPDR. Po jednom bajte sa nastaví príznak ukončenia transakcie a po nastavení SPIE bitu je opäť možné posielať nové dáta.

2.3 File Allocation Table (FAT)

Súborový systém FAT čiastočne spadá pod patent firmy Microsoft. Bol vyvinutý pre operačný systém MS-DOS a ďalej používaný ako primárny systém na uloženie súborov v MS Windows až do verzie Me [14]. Pôvodný zámer vývojárov však siahal najprv k pružným diskom (disketám) o veľkosti menšej ako 500KB [15].

Najväčšia výhoda oproti ostatným systémom (ďalej OS) vyplýva z veľmi dobrej podpory zo strany operačných systémov. Preto používanie FAT na disketách, prenosných diskoch, pamäťových kartách atď. nikoho neprekvapí. Osobne som kedysi používal FAT na zdieľanie dát medzi rôznymi OS. Jednou z výhod je aj relatívne jednoduchá implementácia, čo by sa mohlo využiť aj v oblasti vstavaných systémov. Realizáciu z tohto súdka ponúkne jedna z úloh tohto projektu – naprogramovať podporu pre čítanie/zápis/vymazanie súborov z SD karty.

Typy FAT

Ako väčšina technológií spojených a obmedzených maximálnou veľkosťou pamäte, aj FAT systém musel napredovať a celkovo vyprodukovali tri verzie: FAT12, FAT16 a FAT32. Niektoré dôležité informácie o limitoch jednotlivých verzií nájdeme v Tabuľke 2-5.

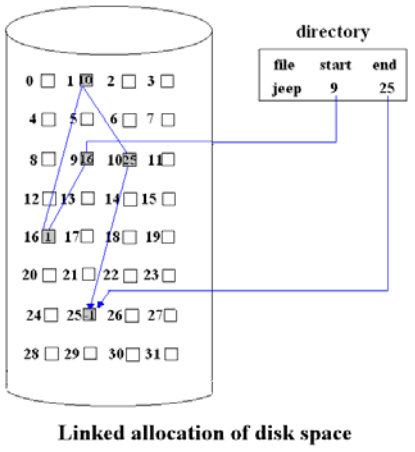
Tabuľka 2-5 Vlastnosti FAT [14]

Vlastnosť	FAT12	FAT16	FAT32
Dátum vydania	1977 (MS DISK BASIC)	November 1987 (Compaq DOS 3.31)	August 1996 (Windows 95 OSR2)
Max veľkosť zväzku	32 MiB	2GiB (4GiB – 64KB clstr)	8 TiB
Max veľkosť súboru	32 MiB	2 GiB	4 GiB
Max počet súborov	4 077	65 517	268 435 437
Max dĺžka názvu	8 + 3 znaky prípona, 255 znakov pri Long File Name (LFN)		
Rozsah dátumov	1. Január 1980 – 31. December 2107		

Významný rozdiel v typoch FAT predstavuje počet bitov vyhradených pre umiestnenie jedného záznamu na disku. Od tejto veľkosti sa odvodzuje aj názov typu FAT. Platí, že čím väčšia je šírka, tým viac záznamov umožní adresovať – preto sa tak výrazne líšia maximálne veľkosti partícií v Tabuľke 2-5. Posledná verzia FAT32 už dosahuje slušné veľkosti zväzkov. Ale nevýhodou je hranica maximálnej veľkosti súboru, ktorá v dnešnej dobe nepostačuje.

Fragmentácia

Za najväčší nedostatok však môžeme pokladať fragmentáciu súborov. Znamená to, že súbor nie je uložený na disku ako ucelený blok, ale rozdelený na niekoľko častí – fragmentov (viď Obr 2-23). Z hľadiska nesúvislého priestorového umiestnenia to nevadí, pretože v reťazovom rozpoložení každý záznam uchováva adresu na ďalší. Problémy prichádzajú zo sekvenčným čítaním/zápisom súborov. Vtedy dochádza k spomaleniu pamäťového média neustálym hľadaním nasledujúceho clusteru.

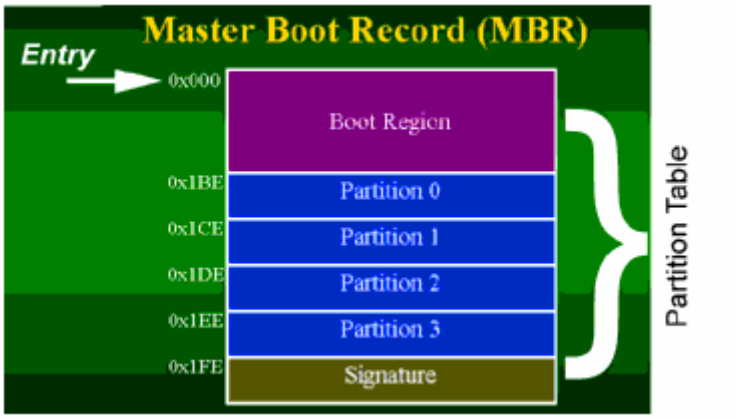


Obr 2-23 Reťazové uloženie súboru – Fragmentácia (Prevzaté z [16])

Vzniká pri mazaní a opätovnom zápise. Na jej elimináciu (usporiadanie do súvislých celkov) do existuje niekoľko programov, napr. Disk Defragmentor, ktorý je súčasťou OS Windows.

2.3.1 Master Boot Record (MBR)

MBR nachádzame na začiatku fyzického disku, príp. iného pamäťového média. Jeho veľkosť je obyčajne jeden sektor, ale môže ich byť aj viac. Obr 2-24 ukazuje logické zloženie MBR. Prvý element tvorí bootovacia oblasť, vytvorená pri formátovaní média. Z nej začína zvyčajne štart PC, obsahuje programový kód zavádzača. Za ním nasledujú štyri 16 bajtové záznamy o partíciách.



Obr 2-24 MBR - logická stavba (prevzaté z [17])

Každý záznam obsahuje informácie o schopnosti bootovania, adresa začiatku a koncu zväzku na médiu a počet sektorov v partícii. Podrobnejšie zloženie nájdeme v Tabuľke 2-6.

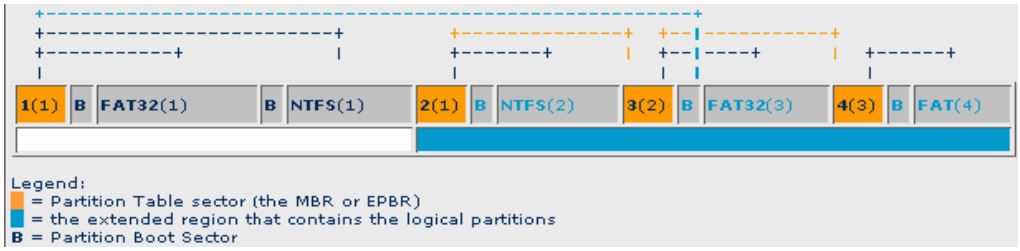
Tabuľka 2-6 Záznam o partícii [18]

Počet bajtov	Príklad hodnôt	Popis
1	0x80	Bootovací príznak: 0x00 – nepovolené, 0x80 - povolené
3	0x00 0x02 0x00	Adresa štartovacej hlavy, cylindra a sektoru
1	0x07	Popisovač súborového systému
3	0x1A 0x5B 0x8C	Adresa poslednej hlavy, cylindra a sektoru v partícii
4	0x02 0x00 0x00 0x00	Adresa prvého sektoru (relatívna k počiatku disku)
4	0x00 0x35 0x0C 0x00	Počet sektorov v partícii

Boot príznak povoľuje zavedenie systému zo zväzku. Ak má hodnotu 0x80, t.j. umožní bootovanie, hovoríme o aktívnej partícii. Na jednom fyzickom disku môže byť súčasne iba jedna aktívna partícia. Pri štarte PC najprv BIOS načíta do pamäte MBR, spustí ho a zistí aktívnu partíciu. Následne nahrá zavádzač štartovací sektor vybrané zväzku a vykoná jeho inštrukcie (zvyčajne to je inštrukcia skoku na zavádzací súbor OS). Niekedy, najmä keď má počítač nainštalovaných viac operačných systémov, môžeme nájsť v MBR aj zložitejší zavádzač. Jeho hlavnou vymoženosťou

býva výber partície, príp. aj iného média (napr. CD, DVD, disketa), z ktorej štartuje OS (napr. Smart Boot Manager, voľne dostupný na URL: <http://www.linuxsoft.cz/en/sw_detail.php?id_item=740>).

Z Obr 2-24 je jasné, že na disku teraz viac ako 4 partície nemôžu byť. Limitáciu vyrieši vytvorenie tzv. extended partície, pretože v rámci nej smieme umiestniť ďalších 24 logických partícií. Extended partícia má vlastnú tabuľku partícií pre logické disky. Ukážkové rozdelenie disku vrátane logických partícií demonštruje Obr 2-25.



Obr 2-25 Príklad rozdelenia disku (Prevzaté z [17])

2.3.2 Štruktúra FAT partície

Každý zväzok s FAT systémom vykazuje štruktúru znázornenú v Tabuľke 2-7.

Tabuľka 2-7 Štruktúra FAT partície

Boot Sector	Reserved sectors	File Allocation Table č. 1	File Allocation Table č. 2	Root dir (iba FAT12/16)	Dáta

Prvý (boot) sektor obsahuje „BIOS parameter block“ (BPB). BPB okrem iných informácií o partícii bežne uchováva aj vyššie zmienený zavádzač OS. Po boot sektore voliteľne nasleduje niekoľko rezervovaných sektorov, ich počet udáva položka Reserved Sector Count v BPB. Dôležitými časťami sú dve kópie alokačných tabuliek súborov (File Allocation Table). Root adresár nachádzame v tesnom uložení za kópiami tabuliek u typov FAT12 a FAT16. FAT32 má svoj koreňový adresár v dátovej časti, jeho adresu prechováva rozširujúca štruktúra (viď Tabuľka X).

Boot Sektor a BPB

Podrobné informácie o štruktúre uvediem v Tabuľke 2-8, dĺžka aj ofset sú v bajtoch. Veľkosť tohto bloku je 36 bajtov.

Tabuľka 2-8 Štruktúra Boot sektoru a BPB [15]

Názov	Ofset	Dĺžka	Popis
BS_jmpBoot	0	3	Inštrukcia skoku (jmp) na adresu zavádzača OS
BS_OEMName	3	8	Názov, obvykle MSDOS5.0 alebo MSWIN4.1
BPB_BytsPerSec	11	2	Počet bajtov na sektor
BPB_SecPerClus	13	1	Počet sektorov v jednom clusteri
BPB_RsvdSecCnt	14	2	Počet rezervovaných sektorov
BPB_NumFATs	16	1	Počet tabuliek FAT, zvyčajne 2

BPB_RootEntCnt	17	2	Maximálny počet záznamov v koreňovom adresári. Pre FAT32=0
BPB_TotSec16	19	2	Celkový počet sektorov. Pre FAT32 = 0
BPB_Media	21	1	0xF8 pre neprenositeľné a 0xF0 pre prenositeľné médiá
BPB_FATSz16	22	2	Veľkosť tabuľky FAT v sektoroch. Pre FAT32 = 0
BPB_SecPerTrk	24	2	Počet sektorov na stopu, iba pre médiá s geometriou.
BPB_NumHeads	26	2	Počet hláv, tiež iba pre médiá s geometriou.
BPB_HiddSec	28	4	Počet skrytých sektorov.
BPB_TotSec32	32	4	Celkové množstvo sektorov pre FAT32.

Ďalšia súčasť boot sektoru má odlišnú štruktúru pre FAT16 a FAT32. Obsahujú informácie o adrese koreňového adresára (pri FAT32), názve partície, identifikačných čísel, verziách. Veľkosť je pre obidva typy FAT rovnaká – 26 bajtov. Výňatok dôležitých informácií z rozširujúcej štruktúry uvediem v Tabuľke 2-9.

Tabuľka 2-9 Rozširujúca štruktúra boot sektoru pre FAT16 a FAT32

Názov	Ofset	Dĺžka	Popis
FAT16			
BS_VolLab	43	11	Menovka partície
BS_FilSysType	54	8	Typ: FAT12 alebo FAT16
FAT32			
BPB_FATSz32	36	4	Veľkosť tabuľky FAT v sektoroch
BPB_RootClus	44	4	Adresa prvého clusteru koreňového adresára
BS_VolLab	71	11	Menovka partície
BS_FilSysType	82	8	Typ: vždy FAT32

File Allocation Table

Dátovú časť partície skladajú do celku clustre s obvyklou veľkosťou 4kB alebo 8kB. Veľkosť clustru je celistvým násobkom veľkosti sektoru, inak povedané určitý počet sektorov sa zoskupuje do jedného clustru. Iným faktorom, ktorý nepriamo vplýva na dĺžku clustru je veľkosť partície (maximálny počet možných adries clustrov * veľkosť clustru = maximálna veľkosť partície). Každý záznam v alokačnej tabuľke patrí jednému clustru v dátovej oblasti. Záznam vo FAT ukazuje buď na ďalší záznam, alebo indikuje koniec reťazca clustrov (EOF), príp. označuje voľný, chybný, rezervovaný, alebo nevyužitý cluster. Postupnosť clustrov, ukončených EOF clustrom, zhromažďuje dáta jedného súboru, resp. adresáru (viď Obr 2-23). Reťazec clustrov si môžeme predstaviť ako jednosmerne viazaný lineárny zoznam.

Počet sektorov koreňového adresára vypočítame:

$$rootDirSecs = \left(BPB\_RootEntCnt * 32 + BPB\_BytsPerSec - 1\right) / BPB\_BytsPerSec$$

Pre FAT32 je vždy počet root sektorov 0. Pre ďalšie výpočty bude potrebná aj adresa prvého dátového sektoru:

$$firstDataSec = BPB\_resvdSecCnt + BPB\_NumFATs * FATSz + rootDirSecs,$$

kde FATSz udáva veľkosť FAT tabuľky (viď Tabuľka 2-8, resp. 2-9 pre FAT32). Teraz môžeme zistiť adresu prvého sektoru v N-tom clustri:

$$firstSecClusN = (N-2) * PBP\_SecPerClus + firstDataSec$$

Táto adresa je relatívna k adrese prvého sektoru partície.

Zistenie typu FAT a adresu záznamu vo FAT

Podľa dokumentácie [15] jediný správny údaj pre získanie verzie FAT predstavuje počet clustrov na disku. Jeho obstaranie vyžaduje nasledovný postup:

$$datSecs = totSecs - (BPB\_resvdSecCnt + PBP\_NumFATs * FATSz + rootDirSecs),$$

kde číslo totSecs označuje celkový počet údajov (vid Tabuľka 2-8).

$$countClus = datSecs / BPB\_SecPerClus$$

A teraz zistíme typ FAT:

- $countClus < 4085$ – FAT12
- $4085 \le countClus < 65525$ – FAT16
- $countClus \ge 65525$ – FAT32

Významnými hodnotami sú aj adresy sektorov a ofset, na ktorom nájdeme N-tú položku v tabuľke FAT:

$$fatSecN = BPB\_\mathrm{Re}svdSecCnt + fatOffset / BPB\_BytsPerSec,$$

$$fatNoffset = fatOffset \mod BPB\_BytsPerSec,$$

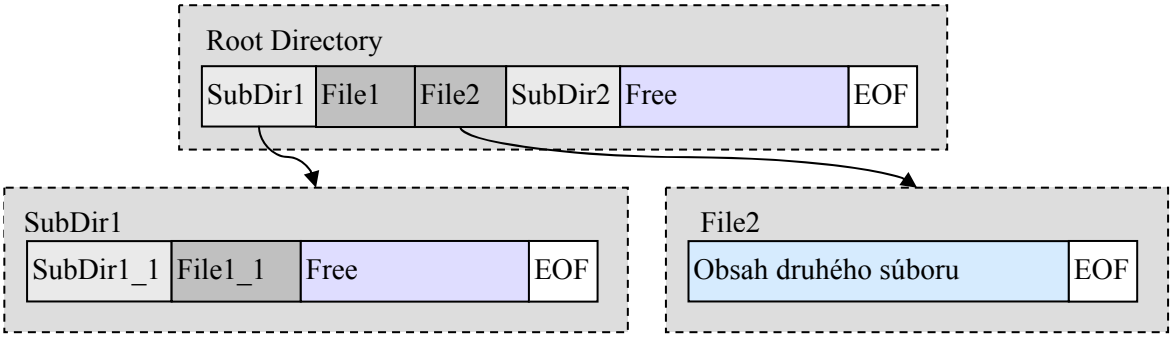
kde *fatOffset* je N*2 pre FAT16 a N*4 pre FAT32.

Adresárová štruktúra

Informácie o umiestnení adresárov a súborov nachádzame v rodičovskom adresári. V hierarchii najvyššie miesto zastáva koreňový adresár. Jeho adresu vypočítame pre FAT16:

$$firstRootDirSec = BPB\_resvdSecCnt + (BPB\_NumFATs * BPB\_FATSz16),$$

FAT32 si tento údaj ukladá do rozširujúcej štruktúry (viď Tabuľka 2-9).



Obr 2-26 Položky v adresári

Záznamy o adresároch aj súboroch sú uložené v rovnakej 32 bajtovej štruktúre (viď Tabuľka 2-10), adresár rozoznáme od súboru atribútom DIR_Attr nastaveným na DIRECTORY. Záznam obsahuje číslo prvého sektoru clustru súboru/adresáru. Na tejto adrese nájdeme buďto ďalší adresár, alebo priamo obsah súboru. Príklad zloženia adresáru a odkazovanie na ďalšie položky podáva Obr 2-26. Neobsadený záznam v adresári indikuje prvý znak v názve položky (hodnota 0). Hodnota 0xE5 znamená, že údaj bol obsadený, ale došlo k jeho zmazaniu a možno ho pokladať znovu za využiteľný.

Tabuľka 2-10 Záznam o položke v adresári

Názov	Ofset	Veľkosť	Popis
DIR_Name	0	11	Názov súboru (8+3 znaky prípona)
DIR_Attr	11	1	Atribúty, napr.: READ_ONLY 0x01, DIRECTORY 0x10
DIR_NTRes	12	1	Rezervované
DIR_CrtTimeTenth	13	1	Čas vytvorenia súboru v 10ms jednotkách: 0-199
DIR_CrtTime	14	2	Čas vytvorenia súboru (počet bitov v hod:5, min:6, secs/2:5)
DIR_CrtDate	16	2	Dátum vytvorenia súboru (rok:7, mesiac:4, deň:5)
DIR_LstAccDate	18	2	Dátum posledného prístupu k súboru (ako predchádzajúci)
DIR_FstClusHi	20	2	Iba pre FAT32: vyššie 2 bajty adresy prvého clustru súboru
DIR_WrtTime	22	2	Čas poslednej zmeny (rozloženie ako v DIR_CrtTime)
DIR_WrtDate	24	2	Dátum poslednej zmeny (rozloženie ako v DIR_CrtDate)
DIR_FstClusLo	26	2	Nižšie dva bajty adresy prvého clustru súboru
DIR_FileSize	28	4	Veľkosť súboru v bajtoch.

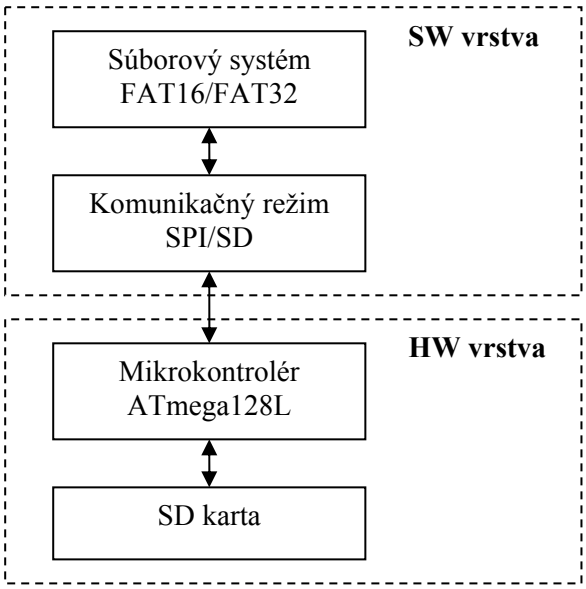
Obmedzenie dĺžky názvu súboru, eliminujú tzv. Long File Names (LFN), ktoré zvýšia veľkosť názvu až na 255 znakov. Detekcia LFN záznamu prebieha porovnaním DIR_Attr, hodnota 0x0F označuje LFN. Umiestnenie rozširujúcich záznamov LFN patriacich jednej položke sú ukladané ako štandardné záznamy. Rozlíšenie spočíva v atribúte položky nastavenej na VOLUME_ID. Staršie OS takéto položky vynechávali.

3 Návrh a implementácia

V tejto časti sa budem zaoberať analýzou problému, navrhnutím a samotnou realizáciou. Na začiatku každého projektu je nutné rozdeliť prácu do ucelených celkov, ktoré je nutné dobre skombinovať a vytvoriť tak výsledné dielo. V tomto prípade sa dá projekt rozčleniť do troch častí:

1. výber vhodného HW pre realizáciu a zostrojenie vývojovej dosky
2. inicializácia a komunikácia s SD kartou na úrovni práce s blokmi dát
3. implementácia súborové systému FAT pre prácu s adresármi a súbormi

Na najnižšej úrovni aplikáciu tvoria dve vrstvy: hardvérová a softvérová. V hardvérovej sa jedná o fyzické spojenie vstupných/výstupných portov mikrokontroléru a SD karty. Softvérová vrstva predstavuje komunikačné spojenie s SD kartou, ktoré využijú pre získavanie a posielanie dát funkcie systému súborov FAT. Na Obr 3-1 je názorná ukážka vrstvového modelu.



Obr 3-1 Model SW a HW vrstvy

3.1 Požiadavky na jednotlivé fáze projektu

Hardwarové nároky

Pri výbere hardwaru budem dbať na široké spektrum využitia a dobré podmienky pre vývoj aplikácií. Pri výbere rozhoduje prijateľná cena pri požadovaných parametroch, efektivita vývojových nástrojov a celkové možnosti použitia. Minimálnymi požiadavkami na mikrokontrolér bola interná flash pamäť 32KB, pamäť SRAM 1KB, hardwarové SPI rozhranie a umožnenie postupovať príkaz za príkazom, tzv. on-chip-debug. Pri výbere je potreba dbať na fyzické možnosti, hlavne flash pamäť a SRAM by mali splňovať aspoň minimálne požiadavky. Obmedzenie na nižšie nároky by vyžadovalo upravenie

zdrojového kódu z vypustením niektorých funkcií, prípadne znížiť veľkosť vyrovnávacej pamäte a dĺžku bloku pri čítaní/zápise z karty.

Pre zvolený mikrokontrolér je nevyhnutné navrhnúť a skonštruovať vývojovú dosku. Doska by mala okrem podporných obvodov ako napájanie obsahovať aj päticu pre pripojenie SD karty. Veľmi užitočným prvkom bude aj obvod pre komunikáciu s počítačom (napr. so sériovým portom RS-232), cez ktorý sa bude reprezentovať funkčnosť. Neodmysliteľnou súčasťou každej vývojovej dosky je aj začlenenie testovacích elementov ako tlačidlá, príp. svetelné diódy LED. Pri tvorbe schémy bude potrebné zohľadniť aj rozhranie pre programovanie mikrokontroléru.

Inicializácia a dátová komunikácia s kartou

Samotné pripojenie SD karty bude súčasťou hardwarovej časti projektu. Problémy pripojení by nemal vzniknúť, pretože karta aj mikrokontrolér operujú na rovnakých napäťových úrovniach. V SD režime bude potreba vyriešiť obojsmerné linky.

Pri inicializovaní karty sa bude postupovať podľa špecifikácie uvedenej v kap. 2.1. V nej musí byť zahrnutá aj kontrola a dohadovanie operačného napätie a pri nekompatibilite karty odmietnuť spoluprácu s kartou, aby sa predišlo príp. kolíziám, alebo poškodeniu hardwaru. Režim prenosu dát bude predstavovať operácie nad blokmi dát a to čítanie, zápis a bezpečné mazanie, t.j. prepísanie celého bloku skupinou 0 alebo 1.

Požiadavky na súborový systém

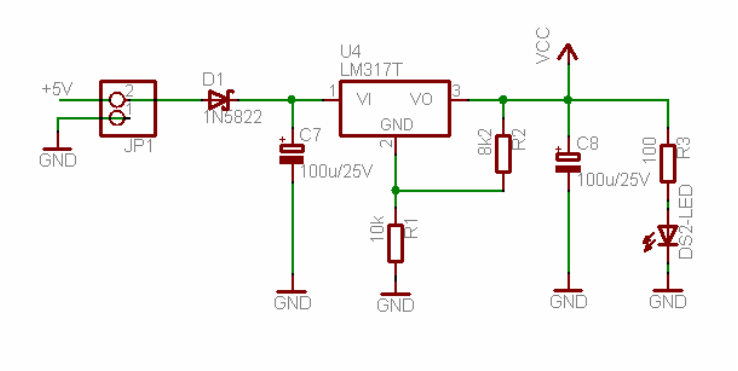
Implementácia FAT systému je čisto softwarovou záležitosťou a bude využívať nižšiu vrstvu – vrstvu dátovej komunikácie s kartou cez rozhranie SPI alebo SD. Správa súborov a adresárov zakryje práca s deskriptormi súborov, resp. adresárov. Funkcie umožnia získať obsah adresárov, čítať, doplňovať, mazať súbory. Doplňujúce informácie o voľnej a celkovej veľkosti pamäte karty budú tiež poskytnuté.

3.2 Návrh a realizácia HW časti projektu

Výber hardwarových komponent

Pri návrhu systému som vychádzal z predchádzajúcich požiadaviek. Začal som s výberom mikrokontroléru. Po prieskume dostupných mikrokontrolérov na súčasnom trhu, som sa rozhodol pre ATmega128L (popis viď kap. 2.2) z dielne firmy Atmel. Celá rodina 8-bitových mikrokontrolérov AVR je založená na rovnakom jadre a preto ho je možné nahradiť takmer ktorýmkoľvek z tejto rady (je nevyhnutné pritom vychádzať z minimálnych nárokov stanovených v predchádzajúcej kapitole a upraviť zapojenie vstupných/výstupných vývodov podľa použitého puzdra). Verziu „L“ som vybral kvôli nižšiemu operačnému napätiu, aby bol plno kompatibilný s väčšinou SD kariet. Rozsah operačných napätí je pre ATmega128L od 2.7V do 5.5V a pre SD karty od 2.7V do 3.6V, ale každá

karta nemusí podporovať celú škálu (viď kap 2.1). Preto som zvolil napájanie obvodov 3.3V. Pre stabilizáciu na úroveň 3.3V použijem regulátor napätia LM317T. Schému zapojenia kompletného napájacieho obvodu je na Obr 3-2.



Obr 3-2 Schéma zapojenia zdroju

Výstupné napätie sa vypočíta podľa nasledujúceho vzorca, kde je potrebné zvoliť hodnoty rezistorov R_1 a R_2:

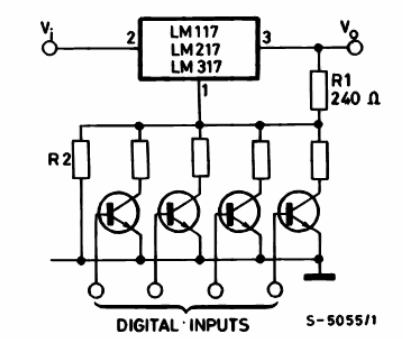
$$U_O = U_{REF}\left(1 + \frac{R_1}{R_{12}}\right) + I_{ADJ} R_2$$

kde $U_{REF} = 1.25V$ a $I_{ADJ} = 50\mu A$ pri 25°C, maximálne $I_{ADJ} = 100\mu A$ (prevzaté z [8]).

Príklad výpočtu naznačí nasledujúci vzorec:

$$U_O = 1.25V(1 + \frac{10000\Omega}{8200\Omega}) + 50 * 10^{-6} A * 10000\Omega = 3.27V$$

Na Obr. 3-3 je naznačené zapojenie obvodu LM317 s digitálnou reguláciou výstupného napätia. Týmto zapojením by bolo možné doplniť pôvodné zapojenie a docieliť tak zníženie operačného napätia na najnižšiu úroveň, ktorú podporujú zároveň aj mikrokontrolér a karta.



Obr 3-3 Zapojenie LM317 s digitálnym riadením výstupného napätia (Prevzaté z [8])

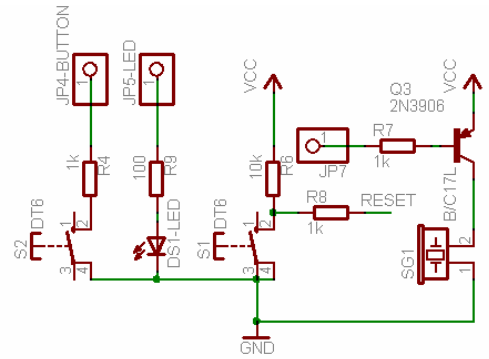
Ďalším komponentom pre výber bol obvod pre sériovú komunikáciu s počítačom, určenú pre predvedenie funkčnosti projektu. Hlavnú rolu pri výbere zohralo už vyššie spomínané napájacie napätie systému, ktoré je 3.3V. Kvôli tomu som siahol pri rozhodovaní po MAX3232 CPE od firmy MAXIM. Tento vysielač - prijímač komunikuje so sériovým portom RS-232 a je pripojiteľný

sériovému rozhraniu mikrokontroléru. Na vstupe z mikrokontroléru do vstupnej logiky obvodu je vhodné zapojiť ochranný rezistor (hodnota 1kΩ).

Návrh a zhotovenie vývojovej dosky

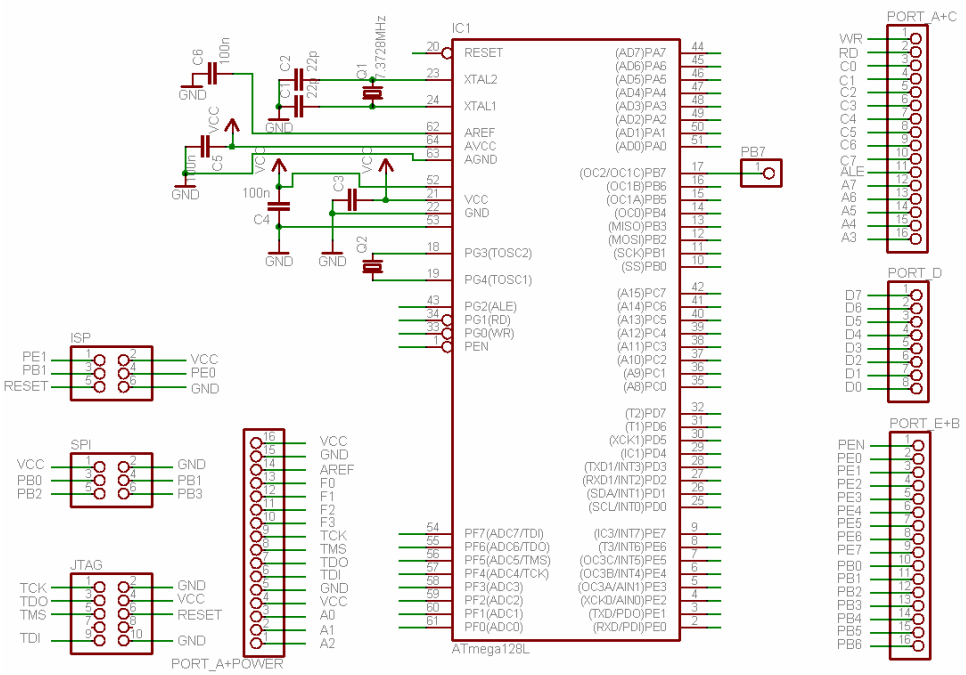
Teraz nastal čas zvoliť softvér pre kreslenie schém a design dosiek plošných spojov. Rozhodol som sa pre voľne dostupný a užívateľsky prijateľný program EAGLE (verzia 4.16r2). Umožňuje navrhnúť schému a k nej zhotoviť dosku plošných spojov (DPS). EAGLE síce disponuje funkciou automatického vytvorenia plošných spojov (tzv. auto-routing), napriek tomu je vo väčšine prípadov nevyhnutné siahnuť po ručnom doladení výsledného designu DPS. Najvhodnejším variantom je manuálne rozmiestenie súčiastok a kritické miesta spojov rozvrhnúť bez použitia routovacieho pomocníka.

Pri návrhu a kreslení schémy som sa nestretol s väčšími problémami. Hlavnou zásadou je dávať pozor a kontrolovať či spoje sú naozaj spojmi a naopak, t.j. správne vytvorenie uzlov (môže sa stať, že uzol sa objaví aj na mieste, kde by sa mali vodiče iba križovať) a niekedy sa vyskytne aj spoj, resp. chýbajúci spoj najmä po korektúrach ako presuny a rotácie. Do vývojovej dosky som zahrnul aj malé testovacie prostredie (viď Obr 3-4) pre vývoj aplikácií, zloženého z jedného tlačidla a LED diódy. Ďalším užitočným elementom je tlačidlo RESET a malý bzučiak zapojený cez spínací tranzistor.



Obr 3-4 Schéma zapojenia testovacích prvkov a RESET

Výroba jednoúčelovej dosky mi pripadala vzhľadom na investície ako dosť neekonomické riešenie. A preto som na vývojovú dosku osadil aj konektory k jednotlivým pinom portov mikrokontroléru, aby som umožnil pripojenie aj iných obvodov a zariadení k mikrokontroléru (viď Obr 3-5). Týmto krokom som dosku pripravil aj pre vývoj iných, príp. rozvoj riešenej aplikácie. S podobnou myšlienkou som počítal aj pri osadení slotu pre SD kartu. Preto tu je šanca pripojiť kartu na iný port mikrokontroléru.



Obr 3-5 Schéma zapojenia ATmega128

Rozhrania pre pripojenie programátorov In-System Programming (ISP) a Joint Test Action Group (JTAG). Doska má aj prípojku SPI rozhrania. Keďže ATmega128L pracuje na frekvencii maximálne 8MHz, použil som kryštál 7.3728MHz (na schéme Q_1).

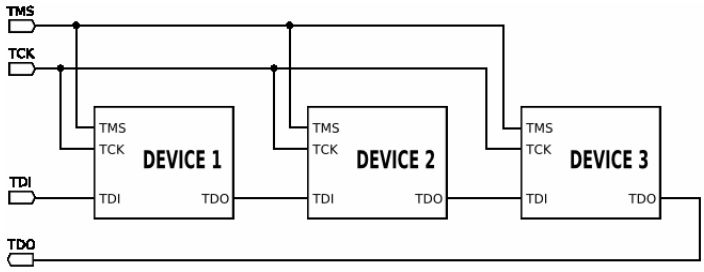
Po navrhnutí konečnej podoby schémy som začal s designom dosky plošných spojov. Súčiastky som sa snažil rozmiestniť tak, aby boli usporiadané do oblastí podľa častí, do ktorých patria, t.j. mikrokontrolér s konektormi, zdroj, testovací článok, sériové rozhranie, slot pre SD kartu. Väčšinu plošných spojov bolo podľa očakávaní nevyhnutné nakresliť ručne, auto-routing našiel uplatnenie hlavne pri pinoch konektorov, ktorých je takmer 50. Návrh dosky plošných spojov je uvedený v prílohe č. 1 až 3 s pohľadmi z oboch strán, vrátane osadenia súčiastok. Materiál pre výrobu dosky som po konzultácii s firmou Plošné spoje fy. Mucha (URL: <http://www.volny.cz/pavel-mucha/>) zvolil obojstranný UMATEX FR4 o sile 1.2mm a sile medi 35µm. Plošný spoj bol vyrobený fotocestou.

Programátor JTAG

Programovanie ATmega128 je možné dvoma spôsobmi. Prvým je ISP, ktorý síce podporuje tzv. on-chip programovanie (nahrávať softvér do už plne zapojeného obvodu), ale neumožňuje debug mód. Preto pri riešení projektu zvíťazil druhý z programátorov – JTAG, kompenzujúci vyššie zmienený nedostatok.

Rozhranie JTAG bolo štandardizované v roku 1990 a primárne vyvinuté pre testovanie prístupových portov obvodov s metódou „boundary scan“. Táto metóda kontroluje prepojenia a pamäťové, logické bloky a pod. bez použitia akýchkoľvek fyzických snímačov (Prevzaté z [10]). Konektor JTAG rozhrania obsahuje okrem napájacích aj 5 pinov pre prenos informácií. Sériovú

komunikáciu zaisťuje jedna vstupná linka Test Data In (TDI) a výstupná Test Data Out (TDO). Ďalší zo signálov je hodinový Test Clock (TCK) – v jednom takte po sériovej linke prejde jeden bit. Test Mode Select (TMS) ovláda stavový automat, ktorý prezentuje konfiguráciu interných registrov a toku dát. Signál Test Reset (TRST) nemusí byť zapojený, ale prevádza obyčajne asynchrónny reset testovacej logiky. V prípade, že TRST chýba, reset sa vykonáva inštrukčne. Obrázok 3-6 znázorňuje zapojenie viacerých blokov k jednému JTAG, t.j. ladenie zložitejších systémov môžeme vykonávať s jedným programátorom.



Obr 3-6 JTAG - pripojenie k viacerým zariadeniam (Prevzaté z [9])

Podľa stránky výrobcu existujú pre ATmega128L konkrétne dva JTAG programátory: AVR JTAGICE mkII a AVR JTAG ICE (zdroj URL: <http://www.atmel.com/>). Z ekonomického hľadiska a pre nedostupnosť pôvodného AVR JTAG ICE som použil alternatívny JtagIce zostrojený podľa plánu zverejnenom na serveri dioda.cz (viď Príloha č. 4). Jednou z jeho výhod je automatická aktualizácia firmware v prostredí AVR Studio.

3.3 Návrh softvérovej vrstvy

Pri návrhu programového vybavenia aplikácie som riešil viacero záležitostí ako výber vývojového prostredia, rozdelenie funkcií do jednotlivých súborov, uloženie konfiguračných nastavení (napr. výber režimu, nastavenie pinov, na ktoré je pripojená karta a pod.).

AVR Studio

Spoločnosť Atmel ponúka na vývoj aplikácií k mikrokontrolérom rady AVR profesionálny nástroj AVR Studio 4 (softvér je voľne stiahnuteľný z oficiálnych stránok [11]). Písanie zdrojových kódov má ešte malé nedostatky (nepodporuje kontextové nápovede a inteligentnejšie formátovanie textu), ale ladenie v spolupráci s JTAG rozhraním je veľmi užitočným a pohodlným pomocníkom pri odhaľovaní chýb. V debug móde je samozrejmosťou prezeranie stavu (hodnôt) registrov, portov, premenných, pamäte atď. V režime simulácie poskytuje AVR Studio aj meranie cyklov jednotlivých inštrukcií a meranie času pri nastavení správnej pracovnej frekvencie. V kombinácii s prekladačom WinAVR (voľne dostupný na oficiálnej stránke [12]) je možné použiť ako programovací jazyk C. Na písanie zdrojového kódu som využil aj inteligentný editor PSPad [13].

Usporiadanie zdrojových súborov

Funkcie pre inicializáciu karty i rozhrania a dátovú komunikáciu budú pre SPI a SD režim uložené v osobitných súboroch (spi.c, sd.c). Výber režimu bude preveditelný nastavením (napr. #define CARD_MODE SPI_SW) v konfiguračnom súbore config.h, kde sa okrem iného prideľujú aj čísla pinov a porty, ku ktorým sú pripojené vývody karty. Blok funkcií pre prácu s FAT systémom nájdeme v súbore fat.c.

3.4 SPI režim

Sériový SPI režim je jedným z dvoch pracovných módov SD kariet (viď aj kap. 2.1.5). Spája sa hlavne s predchodcom SD kariet MMC. V zariadeniach ako sú čítačky kariet ho nájdeme kvôli spätnej kompatibilite kariet. Vo vstavaných systémoch zas pre jeho dobrú hardvérovú podporu. Mikrokontrolér ATmega128L tiež disponuje hardvérovou implementáciou SPI (viď kap. 2.2.3). Pri realizácii práce som sa rozhodol aj pre softvérovú verziu tohto rozhrania. Na porovnaní rýchlosti obidvoch implementácií ukážem výhodu tej hardvérovej oproti softvérovému riešeniu.

3.4.1 Inicializácia hardvérového SPI

Inicializácia SPI režimu začína štandardne nastavením smeru pinov na vstupné alebo výstupné. Vstupom je iba signál MISO (Master In Slave Out). Potom sa povolí zapnutie SPI príznakom SPIE v registri SPCR (SPI Control Register), vyberie režim master alebo slave (príznak MSTR v SPCR) a určí prevádzková frekvencia. Zvolenie frekvencie má na výber z viacero hodnôt – vždy sa jedná o diel frekvencie oscilátora (viď Tabuľka 3-1). Príznaky SPR1 a SPR0 indikujúce deliacu časť sú v SPCR a príznak SPI2X v registri SPSR (SPI Status Register).

Tabuľka 3-1 Nastavenie frekvencie SPI

SPI2X	SPR1	SPR0	Frekvencia
0	0	0	f/4
0	0	1	f/16
0	1	0	f/64
0	1	1	f/128
1	0	0	f/2
1	0	1	f/8
1	1	0	f/32
1	1	1	f/64

Pri inicializácii SD karty je potrebné znížiť frekvenciu v rozsahu od 100-400KHz (toto nariadenie zohľadňuje kritérium pre spätnú kompatibilitu s MMC kartami). Keďže použitý kryštál zariaďuje pracovnú frekvenciu 7.3728, vhodný deliaci pomer kvôli obmedzeniu je 32:

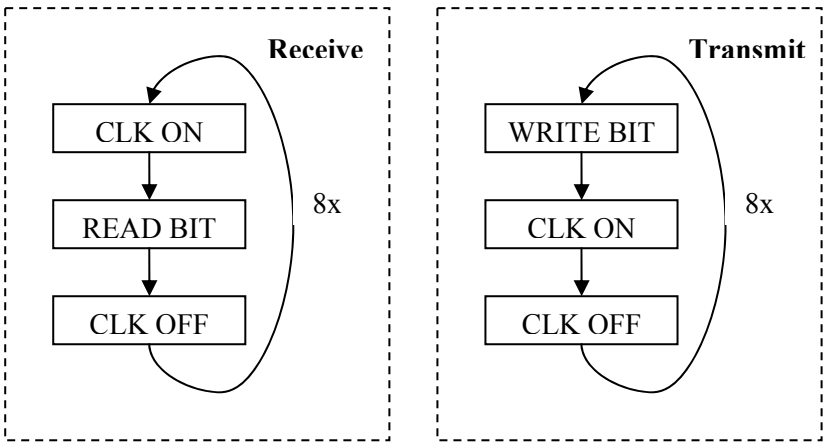
$$f_{SPI} = \frac{f}{32} = \frac{7.3728MHz}{32} = 230.4KHz$$

Po inicializovaní karty frekvenciu možno zvýšiť na najvyššiu hodnotu, t.j. na polovičnú hodnotu oscilátora.

Posielanie dát (po bajtoch) cez SPI vykoná príkaz pre uloženie novej hodnoty do registru SPDR (SPI Data Register) a počkať na potvrdenie o ukončení prenosu (po vyslaní sa nastaví príznak SPIF v SPSR). Následne získané dáta z registra SPDR predstavujú prijaté bity zo slave zariadenia (SD karty).

3.4.2 Inicializácia softvérového SPI

Prvý krok inicializácie, spočívajúci v pridelení smeru signálom, zhodne ako v HW vyčlení ako vstupnú linku iba MISO. Generovanie hodinového cyklu zariadi program vždy pri vyslaní, alebo prijatí bitu. Tu nastáva rozdiel oproti hardvérovému SPI. Prenos jedného bajtu znázorňuje Obr 3-7.



Obr 3-7 SW SPI: a) prijatie znaku, b) odoslanie znaku

Hlavná nevýhoda takéhoto riešenia je jasná – podstatne nižšia prenosová rýchlosť. Réžia, spojená s transferom jedného bitu, je pomerne veľká – rádovo niekoľko desiatok taktov.

3.4.3 Inicializácia karty SD v SPI

Vývojový diagram postupu pri inicializácii je na Obr 2-4. Po pripojení karty do obvodu potrebuje minimálne 74 taktov, kým sa dostane do stavu „idle“, t.j. do stavu, v ktorom prijíma príkazy na reset (CMD0), získanie operačných úrovní napätí (CMD58) a validácie napätia (CMD55 + ACMD41). Momentálne karta pracuje v SD režime a prepnutie do SPI vykoná príkaz CMD0 pri nastavení signálu Chip Select (CS) na logickú 0.

Nasledujúci krok tvorí validácia operačného napätia. Najprv získam hodnoty registra OCR (viď Tabuľka 3-2), ktorý ukrýva indikátory podporujúcich napäťových úrovní. Potom zistím stav príznakov č. 20 a 21 (hodnoty viď Tabuľka 3-2) udávajúce, že karta naozaj podporuje napätia v rozsahu 3.2 -3.4V (mikrokontrolér má na vývodoch asi 3.27V). V prípade, že karta uvedený rozsah nepodporuje ukončím inicializáciu karty a ďalšie príkazy už neposielam (v SD režime sa v tomto prípade posiela karta to neaktívneho stavu). Inak pokračujem posielaním príkazu ACMD41, resp.

kombináciou CMD55 a CMD41, pretože aplikačné príkazy musia byť karte oznámené práve príkazom CMD55. Túto dvojicu príkazov posielam dovtedy, pokiaľ je karta v idle stave.

Tabuľka 3-2 Register OCR - Operating Condition Register (Prevzaté z [4])

OCR Bit	VDD Voltage Window	OCR Bit	VDD Voltage Window
0-3	Reserved	15	2.7 to 2.8
4	1.6 to 1.7	16	2.8 to 2.9
5	1.7 to 1.8	17	2.9 to 3.0
6	1.8 to 1.9	18	3.0 to 3.1
7	1.9 to 2.0	19	3.1 to 3.2
8	2.0 to 2.1	20	3.2 to 3.3
9	2.1 to 2.2	21	3.3 to 3.4
10	2.2 to 2.3	22	3.4 to 3.5
11	2.3 to 2.4	23	3.5 to 3.6
12	2.4 to 2.5	24-30	Reserved
13	2.5 to 2.6	31	Card power-up status bit
14	2.6 to 2.7		

3.4.4 Prenos dát

Transfery dát zaisťujú dva režimy: jedno blokový a mnoho blokový. Vzhľadom k obmedzeným možnostiam mikrokontroléru (veľkosť SRAM, rýchlosť) postačí aj prenos po jednom bloku. Mazanie dát prepíše vybraný blok dát nulami alebo jednotkami. Toto nastavenie určuje bit DATA_STAT_AFTER_ERASE v SCR registri.

Čítanie

Pre čítanie blokov dát som vytvoril funkciu ReadDataBlock(), ktorá uloží do bufferu požadovaný počet načítaných bajtov. Pri práci so systémom súborov však bude využiteľná ďalšia funkcia ReadSector(sector, data). Načíta jeden sektor zo zadanej adresy v parametri funkcie. Veľkosť sektoru je obyčajne 512 bajtov. Keďže parametrom príkazu pre získanie jedného bloku dát z karty (READ_SINGLE_BLOCK) je priamo adresa bajtu v pamäti, na prepočítanie do adresy sektoru stačí vynásobiť číslo sektoru počtom bajtov na sektor. Pred začiatkom prenosu treba počkať na štartovací znak 0xFE. Po ukončení prenosu sú poslané aj dva bajty CRC kódu, ktoré v tomto prípade zanedbávam. Ochrana dát cyklickým kódom je implicitne v SPI zakázaná. Povolenie zabezpečeného prenosu dát vykoná príkaz CRC_ON_OFF (CMD59).

Zápis

Podobne ako pri čítaní aj zápis dát prevádzam po sektoroch. Funkcia WriteSector(sector, data) v parametri dostane index sektoru (nutné prepočítať na ofset v bajtoch). Transfer začína poslaním príkazu WRITE_BLOCK a počkaním na odpoveď. Samotný prenos bajt po bajte predchádza vyslanie začiatočného znaku 0xFE. Na koniec zapíšem ešte dva bajty, predstavujúce CRC kód. Pri nepovolenom kontrolovaní CRC vyšlem vždy iba 0xFF. Na prevedenie zápisu (programovania flash pamäte karty) funkcia čaká dovtedy, kým je vysielaný „busy“ rámec (0x00).

Vymazanie dát

Bezpečné zničenie informácií na médiu zariaďuje funkcia SecureEraseCluster(sector, clusterSize). Bezpečným ho nazývam z dôvodu reálneho odstránenia uložených dát, t.j. prepis celého bloku (clusteru) 1 alebo 0, ako je uvedené vyššie. Vymazanie súboru totiž v systéme FAT v skutočnosti ako také neprebieha. Stačí iba zrušiť (urobiť ho neplatým) záznam o súbore z adresárovej štruktúry. Parameter clusterSize udáva počet bajtov v jednom clustri.

Samotné mazanie dát vykonávajú tri príkazy. Prvým (ERASE_WR_BLK_START) určím počiatočnú adresu bloku (číslo bajtu) a druhým (ERASE_WR_BLK_END) číslo koncového bajtu. Posledný z trojice príkazov spôsobí spomínané deštruktívne zmazanie označeného bloku. Stav zaneprázdnenosti pri vykonávaní príkazu rovnako ako pri zápise indikuje logická 0 na MISO linke.

Načítanie CID registra

Card Identification Register, alebo identifikačný register karty uchováva informácie o výrobcovi, sériovom čísle a pod. Funkcia ReadCid() vracia štruktúru, obsahujúcu informácie z CID registra (význam jednotlivých položiek viď Tabuľka 3-3).

Tabuľka 3-3 CID register

Názov	Typ	Šírka [bit]	Hodnota	Popis
ID výrobcu	Binárny	8	0x03	ID prideľuje SD Card Asoc.
ID aplikácie	ASCII	16	0x53, 0x44	Identifikuje kartu
Názov produktu	ASCII	40	SD16 - SD02G	Názov skrýva kapacitu karty
Verzia	BCD	8	Číslo revízie	Dve binárne-kódované desiatk. čísla
Sériové číslo	Binárny	32	Sér. číslo prodkt.	32-bitový unsigned int
Rezervované	...	4	...	...
Dátum	BCD	12	Dátum výroby	RRM, ofset od roku 200
CRC7	Binary	7	CRC7	Počíta karta
Nevyužitý	..	1	...	...

3.5 SD režim

Softvérové riadenia SD módu je podstatne náročnejšie ako predchádzajúce implementácie. Z reálneho hľadiska využitia tohto režimu nemá veľký prínos. Už pri prvotných pokusoch bolo jasné, že réžia spojená s programovou obsluhou zníži zásadne rýchlosť prenosu dát. Generovanie hodinového signálu zariaďuje ako v SW SPI funkcia, ktorá striedavo prepína na výstupe CLK nuly a jednotky (CLK v SD kartách aktivizuje logická 0).

Ďalší problém nastáva v použití obojsmerných liniek. V tomto prípade by bolo vhodné použiť tri-state, alebo troj-stavovej konfigurácie vývodov mikrokontroléru, ale nepodarilo sa mi ich dostať do plne funkčného stavu. Na pin pripojený pull-up rezistor síce držal úroveň na logickej jednotke, ale prepnutie na nulu nepracovalo podľa predstáv. Zníženie prebehlo na úroveň okolo 1V, čo je

nevyhovujúce pre SD kartu – maximálna hodnota „input low voltage“ (viď Tabuľka 3-4) je $V_{L\max} = 0.25 * V_{cc} = 0.25 * 3.27V = 0.812V$

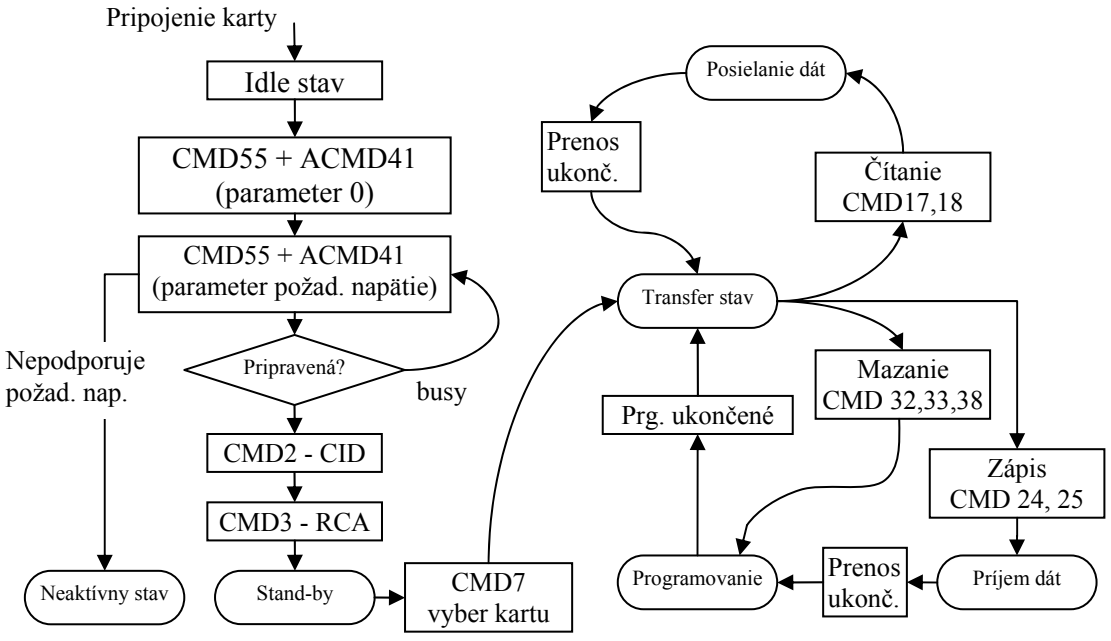
Tabuľka 3-4 Úrovne vstupných/výstupných napätí SD karty

Parameter	Symbol	Min.	Max.	Unit	Conditions
Output high voltage	V_{OH}	0.75*V_{DD}	---	V	I_{OH} = -100 uA@ V_{DD} (minimum)
Output low voltage	V_{OL}	---	0.125*V_{DD}	V	I_{OL} = 100 uA@ V_{DD} (minimum)
Input high voltage	V_{IH}	0.625*V_{DD}	V_{DD} +0.3	V	---
Input low voltage	V_{IL}	V_{SS} -0.3	0.25*V_{DD}	V	---

Preto som použil prepínanie vstupu/výstupu. Tento krok opäť nepríjemne znížil prenosovú rýchlosť, ale pri sekvenčných čítaniach/zápisoch to má len minimálny vplyv.

3.5.1 Inicializácia karty v SD

Príprava pred štartom dátovej komunikácie spočíva v nastavení smeru vývodov (myslí sa tým počiatočné určenie smeru, pretože linky okrem výstupnej CLK, budú prepínané) a inicializácia karty. Štart karty v režime SD (postup viď Obr 3-8) sprevádzajú obdobné kroky ako v SPI. Pri príkaze RESET - CMD0 (nepovinný v SD móde) už nie je CS na nízkej úrovni, lebo by karta prešla do SPI. Navyše je potrebné vyslať príkazy na získanie CID (ALL_SEND_CID – CMD2) a vypýtať si novú relatívnu adresu karty (Relative Card Adress – RCA) príkazom CMD3. RCA vykazuje adresu karty, používanú pre aktiváciu karty príkazom CMD7 a prechodom do stavu transfer. Týmto ukončíme proces inicializácie a karta je teraz schopná dátovej komunikácie (viď tiež Obr 3-8).



Obr 3-8 Inicializácia a dátová komunikácia v SD režime

3.5.2 Prenos dát v SD

Karta musí byť vždy v stave transfer pred vyslaním akéhokoľvek príkazu pre čítanie/zápis/mazanie. Podrobnejší postup krokov pri dátovej komunikácii demonštruje vývojový diagram na Obr 3-8. Funkcie pre čítanie/zápis/mazanie sektorov som vytvoril s rovnakou funkčnosťou a názvami, aby boli plne kompatibilné s funkciami v SPI (viď kap. 3.4.4). Knižnici FAT podprogramov potom nezáleží, ktorý s režimov je aktivovaný.

Implementácia samotného čítanie/zápisu/mazania obnáša značné množstvo riadenia, spojeného už so zmieňovaným spomalením celého prenosu. Veľký pozor musím dávať na synchronizáciu pri prepínaní smerov liniek, aby nedošlo k ich poškodeniu. Réžia pri prenose je navonok podobná ako v SPI. S viditeľnými rozdielmi som sa stretol pri časovaní, kde v SPI sú čakacie intervaly vždy po ôsmich cykloch, v SD rádovo jednotky cyklov. Štartovací znak dátového prenosu v SPI nahradzuje iba jeden bit (logická nula) v režime SD.

3.5.3 Zabezpečovací linkový kód – Cyclic Redundancy Check (CRC)

Na rozdiel od SPI v SD režime je ochrana prenášaných informácií povinná. Pre tento účel používajú CRC7 (vzorec na výpočet viď kap 2.1.5) všetky príkazy a odpovede (okrem typu R_3). Užitočné zabezpečenie blokových prenosov dát prevádzam podľa špecifikácie CRC16-CCITT.

Tabuľka 3-5 Algoritmus CRC

```
1. function getCRC(bit array bitString[1..len], int polynomial) {
2.  shiftRegister := initial value // commonly all 0 bits or all 1 bits
3.  for i from 1 to len {
4.   if most significant bit of shiftRegister xor bitString[i] = 1 {
5.    shiftRegister := (shiftRegister left shift 1) xor polynomial
6.    } else {
7.     shiftRegister := (shiftRegister left shift 1)
8.    }
9.   }
10.  return shiftRegister
11. }
```

Vyššie zmienený algoritmus v Tabuľke 3-5 implementujem vo funkciách pre výpočet CRC kódu. Polynóm pre CRC7 je 0x48, pre CRC16-CCITT 0x8408.

3.6 Súborový systém FAT

Implementácia FAT tvorí najrozsiahlejšiu softvérovú časť projektu. V hierarchii vrstiev je na tej najvyššej, pretože zakrýva prístup ku karte na úrovni práci so súbormi, resp. adresármi. Pri riešení tejto časti som vychádzal z informácií v kap. 2.3.

3.6.1 Inicializácia FAT

Sekcia inicializácie pripravuje na použitie ostatné funkcie pre prístup k FAT. Počiatočným krokom je načítanie tabuľky partícií a získanie ofset na prvý sektor partície. Potom nasleduje nahranie boot sektoru zväzku a výpočty dôležitých informácií (viď tiež kap 2.3): počet clustrov, štartovací dátový sektor, adresu koreňového adresára, veľkosť tabuľky FAT a zistenie typu FAT. Projekt momentálne podporuje FAT16 a FAT32, ktoré sú najpoužívanejšie. Podpora FAT12 síce nie je implementovaná, ale aplikácia s ňou ráta a v prípade potreby ju možno bezstarostne doprogramovať.

Funkcia `InitFAT`(buffer) počíta s alokovanou pamäťou pre buffer, ktorý používam pre čítanie/zápis sektoru z SD karty (detaily viď nasledujúca podkapitola).

Buffering

Pre transfer dát s SD kartou som vytvoril funkcie s využitím vyrovnávajúceho bufferu so implicitnou veľkosťou 512 bajtov (odpovedá jednému sektoru). Najvýznamnejším účelom bufferu je prevencia pred opakovaným čítaním/zápisom toho istého sektoru. Prvotnú funkciu som používal iba pre čítanie. S ďalším rozvojom aplikácie som sa rozhodol využitia overeného princípu aj pre oneskorený zápis. Výhodu neskoršieho zápisu ocení najmä zmena záznamov v rámci jedného adresára. Pri jeho používaní musí programátor dbať na určité zásady, najmä na konci zavolať funkciu `FlushWrite`() (detailnejší popis viď Príloha č. 6).

Výpočet celkového a voľného miesta na médiu

Knižnica realizuje funkcie `GetTotalSpace`() a `GetFreeSpace`(). Totálna veľkosť partície závisí na počtu clustrov získaných pri inicializácii. K vyjadreniu tejto hodnoty v bajtových jednotkách musím vynásobiť počet clustrov veľkosťou sektoru a počtom sektorov v clustri. Zložitejšou záležitosťou je vypočítanie voľného miesta na zväzku. Túto hodnotu dostanem, keď sčítam všetky neobsadené záznamy v tabuľke FAT. Programovo som pokryl túto skutočnosť pevným cyklom, ktorý skontroluje všetky záznamy vo FAT. Pri nájdení záznamu s hodnotou 0 (voľný) zvýšim počítadlo.

3.6.2 Práca s adresármi

Otvorenie adresáru prevedie funkcia `OpenDir`(dirEntry). Parametrom funkcie je záznam o adresári (obsahuje informácie popísané v Tabuľke 2-10), ktorý získame prieskumom adresára – bude popísané neskôr. Ak je záznam nulový (null), znamená to, že otvárame koreňový adresár. Funkcia vráti deskriptor adresáru, s ktorým sa naďalej pracuje. Detailnejší pohľad na štruktúru deskriptoru znázorňuje Tabuľka 3-6.

Tabuľka 3-6 Štruktúra FILE_DESC pre adresáre a súbory

```
BYTE  attr;           //  Attributes from dir entry.
DWORD clusterN;       //  Index of first cluster of file.
DWORD fileSize;       //  Size of file [in Bytes].
DWORD clustEntry;     //  Adress of cluster, where entry is located.
BYTE  offsetSecEntry; //  Offset of sector in cluster.
BYTE  offsetEntry;    //  Offset of this entry in sector.
DWORD curClust;       //  Cluster of actual position in file.
BYTE  curOffset;      //  Offset in cluster of actual position in file
DWORD posInFile;      //  Position in file [0..Size of file]
```

Vlastný deskriptor som navrhol kvôli ušetreniu miesta oproti záznamu v adresári. Uchováva iba dôležité informácie pre prácu s adresárom, resp. súborom. A pridáva ďalšie ako adresu jeho záznamu v rodičovskom adresári alebo aktuálnu pozíciu v súbore pri jeho čítaní, príp. pozíciu v adresári pri prieskume.

Prieskum adresáru zabezpečuje funkcia `GetNextFileInDir`(dir). Parameter je deskriptor adresáru a návratová hodnota štruktúra záznamu popísaná v Tabuľke 2-10. Funkcia vracia postupne všetky položky v adresári. Za každým volaním sa zvýši počítadlo ofsetu aktuálnej položky v adresári. V prípade, že adresár zasahuje aj do ďalšieho clustra, funkcia automaticky vyhľadá vo FAT tabuľke index nasledujúceho clustru.

3.6.3 Správa súborov

Popisovače súborov používajú rovnakú štruktúru ako adresáre (viď Tabuľka 3-6). Pred použitím každého súboru, ho musíme najprv otvoriť funkciou `OpenDir`(dirEntry). Podobne aj pri súboroch je parametrom záznam dirEntry (viď tiež Tabuľku 2-10), získaný volaním funkcie `GetNextFileInDir`().

Čítanie obsahu súboru

Čítanie súboru funguje obdobne ako funkcia `read`() alebo `fread`() v jazyku C. Ako parameter predávam funkcii `ReadFile`(file, data) deskriptor a alokovaný buffer pre uloženie dát. Počet čítaných bajtov je fixne nastavený na veľkosť jedného sektoru. Funkcia vracia počet načítaných bajtov. Ak hodnota klesne pod veľkosť sektoru, alebo na nulu, indikujem tým koniec súboru. Funkcia si poradí aj s fragmentovanými súbormi (viď kapitola 2.3 Fragmentácia).

Pridávanie znakov

Funkcia `AppendFile`(file, data, length) pridá určený počet (length) znakov (data) na koniec súboru (file). Zapisovať dáta možno po ľubovoľne veľkých blokoch, maximálne však 512 bajtov. Taktiež treba brať do úvahy veľkosť SRAM a pamätať na to, že 512 bajtov má alokovaných v SRAM buffer.

Problémy pri zápise do súboru nastávajú keď sa dáta nezmestia do aktuálneho sektoru, príp. ani do clustru. V takom prípade funkcia zapíše dáta do oboch sektorov, resp. alokuje pre súbor ďalší cluster s novými dátami. Pre tento účel využíva prehľadávanie záznamov o clustroch vo FAT tabuľke

a nájdenie prázdneho clustru funkciou `FindFreeCluster`(). Pri pridávaní znakov na koniec súboru využívam i funkciu `FindLastCluster`() pre obstaranie adresy posledného clustru.

Mazanie súborov

Zmazaním súboru v systéme FAT rozumieme iba zmenu položky na neplatnú a označenie všetkých clustrov súboru vo FAT tabuľke za voľné. To znamená, že deštruktívne mazanie nie je nevyhnutné. Ak však potrebujeme dáta na karte bezpečne odstrániť, stačí nastaviť pri mazaní súboru príznak SEC_ERASE_ENABLE. V tomto prípade prevedie fyzické zničenie dát SD karta (všetky bity clustrov súboru nastaví na nuly alebo jednotky). Funkciu pre vymazanie súboru som nazval `DeleteFile`(file, secErase), parametrami sú deskriptor a vyššie zmienený príznak.

4 Vyhodnotenie výsledkov

Pri práci som zápasil s viacerými komplikáciami. Týkali sa najmä softvérových verzií rozhraní. Hardvérová implementácia zakrývala komunikáciu na najnižšej úrovni (riadenie prenosu – vysielanie hodinových impulzov a pod.) a funguje spoľahlivo. S vážnym problémom som sa stretol až pri rozhraní SD. Vytvoril som funkcie na inicializáciu a prenos dát, no nepodarilo sa mi zabezpečiť samotný prenos bit po bite, čo robí funkčnosť SD rozhrania momentálne nepoužiteľným. Pretrvávajúce problémy s časovaním pri posielaní/prijímaní dát sa mi nepodarilo doviesť ku kladným výsledkom. Zistil som, že obrovskú dobu trvá vyslanie jedného bajtu – okolo 510 taktov. To je viac ako 60 taktov na jeden bit a v prepočte to pripadá na frekvenciu na hranici 100KHz (podľa špecifikácie minimálna frekvencia pri inicializácii je práve 100KHz).

Do ďalších pokračovaní projektu bude pre vyriešenie problémov s frekvenciou a časovaním, potrebné optimalizovať kód softvérovej komunikácie a zabezpečiť pravidelné generovanie hodinového signálu. Túto myšlienku by som navrhol realizovať pomocou čítača s prerušením a vyrovnávacieho bufferu, z ktorého by sa vysielali znaky. To znamená, že po pretečení čítača dôjde k prerušeniu a obslužná funkcia za každým vyšle jeden bit z bufferu. S týmto riešením pribudli nové starosti o správu bufferu, ale pri zvýšení pracovnej frekvencii mikrokontroléru to pomôže docieliť generovanie pravidelného hodinového signálu.

Realizácia softvérovej verzie SPI funguje bez väčších problémov. Pri problémoch s inicializáciou karty pomohol kondenzátor 100pF na SCLK signál. Pri porovnaní SPI rozhraní s SD by som spomenul výhodu hardvérovej implementácie SPI, ktorá je bežnou súčasťou mikrokontrolérov. Momentálne som nenašiel na trhu dostupný mikrokontrolér, ktorý by si poradil s hardvérovou realizáciou SD, čo by som označil doposiaľ za najväčšiu nevýhodu tohto rozhrania, pretože pri nárokoch na vyšší dátový prenos bude voľba spadať práve na SD.

Implementácia FAT systému sa podarila podľa zadaných kritérií v kap. 3.1. Spoľahlivo fungujú všetky programované funkcie a aplikácia si poradí s FAT16 aj FAT32 a taktiež aj s fragmentovanými súbormi.

Meranie výkonnosti

Na záver projektu som spravil pár testov na čítanie a zápis rozhrania SPI – hardvérového i softvérového (viď Tabuľka 4-1). V testoch sa jednalo o čítanie/zápis 50kB bloku dát v cykle postupne po 512 bajtoch. Meranie som prevádzal za pomoci čítača s prerušením, ktorý som spustil pred začatím prenosu a zastavil bezprostredne po prenose (počítal 10ms intervaly). V hardvérovej verzii SPI som použil všetky nastavenia deliacich frekvencií.

Tabuľka 4-1 Meranie výkonnosti prenosu dát

Deliaci pomer	Frekvencia [KHz]	Čas čítania [s]	Rýchlosť čítania [kbits/s]	Čas zápisu [s]	Rýchlosť zápisu [kbits/s]
Softvérové SPI					
...	...	20,1	20,376	24,0	17,040
Hardvérové SPI					
f/2	3686,4	5,7	71,856	5,6	73,136
f/4	1843,2	6,8	60,232	6,8	60,232
f/8	921,6	8,4	48,760	8,3	49,344
f/16	460,8	11,8	34,704	11,6	35,304
f/32	230,4	20,7	19,784	19,1	21,440
f/64	115,2	32,8	12,480	33,0	12,408
f/128	57,6	60,8	6,736	64,0	6,400

Dosiahnuté výsledky sú prekvapivo nízke pri hardvérovom SPI. Softvérová verzia SPI dosiahla výkonnosť na úrovni 230,4 KHz pri hardvérovom.

5 Záver

V projekte som realizoval pripojenie karty pomocou dvoch rozhraní a plnú funkčnosť sa mi podarilo zaručiť iba u softvérovej a hardvérovej implementácii režimu SPI. Bližšie zhodnotenie kladných aj záporných výsledkov som uviedol v kap. 4.

Veľmi dôležitým krokom pri pokračovaní na projekte je dokončenie funkcionality rozhrania SD, napr. spomínaným návrhom v kap. 4. Na ďalšie pokračovanie vo vývoji aplikácie by som poukázal na optimalizácie prenosu dát s dosiahnutím lepších výsledkov v rýchlosti prenosu. Taktiež je vhodné rozšíriť obzor implementácie FAT systému – doplniť viac funkcií (napr. kompletná editácia súborov, mazanie celých adresárov), príp. optimalizovať prístup k súborom a adresárom. Projekt by bolo možné použiť ako základ pre návrh zariadení s rozhraním SDIO, ktorý môže priamo využívať dátový priestor pre svoju potrebu.

Práca iste nájde uplatnenie aj vo vstavaných systémoch (napr. zariadenie pre meranie teploty, rôzne ďalšie monitorovacie prístroje atď.) a umožní im zaznamenávať svoje dáta pohodlne na pamäťovú kartu SD. Získať takto uložené dáta je potom bezproblémovou záležitosťou.

Literatúra

[1] SD Card Association. *About SD Memory cards* [online]. Posledné úpravy 2004 [cit. 2006-12-15]. Dokument dostupný na URL: <http://www.sdcard.org/sd_memorycard/index.html>

[2] Wikipedia. *Secure Digital Card* [online]. Posledné úpravy 2006-12-06 [cit. 2006-12-18]. Dokument dostupný na URL: <http://en.wikipedia.org/wiki/Secure_Digital_card>

[3] SD Card Association. *Simplified Version of PHYSICAL LAYER SPEC.* Ver 2.0. [online]. Posledná aktualizácia 2006-09-25 [cit. 2006-12-15]. Dokument dostupný na URL: <http://www.sdcard.org/sd_memorycard/Simplified%20Physical%20Layer%20Specification.PDF>

[4] SanDisk Corporation. *SanDisk SD Card - Product Manual*. Ver 2.2. [online]. Posledná aktualizácia 2006-11 [cit. 2006-12-15]. Dokument dostupný na URL: <http://www.digitalspirit.org/file/?aff=,/docs/sd/ProductManualSDCardv2.2final.pdf>

[5] Wikipedia. *Content Protection for Recordable Media* [online]. Posledné úpravy 2006-12-26 [cit. 2006-12-28]. Dokument dostupný na URL: <http://en.wikipedia.org/wiki/Content_Protection_for_Recordable_Media>

[6] ARKTAKE, inc. *SD Memory Card – Specification Sheet*. Ver 1.0. [online]. Posledná aktualizácia 2005 [cit. 2006-12-17]. Dostupný na URL: <http://www.arktake.co.jp/pdf/sdcard.pdf>

[7] Atmel Corporation. *Atmel ATmega128 Datasheet*. Rev. 24670-AVR-10/2006 [cit. 2006-12-26]. Dokument dostupný na URL: <http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf>

[8] STMicroelectronics, *LM317* [online]. Posledná aktualizácia 2003 [cit. 2007-03-16]. Dokument dostupný na URL: <http://www.ortodoxism.ro/datasheets2/c/0hj5dxz6qa1kdqxjigl5zpok4iky.pdf>

[9] Wikipedia. *Join Test Action Group* [online]. Posledné úpravy 2007-04-27 [cit. 2007-05-04]. Dokument dostupný na URL: <http://en.wikipedia.org/wiki/JTAG>

[10] Wikipedia. *Boundary scan* [online]. Posledné úpravy 2007-04-23 [cit. 2007-05-04]. Dokument dostupný na URL: <http://en.wikipedia.org/wiki/Boundary_scan>

[11] Atmel Corporation. *AVR Studio 4* [online]. Posledné úpravy 2007-03 [cit. 2007-04-10]. Dokument dostupný na URL: <http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725>

[12] WinAVR. *Oficiálna stránka* [online]. Posledné úpravy 2007 [cit. 2007-04-10]. Dokument dostupný na URL: <http://winavr.sourceforge.net/>

[13] PSPad Freeware editor. *Oficiálna stránka* [online]. Ver 4.5.2 (2240) – 2006-10-20 [cit. 2007-04-11]. Dokument dostupný na URL: <http://www.pspad.com>

[14] Wikipedia. *File Allocation Table* [online]. Posledné úpravy 2006-12-21 [cit. 2006-12-23]. Dokument dostupný na URL: <http://en.wikipedia.org/wiki/File_Allocation_Table>

[15] Microsoft. *FAT32 File System Specification.* Rev. 1.03 [online]. Posledné úpravy 2000-12-06 [cit. 2006-12-23]. Dokument dostupný na URL: <http://www.microsoft.com/whdc/system/platform/firmware/fatgen.mspx>

[16] VRSchool. *Link Allocation* [online]. [cit. 2007-05-07]. Dostupný na URL: <http://vrschool.ice.cycu.edu.tw/vrschool/Course/OS/CHAP10-12/os_11-22.htm>

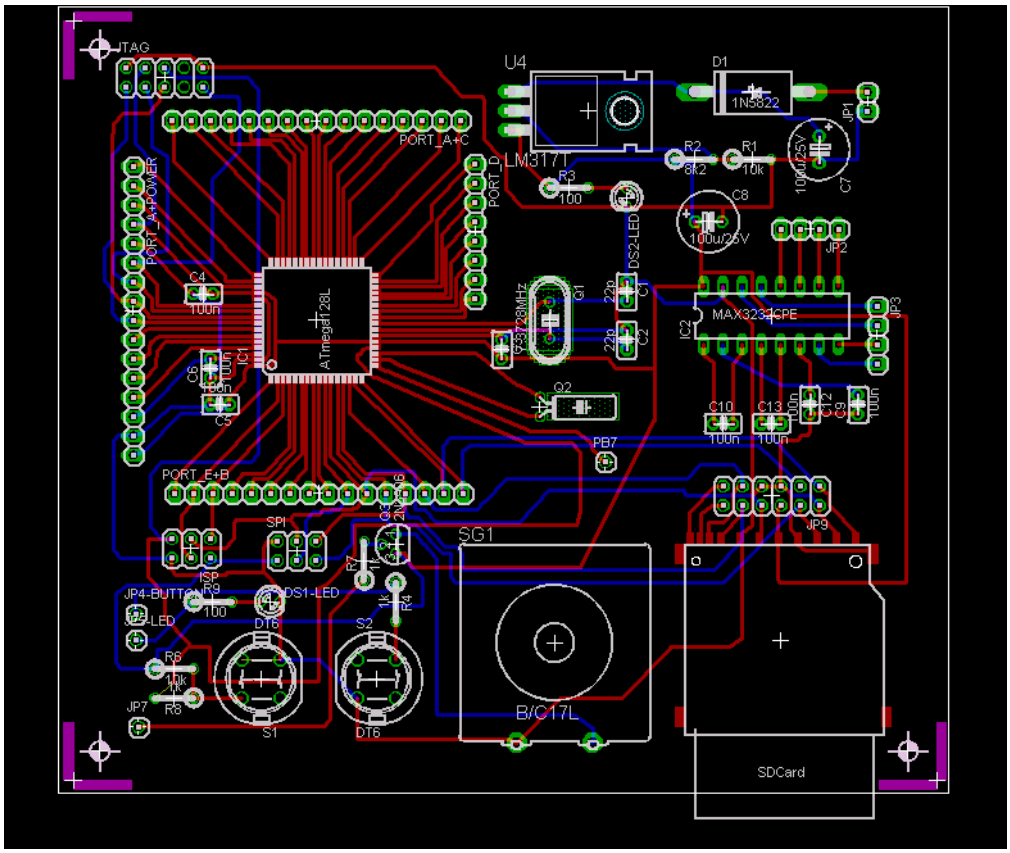
[17] Rabbit Semiconductor. *Appendix A. More FAT Information.* Rev. F [online] [cit. 2007-05-07]. Dokument dostupný na URL: <http://www.rabbitsemiconductor.com/documentation/docs/modules/FileSystem/ModFAT40.htm>

[18] DIY DataRecovery.nl. *MBR (Master Boot Record)* [online]. Posledné úpravy 2006-11-28 [cit. 2007-05-07]. Dokument dostupný na URL: <http://www.diydatarecovery.nl/kb_mbr_article.htm>

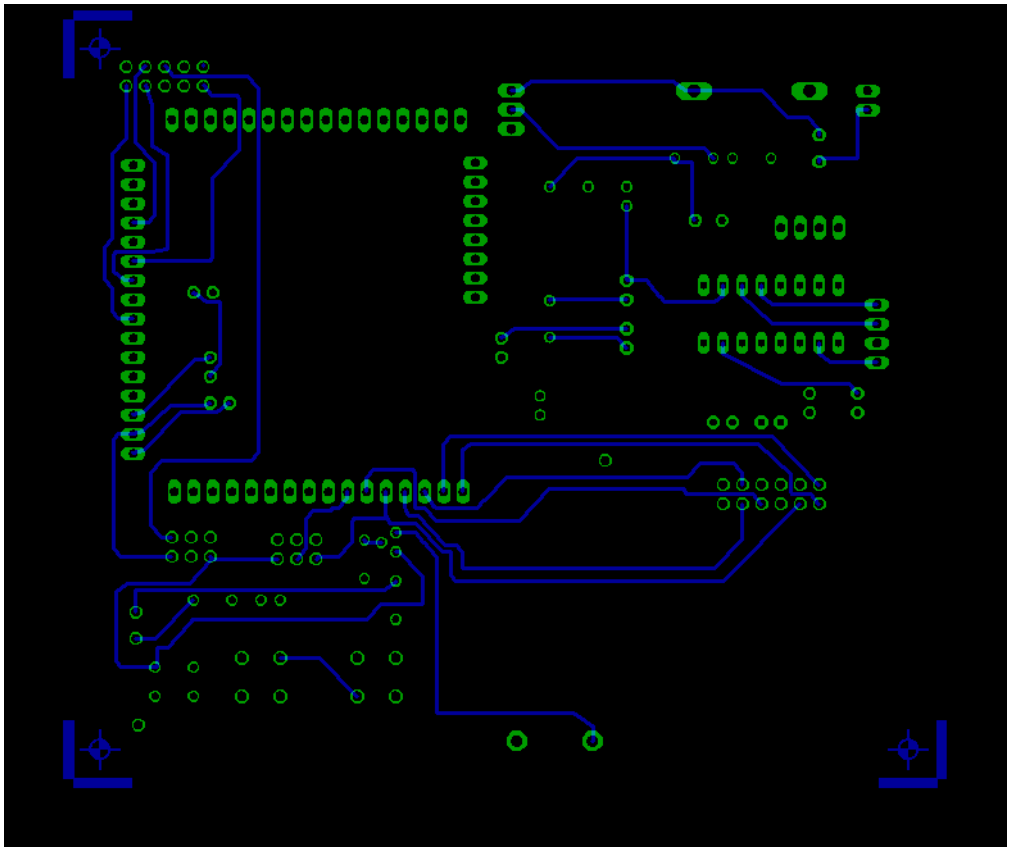
[19] Ing. Karel Radkovský. *Jtag Ice* [online]. [cit. 2007-05-08]. Dokument dostupný na URL: <http://www.dioda.cz/index.php?PageId=7>

Zoznam príloh

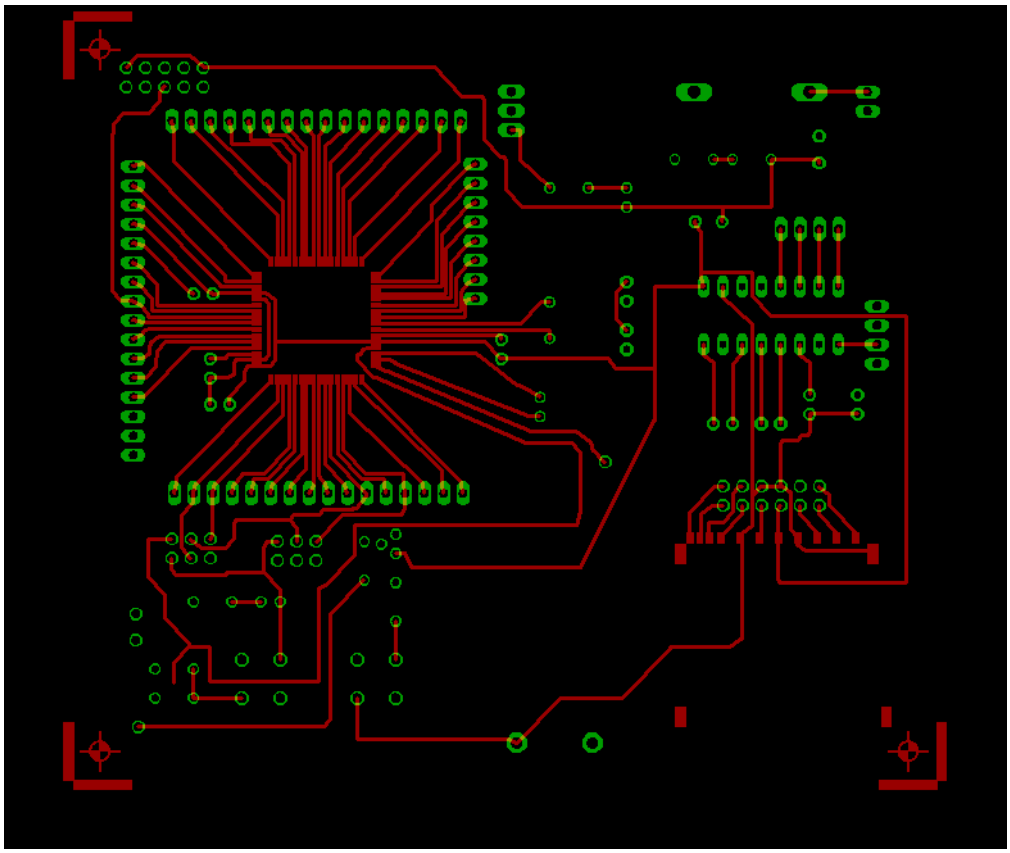
Príloha 1 Pohľad na DPS – osadenie súčiastok



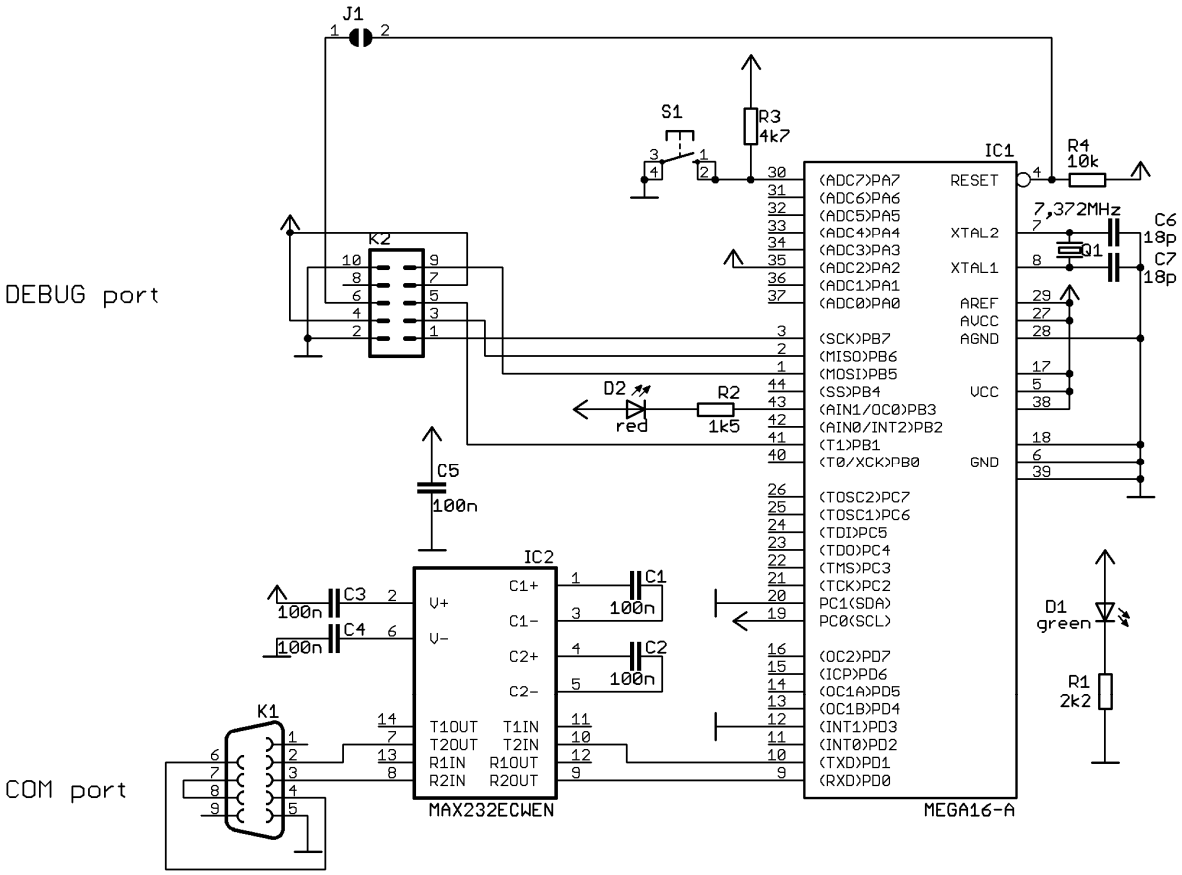
Príloha 2 Pohľad na DPS zo spodnej strany spojov



Príloha 3 Pohľad na DPS z vrchnej strany súčiastok



Príloha 4 Schéma programátoru JTAG [19]



Príloha 5 Hlavičkový konfiguračný súbor config.h

```
/* ============================================================*/
/*   config.h
*/
/*   (c) 2007 Pavel Laurinc
*/
/*
*/
/*   Description
*/
/*     Configuration file for project.
*/
/* ============================================================*/


#ifndef __CONFIG_H_
#define __CONFIG_H_

#include <stdio.h>
#include <string.h>
#include <avr/io.h>
#include <avr/eeprom.h>
#include <avr/interrupt.h>
#include <avr/signal.h>

// determine the card mode
#define SPI_HW 0  //    0...SPI mode (HW)
```

```
#define SPI_SW 1  //    1...SPI mode (SW)
#define SD_SW  2  //    2...SD  mode (SW)
#define CARD_MODE           SPI_HW

// Default Sector size [Bytes]
#define SECTOR_SIZE         512

// define PORTs and PINs for SPI mode (SD_* is in SD Card!)
// PORTx is output: Master(uC) -> Slave(SD Card)
// PINx  is input:  Master(uC) <- Slave(SD Card)
#if (CARD_MODE == SPI_SW || CARD_MODE == SPI_HW)
#define SD_CS               PORT6
#define SD_DI               PORT2
#define SD_SCLK             PORT1
#define SD_DO               PIN3

#define DDR_SPI             DDRB
#define PORT_SPI            PORTB
#define PIN_SPI             PINB
#define DD_MOSI             SD_DI
#define DD_MISO             SD_DO
#define DD_SCK              SD_SCLK
#define DD_SS               PIN0
#define SD_CINS             PIN5
#else
// define PORTs and PINs for SD mode
#define SD_PORT             PORTA
#define SD_DDR              DDRA
#define SD_PINS             PINA

#define SD_DAT3               PIN6
#define SD_CMD              PIN1
#define SD_CLK              PORT2
#define SD_DAT0             PIN3
#define SD_DAT1             PIN4
#define SD_DAT2             PIN5
// detect insertion of card
#define SD_CINS             PIN0
#endif
// define my types
typedef unsigned char       BYTE;
typedef unsigned short int  WORD;
typedef unsigned long  int  DWORD;

// define some useful macros
#define set_bit(sPort, sBit)    sPort |= _BV(sBit)
#define clr_bit(cPort, cBit)    cPort &=~_BV(cBit)
#define out_bit(oPort, oBit, isSet) {if(isSet) set_bit(oPort, oBit);\
                                    else      clr_bit(oPort, oBit);}
#define NOP asm("nop");
#define ERROR 1
#define OK    0

#if (CARD_MODE == SPI_HW || CARD_MODE == SPI_SW)
// define SPI commands for SD Card
#define GO_IDLE_STATE       0   // response R1
//#define SEND_OP_COND        1   // response R1  OLD: use with MMC cards
#define SEND_CID            10  // response R1
#define SEND_CSD            9   // response R1
#define SET_BLOCKLEN        16  // response R1
#define READ_SINGLE_BLOCK   17  // response R1
```

```
#define WRITE_BLOCK          24  // response R1
#define ERASE_WR_BLK_START   32  // response R1
#define ERASE_WR_BLK_END     33  // response R1
#define ERASE                38  // response R1b
#define READ_OCR             58  // R3
#define APP_CMD              55  // R1
#define SD_SEND_OP_COND      41  // R1
#else
// define SD mode commands for SD Card
#define GO_IDLE_STATE        0   // no response
#define SEND_IF_COND         8   // R7
#define APP_CMD              55  // param RCA, response R1
#define SD_SEND_OP_COND      41  // response R3
#define SEND_CID             10  // response R2
#define ALL_SEND_CID         2   // response R2
#endif

//  CID register
typedef struct{
  BYTE  manId;
  WORD  appId;
  BYTE  name[5];
  BYTE  rev;
  DWORD psn;
  WORD  dataCode;
  BYTE  crc7;
}CID;

#endif
```

Príloha 6 Hlavičkový súbor fat.h

```
/* ================================================================= */
/*
/*   fat.h
*/
/*   (c) 2007 Pavel Laurinc
/*   Description
*/
/*     HEADER for Master Boot record and FAT file system functions.
*/
/* ================================================================= */


#ifndef __FAT_H_
#define __FAT_H_
#include "config.h"
#if (CARD_MODE == SPI_HW || CARD_MODE == SPI_SW)
#include "spi.h"
#else
#include "sd.h"
#endif

//  Define some useful macros
//  compute the first sector of N-th cluster
#define FIRST_SECTCLUST(N)  (((N-2) * secPerClus) + firstDataSector +
volumeOffset)
#define THIS_FATSEC_NUM(N)  (resvdSecCnt+((N*fatOffset)/bytsPerSec) +
volumeOffset)
```

```
#define THIS_FATENT_OFS(N)  ((N*fatOffset) % bytsPerSec)

//                  ----  MBR  ----
//  structure for Partition Table in MBR(Master Boot Sector)
//  Count of partiton tables is 4. The first is at offset 0x01BE.
#define FIRST_PART_TABLE_OFFSET   0x01BE
/* Resource of this table is URL:
http://www.diydatarecovery.nl/kb_mbr_article.htm
|=====================================================|
| Byte Count | Description of contents                |
|=====================================================|
|      1     | Boot indicator (0x00 off, 0x80 on)     |
|=====================================================|
|      3     | Starting head, cylinder and sector     |
|=====================================================|
|      1     | Filesystem descriptor                  |
|=====================================================|
|      3     | Ending head, cylinder and sector       |
|=====================================================|
|      4     | Starting sector (offset to disk start  |
|=====================================================|
|      4     | Number of sectors in partition         |
|=====================================================|
*/
//  Size of partition table is 16 Bytes.
typedef struct{
  BYTE  bootIndicator;
  BYTE  startHead;
  BYTE  startCylinder;
  BYTE  startSector;
  BYTE  fsDescriptor;
  BYTE  endHead;
  BYTE  endCylinder;
  BYTE  endSector;
  DWORD startingSectorOffset;   // Most important value (start sector of
volume)
  DWORD sectors;                // indicates number of sectors in
partition
}PartitionTable;

//                  ----  FAT  ----
#define FAT12 12
#define FAT16 16
#define FAT32 32

#define FAT_LAST_CLUSTER  0x0FFFFFF
#define FAT_FREE_CLUSTER  0x0000000
//  Structure for Boot Sector and BPB (size: 36 Bytes)
#define FATEXSTRUCT_OFFSET  0x24  // == 36 Bytes
typedef struct{
  BYTE  BS_jmpBoot [3];   // 3
  BYTE  BS_OEMName [8];
  WORD  BPB_BytsPerSec;
  BYTE  BPB_SecPerClus;
  WORD  BPB_ResvdSecCnt;
  BYTE  BPB_NumFATs;
  WORD  BPB_RootEntCnt;
  WORD  BPB_TotSec16;
  BYTE  BPB_Media;
  WORD  BPB_FATSz16;
  WORD  BPB_SecPerTrk;
```

```
  WORD  BPB_NumHeads;
  DWORD BPB_HiddSec;
  DWORD BPB_TotSec32;   //  4
}BootSector;
//  Structure for BPB/Boot Sector for FAT12/FAT16 (size: 26 Bytes)
typedef struct{
  BYTE  BS_DrvNum;
  BYTE  BS_Reserved1;
  BYTE  BS_BootSig;
  DWORD BS_VoIID;
  BYTE  BS_VolLab[11];
  BYTE  BS_FilSysType[8];
}BPB_ExFAT16;
//  and for FAT32
typedef struct{
  DWORD BPB_FATSz32;
  WORD  BPB_ExtFlags;
  WORD  BPB_FSVer;
  DWORD BPB_RootClus;
  WORD  BPB_FSInfo;
  WORD  BPB_BkBootSec;
  BYTE  BPB_Reserved[12];
  BYTE  BS_DrvNum;
  BYTE  BS_Reserved1;
  BYTE  BS_BootSig;
  DWORD BS_VolID;
  BYTE  BS_VolLab[11];
  BYTE  BS_FilSysType[8];
}BPB_ExFAT32;
//   structure for 32 Bytes directory entry + 6 additional Bytes.
#define DIRENTRY_SIZE 32  //  don't add 6 last Bytes!
typedef struct{
  BYTE  name[11];
  BYTE  attr;
  BYTE  NTRes;
  BYTE  crtTimeTenth;
  WORD  crtTime;
  WORD  crtDate;
  WORD  lstAccDate;
  WORD  fstClusHi;
  WORD  wrtTime;
  WORD  wrtDate;
  WORD  fstClusLo;
  DWORD fileSize;
  // additional info, use for file descriptor
  DWORD clustEntry;
  BYTE  offsetSecEntry;
  BYTE  offsetEntry;
}DIR_ENTRY;
//  define dir/file attributes:
#define READ_ONLY 0x01
#define HIDDEN    0x02
#define SYSTEM    0x04
#define VOLUME_ID 0x08
#define DIRECTORY 0x10
#define ARCHIVE   0x20
#define LONG_NAME 0x0F

//  determine if dir entry is free or not(the first byte in every entry)
#define DIRENT_FREE     0xE5
#define DIRENT_FREESPEC 0x00
```

```
//  My File Descriptor structure
typedef struct{
  BYTE  attr;           //  Attributes from dir entry.
  DWORD clusterN;       //  Index of first cluster of file.
  DWORD fileSize;       //  Size of file [in Bytes].
  DWORD clustEntry;     //  Adress of cluster, where this entry is
located.
  BYTE  offsetSecEntry; //  Offset of sector in cluster.
  BYTE  offsetEntry;    //  Offset of this entry in sector.
  DWORD curClust;       //  Cluster of actual position in file (used by
seek)
  BYTE  curOffset;      //  Offset in cluster of actual position in file
  DWORD posInFile;      //  Position in file [0..Size of file]
}FILE_DESC;
#define ROOT_DIR  0x50  // 0x40|DIRECTORY = 0x40|0x10
//  Functions prototypes
//  PRIVATE:
//  Reading partition table selected by partNum
void ReadPartitonTable(BYTE partNum, PartitionTable *table);
void ReadBootSector(DWORD sector, BootSector *bootSector);
void ReadExFAT16(DWORD sector, BPB_ExFAT16* exFat);
void ReadExFAT32(DWORD sector, BPB_ExFAT32* exFat);
DWORD FindFreeCluster(void);
DWORD FindLastCluster(DWORD cluster);
DWORD GetNextCluster(DWORD cluster);
void  SetFATcluster(DWORD cluster, DWORD value);
BYTE  CacheReadSector(DWORD sector, BYTE* cBuffer);
BYTE  CacheWriteSector(DWORD sector, BYTE* cBuffer);

//  Real cache functions, call StartRealCaching() before read/write
//  if you want to write data to card without waiting,
//  you must call FlushWrite() !!!
//  At the end of real caching call always FlushWrite()!!!
BYTE  RealCacheReadSector(DWORD sector, BYTE* cBuffer);
BYTE  RealCacheWriteSector(DWORD sector, BYTE* cBuffer);
void StartRealCaching(void);
void FlushWrite(void);
//  EXPORT:
BYTE FatInit(BYTE* tempBuffer);
//  Open directory and return file(dir) descriptor.
//  dirEnt...info about dir
//  *parent..descriptor of parent dir,
//           if(parent == NULL || dirEnt == NULL) parent = ROOT_DIR
FILE_DESC OpenDir(DIR_ENTRY *dirEnt, FILE_DESC *parent);
DIR_ENTRY GetNextFileInDir(FILE_DESC* dir);
FILE_DESC OpenFile(DIR_ENTRY dirEnt);

//  Read one sector of file
//  return... On  success,  the number of Bytes read is returned (zero
indicates end
//            of file)
WORD  ReadFile(FILE_DESC* file, BYTE* data);

//  Append data to the file, max length is limited to 512Bytes!
WORD  AppendFile(FILE_DESC* file, BYTE* data, WORD length);
//  Secure erase indicate that all data in file will be replace with 0 or
1
#define SEC_ERASE_ENABLE  1
#define SEC_ERASE_DISABLE 0
BYTE  DeleteFile(FILE_DESC* file, BYTE secErase);
```

```
//  Calculate available (free) space
//  return...value of free space in Bytes
DWORD GetFreeSpace(void);

//  Get total space in Bytes
//  return...value of total space on the disk(SD card)
DWORD GetTotalSpace(void);
#endif
```

Príloha 7 Ukážkový príklad použitia aplikácie

```
/* ================================================================= */
/*
*/
/*   main.c
*/
/*   (c) 2007 Pavel Laurinc
*/
*/
/*   Description
*/
/*     Implementation of main loop of program.
/* ================================================================= */

#include "config.h"
#include "fat.h"

int main()
{
  FILE_DESC fdd;
  DIR_ENTRY ded;
  BYTE buffer[512];
  BYTE resp = 0;
  BYTE i = 0;
  CID cidReg;
  // inicializácia, v prípade chyby koniec programu
  SPI_MasterInit();
  resp = SPI_SlaveInit();
  if(resp == ERROR)
    return 0;
  // takto je možné získať CSD register
  SendCommand(SEND_CSD, 0x00, 0x00, 0x00, 0x00, 0xFF);
  resp = ReadDataBlock(buffer, 16);
  // a CID register
  SendCommand(SEND_CID, 0x00, 0x00, 0x00, 0x00, 0xFF);
  resp = ReadDataBlock(&(buffer[20]), 16);
  // alebo CID register ešte inak:
  cidReg = ReadCID();

  // Initalization of FAT file system
  FatInit(buffer);

  // ukážem ako otvoriť korenový adresár
  fdd = OpenDir(NULL, NULL);
  do{
    // a preskúmať celý jeho obsah
    ded = GetNextFileInDir(&fdd);
```

```
    if(ded.attr != DIRECTORY && ded.attr != ROOT_DIR &&
      ded.name[0]!=DIRENT_FREE && ded.name[0]!=DIRENT_FREESPEC)
    {
      // otvoriť každý súbor
      FILE_DESC fdfile = OpenFile(ded);
      if(ded.name[0] == 'G')
      {
        WORD readBytes = 0;
        // prečítať celý obsah súboru začínajúceho na písmeno G
        while((readBytes = ReadFile(&fdfile, buffer)) > 0);
        // a doplnit ho textom: "doplneny text")
        readBytes = AppendFile(&fdfile, "doplneny text", 13);
      }
      // zmažeme všetky súbory, ktoré začínajú na písmeno 'K'
      if(ded.name[0] == 'K')
        // bezpečná deštrukcia dát je zapnutá
        DeleteFile(&fdfile, SEC_ERASE_ENABLE);

    }
    // skúmam až kým nenarazím na koniec adresára
  }while(ded.name[0] != 0);

  while(1)
  {

  }

  return 0;
}
```