ČVUT v Praze

Fakulta elektrotechnická

DIPLOMOVÁ PRÁCE

2006 **Martin Řehák**

České vysoké učení technické v Praze, fakulta elektrotechnická

Katedra: mikroelektroniky **Školní rok:** 2004/2005

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Martin Řehák
Obor: elektronika
Název tématu: **Datalogger se zápisem na CF kartu**

Z á s a d y p r o v y p r a c o v á n í:

1. Prostudujte problematiku adresování, zápisu a čtení CF karet.
2. Navrhněte schéma dataloggeru.
3. Realizujte funkční vzorek.

Seznam odborné literatury:

[1] Dokumentace k procesoru ATMEGA.
[2] Záhlava, V.: Metodika návrhu plošných spojů, Vydavatelství ČVUT, Praha 2004
[3] Dokumentace a specifikace CF karet.

Vedoucí diplomové práce: **Ing. Vít Záhlava, CSc.**

Datum zadání diplomové práce: **30. 1. 2005**

Termín odevzdání diplomové práce: **27. 1. 2006**

[ZDE VLOŽIT ORIGINÁLNÍ ZADÁNÍ]

Prof. Ing. Miroslav Husák, CSc.
vedoucí katedry

Prof. Ing. Vladimír Kučera, DrSc.
děkan

V Praze dne 30. 1. 2005

Poděkování:

Zde bych poděkoval svému vedoucímu diplomové práce Ing. Vítu Záhlavovi, CSc. za cenné rady a připomínky a také svým rodičům za podporu během studia.

Prohlášení:

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, software atd.) uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 26. 1. 2006 podpis .....

Anotace:

Tato diplomová práce se zabývá připojením paměťových karet standardu Compact Flash a pevných disků IDE k jednočipovému mikropočítači. Cílem je navrhnout jednoduché zařízení a obslužný software, které umožní záznam a čtení dat (z vnějšku nebo interně připojených senzorů) na tato levná vysokokapacitní média. Praktická část popisuje konkrétní realizaci zařízení založenou na mikropočítači Atmel ATmega128 a obslužný software ve formě knihoven v programovacím jazyce C.

Annotation:

This master thesis deals with the Compact Flash memory cards and IDE hard drives to the single-chip microcontroller connection possibilities. The main goal of this work is to develop the microcontroller based recording device including the software equipment to log up the data (from external source or internally attached sensors) to these high capacity data storages. The particular hardware solution based on the Atmel ATmega128 microcontroller as well as the software implementation is described in the practical part of this work.

OBSAH

POUŽITÉ ZKRATKY A POJMY

/	prefix negace (signál je aktivní v log. 0)
0b...	prefix binárních čísel
0x...	prefix hexadecimálních čísel
ASCII	American Standard Code for Information Interchange – tabulka kódování znaků a číslic pomocí 8-bit slova používaná na PC.
API	Application Programming Interface - rozhraní pro programování aplikací
ATA	AT-Attachment – standard připojení pevných disků PC-AT
bit	binary digit – dvojková číslice s hodnotami 0 nebo 1
B, Byte	Byte – jednotka kapacity, slovo o délce 8 bitů
BIOS	Basic Input/Output System – základní systém pro vstup/výstup
BPB	BIOS Parameter Block – blok parametrů BIOSu
CF	Compact Flash – záznamové médium založené na paměti Flash EPROM
CHS	Cylinder-Head-Sector – adresování číslem cylindru, hlavy a sektoru
cluster	blok o daném počtu sektorů (typicky násobky 2^n)
DBR	DOS Boot Record – záznam zavaděče DOSu
DMA	Direct Memory Access – Přímý přístup do paměti (bez účasti procesoru)
DOS	Disk Operating Systém – diskový operační systém
DPS	Deska Plošného Spoje
DWord	dvojslovo o délce 32 bitů, zde uvažujeme pořadí Little Endian
ECC	Error Correction Code – mechanismus pro zjištění a opravu chyb
FAT	File Allocation Table – alokační tabulka souborů
Flash EPROM	Flash Erasable Programmable Read-Only Memory – elektricky mazatelná programovatelná paměť
GND	GrouND – zemní potenciál
HDD	Hard Disk Drive – jednotka pevného disku
IDE	Integrated Drive Electronic – jednotka s integrovanou elektronikou
LBA	Logical Block Address – logické adresování (nebo adresa) bloků
LFN	Long File Name – dlouhý název souboru/adresáře, až 255 znaků
Little Endian	pořadí bytů ve slově je od LSB k MSB (platforma intel a další)
LSb	Least Significant bit – nejméně významný (nejnižší) bit slova
LSB	Least Significant Byte – nejméně významný (nejnižší) byte slova
MBR	Master Boot Record – hlavní záznam zavaděče
MSb	Most Significant bit – nejvíce významný (nejvyšší) bit slova
MSB	Most Significant Byte – nejvíce významný (nejvyšší) byte slova
PAT	PArtition Table – tabulka diskových oddílů

PCMCIA	Personal Computer Memory Cards International Association – sdružení, které vydalo standard původně rozšiřujících paměťových karet (později i periferií) pro přenosné počítače
PIO	Programmed Input/Output – režim řízení přes I/O porty (nikoliv DMA)
RAM	Random Access Memory – přepisovatelná paměť s náhodným přístupem
RISC	Reduced Instruction Set Computer – Počítač s redukovanou instrukční sadou
sektor	blok dat o dané velikosti (typicky 512B u pevných disků)
SMD	Surface Mounted Device – součástka pro povrchovou montáž
THD	Through Hole Device – součástka pro klasickou montáž s vývody skrz
UTF	Unicode Transformation Format – další způsob kódování znaků a číslic podle standardu Unicode, UTF-8 kóduje znaky do 1-4 bytů, UTF-16 kóduje znaky do 16-bitových slov.
Word	slovo o délce 16 bitů, zde uvažujeme pořadí Little Endian

1 ÚVOD

V dnešní době řídí spousty technologických procesů, měření a testů počítače. Komplexnost a objem zpracovávaných dat stále narůstá. Ne vždy je však možné nebo vhodné použití klasického či průmyslového PC s dostatečnou záznamovou kapacitou. V řadě případů se k řízení používají jednoduché průmyslové automaty, které ale nebyly původně určeny k zaznamenávání velkého množství dat. Kompletní modernizace by byla nákladná, vhodnějším řešením je doplnit systém o jednoduché jednoúčelové zařízení na záznam dat (datalogger), které se připojí přes zvolené existující standardizované rozhraní.

Základním požadavkem na takové zařízení je dostatečná záznamová kapacita a snadná přenositelnost dat na PC pro další zpracování. V konkrétním návrhu jsem zvolil jako záznamové médium paměťovou kartu Compact Flash pro své výhodné vlastnosti:

- kapacita v rozsahu 8 MB až 12 GB (stále ve vývoji),
- kompaktní rozměry a malá spotřeba,
- vyšší mechanická odolnost (neobsahuje mech. pohyblivé části),
- kompatabilita s PC a PDA (USB čtečky, IDE/PCMCIA redukce),
- dostupnost a cena.

Další možnou alternativou jsou jiné typy paměťových karet, jejichž kompatabilita ale není tak široká nebo přenosné USB Flash disky („klíčenky“), které se zas omezují na dostupnost USB portu.

Aby byla zaznamenaná data snadno přenositelná na PC, je třeba zvolit vhodný souborový systém běžně podporovaný v současných operačních systémech. Zvolil jsem souborový systém FAT16, který má i přes některé své omezení a nedostatky širokou podporu (nejen na PC) a zároveň je relativně jednoduchý, což umožňuje snadnou implementaci i na jednočipovém mikropočítači.

V následujícím textu bude nejprve popsán princip pevného disku a rozhraní ATA, které je též společné kartám Compact Flash v režimu kompatability TrueIDE. Stručně vysvětlím také princip Flash pamětí. Další část se zabývá souborovým systémem FAT16 a datovými strukturami, které používají operační systémy na PC pro organizaci a ukládání dat. V praktické části je popsán konkrétní návrh a realizace hardware dataloggeru s využitím jednočipového počítače Atmel ATmega128 a komunikačního rozhraní RS232. Poslední část se zabývá návrhem software, kde bylo požadováno vytvoření sady knihoven – jednoduchého API, které bude poskytovat základní souborové funkce pro zápis a čtení dat.

2 TEORETICKÝ ROZBOR

2.1 PRINCIP ZÁZNAMU DAT NA PEVNÝ DISK

Na začátku bych stručně zmínil princip záznamu dat na pevný disk. Ačkoliv karty Compact Flash pracují na úplně jiném principu, mají velice podobné rozhraní (mohou pracovat v režimu kompatabilním s běžnými IDE disky) a používají se zde některé shodné pojmy jako u pevných disků.

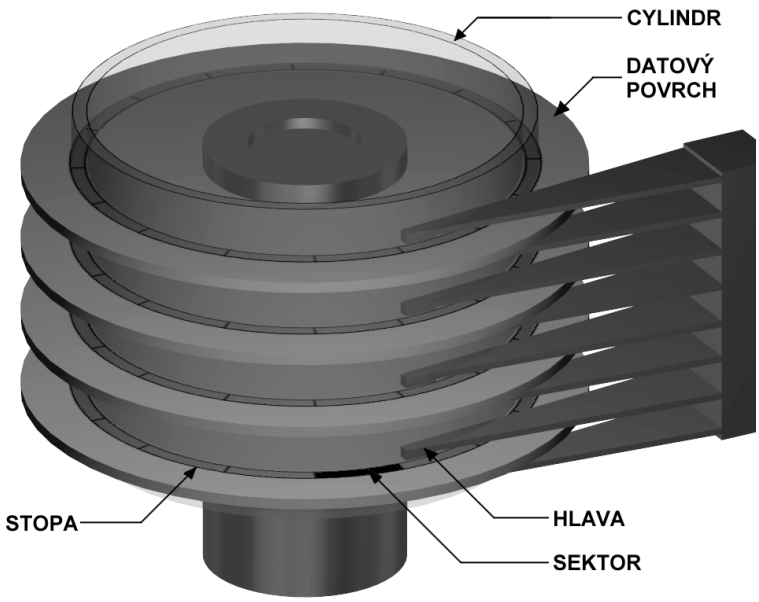
Pevné disky jsou založeny na principu magnetického záznamu, který se používá více než 100 let (drátofony, magnetofony, video, datové pásky, diskety, atd.). V podstatě jde o to, že máme kus magneticky tvrdého materiálu (FeOx, FeNiCr, tvrdé ferity), který si po zmagnetování záznamovou cívkou, kdy se orientují domény materiálu ve směru magnetického pole, udrží svou orientaci dlouho i po zániku pole. Snímací cívkou pak lze (opakovaně) zjistit jak byl materiál magnetován a získat tak zpět zaznamenanou informaci. Magnetický záznam lze smazat buď pomocí vysokofrekvenčního či stacionárního magnetického pole, nebo zahřátím nad Curieovu teplotu (záleží na materiálu, typicky několik set °C).

Nejjednodušší praktické využití je v podobě magnetického pásku, jenž se převíjí mezi dvěma kotouči a je magnetován a snímán dvěma statickými magnetickými hlavami. Pro využití ve výpočetní technice má však zásadní nevýhodu - doba přístupu k náhodně vybraným datům je velmi dlouhá.

Proto bylo vymyšleno nové, zcela odlišné uspořádání. Magnetické záznamové médium v podobě kruhového rotujícího disku, nad nímž se na tenkém (řádově μm) vzduchovém polštáři pohybují hlavy (nedochází k fyzickému kontaktu hlavy a média a tedy opotřebení povrchu). První pevný disk vyrobila firma IBM už v roce 1955, používal padesát 24" ploten a pouze jednu hlavu, kapacita dosahovala 5 MB. Více z historie zde [1].

U běžných disků se používá jedna hlava pro každý aktivní datový povrch (někdy se využívá pouze jeden povrch plotny), viz obr. 2.1.1. Hlavy jsou umístěné na ramínkách a pohybují se všechny současně od kraje ke středu po určitých diskrétních krocích. Pro každý krok si můžeme představit jak hlava opisuje na povrchu plotny kružnici určitého průměru – **stopu** (čísluje se od 0). Souhrn všech stop na jednotlivých plotnách stejného průměru tvoří tzv. **cylindry** (čísluje se od 0). Konkrétní stopa je tedy určena číslem **hlavy** (čísluje se též od 0). Každá stopa je pak dále rozdělena na **sektory** (čísluje se od 1) pevně dané velikosti (typicky 512 B).

Tím byl položen základ tzv. **CHS** (Cylinder-Head-Sector) diskové geometrie, která jednoznačně určuje dané místo na disku (lze chápat jako 3D adresu). Bohužel tehdy na PC programátoři diskových služeb BIOSu INT 0x13 rozvrhli bity registrů pro CHS adresu poměrně nešikovně (10 bitů pro číslo cylindru, 8 bitů pro číslo hlavy a 6 bitů pro číslo sektoru) a tak se začalo brzy narážet na limit 1024 cylindrů. Naopak 256 fyzických hlav asi žádný disk



Ilustrace 2.1.1: Princip HDD

pro PC nikdy nevyužil. Navíc se také u disků IDE změnila metoda záznamu na ZBR (Zone Bit Recording), kde už není konstantní počet sektorů na stopu, ale směrem ke kraji se zvětšuje. Kvůli kompatabilitě byla zavedena **translace** CHS geometrie, kterou provádí transparentně elektronika disku, jenž se pak navenek chová tak, jako by měl více hlav a méně cylindrů, aby čísla CHS padla do povolených limitů. Časem se ale stejně narazilo na limit 2^{24} sektorů, tj. 8,4 GB. Více např. v [2].

Zavedlo se tedy nové lineární adresování sektorů **LBA**, které využívalo ze začátku 28-bitovou lineární adresu (limit 128 GB) a nyní 48-bitovou lineární adresu (limit 131072 TB). Moderní operační systémy používají LBA přístup prostřednictvím ovladače, který komunikuje přímo s řadičem disku bez účasti BIOSu. Kvůli kompatabilitě se i do BIOSu přidaly nové funkce – rozšíření INT 0x13, které pracují s LBA. Umí je využít např MS-DOS 7.0+ nebo FreeDOS. Také disky ještě stále obsahují informaci o CHS, která už však nemá nic společného s fyzickou geometrií. CHS adresu lze na LBA přepočítat podle vztahu 2.1.1

Vztah 2.1.1:

$$LBA = (((c \cdot H) + h) \cdot S) + s - 1$$

kde:

H je celkový počet hlav
S je počet sektorů na stopu
c je číslo cylindru
h je číslo hlavy
s je číslo sektoru

a zpětně z LBA adresy lze při znalosti CHS geometrie odvodit čísla cylindru, hlavy a sektoru podle vztahu 2.1.2.

Vztah 2.1.2:

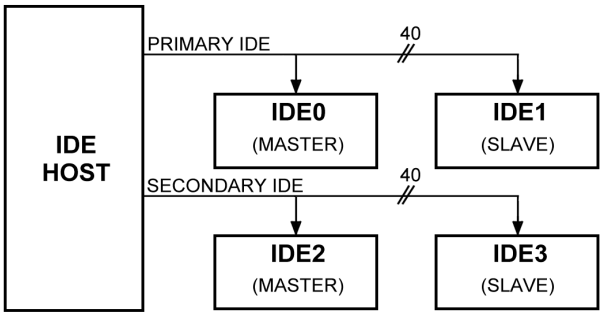
$$c = (LBA / S) / H$$
$$h = (LBA / S) \bmod H$$
$$s = (LBA \bmod S) + 1$$

kde:

H je celkový počet hlav
S je počet sektorů na stopu
mod je operátor modulo

2.2 ROZHRANÍ IDE/ATA

Starší pevné disky měli poměrně jednoduchou elektroniku, která většinou jen uměla posunout hlavy a přečíst/zapsat surová data. Všechny vyšší operace obstarával řadič disku. Pro zvýšení spolehlivosti a rychlosti přenosu a snížení ceny byl roku 1986 firmami Imprimis, Western Digital a Compaq vytvořen nový standard pro připojení disků nazývaný **IDE** nebo **ATA**. IDE znamená Integrated Drive Electronic – jednotka s integrovanou elektronikou, což vystihuje fakt, že disk už má na sobě inteligentní elektroniku a nepotřebuje žádný speciální řadič. ATA znamená AT-Attachment, což zase říká, že disk má téměř všechny signály společné se sběrnicí AT-bus (16-bitová ISA). V praxi se disk nepřipojuje přímo, ale přes oddělovače/ posilovače sběrnic kvůli větší odolnosti proti rušení a dovolenému zatížení sběrnice. Na jeden kanál IDE „řadiče“ lze připojit až 2 disky, které sdílí všechny signály. Aby nedocházelo ke kolizi, musí se jeden disk nastavit do režimu MASTER a druhý do režimu SLAVE, to se provede buď pomocí jumperu na disku nebo pinem CABLE SELECT na datovém konektoru. V jednom okamžiku pochopitelně probíhá komunikace pouze s jedním diskem a druhý je neaktivní. PC mívají standardně 2 nebo 4 kanály, což umožňuje připojit až 4 resp. 8 disků (obr. 2.1.1).

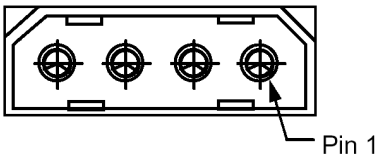


Ilustrace 2.2.1: Typická konfigurace IDE zařízení v PC

Později byl vyvinut protokol ATAPI, který umožňuje posílat některé SCSI příkazy po rozhraní ATA. Ty využívají například optické mechaniky CD/DVD-ROM, které tak mohly přejít z dražšího provedení pro SCSI sběrnici na levnější ATA rozhraní. V roce 2003 bylo ATA po uvedení nového standardu Serial ATA přejmenováno na Parallel ATA (PATA). Více z historie ATA standardu např. v [3].

2.2.1 Popis fyzického a elektrického rozhraní (40-pin)

Dále popíši zapojení klasického 40-pinového konektoru s roztečí 2,54 mm (obr. 2.2.1.2, tab. 2.2.1.2), kterým jsou vybaveny všechny 3,5" IDE disky a význam signálů. Napájení 3,5" disků je zajištěno přes 4-pinový konektor Molex (obr. 2.2.1.1, tab. 2.2.1.1). Pro 2,5" disky se používá menší 44-pinový konektor s roztečí 2 mm, který v sobě zahrnuje i napájení (pouze 5 V – piny 41, 42 a GND – pin 43).

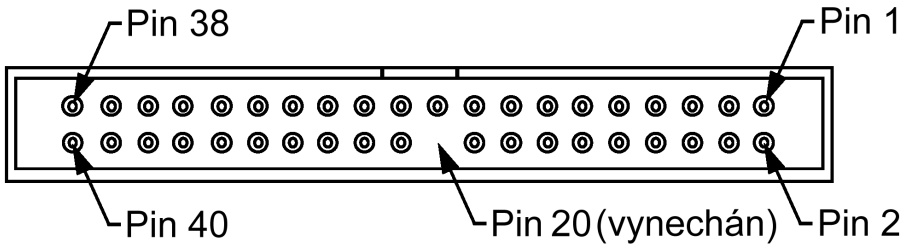


Ilustrace 2.2.1.1: Napájecí konektor Molex (na straně HDD)

Tabulka 2.2.1.1: Zapojení napájecího konektoru Molex

pin	popis
1	+12 V / 400 mA (žlutá)
2	GND (černá)
3	GND (černá)
4	+5 V / 270 mA (červená)

Pozn.: uvedený proudový odběr je zde pouze pro představu a platí pro pevný disk WD 33100 AC v ustáleném stavu, při rozběhu hnacího motoru je několikanásobně větší. Barvy platí pro odpovídající napájecí konektor PC zdroje.



Ilustrace 2.2.1.2: Datový 40-pin konektor (na straně HDD)

Tabulka 2.2.1.2: Zapojení datového konektoru IDE 40-pin

pin	popis	směr
1	/RESET	I
2	GND	-
3	D7	I/O
4	D8	I/O
5	D6	I/O
6	D9	I/O
7	D5	I/O
8	D10	I/O
9	D4	I/O
10	D11	I/O
11	D3	I/O
12	D12	I/O
13	D2	I/O
14	D13	I/O
15	D1	I/O
16	D14	I/O
17	D0	I/O
18	D15	I/O
19	GND	-
20	KEYPIN (neosazený)	-
21	DMARQ	O, 3S
22	GND	-
23	/IOWR	I
24	GND	-
25	/IORD	I
26	GND	-
27	IORDY	O, 3S
28	CABLE SELECT	I
29	/DMAACK	I
30	GND	-
31	INTRQ	O, 3S
32	/IOCS16	O, OC
33	A1	I

pin	popis	směr
34	/PDIAG	I/O
35	A0	I
36	A2	I
37	/CS0	I
38	/CS1	I
39	/DASP	I/O, OC
40	GND	-

Pozn.: signály s „/“ na začátku jsou aktivní v log. 0

/RESET – tento signál se generuje inverzí ze signálu RESET sběrnice ISA, inicializuje disk do výchozího stavu.

D0-D15 – datové linky po kterých se přenáší příkazy a data. všechny linky D0-D15 se využívají pouze k přenosu obsahu datového registru, ostatní registry se přenáší jen pro D0-D7.

KEYPIN – obvykle je vynechán a slouží jako klíč, aby nešel protikus konektoru nasadit obráceně.

DMARQ – slouží pro vyslání žádosti o přímý přístup do paměti při DMA přenosu. V PC se obvykle používá DMA kanál 1 nebo 3. V PIO režimu se nepoužívá.

/IOWR – při zápisovém cyklu generuje řadič pro disk negativní impuls minimální délky.

/IORD – při čtecím cyklu generuje řadič pro disk negativní impuls minimální délky.

IORDY – tímto signálem disk v log. 1 oznamuje, že je schopen přijímat další příkazy/data, pokud potřebuje prodloužit čtecí/zápisový cyklus, nastaví na chvilku log. 0. Signál IORDY se používá až u vyšších PIO módů nebo DMA přenosů.

CABLE SELECT – u některých disků umožňuje nastavení režimu jednotky MASTER

(log. 0) / SLAVE (log, 1), jinde je rezervovaný.

/DMAACK – úrovní log. 0 řadič potvrdí přijetí žádosti disku o DMA.

INTRQ – pomocí tohoto signálu informuje disk řadič o různých událostech, např. dokončení příkazu nebo o chybě.

/IOCS16 – tímto signálem disk v log. 0 oznamuje, že provádí 16-bitový přenos.

A0-A2 – slouží k adresaci vnitřních registrů disku, viz. tab. 2.2.3.1.

/PDIAG – tímto signálem disk informuje řadič o dokončení auto testu po zapnutí.

/CS0, /CS1 – slouží spolu s A0-A2 k adresaci vnitřních registrů disku, viz. tab. 2.2.3.1.

/DASP – Drive Active-Slave Present, tímto signálem informuje disk o své aktivitě (v PC bývá

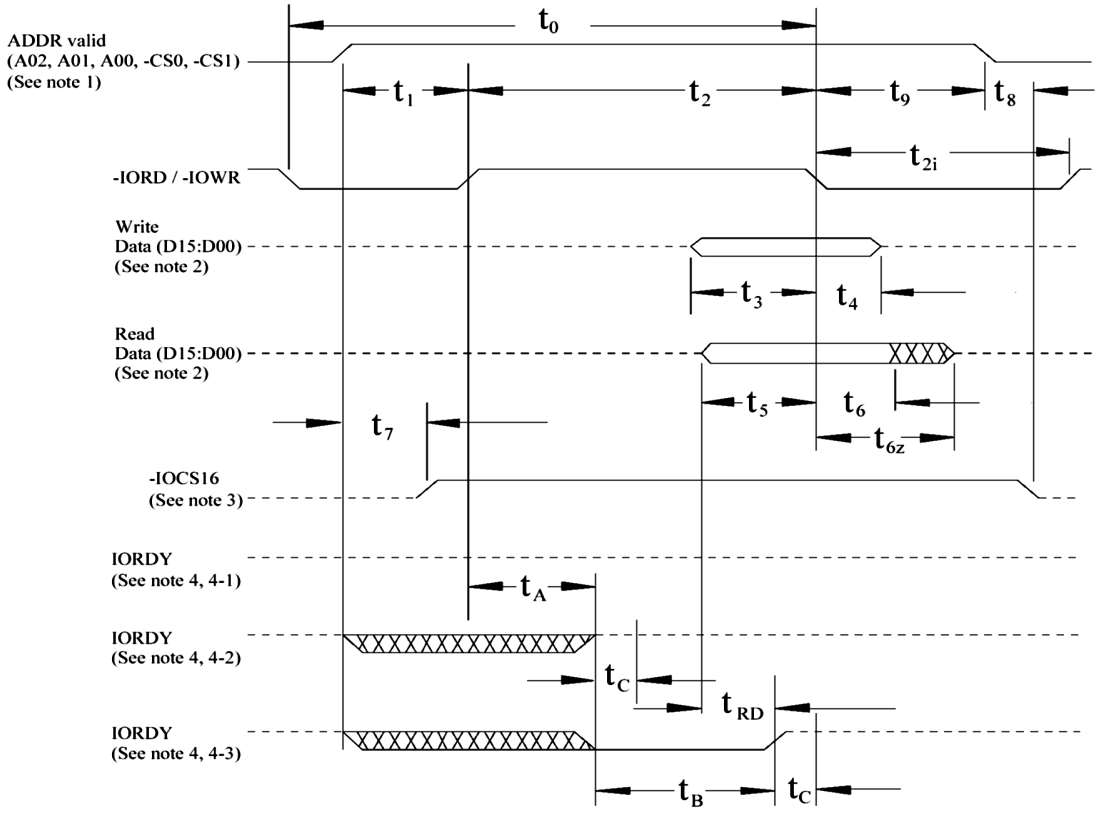
připojen na HDD LED), dále se používá pro detekci druhého disku SLAVE, který po resetu drží tento signál 400ms v log. 0.

Linky používají 5 V logiku, jsou vstupní - I, výstupní - O, 3-stavové - 3S, obousměrné - I/O nebo s otevřeným kolektorem - OC. Elektrické parametry konkrétních linek jsou detailně popsány v [4], kapitola 4. Zařízení se řadiči připojují pomocí 40-žilového plochého kabelu se 2 nebo 3 samořeznými konektory. U zařízení používajících přenos Ultra DMA 66 a vyšší je nutno použít 80-žilový kabel (každý druhý vodič je zem).

2.2.2 Čtecí/zápisový cyklus

Veškeré řízení disku IDE probíhá pomocí několika interních registrů, které budou popsány v další podkapitole. Abychom mohli tyto registry správně číst a zapisovat, je třeba dodržet korektní časování jednotlivých signálů, viz. obr. 2.2.2.1.

Pozor, signály **/IORD**, **/IOWR** a **/IOCS16** jsou nakresleny inverzně!



Notes:
(1) Device address consists of -CS0, -CS1, and A[02::00]
(2) Data consists of D[15::00] (16-bit) or D[07::00] (8 bit)
(3) -IOCS16 is shown for PIO modes 0, 1 and 2. For other modes, this signal is ignored.
(4) The negation of IORDY by the device is used to extend the PIO cycle. The determination of whether the cycle is to be extended is made by the host after tA from the assertion of -IORD or -IOWR. The assertion and negation of IORDY is described in the following three cases:
(4-1) Device never negates IORDY: No wait is generated.
(4-2) Device starts to drive IORDY low before tA, but causes IORDY to be asserted before tA: No wait generated.
(4-3) Device drives IORDY low before tA: wait generated. The cycle completes after IORDY is reasserted. For cycles where a wait is generated and -IORD is asserted, the device shall place read data on D15-D00 for tRD before causing IORDY to be asserted.

Ilustrace 2.2.2.1: Časový diagram čtení/zápisu

Vzhledem k omezeným možnostem běžných jednočipových mikropočítačů se zde nebudu zabývat DMA přenosy, ale pouze PIO přenosy (řízení pomocí I/O portů). Norma ATA definuje režimy PIO 0 až PIO 4, které se liší konkrétními časovými údaji (tab. 2.2.2.1) a tím pádem také maximálními dosažitelnými přenosovými rychlostmi (3,3 – 16,7 MB/s).

Zjednodušeně lze cyklus čtení/zápisu popsat takto:

1) pomocí linek **A0-A2** a **/CS0**, **/CS1** nastavíme adresu požadovaného vnitřního registru

2) při čtení stáhneme linku **/IORD** do log. 0, resp. při zápisu stáhneme linku **/IOWR** do log. 0

3) po určité prodlevě přečteme, resp. zapíšeme data na datové linky **D0-D7** příp. **D0-D15**

(o tom, že máme číst 16-bitově nás informuje aktivní signál **/IOCS16**)

4) vrátíme linku **/IORD**, resp. **/IOWR** do klidového stavu, tj. log. 1

5) po uplynutí další prodlevy můžeme cyklus opakovat.

Signál **IORDY**, kterým zařízení prodlužuje čtecí/zápisový cyklus, je povinný až od režimu PIO 3. Pokud zařízení podporuje režimy PIO 3 a vyšší, mělo by se po resetu nastavit do režimu PIO 0, 1 nebo 2. Vyšší PIO režim pak může být nastaven pomocí ATA příkazu. Další podrobnosti o časování v [4], kapitola 10.

Tabulka 2.2.2.1: Konkrétní časové hodnoty pro dané PIO režimy

	Item	Mode 0 (ns)	Mode 1 (ns)	Mode 2 (ns)	Mode 3 (ns)	Mode 4 (ns)	Note
t0	Cycle time (min)	600	383	240	180	120	1
t1	Address Valid to -IORD/-IOWR setup (min)	70	50	30	30	25	
t2	-IORD/-IOWR (min)	165	125	100	80	70	1
t2	-IORD/-IOWR (min) Register (8 bit)	290	290	290	80	70	1
t2i	-IORD/-IOWR recovery time (min)	-	-	-	70	25	1
t3	-IOWR data setup (min)	60	45	30	30	20	
t4	-IOWR data hold (min)	30	20	15	10	10	
t5	-IORD data setup (min)	50	35	20	20	20	
t6	-IORD data hold (min)	5	5	5	5	5	
t6Z	-IORD data tristate (max)	30	30	30	30	30	2
t7	Address valid to -IOCS16 assertion (max)	90	50	40	n/a	n/a	4
t8	Address valid to -IOCS16 released (max)	60	45	30	n/a	n/a	4
t9	-IORD/-IOWR to address valid hold	20	15	10	10	10	
tRD	Read Data Valid to IORDY active (min), if IORDY initially low after tA	0	0	0	0	0	
tA	IORDY Setup time	35	35	35	35	35	3
tB	IORDY Pulse Width (max)	1250	1250	1250	1250	1250	
tC	IORDY assertion to release (max)	5	5	5	5	5	

2.2.3 Popis vnitřních registrů

Vnitřní registry se dělí do dvou skupin: Control Block Registers (vybírá se přes **/CS1**) a Command Block Registers (vybírá se přes **/CS0**). Není-li aktivní ani **/CS0** ani **/CS1**, jsou

datové linky ve stavu vysoké impedance. Oba dva najednou nesmějí být aktivovány. V tab. 2.2.3.1 jsou uvedeny potřebné adresové kombinace a odpovídající jména registrů. Význam některých registrů je jiný pro čtení a jiný pro zápis. Také pro řadu ATA příkazů se může význam registrů měnit podle potřeby. Ve sloupci adr. jsou I/O adresy v PC, kam IDE řadič mapuje registry primárního kanálu (sekundární kanál je mapován na 0x17x a 0x37x).

Tabulka 2.2.3.1: Vnitřní registry IDE disku

adr.	/CS0	/CS1	A2	A1	A0	čtení (/IORD=0)	zápis (/IOWR=0)
-	0	0	x	x	x	zakázáno	zakázáno
-	1	1	x	x	x	vysoká imped.	nevyužito
0x3Fx	1	0	0	x	x	vysoká imped.	nevyužito
0x3Fx	1	0	1	0	x	vysoká imped.	nevyužito
0x3F6	1	0	1	1	0	ALT_STATUS	DEV_CTRL
0x3F7	1	0	1	1	1	DRV_ADDR	nevyužito
0x1F0	0	1	0	0	0	DATA	DATA
0x1F1	0	1	0	0	1	ERROR	FEATURES
0x1F2	0	1	0	1	0	SECTOR_CNT	SECTOR_CNT
0x1F3	0	1	0	1	1	SECTOR	SECTOR
0x1F4	0	1	1	0	0	CYLINDER_L	CYLINDER_L
0x1F5	0	1	1	0	1	CYLINDER_H	CYLINDER_H
0x1F6	0	1	1	1	0	DRIVE_HEAD	DRIVE_HEAD
0x1F7	0	1	1	1	1	STATUS	COMMAND

DATA – jediný 16-bitový registr sloužící pro přenos dat z/do 512 B interního bufferu. Index bufferu se s každým čtením/zápisem automaticky inkrementuje. Po zadání ATA příkazu, který vyžaduje přenos dat, se index automaticky vynuluje.

ERROR – obsahuje status nebo detaily o případné chybě naposledy provedeného ATA příkazu. Platí pouze je-li nastaven bit ***ERR*** registru **STATUS**, s výjimkou po zapnutí nebo provedení příkazu interní diagnostiky, kdy obsahuje výsledek diagnostiky.

Tabulka 2.2.3.2: Bity registru ERROR

BBK	UNC	-	IDNF	-	ABRT	TK0NF	AMNF

Význam jednotlivých bitů (bit je aktivní, je-li = 1):

BBK – požadovaný sektor je označen jako vadný.

UNC – došlo k neopravitelné chybě.

IDNF – požadovaný sektor nebyl nalezen.

ABRT – provádění příkazu bylo přerušeno kvůli chybě nebo neplatnému ATA příkazu.

TK0NF – během rekalibrace nebyla nalezena nultá stopa.

AMNF – nebyla nalezena adresní značka dat.

FEATURES – sem se zapisuje kód podpříkazu ATA nebo nějaký parametr.

SECTOR_CNT – definuje kolik sektorů se bude během daného příkazu přenášet. Lze zadat hodnotu 0 – 255, kde 0 znamená 256 přenos 256 sektorů. Během přenosu se registr automaticky dekrementuje a udává kolik sektorů ještě zbývá přenést. Po úspěšném dokončení přenosu bude nulový.

SECTOR – sem se zapisuje požadované číslo (počátečního) sektoru dle CHS adresy [1-255], který se má zapsat. V případě LBA adresace se sem ukládá dolních 8 bitů LBA 7:0.

CYLINDER_L, **CYLINDER_H** – slouží pro zápis požadovaného čísla cylindru (LSB a MSB) dle CHS adresy [0-65535], případně 16-ti bitů LBA 23:8.

DRIVE_HEAD – slouží pro zápis požadovaného čísla hlavy a jednotky, viz tab. 2.2.3.3.

Tabulka 2.2.3.3: Bity registru DRIVE_HEAD

1	LBA	SS	MA/SL	H3	H2	H1	H0

LBA – způsob adresace, 0 = CHS, 1 = LBA.

SS – velikost sektoru, 1 = 512 B (jiná hodnota se u IDE HDD nepoužívá).

MA/SL – výběr jednotky na daném kanálu, 0 = MASTER, 1 = SLAVE.

H0-H3 – číslo hlavy [0-15], případně 4 bity LBA 27:24.

STATUS – tento registr slouží ke čtení stavových informací o jednotce, aktualizuje se po každém provedení ATA příkazu. Ostatní bity tohoto registru (a také registrů skupiny Control Block Registers) jsou platné pouze tehdy, je-li bit ***BUSSY*** = 0. Stavové bity viz tab. 2.2.3.4.

Tabulka 2.2.3.4: Bity registru STATUS

BUSY	DRDY	DWF	DSC	DRQ	CORR	INDEX	ERR

Význam jednotlivých bitů (bit je aktivní, je-li = 1):

BUSY – indikuje, že jednotka je zaneprázdněna prováděním příkazu a nelze z ní číst/zapisovat data (žádné jiné registry kromě STATUSu).

DRDY – indikuje, že jednotka je schopna přijímat další příkazy, při chybě zůstává nezměněn, dokud se nepřečte **STATUS** registr, pak se nastaví na 1. Po zapnutí je nulový dokud se neustálí otáčky motoru ploten.

DWF – indikuje chybu zápisu.

DSC – oznamuje, že bylo dokončeno vystavení hlav na danou pozici.

DRQ – oznamuje, že jednotka je připravena přečíst/zapsat slovo z/do portu **DATA**.

CORR – pokud při čtení/zápisu došlo k chybě, která byla ECC mechanismy opravena, je tento bit nastaven, operace čtení/zápisu probíhá dále.

INDEX – při nalezení indexové značky se nastaví jednou za otáčku plotny.

ERR – indikuje, že předchozí příkaz skončil chybou, detaily o chybě jsou v registru **ERROR**.

COMMAND – do tohoto registru se zapíše kód ATA příkazu (viz další podkapitola), který se následně provede, takže je třeba mít předem nastavené všechny ostatní registry.

ALT_STATUS – obsahuje stejné informace jako registr **STATUS**, neovlivňuje přerušení.

DEV_CTRL – umožňuje nastavit generování přerušení s reset jednotky, viz tab. 2.2.3.5.

Tabulka 2.2.3.5: Bity registru DEV_CTRL

-	-	-	-	1	SRST	/DIEN	0

SRST – nastavením bitu do 1 se provede softwarový reset obou jednotek na daném kanálu řadiče.

/DIEN – nastavením bitu do 0 se povolí generování přerušení.

DRV_ADDR – z tohoto portu lze přečíst číslo právě vybrané hlavy a jednotky, viz tab. 2.2.3.6.

Tabulka 2.2.3.6: Bity registru DRV_ADDR

-	/WGT	/H3	/H2	/H1	/H0	/DS1	/DS0

Význam jednotlivých bitů (bit je aktivní, je-li = 0):

/WGT – oznamuje, že jednotka provádí zápis.

/H0-/H3 – negované číslo vybrané hlavy.

/DS1 – indikuje, že je aktivní jednotka 1 (SLAVE).

/DS0 – indikuje, že je aktivní jednotka 0 (MASTER).

Další podrobnosti o registrech v [4], kapitola 7.2.

2.2.4 Popis vybraných ATA příkazů

Standard ATA/ATAPI definuje ve své několikasetstránkové specifikaci [4] (příloha na CD-ROM), kapitola 8 přes 100 různých ATA/ATAPI příkazů, které lze zhruba rozdělit do těchto skupin:

- příkazy pro přenos dat a formátování,
- příkazy pro zjištění informací o jednotce a diagnostiky,
- příkazy pro nastavení různých režimů (PIO, MW-DMA, UDMA,..),
- příkazy pro řízení spotřeby (APM,..),
- příkazy pro on-line monitorování stavu jednotky (S.M.A.R.T.),
- příkazy pro zabezpečení neoprávněného přístupu k jednotce (security),
- specifické příkazy výrobce (např. pro update firmware jednotky).

Specifikace přesně definuje, jaké příkazy jsou pro danou verzi povinné a které volitelné a dále formát a význam předávaných parametrů, datových struktur a návratových hodnot s výjimkou příkazů specifických pro výrobce (ty mají vyhrazený určitý rozsah kódů příkazů). Má-li zařízení splňovat danou verzi specifikace, musí také podporovat všechny povinné příkazy v ní uvedené. Dále stručně uvedu jen příkazy, které jsem v dataloggeru použil s odkazy na konkrétní odstavce v [4].

2.2.4.1 IDENTIFY DEVICE, [4] - 8.12

kód příkazu: **0xEC**

parametry: **DRIVE_HEAD** = číslo jednotky

výstup: **STATUS,** 512 B (256 Word) datová struktura, [4] - 8.12.8

Tento příkaz slouží k identifikaci daného IDE zařízení, z přečtené struktury lze získat spoustu zajímavých informací o parametrech, výrobci, sériové číslo, a pod. Pár příkladů:

Word 1 – celkový počet cylindrů (logická hodnota)

Word 3 – celkový počet hlav (logická hodnota)

Word 6 – celkový počet sektorů na stopu (logická hodnota)

Word 60,61 – celkový počet LBA sektorů

Word 10-19 – sériové číslo (20 ASCII znaků, 1. znak v MSB, 2. znak v LSB)

Word 23-26 – verze firmware (8 ASCII znaků, 1. znak v MSB, 2. znak v LSB)

Word 27-46 – název zařízení (40 ASCII znaků, 1. znak v MSB, 2. znak v LSB)

Word 64 – informace o podpoře vyšších PIO režimů

2.2.4.2 READ SECTOR(S), [4] - 8.27

kód příkazu: **0x20**

parametry: **SECTOR_CNT** = počet čtených sektorů

SECTOR = číslo počátečního sektoru nebo LBA 7:0

CYLINDER_L, CYLINDER_H = číslo cylindru nebo LBA 23:8

DRIVE_HEAD = číslo hlavy nebo LBA 27:24 a jednotky

výstup: **STATUS,** SECTOR_CNT·512 B dat, [4] - 8.27.5

Tento příkaz přečte až 256 sektorů z vybraného disku. Adresu lze zadat v CHS nebo LBA. Po provedení příkazu se z datového bufferu postupně vyčítají sektory od prvního do posledního.

2.2.4.3 READ VERIFY SECTOR(S), [4] - 8.28

kód příkazu: **0x40**

parametry: **SECTOR_CNT** = počet kontrolovaných sektorů

SECTOR = číslo počátečního sektoru nebo LBA 7:0

CYLINDER_L, CYLINDER_H = číslo cylindru nebo LBA 23:8

DRIVE_HEAD = číslo hlavy nebo LBA 27:24 a jednotky

výstup: **STATUS,** [4] - 8.28.5

Tento příkaz provádí v podstatě to samé jako READ SECTORS, jenom nepřenáší žádná data – slouží pouze k ověření čitelnosti zapsaných dat (je rychlejší).

2.2.4.4 WRITE SECTOR(S), [4] - 8.48

kód příkazu: **0x30**

parametry: **SECTOR_CNT** = počet zapisovaných sektorů

SECTOR = číslo počátečního sektoru nebo LBA 7:0

CYLINDER_L, CYLINDER_H = číslo cylindru nebo LBA 23:8

DRIVE_HEAD = číslo hlavy nebo LBA 27:24 a jednotky

DATA = **SECTOR_CNT**·512 B dat

výstup: **STATUS,** [4] - 8.48.5

Tento příkaz zapíše až 256 sektorů na vybraný disk. Adresu lze zadat v CHS nebo LBA. Po provedení příkazu se do datového bufferu postupně zapíšou data všech sektorů od prvního do posledního.

2.2.4.5 EXECUTE DEVICE DIAGNOSTIC, [4] - 8.9

kód příkazu: **0x90**

parametry: **DRIVE_HEAD** = číslo jednotky (diagnostiku však provedou obě dvě)

výstup: **STATUS, ERROR** = výsledný kód diagnostiky, [4] - 8.19.5

SECTOR_CNT, SECTOR, CYLINDER_L, CYLINDER_H, DRIVE_HEAD = signatura, [4] - 9.2

Tento příkaz spustí provedení interní diagnostiky obou jednotek, výsledný kód pak říká která z jednotek má poruchu. Ze signatury lze dále rozlišit, zdali jednotka podporuje paketové příkazy ATAPI.

2.2.4.6 INITIALIZE DEVICE PARAMETERS, [4] - 8.16

kód příkazu: **0x91**

parametry: **SECTOR_CNT** = logický počet sektorů na stopu

DRIVE_HEAD = číslo jednotky a logický počet hlav - 1

výstup: **STATUS** [4], 8.16.5

Některá zařízení vyžadují před vlastním čtením/zápisem provedení tohoto příkazu, který nastaví logickou CHS geometrii pro translaci. V obslužném software tyto hodnoty inicializuji podle hodnot vrácených příkazem **IDENTIFY DEVICE**.

2.2.4.7 SET FEATURES, [4] - 8.37

kód příkazu: **0xEF**

parametry: **FEATURES** = kód podpříkazu, [4] - 8.37.8

SECTOR_CNT, SECTOR, CYLINDER_L, CYLINDER_H, DRIVE_HEAD = závisí na konkrétním podpříkazu, [4] - 8.37.4

výstup: **STATUS** [4], 8.37.5

Tento příkaz slouží k nastavení různých parametrů zařízení, uvedu jen mnou použité podpříkazy:

0x01 – nastav 8-bitový přenosový PIO režim (pouze Compact Flash), [4] - 8.37.9

0x81 – vypni 8-bitový přenosový PIO režim (pouze Compact Flash), [4] - 8.37.9

0x03 – nastav daný přenosový režim (**SECTOR_CNT**=PIO/MW-DMA/UDMA), [4]-8.37.11

Karty Compact Flash umožňují i komunikaci pouze po 8 bitech. Při čtení registru **DATA** se pak používají jen linky D0-D7 a musí se provést dvakrát tolik čtecích/zápisových cyklů, ale zase se ušetří mnohdy nedostatkové porty mikropočítače.

2.3 COMPACT FLASH

Paměťová karta Compact Flash (**CF**) byla původně vymyšlena jako zařízení na ukládání dat pro přenosná elektronická zařízení. První CF kartu založenou na pamětech typu Flash EPROM vyrobila společnost SanDisk Corporation v roce 1994. O rok později vznikla Compact Flash Association (**CFA**), která se stará o standardizaci CF zařízení. CFA definovala standard CF typu I a typu II, které se liší jen v mechanických rozměrech (výška 3,3 mm resp. 5 mm), a později CF+ typu I a II. Standard CF+ se už neomezuje jen na flash karty, ale zahrnuje také zařízení založená na pevném disku (IBM MicroDrive) nebo periferie, např. GPS, WiFi, ethernet, modem, a další karty pro PDA. CF našly široké uplatnění ve spotřební elektronice (digitální fotoaparáty, PDA, přenosné MP3 přehrávače, atd.) ale i v průmyslových a embedded systémech, více v [5]. V současnosti dosahují CF karty kapacity až 12 GB (firma Pretec). Při porovnání flash karty s klasickým pevným diskem bych uvedl

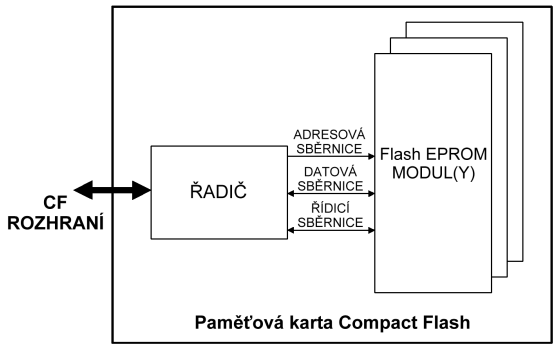
výhody:

- mešní rozměry,
- neobsahuje mechanické pohyblivé části => odolnost proti otřesům,
- nižší spotřeba; 3,3/5 V napájení,
- kompatabilita s PCMCIA a IDE/ATA,

nevýhody:

- menší maximální dostupná kapacita,
- omezený počet zápisových cyklů,
- vyšší cena za MB.

Další popis se bude týkat CF karet založených na pamětech Flash EPROM, blokové schéma karty viz obr. 2.3.1.



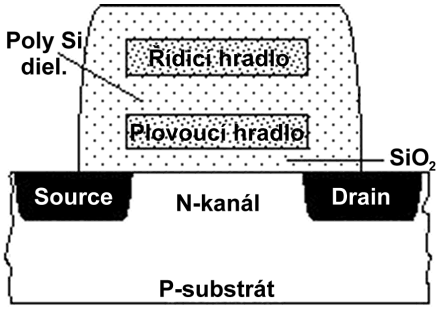
Ilustrace 2.3.1: Blokové schéma CF karty

Srdcem karty je řadič – zákaznický obvod typu ASIC, který se stará o komunikační protokoly, časování, detekci a opravu chyb (ECC), diagnostiku a řízení spotřeby. Výrobci CF karet obvykle vyrábí vlastní řadiče a paměti kupují od velkých výrobců jako Intel, Samsung, Toshiba, Hitachi a další.

2.3.1 Paměť Flash EPROM

Flash EPROM patří mezi tzv. **nevolatilní** paměti, což znamená, že si svůj obsah udrží i po vypnutí napájecího napětí. Historicky prvním druhem tohoto typu pamětí byly **ROM** (Read -Only Memory). U nich byl obsah pevně dán už od výroby a nebyla žádná možnost jej změnit. Tento druh pamětí se stále využívá zejména pro masovou produkci, protože je technologicky nejjednodušší a tudíž i nejlevnější. Ovšem pro malé série by příprava výroby ROM vyšla dost draho a proto byla snaha vyvinout takovou paměť, kterou by si mohl uživatel sám naprogramovat. Paměť **PROM** (Programmable ROM) využívala matici tenkých kovových propojek, které se elektrickým impulsem přetavily a tím se změnil logický stav buňky. Takto naprogramovaná buňka už ale nešla obnovit do původního stavu.

Dalším vývojovým krokem byly paměti **EPROM** (Erasable PROM) založené na tranzistorech MOS s plovoucím hradlem, viz obr. 2.3.1.1.

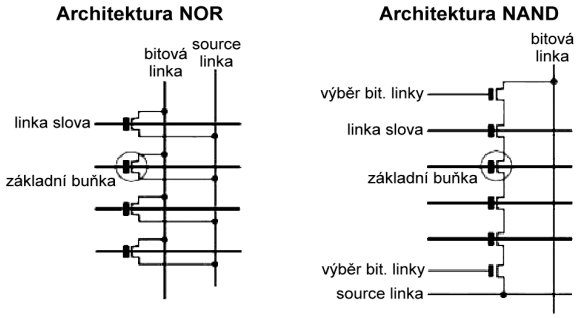


Ilustrace 2.3.1.1: Tranzistor MOSFET s plovoucím hradlem

Stav buňky je dán přítomností/nepřítomností náboje na plovoucím hradle. To je obklopeno izolantem, který zabrání úniku náboje pryč (výrobci uvádějí běžně dobu uchování dat 20-100 let). Při programování se na řídicí hradlo přiloží zvýšené kladné napětí (okolo 13 V) tak, aby došlo k tunelování nosičů náboje skrz tenkou oxidovou vrstvu (desítky nm), přičemž část z nich se zachytí na plovoucím hradle a zůstane zde i po odpojení programovacího napětí. Při čtení se na řídicí hradlo a drain připojí menší napětí (source je na nulovém potenciálu), které nemůže způsobit tunelování. Je-li na plovoucím hradle nahromaděn záporný náboj, tranzistor se neotevře. Pokud je plovoucí hradlo bez náboje, tranzistor se otevře jako normální MOSFET (naindukuje se N-kanál) a začne procházet proud I_{ds}. K vymazání obsahu paměti potřebujeme odstranit náboj z hradel. To lze provést vystavením čipu ionizujícímu (UV či RTG) záření, proto má paměť EPROM skleněné okénko.

Mazání EPROM trvalo řádově minuty a navíc byl třeba zdroj záření, což vylučovalo opakované mazání a programování přímo v zařízení. Proto byly vynalezeny paměti typu **EEPROM** (Electrically Erasable PROM) a Flash, tedy paměti elektricky programovatelné i mazatelné. Hlavní rozdíl mezi EEPROM a Flash je ten, že EEPROM lze mazat a programovat po bitech, zatímco Flash pouze po celých blocích (typ. několik kB). K výmazu se opět využívá tunelový jev, tentokrát se záporným napětím na řídicím hradle. Výmaz bloku trvá asi 1-10ms. Opakovaným programováním a mazáním dochází k degradaci izolačního oxidu mezi plovoucím hradlem a substrátem až do stavu, kdy je buňka dále nepoužitelná. Výrobci dnešních pamětí udávají životnost podle typu 10000 – 1000000 cyklů. Podrobný popis funkce paměťové buňky viz [6].

Podle logického zapojení buňky se Flash dělí na typ NOR a NAND, viz obr. 2.3.1.2. Paměť typu NOR představila poprvé firma Intel v roce 1988. NOR se svou organizací podobá pamětem RAM, umožňuje rychlý přístup k náhodně zvolené buňce. Používá se hlavně k uložení programového kódu bez nutnosti předem načítat kód do RAM. O rok později uvedla Toshiba paměť typu NAND. Ta se zase chová spíše jako pevný disk, pracuje s bloky a je vhodnější pro uložení velkých dat. Paměťové buňky NAND jsou menší, takže lze dosáhnout větší hustoty integrace, navíc mají menší spotřebu a jsou rychlejší při zápisu.



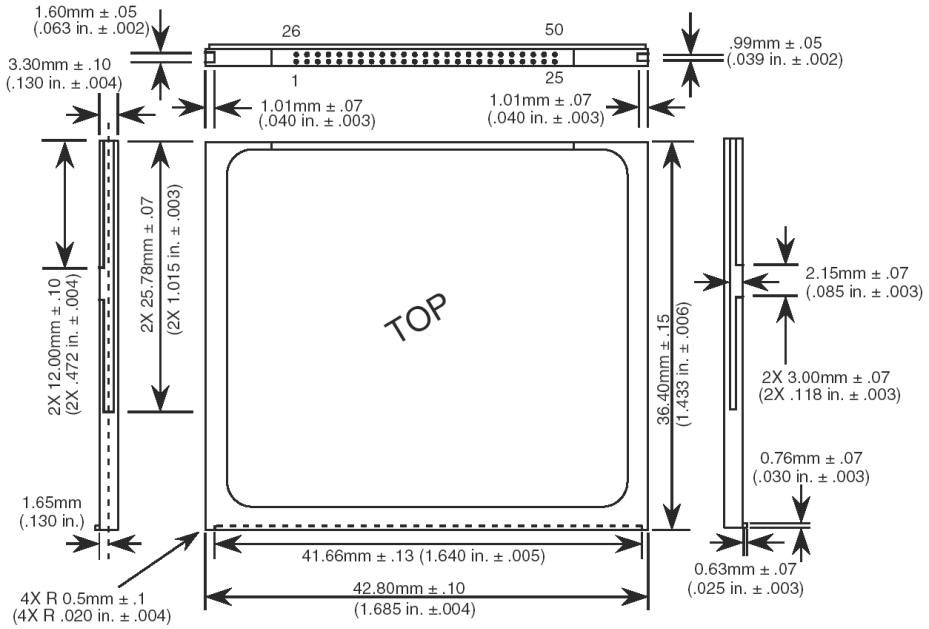
Ilustrace 2.3.1.2: Buňka NOR vs. NAND

V kartách Compact Flash se dnes výhradně používá varianta NAND. Další porovnání NOR vs. NAND viz [7]. Zajímavou možnost, jak zvýšit kapacitu pamětí při stejné ploše čipu přinesla technologie **MLC** (Multi-Level Cell), která do jedné buňky ukládá vícebitovou informaci. Náboj na hradle může mít několik úrovní a při čtení se vyhodnocuje rozdílný proud I_{ds}, který se prahuje na daný počet diskrétních úrovní a dekóduje do více bitů. Současná největší MLC paměť od Samsungu vyráběná 50nm technologií má kapacitu 16Gb na čipu.

2.3.2 Popis fyzického a elektrického rozhraní CF I/II

Rozhraní vychází ze standardu PCMCIA (podporuje I/O i paměťový režim), avšak CF kartu lze přepnout do režimu **TrueIDE**, kdy se chová jako standardní pevný disk IDE (většina CF karet vyhovuje standardu ATA-4 a vyšší). CF kartu lze připojit pomocí jednoduché redukce

sestávající z odpovídajících konektorů a vodivého propojení ke standardnímu IDE řadiči nebo PCMCIA slotu. To umožňuje snadné připojení CF karty do stolního nebo přenosného počítače, kde případně může nahradit klasický pevný disk. Nákres pouzdra CF karty typu I s číslováním konektoru viz obr. 2.3.2.1. Typ II se liší hlavně výškou pouzdra 5 mm, nákres viz [8], figure 4.



Ilustrace 2.3.2.1: Pouzdro a konektor CF typu I

CF konektor už v sobě zahrnuje i napájení, význam dutinek pro režim TrueIDE popisuje tab. 2.3.2.1.

Tabulka 2.3.2.1: Zapojení konektoru Compact Flash

dutinka	popis	směr
1	GND	-
2	D3	I/O
3	D4	I/O
4	D5	I/O
5	D6	I/O
6	D7	I/O
7	/CS0	I
8	A10 (spojit s GND)	I
9	/ATA SELECT (spojit s GND)	I
10	A9 (spojit s GND)	I

dutinka	popis	směr
11	A8 (spojit s GND)	I
12	A7 (spojit s GND)	I
13	VCC	-
14	A6 (spojit s GND)	I
15	A5 (spojit s GND)	I
16	A4 (spojit s GND)	I
17	A3 (spojit s GND)	I
18	A2	I
19	A1	I
20	A0	I
21	D0	I/O
22	D1	I/O
23	D2	I/O
24	/IOCS16	O
25	/CD2 (uvnitř CF spojené s GND)	O
26	/CD1 (uvnitř CF spojené s GND)	O
27	D11 (nutné pouze pro 16-bitový přenos)	I/O
28	D12 (nutné pouze pro 16-bitový přenos)	I/O
29	D13 (nutné pouze pro 16-bitový přenos)	I/O
30	D14 (nutné pouze pro 16-bitový přenos)	I/O
31	D15 (nutné pouze pro 16-bitový přenos)	I/O
32	/CS1	I
33	/VS1 (nezapojovat)	O
34	/IORD	I
35	/IOWR	I
36	/WE (spojit s VCC)	I
37	INTRQ	O
38	VCC	-
39	CABLE SELECT	I
40	/VS2 (nezapojovat)	O
41	/RESET	I
42	IORDY	O
43	RFU (nezapojovat)	O
44	RFU (spojit s VCC)	I

dutinka	popis	směr
45	/DASP	I/O
46	/PDIAG	I/O
47	D8 (nutné pouze pro 16-bitový přenos)	I/O
48	D9 (nutné pouze pro 16-bitový přenos)	I/O
49	D10 (nutné pouze pro 16-bitový přenos)	I/O
50	GND	-

Pozn.: signály s „/“ na začátku jsou aktivní v log. 0

GND, VCC – zem a napájení 3,3 V ±5 % nebo 5 V ±10 %. Každé CF(+) zařízení by mělo umět pracovat při obou napájecích napětích. Maximální efektivní proudový odběr je 75, resp. 100 mA v napájecím režimu 0 a 500 mA v napájecím režimu 1. Režim 1 používají např. CF karty založené na pevném disku pro příkazy vyžadující roztočení ploten, ostatní příkazy mohou být provedeny v režimu 0.

D0-D15 – datové linky po kterých se přenáší příkazy a data. Data se mohou přenášet buď 16-bitově nebo 8-bitově, to lze nastavit ATA příkazem SET FEATURES, viz podkapitola 2.2.4.7.

A0-A2, /CS0, /CS1 – slouží k adresaci vnitřních registrů, viz. tab. 2.2.3.1.

A3-A10 – v režimu TrueIDE se nevyužívá, spojit se zemí.

/ATA SELECT – spojením se zemí se CF přepne do TrueIDE režimu.

/IOCS16 – tímto signálem CF v log. 0 oznamuje, že provádí 16-bitový přenos.

/CD1, /CD2 – tyto dutinky jsou uvnitř CF propojeny se zemí a slouží pro detekci správného zasunutí karty (na protikusu jsou piny o něco delší).

/VS1, /VS2 – rezervováno pro PCMCIA režim.

/IOWR – při zápisovém cyklu generuje řadič pro CF negativní impuls minimální délky.

/IORD – při čtecím cyklu generuje řadič pro CF negativní impuls minimální délky.

/WE – v režimu TrueIDE se nevyužívá, spojit se zemí.

INTRQ – pomocí tohoto signálu informuje CF řadič o různých událostech, např. dokončení příkazu nebo o chybě.

CABLE SELECT – v TrueIDE režimu slouží pro nastavení CF do režimu MASTER (log. 0) / SLAVE (log, 1).

/RESET – tento signál inicializuje CF do výchozího stavu.

IORDY – tímto signálem CF v log. 1 oznamuje, že je schopna přijímat další příkazy/data, pokud potřebuje prodloužit čtecí/zápisový cyklus, nastaví na chvilku log. 0. Signál

IORDY se používá až u vyšších PIO módů.

RFU – rezervováno pro PCMCIA režim.

/DASP – Drive Active-Slave Present, tímto signálem informuje CF o své aktivitě, dále se používá pro detekci druhého disku SLAVE, který po resetu drží tento signál 400ms v log. 0.

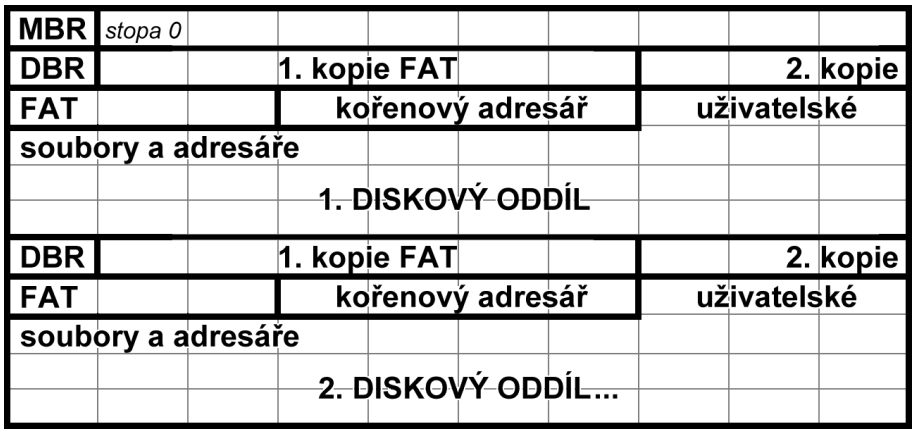
/PDIAG – tímto signálem disk informuje řadič o dokončení auto testu po zapnutí.

2.3.3 Řízení Compact Flash

V TrueIDE režimu CF emuluje vnitřní registry pevného disku IDE tak, jak byly popsány v podkapitole 2.2.3. Přístup k vnitřním registrům se provádí zápisovým/čtecím cyklem pro daný PIO režim, tak jak byl popsán v 2.2.2. Pro řízení se používají ATA příkazy popsané v 2.2.4 a v [8], kapitola 6.2. Ve specifikaci Compact Flash 3.0 přibyly režimy Ultra DMA pro rychlejší přenos dat.

2.4 DŮLEŽITÉ DATOVÉ STRUKTURY A SOUBOROVÝ SYSTÉM

Pro pochopení organizace dat na disku v této kapitole vysvětlím význam jednotlivých datových struktur, jak je používají operační systémy na PC kompatabilní s DOS/Windows. Na obr. 2.4.1 je typické rozložení struktur pro pevné disky. Některá média, např. diskety, však vůbec nepoužívají MBR a obsahují jen jeden oddíl. U CF karet lze použít oba způsoby, já jsem se však z důvodu kompatability přiklonil k rozložení jako u klasického pevného disku.



Ilustrace 2.4.1: Rozmístění důležitých datových struktur na disku

2.4.1 MBR a tabulka rozdělení disku

Nultý sektor záznamového média je vyhrazen pro tzv. Master Boot Record. Ten obsahuje tabulku rozdělení disku (PArtition Table) a spustitelný kód zavaděče, blíže viz tab. 2.4.1.1. Na počítačích typu PC při startu BIOS zavede MBR na adresu 0x7C00, zkontroluje signaturu spustitelného kódu (slovo 0xAA55) a předá mu řízení. Kód zavaděče pak určí aktivní oddíl (případně zobrazí volbu oddílů pro uživatele) z nějž načte počáteční sektor, zkontroluje jeho signaturu a předá mu řízení. Bližší informace např. v [9]. V případě zpracování MBR mikropočítačem s jinou instrukční sadou než x86, nemá pro nás kód zavaděče žádný význam.

Tabulka 2.4.1.1: Struktura MBR

offset	velikost [B]	popis
0x000	446	spustitelný kód zavaděče
0x1BE	16	1. položka tabulky rozdělení disku (PAT)
0x1CE	16	2. položka tabulky rozdělení disku (PAT)
0x1DE	16	3. položka tabulky rozdělení disku (PAT)
0x1EE	16	4. položka tabulky rozdělení disku (PAT)
0x1FE	2	signatura 0xAA55

Důležitá je však tabulka rozdělení disku. Na PC je k dispozici řada nástrojů pro její snadnou editaci, např. program FDISK, který je standardní součástí DOSu, Windows 9x, Linuxu a dalších. MBR obsahuje 4 položky PAT (detail položky viz tab. 2.4.1.2), které mohou definovat až 4 existující primární oddíly. Nevyužité položky jsou vynulované. DOS vidí pouze aktivní primární oddíl (přiřadí mu písmeno C:) a ostatní jsou pro něj skryté. Windows 9x a vyšší už umožňují s dalšími primárními oddíly normálně pracovat. Pro definici více logických oddílů se obvykle v PAT v MBR vytvoří záznam jednoho primárního a jednoho rozšířeného oddílu. Rozšířený oddíl obsahuje na začátku podobnou strukturu jako MBR (obvykle už zde není kód zavaděče) opět se čtyřmi položkami PAT. Ty definují 1 další logický oddíl a 1 další rozšířený oddíl a takto se řetězí dál a dál. Logickým oddílům pak DOS přiřazuje písmena D:, E:, F:, atd.

První byte položky PAT značí, je-li oddíl aktivní (hodnota 0x80) nebo neaktivní (hodnota 0). Dále následuje CHS adresa počátku oddílu. Ta je zde jen z důvodu zpětné kompatability, dnes se už dávno používá LBA. Aby se ušetřilo místo, byla tehdy hodnota cylindru [0-1023] a sektoru [1-63] zapakovaná do jednoho 16-bitového slova, viz tab. 2.4.1.3 tak, aby byla hodnota připravená pro diskové služby BIOSu INT 0x13 - čtení/zápis sektoru pro nahrání do registru CX, detaily viz [10], kapitola funkce ROM-BIOSu. Toto šetření místem pak mimo jiné způsobilo problémy při adresaci disků větších kapacit. První oddíl obvykle začíná na prvním sektoru první stopy. Zbylé místo nulté stopy mohou využívat bootmanažery, různé ovladače disku - rozšíření INT 0x13 nebo také bootviry.

Tabulka 2.4.1.2: Struktura položky PAT

offset	velikost [B]	popis
0x000	**1**	indikátor aktivního oddílu [0x00/0x80]
0x001	**1**	začátek oddílu – hlava [0-255]
0x002	**2**	začátek oddílu – cylindr a sektor 10:6
0x004	**1**	typ souborového systému na oddílu
0x005	**1**	konec oddílu – hlava [0-255]
0x006	**2**	konec oddílu – cylindr a sektor 10:6
0x008	**4**	začátek oddílu – LBA
0x00C	**4**	počet LBA sektorů oddílu

Tabulka 2.4.1.3: Položka cylindr-sektor

C7	C6	C5	C4	C3	C2	C1	C0	C9	C8	S5	S4	S3	S2	S1	S0

Další byte identifikuje typ souborového systému na daném oddílu. V tab. 2.4.1.4 jsou uvedeny čísla několika nejznámějších systémů, kompletní seznam viz např. v [11]. Dále se budu zabývat jen systémem FAT16, který byl použit v implementaci.

Tabulka 2.4.1.4: Položka typ souborového systému

hodnota	typ souborového systému
0x01	FAT12 – primární oddíl (MS-DOS 2.0+), do 15 MB
0x04	FAT16 – primární oddíl (MS-DOS 3.0+), do 32 MB
0x05	rozšířený oddíl (MS-DOS 3.3+), do 2 GB
0x06	FAT16 – primární oddíl (MS-DOS 3.0+), do 2 GB
0x07	NTFS – primární oddíl (OS/2, Windows NT), do 256 TB
0x0B	FAT32 – primární oddíl (MS Windows 95 SR2+), do 2 TB
0x0C	rozšířený oddíl (MS Windows 95 SR2+), do 2 TB
0x0E	VFAT16 – primární oddíl (MS Windows 95+), do 2 GB
0x0F	rozšířený oddíl (MS Windows 95+), do 2 GB

Pak následuje CHS adresa konce oddílu. Nakonec je zde uložena i 32-bitová LBA začátku oddílu (pozor, LBA sektory se číslují od 0) a počet sektorů v oddílu. Z důvodu kompatability musí být LBA a CHS hodnoty synchronizovány. Protože začátek i konec oddílu se v CHS adresování obvykle zarovnává na celé cylindry, nemůže LBA nabývat úplně libovolných hodnot.

2.4.2 DOS Boot Record

Dále uvažujeme, že na oddílu definovaném v PAT byl vytvořen souborový systém typu FAT. Potom se první sektor oddílu nazývá DOS Boot Record, protože jako první začal systém FAT používat operační systém MS-DOS. Při bootování je DBR načten zavaděčem MBR opět na adresu 0x7C00 (zavaděč MBR se musí předtím překopírovat do jiné oblasti) a po kontrole signatury 0xAA55 je mu předáno řízení. Jeho úkolem je najít a zavést systémové soubory operačního systému.

Na začátku DBR (viz. tab. 2.4.2.1) je instrukce skoku na spustitelný kód umístěný ke konci. Dále následuje 8-bajtový ASCII řetězec identifikující obvykle operační systém, kterým byl souborový systém vytvořen. Např. MS-DOS a Windows NT/2000 používají řetězec „MSDOS5.0“ a Windows 95 „MSWIN4.0“ či „MSWIN4.1“ pro verzi OSR2.

Pak jsou zde dvě struktury základního (tab. 2.4.2.2) a rozšířeného (tab. 2.4.2.3) bloku parametrů BIOSu obsahující důležité informace o souborovém systému. Jen dodám, že u systému FAT32 byl BPB rozšířen z 25 B na 53 B na úkor velikosti zavaděče, takže

kompatabilita DBR není úplná. Bližší informace o DBR systému Windows 95 OSR2 v [12] a Windows 98 SE v [13].

Tabulka 2.4.2.1: Struktura DBR FAT16

offset	velikost [B]	popis
0x000	3	instrukce JMP na začátek kódu zavaděče
0x003	8	OEM ID – ASCII řetězec id. systému
0x00B	25	BPB – blok parametrů BIOSu
0x024	26	EBPB – rozšířený blok parametrů BIOSu
0x03E	448	spustitelný kód zavaděče DOSu
0x1FE	2	signatura 0xAA55

Tabulka 2.4.2.2: Struktura BPB FAT16

offset	velikost [B]	popis
0x000	2	velikost sektoru v bytech, u FAT vždy 512 B
0x002	1	počet sektorů na cluster [1-128]
0x003	2	počet rezervovaných sektorů (typ. 1 - DBR)
0x005	1	počet kopií FAT (typ. 2, někdy 1)
0x006	2	počet položek kořenového adresáře
0x008	2	počet sektorů oddílu (pokud=0, platí 32b h.)
0x00A	1	deskriptor média (0xF8 pro HDD)
0x00B	2	velikost FAT v sektorech
0x00D	2	počet sektorů na stopu (dáno CHS)
0x00F	2	počet hlav (dáno CHS)
0x011	4	počet skrytých sektorů před DBR
0x015	4	počet sektorů oddílu (platí,když 16b hod.=0)

První položka BPB udává velikost sektoru. U pevných disků IDE a CF karet je velikost sektoru vždy 512 B (hlavně z důvodu kompatability). Avšak některé SCSI disky nebo CD-ROM jednotky fungující jako bloková zařízení mohou používat jinou velikost, např. 2048 B. Protože počet sektorů u velkokapacitních médií je příliš veliký na efektivní správu, sdružuje souborový systém FAT sektory do tzv. **clusterů.** Cluster může obsahovat 2^n sektorů, teoreticky až 256. Avšak implementace FAT v MS-DOS dovoluje maximálně 64 sektorů na cluster, u Windows NT max. 128 sektorů na cluster. Jelikož FAT16 používá pro adresaci clusterů

16-ti bitové položky, tak z toho také vyplývá maximální velikost diskového oddílu 2 GB, resp. 4 GB. Jelikož FAT adresuje pouze celé clustery, je třeba si uvědomit, že i jednobajtový soubor si alokuje celý cluster a zbytek zůstane nevyužit (tzv. slack, wasted cluster space). Tento problém částečně řeší třeba souborový systém FAT32, který umožňuje adresovat více drobnějších clusterů, ale příliš malá velikost clusterů se zas negativně podepisuje na rychlosti přístupu k datům. Proto je lepší data ukládat do menšího počtu větších souborů.

Počet rezervovaných sektorů udává kolik sektorů včetně DBR je před první tabulkou FAT. Ta obvykle následuje hned za DBR.

DOS a Windows z důvodu bezpečnosti uložených dat standardně používají dvě kopie FAT. V případě poškození jedné FAT lze použít druhou záložní, (nejen) pro kontrolu a opravu FAT slouží např. nástroje Scandisk či Norton Disk Doctor. Při zápisu je potřeba obě FAT synchronizovat. Pro úsporu místa na nízkokapacitních médiích je možno použít jen jednu FAT, takový disk ale nepřečtou starší verze MS-DOS (6.22 a nižší).

Další slovo udává počet položek kořenového adresáře, viz podkapitola 2.4.4. Na offsetu 0x008 je 16-bitová hodnota udávající celkový počet sektorů oddílu. Tuto položku používaly starší verze DOSu s omezením kapacity oddílu na 32MB. U novějších verzí se začala používat 32-bitová hodnota umístěná na offsetu 0x015, ale jen u oddílů přesahujících 32MB. Pokud je 16-bitová hodnota nulová, znamená to, že platí 32-bitová hodnota.

Deskriptor média určuje zda se jedná o pevný disk (hodnota 0xF8) či disketu, podrobný seznam hodnot viz např.[14]. Velikost FAT udává, kolik zabírá jedna kopie FAT sektorů. Závisí na počtu položek FAT, tedy počtu clusterů a určuje se při formátování oddílu. Maximální velikost může být 128 kB. Počet sektorů na stopu a počet hlav odpovídá CHS geometrii daného disku. Počet skrytých sektorů udává kolik sektorů od začátku disku předchází začátku daného oddílu.

Rozšířený blok parametrů BIOSu (viz tab. 2.4.2.3) obsahuje další informativní položky, které nejsou pro implementaci na mikropočítači nezbytné. Číslo disku se používá při volání služeb BIOSu INT 0x13, kde disketové jednotky jsou číslovány od 0 a pevné disky od 0x80.

Tabulka 2.4.2.3: Struktura EBPB FAT16

offset	velikost [B]	popis
0x000	1	číslo disku podle BIOSu (HD0 = 0x80)
0x001	1	rezervováno (0)
0x002	1	signatura 0x29
0x003	4	sériové číslo oddílu
0x007	11	jmenovka oddílu - ASCII řetězec
0x012	8	jméno souborového systému - ASCII řetěz.

Sériové číslo je zapsáno při formátování oddílu a slouží zejména pro identifikaci výměnných médií, rovněž tak jmenovka oddílu, která obsahuje až 11 znaků dlouhý název zarovnaný mezerami. Jak uvidíme dále v podkapitole 2.4.4, jmenovka oddílu může být obsažena i v položce kořenového adresáře a ne vždy je zaručena její synchronizace s položkou v DBR. Řetězec se jménem souborového systému („FAT16“ pro systém FAT16) má také pouze informativní charakter, rozhodující je typ souborového systému uvedený v položce PAT, viz tab. 2.4.1.4.

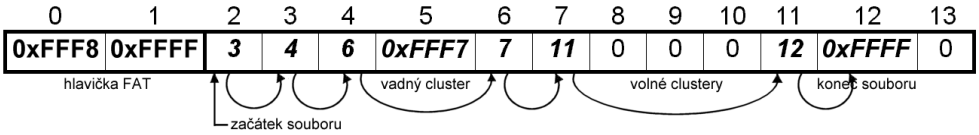
2.4.3 FAT - Alokační tabulka souborů

FAT je struktura typu spojový seznam sloužící k uchovávání informací o přiřazených datových clusterech k souborům a adresářům. Více podrobností v [14] a [15]. V tab. 2.4.3.1 je udaná velikost adresy clusteru a jejich maximální počet, který lze v daných typech FAT alokovat. U FAT32 se mohou limity lišit podle dané implementace v operačním systému, detaily v [16].

Tabulka 2.4.3.1: Typy FAT

typ	vel. adr. clusteru	počet clusterů	max. velikost FAT
FAT12	12 bitů	≤4082	8 kB
FAT16	16 bitů	≤65524	128 kB
FAT32	32 bitů	$<2^{28}$	1 GB

FAT16 můžeme chápat jako lineární pole 16-ti bitových slov, které reprezentují adresy následujících clusterů souboru nebo adresáře, případně vyhrazené hodnoty podle tab. 2.4.3.2.



Ilustrace 2.4.3.1: Ukázka struktury FAT16

Tabulka 2.4.3.2: Možné hodnoty položek FAT16

hodnota	popis
`0x0000`	volný cluster
`0x0001 - 0x0002`	zakázaná hodnota
`0x0003 - 0xFFF6`	adresa dalšího clusteru
`0xFFF7`	vadný cluster
`0xFFF8 - 0xFFFF`	EOF – konec souboru (řetězce clusterů)

Na obr. 2.4.3.1 je příklad FAT16 s jedním souborem. Položky číslo 0 a 1 mají speciální význam a nepoužívají se k adresaci clusterů. Hodnota položky 0 je 0xFFMM, kde MM je hodnota deskriptoru média uvedená v DBR. Položka 1 má obvykle hodnotu 0xFFFF, některé systémy však mohou využívat MSb jako flag označující korektní ukončení práce se souborovým systémem. Pokud má bit hodnotu 0, značí to nekorektní ukončení práce a při dalším spuštění systém pozná, že má provést kontrolu souborového systému (např. Windows automaticky spustí program Scandisk).

Od položky č. 2 už lze zapisovat adresy clusterů přiřazené uživatelským souborům. První adresa clusteru je uložená v adresáři, jak uvidíme v další podkapitole. Zde položka č. 2 obsahuje adresu 3, tedy soubor pokračuje na clusteru 3. V položce č. 3 je adresa 4, ve 4. položce adresa 6 až k hodnotě 0xFFFF, která značí konec souboru. Tato sekvence adres se nazývá **řetězec clusterů** (cluster chain). Další ukázka struktury FAT zobrazená na PC programem Norton Disk Editor je v tab. 2.4.3.3. Tučně je zvýrazněn řetězec clusterů souboru OPTIMIZE.EXE, který začíná na clusteru 79 a končí na clusteru 91. Clustery zde mají velikost 32 kB, z toho plyne alokovaná velikost 393216 B. Skutečná velikost souboru je pouze 369680 B, takže 23536 B (6%) zůstává nevyužito v posledním clusteru. Jak už bylo řečeno, lze toto plýtvání omezit volbou menších clusterů (pokud to dovoluje velikost oddílu) na úkor výkonu.

Tabulka 2.4.3.3: Ukázka struktury FAT16 v programu Norton Disk Editor

Disk Editor

Object Edit Link View Info Tools Help

Sector 1

		<EOF>	<EOF>	<EOF>	6	<EOF>	8
41	<EOF>	<EOF>	<EOF>	<EOF>	<EOF>	20	16
<EOF>	19	<EOF>	18	21	22	23	24
25	26	27	28	29	30	31	32
33	34	15	36	<EOF>	38	65490	40
<EOF>	42	43	8239	8241	<EOF>	48	<EOF>
49	47	51	<EOF>	27682	54	<EOF>	<EOF>
<EOF>	<EOF>	<EOF>	60	1009	<EOF>	65497	<EOF>
65483	66	<EOF>	68	69	<EOF>	71	72
<EOF>	74	75	<EOF>	77	78	<EOF>	**80**
81	**82**	**83**	**84**	**85**	**86**	**87**	**88**
89	**90**	**<EOF>**	92	93	94	95	96
97	98	99	<EOF>	101	<EOF>	103	104
<EOF>	106	<EOF>	108	109	<EOF>	111	<EOF>
<EOF>	<EOF>	<EOF>	<EOF>	<EOF>	118	119	120
121	122	123	124	125	126	127	128
129	130	131	132	133	134	135	136
137	138	139	140	141	142	143	144
145	146	147	148	149	150	151	152
153	154	155	156	157	158	159	160

FAT (1st Copy) Sector 1

C:\QEMM\OPTIMIZE.EXE Cluster 79, hex 4F

Nikde se mi nepodařilo dohledat, proč zrovna konec souboru může nabývat osmi různých hodnot. I když MS-DOS a Windows používají standardně hodnotu 0xFFFF, je lepší na to nespoléhat a kontrolovat celý rozsah hodnot. Hodnotou 0xFFF7 se označuje vadný cluster, který může vzniknout fyzickým poškozením povrchu média (i Flash EPROM paměti mají omezený počet zápisových cyklů) a nelze z něj dále číst/zapisovat (i při chybě v jediném bitu se musí označit celý cluster). Ke kontrole povrchu média je na PC opět k dispozici řada

programů jako Scandisk nebo Norton Disk Doctor.

Jak je vidět z obr. 2.4.3.1, nemusí clustery souboru nebo adresáře vůbec následovat po sobě. K tomu obvykle dojde , když se na disk ukládá postupně za sebou více souborů a pak se nějaký soubor ze začátku disku vymaže (adresy clusterů se vynulují). Při ukládání dalšího souboru obvykle systém najde ve FAT první volný cluster na který začne soubor zapisovat. Pokud je nově zapisovaný soubor větší než mezera vzniklá vymazáním předešlého souboru, musí systém najít další volnou oblast, která už nebude fyzicky navazovat na první oblast. Soubor tak může být uložen v mnoha nesouvislých blocích clusterů, tzv. **fragmentech** po celém povrchu disku. Tento jev se nazývá **fragmentace** souborového systému. Z hlediska logického přístupu k souboru se nejedná o žádný problém, protože adresy clusterů mohou ukazovat kamkoliv po datové oblasti disku. Avšak z hlediska fyzického přístupu způsobuje fragmentace jednoznačně zpomalení přístupu k datům. Jednak je složitější procházení FAT (v případě, že ji nemůžeme mít celou načtenou v RAM) a hlavně u pevných disků trvá řádově déle přesun hlav z jednoho kraje disku na druhý oproti pouhému přesunu na následující stopu. Naštěstí u pamětí Flash EPROM je doba přístupu na všechny adresy stejná. Na odstranění fragmentace, tzv. **defragmentaci** slouží na PC programy jako Defrag nebo Norton Speed Disk.

2.4.4 Kořenový adresář (root directory)

Kořenový adresář je oblast disku konstantní velikosti následující za FAT, která sloužící k ukládání názvů souborů/adresářů, adresy prvního clusteru ve FAT a dalších důležitých atributů. Informace se ukládají do 32 B velkých adresářových položek s následující strukturou:

Tabulka 2.4.4.1: Struktura adresářové položky

offset	velikost [B]	popis
0x00	8	jméno (ASCII, zarovnané mezerami)
0x08	3	přípona (ASCII, zarovnaná mezerami)
0x0B	1	atribut (bitové flagy)
0x0C	1	rezervováno pro Windows NT (0)
0x0D	1	čas vytvoření [0-199] *10ms
0x0E	2	čas vytvoření (pakovaný)
0x10	2	datum vytvoření (pakované)
0x12	2	datum posledního přístupu
0x14	2	rezervováno pro FAT32 (vyšší slovo adr. c.)
0x16	2	čas posledního zápisu (pakovaný)
0x18	2	datum posledního zápisu (pakované)
0x1A	2	adresa počátečního clusteru ve FAT
0x1C	4	velikost souboru v bytech

Jméno o max. 8-mi znacích a přípona max. o 3 znacích je systémem zobrazované obvykle odděleně pomocí tečky, která však není v samotném názvu uložena, z toho plyne název konvence „8.3“. Nevyužité znaky jsou vyplněny mezerami (ASCII kód 0x20). Pro název uživatelských souborů a adresářů nelze použít libovolné ASCII znaky, zejména bychom se měli vyhnout: všem znakům s ASCII kódem ≤0x20 a dále 0x22 ("), 0x2A (*), 0x2B (+), 0x2C (,), 0x2E (.), 0x2F (/), 0x3A (:), 0x3B (;), 0x3C (<), 0x3D (=), 0x3E (>), 0x3F (?), 0x5B ([), 0x5C (\), 0x5D (]), 0x7C (|) a 0xE5 (σ). Některé z těchto znaků mají speciální význam, jak bude vysvětleno dále. DOS při práci se soubory/adresáři automaticky převádí malá písmena na velká, proto nezáleží jak je zadáme.

S příchodem Windows 95 byl souborový systém FAT16 rozšířen o podporu dlouhých názvů (LFN), označuje se jako VFAT16 a je částečně kompatabilní s předchozími verzemi DOSu. Pro ukládání dlouhého názvu (až 255 znaků) se používá více adresářových položek, z nichž jedna nese zkrácený název podle konvence 8.3, který je viditelný pro starší verze DOSu a další položky označené atributem 0x0F (S+H+R+L+) s dlouhým názvem, které jsou pro starší verze DOSu neviditelné. Např. „Dlouhý název.txt“ je zkrácen na „DLOUHY~1.TXT“, kde číslo za vlnovkou slouží k rozlišení v případě, že by vzniklo více stejných zkrácených názvů. Adresářové položky nesoucí dlouhý název jsou uloženy před položkou s krátkým názvem od konce a obsahují pořadové číslo položky dlouhého názvu (nedohledal jsem se z jakého důvodu Microsoft vybral zrovna toto pořadí, dle mého názoru by bylo výhodnější tyto položky ukládat až za položku s krátkým názvem). Dlouhý název je kódovaný v UTF-16 (viz. např. [17]), takže dovoluje použití velkého množství různých národních a speciálních znaků. Do jedné adresářové položky se vejde až 13 UTF-16 znaků, z toho vyplývá že soubor o délce 255 znaků zabere celkem 21 adresářových položek (20 s LFN a 1 s krátkým názvem). Protože kořenový adresář má fixní velikost (typicky 512 položek), může při použití LFN snadno dojít k jejich vyčerpání. Podrobnější popis LFN viz [10], kapitola Dlouhá jména souborů ve Win95.

Byte atributů rozlišuje o jakou položku se vlastně jedná, zda-li soubor nebo adresář, či jmenovku oddílu a další vlastnosti, viz tab. 2.4.4.2. Pokud má daný bit hodnotu 1, je atribut aktivní.

Tabulka 2.4.4.2: Flagy atributu adresářové položky

bit č.	popis atributu
0	Read only – pouze pro čtení (DOS odmítne zápis/výmaz)
1	Hidden – skrytá položka, nevypíše se v příkazu DIR
2	System – systémová položka, nelze ji přesunout na jiný cluster
3	volume Label – položka značí jmenovku disku
4	Directory – položka je adresářem
5	Archive – položka nebyla zálohovaná programem na zálohy.

Atributy R, H, S, A mají spíše jen informativní význam, protože většina programů na PC umožňuje význam těchto atributů potlačit. Např. při editaci souboru a aktivním atributem R+ editor při ukládání jenom zobrazí zprávu, že soubor je jen pro čtení a jestli si ho přejeme opravdu uložit. Nelze to tedy v žádném případě srovnávat s oprávněními v UNIXu.

Při vytváření souborového systému se také v kořenovém adresáři obvykle vytvoří jedna položka s atributem Volume Label, která kopíruje jmenovku disku uvedenou v DBR a nealokuje žádný cluster ve FAT. Příkazy DOSu VOL a LABEL čtou jmenovky z této adresářové položky, LABEL pak synchronizuje položku v DBR.

Datum a čas vytvoření a posledního přístupu k souboru začaly využívat až Windows 9x. MS-DOS standardně ukládá pouze datum a čas posledního zápisu/modifikace souboru. Aby se ušetřilo místo v adresářové položce, je struktura datumu a času zapakovaná do 16-ti bitů jak je uvedeno v tabulce 2.4.4.3 a 2.4.4.4:

Tabulka 2.4.4.3: Pakovaná struktura času

H4	H3	H2	H1	H0	M5	M4	M3	M2	M1	M0	S4	S3	S2	S1	S0

H – hodina: 0-23, M – minuta: 0-59, S·2 – sekunda: 0-59 s krokem po 2

Tabulka 2.4.4.4: Pakovaná struktura datumu

Y6	Y5	Y4	Y3	Y2	Y1	Y0	M3	M2	M1	M0	D4	D3	D2	D1	D0

Y+1980 – rok: 1980-2107, M – měsíc: 1-12, D – den v měsíci: 1-31

Předposlední položka adresářové struktury udává 16-bitovou adresu prvního clusteru ve FAT16 (FAT32 ještě využívá dalších 16 bitů na offsetu 0x14 jako vyšší slovo 32-bitové adresy) – začátek řetězce clusterů. Poslední 32-bitová položka udává velikost souboru v bytech, z toho plyne limit 4 GB pro max. velikost souboru. Adresáře nebo jmenovka mají nastavenou nulovou velikost.

2.4.5 Podadresáře a soubory

MS-DOS od verze 2.0 (1983) podporuje vytváření stromové adresářové struktury. Hloubka stromu není nijak striktně omezena, ale u DOSu bývá problém s omezením délky cesty asi na 120 znaků. Správně by se měl každý adresář kromě kořenového nazývat podadresář. Podadresáře se od kořenového adresáře liší hlavně v tom, že počet jejich položek není pevně daný a může se zvětšovat nebo zmenšovat podle potřeby. Při vytvoření podadresáře v kořenovém adresáři se do kořene zapíše standardní adresářová položka (tab. 2.4.4.1) se jménem, příponou, aktivním atributem D+, nulovou velikostí a adresou prvního clusteru ve FAT. Do tohoto clusteru se pak budou zapisovat adresářové položky souborů nebo podadresářů

obsažených v tomto podadresáři. Až se podadresář zaplní, tak se jednoduše ve FAT alokuje další volný cluster a připojí se do řetězce clusterů podadresáře. Při dopředném procházení adresářové struktury pak jen stačí najít a přečíst položku podadresáře a získat z ní adresu prvního clusteru ve FAT. Po načtení tohoto clusteru lze opět zpracovávat další adresářové položky v něm obsažené stejným způsobem.

Aby bylo možno adresářovou strukturou procházet i zpět směrem ke kořeni, vytváří DOS v novém podadresáři dvě speciální položky se jmény „**.**“ a „**..**“. To je také jeden z důvodů, proč by se tečka neměla vyskytovat v názvech uživatelských souborů. Položka „**.**“ obsahuje adresu prvního clusteru podadresáře v němž je umístěna. Položka „**..**“ pak obsahuje adresu prvního clusteru nadřazeného podadresáře. Je-li nadřazeným adresářem kořen, je tato adresa nulová. DOS ještě používá speciální symbol „\“ pro kořenový adresář, ale to už je jen symbolické označení, fyzicky žádná adresářová položka „\“ neexistuje. V tab. 2.4.5.1 je ukázka zobrazení souborů a adresářů v Norton Disk Editor. Tučně jsou zvýrazněny položky „**.**“ a „**..**“. Jak vidíme, zobrazovaný podadresář začíná na clusteru 36338 a jeho nadřazený adresář je kořen. V jednotlivých sloupcích přehledně vidíme název, příponu, velikost, datum, čas, první cluster a atributy položek.

Tabulka 2.4.5.1: Ukázka adresářové struktury v programu Norton Disk Editor

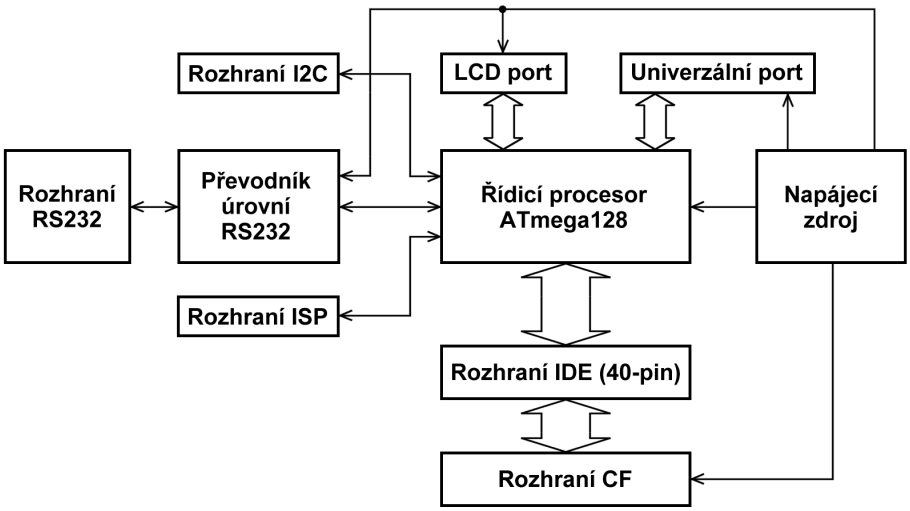
```
                              Disk Editor
   Object  Edit  Link  View  Info  Tools  Help
Name     .Ext     Size      Date      Time    Cluster Arc R/O Sys Hid Dir Vol
Cluster 36,338, Sector 2,326,049
.                        0   1-10-06  11:19 pm  36338                    Dir
..                       0   1-10-06  11:19 pm      0                    Dir
DOCS                     0   1-10-06  11:19 pm  36340                    Dir
MOUSE                    0   1-10-06  11:19 pm  36466                    Dir
PLUS                     0   1-10-06  11:19 pm  36474                    Dir
WIN32                    0   1-10-06  11:19 pm  36526                    Dir
4201     CPI          6404   3-06-01   6:52 pm  36675   Arc
4208     CPI           720   3-06-01   6:52 pm  36676   Arc
5202     CPI           395   3-06-01   6:52 pm  36677   Arc
ACALC    EXE         22851   3-06-01   6:52 pm  36678   Arc
ADOS     CFG            61   3-06-01   6:52 pm  36679   Arc
ADOS     COM         31086   3-06-01   6:52 pm  36680   Arc
ADOS     OVL         92154   3-06-01   6:52 pm  36681   Arc
AM       EXE          7825   3-06-01   6:52 pm  36684   Arc
ANSI     COM          1371  10-28-94   2:15 am  36685   Arc
ANSI     SYS          8906   3-06-01   6:52 pm  36686   Arc
Cluster 36,338, Sector 2,326,050
APPEND   EXE         10774   8-23-99   7:56 am  36687   Arc
ASSIGN   COM          5102   3-06-01   6:52 pm  36688   Arc
   Sub-Directory                                          Cluster 36,338
   C:\DOS                                               Offset 32, hex 20
```

Soubory se od podadresářů liší tím, že nemají aktivní atribut D a mají obvykle nenulovou velikost, dál už je to jen otázka interpretace dat (např. není žádný problém adresáři vynulovat atribut D a číst ho jako soubor). Do alokovaných clusterů souboru lze ukládat libovolná data. Při změně velikosti souboru je třeba jednak zapsat správnou hodnotu velikosti do adresářové položky a také uvolnit případné dále nepoužívané clustery ve FAT. Pokud bychom je neuvolnili, na disku by se hromadily tzv. **ztracené clustery** (lost clusters), až by se disk zaplnil.

3 NÁVRH HARDWARE

V této kapitole popíšu návrh hardware dataloggeru. Při výběru vhodného řídicího procesoru je hardware poměrně jednoduchý a obsahuje minimum vnějších součástek. Vše ostatní je už v kompetenci řídicího software, který bude popsán v kapitole 4. Součástí zadání je také realizace funkčního vzorku. Pro úvodní seznámení s procesorem a CF rozhraním jsem vytvořil jednoduchý prototyp postavený na univerzálním plošném spoji. Po oživení a dostatečném odzkoušení funkce jsem v programu OrCAD vytvořil finální návrh na jedné desce oboustranného plošného spoje integrující všechny potřebné periferie. Mechanické rozměry plošného spoje byly zvoleny tak, aby umožnily zamontování do plastové krabičky U-KP02.

Blokové schéma dataloggeru je na obr. 3.1. Skládá se ze čtyř hlavních částí: řídicí procesor, rozhraní, převodník úrovní pro rozhraní RS232 a napájecí zdroj. Jednotlivé části budou popsány v následujících podkapitolách. Podrobné schéma zapojení, seznam součástek a obrazec plošného spoje viz příloha A.



Ilustrace 3.1: Blokové schéma dataloggeru

3.1 ŘÍDICÍ PROCESOR ATMEL ATMEGA

Pro danou úlohu jsem zvolil 8-bitový RISCový jednočipový mikropočítač ATmega z rodiny AVR firmy Atmel. V prototypu jsem použil typ ATmega32 a ve finálním zařízení pak typ ATmega128. Ten disponuje dostatečnou rezervou výkonu, paměti a periferií pro případnou implementaci dalšího software, který bude využívat stávající knihovny na obsluhu CF a souborové operace. Poprvé jsem se s procesory ATmega setkal na předmětu 31SCS (Speciální číslicové systémy), kde mě zaujaly svým vybavením a výkonem. V neposlední řadě je také sortiment firmy Atmel u nás dobře dostupný za rozumné ceny.

Stručně uvedu některé výhody:

- vysoký výkon až 16 MIPS @16MHz (většina instrukcí se provede v 1 cyklu),
- nízká spotřeba, podpora různých power módů,
- dostatečná velikost interních pamětí,
- široké spektrum integrovaných periferií,
- možnost programování přímo na desce přes rozhraní SPI,
- dostupnost vývojových nástrojů zdarma, překladač jazyka C (GCC + AVR-libC).

Dále se zaměřím na vlastnosti použitého procesoru ATmega128 (více v [18]):

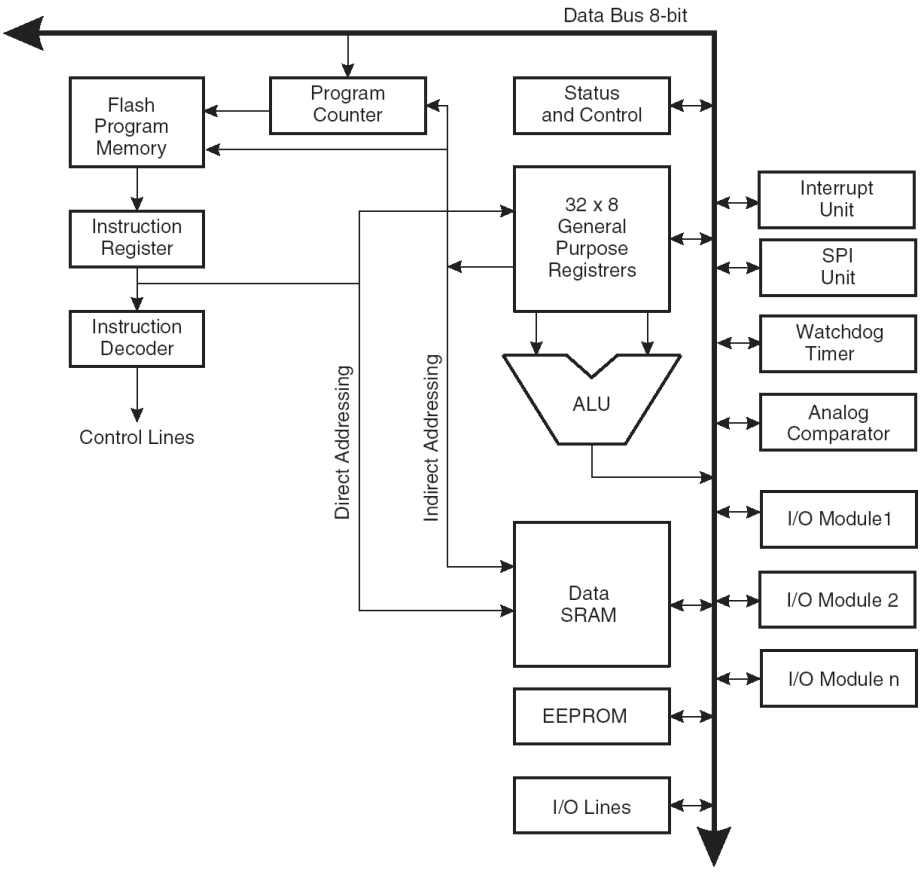
- napájecí napětí: 4,5 – 5,5 V (L varianta 2,7 – 5,5 V),
- kmitočet oscilátoru: 0 – 16 MHz (0 – 8 MHz pro 8 MHz verzi),
- 128 kB Flash EPROM pro uložení programu a dat s výdrží 10000 přepisů,
- 4 kB EEPROM pro uložení dat s výdrží 100000 přepisů,
- 4 kB SRAM pro uložení dat, možnost adresace až 64 kB externí SRAM,
- 4 8-bit programovatelné konfigurační pojistky (zahrnuje ochranu software),
- programování přes rozhraní SPI, JTAG nebo pomocí bootcode programu,
- 2 8-bitové časovače se samostatnými předděličkami,
- 2 16-bitové časovače se samostatnými předděličkami,
- 2 8-bitové PWM kanály a 6 PWM kanálů s program. rozlišením 2 – 16 bitů,
- 8-kanálový 10-bitový AD převodník, 2 kanály s programovatelným ziskem,
- analogový komparátor,
- sériové rozhraní I2C, dvě sériové rozhraní SPI,
- 2 programovatelné USARTy,
- programovatelný RESET po zapnutí a detekce podpětí,
- programovatelný watchdog,
- kalibrovaný interní programovatelný RC oscilátor umožňuje funkci bez krystalu,
- 6 power módů,
- externí a interní přerušení,
- 53 programovatelných I/O linek s můstkovými výstupy a volitelnými pull-upy,
- čtvercové pouzdro TQFP64 pro SMD montáž.

Z toho datalogger využívá jen pár funkcí, ostatní budiž jako rezerva pro další možné rozšíření.

3.1.1 RISCové jádro AVR

Procesor ATmega128 je založen na RISCovém jádru AVR Harvardské architektury, jehož blokové schéma je na obr. 3.1.1.1. To se stará o provádění programu, přístupu k pamětem a periferiím a obsluhu přerušení. Pro zrychlení provádění instrukcí má jednoúrovňovou pipeline, kdy během provádění jedné instrukce se následující instrukce načítá z programové paměti.

ALU (aritmeticko-logická jednotka) má přímý přístup k 32 8-bitovým GPR (registrům pro všeobecné použití) uspořádaných do 8 banků. Během jednoho cyklu ALU načte až 2 operandy z GPR, provede výpočet a výsledek uloží zpět do GPR. 6 8-bitových GPR registrů může být použito jako 3 16-bitové registry (X, Y, Z) pro nepřímé adresování datové paměti a jeden z nich i pro adresaci programové Flash paměti (pro přístup k různým LUT tabulkám, konstantám, a pod.). ALU podporuje standardní aritmetické, logické a bitové operace. Některé AVR procesory, jako např. ATmega navíc mají i hardwarovou násobičku pro celočíselnou a desetinnou aritmetiku (výpočet trvá 2 cykly). Po provedení operace ALU nastaví příslušné flagy ve stavovém registru.



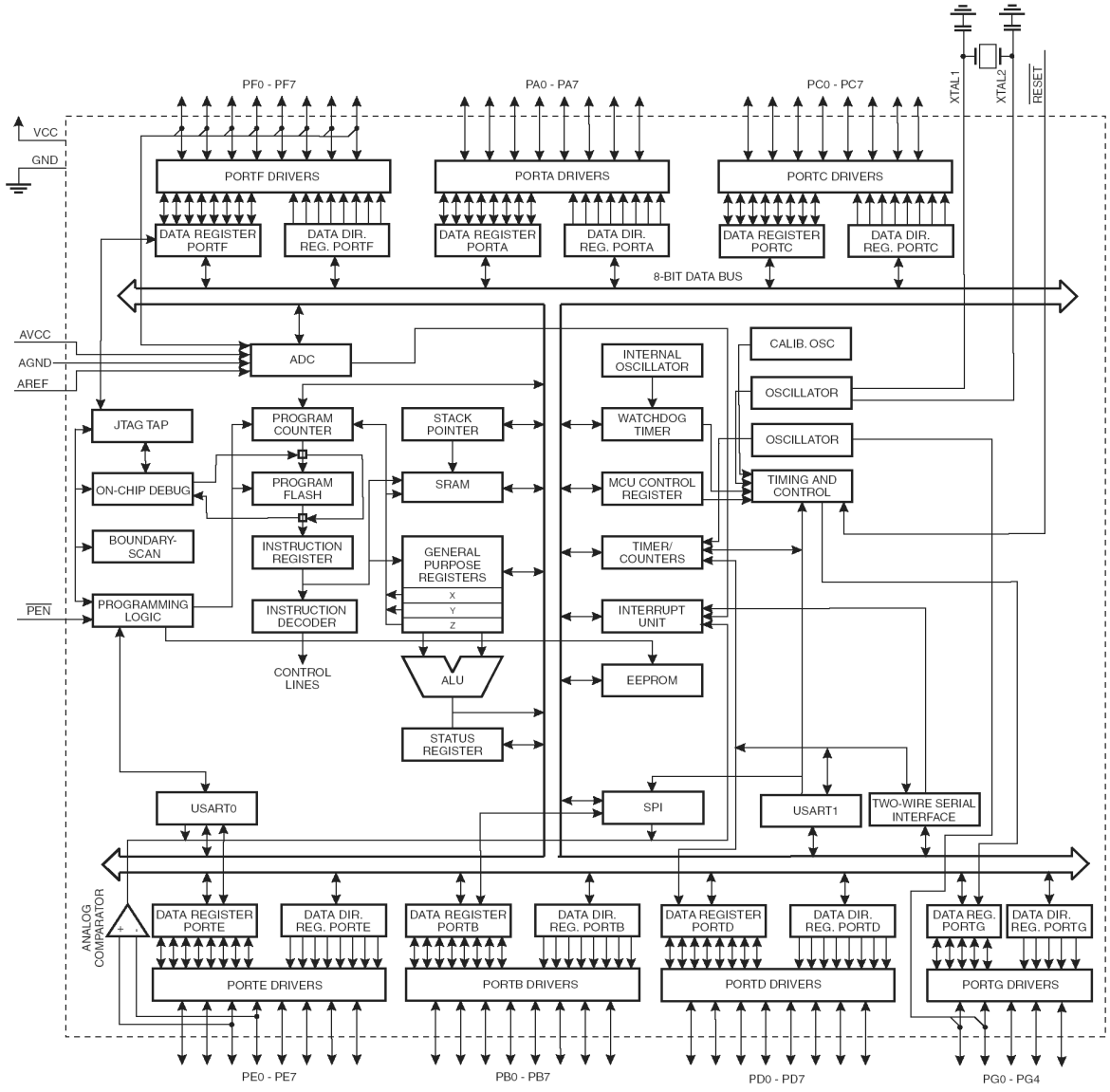
Ilustrace 3.1.1.1: Blokové schéma jádra AVR

Běh programu (registr program counter) lze řídit podmíněnými a nepodmíněnými skoky, voláním a návratem z podprogramu a voláním obsluh interních či externích přerušení.

Programová i datová paměť má lineární adresování. Programová paměť Flash se dělí na oblast aplikační a oblast zavaděče (bootcode). Každá oblast má svoji pojistku proti čtení/zápisu. V zaváděcí oblasti může být uložen program zavaděče, který přes nějaké rozhraní přijme aplikační program a pomocí instrukce SPM (nelze ji volat programem z aplikační oblasti) jej zapíše do aplikační paměti a pak mu předá řízení. Tento mechanismus lze využít pro programování i přes jiné rozhraní, než k tomu určené SPI/JTAG.

Zásobník sdílí interní paměť SRAM a roste směrem dolů. Před voláním podprogramů nebo obsluh přerušení je nutné nastavit ukazatel zásobníku (SP) tak, aby byl k dispozici dostatek paměti pro návratové adresy. Každé přerušení lze individuálně nebo globálně povolit/zakázat. Priorita přerušení je daná adresou vektorů přerušení (čím nižší adresa vektoru, tím vyšší priorita).

Kompletní blokové schéma mikropočítače ATmega128 je na obr. 3.1.1.2.



Ilustrace 3.1.1.2: Blokové schéma vnitřní struktury mikropočítače ATmega128

3.1.2 Programovatelné konfigurační pojistky

Kromě běžné Flash a EEPROM paměti má procesor ATmega ještě čtyři 8-bitové EEPROM buňky, tzv. **pojistky** (fuses), které se programují extra. Tyto pojistky určují některá specifická nastavení mikropočítače pro dané zapojení a obvykle je po správném nastavení není třeba dále měnit. Nastavení ochranných bitů má znemožnit další čtení/zápis programové paměti. Pokud je daná pojistka nenaprogramovaná, má bit hodnotu **1**, pokud je naprogramovaná, má bit hodnotu **0**. V tab 3.1.2.1 je význam pojistkového byte Lock Bits.

Tabulka 3.1.2.1: Pojistkový byte Lock Bits

-	-	BLB12	BLB11	BLB02	BLB01	LB2	LB1

Bity ***BLB**xx* se týkají zaváděcí oblasti a bity ***BL**x* aplikační oblasti. Od výrobce jsou všechny nastaveny na 1, tedy bez žádných přístupových omezení. Např. kombinací LB2 = 1, LB1 = 0 se zabrání dalšímu programování Flash, EEPROM a ostatních pojistek přes SPI/JTAG. Pokud je navíc LB2 = 0, nelze z oblastí ani číst. Další podrobnosti viz [18], str. 288.

V tab. 3.1.2.2 je popis rozšířeného pojistkového byte.

Tabulka 3.1.2.2: Pojistkový byte Extended Fuse

-	-	-	-	-	-	M103C	WDTON

M103C – režim kompatability s mikropočítačem ATmega103. V tomto režimu jsou některé periferie ATMega128 vypnuty nebo omezena jejich funkce tak, aby byl zpětně kompatabilní s ATmega103. Od výroby je tento bit naprogramovaný, pro správnou funkci v dataloggeru je třeba jej vypnout.

WDTON – znemožňuje vypnutí watchdogu (od výroby 1).

Další pojistkový byte je v tab. 3.1.2.3.

Tabulka 3.1.2.3: Pojistkový byte High Fuse

OCDE	JTAGE	SPIE	CKOP	EESAV	BS1	BS0	BRST

OCDE – zapne funkci On-Chip debug pro ladění přes JTAG – TAP (od výroby 1).

JTAGE – zapne rozhraní JTAG – linky TDI, TDO, TMS, TCK místo portu F.4 – F.7 (od výroby 0).

SPIE – zapne programovací rozhraní SPI – linky PDI, PDO místo portu PE.0, PE.1 (od výroby 0).

CKOP – nastavuje režim oscilátoru - pokud je bit naprogramovaný, bude budič externího krystalového oscilátoru používat plný rozkmit napětí což je nutné pro frekvence vyšší jak 8 MHz nebo v zarušeném prostředí. Tento režim má ale větší spotřebu. Pokud je bit nenaprogramovaný, používá budič menší amplitudu napětí (od výroby 1).

EESAV – pokud je bit naprogramovaný, zůstane zachován obsah EEPROM při programování Flash (resp. příkazu Chip Erase, od výroby 1).

BS0, BS1 – nastavuje velikost zaváděcí oblasti 512 (0b11), 1024 (0b10), 2048 (0b01) nebo 4096 B (0b00, od výroby 0b11).

BRST – nastavuje vektor resetu - pokud je bit naprogramovaný, spustí se po resetu zavaděč ze zaváděcí oblasti, jinak se spustí program z aplikační oblasti (od výroby 1).

Poslední pojistkový byte je v tab. 3.1.2.4.

Tabulka 3.1.2.4: Pojistkový byte Low Fuse

BODL	BODE	SUT1	SUT0	CKS3	CKS2	CKS1	CKS0

BODL – určuje úroveň, při jakém poklesu napájecího napětí vyvolá detektor podpětí reset. Pro hodnotu 1 je to 2,7 V a pro 0 je to 4,0 V (od výroby 1).

BODE – zapíná detektor podpětí, tzv. Brown out detector (od výroby 1), podrobnosti v [18], str. 50.

SUT0, SUT1 – nastavuje přídavné časové zpoždění rozběhu procesoru po zapnutí napájecího napětí (od výroby 0b10), podrobnosti v [18], str. 36.

Tabulka 3.1.2.5: Hodnoty přídavného zpoždění náběhu po zapnutí napájení

CKS:	X, CKS0=0	X, CKS0=1	LF Xtal	int. RC	ext. RC
SUTx	doba [ms]	doba [ms]	doba [ms]	doba [ms]	doba [ms]
0b00	4,1	65	4,1	-	-
0b01	65	-	65	4,1	4,1
0b10	-	4,1	65	65	65
0b11	4,1	65	-	-	4,1

CKS0-CKS3 – slouží pro volbu zdroje hodinového signálu (X – externí krystal, LF Xtal – krystal o nízké frekvenci, RC – interní/externí RC oscilátor), viz tab. 3.1.2.6 (od výroby 0b0001), další detaily v [18], str. 35. Frekvence oscilátoru může být dále

dynamicky vydělena pomocí registru XTAL Divide Control Register faktorem 2-129, viz [18], str. 41.

Tabulka 3.1.2.6: Různé druhy zdrojů hodinového signálu

CKSx	popis atributu
0b111x[1]	externí krystal 3 – 8 MHz; pro CKOPT=0 1–16 MHz
0b110x[1]	externí krystal 0,9 – 3 MHz; pro CKOPT=0 1–16 MHz
0b101x[1]	externí krystal 0,4 – 0,9 MHz; pro CKOPT=0 1–16 MHz
0b1001	externí nf krystal, typ. 32 kHz
0b1000	externí RC oscilátor 8 – 12 MHz
0b0111	externí RC oscilátor 3 – 8 MHz
0b0110	externí RC oscilátor 0,9 – 3 MHz
0b0101	externí RC oscilátor 0,1 – 0,9 MHz
0b0100	kalibrovaný interní RC oscilátor 8 MHz
0b0011	kalibrovaný interní RC oscilátor 4 MHz
0b0010	kalibrovaný interní RC oscilátor 2 MHz
0b0001	kalibrovaný interní RC oscilátor 1 MHz
0b0000	externí zdroj hodinového signálu

[1]bit CKS0 slouží spolu se SUT0, SUT1 pro výběr přídavné doby náběhu

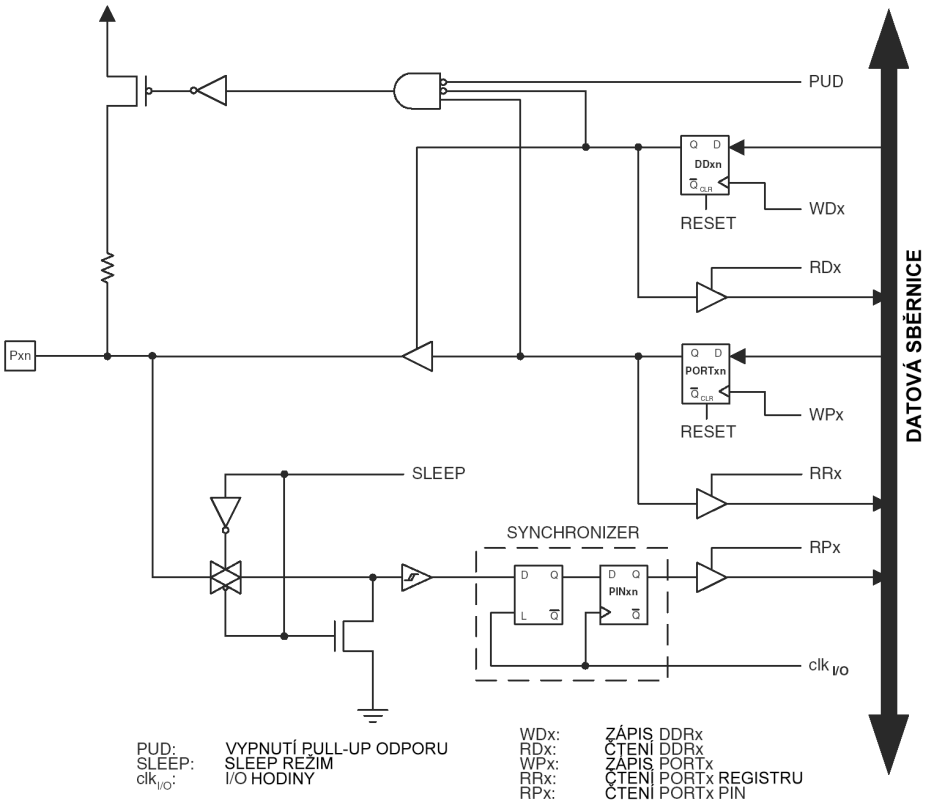
Konkrétní konfigurace pojistek v pro správnou funkci dataloggeru je uvedena v hlavičce zdrojového kódu programu v souboru *MAIN.C*, viz příloha na CD-ROM.

3.1.3 I/O porty

Mikropočítač ATmega128 má 53 programovatelných univerzálních obousměrných portů. Většina z nich má 1 nebo 2 alternativní významy, které se obvykle aktivují zapnutím dané periferie pomocí jejího řídicího registru. Porty jsou uspořádány do osmic (port A – G), které lze nastavovat najednou nebo každý bit individuálně. Výstupy portů jsou řešeny jako komplementární pár CMOS se schopností dodávat/odebírat až 40 mA, celkově však nesmí proud všech I/O portů překročit 400 mA. U každého vstupu zvlášť lze zapnout pull-up rezistor o hodnotě 20 – 50 kΩ. Vstupy mají ochranné diody zapojené proti Vcc a gnd. Na obr. 3.1.3.1 je schéma řídicí logiky jednoho I/O portu. Každá osmice portů má tři 8-bitové řídicí registry:

DDRx – (Data Direction Register) souží pro konfiguraci portu jako vstup (hodnota 0) nebo výstup (hodnota 1).

PORTx – slouží pro nastavení hodnoty na pinu I/O portu, je-li port nakonfigurován registrem DDRx jako vstup, řídí nastavení pull-up rezistoru (hodnota 1 – pull-up je aktivní).



Ilustrace 3.1.3.1: Řídicí logika I/O portu

Všechny pull-upy lze globálně vypnout přes registr SFIOR, bit PUD. všechny dovolené kombinace jsou popsány v tab. 3.1.3.1. Po resetu jsou všechny porty nastaveny jako vstupy s vypnutým pull-upem. Čtení registru PORTx nám dá předchozí zapsanou hodnotu, nikoliv skutečný stav na pinu.

Tabulka 3.1.3.1: Možné konfigurace I/O portu

DDRx	PORTx	PUD	I/O	pull-up	popis
0	0	x	vstup	ne	třetí stav, vysoká impedance
0	1	0	vstup	ano	přes pull-up teče proud z Vcc
0	1	1	vstup	ne	třetí stav, vysoká impedance
1	0	x	výstup	ne	výstup v log. 0
1	1	x	výstup	ne	výstup v log. 1

PINx – souží pro čtení skutečného stavu pinu (nezávisle na stavu DDRx), synchronizuje se s hodinovým signálem.

Z logiky věci dále plyne, že pokud chceme přepnout port z režimu vstup do režimu výstup s danou hodnotou bez nechtěné mezistavové hodnoty, je třeba nejprve zapsat registr

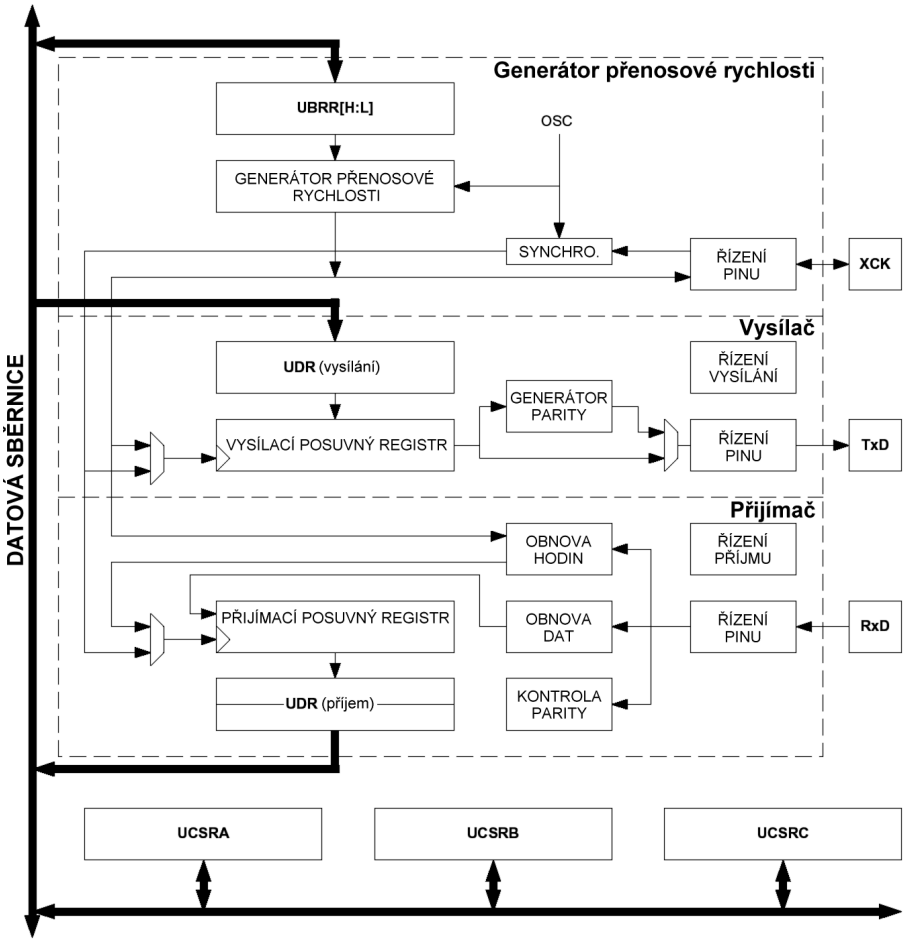
PORTx a pak teprve DDRx. Další popis portů v [18], str. 63.

3.1.4 USART

Mikropočítač ATmega128 má 2 programovatelné USARTy (Universal Synchronous/ Asynchronous Receiver and Transmitter) pro sériovou komunikaci. Jeden z nich ale sdílí piny s programovacím SPI rozhraním, takže v dataloggeru je použitelný jen jeden. Krátký výčet některých schopností USARTu:

- plný duplex (oddělené přijímací a vysílací registry),
- generátor přenosové rychlosti s jemným dělením,
- podpora sériových rámců 5 – 9 datových bitů, 1 start-bit, 1 nebo 2 stop-bity,
- hardwarový generátor sudé i liché parity a kontrola parity,
- šumové filtry (digitální LP filtr) a detekce falešného start-bitu,
- 3 přerušení generovaná ukončením vysílání/příjmu a prázdným vysílacím reg.

Blokové schéma USARTu je na obr. 3.1.4.1. Tučně jsou vyznačeny přístupné registry.



Ilustrace 3.1.4.1: Blokové schéma USARTu

USART se skládá ze 3 základních celků: generátor přenosové rychlosti, vysílač a přijímač. V dataloggeru se používá jako UART (Universal Asynchronous Receiver and Transmitter) pro komunikaci po RS232, pin **XCK** a obvod synchronizace z vnějších hodin je tedy nevyužit. Pro nastavení požadované bitové rychlosti slouží 16-bitový registr **UBRR**. Konkrétní rychlost podle hodnoty **UBRR** registru zjistíme ze vztahu 3.1.1 a potřebnou hodnotu UBRR [0 – 4095] pro požadovanou rychlost nám dá vztah 3.1.2. Detaily v [18], str. 172.

Vztah 3.1.1:

$$BAUD = f_{osc}/[16 \cdot (UBRR + 1)]$$

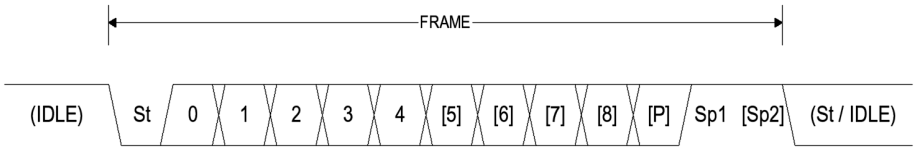
kde:

f_{osc} je hodinový kmitočet procesoru
UBRR je hodnota UBRR registru
BAUD je bitová rychlost v baudech

Vztah 3.1.2:

$$UBRR = f_{osc}/(16 \cdot BAUD) - 1$$

Vysílač obsahuje posuvný registr do nějž se zápisem datového registru **UDR** vloží vysílaná data, která jsou dále automaticky zpracována. Jakmile je **UDR** připraven pro další zápis dat, je vyvoláno přerušení (pokud je povoleno). Podle požadavku je vypočtena sudá nebo lichá (nebo žádná) parita. Řídící logika pak v rytmu hodin generátoru bitové rychlosti vysílá jednotlivé bity doplněné o start, paritu a stop bity přes výstupní budič na pin **TxD**. Jakmile je přenos dokončen, je vyvoláno další přerušení (pokud je povoleno). Rámec sériového přenosu je na obr. 3.1.4.2. St značí start-bit, 0 – 4 povinné datové datové bity, [5] – [9] volitelné datové bity, Sp1 a [Sp2] povinný a volitelný stop-bit a IDLE je klidový stav na lince. Detaily v [18], str. 175.



Ilustrace 3.1.4.2: Rámec sériového přenosu

Přijímač přijímá sériový tok bitů pinem **RxD**. Nejprve se provádí filtrace ze účelem odstranění šumu a rušení a regenerace tvaru signálu. Rekonstruovaným hodinovým signálem se řídí přijímací posuvný registr, který postupně střádá datové bity. Z přijatých dat se spočítá a

zkontroluje parita. Po dokončení příjmu slova je vyvoláno přerušení (pokud je povoleno) a přijatá data lze přečíst z registru **UDR**. Detaily v [18], str. 180.

Další nastavení USARTu a čtení stavových informací se provádí přes tři 8-bitové registry **UCSRA**, **UCSRB** a **UCSRC**, viz tab. 3.1.4.1, 3.1.4.2 a 3.1.4.4. Detaily v [18], str. 189.

Tabulka 3.1.4.1: Konfigurační/stavový registr UCSRA

RXC	TXC	UDRE	FE	DOR	UPE	U2X	MPCM

RXC – je-li nastaven na 1, signalizuje že registr **UDR** obsahuje přijatá data k přečtení. Může generovat přerušení.

TXC – nastaví se do 1 po odeslání dat z posuvného registru. Může generovat přerušení.

UDRE – je-li nastaven na 1, lze do registru **UDR** zapisovat nová data k odeslání. Může generovat přerušení.

FE – je-li nastaven na 1, signalizuje chybu rámce v přijatém slovu, které je v registru **UDR**. Jeho přečtením se flag automaticky vynuluje.

DOR – je-li nastaven na 1, signalizuje přetečení dat (nastane, pokud z registru **UDR** nebyla včas přečtena data, posuvný registr je plný a přijímač detekoval další start-bit).

UPE – je-li nastaven na 1, signalizuje chybu parity přijatého slova, které je v registru **UDR**.

U2X – nastavením do 1 se zvýší bitová rychlost UARTu na dvojnásobek (konstanta 16 ve vztahu 3.1.1 a 3.1.2 se redukuje na 8). Pro synchronní přenos má být nastaven na 0.

MPCM – nastavením do 1 zapíná podporu multiprocesorové komunikace, viz [18], str. 187.

Tabulka 3.1.4.2: Konfigurační/stavový registr UCSRB

RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCZS2	RXB8	TXB8

RXCIE – nastavením do 1 povolí generovaní přerušení při dokončení příjmu (***RXC*** = 1).

TXCIE – nastavením do 1 povolí generovaní přerušení při dokončení vysílání (***TXC*** = 1).

UDRIE – nastavením do 1 povolí generovaní přerušení při vyprázdnění vysílacího registru **UDR** (***UDRE*** = 1).

RXEN – nastavením do 1 zapíná blok přijímače.

TXEN – nastavením do 1 zapíná blok vysílače.

UCZS0-2 – slouží pro nastavení počtu datových bitů v rámci, platné hodnoty viz tab. 3.1.4.3.

Tabulka 3.1.4.3: Výběr nastavení počtu datových bitů rámce

UCZS2	UCZS1	UCZS0	počet datových bitů
0	0	0	5
0	0	1	6
0	1	0	7
0	1	1	8
1	1	1	9

RXB8 – datový bit 8 (příjem) pro 9-ti bitové slovo.

TXB8 – datový bit 8 (zápis) pro 9-ti bitové slovo.

Tabulka 3.1.4.4: Konfigurační/stavový registr UCSRC

-	UMSEL	UPM1	UPM0	USBS	UCZS1	UCZS0	UCPL

UMSEL – volí mezi synchronním (hodnota 1) a asynchronním (hodnota 0) režimem.

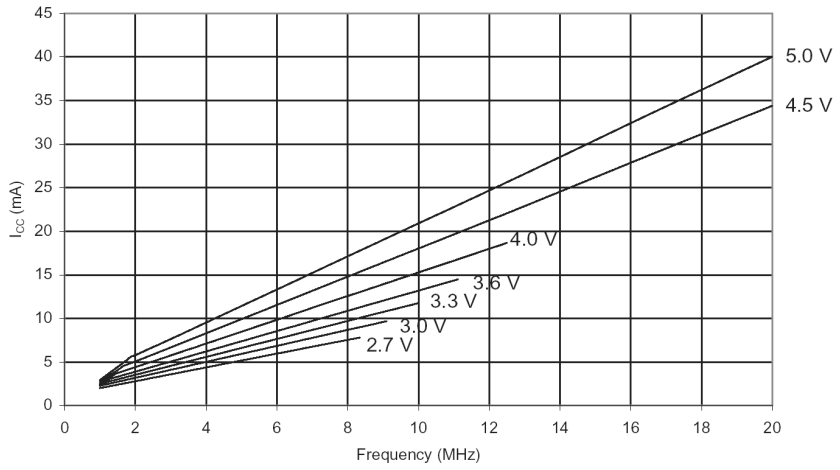
UPM0, UPM1 – nastavuje paritu: žádná (0b00), sudá (0b10), lichá (0b11).

USBS – nastavuje počet stop-bitů: jeden (hodnota 0) nebo dva (hodnota 1).

UCPL – nastavuje polaritu hodin pro synchronní přenos: příjem na sestupnou hranu (hodnota 0) nebo na vzestupnou hranu (hodnota 1). Pro asynchronní přenos se má nastavit na 0.

3.1.5 Proudový odběr

Na obr. 3.1.5.1 je znázorněna závislost proudového odběru procesoru v aktivním režimu



Ilustrace 3.1.5.1: Závislost proudového odběru na pracovní frekvenci

(za podmínky, že není odebírán žádný proud z I/O portů) na pracovní frekvenci. U CMOS obvodů je typická tato téměř lineární závislost. Např. pro 16 MHz a napájecí napětí 5 V je ztrátový výkon 0,16 W. Spotřebu lze snížit přepnutím do některého ze sleep módů, viz [18], str. 42, dynamickým řízením frekvence a také vypnutím nepoužívaných periferií.

3.2 PŘEVODNÍK ÚROVNÍ PRO ROZHRANÍ RS232

RS232 je rozhraní pro sériový přenos informací vytvořené původně pro komunikaci dvou zařízení do vzdálenosti 20 m. Pro větší odolnost proti rušení je informace po propojovacích vodičích přenášena větším napětím, než je standardních 5 V. Běžné PC je vybaveno 1 až 2 porty RS232. Pro přenosné počítače, kde toto rozhraní již téměř vymizelo, existují převodníky RS232/USB (ve Windows se chová jako virtuální COM port).

Protože USART procesoru ATmega používá standardní 5 V úrovně, musí se pro komunikaci s PC použít převodník úrovní na RS232. V tab. 3.2.1 jsou popsány úrovně RS232. Více podrobností o normě RS232 viz [19].

Tabulka 3.2.1: Napěťové úrovně RS232 pro vysílač a přijímač

úroveň	na straně vysílače	na straně přijímače
log. 0	+5 V až + 15 V	+3 V až +25 V
log. 1	-5 V až - 15 V	-3 V až -25 V
zakázaná	-3 V až +3 V	

Výstupní proud budiče na krátko je omezen přibližně na 10 mA

Pro tento účel jsem zvolil rozšířený a dobře dostupný integrovaný obvod MAX232, který obsahuje dva páry převodníků úrovní. Jeho hlavní výhodou je, že ke své funkci potřebuje jen 5 V napájení. Obsahuje totiž zdvojovače a invertory na principu nábojové pumpy, které vytvoří potřebná kladná a záporná napětí pro buzení RS232. Počet potřebných vnějších součástek je minimální a omezuje se na 4 kondenzátory pro nábojové pumpy a jeden blokovací kondenzátor napájecího napětí. Schéma zapojení převodníku je součástí hlavního schématu dataloggeru v příloze A. Detailní popis obvodu MAX232 viz [20].

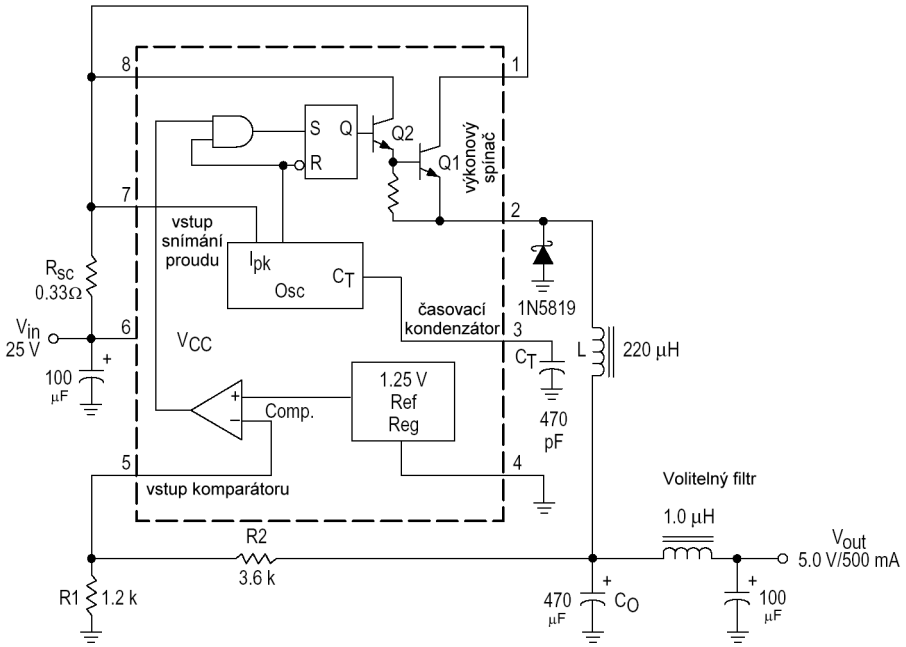
3.3 NAPÁJECÍ ZDROJ

Funkčnost dataloggeru byla požadována pro rozsah napájecích napětí +12 V až +24 V stejnosměrných. Vzhledem k poměrně velkému rozdílu vstupního a výstupního napětí zdroje jsem zavrhl jednoduché lineární stabilizátory řady 78xx a použil snižující pulsní měnič s integrovaným obvodem Motorola MC34063A. Stručný přehled parametrů:

- vstupní napětí +3 V až +40 V
- spínaný proud až 1,5 A

- proudové omezení
- pracovní frekvence až 100 kHz
- nastavitelné výstupní napětí
- interní napěťová reference s přesností 2 %

Zapojení zdroje dataloggeru vychází z katalogového zapojení obvodu v [21], str. 6, viz obr.3.3.1, kde je naznačeno i jeho vnitřní zapojení.



Ilustrace 3.3.1: Katalogové zapojení snižujícího měniče s IO MC34063A

Integrovaný obvod obsahuje tyto hlavní části: tepelně kompenzovanou napěťovou referenci, komparátor, oscilátor s proměnnou střídou, obvod pro snímání a omezení proudu a výkonový spínací tranzistor. Z vnějšku je třeba připojit snímací rezistor Rsc, na němž se snímá úbytek napětí v závislosti na odebíraném proudu a při překročení určité hranice řídicí obvod nedovolí další zvyšování proudu, dále vstupní a výstupní vyhlazovací kondenzátory, časovací kondenzátor pro nastavení základní frekvence oscilátoru, akumulační tlumivku a rychlou (schottkyho) diodu, volitelně výstupní LC filtr pro snížení zvlnění a odporový dělič pro nastavení výstupního napětí, které je dáno dáno vztahem 3.3.1

Vztah 3.3.1:

$$U_{výst} = 1{,}25 \cdot (1 + R2 / R1)$$

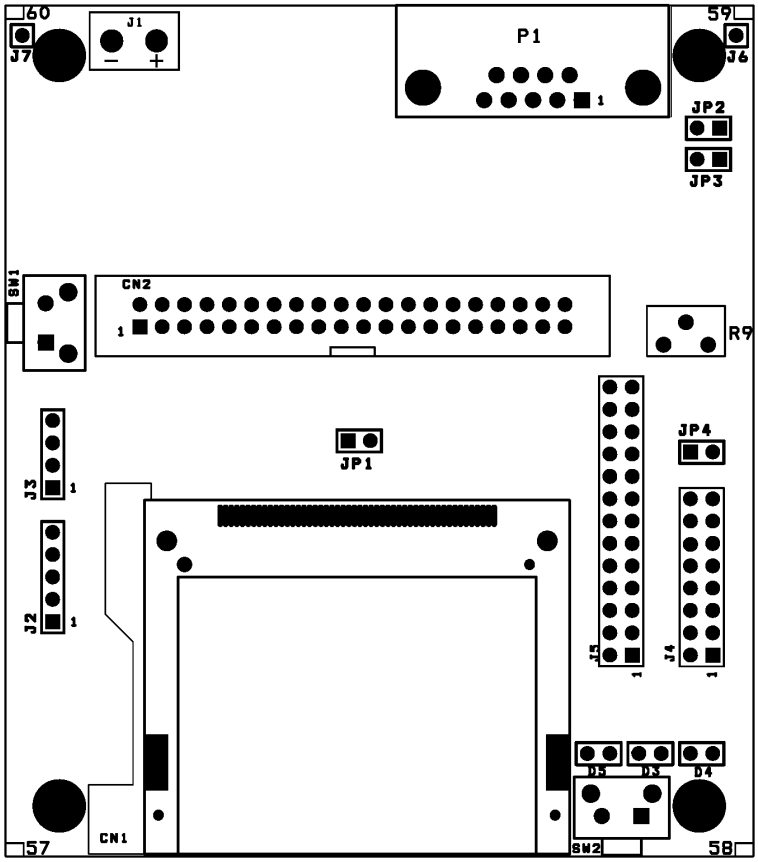
kde:

$U_{výst}$ je výstupní napětí [V]
R1, R2 jsou hodnoty vnějších odporů [Ω]

Na vstup zdroje jsem ještě přidal jednu diodu v sérii jako ochranu proti přepólování a transil pro zachycení případných napěťových špiček. Na výstupu zdroje (za přídavným filtrem) jsem osciloskopem naměřil zvlnění přibližně 70 mVpp, což je dostatečně málo pro funkci použitých digitálních obvodů. Celé zapojení zdroje je součástí hlavního schématu dataloggeru v příloze A.

3.4 ROZHRANÍ

Na obrázku 3.4.1 je patrné rozložení konektorů použitých rozhraní, indikačních a ovládacích prvků při pohledu na DPS shora. Následuje stručný popis a zapojení pinů včetně přiřazení k I/O portům procesoru ATmega128:



Ilustrace 3.4.1: Rozvržení konektorů na desce plošného spoje (z vrchu)

J1 – 2-pólová svorkovnice pro připojení napájecího napětí 12V až 24 V stejnosměrných.

J6, J7 – měřicí špičky spojené se zemí (pro uzemnění sondy osciloskopu).

J2 – programovací rozhraní **SPI**, které lze přes jednoduchou redukci (Brian Dean's Programmer – BSD, [22]), viz tab. 3.4.1 připojit k paralelnímu portu PC (D-SUB25). Pro tento způsob programování se pin /PEN procesoru nechává nezapojený (má vnitřní pull-up rezistor).

Tabulka 3.4.1: Zapojení SPI konektoru a jeho propojení s LPT

pin SPI	pin ATmega	popis	pin LPT	popis
1	(PDI) PE.0	MOSI (data-vstup)	9	D7
2	(PDO) PE.1	MISO (data-výstup)	10	/ACK
3	(SCK) PB.1	SCK (hodiny-vstup)	8	D6
4	/RESET	RESET (AVR reset)	7	D5
5	GND	GND	18	GND

J3 – sériová sběrnice **I2C** pro případné budoucí rozšíření dataloggeru. Toto rozhraní má např. řada inteligentních integrovaných senzorů. Zapojení viz tab. 3.4.2.

Tabulka 3.4.2: Zapojení I2C konektoru

pin	pin ATmega	popis
1	Vcc	+5 V (napájení)
2	(SCL) PD.0	SCL (hodiny-výstup)
3	(SDA) PD.1	SDA (data-obousměrná)
4	GND	GND

J4 – port pro připojení alfanumerického LCD displeje s řadičem HD44780 nebo kompatabilním, zapojení v tab. 3.4.3. Tyto displeje jsou běžně k dostání ve velikosti 1 x 8 až 4 x 20 znaků. Komunikují paralelně pomocí 8 (4) datových a 3 (2) řídicích linek, zde použito potřebné minimum linek. Bližší informace o těchto displejích a jejich řízení v [23]. Software dataloggeru zahrnuje knihovnu pro ovládání těchto displejů, lze je využít např. pro výpis ladicích informací a pod.

Tabulka 3.4.3: Zapojení LCD konektoru

pin	pin ATmega	popis (piny LCD)
1	GND	GND
2	Vcc	+5 V (napájení)
3	-	Vee (řízení kontrastu)
4	PD.5	RS (výběr data/příkaz)
5	GND	R/W (vybrán trvale zápis)
6	PD.6	E (Enable, zápisový puls)
7	GND	D0 (nevyužito, jen 4-bit)

pin	pin ATmega	popis (piny LCD)
8	GND	D1 (nevyužito, jen 4-bit)
9	GND	D2 (nevyužito, jen 4-bit)
10	GND	D3 (nevyužito, jen 4-bit)
11	PF.0	D4 (data/příkazy)
12	PF.1	D5 (data/příkazy)
13	PF.2	D6 (data/příkazy)
14	PF.3	D7 (data/příkazy)
15	Vcc	+5 V (napájení podsvětlení)
16	GND	GND

JP4 – propojka umožňující vypnout/zapnout podsvětlení LCD (je-li osazena, podsvětlení je zapnuto)

R9 – odporový trimr pro nastavení kontrastu LCD. Některé LCD vyžadují záporné napětí Vee, takové zde nelze bez úpravy použít.

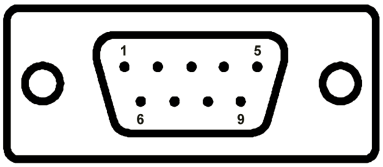
J5 – univerzální rozšiřující port. Na konektor se zapojením v tab. 3.4.4 jsem vyvedl všechny nevyužité porty, které by mohli posloužit pro budoucí rozšíření dataloggeru. Podrobnější popis významu pinů v [18] str. 68.

Tabulka 3.4.4: Zapojení konektoru univerzálního portu

pin	pin ATmega	popis
1	Vcc	+5 V (napájení)
2	GND	GND
3	(TxD1) PD.3	výstup USARTu 1
4	(RxD1) PD.2	vstup USARTu 1
5	(T1) PD.6	vstup časovače 1, I/O
6	(XCK1) PD.5	vstup/výstup hodin USARTu 1, I/O
7	(OC3A/AIN1) PE.3	výstup PWM, -vstup komparátoru,I/O
8	(XCL0/AIN0) PE.2	hodiny USARTu 0, +vstup komp., I/O
9	(OC3C/INT5) PE.5	výstup PWM, přerušení 5, I/O
10	(OC3B/INT4) PE.4	výstup PWM, přerušení 4, I/O
11	(ICP3/INT7) PE.7	IC vstup časovače 3, přerušení 7, I/O
12	(T3/INT6) PE.6	vstup časovače 3, přerušení 6, I/O
13	(ADC0) PF.0	vstup AD převodníku 0, I/O

pin	pin ATmega	popis
14	(TCK/ADC4) PF.4	JTAG, vstup AD převodníku 4, I/O
15	(ADC2) PF.2	vstup AD převodníku 2, I/O
16	(ADC1) PF.1	vstup AD převodníku 1, I/O
17	(TMS/ADC5) PF.5	JTAG, vstup AD převodníku 5, I/O
18	(ADC3) PF.3	vstup AD převodníku 3, I/O
19	(TDI/ADC7) PF.7	JTAG, vstup AD převodníku 7, I/O
20	(TDO/ADC6) PF.6	JTAG, vstup AD převodníku 6, I/O
21	(/RD) PG.1	řízení čtení externí paměti, I/O
22	(/WR) PG.0	řízení zápisu externí paměti, I/O
23	(TOSC2) PG.3	vstup krystalu RT časovače, I/O
24	(ALE) PG.2	řízení adresace externí paměti, I/O
25	GND	GND
26	(TOSC1) PG.4	vstup krystalu RT časovače, I/O

P1 – rozhraní RS232, zapojení konektoru D-SUB9 (obr. 3.4.2) odpovídá zapojení na PC vyjma chybějících hardwarových handshake signálů, viz tab. 3.4.5.



Ilustrace 3.4.2: RS232 - konektor D-SUB9

Tabulka 3.4.5: Zapojení RS232 (D-SUB9) konektoru

pin	pin ATmega	popis
2	(RxD1) PD.2 přes MAX232	RxD (přijímaná dat)
3	(TxD1) PD.3 přes MAX232	TxD (vysílaná data)
5	GND	GND

JP2, JP3 – tyto propojky slouží k odpojení USARTu ATmega128 od převodníku MAX232 (jsou-li propojky zasunuty, je MAX232 připojen). To může být užitečné v případě připojení budiče jiného rozhraní, např. RS485 apod. přes univerzální port.

JP1 – nastavuje režim vložené Compact Flash karty MASTER/SLAVE. Je-li propojka

zasunuta, pracuje CF v režimu MASTER.

CN1 – konektor pro připojení paměťové karty Compact Flash typu I nebo II. Konektor je vybaven vyhazovačem pro snadnější vyjmutí karty.

CN2 – standardní 40-pinový IDE konektor pro připojení pevného disku. Současně lze připojit 1 pevný disk a 1 CF kartu nebo 2 pevné disky a žádnou CF kartu. Jeden z disků musí být nastaven v režimu MASTER a druhý v režimu SLAVE. U CF karty k tomuto nastavení slouží propojka **JP1**. V tab. 3.4.6 je uvedeno pouze přiřazení potřebných linek ATA rozhraní k I/O portům procesoru ATmega128, ostatní linky jsou zapojeny podle schématu v příloze A. U prototypu s ATmega32 jsem využíval pouze datové linky D0-D7.

Tabulka 3.4.6: Připojení CF/IDE k procesoru ATmega128

linky ATA	pin ATmega	popis
D0-D7	PA.0-PA.7	dolní osmice datových linek
D8-D15	PC.0-PC.7	horní osmice datových linek
A0-A2	PB.2-PB4	linky pro adresaci vnitřních registrů
/RESET	PB.0	signál pro inicializaci zařízení
/IOWR	PB.5	signál zápisového pulsu
/IORD	PB.6	signál čtecího pulsu
IORDY	PB.6	signál prodloužení čtecího/zápis. pulsu

SW1 – RESET procesoru ATmega128, inicializuje datalogger do výchozího stavu.

SW2 – slouží pro ukončení diskových operací před vyjmutím karty, nastaví datové linky ATA jako vstupy, /IOWR a /IORD do klidového stavu. Tlačítko je připojeno k portu PD.4.

D5 – modrá LED, signalizuje ukončení diskových operací a že je možno CF kartu vyjmout. Je připojena na port PD.7.

D3 – červená LED, signalizuje přístup na CF kartu. Je připojena na ATA linku /DASP.

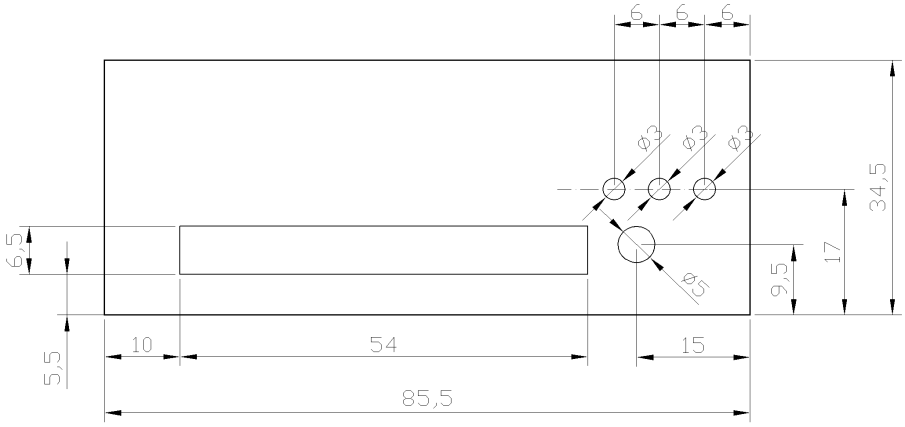
D4 – zelená LED, indikuje stav napájení dataloggeru. Je připojená na Vcc.

3.5 MECHANICKÉ PROVEDENÍ

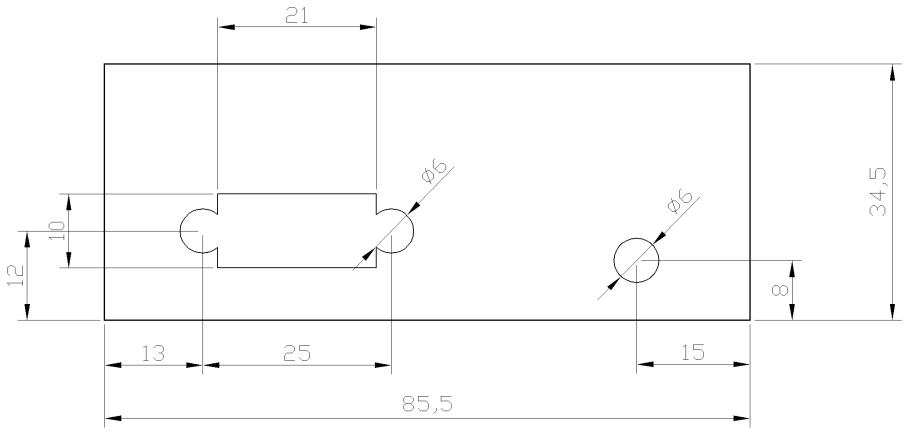
Obvodové zapojení je realizováno na oboustranné desce plošného spoje v třídě přesnosti 5 s prokovy pro použití součástek SMD a THD. Návrh plošného spoje jsem provedl v programu OrCAD 10.3. Soubory s návrhem jsou součástí přílohy na CD-ROM. Všechny SMD součástky jsou osazeny ze spodní strany s výjimkou CF konektoru a všechny THD součástky jsou osazeny z vrchu. Osazení jsem provedl ručně mikropájkou a po očištění přepáleného

tavidla nalakoval pájené spoje ochranným lakem.

Rozměr DPS (84 x 96,5 mm) byl zvolen tak, aby ji bylo možno zamontovat do běžně dostupné plastové krabičky U-KP02. Krabička se skládá ze shodného horního a dolního dílu, předního a zadního panelu a 4 nožiček. Z vnitřku krabičky je potřeba nejprve odstranit střední plastový sloupek a pak vyvrtat v rozích (v ose nízkých plastových distančních sloupků) 4 otvory průměru 3 mm a z vnější strany zahloubení pro zápustné hlavy šroubků. Potřebné otvory pro konektory a ovládací prvky jsem do čelního a zadního panelu vyvrtal a vyřezal lupenkovou pilkou a vypiloval pilníky podle nákresu 3.5.1 a 3.5.2.



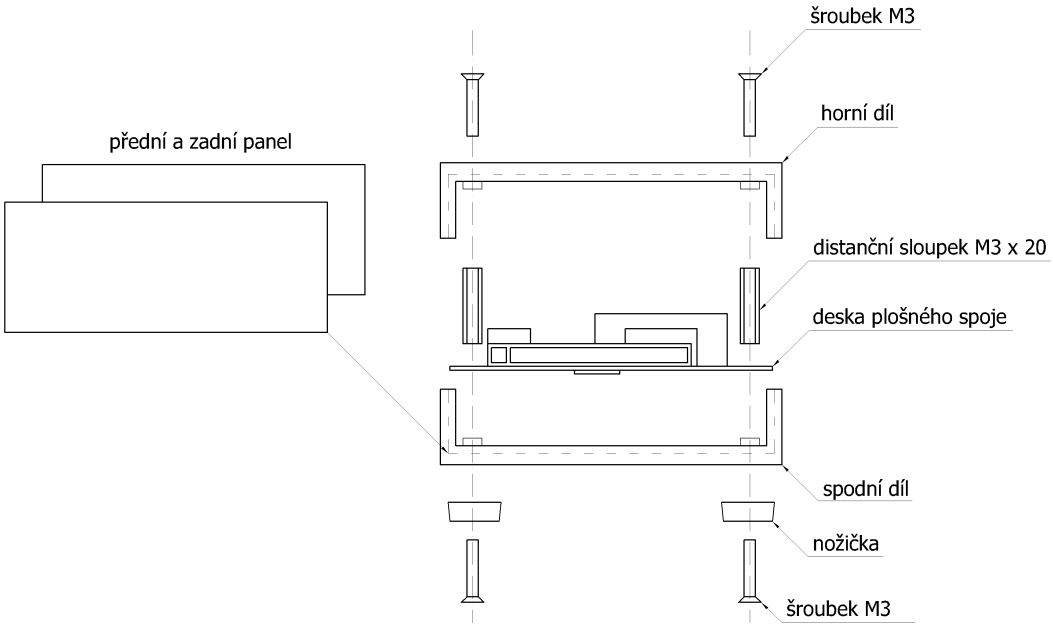
Ilustrace 3.5.1: Přední panel



Ilustrace 3.5.2: Zadní panel

DPS se pak s nasazenými panely vloží do spodního dílu krabičky. Čela se zasunou do příslušných drážek a DPS usedne na 4 rohové plastové distanční sloupky. Zespodu se dírami prostrčí šroubky s plastovými nožičkami a z horní strany se na ně našroubují kovové distanční sloupky výšky 20 mm. Pak se nasadí horní díl krabičky, tak aby čela zapadla do drážek.

Nakonec se dírami v horním dílu prostrčí 4 šroubky se zápustnou hlavou a zašroubují se do závitů distančních sloupků. Celý postup je zřejmý z nákresu 3.5.3.



Ilustrace 3.5.3: Montážní plánek

4 NÁVRH SOFTWARE

V této kapitole stručně popíšu použité vývojové nástroje, způsob překladu zdrojových kódů a programování procesoru, organizaci zdrojového kódu do souborů a na závěr hlavičky funkcí s popisem parametrů a návratových hodnot. všechny dále uvedené zdrojové kódy a nástroje jsou součástí přílohy na CD-ROM.

4.1 VÝVOJOVÉ NÁSTROJE

Dříve bylo běžné programování jednočipových mikropočítačů v jazyce symbolických adres (assembleru). Výrobci jednočipů obvykle dodávali i vlastní překladače (+disassemblery, debuggery, simulátory, atd.). Řada softwarových firem pak dělala univerzální překladače, jejichž syntaxe se od ostatních více či méně lišila, což mohlo působit problémy při kombinaci více různých zdrojových kódů.

Hlavní výhodou programování v assembleru je, že programátor má absolutní kontrolu nad výsledným kódem, kde není žádný balast a při dobré znalosti hardware může výsledný kód velmi efektivně optimalizovat na rychlost nebo velikost. Toto má význam především u nejlevnějších jednočipů s malou pamětí a nižším výkonem, kam by se třeba výsledný kód z překladačů vyšších jazyků vůbec nevešel.

Nevýhodou je pak to, že zdrojový kód v assembleru je vázán na konkrétní instrukční sadu daného procesoru nebo rodiny a nelze ho tak snadno portovat na jinou platformu. Assembler také klade vyšší nároky na programátora, který musí znát konkrétní instrukční sadu, musí umět daný algoritmus efektivně rozložit na elementární operace odpovídající používaným instrukcím a musí si sám spravovat veškeré zdroje jako paměť a registry.

4.1.1 Programovací jazyk C

S příchodem moderních rychlých jednočipů s dostatečnou pamětí se stále častěji používá překladačů vyšších programovacích jazyků, zejména C. Jazyk C vyvinuli začátkem 70-tých let Ken Thompson a Dennis Ritchie pro účely operačního systému UNIX – standard K&R a v roce 1989 vyšel standard ANSI C, jeho stručná charakteristika:

- univerzální programovací jazyk nízké úrovně,
- má úspornou syntaxi a je strukturovaný,
- má velký soubor operátorů a moderní datové struktury,
- velká efektivita kódu, rychlostí se téměř vyrovná programu v assembleru,
- není specializovaný na jednu oblast používání,
- je nezávislý na použité platformě.

Pro úvodní seznámení s C mi dobře posloužila kniha [24]. Dnes téměř pro každou platformu

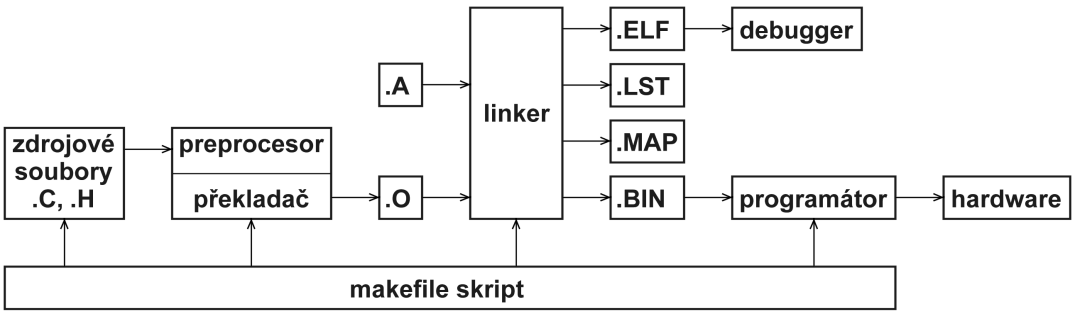
existuje alespoň jeden překladač C a ne jinak je tomu i na platformě AVR. Zde si můžeme vybrat z celé řady komerčních překladačů, např. CodeVisionAVR C Compiler, Imagecraft ICCAVR, IAR's C Compiler a další nebo volně dostupných open-source překladačů založených na linuxovém GCC (porty pro Linux, FreeBSD, DOS, Win32), kompletní výčet v [25]. Já jsem zvolil balík WinAVR (port avr-gcc pro Windows), který lze zdarma stáhnout zde [26].

4.1.2 WinAVR

Balík WinAVR obsahuje vše potřebné pro přeložení zdrojového kódu a naprogramování jednočipu:

- GCC – překladač ANSI C a C++,
- Binutils – nástroje pro tvorbu a práci s binárními soubory (linker, assembler, atd.),
- AVR-libC – knihovna standardních C funkcí pro AVR (detailní popis v [27]),
- AVRdude – programátor jednočipů AVR s širokou podporou hardware + GUI,
- GDB – debugger + GUI,
- AvarICE – podpora pro debugování přes JTAG kabel pro GDB,
- SimulAVR – podpora simulace pro GDB,
- Programmer's Notepad – textový editor,
- PDF/HTML dokumentace a další.

Překlad probíhá podle schématu na obr. 4.1.2.1. Zdrojové soubory *.C* jsou nejprve zpracovány preprocesorem, který načte příslušné hlavičkové soubory a expanduje makra. Překladač je pak přeloží do relativního (tzv. objektového) kódu (strojový kód s relativními adresami identifikátorů) a předá linkeru (sestavovací program). Ten načte další potřebný objektový kód knihovních funkcí a sestaví výsledný binární soubor s už absolutními adresami. Program v binárním nebo hexadecimálním souboru můžeme načíst v programátoru a přes příslušný hardware (v mém případě jednoduchý SPI kabel, viz tab. 3.4.1) nahrát do paměti jednočipu. Vedlejším produktem linkeru jsou textové výpisy *.LST* a *.MAP*.

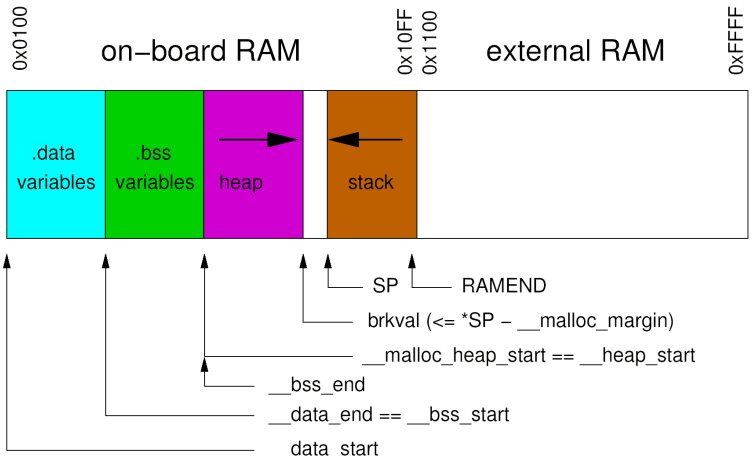


Ilustrace 4.1.2.1: Proces překladu zdrojových kódů

V *.LST* souboru je zdrojový kód prokládaný řádky v assembleru tak jak jej překladač přeložil. Lze tak snadno dohledat konkrétní (i knihovní) funkci a zjistit její assemblerovský zápis. V *.MAP* souboru je pak výpis přiřazení paměťových adres identifikátorům. Linker ještě podporuje binární formát ELF, který lze načíst do debuggeru GDB nebo AVR Studia. Celý proces překladu i programování lze zautomatizovat pomocí textového skriptu makefile pro program make, který zajistí volání všech potřebných programů podle skriptu.

4.1.3 Organizace datové paměti SRAM

Překladač provádí za programátora správu registrů a paměti. Programátor se tedy nemusí starat o to, kam překladač danou proměnnou umístí. V případě potřeby si může skutečnou adresu proměnné zjistit operátorem reference (&). Vzhledem k malé interní kapacitě datové paměti SRAM, je ale dobré mít alespoň hrubou představu o její organizaci, viz obr. 4.1.3.1.



Ilustrace 4.1.3.1: Organizace datové paměti SRAM

Oblast **.data** je vyhrazena pro inicializované proměnné (jejichž hodnota je známa už během překladu) a oblast **.bss** pro neinicializované proměnné. Dynamické proměnné alokované funkcí ***malloc()*** jsou umisťovány na haldu (**heap**) a rostou směrem nahoru. Návratové adresy při volání funkcí, jejich parametry a lokální proměnné třídy auto se ukládají na zásobník (**stack**), který roste směrem dolů.

Pokud dojde ke kolizi haldy a zásobníku nastane pravděpodobně zhroucení programu nebo alespoň poškození dynamických případně i statických dat. Překladač tento problém nehlásí, protože během překladu nelze určit, kolik paměti ze zásobníku se použije třeba při rekurzivním volání funkce. Bohužel nehlásí ani případ, kdy se statické proměnné nevejdou do RAM. AVR-libC neposkytuje žádnou funkci na zjištění volné RAM. Z výpisu v *.MAP* souboru lze alespoň přečíst hodnotu **__bss_end** a zjistit tak zbývající volné místo pro dynamickou paměť a zásobník. Za běhu programu můžeme číst proměnnou **__malloc_heap_start,** která je

implicitně nastavena na hodnotu **__bss_end** a obsah registru **SP** (vrchol zásobníku) a z toho získat velikost volné paměti pro dynamická data. Je však třeba počítat s tím, že se hodnota **SP** může ještě snížit např. při volání obsluhy přerušení.

4.1.4 Využití programové paměti pro konstanty

Během psaní programu se mi stalo, že řetězcové konstanty zaplnily oblast **.data** tak, že došlo ke kolizi dat se zásobníkem a program při volání určitých funkcí zhavaroval. Některé procesory z rodiny AVR (např. ATmega) přitom mají instrukce pro čtení dat z programové paměti (LPM nebo ELPM), které využívají některé funkce knihovny AVR-libC definované v hlavičkovém souboru *PGMSPACE.H*.

Má-li být proměnná uložena v programové paměti místo oblasti **.data** v RAM, musí být označena atributem **PROGMEM**. Pro celočíselné datové typy jsou k dispozici předdefinované datové typy s prefixem **prog_**, např **prog_char**, **prog_int16_t**, atd. V případě, že používáme řetězec přímo v argumentu funkce, lze použít přetypování pomocí makra **PSTR()**. Základními funkcemi pro přístup do programové paměti jsou ***pgm_read_byte()***, ***pgm_read_word()*** a ***pgm_read_dword()***. K dispozici jsou dále ekvivalenty běžných řetězcových funkcí s postfixem **_P**, např. ***memcpy_P()***, ***sprintf_P()***, atd. Kompletní výčet funkcí viz [27], kapitola 5.5. Následující příklad zdrojového kódu v C ukazuje použití výše zmíněných funkcí:

```
#include <avr/pgmspace.h>        // vložení hlavičkového souboru funkcí

char PROGMEM *s1="abcde";        // konstantní řetězec v prog. paměti
prog_char s2[]="fghi12";         // konstantní řetězec v prog. paměti
char s3[10];                     // neinicializovaný řetězec v RAM (.bss)
int a=13;                        // inicializovaná proměnná v RAM (.data)
char b;                          // neinicializovaná proměnná v RAM (.bss)

int main(void)                   // hlavní funkce
{
  b=pgm_read_byte(s1+2);         // přečte třetí znak z řetězce s1
  memcpy_P(s3,s2,4);             // zkopíruje 4 znaky z řetězce s2 do s3

/* podle formátovacího řetětězce v programové paměti vytisnke tento
řetězec a hodnoty proměnných a, b do řetězce s3 v RAM */
  sprintf_P(s3,PSTR("Var a = %d, b = %c\n"),a,b);

  return 0;
}
```

4.1.5 Zápis a čtení I/O portů

Díky tomu, že řídicí registry periferií včetně I/O portů jsou mapované do vnitřní SRAM (adresy 0x20 – 0x60 nebo 0x100 podle typu), umožňuje avr-gcc pracovat s registrem jako se standardní proměnnou umístěnou v RAM. Na začátek programu je jenom třeba vložit hlavičkový soubor *IO.H*, který se odkazuje na další hlavičkový soubor podle konkrétního typu procesoru, v němž jsou definice registrů. Pak už lze s registry pracovat zcela standardním

způsobem, viz následující příklad:

```
#include <avr/io.h>              // vložení hlavičkového souboru def. reg.

// definice maker pro nastavení/vynulování/přečtení jednoho bitu
#define SETB(data,bitnum) (data|=(1<<bitnum))  // nastav jeden bit
#define CLRB(data,bitnum) (data&=~(1<<bitnum)) // vynuluj jeden bit
#define FLIPB(data,bitnum) (data^=(1<<bitnum)) // invertuj jeden bit
#define GETB(data,bitnum) ((data>>bitnum)&1)   // přečti jeden bit

char a;                          // neinicializovaná proměnná v RAM (.bss)

int main(void)                   // hlavní funkce
{
  SETB(PORTA,7);                 // nastav bit č. 7 portu A do 1
  DDRA=0xFF;                     // nastav port A jako výstupní (všech 8l)
  FLIPB(PORTA,7);                // invertuj bit č. 7 portu A (z 1 do 0)
  return 0;
}
```

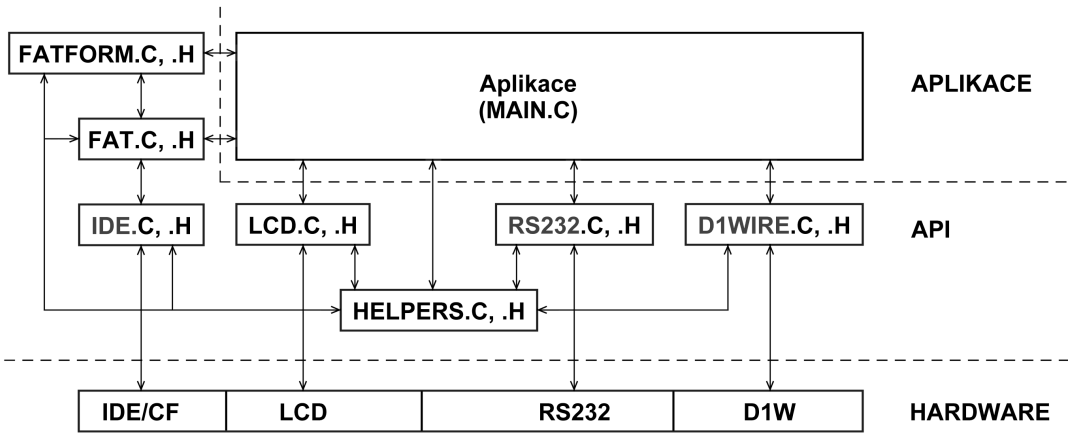
4.1.6 Obsluha přerušení

Nejprve je třeba definovat funkci pro obsluhu přerušení a do tabulky vektorů přerušení na příslušnou pozici zapsat pointer na onu funkci. To lze provést např. pomocí makra **SIGNAL(**jméno signálu**)** místo klasické hlavičky funkce, které je definované v hlavičkovém souboru *INTERRUPT.H* (dříve *SIGNAL.H*). Jméno signálu přerušení je definováno v tomtéž souboru. V hlavním programu je pak třeba globálně povolit přerušení makrem **sei()** a také v příslušném řídicím registru periferie povolit konkrétní přerušení. Pokud funkce obsluhy přerušení pracuje s nějakou globální proměnnou, je nutné aby tato proměnná byla označena atributem **volatile**. Ukázka definice obslužné funkce je v následujícím příkladu:

```
#include <avr/interrupt.h>       // vložení hlavičkového souboru def. int.
#include <avr/io.h>              // vložení hlavičkového souboru def. reg.

volatile char a;                 // neinicializovaná proměnná v RAM (.bss)

// definice funkce obsluhy přerušení při prijetí znaku po UARTu
SIGNAL(SIG_UART_RECV)
{                                // tělo funkce obsluhy přerušení
  if (a==0)
    return;
  a=UDR;                         // přečti přijatý znak
}

int main(void)                   // hlavní funkce
{
  // zapni přijímač a vysílač, povol přerušení od přijímače (UART CReg.B)
  UCSRB=(1<<RXEN)|(1<<TXEN)|(1<<RXCIE);
  sei();                         // povol globálně přerušení
  return 0;
}
```

4.2 ORGANIZACE ZDROJOVÉHO KÓDU

Zdrojové kódy software dataloggeru jsou rozčleněny do několika knihoven - souborů podle své logické funkce, viz obr. 4.2.1. Dohromady tvoří aplikační programovací rozhraní, které odděluje aplikaci od samotného hardware. Každá knihovna se skládá z *.C* souboru, kde jsou definice jednotlivých funkcí a globálních proměnných a *.H* souboru, kde jsou přiřazení I/O portů pro dané rozhraní, definice maker, datových typů a hlavičky funkcí.



Ilustrace 4.2.1: Organizace zdrojového kódu

V knihovně *HELPERS* jsou obecné datové typy, makra a funkce používané všemi dalšími knihovnami. Pro obsluhu pevného disku IDE a CF karty slouží knihovna *IDE*, která obsahuje low-level funkce na úrovni IDE registrů, ATA příkazů a sektorů. Funkce na úrovni souborového systému (FAT16) pak poskytuje knihovna *FAT* a její rozšíření *FATFORM*. Pro obsluhu LCD displeje s řadičem HD44780 nebo kompatabilním slouží knihovna *LCD*. Jednoduchou komunikaci po RS232 umožňují funkce knihovny *RS232*. Nakonec je zde experimentální knihovna pro obsluhu sériové sběrnice Dallas One-Wire, kterou používají některé periferní obvody a senzory firmy Maxim/Dallas. Podrobnější popis knihoven a jejich funkcí je uveden v podkapitole 4.3.

4.2.1 Makefile skript

Pro snadnou kompilaci zdrojových kódů jsem vytvořil soubor *MAKEFILE* pro program **MAKE**. Na příkazovém řádku v adresáři se zdrojovými soubory stačí zavolat program **make** (bez parametrů) a ten pak provede překlad podle skriptu *MAKEFILE* (je třeba mít předem nastavenou cestu k souborům překladače WinAVR např. dávkou *SETAVR.BAT* nebo v systémovém souboru autoexec.bat). Výsledkem úspěšného překladu jsou soubory relativního objektového kódu *.O*, výpisy *.LST* a *.MAP*, binární soubor *.ELF* pro debugger, binární soubor *.BIN* a hexadecimální soubor *.HEX* pro programátor AVRdude.

Na začátku souboru *MAKEFILE* je několik důležitých konfiguračních řádků:

```
PROGNAME        = MAIN
OBJS            = helpers.o d1wire.o lcd.o rs232.o ide.o fat.o fatform.o...
MCU_TARGET      = atmega128 # atmega32
OPTIMIZE        = -O2 # -Os for size, -O2 for speed
```

První řádek s proměnnou **PROGNAME** říká, jak se bude jmenovat výsledný přeložený soubor. O tom, které soubory relativního objektového kódu budou do výsledného programu zahrnuty, rozhoduje proměnná **OBJS**. Typ jednočipu, pro který program překládáme, je nutné nastavit proměnnou **MCU_TARGET**. Optimalizaci výsledného kódu lze široce ovlivnit parametry překladače **gcc** v proměnné **OPTIMIZE**, popis parametrů např. v [28].

Další funkce skriptu se zobrazí po zadání příkazu **make help** na příkazové řádce. Stručný seznam dalších příkazů viz tab. 4.2.1.1.

Tabulka 4.2.1.1: Parametry makefile skriptu

make flash – nahraje binární soubor *.BIN* do programové paměti jednočipu,
make dump – přečte obsah programové paměti jednočipu do *.BIN* souboru,
make efuse – naprogramuje pojistku Extended Fuse ze souboru *FUSEEXT.BIN*,
make lfuse – naprogramuje pojistku Low Fuse ze souboru *FUSELOW.BIN*,
make hfuse – naprogramuje pojistku High Fuse ze souboru *FUSEHIGH.BIN*,
make fuser – přečte současnou hodnotu pojistek do výše zmíněných souborů,
make clean – vymaže všechny soubory vzniklé při překladu.

4.3 POPIS KNIHOVEN A HLAVIČEK FUNKCÍ

V této podkapitole popíšu jednotlivé knihovny a hlavičky funkcí (tučná kurzíva), makra (tučně), příp. globální proměnné (tučně) v nich obsažené. Další podrobnosti lze vyčíst z okomentovaných zdrojových souborů v příloze na CD-ROM.

4.3.1 Knihovna *HELPERS*

Tato knihovna obsahuje obecné definice datových typů, konstanty, makra a funkce používané v dalších knihovnách i hlavním programu. Pro správnou funkci programu je důležité nastavit konstantu **F_CPU** na skutečný kmitočet oscilátoru procesoru:

```
#define F_CPU 16000000UL        // kmitočet oscilátoru CPU [Hz].
```

Ve všech dalších knihovnách se používají následující datové typy u nichž je nezbytné zachovat odpovídající velikost:

```
#define Byte uint8_t            // 8-bitů, neznaménkový
#define Word uint16_t           // 16-bitů, neznaménkový
#define DWord uint32_t          // 32-bitů, neznaménkový
#define QWord uint64_t          // 64-bitů, neznaménkový
```

Dále jsou zde definována makra pro bitové operace:

```
SETB(data,bitnum)                // nastav jeden bit
CLRB(data,bitnum)                // vynuluj jeden bit
FLIPB(data,bitnum)               // invertuj jeden bit
GETB(data,bitnum)                // přečti jeden bit
```

a makra a funkce pro časové prodlevy:

```
delay_ns(t)          // limit: 762000ns/F_CPU_MHz, 0=min.: 3/F_CPU_MHz us
delay_us(t)          // limit: 768us/F_CPU_MHz, 0=maximální prodleva
delay_us2(t)         // limit: 262144us/F_CPU_MHz, 0=maximální prodleva
delay_ms(t)          // limit: 262ms/F_CPU_MHz, 0=maximální prodleva
delay_ms2(Word t);   // limit: 65535ms, 0=žádná prodleva
```

a makro na odhad volné paměti pro dynamické proměnné:

```
mem_avail()                      // SP-__bss_end.
```

4.3.2 Knihovna *IDE*

Tato knihovna obsahuje nízkoúrovňové funkce pro čtení/zápis vnitřních registrů a datového bufferu IDE zařízení, posílání ATA příkazů, definice konstant ATA příkazů a další. Pro správnou funkci je třeba na začátku hlavičkového souboru přiřadit I/O porty jednočipu k daným linkám ATA rozhraní. Pokud není definován I/O port pro datové linky D8-D15, použije se automaticky pouze 8-bitový přenos. Další konfigurační konstanty:

```
#define IDE_RW_PULSE_DELAY 180   // délka IORD# a IOWR# pulsu [ns]
#define IDE_RETRY_COUNT  65535   // počet opakování čtení/zápisu při chybě
#define IDE_SECTOR_SIZE    512   // velikost sektoru [B].
```

Globální proměnné:

```
Byte ide_sector_buffer[IDE_SECTOR_SIZE]; // buffer pro čtení/zápis dat
Word ide_device_c;           // počet cylindrů zařízení, c: [0..C-1]
Word ide_device_h;           // počet hlav zařízení, h: [0..H-1]
Word ide_device_s;           // počet sektorů na stopu zaříz., s: [1..S]
DWord ide_device_lba;       // počet LBA sektorů zařízení, lba: [0..LBA-1]
DWord ide_device_current_lba; // LBA pávě čteného/zapisovaného sektoru
Byte ide_device_rscache;     // pokud >0,tak při opakovaném čtení stejného
// sektoru skončí, předpokládá se že data jsou už načtena v bufferu
Byte ide_device_pio;         // nejrychlejší podporovaný PIO režim [0-4].
```

Dále uvedu některé hlavičky funkcí (nebudu uvádět funkce sloužící převážně pro interní potřebu).

```
void ide_init(void);
```

Nastaví I/O porty IDE rozhraní do výchozího stavu, pošle zařízení /RESET puls.

```
Word ide_read_cmdreg(Byte reg);
```

Přečte požadovaný vnitřní registr zařízení, argument je adresa registru (viz definice v *IDE.H*) a vrací 16-bitovou hodnotu (pokud se komunikuje pouze po 8-bitech, platí jen dolních 8 bitů).

```
void ide_write_cmdreg(Byte reg, Word data);
```

Zapíše požadovanou hodnotu do vnitřního registru zařízení.

```
Byte ide_exec_cmd(Byte drive, Byte cmd, Byte feat, Byte scnt, DWord lba);
```

Pošle zařízení daný ATA příkaz s parametry: číslo zařízení (konstanta IDE_DRVSEL_MA nebo IDE_DRVSEL_SL pro MASTER, resp. SLAVE), features, sector_count, LBA adresa. Význam parametrů odpovídá vnitřním registrům a je specifický podle konkrétního ATA příkazu. Návratové hodnoty: 0 – příkaz proběhl v pořádku, bit 0 = 1 – po provedení příkazu byl nastaven flag ERR, bit 1 = 1 – vypršel timeout při čekání na flag bussy před provedením příkazu, bit 2 = 1 – vypršel timeout při čekání na flag ready před provedením příkazu, bit 3 = 1 – vypršel timeout při čekání na flag bussy po provedení příkazu.

```
void ide_read_sector_buffer(Byte *p_buffer);
```

Přečte data z vnitřního bufferu zařízení do daného bufferu, argument je pointer na 512B buffer.

```
void ide_write_sector_buffer(Byte *p_buffer);
```

Zapíše data z daného bufferu do vnitřního bufferu zařízení, argument je pointer na 512B buffer.

```
Byte ide_init_device(Byte drive,Byte *p_buffer,IDE_DEVICE_STRINGS *p_dss);
```

Detekuje, diagnostikuje a inicializuje dané zařízení, provádí postupně ATA příkazy: EXECUTE DEVICE DIAGNOSTIC, SET FEATURES (jen při nastavení 8-bitového přenosu), IDENTIFY DEVICE a INITIALIZE DEVICE PARAMETERS. Argumenty jsou číslo zařízení, pointer na 512B buffer pro načtení identifikačních dat a pointer na strukturu IDE_DEVICE_STRINGS, kam se uloží textové řetězce se sériovým číslem, názvem a verzí firmware zařízení. Pokud tyto údaje nepotřebujeme, zadáme nulový pointer NULL. Návratové hodnoty: 0 – vše proběhlo v pořádku, 1 – zařízení nepodporuje LBA (u CF karet je podpora LBA vždy zajištěna), 2 – chyba při identifikaci zařízení, 3 – chyba při nastavení 8-bitového režimu (u pevných disků), 4 – chybný výsledek diagnostiky, 5 – chyba při průběhu diagnostiky.

```
Byte ide_read_sector(Byte drive, DWord lba, Byte *p_buffer);
```

Přečte vybraný sektor do daného bufferu, argumenty jsou číslo zařízení, LBA adresa sektoru a pointer na 512B buffer pro uložení dat. Vzhledem k omezené kapacitě vnitřní SRAM jsem čtení a zápis omezil na 1 sektor. Návratová hodnota je 0, pokud vše proběhlo v pořádku, jinak je předán chybový kód funkce ***ide_exec_cmd()***.

```
Byte ide_write_sector(Byte drive, DWord lba, Byte *p_buffer);
```

Zapíše data z daného bufferu do zvoleného sektoru, argumenty jsou číslo zařízení, LBA adresa sektoru a pointer na 512B buffer s daty pro zápis. Návratové hodnoty jsou stejné jako u ***ide_read_sector().***

4.3.3 Knihovna *FAT*

Tato knihovna poskytuje funkce na úrovni souborového systému. Přímo nepřistupuje k žádnému hardware ale využívá funkce nižší vrstvy – knihovny *IDE*. Díky tomu je přenositelná např. na platformu PC. Většinu funkcí jsem nejprve překládal na PC pomocí 32-bitového překladače DJGPP/GCC pro DOS a po odladění portoval s naprosto minimálními úpravami do prostředí WinAVR.

Knihovna obsahuje řadu konstant, datových typů a struktur podle popisu v kapitole 2.4.

Datové struktury:

```
FAT_MBR_PAT_ENTRY          // struktura položky PArtition Tabulky (v MBR)
FAT_MBR                    // struktura MBR (Master Boot Record)
FAT_DBR                    // struktura DBR (DOS Boot Record)
FAT_DIR_ENTRY              // struktura adresářové položky
FAT_STAT                   // struktura statistiky FAT, viz fat_get_stat()
FAT_FILE                   // struktura souboru, viz fat_fopen().
```

Makra pro práci s pakovanou strukturou datumu a času:

```
fat_dir_get_time_h(ptime) // rozpakuj hodiny
fat_dir_get_time_m(ptime) // rozpakuj minuty
fat_dir_get_time_s(ptime) // rozpakuj sekundy
fat_dir_put_time(h,m,s)   // zapakuj hodiny, minuty a sekundy
fat_dir_put_times(utime)  // zapakuj hodiny, minuty a sekundy ze strukt.

fat_dir_get_date_d(pdate) // rozpakuj den
fat_dir_get_date_m(pdate) // rozpakuj měsíc
fat_dir_get_date_y(pdate) // rozpakuj rok
fat_dir_put_date(d,m,y)   // zapakuj den, měsíc, rok
fat_dir_put_dates(udate)  // zapakuj den, měsíc, rok ze struktury.
```

Další makra:

```
fat_lba_begin2             // vrací LBA začatku 2. FAT tabulky
fat_cluster2lba(cluster)   // vrací LBA 1. sektoru požadovaného clusteru
fat_find_dir_init(dir_base_cluster) // inicializace prohledávání adresáře,
// pokud chceme hledat od začítku, viz funkce fat_find_dir_entry()
fat_ftell(file)           // vrátí aktuální pozici v souboru.
```

Globální proměnné:

```
Byte fat_current_drive;   // číslo aktuálního zařízení (jako v knihovně
// IDE) na kterém se budou provádět veškeré operace, nastaví uživatel
Byte fat_type;            // typ souborového systému FAT aktivního oddílu
DWord fat_plba_begin;     // LBA začátku aktivního oddílu
DWord fat_plba_size;      // LBA velikost aktivního oddílu
char fat_oem_id[FAT_DBR_OEMIDL+1]; // ASCII řetězec s OEM ID z DBR
Byte fat_media_descriptor;// média deskriptor aktivního oddílu z DBR
Byte fat_number_of_fats;  // počet FAT tabulek na aktivním oddílu
Byte fat_sectors_per_cluster; // počet sektorů na cluster z DBR
Word fat_total_clusters;  // celkový počet dat. clusterů na akt. oddílu
Word fat_root_entries;    // počet položek v kořenovém adresáři
Word fat_sectors_per_fat; // počet sektorů na jednu FAT tabulku
DWord fat_lba_begin       // LBA začátku 1. FAT na aktivním oddílu
DWord fat_root_lba_begin; // LBA začátku kořenového adresáře
DWord fat_data_lba_begin; // LBA začátku dat
Word fat_current_dir;     // číslo 1.clusteru aktualního adresaře (root=0)
```

```
Word fat_current_dir_entry; // číslo aktualní položky adresáře
Word fat_current_cluster; // číslo aktualního clusteru ve FAT
```

Funkce:

```
Byte fat_read_mbr(DWord lba);
```

Přečte sektor z dané LBA zařízení **fat_current_drive** (tuto proměnnou nastaví uživatel, implicitně je nastavena na zařízení MASTER) do bufferu **ide_sector_buffer** a zpracuje ho jako MBR. Zkontroluje signaturu a určí aktivní oddlíl v PAT. Pokud žádný oddíl není označen jako aktivní, pokusí se najít první oddíl s FAT16. Zkontroluje typ oddílu a je-li FAT16 přečte z PAT LBA začátku oddílu a velikost a podle toho inicializuje globální proměnné **fat_type**, **fat_plba_begin**, **fat_plba_size**. Argument je LBA sektoru MBR (typicky 0). Návratové hodnoty: 0 – vše proběhlo v pořádku, 1 – neplatná LBA oddílu, 2 – nenalezen žádný známý souborový systém, 3 – neplatná signatura (0x55AA), 4 – chyba při čtení sektoru

```
Byte fat_read_dbr(DWord lba);
```

Přečte sektor z dané LBA do bufferu **ide_sector_buffer** a zpracuje ho jako DBR. Zkontroluje jeho signaturu, přečte OEM ID do proměnné **fat_oem_id,** zkontroluje velikost sektoru na oddílu (podpora jen 512B sektorů), přečte počet rezervovaných sektorů, vypočítá LBA začátku 1. FAT tabulky a uloží ji do proměnné **fat_lba_begin,** přečte počet FAT tabulek na oddílu a uloží ho do proměnné **fat_number_of_fats**, přečte počet položek kořenového adresáře a uloží ho do proměnné **fat_root_entries**, přečte deskriptor média (pro pozdější kontrolu signatury FAT) do proměnné **fat_media_descriptor**, přečte celkový počet sektorů oddílu a porovná ho s hodnotou v PAT (není-li větší než údaj v PAT) a přepíše proměnnou **fat_plba_size**, přečte počet sektorů na jednu FAT a uloží ho do proměnné **fat_sectors_per_fat**, vypočítá LBA začátku kořenového adresáře a LBA začátku vlastních dat. Argument je LBA sektoru DBR (typicky první sektor aktivního oddílu). Návratové hodnoty: 0 – vše proběhlo v pořádku, 1 – velikost oddílu v DBR je větší než je definovaná v PAT, 2 – neplatná LBA FAT, 3 – jiná velikost sektoru než 512 B, 4 – neplatná signatura (0x55AA), 5 – chyba při čtení sektoru.

```
Byte fat_read_dir_entry(Word dir_cluster, Word entrynum, FAT_DIR_ENTRY
*dire);
```

Přečte požadovanou adresářovou položku z adresáře na daném clusteru do struktury. Argumenty jsou číslo clusteru adresáře ve FAT, pořadové číslo adresářové položky a pointer na strukturu FAT_DIR_ENTRY. Vrátí 0 pokud vše proběhlo v pořádku, jinak předá chybový kód funkce ***ide_read_sector()***.

```
Byte fat_write_dir_entry(Word dir_cluster, Word entrynum, FAT_DIR_ENTRY
*dire);
```

Zapíše danou adresářovou položku ze struktury na daný cluster adresáře. Argumenty jsou stejné jako u předchozí funkce. Vrátí 0 pokud vše proběhlo v pořádku, jinak předá chybový

kód funkce ***ide_write_sector()***.

```
Word fat_read_cluster(DWord fat_lba_base, Word cluster);
```

Přečte požadovanou hodnotu položky FAT s danou bázovou LBA z daného čísla položky. Argumenty jsou LBA prvního sektoru FAT – báze (lze tedy číst z 1. nebo 2. FAT) a číslo položky FAT jejíž hodnotu chceme přečíst. Při chybě čtení vrátí neplatnou hodnotu položky FAT_CLUSTER_NA1.

```
Byte fat_write_cluster(DWord fat_lba_base1, DWord fat_lba_base2, Word
cluster, Word value);
```

Zapíše požadovanou hodnotu položky FAT do obou FAT tabulek s danými bázovými LBA. Argumenty jsou bázové LBA 1. a 2. FAT tabulky (pokud je na oddílu jen jedna FAT, tak bude druhý argument ignorován), číslo položky FAT kam se má zapisovat a hodnota položky. Vrátí 0 pokud vše proběhlo v pořádku, jinak předá chybový kód funkce ***ide_write_sector()***.

```
Word fat_find_cluster_entry(DWord fat_lba_base, Word fat_start, Word
cluster_type);
```

Prohledá FAT od zadané počáteční položky a vrátí číslo položky požadovaného typu. Používá se např. při alokaci volného místa ve FAT pro adresáře nebo soubory. Argumenty jsou bázová LBA FAT, číslo počáteční položky (≥2) od které probíhá hledání a požadovaná hodnota položky. Návratové hodnoty: 0 – došlo k chybě čtení sektoru FAT, 1 – položka dané hodnoty nenalezena, ≥2 – číslo položky s prvním výskytem požadované hodnoty.

```
Word fat_get_stat(DWord fat_lba_base, FAT_STAT *stat)
```

Projde celou FAT a spočítá statistiku kolik clusterů je volných, obsazených a vadných. Z počtu volných clusterů a a velikosti clusteru lze zjistit zbývající volné místo na oddílu. Argumenty jsou bázová LBA FAT a pointer na strukturu FAT_STAT. Vrátí 0 pokud vše proběhlo v pořádku, jinak vrátí relativní číslo sektoru (od bázové LBA FAT) +1 na kterém došlo k chybě čtení.

```
void fat_expand_filename(char *packed_name, char *unpacked_name);
```

Rozšíří zadaný řetězec názvu souboru nebo adresáře o 1-11 znacích na celých 11 znaků s případným zarovnáním mezerami a přidá ukončovací 0. Umožňuje zadat tzv. wildcards pro filtrování většího množství souborů, kde * zastupuje 1 a více libovolných znaků, ? zastupuje právě jeden libovolný znak a | na začátku se převede na znak 0xE5 – znak smazaného souboru. Argumenty jsou pointery na zdrojové pole znaků s názvem o proměnné délce a cílové pole znaků o pevné délce.

```
Byte fat_find_dir_entry(char *filename, Byte attribm, FAT_DIR_ENTRY
*dire);
```

Projde adresář na který ukazuje globální proměnná **fat_current_cluster** a pokusí se najít položku odpovídající zadanému názvu a atributové masce. Je-li položka nalezena, načte ji do struktury. Nastaví globální proměnné **fat_current_cluster** a **fat_current_dir_entry** tak, že ukazují buď na nalezenou položku nebo na první volné místo v adresáři. Před novým hledáním je obvykle třeba použít makro **fat_find_dir_init**, které nastaví globální proměnné tak, aby se hledalo od začátku zadaného adresáře, jinak se pokračuje z předchozí pozice. Argumenty jsou název adresářové položky (1-11 znaků včetně wildcards), atributová maska složená z příslušných konstant definovaných ve *FAT.H* a pointer na strukturu FAT_DIR_ENTRY, kam se případně nalezená položka načte. Návratové hodnoty: 0 – vše proběhlo v pořádku, bit 6 = 1 – konec procházení adresáře, bit 7 = 1 – konec alokovaného prostoru adresáře.

```
Byte fat_alloc_dir_entry(Word dir_base_cluster, FAT_DIR_ENTRY *temp_dire);
```

Projde zadaný adresář, pokusí se najít místo pro novou položku (buď volnou nebo smazanou) a případně alokuje nový cluster adresáře a připojí ho do řetězce. Pak nastaví globální proměnné **fat_current_cluster** a **fat_current_dir_entry** na tuto budoucí položku. V případě plného kořenového adresáře nelze alokovat další prostor, protože má fixní velikost. Argumenty jsou 1. cluster adresáře (0 pro kořenový adresář) a pointer na strukturu FAT_DIR_ENTRY pro dočasné použití během hledání. Návratové hodnoty: 0 – vše proběhlo v pořádku, 1 – chyba při inicializaci nově alokovaného clusteru, 2 – chyba připojení nově alokovaného clusteru na konec řetězce, 3 – chyba při ukončení řetězce (zápis EOF), 4 – chyba při alokaci clusteru, 5 – disk je plný, 6 – kořenový adresář je plný, 7 – chyba při čtení sektorů během prohledávání adresáře.

```
Byte fat_mkdir(Word dir_base_cluster, char *dir_name);
```

Vytvoří podadresář v daném adresáři. Argumenty jsou 1. cluster rodičovského adresáře a jméno podadresáře. Návratové hodnoty: 0 – vše proběhlo v pořádku, 1,2 – chyba při vytvoření položek . a .., 3 – chyba při inicializaci nově alokovaného clusteru, 4 – chyba při zápisu adresářové položky, 5 – chyba zápisu do FAT, 6 – disk je plný, 7 – položka daného jména již existuje, 8 – neplatné znaky ve jménu, ≥9 – chyba alokace adresářové položky.

```
Byte fat_chdir(char *dir_name); // navratove kody: 0=OK, 1=prazdne jmeno
```

Změní aktuální adresář podle názvu adresáře. Nelze přeskakovat přes více než jednu úroveň adresářů vyjma přechodu na kořenový adresář (např ***fat_chdir***("\adr1")). Symboly '\' nebo '/', '.', '..' značí kořenový adresář, aktuální adresář a nadřazený adresář. Návratové hodnoty: 0 – vše proběhlo v pořádku, 1 – prázdné jméno adresáře, 2 – adresář nenalezen, 3 – nalezená položka nemá atribut adresáře.

```
Byte fat_remove(Word dir_base_cluster, char *dir_name);
```

Z daného adresáře odstraní požadovanou položku (lze mazat jen prázdné podadresáře, jinak dojde ke vzniku lost clusters). Argumenty jsou 1. cluster adresáře a jméno adresářové položky. Návratové hodnoty: 0 – vše proběhlo v pořádku, 1 – chyba při zápisu do FAT, 2 – chyba zápisu adresářové položky, 3 – položka nenalezena, 4 – pokus o mazání aktuálního adresáře (nelze).

```
FAT_FILE *fat_fopen(Word dir_base_cluster, char *file_name, Byte mode);
```

Otevře soubor v daném adresáři; umožňuje několik režimů činnosti. Argumenty jsou 1. cluster rodičovského adresáře, jméno souboru (nesmí obsahovat wildcards ani jiné nepovolené znaky) a režim činnosti. V režimu čtení – '**R**' (read), je v rodičovském adresáři nalezena odpovídající položka a podle ní zinicializována struktura FAT_FILE, jejíž pointer funkce vrátí (pro strukturu se alokuje potřebné množství dynamické paměti). Pokud položka neexistuje, vrátí NULL. V režimu zápisu – '**W**' (write), se funkce nejprve pokusí najít existující položku. Pokud existuje, zavolá funkci ***fat_remove()***, která soubor odstraní a uvolněnou položku inicializuje výchozími hodnotami. Pokud položka neexistuje, pokusí se o alokaci nové. Dále inicializuje strukturu FAT_FILE a vrátí pointer na ní. V režimu připisování na konec souboru – '**A**' (append), se v případě neexistence stávajícího souboru postupuje jako v režimu '**W**', jinak se zpracují údaje ze stávající položky a ukazatel pozice v souboru se nastaví na konec.

```
void fat_fclose(FAT_FILE *file);
```

Uzavře soubor otevřený funkcí ***fat_fopen()*** a odalokuje paměť struktury FAT_FILE.

```
Byte fat_fseek(FAT_FILE *file, long offset, Byte mode);
```

Nastaví ukazatel pozice v souboru; umožňuje několik režimů činnosti. Argumenty jsou pointer na strukturu FAT_FILE, offset (kladné nebo záporné číslo) a režim činnosti: '**A**' (absolute) – offset je kladné číslo vztažené k začátku souboru, '**R**' (relative) – offset je kladné nebo záporné číslo vztažené k aktuální pozici v souboru a '**E**' (end) – offset je záporné číslo vztažené ke konci souboru. Návratové hodnoty: 0 – vše proběhlo v pořádku, 1 – chyba při čtení FAT, 2 – offset mimo rozsah souboru, 3 – byl předán NULLový pointer.

```
int fat_fgetc(FAT_FILE *file);
```

Přečte jeden byte z otevřeného souboru. Argumentem je pointer na strukturu FAT_FILE. Návratové hodnoty: ≥0 – vše proběhlo v pořádku (je vrácen platný znak), -1 – pokus o čtení za koncem souboru, -2 – chyba při čtení sektoru, -3 – neplatné číslo clusteru ve struktuře FAT_FILE, -4 – byl předán NULLový pointer.

```
Byte fat_fputc(Byte c, FAT_FILE *file);
```

Zapíše jeden byte na aktuální pozici v otevřeném souboru. Argumenty jsou pointer na strukturu FAT_FILE a zapisovaný byte. Návratové hodnoty: 0 – vše proběhlo v pořádku, 1 – chyba aktualizace adresářové položky souboru, 2 – chyba při zápisu sektoru, 3 – chyba při čtení

sektoru, 4 – chyba při zápisu do FAT, 5 – chyba při alokaci nového clusteru, 6 – chyba čtení adresářové položky souboru, 7 – byl předán NULLový pointer.

4.3.4 Knihovna *FATFORM*

Tato knihovna rozšiřuje knihovnu *FAT* o funkce pro formátování disku a vytvoření souborového systému FAT16. Používanou CF kartu nebo disk lze naformátovat na standardním PC. Avšak pod operačním systémem **Windows 9x** může být u menších karet problém s nemožností zvolit typ souborového systému, kde Windows 9x automaticky zvolí typ FAT12. Existuje sice nedokumentovaný přepínač /z programu

```
format drive: /z:sectorspercluster,
```

ale ten nelze použít na výměnné disky (CF kartu ve čtečce). Pod systémem **Windows 2000/XP** už lze ovlivnit typ souborového systému takto:

```
format drive: /fs:FAT /a:clustersize,
```

kde hodnota clustersize (v bytech) musí být taková, aby počet clusterů na disku překročil hranici 4096, pak bude zvolen systém FAT16.

Abych se vyhnul problémům s formátováním na různých systémech, implementoval jsem vlastní formátovací funkci:

```
Byte fat_format_drive(Byte drive, Byte form_level);
```

která vytvoří oblast MBR s jedním primárním oddílem typu FAT16, oblast DBR, FAT a kořenového adresáře. Volitelně může vynulovat celou nultou stopu nebo celý disk. Image MBR a DBR jsou uloženy v programové paměti a dohromady zabírají 1 kB. Argumenty jsou číslo zařízení a úroveň formátování: FAT_FORMAT_FAST – vytvoří jen potřebné struktury, FAT_FORMAT_WIPET0 – navíc vynuluje stopu 0, FAT_FORMAT_WIPEALL – navíc vynuluje celý disk. Návratové hodnoty: 0 – vše proběhlo v pořádku, 1 – chyba při formátování datových sektorů, 2 – chyba při inicializaci oblasti kořenového adresáře, 3 – chyba při inicializaci oblasti FAT, 4 – chyba při formátování zbývajících sektorů stopy 0, 5 – chyba při inicializaci DBR, 6 – oddíl je příliš veliký (>4 GB, funkce by však měla velikost oříznout na 4 GB – netestováno), 7 – oddíl je příliš malý pro FAT16 (<2,5 MB), 8 – chyba při inicializaci MBR, 9 – chyba při inicializaci zařízení.

4.3.5 Knihovna *RS232*

Tato knihovna obsahuje několik funkcí pro jednoduchou sériovou komunikaci po RS232. Na začátku hlavičkového souboru se nachází definice konstant parametrů přenosu:

```
#define UART_BAUD_RATE   9600 // nastavení bitové rychlosti [baud]
#define UART_PARITY_MODE 0    // nastavení parity [0=žádní,2=sudá,3=lichá]
#define UART_DATA_BITS   8    // nastavení počtu datových bitů [5,6,7,8]
#define UART_STOP_BITS   1    // nastavení počtu stop bitů [1,2]
#define UART_NEW_LINE_MODE 1  // sekvence pro odřádkování na terminálu:
// [0=UNIX(LF), 1=DOS/WIN(CR+LF), 2=Apple/MAC(CR)].
```

Funkce:

```
void rs232_init(void);
```

Inicializuje USART podle konfiguračních konstant – z bitové rychlosti UART_BAUD_RATE vypočte a nastaví registr **UBRR**, zapne přijímač a vysílač, povolí přerušení od přijímače a podle dalších konstant nastaví parametry rámce přenosu (registry **UCSRA**, **UCSRB**, **UCSRC**).

```
void rs232_putc(Byte data);
```

Pošle jeden znak po RS232:

```
void rs232_putnl(void);
```

Pošle specifickou sekvenci znaků pro ukončení řádky:

```
void rs232_print(char *str);
```

Pošle postupně znaky řetězce, argumentem je pointer na pole znaků ukončené nulou.

```
void rs232_print_P(PGM_P str);
```

Pošle postupně znaky řetězce uloženého v programové paměti, argumentem je pointer do programové paměti na pole znaků ukončené nulou.

4.3.6 Knihovna *LCD*

LCD není součástí hardware dataloggeru, ale lze ho dodatečně připojit na rozšiřující konektor J4. U prototypu s ATmega32 jsem LCD používal k různým výpisům ladicích informací. Knihovna je navržena pouze pro 4-bitovou komunikaci. Pro správnou funkci je třeba na začátku hlavičkového souboru přiřadit I/O porty jednočipu k daným linkám LCD. Knihovna obsahuje definice konstant příkazů řadiče LCD HD44780 a globální proměnné:

```
Byte lcd_wherex;              // pozice kurzoru x (od 1)
Byte lcd_wherey;              // pozice kurzoru y (od 1),
```

které slouží k uložení aktuální pozice kurzoru. Dále uvedu některé hlavičky funkcí (nebudu uvádět funkce sloužící převážně pro interní potřebu).

```
void lcd_init(Byte lines);
```

Inicializuje LCD, nastaví 4-bitovou komunikaci a 1 nebo 2 aktivní řádky, argument je počet řádků LCD.

```
void lcd_print_char(char c, Byte chardelay);
```

Napíše na aktuální pozici kurzoru LCD 1 znak, argumenty jsou znak a přídavná prodleva v ms (jen na efekt). Aktualizuje globální proměnné **lcd_wherex, lcd_wherey**. Při přetečení prvního řádku se automaticky posune na začátek druhého řádku.

```
void lcd_print(char *str, Byte chardelay);
```

Pomocí předchozí funkce vypíše celý řetězec, vyskytuje-li se v něm znak '**n**', tak odřádkuje a pokračuje. Argumenty jsou pointer na pole znaků ukončené nulou a přídavná prodleva v ms.

```
void lcd_print_P(PGM_P str, Byte chardelay);
```

Tato funkce provádí totéž, pouze s řetězcem uloženým v programové paměti.

```
void lcd_set_user_char(Byte asciicode, Byte *chardata);
```

Řadič LCD HD44780 umožňuje nahrát do svého znakového generátoru definice až osmi uživatelských znaků. Hlavičkový soubor obsahuje na ukázku několik definicí znaků ve formě pole konstant. Tato funkce je pak umožní nahrát do znakového generátoru, argumenty jsou ASCII kód uživatelského znaku 0 – 7 a pointer na pole 8 bytů s definicí znaku.

Další příkazy LCD jsou implementovány jako makra:

```
lcd_clear()           // vymaže LCD a nastaví kurzor na začátek (1,1)
lcd_mode(mode)        // zapne/vypne LCD a nastaví typ kurzoru
lcd_entry_mode(mode) // nastaví směr posunu kurzoru při zápisu znaku
lcd_putc(c)           // napíše 1 znak na aktuální pozici LCD
lcd_move_cursor(direction) // posune kurzor o 1 znak doleva/doprava/zač.
lcd_gotoxy(x,y)       // nastaví kurzor na danou pozici.
```

4.3.7 Knihovna *D1WIRE*

Tato experimentální knihovna slouží ke komunikaci s obvody vybavenými sériovou sběrnicí Dallas One-Wire (obousměrný přenos dat a napájení po jednom vodiči). Další funkce jsou určeny pro přesný digitální teploměr DS18B20 firmy Maxim/Dallas, více informací v [29]. Pro správnou funkci je třeba na začátku hlavičkového souboru přiřadit I/O port jednočipu k lince 1-wire sběrnice. Dále jsou zde definice konstant příkazů pro DS18B20 a datové struktury ROM a fixed point čísla 8:8:8.

Globální proměnné:

```
Byte ds18b20_scratchpad[9]; // pro uložení obsahu 9-bajtové RAM DS18B20
DS18B20_ROM ds18b20_rom;    // struktura pro uložení ROM DS18B20 (64bitů).
```

Dále uvedu hlavičky některých funkcí pro obsluhu sběrnice 1-wire.

```
Byte d1w_reset(void);
```

Provede inicializaci všech zařízení na sběrnici a zjistí, jestli je nějaké zařízení připojeno. Návratová hodnota 0 znamená, že je na sběrnici alespoň jedno zařízení, 1 že není připojeni žádné zařízení.

```
Byte d1w_readbyte(void);
```

Přečte 1 byte vyslaný zařízením.

```
void d1w_writebyte(Byte d);
```

Pošle na sběrnici 1 byte.

Další funkce se týkají DS18B20.

```
Byte ds18b20_read_rom(void);
```

Přečte obsah 64-bitové ROM zařízení do globální proměnné **ds18b20_rom**. ROM obsahuje identifikační byte zařízení (pro DS18B20 hodnota 0x28), 48-bitové jedinečné sériové číslo a 1-bytový CRC. Návratové hodnoty: 0 – zařízení nalezeno a CRC je správný, 1 – špatný CRC, 2 – zařízení nenalezeno.

```
Byte ds18b20_calc_block_crc(Byte *p_block, Byte len);
```

Spočítá CRC bloku dat dané délky, argumenty jsou pointer na pole bytů a délka pole. Návratová hodnota je CRC.

```
Byte ds18b20_read_scratchpad(void);
```

Přečte obsah 9-bytové RAM (zápisník) zařízení do globální proměnné **ds18b20_scratchpad**. Návratové hodnoty: 0 – CRC je správný, 1 – CRC je špatný.

```
Byte ds18b20_write_scratchpad(void);
```

Zapíše globální proměnnou **ds18b20_scratchpad do** zápisníku zařízení. Návratové hodnoty: 0 – zařízení bylo nalezeno, 1 – zařízení nebylo nalezeno.

```
int ds18b20_measure_read_temp(Byte retry_count);
```

Spustí měření teploty, načte zápisník a vrátí teplotu v podobě 16-bitového int. Parametrem je počet opakování při chybném CRC zápisníku.

```
SFIX888 ds18b20_convert_temp(int t);
```

Zkonvertuje teplotu z 16-bitového int do 3-bajtové struktury obsahující znaménko (ASCII) , celočíselnou [0-255] a desetinnou [0-99] část.

4.3.8 Hlavní program *MAIN*

Hlavní program byl vytvořen pouze pro demonstrační účely jednotlivých funkcí a dále si jej upraví uživatel podle konkrétního zadání. Pro nezbytné fungování obsluhy CF/IDE a souborových operací je třeba pouze zavolat tyto funkce v daném pořadí:

```
ide_init();                 // inicializace ATA rozhraní, reset zařízení
// inicializace daného zařízení a globálních proměnných knihovny IDE
ide_init_device(IDE_DRVSEL_MA,ide_sector_buffer,NULL);

// načtení údajů z MBR a PAT, inicializace globálních proměnných knih. FAT
```

```
fat_read_mbr(FAT_MBR_LBA);

// načtení údajů z DBR o souborovém systému, inicializace glob.prom.k. FAT
fat_read_dbr(fat_plba_begin);
```

Pak už lze volat libovolné funkce knihovny *FAT, FATFORM* nebo *IDE*.

Program obsahuje jednoduchou obsluhu USARTu umožňující terminálovou komunikaci s PC po RS232. Komunikační parametry jsou: bitová rychlost 9600 baudů, 8 datových bitů, 1 stop bit, žádná parita a odřádkování sekvencí CR+LF (lze změnit v *RS232.H*). Tato obslužná funkce umožňuje přijímat příkazy zadávané z klávesnice na straně PC. Po přijmutí celého příkazu (ukončeného entrem) je tento příkaz vykonán (pokud je platný) a výsledek se odešle na obrazovku terminálu.

Po zapnutí/resetu dataloggeru by se mělo zobrazit úvodní hlášení o parametrech přenosu. Seznam platných příkazů lze získat zadáním příkazu **help**, viz tab. 4.3.8.1.

Tabulka 4.3.8.1: seznam terminálových příkazů

```
mem      - zobrazí odhad volné paměti (vnitřní datové SRAM)
idecmr   - zobrazí obsah vnitřních registrů IDE zařízení
ideerr   - zobrazí flagy registru STATUS
idedb    - pošle obsah proměnné ide_sector_buffer v binární formě
idedbx   - pošle obsah proměnné ide_sector_buffer v hex. formě
ideid    - provede inicializaci a identifikaci zařízení, vypíše informace
idesd    - otestuje čitelnost všech sektorů zařízení
iders #lba – načte sektor z dané LBA do glob. proměnné ide_sector_buffer
idews #lba – zapíše obsah proměnné ide_sector_buffer do daného sektoru
fatmbr   - načte a zpracuje MBR, zobrazí informace
fatdbr   - načte a zpracuje DBR, zobrazí informace
fatfmt   - naformátuje zařízení souborovým systémem FAT16
fatst    - zobrazí statistiku FAT (počet volných/obsazených/vadných cl.)
fatcch $name – zobrazí řetězec clusterů daného souboru nebo adresáře
dir      - vupíše obsah aktuálního adresáře
cd $name – změní aktuální adresář
md $name – vytvoří nový podadresář
rm $name – smaže soubor nebo prázdný podadresář
cat $name – pošle obsah daného souboru v binární formě
log $name – otevře log soubor, do nějž se budou zapisovat přijaté příkazy
slog     - zastaví logování příkazů do souboru a uzavře ho
finfo    - zobrazí informace o otevřeném log souboru
```

5 ZÁVĚR

Závěrem bych zhodnotil výsledky a průběh své práce. V první etapě jsem prostudoval specifikaci ATA a Compact Flash a vyzkoušel si nízkoúrovňové programování pevného disku IDE přes I/O porty řadiče na PC v prostředí DOS/DJGPP. Po seznámení s procesory rodiny Atmel AVR jsem navrhl a postavil na univerzálním DPS prototyp s procesorem ATmega32 a minimem potřebných periferií. CF kartu jsem připojil přes navrženou redukci.

V další etapě jsem pro tento hardware napsal a odladil základní rutiny pro obsluhu ATA na úrovni řízení jednotlivých signálů, registrů a ATA příkazů. Během ladění jsem narazil na několik zádrhelů, např. díky implicitně (od výroby) zapnutému JTAG rozhraní na ATmega32 měly porty C.2-C.5 vypnuté pull-up rezistory, což způsobovalo nekorektní načítání dat z CF nebo opomenutí atributu volatile u globální proměnné využívané obsluhou USARTu mělo za následek její nefunkčnost.

Když byly odladěny tyto nízkoúrovňové záležitosti, začal jsem návrh finální podoby DPS dataloggeru s výkonnějším procesorem ATmega128. Důraz byl kladen na univerzálnost a modularitu. Proto deska obsahuje řadu konektorů připravených pro připojení rozšiřujících periferií. Jako záznamové zařízení lze připojit CF a nebo pevný disk(y) IDE.

Osazení a oživení DPS proběhlo naprosto bez problémů. Nízkoúrovňové rutiny pro obsluhu periferií jsem napsal dostatečně flexibilně, takže stačilo pouze změnit definice přiřazení I/O portů v hlavičkových souborech a typ cílového procesoru ve skriptu makefile.

V poslední etapě jsem se zaměřil na programování funkcí vyšší úrovně pro práci se souborovým systémem FAT16 (MBR, PAT, DBR, FAT, adresáře a soubory). Díky potrabilitě jazyka C jsem mohl všechny tyto funkce pohodlně psát a ladit na PC a poté je s drobnými úpravami přeložit pro jednočip, kde jsem je ještě znovu otestoval.

Zápis a čtení jsem úspěšně vyzkoušel na několika CF kartách (SanDisk, Canon, Pretec) v kapacitě 8-256 MB a dále na pevných discích Western Digital WDC2540, WDC2635, WDC33100 a Seagate ST51080A. U disků WDC2540 a WDC2635 se ukázalo jako nezbytné provést při inicializaci ATA příkaz INITIALIZE DEVICE PARAMETERS pro nastavení logické CHS geometrie, který ostatní zařízení nevyžadovala.

Výsledkem je tedy funkční hardware schopný zápisu a čtení dat na CF/IDE, který stačí případně dovybavit příslušnými periferiemi podle konkrétního nasazení. Pro danou úlohu je třeba sestavit hlavní řídicí program, který bude využívat potřebné funkce z knihoven.

Náměty na další vylepšení: v současné verzi jsou podporovány pouze základní funkce na čtení/zápis znaku do souboru, v řadě případů by se využily i rychlejší blokové operace (je zde však omezení ze strany velikosti vnitřní datové RAM). V případě větších nároků na kapacitu by bylo možno implementovat podporu souborového systému FAT32. Také je zde prostor pro rychlostní optimalizace nebo podporu paralelního běhu programů pro real-time aplikace.

6 SEZNAM POUŽITÉ LITERATURY

[1] Wikipedia-The Free Encyclopedia. *Hard disk*, 2006 [online]. URL: <http://en.wikipedia.org/wiki/Harddisk>.

[2] Minasi, Mark. *PC Velký průvodce hardwarem*. Praha: Grada, 1998. ISBN 80-7169-667-6.

[3] ATA-ATAPI.COM. *ATA/ATAPI History*, 2005 [online]. URL: <http://www.ata-atapi.com/hist.htm>.

[4] ANSI, ITIC, NCITS, T13. *AT-Attachment with Packet Interface-6 (ATA/ATAPI-6)*, 2000 [online]. URL: <http://pdos.csail.mit.edu/6.828/2005/readings/hardware/ATA-d1410r3a.pdf>.

[5] Wikipedia-The Free Encyclopedia. *CompactFlash*, 2006 [online]. URL: <http://en.wikipedia.org/wiki/Compact_flash>.

[6] Makwana, J. J., Schroder, D. K.. *A Nonvolatile Memory Overview*, 2004 [online]. URL: <http://aplawrence.com/Makwana/nonvolmem.html>.

[7] M-Systems. *Two Technologies Compared: NOR vs. NAND*, 2003 [online]. URL: <www.m-sys.com/NR/rdonlyres/F4F96D49-7EA0-4FA8-882D-21AEAA0CB9C5/229/NOR_vs_NAND.pdf%7CNOR_vs_NAND.pdf>.

[8] Compact Flash Association. *CF+ and Compact Flash Specification Revision 2.0*, 2003 [online]. URL: <http://www.compactflash.org>.

[9] Ray Knights. *Windows 95b (OSR2) MBR*, 2004 [online]. URL: <http://home.att.net/~rayknights/pc_boot/w95bboot.htm>.

[10] Lízal, Vladimír, Hrůza, Petr. *Systémový Manuál*, 1995 [online]. URL: <http://mischa.xf.cz/prirucky/sysman.rar>.

[11] Reifsnyder, B. E.. *Partition Information File for Free FDISK*, 2003 [online]. URL: <http://www.23cc.com/free-fdisk>.

[12] Knights, Ray. *Windows 95b Boot Sector*, 2003 [online]. URL: <http://home.att.net/~rayknights/pc_boot/w95bboot.htm>.

[13] Knights, Ray. *Windows 98 SE Boot Sector*, 2003 [online]. URL: <http://home.att.net/~rayknights/pc_boot/w98bboot.htm>.

[14] Thygesen, Krogh Lasse. *FAT16 File System*, 1999 [online]. URL: <http://www.maverick-os.dk/FileSystemFormats/FAT16_FileSystem.html>.

[15] Wikipedia-The Free Encyclopedia. *File Allocation Table*, 2006 [online]. URL: <http://en.wikipedia.org/wiki/File_Allocation_Table>.

[16] Microsoft. *Limitations of FAT32 File System*, 2004 [online]. URL: <http://support.microsoft.com/default.aspx?scid=KB;EN-US;Q184006&>.

[17] Wikipedia-The Free Encyclopedia. *UTF-16/UCS-2*, 2006 [online]. URL: <http://en.wikipedia.org/wiki/UTF-16>.

[18] Atmel. *8-bit AVR Microcontroller with 128K Bytes Flash In-System Programmable Flash ATmega128(L)*, 2004 [online]. URL: <http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf>.

[19] HW server. *RS-232*, 2003 [online]. URL: <http://rs232.hw.cz>.

[20] Maxim. *MAX232 +5V-Powered, Multichannel RS232 Driver/Receiver*, 2000 [online]. URL: <http://www.trash.net/~luethi/microchip/datasheets/max232.pdf>.

[21] Motorola. *MC34063A DC-to-DC Control Circuit*, 1996 [online]. URL: <http://www.kh-gps.de/mc34063a.pdf>.

[22] Dean, Brian. *AVRDUDE (Formerly AVRPROG)*, [online]. URL: <http://www.bsdhome.com/avrdude>.

[23] Ouwehand, Peter. *How to control a HD44780-based Character-LCD*, 2005 [online]. URL: <http://home.iae.nl/users/pouweha/lcd/lcd.shtml>.

[24] Herout, Pavel. *Učebnice jazyka C (3. vydání)*. České Budějovice: Kopp, 1998. ISBN 80-85828-21-9.

[25] Torvmark, K. H.. *Compiler overview*, 2002 [online]. URL: <http://www.omegav.ntnu.no/~karlto/avr/ccomp.html>.

[26] Weddington, E., Wunsch, J., Flynn, C.. *WinAVR Package*, 2006 [online]. URL: <http://sourceforge.net/projects/winavr>.

[27] AVR-libC community. *Avr-libc Reference Manual 1.4.2*, 2006 [online]. URL: <http://savannah.nongnu.org/projects/avr-libc/>.

[28] Stallman, R. M., GCC Developer Community. *Using the GNU Compiller Collection*, 2004 [online]. URL: <http://www.gnu.org/software/gcc/onlinedocs/>.

[29] Maxim/Dallas. *DS18B20 Programmable Resolution 1-wire Digital Thermometer*, 2002 [online]. URL: <http://www.rentron.com/Files/ds18b20.pdf>.

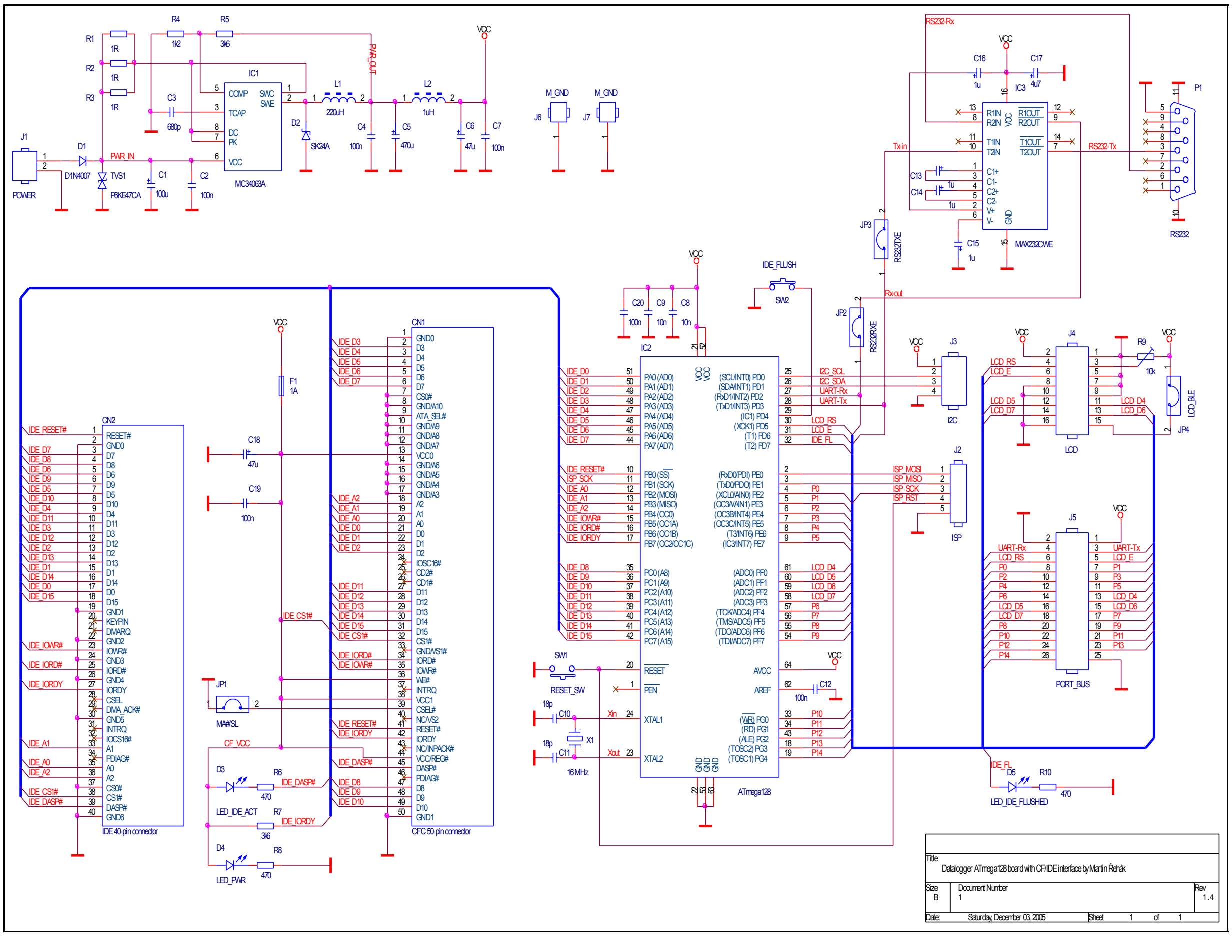
Pozn.: dostupnost všech výše uvedených URL byla ověřena ke dni 26.1.2006.

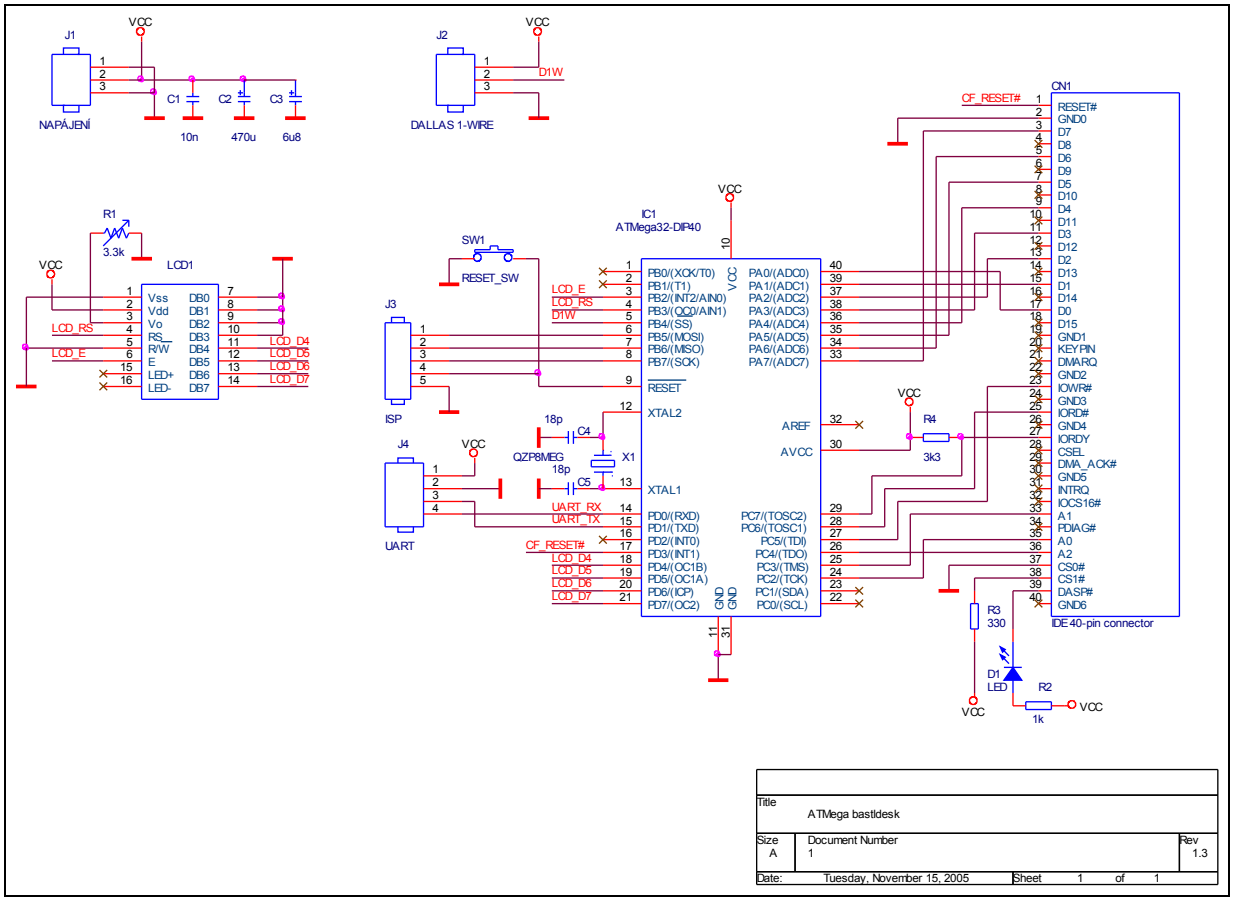
Seznam tabulek

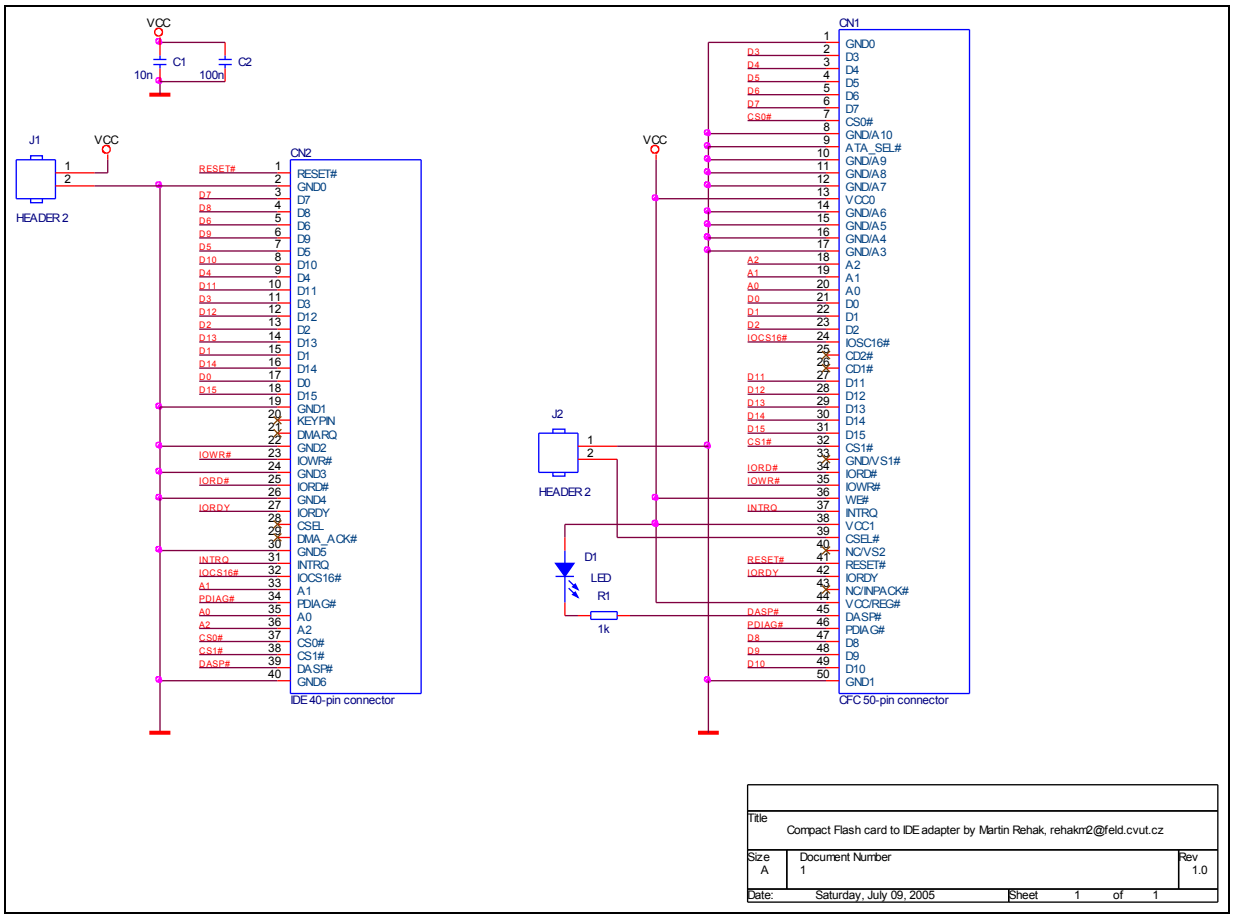
Seznam ilustrací

PŘÍLOHA A

Schéma zapojení dataloggeru a návrh plošného spoje


Title
Datalogger ATmega128 board with CF/IDE interface by Martin Řehák
Size
B
Document Number
1
Rev
1.4
Date:
Saturday, December 03, 2005
Sheet
1
of
1


J1
NAPÁJENÍ
C1
10n
C2
470u
C3
6u8
J2
DALLAS 1-WIRE
D1W
R1
3.3k
LCD1
LCD_RS
LCD_E
LCD_D4
LCD_D5
LCD_D6
LCD_D7
SW1
RESET_SW
IC1
ATMega32-DIP40
J3
ISP
J4
UART
UART_RX
UART_TX
18p
C4
C5
QZP8MEG
X1
CF_RESET#
R4
3k3
CN1
IDE 40-pin connector
R3
330
D1
LED
R2
1k
Title
ATMega bastldesk
Size
A
Document Number
1
Rev
1.3
Date:
Tuesday, November 15, 2005
Sheet
1
of
1


VCC
C1
10n
C2
100n
J1
VCC
1
2
HEADER 2
CN2
RESET#
GND0
D7
D8
D6
D9
D5
D10
D4
D11
D3
D12
D2
D13
D1
D14
D0
D15
GND1
KEYPIN
DMARQ
GND2
IOWR#
GND3
IORD#
GND4
IORDY
CSEL
DMA_ACK#
GND5
INTRQ
IOCS16#
A1
PDIAG#
A0
A2
CS0#
CS1#
DASP#
GND6
IDE 40-pin connector
VCC
J2
1
2
HEADER 2
D1
LED
R1
1k
CN1
GND0
D3
D4
D5
D6
D7
CS0#
GND/A10
ATA_SEL#
GND/A9
GND/A8
GND/A7
VCC0
GND/A6
GND/A5
GND/A4
GND/A3
A2
A1
A0
D0
D1
D2
IOSC16#
CD2#
CD1#
D11
D12
D13
D14
D15
CS1#
GND/VS1#
IORD#
IOWR#
WE#
INTRQ
VCC1
CSEL#
NC/VS2
RESET#
IORDY
NC/INPACK#
VCC/REG#
DASP#
PDIAG#
D8
D9
D10
GND1
CF 50-pin connector
Title
Compact Flash card to IDE adapter by Martin Rehak, rehakm2@feld.cvut.cz
Size
A
Document Number
1
Rev
1.0
Date:
Saturday, July 09, 2005
Sheet
1
of
1

PŘÍLOHA B

Vybrané stránky z katalogových listů použitých integrovaných obvodů

PŘÍLOHA C

Ukázka výpisu z terminálové komunikace PC - datalogger

```
Atmel ATMEGA128 zdravi!
komunikacni parametry: 9600 baud, 8, none, 1

help
mem    - show free (heap) memory
idecmr - IDE command registers dump
ideerr - IDE error details
idedb  - IDE sector buffer dump
idedbx - IDE sector buffer hex-dump
ideid  - IDE init & identify device
idesd  - IDE scan disk test
iders #lba - IDE read LBA sector
idews #lba - IDE write LBA sector
fatmbr - read MBR info
fatdbr - read DBR info
fatfmt - format disk & create FAT16 filesystem
fatst  - display FAT statistics
fatcch #base_cluster - display FAT cluster chain
dir    - display current directory list
cd $name - change current directory
md $name - make directory
rm $name - delete file or empty directory
cat $name - display content of entire file
log $name - start logging commands to file
slog   - stop logging commands and close file
finfo  - display info about opened logfile

mem
free memory = 3245 B

ideid
IDE init & identify device:
OK
model: SanDisk SDCFB-16
s/n:       101417F0703X1735
firmw: Vdg 1.23
max. transfer mode: PIO4
C = 490, H = 2, S = 32
LBA = 31360
total capacity = 15 MB

fatmbr
MBR data:
OK
active/known partition found at LBA: 32, size: 16007168 B
filesystem: FAT16

fatdbr
DBR data:
OK
filesystem OEM ID: ,wwHYIHC
total clusters = 7792
sectors/cluster = 4
number of FATs = 2
1st FAT found at LBA: 33
2nd FAT found at LBA: 64
sectors/FAT = 31
root found at LBA = 95
number of root entries = 512
media descriptor = F8h
volume label: DOS_DISK001
filesystem ID: FAT16

dir
filename.ext -- size -- RHSLDA -- date ---- time - cluster
DOS_DISK 001          0 ---+-- 01-01-2006 00:00:00     0
POKUS1                0 ----+- 26-01-2006 07:02:50    87
POKUS2                0 ----+- 01-01-2006 00:00:00     3
```

```
POKUS3               0 ----+- 01-01-2006 00:00:00     4
AUTOEXEC BAT       795 -----+ 26-01-2006 07:03:18   221
AUTOEXEC DOS      2652 -----+ 25-12-2005 03:24:10   222
BOOT     INI       386 -----+ 20-12-2005 19:30:26   227
MSDOS    DOS     38138 -----+ 12-04-1997 11:23:04   459
4        TXT     98304 -----+ 26-01-2006 14:13:38   173
MANUAL   TXT     51915 -----+ 15-02-2005 19:06:10   357
COMMAND  COM     94706 -----+ 05-05-1999 22:22:00    78
2        TXT    142465 +++--+ 26-01-2006 04:22:18     2
BOOTFONT BIN      4952 -----+ 20-09-2001 13:00:00   223
BOOTSECT DOS       512 -----+ 21-12-2001 00:01:52    85
BOOTSECT NT        512 -----+ 21-12-2001 00:04:10    86
LOG      TXT      2164 -----+ 01-01-2006 12:00:00   383
COMMAND  DOS     54645 -----+ 31-05-1994 06:22:00   293
CONFIG   DOS      2786 -----+ 25-12-2005 02:23:06   320
IO       SYS    222670 -----+ 01-12-2001 09:05:00    74
DYM      TXT       388 -----+ 27-01-2006 07:05:20   478

log log.txt
Log file: log.txt opened

fatst
FAT statistics: OK
FAT signature: FFF8
total clusters =  7792
free clusters =   7351
used clusters =    441
bad clusters =       0

fatcch 2.txt
cluster 2 chain: 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16
 -> 17 -> 18 -> 19 -> 20 -> 21 -> 22 -> 23 -> 24 -> 25 -> 26 -> 27 -> 28 -> 29 -
> 30 -> 31 -> 32 -> 33 -> 34 -> 35 -> 36 -> 37 -> 38 -> 39 -> 40 -> 41 -> 42 ->
43 -> 44 -> 45 -> 46 -> 47 -> 48 -> 49 -> 50 -> 51 -> 52 -> 53 -> 54 -> 55 -> 56
 -> 57 -> 58 -> 59 -> 60 -> 61 -> 62 -> 63 -> 64 -> 65 -> 66 -> 67 -> 68 -> 69 -
> 70 -> 71 -> 72 -> 73 -> eof

cat dym.txt
Content of file: dym.txt
Dymovnice:
**********
Pouzitelne jako dymovnice.
Hori pomalu, speka se na strusku.

     KNO3 (Dusicnan draselny) ... 30%
     Cukr moucka              ... 60%
     Kakao / drevene piliny   ... 10%

Rozdrtime na jemny prasek a dukladne promichame. Nejvic cadi prasek jen tak,
nebo ho nasypeme do "vacku" z alobalu. Bez kakaa / pilin se da pouzit
jako palivo do raketoveho motorku.


EOF

cd pokus1
Change current directory to: pokus1
current directory cluster = 87

dir
filename.ext -- size -- RHSLDA -- date ---- time - cluster
.                    0 ----+- 26-01-2006 07:02:50    87
..                   0 ----+- 26-01-2006 07:02:50     0
A                    0 ----+- 26-01-2006 07:02:56    88
B                    0 ----+- 26-01-2006 07:02:58    89
BOOTSECT W2K       512 -----+ 22-12-2004 16:17:12    90
BOOTSECT W98       512 -----+ 21-12-2001 00:13:52    91
COMMAND  COM     94706 -----+ 05-05-1999 22:22:00    92
```

```
COMMAND  DOS      54645 -----+ 31-05-1994 06:22:00   139

md new
Make directory: new
OK

mkf newfile.txt
Make file: newfile.txt
dir
filename.ext -- size -- RHSLDA -- date ---- time - cluster
.                     0 ----+- 26-01-2006 07:02:50    87
..                    0 ----+- 26-01-2006 07:02:50     0
A                     0 ----+- 26-01-2006 07:02:56    88
B                     0 ----+- 26-01-2006 07:02:58    89
BOOTSECT W2K        512 -----+ 22-12-2004 16:17:12    90
BOOTSECT W98        512 -----+ 21-12-2001 00:13:52    91
COMMAND  COM      94706 -----+ 05-05-1999 22:22:00    92
COMMAND  DOS      54645 -----+ 31-05-1994 06:22:00   139
NEW                   0 ----+- 01-01-2006 12:00:00   384
NEWFILE  TXT          4 -----+ 01-01-2006 12:00:00   385

cat newfile.txt
Content of file: newfile.txt
Ahoj
EOF

rm newfile.txt
Delete file/directory: newfile.txt
OK

cd /
Change current directory to: /
current directory cluster = 0

slog
Logging stopped

cat log.txt
Content of file: log.txt
log LOG.TXT
fatst
fatcch 2.txt
cat dym.txt
cd pokus1
dir
md new
mkf newfile.txt
cat newfile.txt
rm newfile.txt

EOF
```

PŘÍLOHA D

Obsah přiloženého CD-ROM:

- text diplomové práce ve formátu PDF,
- specifikace ATA/ATAPI, Compact Flash a další použitá literatura,
- datasheety použitých integrovaných obvodů,
- schémata a návrhové soubory DPS programu OrCAD,
- vývojový balík nástrojů WinAVR (avr-gcc 3.4.3, avr-libc 1.4.3, avrdude 5.1),
- zdrojové a binární soubory software dataloggeru.