Vážení zákazníci,

dovolujeme si Vás upozornit, že na tuto ukázku knihy se vztahují autorská práva, tzv. copyright.

To znamená, že ukázka má sloužit výhradně pro osobní potřebu potenciálního kupujícího (aby čtenář viděl, jakým způsobem je titul zpracován a mohl se také podle tohoto, jako jednoho z parametrů, rozhodnout, zda titul koupí či ne).

Z toho vyplývá, že není dovoleno tuto ukázku jakýmkoliv způsobem dále šířit, veřejně či neveřejně např. umisťováním na datová média, na jiné internetové stránky (ani prostřednictvím odkazů) apod.

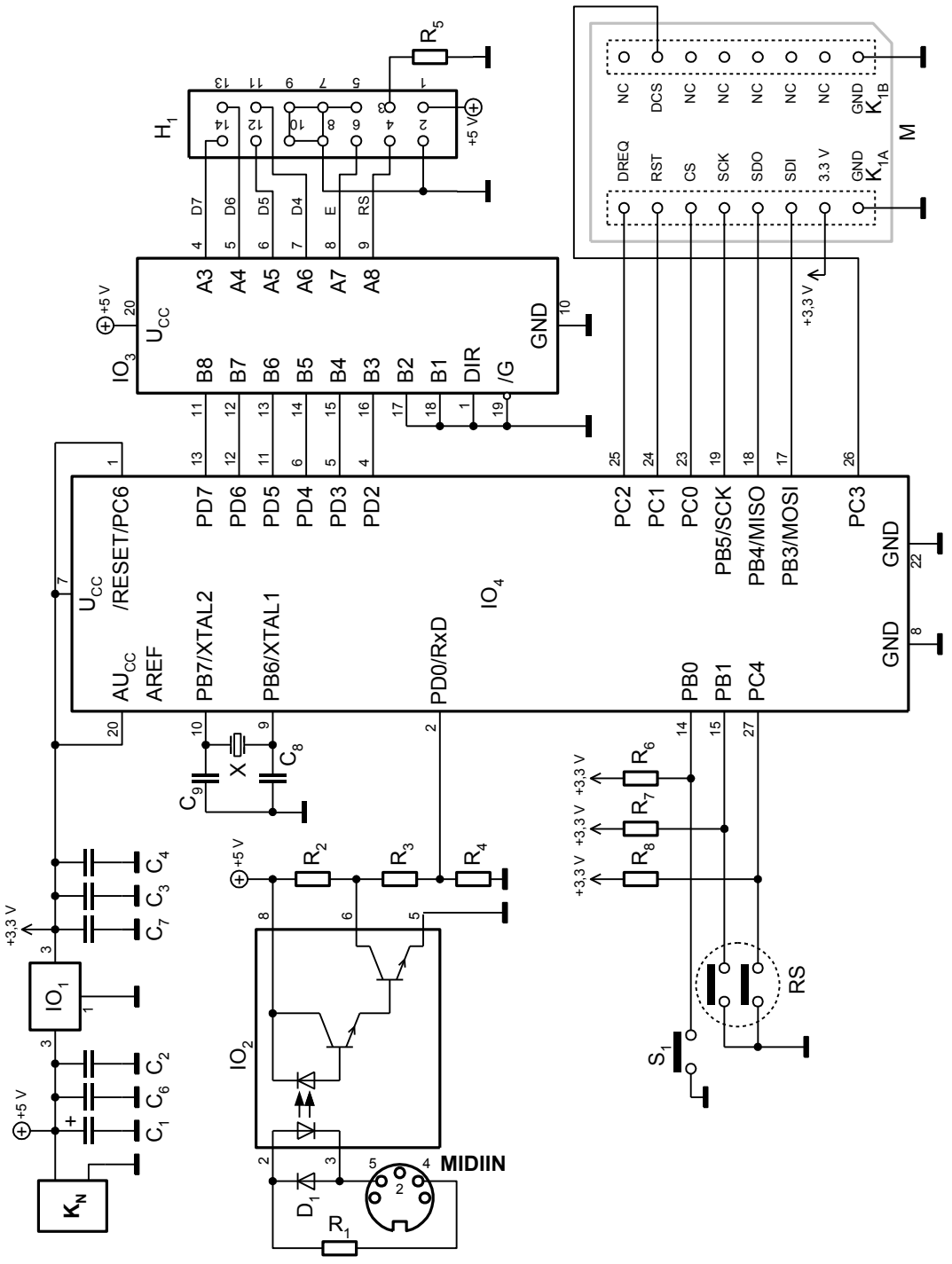
redakce nakladatelství BEN – technická literatura
redakce@ben.cz

15 MIDISYNT – MIDI syntezátor

MIDI syntezátor je zařízení, které přijímá MIDI zprávy a reaguje na ně tím, že provádí příslušné příkazy. Například přehrává příslušné noty nebo mění programy (typ nástroje). Režim real-time MIDI, který je implementován v kodeku VS1053b, umožňuje vytvořit MIDI syntezátor relativně jednoduché konstrukce.

15.1 Přípravek MIDISYNT – MIDI syntezátor

Schéma zapojení MIDI syntezátoru **MIDISYNT** je uvedeno na obr. 15.1. Poklady pro výrobu naleznete v příloze A.11.



*Obr. 15.1 Schéma zapojení přípravku **MIDISYNT***

Jádrem zapojení je mikrokontrolér **IO4** typu **ATmega8A-PU**, který pomocí vstupního konektoru MIDI přijímá zprávy, které pak předává na kodek. Spínače slouží pro ovládání hlasitosti, která se zobrazuje na displeji.

Celek je napájen přes konektor **KN**. Pro napájení lze použít adaptér 5 V/1 A, doporučujeme **ET328A** (GME) resp. **AC/DC-LV5/1** (TME).

Pro zjednodušení propojení mezi mikrokontrolérem a kodekem jsme se rozhodli použít napájení 3,3 V (odpadají pak odporové děliče pro úpravu napětí řídicích signálů). Stabilizátor **IO1** poskytuje napětí 3,3 V na svém výstupu. Toto napájení je pak přímo použito pro mikrokontrolér **IO4** a modul kodeku **M**.

Vstupní signál se připojuje na konektor **MIDIIN**, pro galvanické oddělení (viz obr. 9.1) je použit optočlen **IO2** (6N138), který pracuje na výstupní straně s napájením 5 V (požadovaná vysoká přenosová rychlost je totiž garantována pro 5 V). Proto je na výstupu ještě pomocný dělič R3, R4, který napětí sníží na rozsah zhruba 0 až 3,3 V. Získaný signál je pak již přiveden na vývod **RxD** (PD0) mikrokontroléru. Zpětné natvarování na obdélníkový signál s ostrými hranami je zajištěno přímo ve vstupním obvodu vývodu mikrokontroléru Schmittovým klopným obvodem (viz obr. 3.30).

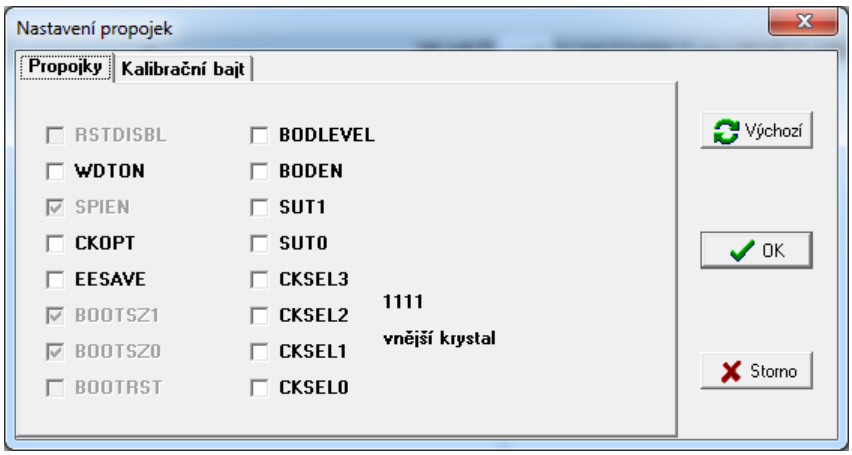
Budič sběrnice **IO3** (74HCT245) slouží pro úpravu napěťových úrovní z 3,3 V logiky na logiku 5 V. Tato úprava je nutná pro řízení displeje, který je napájen z 5 V. Pro zjednodušení zapojení jsou vývody mezi portem PD a displejem přetočeny (přetočení bitů při odesílání dat a příkazů na displej je řešeno programově).

LCD je v klasickém 2řádkovém provedení s 2×7 vývody připojen přes „header“ 2×7 kolíků označený **H1**. V tomto zapojení doporučujeme použít LCD typu **WD-C1602L-7GNNa**, viz: http://mujweb.cz/hezky.den/datasheet/lcd-wd-c1602l-7gnna.htm. Displej slouží pro zobrazení nastavené hlasitosti.

Spínač **S1** zajišťuje funkci MUTE (rychlé vypnutí/zapnutí zvuku), je připojen na vývod PB0. Rotační spínač **RS** slouží pro plynulou regulaci hlasitosti, je připojen na vývody PB1 a PC4 (vývod PB2, který odpovídá funkci SS – Slave Select, nelze používat ve vstupním směru – jsou pak problémy s jednotkou SPI).

Modul kodeku **M** je připojen pomocí konektorů **K1A**, **K1B** na vývody portů PB a PC. Napájení je 3,3 V.

Pro přesné časování USART je nutné použít vnější krystal 8 MHz označený **X**, který používá vazební kondenzátory **C8**, **C9**. Nastavení pro tento případ a další potřebné volby ukazuje obr. 15.2 (byl zvolen dlouhý rozběh pomocí propojek SUT0 a SUT1).



Obr. 15.2 Nastavení propojek ATmega8 pro vnější krystal 8 MHz + další volby

15.2 Firmware pro MIDISYNT

Program najdete v doprovodném ZIP archivu v adresáři **FIRMWARE/MIDISYNT**.

Pro přehlednost jsou jednotlivé části programu rozděleny do několika jednotek podle účelu jejich použití. Celkově má program tuto modulární strukturu [17]:

- **LCD** – ovládání displeje,
- **VS1053B** – ovládání kodeku VS1053b,
- **MISC** – pomocné funkce (konfigurace portů, ovládání USART, ovládání spínačů),
- **MIDISYNT** – integrující jednotka, obsahuje hlavní program.

15.2.1 Jednotka LCD

Jednotka **LCD** exportuje symboly příkazů pro smazání displeje, nastavení na začátek prvního nebo druhého řádku (**LCD_CLEAR**, **LCD_HOME** a **LCD_AT64**). Dále jsou exportovány funkce:

- **prikazLCD** – odešle určený příkaz (například smazání displeje),
- **iniLCD** – provede inicializaci displeje pro jeho použití, rovněž přesměruje standardní výstup pro použití displeje (funkce **printf** pak provádí výpisy na displeji),
- **iniUDG** – nahraje do CGRAM displeje 8 znaků, které se zobrazují při výpisu hlasitosti (viz níže).
- **pisUDG** – vypíše na LCD 0 až 8 znaků pro zobrazení hlasitosti (viz níže),
- **pisLCD** – vypíše na LCD určený znak.

LCD.H:

```
#ifndef LCD_H
#define LCD_H

#define LCD_CLEAR 1      //smazani displeje
#define LCD_HOME  2      //navrat na zacatek
#define LCD_AT64  128+64 //zacatek 2. radku

void prikazLCD(unsigned char d);
void iniLCD();
void iniUDG();
void pisUDG(unsigned char p);
void pisLCD(char c);

#endif
```

Funkce pro ovládání LCD jsou přejaty z [7]. Nebudeme je nyní podrobněji popisovat (jejich výpis také zkrátíme). Zastavíme se pouze u nově přidaných funkcí.

Funkce **pisport** slouží pro odeslání dat na port D. Tato funkce zajišťuje nutné přetočení bitů a dále ponechává vždy připojený pull-up rezistor na vývodu PD0.

Funkce **iniUDG** slouží pro uložení uživatelsky definovaných znaků do CGRAM displeje. Viz [18] str. 16 a 18. Tyto znaky odpovídají sloupcům proměnné výšky 1 až 8. Jak ukazuje obr. 15.3. Obrazce znaků jsou čteny z pole konstant **UDG**.

Funkce **pisUDG** slouží pro výpis požadovaného počtu sloupců. Parametr určuje, kolik sloupců je vypsáno (0 až 8). Sloupce se vypisují zleva doprava od pozice devátého sloupce ve druhém řádku. Prázdné pozice jsou pak přepisovány mezerami.

Obr. 15.3 Obrazce UDG znaků pro výpis hlasitosti (obrazce jsou uloženy v CGRAM)

LCD.C:

```
#define __AVR_ATmega8__ 1
#define F_CPU 8000000UL

#include <avr/io.h>
#include <avr/pgmspace.h>
#include <util/delay.h>
#include <stdio.h>
#include "LCD.h"

#define LCD_E_BIT  3  //cislo bitu E
#define LCD_RS_BIT 2  //cislo bitu RS

//posila bity pres port D (otocene poradi)
void pisport(unsigned char data)
{
  unsigned bity=((data&128)>>5)|((data&64)>>3)
              |((data&32)>>1)|((data&16)<<1)
              |((data&8)<<3)|((data&4)<<5);
  PORTD=bity|(1<<PORTD0);
}

//posila 4 bity, rs urcuje prikaz (rs=0) nebo data (rs=1)
void pisLCD4bity(unsigned char d,unsigned char rs)
{
  pisport(d|(0<<LCD_E_BIT)|(rs<<LCD_RS_BIT));  //E=0
  _delay_us(0.140);
  pisport(d|(1<<LCD_E_BIT)|(rs<<LCD_RS_BIT));  //E=1
  _delay_us(0.500);
  pisport(d|(0<<LCD_E_BIT)|(rs<<LCD_RS_BIT));  //E=0
  _delay_us(0.500);
}

//posila 8 bitu nadvakrat,
//rs urcuje prikaz (rs=0) nebo data (rs=1)
void pisLCD8bitu(unsigned char d,unsigned char rs)
...

//posle prikaz:
void prikazLCD(unsigned char d)
...

//presmeruje printf:
int putcLCD(char c, FILE *stream)
...

//posle znak:
void pisLCD(char c)
...
```

```
//uzivatelske znaky displeje
const unsigned char UDG[] PROGMEM=
  {00,00,00,00,00,00,00,31,
   00,00,00,00,00,00,31,31,
   00,00,00,00,00,31,31,31,
   00,00,00,00,31,31,31,31,
   00,00,00,31,31,31,31,31,
   00,00,31,31,31,31,31,31,
   00,31,31,31,31,31,31,31,
   31,31,31,31,31,31,31,31};
//ulozi znaky do CGRAM
void iniUDG()
{
  prikazLCD(64);  //uk. CGRAM=0
  for(unsigned char i=0;i<64;i++) //odesle kody
    pisLCD(pgm_read_byte(&UDG[i]));
}

//vypise tolik sloupcu, kolik udava p
//p muze byt 0 az 8
void pisUDG(unsigned char p)
{
  //pozice: 2. radek, 9. sloupec:
  prikazLCD(LCD_AT64+8);
  //pise sloupce nebo mezery:
  for(unsigned char i=0;i<8;i++)
  {
    if(i<p)
      pisLCD(i);  //sloupec i
    else
      pisLCD(' ');  //nebo mezera
  }
}

//inicializace LCD:
void iniLCD()
...
```

15.2.2 Jednotka VS1053B

Jednotka **VS1053B** vychází z funkcí, které jsme vytvořili v předchozích kapitolách. Významné změny jsou hlavně v jiném napojení vývodů procesoru, které musí být ve zdrojovém textu jednotlivých funkcí zohledněny.

VS1053B.H:

```
#ifndef VS1053B_H
#define

//cisla bitu VS1053b (port C):
#define RST  1  //RST
#define CS   0  //CS
#define DCS  3  //DCS
#define DREQ 2  //DREQ
```

```
//cisla bitu VS1053b (port B):
#define SCK  5  //SCK
#define MISO 4  //MISO (SDO)
#define MOSI 3  //MOSI (SDI)
#define SS   2  //SS

//cisla SCI registru:
#define SCI_MODE   0x00
#define SCI_VOL    0x0b

void initSPI();
void resetCodec();
void writeSCI(unsigned char addr,unsigned data);
void rtMIDIStart();
void midiSPISendByte(unsigned char d);
void midiSPIChangePrg(unsigned char program);
void midiSPINoteOn(unsigned char note);
void midiSPINoteOff(unsigned char note);

#endif
```

Důležitá změna je provedena také v inicializaci jednotky SPI (funkce **initSPI**). Připomeňme, že SPI sběrnice kodeku musí používat hodinový signál rovný čtvrtině resp. sedmině hodinového kmitočtu kodeku. Jelikož jsme hodinový kmitočet mikrokontroléru zvýšili na 8 MHz, musela být provedena změna nastavení přenosové rychlosti SPI na 1 MHz.

VS1053B.C:

```
#define __AVR_ATmega8__ 1
#define F_CPU 8000000UL

//kodek
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <util/delay.h>
#include <stdio.h>
#include "VS1053B.h"

//inicializace SPI
void initSPI()
{
  SPCR=(1<<SPE)|(1<<MSTR)|(0<<SPR1)|(1<<SPR0);   //režim SPI
  SPSR=(1<<SPI2X);           //rychlost f0/8 (1 MHz)
}

//posle jeden bajt pres SPI
void inline sendSPI(unsigned char d)
...
```

```
//ceka na uvolneni DREQ
void inline waitDREQ()
{
  while(!(PINC&(1<<DREQ))); //ceka na DREQ=1
}

//hw reset kodeku
void resetCodec()
{
  PORTC&=~(1<<RST);//RST=0
  _delay_ms(1);
  PORTC|=(1<<RST); //RST=1
  _delay_ms(300);  //pro uspesne provedeni
}

//zapise do SCI registru
void writeSCI(unsigned char addr,unsigned data)
{
  PORTC&=~(1<<CS);  //CS=0
  sendSPI(0b00000010); //instrukce pro zapis
  sendSPI(addr);       //adresa registru
  sendSPI(data>>8);    //horni bajt dat
  sendSPI(data&0xff);  //dolni bajt dat
  PORTC|=(1<<CS);   //CS=1
  waitDREQ();    //ceka na DREQ
  _delay_ms(1);  //dalsi pauza
}

//pole konstant pro prepnuti do run-time MIDI rezimu
const unsigned char rtreg[22] PROGMEM=
...
const unsigned rtcode[22] PROGMEM=
...
//prepne kodek do run-time MIDI rezimu
void rtMIDIStart()
...

//posle jeden bajt pro MIDI pres SPI
void midiSPISendByte(unsigned char d)
{
  PORTC&=~(1<<DCS);  //DCS=0
  sendSPI(0x00);     //odeslani 0
  sendSPI(d);        //odeslani bajtu
  PORTC|=(1<<DCS);   //DCS=1
  waitDREQ();        //test DREQ
}

//zmeni program MIDI na kanalu 0
void midiSPIChangePrg(unsigned char program)
...

//zapne notu na kanalu 0
void midiSPINoteOn(unsigned char note)
...
```

```
//cte aktualni stav kontaktu rotacniho spinace
unsigned char getEncStatus()
{
  unsigned char a,b;

  do
  {
    a=ROT_A|ROT_B;  //1. vzorek
    _delay_us(1000);
    b=ROT_A|ROT_B;  //2. vzorek
  }
  while (a!=b);

  return a;
}

//cte stav rotacniho spinace
unsigned char getEncoder()
{
  unsigned char a,b,c;

  a=getEncStatus();  //pozice 1
  _delay_us(1000);
  b=getEncStatus();  //pozice 2

  c=(a<<4)|b;  //sestaveni kodu

  if(c==0x01 || c==0x13 || c==0x32 || c==0x20)
    return 1;  //po smeru hod. rucek
  if(c==0x02 || c==0x23 || c==0x31 || c==0x10)
    return 2;  //proti smeru hod. rucek

  return 0;  //zadne otaceni
}
```

15.2.4 Jednotka MIDISYNT

Jednotka **MIDISYNT** je hlavní jednotkou programu. Využívá připravené funkce a obsahuje hlavní program.

Symboly **VOL_MAX**, **VOL_INIT**, **VOL_SHR**, **VOL_SHL** souvisí s ovládáním hlasitosti. Hlasitost se nastavuje rotačním spínačem **RS** v rozsahu 0 až VOL_MAX (tedy v 64 stupních). Výchozí hodnota hlasitosti po resetu je dána jako VOL_INIT (50). Ostatní symboly slouží pro přepočet hlasitosti jak pro zobrazení na displeji tak pro odeslání na kodek (viz níže).

Použité proměnné: **h** (hlasitost), **mute** (indikuje MUTE funkci), **enc** (pro čtení rotačního spínače RS), **m** (pro čtení tlačítka S_1).

Na začátku je provedena inicializace portů, LCD, UDG znaků, SPI jednotky, kodeku včetně přepnutí do režimu run-time MIDI, jednotky USART. Do prvního řádku displeje je vypsán text MIDISYNT.

Program poté vstupuje do hlavní smyčky:

Do proměnné **enc** je čten stav rotačního spínače. Podle toho se zvýší nebo sníží hlasitost. Podle aktuálního obsahu proměnné **h** se vypíše nastavená hlasitost funkcí **pisUDG** (pro přepočet na rozsah 0 až 8 se používá posuv doprava, který upraví původní hodnotu hlasitosti v rozsahu 0 až 63).

Funkce **getMute** čte stav tlačítka S_1. Při stisku dojde ke změně obsahu proměnné **mute**, která řídí režim MUTE.

Je-li zvolen režim MUTE, je do druhého řádku vypsán text MUTE (v obráceném případě se text přemaže mezerami).

Nakonec se odešle nová hodnota hlasitosti do SCI registru **SCI_VOL**. Pokud je aktivní režim MUTE, odešle se hodnota 0xFFFF (vypnuto) jinak se odešle přepočítaná hodnota proměnné **h** (přepočet je proveden posuvem doleva, oba bajty mají stejnou hodnotu, bitová negace odpovídá tomu, že maximální hlasitost je 0x0000). Vykonání funkce **writeSCI** musí být provedeno při zakázaném přerušení. Jinak hrozí střet mezi zápisem přes rozhraní SDI a SCI. Je-li přerušení zakázáno, nemůže obsluha přerušení reagovat na nově přijaté MIDI zprávy (odeslání proběhne po povolení přerušení).

Nakonec je zařazeno čekání na uvolnění tlačítka S_1, aby se režim MUTE mohl měnit až po předchozím uvolnění tlačítka.

MIDISYNC.C:

```
#define __AVR_ATmega8__ 1
#define F_CPU 8000000UL

//firmware MIDISYNT
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <util/delay.h>
#include <stdio.h>
#include "LCD.h"
#include "VS1053B.h"
#include "MISC.h"

#define VOL_MAX  63  //maximalni hlasitost
#define VOL_INIT 50  //vychozi hlasitost
#define VOL_SHR  3   //rotace pro zobrazeni
#define VOL_SHL  2   //rotace pro SCI_VOL

int main()
{
  unsigned char h=VOL_INIT;  //hlasitost
  unsigned char mute=0;      //mute
  unsigned char enc;   //stav rot. spinace
  unsigned char m;     //stav tl. MUTE

  initports();   //inicializace portu ATmega
  iniLCD();      //inicializace LCD
  iniUDG();

  initSPI();     //inicializace SPI
  resetCodec();  //reset kodeku
  rtMIDIStart(); //prepne na run-time MIDI

  initUSART();   //inicializace USART

  //uvodni text
  prikazLCD(LCD_HOME);
  printf_P(PSTR("MIDISYNT"));
```

16 SD karta a její ovládání

V této kapitole vysvětlíme způsob uložení dat na SD kartě a dále její ovládání. Níže budeme předpokládat, že SD karta je naformátována pro systém FAT32 (což je asi nejobvyklejší případ). Dovolujeme si připomenout, že pro ukládání vícebajtových čísel se standardně používá formát **Little Endian** (na nejnižší adrese je uveden nejméně významný bajt čísla).

16.1 Souborový systém FAT32

Data na disku jsou rozdělena do alokačních jednotek. Nejmenší přímo přístupná jednotka je tzv. **sektor**, který má obvykle velikost **512 B**. Pro disky větší kapacity je nutné sdružovat sektory do větších alokačních jednotek, které označujeme jako **clustery** (délka clusteru je celočíselný násobek délky sektoru). Každý disk má tyto čtyři základní oblasti:

1. **Zaváděcí záznam (MBR)** – odpovídá prvnímu fyzickému sektoru (sektor číslo 0).
2. **Tabulka obsazení disku (FAT)** – určuje rozmístění informací na disku.
3. **Kořenový adresář (Root Directory)** – informace o souborech.
4. **Datová oblast (Data Area)** – data jednotlivých souborů.

Zaváděcí záznam (MBR – Master Boot Record)

MBR obsahuje kód systémového zavaděče (pokud se jedná o systémový disk), velikost sektoru, počet sektorů clusteru, počet rezervovaných sektorů na začátku disku, počet kopií FAT, počet položek v kořenovém adresáři, celkový počet sektorů, počet povrchů, počet skrytých sektorů. Viz [19]. Základní popis viz tab. 16.1.

Tab. 16.1 Popis MBR

Ofset	Popis	Velikost [B]
0x000	kód zavaděče	446
0x1be	údaje 1. partition	16
0x1ce	údaje 2. partition	16
0x1de	údaje 3. partition	16
0x1ee	údaje 4. partition	16
0x1fe	signatura (0x55, 0xAA)	2

Signatura (bajty hodnot **0x55 0xAA** za sebou) představuje ukončení sektoru. Podle toho, kolik logických disků (partitions) je na disku, jsou vyplněny údaje o partitions, viz tab. 16.2 (ofsety jsou stanoveny relativně k začátku 0x1be, resp. 0x1ce, 0x1de, 0x1fe).

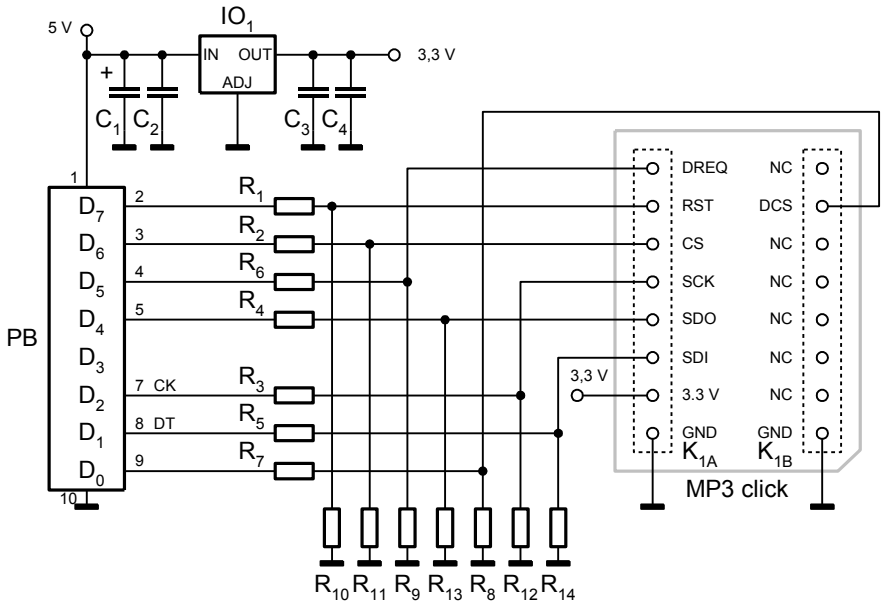
Tab. 16.2 Popis údajů o jedné partition

Ofset	Popis	Velikost [B]
+0x0	stav partition (0 – aktivní)	1
+0x1	absolutní CHS (cilinder/head/sector) adresa prvního sektoru	3
+0x4	typ partition	1
+0x5	absolutní CHS adresa posledního sektoru	3
+0x8	*absolutní LBA adresa prvního sektoru*	4
+0xc	*počet sektorů*	4

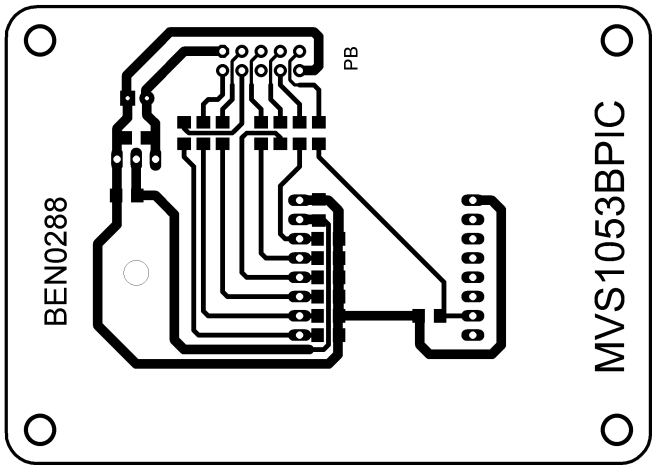
Nejzajímavější jsou pro nás údaje na ofsetu **+0x8** (jedná se o adresu prvního sektoru dané partition v LBA; LBA značí Logical Block Addressing – adresa je lineární ne ve formátu 3D jako u CHS adresování) a na ofsetu **+0xc** (počet sektorů).

A.10 MVS1053BPIC – úprava desky MVS1053B pro PIC16F628A

Schéma zapojení přípravku **MVS1053BPIC** je uvedeno na obr. A.40.



*Obr. A.40 Schéma zapojení přípravku **MVS1053BPIC***



*Obr. A.41 Výkres desky plošných spojů přípravku **MVS1053BPIC***

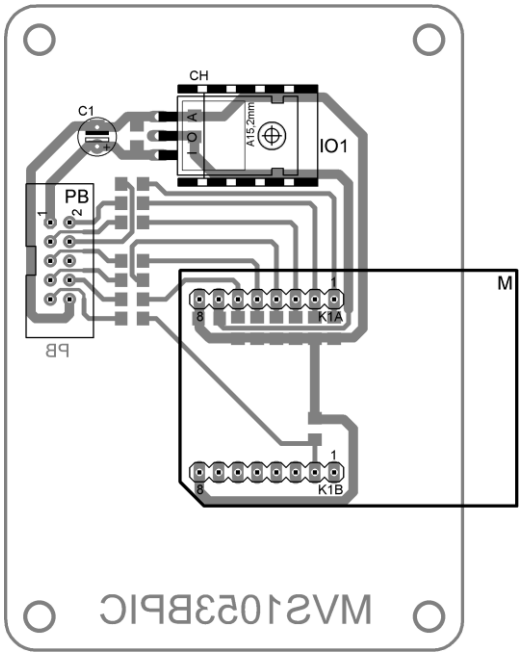
Seznam součástek (opisný):

R_1 až R_7	rezistor SMD 4,7 kΩ, velikost 1206	7 ks
R_8 až R_{14}	rezistor SMD 10 kΩ, velikost 1206	7 ks
C_1	elektrolyt. kondenzátor 100 μF/16V, radiální vývody, rozteč 2,5 mm	1 ks
C_2 až C_4	keramický SMD kondenzátor 100 nF, velikost 1206	3 ks
IO_1	nízkoúbytkový stabilizátor 3,3 V LD1117V33 + chladič	1 ks
K_{1A}, K_{1B}	dutinková lišta do DPS, 8 kolíků, rozteč 2,54 mm	2 ks
M	kit MP3 click se zapájenými kolíkovými lištami	1 ks
PB	konektor pro ploché kabely do DPS, 2×5 kolíků, rozteč 2,54 mm	1 ks

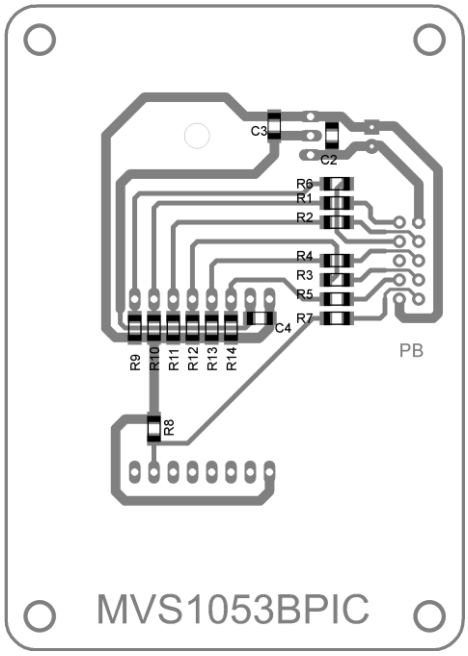
Rozpis součástek:

Označení	dle GME	dle TME	počet
R_1 až R_7	R1206 4K7	RC1206JR-074K7	7 ks
R_8 až R_{14}	R1206 10K	RC1206JR-0710K	7 ks
C_1	CE 100u/16V	CE-100/16PHT-Y	1 ks
C_2 až C_4	CKS1206 100n	CC1206KKX7R0104	3 ks
IO_1	TS1084CZ33 CO +chladič DO1A	LD1117V33 +chladič D01A	1 ks
K_{1A}, K_{1B}	2×BL804G	ZL262-8SG	2 ks
M	*není k dispozici v GME*	modul MP3 click	1 ks
PB	MLW10G	ZL231-10PG	1 ks

Modul MP3 click se osadí do dvou dutinkových lišt **K_{1A}**, **K_{1B}**. Zespodu se osazují pouze SMD rezistory a kondenzátory. Chladič stabilizátoru **IO_1** připevníme pomocí šroubku M3.



Obr. A.42 Osazovací plánek (strana součástek)



Obr. A.43 Osazovací plánek (strana spojů, SMD)

Jako dno lze použít jeden díl krabičky **KM22** (GME) resp. **KM-22** (TME).