Technická dokumentace

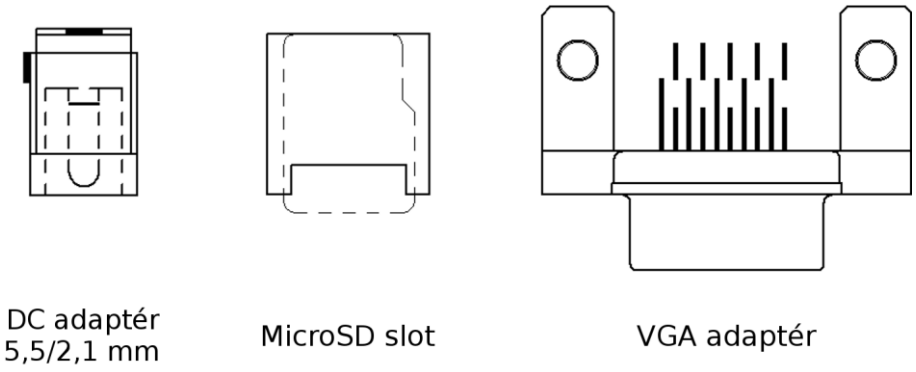
Mikroprocesorová vykreslovací jednotka

2015 Tomáš Pazdiora

Obsah

1. Hardwarová specifikace

1.1. Vstupy a výstupy



Obr. 1 – 1: Vstupy a výstupy

1.2. Elektrické vlastnosti

Základní parametry	
Vstupní napájení	12 V DC
Maximální spotřeba	1 W
Typická spotřeba	0,75 W

Tab. 1 – 1: Základní elektrické parametry

Detailní parametry		
Část zařízení	**Napájení**	**Typický proud**
ATtiny4313	5,0 V	16 mA
ATmega32	5,0 V	26 mA
RAM modul	5,0 V	18 mA
DAC	5,0 V	10 mA
SD karta	3,3 V	15 mA
	Napájení	**Efektivita**
Regulátor	12 V DC	>75 %

Tab. 1 – 2: Detailní elektrické parametry

1.3. MIKROPROCESORY

Použité mikroprocesory jsou od firmy Atmel, mají 8bitové jádro AVR, jsou programovatelné skrz rozhraní SPI nebo paralelně.

Jako zobrazovací mikroprocesor je použit mikroprocesor **ATtiny4313**. Pracuje na frekvenci 20 MHz při napájecím napětí 5 V. Kód zabírá 97.9 % paměti flash. Pokud by bylo potřeba uvolnit místo, je doporučeno odstranit podporu kopírování z jedné paměti do druhé (kód hledejte pod návěštím „copy“).

Vykreslovacím mikroprocesorem je mikroprocesor **ATmega32**. Pracuje na frekvenci 16MHz při napájecím napětí 5 V. Velikost kódu je závislá na použitých knihovnách.

V následující tabulce jsou vypsána nastavení pojistek obou mikroprocesorů.

Nastavení pojistek		
Typ pojistky	**ATtiny4313**	**ATmega32**
Low	0xFF	0x3F
High	0xD9	0xD9
Extended	0xFF	---

Tab. 1 – 3: Nastavení pojistek mikroprocesorů

2. INTERPRETER

Toto zařízení má základní podporu interpretace jednoduchých příkazů. Interpreter načítá příkazy ze souboru „**main.txt**“ (platí pouze pro aktuální program nahraný v mikroprocesoru). *Soubor, ze kterého se načítají příkazy, je zvolen při volání funkce „load_commands“ (tato funkce je obsažena v knihovně „cmd_parser.h“).*

Interpreter aktuálně podporuje 14 příkazů. *Další příkazy mohou být implementovány úpravou funkce „process_command“ v knihovně „cmd_parser.h“ vykreslovacího mikroprocesoru.*

Syntax vychází ze zápisu volání funkcí v jazyce C. Jsou podporovány jednořádkové a víceřádkové komentáře. Syntax vypadá následovně:

```
// Jednořádkový komentář

/*    Víceřádkový komentář
      Víceřádkový komentář
      Víceřádkový komentář    */

// příkaz (argument1, argument2, ... );

// následují příklady zápisu příkazů

bmp_config(0);
clear_color(0,0,0);
draw_bmp ("images/22.bmp");
repeat();
```

2.1. INTERPRETOVANÉ PŘÍKAZY

`draw_bmp(file_path);`

Vykreslení BMP obrázku.

- file_path – cesta k obrázku včetně uvozovek (např. "slozka/obrazek.bmp")

`draw_bmp_from_folder(folder);`

Vykreslení jakéhokoliv bmp obrázku z dané složky.

- folder – cesta k dané složce včetně uvozovek (např. "slozka/podslozka")

`draw_pal(swap,copy);`

Vykreslení palety.

- swap – prohození vykreslovací a zobrazovací paměti po dokončení vykreslování (0 nebo 1)
- copy – postupné vykreslování (0 nebo 1)

`draw_hsv_pal(swap,copy);`

Vykreslení HSV palety.

- swap – prohození vykreslovací a zobrazovací paměti po dokončení vykreslování (0 nebo 1)
- copy – postupné vykreslování (0 nebo 1)

`draw_test(test_number,swap,copy);`

Vykreslení funkcí z „draw_test.h“.

- číslo vykreslovací funkce (0 až 3)
- swap – prohození vykreslovací a zobrazovací paměti po dokončení vykreslování (0 nebo 1)
- copy – postupné vykreslování (0 nebo 1)

`clear_color(r,g,b);`

Vykreslení jedné barvy přes celou obrazovku.

- r – červená složka (0 až 255)
- g – zelená složka (0 až 255)
- b – modrá složka (0 až 255)

`delay(seconds);`

Pozastavení na daný počet sekund.

`bmp_config(enable_scaling);`

Je-li nastaveno na „1“, povoluje se škálování bitmapy – dělí se šířka obrázku dvěma. Tímto se částečně kompenzuje nesprávná šířka pixelu zobrazovací jednotky. Pokud je šířka bitmapy upravena předem, nastavujeme na „0“.

`processing_config(enable_dithering);`

Povolení ditheringu. (0 nebo 1)

`set_color_mode(mode);`

Nastavení barevného režimu. (RGB nebo BW, bez uvozovek)

`swap_buffers();`

Prohození vykreslovací a zobrazovací paměti.

`copy_buffers(direction);`

Kopírování z jedné paměti do druhé.

„direction“ (směr):

- 1 – z vykreslovací paměti do zobrazovací
- 0 – ze zobrazovací paměti do vykreslovací

`repeat_from_here();`

Označení pozice, odkud se má kód opakovat při zavolání funkce „repeat“.

`repeat();`

Opakovat.

3. Knihovny vykreslovacího mikroprocesoru

Některé knihovny obsahují globální proměnné pro své vnitřní výpočty. Většina těchto proměnných tu není uvedena.

3.1. BMP.h

```
unsigned char bmp_2to1_scaling;
```

Je-li nastaveno na „1“, povoluje se škálování bitmapy – dělí se šířka obrázku dvěma. Tímto se částečně kompenzuje nesprávná šířka pixelu zobrazovací jednotky. Pokud je šířka bitmapy upravena předem, nastavujeme bmp_2to1_scaling na „0“.

```
int BMP_decode_start();
```

Dekóduje hlavičku bitmapy. Podporuje pouze bitmapy s 24bitovou barevnou hloubkou a pouze bitmapy se standardní hlavičkou. Pokud navrací „1“, jde o nepodporovanou bitmapu, je-li vše v pořádku, navrací „0“.

```
vec3 BMP_next_pixel();
```

Navrací další pixel.

```
void BMP_next_y();
```

Posouvá se na další Y souřadnici. Souřadnice jsou čteny zespod obrázku nahoru. (adresy od „výška - 1“ do „0“)

```
void BMP_center_y(int input_y);
```

Centrování bitmapy. Volá se před vykreslením pixelu, jednou na každé lince.

```
void BMP_center_x(int input_x);
```

Centrování bitmapy. Volá se před vykreslením pixelu.

```
void BMP_render(unsigned long starting_cluster);
```

Vykreslení bitmapy.

- starting_cluster – cluster BMP souboru, který chceme vykreslit.

3.2. cmd_parser.h

```
void process_command(char *cmd, char *arg);
```

Zpracuje příkaz.

- cmd – příkaz (string)
- arg – argument (string)

```
void load_commands(char *file_path);
```

Načítá soubor s příkazy určenými k interpretaci.

- file_path – cesta k souboru (např. „slozka/soubor.txt“)

```
unsigned char cmd_compare(char* cmd, char* text);
```

Porovnává příkaz (string) s textem.

- cmd – příkaz (string)
- text – text (string)

```
int get_argument_i(char *arg, unsigned char arg_number);
```

Vytahuje z argumentu (string) argument s číslem „arg_number“ a navrací jej.

- arg – argument (string)
- arg_number – číslo požadovaného argumentu

```
void get_argument_str(char *arg);
```

Vytahuje string z argumentu (string) a navrací jej zpět do argumentu.

- arg – argument (string)

```
void get_argument_enum(char *arg);
```

Vytahuje enum z argumentu (string) a navrací jej zpět do argumentu.

- arg – argument (string)

3.3. color_process.h

```
unsigned char dither_enable;
```

Povolení ditheringu (0, 1).

```
PROGMEM unsigned const char threshold_matrix3b[4][4];
```

Prahová mapa 4x4 pro ordered dithering, pronásobená velikostí kvantizační hladiny o velikosti 32.

```
PROGMEM unsigned const char threshold_matrix2b[4][4];
```

Prahová mapa 4x4 pro ordered dithering, pronásobená velikostí kvantizační hladiny o velikosti 64.

```
vec3 dither_rgb(vec3 pixel, int x, int y);
```

Převede pomocí ordered ditheringu pixel s 24bitovou hloubkou na pixel s 8bitovou hloubkou (kompatibilní s formátem BBGGGRRR).

```
vec3 rgb_to_bw(vec3 pixel24b);
```

Pomocí funkce relativního jasu převede barevný pixel na černobílý.

```
unsigned char rgb_to_bw_value(vec3 pixel24b);
```

Pomocí funkce relativního jasu převede barevný pixel s 24bitovou hloubkou na hodnotu o 256 odstínech šedi.

```
vec3 truncate_rgb(vec3 pixel);
```

Oseká pixel s 24bitovou hloubkou na pixel s 8bitovou hloubkou (kompatibilní s formátem BBGGGRRR).

```
unsigned char compose_rgb8(vec3 pixel);
```

Navrací formát BBGGGRRR.

- pixel – pixel s 8bitovou hloubkou (kompatibilní s formátem BBGGGRRR)

```
unsigned char compose_rgb24(vec3 pixel);
```

Oseká pixel s 24bitovou hloubkou na pixel s 8bitovou hloubkou (kompatibilní s formátem BBGGGRRR) a navrací formát BBGGGRRR.

- pixel - pixel s 24bitovou hloubkou

```
vec3 decompose_rgb8(unsigned char bbgggrrr);
```

Rozložení formátu BBGGGRRR na tři složky.

- bbgggrrr – vstupní hodnota formátu BBGGGRRR

3.4. commands.h

Obsahuje seznam příkazů pro funkci „driver_command“, která je deklarovaná v knihovně „UART.h“.

3.4.1. Žádosti

`COPY_REQUEST_CMD(n)`

Kopírování n-té čtveřice linek. „n“ je v rozmezí od 0 do 119.

`VIDEO_DISABLE`

Zakázání zobrazování obrazu.

`VIDEO_ENABLE`

Povolení zobrazování obrazu.

`SWAP_STATE_CMD(state)`

Prohodit paměti do konkrétního stavu. „state“ je buď „0“ nebo „1“.

`COPY_DIR_CMD(dir)`

Směr kopírování. „dir“ může být copy_to_video (kopírování z vykreslovací paměti do zobrazovací) nebo copy_to_data (kopírování ze zobrazovací paměti do vykreslovací).

`SWAP_REQUEST_CMD`

Prohození pamětí.

`DRIVER_READY_REQ_CMD`

Zjišťuje, je-li zobrazovací mikroprocesor připraven.

3.4.2. Odpovědi

`DRIVER_DONE_CMD`

Příkaz byl dokončen v pořádku.

`DRIVER_READY_RESP_CMD`

Zobrazovací mikroprocesor je připraven.

`DRIVER_TRANSMIT_ERROR`

Chyba v přenosu.

`DRIVER_ILLEGAL_CMD`

Nepodporovaný příkaz.

3.5. draw_test.h

```
unsigned char draw0(int input_x,int input_y);
```

Vykreslovací funkce „srdce“.

```
unsigned char draw1(int input_x,int input_y);
```

Vykreslovací funkce „základní fraktál“.

```
unsigned char draw2(int input_x,int input_y);
```

Vykreslovací funkce.

```
unsigned char draw3(int input_x,int input_y);
```

Vykreslovací funkce.

3.6. drawing.h

```
unsigned long time;
```

Tato hodnota je určená pro vlastní výpočty. Při startu programu je inicializována na „0“. Funkce „render“ navyšuje tuto hodnotu o 1.

```
void clear_color(unsigned char color, unsigned char swap);
```

Vykreslení jedné barvy do paměti.

- color – barva ve formátu BBGGGRRR
- swap – prohození bufferů po dokončení vykreslování barvy (0, 1)

```
void render_loading_bar(unsigned int input_y);
```

Vykreslení loading baru.

- input_y – aktuální adresa y

```
void render(    unsigned char (*compute_pixel)(int,int),
                unsigned char swap,
                unsigned char copy    );
```

- (*compute_pixel)(int,int) – vstupní vykreslovací funkce (např. draw_pal)
- swap – prohození bufferů po dokončení vykreslování barvy (0, 1)
- copy – postupné vykreslování

```
unsigned char draw_pal(int input_x,int input_y);
```

Vykreslovací funkce pro vykreslení palety.

```
unsigned char draw_hsv_pal(int input_x,int input_y);
```

Vykreslovací funkce pro vykreslení HSV palety.

```
void delay(unsigned int seconds);
```

Pozastavení programu.

- seconds – doba pozastavení v sekundách

3.7. FAT.h

```
void FAT_init();
```

Inicializace FATu.

```
unsigned long cluster_to_lba(unsigned long cluster);
```

Převádí cluster na adresu LBA a tu navrací.

```
int FAT_get_new_cluster();
```

Posune čtení se na následující cluster. Pokud je aktuální cluster poslední v řetězci clusterů, navrací „1“, jinak „0“.

```
void FAT_read_start(unsigned long cluster);
```

Započne čtení na daném clusteru.

```
unsigned char FAT_read_next();
```

Navrací další 8bitové nezáporné celé číslo.

```
unsigned int FAT_next_ui();
```

Navrací další 16bitové nezáporné celé číslo.

```
unsigned long FAT_next_ul();
```

Navrací další 32bitové nezáporné celé číslo.

```
void FAT_read_skip(int bytes_to_skip);
```

Přeskočení několika bytů.

- bytes_to_skip – počet bytů k přeskočení

```
unsigned long FAT_find_entry_cluster( char *name ,
     unsigned char is_directory);
```

Navrací cluster hledaného souborového zápisu. Není-li zápis nalezen, vrací „0“.

- name – jméno hledaného souborového zápisu (např. „soubor.txt“)
- is_directory – jedná se o složku (1, 0)

```
void FAT_open_directory(    char *directory_path,
                            unsigned char name_is_file_path );
```

Otevírá složku.

- directory_path – cesta ke složce (např. „slozka/podslozka1/podslozka2“)
- name_is_file_path – cesta ke složce obsahuje na konci zápisu soubor (1, 0)

```
unsigned long FAT_find_file_cluster(char *file_path);
```

Navrací cluster hledaného souboru.

- file_path – cesta k souboru (např. „slozka/podslozka/soubor.txt“)

```
unsigned long FAT_file_by_extension_cluster(
    char* directory_name, char *extension);
```

Otevírá soubor s danou příponou (přípona není citlivá na velikost písmen). Při vícenásobném volání otevírá soubory postupně za sebou.

- directory_name – cesta ke složce (např. „slozka/podslozka1/podslozka2“)
- extension – přípona souborů (pouze třímístné, např. „bmp“, „txt“)

3.8. RAM.h

```
unsigned char const PROGMEM translate_x_data[33];
```

Konstantní data využitá pro přeložení standartního adresování „od 0 do n“ na „posloupnost rychlého sekvenčního čtení“.

```
void set_Y(unsigned int _value);
```

Nastavení adresy Y (standardní adresování „od 0 do n“).

- _value – vstupní hodnota adresy Y

```
void set_X(unsigned char _value);
```

Nastavení adresy X (standardní adresování „od 0 do n“).

- _value – vstupní hodnota adresy X

```
unsigned char translate_x(unsigned char x);
```

Přeložení adresy standardního adresování „od 0 do n“ na adresu „posloupnosti rychlého sekvenčního čtení“. Tato funkce je používána interně.

```
void inc_X();
```

Zvětšení hodnoty adresy X o 1. Tato funkce je optimalizována pro rychlost. Pokud je to možné, je doporučeno používat tuto funkci místo funkce set_X.

```
void reset_X();
```

Vynuluje adresu X. Tato funkce je optimalizována pro rychlost. Pokud je to možné, je doporučeno používat tuto funkci místo funkce set_X.

```
unsigned char RAM_read();
```

Vrací hodnotu z externí RAM, z aktuálně nastavené adresy.

```
void RAM_write(unsigned char data);
```

Zapisuje do externí RAM na aktuálně nastavenou adresu.

- data – vstupní data

3.9. SD.h

```
void SD_set_speed(unsigned char speed);
```

Nastavení rychlosti přenosu SD karty.

Hodnota proměnné „speed“	Rychlost přenosu [kbps]
0	125
1	250
2	500
3	1000
4	2000
5	4000
6	8000

Tab. 3 – 1: Funkce „SD_set_speed“ – popis parametru „speed“

```
int SD_init_basic();
```

Základní inicializace SD karty.

```
int SD_init();
```

Inicializace SD karty. Aktuálně wrapper pro SD_init_basic.

```
unsigned int crc16(unsigned int  old_crc, unsigned char data);
```

Jde o CRC-16-CCITT. Navrací novou hodnotu CRC.

- old_crc – stará hodnota CRC
- data – vstupní byte do CRC výpočtu

```
unsigned char crc7(unsigned char *message);
```

Navrací poslední byte SD příkazu (CRC posunuté o bit doleva a stop bit).

- message – SD příkaz, pro který se počítá CRC-7

```
void SD_command(unsigned char cmd, unsigned long arg);
```

Posílá SD příkaz.

- cmd – číslo příkazu
- arg - argument

```
unsigned char SD_command_R1(unsigned char cmd,
    unsigned long arg, unsigned char expected_response);
```

Posílá SD příkaz a čeká na odpověď R1.

- cmd – číslo příkazu
- arg – argument
- expected_response – požadovaná odpověď

```
void SD_read_start(unsigned long address, unsigned long blocks);
```

Započne čtení.

- address – adresa, na které se má začít číst
- blocks – počet bloků, které se budou číst

```
void SD_read_stop();
```

Zastavení přenosu dat. Není nutné používat, volá se automaticky dle potřeby.

```
unsigned char SD_read_next();
```

Navrací další 8bitové nezáporné celé číslo.

```
unsigned int SD_next_ui();
```

Navrací další 16bitové nezáporné celé číslo.

```
unsigned long SD_next_ul();
```

Navrací další 32bitové nezáporné celé číslo.

```
void SD_read_skip(int bytes_to_skip);
```

Přeskočení několika bytů.

- bytes_to_skip – počet bytů k přeskočení

3.10. settings.h

Obsahuje softwarová a hardwarová nastavení. Jejich funkce je dle názvu zřejmá, proto zde nejsou nastavitelné parametry popsány.

```
void set_color_mode(unsigned char color_mode);
```

Nastavuje barevný režim.

- color_mode - typ barevného režimu (RGB, BW).

```
unsigned char get_color_mode();
```

Vrací aktuálně nastavený barevný režim (RGB, BW).

```
unsigned char informative_color();
```

Vrací barvu pro využití k informativním účelům (načítání, chyby, …) ve formátu BBGGGRRR. Vrácená barva závisí na nastaveném barevném režimu.

3.11. SPI.h

```
void SPI_init();
```

Inicializace SPI.

```
void SPI_set_speed(int speed);
```

Nastavení rychlosti SPI.

Hodnota proměnné „speed“	Rychlost přenosu [kbps]
0	125
1	250
2	500
3	1000
4	2000
5	4000
6	8000

Tab. 3 – 2: Funkce „SPI_set_speed“ – popis parametru „speed“

```
unsigned char SPI_transfer(unsigned char data_in);
```

SPI přenos. Navrací příchozí data.

- data_in – data pro odeslání

3.12. touch_button.h

Tato knihovna není hardwarově podporována.

3.13. UART.h

```
void UART_init();
```

Inicializace UARTu

```
void driver_init();
```

Vyčkání na inicializaci zobrazovacího mikroprocesoru.

```
void driver_command(unsigned char command);
```

Poslání příkazu zobrazovacímu mikroprocesoru. Všechny možné příkazy jsou sepsány v souboru „commands.h“.

- command – příkaz, který se pošle

3.14. vecmath.h

```
typedef struct
{
   unsigned char r;
   unsigned char g;
   unsigned char b;
}vec3;
```

Definuje typ vec3 – vektor o třech 8bitových složkách.

```
vec3 add(vec3 a, vec3 b);
```

Součet dvou vektorů „a“ a „b“.

```
vec3 add_value(vec3 a, unsigned char v);
```

Součet vektoru „a“ a hodnoty „v“.

```
vec3 mul_value(vec3 a, float v);
```

Násobení vektoru „a“ hodnotou „v“.

```
vec3 average(vec3 a, vec3 b);
```

Zprůměrování vektorů „a“ a „b“.

```
vec3 mix(vec3 a, vec3 b, unsigned char alpha);
```

Smíchání vektorů „a“ a „b“ v poměru daném hodnotou „alpha“:

Hodnota „alpha“	Vektor „a“	Vektor „b“
0	100 %	0 %
127	50 %	50 %
255	0 %	100 %

Tab. 3 – 3: Funkce „mix“ – popis parametru „alpha“

```
unsigned char clamp(int x);
```

Osekání hodnoty „x“ do rozmezí od 0 do 255.

```
float absolute(float x);
```

Navrací absolutní hodnotu „x“.

Seznam obrázků, grafů a tabulek