LCD přes I2C

Poznámka o programování

Mnoho lidí „programuje“ způsobem, že si stahují hotové řešení z knihoven:

```
#include "MyCoolLCD.h"
MyCoolLCD lcd(0x37);

void setup() {
  lcd.begin();
  Serial.begin(9600);
}

void loop() {
  if(Serial.available()) {
    byte chr = Serial.read();
    lcd.print(chr);
  }
}
```

V podstatě je to volání programu `MyCoolLCD` s parametrem. Pokud by pak programátor měl doplnit ošetření klávesy backspace a šipky, musel by prostudovat knihovnu `MyCoolLCD`.
V reálu však programátoři googlí a hledají pokud možno příklady, které dělají, co potřebují. Programují pak úpravami zkopírovaného kódu. Vyzkoušejí, jestli to funguje, zkusí otestovat ještě pár vstupů a prohlásí problém za vyřešený. Programátor pak nikdy neví, co se stane za podmínek, které netestoval: Co když se zakáže přerušení? Bude to fungovat i na mikrokontroleru s pomalejším oscilátorem? Pokud se objeví chyba, pak opět Google a úpravy kódu. V případě dlouhodobě udržovaného projektu je vhodné najít si čas a vše prostudovat do technických detailů. Díky tomu pak programátor může rozumět příčinám chyb. Člověk si ušetří spoustu práce, pokud odmítne převzít kód po programátorovi, který komentuje stylem „*tady jsem musel trochu zvětšit delay a tady jsem musel znovu vytvořit objekt*“. To je programování záplatováním, které lajdáci obhajují frází, že není vhodné znovuvynalézat kolo. To platí pouze tehdy, lze-li problém vyřešit změnou parametrů řádně otestovaného řešení, ale ne při vývoji nových funkcí.

Analýza I2C displeje

Vygooglíme, že displej je od společnosti Hitachi ze série HD44780, stáhneme si jeho technickou dokumentaci (*datasheet*). Pohledem zjistíme, že I2C převodník z druhé strany je PCF8574, jehož datasheet si také stáhneme. Vyhradíme si dva ničím nerušené dny a pustíme se do čtení těchto magických knih, které z nás udělají experty na tyto Hitachi displeje.
Zjistíme, že

- Displej funguje s napájením 2.7 až 5.5 V, můžeme ho tedy bez obav použít pro Arduino i Raspberry

- Displej obsahuje 1.24 kB CGROM (character generator ROM), kde jsou uloženy bitmapy znaků (Hitachi vyrábí CGROM s „evropským“ a japonským fontem, je však možné se domluvit i na sériové výrobě displejů s vlastním fontem). Dále displej obsahuje 64 B CGRAM pro definici vlastních znaků. Náš displej (pohledem na matrici) obsahuje font 5×8 bodů. Znak zabírá 8B v paměti, je tedy k dispozici 155 znaků z CGROM a 8 uživatelsky definovaných znaků z CGRAM. Data znaku se posílají sérioparalelně po pěti sloupcích do konvertoru, kam jde též vstup s údaji o kurzoru (který může blikat) a paralelně po 40žilové sběrnici (!) je posílán do posuvného registru, který data sériově posílá přes pin D a paralelně na piny SEG1 až SEG40.
- Displej dále obsahuje DDRAM (data display RAM) 80 B (20×4). Adresa pozice se posílá po 7žilové sběrnici (2^7=128>80). Data se do obou RAM posílají přes datový registr (DR), se kterým může komunikovat mikrokontroler paralelně po vodičích DB0 až DB7.
- Řídící sběrnice má vyvedené piny
    - RS (register select: 0=instrukce, 1=data). Instrukce se načítají do instrukčního registru (IR)
    - R/W (read, not write: 0=zápis, 1=čtení). Na I2C převodníku jsme omezeni na zápis, jelikož pin pro přerušení čtení není vyveden.
    - E (enable: 0=ignorování, 1=zpracování signálu). V průběhu vystavování pinů je třeba dát E na 0, a poté krátkým pulzem (min 450ns) dát signál pro zpracování
    - napájení a zem pro displej (až 11 V) a čip (až 5.5 V)
- Na desce displeje nejsou vyvedeny piny CL1, CL2, M, D, COMn a SEGn použitelné pro analýzu
- Z příznakového registru má čip pouze BF bit (busy flag) indikující probíhající operaci. Tento si nemůžeme přes I2C převodník přečíst :-(Adresa instrukcí (CG i DD segmentu) se zapisuje do AC registru, který přes I2C také nelze číst.
- Čip počítá pouze s 1 či 2 řádkovým displejem: 1řádkový ma adresy DDRAM 0-39, dvouřádkový 1-79. Displej 2×20 může být tedy implementován i jako jednořádkový a podle toho se může lišit zadávání souřadnic (fyzicky máme 4řádkový displej)

Abstraktní funkce

Vytvoříme třídu `HD44780` pro implementaci komunikace s LCD. Uděláme ji nezávislou na způsobu připojení (I2C či paralelní), což zajistíme následujícími ryze virtuálními metodami pro práci se sběrnicemi:

```
virtual void control(bool rs, bool rw, bool en)=0;  // řídící sběrnice
virtual void command(byte com, bool rs, bool rw)=0; // datová sběrnice
```

Krátká instrukce trvá nejdéle 37 mikrosekund, můžeme tedy vytvořit metodu pro synchronní zpracování instrukcí:

```
void instruction(byte com, bool rs=false, bool rw=false) {
  command(com, rs, rw);
  delayMicroseconds(37);
}
```

Řídící signály se vystavují následovně

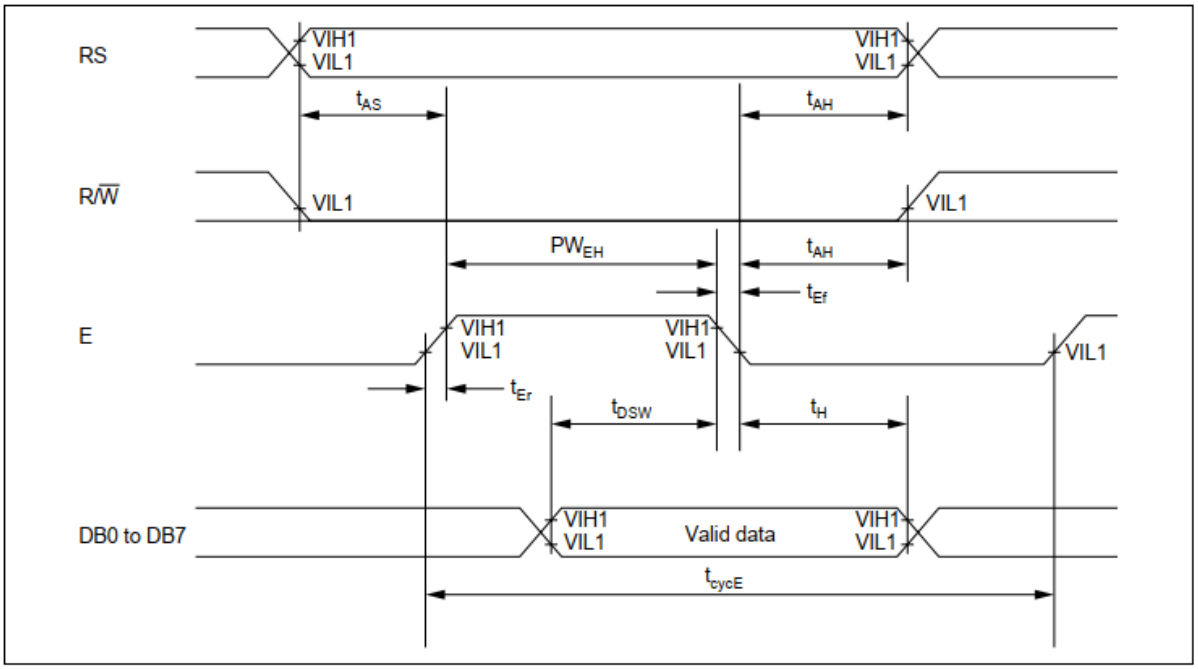


Figure 25 Write Operation

Z toho plyne, že signály RS a R/W musí být vystaveny o dobu t_{AS} před signálem E, jehož délka musí být PW_{EH}. Data musí být platná po dobu $t_{DSW}+t_H$ a signály RS a R/W musí po celou dobu zůstat nezměněné. Tyto údaje jsou popsány v datasheetu, jsou minimální a jsou všechny v řádu desítek nanosekund. Můžeme tedy napsat metodu pro generování E pulzu:

```
void en_pulse(bool rs, bool rw) {
  delayMicroseconds(1);   // > tAS
  control(rs, rw, true);
  delayMicroseconds(1);   // > PWEH, > tDSW+tH, > tAH
  control(rs, rw, false);
}
```

Pokud před tímto pulzem vystavíme data, RS a R/W, splníme časové požadavky diagramu.

Instrukce displeje

Instruction	Code RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (max) (when f_{cp} or f_{osc} is 270 kHz)
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.	
Return home	0	0	0	0	0	0	0	0	1	—	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 µs
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).	37 µs
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	—	—	Moves cursor and shifts display without changing DDRAM contents.	37 µs
Function set	0	0	0	0	1	DL	N	F	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).	37 µs
Set CGRAM address	0	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	Sets CGRAM address. CGRAM data is sent and received after this setting.	37 µs
Set DDRAM address	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	Sets DDRAM address. DDRAM data is sent and received after this setting.	37 µs
Read busy flag & address	0	1	BF	AC	AC	AC	AC	AC	AC	AC	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0 µs
Write data to CG or DDRAM	1	0	Write data								Writes data into DDRAM or CGRAM.	37 µs t_{ADD} = 4 µs*
Read data from CG or DDRAM	1	1	Read data								Reads data from DDRAM or CGRAM.	37 µs t_{ADD} = 4 µs*
	I/D = 1: Increment I/D = 0: Decrement S = 1: Accompanies display shift S/C = 1: Display shift S/C = 0: Cursor move R/L = 1: Shift to the right R/L = 0: Shift to the left DL = 1: 8 bits, DL = 0: 4 bits N = 1: 2 lines, N = 0: 1 line F = 1: 5 × 10 dots, F = 0: 5 × 8 dots BF = 1: Internally operating BF = 0: Instructions acceptable										DDRAM: Display data RAM CGRAM: Character generator RAM ACG: CGRAM address ADD: DDRAM address (corresponds to cursor address) AC: Address counter used for both DD and CGRAM addresses	Execution time changes when frequency changes Example: When f_{cp} or f_{osc} is 250 kHz, $37\ \mu s \times \frac{270}{250} = 40\ \mu s$

Instrukce pro čtení nelze s I2C převodníkem implementovat. Můžeme napsat kód pro dlouhé bezparametrické instrukce `clear()` a `home()`:

```
void home() {
  command(1<<1);
  delayMicroseconds(1520);
}
```

Obdobně můžeme napsat kód pro krátké parametrické instrukce:

```
void display_cursor_blink(bool d=true, bool c=false, bool b=false) {
  byte data = (1<<3) | (d<<2) | (c<<1) | b;
  instruction(data);
}
```

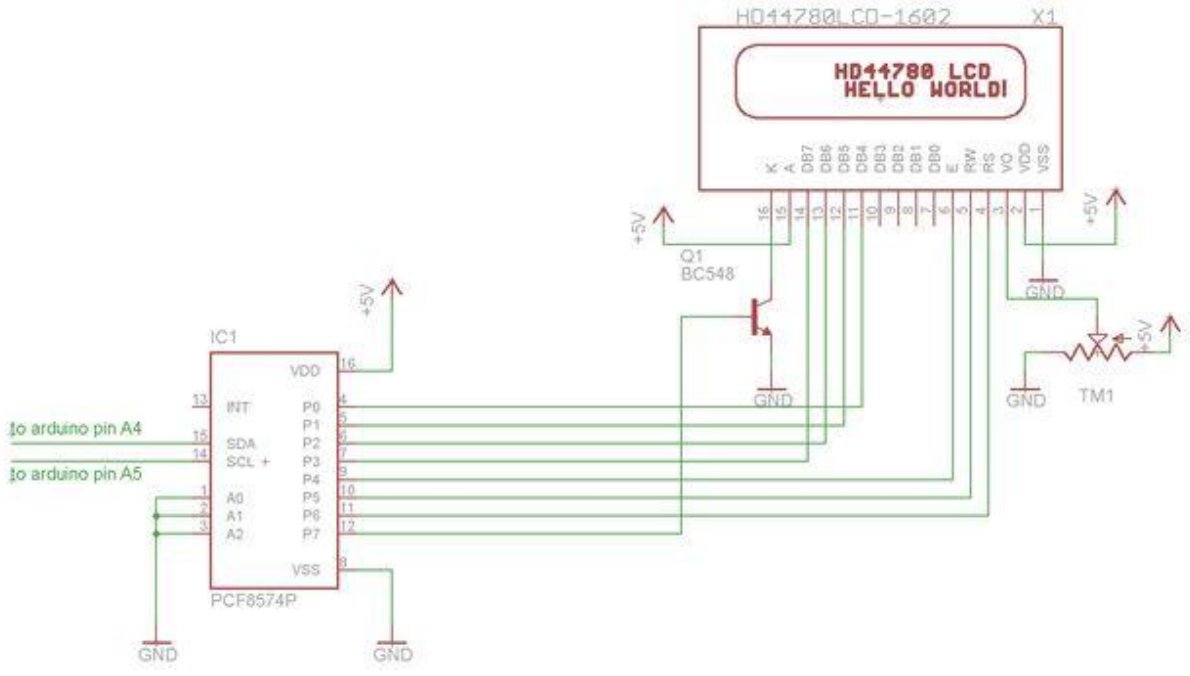
Podobnou technikou, kdy z virtuálních metod děláme posloupnost abstraktních kroků, funguje i třída `Print`: zde jsou implementovány všechny varianty, jaké lze volat u `Serial.print`, pouze varianta pro tisk znaku je ryze virtuální. Pokud ji implementujeme, zdědíme tak i všechny ostatní metody pro výpis:

```
size_t write(uint8_t chr) {
  if(!dd_selected) return 0;
  instruction(chr, true);
  return 1;
}
```

Metoda má vrátit 1, pokud se znak podařilo napsat. V našem případě bychom měli znak zapsat pouze tehdy, pracujeme-li s DDRAM. Typ paměti (CGRAM nebo DDRAM) můžeme vybrat instrukcí (viz tabulka výše). Povolíme tedy zápis metodou `write`, pouze pokud byla zvolena DDRAM (k tomu slouží privátní proměnná `dd_selected`).

I2C převodník a 4bitový mód

I2C převodník PCF8574 má 8 datových pinů, nemůžeme je ale použít všechny, protože displej potřebuje ještě řídící signály RS, R/W a E. Čip displeje však instrukcí `function_set` umožňuje přepnout do 4bitového módu, kde používají pouze 4 datové vodiče, které přenesou byte ve dvou cyklech: v prvním se přenese vyšší a v druhém nižší nibble. Připojení vypadá takto:



Obrázek 1: I2C převodník a displej, zdroj: https://www.microchip.com/forums/m830041-p2.aspx

Volný bit P7 je použit k podsvícení displeje, které tak lze také ovládat softwarově. V našem případě (pohledem na vedení vodičů na desce převodníku) však zjistíme, že jsou datové nibble prohozené: P4-P7 jsou datové, P0-P3 řídící sběrnice.

Napíšeme si tedy funkci pro přenos nibble po I2C sběrnici:

```
void writeI2C(byte nibble, bool rs=false, bool en=false) {
  data = nibble;    // uloženo pro následné poslání řídících signálů
  if(!addr) return; // I2C adresa z konstruktoru
  byte packet = nibble<<4;
  packet|= (rs<<0) | (en<<2) | (light<<3);
  Wire.beginTransmission(addr);
  Wire.write(packet);
  Wire.endTransmission();
}
```

Posílaná data si uložíme do bufferu (při zaslání E pulzu musí být nezměněná). `light` je veřejná proměnná. Pokud ji změníme, displej se rozsvítí/zhasne při následujícím příkazu. Implementujte ji v souladu se zapouzdřením jako property (při změně pošlete instrukci NOP – samé nuly). Poté můžeme např. zablikat displejem (jako při budíku).

Virtuální metodu `control` pak můžeme implementovat takto:

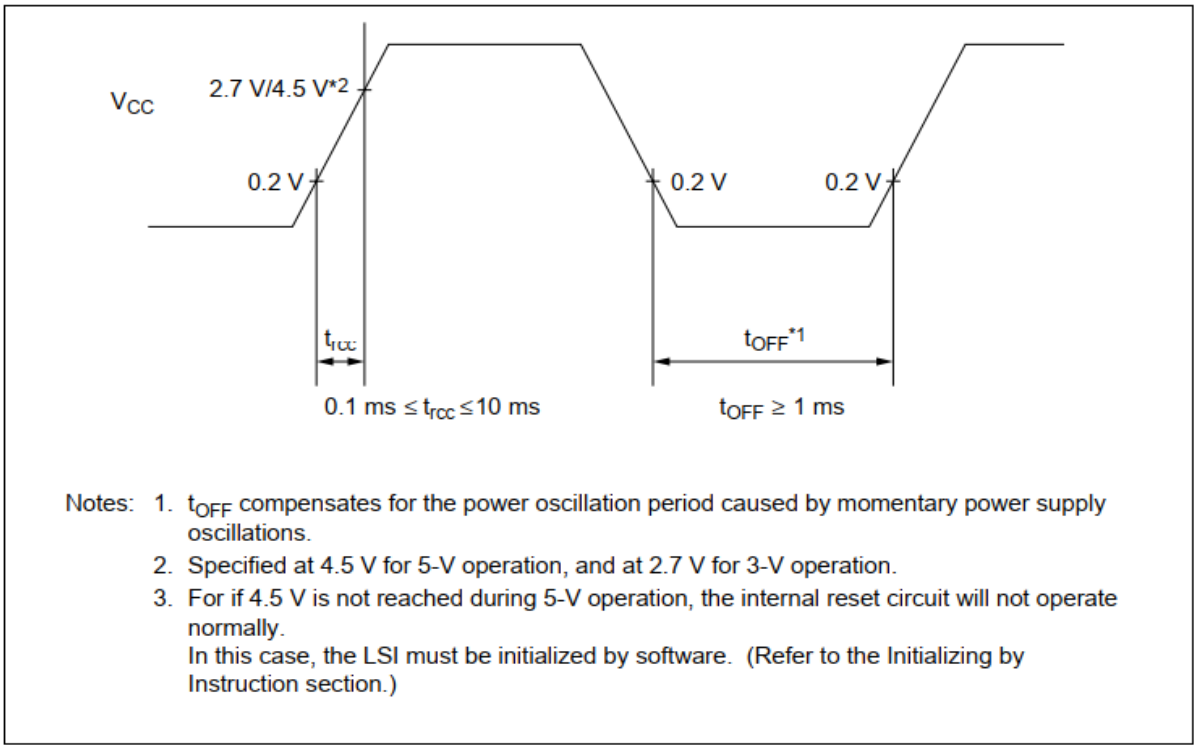
```
void control(bool rs, bool rw, bool en) {
  rw = rw; // unused
  writeI2C(data, rs, en);
}
```

Tato metoda se volá 2× v `en_pulse` (vzestupná a sestupná hrana). Proto můžeme virtuální metodu `command` implementovat takto:

```
void command(byte com, bool rs=false, bool rw=false) {
  byte hiByte = com >> 4;
  byte loByte = com & 0b1111;
  data = hiByte; en_pulse(rs, rw); // poslání horního nibble s E pulsem
  data = loByte; en_pulse(rs, rw); // poslání dolního nibble s E pulsem
}
```

Ještě je zapotřebí vyřešit inicializaci displeje, který se po restartu zapne do 8bitového režimu, a tedy první instrukci `function_set` pro přepnutí do 4bitového režimu musíme provést v 8bitovém módu. Dolní 4 bity sice nemáme zapojené, ale nejsou zapotřebí. Poté zavoláme `function_set` ještě jednou: tentokráte ve 4bitovém módu, kde pomocí dolních 4 bitů nastavíme dvouřádkový displej. Poté už počet řádků nelze měnit až do resetu.

Tuto krátkou sekvenci můžeme provést, pokud se správně provedl HW reset, jehož podmínky jsou tyto:



Notes:
1. t_{OFF} compensates for the power oscillation period caused by momentary power supply oscillations.
2. Specified at 4.5 V for 5-V operation, and at 2.7 V for 3-V operation.
3. For if 4.5 V is not reached during 5-V operation, the internal reset circuit will not operate normally.
   In this case, the LSI must be initialized by software. (Refer to the Initializing by Instruction section.)

Figure 28 Internal Power Supply Reset

Pokud tyto podmínky nejsou dodrženy, displej nebude pracovat správně a je nutno provést softwarový reset popsaný v diagramu č. 23 na stránce 45 v manuálu. V kódu:

```
void set_4bit() {
  writeI2C(0x00); // klidový stav sběrnice
  delay(50);      // ustálení napětí
  data = 0x03; en_pulse(false, false); // function_set v 8bitovém módu
  delayMicroseconds(5000);
  data = 0x03; en_pulse(false, false); // function_set v 8bitovém módu
  delayMicroseconds(150);
  data = 0x03; en_pulse(false, false); // function_set v 8bitovém módu
  delayMicroseconds(37);
  data = 0x02; en_pulse(false, false); // function_set v 8bitovém módu
  delayMicroseconds(37);
}
```

Po nevydařeném HW resetu může být displej v náhodném stavu, metoda `begin` pro inicializaci by měla vypadat takto:

```
void begin(int cols, int rows) {
  this->rows = rows;
  this->cols = cols;
  set_4bit();
  function_set(false, rows>0, false);
  clear();
  display_cursor_blink(true, false, false);
  entry_mode_set(true, false);
}
```

HW potřebujeme sloupce pouze pro správnou funkci jezdícího textu, SW se hodí řádky a sloupce k pohodlnému nastavení kurzoru přes souřadnice:

```
void gotoxy(int X, int Y) {
  if(X<0) X = cols+X;
  if(Y<0) Y = rows+Y;
  byte pos = (byte)(Y*cols+rows);
  set_ddram_addr(pos);
}
```

Offsety začátků řádků se však mohou lišit u 4řádkových displejů, viz analýza v úvodu. Pokud jsme nerolovali displejem (jezdícím textem), je efektivnější volání `gotoxy(0,0)` než `home()`.

Poslední věc nutná ke správnému zapojení je správně nastavená adresa I2C sběrnice. O tom podrobněji někdy jindy, zde si vystačíme s knihovnou `wire.h` a intuitivním předpokladem, že zařízení přenos ignoruje, pokud se adresa neshoduje s jeho. Adresa je daná většinou hardwarově v dokumentaci, na některých převodnících je možné ji změnit pomocí přepínačů.

Je-li na sběrnici pouze jedno zařízení, jeho adresu lze pomocí knihovny `wire.h` detekovat automaticky. Tento kód je vhodné umístit do konstruktoru LCDI2C:

```
Wire.begin(); // inicializace wire.h knihovny
for(int i=1; i<128; ++i) {
  Wire.beginTransmission(i);
  if(Wire.endTransmission()==0) {
    this->addr = i;
    break;
  }
}
```

Naši knihovnu pak můžeme volat např. takto

```
LCDI2C* lcd = new LCDI2C;
lcd->begin(20,4);
lcd->gotoxy(-4,1); // text do pravého dolního rohu
lcd->print("Ahoj"); // DDRAM AC je na 0 (levý horní roh)
```

Poznámka o designu LCD přes I2C

I2C je HW kompatibilní s datovou sběrnicí, řídící sběrnice potřebuje ale své piny nastavovat a měnit nezávisle a nehodí se proto k sériovému přenosu dat. Vhodnější by bylo zařadit posuvný registr jako cache za datovou a řídící sběrnici a přenášená data mezi nimi multiplexovat, případně řídící sběrnici ponechat paralelní (pouhé 3 vodiče navíc). Kód by pak mohl být jednodušší a bylo by možné využít 8bitový režim a čtecí instrukce displeje.

Příloha 1: třída HD44780

```
class HD44780 : public Print {
protected:
  bool dd_selected;
  HD44780() : dd_selected(true) { }

  void en_pulse(bool rs, bool rw) {
    delayMicroseconds(1);
    control(rs, rw, true);
    delayMicroseconds(1);
    control(rs, rw, false);
  }
  void instruction(byte com, bool rs=false, bool rw=false) {
    command(com, rs, rw);
    delayMicroseconds(37);
  }

  virtual void control(bool rs, bool rw, bool en)=0;
  virtual void command(byte com, bool rs, bool rw)=0;

  size_t write(uint8_t chr) {
    if(!dd_selected) return 0;
    instruction(chr, true);
    return 1;
  }

  // right = moves cursor position after writing char: true=right,
false=left
  // roll = rolls display after writing char to the "right"
  void entry_mode_set(bool right=true, bool roll=false) {
    byte data = (1<<2) | (right<<1) | roll;
    instruction(data);
  }

  // d = turn the display on
  // c = show cursor on display
  // b = blink cursor on display (409.6 ms)
  void display_cursor_blink(bool d=true, bool c=false, bool b=false) {
    byte data = (1<<3) | (d<<2) | (c<<1) | b;
    instruction(data);
  }

  // right = moves cursor position: true=right, false=left
  // roll = rolls display to the "right"
  void cursor_display_shift(bool right=true, bool roll=false) {
    byte data = (1<<4) | (roll<<3) | (right<<2);
    instruction(data);
  }

  // bits8 = 8bit mode
  // lines = multiline display
  // bigfont = 5x10 instead of 5x8 (if supported)
  void function_set(bool bits8=true, bool lines=true, bool bigfont=false) {
    byte data = (1<<5) | (bits8<<4) | (lines<<3) | (bigfont<<2);
    instruction(data);
  }
```

```
  void set_cgram_addr(byte addr) {
    byte data = (1<<6) | (addr & 0b111111);
    dd selected = false;
    instruction(data);
  }

  void set_ddram_addr(byte addr) {
    byte data = (1<<7) | (addr & 0b1111111);
    dd_selected = true;
    instruction(data);
  }

  void write_ram_value(byte data) {
    instruction(data, true);
  }

public:
  void clear() {
    command(1<<0);
    delayMicroseconds(1520); // long executing instruction
  }
  void home() {
    command(1<<1);
    delayMicroseconds(1520); // long executing instruction
  }

};
```

Příloha 2: třída LCDI2C

```
class LCDI2C : public HD44780 {
  void set_4bit() {
    writeI2C(0x00);
    delay(50);
    data = 0x03; en_pulse(false, false);
    delayMicroseconds(5000);
    data = 0x03; en pulse(false, false);
    delayMicroseconds(150);
    data = 0x03; en_pulse(false, false);
    delayMicroseconds(37);
    data = 0x02; en_pulse(false, false);
    delayMicroseconds(37);
  }
protected:
  byte rows, cols;
  byte addr, data;
  void command(byte com, bool rs=false, bool rw=false) {
    byte hiByte = com >> 4;
    byte loByte = com & 0b1111;
    data = hiByte; en_pulse(rs, rw);
    data = loByte; en_pulse(rs, rw);
  }
  void control(bool rs, bool rw, bool en) {
    rw = rw; // unused
    writeI2C(data, rs, en);
  }
  void writeI2C(byte nibble, bool rs=false, bool en=false) {
    data = nibble;
    if(!addr) return;
    if(en) Serial.println(nibble,BIN);
    byte packet = nibble<<4;
    packet|= (rs<<0) | (en<<2) | (light<<3);
    Wire.beginTransmission(addr);
    Wire.write(packet);
    Wire.endTransmission();
  }
public:
  bool light;
  LCDI2C(byte addr=0, bool light=true) : addr(addr), light(light) {
    Wire.begin();
    if(addr>0) return;
    // autodetect
    for(int i=0; i<128; ++i) {
      Wire.beginTransmission(i);
      if(Wire.endTransmission()==0) {
        this->addr = i;
        break;
      }
    }
  }
  void begin(int cols, int rows) {
    this->rows = rows;
    this->cols = cols;
    set_4bit();
    function_set(false, rows>0, false);
    clear();
    display_cursor_blink(true, false, false);
    entry_mode_set(true, false);
  }
```

```
  void gotoxy(int X, int Y) {
    if(X<0) X = cols+X;
    if(Y<0) Y = rows+Y;
    byte pos = (byte)(Y*cols+rows);
    set_ddram_addr(pos);
  }
};
```