

LCD shield pro Raspberry Pi B+/B

1. POPIS

Tento shield je kompatibilní s vývojovými kity Raspberry Pi B+/B.

Základní charakteristika:

- displej typu 1602 (16 znaků, 2 řádky)
- 3x indikační LED
- 5x tlačítko
- 7 volitelných barev podsvícení
- Potenciometr pro nastavení jasu
- Komunikace přes rozhraní I²C



2. SPECIFIKACE

Napájecí napětí	5 VDC	Počet znaků	16
Max. proud	60 mA	Počet řádků	2
Komunikační rozhraní	I ² C	Počet barev podsvícení	7
Kompatibilita HW	Raspberry Pi B+/B	Rozměry (mm)	82 x 58
Kompatibilita SW	Raspbian a podobné	Provozní teplota (°C)	0 až 50
Typ displeje	negativní LCD	Hmotnost (g)	102

3. ZAPOJENÍ

Tento shield nevyžaduje žádné externí zapojení, pouze vsuňte modul do vývojového kitu Raspberry.

00101
01001
00001

4. UKÁZKA PROGRAMU

Ukázka převzata z https://github.com/adafruit/Adafruit_Python_CharLCD.

```
# Copyright (c) 2014 Adafruit Industries
# Author: Tony DiCola
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
# THE SOFTWARE.
import time

import Adafruit_GPIO as GPIO
import Adafruit_GPIO.I2C as I2C
import Adafruit_GPIO.MCP230xx as MCP
import Adafruit_GPIO.PWM as PWM

# Commands
LCD_CLEARDISPLAY      = 0x01
LCD_RETURNHOME        = 0x02
LCD_ENTRYMODESET      = 0x04
LCD_DISPLAYCONTROL    = 0x08
LCD_CURSORSHIFT       = 0x10
LCD_FUNCTIONSET       = 0x20
LCD_SETCGRAMADDR      = 0x40
LCD_SETDDRAMADDR      = 0x80

# Entry flags
LCD_ENTRYRIGHT        = 0x00
LCD_ENTRYLEFT         = 0x02
LCD_ENTRYSHIFTINCREMENT = 0x01
LCD_ENTRYSHIFTDECREMENT = 0x00

# Control flags
LCD_DISPLAYON         = 0x04
LCD_DISPLAYOFF        = 0x00
LCD_CURSORON          = 0x02
LCD_CURSOROFF         = 0x00
LCD_BLINKON           = 0x01
LCD_BLINKOFF          = 0x00

# Move flags
LCD_DISPLAYMOVE       = 0x08
LCD_CURSORMOVE        = 0x00
LCD_MOVERIGHT         = 0x04
LCD_MOVELEFT          = 0x00

# Function set flags
LCD_8BITMODE          = 0x10
LCD_4BITMODE          = 0x00
LCD_2LINE              = 0x08
LCD_1LINE              = 0x00
LCD_5x10DOTS          = 0x04
LCD_5x8DOTS           = 0x00

# Offset for up to 4 rows.
```

```

LCD_ROW_OFFSETS          = (0x00, 0x40, 0x14, 0x54)

# Char LCD plate GPIO numbers.
LCD_PLATE_RS             = 15
LCD_PLATE_RW             = 14
LCD_PLATE_EN             = 13
LCD_PLATE_D4             = 12
LCD_PLATE_D5             = 11
LCD_PLATE_D6             = 10
LCD_PLATE_D7             = 9
LCD_PLATE_RED            = 6
LCD_PLATE_GREEN          = 7
LCD_PLATE_BLUE           = 8

# Char LCD plate button names.
SELECT                   = 0
RIGHT                    = 1
DOWN                     = 2
UP                       = 3
LEFT                     = 4

class Adafruit_CharLCD(object):
    """Class to represent and interact with an HD44780 character LCD display."""

    def __init__(self, rs, en, d4, d5, d6, d7, cols, lines, backlight=None,
                  invert_polarity=True,
                  enable_pwm=False,
                  gpio=GPIO.get_platform_gpio(),
                  pwm=PWM.get_platform_pwm(),
                  initial_backlight=1.0):
        """Initialize the LCD. RS, EN, and D4...D7 parameters should be the pins
        connected to the LCD RS, clock enable, and data line 4 through 7 connections.
        The LCD will be used in its 4-bit mode so these 6 lines are the only ones
        required to use the LCD. You must also pass in the number of columns and
        lines on the LCD.

        If you would like to control the backlight, pass in the pin connected to
        the backlight with the backlight parameter. The invert_polarity boolean
        controls if the backlight is one with a LOW signal or HIGH signal. The
        default invert_polarity value is True, i.e. the backlight is on with a
        LOW signal.

        You can enable PWM of the backlight pin to have finer control on the
        brightness. To enable PWM make sure your hardware supports PWM on the
        provided backlight pin and set enable_pwm to True (the default is False).
        The appropriate PWM library will be used depending on the platform, but
        you can provide an explicit one with the pwm parameter.

        The initial state of the backlight is ON, but you can set it to an
        explicit initial state with the initial_backlight parameter (0 is off,
        1 is on/full bright).

        You can optionally pass in an explicit GPIO class,
        for example if you want to use an MCP230xx GPIO extender. If you don't
        pass in a GPIO instance, the default GPIO for the running platform will
        be used.
        """
        # Save column and line state.
        self.cols = cols
        self.lines = lines
        # Save GPIO state and pin numbers.
        self._gpio = gpio
        self.rs = rs
        self.en = en
        self._d4 = d4
        self._d5 = d5
        self._d6 = d6
        self._d7 = d7
        # Save backlight state.
        self.backlight = backlight
        self._pwm_enabled = enable_pwm
        self._pwm = pwm
        self._blpol = not invert_polarity

```

```

# Setup all pins as outputs.
for pin in (rs, en, d4, d5, d6, d7):
    gpio.setup(pin, GPIO.OUT)
# Setup backlight.
if backlight is not None:
    if enable_pwm:
        pwm.start(backlight, self._pwm_duty_cycle(initial_backlight))
    else:
        gpio.setup(backlight, GPIO.OUT)
        gpio.output(backlight, self._blpol if initial_backlight else not self._blpol)
# Initialize the display.
self.write8(0x33)
self.write8(0x32)
# Initialize display control, function, and mode registers.
self.displaycontrol = LCD_DISPLAYON | LCD_CURSOROFF | LCD_BLINKOFF
self.displayfunction = LCD_4BITMODE | LCD_1LINE | LCD_2LINE | LCD_5x8DOTS
self.displaymode = LCD_ENTRYLEFT | LCD_ENTRYSHIFTDECREMENT
# Write registers.
self.write8(LCD_DISPLAYCONTROL | self.displaycontrol)
self.write8(LCD_FUNCTIONSET | self.displayfunction)
self.write8(LCD_ENTRYMODESET | self.displaymode) # set the entry mode
self.clear()

def home(self):
    """Move the cursor back to its home (first line and first column)."""
    self.write8(LCD_RETURNHOME) # set cursor position to zero
    self._delay_microseconds(3000) # this command takes a long time!

def clear(self):
    """Clear the LCD."""
    self.write8(LCD_CLEARDISPLAY) # command to clear display
    self._delay_microseconds(3000) # 3000 microsecond sleep, clearing the display takes a long time

def set_cursor(self, col, row):
    """Move the cursor to an explicit column and row position."""
    # Clamp row to the last row of the display.
    if row > self._lines:
        row = self._lines - 1
    # Set location.
    self.write8(LCD_SETDRAMADDR | (col + LCD_ROW_OFFSETS[row]))

def enable_display(self, enable):
    """Enable or disable the display. Set enable to True to enable."""
    if enable:
        self.displaycontrol |= LCD_DISPLAYON
    else:
        self.displaycontrol &= ~LCD_DISPLAYON
    self.write8(LCD_DISPLAYCONTROL | self.displaycontrol)

def show_cursor(self, show):
    """Show or hide the cursor. Cursor is shown if show is True."""
    if show:
        self.displaycontrol |= LCD_CURSORON
    else:
        self.displaycontrol &= ~LCD_CURSORON
    self.write8(LCD_DISPLAYCONTROL | self.displaycontrol)

def blink(self, blink):
    """Turn on or off cursor blinking. Set blink to True to enable blinking."""
    if blink:
        self.displaycontrol |= LCD_BLINKON
    else:
        self.displaycontrol &= ~LCD_BLINKON
    self.write8(LCD_DISPLAYCONTROL | self.displaycontrol)

def move_left(self):
    """Move display left one position."""
    self.write8(LCD_CURSORSHIFT | LCD_DISPLAYMOVE | LCD_MOVELEFT)

def move_right(self):
    """Move display right one position."""
    self.write8(LCD_CURSORSHIFT | LCD_DISPLAYMOVE | LCD_MOVERIGHT)

```

```

def set_left_to_right(self):
    """Set text direction left to right."""
    self.displaymode |= LCD_ENTRYLEFT
    self.write8(LCD_ENTRYMODESET | self.displaymode)

def set_right_to_left(self):
    """Set text direction right to left."""
    self.displaymode &= ~LCD_ENTRYLEFT
    self.write8(LCD_ENTRYMODESET | self.displaymode)

def autoscroll(self, autoscroll):
    """Autoscroll will 'right justify' text from the cursor if set True,
    otherwise it will 'left justify' the text.
    """
    if autoscroll:
        self.displaymode |= LCD_ENTRYSHIFTINCREMENT
    else:
        self.displaymode &= ~LCD_ENTRYSHIFTINCREMENT
    self.write8(LCD_ENTRYMODESET | self.displaymode)

def message(self, text):
    """Write text to display. Note that text can include newlines."""
    line = 0
    # Iterate through each character.
    for char in text:
        # Advance to next line if character is a new line.
        if char == '\n':
            line += 1
            # Move to left or right side depending on text direction.
            col = 0 if self.displaymode & LCD_ENTRYLEFT > 0 else self.cols-1
            self.set_cursor(col, line)
        # Write the character to the display.
        else:
            self.write8(ord(char), True)

def set_backlight(self, backlight):
    """Enable or disable the backlight. If PWM is not enabled (default), a
    non-zero backlight value will turn on the backlight and a zero value will
    turn it off. If PWM is enabled, backlight can be any value from 0.0 to
    1.0, with 1.0 being full intensity backlight.
    """
    if self.backlight is not None:
        if self._pwm_enabled:
            self.pwm.set_duty_cycle(self.backlight, self.pwm_duty_cycle(backlight))
        else:
            self.gpio.output(self.backlight, self.blpol if backlight else not self.blpol)

def write8(self, value, char_mode=False):
    """Write 8-bit value in character or data mode. Value should be an int
    value from 0-255, and char mode is True if character data or False if
    non-character data (default).
    """
    # One millisecond delay to prevent writing too quickly.
    self.delay microseconds(1000)
    # Set character / data bit.
    self.gpio.output(self._rs, char_mode)
    # Write upper 4 bits.
    self.gpio.output pins({ self.d4: ((value >> 4) & 1) > 0,
                            self.d5: ((value >> 5) & 1) > 0,
                            self.d6: ((value >> 6) & 1) > 0,
                            self.d7: ((value >> 7) & 1) > 0 })

    self._pulse_enable()
    # Write lower 4 bits.
    self.gpio.output pins({ self.d4: (value & 1) > 0,
                            self.d5: ((value >> 1) & 1) > 0,
                            self.d6: ((value >> 2) & 1) > 0,
                            self.d7: ((value >> 3) & 1) > 0 })

    self._pulse_enable()

def create_char(self, location, pattern):
    """Fill one of the first 8 CGRAM locations with custom characters.
    The location parameter should be between 0 and 7 and pattern should
    provide an array of 8 bytes containing the pattern. E.g. you can easily

```

```

design your custom character at http://www.quinapalus.com/hd44780udg.html
To show your custom character use eg. lcd.message('\x01')
"""
# only position 0..7 are allowed
location &= 0x7
self.write8(LCD_SETCGRAMADDR | (location << 3))
for i in range(8):
    self.write8(pattern[i], char mode=True)

def _delay_microseconds(self, microseconds):
    # Busy wait in loop because delays are generally very short (few microseconds).
    end = time.time() + (microseconds/1000000.0)
    while time.time() < end:
        pass

def _pulse_enable(self):
    # Pulse the clock enable line off, on, off to send command.
    self._gpio.output(self._en, False)
    self._delay_microseconds(1) # 1 microsecond pause - enable pulse must be > 450ns
    self._gpio.output(self._en, True)
    self._delay_microseconds(1) # 1 microsecond pause - enable pulse must be > 450ns
    self._gpio.output(self._en, False)
    self._delay_microseconds(1) # commands need > 37us to settle

def pwm_duty_cycle(self, intensity):
    # Convert intensity value of 0.0 to 1.0 to a duty cycle of 0.0 to 100.0
    intensity = 100.0*intensity
    # Invert polarity if required.
    if not self.blpol:
        intensity = 100.0-intensity
    return intensity

class Adafruit_RGBCharLCD(Adafruit_CharLCD):
    """Class to represent and interact with an HD44780 character LCD display with
    an RGB backlight."""

    def __init__(self, rs, en, d4, d5, d6, d7, cols, lines, red, green, blue,
                 gpio=GPIO.get_platform_gpio(),
                 invert_polarity=True,
                 enable_pwm=False,
                 pwm=PWM.get_platform_pwm(),
                 initial_color=(1.0, 1.0, 1.0)):
        """Initialize the LCD with RGB backlight. RS, EN, and D4...D7 parameters
        should be the pins connected to the LCD RS, clock enable, and data line
        4 through 7 connections. The LCD will be used in its 4-bit mode so these
        6 lines are the only ones required to use the LCD. You must also pass in
        the number of columns and lines on the LCD.

        The red, green, and blue parameters define the pins which are connected
        to the appropriate backlight LEDs. The invert_polarity parameter is a
        boolean that controls if the LEDs are on with a LOW or HIGH signal. By
        default invert_polarity is True, i.e. the backlight LEDs are on with a
        low signal. If you want to enable PWM on the backlight LEDs (for finer
        control of colors) and the hardware supports PWM on the provided pins,
        set enable_pwm to True. Finally you can set an explicit initial backlight
        color with the initial_color parameter. The default initial color is
        white (all LEDs lit).

        You can optionally pass in an explicit GPIO class,
        for example if you want to use an MCP230xx GPIO extender. If you don't
        pass in a GPIO instance, the default GPIO for the running platform will
        be used.
        """
        super(Adafruit_RGBCharLCD, self).__init__(rs, en, d4, d5, d6, d7,
                                                  cols,
                                                  lines,
                                                  enable_pwm=enable_pwm,
                                                  backlight=None,
                                                  invert_polarity=invert_polarity,
                                                  gpio=gpio,
                                                  pwm=pwm)

        self._red = red

```

```

self._green = green
self._blue = blue
# Setup backlight pins.
if enable_pwm:
    # Determine initial backlight duty cycles.
    rdc, gdc, bdc = self._rgb_to_duty_cycle(initial_color)
    pwm.start(red, rdc)
    pwm.start(green, gdc)
    pwm.start(blue, bdc)
else:
    gpio.setup(red, GPIO.OUT)
    gpio.setup(green, GPIO.OUT)
    gpio.setup(blue, GPIO.OUT)
    self.gpio.output_pins(self._rgb_to_pins(initial_color))

def _rgb_to_duty_cycle(self, rgb):
    # Convert tuple of RGB 0-1 values to tuple of duty cycles (0-100).
    red, green, blue = rgb
    # Clamp colors between 0.0 and 1.0
    red = max(0.0, min(1.0, red))
    green = max(0.0, min(1.0, green))
    blue = max(0.0, min(1.0, blue))
    return (self._pwm_duty_cycle(red),
            self._pwm_duty_cycle(green),
            self._pwm_duty_cycle(blue))

def _rgb_to_pins(self, rgb):
    # Convert tuple of RGB 0-1 values to dict of pin values.
    red, green, blue = rgb
    return { self._red: self._blpol if red else not self._blpol,
            self._green: self._blpol if green else not self._blpol,
            self._blue: self._blpol if blue else not self._blpol }

def set_color(self, red, green, blue):
    """Set backlight color to provided red, green, and blue values. If PWM
    is enabled then color components can be values from 0.0 to 1.0, otherwise
    components should be zero for off and non-zero for on.
    """
    if self._pwm_enabled:
        # Set duty cycle of PWM pins.
        rdc, gdc, bdc = self._rgb_to_duty_cycle((red, green, blue))
        self.pwm.set_duty_cycle(self._red, rdc)
        self.pwm.set_duty_cycle(self._green, gdc)
        self.pwm.set_duty_cycle(self._blue, bdc)
    else:
        # Set appropriate backlight pins based on polarity and enabled colors.
        self.gpio.output_pins({self._red: self._blpol if red else not self._blpol,
                              self._green: self._blpol if green else not self._blpol,
                              self._blue: self._blpol if blue else not self._blpol })

def set_backlight(self, backlight):
    """Enable or disable the backlight. If PWM is not enabled (default), a
    non-zero backlight value will turn on the backlight and a zero value will
    turn it off. If PWM is enabled, backlight can be any value from 0.0 to
    1.0, with 1.0 being full intensity backlight. On an RGB display this
    function will set the backlight to all white.
    """
    self.set_color(backlight, backlight, backlight)

class Adafruit_CharLCDPlate(Adafruit_RGBCharLCD):
    """Class to represent and interact with an Adafruit Raspberry Pi character
    LCD plate."""

    def __init__(self, address=0x20, busnum=I2C.get_default_bus(), cols=16, lines=2):
        """Initialize the character LCD plate. Can optionally specify a separate
        I2C address or bus number, but the defaults should suffice for most needs.
        Can also optionally specify the number of columns and lines on the LCD
        (default is 16x2).
        """
        # Configure MCP23017 device.
        self._mcp = MCP23017(address=address, busnum=busnum)

```

```

# Set LCD R/W pin to low for writing only.
self.mcp.setup(LCD_PLATE_RW, GPIO.OUT)
self.mcp.output(LCD_PLATE_RW, GPIO.LOW)
# Set buttons as inputs with pull-ups enabled.
for button in (SELECT, RIGHT, DOWN, UP, LEFT):
    self._mcp.setup(button, GPIO.IN)
    self._mcp.pullup(button, True)
# Initialize LCD (with no PWM support).
super(Adafruit_CharLCDPlate, self).init(LCD_PLATE_RS, LCD_PLATE_EN,
    LCD_PLATE_D4, LCD_PLATE_D5, LCD_PLATE_D6, LCD_PLATE_D7, cols, lines,
    LCD_PLATE_RED, LCD_PLATE_GREEN, LCD_PLATE_BLUE, enable_pwm=False,
    gpio=self._mcp)

def is_pressed(self, button):
    """Return True if the provided button is pressed, False otherwise."""
    if button not in set((SELECT, RIGHT, DOWN, UP, LEFT)):
        raise ValueError('Unknown button, must be SELECT, RIGHT, DOWN, UP, or LEFT.')
    return self._mcp.input(button) == GPIO.LOW

```