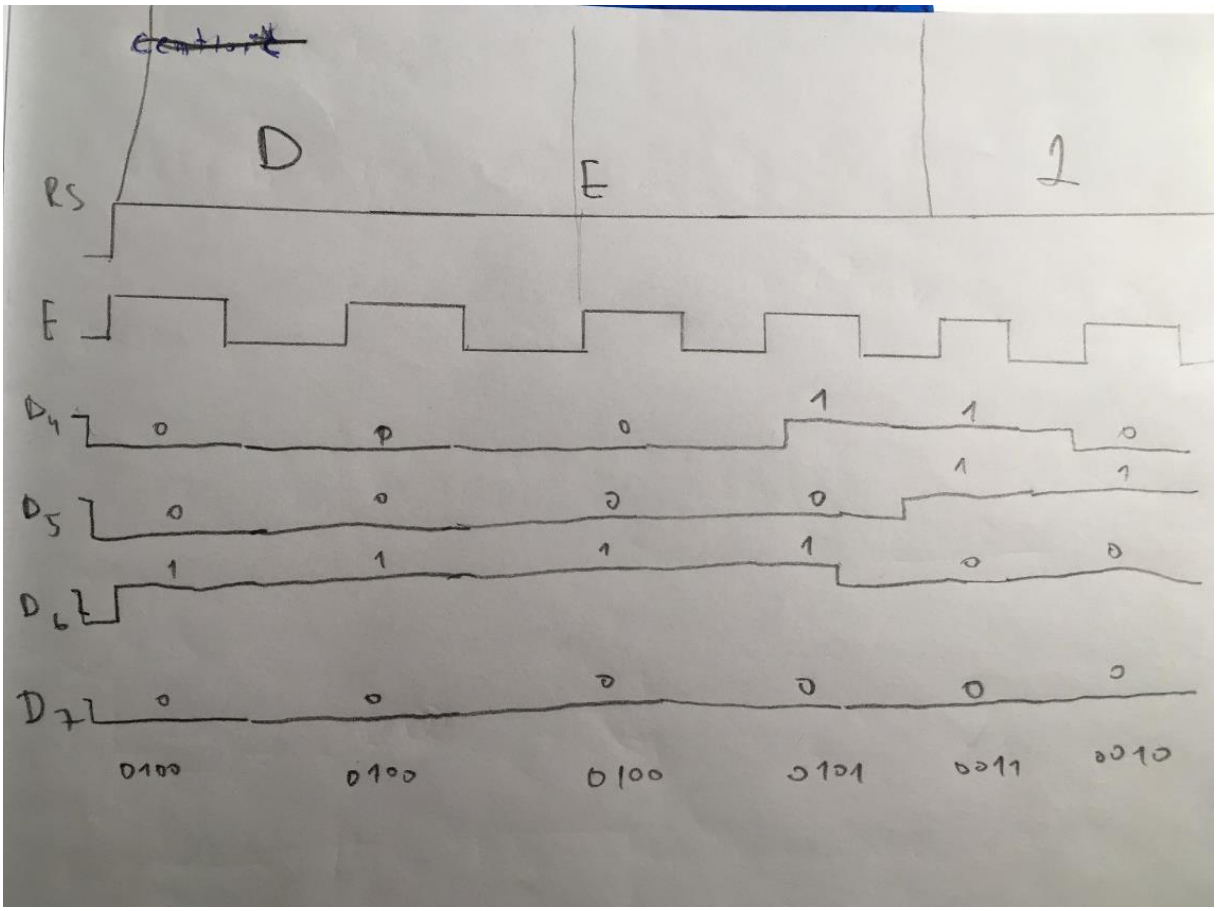
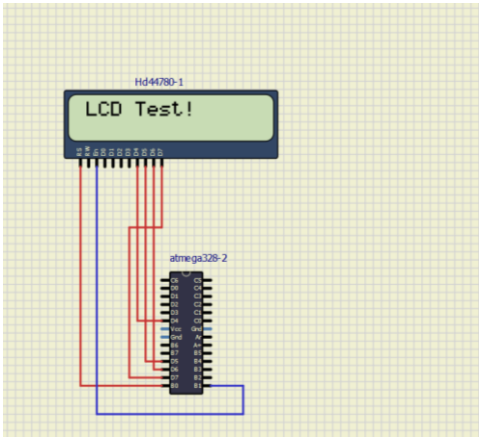
Kamil Káčer 211777

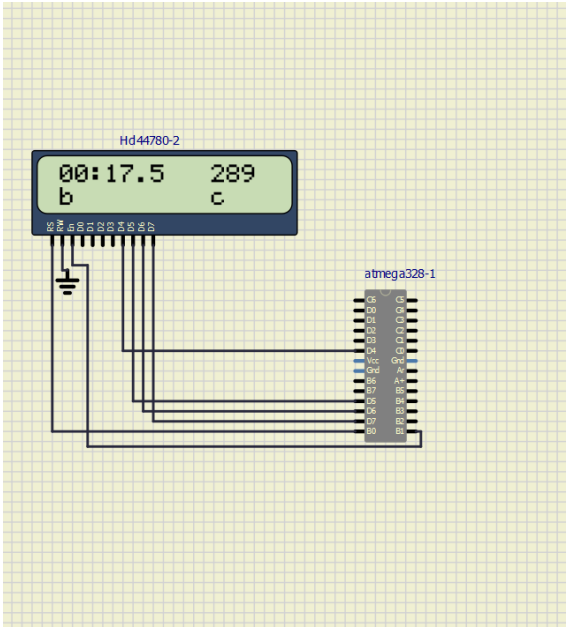
ASCII je kodovací systém znakov anglickej abecedy, číslic slúžiacich k riadeniu dátového prenosu

Obrázok časového signálu pri prenose dat ‘DE2‘





Lcd test



Timer with powered seconds.

Custom chararacter definition.

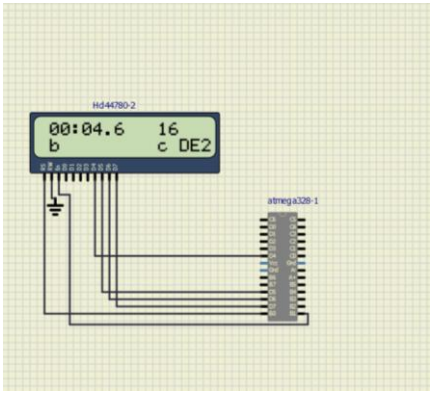
```
uint8_t customChar[] = {
    // addr 0: .....
    0b00000, 0b00000, 0b00000, 0b00000, 0b00000, 0b00000, 0b00000, 0b00000,
    // addr 1: |....
    0b10000, 0b10000, 0b10000, 0b10000, 0b10000, 0b10000, 0b10000, 0b10000,
    // addr 2: |....
    0b11000, 0b11000, 0b11000, 0b11000, 0b11000, 0b11000, 0b11000, 0b11000,
    // addr 3: |....
    0b11100, 0b11100, 0b11100, 0b11100, 0b11100, 0b11100, 0b11100, 0b11100,
    // addr 4: |....
    0b11110, 0b11110, 0b11110, 0b11110, 0b11110, 0b11110, 0b11110, 0b11110,
    // addr 5: |....
    0b11111, 0b11111, 0b11111, 0b11111, 0b11111, 0b11111, 0b11111, 0b11111
};
/* Function definitions ----------------------------------------------*/
/**
 * Main function where the program execution begins. Update stopwatch
 * value on LCD display when 8-bit Timer/Counter2 overflows.
 */
int main(void)
{
    // Initialize LCD display
    lcd_init(LCD_DISP_ON);
    // Set pointer to beginning of CGRAM memory
    lcd_command(1 << LCD_CGRAM);
    for (uint8_t i = 0; i < 48; i++)
    {
        // Store all new chars to memory line by line
        lcd_data(customChar[i]);
    }
    // Set DDRAM address
    lcd_command(1 << LCD_DDRAM);
```

Stop watch code:

```
/***********************************************************************
 *
 * Stopwatch with LCD display output.
 * ATmega328P (Arduino Uno), 16 MHz, AVR 8-bit Toolchain 3.6.2
 *
 * Copyright (c) 2017-2020 Tomas Fryza
 * Dept. of Radio Electronics, Brno University of Technology, Czechia
 * This work is licensed under the terms of the MIT license.
 *
 **********************************************************************/

/* Includes ----------------------------------------------------------*/
#include <avr/io.h>         // AVR device-specific IO definitions
#include <avr/interrupt.h>  // Interrupts standard C library for AVR-GCC
#include "timer.h"          // Timer library for AVR-GCC
#include "lcd.h"            // Peter Fleury's LCD library
#include <stdlib.h>         // C library. Needed for conversion function

/* Function definitions ----------------------------------------------*/
/**
 * Main function where the program execution begins. Update stopwatch
 * value on LCD display when 8-bit Timer/Counter2 overflows.
 */
int main(void)
{
    // Initialize LCD display
        lcd_init(LCD_DISP_ON);

    // Put string(s) at LCD display
       lcd_gotoxy(1, 0);
```

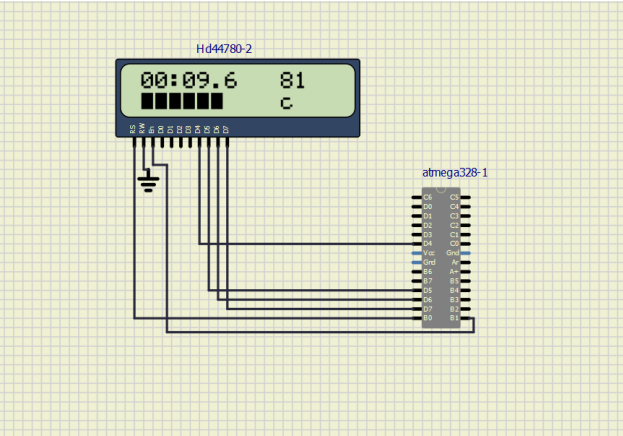
```
        lcd_puts("00:00.0");
        lcd_gotoxy(11, 0);
        lcd_putc('a');
        lcd_gotoxy(1, 1);
        lcd_putc('b');
        lcd_gotoxy(11, 1);
        lcd_putc('c');

    // Configure 8-bit Timer/Counter2 for Stopwatch
    // Set prescaler and enable overflow interrupt every 16 ms
        TIM2_overflow_16ms();
        TIM2_overflow_interrupt_enable()


    // Enables interrupts by setting the global interrupt mask
    sei();

    // Infinite loop
    while (1)
    {
        /* Empty loop. All subsequent operations are performed exclusively
         * inside interrupt service routines ISRs */
    }

    // Will never reach this
    return 0;
}

/* Interrupt service routines ----------------------------------------*/
/**
 * ISR starts when Timer/Counter2 overflows. Update the stopwatch on
 * LCD display every sixth overflow, ie approximately every 100 ms
 * (6 x 16 ms = 100 ms).
 */
ISR(TIMER2_OVF_vect)
{
      static uint8_t number_of_overflows = 0;
       static uint8_t tens = 0;
       static uint8_t secs = 0;
       static uint8_t min = 0;
       uint16_t squaresecs = secs * secs;

       //static uint8_t secs_1 = 0;
    char lcd_string[2] = " ";     //string for converting numbers
    number_of_overflows++;
    if (number_of_overflows >= 6)
    {
        // Do this every 6 x 16 ms = 100 ms
        number_of_overflows = 0;
              tens ++;
              if (tens > 9)
              {
              tens = 0;
              secs ++;
              }
              itoa(tens, lcd_string, 10);
              lcd_gotoxy(7, 0);
              lcd_puts(lcd_string);

              if (secs > 59)
              {
                     secs = 0;
```

```
                min ++;
                lcd_gotoxy(4, 0);
                lcd_putc('0');
        }
        itoa(secs, lcd_string, 10);
        if (secs > 9)
        {
                lcd_gotoxy(4,0);
        }
        else
        {
                lcd_gotoxy(5,0);
        }
        lcd_puts(lcd_string);

        if (min > 60)
        {
                min = 0;
                lcd_putc('0');
                lcd_gotoxy(2, 0);

        }
        itoa(min, lcd_string, 10);
        if (min > 9)
        {
                lcd_gotoxy(1,0);
        }
        else
        {
                lcd_gotoxy(2,0);
        }
        lcd_puts(lcd_string);

        itoa(squaresecs, lcd_string, 10);
        if (squaresecs = 0)
        {
                lcd_gotoxy(11, 0);
                lcd_puts("0  ");
        }
        else
        {
                lcd_gotoxy(11,0);
        }
        lcd_puts(lcd_string);
    }
}
```



Function name	Function parameters	Description	Example
lcd_init	LCD_DISP_OFF	Display off	lcd_init(LCD_DISP_OFF);
	LCD_DISP_ON	Display on	
	LCD_DISP_ON_CURSOR	Cursor on	
	LCD_DISP_ON_CURSOR_BLINK	Blinking cursor	
lcd_gotoxy	x horizontal position (0: left most position)	Set cursor to specified position.	lcd_gotoxy(1,10)
	y vertical position (0: first line)		
lcd_clrscr		Clear display and set cursor to home position.	lcd_clrscr();
lcd_putc	c character to be displayed	Display character at current cursor position.	lcd_putc('c')
lcd_puts	sample string to be displayed	Display string	lcd_puts("sample")
lcd_command	cmd instruction to send to LCD controller	Send LCD controller instruction command	lcd_command (uint8_t cmd)
lcd_data	data byte to send to LCD controller	Send data byte to LCD controller.	lcd_data (uint8_t data)

Lcd signals	AVR pins	Description
RS	PB0	Register selection signal Selection between instruction register and data register
R/W	GND	Read or Write
D[3:0]	x	Low order bidirectional bus
D[7:4]	PD[7:4]	High order bidirectional transfer data bus



Loading bar

```
ISR(TIMER0_OVF_vect)
{
	static uint8_t symbol = 0;
	static uint8_t position = 0;
	static uint8_t ovrflov = 0;

	ovrflov ++;
	if(ovrflov >= 1)

	ovrflov = 0;
	lcd_gotoxy(1 + position, 1);
	lcd_putc(symbol);
	symbol ++;
	if(symbol > 5)
	{
		position ++;
		symbol = 0;

		if(position == 9)
		{
			position = 0;

			lcd_gotoxy(9,1);
			lcd_putc(0);
			lcd_gotoxy(8,1);
			lcd_putc(0);
			lcd_gotoxy(7,1);
			lcd_putc(0);
			lcd_gotoxy(6,1);
			lcd_putc(0);
			lcd_gotoxy(5,1);
			lcd_putc(0);
			lcd_gotoxy(4,1);
			lcd_putc(0);
			lcd_gotoxy(3,1);
			lcd_putc(0);
			lcd_gotoxy(2,1);
			lcd_putc(0);
			lcd_gotoxy(1,1);
			lcd_putc(0);

			}

		}
```

```
}
```