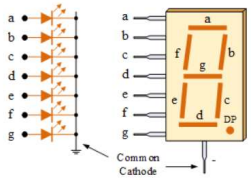
1A) Anode 7-segment display

Digit	A	B	C	D	E	F	G	DP
0	0	0	0	0	0	0	1	1
1	1	0	0	1	1	1	1	1
2	0	0	1	0	0	1	0	1
3	0	0	0	0	1	1	0	1
4	1	0	0	1	1	0	0	1
5	0	1	0	0	1	0	0	1
6	0	1	0	0	0	0	0	1
7	0	0	0	1	1	1	1	1
8	0	0	0	0	0	0	0	1
9	0	0	0	0	1	0	0	1

1B) Describe the difference between Common Cathode and Common Anode 7-segment display

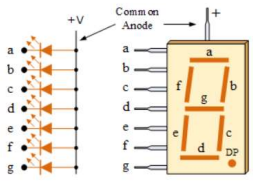
V případě přivedení log.1 na segmenty a-g u Katodového 7-segment displeje, pak tyto segmenty budou svítit. (ledky mají společné uzemnění)

Common Cathode 7-segment Display



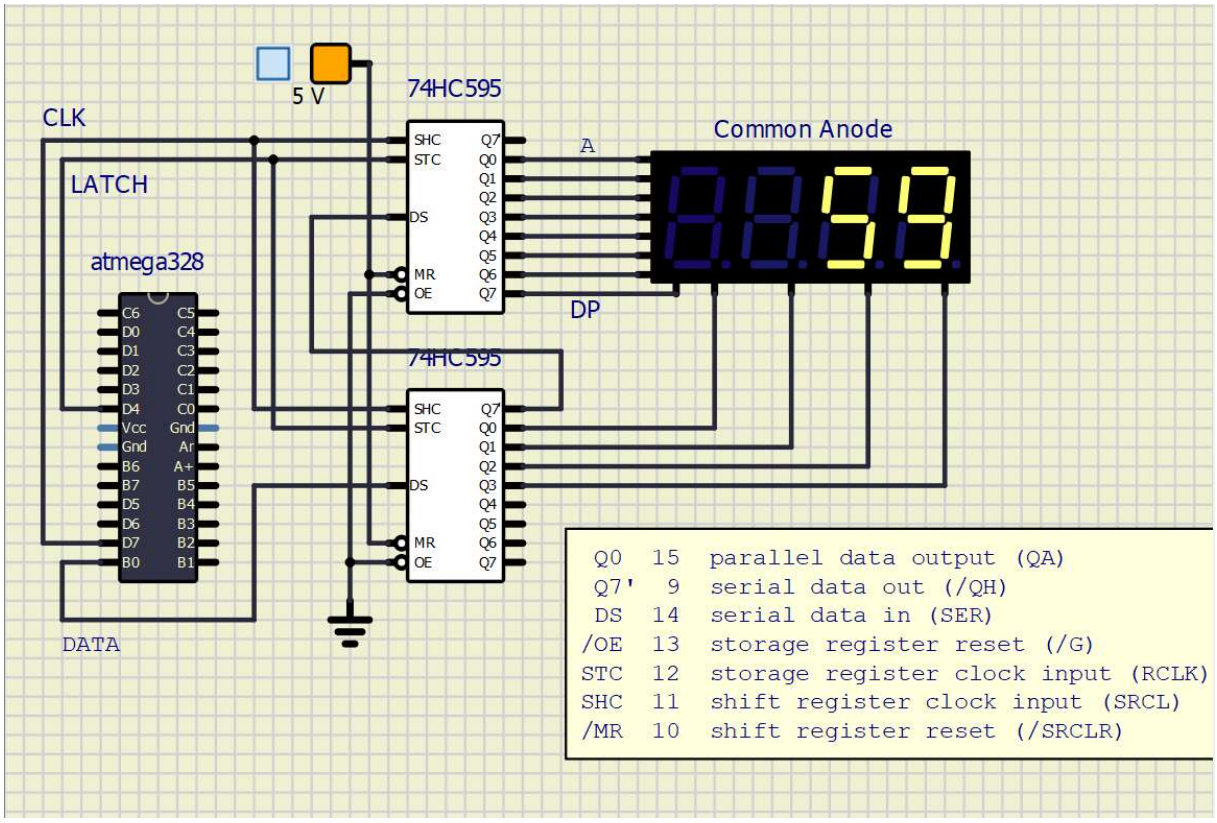
V případě přivedení log.0 na segmenty a-g u Anodového 7-segment displeje, pak tyto segmenty budou svítit. (ledky mají společné +V)

Common Anode 7-segment Display



Interup	Vector name	Pins	Operation	I/O register	Bit(s)
Pin Change Interrupt 0	PCINT0_vect	PB[7:0]	Interrupt enable Select pins	PCICR PCMSK0	PCIE0 PCINT[7:0]
Pin Change Interrupt 1	PCINT1_vect	PB[14:8]	Interrupt enable Select pins	PCICR PCMSK1	PCIE1 PCINT[14:8]
Pin Change Interrupt 2	PCINT2_vect	PB[23:16]	Interrupt enable Select pins	PCICR PCMSK2	PCIE2 PCINT[23:16]

2A,B,C)



3A,B)

Snake	A	B	C	D	E	F	G	DP
A	0	1	1	1	1	1	1	1
B	1	0	1	1	1	1	1	1
C	1	1	0	1	1	1	1	1
D	1	1	1	0	1	1	1	1
E	1	1	1	1	0	1	1	1
F	1	1	1	1	1	0	1	1

```c
/***********************************************************************
 *
 * Seven-segment display library for AVR-GCC.
 * ATmega328P (Arduino Uno), 16 MHz, AVR 8-bit Toolchain 3.6.2
 *
 * Copyright (c) 2019-2020 Tomas Fryza
 * Dept. of Radio Electronics, Brno University of Technology, Czechia
 * This work is licensed under the terms of the MIT license.
 *
 **********************************************************************/

/* Includes ----------------------------------------------------------*/
#define F_CPU 16000000
#include <util/delay.h>
#include "gpio.h"
#include "segment.h"

/* Variables ---------------------------------------------------------*/
// Active-low digit 0 to 9
uint8_t segment_value[] = {
    // abcdefgDP
    0b00000011,     // Digit 0
    0b10011111,     // Digit 1
    0b00100101,     // Digit 2
    0b00001101,     // Digit 3
    0b10011001,     // Digit 4
    0b01001001,     // Digit 5
    0b01000001,     // Digit 6
    0b00011111,     // Digit 7
    0b00000001,     // Digit 8
    0b00001001      // Digit 9
};

// Active-high position 0 to 3
uint8_t segment_position[] = {
    // p3p2p1p0....
    0b00010000,     // Position 0
    0b00100000,     // Position 1
    0b01000000,     // Position 2
    0b10000000      // Position 3
};


/* Function definitions ----------------------------------------------*/
void SEG_init(void)
{
    /* Configuration of SSD signals */
    GPIO_config_output(&DDRD, SEGMENT_LATCH);
    GPIO_config_output(&DDRD, SEGMENT_CLK);
    GPIO_config_output(&DDRB, SEGMENT_DATA);
}

/*--------------------------------------------------------------------*/
```

```
54  void SEG_update_shift_regs(uint8_t segments, uint8_t position)
55  {
56      uint8_t bit_number;
57      segments = segment_value[segments];     // 0, 1, ..., 9
58      position = segment_position[position];  // 0, 1, 2, 3
59
60      // Pull LATCH, CLK, and DATA low
61      GPIO_write_low(&PORTD, SEGMENT_LATCH);
62      GPIO_write_low(&PORTD, SEGMENT_CLK);
63      GPIO_write_low(&PORTB, SEGMENT_DATA);
64
65      // Wait 1 us
66      _delay_us(1);
67
68      // Loop through the 1st byte (segments)
69      // a b c d e f g DP (active low values)
70      for (bit_number = 0; bit_number < 8; bit_number++)
71      {
72          // Output DATA value (bit 0 of "segments")
73          if ((segments & 1) == 0)
74              GPIO_write_low(&PORTB, SEGMENT_DATA);
75          else
76              GPIO_write_high(&PORTB, SEGMENT_DATA);
77
78          // Wait 1 us
79          _delay_us(1);
80
81          // Pull CLK high
82          GPIO_write_high(&PORTD, SEGMENT_CLK);
83
84          // Wait 1 us
85          _delay_us(1);
86
87          // Pull CLK low
88          GPIO_write_low(&PORTD, SEGMENT_CLK);
89
90          // Shift "segments"
91          segments = segments >> 1;
92      }
93
94      // Loop through the 2nd byte (position)
95      // p3 p2 p1 p0 . . . . (active high values)
96      for (bit_number = 0; bit_number < 8; bit_number++)
97      {
98          // Output DATA value (bit 0 of "position")
99          if ((position % 2) == 0)
100             GPIO_write_low(&PORTB, SEGMENT_DATA);
101         else
102             GPIO_write_high(&PORTB, SEGMENT_DATA);
103
104        // Wait 1 us
105        _delay_us(1);
106
```

```
107         // Pull CLK high
108         GPIO_write_high(&PORTD, SEGMENT_CLK);
109 
110         // Wait 1 us
111         _delay_us(1);
112 
113         // Pull CLK low
114         GPIO_write_low(&PORTD, SEGMENT_CLK);
115 
116          // Shift "position"
117          position = position >> 1;
118     }
119 
120     // Pull LATCH high
121     GPIO_write_high(&PORTD, SEGMENT_LATCH);
122 
123     // Wait 1 us
124     _delay_us(1);
125 }
126 
127 /*-------------------------------------------------------------------*/
128 /* SEG_clear */
129 void SEG_clear(void)
130 {
131     uint8_t bit_number, segments = 0b11111111, position = 0;
132 
133     // Pull LATCH, CLK, and DATA low
134     GPIO_write_low(&PORTD, SEGMENT_LATCH);
135     GPIO_write_low(&PORTD, SEGMENT_CLK);
136     GPIO_write_low(&PORTB, SEGMENT_DATA);
137 
138     // Wait 1 us
139     _delay_us(1);
140 
141     // Loop through the 1st byte (segments)
142     // a b c d e f g DP (active low values)
143     for (bit_number = 0; bit_number < 8; bit_number++)
144     {
145         // Output DATA value (bit 0 of "segments")
146         if ((segments & 1) == 0)
147             GPIO_write_low(&PORTB, SEGMENT_DATA);
148         else
149             GPIO_write_high(&PORTB, SEGMENT_DATA);
150 
151         // Wait 1 us
152         _delay_us(1);
153 
154         // Pull CLK high
155         GPIO_write_high(&PORTD, SEGMENT_CLK);
156 
157         // Wait 1 us
158         _delay_us(1);
159 
```

```
160         // Pull CLK low
161         GPIO_write_low(&PORTD, SEGMENT_CLK);
162 
163         // Shift "segments"
164         segments = segments >> 1;
165     }
166 
167     // Loop through the 2nd byte (position)
168     // p3 p2 p1 p0 . . . . (active high values)
169     for (bit_number = 0; bit_number < 8; bit_number++)
170     {
171         // Output DATA value (bit 0 of "position")
172         if ((position % 2) == 0)
173             GPIO_write_low(&PORTB, SEGMENT_DATA);
174         else
175             GPIO_write_high(&PORTB, SEGMENT_DATA);
176 
177         // Wait 1 us
178         _delay_us(1);
179 
180         // Pull CLK high
181         GPIO_write_high(&PORTD, SEGMENT_CLK);
182 
183         // Wait 1 us
184         _delay_us(1);
185 
186         // Pull CLK low
187         GPIO_write_low(&PORTD, SEGMENT_CLK);
188 
189         // Shift "position"
190         position = position >> 1;
191     }
192 
193     // Pull LATCH high
194     GPIO_write_high(&PORTD, SEGMENT_LATCH);
195 
196     // Wait 1 us
197     _delay_us(1);
198 
199 }
200 
201 /*-------------------------------------------------------------------*/
202 /* SEG_clk_2us */
203 void SEG_clk_2us(void)
204 {
205     // Wait 1 us
206     _delay_us(1);
207 
208     // Pull CLK high
209     GPIO_write_high(&PORTD, SEGMENT_CLK);
210 
211     // Wait 1 us
212     _delay_us(1);
```

```
213
214     // Pull CLK low
215     GPIO_write_low(&PORTD, SEGMENT_CLK);
216 }
```

```c
/***********************************************************************
 *
 * Decimal counter with 7-segment output.
 * ATmega328P (Arduino Uno), 16 MHz, AVR 8-bit Toolchain 3.6.2
 *
 * Copyright (c) 2018-2020 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 "segment.h"        // Seven-segment display library for AVR-GCC

uint8_t singles = 0;
uint8_t decimals = 0;


/* Function definitions ----------------------------------------------*/
/**
 * Main function where the program execution begins. Display decimal
 * counter values on SSD (Seven-segment display) when 16-bit
 * Timer/Counter1 overflows.
 */
int main(void)
{
    // Configure SSD signals
    SEG_init();



    // Test of SSD: display number '3' at position 0
    SEG_update_shift_regs(3, 0);

    //SEG_clear();


    /* Configure 8-bit Timer/Counter0
     * Set prescaler and enable overflow interrupt */
    TIM0_overflow_4ms();
    TIM0_overflow_interrupt_enable();

    /* Configure 16-bit Timer/Counter1
     * Set prescaler and enable overflow interrupt */
    TIM1_overflow_262ms();
    TIM1_overflow_interrupt_enable();

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

```

```c
    // 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/Counter0 overflows. Display value on SSD.
 */
ISR(TIMER0_OVF_vect)
{
    static uint8_t position = 0;
    if (position == 0)
        SEG_update_shift_regs(singles, 0); //first position
    else
        SEG_update_shift_regs(decimals, 1); //second position
    position = !position; //change position (0 1)
}

/**
 * ISR starts when Timer/Counter1 overflows. Increment decimal counter.
 */
ISR(TIMER1_OVF_vect)
{
    // number AB = 0-5 0-9
    // LEd counter 0-59
    singles++;
    if(singles > 9)
    {
        singles = 0;
        decimals++;
        if(decimals > 5)
            decimals = 0;
    }
}
```

```c
/***********************************************************************
 *
 * Seven-segment display library for AVR-GCC.
 * ATmega328P (Arduino Uno), 16 MHz, AVR 8-bit Toolchain 3.6.2
 *
 * Copyright (c) 2019-2020 Tomas Fryza
 * Dept. of Radio Electronics, Brno University of Technology, Czechia
 * This work is licensed under the terms of the MIT license.
 *
 **********************************************************************/

/* Includes ----------------------------------------------------------*/
#define F_CPU 16000000
#include <util/delay.h>
#include "gpio.h"
#include "segment.h"

/* Variables ---------------------------------------------------------*/
// Active-low digit a-f
uint8_t segment_value[] = {
    // abcdefgDP
    0b01111111,     // Digit a
    0b10111111,     // Digit b
    0b11011111,     // Digit c
    0b11101111,     // Digit d
    0b11110111,     // Digit e
    0b11111011,     // Digit f
};

// Active-high position 0 to 3
uint8_t segment_position[] = {
    // p3p2p1p0....
    0b00010000,     // Position 0
    0b00100000,     // Position 1
    0b01000000,     // Position 2
    0b10000000      // Position 3
};


/* Function definitions ----------------------------------------------*/
void SEG_init(void)
{
    /* Configuration of SSD signals */
    GPIO_config_output(&DDRD, SEGMENT_LATCH);
    GPIO_config_output(&DDRD, SEGMENT_CLK);
    GPIO_config_output(&DDRB, SEGMENT_DATA);
}

/*-------------------------------------------------------------------*/
void SEG_update_shift_regs(uint8_t segments, uint8_t position)
{
    uint8_t bit_number;
    segments = segment_value[segments];     // 0, 1, ..., 5
```

```
54      position = segment_position[position];  // 0
55
56      // Pull LATCH, CLK, and DATA low
57      GPIO_write_low(&PORTD, SEGMENT_LATCH);
58      GPIO_write_low(&PORTD, SEGMENT_CLK);
59      GPIO_write_low(&PORTB, SEGMENT_DATA);
60
61      // Wait 1 us
62      _delay_us(1);
63
64      // Loop through the 1st byte (segments)
65      // a b c d e f g DP (active low values)
66      for (bit_number = 0; bit_number < 8; bit_number++)
67      {
68          // Output DATA value (bit 0 of "segments")
69          if ((segments & 1) == 0)
70              GPIO_write_low(&PORTB, SEGMENT_DATA);
71          else
72              GPIO_write_high(&PORTB, SEGMENT_DATA);
73
74          // Wait 1 us
75          _delay_us(1);
76
77          // Pull CLK high
78          GPIO_write_high(&PORTD, SEGMENT_CLK);
79
80          // Wait 1 us
81          _delay_us(1);
82
83          // Pull CLK low
84          GPIO_write_low(&PORTD, SEGMENT_CLK);
85
86          // Shift "segments"
87          segments = segments >> 1;
88      }
89
90      // Loop through the 2nd byte (position)
91      // p3 p2 p1 p0 . . . . (active high values)
92      for (bit_number = 0; bit_number < 8; bit_number++)
93      {
94          // Output DATA value (bit 0 of "position")
95          if ((position % 2) == 0)
96              GPIO_write_low(&PORTB, SEGMENT_DATA);
97          else
98              GPIO_write_high(&PORTB, SEGMENT_DATA);
99
100         // Wait 1 us
101         _delay_us(1);
102
103         // Pull CLK high
104         GPIO_write_high(&PORTD, SEGMENT_CLK);
105
106         // Wait 1 us
```

```
107         _delay_us(1);
108 
109         // Pull CLK low
110         GPIO_write_low(&PORTD, SEGMENT_CLK);
111 
112          // Shift "position"
113          position = position >> 1;
114     }
115 
116     // Pull LATCH high
117     GPIO_write_high(&PORTD, SEGMENT_LATCH);
118 
119     // Wait 1 us
120     _delay_us(1);
121 }
122 
123 /*-------------------------------------------------------------------- */
124 /* SEG_clear */
125 void SEG_clear(void)
126 {
127     uint8_t bit_number, segments = 0b11111111, position = 0;
128 
129     // Pull LATCH, CLK, and DATA low
130     GPIO_write_low(&PORTD, SEGMENT_LATCH);
131     GPIO_write_low(&PORTD, SEGMENT_CLK);
132     GPIO_write_low(&PORTB, SEGMENT_DATA);
133 
134     // Wait 1 us
135     _delay_us(1);
136 
137     // Loop through the 1st byte (segments)
138     // a b c d e f g DP (active low values)
139     for (bit_number = 0; bit_number < 8; bit_number++)
140     {
141         // Output DATA value (bit 0 of "segments")
142         if ((segments & 1) == 0)
143             GPIO_write_low(&PORTB, SEGMENT_DATA);
144         else
145             GPIO_write_high(&PORTB, SEGMENT_DATA);
146 
147         // Wait 1 us
148         _delay_us(1);
149 
150         // Pull CLK high
151         GPIO_write_high(&PORTD, SEGMENT_CLK);
152 
153         // Wait 1 us
154         _delay_us(1);
155 
156         // Pull CLK low
157         GPIO_write_low(&PORTD, SEGMENT_CLK);
158 
159         // Shift "segments"
```

```
160         segments = segments >> 1;
161     }
162 
163     // Loop through the 2nd byte (position)
164     // p3 p2 p1 p0 . . . . (active high values)
165     for (bit_number = 0; bit_number < 8; bit_number++)
166     {
167         // Output DATA value (bit 0 of "position")
168         if ((position % 2) == 0)
169             GPIO_write_low(&PORTB, SEGMENT_DATA);
170         else
171             GPIO_write_high(&PORTB, SEGMENT_DATA);
172 
173         // Wait 1 us
174         _delay_us(1);
175 
176         // Pull CLK high
177         GPIO_write_high(&PORTD, SEGMENT_CLK);
178 
179         // Wait 1 us
180         _delay_us(1);
181 
182         // Pull CLK low
183         GPIO_write_low(&PORTD, SEGMENT_CLK);
184 
185         // Shift "position"
186         position = position >> 1;
187     }
188 
189     // Pull LATCH high
190     GPIO_write_high(&PORTD, SEGMENT_LATCH);
191 
192     // Wait 1 us
193     _delay_us(1);
194 
195 }
196 
197 /*-------------------------------------------------------------------*/
198 /* SEG_clk_2us */
199 void SEG_clk_2us(void)
200 {
201     // Wait 1 us
202     _delay_us(1);
203 
204     // Pull CLK high
205     GPIO_write_high(&PORTD, SEGMENT_CLK);
206 
207     // Wait 1 us
208     _delay_us(1);
209 
210     // Pull CLK low
211     GPIO_write_low(&PORTD, SEGMENT_CLK);
212 }
```

```c
/***********************************************************************
 *
 * Decimal counter with 7-segment output.
 * ATmega328P (Arduino Uno), 16 MHz, AVR 8-bit Toolchain 3.6.2
 *
 * Copyright (c) 2018-2020 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 "segment.h"        // Seven-segment display library for AVR-GCC

uint8_t singles = 0;


/**
 * Main function where the program execution begins. Display decimal
 * counter values on SSD (Seven-segment display) when 16-bit
 * Timer/Counter1 overflows.
 */
int main(void)
{
    // Configure SSD signals
    SEG_init();

    /* Configure 8-bit Timer/Counter0
     * Set prescaler and enable overflow interrupt */
    TIM0_overflow_4ms();
    TIM0_overflow_interrupt_enable();

    /* Configure 16-bit Timer/Counter1
     * Set prescaler and enable overflow interrupt */
    TIM1_overflow_262ms();
    TIM1_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;
}
```

```
54
55  /* Interrupt service routines ----------------------------------------*/
56  /**
57   * ISR starts when Timer/Counter0 overflows. Display value on SSD.
58   */
59  ISR(TIMER0_OVF_vect)
60  {
61          SEG_update_shift_regs(singles, 0); //first position
62  }
63
64  /**
65   * ISR starts when Timer/Counter1 overflows. Increment decimal counter.
66   */
67  ISR(TIMER1_OVF_vect)
68  {
69      // SNAKE counter 0-5
70      singles++;
71      if(singles > 5)
72      {
73          singles = 0;
74      }
75  }
```