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

Rozdíl mezi displayi se společnou katodou a anodou je v řízení, kdy v případě zapojení se společnou anodou je display aktivní v případě že je na jeho výstup přivedena úroveň low, u zapojení se společnou katodou je to naopak.

Segment.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"

/* 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);
}

/* Variables ---------------------------------------------------------*/
// Active-low digits 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


/*-------------------------------------------------------------------*/
void SEG_update_shift_regs(uint8_t segments, uint8_t position)
{
	uint8_t bit_number;

	segments = segment_value[segments];	// 0, 1, ..., 9
```

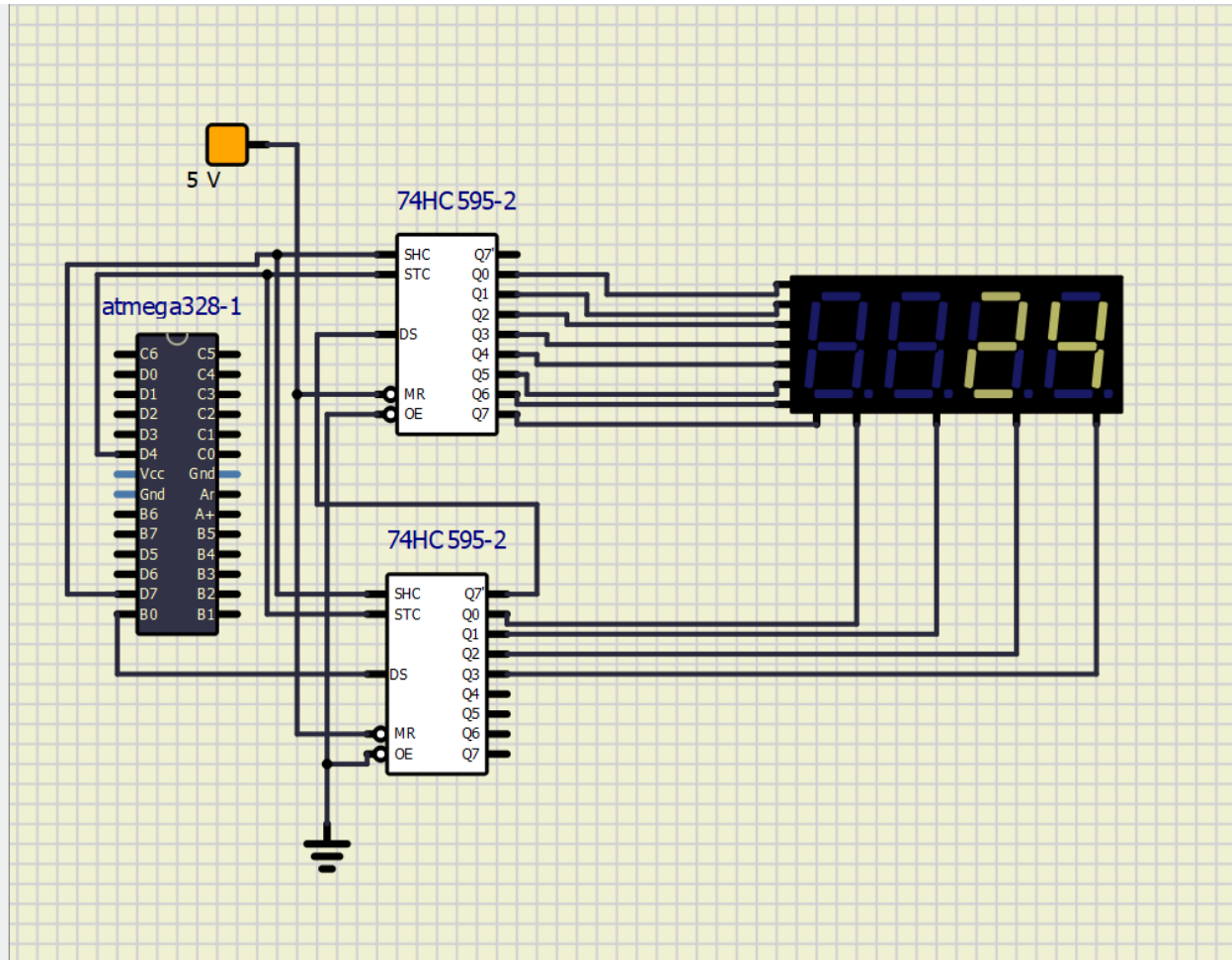
```
position = segment_position[position];
// Pull LATCH, CLK, and DATA low
GPIO_write_low(&PORTD,SEGMENT_LATCH);
GPIO_write_low(&PORTD,SEGMENT_CLK);
GPIO_write_low(&PORTB,SEGMENT_DATA);
// Wait 1 us
_delay_us(1);
// Loop through the 1st byte (segments)
// a b c d e f g DP (active low values)
for (bit_number = 0; bit_number < 8; bit_number++)
{
	// Output DATA value (bit 0 of "segments")
	if(segments & 1)
	{
		GPIO_write_high(&PORTB,SEGMENT_DATA);
	}else
	{
		GPIO_write_low(&PORTB,SEGMENT_DATA);
	}

	// Wait 1 us
	_delay_us(1);
	// Pull CLK high
	GPIO_write_high(&PORTD,SEGMENT_CLK);
	// Wait 1 us
	_delay_us(1);
	// Pull CLK low
	GPIO_write_low(&PORTD,SEGMENT_CLK);
	// Shift "segments"
	segments = segments >> 1;
}

// Loop through the 2nd byte (position)
// p3 p2 p1 p0 . . . . (active high values)
for (bit_number = 0; bit_number < 8; bit_number++)
{
	// Output DATA value (bit 0 of "position")
	if(position & 1)
	{
		GPIO_write_high(&PORTB,SEGMENT_DATA);
	}else
	{
		GPIO_write_low(&PORTB,SEGMENT_DATA);
	}

	// Wait 1 us
	_delay_us(1);
	// Pull CLK high
	GPIO_write_high(&PORTD,SEGMENT_CLK);
	// Wait 1 us
	_delay_us(1);
	// Pull CLK low
	GPIO_write_low(&PORTD,SEGMENT_CLK);

	// Shift "position"
	position = position >> 1;
}
```

```
        // Pull LATCH high
        GPIO_write_high(&PORTD,SEGMENT_LATCH);
        // Wait 1 us
        _delay_us(1);
}

/*-------------------------------------------------------------------*/
/* SEG_clear */

/*-------------------------------------------------------------------*/
/* SEG_clk_2us */
```

Main.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



/* Function definitions ----------------------------------------------*/
/**
 * Main function where the program execution begins. Display decimal
 * counter values on SSD (Seven-segment display) when 16-bit
 * Timer/Counter1 overflows.
 */
uint8_t singles=0;
uint8_t deciamals=0;
int main(void)
{
    // Configure SSD signals
    SEG_init();


    /* Configure 16-bit Timer/Counter1
     * Set prescaler and enable overflow interrupt */
      TIM1_overflow_262ms();
      TIM1_overflow_interrupt_enable();
      /* Configure 8-bit Timer/Counter0
     * Set prescaler and enable overflow interrupt */
      TIM0_overflow_4m();
      TIM0_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/Counter1 overflows. Increment decimal counter
 * value and display it on SSD.
 */
ISR(TIMER1_OVF_vect) // routine for counting upto 59 from 00
{

        if(singles > 8)
        {
              if(deciamals<5)
              {
                    deciamals++;
              }
              else
              {
                    deciamals=0;
              }
              singles=0;
       }
       else
              singles++;



}

ISR(TIMER0_OVF_vect) // routine for displaying numbers on 4 SSDs
{
       static uint8_t pos=0;

       if(pos==0)
       {
              SEG_update_shift_regs(singles, pos);
              pos=1;
       }else
       {
              SEG_update_shift_regs(deciamals, pos);
              pos=0;
       }



}
```



Snake lookup table

```
uint8_t segment_value[] = {
       // Snake
       0b11011111,          // segment lower right
       0b11101111,          // segment lower
       0b11110111,          // segment lower left
       0b11111011,          // Segment upper left
       0b01111111,          // Segment upper
       0b10111111           // Segment upper right
};
```

Snake

```
/***********************************************************************
 *
 * Snake  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



/* Function definitions ----------------------------------------------*/
/**
 * Main function where the program execution begins. Display decimal
 * counter values on SSD (Seven-segment display) when 16-bit
 * Timer/Counter1 overflows.
 */
uint8_t value=0;

int main(void)
{
    // Configure SSD signals
    SEG_init();


    /* 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;
}

/* Interrupt service routines ----------------------------------------*/
/**
 * ISR starts when Timer/Counter1 overflows. Increment decimal counter
 * value and display it on SSD.
```

```
 */
ISR(TIMER1_OVF_vect) // routine for moving snake around 7 segment display
{

	if(value<5)
	{
		SEG_update_shift_regs(++value, 0);
	}
	else
	{
		value=0;
		SEG_update_shift_regs(value, 0);
	}



}
```