Link to my GitHub Reporsitory

Table 1:Segments values for display 0 to 9

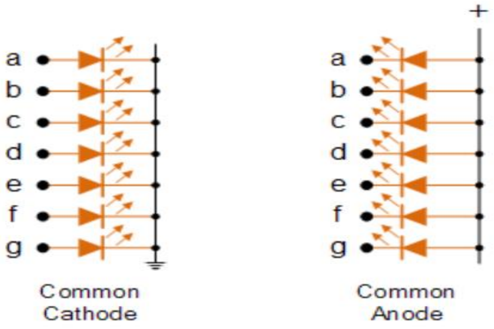
Digit	A	B	C	D	E	F	G	DP
0	0	0	0	0	0	0	1	1
1	1	0	0	1	1	1	1	1
2	0	0	1	0	0	1	0	1
3	0	0	0	0	1	1	0	1
4	1	0	0	1	1	0	0	1
5	0	1	0	0	1	0	0	1
6	0	1	0	0	0	0	0	1
7	0	0	0	1	1	1	1	1
8	0	0	0	0	0	0	0	1
9	0	0	0	0	1	0	0	1

Table 2: Look up Table with Snake Definition

Segments	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

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

It becomes a common anode in seven segment displays, when all the anodes are connected to one point. Common cathode means that all of a 7-segment display's seven cathodes are connected together



A positive voltage should be supplied to the common anode in order to operate and the common cathode should be grounded. The Cathode(-) side of led's are connected to a, b , c, d , e, f, g pins of seven segment display in common Anode. The anode(+) side's of Common Cathode led are connected to a, b , c, d , e, f, g pins of seven segment display

Listing of library source file 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"
uint8_t clearsegments = 0;

/* 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);
}

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


        if(clearsegments==0){
         segments = segment_value[segments];     // 0, 1, ..., 9
         position = segment_position[position];  // 0, 1, 2, 3
               }
        else if(clearsegments==1){
               segments=0b11111111;// in order to Turn off the all segments we set all
bit 1 because of aktiv low connection
               }

    // Pull LATCH, CLK, and DATA low
       GPIO_write_low(&PORTD, SEGMENT_LATCH);   // LATCH
       GPIO_write_low(&PORTD, SEGMENT_CLK);            // CLK
       GPIO_write_low(&PORTB, SEGMENT_DATA);    // 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 % 2)==0) // LSB is 0
                     GPIO_write_low(&PORTB, SEGMENT_DATA);
              else
                     GPIO_write_high(&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 % 2)==0)// LSB is 0

                     GPIO_write_low(&PORTB, SEGMENT_DATA);
              else
                    GPIO_write_high(&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 */


void SEG_clear()// this function will be used for the Turn off all segments at all
postions
{

       clearsegments=1;

}

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

void SEG_clk_2us ()
{
              GPIO_write_high(&PORTD, SEGMENT_CLK);     // CLK
              _delay_us(1);
              GPIO_write_low(&PORTD, SEGMENT_CLK);
              _delay_us(1);
}
```

Listing of decimal counter application main.c

```
/* 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.
 */

/*Global variable*/

uint8_t cnt0 = 0;    // Decimal counter value
uint8_t cnt1 = 0;   // For the second Display we need to add an other global variable

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

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



       /* Configure 8-bit Timer/Counter0
     * Set prescaler and enable overflow interrupt */

       TIM0_overflow_4ms();// we set an other timer and each 4ms we change the position
           TIM0_overflow_interrupt_enable();

       /* Configure 16-bit Timer/Counter1
     * Set prescaler and enable overflow interrupt */
              TIM1_overflow_1s();
              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)
{

    cnt0++;
        if(cnt0>=10){

              cnt0=0;// we have to reset it when the first Display reached the maximum

              cnt1++; //but in the same time we have to increment the next counter.
        if(cnt1>=6){
               cnt1=0; //if the second Display reach the maximum as well it has to be
reseted too.
        }
       }
}


 ISR(TIMER0_OVF_vect)
 {
        //  SEG_clear();
       static uint8_t pos = 0;//we use static variable to keep remember the current
position

              uint8_t display = 0;
       if(pos==0)
       {
              SEG_update_shift_regs(cnt0,display);
              pos=1;
       }
       else
       {
              display++;
              SEG_update_shift_regs(cnt1,display);
              pos=0;
       }



 }
```

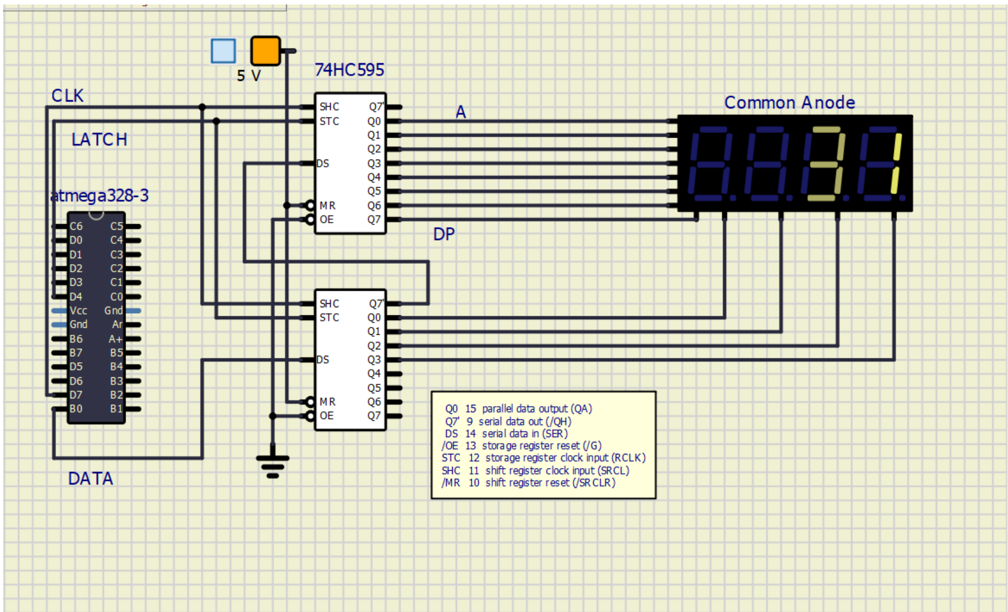

5 V
74HC595
CLK
LATCH
atmega328-3
A
DP
Common Anode
DATA
SHC
STC
DS
MR
OE
Q7
Q0
Q1
Q2
Q3
Q4
Q5
Q6
C6
D0
D1
D2
D3
D4
Vcc
Gnd
B6
B7
D5
D6
D7
B0
C5
C4
C3
C2
C1
C0
Gnd
Ar
A+
B5
B4
B3
B2
B1
Q0 15 parallel data output (QA)
Q7 9 serial data out (/QH)
DS 14 serial data in (SER)
/OE 13 storage register reset (/G)
STC 12 storage register clock input (RCLK)
SHC 11 shift register clock input (SRCL)
/MR 10 shift register reset (/SRCLR)

Figure 1:Screenshot of SimulIDE circuit

Listing of snake cycling application main.c

```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.
 */

/*Global variable*/


uint8_t cnt0 = 0;    // Decimal counter value
uint8_t cnt1 = 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)
{
       //Snake on one Display
       cnt0++;
       if (cnt0>=6)

       cnt0=0;
    SEG_update_shift_regs(cnt0,0);

}
```

Listing of the segment.c file for the snake application

```
/***********************************************************************
 *
 * 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"
uint8_t clearsegments = 0;

/* Variables ---------------------------------------------------------*/
// Active-low digit 0 to 9



uint8_t segment_value[] = {
       // abcdefgDP

       0b01111111,     // Segment A
       0b10111111,     // Segment B
       0b11011111,     // Segment C
       0b11101111,     // Segment D
       0b11110111,     // Segment E
       0b11111011,     // Segment F
       0b01111111,     // Segment A

};


uint8_t segment_position[] = {
       // p0p1p2p3
              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;



        if(clearsegments==0){
         segments = segment_value[segments];     //  A,B,C,D,E,F... snake
         position = segment_position[position];  // 0, 1, 2, 3
               }
        else if(clearsegments==1){
               segments=0b11111111;// in order to Turn off the all segments we set all
bit 1 because of aktiv low connection

                }
    // Pull LATCH, CLK, and DATA low
       GPIO_write_low(&PORTD,SEGMENT_LATCH);    // LATCH
       GPIO_write_low(&PORTD,SEGMENT_CLK);             // CLK
       GPIO_write_low(&PORTB,SEGMENT_DATA);     // 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 % 2)==0) // LSB is 0
                     GPIO_write_low(&PORTB, SEGMENT_DATA);
              else
                     GPIO_write_high(&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 % 2)==0)// LSB is 0

                    GPIO_write_low(&PORTB, SEGMENT_DATA);

              else
                    GPIO_write_high(&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 */


void SEG_clear()
{

       clearsegments=1;

}

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

void SEG_clk_2us ()
{
              GPIO_write_high(&PORTD,SEGMENT_CLK);      // CLK
              _delay_us(1);
              GPIO_write_low(&PORTD,SEGMENT_CLK);
              _delay_us(1);


}
```