Session 5

-

Display devices, 7-segment display

Author: Guillermo Cortés Orellana

Teacher: Tomáš Frýza

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

Lab assignment

1. **Preparation tasks**

- Table with segments values for display 0 to 9 on a common 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

- In your words, describe the difference between Common Cathode and Common Anode 7-segment display

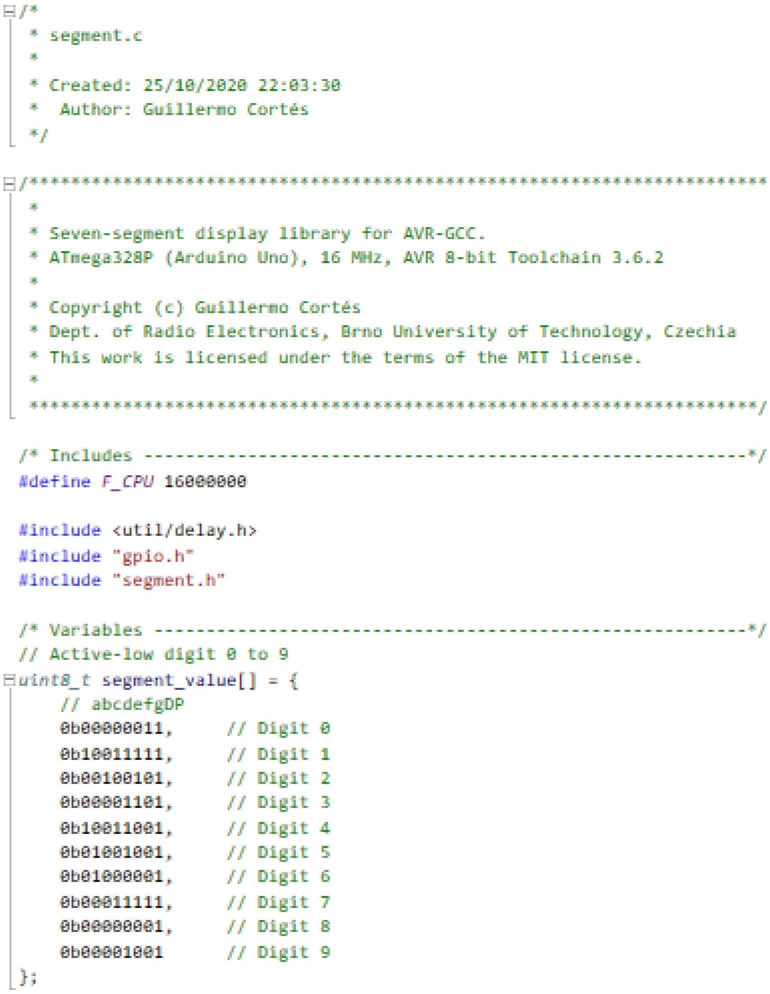
Common Cathode 7-segment display, has the cathodes of 7 segments connected to each other, instead of **Common Annode 7-segment display,** which has the anodes of 7 segments connected to each other.

In addition, **Common Cathode** turn on the LED with ‘1’ and turn off with ‘0’, as opposed to **Common Annode,** which turn on the LED with ‘0’ and turn off with ‘1’.

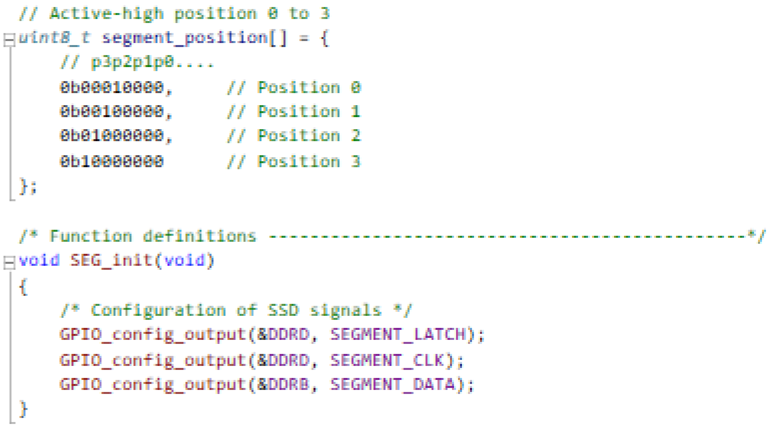
2. **7-segment library**

- Listing of library source file ***segment.c***

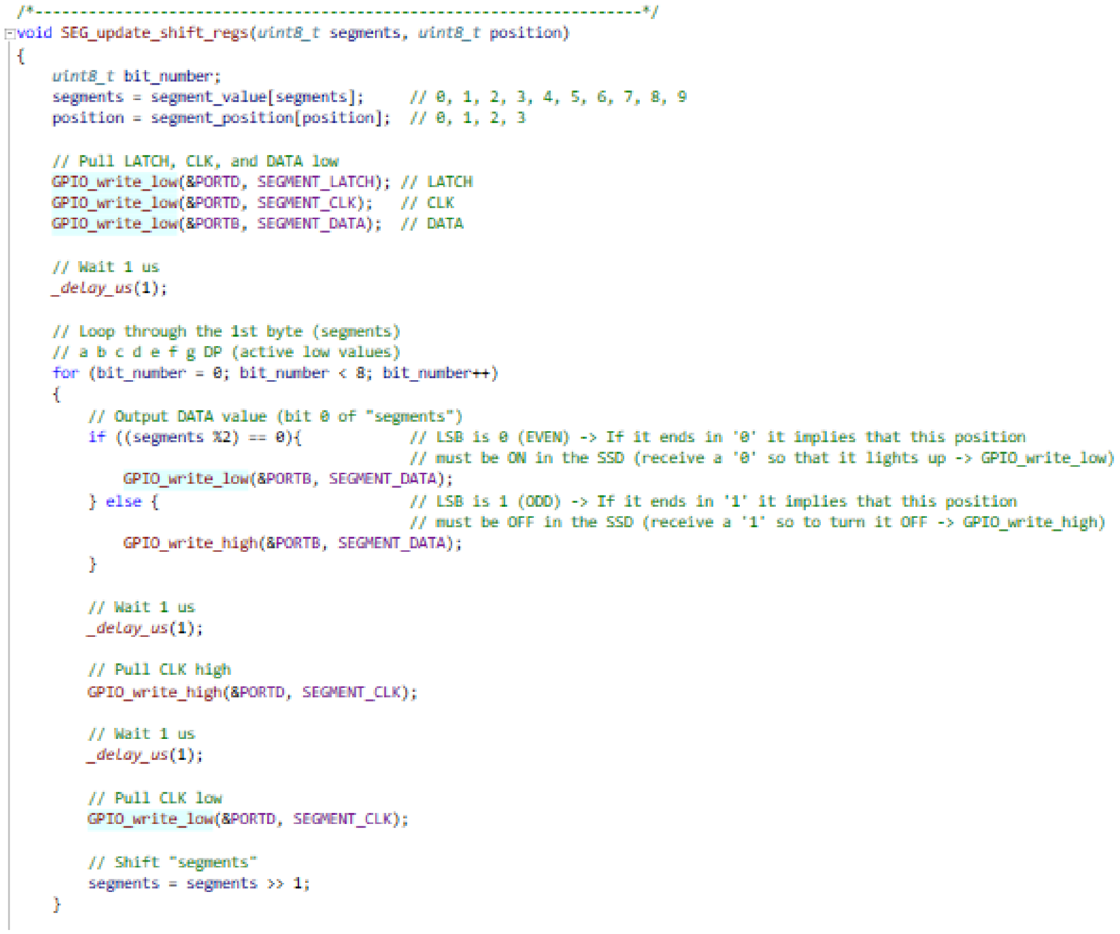
<u>Note</u>: `SEG_clk_2us()` function is designed so that it will generate 1 period of a clock signal with a **frequency of 800kHz** such as it is explained in the section ‘Experiments on your own’



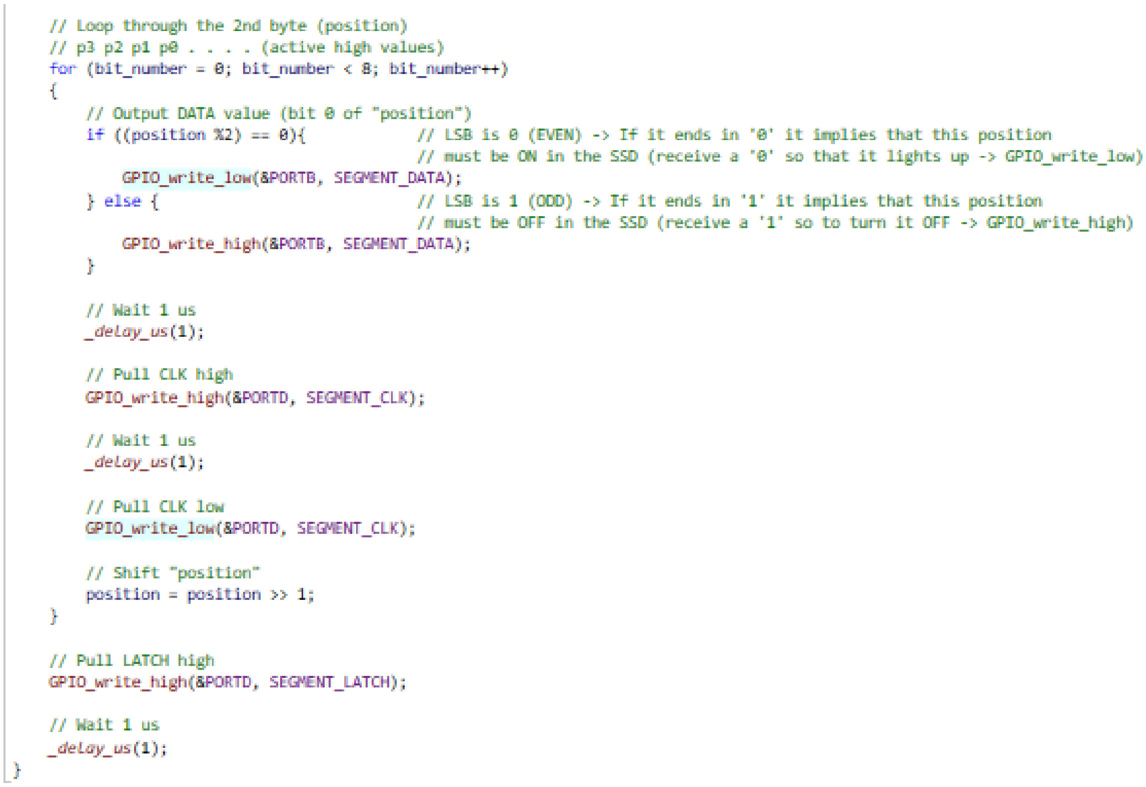
```c
/*
 * segment.c
 *
 * Created: 25/10/2020 22:03:30
 *  Author: Guillermo Cortés
 */

/***********************************************************************
 *
 * Seven-segment display library for AVR-GCC.
 * ATmega328P (Arduino Uno), 16 MHz, AVR 8-bit Toolchain 3.6.2
 *
 * Copyright (c) Guillermo Cortés
 * 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
};
```



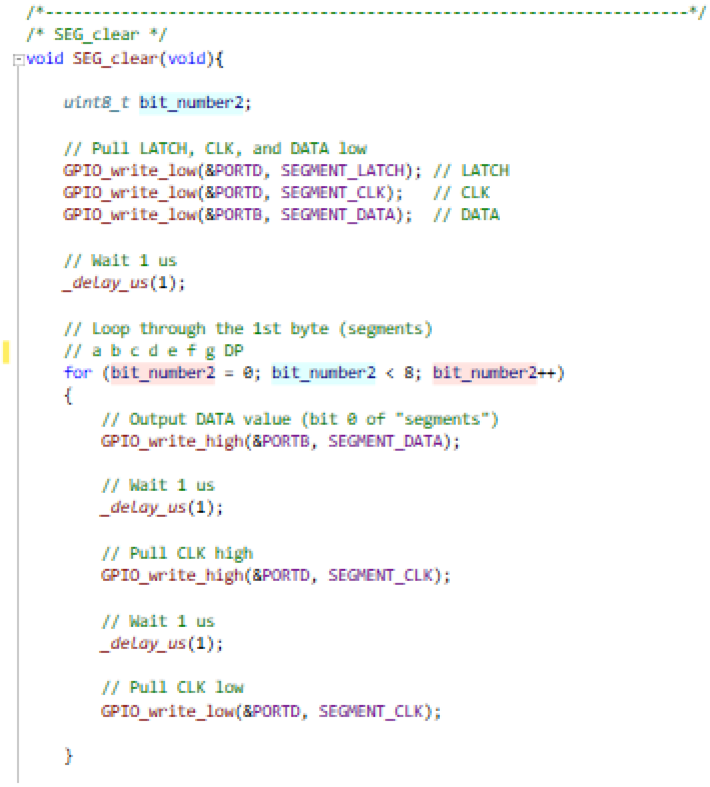
```c
// Active-high position 0 to 3
uint8_t segment_position[] = {
    // p3p2p1p0....
    0b00010000,     // Position 0
    0b00100000,     // Position 1
    0b01000000,     // Position 2
    0b10000000      // Position 3
};

/* 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);
}
```



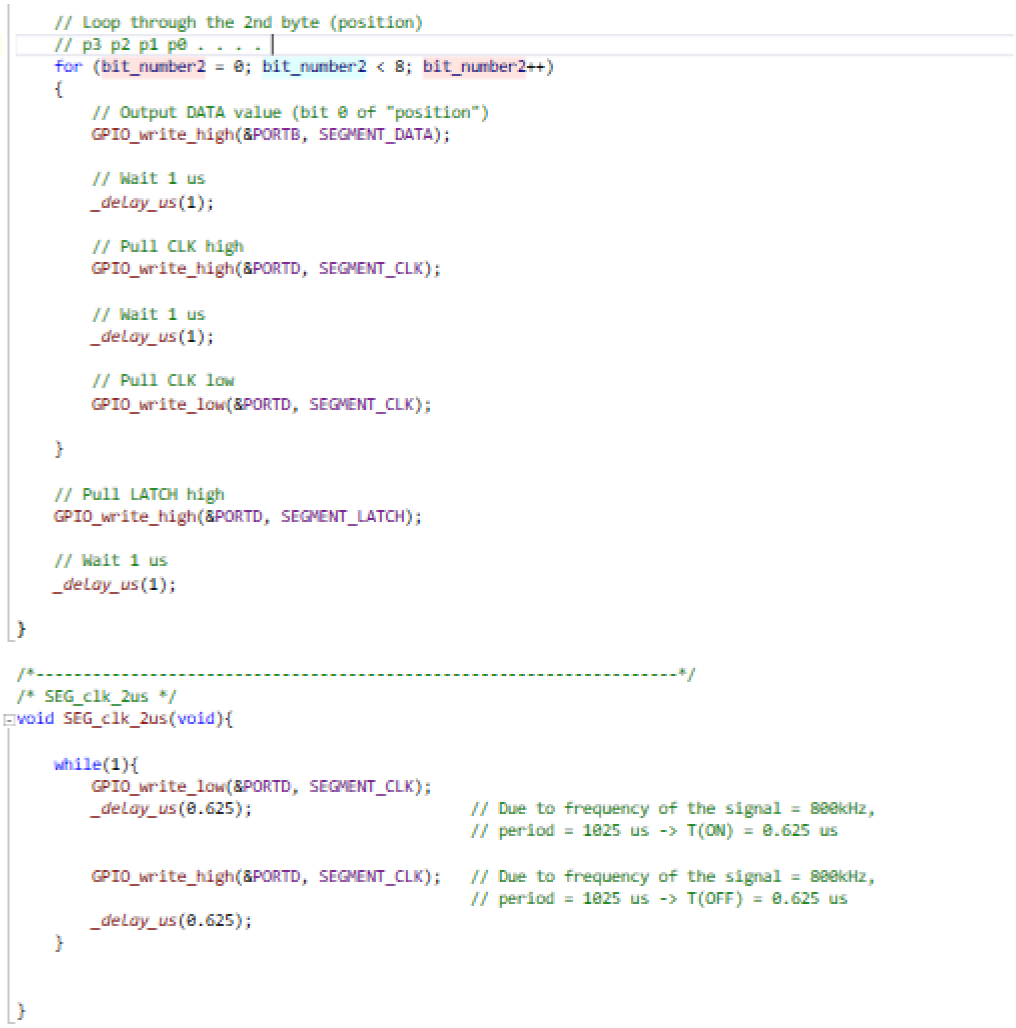
```c
/*-------------------------------------------------------------------*/
void SEG_update_shift_regs(uint8_t segments, uint8_t position)
{
    uint8_t bit_number;
    segments = segment_value[segments];     // 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
    position = segment_position[position];  // 0, 1, 2, 3

    // 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 (EVEN) -> If it ends in '0' it implies that this position
                                            // must be ON in the SSD (receive a '0' so that it lights up -> GPIO_write_low)
            GPIO_write_low(&PORTB, SEGMENT_DATA);
        } else {                            // LSB is 1 (ODD) -> If it ends in '1' it implies that this position
                                            // must be OFF in the SSD (receive a '1' so to turn it OFF -> GPIO_write_high)
            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 (EVEN) -> If it ends in '0' it implies that this position
                                            // must be ON in the SSD (receive a '0' so that it lights up -> GPIO_write_low)
            GPIO_write_low(&PORTB, SEGMENT_DATA);
        } else {                            // LSB is 1 (ODD) -> If it ends in '1' it implies that this position
                                            // must be OFF in the SSD (receive a '1' so to turn it OFF -> GPIO_write_high)
            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(void){

    uint8_t bit_number2;

    // 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
    for (bit_number2 = 0; bit_number2 < 8; bit_number2++)
    {
        // Output DATA value (bit 0 of "segments")
        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);

    }
```

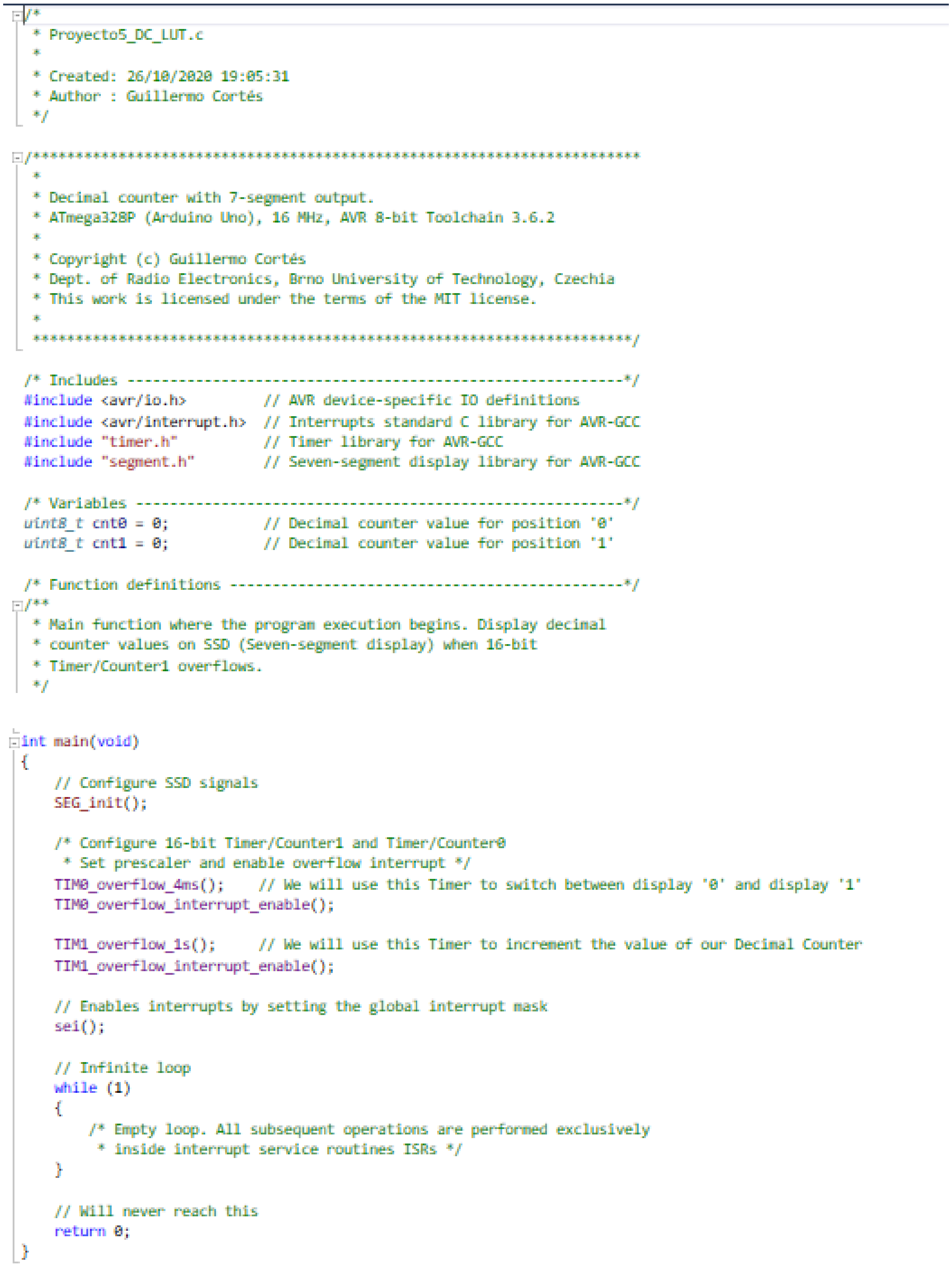


```
    // Loop through the 2nd byte (position)
    // p3 p2 p1 p0 . . . .
    for (bit_number2 = 0; bit_number2 < 8; bit_number2++)
    {
        // Output DATA value (bit 0 of "position")
        GPIO_write_high(&PORTB, SEGMENT_DATA);

        // 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);

    }

    // Pull LATCH high
    GPIO_write_high(&PORTD, SEGMENT_LATCH);

    // Wait 1 us
    _delay_us(1);

}

/*--------------------------------------------------------------------*/
/* SEG_clk_2us */
void SEG_clk_2us(void){

    while(1){
        GPIO_write_low(&PORTD, SEGMENT_CLK);
        _delay_us(0.625);                       // Due to frequency of the signal = 800kHz,
                                                // period = 1025 us -> T(ON) = 0.625 us

        GPIO_write_high(&PORTD, SEGMENT_CLK);   // Due to frequency of the signal = 800kHz,
                                                // period = 1025 us -> T(OFF) = 0.625 us
        _delay_us(0.625);
    }


}
```

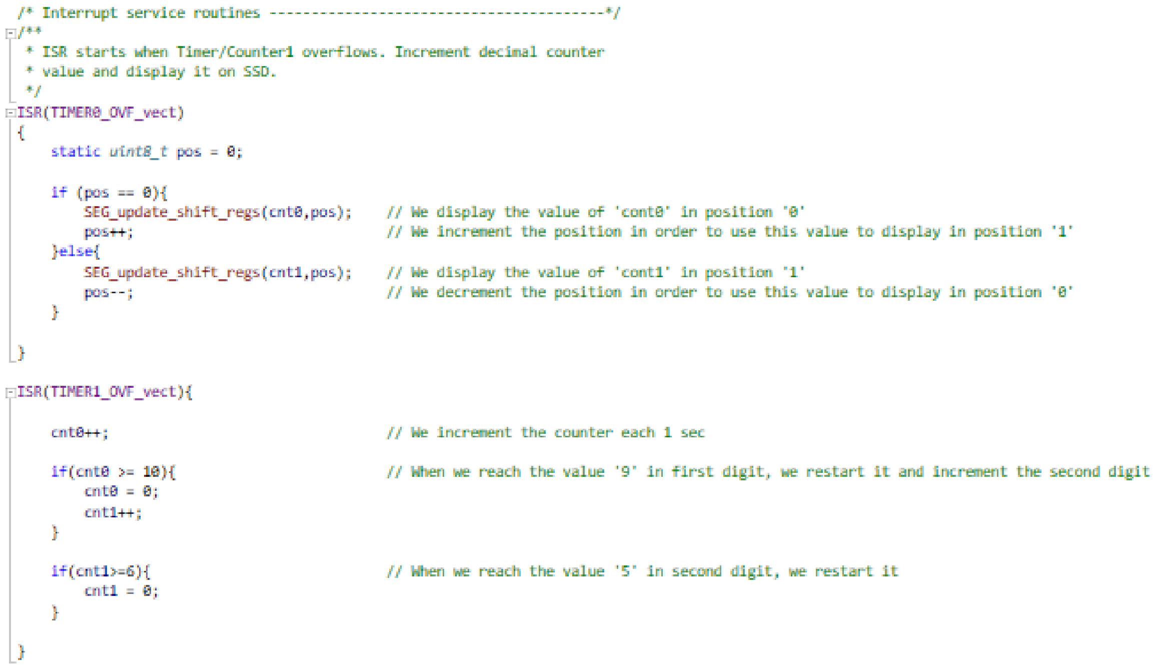
You can find the code on my GitHub:

https://github.com/GuicoRM/Digital-Electronics-2

- Listing of decimal counter application ***main.c*** (two digits, from 00 to 59)



```c
/*
 * Proyecto5_DC_LUT.c
 *
 * Created: 26/10/2020 19:05:31
 * Author : Guillermo Cortés
 */

/***********************************************************************
 *
 * Decimal counter with 7-segment output.
 * ATmega328P (Arduino Uno), 16 MHz, AVR 8-bit Toolchain 3.6.2
 *
 * Copyright (c) Guillermo Cortés
 * 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

/* Variables ---------------------------------------------------------*/
uint8_t cnt0 = 0;           // Decimal counter value for position '0'
uint8_t cnt1 = 0;           // Decimal counter value for position '1'

/* 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();

    /* Configure 16-bit Timer/Counter1 and Timer/Counter0
     * Set prescaler and enable overflow interrupt */
    TIM0_overflow_4ms();    // We will use this Timer to switch between display '0' and display '1'
    TIM0_overflow_interrupt_enable();

    TIM1_overflow_1s();     // We will use this Timer to increment the value of our Decimal Counter
    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;
}
```

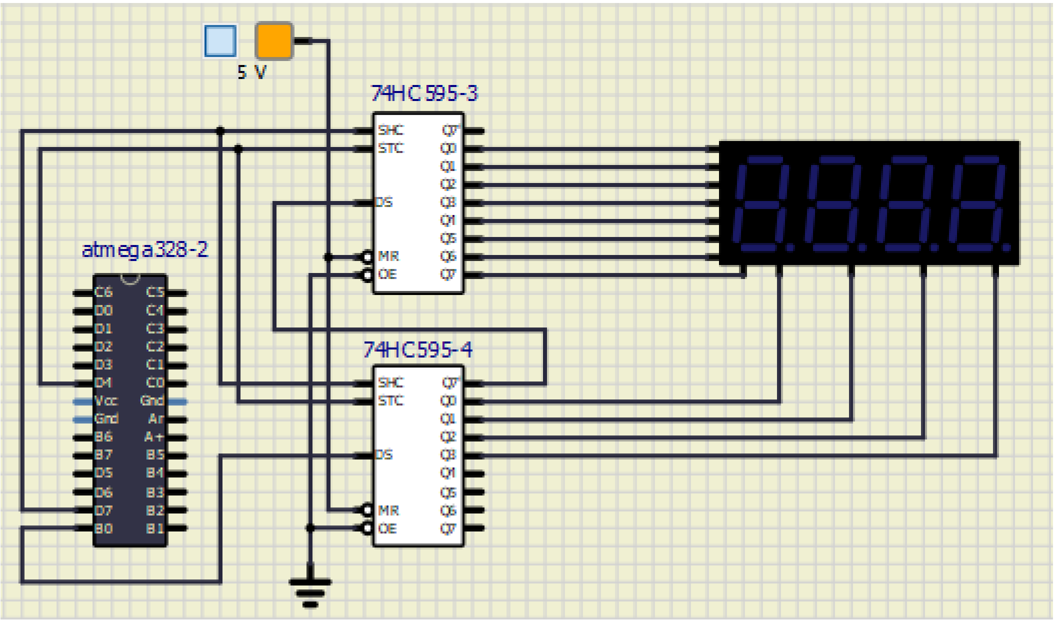


```c
/* Interrupt service routines ----------------------------------------*/
/**
 * ISR starts when Timer/Counter1 overflows. Increment decimal counter
 * value and display it on SSD.
 */
ISR(TIMER0_OVF_vect)
{
    static uint8_t pos = 0;

    if (pos == 0){
        SEG_update_shift_regs(cnt0,pos);    // We display the value of 'cont0' in position '0'
        pos++;                              // We increment the position in order to use this value to display in position '1'
    }else{
        SEG_update_shift_regs(cnt1,pos);    // We display the value of 'cont1' in position '1'
        pos--;                              // We decrement the position in order to use this value to display in position '0'
    }

}

ISR(TIMER1_OVF_vect){

    cnt0++;                                 // We increment the counter each 1 sec

    if(cnt0 >= 10){                         // When we reach the value '9' in first digit, we restart it and increment the second digit
        cnt0 = 0;
        cnt1++;
    }

    if(cnt1>=6){                            // When we reach the value '5' in second digit, we restart it
        cnt1 = 0;
    }

}
```

You can find the code on my GitHub:

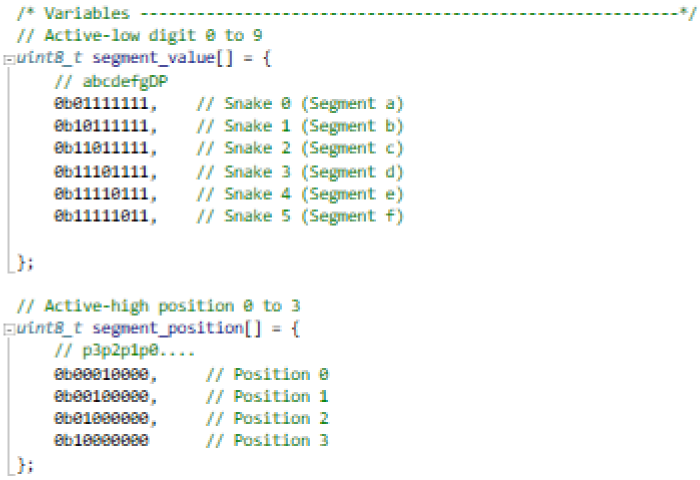
https://github.com/GuicoRM/Digital-Electronics-2

- Screenshot of SimulIDE circuit



3. **Snake**

- Look-up table with snake definiton

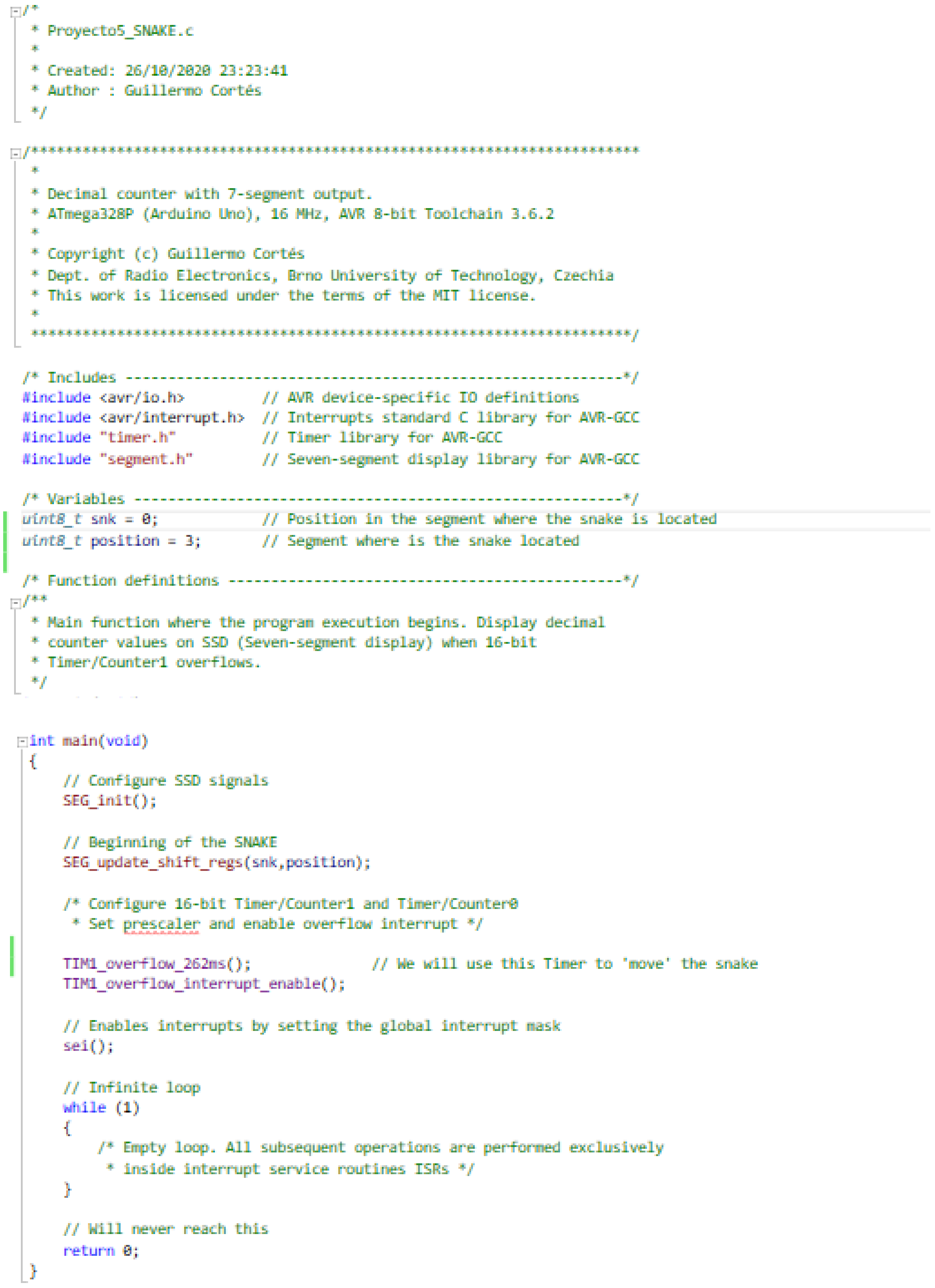


```
/* Variables ---------------------------------------------------------*/
// Active-low digit 0 to 9
uint8_t segment_value[] = {
    // abcdefgDP
    0b01111111,    // Snake 0 (Segment a)
    0b10111111,    // Snake 1 (Segment b)
    0b11011111,    // Snake 2 (Segment c)
    0b11101111,    // Snake 3 (Segment d)
    0b11110111,    // Snake 4 (Segment e)
    0b11111011,    // Snake 5 (Segment f)

};

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

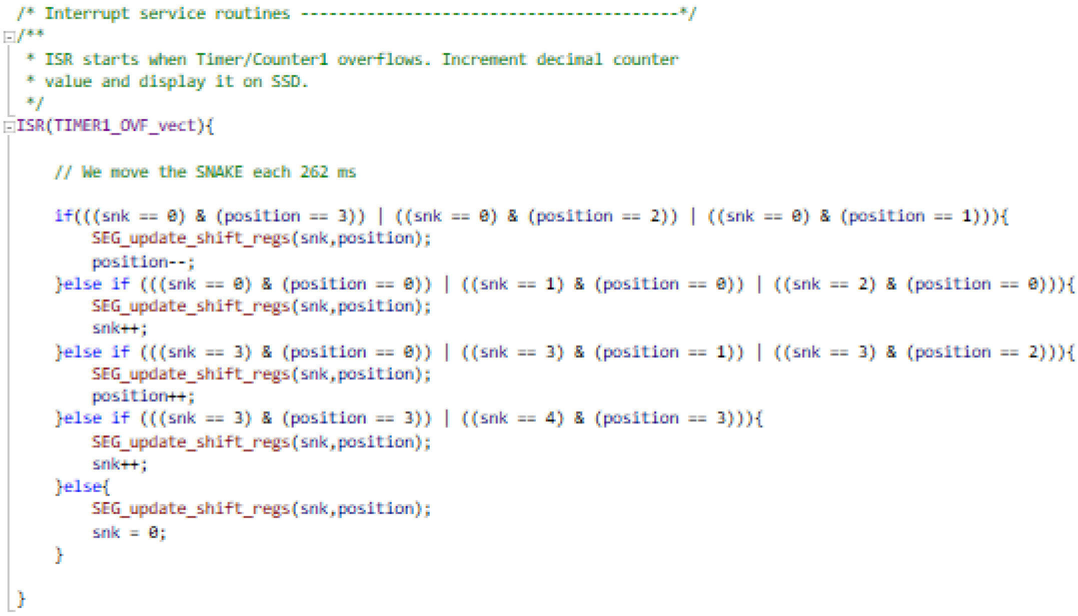
You can find the code on my GitHub:

https://github.com/GuicoRM/Digital-Electronics-2

- Listing of snake cycling application ***main.c*** (four digits snake)



```c
/*
 * Proyecto5_SNAKE.c
 *
 * Created: 26/10/2020 23:23:41
 * Author : Guillermo Cortés
 */

/***********************************************************************
 *
 * Decimal counter with 7-segment output.
 * ATmega328P (Arduino Uno), 16 MHz, AVR 8-bit Toolchain 3.6.2
 *
 * Copyright (c) Guillermo Cortés
 * 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

/* Variables ---------------------------------------------------------*/
uint8_t snk = 0;            // Position in the segment where the snake is located
uint8_t position = 3;       // Segment where is the snake located

/* 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();

    // Beginning of the SNAKE
    SEG_update_shift_regs(snk,position);

    /* Configure 16-bit Timer/Counter1 and Timer/Counter0
     * Set prescaler and enable overflow interrupt */

    TIM1_overflow_262ms();              // We will use this Timer to 'move' the snake
    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;
}
```



```c
/* Interrupt service routines ----------------------------------------*/
/**
 * ISR starts when Timer/Counter1 overflows. Increment decimal counter
 * value and display it on SSD.
 */
ISR(TIMER1_OVF_vect){

    // We move the SNAKE each 262 ms

    if(((snk == 0) & (position == 3)) | ((snk == 0) & (position == 2)) | ((snk == 0) & (position == 1))){
        SEG_update_shift_regs(snk,position);
        position--;
    }else if (((snk == 0) & (position == 0)) | ((snk == 1) & (position == 0)) | ((snk == 2) & (position == 0))){
        SEG_update_shift_regs(snk,position);
        snk++;
    }else if (((snk == 3) & (position == 0)) | ((snk == 3) & (position == 1)) | ((snk == 3) & (position == 2))){
        SEG_update_shift_regs(snk,position);
        position++;
    }else if (((snk == 3) & (position == 3)) | ((snk == 4) & (position == 3))){
        SEG_update_shift_regs(snk,position);
        snk++;
    }else{
        SEG_update_shift_regs(snk,position);
        snk = 0;
    }

}
```

You can find the code on my GitHub:

https://github.com/GuicoRM/Digital-Electronics-2