VUT FEKT

Name and Surname: *Kreshnik Shala*

Person ID: *226108*

[BPA-DE2] Digital Electronics 2

Assignment 5

Date: Tuesday, October 27, 2020

GitHub: *https://github.com/ShalaKreshnik*

1 Table of Contents

Lab assignment 5

VUT FEKT

Name and Surname: *Kreshnik Shala*

Person ID: *226108*

[BPA-DE2] Digital Electronics 2

Assignment 5

Date: Tuesday, October 27, 2020

GitHub: *https://github.com/ShalaKreshnik*

1.1 Preparation tasks. Submit:

1.1.1 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

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

In Common Anode configuration all the anodes of LEDs are connected together, to turn ON the individual segments logic ‘0’ is applied. But in Common Cathode configuration all cathodes of LEDs are connected together and to turn ON the individual segments logic 1 or HIGH is applied.

Name and Surname: *Kreshnik Shala*
Person ID: *226108*

Date: Tuesday, October 27, 2020

1.2 7-segment library. Submit:

1.2.1 Listing of library source file segment.c:

```
/*////////////////////////////////////////////////////////////////////////////////
///                                                                              ///
/// VUT FEKT                                    Name and Surname: Kreshnik Shala ///
/// [BPA-DE2] Digital Electronics 2             Person ID: 226108                ///
/// Date: Tuesday, October 27, 2020                                              ///
/// GitHub: https://github.com/ShalaKreshnik                                     ///
///                                                                              ///
*////////////////////////////////////////////////////////////////////////////////
/* Includes ------------------------------------------------------------*/
#define F_CPU 16000000
#define OFF 0b11111111
#include <util/delay.h>
#include "gpio.h"
#include "segment.h"

/* Variables -----------------------------------------------------------*/
// Active-low digits 0 to 9
uint8_t clear_flag = 0;
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

uint8_t segment_value_pos_3[] = {
       // abcdefgDP
       0b00000010,      // Digit 0
       0b10011110,      // Digit 1
       0b00100100,      // Digit 2
       0b00001100,      // Digit 3
       0b10011000,      // Digit 4
       0b01001000,      // Digit 5
       0b01000000,      // Digit 6
       0b00011110,      // Digit 7
       0b00000000,      // Digit 8
       0b00001000};     // 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 (clear_flag == 0) // iF CLEAR CALL HAS NOT BEEN MADE
       {
       if (position == 2) // Third ssd
       {
              segments = segment_value_pos_3[segments]; // 0, 1, ..., 9
       }
       else
       {
              segments = segment_value[segments];          // 0, 1, ..., 9
       }
       }

       else if (clear_flag == 1) // IF CLEAR CALL HAS BEEN MADE
       {
              segments = OFF;      // TURN OFF THE SEGMENTS
       }

        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 0of "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);  // LATCH
    // Wait 1 us
       _delay_us(1);

}
/*-------------------------------------------------------------------*/
/* SEG_clear */
void SEG_clear()
{
       clear_flag = 1; // Will be used to TURN OFF ALL THE SEGMENTS (NO SEGMENT WILL BE
ON)
}
/*-------------------------------------------------------------------*/
/* SEG_clk_2us */
void SEG_clk_2us()
{
       /*1 instruction performed in(by a clock of 800kHz freq) = 1/800k = 1.25us;
```

```
       we need 2us delay. No of instructions = 2u/1.25u = 1.6
       TCNT0 value = 256 - 1.6 = 254.4 (Subtracting from 256 because 8 bit timer can
perform maximum 256 instructions)*/

       TCNT0 = 254;
       TCCR0A = 0; //Normal mode
       TCCR0B = 2; //prescaler of 1
       while((TIFR0&(1<<OCF0A))==0)
       { } //wait until OCF0A is set
       TCCR0B = 0; //stop timer0
       TIFR0 = 1<<OCF0A;//clear flag
}
```

1.2.2 Listing of decimal counter application main.c

```
/*////////////////////////////////////////////////////////////////////////////////
///                                                                              ///
/// VUT FEKT                                  Name and Surname: Kreshnik Shala   ///
/// [BPA-DE2] Digital Electronics 2           Person ID: 226108                  ///
/// Date: Tuesday, October 27, 2020                                              ///
/// GitHub: https://github.com/ShalaKreshnik                                     ///
///                                                                              ///
*////////////////////////////////////////////////////////////////////////////////
/* 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-
#include "Pinchange_interrupt.h"

#define BTN PC1
/* Variables -----------------------------------------------------------*/

uint8_t cnt0 = 0;                // Decimal counter value
uint8_t cnt1 = 0;                // TEN counter value
uint8_t cnt2 = 0;                // HUNDRED counter value
uint8_t cnt3 = 0;                // THOUSAND counter value
static uint8_t pos = 0;          // POSITION OF SSD

/* 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
```

```
    /* Configure 16-bit Timer/Counter1
     * Set prescaler and enable overflow interrupt */
       TIM1_overflow_1s ();
       TIM1_overflow_interrupt_enable ();

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

       PCINT9_Enable();
       PCINT_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)
{
    // WRITE YOUR CODE HERE
       cnt0++;

       if (cnt0 >= 10) // Reached 10 seconds
       {
              cnt0 = 0;
              cnt1++;
       }
       if (cnt1 >= 6) // If the stopwatch has reached 60 seconds
       {
              cnt0 = 0; // Decimal value 0
              cnt1 = 0; // Ten value 0 (Ten value of seconds)
              cnt2++; // Increase in minute (60 seconds has passed)
       }
       if (cnt2 >= 10) // If 10 minute have passed
       {
              cnt0 = 0;
              cnt1 = 0;
              cnt2 = 0; // Decimal value of minutes = 0
```

Name and Surname: *Kreshnik Shala*
Person ID: *226108*

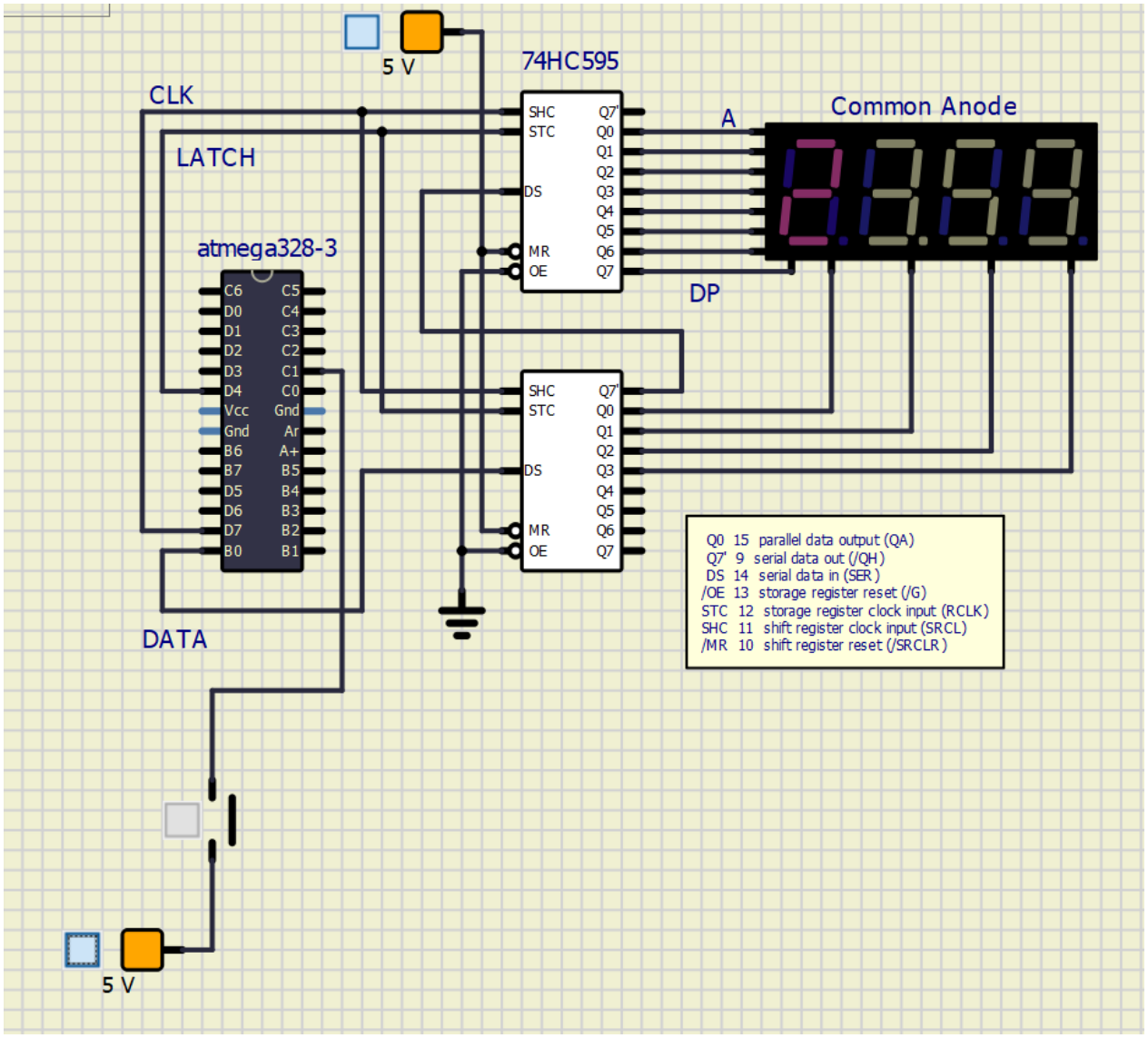
Date: Tuesday, October 27, 2020

```
                cnt3++; // Ten value of minutes increased
        }
        if (cnt3 >= 6) // IF 60 minutes have passed
        {
                // INITIALIZE THE COMPLETE WATCH STORE 00.00

                cnt0 = 0;
                cnt1 = 0;
                cnt2 = 0;
                cnt3 = 0;
        }
}
ISR(TIMER0_OVF_vect)
{
        if (pos == 0)
        {
                SEG_update_shift_regs(cnt0, pos);
                pos = 1; // Ready to move to second SSD
        }
        else if (pos == 1)
        {
                SEG_update_shift_regs(cnt1, pos);
                pos = 2; // Position changed to third SSD
        }
        else if (pos == 2)
        {
                SEG_update_shift_regs(cnt2, pos);
                pos = 3; // POSITION CHANGED TO 4TH SSD
        }
        else if (pos == 3)
        {
                SEG_update_shift_regs(cnt3, pos);
                pos = 0; // POSITION CHANGED TO FIRST SSD
        }
}

ISR (PCINT1_vect)
{
                SEG_clear(); // CLEAR SSD FUNCTION
}
```

Name and Surname: *Kreshnik Shala*
Person ID: *226108*

Date: Tuesday, October 27, 2020

1.2.3 Screenshot of SimullDE circuits:



Name and Surname: *Kreshnik Shala*
Person ID: *226108*

Date: Tuesday, October 27, 2020

1.3 Snake. Submit:

1.3.1 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
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
a	0	1	1	1	1	1	1	1

1.3.2 Listing of your snake cycling application main.c:

```
/*////////////////////////////////////////////////////////////////////////////////
///                                                                              ///
/// VUT FEKT                                  Name and Surname: Kreshnik Shala   ///
/// [BPA-DE2] Digital Electronics 2           Person ID: 226108                  ///
/// Date: Tuesday, October 27, 2020                                              ///
/// GitHub: https://github.com/ShalaKreshnik                                     ///
///                                                                              ///
*////////////////////////////////////////////////////////////////////////////////

/* 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-
#include "Pinchange_interrupt.h"

/* Variables -----------------------------------------------------------*/

uint8_t cnt0 = 0;                // Decimal counter value
uint8_t shift = 0;               // Will use 0 or 1 to Shift LEDs
/* 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)
```

Name and Surname: *Kreshnik Shala*
Person ID: *226108*

Date: Tuesday, October 27, 2020

```
{

    // Configure SSD signals
    SEG_init();

    // Test of SSD: display number '3' at position 0
       SEG_update_shift_regs(cnt0, 0);

       /* 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)
{
    // WRITE YOUR CODE HERE
       cnt0++;

       if (cnt0 == 4)
       {
              shift = 1; // Using this to shift the snake to the next LED
       }

       if (cnt0 >= 8)
       {
              cnt0 = 0; // Coming back to its original value
              shift = 0; // Shifting back to first LED
       }

       SEG_update_shift_regs(cnt0, shift);
}
```