1. **Preparation tasks (done before the lab at home):**
   * **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.**

7-segment display with common cathode has common cathode for LEDs of all 7 segments and decimal point. So any segment can be lit seting its anode pin to 0.

Common anod is oposite. Segments are lit with 1.

1. **7-segment library. Submit:**
   * **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"

/\* 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

/\* 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, ..., 9

position = segment\_position[position]; // 0, 1, 2, 3

// 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) == 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)

{

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()

{

*uint8\_t* seg\_off = 0b00000000;

*uint8\_t* data\_clear = 0b11111111;

*uint8\_t* bit\_number;

GPIO\_write\_low(&PORTD,SEGMENT\_LATCH);

GPIO\_write\_low(&PORTD,SEGMENT\_CLK);

GPIO\_write\_low(&PORTB,SEGMENT\_DATA);

for (bit\_number = 0; bit\_number < 8; bit\_number++)

{

// Output DATA value (bit 0 of "position")

if ((data\_clear & 1) == 0)

{

GPIO\_write\_low(&PORTB,SEGMENT\_DATA);

}

else

{

GPIO\_write\_high(&PORTB,SEGMENT\_DATA);

}

*\_delay\_us*(1); // Wait 1 us

GPIO\_write\_high(&PORTD,SEGMENT\_CLK); // Pull CLK high

*\_delay\_us*(1); // Wait 1 us

GPIO\_write\_low(&PORTD,SEGMENT\_CLK); // Pull CLK low

data\_clear = data\_clear >> 1;

}

for (bit\_number = 0; bit\_number < 8; bit\_number++)

{

// Output DATA value (bit 0 of "position")

if ((seg\_off & 1) == 0)

{

GPIO\_write\_low(&PORTB,SEGMENT\_DATA);

}

else

{

GPIO\_write\_high(&PORTB,SEGMENT\_DATA);

}

*\_delay\_us*(1); // Wait 1 us

GPIO\_write\_high(&PORTD,SEGMENT\_CLK); // Pull CLK high

*\_delay\_us*(1); // Wait 1 us

GPIO\_write\_low(&PORTD,SEGMENT\_CLK); // Pull CLK low

seg\_off = seg\_off >> 1;

}

GPIO\_write\_high(&PORTD,SEGMENT\_LATCH); // Pull LATCH high

*\_delay\_us*(1); // Wait 1 us

}

/\*--------------------------------------------------------------------\*/

/\* SEG\_clk\_2us \*/

void SEG\_clk\_2us()

{

GPIO\_write\_high(&PORTD,SEGMENT\_CLK); // set pin "Segment\_clk" to 1

*\_delay\_us*(1); // Wait 1 us

GPIO\_write\_low(&PORTD,SEGMENT\_CLK); // set pin "Segment\_clk" to 0

*\_delay\_us*(1); // Wait 1 us

}

* + **Listing of decimal counter application main.c (at least two-digit decimal counter, ie. from 00 to 59),**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* 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();

/\* Configure 16-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\_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/Counter0 overflows. Display value on SSD

\*/

ISR(TIMER0\_OVF\_vect)

{

static *uint8\_t* position = 0;

if (position == 0)

{

SEG\_update\_shift\_regs(singles, 0);

position = 1;

}

else

{

SEG\_update\_shift\_regs(decimals, 1);

position = 0;

}

}

/\*\*

\* ISR starts when Timer/Counter1 overflows. Increment decimal counter

\*/

ISR(TIMER1\_OVF\_vect)

{

singles++;

if(singles > 9)

{

singles = 0;

decimals++;

if(decimals > 5)

{

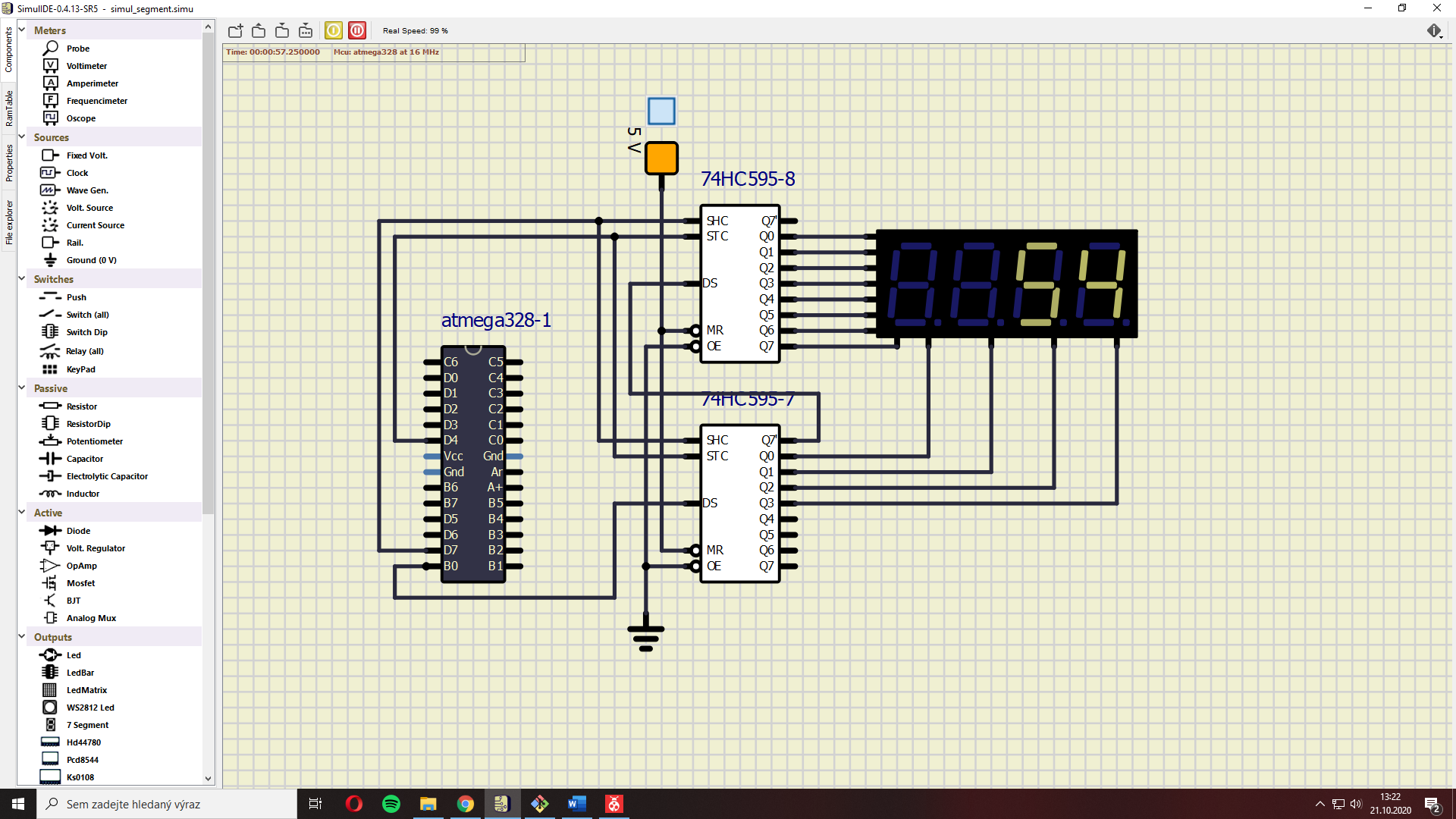
decimals = 0;

}

}

}

* + **Screenshot of SimulIDE circuit.**

****

1. **Snake. Submit:**
   * **Look-up table with snake definition,**

|  |  |  |
| --- | --- | --- |
| **number** | **snake position** | **7seg binary code** |
| 0 |  | 0b11111111 |
| 1 | ¯ | 0b01111111 |
| 2 | ‘ | 0b10111111 |
| 3 | , | 0b11011111 |
| 4 | \_ | 0b11101111 |
| 5 | , | 0b11110111 |
| 6 | ‘ | 0b11111011 |

* + **Listing of your snake cycling application 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

*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();

/\* Configure 16-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;

}

/\* Interrupt service routines ----------------------------------------\*/

/\*\*

\* ISR starts when Timer/Counter0 overflows. Display snake on SSD

\*/

ISR(TIMER0\_OVF\_vect)

{

static *uint8\_t* position = 0;

if (position == 0)

{

SEG\_update\_shift\_regs(singles, 0);

position = 1;

}

else

{

SEG\_update\_shift\_regs(decimals, 1);

position = 0;

}

}

/\*\*

\* ISR starts when Timer/Counter1 overflows. Increment decimal counter

\*/

ISR(TIMER1\_OVF\_vect)

{

if(decimals == 0)

{

singles++;

if(singles > 4)

{

singles = 0;

decimals = 4;

}

}

else if(decimals > 3)

{

decimals++;

if (decimals > 6)

{

decimals = 1;

}

}

else if(decimals == 1)

{

decimals = 0;

singles = 1;

}

}