Lab assignment 5

# Preparation Task

1. **Binary values of the segments 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. **Common cathode vs anode**

In the common cathode display (cc), all the cathode connections of the 7-segments are joined together and connected to ground (logic ‘0’) and the individual segments are illuminated by the application of a High (logic ‘1’) signal via a current limiting resister to forward bias the individual Anode terminals. While in the common anodes of the LED segments are connected together to High (logic ‘1’) and the individual segments are illuminated by the application of the logic “0” (LOW) via a current limiting resistor to their Cathodes.

# 2. 7-segment library

* 1. **Listing of segment.c library**

/\*

\* segment.c

\*

\* Created: 10/21/2020 11:22:21

\* Author: masau

\*/

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

#define *F\_CPU* 16000000 // MCUs CPU frequency

#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

// Active-high position 0 to 3

*uint8\_t* segment\_position[] = {

// p3p2p1p0....

0b00010000, // Position 0

0b00100000, // Position 1

0b01000000, // Position 2

0b1000000}; // 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 \*/

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

/\* SEG\_clk\_2us \*/

* 1. **Listing of decimal counter main.c ( 00 to 59)**

/\*

\* segment.c

\*

\* Created: 10/21/2020 11:18:03

\* Author : masau

\*/

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

\*

\* 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* ones = 0;

*uint8\_t* tens = 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();

// Test of SSD: display number '3' at position 0

//SEG\_update\_shift\_regs(4, 0);

/\* Configure 8-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 value on SSD.

\*/

ISR(TIMER0\_OVF\_vect)

{

static *uint8\_t* position = 0;

if(position == 0)

{

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

position = 1;

}

else

{

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

position = 0;

}

}

/\*\*

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

\*/

ISR(TIMER1\_OVF\_vect)

{

ones++;

if(ones > 9)

{

ones = 0;

tens++;

if(tens > 5)

{

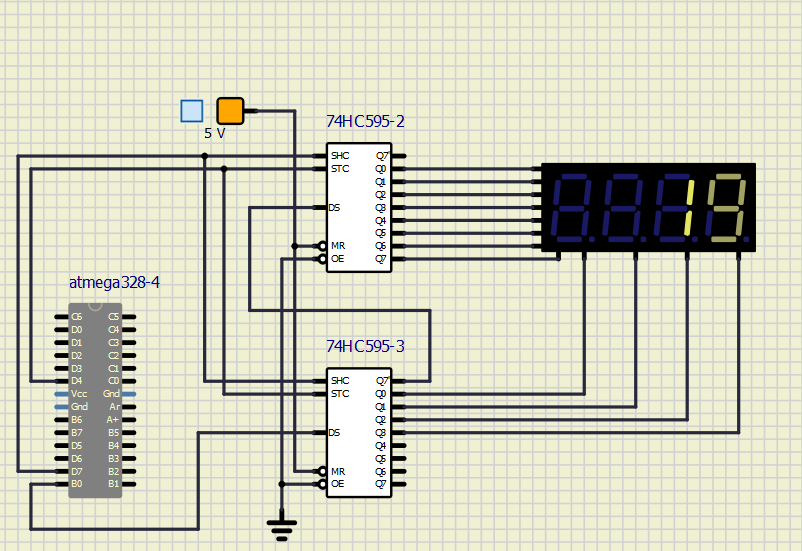
tens = 0;

}

}

}

**2.3 7-segment Circuit**



# Snake application

1. **Snake Look-up table**

| Segment | Binary code | snake position |
| --- | --- | --- |
| NON | 0b11111111 |  |
| A | 0b01111111 | - |
| B | 0b10111111 | I |
| C | 0b11011111 | I |
| D | 0b11101111 | \_ |
| E | 0b11110111 | I |
| F | 0b11111011 | I |

1. **Main.c**

/\*

\* snake.c

\*

\* Created: 10/21/2020 11:18:03

\* Author : masau

\*/

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

\*

\* 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* ones = 0; // first digit on the 4-digit 7-segment display

*uint8\_t* tens = 0; // second digit on the display

/\* 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

//SEG\_update\_shift\_regs(4, 0);

/\* Configure 8-bit Timer/Counter0

\* Set prescaler and enable overflow interrupt \*/

TIM0\_overflow\_1ms();

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 value on SSD.

\*/

ISR(TIMER0\_OVF\_vect)

{

static *uint8\_t* position = 0;

if(position == 0)

{

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

position = 1;

}

else

{

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

position = 0;

}

}

/\*\*

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

\*/

ISR(TIMER1\_OVF\_vect)

{

if(tens == 0)

{

ones++;

if(ones > 4)

{

ones = 0;

tens = 4;

}

}

else if (tens > 3)

{

tens++;

if (tens > 6)

{

tens = 1;

}

}

else if (tens == 1)

{

tens = 0;

ones = 1;

}

}