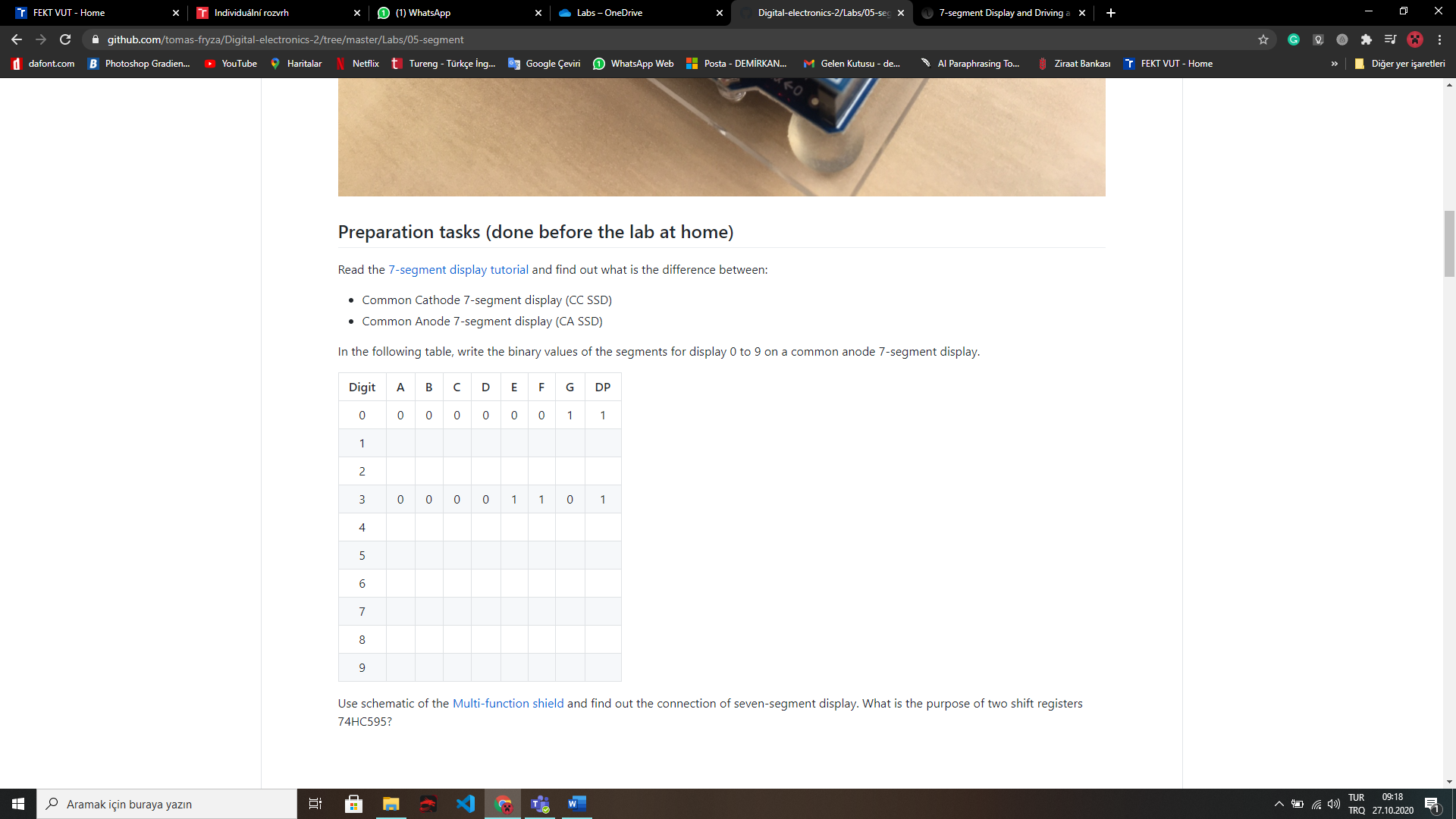
**DIGITAL ELECTRONICS 2 LAB ASSIGNMENT 5**

Name: Demirkan Korbey Baglamac

1. 

0 0 0 0 1 0 0 1

0 0 0 0 0 0 0 1

0 0 0 1 1 1 1 1

1 0 0 1 1 1 1 1

0 0 1 0 0 1 0 1

1 0 0 1 1 0 0 1

0 1 0 0 1 0 0 1

0 1 0 0 0 0 0 1

In Common Cathode, as you can understand from the name, for all the leds cathode terminal is same(common).

And in Common Anode this time for the all leds, anode terminal is same(common).

1. **Segment.c:**

/\* 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

};

// 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* input\_type)

{

*uint8\_t* bit\_number;

if(input\_type == 1) {

// Getting segment and position values from the arrays

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); // 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);

}

**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

/\* 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();

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

SEG\_update\_shift\_regs(cnt0, 0, 1);

SEG\_update\_shift\_regs(cnt0, 2, 1);

/\* 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();

// 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(TIMER0\_OVF\_vect)

{

static *uint8\_t* pos = 0;

if(pos == 0) {

SEG\_update\_shift\_regs(cnt0, pos, 1);

pos = 1;

}else {

SEG\_update\_shift\_regs(cnt1, pos, 1);

pos = 0;

}

}

/\*\*

\* 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;

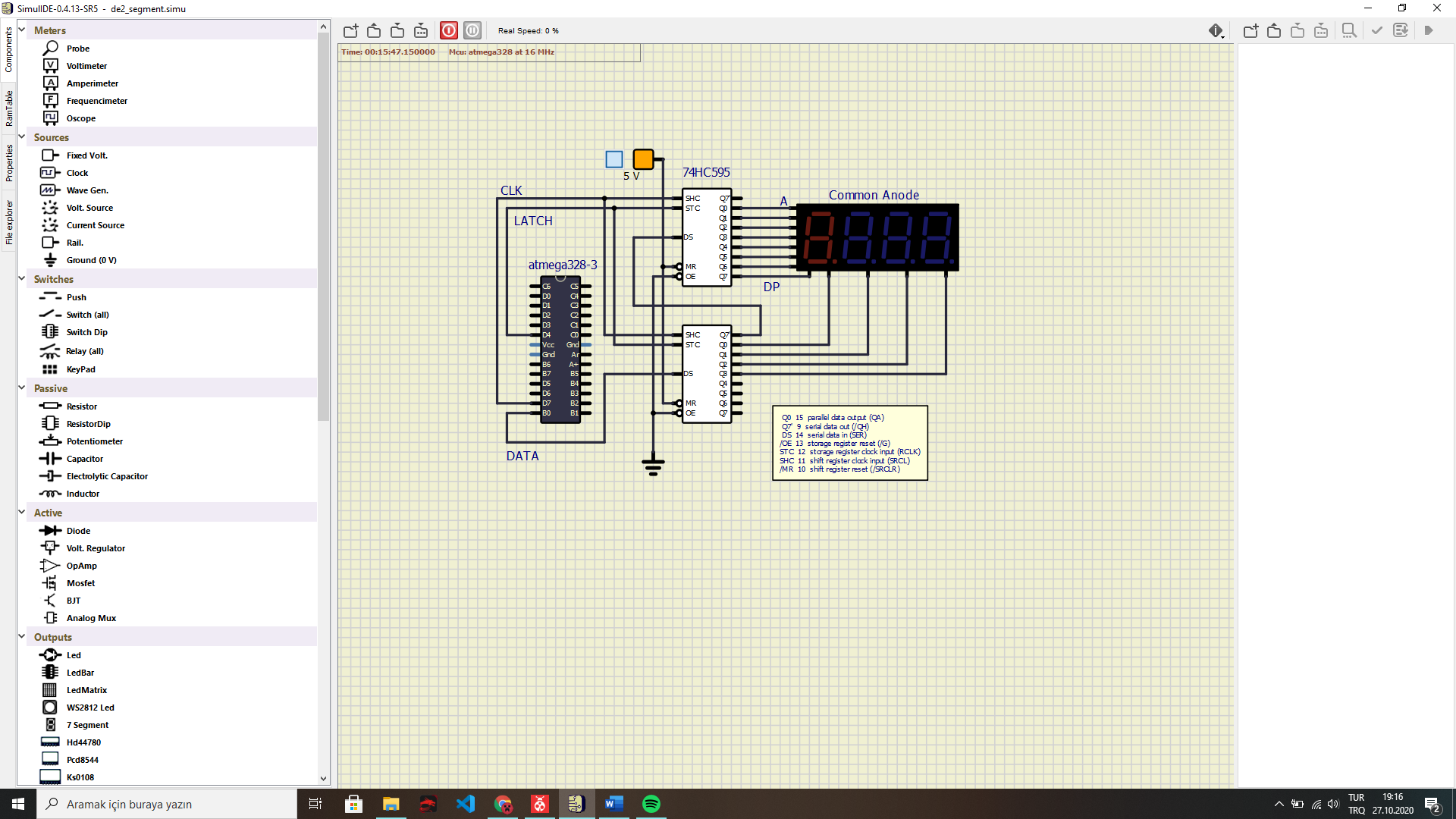
cnt1++;

if(cnt1 >= 6)

cnt1 = 0;

}

}

**Circuit:**

1. **Look-up table for the snake application:**

|  |  |  |
| --- | --- | --- |
|  | **Segment Value** | **Segment Position** |
| **Position 0, Led D** | 0b11101111 | 0b00010000 |
| **Position 0, Led C** | 0b11011111 | 0b00010000 |
| **Position 0, Led B** | 0b10111111 | 0b00010000 |
| **Position 0, Led A** | 0b01111111 | 0b00010000 |
| **Position 1, Led A** | 0b01111111 | 0b00100000 |
| **Position 1, Led F** | 0b11111011 | 0b00100000 |
| **Position 1, Led E** | 0b11110111 | 0b00100000 |
| **Position 1, Led D** | 0b11101111 | 0b00100000 |

**main.c:**

/\*

\* snake\_application.c

\*

\* Created: 27.10.2020 11:51:05

\* Author : dkorb

\*/

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

#define *F\_CPU* 16000000

#include <util/delay.h>

#include <avr/io.h> // AVR device-specific IO definitions

#include "segment.h" // Seven-segment display library for AVR-GCC

// Note: Specially for this application, "input\_type" parameter added to the

// "SEG\_update\_shift\_regs" function in segment.h, for setting the

// input type for the function.

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

int main(void)

{

// Configure SSD signals

SEG\_init();

/\* Replace with your application code \*/

while (1)

{

SEG\_update\_shift\_regs(0b11101111, 0b00010000, 0); // Position 0, Led D

*\_delay\_ms*(250);

SEG\_update\_shift\_regs(0b11011111, 0b00010000, 0); // Position 0, Led C

*\_delay\_ms*(250);

SEG\_update\_shift\_regs(0b10111111, 0b00010000, 0); // Position 0, Led B

*\_delay\_ms*(250);

SEG\_update\_shift\_regs(0b01111111, 0b00010000, 0); // Position 0, Led A

*\_delay\_ms*(250);

SEG\_update\_shift\_regs(0b01111111, 0b00100000, 0); // Position 1, Led A

*\_delay\_ms*(250);

SEG\_update\_shift\_regs(0b11111011, 0b00100000, 0); // Position 1, Led F

*\_delay\_ms*(250);

SEG\_update\_shift\_regs(0b11110111, 0b00100000, 0); // Position 1, Led E

*\_delay\_ms*(250);

SEG\_update\_shift\_regs(0b11101111, 0b00100000, 0); // Position 1, Led D

*\_delay\_ms*(250);

}

}