Kamil Káčer 211777

Binárne hodnoty 7 segmentoveho displaya



**Spoločná anóda:** Segmenty na 7segmentovom displayi su zasvietene ked privedieme nulu na piny(a,b,c,d,e,f,g,dp)

**Spoločná katóda:** Segmenty na 7segmentovom displayi sú zasvietene keď privedieme jednotku na piny(a,b,c,d,e,f,g,dp)

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

0b00011001}; // Digit 9

// Active-high position 0 to 3

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

// p3p2p1p0....

0b00010000, // Position 0

0b00100000, // Position 1

0b0100000, // 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(&PORTD,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 & 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 "position"

position = position >> 1;

}

// Pull LATCH high

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

// Wait 1 us

*\_delay\_us*(1);

}

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

/\* SEG\_clear \*/

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

/\* SEG\_clk\_2us \*/

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 8-bit Timer/Counter1

\* Set prescaler and enable overflow interrupt \*/

TIM0\_overflow\_4ms();

TIM0\_overflow\_interrupt\_enable();

// Enables interrupts by setting the global interrupt mask

/\* 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(singles,0);

position = 1;

}

else

{

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

position = 0;

}

}

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

/\*\*

\* ISR starts when Timer/Counter1 overflows. Increment decimal counter display in on SSD.

\*/

ISR(TIMER1\_OVF\_vect)

{

singles++;

if (singles > 9)

{

singles = 0;

decimals ++;

if (decimals > 5)

{

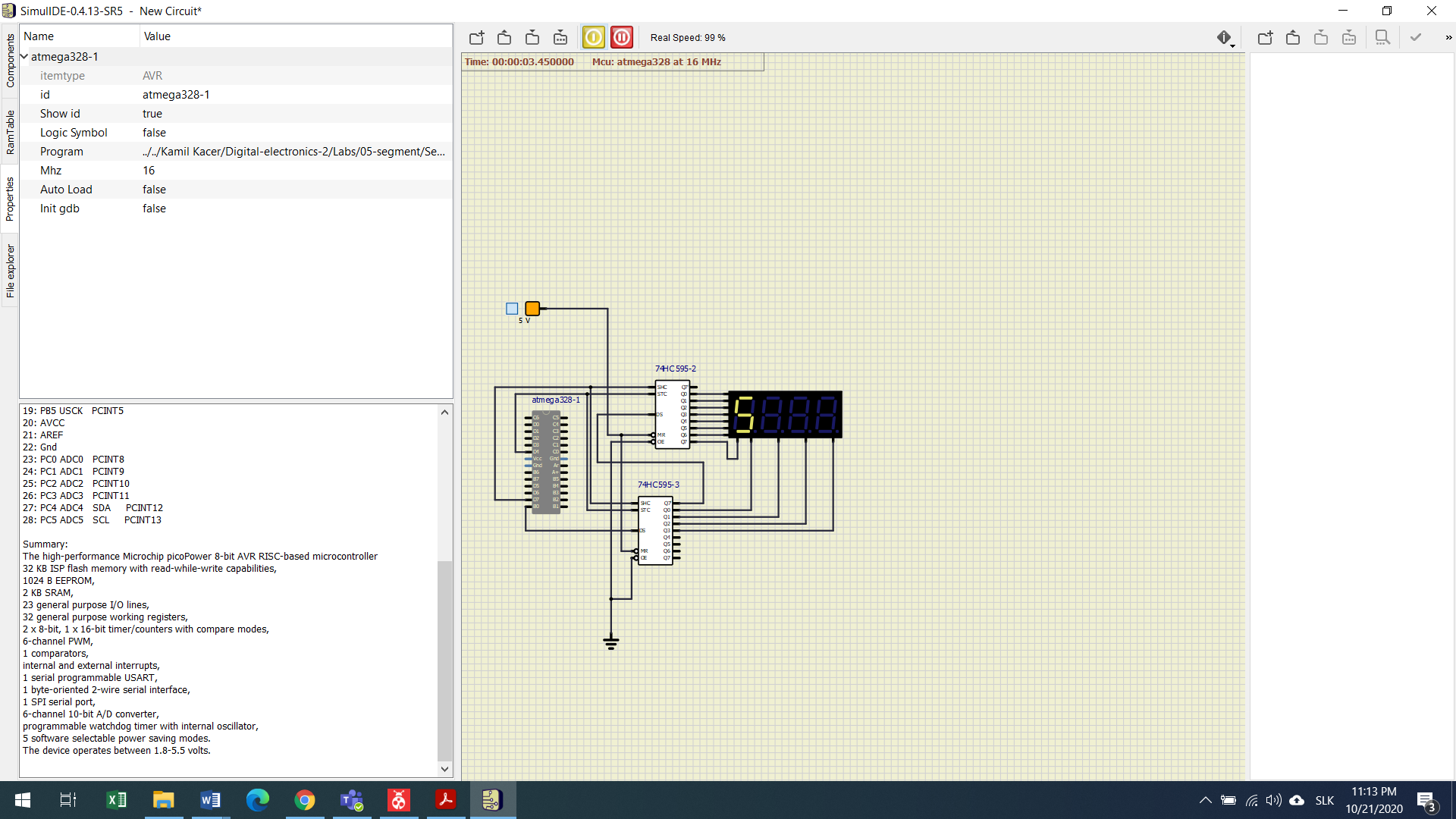
decimals = 0;

}

}

}

Displaying single digit



**Snake Code**

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

\*

\* 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;

*uint8\_t* snake = 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();

//SEG\_update\_shift\_regs(2,);

/\* Configure 8-bit Timer/Counter0

\* Set prescaler and enable overflow interrupt \*/

TIM0\_overflow\_16us();

TIM0\_overflow\_interrupt\_enable();

// Enables interrupts by setting the global interrupt mask

/\* 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(snake,0);

}

}

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

/\*\*

\* ISR starts when Timer/Counter1 overflows. Increment decimal counter display in on SSD.

\*/

ISR(TIMER1\_OVF\_vect)

{

snake++;

if(snake>5)

{

snake = 0;

}

}

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

// Active-low digits 0 to 9

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

// abcdefgDP

0b01111111, // Segments we need for the snake to go around in position 0

0b10111111,

0b11011111,

0b11101111,

0b11110111,

0b11111011};

// Active-high position 0 to 3

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

// p3p2p1p0....

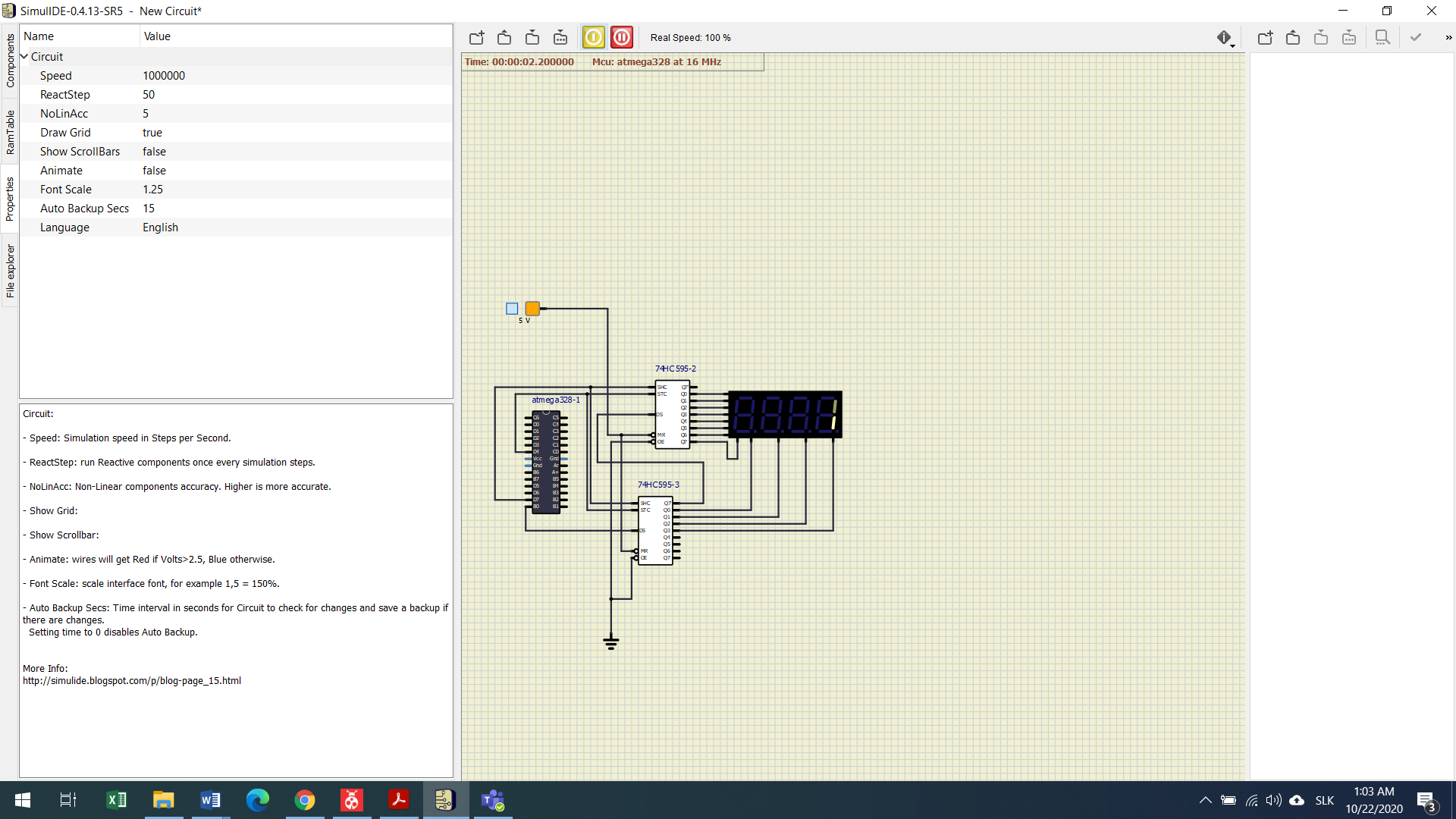
0b00010000, // Position 0

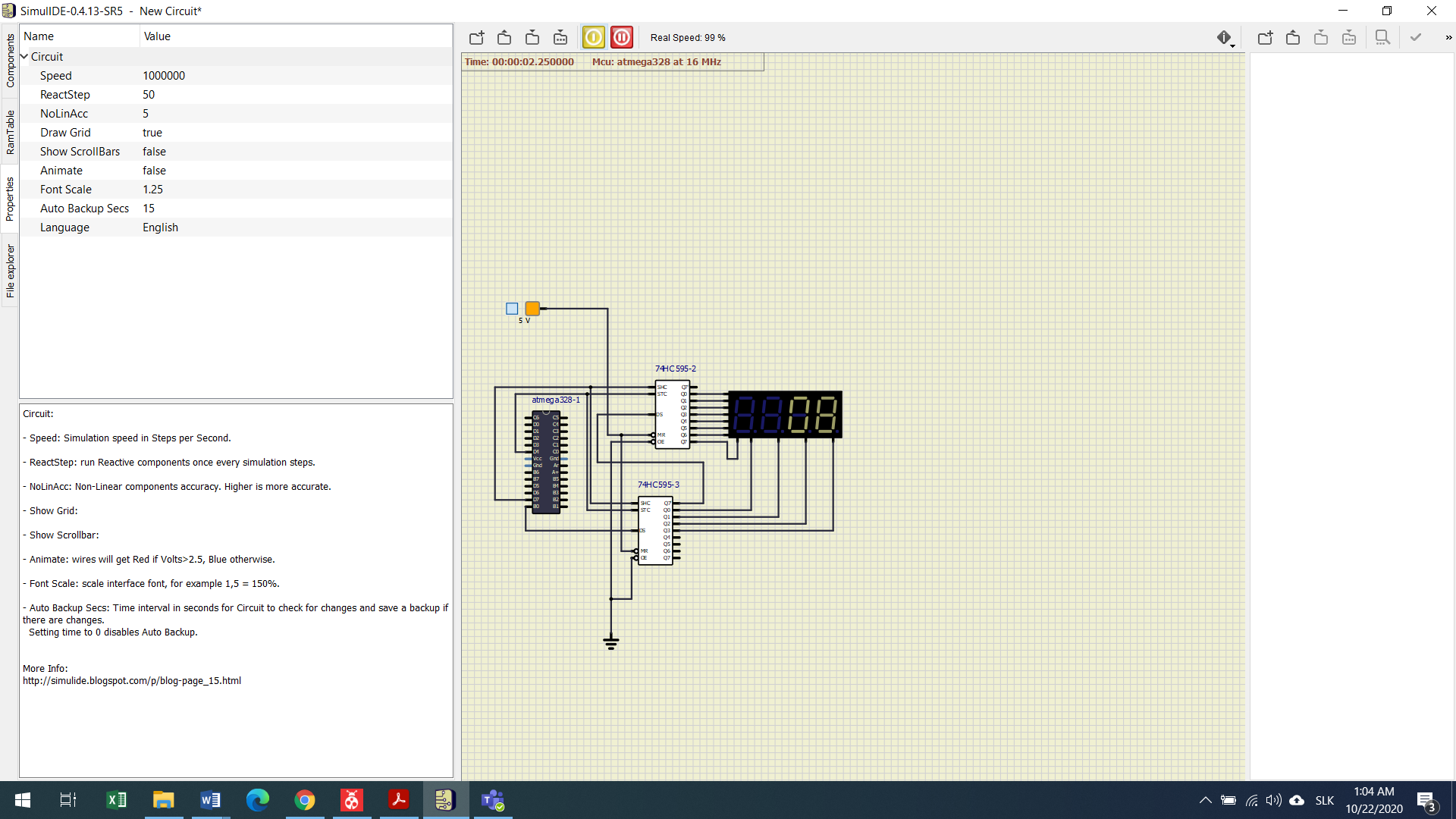
0b00100000, // Position 1

0b01000000, // Position 2

0b10000000}; // Position 3

Snake





Count up