master Digital-Electronics-2 / labs / lab5 / Go to file Add file

FilipPaul simulations ... 1 minute ago History

..		
.vscode	lab 5	6 days ago
include	lab 5	6 days ago
lib	lab 5	1 hour ago
pictures	lab 5	3 minutes ago
snake	lab 5	3 minutes ago
src	lab 5	3 minutes ago
test	lab 5	6 days ago
.gitignore	lab 5	6 days ago
Readme.md	simulations	1 minute ago
platformio.ini	lab 5	6 days ago
simulation.simu	lab 5	1 hour ago

Readme.md

Lab 5

Preparation Task

- Common Cathode 7-segment display (CC SSD) - active HIGH
- * Common Anode 7-segment display (CA SSD) - active LOW

In the following table, write the 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

Use schematic of the Multi-function shield and find out the connection of seven-segment display. What is the purpose of two shift registers 74HC595?
Shift registers are used for creating more output pins, this component alow us to create (8 output pins for each register) using only 3 wires (CLK, LATCH and Data), more registers can be connected in series where Q7' of first shift register is connected to data pin of second shift register

Lab Results

segment.cpp

```
/*
 *
 * 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"

uint8_t segment_values[]={
    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_positions[]={
    0b10000000,
    0b01000000,
    0b00100000,
    0b00010000,
    0b00001000,
    0b00000100,
    0b00000010,
    0b00000001};


// 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 segment_value = segment_values[segments];
    uint8_t position_value = segment_positions[position];
    uint8_t bit_number;

    // Pull LATCH, CLK, and DATA low
    GPIO_write_low(&PORTD,SEGMENT_LATCH);

    // 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 ((segment_value & 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"
        segment_value = segment_value >> 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_value & 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"

        // Shift "position"
        position_value = position_value >> 1;
    }

    // Pull LATCH high
    GPIO_write_high(&PORTD, SEGMENT_LATCH);

    // Wait 1 us
    _delay_us(1);
}

//-------------------------------------------------------------------
/* SEG_clear */
void SEG_clear()
{
    uint8_t bit_number;

    // Pull LATCH, CLK, and DATA low
    GPIO_write_low(&PORTD,SEGMENT_LATCH);

    // 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 < 16; bit_number++)
    {
```

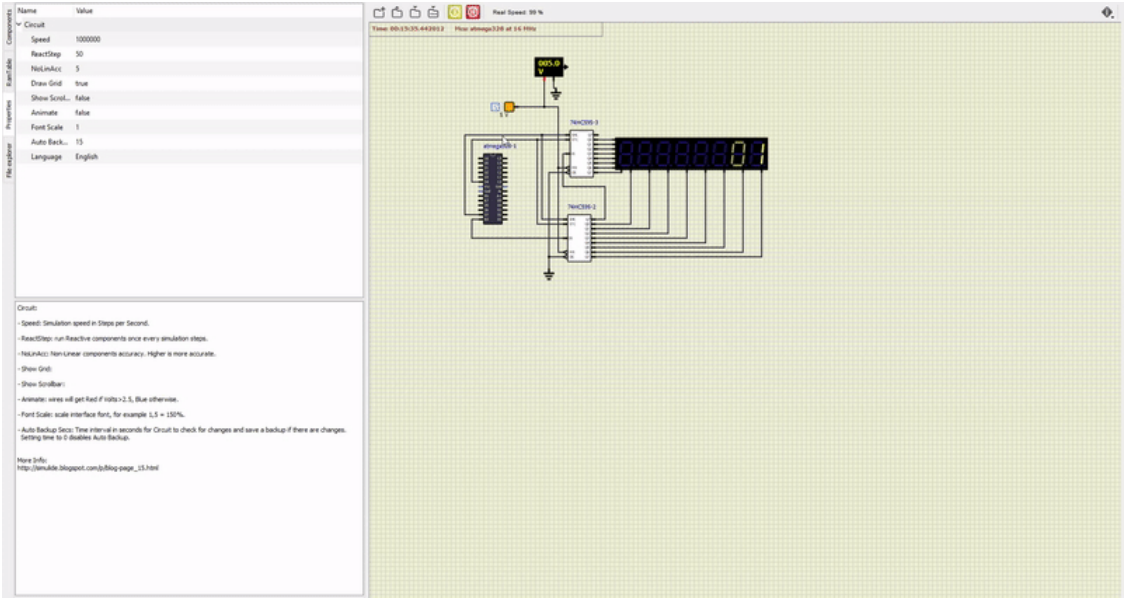
```
        GPIO_write_low(&PORTB, SEGMENT_DATA);

        // 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"
    }

    // Pull LATCH high
    GPIO_write_high(&PORTD, SEGMENT_LATCH);

    // Wait 1 us
    _delay_us(1);
}
/*-------------------------------------------------------------------*/
/* SEG_clk_2us */
void SEG_clk()
{
    // Pull CLK high
    GPIO_write_high(&PORTD, SEGMENT_CLK);
    // Wait 1 us
    _delay_us(1);
    // Pull CLK low
    GPIO_write_low(&PORTD, SEGMENT_CLK);
    // Wait 1 us
    _delay_us(1);
}
```

main.cpp

```
#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
#include "gpio.h"
#include <Arduino.h>
uint8_t segment[] = {0,0,0,0,0,0,0,0}; // initial value
uint8_t muxflag = 1; // variable for multiplexing digits

int main(void)
{
    // timer for multiplexing
    TIM2_overflow_512us(); // 512us for each digit
    TIM2_overflow_interrupt_enable();

    // Configure SSD signals
    SEG_init();
    TIM1_overflow_262ms() ;
    TIM1_overflow_interrupt_enable();
    sei();
    // Infinite loop
    while (1)
    {

    // Will never reach this
    }
    return 0;
}

/* Interrupt service routines ----------------------------------------*/
ISR(TIMER1_OVF_vect)
{

    if(segment[7] == 9){
      segment[6] = segment[6] + 1;
      segment[7] = 0;
    }

    else
    {
```

```
        segment[7] = segment[7] + 1;
    }

    if(segment[6] == 6){
        segment[6] = 0;
        segment[7] = 0;
    }
}
// multiplexing digits
ISR(TIMER2_OVF_vect){
    if (muxflag == 1){
    SEG_update_shift_regs(segment[7], 7);
    }
    else
    {
     SEG_update_shift_regs(segment[6], 6);
    }
    muxflag = !muxflag; //

}
```

Simulation



Snake.cpp

```
#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
#include "gpio.h"
#include <Arduino.h>
int main(void)
{
 // Configure SSD signals
  SEG_init();


  while (1)
  {
    for (size_t i = 0; i < 6; i++)
    {
      my_snake(i,7);
      _delay_ms(400);
    }


  // Will never reach this
  }
  return 0;
```

```
}
```

segment.cpp

I added In file segment.cpp following lines.

```
uint8_t snake_values[]={
 0b01111111,
 0b10111111,
 0b11011111,
 0b11101111,
 0b11110111,
 0b11111011,
 0b11111101,
 0b11111110,
};
void my_snake(uint8_t segments, uint8_t position)
{   uint8_t segment_value = snake_values[segments];
 uint8_t position_value = segment_positions[position];
 uint8_t bit_number;

 // Pull LATCH, CLK, and DATA low
 GPIO_write_low(&PORTD,SEGMENT_LATCH);

 // 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 ((segment_value & 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"
     segment_value = segment_value >> 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_value & 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"

        // Shift "position"
        position_value = position_value >> 1;
    }

    // Pull LATCH high
    GPIO_write_high(&PORTD, SEGMENT_LATCH);

    // Wait 1 us
    _delay_us(1);
}
```

Snake Simulation