













master ▾

...

Digital-Electronics-2 / labs / lab3 /

 FilipPaul ...	1 minute ago 
..	
 .vscode	3 hours ago
 include	3 hours ago
 lib	1 hour ago
 pictures	8 minutes ago
 src	1 hour ago
 test	3 hours ago
 .gitignore	3 hours ago
 README.md	1 minute ago
 platformio.ini	1 hour ago
 simulation.simu	19 minutes ago

README.md



Lab 3

Preparation tasks

1. What is the meaning of `volatile` keyword in C? What is the difference between operators `*` and `&`, such as `*reg` and `&DDRB`?

volatile: disables compiler optimizations, It tells the compiler that the value of the variable may change at any time--without any action being taken by the code, For Ex: this datatype is needed when using interrupts on some microcontrollers like ESP32 and etc.

pointer *: this datatypes 'points' to value at address -- syntax: *variable

&: adress of "smthing" -- syntax: &variable

***reg:** it points to the value at address of register name stored in variable reg

&DDRB: returns adress of DDRB register

2. Complete the following table with C data types.

Data type	Number of bits	Range	Description
uint8_t	8	0, 1, ..., 255	Unsigned 8-bit integer
int8_t	8	-128...0...127	Signed 8-bit integer
uint16_t	16	0..65535	Unsigned 16-bit integer
int16_t	16	-65536...0...65535	Signed 16-bit integer
float	32	-3.4e+38, ..., 3.4e+38	Single-precision floating-point
void	x	x	keyword to use as a placeholder where you would put a data type, to represent "no data".

Complete the code

```
#include <avr/io.h>

// Function declaration (prototype)
uint16_t calculate(uint8_t x, uint8_t y);

int main(void)
{
    uint8_t a = 156;
    uint8_t b = 14;
    uint16_t c;

    // Function call
    c = calculate(a, b);

    while (1)
    {
    }
    return 0;
}
```

```
// Function definition (body)
uint16_t calculate(uint8_t x, uint8_t y)
{
    uint16_t result;    // result = x^2 + 2xy + y^2
    result = x*x+ 2*x*y + y*y;
    return result;
}
```

Lab results

gpio.cpp

```

/*****
 *
 * GPIO 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 -----*/
#include "gpio.h"

/* Function definitions -----*/
void GPIO_config_output(volatile uint8_t *reg_name, uint8_t pin_num)
{
    *reg_name = *reg_name | (1<<pin_num);
}

/*-----*/
void GPIO_config_input_nopull(volatile uint8_t *reg_name, uint8_t pin_num)
{
    *reg_name = *reg_name | (1<<pin_num);
    *reg_name++;
    *reg_name = *reg_name | (1<<pin_num);
}

/*-----*/
void GPIO_config_input_pullup(volatile uint8_t *reg_name, uint8_t pin_num)
{
    *reg_name = *reg_name & ~(1<<pin_num); // Data Direction Register
    *reg_name++;                          // Change pointer to Data Register setting
    *reg_name = *reg_name | (1<<pin_num);  // Data Register
}

/*-----*/

```

```

void GPIO_write_low(volatile uint8_t *reg_name, uint8_t pin_num)
{
    *reg_name = *reg_name & ~(1<<pin_num);
}

/*-----*/
void GPIO_write_high(volatile uint8_t *reg_name, uint8_t pin_num)
{
    *reg_name = *reg_name | (1<<pin_num);
}

/*-----*/
void GPIO_toggle(volatile uint8_t *reg_name, uint8_t pin_num)
{
    *reg_name = *reg_name ^ (1<<pin_num);
}

/*-----*/
uint8_t GPIO_read(volatile uint8_t *reg_name, uint8_t pin_num)
{
    if (bit_is_clear(*reg_name, pin_num))
    {
        return 0;
    }
    else
    {
        return 1;
    }
}

```

main.cpp

```

#include <Arduino.h>

/* Defines -----*/
#define LED_GREEN    PB5    // AVR pin where green LED is connected
#define BLINK_DELAY  500
#define BTN          PD5
#define LED_RED      PB4
#ifndef F_CPU
#define F_CPU 16000000    // CPU frequency in Hz required for delay
#endif

/* Includes -----*/
#include <util/delay.h>    // Functions for busy-wait delay loops
#include <avr/io.h>        // AVR device-specific IO definitions
#include <gpio.h>          // GPIO library for AVR-GCC

int main(void)

```

```

{
    /* GREEN LED */
    GPIO_config_output(&DDRB, LED_GREEN);
    GPIO_write_low(&PORTB, LED_GREEN);

    /* RED LED */
    GPIO_config_output(&DDRB, LED_RED);
    GPIO_write_low(&PORTB, LED_RED);

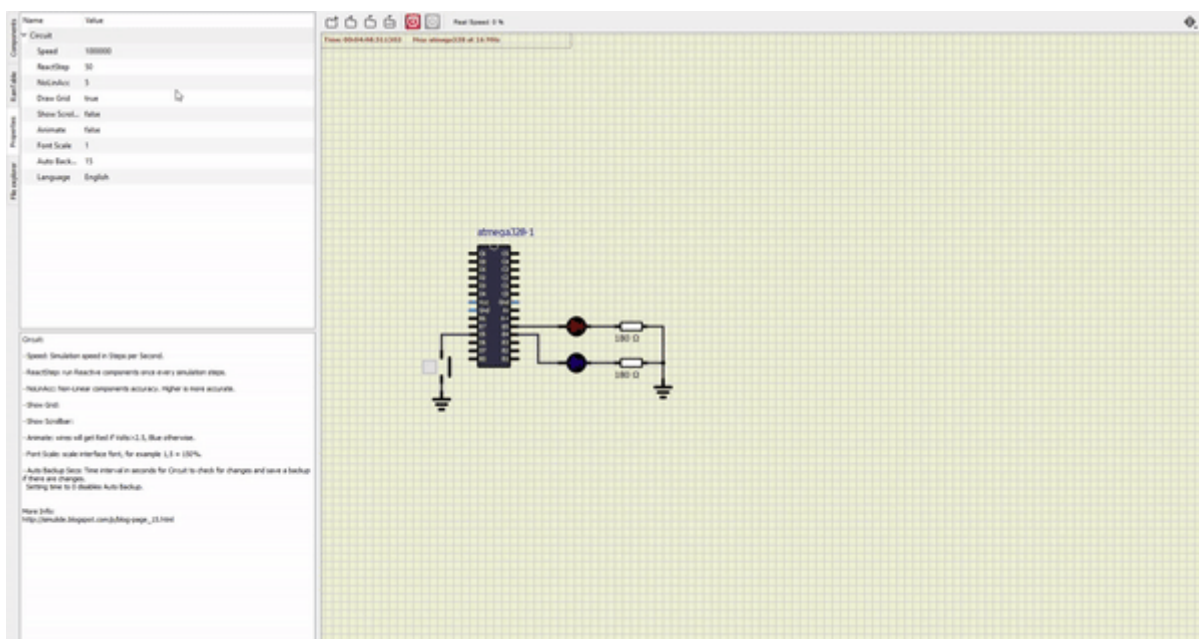
    /* push button */
    GPIO_config_input_pullup(&DDRD, BTN);

    // Infinite loop
    while (1)
    {
        // Pause several milliseconds
        _delay_ms(BLINK_DELAY);
        if (!GPIO_read(&PIND, BTN))
        {
            GPIO_toggle(&PORTB, LED_GREEN); //toggle leds
            GPIO_toggle(&PORTB, LED_RED);
        }
        if (GPIO_read(&PIND, BTN))
        {
            GPIO_write_low(&PORTB, LED_GREEN); //turn off if button isnt pushed
            GPIO_write_low(&PORTB, LED_RED);
        }
    }

    // Will never reach this
    return 0;
}

```

Simulation



Declaration vs definition of function

Declaration means, that you create "variable (function)", by doing that, this function can be called in your code, but does nothing. Definition of function means, that you create an algorithm, that will be executed, when function is called. Examples of these are in main.cpp (here you call functions), in gpio.h (here you declare functions) and in gpio.cpp (here you define functions).
