Lab 3: User library for GPIO control

David Garcia Torre

- Table with data types,

Data type	Number of bits	Range	Description
uint8_t	8	0, 1, ..., 255	Unsigned 8-bit integer
int8_t	8	-128...127	Signed 8-bit integer
uint16_t	16	0...65535	Unsigned 16-bit integer
int16_t	16	-32768...32767	Signed 16-bit integer
float	32	-3.4e+38, ..., 3.4e+38	Single-precision floating-point
void	64/128/256	2^128 – 1	Single-precision floating-point

- Completed source code from the example.

```
/*
 * lab3.c
 * Author : TheGT23
 */

#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;
}
```

Listing of library source file `gpio.c`

```
/**********************************************************************
 *
 * 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);  //Set bit (or¡)
}

/*-------------------------------------------------------------------*/
void GPIO_config_input_nopull (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
	*reg_name = *reg_name & ~ (1<<pin_num);   // Data Register
}


/*-------------------------------------------------------------------*/
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
    *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);  //Clear bit(and not)
}

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

```c
/*-------------------------------------------------------------------*/
void GPIO_toggle(volatile uint8_t *reg_name, uint8_t pin_num)
{
	*reg_name = *reg_name ^ (1<<pin_num);  //Toggle the bit
}

/*-------------------------------------------------------------------*/
uint8_t GPIO_read(volatile uint8_t *reg_name, uint8_t pin_num)
{
	uint8_t result = 0;

	if(bit_is_clear(*reg_name,pin_num)){ // if 'PUSH' (0) -> I enter de 'if'
		result = 1;
	}

	return result;
}
```

gpio.c

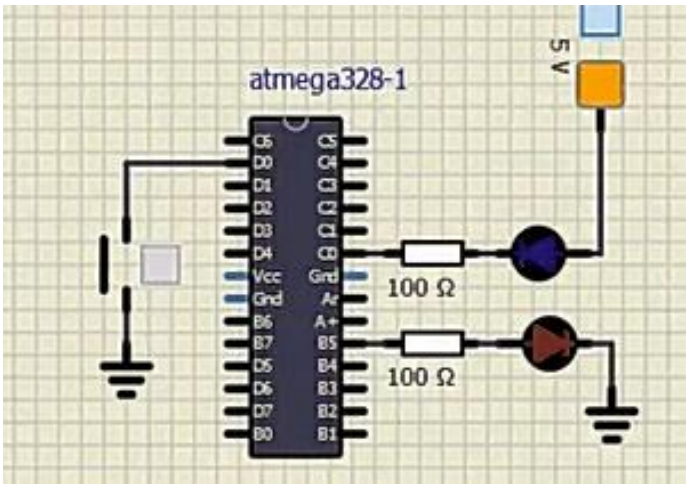
```c
/*****************************************************************
 *
 * GPIO library for AVR-GCC.
 * ATmega328P (Arduino Uno), 16 MHz, AVR 8-bit Toolchain 3.6.2
 *
 * Copyright (c) 2019-2020 Tomas Fryza
 * Dept. of Radio Electronics, Brno University of Technology, Czechia
 * This work is licensed 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);  //Set bit (or¡)
}

/*-------------------------------------------------------------------*/
void GPIO_config_input_nopull (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
	*reg_name = *reg_name & ~ (1<<pin_num);   // Data Register
}


/*-------------------------------------------------------------------*/
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
    *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);  //Clear bit(and not)
}
```

```c
/*-------------------------------------------------------------------*/
void GPIO_write_high(volatile uint8_t *reg_name, uint8_t pin_num)
{
	*reg_name = *reg_name | (1<<pin_num);  //Set bit(or)
}

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

/*-------------------------------------------------------------------*/
uint8_t GPIO_read(volatile uint8_t *reg_name, uint8_t pin_num)
{
	uint8_t result = 0;

	if(bit_is_clear(*reg_name,pin_num)){ // if 'PUSH' (0) -> I enter de 'if'
		result = 1;
	}

	return result;
}
```

Blinking of two LEDs

```c
/************************************************************************
 *
 * Alternately toggle two LEDs when a push button is pressed. Use
 * functions from GPIO library.
 * 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.
 *
 ************************************************************************/

/* Defines -----------------------------------------------------------*/
#define LED_GREEN   PB5     // AVR pin where green LED is connected
#define LED_RED   PC0     // AVR pin where red LED is connected
#define BUTTON   PD0     // AVR pin where the button is connected

#define BLINK_DELAY 500
#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

/* Function definitions ----------------------------------------------*/
/**
 * Main function where the program execution begins. Toggle two LEDs
 * when a push button is pressed. Functions from user-defined GPIO
 * library is used instead of low-level logic operations.
 */
```

```c
int main(void)
{
        /* PUSH */
        GPIO_config_input_pullup(&DDRD, BUTTON);

        /* RED LED */
        GPIO_config_output(&DDRC, LED_RED);
        GPIO_write_high(&PORTC, LED_RED);

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

    // Infinite loop
    while (1)
    {
        // Pause several milliseconds
        _delay_ms(BLINK_DELAY);

        if(GPIO_read(&PIND,BUTTON) == 1){

              GPIO_toggle(&PORTC,LED_RED);
              GPIO_toggle(&PORTB,LED_GREEN);

        }

    }

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



The function declaration serves so that the compiler need the information, basic information for knowing about the variables that needs to use, or where need to save the information.

On the other hand, the function is the place where we write what we want the program do. We write the code and when the function is called, it receives some variables and use them to send back to the place where is the declaration a value.