Lab 3: Creation of user library for GPIO control

Contents

Preparation tasks

Table of bits and numeric range for the selected data types defined by C.

Data type	Number of bits	Range	Description
`uint8_t`	8	0, 1, ..., 255	Unsigned 8-bit integer
`int8_t`	8	integers -127 to 127	Signed 8-bit integer, MSb defines polarity
`uint16_t`	16	integers 0 to 65 535	Unsigned 16-bit integer
`int16_t`	16	integers -32 768 to 32 767	Signed 16-bit integer, MSb defines polarity
`float`	32	-3.4e+38, ..., 3.4e+38	Single-precision floating-point
`void`	0	none	Function return type

Completed code:

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

uint16_t calculate(uint8_t,uint8_t);

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

    c = calculate(a, b);

    while (1)
    {
    }
    return 0;
}

uint16_t calculate(uint8_t x, uint8_t y)
{
    uint16_t result;    // result = x^2 + 2xy + y^2

    result = x*x;
    result += 2 * x * y;
    result += 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 ----------------------------------------------*/

/* GPIO_config_output */
void GPIO_config_output(volatile uint8_t *reg_name, uint8_t pin_num)
{
    *reg_name = *reg_name | (1<<pin_num); // Configurate output in register on
given pin
}

/*--------------------------------------------------------------------*/

/* GPIO_config_input_nopull */
void GPIO_config_input_nopull(volatile uint8_t *reg_name, uint8_t pin_num)
{
    *reg_name = *reg_name & ~(1<<pin_num); //DDR
    *reg_name++;                            // change register to PORT
    *reg_name = *reg_name & ~(1<<pin_num);
}

/*--------------------------------------------------------------------*/

/* GPIO_config_input_pullup */
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
}

/*--------------------------------------------------------------------*/
```

```
/* GPIO_write_low */
void GPIO_write_low(volatile uint8_t *reg_name, uint8_t pin_num)
{
    *reg_name = *reg_name & ~(1<<pin_num); // Write low-value to a set bit in
given register
}

/*--------------------------------------------------------------------*/

/* GPIO_write_high */
void GPIO_write_high(volatile uint8_t *reg_name, uint8_t pin_num)
{
    *reg_name |= (1<<pin_num); //Write high-value to a set bit in given register
}
/*--------------------------------------------------------------------*/

/* GPIO_toggle */
uint8_t GPIO_toggle(volatile uint8_t *reg_name, uint8_t pin_num)
{
    *reg_name ^= (1<<pin_num);  // Flib given bit in register
}


/*--------------------------------------------------------------------*/

/* GPIO_read */
uint8_t GPIO_read(volatile uint8_t *reg_name, uint8_t pin_num)
{
    return(bit_is_set(*reg_name, pin_num)); // if bit is set, returns 1, if not,
returns 0

}
```

C code of the application main.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 BTN         PD0
```

```
#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.
 */
int main(void)
{
    /* GREEN LED */
    GPIO_config_output(&DDRB, LED_GREEN);
    GPIO_write_low(&PORTB, LED_GREEN);

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

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

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

        if(bit_is_clear(PIND,BTN)) // Could use GPIO_read function but it is quite
unnecesary, this is less time consuming
        {
            GPIO_toggle(&PORTB, LED_GREEN);
            GPIO_toggle(&PORTC, LED_RED);
        }
    }

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

Creating a function

Declaration

Declaration of function is creating information about functions existence, it's name, number of inputs, input types and output type. It is usually done in header file for better lucidity and faster, more effective, compiling, thou it can be also done in main file.

Definition

Definition assigns certain algorithm to declared function. It is usually done in separate .c file designated for defining functions.

Calling a function

Calling function is as simple as using it in your code. It requires knowledge of the function for correct usage.

Notes

Writing declaration and definition in designated files gives the compiler ability, to choose only those functions included in main file.