Session 3

-

User library for GPIO control

Author: Guillermo Cortés Orellana

Teacher: Tomáš Frýza

Lab assignment

1. **Preparation tasks**

- Table with data type

Data type	Number of bits	Range	Description
uint8_t	8	0, 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	-3.4E38, 3.4E38	Single-precision floating-point
void	0	-	No value

- Source code completed from the example

```c
#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;
    result = result + (2*x*y);
    result = result + (y*y);

    return result;
}
```

2. **GPIO library**

- Listing of library source file ***gpio.c***

  Note: functions prototypes are in ***gpio.h*** file

```c
/*
 * gpio.c
 *
 * Created: 10/10/2020 21:06:48
 *  Author: Guillermo Cortés
 */

/************************************************************************
 *
 * GPIO library for AVR-GCC.
 * ATmega328P (Arduino Uno), 16 MHz, AVR 8-bit Toolchain 3.6.2
 *
 * Copyright (c) Guillermo Cortés
 * 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 ------------------------------------------------*/

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

/*INPUT*/
/*NO PULL*/
void GPIO_config_input_nopullup(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
}
```

```c
/*PULL*/
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
}

/*LOW*/
void GPIO_write_low(volatile uint8_t *reg_name, uint8_t pin_num)
{
    *reg_name = *reg_name & ~(1<<pin_num); // Clear the bit (and not)
}

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

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

/*READ*/
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' -> Si hay un '0' (ACTIVO BAJO), entro en el 'if'
        result = 1;
    }

    return result;
}
```

You can find the code on my GitHub:

https://github.com/GuicoRM/Digital-Electronics-2

- C code of the application ***main.c***

```
/*
 * Proyecto3_Puls_2LEDS.c
 *
 * Created: 10/10/2020 14:42:26
 * Author : Guillermo Cortés Orellana
 */

/*****************************************************************************
 *
 * 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) Guillermo Cortés
 * 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_BLUE    PC0     // AVR pin where blue LED is connected
#define BTN         PD0    // AVR pin where blue PUSH 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
```

```c
/* 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); // LED off because active HIGH

    /* BLUE LED */
    GPIO_config_output(&DDRC, LED_BLUE);
    GPIO_write_high(&PORTC, LED_BLUE); // LED off because active LOW

    /* push button */
    GPIO_config_input_pullup(&DDRD, BTN); // Configure DDRx and PORTx in 'pullup mode'

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

        if (GPIO_read(&PIND, BTN) == 1){ // If we push the button (0), I enter de 'if'

            GPIO_toggle(&PORTB, LED_GREEN); // Switch (blink)
            GPIO_toggle(&PORTC, LED_BLUE); // Switch (blink)
        }
    }

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

You can find the code on my GitHub:

https://github.com/GuicoRM/Digital-Electronics-2

- Difference between the declaration and the definition of the function in C

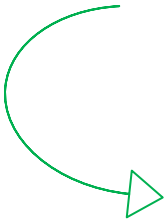
  - **Declaration:** part of the code where is included some information related to the own function (name, parameters, return type). It could help others programmers to understand the code and the goal of the function. It is normally included in **.h** file.

  - **Definition:** in contradistinction to the declaration, the definition implements the task which the function will carry out. It is normally included in **.c** file and "contains" the structure of the declaration.

  - **Example:**

Declaration → .h file

```
int example_add_2_numbers (int num1, int num2);
```



Definition → .c file

```
int example_add_2_numbers (int num1, int num2){

    *Code that allows adding 2 numbers*

}
```