Lab Assignment 3.

# Task 1: Preparation task.

**Data types in C and their ranges.**

| **Data type** | **Number of bits** | **Range** | **Description** |
| --- | --- | --- | --- |
| uint8\_t | 8 | 0, 1, ..., 255 | Unsigned 8-bit integer |
| int8\_t | 8 | -128,-127...127 | Signed 8-bit integer |
| uint16\_t | 16 | 0,1...65538 | 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 | \* | \* | Used for pointers |

\*The number of bits and the range for void varies with the system type

**Function phototype Example**

#include <avr/io.h>

// Function declaration (prototype)

uint16\_t calculate(uint8\_t, uint8\_t );

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 += 2\*x\*y;

result += y\*y;

return result;

}

# Task 2: GPIO library.

1. **Listing of library source file gpio.c,**

|  |
| --- |
| /\* |
|  | \* gpio.c |
|  | \* |
|  | \* Created: 10/7/2020 11:36:30 |
|  | \* Author: masau |
|  | \*/ |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  | \* |
|  | \* 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); // Data Register |
|  | } |
|  |  |
|  | /\*--------------------------------------------------------------------\*/ |
|  | 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); |
|  | } |
|  |  |
|  | /\*--------------------------------------------------------------------\*/ |
|  | 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; |
|  | } |
|  | } |

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

|  |
| --- |
|  |
| /\* |
|  | \* gpio.c |
|  | \* |
|  | \* Created: 10/7/2020 11:13:50 |
|  | \* Author : masau |
|  | \*/ |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  | \* |
|  | \* 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); |
|  |  |
|  | /\* RED LED \*/ |
|  | GPIO\_config\_output(&DDRC, LED\_RED); |
|  | GPIO\_write\_low(&PORTC, 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); |
|  | GPIO\_toggle(&PORTC, LED\_RED); |
|  | } |
|  |  |
|  | } |
|  |  |
|  | // Will never reach this |
|  | return 0; |
|  | } |
|  |  |

1. **In your words, describe the difference between the declaration and the definition of the function in C. Give an example**

Declaration of a function provides the compiler with the basic attributes of the function i.e. the name of the function, the number and type of arguments it takes and its return type. For example:

int func(int, int);

Whereas the definition of the function is used for allocating memory for the function and provide extra details of what a particular function does. E.g.

int add(int x, int y)

{

return (x + y);

}

Differences between function declaration and definition

| **Declaration** | **Definition** |
| --- | --- |
| A function can be declared any number of times | A function can be defined only once |
| Memory will not be allocated during declaration | Memory will be allocated |
| int func(int); | int func(int x)  {  return x;  } |

1. **Reprogramed Knight raider.**

/\*

\* gpio.c :m

\*

\* Created: 10/7/2020 11:13:50

\* Author : masau

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* 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 -----------------------------------------------------------\*/

//Defines

#define LED\_GREEN\_5 PB5 // AVR pin where green LED is connected

#define LED\_RED\_4 PB4 // AVR pin where RED LED is connected

#define LED\_GREEN\_3 PB3 // AVR pin where green LED is connected

#define LED\_RED\_2 PB2 // AVR pin where RED LED is connected

#define LED\_GREEN\_1 PB1 // AVR pin where green LED is connected

#define BTN PD0 // AVR pin where PUSH BUTTONis connected

#define BLINK\_DELAY 50 // TIME DELAY IN MILISECONDS

#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 FIVE LEDs

\* when a push button is pressed.

\*/

//This is the modified version of knight raider

int main(void)

{

/\* GREEN LED \*/

GPIO\_config\_output(&DDRB, LED\_GREEN\_1); //output configuration

GPIO\_write\_low(&PORTB, LED\_GREEN\_1); //port assignment

GPIO\_config\_output(&DDRB, LED\_GREEN\_3);

GPIO\_write\_low(&PORTB, LED\_GREEN\_3);

GPIO\_config\_output(&DDRB, LED\_GREEN\_5);

GPIO\_write\_low(&PORTB, LED\_GREEN\_5);

/\* RED LED \*/

GPIO\_config\_output(&DDRB, LED\_RED\_2);

GPIO\_write\_low(&PORTB, LED\_RED\_2);

GPIO\_config\_output(&DDRB, LED\_RED\_4);

GPIO\_write\_low(&PORTB, LED\_RED\_4);

/\* push button \*/

GPIO\_config\_input\_pullup(&DDRD, BTN);

// Infinite loop

while (1)

{

// Pause several milliseconds

*\_delay\_ms*(BLINK\_DELAY);

if (!GPIO\_read(&PIND, BTN))

{

//forward toggle

GPIO\_toggle(&PORTB, LED\_GREEN\_1);

*\_delay\_ms*(BLINK\_DELAY);

GPIO\_toggle(&PORTB, LED\_GREEN\_1);

GPIO\_toggle(&PORTB, LED\_RED\_2);

*\_delay\_ms*(BLINK\_DELAY);

GPIO\_toggle(&PORTB, LED\_RED\_2);

GPIO\_toggle(&PORTB, LED\_GREEN\_3);

*\_delay\_ms*(BLINK\_DELAY);

GPIO\_toggle(&PORTB, LED\_GREEN\_3);

GPIO\_toggle(&PORTB, LED\_RED\_4);

*\_delay\_ms*(BLINK\_DELAY);

GPIO\_toggle(&PORTB, LED\_RED\_4);

GPIO\_toggle(&PORTB, LED\_GREEN\_5);

*\_delay\_ms*(BLINK\_DELAY);

//backward toggle

GPIO\_toggle(&PORTB, LED\_GREEN\_5);

GPIO\_toggle(&PORTB, LED\_RED\_4);

*\_delay\_ms*(BLINK\_DELAY);

GPIO\_toggle(&PORTB, LED\_RED\_4);

GPIO\_toggle(&PORTB, LED\_GREEN\_3);

*\_delay\_ms*(BLINK\_DELAY);

GPIO\_toggle(&PORTB, LED\_GREEN\_3);

GPIO\_toggle(&PORTB, LED\_RED\_2);

*\_delay\_ms*(BLINK\_DELAY);

GPIO\_toggle(&PORTB, LED\_RED\_2);

GPIO\_toggle(&PORTB, LED\_GREEN\_1);

}

}

// Will never reach this

return 0;

}

