Lab 3: User library for GPIO control

* 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 |  |  |  |

* Completed source code from the example.

#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;

}

* 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)

}

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

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;

}

* 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 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.

\*/

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, BUTTON);

// Infinite loop

while (1)

{

// Pause several milliseconds

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

if(GPIO\_read(&PIND,BUTTON) == 1){

// Invert LED in Data Register

// PORTB = PORTB xor 0010 0000

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

GPIO\_toggle(&PORTC,LED\_RED);

}

}

// Will never reach this

return 0;

}

In the function declaration there are the things that the compiler needs , the name, the return type and the parameters. In the other hand, in the function definition is the code that has to be done when you call the function.