L_02

Hardware I/O – Digital inputs control

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this laboratory work is to get familiarized with the setup and use of digital inputs of a LPC1769 microcontroller.

1.2 Objective

(1) Implementation of an embedded C application for controlling the digital outputs based on the state of two user buttons (digital inputs) handled by GPIO interrupts.

2. General Description

The AOAA Board contains two user buttons, placed under the RGB LEDs, which are configurable for custom applications. The signal provided by the state switch of the buttons is a digital one (high or low). The GPIO pins associated with these buttons are: P2_11 and P2_12.

2.1 Software template description

The software template contains all the setups to run a standalone application for reading the input state change of the user buttons and for driving the board RGB LESs modules.
This template has three important sections in the code:

- the GPIO (General Purpose Input Output) Interrupt handler function (Fig. 1)
- the LEDs task running continuously from "*tasks\rc_task_leds\rc_task_leds.c*"
- the application logic implemented in the LEDs task, which is based on a simple State-Machine, changed by the user via user buttons and interrupts handler.

```
void EINT3_IRQHandler(void)
{
    //Button1 Push
    if(E_RC_TRUE == rcgpio_GetInterruptStatus(RC_SW2_PORT, RC_SW2_PIN, E_RCGPIO_RISING))
    {
        rcleds_SetState(E_RCLEDS_STATE_RED);
        rcgpio_ClearInterrupt(RC_SW2_PORT, RC_SW2_PIN);
    }
    //Button1 Release
    if(E_RC_TRUE == rcgpio_GetInterruptStatus(RC_SW2_PORT, RC_SW2_PIN, E_RCGPIO_FALLING))
    {
        rcgpio_ClearInterrupt(RC_SW2_PORT, RC_SW2_PIN);
    }

    //Button2 Push
    if(E_RC_TRUE == rcgpio_GetInterruptStatus(RC_SW3_PORT, RC_SW3_PIN, E_RCGPIO_RISING))
    {
        rcleds_SetState(E_RCLEDS_STATE_BLUE);
        rcgpio_ClearInterrupt(RC_SW3_PORT, RC_SW3_PIN);
    }
    //Button2 Release
    if(E_RC_TRUE == rcgpio_GetInterruptStatus(RC_SW3_PORT, RC_SW3_PIN, E_RCGPIO_FALLING))
    {
        rcgpio_ClearInterrupt(RC_SW3_PORT, RC_SW3_PIN);
    }
}
```

Figure 1. User buttons state interrupt handler

In the "*EINT3_IRQHandler*" function, the application checks for the state of the two user buttons. The interrupt handler is called between two atomic operations in case one of the

buttons switches its state (from LOW to HIGH or from HIGH to LOW – rising edge or falling edge).

The current state of the button(s) is checked using "*rcgpio_GetInterruptStatus*" function. The last parameter of the function determines which transition shall be checked while in interrupt: rising or falling edge (*see function description*). The "*if*" block contain the user functions (actions) that need to be taken. In this example, the action taken is a state change of the LEDs using "*rcleds_SetState*" and the new state is parsed in the LEDs task.

The interrupt events are handled by the Interrupt controller (NVIC – Nested Vectored Interrupt Controller), built in the microcontroller core. Once an interrupt occurs, the controller sets a flag associated with the trigger interrupt. Between the atomic operations of the processor, the interrupt controller checks for the active interrupt flags and calls the appropriate Interrupt Handler functions. After a call of the interrupt handler function, it is recommended to clear manually the trigger flag using "*rcgpio_ClearInterrupt*".

The GPIO interrupts are activated individually for each input pin and for each transition (falling or rising edge – Fig. 2)

```
/* Enable interrupts */
rcgpio_EnableInterrupt(RC_SW2_PORT, RC_SW2_PIN, E_RCGPIO_RISING);
rcgpio_EnableInterrupt(RC_SW2_PORT, RC_SW2_PIN, E_RCGPIO_FALLING);
rcgpio_EnableInterrupt(RC_SW3_PORT, RC_SW3_PIN, E_RCGPIO_RISING);
rcgpio_EnableInterrupt(RC_SW3_PORT, RC_SW3_PIN, E_RCGPIO_FALLING);
NVIC_EnableIRQ(EINT3_IRQn);
```

Figure 2. GPIO Interrupts setup

The LEDs state is parsed in the LEDs task ("*tasks\rc_task_leds\rc_task_leds.c*"). In this example, there are two defined states: *E_RCLEDS_STATE_RED* and *E_RCLEDS_STATE_BLUE*. Based on the buttons state, the state is switched between the two defined ones. As functionality, the application turns ON the RED LEDs if user button 1 is pressed and turns ON the BLUE LEDs if the user button 2 is pressed. In each state, all LEDs are switched off as a first action.

```
while(1)
{
    //Read current LEDs state
    le_LedsState = rcleds_GetState();

    //Check the current state of the task
    switch(le_LedsState)
    {
    case E_RCLEDS_STATE_RED:
        //Turn off all LEDS
        rcleds_ClearLEDs();
        //Turn on RED LEDs
        rcgpio_ClearPin(RC_RGBLEDS_PORT, RC_RGBLED1_RED_PIN);
        rcgpio_ClearPin(RC_RGBLEDS_PORT, RC_RGBLED2_RED_PIN);
        //Switch to State None
        rcleds_SetState(E_RCLEDS_STATE_NONE);
        break;
    case E_RCLEDS_STATE_BLUE:
        //Turn off all LEDS
        rcleds_ClearLEDs();
        //Turn on BLUE LEDs
        rcgpio_ClearPin(RC_RGBLEDS_PORT, RC_RGBLED1_BLUE_PIN);
        rcgpio_ClearPin(RC_RGBLEDS_PORT, RC_RGBLED2_BLUE_PIN);
        //Switch to State None
        rcleds_SetState(E_RCLEDS_STATE_NONE);
        break;
    default:
        //Do nothing
        break;
    }
    vTaskDelay(100);
}
```

Figure 3. LEDs task

3. Requirements

Based on the example described in the *2. General Description*, update the application template with the following requirements:

(1) The application shall start blinking the RED LEDs once the user button is pushed. The timing between blinks shall be 2 seconds and the blinking period (ON + OFF) shall be 500 ms.
(2) Every new button push shall increment the number of blinks, considering the timings at req. (1). The maximum number of blinks shall be 5 (button push shall be ignored if the blink counter is higher than 5)
(3) User button 2 shall be used to reset the blinks counter and turn OFF all LEDs.