GPIO interrupts

For the following exercises you need to study chapter 12 of LPC1549 user manual and pinint15xx.h.

There are eight pin interrupts available. Their interrupt numbers are defined in cmsis.h. The numbers are: *PIN_INT0_IRQn - PIN_INT7_IRQn*.

Interrupt handlers: `void PIN_INT0_IRQHandler(void)` - `void PIN_INT7_IRQHandler(void)`.

Remember that you need to do the following:

- Initialize pin interrupt hardware
- Set priority of each pin interrupt you plan to use (NVIC)
- Configure and enable pin interrupts in the pin interrupt hardware
- Write interrupt handlers for each interrupt
- Enable each interrupt on NVIC

Remember that interrupts must be enabled on both pin interrupt hardware and NVIC.

You need to use pin interrupts in all of the following exercises.

Exercise 1

Write a program that configures SW1-SW3 as inputs with pull-ups and enables falling edge interrupt for each pin. The ISR of a pin sends the number of the pin into a queue.

A task waits on the queue and counts the number of successive presses of the same button. The number of button presses counted is printed when a different button is pressed.

For example if user presses the buttons in the following way:

1, 1, 1, 2, 2, 3, 1, 1, 1, 2, 2, 2, 2, 2

The program prints:

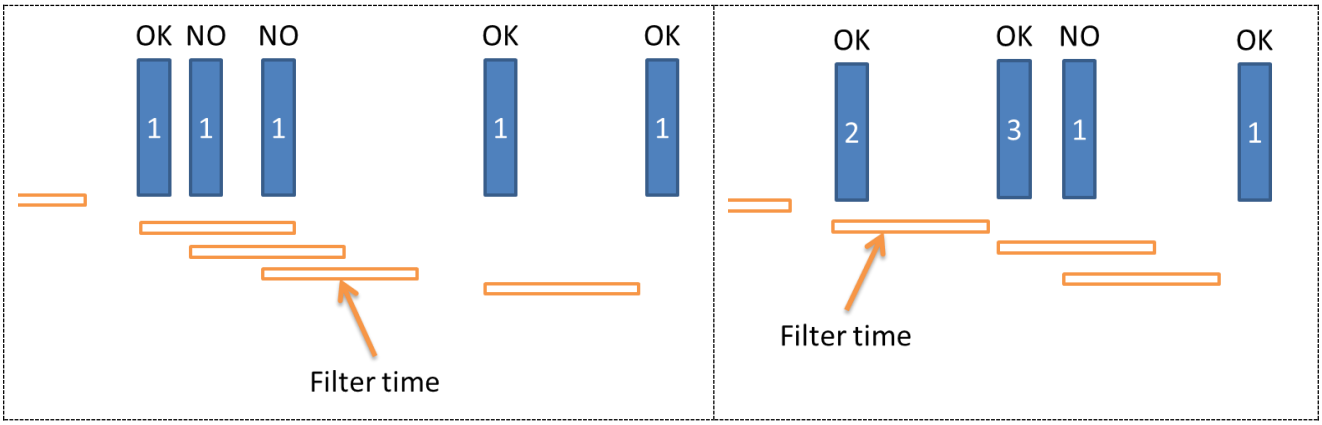
```
Button 1 pressed 3 times.
Button 2 pressed 2 times.
Button 3 pressed 1 times.
Button 1 pressed 3 times.
```

Note that button presses of button 2 at the end of the sequence are not printed until some other button is pressed!

Exercise 2

Write a program that configures SW1-SW3 as inputs with pull-ups and enables falling edge interrupt for each pin. The ISR of a pin sends the number of the pin and the current tick count value into a queue. (Use an object/structure for the data.)

A task waits on the queue and reads the button information from the queue. The task prints the time elapsed between button presses. The task has a configurable filter that ignores button presses that occur too soon after previous button press.



The filter set commands are read from the serial port. The filter commands is called “filter” which is followed by the minimum number of milliseconds that must pass between acceptable button presses. When new filter time is set the new value is printed to notify user about the change. The default filter time is 50 ms.

The counting always starts from the previous button press even if it was filtered out. Note that the time is always counted from previous button press regardless of the button number.

The program prints (for example):

```
Started waiting:
400 ms Button 1
800 ms Button 2
1.2 s Button 3
700 ms Button 3
8.3 s Button 2
```

Exercise 3

Use pin interrupts for SW1-SW2. Configure switches to interrupt on falling edge and send button presses to a queue. Implement a task that monitors SW3 and if it pressed continuously for three seconds then a notification is sent to the queue.

Implement a code lock with an 8-bit access code. The lock must accept the sequence at any point in the keying sequence and there is 15 s timeout. If no key is pressed for 15 seconds all previous key presses are ignored.

When the correct code has been entered the door opens for 5 seconds. Red led is on when door is “locked” and green led is on when door is “open”. Code lock reads button presses from the queue. SW1 is 0 and SW2 is 1. When SW3 is notification is received the blue led is switched on and lock goes into a learning mode and records a new code that is entered. After the new code has been recorded the blue led is switched off and red is switched on and the lock goes back to normal operation mode. Employ filtering to eliminate false presses from switch bounce.

The default code after reset is: 11110110. Note that code must be accepted at any point in the sequence. For example is user presses: 1111111110110, the door must open!