

Polling switches

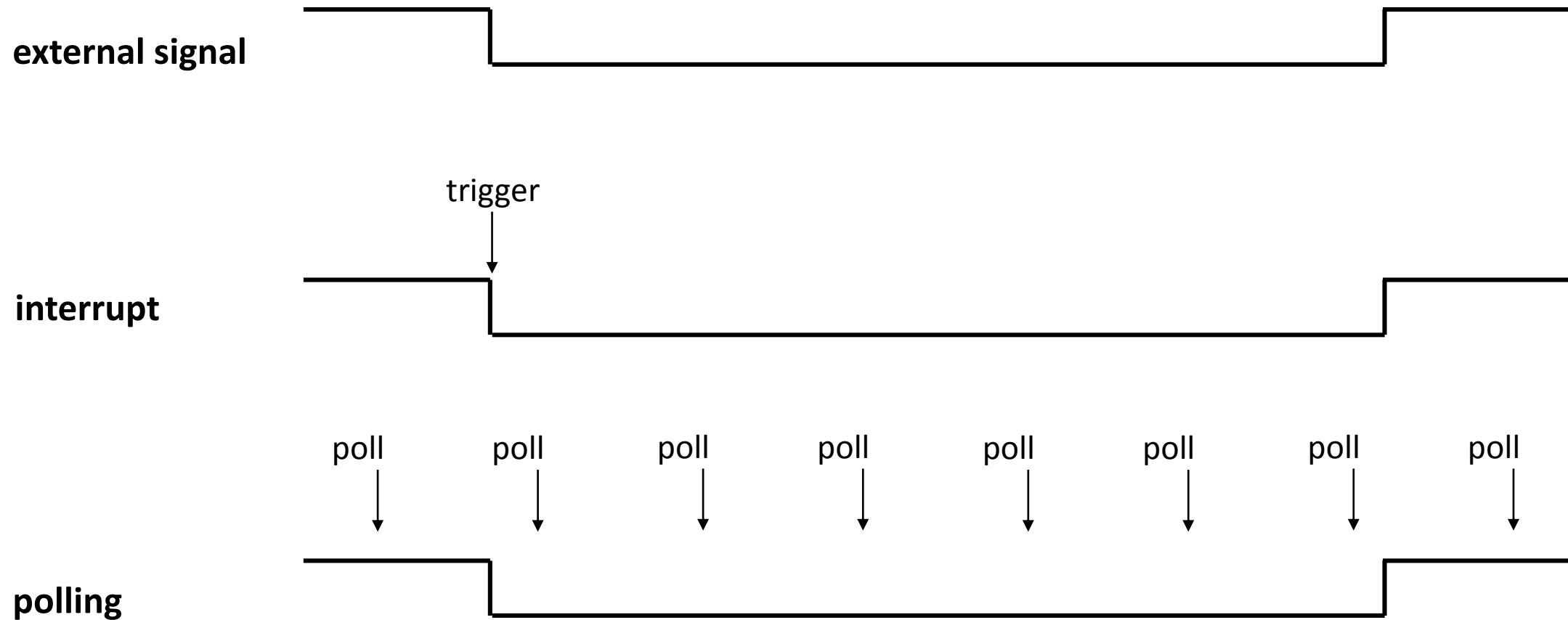
Case of study: the Joystick

P. Bernardi

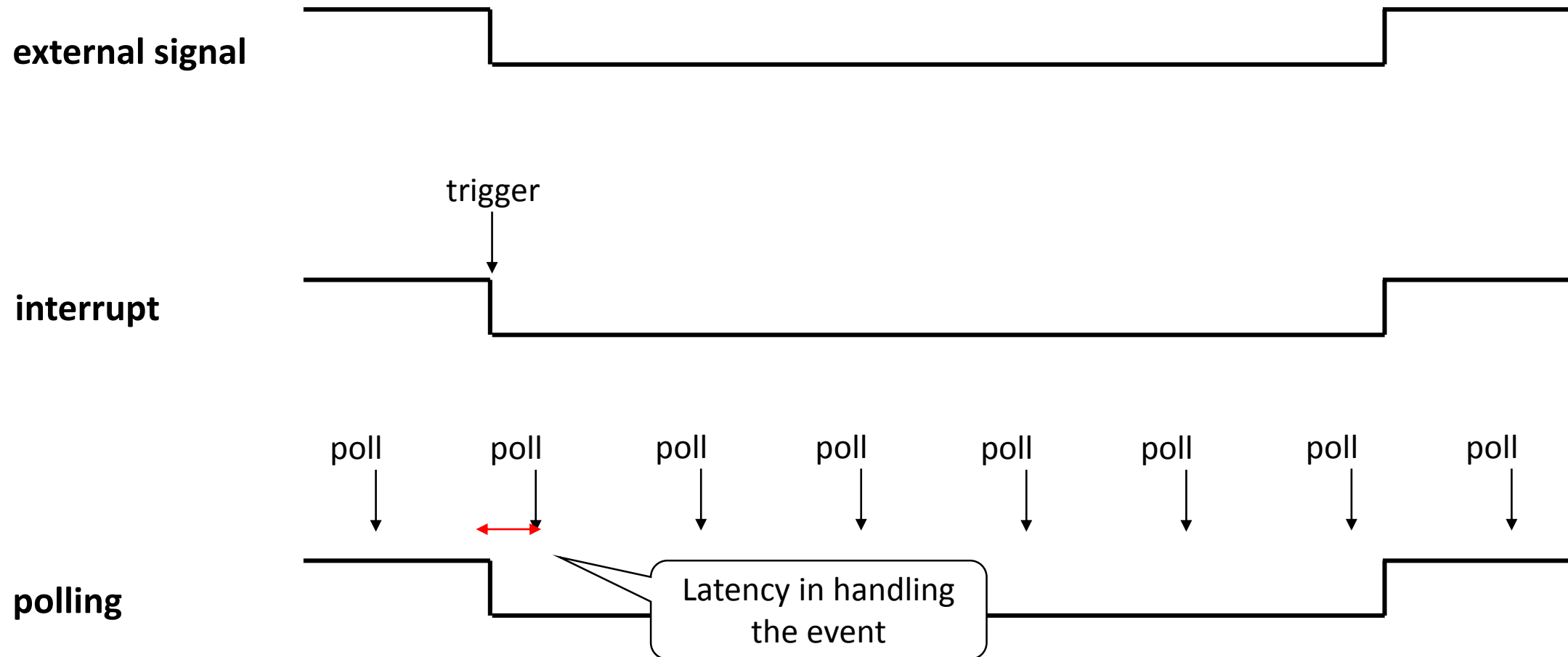
Polling vs Interrupt

- It is usually preferred to use Interrupt functionalities to be noticed about external events
 1. It is more **timing** efficient because the event is handled as soon as required, without any latency (this is fundamental for safety)
 2. It is more **power** efficient because the system sleeps between requests (this is fundamental for mobile)
- Unfortunately, not all events are triggering interrupt
 - Peripheral cores, inside the SoC, which are not connected to the Interrupt Controller (quite unusual)
 - External devices connected to pins that cannot be configured as external interrupt sources (case of study: the joystick).

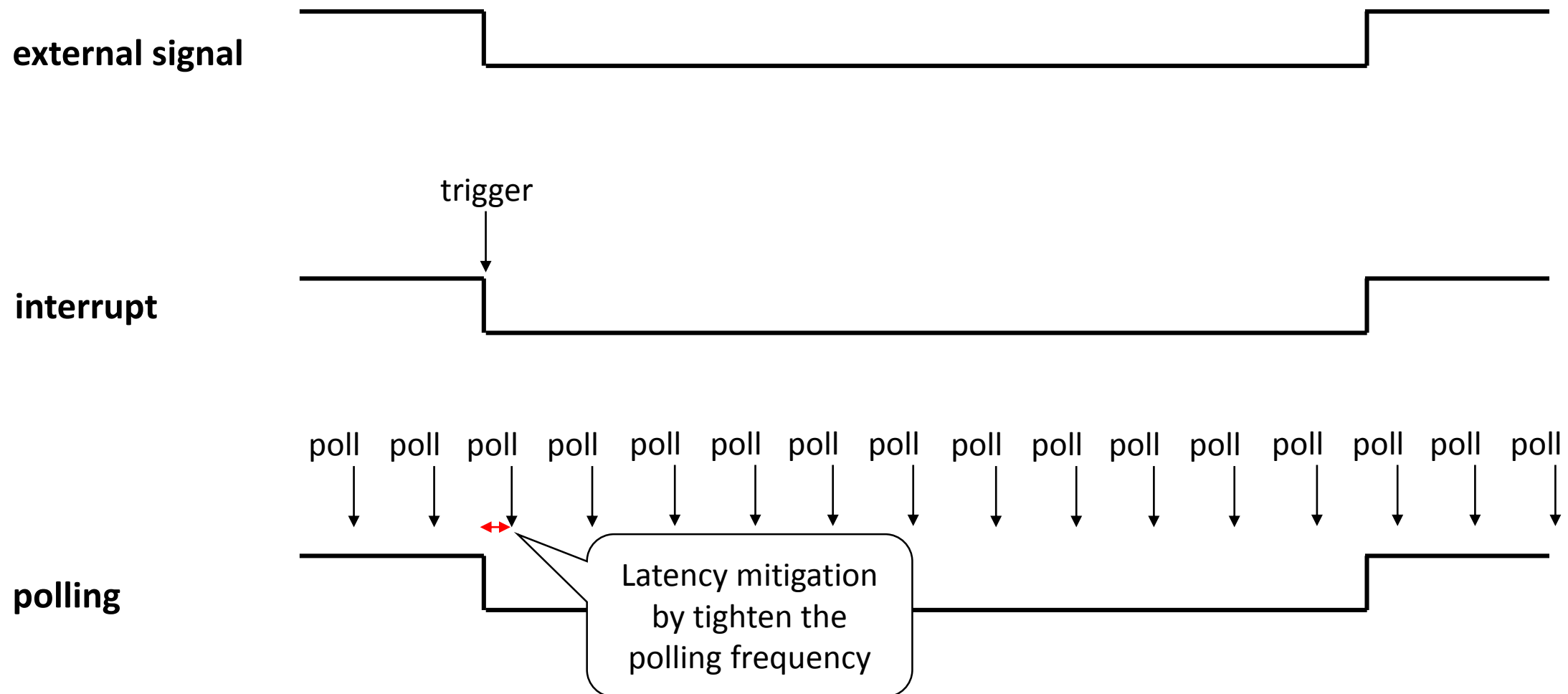
Polling vs Interrupt (II)



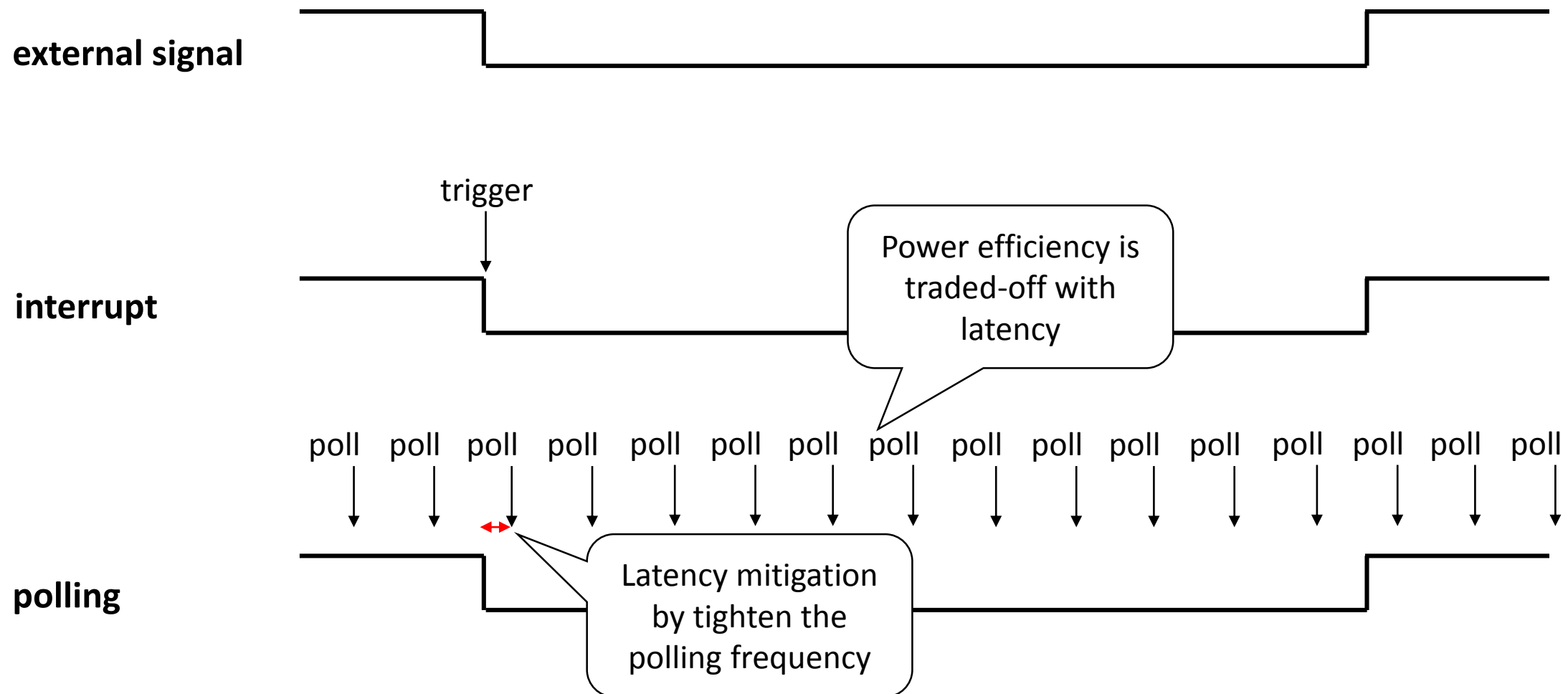
Latency



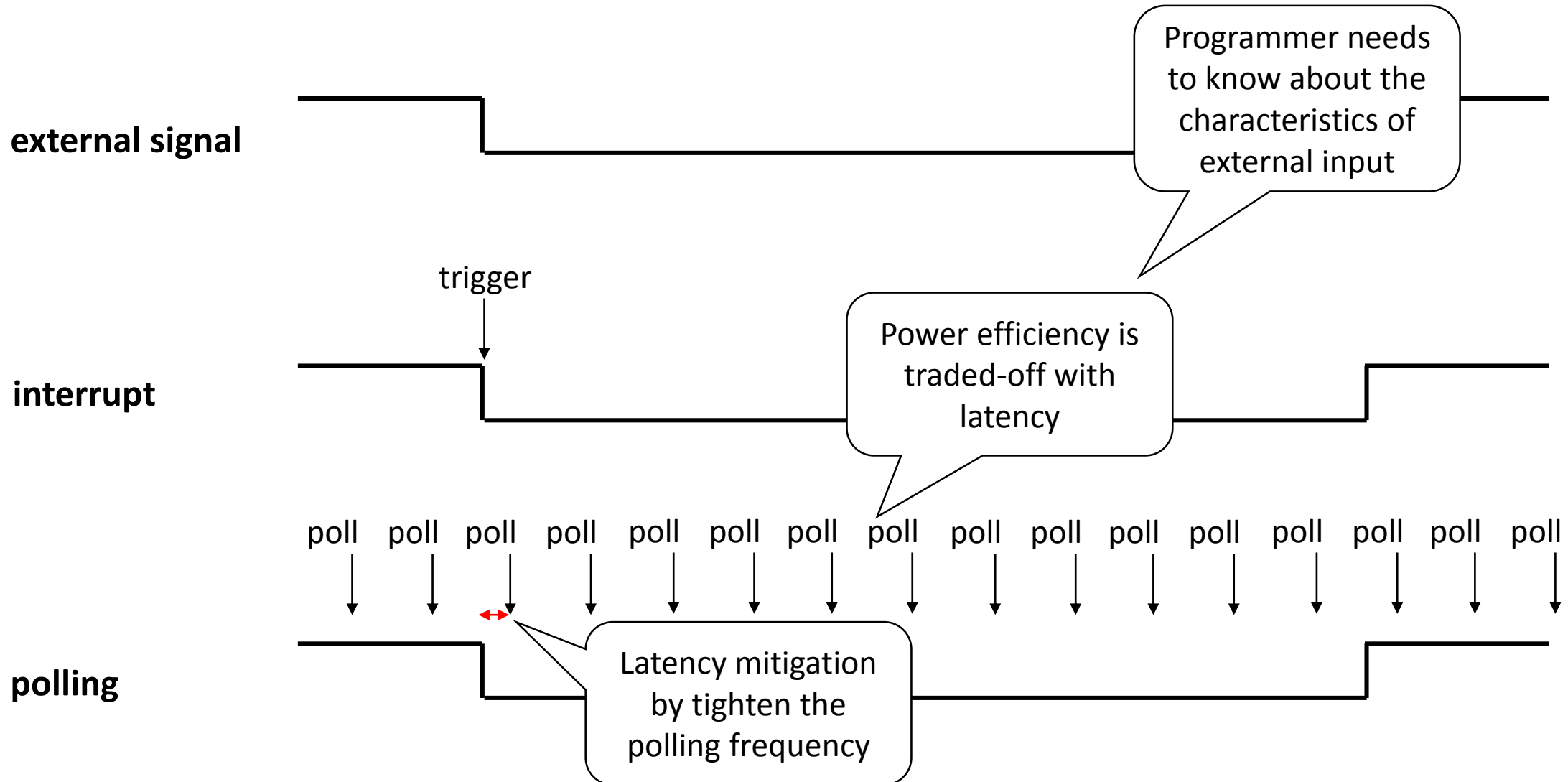
Latency



Latency



Latency



Polling implementation

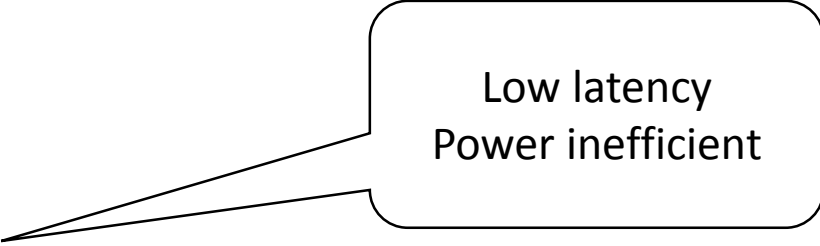
- Software approach

```
while(1)
```

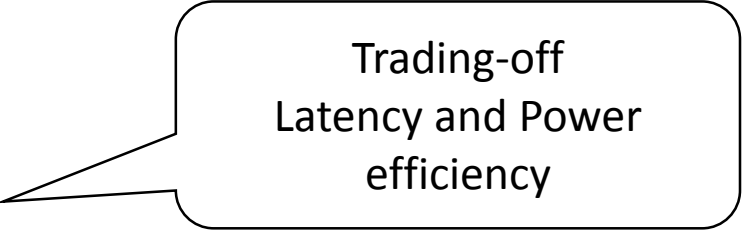
```
    poll(register);
```

- Timer based approach

- To trigger an interruption at regular intervals
- The system sleeps while the timer is counting



Low latency
Power inefficient

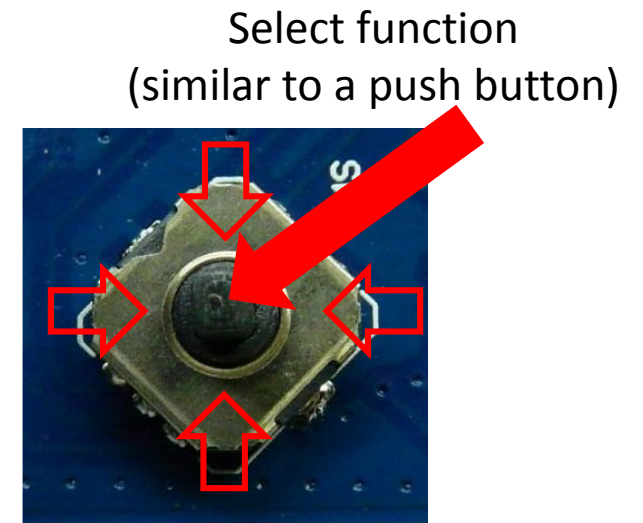


Trading-off
Latency and Power
efficiency

Case of study

The Joystick of the landtiger board

- The LandTiger LPC17XX board features a 5 way digital joystick (SW5).
- The joystick may be used for example to select options in a menu shown on the LCD.
- Each direction (up, down, left, right) and the Select function are connected to a dedicated digital input pin on the LPC1768.
- Multiple keys can be pressed at the same time (e.g. up and right).
- Input pins are active low when a key is pressed.
- The input pins are hardware debounced.

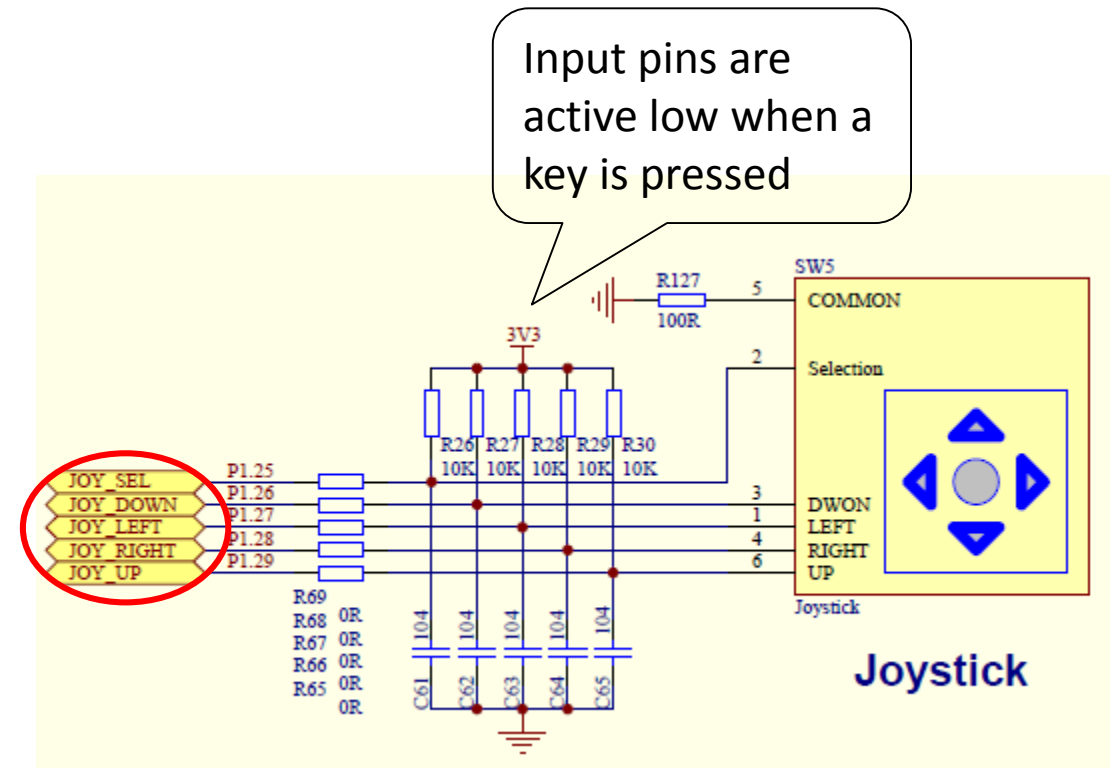
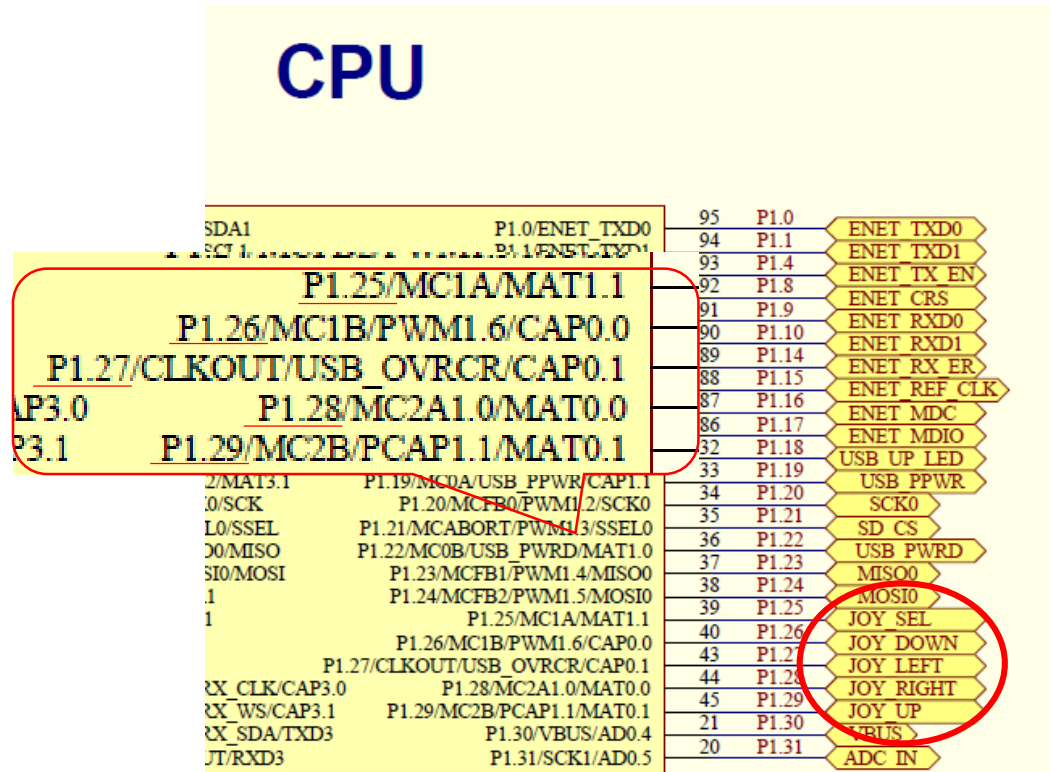


Limitations

- The joystick is connect to pins not owning external interrupt capabilities
- Therefore the only way to read its value is to
 - Setup the pins connected to the 5-way joystick actuators as
 - GPIO
 - in input direction
 - Poll the GPIO register value
 - Retrieving a full port value
 - Then making the proper bits to selectively notice a change of status

GPIO identification

- From the schematic document of the board.



- The functionalities of the RIT (Repetitive Interrupt Timer) are used to implement the polling functionalities
- Every time (50ms) the RIT triggers an interrupt
 - This timing looks fine for interfacing human a behavior (finger pressure)
- Importantly, the input pins are hardware debounced

Joystick - Select function

```
26 void RIT_IRQHandler (void)
27 {
28     static int select=0;
29
30     if((LPC_GPIO1->FIOPIN & (1<<25)) == 0){
31         /* Joytick Select pressed */
32         select++;
33         switch(select){
34             case 0:
35                 /* your action here */
36                 break;
37             default:
38                 break;
39         }
40     }
41     else{
42         select=0;
43     }
```

- In the RIT Handler
 - A. The value of the port GPIO1 is read
 - B. If the value of bit 25 is 0 then
 1. If it is the first time: an action is performed
 2. If it is a pressure repetition, no action is performed (push button functionality).

Joystick - Select function

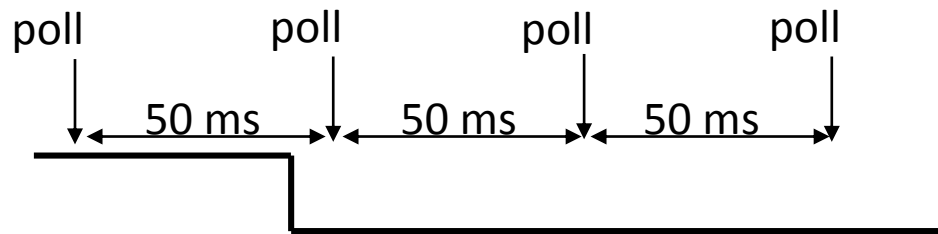
```
26 void RIT_IRQHandler (void)
27 {
28     static int select=0;
29
30     if((LPC_GPIO1->FIOPIN & (1<<25)) == 0){
31         /* Joystick Select pressed */
32         select++;
33         switch(select){
34             case 1:
35                 /* your action here */
36                 break;
37             default:
38                 break;
39         }
40     }
41     else{
42         select=0;
43     }
```

Input pins are active low when a key is pressed

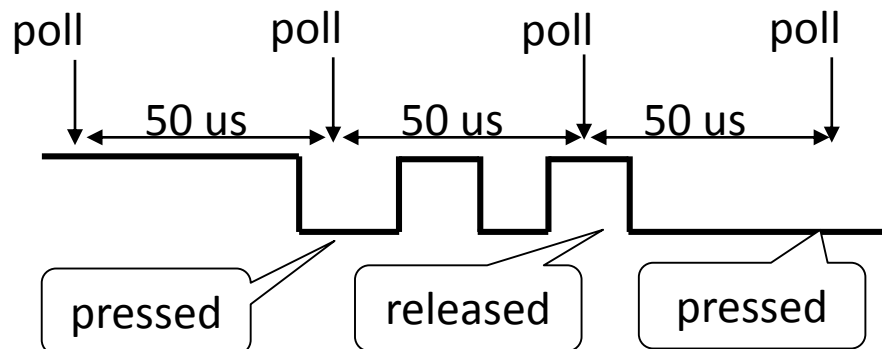
No action is performed if there is a prolonged pressure

Bouncing and prolonged pressure

- Current scenario (hw debounced)



- Potential scenario (with faster polling and bouncing issues)



```
26 void RIT_IRQHandler (void)
27 {
28     static int select=0;
29
30     if( (LPC_GPIO1->FIOPIN &
31         /* Joytick Select pre
32         select++;
33         switch(select){
34             case 1:
35                 /* your action here
36                 break;
37             default:
38                 break;
39         }
40     }
41     else{
42         select=0;
43     }
```

This value may be changed by a multiple of the polling interval, depending on the bouncing timing characteristics

In the potential scenario could be 3

Other cases can be included to manage special joystick functionalities, like prolonged pressure

Exercise (from slides 18_Switch_Bouncing)

- Experiment switch bouncing with your board and try to mitigate Key bouncing: they must use the external interrupt functionalities

Advanced -> Joystick: i
also able to mitigate d

Quite Advanced -> c
contemporary use of b

RIT is used for polling the joystick, so it cannot be enabled/disabled as ordinary done for debouncing the button

controlled polling strategy»

sur of many buttons or the

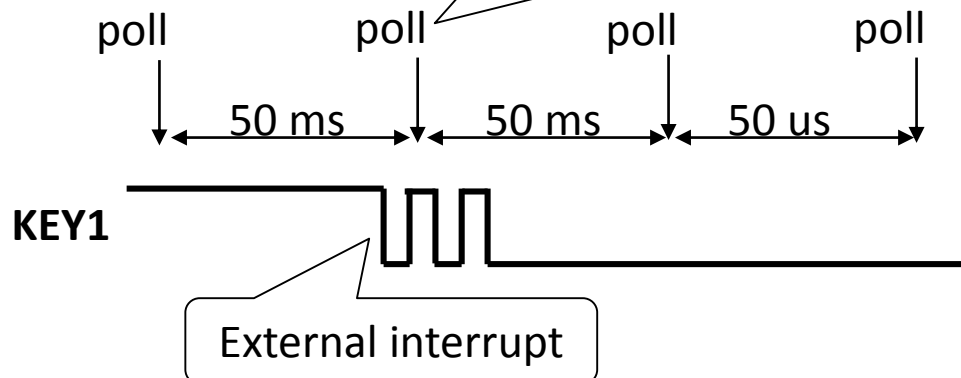
Super-Advanced -> implement button and joystick debouncing by using the RIT only.

Button and joystick by using the RIT only

```
4 extern int down;
5
6 void EINT1_IRQHandler (void) /* KEY1
7 {
8     NVIC_DisableIRQ(EINT1_IRQn); /* disable
9     LPC_PINCON->PINSEL4    &= ~(1 << 22);
10    down=1;
11    LPC_SC->EXTINT &= (1 << 1); /* clear
12 }
```

The down variable is used to synchronize the handlers

This read operation may be unreliable



```
24 volatile int down=0;
25
26 void RIT_IRQHandler (void)
27 {
28     static int select=0;
29
30     if((LPC_GPIO1->FIOPIN & (1<<25)) == 0){ /* Joytick Select pre
31         select++;
32         switch(select){
33             case 1:
34                 /* your action here */
35                 break;
36             default:
37                 break;
38         }
39     }
40     else
41         select=0;
42
43     if(down!=0){ /* button management */
44         if((LPC_GPIO2->FIOPIN & (1<<11)) == 0){ /* KEY1 pressed */
45             down++;
46             switch(down){
47                 case 2:
48                     /* your action here */
49                     break;
50                 default:
51                     break;
52             }
53         }
54         else { /* button released */
55             down=0;
56             NVIC_EnableIRQ(EINT1_IRQn); /* enable Button
57             LPC_PINCON->PINSEL4    |= (1 << 22); /* External inte
58         }
59     }
```

Declare volatile to avoid optimization issues

If the Ext Int handler was executed, down is different from 0

Since the first polling read can be unreliable, the cofirmation of the pressure is given after one polling cycle