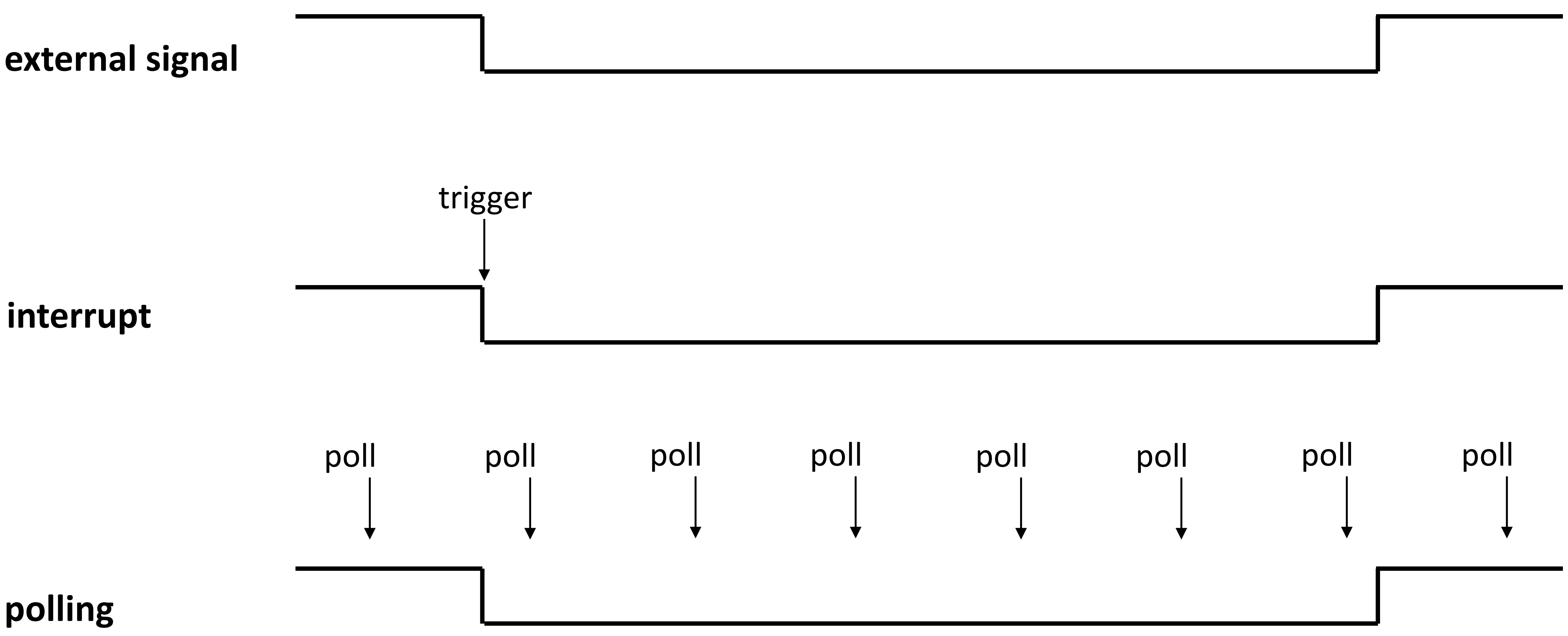
Polling switches
Case of study: the Joystick

P. Bernardi

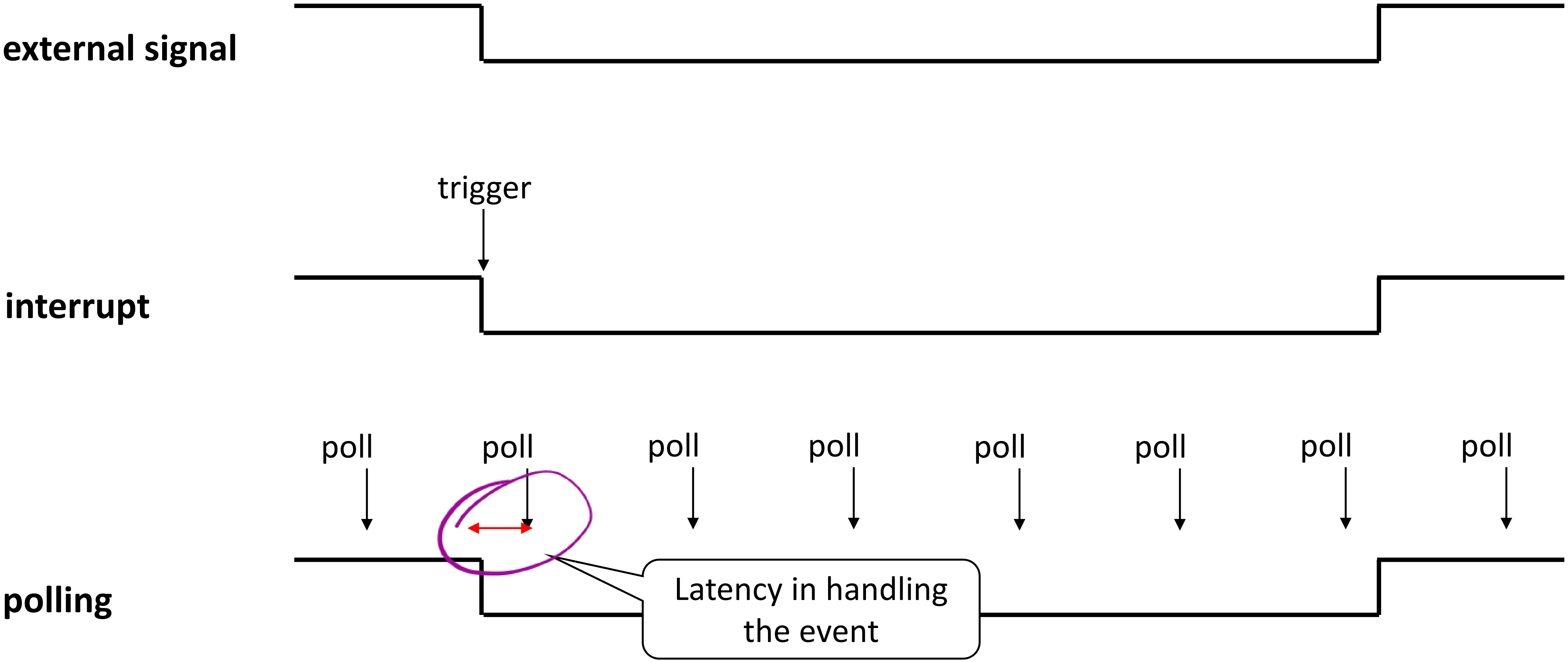
Polling Vs Interrupt

- It is usually preferred to use Interrupt functionalities to be notified about external events
  1. It is more **timing** efficient beacuse the event is handled as soon as required, without any latency (this is fundamental for safety and real time applications)
  2. It is more **power** efficient because the system sleeps between requests (this is fundamental for personal mobile devices)
- Unfortunately, not all the events are triggering interrupts
  - 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).

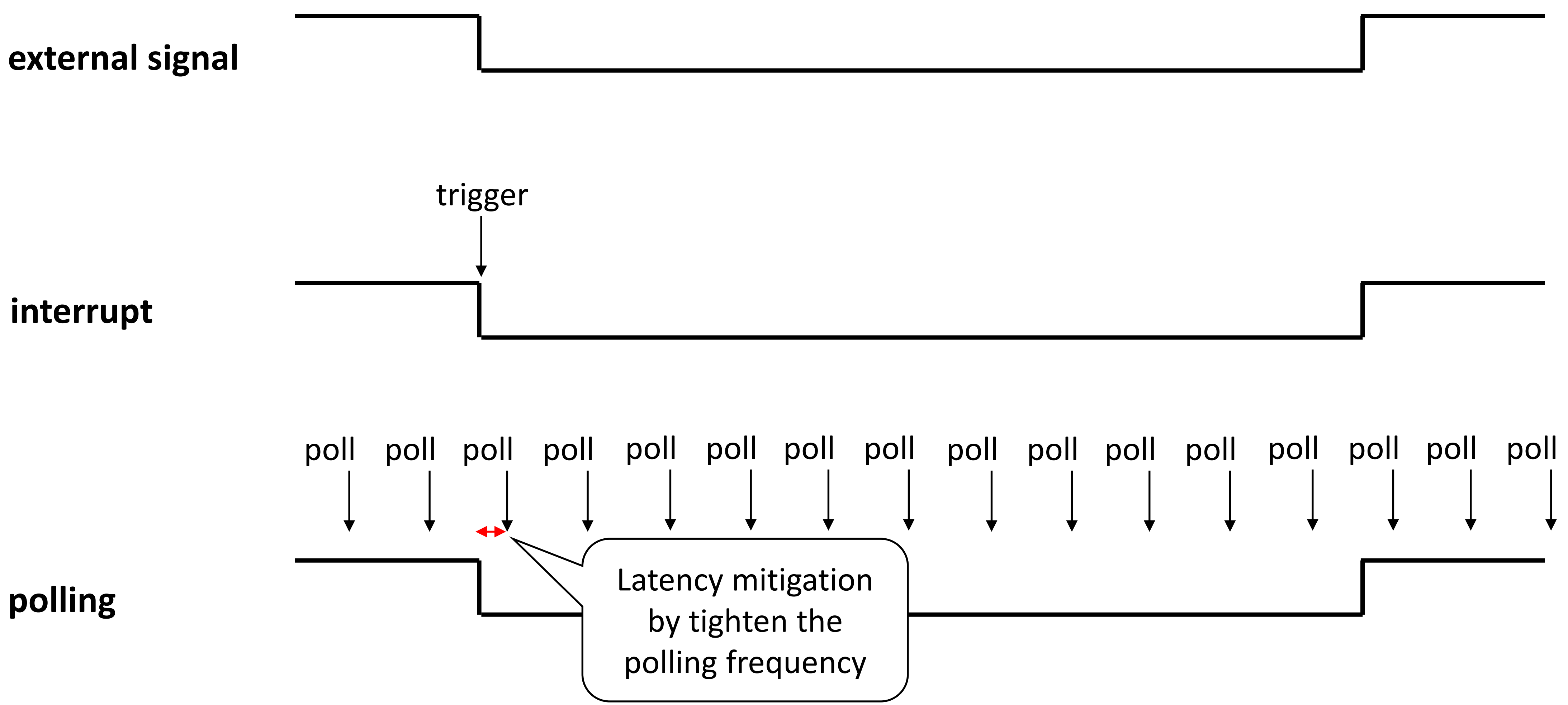
Polling vs Interrupt (II)



Latency

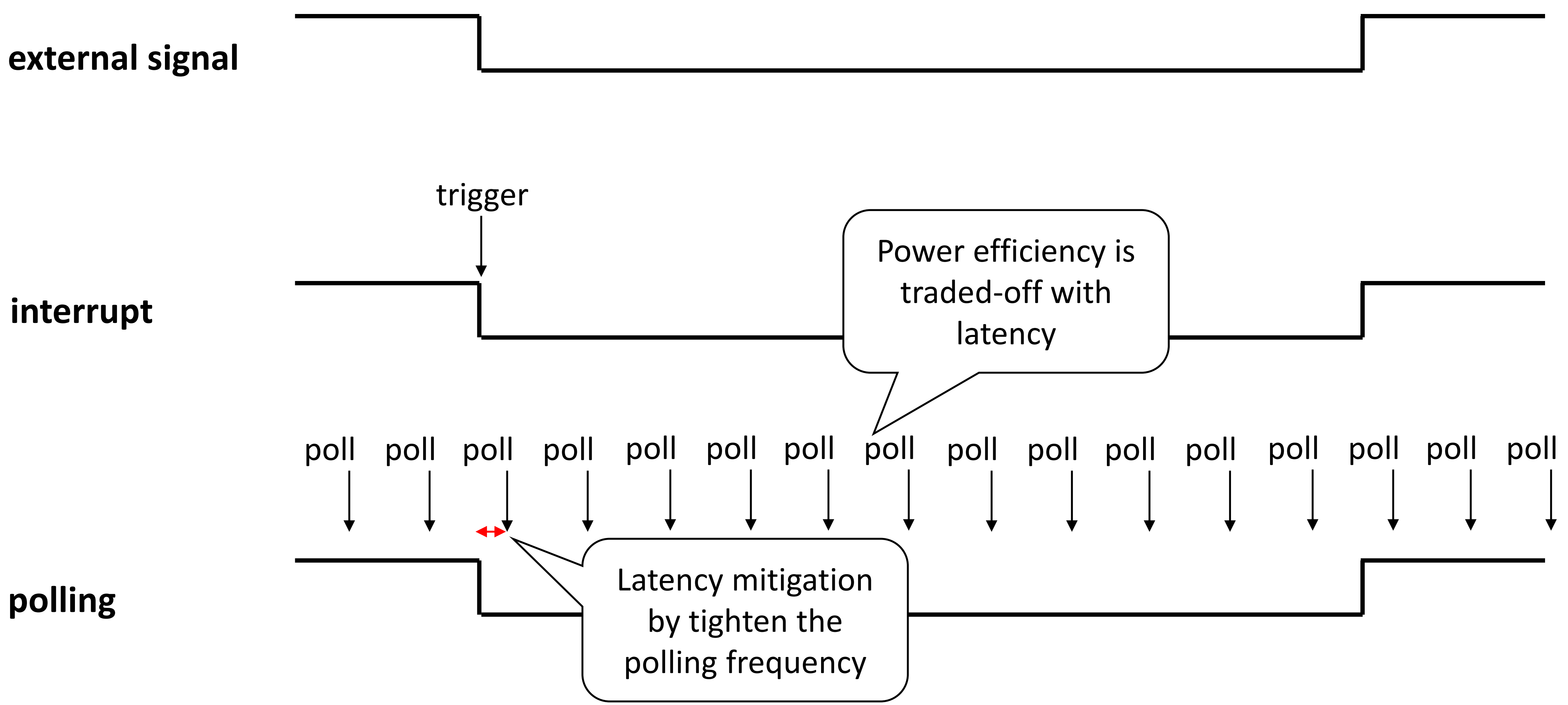

external signal
trigger
interrupt
poll
poll
poll
poll
poll
poll
poll
poll
polling
Latency in handling the event

Latency



Latency mitigation by tighten the polling frequency

Latency



external signal

trigger

interrupt

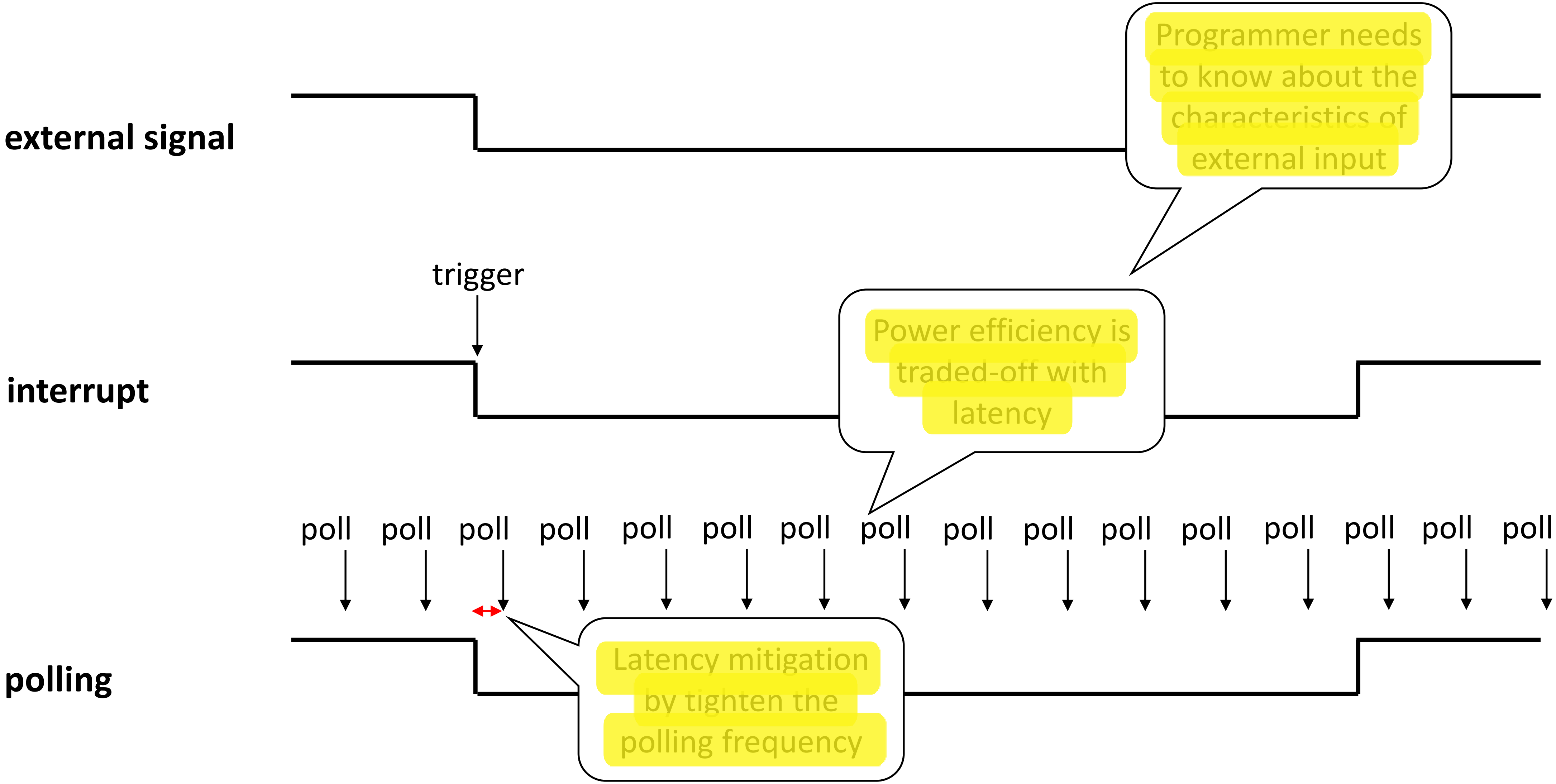
Power efficiency is traded-off with latency

poll poll poll poll poll poll poll poll poll poll poll poll poll poll poll poll

polling

Latency mitigation by tighten the polling frequency

Latency



Programmer needs to know about the characteristics of external input

Power efficiency is traded-off with latency

Latency mitigation by tighten the polling frequency

Polling implementation

- Software approach

```
while(1)
{
        poll(register1);
        poll(register2);
        poll(register3);
}
```

Latency and
power inefficient
issues

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

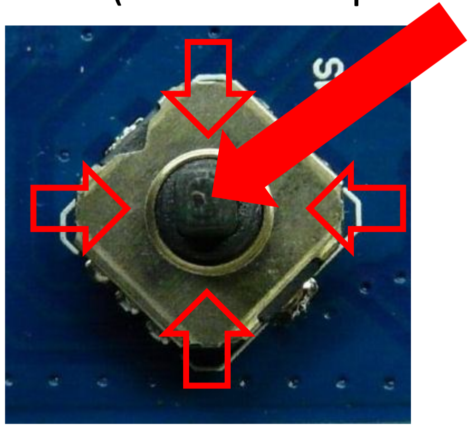
Trading-off
Latency and Power
efficiency

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 (push) 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.

Non va implementato bouncing

Select function
(similar to a push button)

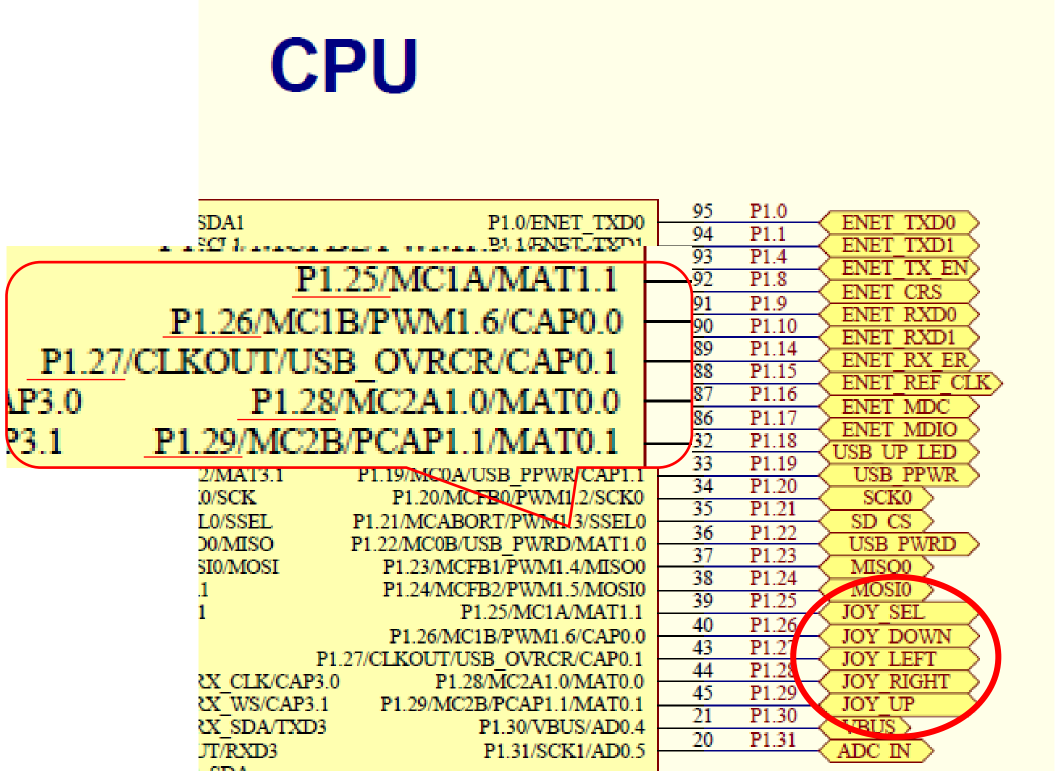


Limitations

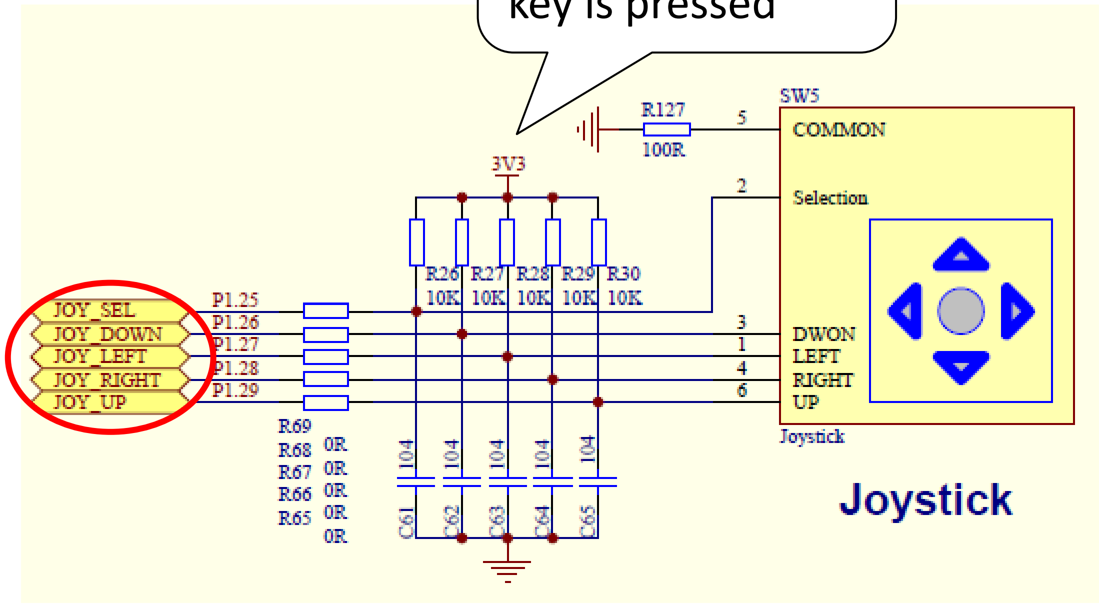
- The joystick is connect to pins not owning external interrupt capabilities
- Therefore the only way to read its value is to
  - Setup the pins connected to the 5-way joystick actuators as
    - GPIO
    - in input direction
  - Poll the GPIO register value
    - Retrieving a full port value
    - Then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

GPIO identification

- From the schematic document of the board.

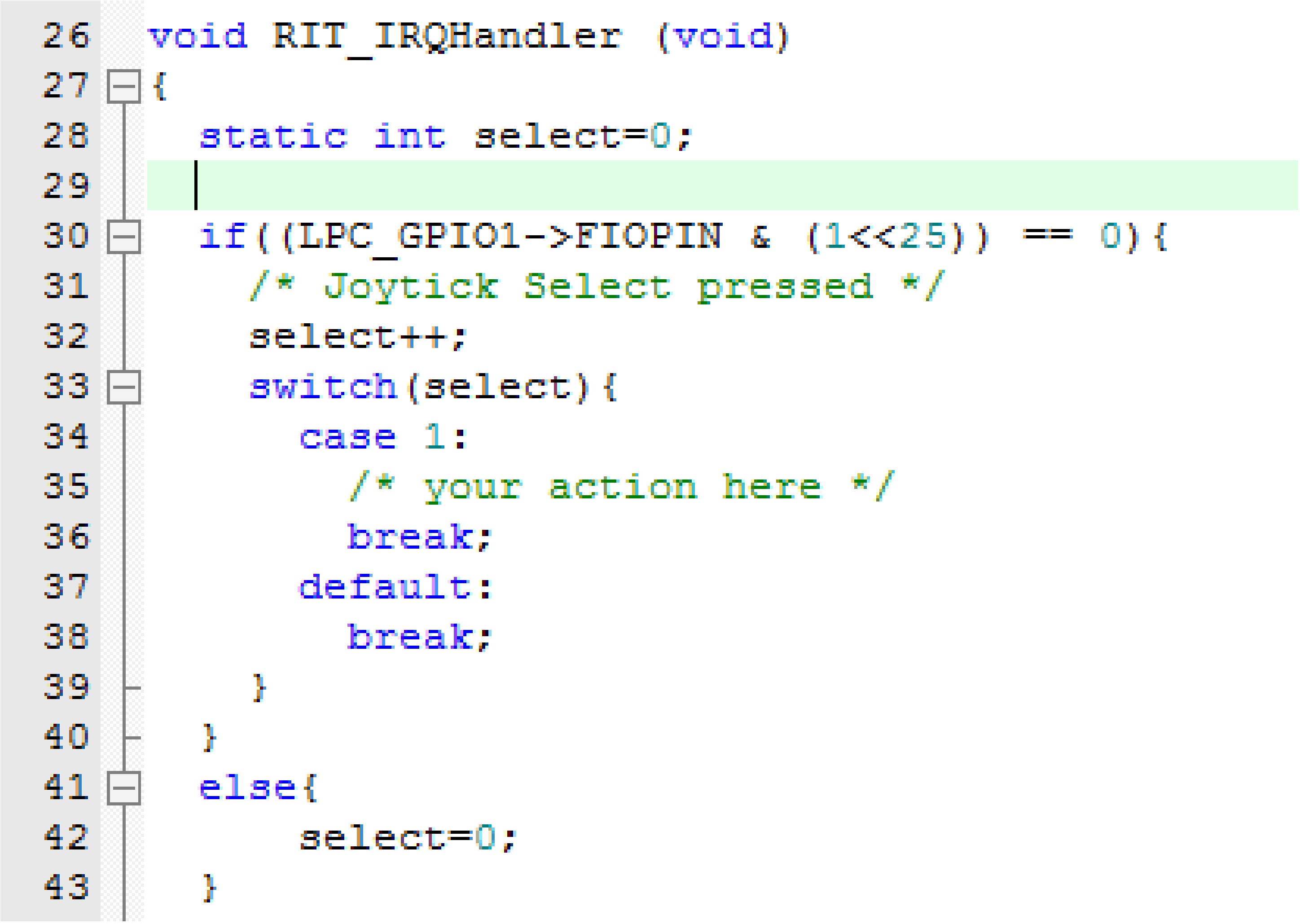


Input pins are active low when a key is pressed



Joystick - Select function

- 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 is fine for interfacing human behavior (finger pressure)
- Importantly, the input pins are <u>hardware debounced</u>



```
void RIT_IRQHandler (void)
{
  static int select=0;

  if((LPC_GPIO1->FIOPIN & (1<<25)) == 0){
    /* Joytick Select pressed */
    select++;
    switch(select){
      case 1:
        /* your action here */
        break;
      default:
        break;
    }
  }
  else{
      select=0;
  }
```

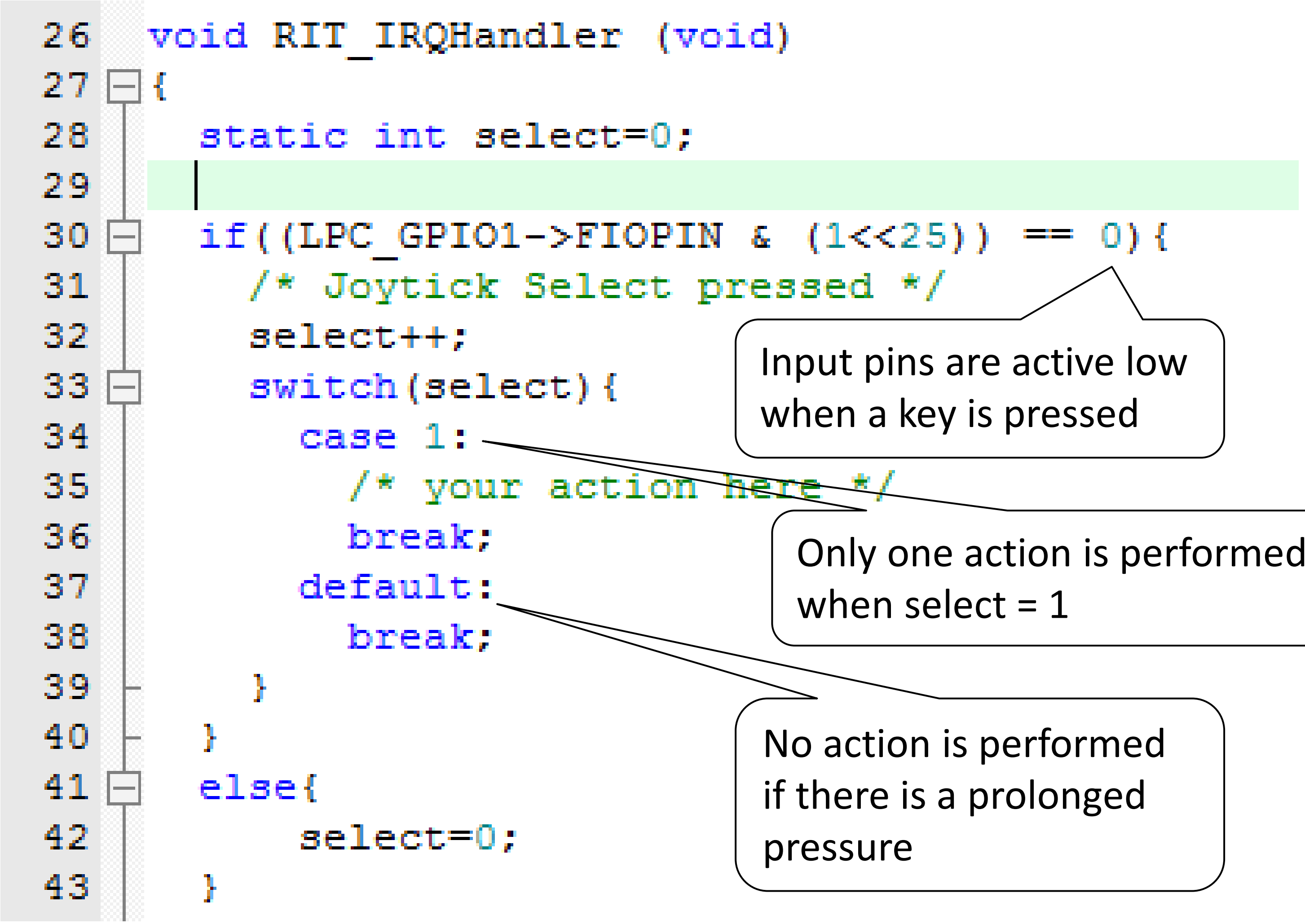
Joystick - Select function

- 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: one action is performed
2. If it is a pressure repetition, no action is peformed (push button functionality).



```
void RIT_IRQHandler (void)
{
  static int select=0;

  if((LPC_GPIO1->FIOPIN & (1<<25)) == 0){
    /* Joytick Select pressed */
    select++;
    switch(select){
      case 1:
        /* your action here */
        break;
      default:
        break;
    }
  }
  else{
      select=0;
  }
```

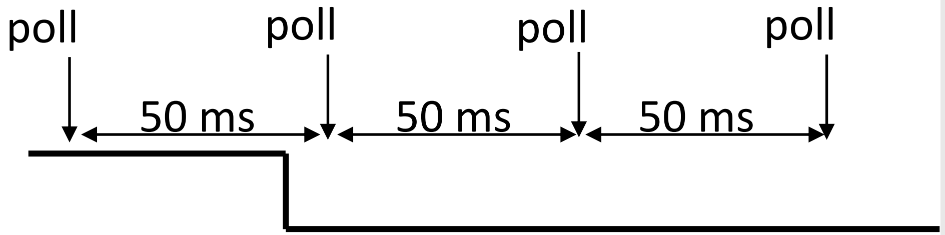
Input pins are active low when a key is pressed

Only one action is performed when select = 1

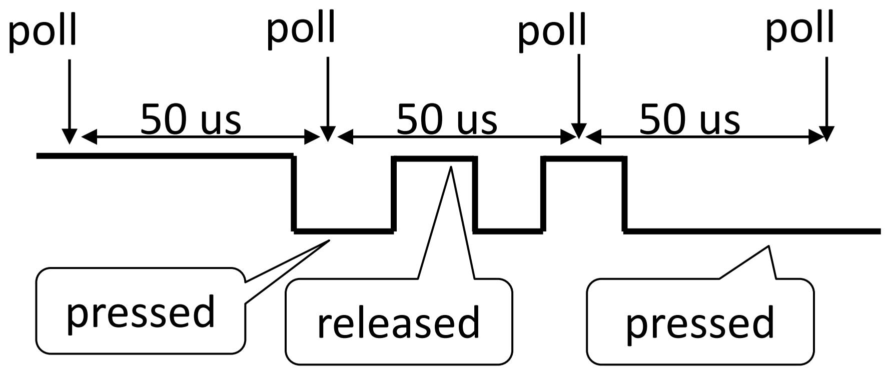
No action is performed if there is a prolonged pressure

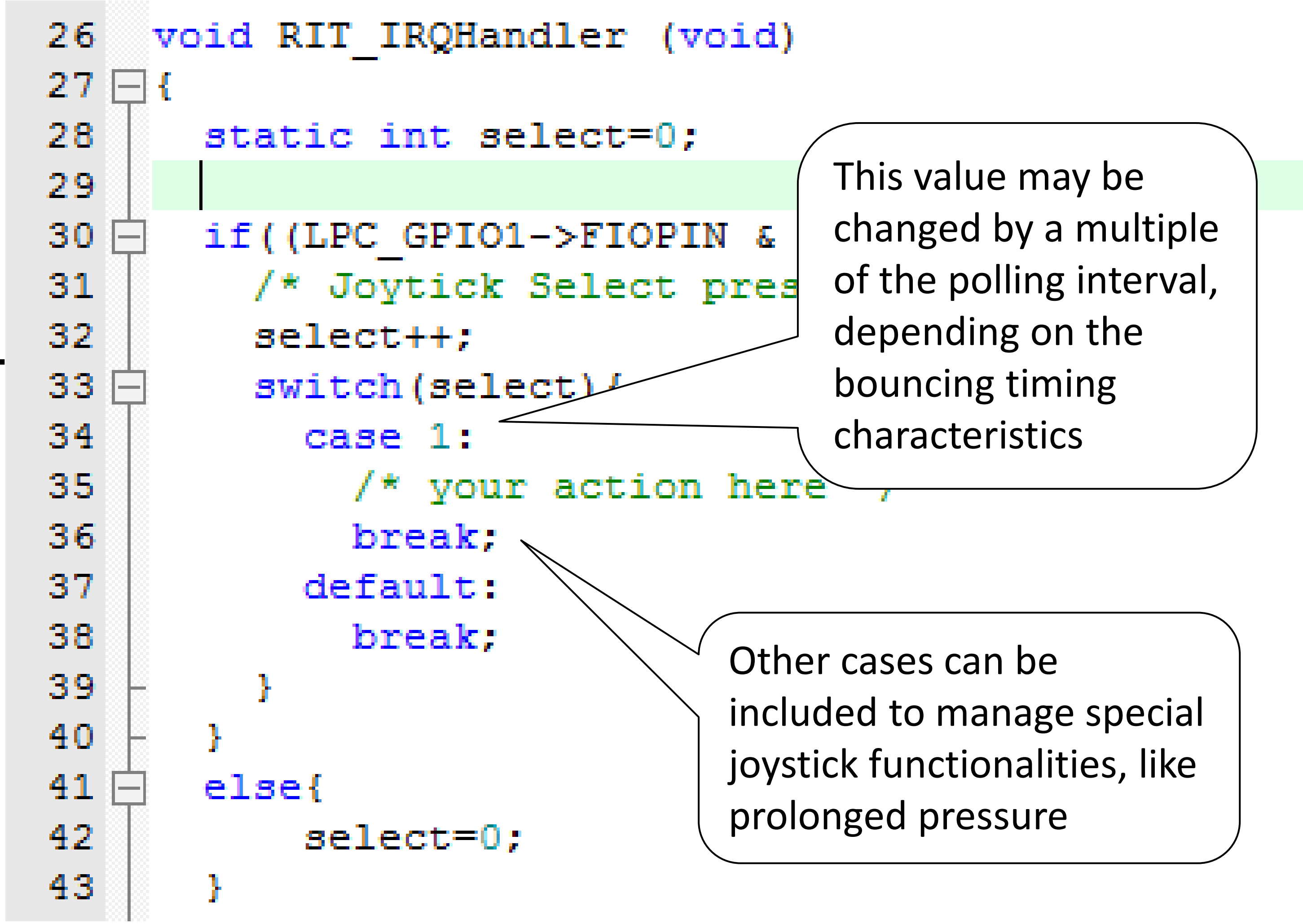
Bouncing and prolonged pressure

- Current scenario (hw debounced)



- Potential scenario (with faster polling and bouncing issues)



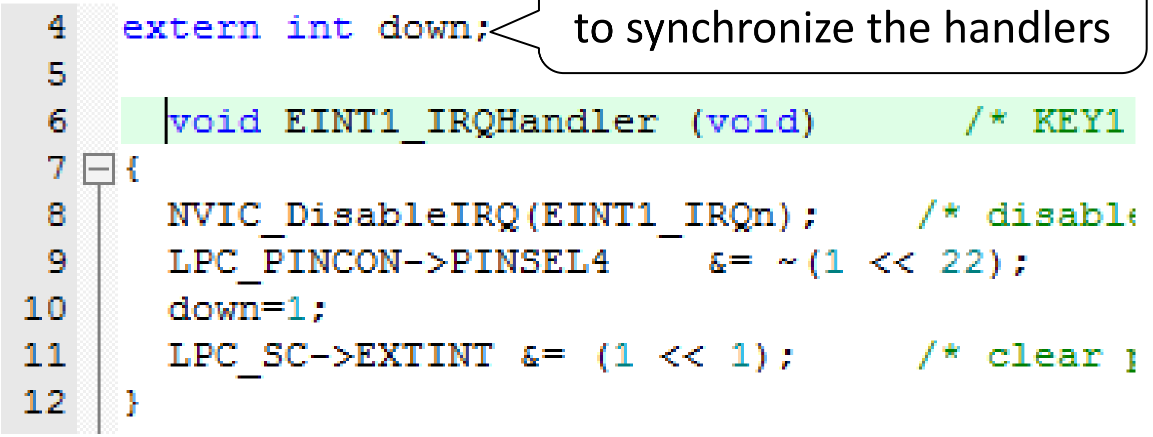


```
void RIT_IRQHandler (void)
{
  static int select=0;

  if((LPC_GPIO1->FIOPIN &
    /* Joytick Select pres
    select++;
    switch(select){
      case 1:
        /* your action here
        break;
      default:
        break;
    }
  }
  else{
      select=0;
  }
```

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

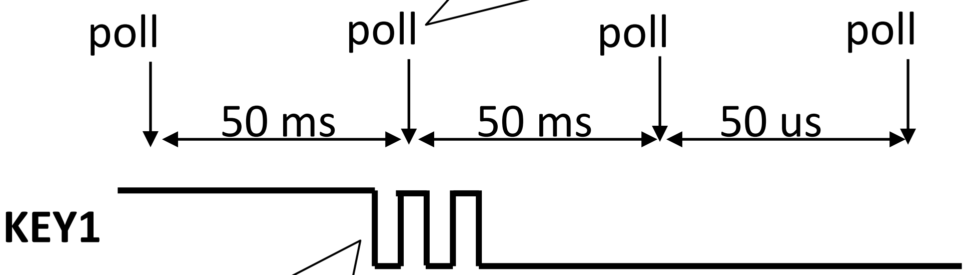
Other cases can be included to manage special joystick functionalities, like prolonged pressure

Button and joystick by using the RIT only



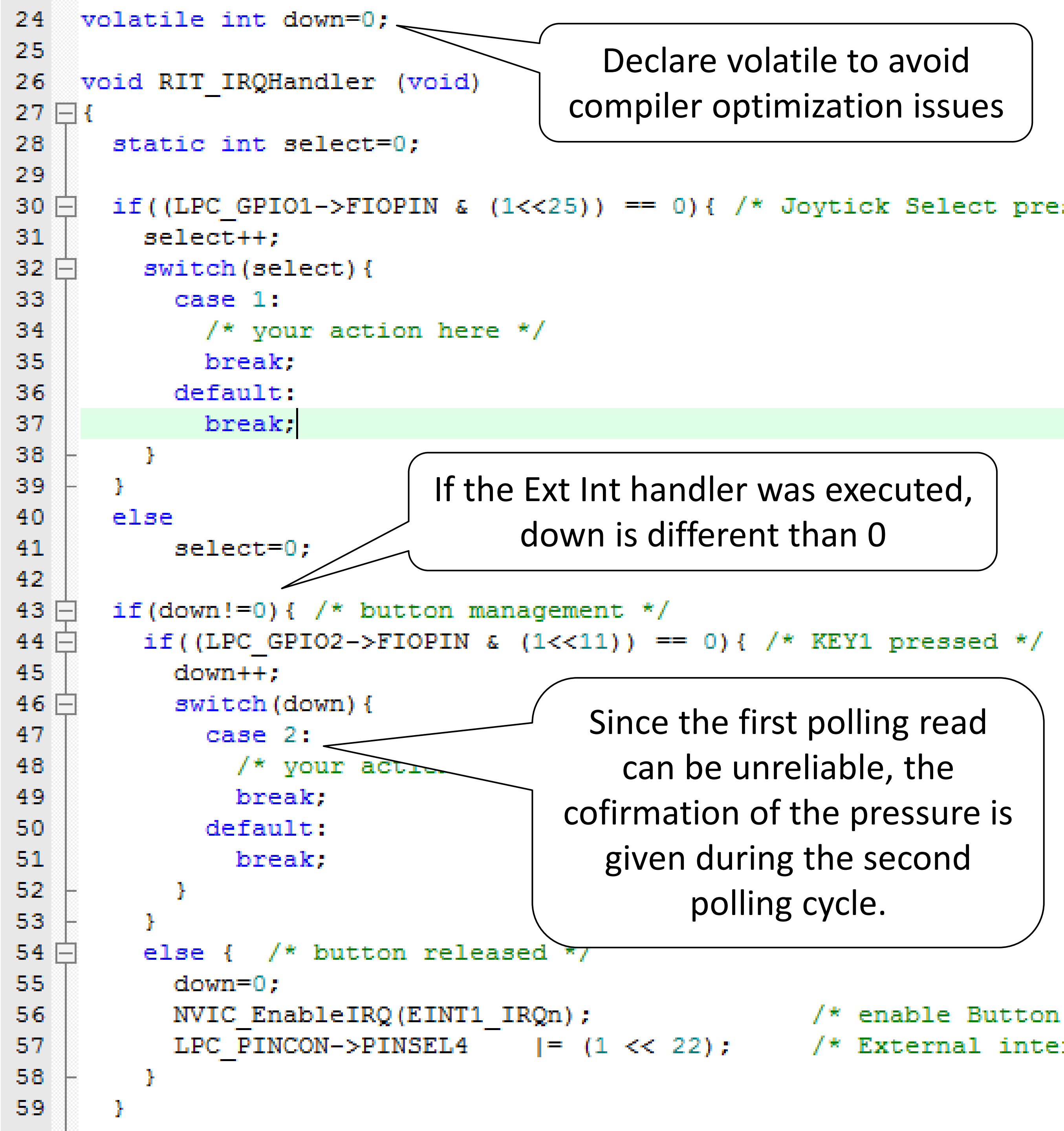
```
extern int down;

void EINT1_IRQHandler (void)          /* KEY1
{
  NVIC_DisableIRQ(EINT1_IRQn);        /* disable
  LPC_PINCON->PINSEL4     &= ~(1 << 22);
  down=1;
  LPC_SC->EXTINT &= (1 << 1);         /* clear
}
```

The down variable is used to synchronize the handlers



This read operation may be unreliable

External interrupt



```
volatile int down=0;

void RIT_IRQHandler (void)
{
  static int select=0;

  if((LPC_GPIO1->FIOPIN & (1<<25)) == 0){ /* Joytick Select pre
    select++;
    switch(select){
      case 1:
        /* your action here */
        break;
      default:
        break;
    }
  }
  else
      select=0;

  if(down!=0){ /* button management */
    if((LPC_GPIO2->FIOPIN & (1<<11)) == 0){ /* KEY1 pressed */
      down++;
      switch(down){
        case 2:
          /* your acti
          break;
        default:
          break;
      }
    }
    else {  /* button released */
      down=0;
      NVIC_EnableIRQ(EINT1_IRQn);                     /* enable Button
      LPC_PINCON->PINSEL4    |= (1 << 22);            /* External inte
    }
  }
```

Declare volatile to avoid compiler optimization issues

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

Since the first polling read can be unreliable, the cofirmation of the pressure is given during the second polling cycle.

Exercise (from slides 16_Switch_Bouncing)

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

Advanced -> Joystick: implement a «timer controlled polling strategy» also able to mitigate debouncing

Quite Advanced -> can you manage the pressur of many buttons or the contemporary use of buttons and Joystick?

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

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

Exercise

- Using the joystick up and down functions, set a value from 0-255 and show this in the LEDs.
    - Up increases the value by one
    - Down decreases the value by one
- If there is a prolonged pressure, increase or decrease the value every 400ms
- If the pressure lasts by more than 2 s, increase or decrease the value every 200ms.