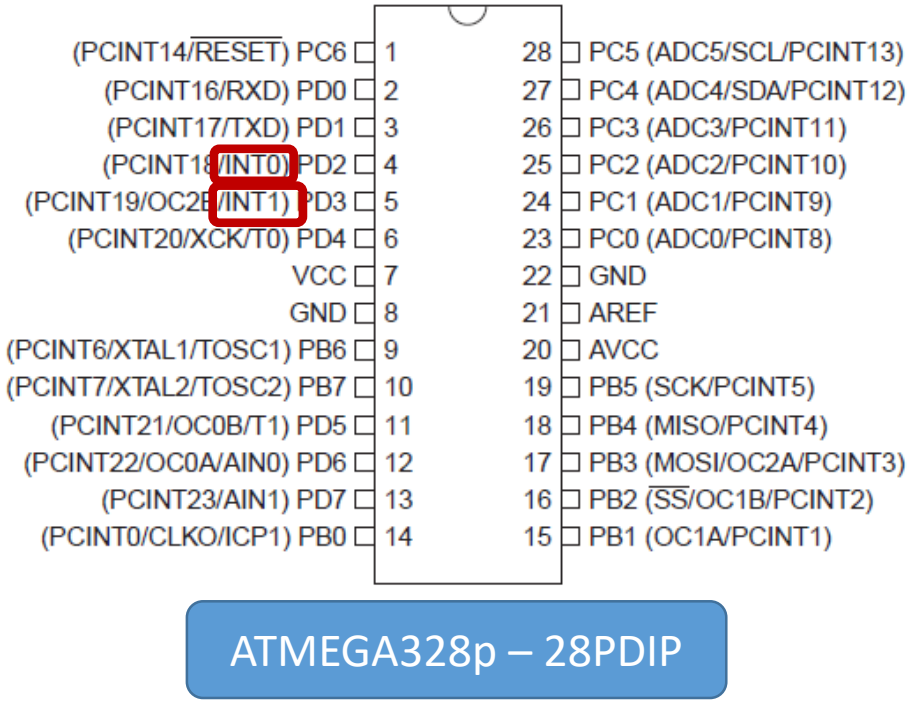
1. INTERRUPTS

External interrupts



ATMEGA328p – 28PDIP

EICRA – External Interrupt Control Register A

The External Interrupt Control Register A contains control bits for interrupt sense control.

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

INT1

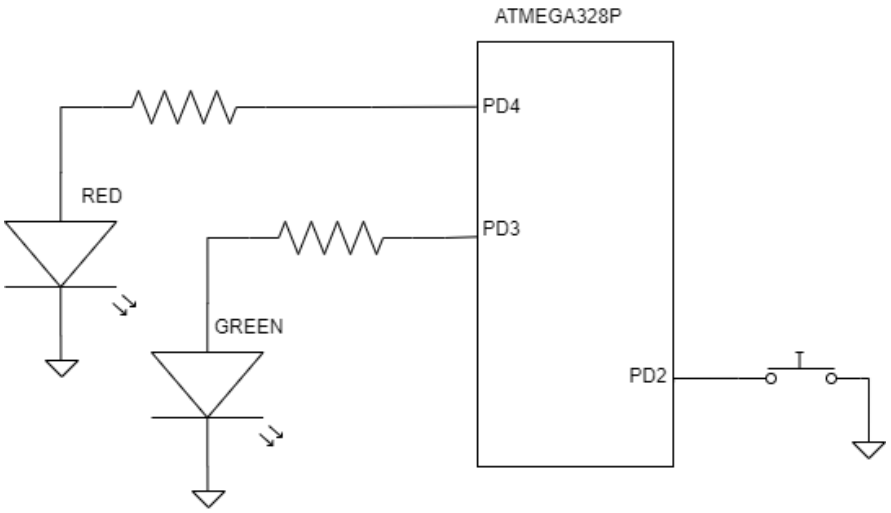
ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

INT0

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

Exercise 1

Toggle a RED LED with a pushbutton (do not take into account the bouncing issue), by means of external interrupts (INT0), while the main program is blinking a GREEN LED.



In this example, we are using a pushbutton as an input. The pushbutton is connected to PD2 of the microcontroller, which is the INT0.

The main program may be doing whatever thing, but polling to know the state of the pushbutton is not necessary. The ISR will take the control to switch ON/OFF the LED.

The problem with interrupts and buttons is that a button gives a lot of edges, and thus one button press might invoke an interrupt dozens of times. This is usually not what you want. Also, some MCUs do not like very short pulses on interrupt input, but this is not the case for AVRs.

EICRA – External Interrupt Control Register A

The External Interrupt Control Register A contains control bits for interrupt sense control.

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

EIMSK – External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	–	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

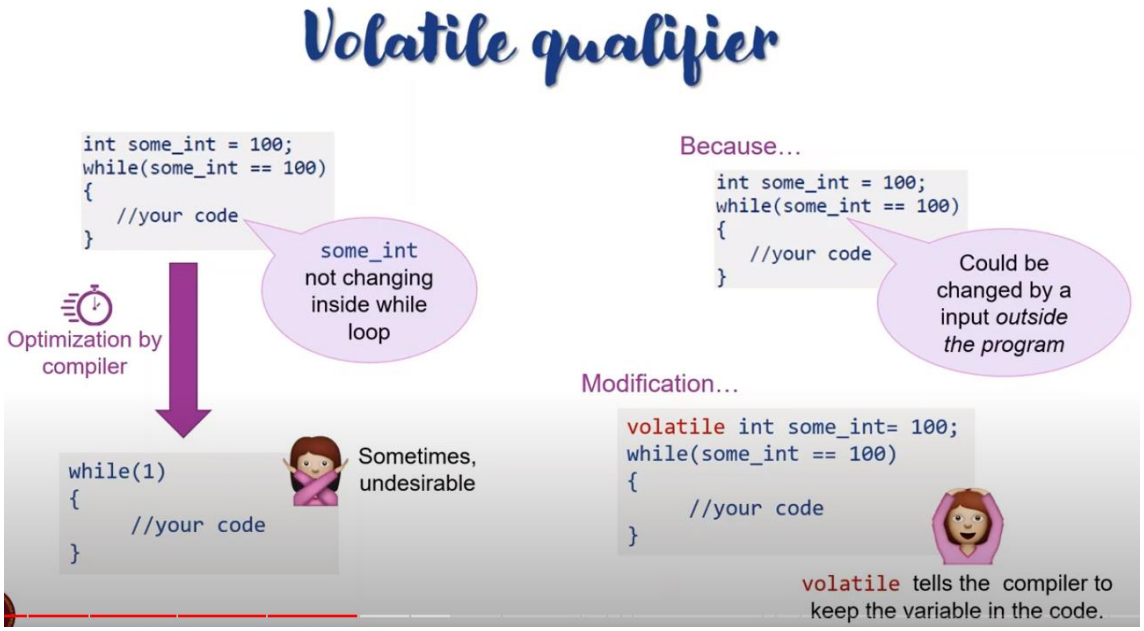
```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include "PORTS.h"
#include "MACROSh"
#include "PARAMETERS.h"

///-------- Interrupt Service Routines --------------------------
/// If INT0 interrupt is enabled, each time the pushbutton is
/// pressed, the int interrupt is triggered. Then all the actions
/// written into the ISR are executed. When it finishes, the flag is
/// cleared and the main program continues.

ISR(INT0_vect) {
      ///---------------------------------------------------
      /// 1.- Toggle the red LED
      ///---------------------------------------------------
      TOGGLEBIT (GPIO_1_OUT, RED_LED_PIN);     // Toggle the RED_LED
      //-------------------------------------------------------------------
      // 2. To avoid a new interrupt due to a falling edge caused by bouncing
      //-------------------------------------------------------------------
      _delay_ms (100);     // Although is not recommended
}


int main(void) {
/// ------------ SETUP ----------------------------------------
      /// ---------------------------------------------------------
      /// 1.- Configure ports
      /// ---------------------------------------------------------
      GPIO_init();
      /// ---------------------------------------------------------
      /// 2.-  Configure external interrupt
      /// ---------------------------------------------------------
      EICRA = (1<<ISC01) |(0<<ISC00) |   // INT0 is triggered by a falling edge
              (0<<ISC11) |(0<<ISC10);    // INT1 is set as default value (0)
      EIMSK |= (1<<INT0);                // Enable the ISR of INT0;
      /// ---------------------------------------------------------
      /// 3.- Enable global interrupts
      /// ---------------------------------------------------------
      sei();                             // Equivalent to SREG |=(1<<I);

///------------ LOOP ----------------------------------------
      while (1) {
      /// ---------------------------------------------------------
      /// 1.- Toggle the green LD
      /// ---------------------------------------------------------

            TOGGLEBIT (GPIO_1_OUT, GREEN_LED_PIN);
            _delay_ms(BLINK_TIME);
   }
}
```

Exercise 2

Assembly an electronic circuit and write a code to show (binary format) how many times a pushbutton is pressed. Use the external interrupt INT0 (PD2). Connect 4 LED to the port D (bits 3,4,5 and 6) of the microcontroller to show the number in binary format (Maximum value is 15)

Draw the proposed electronic schematic.

Note: remember to use a **volatile** variable to store the counter as it will be modified out of the main program, the ISR in this case.



Source: Lecturer Shahidatul Sadiah (University of Malasya)

URL: https://www.youtube.com/watch?v=t405Bpc8b7U&list=PLbnXeqW-iDlnWDkh5uNtHVDV4SqlnxRP9&index=15

Exercise 3

a) Assembly an electronic circuit with 4 LED connected to the port D (pins 7, 6, 5, 4). Add a pushbutton in the pin 2 of the same port.

b) The circuit must show a range of values, from 0 to F (a counter automatically increments by one). Then, every time the pushbutton is pressed, the increment of the counter is incremented by one. Set a limit to set the maximum increment, for example 4.

Tip: It is recommended to define a macro called "MAX_INCREMENT" as a user parameter.

c) After checking the system works fine, create a header file named "EXT_INT.h" and add the following inline functions to encapsulate the configuration related to the interrupt INT0. This new file should be in the folder "DCE_LIBRARIES".

- `INT0_config (uint8_t typeTrigger)`

By defining the following macros, you may pass the argument type of trigger by writing a name instead a number.

```
#define LOW_LEVEL   0    // Decimal value equivalent to ICS01=0 and ICS00=0
#define CHANGE      1    // Decimal value equivalent to ICS01=0 and ICS00=1
#define FALLING     2    // Decimal value equivalent to ICS01=1 and ICS00=0
#define RISING      3    // Decimal value equivalent to ICS01=1 and ICS00=1
```

```
static inline void INT0_config (uint8_t typeTrigger)
 {
      EICRA |= typeTrigger;
 }
```

- `INT0_enable ()`
- `INT0_disable ()`

d) Modify the "main.c" file to configure the interruption by means of the new functions.

e) For the future, add to the "EXT_INT.h" file , similar functions for the external interrupt INT1.

Exercise 4

By using the previously defined header file, connect 4 LED to the port D (pins 7,6,5,4). Add a pushbutton in pin D2. Write a code to do the next sequence (FSM):

State "allON" → All the LED are ON

State "oddON" → Only the odd LED are ON

State "evenON" → Only the even LED are ON

State "oddOFF" → All the LED are OFF

If the pushbutton is pressed the system should return to the "allON" state, regardless of what is the current and the next state.

Exercise 5

Repeat the exercise 1 but using a Pin Change interrupt instead of INT0.

a) Do you watch any difference in the performance of the exercise 1 using INT0 or PCINT? Why do you think is the reason? (You can find the explanation below)

PCICR – Pin Change Interrupt Control Register

Bit	7	6	5	4	3	2	1	0	
(0x68)	–	–	–	–	–	PCIE2	PCIE1	PCIE0	PCICR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

'1' → Enable
'0' → Disable

PCMSK2 – Pin Change Mask Register 2

Bit	7	6	5	4	3	2	1	0	
(0x6D)	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	PCMSK2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:0 – PCINT[23:16]: Pin Change Enable Mask 23...16**

Each PCINT[23:16]-bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT[23:16] is set and the PCIE2 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT[23:16] is cleared, pin change interrupt on the corresponding I/O pin is disabled.

Take into account that a pushbutton is not the most suitable device to connect to a PCINT pin. The reasons are:

- A PCINT with a pushbutton connected, triggers two interrupts, the first when the pushbutton is pressed (falling-edge) and the second when it is released (rising-edge)
- A pushbutton suffer from bouncing and it is not recommended using "delay" function within the ISR in order to debounce.

PCMSK1 – Pin Change Mask Register 1

Bit	7	6	5	4	3	2	1	0	
(0x6C)	–	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	PCMSK1
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PCMSK0 – Pin Change Mask Register 0

Bit	7	6	5	4	3	2	1	0	
(0x6B)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

b) Add static inline functions for this type of external interrupts to the header file "EXT_INT.h", and check it the right performance by means a new "main.c" file.

TIPS:

- *The names such as "PCINT18" are define in the "avr/io.h" file. They can be used to pass an argument to a function. (But, keep in mind those are specific names for ATMEGA)*

Explanation

Selecting only one edge from the pushbutton.

- When the ISR is triggered, reading the pin with the pushbutton connected.
- Only if the value is '0' (pushbutton is pressed, falling-edge) the ISR do something.
- When the pushbutton is released, the ISR read the input as '1' and do nothing.

This strategy is also recommended when more than one pin is capable to trigger the PCINT. In this way the action carried out depends on the pin which triggers the ISR.