Sistemas Baseados em Microprocessadores

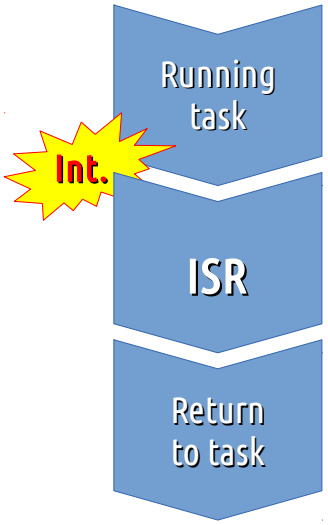
Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

ATmega328p – Interrupt System

João Paulo de Sousa

Interrupts in general

- Unexpected event
- What happens inside the CPU:
  1. It interrupts the running task
  2. It executes the correspondent ISR
     (ISR = Interrupt Service Routine)
  3. Upon completion of the ISR
     it returns to the original task
- During the ISR execution the original task is stopped!

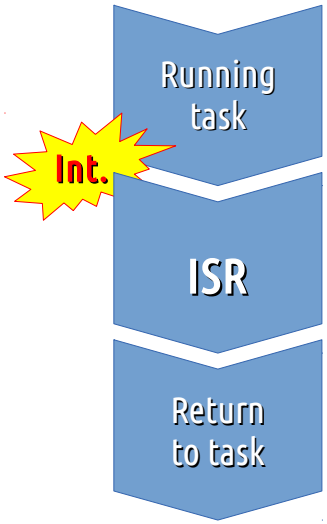


Interrupts in general

- Three important concepts:
  1) Stack: memory zone where temporary data is kept. Access of type LIFO (last in, first out)
  2) SP register (stack pointer): always points to the top of the stack
  3) PC register (program counter): always points to the address of the next instruction to execute

Interrupts in general

- Servicing an interrupt request:
  - Terminate the current instruction
  - Store PC on the top of the stack (save the address of the next instruction)
  - Load PC with the start address of the ISR ("jump" to the ISR)
- Return from an ISR
  - Load PC with the top of the stack



Dangerous! Why?

ATmega328p - Interrupt handling

- Triggered by unexpected events:
  - External: a pin changes state, new data arrives at the serial port, …
  - Internal: a timer reaches zero, end of conversion at the ADC, …
- 26 different interrupt sources. Priority:
  - RESET
  - INT0, INT1, 3xPCINT, WDT, 3xTimer2, 4xTimer1, 3xTimer0, SPI, 3xUSART, ADC, EEPROM, Analog comparator, TWI, SPM

ATmega328p - Interrupt handling

- Service:
  - By a specific routine for each request
  - Automatically executed (no software call)
  - Starting at fixed addresses of the program memory
    - RESET (address 0x0000)
    - INT0 (address 0x0002), INT1 (address 0x0004), …
- Configuration:
  - Define the ISR
  - Activate the global and individual enable bits

ATmega328p - Interrupt handling

- Definition of the ISR:

  `ISR(addr){…}`

  - ISR - macro defined in interrupt.h
  - addr - symbolic name defined in io.h:
    - `INT0_vect`,
    - `TIMER1_OVF_vect`,
    - …

- Example:

```
#include <avr/io.h>
#include <avr/interrupt.h>

ISR(INT0_vect){
  if(MAX == time) time = 1;
  else time++;
}

ISR(0x0002){
  if(MAX == time) time = 1;
  else time++;
}



ISR(TIMER1_OVF_vect){
  TCCR1 = TCCR1VAL;
  PORTB = PORTB ^ (1<<LED);
}
```

ATmega328p - Interrupt handling

- Individual bits
  - Defined using masks
- Global bit
  - Using masks
  - Using special macros defined in interrupt.h
    - `sei()`
    - `cli()`

- Example:

```
#include <avr/io.h>
#include <avr/interrupt.h>

/* Configure INT0 and INT1 */
DDRD &= ~(1<<INT0 | 1<<INT1);
PORTD |= (1<<INT0 | 1<<INT1);
EICRA = 0b00000000;



/* Enable INT0 and INT1 */
EIMSK = (1<<INT0) | (1<<INT1);

/* Enable global INT */
sei();
```

ATmega328p - Interrupt handling

- Serving an interrupt request:
  - Terminate the current instruction: **1..3** clock cycles
  - Clear the global enable bit (why?): **1** cycle
  - Save PC on stack and jump to ISR: **3** cycles

- Conclusion: **5..7** clock cycles to change context…

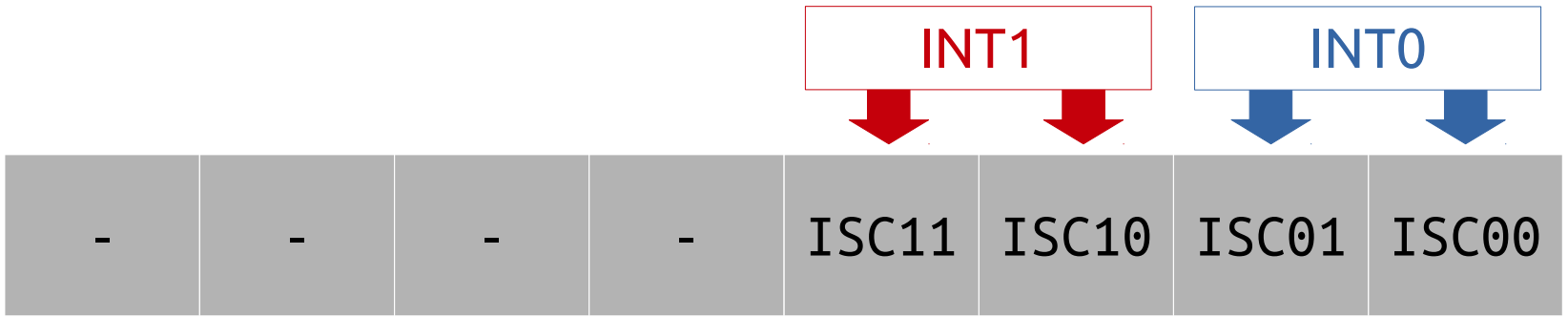
Next: external interrupts

ATmega328p - external interrupts

- Associated configuration registers:
  - Pins INT0 and INT1
    - EICRA – External Interrupt **Control** Register A
    - EIMSK – External Interrupt **Mask** Register
    - EIFR – External Interrupt **Flag** Register
  - Other pins
    - PCICR – Pin Change Interrupt Control Register
    - PCMSK2/1/0 – Pin Change Mask Registers 2, 1 and 0
    - PCIFR – Pin Change Interrupt Flag Register

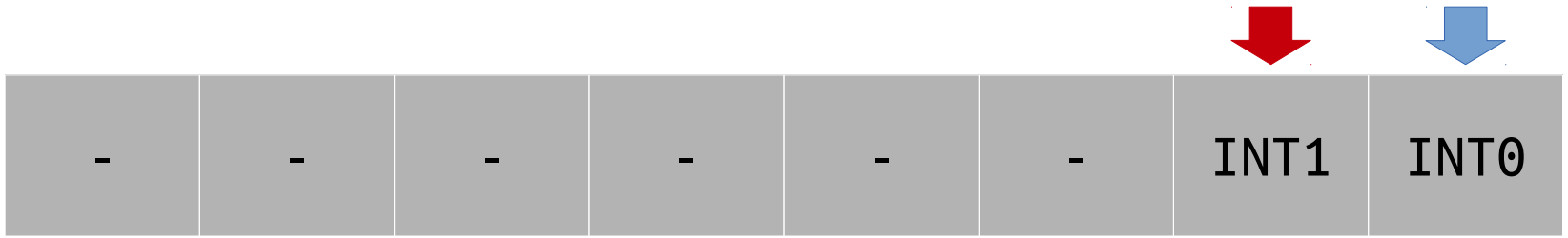
ATmega328p - external interrupts

- **EICRA** – External Interrupt Control Register A
- Two configuration bits per pin:
  - `(1,1)` Interrupt request at rising edge
  - `(1,0)` Interrupt request at falling edge
  - `(0,1)` Interrupt request at any edge
  - `(0,0)` Interrupt request at low level



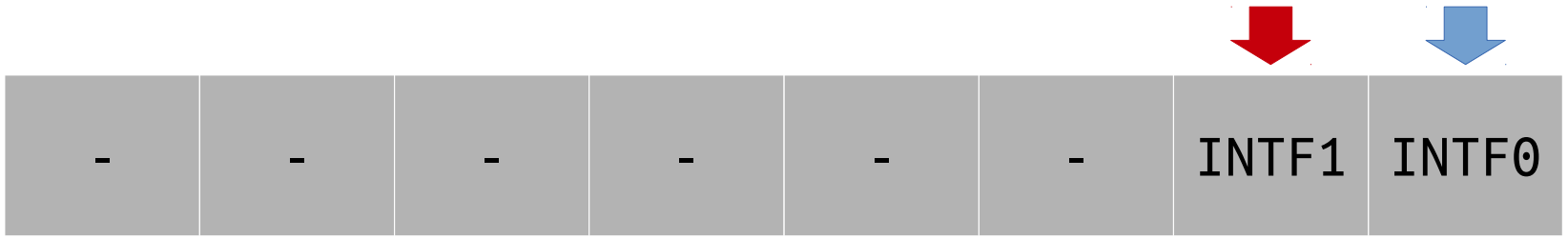
ATmega328p - external interrupts

- **EIMSK** – External Interrupt Mask Register
- One configuration bit per pin:
  - Bit INT1: controls the service of requests at pin INT1
  - Bit INT0: controls the service of requests at pin INT0
- Do not forget the global configuration bit
  - sei(), cli()



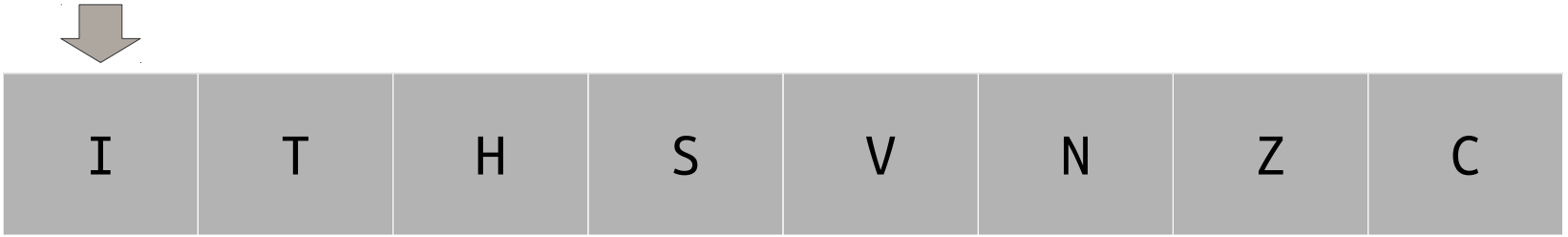
ATmega328p - external interrupts

- **EIFR** – External Interrupt Flag Register
- Two state bits (active high):
  - Signal an interrupt request at the correspondent pin
  - Automatically cleared when returning from the ISR
  - Manually cleared by writing a one (!)
  - Usually we don't need to test them (why?)



ATmega328p - external interrupts

- Global configuration bit
  - Most significant bit of the state register (flags)
  - Saved & cleared automatically when entering the ISR
  - Restored when leaving the ISR
- Usually handled by the `sei()` and `cli()` macros defined in interrupt.h



ATmega328p - external interrupts

- Problem:
  - A buzzer sounds at a fixed rate, initially 320ms
  - Two push buttons control the rate: each time, one button doubles the rate and the other halves it

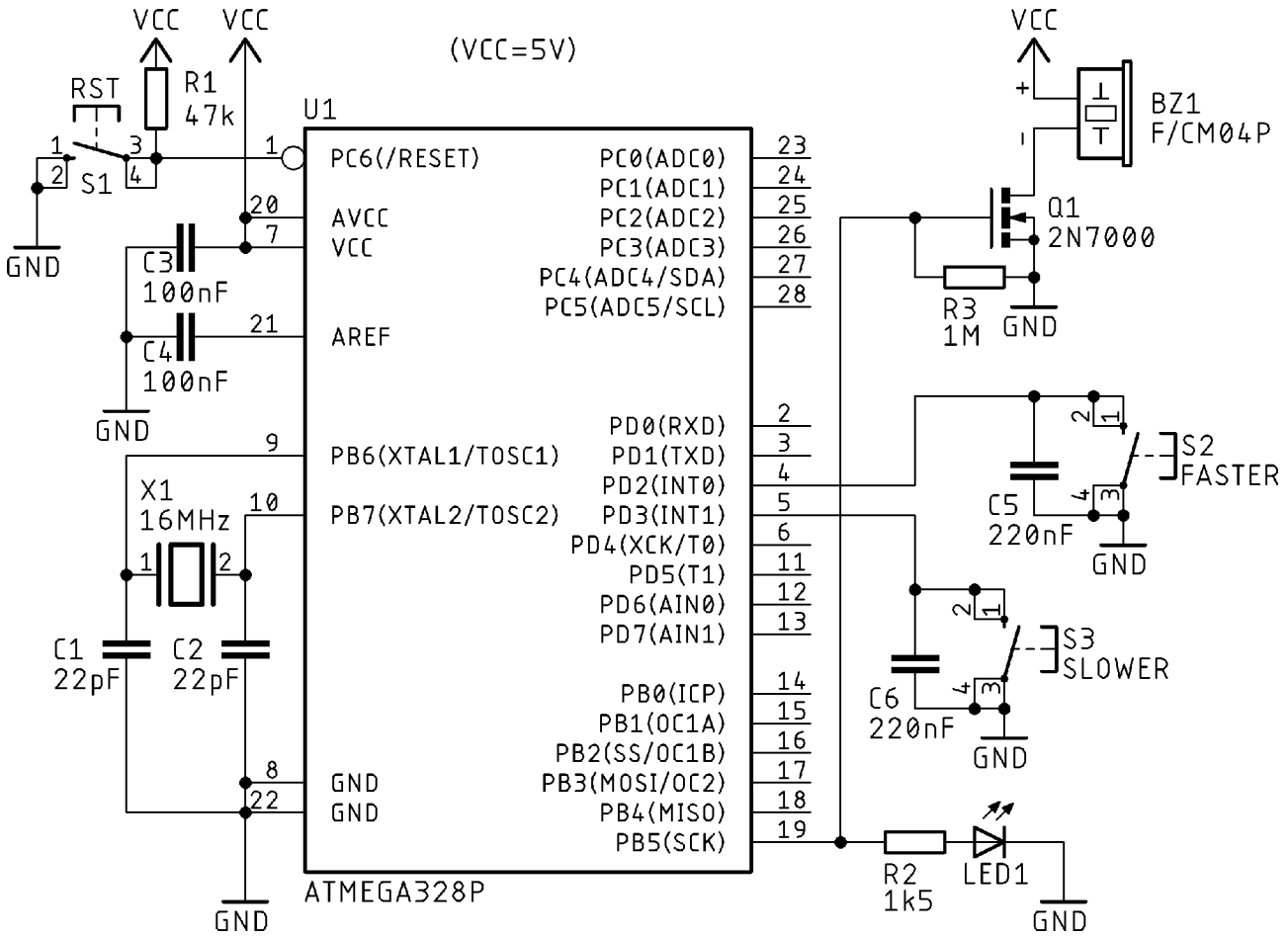
ATmega328p - external interrupts example

- Problem:
  - A buzzer sounds at a fixed rate, initially 320ms
  - Two push buttons control the rate: each time, one button doubles the rate and the other halves it
- Solution:
  - The toggling rate is stored in a variable
  - Each button is connected to one external interrupt pin
  - One ISR doubles the variable, the other halves it
  - main() assures an endless loop

ATmega328p - external interrupts example

Hardware:

- PB5 as output
- PD2, PD3 as inputs with pull-up
- Values?
  - C5, C6
  - R3
  - R2



ATmega328p - external interrupts example

- Software:

```
/*****************************************
 *  interrupts.c
 *  External interrupts demo in plain C
 *  Created on: 16/09/2014 (eclipse, avr-gcc)
 *      Author: jpsousa@fe.up.pt
 *****************************************/

#include <avr/io.h>             /* Registers */
#include <avr/interrupt.h>     /* Interrupts */
#include <util/delay.h>    /* Delays library */

#define OUT PB5            /* LED+BUZ at PB5 */
#define FASTER PD2               /* PD2=INT0 */
#define SLOWER PD3               /* PD3=INT1 */

#define RATEVAL 32     /* Initial rate value */

uint8_t i,rate;              /* Range 0..255 */
```

```
void hw_init(void) {
  /* set LED+Buzzer pin as output */
  DDRB = DDRB | (1<<OUT);

  /* Set Interrupt pins as input
   * and activate internal pull-ups */
  DDRD = DDRD & ~((1<<FASTER) | (1<<SLOWER));
  PORTD = PORTD | (1<<FASTER) | (1<<SLOWER);

  /* Interrupt request at falling edge
   * for INT1 and INT0 */
  EICRA = EICRA | (2<<ISC10) | (2<<ISC00);

  /* Enable INT1 and INT0 */
  EIMSK = EIMSK | (1<<INT1) | (1<<INT0);

  /* Enable global interrupt flag */
  sei();
}
```

ATmega328p - external interrupts example

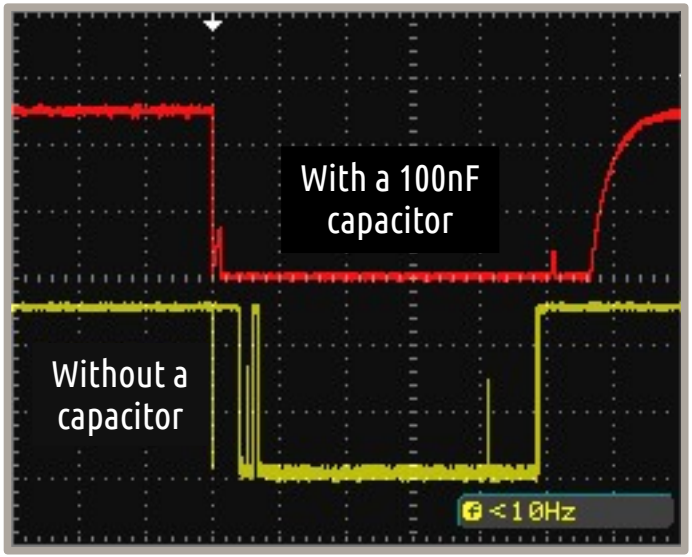
- Software:

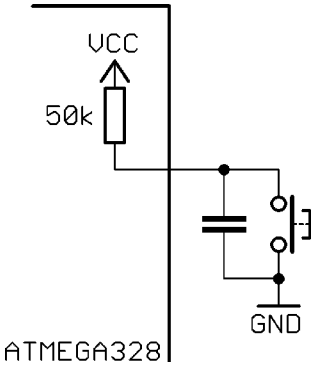
```
/* ISRs will change the rate value
 * from the list: 128-64-32-16-8-4-2 */


/* INT0 will double the rate */
ISR(INT0_vect){
  if (rate<128) rate = rate*2;
}


/* INT1 will halve it */
ISR(INT1_vect){
  if (rate>2) rate = rate/2;
}
```

```
/*
 *  Start at 320ms and increase or decrease
 *  the rate: 1280-640-320-160-80-40-20
 */

int main(void) {
  hw_init();
  rate=RATEVAL;
  while(1) {
    /* Toggle LED at a given rate */
    PORTB = PORTB ^ (1<<OUT);
    for(i=1;i<rate;i++){
      _delay_ms(10);
    }
  }
}
```

External interrupts - Discussion

- Contact bouncing
  - Low-pass filter
    (rising time: T≈2,2RC, R≈50k)
  - T=10ms @ C=100nF
  - T=25ms @ C=220nF
- Internal pull-up resistors
  - Save time and assembly costs
  - Can create problems in low-power modes...





To further explore...

- Chapters 16 and 17 of the datasheet (Moodle)
- Application notes:
  - AVR1200 - External Interrupts for megaAVR
  - AVR1201 - External Interrupts for tinyAVR
- Most important:
  - Try the example and explore variants at home:
    - Single key increases in a circular way
      (2, 4, 8, 16, 32, 64, 128, 2, 4, …)
    - Two keys increase and decrease in a circular way