

System-Level Programming

19 Interrupts – Example

J. Kleinöder, D. Lohmann, V. Sieh, P. Wägemann

Lehrstuhl für Informatik 4
Systemsoftware

Friedrich-Alexander-Universität
Erlangen-Nürnberg

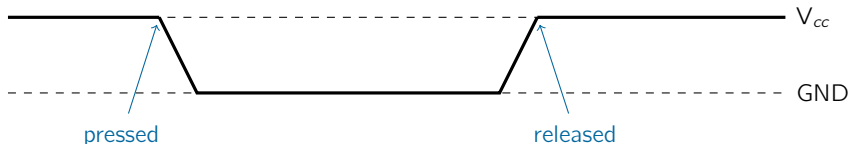
Summer Term 2024

<http://sys.cs.fau.de/lehre/ss24>



Level- and Edge-Triggered Interrupts

- Example: Signal of an **idealized** button (*active low*)



- Edge-triggered interrupt
 - Interrupt is triggered by a change of voltage (edge)
 - Usually configurable which edge (rising/falling/both) triggers an interrupt
- Level-triggered interrupt
 - Interrupt is triggered as long as a specific voltage level is present



Interrupt Control for AVR ATmega

- IRQ sources for the ATmega328PB (IRQ \mapsto *Interrupt ReQuest*)
 - 45 IRQ sources [1, S. 78]
 - individual (de)activation
 - IRQ \rightsquigarrow jump to vector address

- Wiring SPiCboard
(\hookrightarrow 17-17 \hookrightarrow 2-9)

- INT0 \mapsto PD2 \mapsto Button0
(debounced by hardware)
- INT1 \mapsto PD3 \mapsto Button1

Vector No	Program Address	Source	Interrupts definition
1	0x0000	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 0
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2_COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2_COMPB	Timer/Counter2 Compare Match B
10	0x0012	TIMER2_OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1_CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1_COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1_COMPB	Timer/Counter1 Compare Match B
14	0x001A	TIMER1_OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0_COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0_COMPB	Timer/Counter0 Compare Match B
17	0x0020	TIMER0_OVF	Timer/Counter0 Overflow
18	0x0022	SPI0_STC	SPI1 Serial Transfer Complete
19	0x0024	USART0_RX	USART0 Rx Complete
20	0x0026	USART0_UDRE	USART0, Data Register Empty
21	0x0028	USART0_TX	USART0, Tx Complete
22	0x002A	ADC	ADC Conversion Complete



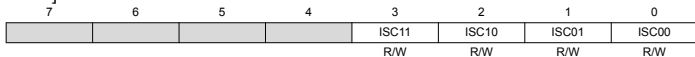
External Interrupts: Registers

Control registers for INT0 and INT1

- **EIMSK** **External Interrupt Mask Register:** Determines whether the sources INT i IRQs trigger (bit INT i =1) or are deactivated (bit INT i =0) [1, S. 84]



- **EICRA** **External Interrupt Control Register A:** Determines for external interrupts INT0 and INT1 the *cause* for activation (edge/level controlled) [1, S. 83]



Thereby, two *Interrupt-Sense-Control* bits (ISC $i0$ and ISC $i1$) control the exact trigger (table for INT1, INT0 has analogous table):

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.



External Interrupts: Usage

- **Step 1:** Installation of `interrupt-service routine`
 - ISR in high level language \leadsto store and restore contents of registers
 - Support by `avrlibc`: Macro `ISR(SOURCE_vect)` (module `avr/interrupt.h`)

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

ISR(INT1_vect) { // invoked for every INT1 IRQ
    static uint8_t counter = 0;
    sb_7seg_showNumber(counter++);
    if (counter == 100) counter = 0;
}

void main(void) {
    ... // setup
}
```



■ **Step 2:** Configuration of interrupt control

- Initialize control registers as required
- Support by `avrlibc`: macros for bit indices (module `avr/interrupt.h` and `avr/io.h`)

```
...  
void main(void) {  
    DDRD  &= ~(1<<PD3);           // PD3: input with pull-up  
    PORTD |= (1<<PD3);  
    EICRA &= ~(1<<ISC10 | 1<<ISC11); // INT1: IRQ on level=low  
    EIMSK |= (1<<INT1);          // INT1: enable  
    ...  
    sei();                       // global IRQ enable  
    ...  
}
```

■ **Step 3:** Globally enable interrupts

- After finishing the device initialization
- Support by `avrlibc`: Instruction `sei()` (module `avr/interrupt.h`)



- **Step 4:** If there is nothing left to do, enter **power-saving mode**
 - The `sleep` instruction halts the CPU until an IRQ occurs
 - This state only has a comparably low power demand
 - Support by `avrlibc` (module `avr/sleep.h`):
 - `sleep_enable()` / `sleep_disable()`: sleep mode is activated/deactivated
 - `sleep_cpu()`: sleep mode entered



```
#include <avr/sleep.h>
...
void main(void) {
...
    sei(); // global IRQ enable
    while(1) {
        sleep_enable();
        sleep_cpu(); // wait for IRQ
        sleep_disable();
    }
}
```



- [1] *ATmega328PB 8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash*. [Atmel Corporation](#). Okt. 2015.

