HOCHSCHULE
RHEIN-WAAL
Rhine-Waal University
of Applied Sciences

LABORATORY

Microcontroller

4

EXPERIMENT:

Interrupts

1 Interrupts Basics

The simple way for a running programs is to execute it step after step. For some tasks a very fast reaction is required. To react on defined events the program need to ”look” for the event as often as possible. This can be a problem, it is inefficient and bad program code, see Figure 1.

To explain this, think about an example from the real world. A person is cleaning up his home and waiting for a guest. It is an annoying way to interrupt cleaning every minute and check if the guest is waiting in front of the door. The real world solution for this problem is the doorbell. The guest will use the doorbell, in the microcontroller word this is called interrupt.

A microcontroller interrupt can have several sources. In the real world this could be the doorbell, the phone bell or an alarm clock. Most microcontrollers have a sleep mode, it is used to save energy. Interrupts are often the only way to wake up a microcontroller.

Typical interrupt events:

- external interrupts (digital input)
- Timer / Counter compare register
- Timer / Counter overflow
- communication events (incoming data, sending complete, ...)
- watchdog

On most microcontrollers every event has his own event handler. This handler is called **I**nterrupt **S**ervice **R**outine (ISR). The interrupt hardware stops the normal program and starts the ISR. After the ISR is done the normal program will continue, see Figure 2.

To prevent troubleshooting between the normal program and the ISR, the ISR must be as short as possible. The remaining time for the normal program must be long enough.

The normal program can disable interrupts. This is necessary if:

- normal program and ISR share memory (prevent double access)
- a part of the normal program is time critical

The CPU can work with enabled or disabled interrupts. By default the program can not be interrupted. The program must enable the global interrupt switch with *sei();*

Inside an ISR other interrupt events are only stored and processed back to back. While an ISR is running the global interrupt switch is disabled until the ISR is done. This is done automatically.

It is possible to allow interrupts inside an ISR. This generates a low priority ISR. A program can become very complex with interruptable ISR, use it only if it is really necessary.

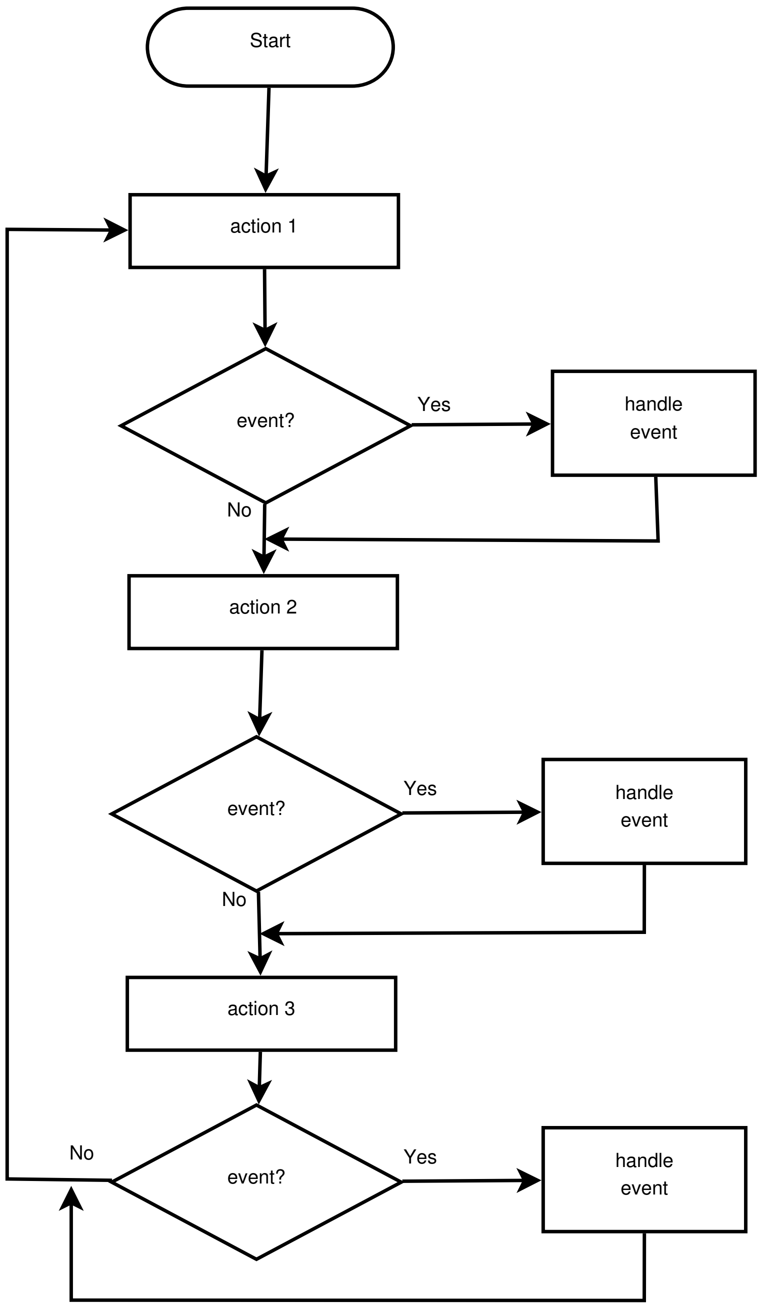


Figure 1: Without Interrupts

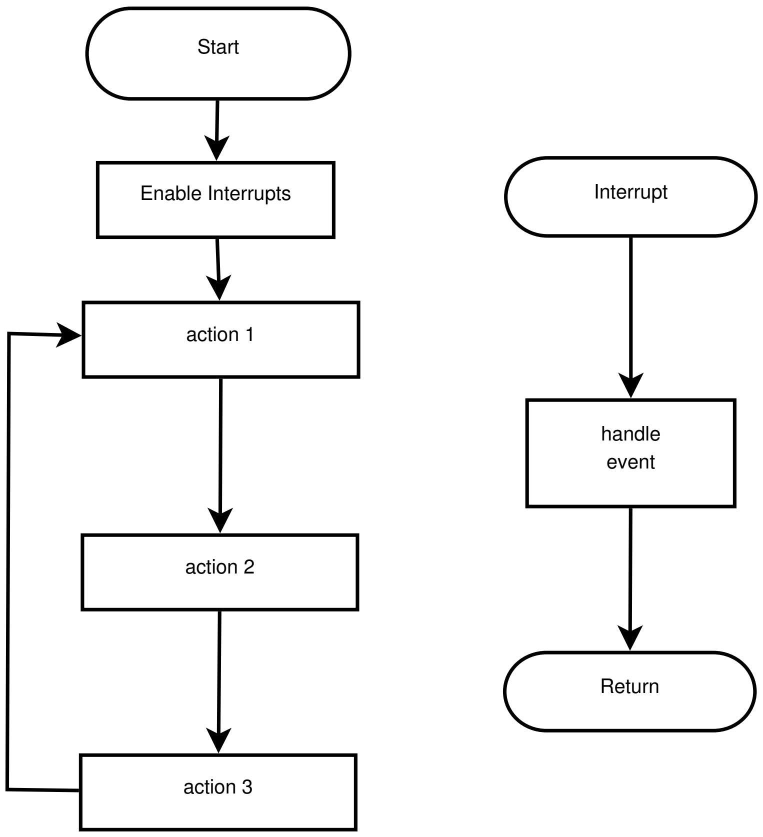
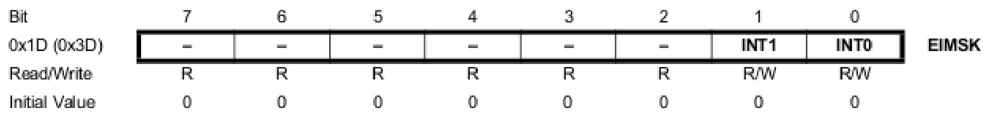


Figure 2: With Interrupts



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	

Figure 3: External Interrupt Mask Register

1.1 Interrupts on the Atmel ATmega88PA

The interrupt handling is controlled with registers. Inside the Status Register is the bit: **Global Interrupt Enable**. In C there is a program library with functions to set this bit: <avr/interrupt.h>

To enable interrupts use: void sei(void);
To disable interrupts use: void cli(void);

This is the general interrupt setting. The interrupt sources must be enabled separately! It is like a master switch and device switches.

Please note that the hardware executes interrupts without any security check. If an interrupt is enabled but not defined the microcontroller can crash (no hardware damage)!

The input pins PD2 and PD3 can be used for external inputs. To enable the external interrupts set the bits INT0 / INT1 in the EIMSK register, see Figure 3.

Most of the physical pins have multiple functions. On the MyAVR board the pins for external interrupts are used to communicate with the LCD display. To use this interrupts, please disconnect the LCD display and do not add any LCD functions to the source code.

1.2 ISR in C

In C every Interrupt is a simple function. Because the ISR is called by an event and not by another software function there is no return value. To distinguish between the different interrupt events, there is a event parameter. Look at the defines in the following list. The list is an excerpt and not complete.

```
//...
/* $Id : iom8.h, v1.132005/10/3022 : 11 : 23joerg_wunschExp$ */

/* avr/iom88p.h - definitions for ATmega88pa */
//...

/* Interrupt vectors */

/* External Interrupt Request 0 */
#define INT0_vect _VECTOR(1)
#define SIG_INTERRUPT0 _VECTOR(1)

/* External Interrupt Request 1 */
#define INT1_vect _VECTOR(2)
#define SIG_INTERRUPT1 _VECTOR(2)

/* Timer/Counter2 Compare Match */
#define TIMER2_COMP_vect _VECTOR(3)
#define SIG_OUTPUT_COMPARE2 _VECTOR(3)

/* Timer/Counter2 Overflow */
#define TIMER2_OVF_vect _VECTOR(4)
#define SIG_OVERFLOW2 _VECTOR(4)

/* Timer/Counter1 Capture Event */
#define TIMER1_CAPT_vect _VECTOR(5)
#define SIG_INPUT_CAPTURE1 _VECTOR(5)

/* Timer/Counter1 Compare Match A */
#define TIMER1_COMPA_vect _VECTOR(6)
#define SIG_OUTPUT_COMPARE1A _VECTOR(6)

/* Timer/Counter1 Compare Match B */
#define TIMER1_COMPB_vect _VECTOR(7)
```

```
#define SIG_OUTPUT_COMPARE1B _VECTOR(7)

//...
```

Examples for some ISR functions in C:

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

//...

void main()
{
//...
sei();
//...
}

ISR(INT0_vect)
{
/* Interrupt Code */
}

ISR(TIMER0_OVF_vect)
{
/* Interrupt Code */
}

ISR(USART_RXC_vect)
{
/* Interrupt Code */
}
```

Please note, the compiler will generate different ISR(parameter) functions. The parameter is not the same like a normal C function with a parameter, in this case it is a macro.

2 Using interrupts

Before flashing and running the programs remove the LCD display from the board and connect:

- Pin D2 to key 1
- Pin D3 to key 2

- Pin B0 to red LED
- Pin B1 to yellow LED
- Pin B2 to green LED
- Pin C0 to potentiometer 1

Use the template in "Template/Experiment4", the init function in this template is not complete.

2.1 Naive Projection

Task 1:
Write a program (without using interrupts) that:

- enables the red LED when button 1 is pressed
- disables the red LED when button 2 is pressed

If this program runs, expand the program:

- the yellow LED should blink with 0.5 sec delay (use _delay_ms(250) two times)
  - In order to use _delay_ms(250) it is necessary to add #define F_CPU 8000000UL and #include <util/delay.h> to the code.

Now you can see, that your program has a bad reaction on the buttons. This is because the device is hanging while the delay function is waiting.

2.2 Digital Input Interrupts

Task 2:
To solve the problem from 2.1 it is time to interrupt the CPU in the waiting time. Move the enabling and disabling from the main function to the interrupts INT0 and INT1.

The first step is to set the correct register values. To use INT0 and INT1 it is required to:

- set bits in register EICRA
- set bits in register EIMSK
- enable the global interrupt handling (already done here)
- write the two ISR functions

2.3 Timer Interrupts

Task 3:
Use Timer/Counter1 to flash the green LED with the overflow interrupt from timer one. For the prescaler use the setting 256.

The Timer/Counter parts of the Atmel Mega88PA can do more. So what if we want a flashing LED that has a short on-time and a long off-time (low duty cycle)? With the solution until now we can just realize a 50 percent duty cycle.

So let use the compare interrupts. This interrupts are called if the timer/counter reaches a defined value.

Task 4:
Use the interrupt for the compare register A and the overflow interrupt to:

- enable the green LED when the Timer/Counter 1 overflows
- disable the green LED when Timer/Counter 1 reaches the value 1000

2.4 Analog input recap

Task 5 (Optional):
Expand the main loop. Read the value from the potentiometer one (ADC0) and set the prescaler from Timer/Counter 1 to one of following values:

- 1
- 8
- 64
- 256

The potentiometer should work as a four step speed regulator for the flashing green LED.