RUHR-UNIVERSITÄT BOCHUM

RUB

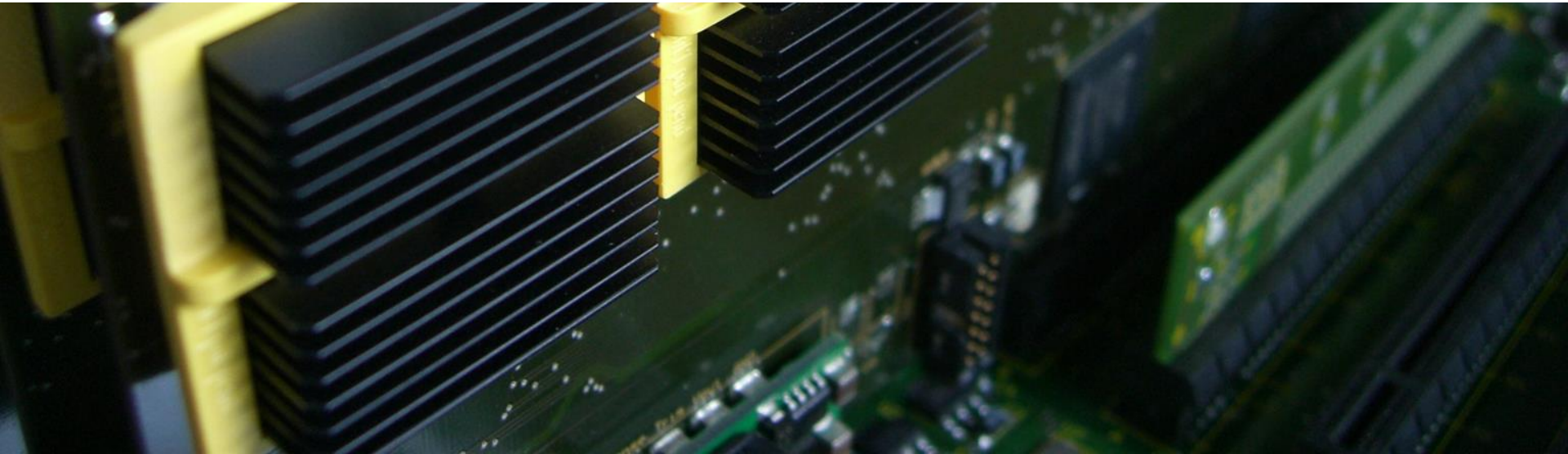
# Eingebettete Prozessoren

*SS 2014*

Übung 9: Interrupts

**Dipl.-Ing. Thomas Pöppelmann**
Arbeitsgruppe Sichere Hardware
Horst Görtz Institut für IT-Sicherheit

**26.06.2014**



# Agenda

1. Organisatorisches
2. Besprechung Übung 8
3. Interrupts

# Organisatorisches

- Seit Übung 8 Betreuung durch mich (Thomas Pöppelmann*)
- Fragen bitte nicht per Email sondern im Blackboard stellen
  - Kommilitonen haben vielleicht die selbe Frage
  - Antwort vermutlich schneller, da auch die Korrekteure, Ingo v. Maurich und Kommilitonen Zugriff haben
  - Ggf. anonym fragen
- Musterklausur inkl. Datenblatt im Blackboard (Kursunterlagen -> Musterklausur)
  - Keine Hilfsmittel in der Klausur gestattet (außer Taschenrechner)
  - Auszug aus dem Datenblatt wird gestellt

*) **http://www.sha.rub.de/group/staff/thomas_poeppelmann/**
thomas.poeppelmann@rub.de

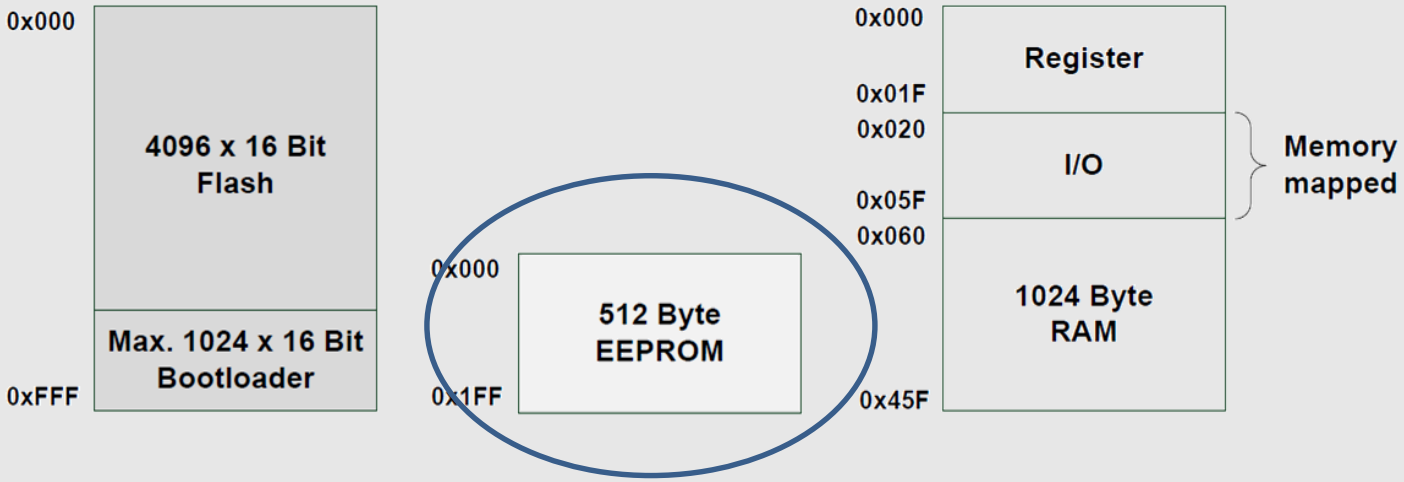
# Organisatorisches

- Infoveranstaltung „Studium in USA“
  - Freitag 10-12 Uhr ID 03/411
  - Poster im Blackboard
  - Purde Programm vermutlich eine der komfortabelsten Möglichkeiten an der RUB um in den USA zu studieren (Infrastruktur, Unterstützung, Kontakte)
  - In der Regel gute Chancen einen Platz zu erhalten
- Auslandsaufenthalte über andere Programme (Erasmus) sind auch möglich. Ggf. jetzt schon Gedanken machen und sich beraten lassen.

# Besprechung Übung 8

# Besprechung Übung 8

- Nicht-flüchtiger, Byte adressierbarer Speicher
- Separater Adressraum



# Besprechung Übung 8

- (**Aufgabe 1**) Warum gibt es im EEPROM Kontrollregister EECR zwei Steuerbits (EEMWE, EEWE) zum Schreiben, und nur eines zum Lesen (EERE)?
  - Der EEPROM hat nur eine begrenzte Lebensdauer, daher gibt es für Schreiboperationen eine Sicherheitsabfrage, die unbeabsichtigtes Beschreiben des EEPROM verhindern soll. Aus diesem Grund gibt es das Master Write Enable (EEMWE) Bit, welches ein Sicherheitsabfrage darstellt, bevor mit dem eigentlichen Schreibvorgang per Write Enable EEWE begonnen werden kann. Beim Lesen ist diese Vorsichtsmaßnahme nicht notwendig, so dass ein Signal "Read Enable" (EERE) hier genügt.
- (**Aufgabe** 2) Es gibt zwei unabhängige Steuerleitungen zum Lesen (EERE) und Schreiben (EEWE). Kann daher der EEPROM parallel gelesen und beschrieben werden?
  - Nein. Z.B. teilen sich beide Operationen das Datenregister EEDR von dem natürlich NICHT gleichzeitig gelesen und geschrieben werden kann.

# Besprechung Übung 8

- (**Aufgabe 3**) Erklären Sie, was folgender Assembler Code beim Programmieren des EEPROM bezweckt.

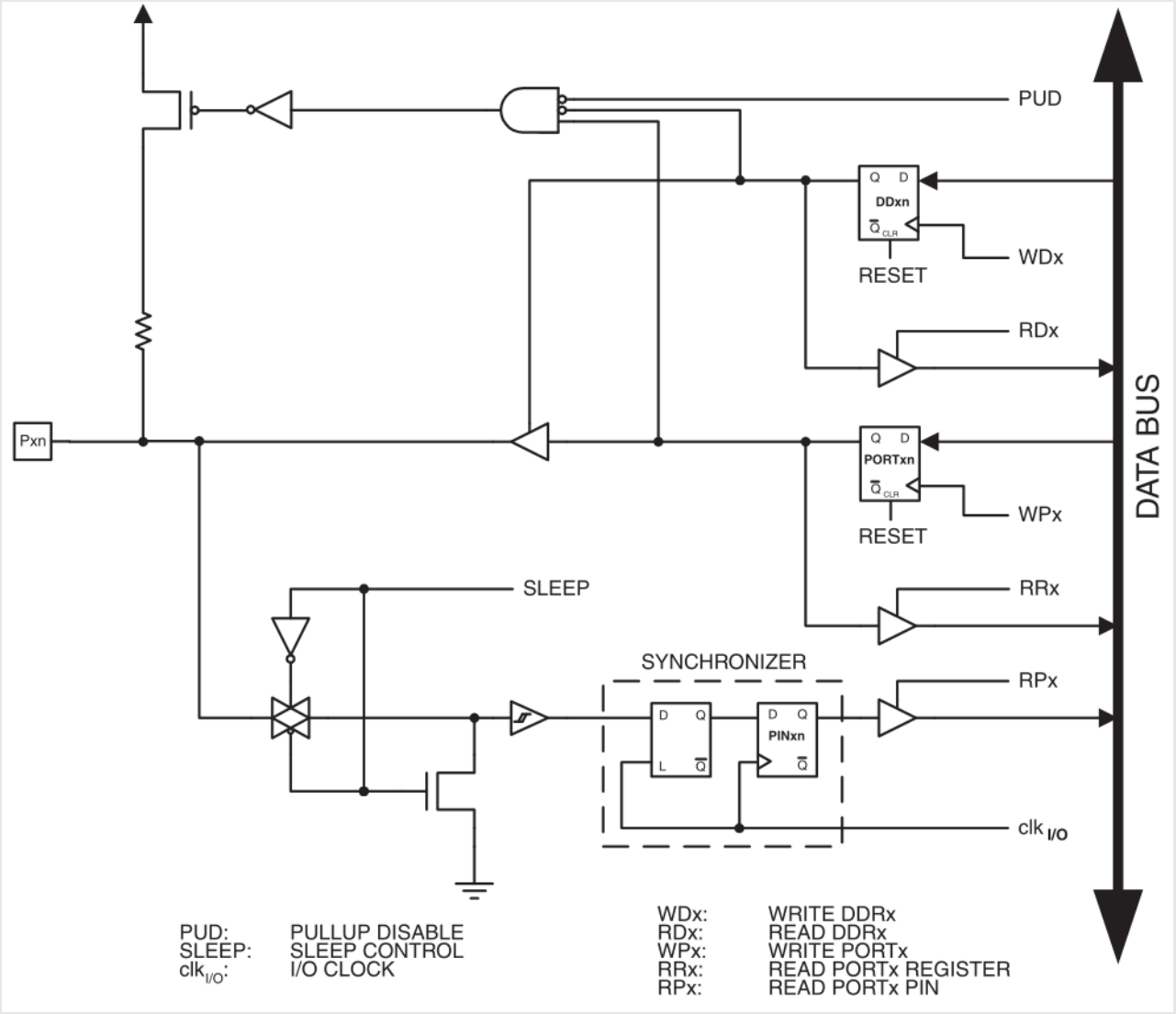
```
<...>
EEPROM_WAIT:
SBIC EECR, EEWE
RJMP EEPROM_WAIT
<...>
```

  - Antwort ohne Erklärung: Test ob der Schreibvorgang beendet ist.

# Besprechung Übung 8

- **(Aufgabe 4a)** Unterschied zwischen (1) PORTB mit OUT PORTB, R16 beschreiben und anschließend den Inhalt des Registers mit IN R16, PORTB zurücklesen und (2) IN R16, PINB verwenden
  - Vom PORTB, liest man nur den Inhalt des Ausgaberegisters zurück, welchen man ggfs. zuvor gesetzt hat
  - Liest man von PINB, tastet man die tatsächlichen externen Signalpins ab (Mit Verzögerung durch die Synchronisation)

# 1. PORTx vs. PINx



# Besprechung Übung 8

- **(Aufgabe 4b)** Sicheres setzen eines Bits im Register.
  - SBI und CBI Befehle verwenden
- **(Aufgabe 4c)** Spielt es bei der Konfiguration der Eingabe und Ausgabe sowie der Pull-Up Widerstände eine Rolle, ob man zuerst das DDR-Register beschreibt und dann das PORT-Register oder andersherum? Gibt es hier ggf. problematische Konfigurationen?
  - Das Standardverfahren: Zuerst Pull-Ups bzw. initiale Ausgabewerte, dann das DDR-Register zur Bestimmung der Ausgabe/Eingabefunktion.
    Sonderfall: (Wechsel von Port als Tristate zu Ausgabe mit Eins bzw. Wechsel von Eingabe mit Pullup zu Ausgabe Null). Kann aber grundsätzlich ebenfalls mit dem Standardverfahren (erst PORT dann DDR setzen) abgedeckt werden können.

# Besprechung Übung 8

- **(Aufgabe 6)** Diskobeleuchtung
  - Siehe *button_interrupt.asm* im BlackBoard

# (Externe) Interrupts

# Interrupts

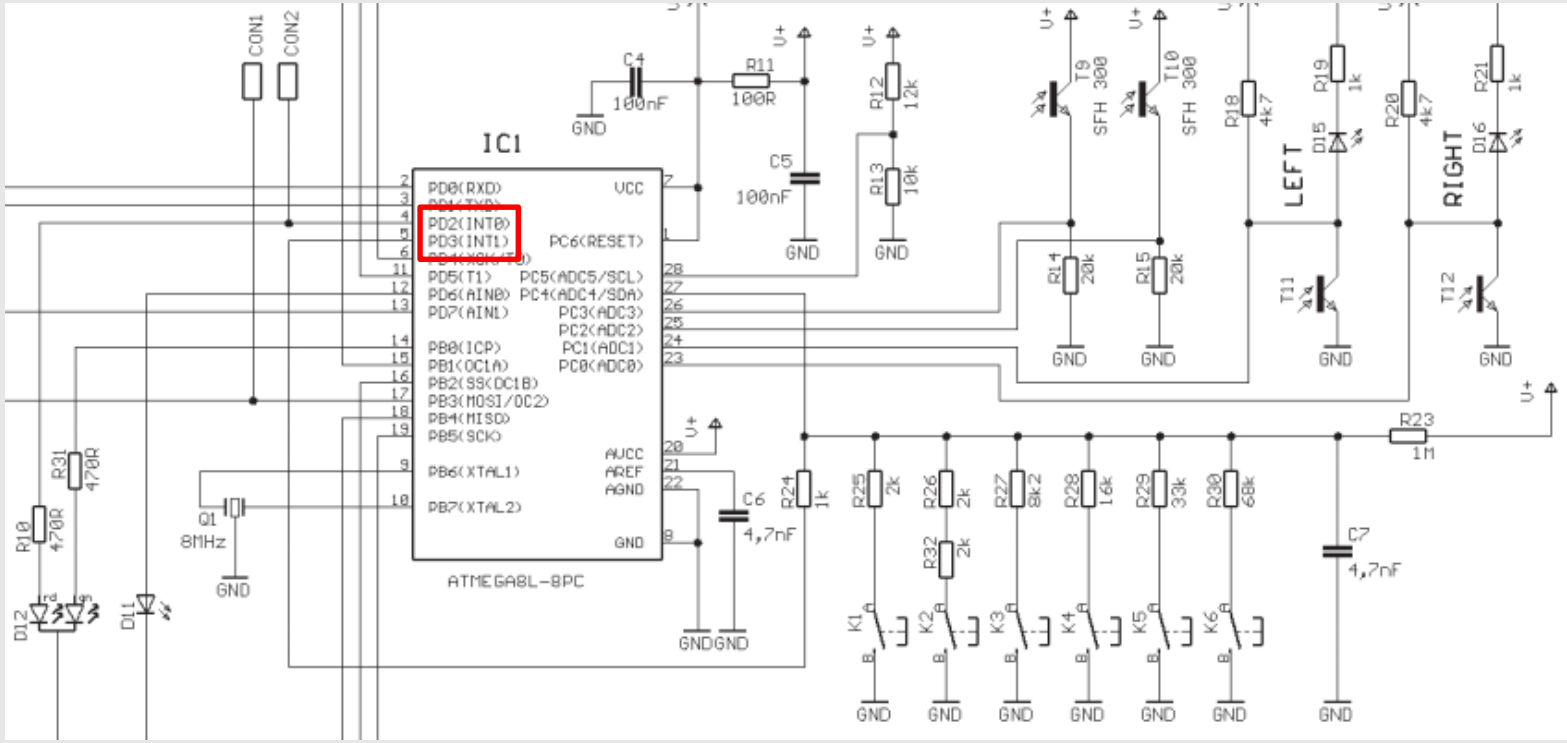
Wie laufen Interrupts ab?

- Unterbrechung des aktuell laufenden Programms
- Interrupttabelle enthält Einsprungsvektor in eine Interruptserviceroutine (ISR)
- ISR behandelt den Interrupt
  - Direkt und unabhängig vom Hauptprogramm
  - Setzt z.B. Flag auf welches das Hauptprogramm später reagiert
- Rücksprung in das Hauptprogramm
  - ISR darf Ablauf des Hauptprogramms nicht schädlich beeinflussen => SREG sichern

- Idee: Hauptprogramm kann „schlafen“ (Energie sparen) oder sich um wichtige Dinge kümmern und wird nur „aufgeweckt“ bzw. aktiv, wenn etwas passiert
- Informationen im Datenblatt unter „Interrupts“ und „External Interrupts“

# Interrupts

- Interrupts können z.B. von einem Timer, UART oder extern ausgelöst werden
- Heute behandeln wir externe Interrupts (INT0, INT1)



# Interrupts

- *INT0* und *INT1* müssen im General Interrupt Control Register *GICR* aktiviert werden
- Status der IRQs befindet sich im *GIFR*
- Konfiguration des auslösenden Ereignisses: Interrupts mit dem MCU (Mikrocontroller unit) Control Register *MCUCR*
- Damit Interrupt-Vektor ausgeführt wird muss zusätzlich das Global Interrupt Enable/Disable Flag im *SREG* gesetzt sein (SEI, CLI)

**Datenblatt**

# Interrupts

Event kann im *MCUCR* Register eingestellt werden:

- IRQ bei anliegendem Nullpegel (ISCx1:ISCx0=00)
- IRQ bei jedem Flankenwechsel (ISCx1:ISCx0=01)
- IRQ bei fallender Flanke (ISCx1:ISCx0=10)
- IRQ bei steigender Flanke (ISCx1:ISCx0=11)

# Interrupts

Interrupts verwenden – Einsprungvektor definieren

- Wenn ein Interrupt ausgelöst wird, wir der Befehl an der zum Interrupt gehörenden Adresse ausgeführt (Table 18, ATmega8 Datenblatt), der Program Counter gesichert und Interrupts deaktiviert
- Mit *.org* wird die Position von Programmcode im Code Segment definiert. Damit können Einsprungvektoren in die ISR definiert werden
- *.org 0x000* weißt den Assembler an, den nachfolgenden Code an die Adresse 0x000 im Programmspeicher zu legen

# Interrupts

**Table 18.** Reset and Interrupt Vectors

| Vector No. | Program Address[2] | Source | Interrupt Definition |
|---|---|---|---|
| 1 | 0x000[1] | RESET | External Pin, Power-on Reset, Brown-out Reset, and Watchdog Reset |
| 2 | 0x001 | INT0 | External Interrupt Request 0 |
| 3 | 0x002 | INT1 | External Interrupt Request 1 |
| 4 | 0x003 | TIMER2 COMP | Timer/Counter2 Compare Match |
| 5 | 0x004 | TIMER2 OVF | Timer/Counter2 Overflow |
| 6 | 0x005 | TIMER1 CAPT | Timer/Counter1 Capture Event |
| 7 | 0x006 | TIMER1 COMPA | Timer/Counter1 Compare Match A |
| 8 | 0x007 | TIMER1 COMPB | Timer/Counter1 Compare Match B |
| 9 | 0x008 | TIMER1 OVF | Timer/Counter1 Overflow |
| 10 | 0x009 | TIMER0 OVF | Timer/Counter0 Overflow |
| 11 | 0x00A | SPI, STC | Serial Transfer Complete |
| 12 | 0x00B | USART, RXC | USART, Rx Complete |
| 13 | 0x00C | USART, UDRE | USART Data Register Empty |
| 14 | 0x00D | USART, TXC | USART, Tx Complete |
| 15 | 0x00E | ADC | ADC Conversion Complete |
| 16 | 0x00F | EE_RDY | EEPROM Ready |
| 17 | 0x010 | ANA_COMP | Analog Comparator |
| 18 | 0x011 | TWI | Two-wire Serial Interface |
| 19 | 0x012 | SPM_RDY | Store Program Memory Ready |

# Interrupts

```
.include "m8def.inc"
.org 0x000
        rjmp main                   ; Reset Handler
.org 0x001
        rjmp handle_INT0            ; Externer Interrupt INT0
.org 0x002
        rjmp handle_INT1            ; Externer Interrupt INT1

main:                               ; Hauptprogramm
…
```

# Interrupts

- (Externe) Interrupts aktivieren
  1. Reaktion auf Flanke festlegen
  2. Speziellen Interrupt aktivieren (INT0 oder INT1)
  3. Interrupts allgemein aktivieren
- Initialisieren des Stacks nicht vergessen!

# Interrupts

```
...
;INT0: Jeder Flankenwechsel am Pin PD2 loest Interrupt INT0 aus
;INT1: Jeder Flankenwechsel am Pin PD3 loest Interrupt INT1 aus
in temp, MCUCR
sbr temp, (1<<ISC00) | (1<<ISC10)
cbr temp, (1<<ISC01) | (1<<ISC11)
out MCUCR, temp
;Aktivieren der externen Interrupts INT0/INT1
in temp, GICR
sbr temp, (1<<INT0) | (1<<INT1)
out GICR, temp
;Interrupts global aktivieren
sei
...
```

# Interrupts

- Interrupt behandeln
  - ISR behandelt den Interrupt
  - Möglichst kurz halten
  - SREG und in ISR verwendete Register sichern
    - Push/Pop
  - Rücksprung mit RET**I**

# Interrupts

```
handle_INT0:
push temp
in temp, SREG
inc count_int0          ; Bsp. Zaehle Anzahl der ISR Aufrufe
out SREG, temp
pop temp
reti

handle_INT1:
push temp
in temp, SREG
inc count_int1          ; Bsp. Zaehle Anzahl der ISR Aufrufe
out SREG, temp
pop temp
reti
```

Demo

- Beispiel im Debugger (nach der Übung im BlackBoard)
  - Zum Testen PIN (PD2 / PD3) im Debugger anklicken und dadurch dessen Zustand ändern

- Gruppenaufgabe
  - *button_interrupt.asm* Hauptprogramm
  - *led_busy_lsg.asm* Front-LED leuchtet bei Tastendruck (nicht nebenläufig)