	<b>Mikrocontroller</b>	Name: Rahm Datum: 13.02.2018 4_FA205_Befehlsliste_und_Zusatzbefehle.docx
	Befehlsliste technischen Richtlinie (Kurzform)	1.1

## delay.h

### Delay

Verzögert den Programmablauf für die angegebene Zeitdauer (Software)

```
delay_ms ( millisec );
delay_100us ( mikrosec );
```

#### Parameterliste:

```
uint8_t millisec: 0...65535 (x 1ms)
uint16_t mikrosec: 0...65535 (x 100µs)
```

## in\_out.h

### Bit Ein-/Ausgabe

Port-I/O

```
bit_init ( port, bitnr, direction );
bit_write ( port, bitnr, value );
value = bit_read ( port, bitnr );
```

### Byte Ein-/Ausgabe

Port-I/O

```
byte_init ( port, direction );
byte_write ( port, value );
value = byte_read ( port );
```

#### Parameterliste :

```
uint8_t port : PORTx, PORTy
AVR: _PORTB_, _PORTC_, ...
uint8_t bitnr : 0...7
uint8_t direction : IN = 0, OUT = 1
uint8_t value : Bit: 0,1
Byte: 0 ... 255
uint8_t status : Bit-Status: 0,1
```

### Pulsweitenmodulation

Digitaler Ausgang (PWM Out) zur Ausgabe eines pulswidenmodulierten Signals.

```
pwm_init ( ); // Initialwert für Tastgrad 50%
pwm_start ( );
pwm_stop ( );
pwm_duty_cycle ( value );
```

#### Parameterliste :

```
uint8_t value : 0 ... 255
(entspricht Tastgrad 0...100%)
```

### Analog-Digital Konverter

Zwei analoge Eingänge, je 8 Bit Auflösung.

```
adc_init ( );
value = adc_in1 ( );
value = adc_in2 ( );
```

#### Rückgabewert :

```
uint8_t value : 0 ... 255
```

## interrupt.h

### Externer Interrupt

Bei fallender Flanke am Interrupteingang wird ext\_interrupt\_isr() aufgerufen.

```
ext_interrupt_init ( ext_interrupt_isr );
ext_interrupt_enable ( );
ext_interrupt_disable ( );
ext_interrupt_isr ( );
```

### Timer

Interner Timer ruft alle 1ms die timer1ms\_isr() auf (Hardware-Timer).

```
timer1ms_init ( timer1ms_isr );
timer1ms_enable ( );
timer1ms_disable ( );
timer1ms_isr ( );
```

## lcd.h

### LC-Display

Routinen zur Ansteuerung eines Textdisplays.

```
lcd_init ( );
lcd_clear ( );
lcd_setcursor ( row, column );
lcd_print ( text[ ] ); // Bsp.: "Hallo Welt"
lcd_char ( ascii ); // 'A', 'B', ':', '?', ....
lcd_byte ( byte ); // 0 ... 255
lcd_int ( word ); // 0 ... 65535
```

#### Parameterliste :

```
uint8_t row: zeile = 1, 2, ...
uint8_t column: spalte = 1, 2, ...
uint8_t text [ ]: Zeichenkette ('\0'-terminiert)
uint8_t ascii: Zeichen (ASCII-Code)
uint8_t byte: 8 Bit (3 stelliger Dezimalwert)
uint16_t word: 16 Bit (5 stelliger Dezimalwert)
```

## communication.h

### RS232

Serielle Schnittstellenroutinen zur Kommunikation.

```
rs232_init ( ); // 9600 Baud, 1 Startbit, 8 Datenbit, 1 Stopbit
value = rs232_get ( ); // Liefert \0, wenn kein Zeichen empfangen
rs232_put ( value ); // Ausgabe eines Bytes
rs232_print ( text[ ] ); // Ausgabe eines Strings
```

#### Parameterliste :

```
uint8_t value: Zeichen (8 Bit)
uint8_t text [ ]: Zeichenkette ('\0'-terminiert)
```


### I2C-Bussystem

Schnittstellenroutinen zur Kommunikation

```
mit I2C-Bus Komponenten.
i2c_init ( );
i2c_start ( );
i2c_stop ( );
ack = i2c_write ( value );
value = i2c_read ( ack );
```

#### Parameterliste :

```
uint8_t ack : ACK = 0, NACK = 1
uint8_t value: Byte (8 Bit)
```

 Friedrich-Ebert-Schule Esslingen FES	<b>Mikrocontroller</b>	Name: Rahm Datum: 13.02.2018 4_FA205_Befehlsliste_und_Zusatzbefehle.docx
	Befehlsliste technischen Richtlinie (Kurzform)	1.2

## Erweiterte Funktionen

### 1. delay.h

```
void delay_us ( uint16_t mikrosekunden);
void delay_s ( uint16_t sekunden);
```

### 2. In\_out.h

```
void bit_toggle ( uint8_t *port, uint8_t bitnr, uint8_t *status );

void pwm2_init ( void );           // pwm2-Ausgang an Port PD3
void pwm2_start ( void );
void pwm2_stop ( void );
void pwm2_duty_cycle ( uint8_t value);

void pwm3_init ( void );           // pwm3-Ausgang an Port PB2
void pwm3_start ( void );
void pwm3_stop ( void );
void pwm3_duty_cycle ( uint8_t value);

uint8_t adc_in ( uint8_t channel ); // channel: ch1=pc3, ch2=pc0, ch3, ch4, ch5
```

### 3. interrupt.h

Erweiterte Triggermöglichkeiten für die externen Interrupts werden über das Auskommentieren des entsprechenden Schalters aktiviert.


```
// externer Interrupt Trigger
// #define _RISING_EDGE_TRIGGER_
#define _FALLING_EDGE_TRIGGER_
// #define _ANY_EDGE_TRIGGER_

// Funktionsprototypen für externen Interrupt1 (Interrupteingang Int1: Port PD3)
void ext_interrupt1_init ( void (*ip) (void));
void ext_interrupt1_enable ( void );
void ext_interrupt1_disable ( void );
void ext_interrupt1_isr ( void );

// Funktionsprototypen für den seriellen Empfangs-Interrupt (data received)
void serial_interrupt_init ( void (*sr) (void));
void serial_interrupt_enable ( void );
void serial_interrupt_disable ( void );
void serial_interrupt_isr ( void );

// Erweiterter Timer mit variabler Zeit: millisekunden = (16.2 ... 0.0001) ms
// Die Funktionen werden für timer1ms benötigt. Sind aber nur beim
// ATmega328 verfügbar. Die Genauigkeit hängt vom gewählten Zeitwert ab!
void timer_ms_init ( void (*ti) (void), float millisec );
void timer_ms_enable ( void );
void timer_ms_disable ( void );
void timer_ms_isr ( void );

// Routinen zur Soundausgabe an Port B.3 (Verwendet Timer 0 = timer_ms_)
// key:      Midi-Tastennummern (49 = a' = 440Hz Kammerton)
//          1 = „A = 27,5Hz (real sind ca. 32Hz realisierbar!)
//          bis 88 = a'''''' = 4186,01Hz
// duration: Tondauer in ms
// silence:   Ruhe nach Ton in ms
// Änderung des Soundports hier möglich:
#define TON_PORT _PORTB_
#define TON_BIT 3
void sound_init (void);
void play_note (uint8_t key, uint16_t duration, uint16_t silence);
// frequenz: Tonfrequenz in Hz
void note_on (float frequenz);
void note_off (void);
```

 Friedrich-Ebert-Schule Esslingen FES	<b>Mikrocontroller</b>	Name: Rahm Datum: 13.02.2018 4_FA205_Befehlsliste_und_Zusatzbefehle.d docx
	Befehlsliste technischen Richtlinie (Kurzform)	1.3

#### 4. lcd.h

```
// Definition von 7 eigenen Zeichen. char_nr: 1 bis 7
// char_nr = 0 funktioniert nur, wenn das Zeichen nicht in Zeichenketten
// ('\0' = 0) verwendet wird.
void lcd_defchar ( uint8_t *pixtab, uint8_t char_nr );
// löscht die angegebene Zeile (lineNr = 1...4)
void lcd_clearline ( uint8_t lineNr );
// Ausgabe von 00..99 z.B. für Datum, Uhrzeit
void lcd_dd (uint8_t val);
// Gibt ein Byte als Bitfolge am LC-Display aus. Z.B.: 01101101
void lcd_debug (uint8_t byte);
```

#### 5. communication.h

```
// Ändert die Baudrate (4800,9600,14400,19200,38400,57600,115200,128000)
void rs232_baud ( uint32_t baud );
// Erzeugt ein Eingabeprompt für eine Zahl von 00 ... 99
uint8_t rs232_inputdd ( void );
// Ausgabe einer Dezimalzahl 00..99 auf RS232
void rs232_printdd ( uint8_t value );
// Ausgabe einer Dezimalzahl 0...65535 auf RS232
void rs232_int ( uint16_t value );
```

### Zusatzbibliotheken

#### 1. dht11.h

```
// Einstellung des Sensor-Out-Anschluss des DHT11
#define DHT11_PORT _PORTD_
#define DHT11_BIT 2
// Liest den dht11 Feuchte und Temperatursensor
// data[0] = Feuchte (20 .. 95%)
// data[1] = Feuchte dezimal (0 bei DHT11)
// data[2] = Temperatur (0..50°C)
// data[3] = Temperatur dezimal (0 bei DHT11)
// data[4] = Checksumme (data[4] = data[0]+data[1]+data[2]+data[3])
// Rückgabewert: 0 = error
uint8_t dht11_read(uint8_t data[]);
```

#### 2. srf04.h


```
void srf04_start ( void );
uint32_t srf04_stop ( void );
```

#### 3. nunchuk.h

```
//Globale Variablen zur Übergabe der Nunchuk Sensordaten
uint16_t X_accel, Y_accel, Z_accel;
uint8_t X_joy, Y_joy;
uint8_t Z_button, C_button;
//Nunchuk Funktionsprototypen
void nunchuk_init ( void );
void nunchuk_read ( void ); //Auslesen des Nunchuk. Schreiben der glob. Var.
void nunchuk_debug ( void ); //Ausgabe der Sensorwerte auf LC-Display
```

#### 4. rtc.h (für DS1307)

```
// Funktionsprototypen für Echtzeituhr
void rtc_init (void); // Initialisierung
// Schreiben der Timekeeperregister (BCD-Format)
// reg = _SEC_, _MIN_, _HR_, _DAY_, _DATE_, _MONTH_, _YEAR_
// value = 00...59 , 59, 23, 31, 7, 12, 99
void rtc_write (uint8_t reg, uint8_t value);
// Lesen der Timekeeper-Register (BCD-Format)
uint8_t rtc_read (uint8_t reg);
// Byteweises Schreiben ins interne RAM (Binär-Format).
// Z.B. Speichern von batteriegepufferten Werten.
// reg = 0x08...0x3F, (0x00..0x07 = Timekeeper-Register!!!)
// value = 0x00...0xff
void rtc_lowlevel_write (uint8_t reg, uint8_t value);
//Byteweises Lesen aus dem internen RAM
```

 Friedrich-Ebert-Schule Esslingen FES	<b>Mikrocontroller</b>	Name: Rahm Datum: 13.02.2018 4_FA205_Befehlsliste_und_Zusatzbefehle.d docx
	Befehlsliste technischen Richtlinie (Kurzform)	1.4

```
uint8_t rtc_lowlevel_read (uint8_t reg);
```

## 5. eeprom.h (für 24LCxx)

```
#define EEPROM_END_ADDRESS 0x7fff // 32kByte 24LC256
//**** EEPROM-Funktionsprototypen
void eeprom_init (void); // Initialisierung
// Schreibt das Byte "value" an die angegebene Speicheradresse
// address = 0x0000...EEPROM_END_ADDRESS
// im angegebenen EEPROM: i2c_address = EEPROM_1, EEPROM_2, EEPROM_3
void eeprom_write (uint8_t i2c_address, uint16_t address, uint8_t value);
// dito. ein Byte lesen
uint8_t eeprom_read (uint8_t i2c_address, uint16_t address);
// führt einen Speichertest des angegebenen EEPROM durch.
// getestet wird Adresse 0x0000 bis EEPROM_END_ADDRESS
// Rückgabewert: -1 = Fehler, 0 = OK
int8_t eeprom_memtest (uint8_t i2c_address);
```

## 6. datalogger.h (Projektspezifische Funktionen für Datenlogger-Platine!!)

```
// Globale Variablen als Übergabe-Parameter für das Datenlogger-Programm (main)
volatile uint8_t jahr, monat, tag, stunde, minute, sekunde;
volatile int16_t temperatur;
volatile uint16_t aktueller_Datensatz; // -> im gewählten EEPROM
#define _REC_SIZE_ 8 // Bytes je Datensatz
#define RECORD_MSB 0x08 // Speicheradressen für aktuellen Datensatz
#define RECORD_LSB 0x09 // wird im RTC-Ram batteriegepuffert
#define SAMPLETIME_MSB 0x0A // Speicheradressen für Sample-Time
#define SAMPLETIME_LSB 0x0B // wird im RTC-Ram batteriegepuffert
#define MAX_RECORD 4095 // Maximale Anzahl an Datensätzen je EEPROM
// 24LC256: 32kByte / _REC_SIZE_ - 1 = 4095

//**** Datenlogger-Funktionen
void eeprom_speichertest (void);
// Eingabe der aktuellen rtc-Zeit über ein Eingabeprompt
// Wartet nach Aufruf 5s auf Eingabe.
void rs232_set_time (void);
// Speichert oder holt den aktuellen Datensatz aus dem EEPROM
// Parameter: i2c_address = EEPROM_1, EEPROM_2, EEPROM_3
// record = Datensatznummer (= Speicheradresse im EEPROM)
void eeprom_get_record (uint8_t i2c_address, uint16_t record);
void eeprom_set_record (uint8_t i2c_address, uint16_t record);
// Gibt den angegebenen Datensatz über die serielle Schnittstelle aus
// Format: DD.MM.YY hh:mm:ss;(-)xx,y
// Tag.Monat.Jahr Stunde:Minute:Sekunde;Temperatur,Dezimale(0,5)
// Bsp: 02.11.17 20:47:34;21,5
void rs232_print_record (uint8_t i2c_address, uint16_t record);
// Speichert oder holt die aktuelle Uhrzeit im Uhrenbaustein
// Liest/speichert in den globalen Variablen: jahr, monat, tag, stunde, minute, sekunde
void rtc_set (void);
void rtc_get (void);
// Schreibt oder liest die aktuelle Datensatznummer spannungsausfallsicher
// im Uhren-RAM (Adresse: RECORD_MSB, RECORD_LSB)
// record_number: 0x0000 ... EEPROM_END_ADDRESS (0x7fff)
uint16_t read_current_recordnumber_from_rtc (void);
void write_current_recordnumber_to_rtc (uint16_t record_number);
uint16_t read_current_sampletime_from_rtc (void);
void write_current_sampletime_to_rtc (uint16_t sample_time);
// Gibt alle gespeicherten Datensätze über RS232 aus
void serial_print_all_records (void);
// Liest den aktuellen Temperaturwert vom LM75-Temperatursensor
int16_t lm75_read (void);
// Zeigt den Temperaturwert auf dem LC-Display an. (z.B.: 24,5°C)
void lcd_print_temperatur (int16_t degree);
// Gibt den Temperaturwert über RS232 aus.
// mode = 0: Mit Kurvenname T1, mit \n (z.B.: T1=23,5)
// mode = 1: ohne Name, mit \r (z.B.: 23,5)
// mode = 2: ohne Name, mit °C mit \r (z.B.: 23,5°C)
void rs232_print_temperatur (int16_t degree, uint8_t mode);
uint32_t rs232_get_sampletime (void);
```