

1 ATmega32u4

1.1 Register beschreiben

Ein Bit in einem Register kann entweder auf 0 oder 1 gesetzt werden.

Um es auf 0 zu setzen, muss es mit 0 ge-UND-et werden und wird mit dem Zeichen & dargestellt.

Der Code, um das **4. Bit** (es wird bei 0 angefangen zu zählen), auf 0 zu setzen, sieht folgendermaßen aus:

```
1 // REGISTER = REGISTER &~ (0 << POSITION IM REGISTER)
2 DDRD = DDRD &~ (1 << 3);
```

Die Tilde (~) ist hier ein Negator, d.h. es ist **nicht** 1, also 0; die Pfeile sind Shiebeoperatoren, um das korrekte Bit anzusprechen.

Ähnlich ist es beim Setzen eines Bits auf 1: Hier wird mit 1 ge-ODER-et, was mit dem Zeichen | gezeigt wird:

```
1 // REGISTER = REGISTER | (1 << POSITION IM REGISTER)
2 DDRD = DDRD | (1 << 3);
```

Es können jeweils **mehrere** Bits eines Registers in einer Zeile auf 1 **oder** 0 gesetzt werden. Allerdings darf in einer Zeile ein Bit nicht auf 0, während ein anderes auf 1 gesetzt werden.

- Erlaubt:

```
DDRD = DDRD &~ (1 << 3) &~ (1 << 3) &~ (1 << 3);
```

- Nicht erlaubt:

```
DDRD = DDRD | (1 << 3) | (1 << 3) &~ (1 << 3);
```

1.2 Takt

Der Takt des ATmega32u4 kann per Software verringert werden und wird über das CLKPR-Register getan. Bevor dieses beschrieben werden kann, muss 0x80 in das Register geschrieben werden.

6.11.4 CLKPR – Clock Prescaler Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---------------|--------|---|---|---|---------------------|--------|--------|--------|-------|
| | CLKPCE | – | – | – | CLKPS3 | CLKPS2 | CLKPS1 | CLKPS0 | CLKPR |
| Read/Write | R/W | R | R | R | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | See Bit Description | | | | |

Table 6-10. Clock Prescaler Select

| CLKPS3 | CLKPS2 | CLKPS1 | CLKPS0 | Clock Division Factor |
|--------|--------|--------|--------|-----------------------|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 2 |
| 0 | 0 | 1 | 0 | 4 |
| 0 | 0 | 1 | 1 | 8 |
| 0 | 1 | 0 | 0 | 16 |
| 0 | 1 | 0 | 1 | 32 |
| 0 | 1 | 1 | 0 | 64 |
| 0 | 1 | 1 | 1 | 128 |
| 1 | 0 | 0 | 0 | 256 |
| 1 | 0 | 0 | 1 | Reserved |
| 1 | 0 | 1 | 0 | Reserved |
| 1 | 0 | 1 | 1 | Reserved |
| 1 | 1 | 0 | 0 | Reserved |
| 1 | 1 | 0 | 1 | Reserved |
| 1 | 1 | 1 | 0 | Reserved |
| 1 | 1 | 1 | 1 | Reserved |

Beispiel

Der externe Takt hat 16MHz und soll auf 8MHz heruntergesetzt werden.

```
1 CLKPR = 0x80;  
2 CLKPR = 0x01;
```

1.3 GPIO

Die **General-Purpose-Input-Output**-Pins (GPIO-Pins) können folgenden Status haben:

Table 10-1. Port Pin Configurations

| DDxn | PORTxn | PUD (in MCUCR) | I/O | Pull-up | Comment |
|------|--------|-------------------|--------|---------|--|
| 0 | 0 | X | Input | No | Tri-state (Hi-Z) |
| 0 | 1 | 0 | Input | Yes | Pxn will source current if ext. pulled low |
| 0 | 1 | 1 | Input | No | Tri-state (Hi-Z) |
| 1 | 0 | X | Output | No | Output Low (Sink) |
| 1 | 1 | X | Output | No | Output High (Source) |

Beispiel

Pin-D7 auf HIGH setzen.

```
1 DDRD = DDRD | (1 << DDD7);  
2 PORTD = PORTD | (1 << PORTD7);
```

Wichtig: Bei der Verwendung von Hardwareeinheiten (Timer, UART, etc.) muss GPIO immer zuerst auf Input bzw. Output eingestellt werden.

1.4 ADC

- Single-Ended:
Spannung von ADC-Pin zu GND wird gemessen.
- Differenziell:
Spannung zwischen zwei ADC-Pins wird gemessen. (Siehe Kapitel ??)
- Referenzspannung:
Es gibt drei verschiedene Spannungsreferenzen:
 - Interne 2,56V Referenz
 - Externer `AREF`-Pin
 - Externer `AVCC`-Pin
- Auto-Trigger Mode
Es wird periodisch gemessen, wofür die Taktquelle eingestellt werden muss. (Siehe Kapitel 1.4.2)

1.4.1 Differenziell

Wenn differenziell gemessen wird, ist das Ergebnis im Zweierkomplement dargestellt - ein Zahlensystem um negative Zahlen (in binär) darzustellen.

Beispiel

" -2_d " im Zweierkomplement

1. Zunächst wird der Binärwert des Betrags der Zahl invertiert: $2_d = 0010_b \Rightarrow 1101_b$
2. Danach wird zu diesem Wert 1_b addiert: $1101_b + 1_b = 1110_b = -2_d$

1.4.2 Auto-Trigger Mode

Die Taktquelle wird folgendermaßen eingestellt:

| | | | | | | | | | |
|---------------|-------|------|------|---|-------|-------|-------|-------|--------|
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| | ADHSM | ACME | MUX5 | – | ADTS3 | ADTS2 | ADTS1 | ADTS0 | ADCSRB |
| Read/Write | R/W | R/W | R | R | R | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

Table 24-6. ADC Auto Trigger Source Selections

| ADTS3 | ADTS2 | ADTS1 | ADTS0 | Trigger Source |
|-------|-------|-------|-------|--------------------------------|
| 0 | 0 | 0 | 0 | Free Running mode |
| 0 | 0 | 0 | 1 | Analog Comparator |
| 0 | 0 | 1 | 0 | External Interrupt Request 0 |
| 0 | 0 | 1 | 1 | Timer/Counter0 Compare Match A |
| 0 | 1 | 0 | 0 | Timer/Counter0 Overflow |
| 0 | 1 | 0 | 1 | Timer/Counter1 Compare Match B |
| 0 | 1 | 1 | 0 | Timer/Counter1 Overflow |
| 0 | 1 | 1 | 1 | Timer/Counter1 Capture Event |
| 1 | 0 | 0 | 0 | Timer/Counter4 Overflow |
| 1 | 0 | 0 | 1 | Timer/Counter4 Compare Match A |
| 1 | 0 | 1 | 0 | Timer/Counter4 Compare Match B |
| 1 | 0 | 1 | 1 | Timer/Counter4 Compare Match D |

Ergebnis

Das Messergebnis des ADC befindet sich in zwei Registern: ADCL (ADC-Low) und ADCH (ADC-High).

Der Messwert kann in zwei Arten dargestellt werden:

1. Linksbündig:

ADLAR = 1

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
|---------------|------|------|------|------|------|------|------|------|------|
| | ADC9 | ADC8 | ADC7 | ADC6 | ADC5 | ADC4 | ADC3 | ADC2 | ADCH |
| | ADC1 | ADC0 | - | - | - | - | - | - | ADCL |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| Read/Write | R | R | R | R | R | R | R | R | |
| | R | R | R | R | R | R | R | R | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

2. Rechtsbündig:

ADLAR = 0

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
|---------------|------|------|------|------|------|------|------|------|------|
| | - | - | - | - | - | - | ADC9 | ADC8 | ADCH |
| | ADC7 | ADC6 | ADC5 | ADC4 | ADC3 | ADC2 | ADC1 | ADC0 | ADCL |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| Read/Write | R | R | R | R | R | R | R | R | |
| | R | R | R | R | R | R | R | R | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

ADCL muss immer vor ADCH ausgelesen werden; folgende Beispiele verwenden Linksbündigkeit.

1.4.3 Messdauer berechnen

Die ADC Messdauer muss eingestellt werden: kurze Messdauern führen zu ungenaueren Ergebnissen, bei zu langen kann die Dauer zwischen Abtastpunkten zu groß werden. Generell sollte die Messfrequenz des ATmega32u4 zwischen $50kHz$ und $200kHz$ sein (wenn die Messgeschwindigkeit realisierbar ist.)

ADC Control and Status Register A – ADCSRA

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---------------|-------------|-------------|--------------|-------------|-------------|--------------|--------------|--------------|---------------|
| | ADEN | ADSC | ADATE | ADIF | ADIE | ADPS2 | ADPS1 | ADPS0 | ADCSRA |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

Table 24-5. ADC Prescaler Selections

| ADPS2 | ADPS1 | ADPS0 | Division Factor |
|-------|-------|-------|-----------------|
| 0 | 0 | 0 | 2 |
| 0 | 0 | 1 | 2 |
| 0 | 1 | 0 | 4 |
| 0 | 1 | 1 | 8 |
| 1 | 0 | 0 | 16 |
| 1 | 0 | 1 | 32 |
| 1 | 1 | 0 | 64 |
| 1 | 1 | 1 | 128 |

Der Wert des obigen Diagramms muss in die gemessene Spannung umgerechnet werden.

- Single-Ended:

$$V_{IN} = \frac{ADC \cdot V_{REF}}{1023} \quad (1.1)$$

- Differenziell:

$$V_{POS} - V_{NEG} = \frac{ADC \cdot V_{REF}}{GAIN \cdot 512} \quad (1.2)$$

Single-Ended

Der entsprechende Code um die gemessene Spannung zurückzubekommen:

```
1 // SETUP:
2 DDRF = DDRF &~ (1 << DDF0); // PF0-Input
3 ADMUX = ADMUX | (1 << ADLAR) | (1 << REFS0); // Left adjust ADC
   result; Voltage reference
4
5 // Uncomment for Auto trigger mode;
6 // ADCSRA = ADCSRA | (1 << ADSC);
7 // ADCSRB = ADCSRB | (1 << ADTS1) | (1 << ADTS0); // Auto
   trigger mode Taktquelle (Timer0)
8
9 // Enable Interrupt
10 // Enable ADC
11 // Prescaler = 64: ADC_f = 8M/64
12 ADCSRA = ADCSRA | (1 << ADEN) | (1 << ADSC) | (1 << ADPS2) | (1
   << ADPS1);
13 DIDR0 = DIDR0 | (1 << ADC0D); // Disable digital function of PF0
14
15 // READ:
16 adcRead() {
17     uint16_t adc_value;
18     float out;
19     unsigned char adcl, adch;
20
21     // Start conversation
22     // Put in comment in auto trigger mode
23     ADCSRA = ADCSRA | (1 << ADSC);
24
25     // Wait for ADC to finish
26     while(ADCSRA & (1 << ADSC)) {}
27
28     adcl = ADCL;
29     adch = ADCH;
30     adc_value = (adcl >> 6) + (adch << 2);
```



```

31     out = (float)((adc_value * 5) / 1023.0);
32     return out;
33 }

```

No re-
turn
type?

Differenziell

Auch hier der Code, um die Spannung returniert zu bekommen:

```

1  // SETUP:
2  DDRF = DDRF &~ (1 << DDF0) &~ (1 << DDF1); // PF0-Input
3  ADMUX = ADMUX | (1 << ADLAR) | (1 << REFS0); // Left adjust ADC
    result; Voltage reference
4  ADMUX = ADMUX | (1 << MUX4); // PINS(P:ADC0; N: ADC1); GAIN: 1
5
6  // Uncomment for auto trigger mode
7  // ADCSRA = ADCSRA | (1 << ADSC);
8  // ADCSRB = ADCSRB | (1 << ADTS1) | (1 << ADTS0); // Auto
    trigger mode Taktquelle (Timer0)
9
10 // Enable interrupt enable
11 // Enable ADC
12 // Prescaler = 64: ADC_f = 8M/64
13 ADCSRA = ADCSRA | (1 << ADEN) | (1 << ADSC) | (1 << ADPS2) | (1
    << ADPS1);
14
15 // READ:
16 adcRead() {
17     uint16_t adc_value;
18     float out;
19     unsigned char adcl, adch;
20
21     // Start conversation
22     // Put in comment in auto trigger mode
23     ADCSRA = ADCSRA | (1 << ADSC);
24
25     // Wait for ADC to finish
26     while(ADCSRA & (1 << ADSC)) {}

```

```

27     adcl = ADCL;
28     adch = ADCH;
29     adc_value = (adcl >> 6) + (adch << 2);
30     out = (float)((adc_value * 5) / (1.0 * 1023.0));
31     return out;
32 }

```

1.5 Sleep Mode

Idle Mode

Der CPU- und Flash-Clock wird gestoppt, alle anderen Clocks laufen weiter. Jedoch funktionieren Peripherien, wie: Timer, USB, SPI, USART, ADC, Analog-Komperator, I2C Watchdog, Interrupts.

Der Mikrocontroller kann durch interne und externe Interrupts - z.B. Timer-Overflow, USART, Transmition-Complete, etc. - aufgeweckt werden.

ADC Noise Reduction Mode

Dieser Modus ist dafür da, um die Genauigkeit der ADC-Messungen zu erhöhen. Wenn der ADC aktiviert ist, und dieser Noise Reduction Mode ebenso, startet automatisch eine ADC-Messung.

Alles bis auf ADC, externe Interrupts, I2C-Address-Matching und den Watchdog-Timer wird abgeschaltet.

Der Mikrocontroller kann durch die Vollendung der ADC-Messung, Reset, Watchdog-Timer, Brownout-Reset, I2C-Interrupt, SPM/EEPROM-Interrupt und externe Interrupts - an den Pins `INT3:0`, `INT6` - oder Pin-Change-Interrupts aufgeweckt werden.

Power-Down/-Save Mode

Der externe Clock wird deaktiviert, wodurch asynchrone Peripherien - externe Interrupts, I2C-Interrupt, Watchdog - weiterarbeiten.

Der Controller kann durch Reset, Watchdog, Brownout-Reset, I2C-Address-Match und externe Interrupts (an den Pins `INT3 : 0`, `INT6`) Pin-Change-Interrupts aufgeweckt werden.

Merke, dass dieser Modus mehr Zeit benötigt, um wieder aufzuwachen.

(Extended) Standby Mode

Dieser Modus ist im Endeffekt gleich wie der Power-Down Mode, nur hier ist der verwendete Oszillator nicht gestoppt wird; dadurch erwacht der Mikrocontroller schneller.

Register

Um die Sleep-Modi zu aktivieren, muss die `avr/sleep.h`-Library inkludiert werden. Zunächst muss eingestellt werden, wie der Controller aufgeweckt wird. Eine einfache Methode dafür ist der Watchdog-Timer, der, sobald er aktiviert wurde, immer im Hintergrund läuft. Um die Bits `WDE` oder `WDPx` des Watchdog-Registers `WDTCSR` verändern zu können, muss gleichzeitig auch das `WDCE`-Bit gesetzt werden. (Dieses Bit wird automatisch zurückgesetzt.)

Mit dem `WDIE`-Bit des `WDTCSR`-Registers, wird der Interrupt, welcher den Mikrocontroller aufweckt, aktiviert.

Mit den Bits `WDP0` bis `WDP3` wird die Sleep-Zeit eingestellt.

Mit der Funktion `set_sleep_mode(MODE)` ; wird der Sleep-Mode eingestellt. Folgendes kann für `MODE` eingesetzt werden:

- `SLEEP_MODE_IDLE`
- `SLEEP_MODE_PWR_DOWN`
- `SLEEP_MODE_PWR_SAVE`
- `SLEEP_MODE_ADC`
- `SLEEP_MODE_STANDBY`

- SLEEP_MODE_EXT_STANDBY

Mit der Funktion `sleep_mode()` ; wird der Sleep-Mode aktiviert.

Beispiel

```

1  #include <avr/io.h>
2  #include <util/delay.h>
3  #include <avr/interrupt.h>
4  #include <avr/sleep.h>
5
6  int main(void) {
7      DDRB = 255;
8
9      WDTCSR = 0b01010100;
10     WDTCSR = 0b01010100; // Sicherheitshalber 2-mal schreiben
11
12     // Global ISR (Interrupt Service Routine) aktivieren
13     sei();
14
15     set_sleep_mode(SLEEP_MODE_PWR_DOWN); // Sleep-Mode setzen
16
17     while(1) {
18         PORB ^= PORTB;
19         sleep_mode();
20     }
21 }
```

1.6 Power Saving

1.6.1 Peripherien

Mit Hilfe der Power-Reduction-Register `PRR0` und `PRR1` kann der Clock zu einzelnen Peripherien ausgeschaltet werden. Diese werden "eingefroren"; sobald sie wieder aktiviert werden, arbeiten sie dort weiter, wo sie aufgehört haben.

Merke, dass einige der Bit-Namen des `PRR1`-Registers nicht richtig hinterlegt sind und deswegen Compilefehler auftauchen können.

1.6.2 Pins

Weiterhin sollten alle nicht verwendeten Pins auf Input-mit-Pullup geschalten werden, d.h. im Programm:

```
1   DDRx = 0;  
2   PORTx = 255;
```

Weil diese Pins jetzt keine Outputs mehr sind, geht kein Strom "verloren". Sie als Input-Pullup zu definieren, verhindert, dass die CMOS-Eingänge ununterbrochen schalten (das auch zu Verlusten führen würde).

1.7 Externe Interrupts

Externe Interrupts lösen eine Funktion aus, wenn eine Zustandsänderung an einem Pin eintritt. Dieser Pin muss zuvor richtig als Eingang definiert werden. Außerdem müssen Interrupts global mittels `sei()` ; aktiviert sein.

und Ausgang?

Es gibt zwei verschiedene Arten von externen Interrupts:

1. "Normale" Interrupts

Hier wird ein Pin einzeln betrachtet; mit dem `EICRA`- bzw. `EICRB`-Register wird die Flanke, auf die geachtet werden soll, eingestellt.

11.1.1 External Interrupt Control Register A – EICRA

The External Interrupt Control Register A contains control bits for interrupt sense control.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---------------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| | ISC31 | ISC30 | ISC21 | ISC20 | ISC11 | ISC10 | ISC01 | ISC00 | EICRA |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

| ISCn1 | ISCn0 | Description |
|-------|-------|---|
| 0 | 0 | The low level of INTn generates an interrupt request. |
| 0 | 1 | Any edge of INTn generates asynchronously an interrupt request. |
| 1 | 0 | The falling edge of INTn generates asynchronously an interrupt request. |
| 1 | 1 | The rising edge of INTn generates asynchronously an interrupt request. |

Note: 1. n = 3, 2, 1, or 0.

When changing the ISCn1/ISCn0 bits, the interrupt must be disabled by clearing its Interrupt Enable bit in the EIMSK Register. Otherwise an interrupt can occur when the bits are changed.

Mit dem EIMSK-Register werden die einzelnen Pin-Interrupts freigegeben.

11.1.3 External Interrupt Mask Register – EIMSK

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---------------|-----|------|-----|-----|------|------|------|------|-------|
| | - | INT6 | - | - | INT3 | INT2 | INT1 | INT0 | EIMSK |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

2. Pin-Change Interrupts

Alle Pins des B-Registers können einen gemeinsamen Interrupt auslösen - egal auf welchem Pin die Zustandsänderung auftritt, es wird derselbe Interrupt ausgelöst.

Die Flanke, auf die geachtet wird, kann nicht eingestellt werden; es wird auf eine beliebige Flankenänderung gewartet.

Mit dem PCICR-Register wird der Pin-Change Interrupt aktiviert.

11.1.5 Pin Change Interrupt Control Register - PCICR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---------------|---|---|---|---|---|---|---|-------|-------|
| | - | - | - | - | - | - | - | PCIE0 | PCICR |
| Read/Write | R | R | R | R | R | R | R | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

11.1.7 Pin Change Mask Register 0 – PCMSK0

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---------------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| | PCINT7 | PCINT6 | PCINT5 | PCINT4 | PCINT3 | PCINT2 | PCINT1 | PCINT0 | PCMSK0 |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |