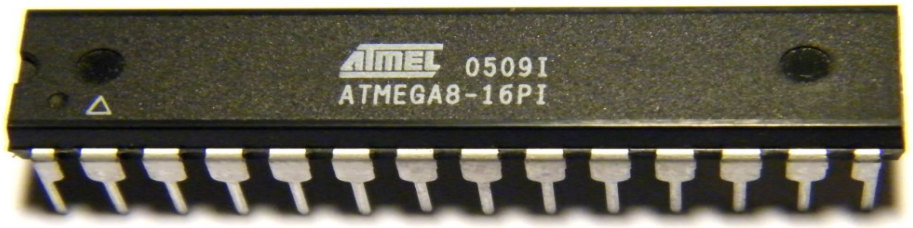
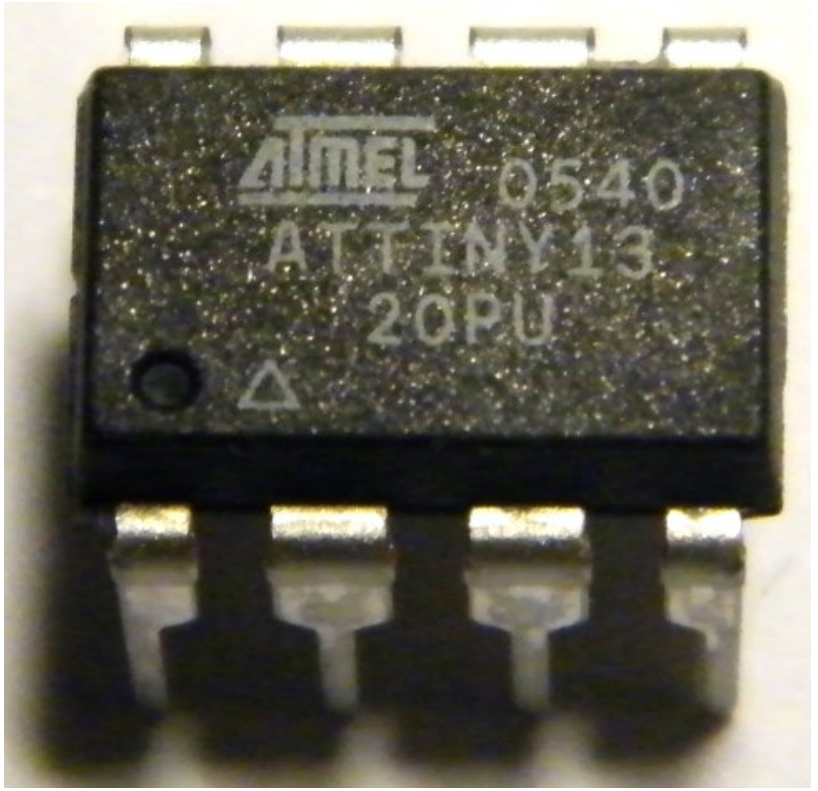
Kurs "Mikrocontroller Hard- und Software

Gerhard Schmidt
Kastanienallee 20
64289 Darmstadt





http://www.avr-asm-tutorial.net

Gliederung

- Hardware
  - Hardware-Aufbau
    - Controller
    - Experimentierboard
  - I/O-Ports
  - Timing
  - Hardwaretimer
  - AD-Wandler
  - Interruptsteuerung
  - Fuses

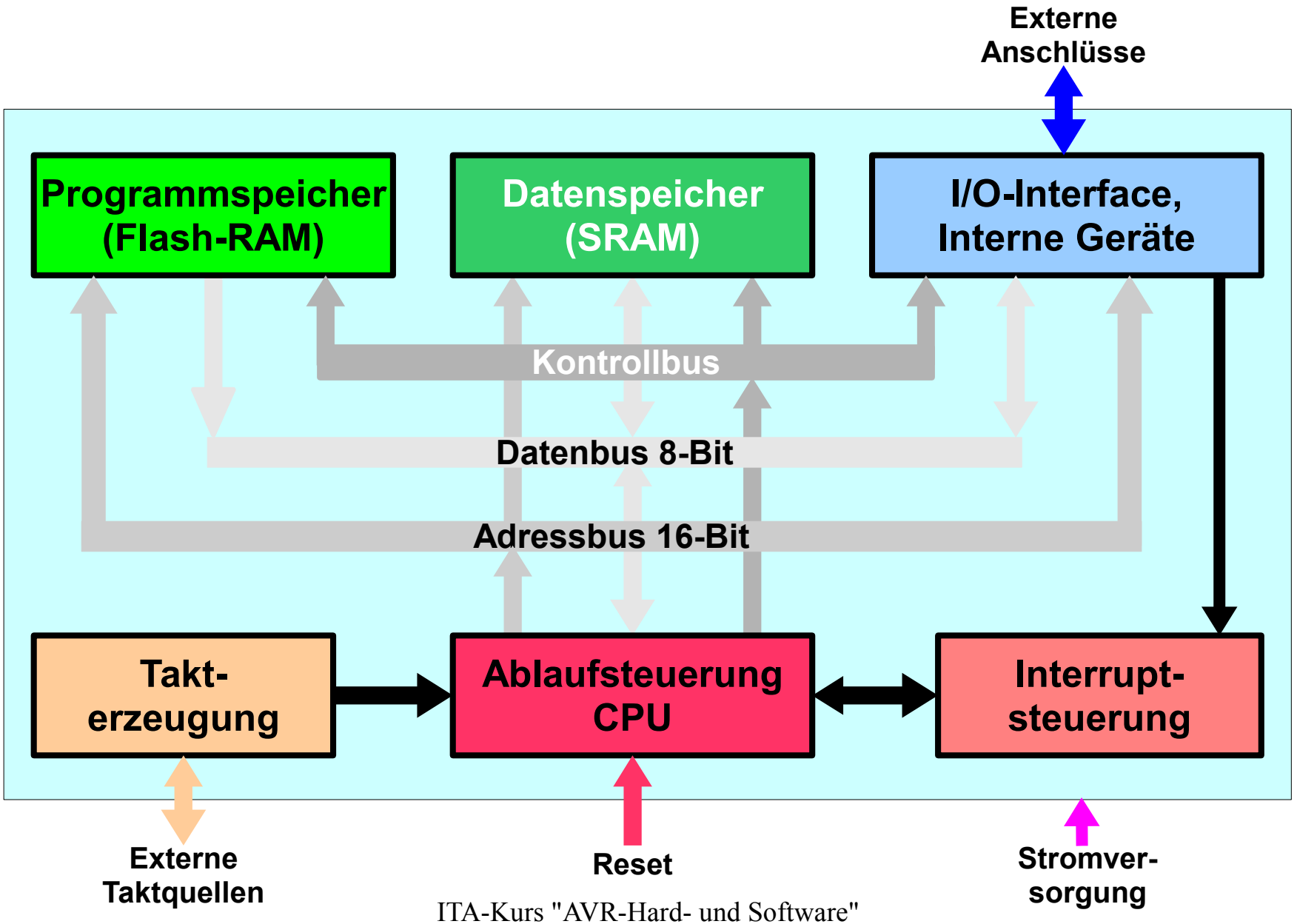
- Software (Assembler)
  - Studio-Programmierung
  - Instruktionen
  - Flash-Programmierung
  - Port-Programmierung
  - Schleifenprogramme
  - Timerprogrammierung
  - ADC-Programme
  - Interruptprogramme
  - Fuse-Einstellung

Features des ATtiny13

Features

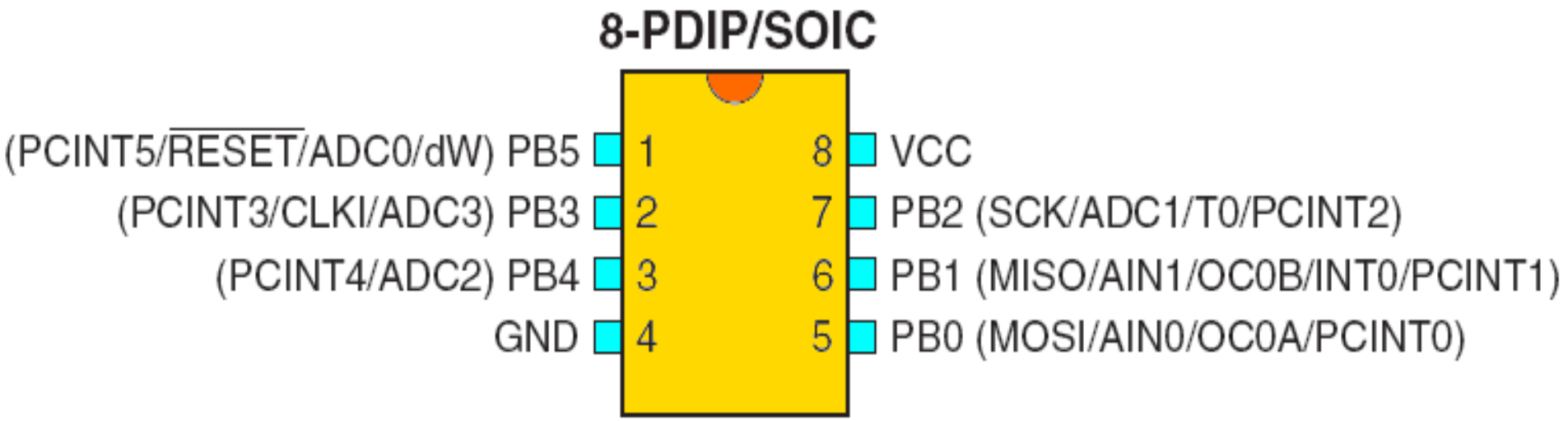
- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 120 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Througput at 20 MHz
- High Endurance Non-volatile Memory segments
  - 1K Bytes of In-System Self-programmable Flash program memory
  - 64 Bytes EEPROM
  - 64 Bytes Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 Years at 85°C/100 Years at 25°C (see page 6)
  - Programming Lock for Self-Programming Flash & EEPROM Data Security
- Peripheral Features
  - One 8-bit Timer/Counter with Prescaler and Two PWM Channels
  - 4-channel, 10-bit ADC with Internal Voltage Reference
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator

Interner Aufbau des ATtiny13



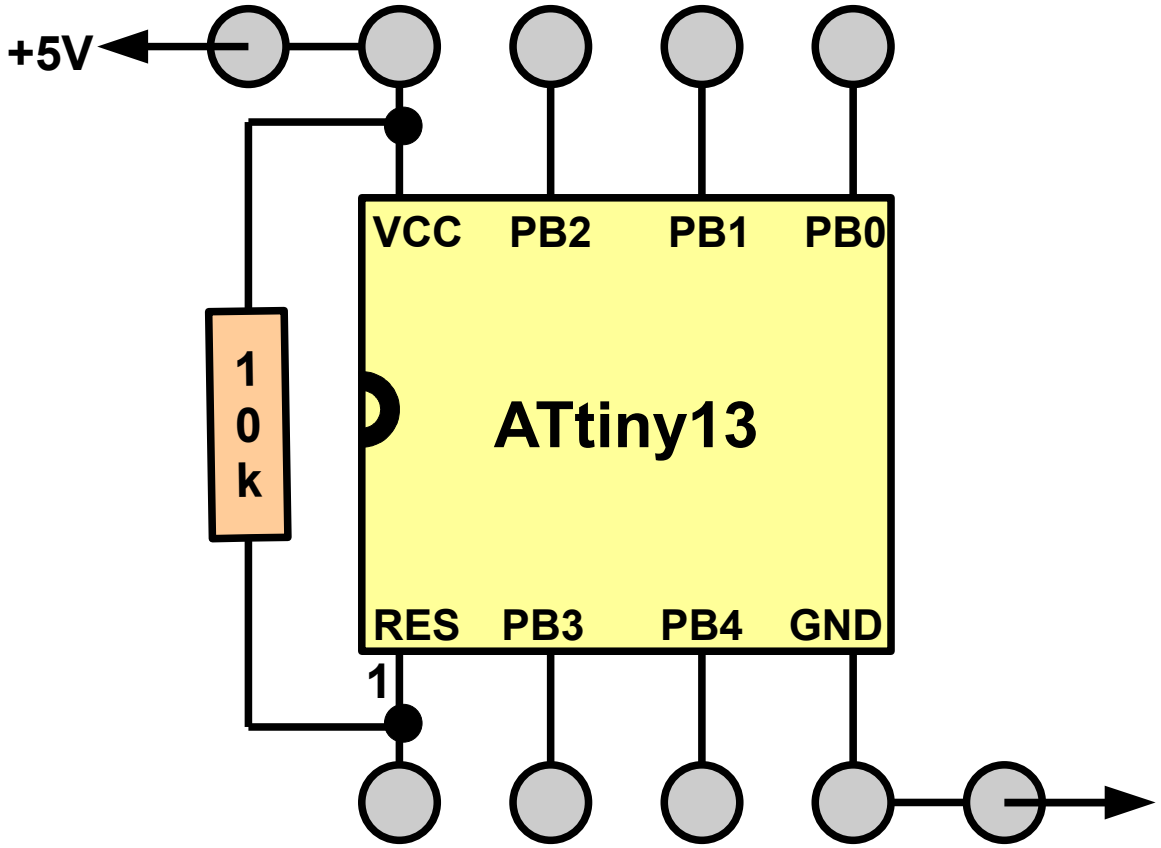
Externe Anschlüsse

- **VCC: +1,8..5,5 V Versorgungsspannung; GND: 0 V**
- **RESET bzw. PB5: Neustart des Prozessors**
- **CLKI, MOSI, MISO: Programmier-Interface für In-System-Programmierung (ISP)**
- **PB0 bis PB4: Ein- oder Ausgabe-Pins**

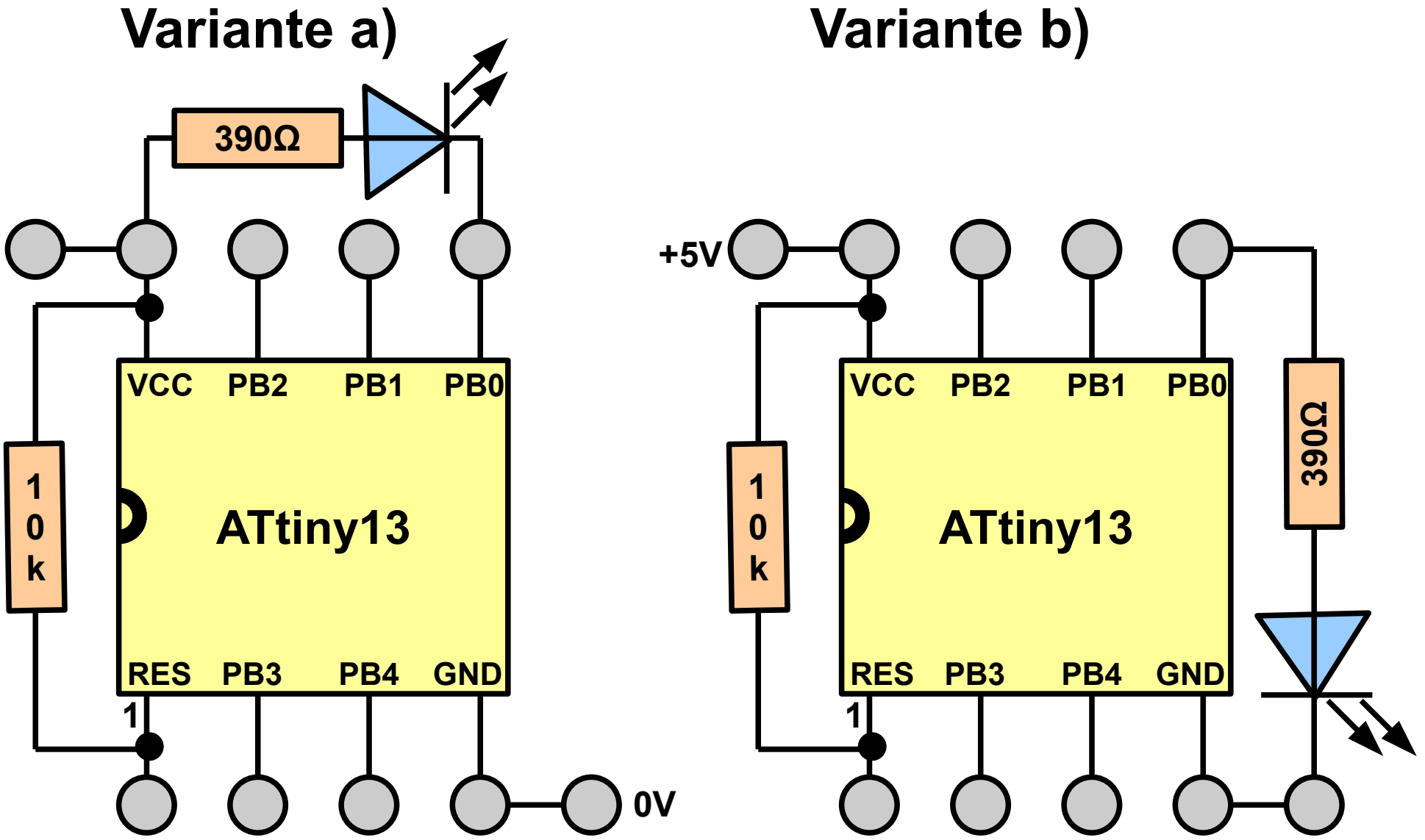


Tn13-Experimentierboard

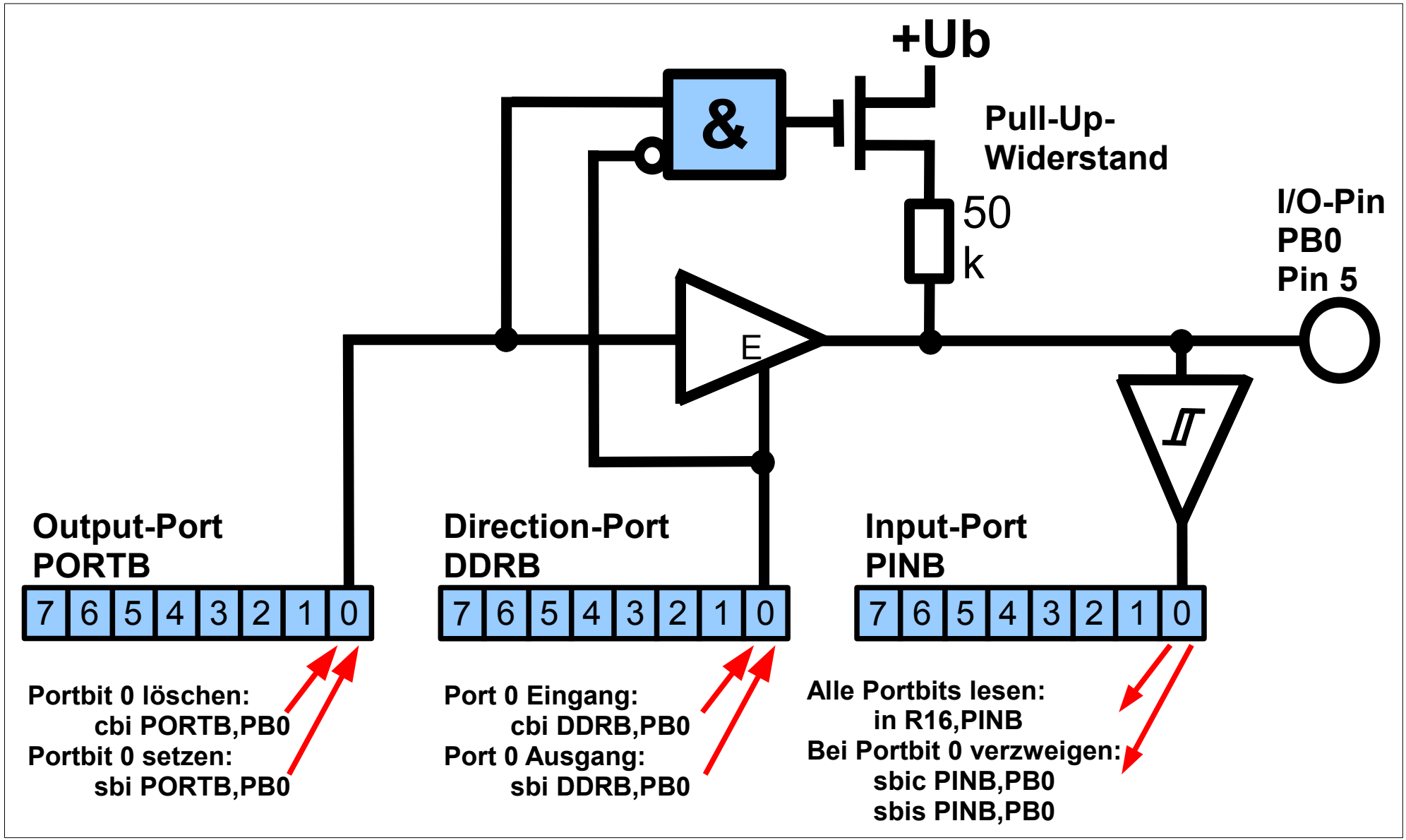
- ATtiny13 mit ISP10-Schnittstelle und 5V-Stromversorgung
- Ausführliche Beschreibung in der Datei tn13exp-beschreibung.pdf



Aufgabe 1: Eine LED einschalten



I/O-Port im AVR: Innenaufbau



Ports im ATtiny13, Teil 1

20. Register Summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x3F	SREG	I	T	H	S	V	N	Z	C
0x3E	Reserved	–	–	–	–	–	–	–	–
0x3D	SPL	SP[7:0]							
0x3C	Reserved	–	–	–	–	–	–	–	–
0x3B	GIMSK	–	INT0	PCIE	–	–	–	–	–
0x3A	GIFR	–	INTF0	PCIF	–	–	–	–	–
0x39	TIMSK0	–	–	–	–	OCIE0B	OCIE0A	TOIE0	–
0x38	TIFR0	–	–	–	–	OCF0B	OCF0A	TOV0	–
0x37	SPMCSR	–	–	–	CTPB	RFLB	PGWRT	PGERS	SELFPR-
0x36	OCR0A	Timer/Counter – Output Compare Register A							
0x35	MCUCR	–	PUD	SE	SM1	SM0	–	ISC01	ISC00
0x34	MCUSR	–	–	–	–	WDRF	BORF	EXTRF	PORF
0x33	TCCR0B	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00
0x32	TCNT0	Timer/Counter (8-bit)							
0x31	OSCCAL	Oscillator Calibration Register							
0x30	BODCR	–	–	–	–	–	–	BODS	BODSE
0x2F	TCCR0A	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00
0x2E	DWDR	DWDR[7:0]							
0x2D	Reserved	–							
0x2C	Reserved	–							
0x2B	Reserved	–							
0x2A	Reserved	–							
0x29	OCR0B	Timer/Counter – Output Compare Register B							
0x28	GTCCR	TSM	–	–	–	–	–	–	PSR10
0x27	Reserved	–							
0x26	CLKPR	CLKPCE	–	–	–	CLKPS3	CLKPS2	CLKPS1	CLKPS0
0x25	PRR	–	–	–	–	–	–	PRTIM0	PRADC

Ports im ATtiny13, Teil 2

0x1F	Reserved	–							
0x1E	EEARL	–	–	EEPROM Address Register					
0x1D	EEDR	EEPROM Data Register							
0x1C	EECR	–	–	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE
0x1B	Reserved	–							
0x1A	Reserved	–							
0x19	Reserved	–							
0x18	PORTB	–	–	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
0x17	DDRB	–	–	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
0x16	PINB	–	–	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
0x15	PCMSK	–	–	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0
0x14	DIDR0	–	–	ADC0D	ADC2D	ADC3D	ADC1D	AIN1D	AIN0D
0x13	Reserved	–							
0x12	Reserved	–							
0x11	Reserved	–							
0x10	Reserved	–							
0x0F	Reserved	–							
0x0E	Reserved	–							
0x0D	Reserved	–							
0x0C	Reserved	–							
0x0B	Reserved	–							
0x0A	Reserved	–							
0x09	Reserved	–							
0x08	ACSR	ACD	ACBG	ACO	ACI	ACIE	–	ACIS1	ACIS0
0x07	ADMUX	–	REFS0	ADLAR	–	–	–	MUX1	MUX0
0x06	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
0x05	ADCH	ADC Data Register High Byte							
0x04	ADCL	ADC Data Register Low Byte							
0x03	ADCSRB	–	ACME	–	–	–	ADTS2	ADTS1	ADTS0
0x02	Reserved	–							
0x01	Reserved	–							
0x00	Reserved	–							

Das Programm schreiben

1) Starten der Studio-Software
2) Neues Projekt anlegen
3) Assembler-Programm in Editor eintippen
4) Quellcode assemblieren (Bild 4)
5) mit AVRISPmkII-Programmiergerät in den Tiny13 programmieren

Siehe avrisp_mk_II_beschreibung.pdf

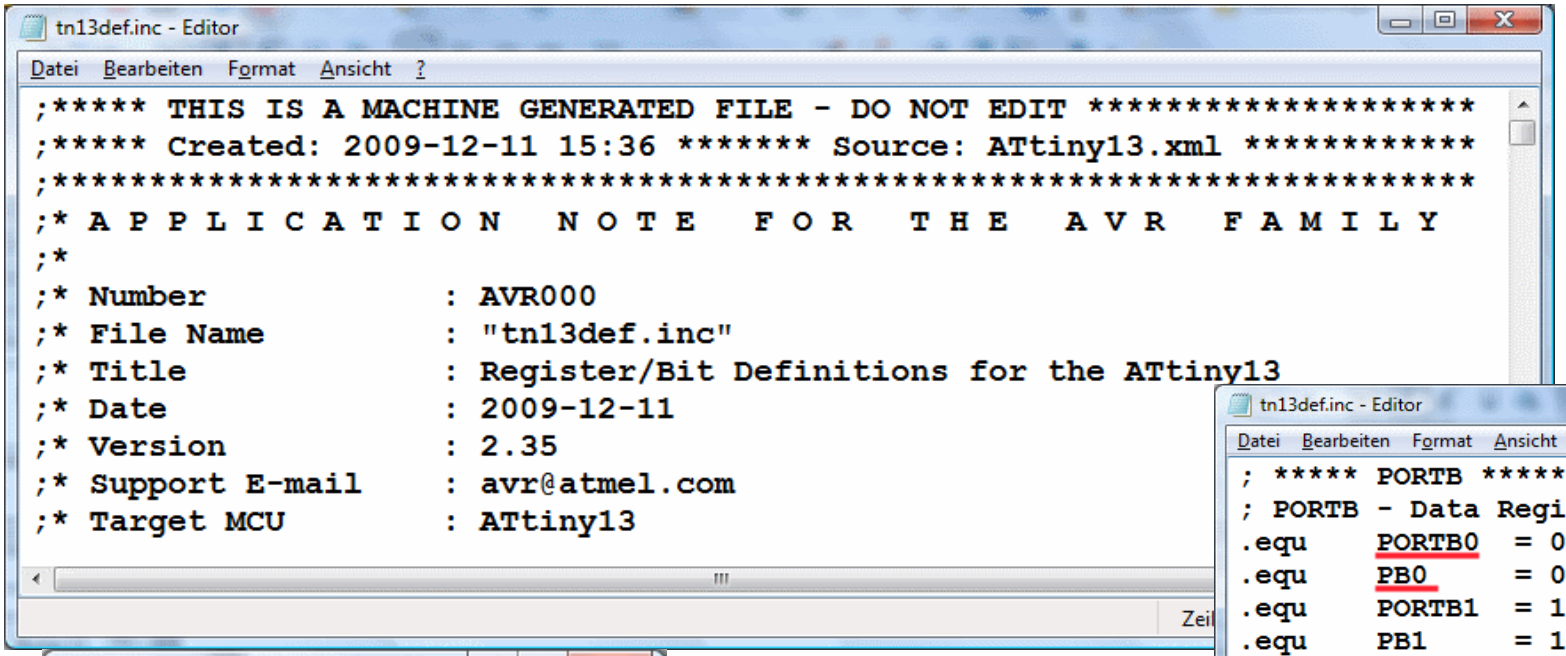
Der Programmaufbau

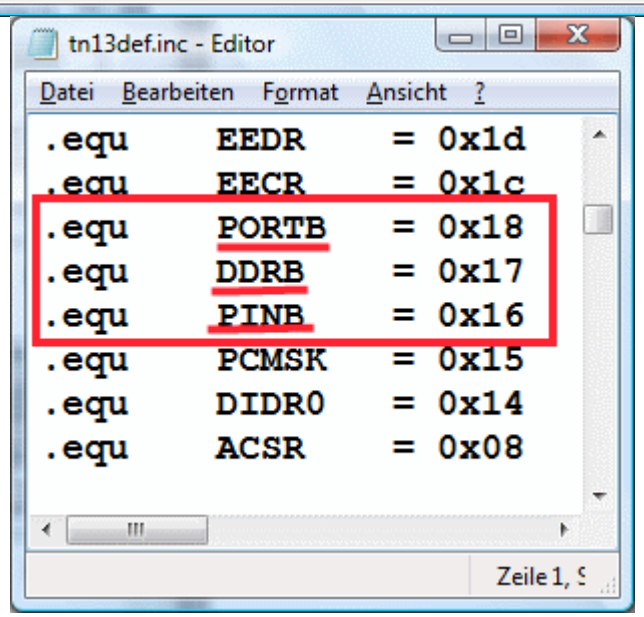
```
;
; Test schaltet Lampe
; an PB0 ein
;

.nolist
.include "tn13def.inc"
.list

    sbi DDRB,PB0
    cbi PORTB,PB0

loop:
    rjmp loop
```

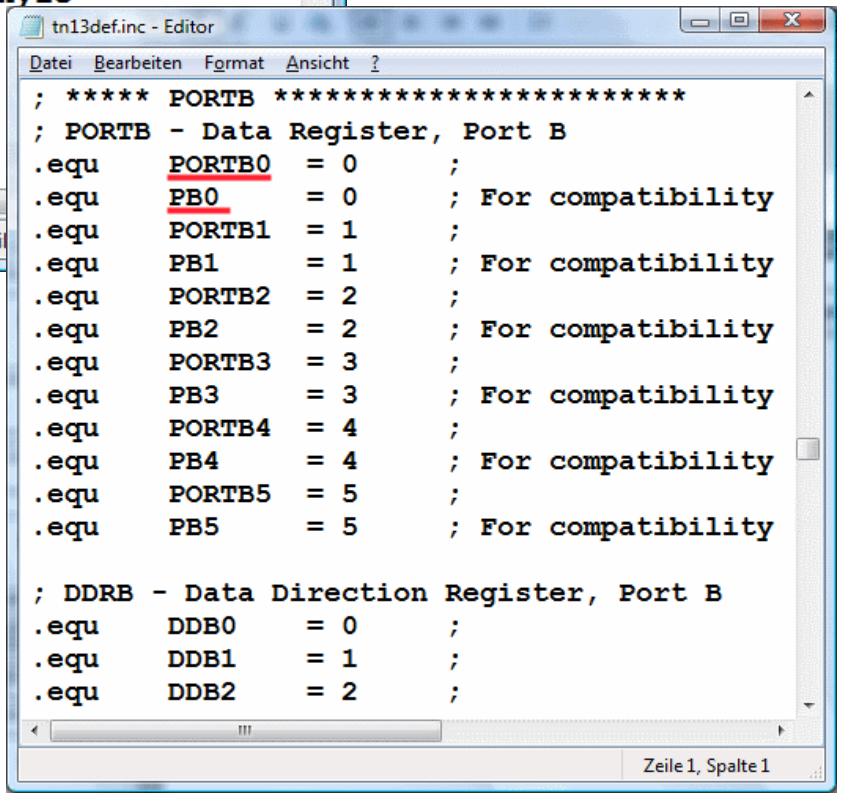
- Kommentar, beginnt mit ";"
- "." sind Assemblerdirektiven:
  .NOLIST schaltet Listing aus
  .INCLUDE liest Datei mit Typ-spezifischen Definitionen ein
  .LIST schaltet Listing wieder an
- Assemblerinstruktionen
  SBI: Set Bit in I/O-Port
  CBI: Clear Bit in I/O-Port
  DDRB,PORTB: Ports
  PB0: Portbit
- "loop: rjmp loop": Unendliche Schleife
  loop: Sprungziel, Label
  RJMP: Relativer Sprung (Jump)

Include Header-Datei "tn13def.inc"

Ort: „C:\Program Files\Atmel\AVR Tools\AvrAssembler2\Appnotes“

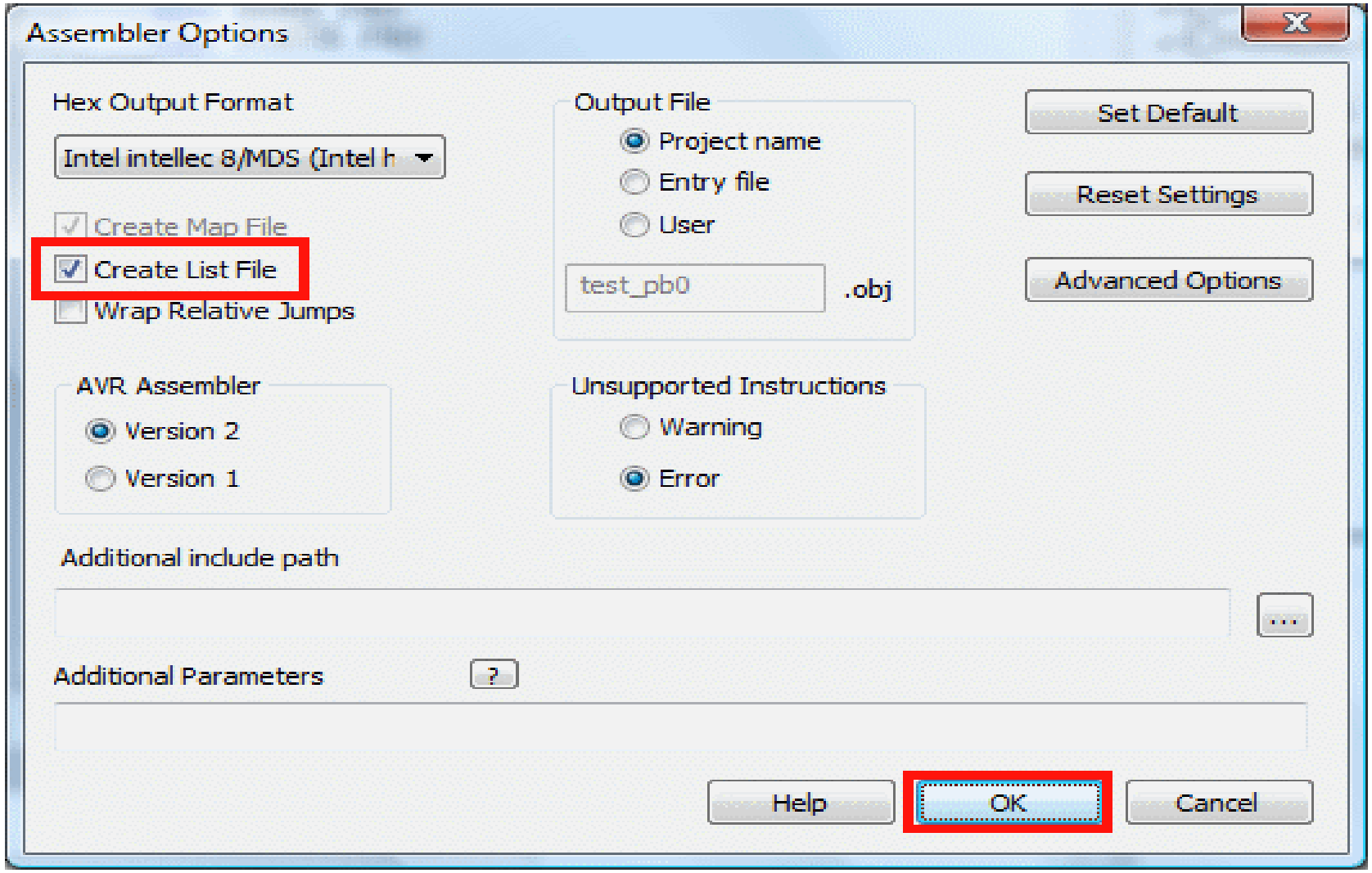






Das Listing des Programms

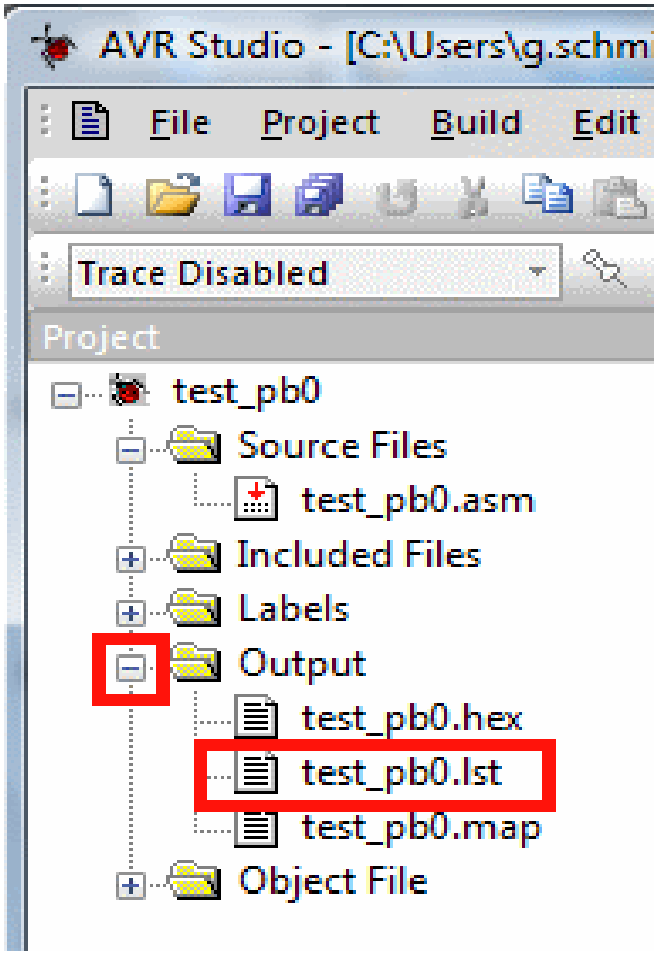
Im Studio „Project/Assembler Options“ öffnen:



Button „Create List File“ ein-schal-ten, mit „OK“ schlie-ßen

Das Listing des Programms

Im „Project“-Fenster die Datei „Output/test_pb0.lst“ auswählen



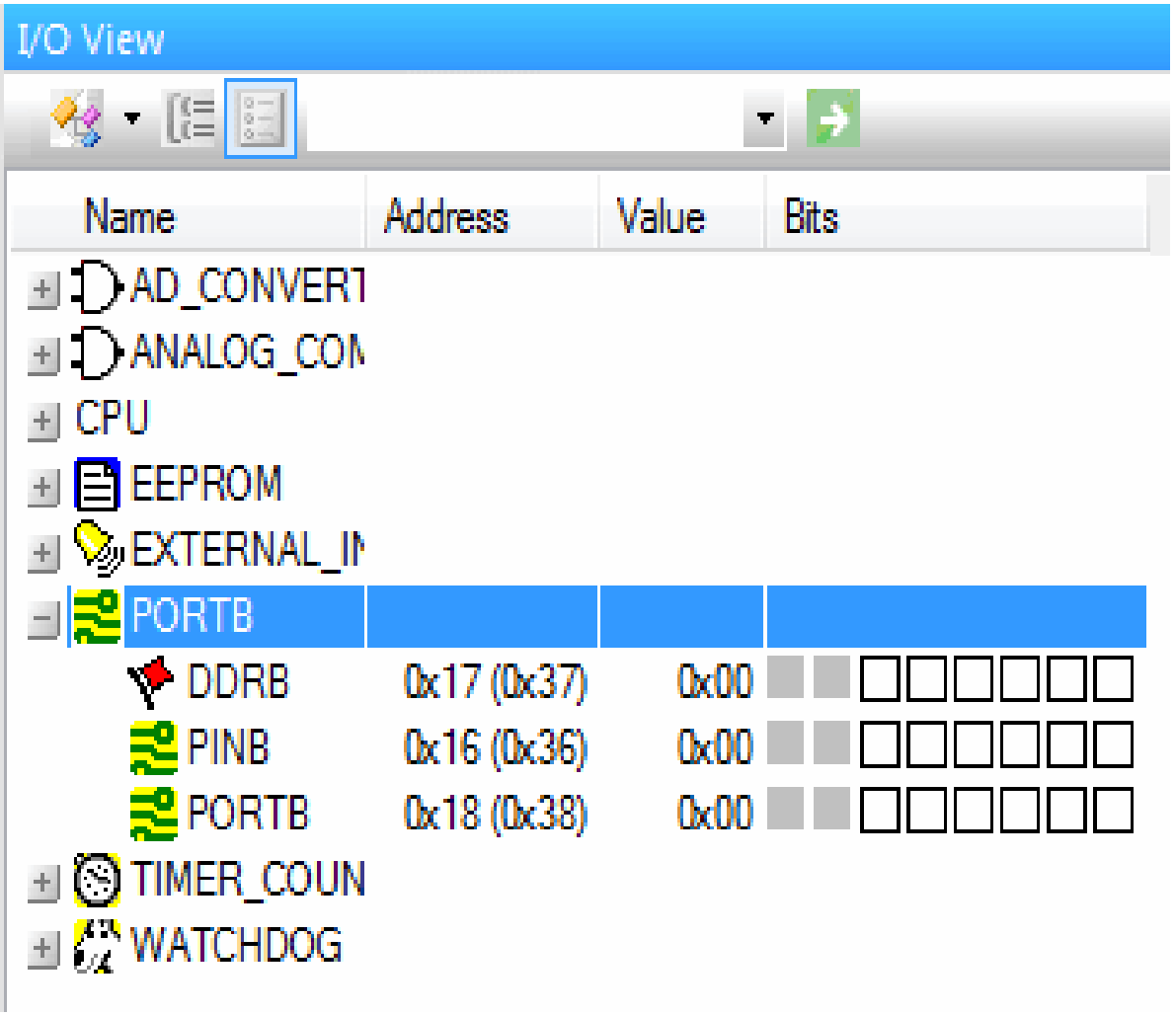

```
AVRASM ver. 2.1.42  C:\Users\g.schmi

C:\Users\g.schmidt\Documents\dev\AVR

                        ; test PB0
                        ;
                        .list
                        ;
000000 9ab8                sbi DDRB,PB0
000001 98c0                cbi PORTB,PB0
                        loop:
000002 cfff                rjmp loop
```

Rot: Adresse der Instruktion; Grün: Opcode der Instruktion; Blau: Quellcode-Zeile der Instruktion

Simulation des Programms

Mit „Build/Build and Run“ den Simulator starten, mit „View-Toolsbars-I/O“ den Ein- und Ausgabebereich sichtbar machen, darin den PORTB auswählen

```
;
; test PB0
;
.nolist
.include "tn13def.inc"
.list
;
    sbi DDRB,PB0
    cbi PORTB,PB0
loop:
    rjmp loop
```

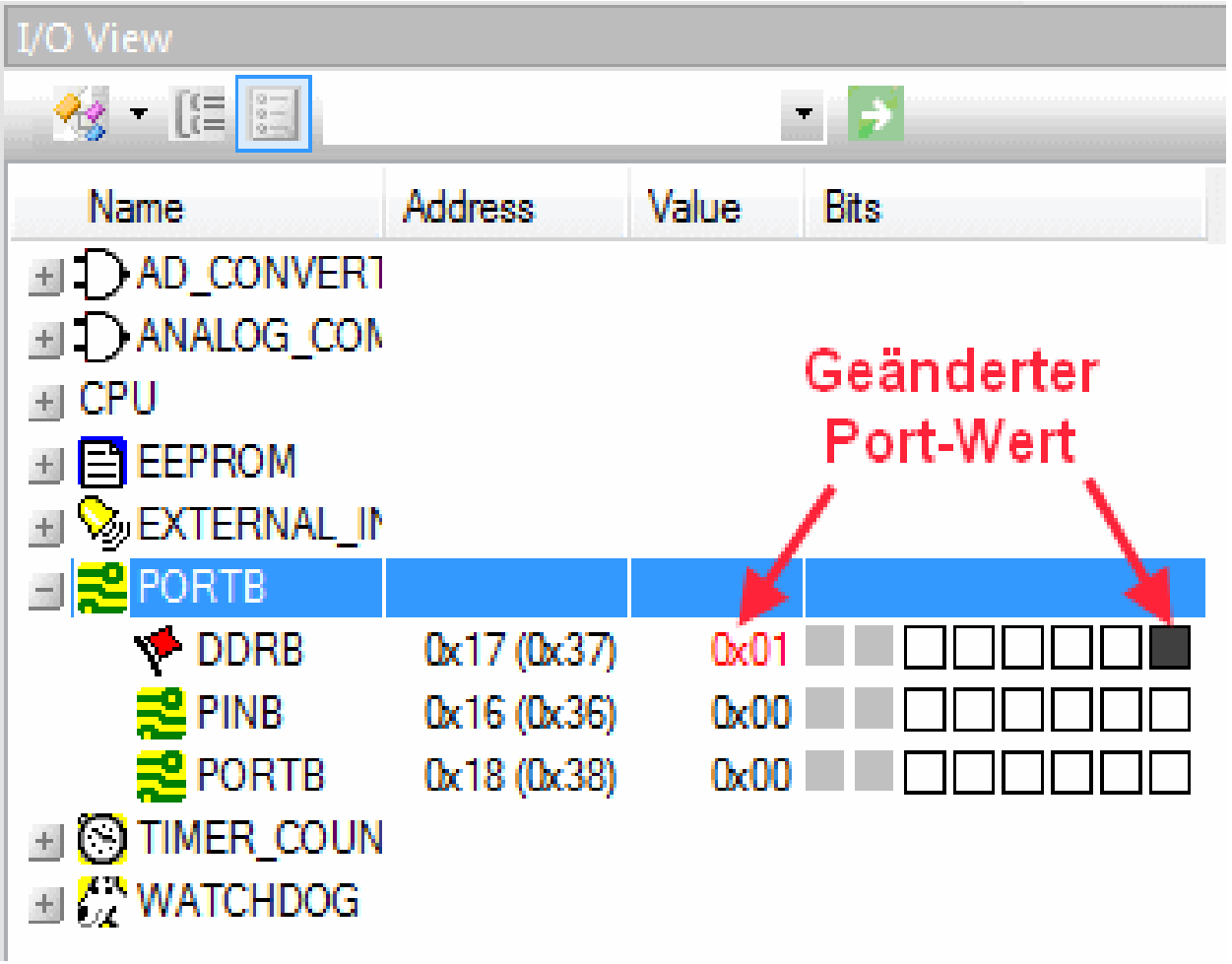


Simulation des Programms

Mit „Debug/StepInto“ oder F11 einen Einzelschritt ausführen

```
;
;  test PB0
;
.nolist
.include "tn13def.inc"
.list
;
    sbi DDRB,PB0
    cbi PORTB,PB0
loop:
    rjmp loop
```

Nächste Instruktion



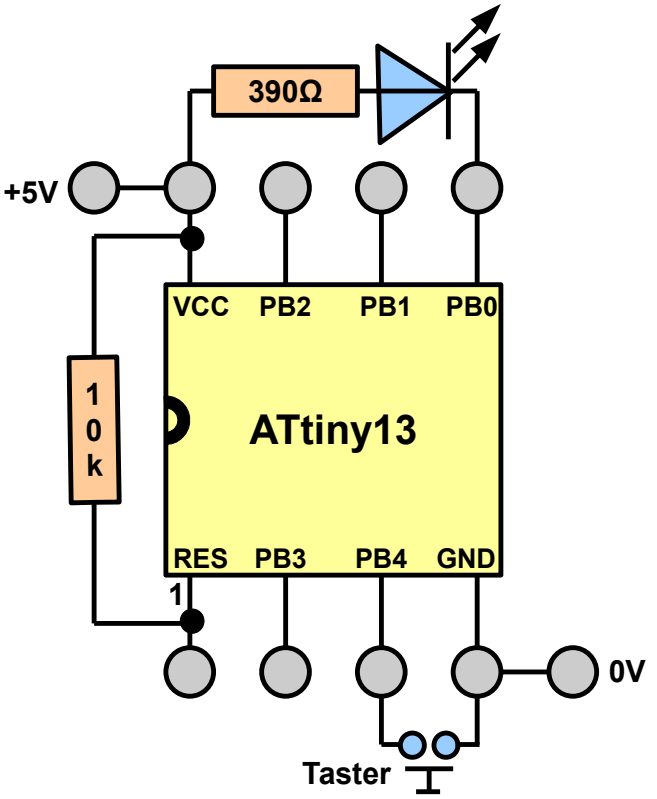
Die Änderungen durch die gerade ausgeführte Instruktion werden sichtbar!

Aufgabe 2: Ein- und Ausschalten der LED

- Aufgabe:
  Schreibe test_pb0.asm so um, dass die LED dauerhaft wechselnd ein- und ausgeschaltet wird.
- Ermittle für alle verwendeten Instruktionen die Ausführungszeiten bei einer Taktfrequenz von 1,2 MHz (Hinweis: Spalte "#Clocks" in der Instruktionstabelle).
- Mit welcher Frequenz und Pulsweite leuchtet die LED?

Aufgabe 3: Einschalten der LED nur bei gedrücktem Taster

- Aufgabe:
  Schreibe test_pb0.asm so um, dass die LED bei gedrücktem Taster an PB4 (Pin 3) eingeschaltet wird.
- Simuliere das Programm im Studio-Simulator.
- Hinweise:
  - Einlesen des Bits an PB4 und verzweigen mit den Instruktionen SBIC PINB,PB4 oder SBIS PINB,PB4
  - SBIC = Skip if Bit in I/o register is Cleared, überspringt nächste Instruktion, falls Portbit Null ist
  - SBIS = dto., falls Portbit Eins ist



Register im AVR

- 32 Byte 8-Bit-Lese-/Schreib-Speicher
- Schnellzugriff durch CPU, aufgrund der Vielzahl auch für oft gebrauchte Variablen nutzbar
- Benennung in Assembler: R0 bis R31
- Setzen eines Registers auf einen Wert:
  - LDI R16,128 ; setzt Register R16 auf dezimal 128
  - LDI R16,0x80 ; setzt Register R16 auf hex 80
  - geht aber nur mit den Registern R16 bis R31!
- Direkt adressierbar von der CPU:
  Beispiel einer 8-Bit-Addition:
  - Instruktion: ADD R0,R1 ; addiere Register R1 zu R0 und speichere Resultat in R0

Eine Verzögerungsschleife

```
; 8-Bit
; Verzögerung
   ldi R16,200


noch_mal:


   dec R16


   brne noch_mal
```

- Kommentare
- Lade Dezimalzahl 200 in das Register R16 = Anzahl Wiederholungen der Schleife
- Label (Marke) zur Wiederholung der Schleife
- Dekrementiere das Register R16 = zieht eine Eins ab, setzt das Z-Flag im Statusregister der CPU, wenn Registerinhalt = Null
- "Branch Relative if Not Equal" = Verzweige zum Label, wenn nicht gleich Null, fragt Z-Flag im Statusregister ab

Verzögerungszeit der Schleife

```
; 8-Bit
; Verzögerung
   ldi R16,200 ; 1 Takt
noch_mal:
   dec R16 ; 1 Takt
   brne noch_mal
; brne: 1 Takt bei Durch-
; lauf, 2 Takte bei Ver-
; zweigung
```

199 Durchläufe der Schleife mit Verzweigung plus 1 Durchlauf der Schleife ohne Verzweigung

n(Takte) = 199 * (1+2) + 1 * (1+1)
= 597 + 2
= 599

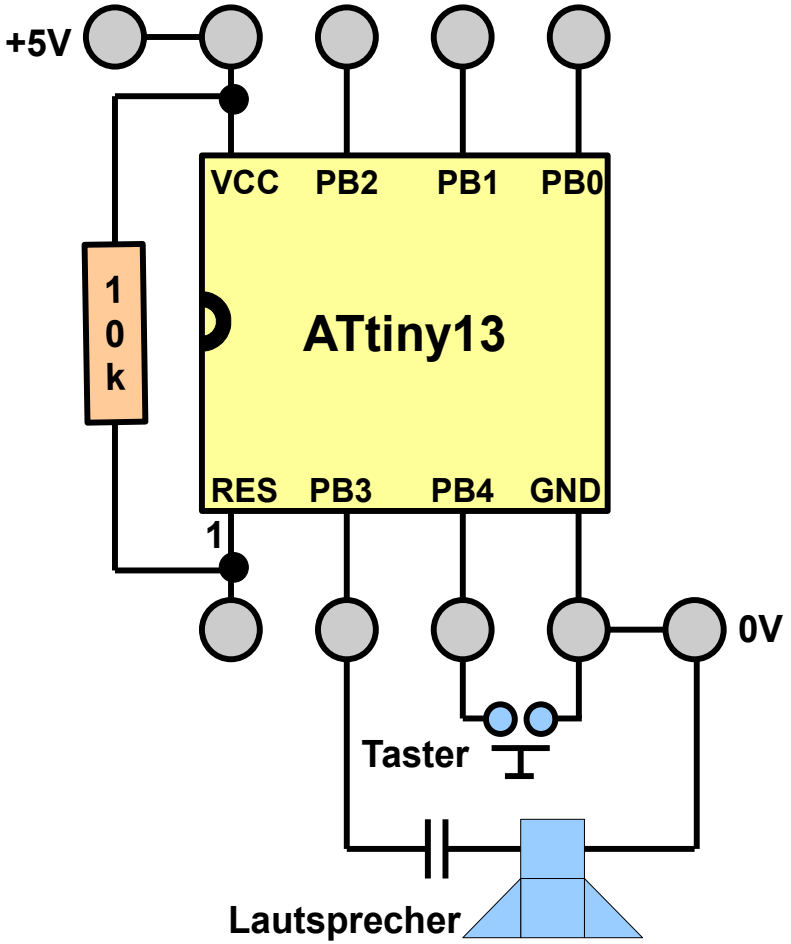
Verzögerungszeit bei 1,2 MHz Takt:
t(Zeit) = 599 / 1,2 [µs]
= 499 [µs]

Frequenz bei zwei Schleifen:
f (Ton) = 1 / (2 * t)
= 1002 Hz

Allgemein (c = Konstante):
n(Takte) = 3*(c-1)+ 2 = 3 * c - 1
t(Zeit) = (3 * c - 1) / 1,2 [µs]
f(Ton) = 600000 / (3 * c – 1) [Hz]
c = (600000 / f(Ton) – 1) / 3

Aufgabe 4: Tonerzeugung mit Schleifen

- Erzeuge mit zwei 8-Bit-Verzögerungsschleifen und einem Lautsprecher an PB3 einen Ton von 1760 Hz, wenn die Taste gedrückt wird.
- Simuliere das Programm
  - Stelle mit "Debug-Simulator Options" die Simulatorfrequenz auf 1,2 MHz um (Texteingabe "1.2" in die Zeile).
  - Setze Breakpoints auf die Ein- und Ausschalteinstruktionen: Cursor in die Zeile setzen, rechte Maustaste, "Toggle Breakpoint", rote Markierung erscheint
  - Zähle die Takte mit "View-Toolbars-Processor", dort mit der "Stopwatch", mit der rechten Maustaste kann diese auf Null gesetzt werden.
  - Mit "Debug – Run" oder F5 läuft der Simulator von Breakpoint bis Breakpoint.



Doppelregister (16-Bit)

- Doppelregister sind zwei benachbarte Register,
  - für die 16-Bit-Instruk-tionen möglich sind
  - die sich als 16-Bit-Adresszähler (Pointer) eignen
- Register:
  - R27:R26 = X; XH:XL
  - R29:R28 = Y; YH:YL
  - R31:R30 = Z; ZH:ZL

- Instruktionen:
  - Laden:
    .equ c = 65535
    ldi XH,HIGH(c)
    ldi XL,LOW(c)
  - Um Eins erhöhen:
    adiw XL,1
  - Um Eins erniedrigen:
    sbiw XL,1
  - Register in Speicher-zelle schreiben/lesen:
    st X,R0 bzw. ld R0,X

16-Bit-Schleifen

```
; 16-Bit
; Verzögerung
.equ zahl = 60000
   ldi YH,HIGH(zahl)
   ldi YL,LOW(zahl)
noch_mal:
   sbiw YL,1 ; 2 Takte
   brne noch_mal
; 1 Takt bei Durchlauf,
; 2 Takte bei Verzweigung
```

59.999 Durchläufe der Schleife mit Verzweigung plus 1 Durchlauf der Schleife ohne Verzweigung

n(Takte) = 59999 * (2+2) + 1 * (2+1)
= 239.996 + 3
= 239.999

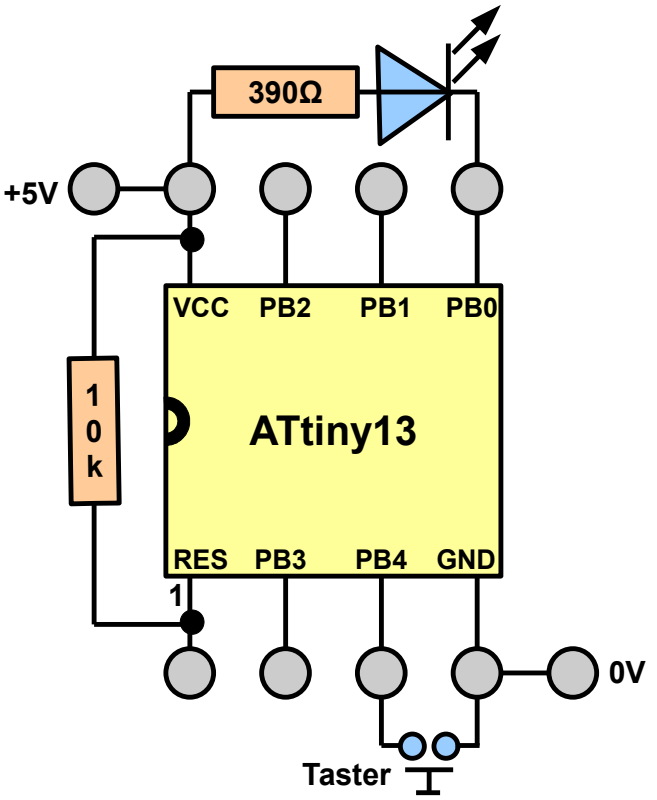
Verzögerungszeit bei 1,2 MHz Takt:
t(Zeit) = 239.999 / 1,2 [µs]
= 199.999 [µs]

Blinkfrequenz bei zwei Schleifen:
f (Blink) = 1 / (2 * t)
= 2,5 Hz

Allgemein (c = Konstante):
n(Takte) = 4 * (c-1) + 3 = 4 * c - 1
t(Zeit) = (4 * c - 1) / 1,2 [µs]
f(Blink) = 600000 / (4 * c – 1) [Hz]
c = (600000 / f(Blink) – 1) / 4

Aufgabe 5: 4 Hz-LED mit 16-Bit-Schleife

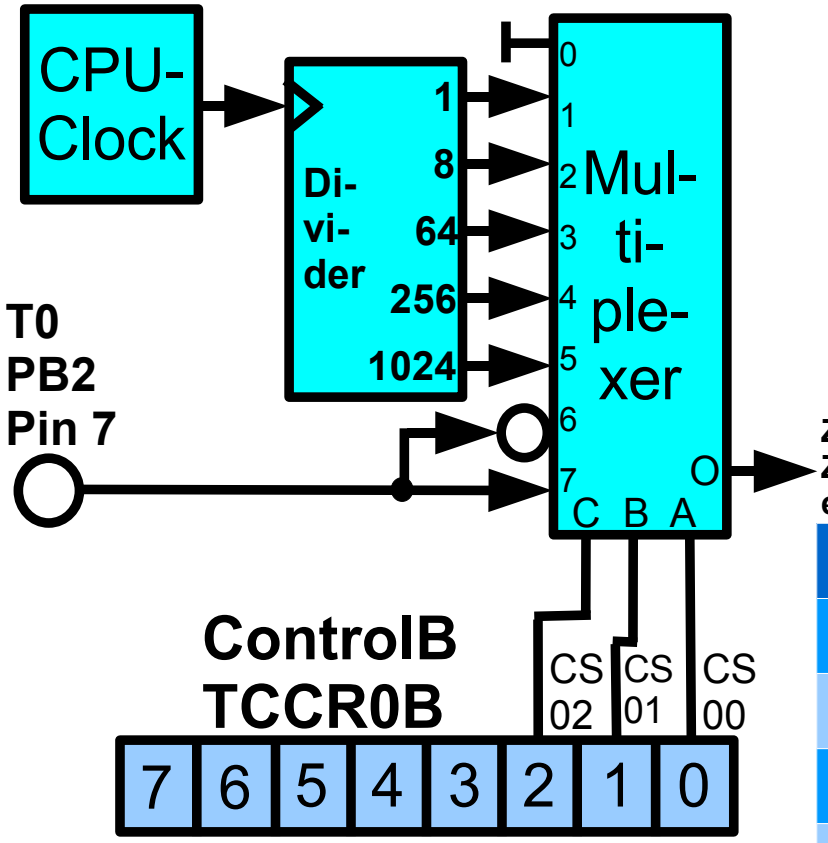
- Aufgabe:
  Schreibe ein Programm mit 16-Bit-Schleifen, das die LED bei gedrücktem Taster an PB4 (Pin 3) mit 1 Hz aus- und einschaltet.
- Hinweise:
  - Einlesen des Bits an PB4 und verzweigen mit den Instruktionen SBIC PINB,PB4 oder SBIS PINB,PB4
  - SBIC = Skip if Bit in I/o register is Cleared, überspringt nächste Instruktion, falls Portbit Null ist
  - SBIS = dto., falls Portbit Eins ist
  - Da bei 1 Hz die Konstante über 65536 liegt (größte 16-Bit-Zahl), müssen drei Zählschleifen kombiniert werden!



Timer/Counter

- Für Zähl- und Timing-Aufgaben haben alle AVR Zähler-/Timer-Bausteine (T/C) an Bord.
- Kleine AVR haben mindestens einen 8-Bit-T/C, größere AVR weitere 8- und 16-Bit-T/C.
- Ein T/C besteht aus:
  - einem wählbaren Vorteiler für den Systemtakt (Prescaler),
  - einem 8- oder 16-Bit breiten Zählregister,
  - zwei entsprechend breiten Vergleichsregistern (Compare A und Compare B).

T/C-Vorteiler



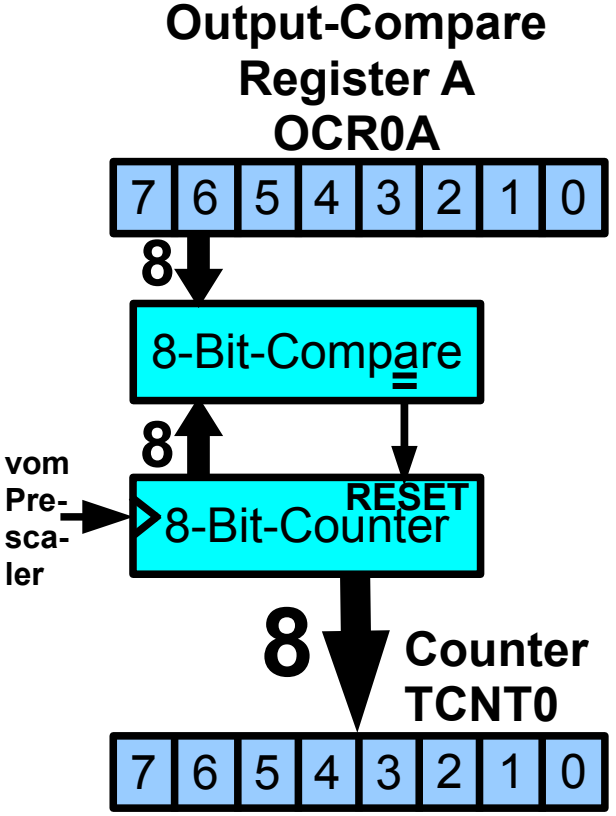
- Der Vorteiler wird durch das Kontrollwort TCCR0B gesteuert.
- Die drei CS-Bits kontrollieren, von welcher Quelle das Zählsignal abgeleitet wird.

2	1	0	Zähler/Timer-Steuerung
0	0	0	Zähler/Timersignal ausgeschaltet
0	0	1	Timer, zählt CPU-Takt
0	1	0	Timer, zählt CPU-Takt / 8
0	1	1	Timer, zählt CPU-Takt / 64
1	0	0	Timer, zählt CPU-Takt / 256
1	0	1	Timer, zählt CPU-Takt / 1024
1	1	0	Counter, zählt ansteigende Pulse an PB2
1	1	1	Counter, zählt abfallende Pulse an PB2

Programmieren des Vorteilers

```
; Timer starten, Vorteiler = 1024
   ldi R16,(1<<CS2) | (1<<CS0)
   out TCCR0B,R16
; Timer stoppen
   clr R16
   out TCCR0B,R16
```

Counter und Compare



- Der Counter zählt die vom Vorteiler kommenden Impulse. Er zählt aufwärts.
- Sein Zählerstand ist durch Lesen des Ports TCNT0 zugänglich.
- Der Zählerstand wird laufend mit dem Inhalt des Compare-Registers OCR0A (und OCR0B) verglichen.
- Bei Gleichheit können weitere Aktionen ausgelöst werden.

Programmieren des Counters

- **Lesen des Counters**
  in R16,TCNT0
- **Schreiben des Counters**
  ldi R16,100
  out TCNT0,R16

Programmieren des Compare Registers

- **Schreiben des Compare-Registers A**
  ldi R16,128
  out OCR0A,R16

Timer-Modi

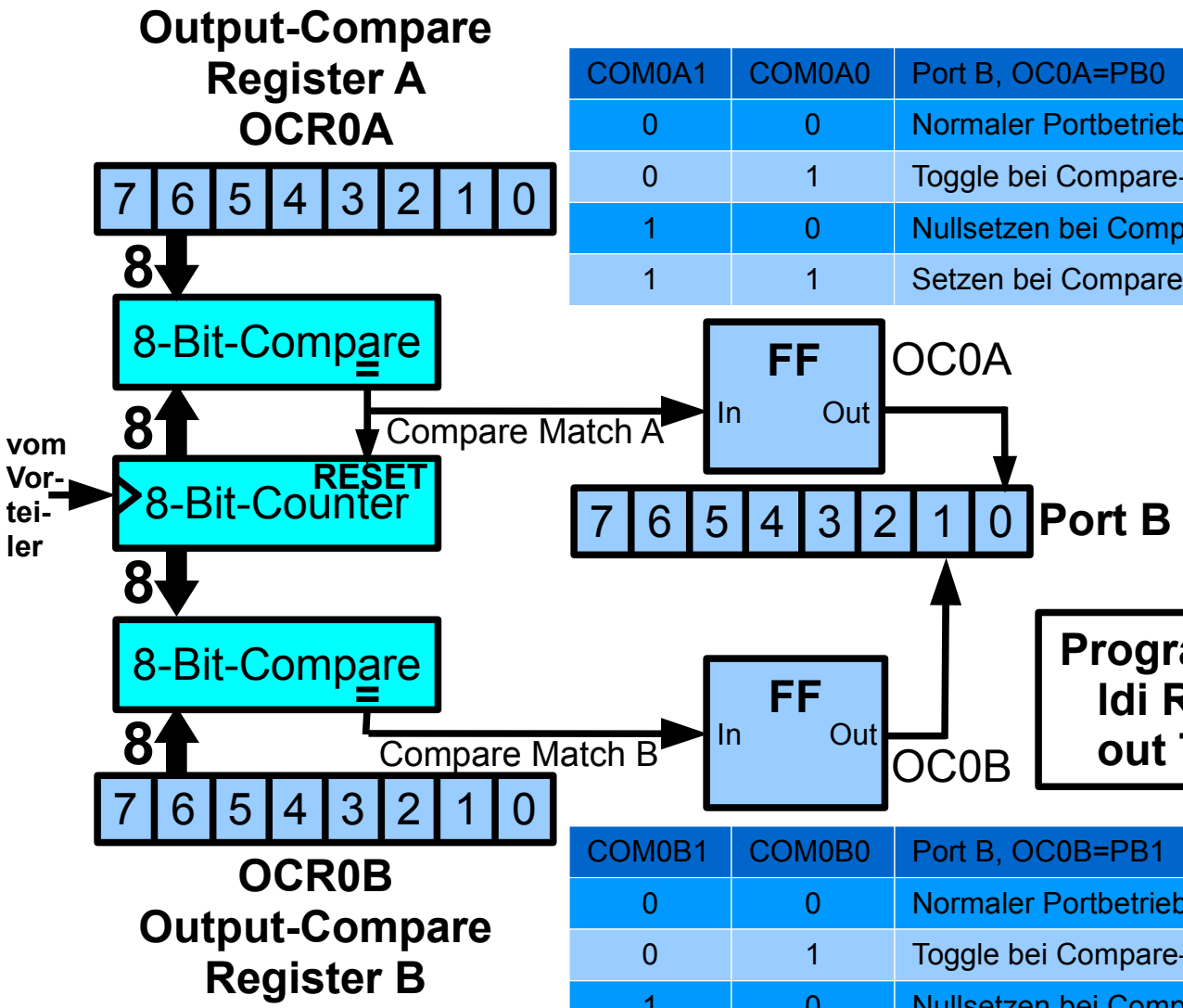
Acht verschiedene Arten, den Timer zu betreiben.
Auswahl über das Setzen der WGM-Bits in den Kontrollregistern TCCR0A (WGM00 und WGM01) und TCCR0B (WGM02)

	WGM			Modus	Top	Update	Int
#	2	1	0			OCRx	TOV
0	0	0	0	Normal (Timing, Zählen)	0xFF	Sofort	0xFF
1	0	0	1	Phasenkorrekter PWM, 8 Bit	0xFF	TOP	0x00
2	0	1	0	CTC (Clear Timer on Compare)	OCRA	Sofort	0xFF
3	0	1	1	Schnelle PWM, 8 Bit	0xFF	TOP	0xFF
4	1	0	0	-			
5	1	0	1	Phasenkorrekter PWM, <8 Bit	OCRA	TOP	0x00
6	1	1	0	-			
7	1	1	1	Schnelle PWM, <8 bit	OCRA	TOP	TOP

Clear Timer on Compare (CTC)

- CTC-Modus: Bei Erreichen des Comparewertes im Compare-Register A wird der Zähler wieder auf Null gesetzt.
- Ermöglicht in weiten Bereichen einstellbare Teilerraten.
- Beispiel:
  - Taktfrequenz: 1,2 MHz, Vorteiler: 1064, Cmp A = 255
    1,2 MHz / 1064 / 255 = 4,42 Hz
  - Taktfrequenz: 9,6 MHz, Vorteiler: 1, CmpA = 2
    9,6 MHz / 1 / 2 = 4,8 MHz

Timer-Signale und Ports



COM0A1	COM0A0	Port B, OC0A=PB0
0	0	Normaler Portbetrieb, nicht verbunden
0	1	Toggle bei Compare-Match A
1	0	Nullsetzen bei Compare-Match A
1	1	Setzen bei Compare-Match A

Der Timer kann selbst-ständig Ausgangspins am Port B steuern, z.B. Toggle (0→1→0→1)

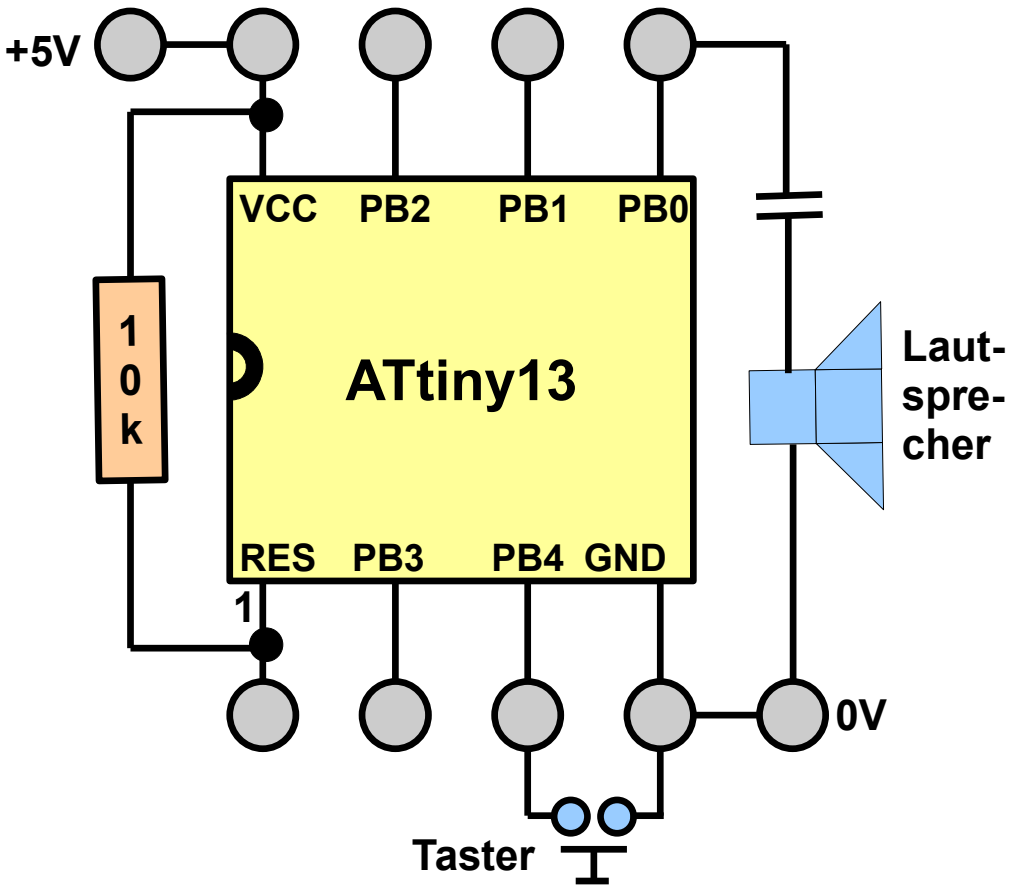
Programmierung CTC+Toggle OC0A:

```
ldi R16,(1<<COM0A0) | (1<<WGM01)
out TCCR0A,R16
```

COM0B1	COM0B0	Port B, OC0B=PB1
0	0	Normaler Portbetrieb, nicht verbunden
0	1	Toggle bei Compare-Match B
1	0	Nullsetzen bei Compare-Match B
1	1	Setzen bei Compare-Match B

Aufgabe 6: Sirene an PB0

- Schreibe ein Programm, das an PB0 den Kammerton a' ausgibt (880 Hz), wenn der Taster gedrückt ist, sonst Kammerton a (440 Hz).
- Verwende den Timer im CTC-Modus mit variabler Frequenz mit OC0A zur Ausgabe.
- Hinweise:
  - Vergiss nicht den Ausgangstreiber von PB0 einzuschalten!
  - Beim Programmieren muss der Lautsprecher abgestöpselt werden! (Warum?)
  - Wähle den Wert für den Vorteiler so klein wie möglich (Warum?).



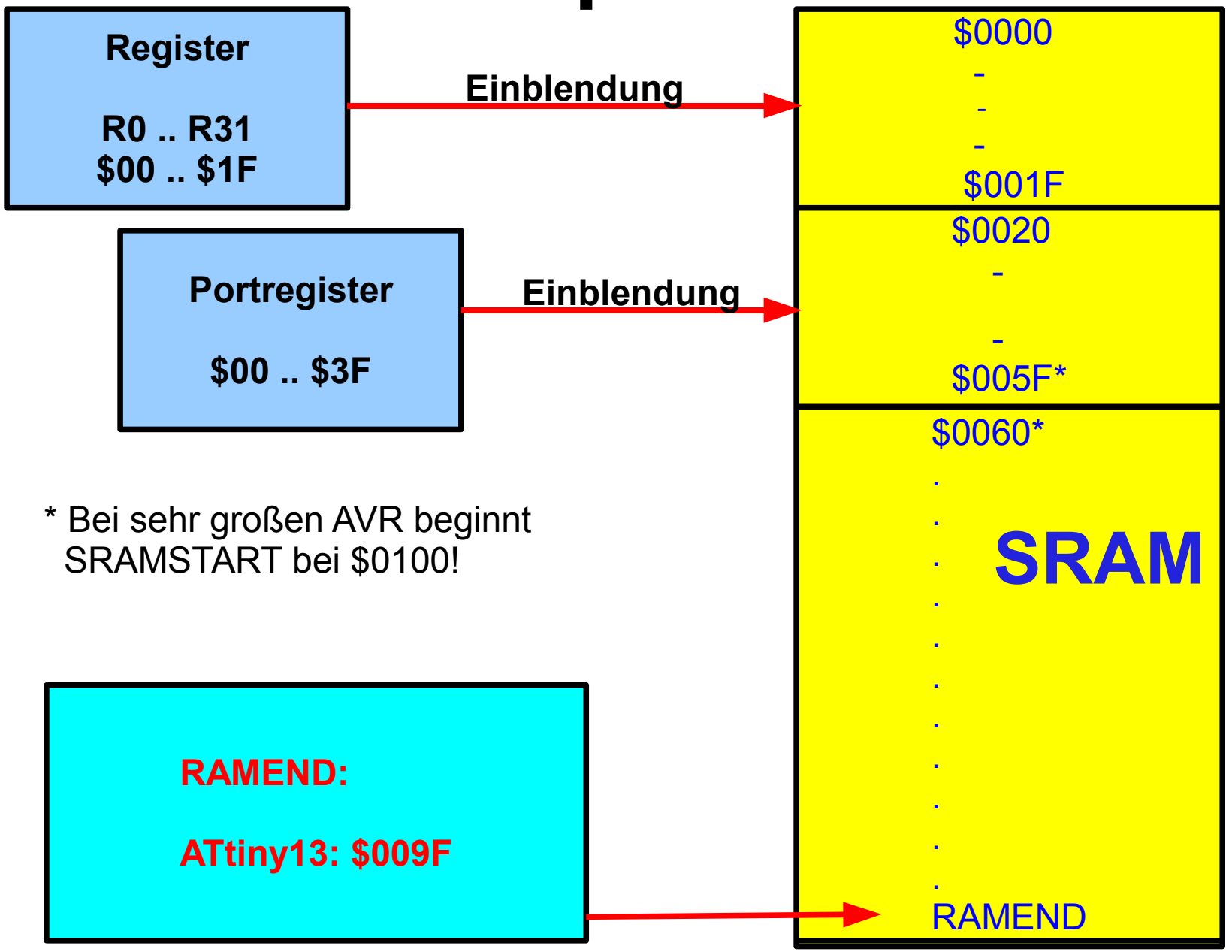
Interne Speicher : SRAM

- **SRAM: Tiny13 verfügt über 64 Bytes SRAM**
- **SRAM ist flüchtig, bei Spannungsverlust unter die Untergrenze der Betriebsspannung gehen die gespeicherten Daten verloren.**
- **Verwendung von SRAM:**
  - **Ablegen von Daten, für die die Register nicht ausreichen, Instruktionen:**
    - "sts adresse,register" ; speichert Registerinhalt im SRAM.
    - "lds register,adresse" ; liest SRAM-Datum in Register
  - **Anlegen von Datenpuffern, Manipulation mit Zeigern:**
    - "st [X|Y|Z],register" ; schreibt ins SRAM, Zeiger X, Y, Z gibt Adresse an
    - "ld register,[X|Y|Z]" ; liest aus SRAM, Zeiger X, Y, Z gibt Adresse an
    - statt "[X|Y|Z]" auch "[X+|Y+|Z+]" mit nachfolgendem Auto-Inkrement oder [-X|-Y|-Z] mit vorausgehendem Auto-Dekrement
    - mit Y und Z auch: "std Y+d,register" oder "ldd register,Y+d", mit d = 0..63 (d = Displacement)

Interne Speicher : SRAM

- **Verwendung von SRAM:**
  - **Ablegen von Daten, für die die Register nicht ausreichen:**
    - "sts adresse,register" ; speichert Registerinhalt im SRAM.
    - "lds register,adresse" ; liest SRAM-Datum in Register
  - **Anlegen von Datenpuffern, Manipulation mit Zeigern:**
    - "st [X|Y|Z],register" ; schreibt ins SRAM, Zeiger X, Y, Z gibt Adresse an
    - "ld register,[X|Y|Z]" ; liest aus SRAM, Zeiger X, Y, Z gibt Adresse an
    - statt "[X|Y|Z]" auch "[X+|Y+|Z+]" mit nachfolgendem Auto-Inkrement oder [-X|-Y|-Z] mit vorausgehendem Auto-Dekrement
    - mit Y und Z auch: "std Y+d,register" oder "ldd register,Y+d", mit d = 0..63 (d = Displacement)

Interne Speicher : SRAM



* Bei sehr großen AVR beginnt SRAMSTART bei $0100!

Interne Speicher : SRAM-Stapel

- **Verwendung von SRAM:**
  - **Einrichten eines LIFO-Stapels:**
    - ldi R16,LOW(RAMEND) ; lower Byte des SRAM-Endes
    - out SPL,R16 ; in lower Byte des Stapelzeigers
    - Bei >256 Byte SRAM: High-Byte des SRAM-Endes in SPH
  - **Ablegen von zeitweise nicht benötigten Registern:**
    - push register ; Ablegen eines Registers auf dem Stapel
    - pop register ; Holen des Registerinhalts vom Stapel
  - **Unterprogramme: Ablegen der Rücksprungadresse auf dem Stapel:**
    - rcall label ; legt Adresse+1 auf den Stapel und verzweigt nach label
    - ret ; Rückkehr nach Adresse+1, holt Adresse vom Stapel
  - **Interrupts: Ablegen der Rücksprungadresse auf dem Stapel:**
    - Unterbrechung legt Rücksprungadresse auf Stapel und verzweigt an fest eingestellte Interrupt-Adresse (Interrupt-Vektor)
    - reti ; beendet die Unterbrechung und kehrt zu vorheriger Adresse zurück

Interne Speicher: EEPROM

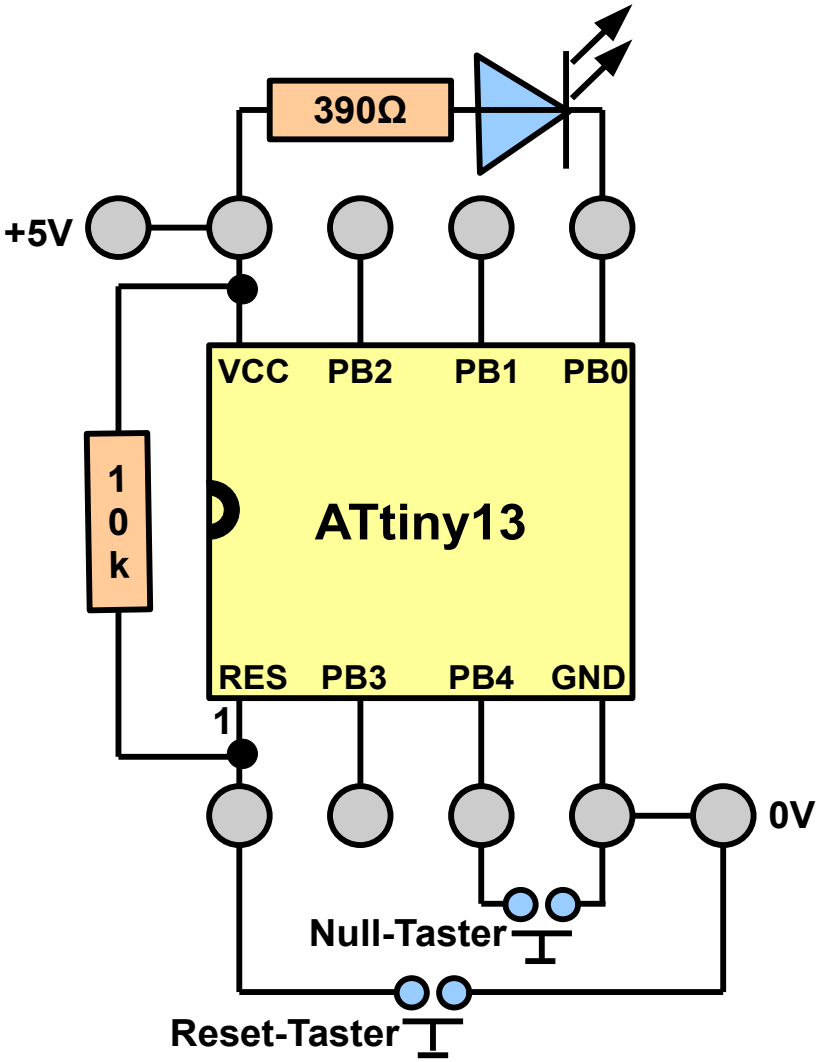
- **EEPROM: Nichtflüchtiges Speichern von Daten, bleiben auch bei Abschalten erhalten**
- **Bei Tiny13: 64 Bytes EEPROM verfügbar, 0x00..0x3F**
- **Lesen:**
  - **ldi R16,0x10 ; Adresse der EEPROM-Zelle in Register**
  - **out EEARL,R16 ; in EEPROM-Adressregister schreiben**
  - **sbi EECR,EERE ; setze Read Enable Bit**
  - **in R16,EEDR ; Lese Inhalt der Zelle in das Register**
- **Schreiben:**
  - **Warte: sbic EECR,EEPE ; frage Control Register ab ob Schreiben möglich**
  - **rjmp Warte ; Schreiben noch nicht möglich, warte**
  - **ldi R16,(0<<EEPM1) | (0<<EEPM0) ; Setze Schreibmodus = 00**
  - **out EECR,R16 ; in Kontrollregister**

Interne Speicher: EEPROM

- **Schreiben (Fortsetzung):**
  - **ldi R16,0x10 ; Adresse der EEPROM-Zelle in Register**
  - **out EEARL,R16 ; in EEPROM-Adressregister schreiben**
  - **ldi R16,0x01 ; Zu schreibendes Datum in Register schreiben**
  - **out EEDR,R16 ; in Datenregister**
  - **sbi EECR,EEMPE ; setze Enable Bit**
  - **sbi EECR,EEPE ; setze programming Enable**
- **Schreiben benötigt 3,4 ms**
- **Für sequentielles Schreiben in mehrere Speicherzellen: EEPROM kann Interrupt auslösen, wenn Schreiben einer Zelle beendet.**

Aufgabe 7: Einschaltzähler

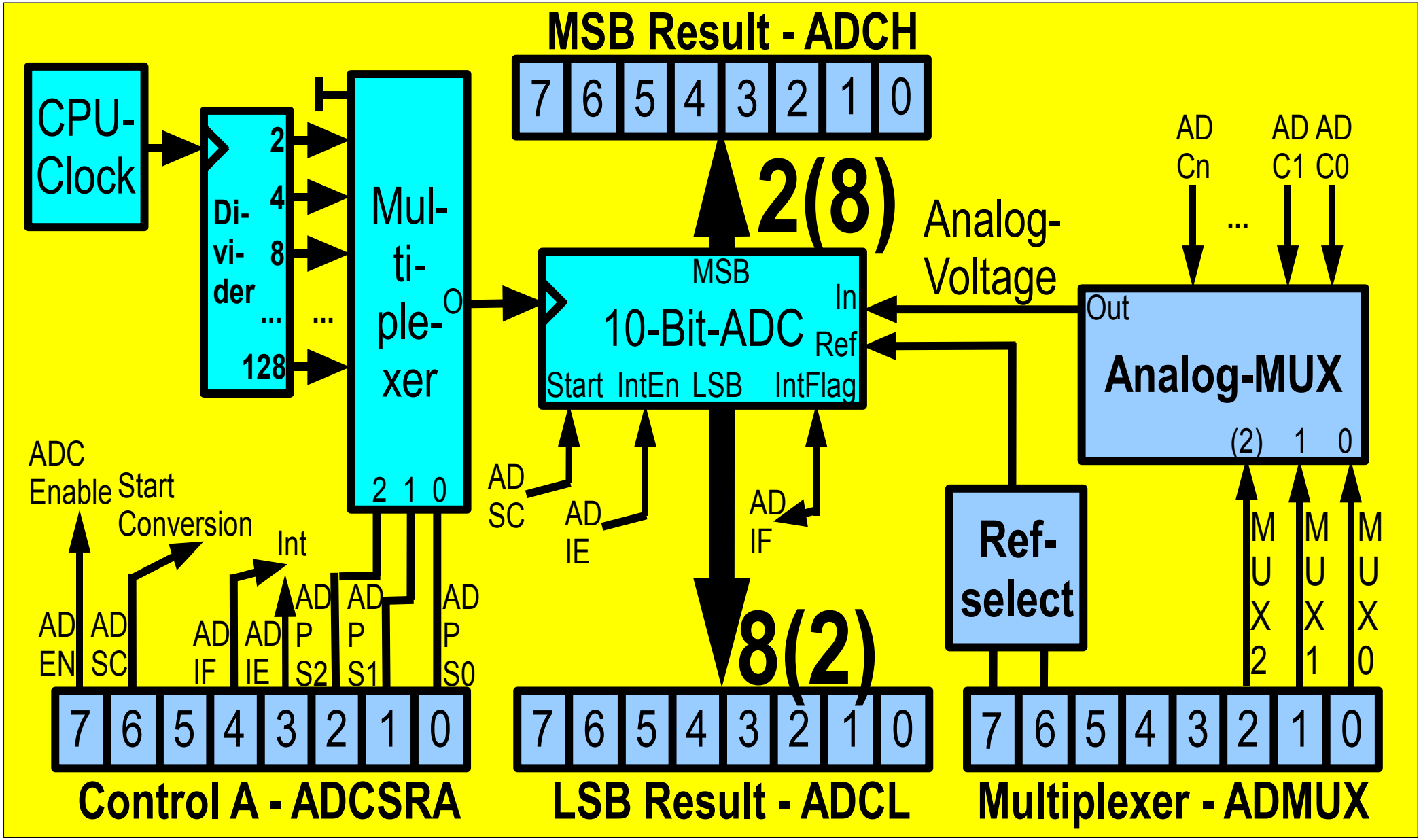
- Programmiere einen Zähler im EEPROM, der bei jedem Einschalten (Reset-Taster) um eins erhöht wird.
- Der Zählerstand soll auf Null gesetzt werden, wenn beim Einschalten die Taste an PB4 gedrückt ist.
- Der Zählerstand ist durch n-maliges Blinken der LED an PB0 auszugeben (0,1 Sekunden an und 0,2 Sekunden aus).
- Hinweise:
  - Zur Vermeidung von Prellen des Reset-Tasters ist zu Beginn eine 20 ms lange Pause einzulegen.
  - Verwende für die Verzögerungs-schleifen Unterprogramme.
  - Bitte Deinen Nachbarn um einen zweiten Taster.



10-Bit-AD-Wandler (ADC)

- **Ein ADC wandelt eine Analogspannung in eine 10 Bit breite Zahl um. Das Ergebnis steht in den Ports ADCH:ADCL. Der Kanal (Portpin) wird in ADMUX vorgewählt.**
- **Lesen des Ergebnisses: zuerst ADCL, danach ADCH! Ist das Bit ADLAR in ADMUX gesetzt, steht das Ergebnis linksbündig in den beiden Registern. Zum Lesen der oberen acht Bit reicht das Lesen von ADCH aus.**
- **Die Wandlung erfolgt aus einem Takt, der aus dem CPU-Takt durch Teilung abgeleitet ist (Bits ADPS2, ADPS1 und ADPS0 in ADCSRA).**
- **Jede Wandlung erfordert 25 (AD-Wandler war vorher abgeschaltet) bzw. 13 ADC-Takte.**
- **Der Vergleich erfolgt entweder mit der Betriebsspannung oder der eingebauten 1,1 V Spannungsreferenz.**

ADC: Interner Aufbau



Ansteuern des ADC

AD-Kanal auswählen:

```
ldi R16,(1<<ADLAR) | (1<<MUX1) | (1<<MUX0)
out ADMUX,R16 ; links adjust, AD-Kanal PB3
```

ADC starten und Taktrate wählen:

```
ldi R16,(1<<ADEN) | (1<<ADSC) | (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0)
out ADCSRA,R16
```

Warten bis Wandler fertig ist:

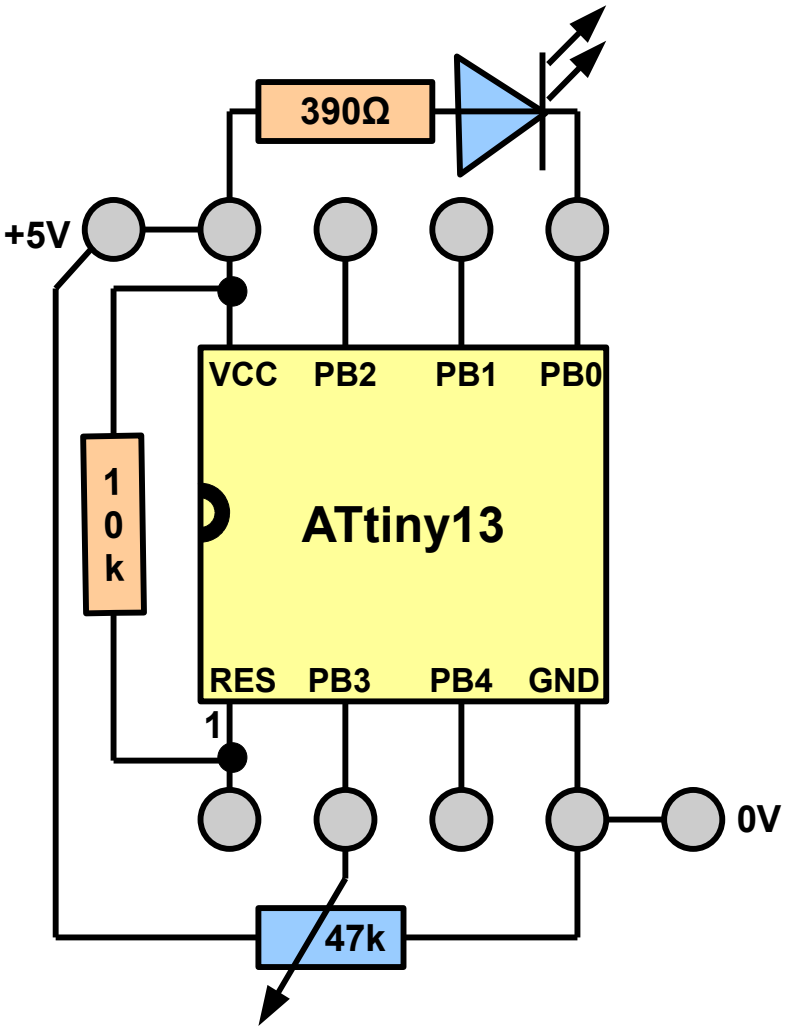
```
Warte:
  sbic ADCSRA,ADSC ; warte bis ADC fertig
  rjmp Warte ; noch nicht fertig
```

Ergebnis des Wandlers auslesen:

```
in R16,ADCH ; lese obere 8 Bit Ergebnis
```

Aufgabe 8: LED-Dimmer

- **Schreibe ein Programm, das den mit dem Poti an PB3 eingestellten Spannungswert misst und in ein pulsweiten-moduliertes Signal (PWM) an OC0A umsetzt und damit eine LED verschieden hell leuchten lässt.**
- **Hinweise:**
  - **Beachte, dass die LED bei längeren Null-Phasen heller leuchtet.**
  - **Timer als PWM: Verwende den Fast-PWM-Modus des Timers und eine möglichst hohe PWM-Frequenz.**
  - **ADC: Verwende den ADLAR-Modus des ADC (8-Bit) und die niedrigst-mögliche Abtastrate des ADC.**
  - **Da die PWM nur An (+5 V) und Aus (0 V) verwendet, ist die Durchlassspannung der LED (ca. 1,9 V) bei dieser Anwendung ohne Bedeutung!**

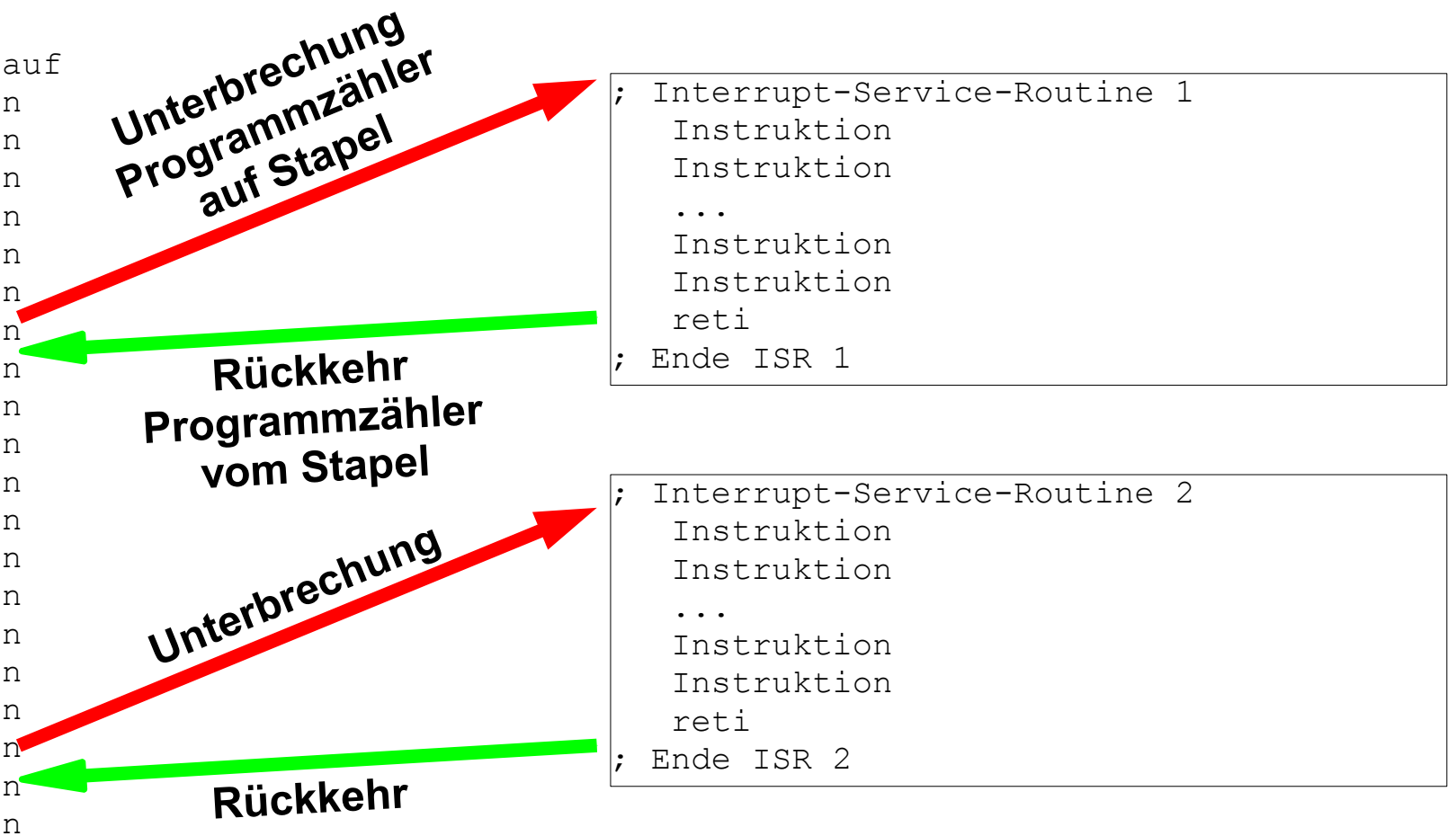


Interrupts

- **Interrupts sind asynchrone Unterbrechungen des normalen Programmablaufs. Sie signalisieren den Bedienbedarfs eines Geräts.**
- **Im auslösenden Gerät muss vorher der Interrupt ermöglicht werden (Interrupt-Enable-Bit, IE).**
- **Das I-Bit im Statusregister muss gesetzt sein (Instruktion SEI), damit die CPU Interrupts akzeptiert.**
- **Tritt der Interrupt ein, wird der aktuelle Programmzähler auf den Stapel gelegt und an eine gerätespezifische Adresse (Interrupt-Vektor) verzweigt. Dort muss zu einer Interrupt-Service-Routine (ISR) gesprungen werden. Ist der Interrupt bearbeitet, muss mit RETI wieder an den unterbrochenen Programmablauf zurückgekehrt werden.**
- **Während der ISR sind weitere Interrupts blockiert. ISRs müssen daher kurz sein. Mit RETI werden Interrupts wieder zugelassen.**
- **Treten zwei Interrupts gleichzeitig auf, wird der höherwertige Interrupt zuerst bearbeitet. Je niedriger die Vektoradresse, desto höher die Priorität.**

Interrupt-Ablauf

```
; Programmablauf
   Instruktion
   Instruktion
   Instruktion
   Instruktion
   Instruktion
   Instruktion
   Instruktion
   Instruktion
   Instruktion
   Instruktion
   Instruktion
   Instruktion
   Instruktion
   Instruktion
   Instruktion
   Instruktion
   Instruktion
   Instruktion
   Instruktion
   Instruktion
   Instruktion
   Instruktion
   Instruktion
   Instruktion
   Instruktion
   Instruktion
   Instruktion
   Instruktion
   Instruktion
   Instruktion
```




```
; Interrupt-Service-Routine 1
   Instruktion
   Instruktion
   ...
   Instruktion
   Instruktion
   reti
; Ende ISR 1
```

```
; Interrupt-Service-Routine 2
   Instruktion
   Instruktion
   ...
   Instruktion
   Instruktion
   reti
; Ende ISR 2
```

Interruptvektoren im Tiny13

Vector No.	Program Address	Source	Interrupt Definition
1	0x0000	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset
2	0x0001	INT0	External Interrupt Request 0
3	0x0002	PCINT0	Pin Change Interrupt Request 0
4	0x0003	TIM0_OVF	Timer/Counter Overflow
5	0x0004	EE_RDY	EEPROM Ready
6	0x0005	ANA_COMP	Analog Comparator
7	0x0006	TIM0_COMPA	Timer/Counter Compare Match A
8	0x0007	TIM0_COMPB	Timer/Counter Compare Match B
9	0x0008	WDT	Watchdog Time-out
10	0x0009	ADC	ADC Conversion Complete

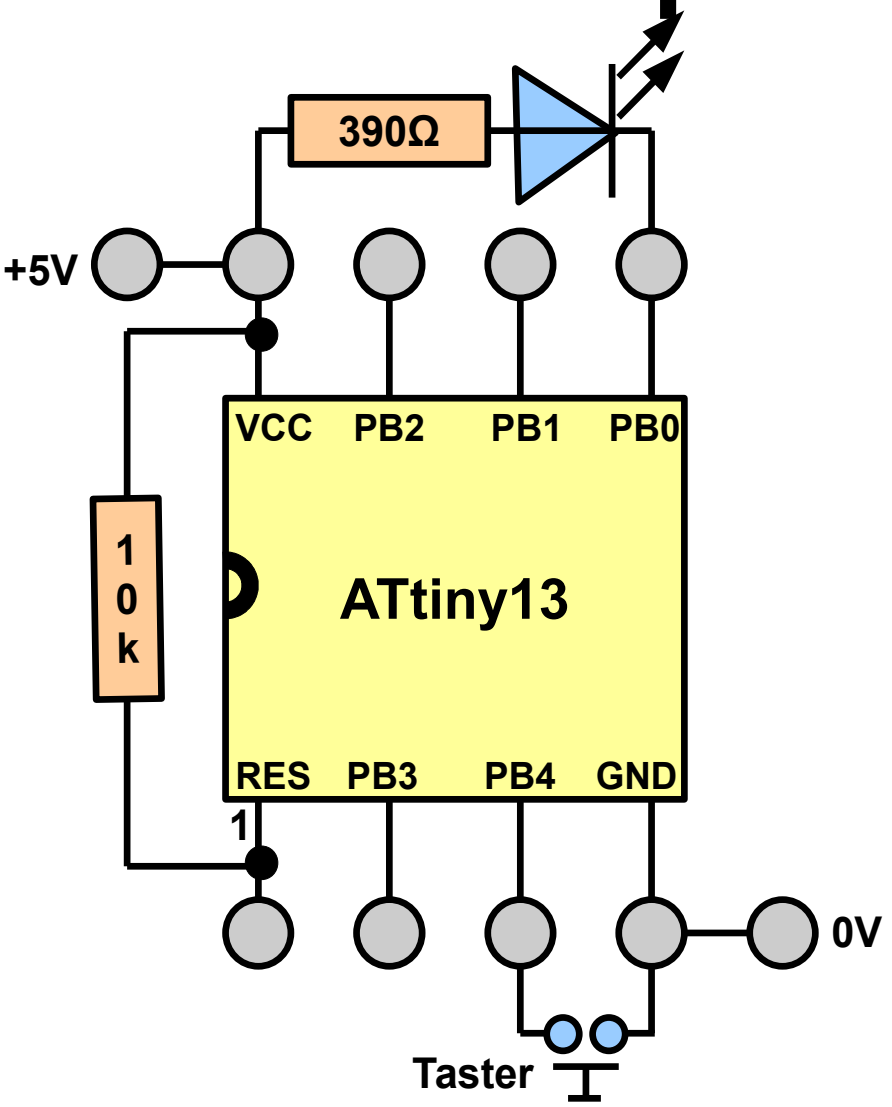
Interrupts, Beispiel

```
; Timer-Interrupt mit Blinker
; Blinkt drei Sekunden
;
.nolist
.include "tn13def.inc"
.list
;
; Register
.def rsreg = R15 ; Sichern des SREG
.def rmp = R16 ; Multipurpose
.def rimp = R17 ; innerhalb ints
.def cnt = R18 ; Zähler für Ints
.def inv = R19 ; Invertiermaske LED
;
; Reset- und Interruptvektoren
   rjmp Init ; Reset-Vektor
   reti ; INT0-Vektor, nicht benutzt
   reti ; PCINT0-Vektor, nicht benutzt
   rjmp TimerIsr ; Timer-Overflow
   reti ; EERDY-Vektor, nicht benutzt
   reti ; ANACOMP-Vektor, nicht benutzt
   reti ; TIM0COMPA-Vektor, nicht benutzt
   reti ; TIM0COMPB-Vektor, nicht benutzt
   reti ; WDT-Vektor, nicht benutzt
   reti ; ADC-Vektor, nicht benutzt
;
```

```
Init:
   ldi rmp,LOW(RAMEND) ; Stapel init
   out SPL,rmp ; Stapel setzen
   ldi cnt,10 ; Zähler setzen
   ldi inv,0x01 ; PB0 invertieren
   sbi DDRB,PB0 ; LED-Ausgang
   ldi rmp,1<<TOIE0 ; Enable Overfl Ints
   out TIMSK0,rmp
   ldi rmp,(1<<CS02) | (1<<CS00) ; 1024
   out TCCR0B,rmp ; Vorteiler
   sei ; enable ints
loop:
   rjmp loop
;
TimerIsr: ; Timer-Interrupts
   in rsreg,SREG ; sichere Status
   dec cnt ; Zähler vermindern
   brne TimerIsrRet ; nicht Null
   ldi cnt,10 ; Zähler neu starten
   in rimp,PORTB ; lese Port
   eor rimp,inv ; LED invertieren
   out PORTB,rimp ; und in Port zurück
TimerIsrRet:
   out SREG,rsreg ; Herstellen Status
   reti ; Rückkehr
;
```

Aufgabe 9: Taster mit Interrupt

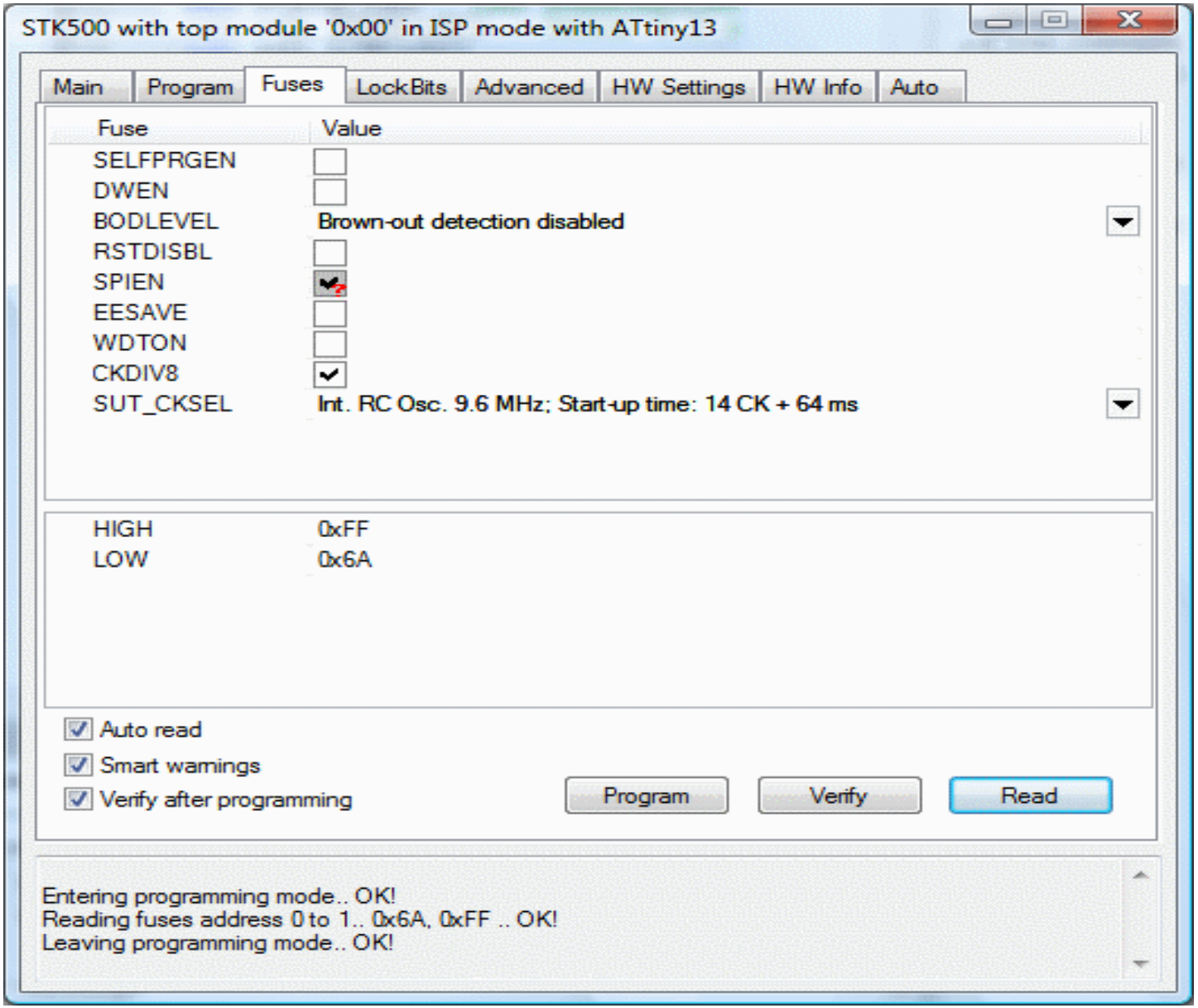
- Schreibe ein Programm, das interruptgesteuert bei gedrückter Taste die LED timer-gesteuert drei Mal kurz leuchten lässt.
- Hinweise:
  - Verwende den PCINT0-Vektor zum Erkennen der Tasteränderungen.
  - Verwende den TIM0-Overflow-Vektor zum Blinken.



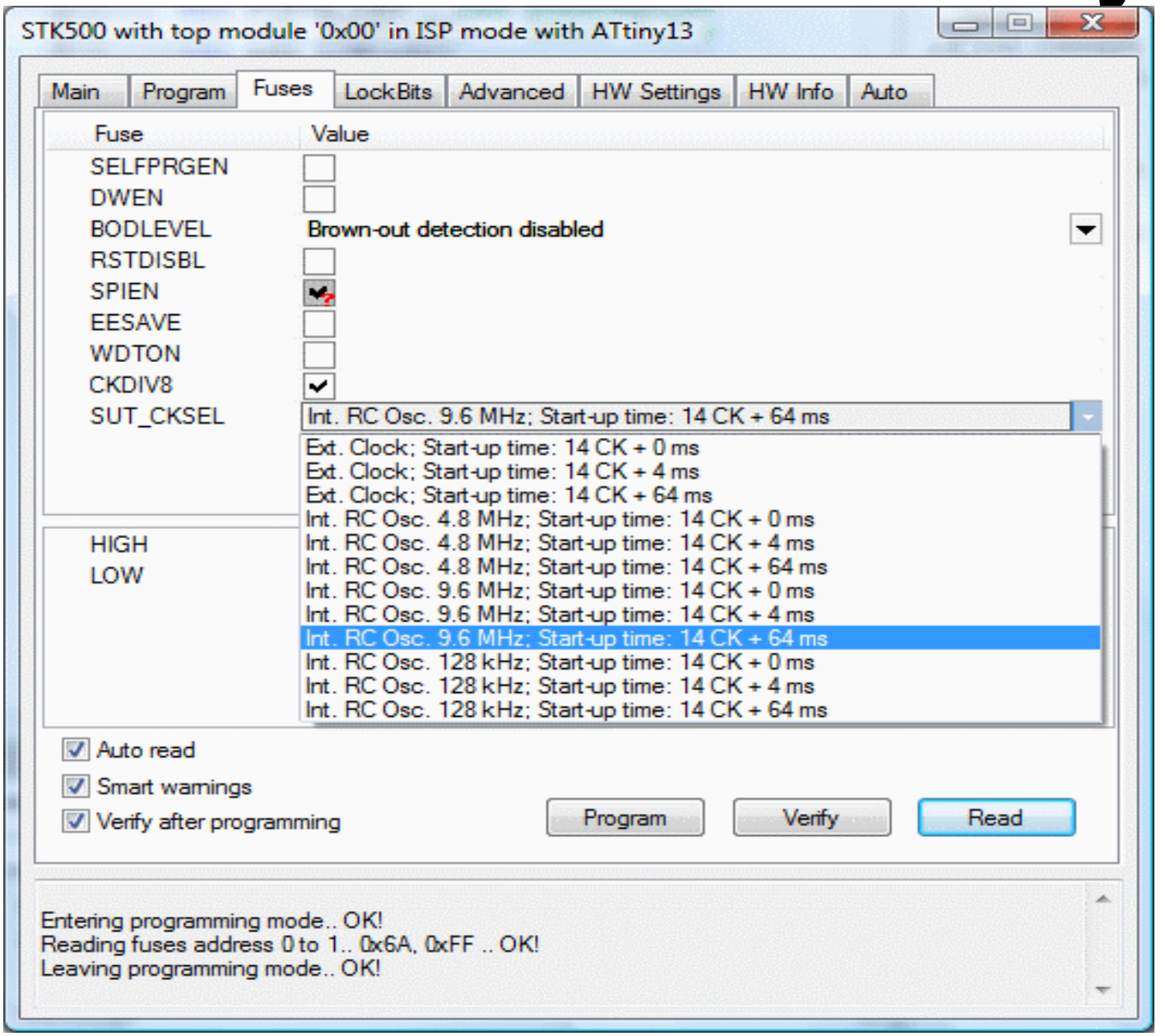
Fuse-Programmierung

- **Fuses sind Speicherzellen, die beim Programmieren gesetzt werden können und die das Hardware-Verhalten des Prozessors beeinflussen, z. B.**
  - **welche Taktquelle verwendet wird (interner RC-Oszillator, externer Oszillator, externer Quarz. Achtung! Ohne angeschlossenen externen Oszillator oder Quarz lassen sich dann keine weiteren Programmierversuche mehr durchführen!),**
  - **ob ein interner Teiler für den Systemtakt (CKDIV8) ein- oder ausgeschaltet ist (Ausschalten macht Prozessor acht Mal schneller, steigert aber überproportional den Stromverbrauch!),**
  - **ob der RESET-Pin aktiv sein soll oder stattdessen ein Porteingang sein soll (Achtung! AVR ist danach nur noch mit Hochvolt-Programmierung zugänglich! Erfordert +12 V am RESET-Eingang! Nur hochwertige Programmiergeräte beherrschen dies (STK500, AVR-Dragon, etc.)),**
  - **ob das Lesen und Verifizieren (Kopierschutz) oder das Überschreiben des programmierten Speichers (Schreibschutz) noch möglich sein soll (Lock-Bits) oder unter Löschen des Speicherinhalts nur noch per "ERASE DEVICE" ansprechbar sein soll.**

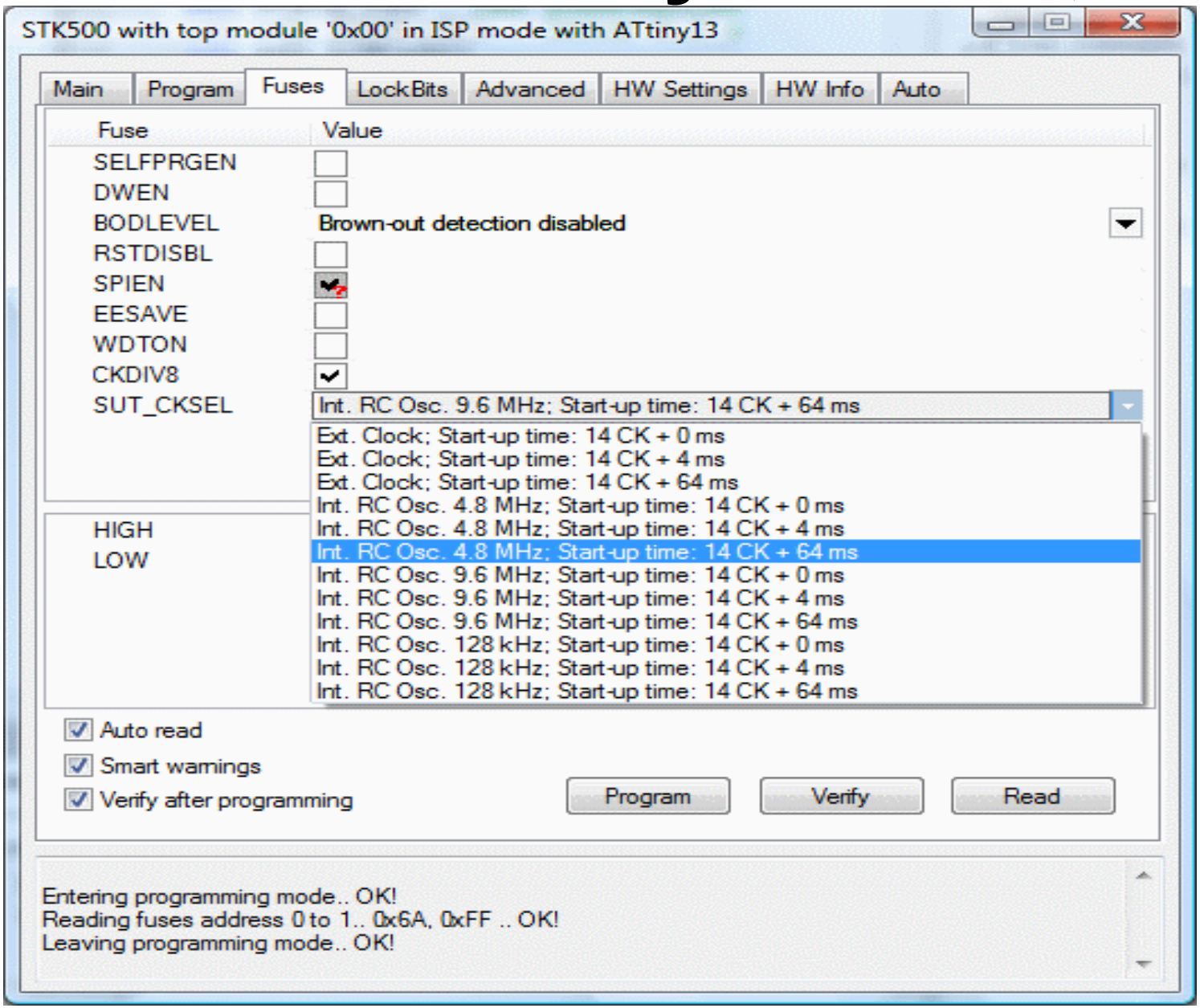
Das Fuse-Fenster beim Tiny13



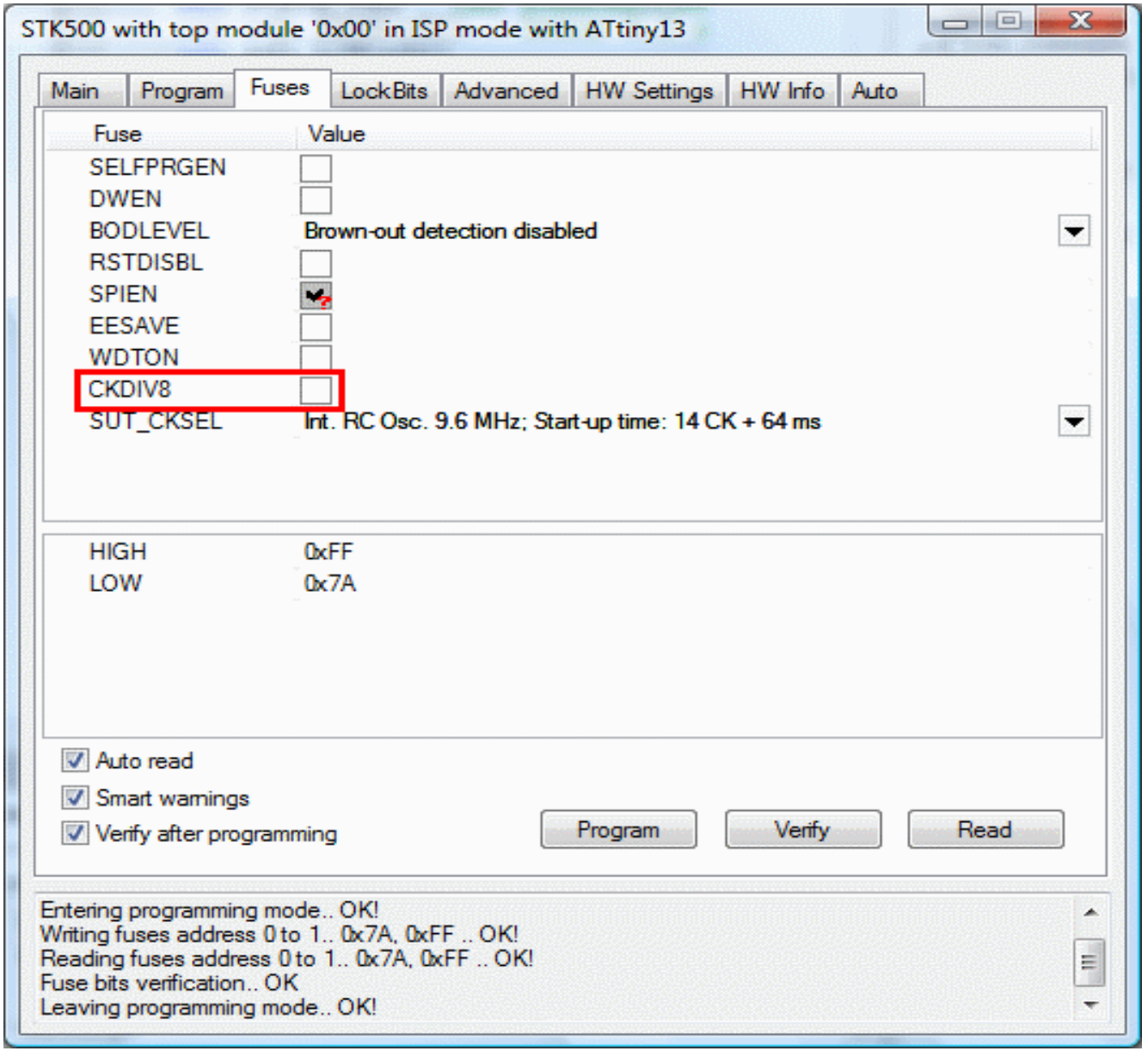
Die Clock-Fuses beim Tiny13



Umstellen des Tiny13 auf 4,8 MHz



Ausschalten von CKDIV8



Die Lock-Bits des Tiny13