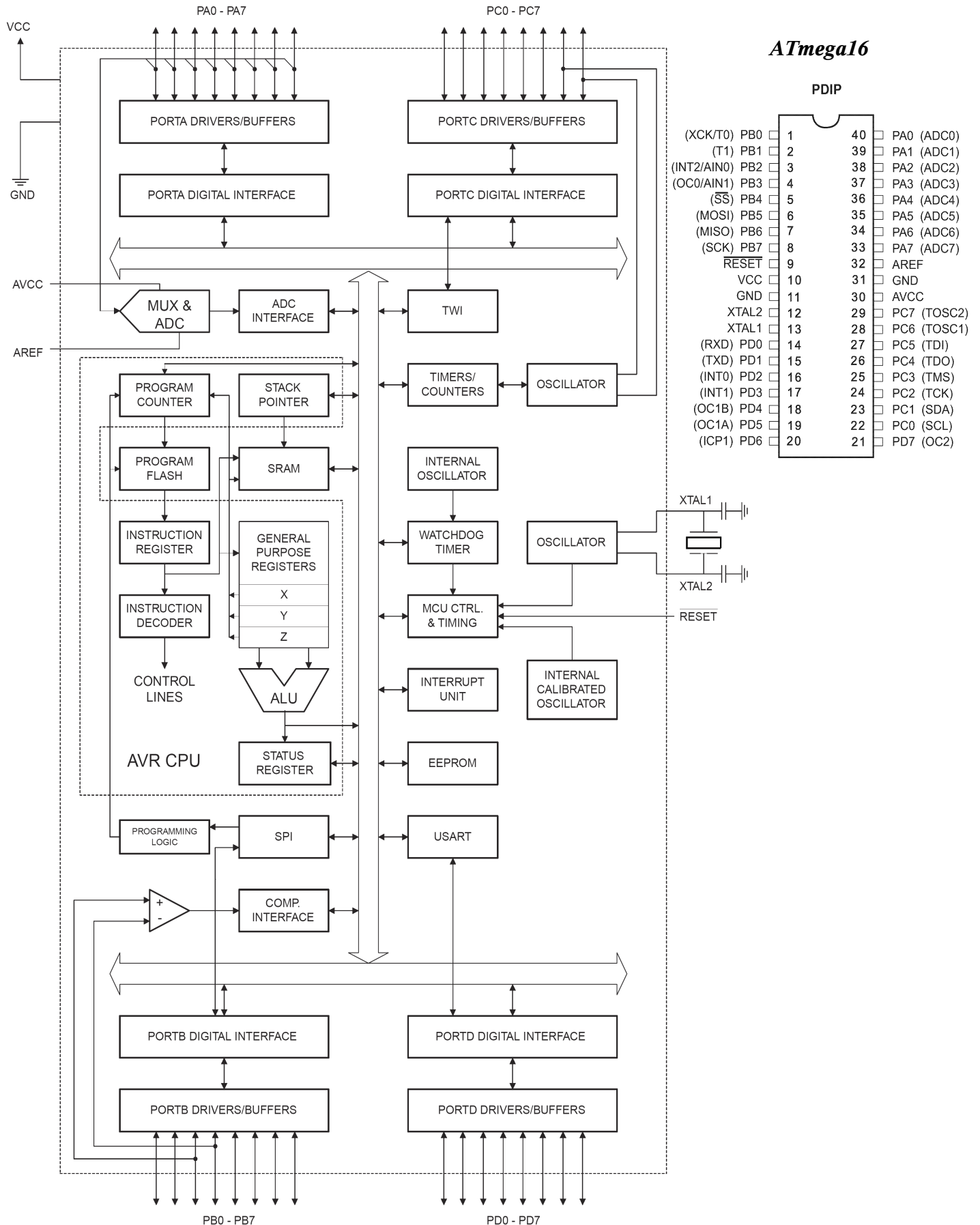


Mikrocontroller ATmega16

Übersichtsschaltbild und Anschlussbelegung



Befehlsliste ATmega16
Status register (SREG)

C : Carry flag in status register
Z : Zero flag in status register
N : Negative flag in status register
V : Two's complement overflow indicator
S : N [+] V, For signed tests
H : Half Carry flag in the status register
T : Transfer bit used by BLD and BST instructions
I : Global interrupt enable/disable flag

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
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	

Registers and Operands

Rd : Destination (and source) register in the register file
Rr : Source register in the register file
R : Result after instruction is executed
K : Constant data
k : Constant address
b : Bit in the register file or I/O register (3 bit)
s : Bit in the status register (3 bit)
X,Y,Z : Indirect address register (X=R27:R26, Y=R29:R28 and Z=R31:R30)
A : I/O location address
q : Displacement for direct addressing (6 bit → q = 0 ... 63)

Instruction Set Summary

Hinweise: Immediate-Befehle (z. B. **LDI R16, 0x5A**) nur möglich mit den Registern **R16 ... R31**

Befehl **ADIW** nur mit den Registern **R25:R24, R27:R26 (X-Reg), R29:R28 (Y-Reg), R31:R30 (Z-Reg)**

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	$Rd \leftarrow Rd + Rr$	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	$Rd \leftarrow Rd + Rr + C$	Z,C,N,V,H	1
ADIW	RdI,K	Add Immediate to Word	$RdH:RdL \leftarrow RdH:RdL + K$	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	$Rd \leftarrow Rd - Rr$	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	$Rd \leftarrow Rd - K$	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	$Rd \leftarrow Rd - Rr - C$	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	$Rd \leftarrow Rd - K - C$	Z,C,N,V,H	1
SBIW	RdI,K	Subtract Immediate from Word	$RdH:RdL \leftarrow RdH:RdL - K$	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	$Rd \leftarrow Rd \cdot Rr$	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	$Rd \leftarrow Rd \cdot K$	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	$Rd \leftarrow Rd \vee Rr$	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	$Rd \leftarrow Rd \vee K$	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	$Rd \leftarrow Rd \oplus Rr$	Z,N,V	1
COM	Rd	One's Complement	$Rd \leftarrow \$FF - Rd$	Z,C,N,V	1
NEG	Rd	Two's Complement	$Rd \leftarrow \$00 - Rd$	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	$Rd \leftarrow Rd \vee K$	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	$Rd \leftarrow Rd \cdot (\$FF - K)$	Z,N,V	1
INC	Rd	Increment	$Rd \leftarrow Rd + 1$	Z,N,V	1
DEC	Rd	Decrement	$Rd \leftarrow Rd - 1$	Z,N,V	1
TST	Rd	Test for Zero or Minus	$Rd \leftarrow Rd \cdot Rd$	Z,N,V	1
CLR	Rd	Clear Register	$Rd \leftarrow Rd \oplus Rd$	Z,N,V	1
SER	Rd	Set Register	$Rd \leftarrow \$FF$	None	1
MUL	Rd, Rr	Multiply Unsigned	$R1:R0 \leftarrow Rd \times Rr$	Z,C	2
MULS	Rd, Rr	Multiply Signed	$R1:R0 \leftarrow Rd \times Rr$	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	$R1:R0 \leftarrow Rd \times Rr$	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	$R1:R0 \leftarrow (Rd \times Rr) \ll 1$	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	$R1:R0 \leftarrow (Rd \times Rr) \ll 1$	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	$R1:R0 \leftarrow (Rd \times Rr) \ll 1$	Z,C	2
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	$PC \leftarrow PC + k + 1$	None	2
JMP		Indirect Jump to (Z)	$PC \leftarrow Z$	None	2
JMP	k	Direct Jump	$PC \leftarrow k$	None	3
RCALL	k	Relative Subroutine Call	$PC \leftarrow PC + k + 1$	None	3
ICALL		Indirect Call to (Z)	$PC \leftarrow Z$	None	3
CALL	k	Direct Subroutine Call	$PC \leftarrow k$	None	4
RET		Subroutine Return	$PC \leftarrow STACK$	None	4
RETI		Interrupt Return	$PC \leftarrow STACK$	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) $PC \leftarrow PC + 2$ or 3	None	1 / 2 / 3
CP	Rd,Rr	Compare	$Rd - Rr$	Z, N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	$Rd - Rr - C$	Z, N,V,C,H	1
CPI	Rd,K	Compare Register with Immediate	$Rd - K$	Z, N,V,C,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) $PC \leftarrow PC + 2$ or 3	None	1 / 2 / 3
SBRs	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) $PC \leftarrow PC + 2$ or 3	None	1 / 2 / 3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) $PC \leftarrow PC + 2$ or 3	None	1 / 2 / 3
SBSI	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) $PC \leftarrow PC + 2$ or 3	None	1 / 2 / 3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then $PC \leftarrow PC + k + 1$	None	1 / 2
BREQ	k	Branch if Equal	if (Z = 1) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRNE	k	Branch if Not Equal	if (Z = 0) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRCS	k	Branch if Carry Set	if (C = 1) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRCC	k	Branch if Carry Cleared	if (C = 0) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRSH	k	Branch if Same or Higher	if (C = 0) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRLO	k	Branch if Lower	if (C = 1) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRMI	k	Branch if Minus	if (N = 1) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRPL	k	Branch if Plus	if (N = 0) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRGE	k	Branch if Greater or Equal, Signed	if (N \oplus V = 0) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRLT	k	Branch if Less Than Zero, Signed	if (N \oplus V = 1) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRTS	k	Branch if T Flag Set	if (T = 1) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then $PC \leftarrow PC + k + 1$	None	1 / 2

Mnemonics	Operands	Description	Operation	Flags	#Clocks
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1 / 2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1 / 2
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd, Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z + 1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q, Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q, Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z + 1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
IN	Rd, P	In Port	Rd ← P	None	1
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2
BIT AND BIT-TEST INSTRUCTIONS					
SBI	P, b	Set Bit in I/O Register	I/O(P, b) ← 1	None	2
CBI	P, b	Clear Bit in I/O Register	I/O(P, b) ← 0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z, C, N, V	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z, C, N, V	1
ROL	Rd	Rotate Left Through Carry	Rd(0) ← C, Rd(n+1) ← Rd(n), C ← Rd(7)	Z, C, N, V	1
ROR	Rd	Rotate Right Through Carry	Rd(7) ← C, Rd(n) ← Rd(n+1), C ← Rd(0)	Z, C, N, V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z, C, N, V	1
SWAP	Rd	Swap Nibbles	Rd(3..0) ← Rd(7..4), Rd(7..4) ← Rd(3..0)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Twos Complement Overflow	V ← 1	V	1
CLV		Clear Twos Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1

Mnemonics	Operands	Description	Operation	Flags	#Clocks
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
MCU CONTROL INSTRUCTIONS					
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	1
WDR		Watchdog Reset	(see specific descr. for WDR/timer)	None	1
BREAK		Break	For On-Chip Debug Only	None	N/A

Ergänzungen:

Division

Der Mikrokontrollertyp AVR hat keinen implementierten Befehl zur Division von 8-Bit-Zahlen, deshalb das Makro **mdivx8 r1, r0** verwenden. Alle Register mit Ausnahme von r2 und r3 möglich. Wirkung des Makros: Division **r1/r0** wird ausgeführt, **r0** enthält Quotient, **r1** den Divisionsrest.

Addition Inhalt eines 8-Bit-Registers zu Doppelregister (Doppelregister X, Y oder Z)

Beispiel: der Inhalt von Register r16 (8 Bit) wird zum Inhalt von Doppelregister Z (16 Bit) addiert

```
add    ZL,r16
clr    r16      ; r16 <- 0, Carry-Flag unverändert!
adc    ZH,r16    ; Null + Übertrag
```

Assembler-Direktiven

Direktive	Operand	Anwendung	Beispiel
.include	“Dateiname.Typ“	Assembler liest Inhalt der Datei	.include“m16def.inc“
.def	Bezeichner = Register	Für Register r0 ... r31	.def temp = r16
.equ	Bezeichner = Ausdruck	Unveränderliche Definition	.equ anf_wert = 0xff
.set	Bezeichner = Ausdruck	Veränderliche Definition	.set wert = 123 ; alt .set wert = 89 ; neu
.org	Ausdruck	Legt Adresse fest	.org 0x100

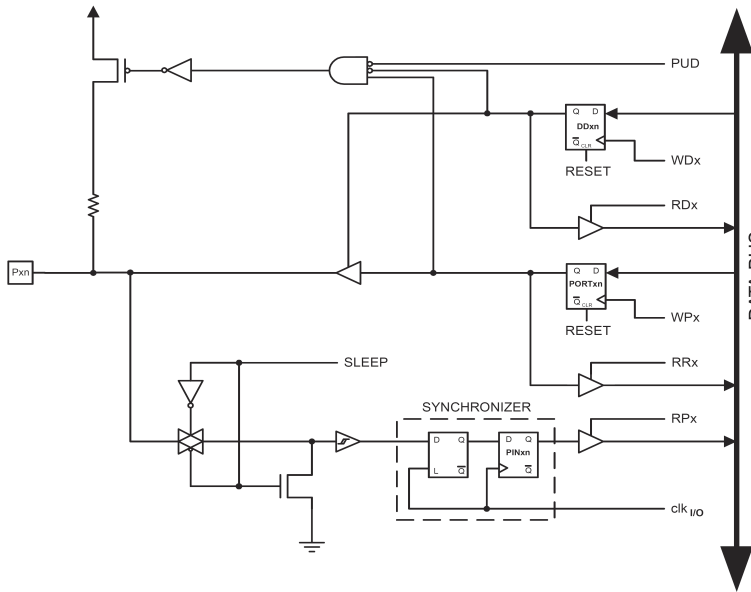
Speicher-Direktiven

Direktive	Operand	Anwendung	Beispiel
.db	Liste mit Bytekonstanten	Konstanten im Flash (oder EEPROM), z. B. Tabelle	Byte_tab: .db 0x41 .db 'A'
.dw	Liste mit Wortkonstanten	16-Bit-Worte im Flash (oder EEPROM)	Wort_tab: .dw 0x1234
.byte	Anzahl n	Reserviert Bytes in SRAM (EEPROM)	Werte: .byte 10 ;10 Bytes
.cseg		Programm im Flash (Code Segment)	.cseg
.dseg		Variablen im SRAM (Data Segment)	.dseg

Assembler-Funktionen

Funktion	Wirkung	Beispiel
LOW (Ausdruck)	Liefert Bit 7 ... Bit 0, also Low-Byte	Low (0x045f) gibt 0x5f Low (RAMEND) gibt 0x5f
HIGH (Ausdruck)	Liefert Bit 15 ... Bit 8, also High-Byte	High (0x045f) gibt 0x04 High (RAMEND) gibt 0x04

I/O-Ports A, B, C, D



Prinzipschaltung für einen Anschluss eines 8-Bit I/O-Ports

Konfiguration eines Pins eines I/O-Ports

x steht für Port A, B, C, D; n steht für Bit 7 ... Bit 0.

DDxn	PORTxn	I/O	Pullup-R	Bemerkung
0	0	IN	Nein	Hochohmig
0	1	IN	Ja	High (Pullup-R)
1	0	OUT	Nein	Low
1	1	OUT	Nein	High

Beispiel zur Konfiguration des Port B:
PB2 als Ausgang, PB1 und PB0 als Eingänge mit Pullup-Widerständen.

Assemblersprache

```
ldi r16, 0b00000011 ; Pullup-R konfigur.
out PORTB, r16
ldi r16, 0b00000100 ; Datenrichtung konfigur.
out DDRB, r16
```

Hochsprache C (GCC)

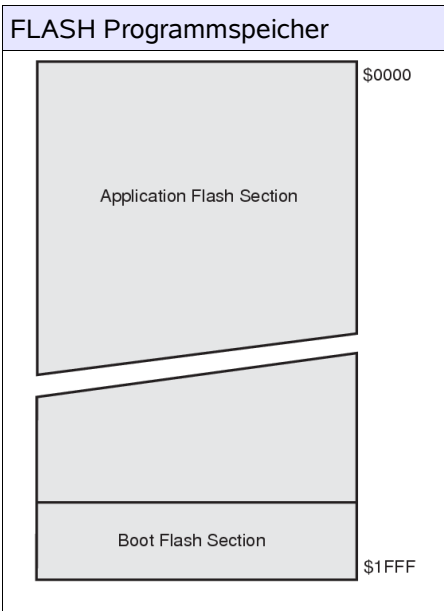
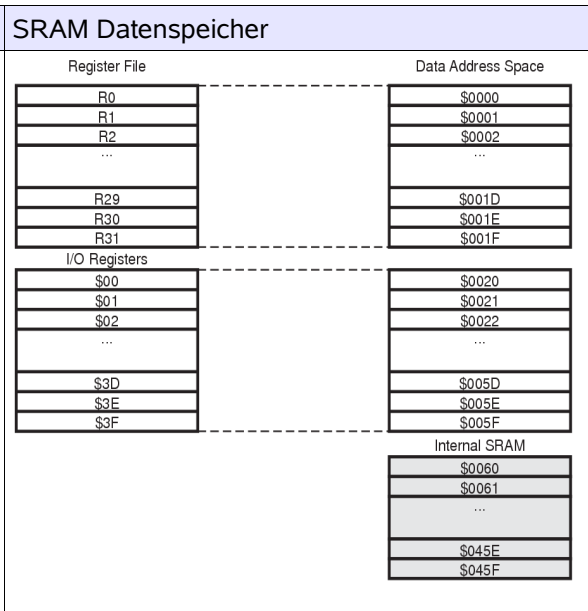
```
PORTB = (1<<PB1)|(1<<PB0); // Pullup-R konfigur.
DDRB = (1<<DDB2); // Datenrichtung konfigur.
```

Befehle für I/O-Ports

Hinweis: Operand **A** gibt I/O-Adresse an, sie kann absolut oder symbolisch sein. Beisp.: A = \$18 = 0x18 = PORTB

Befehl	Operand	Beschreibung	Beispiel	Kommentar zum Beispiel (s. auch Schaltplan oben)
IN	Rd, A	Lesen I/O-Reg. A, Ziel ist Register	in r16, PIND	Pegel am Eingaberegister PIND des Port D lesen und nach r16 kopieren
OUT	A, Rd	Ausgabe Registerinhalt an I/O-Reg. A	out PORTB, r16	Inhalt von Reg. r16 an I/O-Reg. PORTB ausgeben
SBI	A, b	Setzen von Bit b in I/O-Reg A	sbi PORTB, 2	Flipflop PORTB2 des Port B setzen
CBI	A, b	Löschen von Bit b in I/O-Reg A	cbi PORTB, 2	Flipflop PORTB2 des Port B löschen
SBIC	A, b	Skip, wenn Bit b in I/O-Reg. A gelöscht	sbic PIND, 1	Wenn Flipflop PIND1 = 0 wird der nachfolgende Befehl übersprungen
SBIS	A, b	Skip, wenn Bit b in I/O-Reg. A gesetzt	sbis PIND, 1	wenn Flipflop PIND1 = 1 wird der nachfolgende Befehl übersprungen

Speicher

FLASH Programmspeicher	SRAM Datenspeicher	Bemerkungen
		<p>FLASH: Adr. 0 ... 0x1FFF: 16 KByte, organisiert in 8 K Worten zu je 16 Bit,</p> <p>SRAM: Byteweise organisiert.</p> <ul style="list-style-type: none"> Adr. 0 ... 0x1F: Register r16 ... r31 Adr. 0x20 ... 0x5F: I/O-Register (s. auch Register Summary) Adr. 0x60 ... 0x45F: 1 KByte Datenspeicher <p>EEPROM (wird hier nicht behandelt): Byteweise organisiert. Spezielle Befehle.</p>

Assemblersprache	Hochsprache C (GCC)
<pre>ldi XL, LOW(tab) ;Tabellenzeiger ldi XH, HIGH(tab) ldi R24, 2 ;erster Wert st X+, R24 ldi R24, 4 ;zweiter Wert st X+, R24 .dseg ; Daten im SRAM tab: .byte 2 ; 2 Byte reserviert</pre>	<pre>unsigned char tab[2] = {2,4};</pre>

Liste der I/O-Register im Data Address Space (SRAM)

Register Summary

Quelle: ATMEL, Datenblatt ATmega16 2466G-AVR-10/03 Seite 329 f.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
\$3F (\$5F)	SREG	I	T	H	S	V	N	Z	C
\$3E (\$5E)	SPH	–	–	–	–	–	SP10	SP9	SP8
\$3D (\$5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
\$3C (\$5C)	OCR0	Timer/Counter0 Output Compare Register							
\$3B (\$5B)	GICR	INT1	INT0	INT2	–	–	–	IVSEL	IVCE
\$3A (\$5A)	GIFR	INTF1	INTF0	INTF2	–	–	–	–	–
\$39 (\$59)	TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0
\$38 (\$58)	TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
\$37 (\$57)	SPMCR	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN
\$36 (\$56)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
\$35 (\$55)	MCUCR	SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00
\$34 (\$54)	MCUCSR	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF
\$33 (\$53)	TCCR0	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
\$32 (\$52)	TCNT0	Timer/Counter0 (8 Bits)							
\$31 ⁽¹⁾ (\$51) ⁽¹⁾	OSCCAL	Oscillator Calibration Register							
	OCDR	On-Chip Debug Register							
\$30 (\$50)	SFIOR	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10
\$2F (\$4F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10
\$2E (\$4E)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10
\$2D (\$4D)	TCNT1H	Timer/Counter1 – Counter Register High Byte							
\$2C (\$4C)	TCNT1L	Timer/Counter1 – Counter Register Low Byte							
\$2B (\$4B)	OCR1AH	Timer/Counter1 – Output Compare Register A High Byte							
\$2A (\$4A)	OCR1AL	Timer/Counter1 – Output Compare Register A Low Byte							
\$29 (\$49)	OCR1BH	Timer/Counter1 – Output Compare Register B High Byte							
\$28 (\$48)	OCR1BL	Timer/Counter1 – Output Compare Register B Low Byte							
\$27 (\$47)	ICR1H	Timer/Counter1 – Input Capture Register High Byte							
\$26 (\$46)	ICR1L	Timer/Counter1 – Input Capture Register Low Byte							
\$25 (\$45)	TCCR2	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20
\$24 (\$44)	TCNT2	Timer/Counter2 (8 Bits)							
\$23 (\$43)	OCR2	Timer/Counter2 Output Compare Register							
\$22 (\$42)	ASSR	–	–	–	–	AS2	TCN2UB	OCR2UB	TCR2UB
\$21 (\$41)	WDTCR	–	–	–	WDTOE	WDE	WDP2	WDP1	WDP0
\$20 ⁽²⁾ (\$40) ⁽²⁾	UBRRH	URSEL	–	–	–	UBRR[11:8]			
	UCSRC	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL
\$1F (\$3F)	EEARH	–	–	–	–	–	–	–	EEAR8
\$1E (\$3E)	EEARL	EEPROM Address Register Low Byte							
\$1D (\$3D)	EEDR	EEPROM Data Register							
\$1C (\$3C)	EECR	–	–	–	–	EERIE	EEMWE	EERE	EERE
\$1B (\$3B)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0
\$1A (\$3A)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0
\$19 (\$39)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0
\$18 (\$38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
\$17 (\$37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
\$16 (\$36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
\$15 (\$35)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
\$14 (\$34)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
\$13 (\$33)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
\$12 (\$32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
\$11 (\$31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
\$10 (\$30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
\$0F (\$2F)	SPDR	SPI Data Register							
\$0E (\$2E)	SPSR	SPIF	WCOL	–	–	–	–	–	SPI2X
\$0D (\$2D)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
\$0C (\$2C)	UDR	USART I/O Data Register							
\$0B (\$2B)	UCSRA	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM
\$0A (\$2A)	UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8
\$09 (\$29)	UBRRL	USART Baud Rate Register Low Byte							
\$08 (\$28)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0
\$07 (\$27)	ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0
\$06 (\$26)	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
\$05 (\$25)	ADCH	ADC Data Register High Byte							
\$04 (\$24)	ADCL	ADC Data Register Low Byte							
\$03 (\$23)	TWDR	Two-wire Serial Interface Data Register							
\$02 (\$22)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
\$01 (\$21)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWSP1	TWSP0
\$00 (\$20)	TWBR	Two-wire Serial Interface Bit Rate Register							

Interrupt

Vektortabelle ATmega16 mit Interrupt-Definition für Assemblersprache bzw. Bezeichner für C (GCC)

	Program Address	Source	Interrupt Definition	Assembler	Hochsprache C
niedrig <- Priorität -> hoch	\$000	RESET	External Pin, Power-on Reset, Brown-out, Reset, Watchdog Reset, and JTAG AVR Reset		
	\$002	INT0	External Interrupt Request 0	INT0addr	INT0_vect
	\$004	INT1	External Interrupt Request 1	INT1addr	INT1_vect
	\$006	TIMER2	COMP Timer/Counter2 Compare Match	OC2addr	TIMER2_COMP_vect
	\$008	TIMER2	OVF Timer/Counter2 Overflow	OVF2addr	TIMER2_OVF_vect
	\$00A	TIMER1	CAPT Timer/Counter1 Capture Event	ICP1addr	TIMER1_CAPT_vect
	\$00C	TIMER1	COMP A Timer/Counter1 Compare Match A	OC1Aaddr	TIMER1_CMPA_vect
	\$00E	TIMER1	COMP B Timer/Counter1 Compare Match B	OC1Baddr	TIMER1_CMPB_vect
	\$010	TIMER1	OVF Timer/Counter1 Overflow	OVF1addr	TIMER1_OVF_vect
	\$012	TIMER0	OVF Timer/Counter0 Overflow	OVF0addr	TIMER0_OVF_vect
	\$014	SPI, STC	Serial Transfer Complete	SPIaddr	SPI_STC_vect
	\$016	USART, RXC	USART, Rx Complete	URXCaddr	USART_RXC_vect
	\$018	USART, UDRE	USART Data Register Empty	UDREaddr	USART_UDRE_vect
	\$01A	USART, TXC	USART, Tx Complete	UTXCaddr	USART_TXC_vect
	\$01C	ADC	ADC Conversion Complete	ADCCaddr	ADC_vect
	\$01E	EE_RDY	EEPROM Ready	ERDYaddr	EE_RDY_vect
	\$020	ANA_COMP	Analog Comparator	ACIaddr	ANA_COMP_vect
	\$022	TWI	Two-wire Serial Interface	TWIaddr	TWI_vect
	\$024	INT2	External Interrupt Request 2	INT2addr	INT2_vect
	\$026	TIMER0	COMP Timer/Counter0 Compare Match	OC0addr	TIMER0_COMP_vect
	\$028	SPM_RDY	Store Program Memory Ready	SPMRaddr	SPM_RDY_vect

Allgemeine Interrupt-Freigabe durch Setzen des I-Flag im Statusregister (SREG)

- Assemblersprache: **sei** ; **Interrupt freigeben**
- Hochsprache C (GCC): **sei()** ; **//Interrupt freigeben**

Während der Abarbeitung einer Interrupt Service Routine (ISR) wird das I-Flag im Statusregister automatisch rückgesetzt, nach dem Beenden der Interrupt Service Routine wird das I-Flag automatisch wieder gesetzt. D. h. während der ISR sind im Normalfall weitere Unterbrechung gesperrt. Diese Sperre kann aufgehoben werden, indem der Programmierer in der Interrupt Service Routine das I-Flag setzt, z. B. durch **sei** bzw. **sei()**.

Allgemeine Interrupt-Spernung durch Rücksetzen des I-Flag im Statusregister (SREG)

- Assemblersprache: **cli** ; **Interrupt sperren**
- Hochsprache C (GCC): **cli()** ; **//Interrupt sperren**

Interrupt extern (Hinweis: Wir beschränken uns auf INT0 und INT1. INT2 wird hier weggelassen)

Externe Interrupts können an den PINs INT0 (PD2) und INT1 (PD3) ausgelöst werden. Verhalten wird konfiguriert im MCUCR (MCU Control Register) mit den Bits ISC11, ISC10, ISC01 und ISC00. Interrupt wird ausgeführt, wenn im Register GICR (General Interrupt Control Register) das entsprechende Bit (INT0 bzw. INT1) gesetzt ist und im SREG (StatusREGISTER) Interrupts freigegeben sind d. h., Flag I ist gesetzt.

MCU Control Register – MCUCR

The MCU Control Register contains control bits for interrupt sense control and general MCU functions.

Bit	7	6	5	4	3	2	1	0	
	SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
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	

INT0 Sense Control

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

INT1 Sense Control

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

General Interrupt Control Register – GICR

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Beispiel: Steigende Flanke am Eingang INT0 = PD2 löst Interrupt aus.

Assemblersprache	Hochsprache C (GCC)
<pre> #include "m16def.inc" .org 0x00 jmp RESET ; Adr. 0 Reset .org INT0addr jmp EXT_INT0 ; Adr. 2 IRQ0 ;Ende der INT Vektoren .org 0x2A ;---- Interrupt Service Routine ---- EXT_INT0: in temp, SREG ;SREG sichern push temp ; hier folgen weitere Befehle der ISR pop temp out SREG, temp ;SREG wiederherstellen reti ;---- Hauptprogramm ---- RESET: ;Stack definieren ldi temp, high(RAMEND) out SPH, temp ldi temp, low(RAMEND) out SPL, temp ;Ports konfigurieren ... ;INT0 konfig: MCUCR (MCU Control Register) ldi temp, (1<<ISC01) (1<<ISC00) out MCUCR, temp ;INT0 freig: GICR (General Int Control Reg) ldi temp, (1<<INT0) out GICR, temp ;Interrupt freigeben (INT-Bit in SREG) sei endlos: ; hier folgen weitere Befehle jmp endlos </pre>	<pre> #include <avr/io.h> #include <avr/interrupt.h> ISR(INT0_vect) { // Interrupt Service Routine // hier folgen weitere Befehle der ISR } void main(void) { //Ports konfigurieren ... //INT0 konfig: MCUCR (MCU Control Reg) MCUCR = (1<<ISC01) (1<<ISC00); //INT0 freig: GICR (General Int Contr Reg) GICR = (1<<INT0); //Interrupt freigeben (INT-Bit in SREG) sei(); while(1) { // Hauptprogramm Endlosschleife } // Ende while } // Ende main </pre>

8 Bit Timer/Counter0

8 Bit Timer/Counter0 im Normal Mode

Zählt vorwärts bis zum Endwert 0xFF und beginnt erneut bei Anfangswert, Überlauf von TCNT0 setzt TOV0

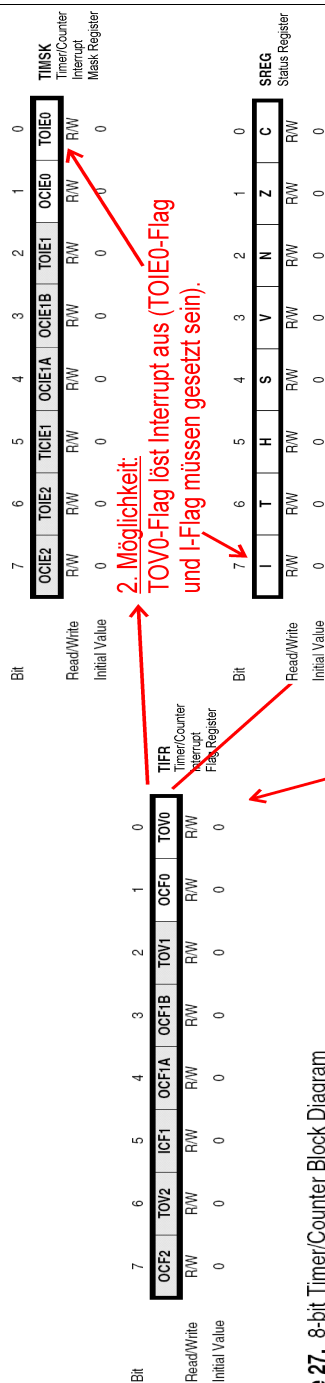
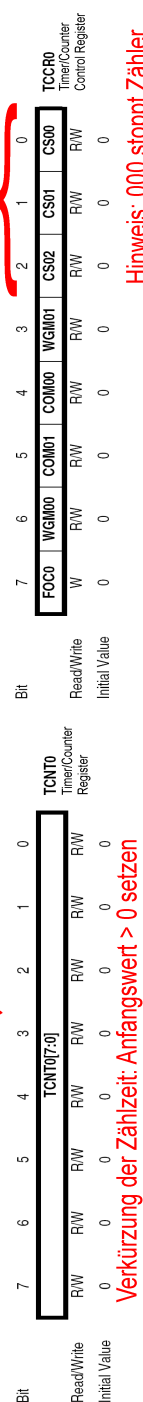
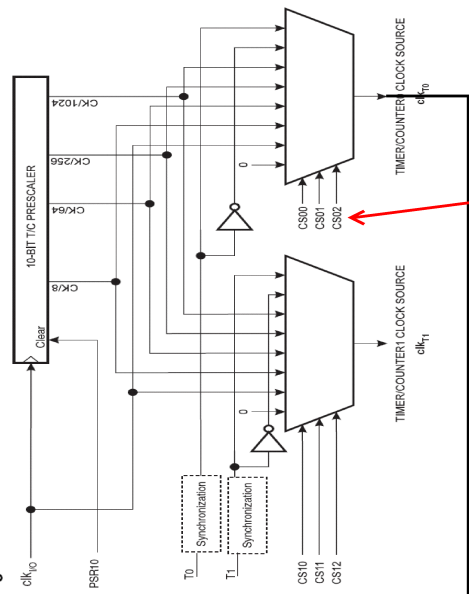


Figure 27. 8-bit Timer/Counter Block Diagram

Überlauf (OVL) von TCNT0 setzt Flag TOV0, Zähler beginnt bei 0x00

Figure 39. Prescaler for Timer/Counter0 and Timer/Counter1⁽¹⁾



Verkürzung der Zählzeit: Anfangswert > 0 setzen

Hinweis: 000 stoppt Zähler

Verzögerungszeiten mit TimerCounter0

Zeit zwischen INT-Aufruf und Beginn der Abarbeitung der ISR wird hier nicht berücksichtigt.

Systemtakt: 1 MHz

Wenn TCNT0 nicht bei 0 beginnen soll, dann muss innerhalb der Interrupt Service Routine der gewünschte Startwert in das Register TCNT0 geladen werden.

Konfigurationsbits in TCCR0			Vorteiler	Berechnung	Verzögerungszeit
CS02	CS01	CS00			
0	0	1	1/1	$256 \cdot 1 \mu s$	256 μs
0	1	0	1/8	$256 \cdot 8 \mu s$	2,048 ms
0	1	1	1/64	$256 \cdot 64 \mu s$	16,384 ms
1	0	0	1/256	$256 \cdot 256 \mu s$	65,536 ms
1	0	1	1/1024	$256 \cdot 1024 \mu s$	262,144 ms

Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$ /(No prescaling)
0	1	0	$clk_{I/O}/8$ (From prescaler)
0	1	1	$clk_{I/O}/64$ (From prescaler)
1	0	0	$clk_{I/O}/256$ (From prescaler)
1	0	1	$clk_{I/O}/1024$ (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

Beispiel: Timer/Counter0 zählt die Impulse eines externen Signals am Pin PB0 = T0, Ausgabe fortlaufend an Port A

Assemblersprache	Hochsprache C (GCC)
<pre>.include "m16def.inc" .def temp = r16 start: ;Ports konfigurieren ;PB0 als Eingang, Pullup-R aktivieren ldi temp, 0b00000001 out PORTB, temp ;Port A als Ausgabe ldi temp, 0xff out DDRA, temp ;TimerCounter0 konfig als Zaehler in temp, TCCR0 ori temp, 0b110 ;ext. Takt ;fallende Flanke ;auch moeglich: ;ldi temp, (1<<CS02) (1<<CS01) out TCCR0, temp endlos: ;Ausgabe aktueller Zaehlerstand an Port A in temp, TCNT0 ;aktueller Zaehlerstand out PORTA, temp jmp endlos</pre>	<pre>#include <avr/io.h> void main(void) { //Ports konfigurieren //PB0 als Eingang, Pullup-R aktivieren PORTB = 0b00000001; //Port A als Ausgabe DDRA = 0xff; //TimerCounter0 konfig als Zaehler TCCR0 = TCCR0 0b110; //ext. Takt //fallende Flanke //auch moeglich: // TCCR0 = (1<<CS02) (1<<CS01) while(1) { //Ausg. aktueller Zaehlerst. an Port A PORTA = TCNT0; //aktueller Zaehlerstand } //Ende while } //Ende main</pre>

16 Bit Timer/Counter1 (Hinweis: weitere Info auf folgender Seite)

Verzögerungszeiten mit TimerCounter1

Zeit zwischen INT-Aufruf und Beginn der Abarbeitung der ISR wird hier nicht berücksichtigt.

Systemtakt: 1 MHz

Wenn TCNT1 nicht bei 0 beginnen soll, dann muss innerhalb der Interrupt Service Routine der gewünschte Startwert in das Register TCNT1 geladen werden.

Hinweis: Die Teileereinheit für Timer/Counter0 und Timer/Counter1 ist gemeinsam, trotzdem kann für jeden Timer/Counter sein eigenes Teileerverhältnis gewählt werden.

Konfigurationsbits in TCCR1B			Vorteiler	Berechnung	Verzögerungszeit
CS12	CS11	CS10			
0	0	1	1/1	$65536 * 1 \mu s$	65,536 ms
0	1	0	1/8	$65536 * 8 \mu s$	524,288 ms
0	1	1	1/64	$65536 * 64 \mu s$	4,1934 s
1	0	0	1/256	$65536 * 256 \mu s$	16,77216 s
1	0	1	1/1024	$65536 * 1024 \mu s$	67,108864 s

Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{IO}/1$ (No prescaler)
0	1	0	$clk_{IO}/8$ (From prescaler)
0	1	1	$clk_{IO}/64$ (From prescaler)
1	0	0	$clk_{IO}/256$ (From prescaler)
1	0	1	$clk_{IO}/1024$ (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

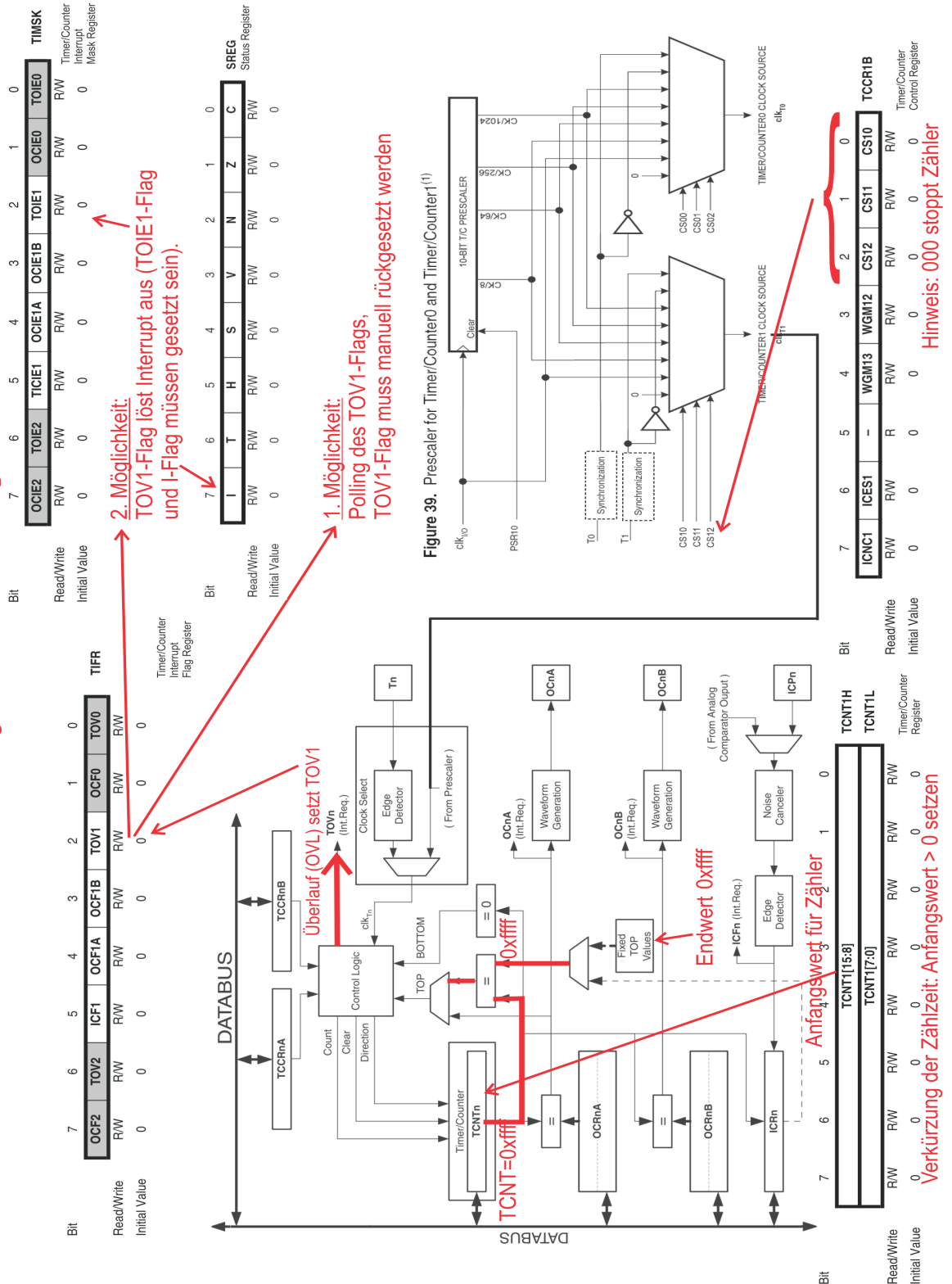
Fortsetzung 16 Bit Timer/Counter1

Reihenfolge beim Schreiben/Lesen eines 16-bit-Registers:

- Schreiben in 16 Bit Register: zuerst Hi-Byte (z. B. TCNT1H), dann Lo-Byte (z. B. TCNT1L)
- Lesen aus 16 Bit Register: zuerst Lo-Byte (z. B. TCNT1L), dann Hi-Byte (z. B. TCNT1H)

16 Bit Timer1 im Normal Mode

Zählt vorwärts bis zum Endwert 0xffff und beginnt erneut bei Anfangswert, Überlauf setzt TOV1



Beispiel: Timer/Counter1 wechselt nach 1 s das Ausgangssignal an Port A (Systemtakt 1 MHz).

Assemblersprache	Hochsprache C (GCC)
<pre> ;Port A blinkt im Sekundentakt #include und Registerzuweisungenorg 0 jmp RESET .org 0x0010 jmp TIM1_OVF ; Timer1 Overflow Handler .org 0x002a ;---- Interrupt Service Routine ---- TIM1_OVF: ;Register retten, falls erforderlich push temp1 push temp2 ;SREG retten (wird durch eor zerstört) in temp1, SREG push temp1 ;Timer1 neu starten (Anfangswert laden) ;schreiben 16-Bit-Reg: Hi-, dann Lo-Byte! ; 1 Sekunde bei 1 MHz Takt, Vorteiler 1/64 ldi temp2,0xff-high(15625) out TCNT1H, temp2 ldi temp1,0xff-low(15625) out TCNT1L, temp1 ;Ausgabe invertieren ldi temp2, 0xff ;Maske fuer Invertierung in temp1, PINA eor temp1, temp2 ;zerstört SREG out PORTA, temp1 ;Restauration von SREG und Arbeitsreg pop temp1 out SREG, temp1 pop temp2 pop temp1 reti ;Ende ISR ;---- Hauptprogramm ---- RESET: ;Stack definieren ldi temp1, high(RAMEND) out SPH, temp1 ldi temp1, low(RAMEND) out SPL, temp1 ;Konfig PortA als Anzeigeport fuer Blinken ldi temp1, 0xff out DDRA, temp1 ;----- Timer1 konfig und starten ----- ; Freigabe von INT bei Ueberlauf von TCNT1 in temp1, TIMSK ori temp1, (1<<TOIE1) ;oder 0b00000100 out TIMSK, temp1 ;Overflow loest INT aus ; Anfangswert Timer setzen, 1. Durchlauf ; Kommentare siehe ISR ldi temp2,0xff-high(15625) out TCNT1H, temp2 ldi temp1,0xff-low(15625) out TCNT1L, temp1 ; Timer starten, Clock Select: Clock/64 in temp1, TCCR1B ori temp1, (1<<CS11) (1<<CS10) ;0b00000011 out TCCR1B,temp1 ; Allg. INT freigeben sei endlos: ;diese Schleife wird unterbrochen! jmp endlos ;Endlosschleife </pre>	<pre> //Port A blinkt im Sekundentakt //include ... ISR(TIM1_OVF_vect) { // INT-Service Routine //Timer1 neu starten (Anfangswert laden) // 1 Sekunde bei 1 MHz Takt, Vorteiler 1/64 TCNT1 = 0xffff-15625; //Ausgabe invertieren PORTA = ~PINA; } void main(void) { //Konfig PortA als Anzeigeport fuer Blinken DDRA = 0xff; //----- Timer1 konfig und starten ----- //Freigabe von INT bei Ueberlauf von TCNT1 TIMSK = TIMSK (1<<TOIE1); // oder 0b00000100 //Anfangswert Timer setzen, 1. Durchlauf //Kommentare siehe ISR TCNT1 = 0xffff-15625; //Timer starten, Clock Select: Clock/64 TCCR1B = TCCR1B ((1<<CS11) (1<<CS10)); //Allg. INT freigeben sei(); while(1) { // diese Schleife wird unterbrochen } } </pre>