ANHANG

F

Copyright ©

Inhaltsverzeichnis ANHANG F

F0 Anhang zum ATmega32A

Befehlssatz des ATmega32A

(Quelle: Atmel® Datenblatt ATmega32A)

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

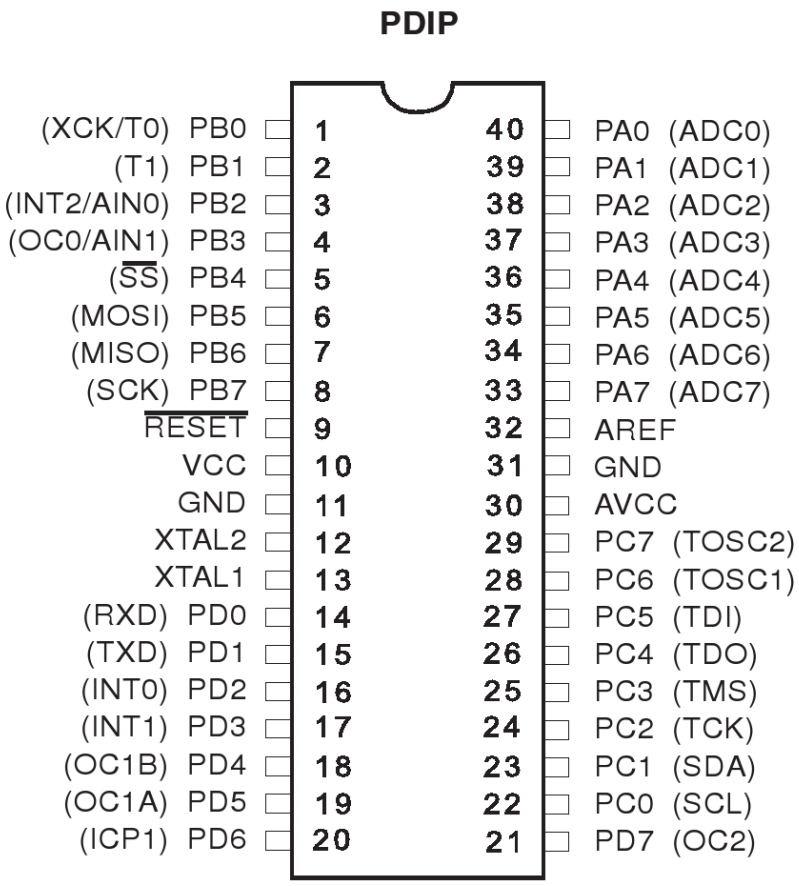
Mnemonics	Operands	Description	Operation	Flags	#Clocks
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1 / 2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1 / 2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1 / 2
BRIE	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
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

Im Befehlssatz werden für Operanden folgende Abkürzungen verwendet:

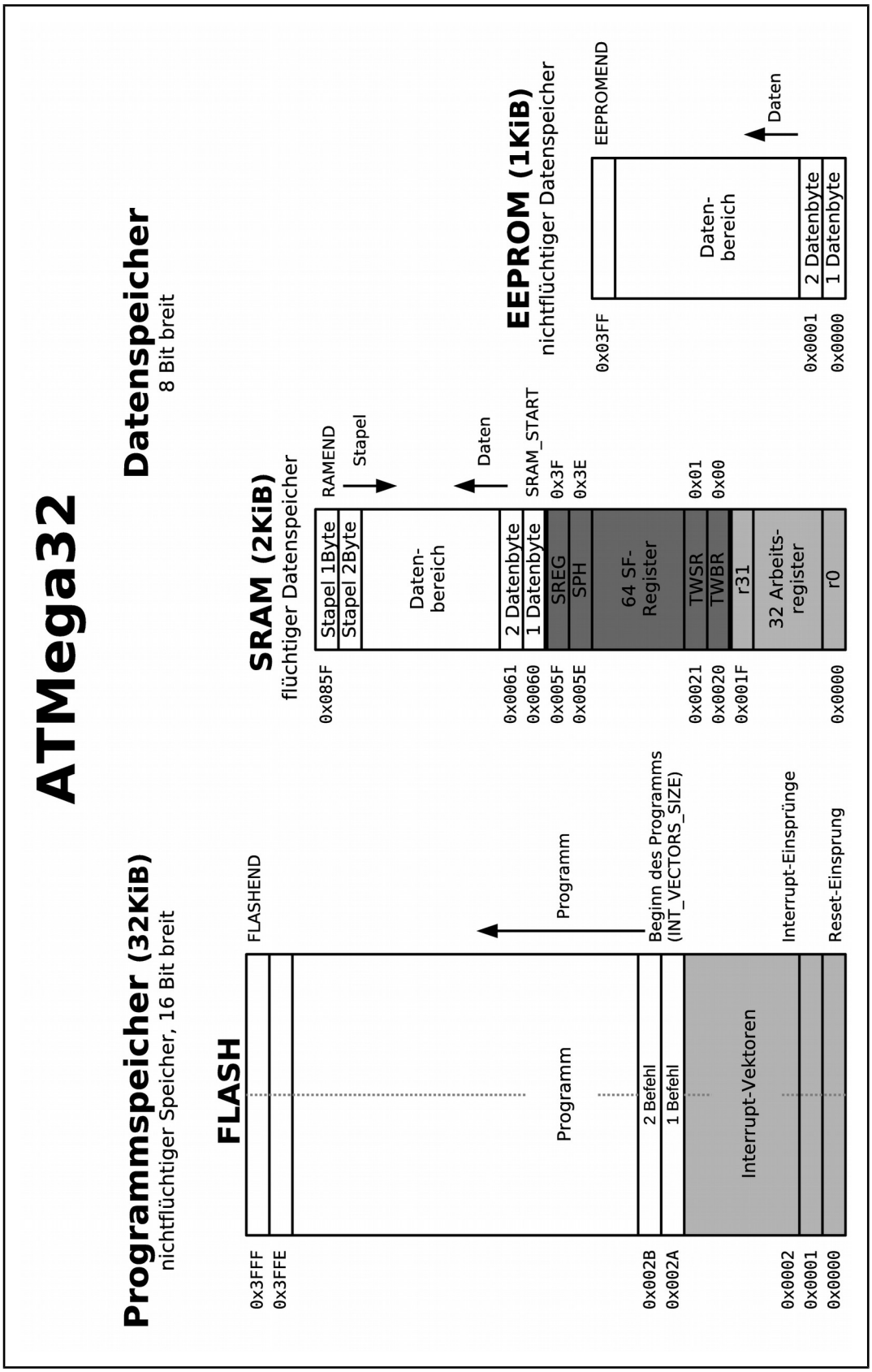
Rd	Meist Ziel-Arbeitsregister (destination, bei einigen Befehlen auch Quelle) **r0**-**r31** (bei immediate Befehlen nur **r16**-**r31**)
Rdl	Niederwertiges Byte (LByte) eines 16 Bit Ziel-Arbeitsregister
Rr	Quell- oder Sende-Arbeitsregister (**r0**-**r31**, source)
K	Daten-Konstante 8 Bit (0-255)
k	Adress-Konstante für Operationen mit dem "program counter PC". (Bsp.: Label für einen Sprung)
b	Bitkonstante 3 Bit (0-7), zum Auswählen eines Bits in einem Arbeits- oder SF-Registers
P	Adresse eines SF-Registers 6 Bit (0-63)
s	Bitkonstante 3 Bit (0-7), zum Auswählen eines Bits im Statusregister
X,Y,Z	Doppelregister (Pointer, Adresszeiger) zur direkten Adressierung **X** ⇒ **r27:r26**; **Y** ⇒ **r29:r28**, **Z** ⇒ **r31:r30**

Pinbelegung des ATmega32A

(Quelle: Atmel® Datenblatt ATmega32A)



Speicherorganisation des ATmega32A



Interrupt-Vektortabelle des ATmega32A[1]

Vektor-nummer	Adresse im Flash	Name in "m32def.inc"	Quelle des Interrupts	Verantwortlich für den Interrupt
21	0x0028	SPMRaddr	SPM_RDY	Programmspeicher (Flash)-Programmierung fertig
20	0x0026	TWIaddr	TWI	I^2C-Interface (TWI-Interface)
19	0x0024	ACIaddr	ANA_COMP	Analog-Komparator
18	0x0022	ERDYaddr	EE_RDY	EEPROM-Programmierung (schreiben) fertig
17	0x0020	ADCCaddr	ADC	A/D-Wandlung vollständig
16	0x001E	UTXCaddr	USART, TXC	USART, Zeichen gesendet (Senderegister leer)
15	0x001C	UDREaddr	USART, UDRE	USART, UDR-Register leer
14	0x001A	URXCaddr	USART, RXC	USART, Empfangsregister voll
13	0x0018	SPIaddr	SPI, STC	Serielle Übertragung beendet (SPI)
12	0x0016	OVF0addr	TIMER0 OVF	Overflow von Timer 0
11	0x0014	OC0addr	TIMER0 COMP	Compare Match von Timer 0
10	0x0012	OVF1addr	TIMER1 OVF	Overflow von Timer 1
9	0x0010	OC1Baddr	TIMER1 COMPB	Compare Match B von Timer 1
8	0x000E	OC1Aaddr	TIMER1 COMPA	Compare Match A von Timer 1
7	0x000C	ICP1addr	TIMER1 CAPT	Capture Event von Timer 2
6	0x000A	OVF2addr	TIMER2 OVF	Overflow Match von Timer 2
5	0x0008	OC2addr	TIMER2 COMP	Compare Match von Timer 2
4	0x0006	INT2addr	INT2	Interrupt-Eingang 2 (externer PIN PB2)
3	0x0004	INT1addr	INT1	Interrupt-Eingang 1 (externer PIN PD3)
2	0x0002	INT0addr	INT0	Interrupt-Eingang 0 (externer PIN PD2)
1	0x0000		RESET	RESET Pin (extern Pin 9), Power-On-Reset, Brown-Out-Reset, Watchdog Reset und JTAG AVR-Reset

Bemerkungen:

Das eigentliche Programm kann erst an der Adresse **0x002A** beginnen. Der in der Definitionsdatei vorgesehene Name für diese Adresse heißt: **INT_VECTORS_SIZE**.

Wenn das **BOOTRST**-Fuse-Bit programmiert wurde springt der Controller nach einem **RESET** automatisch in den Bootbereich (Bootloader-Programm) im oberen Adressbereich des Flash-Speichers. Mit Hilfe des **IVSEL**-Bit im SF-Register **GICR** kann die Interrupt-Vektortabelle in den Anfang des Bootbereichs verlegt werden.

1 Die Interrupt-Vektortabelle des ATmega16A entspricht nicht der Interrupt-Vektortabelle des ATmega32A!

SF-Registersatz des ATmega32A

(Quelle: Atmel® Datenblatt ATmega32A)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
$3F ($5F)	SREG	I	T	H	S	V	N	Z	C	8
$3E ($5E)	SPH	–	–	–	–	SP11	SP10	SP9	SP8	11
$3D ($5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	11
$3C ($5C)	OCR0	Timer/Counter0 Output Compare Register								86
$3B ($5B)	GICR	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	48, 71
$3A ($5A)	GIFR	INTF1	INTF0	INTF2	–	–	–	–	–	71
$39 ($59)	TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	87, 117, 136
$38 ($58)	TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	87, 117, 136
$37 ($57)	SPMCR	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	264
$36 ($56)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	202
$35 ($55)	MCUCR	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	36, 69
$34 ($54)	MCUCSR	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF	42, 70, 251
$33 ($53)	TCCR0	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	84
$32 ($52)	TCNT0	Timer/Counter0 (8 Bits)								86
$31[1] ($51)[1]	OSCCAL	Oscillator Calibration Register								32
	OCDR	On-Chip Debug Register								232
$30 ($50)	SFIOR	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	66,90,137,206,226
$2F ($4F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	112
$2E ($4E)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	114
$2D ($4D)	TCNT1H	Timer/Counter1 – Counter Register High Byte								116
$2C ($4C)	TCNT1L	Timer/Counter1 – Counter Register Low Byte								116
$2B ($4B)	OCR1AH	Timer/Counter1 – Output Compare Register A High Byte								116
$2A ($4A)	OCR1AL	Timer/Counter1 – Output Compare Register A Low Byte								116
$29 ($49)	OCR1BH	Timer/Counter1 – Output Compare Register B High Byte								116
$28 ($48)	OCR1BL	Timer/Counter1 – Output Compare Register B Low Byte								116
$27 ($47)	ICR1H	Timer/Counter1 – Input Capture Register High Byte								116
$26 ($46)	ICR1L	Timer/Counter1 – Input Capture Register Low Byte								116
$25 ($45)	TCCR2	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	132
$24 ($44)	TCNT2	Timer/Counter2 (8 Bits)								135
$23 ($43)	OCR2	Timer/Counter2 Output Compare Register								135
$22 ($42)	ASSR	–	–	–	–	AS2	TCN2UB	OCR2UB	TCR2UB	135
$21 ($41)	WDTCR	–	–	–	WDTOE	WDE	WDP2	WDP1	WDP0	43
$20[2] ($40)[2]	UBRRH	URSEL	–	–	–	UBRR[11:8]				171
	UCSRC	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	170
$1F ($3F)	EEARH	–	–	–	–	–	–	EEAR9	EEAR8	20
$1E ($3E)	EEARL	EEPROM Address Register Low Byte								20
$1D ($3D)	EEDR	EEPROM Data Register								21
$1C ($3C)	EECR	–	–	–	–	EERIE	EEMWE	EEWE	EERE	21
$1B ($3B)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	66
$1A ($3A)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	66
$19 ($39)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	66
$18 ($38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	67
$17 ($37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	67
$16 ($36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	67
$15 ($35)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	67
$14 ($34)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	67
$13 ($33)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	67
$12 ($32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	67
$11 ($31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	67
$10 ($30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	68
$0F ($2F)	SPDR	SPI Data Register								145
$0E ($2E)	SPSR	SPIF	WCOL	–	–	–	–	–	SPI2X	145
$0D ($2D)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	143
$0C ($2C)	UDR	USART I/O Data Register								167
$0B ($2B)	UCSRA	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	168
$0A ($2A)	UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	169
$09 ($29)	UBRRL	USART Baud Rate Register Low Byte								171
$08 ($28)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	206
$07 ($27)	ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	222
$06 ($26)	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	224
$05 ($25)	ADCH	ADC Data Register High Byte								225
$04 ($24)	ADCL	ADC Data Register Low Byte								225
$03 ($23)	TWDR	Two-wire Serial Interface Data Register								203
$02 ($22)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	204
$01 ($21)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	203
$00 ($20)	TWBR	Two-wire Serial Interface Bit Rate Register								201

Notes:
1. When the OCDEN Fuse is unprogrammed, the OSCCAL Register is always accessed on this address. Refer to the debugger specific documentation for details on how to use the OCDR Register.
2. Refer to the USART description for details on how to access UBRRH and UCSRC.
3. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
4. Some of the Status Flags are cleared by writing a logical one to them. Note that the CBI and SBI instructions will operate on all bits in the I/O Register, writing a one back into any flag read as set, thus clearing the flag. The CBI and SBI instructions work with registers $00 to $1F only.

Parallele digitale Ein-/Ausgabeports (12)

Beim ATmega32A existieren vier 8-Bit parallele digitale Ein-/Ausgabeports PORT A bis PORT D. Pro Port werden drei Register benötigt: das Datenrichtungsregister **DDRx**, das Datenausgaberegister **PORTx** und ein Dateneingaberegister **PINx**[2] über das die Zustände des Ports eingelesen werden können. Jedes einzelne Pin kann individuell konfiguriert werden. Zusätzlich kann ein interner Pull-Up-Widerstand zugeschaltet werden.

DDRx = Data Direction Register PORTx

Bit	7	6	5	4	3	2	1	0
DDRx	**DDx7**	**DDx6**	**DDx5**	**DDx4**	**DDx3**	**DDx2**	**DDx1**	**DDx0**
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

DDxn ***Data Direction Bit n (Port x)***

0 Das betreffende Pin als Eingang aktiv. Um Kurzschlüsse zu vermeiden sollten alle nicht benötigten Pins auf Eingang geschaltet werden (Startwert).

1 Das betreffende Pin ist als Ausgang beschaltet.

PORTx =PORTx Data Register

Bit	7	6	5	4	3	2	1	0
PORTx	**PORTx7**	**PORTx6**	**PORTx5**	**PORTx4**	**PORTx3**	**PORTx2**	**PORTx1**	**PORTx0**
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

PORTxn ***PORTx Data Pin n***

a) Das Pin ist als Ausgang konfiguriert DDRxn = 1

0 Das Pin liegt auf Masse. Der Strom kann zum Pin fließen (Stromsenke).

1 Das Pin liegt auf VCC und kann als Quelle einen Strom liefern.

b) Das Pin ist als Eingang konfiguriert DDRxn = 1

0 Das Pin liegt ist hochohmig (Tri-State).

1 Ist das **PUD**-Bit (*pull-up disable*)im Register **SFIOR** gelöscht (Startwert) so wird intern ein Pull-Up-Widerstand gegen **VCC** geschaltet. Dadurch ist es möglich auf externe Pull-Up-Widerstände zu verzichten, wenn das Pin als Eingang benutzt wird. Bei gesetztem **PUD**-Bit ist der Ausgang hochohmig.

PINx = PORTx INput Pins Adress

Bit	7	6	5	4	3	2	1	0
PINx	**PINx7**	**PINx6**	**PINx5**	**PINx4**	**PINx3**	**PINx2**	**PINx1**	**PINx0**
Startwert	-	-	-	-	-	-	-	-
Read/Write	R	R	R	R	R	R	R	R

PINxn ***PORTx INput Pin n***

Unabhängig vom Datenrichtungsregister kann über die **PINx** Adresse der Zustand eines Pins eingelesen werden. Das Pin sollte dabei immer einen definierten Zustand haben (Pull-Up oder Pull-Down-Widerstand zuschalten!).

2 x steht für den Port-Buchstaben (A-D), n für das jeweilige Bit (0-7).

Statusregister SREG (1)

Eines der wichtigsten Register überhaupt ist das Statusregister (Zustands -oder Flagregister). Hier werden in den Bits 0-5 nach Rechenoperationen (arithmetischen Befehle, Vergleichsbefehle, logische Befehle, Rotationsbefehle) alle wichtigen Zustände der Operation abgelegt. Diese dienen dann dazu später die Resultate der Operationen zu interpretieren. Dies Zustandsbits werden auch noch als Flags (Signal-Fahnen) bezeichnet und können durch spezielle Befehle zur Bitbeeinflussung gesetzt und gelöscht werden. Zusätzlich sind im Zustandsregister noch ein Bit enthalten mit dem global Interrupts zugelassen oder gesperrt werden können (**I**-Flag), und ein Bit (**T**-Flag), das dazu dient dem Anwender das Zwischenspeichern eines Zustandes (Bit, Flag) zu vereinfachen.

SREG = Status Register

Bit	7	6	5	4	3	2	1	0
SREG 0x3F	**I** Interrupt	**T** Transfer	**H** Halfcarry	**S** Sign	**V** Overflow	**N** Negative	**Z** Zero	**C** Carry
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

I Das **"Global Interrupt Enable/Disable"**-Flag **I** wird vom Benutzer gesetzt (1) oder rückgesetzt (0) um global Unterbrechungen zu erlauben bzw. zu verbieten (Befehle: **sei**, **cli**).

T Das **"Transfer"**-Flag **T** erlaubt mit Hilfe der Befehle **bld** und **bst** ein einzelnes Bit aus einem Arbeitsregister abzuspeichern (retten) und wieder zu laden (wiederherstellen). Es kann mit den Befehlen **set** und **clt** auch einfach gelöscht oder gesetzt werden.

H Das **"Halfcarry"**-Flag **H** (Auxiliary-Carry) kennzeichnet einen Übertrag von Bit 3 auf Bit 4 und wird bei Berechnungen mit Nibbles (4 Bit) benötigt. Es kann mit den Befehlen **seh** und **clh** auch einfach gelöscht oder gesetzt werden.

S Das **"Sign"**-Flag **S** wird bei der Berechnung mit vorzeichenbehafteten Zahlen eingesetzt. Es entspricht der Exklusiv-Oder Verknüpfung des Negative- und des Overflow-Flags: $S = N \; xor \; V = (\overline{N} \wedge V) \vee (N \wedge \overline{V})$. Es kann mit den Befehlen **ses** und **cls** auch einfach gelöscht oder gesetzt werden.

V Das **"Overflow"**-Flag **V** zeigt einen Überlauf bei der Zweierkomplement-Berechnung, also bei vorzeichenbehafteten Zahlen an. Es kann mit den Befehlen **sev** und **clv** auch einfach gelöscht oder gesetzt werden

N Das **"Negative"**-Flag **N** entspricht dem höchstwertigen Bit des Resultats.

8 Bit: $N = r7$

16 Bit: $N = r15$.

Es kann mit den Befehlen **sen** und **cln** auch einfach gelöscht oder gesetzt werden.

Z Das **"Zero"**-Flag **Z** wird auf Eins gesetzt wenn das Ergebnis einer Operation Null ist. Es kann mit den Befehlen **sez** und **clz** auch einfach gelöscht oder gesetzt werden

C Das **"Carry"**-Flag **C** wird Eins, wenn ein Übertrag an der höchsten Stelle entsteht. Es kann mit den Befehlen **sec** und **clc** auch einfach gelöscht oder gesetzt werden.

Interrupts (4)

Drei externe Interrupts, **INT0** (**PD2**), **INT1** (**PD3**) und **INT2** (**PB2**) stehen beim ATmega32A zur Verfügung. Aktivieren kann man sie, falls Interrupts global zugelassen sind (**sei**), mit dem **GICR**-Register[3]. Trifft ein Interrupt ein, so wird dies mit einem Flag im **GIFR**-Register gespeichert. Auf welche Flanke bzw. auf welchen Pegel der Interrupt reagiert wird im **MCUCR** und **MCUCSR**-Register festgelegt.

GICR = General Interrupt Control Register

Bit	7	6	5	4	3	2	1	0
GICR 0x3B	**INT1**	**INT0**	**INT2**	**-**	**-**	**-**	**IVSEL**	**IVCE**
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W

INTn ***External INTerrupt Request n Enable***

0 Das betreffende Interrupt ist nicht aktiviert.

1 Das betreffende Interrupt ist aktiviert, wenn ebenfalls das **I**-Bit in **SREG** gesetzt ist (Interrupts global erlaubt, **sei**). Ein Interrupt wird auch erkannt, wenn das betreffende Pin als Ausgang initialisiert wurde.

GIFR = General Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0
GIFR 0x3A	**INTF1**	**INTF0**	**INTF2**	**-**	**-**	**-**	**-**	**-**
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R	R	R	R	R

INTFn ***External INTerrupt Flag n***

0 Es trat kein Interrupt auf. Das Flag wird automatisch gelöscht wenn die Interruptroutine aufgerufen wird. Es ist beim **INT0** bzw. **INT1** dauernd gelöscht, wenn die Interrupts auf Pegel reagieren sollen.

1 Es wurde ein Interrupt erkannt und dies wird im Flag gespeichert. Passiert dies innerhalb einer Interruptroutine (wo standardmäßig Interrupts global gesperrt sind), so wird das Interrupt nach dem Verlassen der Routine ausgeführt.
Mit dem **Schreiben einer Eins!** kann das Interrupt-Flag manuell gelöscht werden.

MCUCR = MCU Control Register

Bit	7	6	5	4	3	2	1	0
MCUCR 0x35	**SE**	**SM2**	**SM1**	**SM0**	**ISC11**	**ISC10**	**ISC01**	**ISC00**
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

ISCn ***Interrupt Sense Control n ISCn1, ISCn0***

Mit je zwei Bit kann bei **INT0** und bei **INT1** ausgewählt werden, ob der Interrupt auf einen Zustand (Pegel) oder auf eine Flanke reagieren soll (zustandsgesteuert bzw. flankengesteuert).

00 **Der Low-Zustand löst einen Interrupt aus.**

01 **Jede Zustandsänderung löst einen Interrupt aus.**

10 **Eine fallende Flanke löst einen Interrupt aus.**

11 **Eine steigende Flanke löst einen Interrupt aus.**

3 Achtung! Die Reihenfolge in beiden Registern (**GICR** und **GIFR**) ist 1,0,2!

MCUCSR = MCU Control and Status Register

Bit	7	6	5	4	3	2	1	0
MCUCSR 0x34	**JTD**	**ISC2**	**-**	**JTRF**	**WDRF**	**BORF**	**EXTRF**	**PORF**
Startwert	0	0	0	-	-	-	-	-
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W

ISC2 ***Interrupt Sense Control 2***

INT2 kann nur flankengesteuert betrieben werden.

0 Eine fallende Flanke löst einen Interrupt aus.
1 Eine steigende Flanke löst einen Interrupt aus.

Serielle Schnittstelle (6)

Mit Hilfe von drei Kontroll- und Statusregistern **UCSRA**, **UCSRB** und **UCSRC** können bei der Initialisierung wichtige Parameter (Zeichenrahmen, Freigabe von Interrupts, ...) eingestellt werden und während des Programms wichtige Zustände (Errorflags, Interruptflags) abgefragt werden. Mit dem Doppelregister **UBRR** (**UBRRL**, **UBRRH**[4]) wird die Übertragungsgeschwindigkeit eingestellt. Daten werden über das Datenregister **UDR** empfangen oder gesendet.

UCSRA = USART Control and Status Register A

Bit	7	6	5	4	3	2	1	0
UCSRA 0x0B	**RXC**	**TXC**	**UDRE**	**FE**	**DOR**	**PE**	**U2X**	**MPCM**
Startwert	0	0	1	0	0	0	0	0
Read/Write	R	R/W	R	R	R	R	R/W	R/W

RXC ***USART Receive Complete***

0 Wenn sich keine Daten mehr im Empfangspuffer befinden oder wenn der Empfänger ausgeschaltet ist (Startwert).

1 Wird auf Eins gesetzt, wenn sich Daten im Empfangspuffer befinden. Zeigt also an, dass die Daten ausgelesen werden können. Kann zusätzlich einen Interrupt auslösen (siehe **UCSRB**).

TXC ***USART Transmit Complete***

0 Es sind noch Daten im Schieberegister oder Datenregister vorhanden. Es kann noch kein neues Zeichen gesendet werden (Startwert).

1 Wird auf Eins gesetzt, wenn alle Daten (gesamter Rahmen) aus dem Schieberegister ausgegeben wurden und keine neuen Daten im **UDR**-Datenregister (Sendepuffer) vorliegen. Muss manuell mit einer Eins! vor der Ausgabe gelöscht werden!
Kann zusätzlich einen Interrupt auslösen (siehe **UCSRB**).

UDRE ***USART Data Register Empty***

0 Datenregister **UDR** besetzt.

1 Wird auf Eins gesetzt, wenn das Datenregister **UDR** (Sendepuffer) leer ist und der USART somit bereit ist neue Daten anzunehmen (Startwert nach **RESET**!). Kann zusätzlich einen Interrupt auslösen (siehe **UCSRB**).

FE ***Frame Error***

0 kein Rahmenfehler.

1 **Rahmenfehler**, kein gültiges Stoppbit erkannt.

DOR ***Data OverRun***

0 kein Überlauffehler.

1 **Überlauffehler**; tritt auf wenn beide Pufferregister (**RXB** bzw. **UDR**) und das Schieberegister belegt sind, und dann ein gültiges Startbit erkannt wird.

PE ***Parity Error***

0 kein Paritätsfehler.

1 **Paritätsfehler.**

4 Da sich **UCSRC** und **UBRRH** intern ein Register teilen müssen wird die Auswahl des Registers mit dem höchstwertigen Bit festgelegt.

UCSRB = USART Control and Status Register B

Bit	7	6	5	4	3	2	1	0
UCSRB 0x0A	**RXCIE**	**TXCIE**	**UDRIE**	**RXEN**	**TXEN**	**UCSZ2**	**RXB8**	**TXB8**
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W

RXCIE *RXC Interrupt Enable*

0 kein **RXC**-Interrupt erlaubt.

1 Das Setzen dieses Bit **ermöglicht das Auslösen eines Interrupts** in dem Moment, wo das **RXC**-Flag (**UCSRA**) gesetzt wird, also neue Daten im Empfangspuffer vorhanden sind. Interrupts müssen dazu global frei gegeben sein (**I**=1 im **SREG** mit "**sei**").

TXCIE *TXC Interrupt Enable*

0 kein **TXC**-Interrupt erlaubt.

1 Das Setzen dieses Bit **ermöglicht das Auslösen eines Interrupts** in dem Moment, wo das **TXC**-Flag (**UCSRA**) gesetzt wird, also Schieberegister und **UDR**-Register (Sendepuffer) leer sind. Interrupts müssen dazu global frei gegeben sein (**I**=1 im **SREG** mit "**sei**").

UDRIE *UDRE Interrupt Enable*

0 kein **TXC**-Interrupt erlaubt.

1 Das Setzen dieses Bit **ermöglicht das Auslösen eines Interrupts** in dem Moment, wo das **UDRE**-Flag (**UCSRA**) gesetzt wird, also das **UDR**-Datenregister (Sendepuffer) leer ist. Interrupts müssen dazu global frei gegeben sein (**I**=1 im **SREG** mit "**sei**").

RXEN *Receiver ENnable*

0 schaltet den Empfänger aus.

1 **schaltet den Empfänger ein**. Pin **RxD** (**PORTD0**) ist dann als Eingang reserviert (muss nicht extra initialisiert werden) und ist für andere Aktionen nicht mehr zugänglich.

TXEN *Transmitter ENnable*

0 schaltet den Sender aus.

1 **schaltet den Sender ein**. Pin **TxD** (**PORTD1**) ist als Ausgang reserviert (muss nicht extra initialisiert werden) und ist für andere Aktionen nicht mehr zugänglich.

UCSZ2 *USART Character Size 2*

siehe nächstes Register **UCSRC**

UCSRC = USART Control and Status Register C

Bit	7	6	5	4	3	2	1	0
UCSRC 0x20	**URSEL**	**UMSEL**	**UPM1**	**UPM0**	**USBS**	**UCSZ1**	**UCSZ0**	**UCPOL**
Startwert	1	0	0	0	0	1	1	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

URSEL ***USART Register SELect***

Der Speicherplatz ist mit zwei Registern belegt.

0 das Register **UBRRH wird ausgewählt**. Beim Lesen von **UBRRH** ist das Bit Null.

1 das Register **UCSRC wird ausgewählt** (default). Beim Lesen von **UCSRC** ist das Bit Eins.

UMSEL ***USART Mode SELect***

0 **Asynchroner Modus**

1 **Synchroner Modus**

UPMn ***USART Parity Mode UPM1, UPM0***

Ist die Parität eingeschaltet (gerade oder ungerade Parität) wird hardwaremäßig die Parität generiert und in den Zeichenrahmen mit integriert. Als Empfänger kontrolliert der USART die Parität. Ist ein Fehler aufgetreten, so wird das **PE**-Flag (Parity Error) in **UCSRA** gesetzt.

00 **keine Parität**

01 reserviert

10 **gerade Parität** eingeschaltet

11 **ungerade Parität** eingeschaltet.

USBS ***USART Stop Bit Select***

0 **1 Stoppbit**

1 **2 Stoppbit**

UCSZn ***USART Character Size*** ***UCSZ2, UCSZ1, UCSZ0***

Mit diesen drei Bit wird die **Anzahl der Datenbits (Zeichengröße) eingestellt**. **UCSZ2** befindet sich auf Bit 2 im **UCSRB**-Register und wird nur benötigt wenn 9 Bit benötigt werden. Sonst kann dieses Bit unberücksichtigt bleiben, da sein Default-Wert Null ist.

UCSZ2 UCSZ1 UCSZ0 2^2 2^1 2^0	Anzahl der Datenbits
000	5
001	6
010	7
011	8
100	reserviert
101	reserviert
110	reserviert
111	9

UCPOL ***USART Clock POLarity***

Wird nur bei synchroner Datenübertragung benutzt

0 bei **asynchroner** Übertragung

UBRRH = USART Baud Rate Register High

Bit	7	6	5	4	3	2	1	0
UBRRH 0x20	**URSEL**	**-**	**-**	**-**	**UBRR 11**	**UBRR 10**	**UBRR 9**	**UBRR 8**
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W

UBRRL = USART Baud Rate Register Low

Bit	7	6	5	4	3	2	1	0
UBRRL 0x09	**UBRR 7**	**UBRR 6**	**UBRR 5**	**UBRR 4**	**UBRR 3**	**UBRR 2**	**UBRR 1**	**UBRR 0**
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

URSEL ***USART Register SELect***

Der Speicherplatz ist mit zwei Registern belegt.

0 das Register **UBRRH wird ausgewählt**. Beim Lesen von **UBRRH** ist das Bit Null.

1 das Register **UCSRC wird ausgewählt** (Startwert). Beim Lesen von **UCSRC** ist das Bit Eins.

UBRRn ***USART Baud Rate Register Bit n***

In diesen 12 Bit befindet sich der Teiler aus dem sich die Baudrate berechnen lässt. Die Abtastung des Eingangsignals **RxD** erfolgt mit dem 16-fachen Übertragungstakt (Baudrate). Die Formel lautet:

$$Baudrate = \frac{Systemtakt}{16 \cdot (Teiler + 1)}$$

Umgekehrt lässt sich natürlich auch der Teiler bei erwünschter Baudrate errechnen:

$$Teiler = \frac{Systemtakt}{16 \cdot Baudrate} - 1$$

Die mit dem Teiler erreichte Baudrate entspricht nicht immer dem genauen Standardwert. Der Fehler (in Prozent) errechnet sich mit:

$$Fehler[\%] = \left(\frac{Baudrate}{Standardbaudrate} - 1 \right) \cdot 100\,\%$$

Fehler unter 0,5% sollen angestrebt werden, da sonst die Fehlerrate bei der Übertragung zunimmt. Die kann zum Beispiel durch das Auswechseln des Quarzes erfolgen oder durch Verdoppeln der Baudrate mit **U2X** in **UCSRA** (siehe Datenblatt).

UDR = USART I/O Data Register

Bit	7	6	5	4	3	2	1	0
UDR r/w 0x0C	RXB7 TXB7	RXB6 TXB6	RXB5 TXB5	RXB4 TXB4	RXB3 TXB3	RXB2 TXB2	RXB1 TXB1	RXB0 TXB0
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

TXBn ***Transmit Data Buffer Register Bit n***
Sendepuffer.

RXBn ***Receive Data Buffer Register Bit n***
Empfangspuffer.

A/D-Wandler (4)

Für die Steuerung des A/D-Wandler sind drei Register zuständig: **ADMUX**, **ADCSRA** und **SFIOR**. Das Ergebnis der A/D-Wandlung (10 Bit) steht im Doppelregister **ADC** (**ADCH**, **ADCL**).

ADMUX = ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0
ADMUX 0x07	**REFS1**	**REFS0**	**ADLAR**	**MUX4**	**MUX3**	**MUX2**	**MUX1**	**MUX0**
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

REFSn ***Reference Selection Bits*** ***REFS1, REFS0***

Mit diesen beiden Bit wird die **Quelle für die Referenzspannung ausgewählt** (3 Möglichkeiten). Änderungen an diesen Bits werden erst nach einer eventuell stattfindenden Wandlung berücksichtigt. Liegt eine externe Referenzspannung an Pin **AREF**, so soll nicht softwaremäßig auf interne Spannungsquelle geschaltet werden!

00 **Externe Referenzspannung am Pin AREF des Chip (zwischen 2 V und VCC, interne Spannungsquelle VREF abgeschaltet)**
01 **Die Spannung des Wandler am Pin AVCC wird als Referenz benutzt*.**
10 **reserviert**
11 **Eine interne Referenzspannung VREF von 2,56 V wird benutzt*.**

***** Der offene Eingang **AREF** soll zur Störunterdrückung mit einem Kondensator (z.B. 100 nF) gegen Masse geschaltet werden. Weitere Informationen zur Störunterdrückung im Datenblatt Seite 210 unter "Analog Noise Canceling Techniques"

ADLAR ***ADC Left Adjust Result***

0 Das 10-Bit-Ergebnis der A/D-Wandlung wird **rechtsbündig** im Doppelregister ADC abgelegt (siehe **ADCL** und **ADCH**).
1 Das 10-Bit-Ergebnis der A/D-Wandlung wird **linksbündig** im Doppelregister ADC abgelegt (siehe **ADCL** und **ADCH**).

MUXn ***Analog Channel and Gain Selection Bits MUX0-MUX4***

Mit diesen fünf Bit wird der bzw. die **Eingangspins** am Port A (Eingangskanal), die **Betriebsart** (unsymmetrisch (unipolar, Spannung gegen Masse) oder symmetrisch (differenziell, bipolar, Spannung zwischen zwei Pins)) und der **Verstärkungsfaktor**[5] nach der folgenden Tabelle **ausgewählt**. Änderungen an diesen Bits werden erst nach einer eventuell stattfindenden Wandlung berücksichtigt.

5 Die 200-fache Verstärkung ist nicht für PDIP-Gehäuseformen getestet.

MUXn	UNSYMETRISCH	SYMETRISCH		
43 210 $2^4 2^3 2^2 2^1 2^0$	(Pin gegen Masse)	Positiver (nicht-invert.) Eingang	Negativer (invert.) Eingang	Verstärkung
00 000	PA0 (ADC0)			
00 001	PA1 (ADC1)			
00 010	PA2 (ADC2)			
00 011	PA3 (ADC3)			
00 100	PA4 (ADC4)			
00 101	PA5 (ADC5)			
00 110	PA6 (ADC6)			
00 111	PA7 (ADC7)			
01 000		PA0 (ADC0)	PA0 (ADC0)	10x
01 001		PA1 (ADC1)	PA0 (ADC0)	10x
01 010		PA0 (ADC0)	PA0 (ADC0)	200x
01 011		PA1 (ADC1)	PA0 (ADC0)	200x
01 100		PA2 (ADC2)	PA2 (ADC2)	10x
01 101		PA3 (ADC3)	PA2 (ADC2)	10x
01 110		PA2 (ADC2)	PA2 (ADC2)	200x
01 111		PA3 (ADC3)	PA2 (ADC2)	200x
10 000		PA0 (ADC0)	PA1 (ADC1)	1x
10 001		PA1 (ADC1)	PA1 (ADC1)	1x
10 010		PA2 (ADC2)	PA1 (ADC1)	1x
10 011		PA3 (ADC3)	PA1 (ADC1)	1x
10 100		PA4 (ADC4)	PA1 (ADC1)	1x
10 101		PA5 (ADC5)	PA1 (ADC1)	1x
10 110		PA6 (ADC6)	PA1 (ADC1)	1x
10 111		PA7 (ADC7)	PA1 (ADC1)	1x
11 000		PA0 (ADC0)	PA2 (ADC2)	1x
11 001		PA1 (ADC1)	PA2 (ADC2)	1x
11 010		PA2 (ADC2)	PA2 (ADC2)	1x
11 011		PA3 (ADC3)	PA2 (ADC2)	1x
11 100		PA4 (ADC4)	PA2 (ADC2)	1x
11 101		PA5 (ADC5)	PA2 (ADC2)	1x
11 110	1,22 V (Bandgap-Referenz V_{BG})			
11 111	0 V (GND)			

ADCSRA = ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0
ADCSRA 0x06	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

ADEN ***ADC Enable***

0 schaltet den A/D-Wandler aus (Baugruppe verbraucht keinen Strom!). Geschieht dies während einer Wandlung, so wird diese abgebrochen.

1 **schaltet den A/D-Wandler ein**.

ADSC *ADC Start Conversion*

0 wird nach einer Wandlung automatisch wieder auf Null gesetzt. Das Löschen des Bit während der Wandlung hat keinen Einfluss.

1 **startet die A/D-Wandlung** und kann dazu benutzt werden festzustellen ob momentan eine Wandlung durchgeführt wird.
Single Conversion Mode: Jede einzelne Wandlung wird durch erneutes Schreiben des Bit eingeleitet.
Free Running Mode: Durch (einmalige) Setzen des Bit wird die erste Wandlung eingeleitet. Die erste Wandlung dauert länger (25 statt 13 A/D-Taktzyklen), da ein zusätzlicher Umwandlungszyklus vorangestellt wird, der zur Initialisierung des Wandlers dient.
ADSC und **ADEN** können gleichzeitig gesetzt werden.

ADATE *ADC Auto Trigger Enable*

0 beendet den Auto Trigger Modus.

1 **startet den Auto Trigger Modus**.
Hierbei handelt es sich um einen Trigger Modus, der auf eine von acht Interrupt- Quellen (Trigger-Quellen) reagieren kann. Der A/D-Wandler startet eine Wandlung, wenn eine **positive Flanke** der Interrupt-Quelle erkannt wird. Die Interrupt-Quelle wird mit drei Bit (**ADTS0 - ADTS2**) im **SFIOR**-Register ausgewählt.

ADIF *ADC Interrupt Flag*

0 keine aktuelle Daten im Datenregister **ADC**

1 wird **Eins** sobald der **Wandlungszyklus beendet** ist und die Daten im Doppelregister **ADC** (**ADCL**, **ADCH**) aktualisiert wurden. Gleichzeitig wird ein ADC Complete Interrupt ausgelöst, wenn Interrupts global erlaubt sind (**I** im **SREG**) und das **ADIE**-Bit gesetzt wurde. Wurde ein Interrupt ausgelöst, so wird das Bit hardwaremäßig nach dem Ausführen des Interrupt gelöscht.
Im Polling-Betrieb kann das Bit manuell durch Schreiben einer Eins gelöscht werden!

ADIE *ADC Interrupt Enable*

0 der A/D-Wandler Interrupt ist gesperrt.

1 Das Setzen dieses Bit **ermöglicht das Auslösen eines Interrupts** in dem Moment, wo die A/D-Wandlung beendet ist und neue Daten anliegen (**ADI**-Flag im **ADCSRA**). Interrupts müssen dazu global frei gegeben sein (**I**=1 im **SREG** mit "**sei**").

ADPSn *ADC Prescaler Select Bits ADPS2, ADPS1, ADPS0*

Mit diesen drei Bit wird der Teilungsfaktor des Vorteilers ausgewählt.
Die Taktfrequenz des A/D-Wandlers soll zwischen 50 und 200 kHz liegen! Sie errechnet sich mit:

$$Wandlungstakt = \frac{Systemtakt}{Teilungsfaktor}$$

ADPS2 ADPS1 ADPS0 2^2 2^1 2^0	Teilungs-faktor
000	2
001	2
010	4
011	8
100	16
101	32
110	64
111	128

SFIOR = Special Function IO Register

Bit	7	6	5	4	3	2	1	0
SFIOR 0x30	**ADTS2**	**ADTS1**	**ADTS0**	**-**	**ACME**	**PUD**	**PSR2**	**PSR10**
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W

ADTSn ***ADC Auto Trigger Source*** ***ADTS2, ADTS1, ADTS0***

Falls der Auto Trigger Modus eingeschaltet wurde (**ADATE** im **ADCSRA**-Register) werden diese drei Bit im **SFIOR**-Register benötigt um die Trigger-Quelle auszuwählen. Eine A/D-Wandlung wird durch die steigende Flanke der ausgewählten Interrupt-Quelle ausgelöst.

ADTS2 ADTS1 ADTS0 2^2 2^1 2^0	Trigger-Quelle
000	Free Running Modus
001	Analoger Komparator
010	Externer Interrupt INT0
011	Timer 0 Compare
100	Timer 0 Overflow
101	Timer 1 Compare B
110	Timer 1 Overflow
111	Timer 1 Capture Event

ADCH = ADC Data Register High

Bit	7	6	5	4	3	2	1	0
ADCH 0x05	**- ADC9**	**- ADC8**	**- ADC7**	**- ADC6**	**- ADC5**	**- ADC4**	**ADC9 ADC3**	**ADC8 ADC2**
Startwert	0	0	0	0	0	0	0	0
Read/Write	R	R	R	R	R	R	R	R

ADCL = ADC Data Register Low

Bit	7	6	5	4	3	2	1	0
ADCL 0x04	**ADC7 ADC1**	**ADC6 ADC0**	**ADC5 -**	**ADC4 -**	**ADC3 -**	**ADC2 -**	**ADC1 -**	**ADC0 -**
Startwert	0	0	0	0	0	0	0	0
Read/Write	R	R	R	R	R	R	R	R

ADCn ***ADC Data Register Bit n***

Das Resultat einer A/D-Wandlung befindet sich in diesem 16-Bit Doppelregister. Werden differentielle (symmetrische) Eingänge benutzt, so liegt das Resultat in Zweierkomplementform vor!

ADLAR = 0: Ist das **ADLAR**-Bit im Register **ADMUX** gelöscht, so sind die 10 Bits rechtsbündig angeordnet (obere Zeile). Diese Einstellung ist günstig, wenn alle 10 Bit benötigt werden. Die Wertigkeit der Bits entspricht dann einer 16-Bit Dualzahl. **Das niederwertige Register muss zuerst gelesen werden und dann das hochwertige Register!** Erst nach dem Lesen des hochwertigen Register werden neue Daten in **ADC** abgelegt.

ADLAR = 1: Ist das **ADLAR**-Bit im Register **ADMUX** gesetzt, so sind die 10 Bits linksbündig angeordnet (untere Zeile). Diese Einstellung ist günstig, wenn die zwei niederwertigsten Bit vernachlässigt werden können, man also nur mit 8 Bit weiterarbeitet. Es reicht dann nur **ADCH** auszulesen.

Timer/Counter 0 (5)

Für die Steuerung der Timereinheit sind drei Register zuständig: **TCCR0**, **TIMSK**, **TIFR** und **TIFR**. Das eigentliche Zählregister heißt **TCNT0**. Zusätzlich gibt es ein Vergleichsregister **OCR0**.

TCCR0 = Timer/Counter Control Register 0

Bit	7	6	5	4	3	2	1	0
TCCR0 0x33	**FOC0**	**WGM00**	**COM01**	**COM00**	**WGM01**	**CS02**	**CS01**	**CS00**
Startwert	0	0	0	0	0	0	0	0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

WGM0n ***Waveform Generation Mode*** ***WGM01, WGM00***

Mit diesen zwei Bit wird der Operationsmodus des Timer festgelegt:

WGM01 WGM00 2^1 2^0	Modus:
00	Normaler Modus
01	Phasenkorrekter PWM-Modus
10	CTC-Modus (clear timer on compare match)
11	Fast PWM-Modus

COM0n ***Compare Match Output Mode*** ***COM01, COM00***

Mit diesen zwei Bit wird das Verhalten des Ausgangspin **OC0** festgelegt.
Je nach Modus ändert das Verhalten:

COM01 COM00 2^1 2^0	Verhalten im normalen Modus:
00	**OC0** abgeschaltet (normales Port-Pin)
01	Toggele **OC0** bei Vergleichsübereinstimmung
10	Lösche **OC0** bei Vergleichsübereinstimmung
11	Setze **OC0** bei Vergleichsübereinstimmung
COM01 COM00 2^1 2^0	**Verhalten im Fast-PWM Modus:**
00	**OC0** abgeschaltet (normales Port-Pin)
01	Reserviert
10	nicht-invertierender Fast-PWM-Modus
11	invertierender Fast-PWM-Modus

CS0n ***Clock Select Timer 0*** ***CS02, CS01, CS00***

Mit diesen drei Bit wird die Taktquelle für Timer 0 festgelegt.

CS02 CS01 CS00 2^2 2^1 2^0	Taktquelle:
000	Timer Stopp (verbraucht keinen Strom, default nach RESET!)
001	Systemtakt
010	Systemtakt / 8
011	Systemtakt / 64
100	Systemtakt / 256
101	Systemtakt / 1024
110	externer Takt: fallende Flanke an **T0** (**PB0**)

111	externer Takt: steigende Flanke an T0 (PB0)

TIMSK = Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0
TIMSK 0x39	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

OCIE0 *Timer/Counter 0 Output Compare Match Interrupt Enable*

0 kein OCF0-Interrupt erlaubt.

1 Das Setzen dieses Bit **ermöglicht das Auslösen eines Interrupts** in dem Moment, wo das OCF0-Flag (TIFR) gesetzt wird, also eine Übereinstimmung zwischen dem Zählregister TCNT0 und dem Vergleichsregister OCR0 stattgefunden hat, oder wenn das Flag manuell gesetzt wurde. Interrupts müssen dazu global frei gegeben sein (I=1 im SREG mit "sei").

TOIE0 *Timer/Counter 0 Overflow Interrupt Enable*

0 kein TOV0-Interrupt erlaubt.

1 Das Setzen dieses Bit **ermöglicht das Auslösen eines Interrupts** in dem Moment, wo das TOV0-Flag (TIFR) gesetzt wird, also ein Überlauf auftrat oder wenn das Flag manuell gesetzt wurde. Interrupts müssen dazu global frei gegeben sein (I=1 im SREG mit "sei").

TIFR = Timer/Counter Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0
TIFR 0x38	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

OCF0 *Output Compare Flag 0*

0 Keine Übereinstimmung des Inhalts des Zählregister mit dem Inhalt des Vergleichsregister.

1 Bei Übereinstimmung (*compare match*) des Wertes im im Zählregister TCNT0 mit dem Wert im Vergleichsregister OCR0 wird das Flag gesetzt. Das Flag wird automatisch gelöscht, wenn der OC-Interrupt ausgeführt wird. Manuell kann das Flag durch das **Schreiben einer Eins!** gelöscht werden.

TOV0 *Timer/Counter 0 Overflow Flag*

0 Keine Überlauf eingetreten.

1 Tritt ein Überlauf des Zählregisters TCNT0 auf so wird das Flag gesetzt. Das Flag wird automatisch gelöscht, wenn der OV-Interrupt ausgeführt wird. Manuell kann das Flag durch das **Schreiben einer Eins!** gelöscht werden.

TCNT0 = Timer/Counter Register 0

Bit	7	6	5	4	3	2	1	0
TCNT0 0x32	TCNT07	TCNT06	TCNT05	TCNT04	TCNT03	TCNT02	TCNT01	TCNT00
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

OCR0 = Output Compare Register 0

Bit	7	6	5	4	3	2	1	0
OCR0 0x3C	OCR07	OCR06	OCR05	OCR04	OCR03	OCR02	OCR01	OCR00
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

TWI (I^2C) (5)

Für die I^2C-Einheit werden vier (Master) bzw. fünf (Slave) SF-Register benötigt. Die gesamte Initialisierung wird im **TWI-Kontrollregister** **TWCR** , im **TWI-Bitraten-Register** **TWBR** und in 2 Bit des **TWI-Statusregister** **SPSR** vorgenommen. Mit 5 Bit des TWI-Statusregister kann die korrekte Funktion des Protokolls überwacht werden. Daten werden über das **TWI-Datenregister** **TWDR** ausgetauscht. Im Slave-Modus wird für die Adresse zusätzlich das **TWI-Adress-Register** **TWAR** benötigt.

TWCR = TWI Control Register

Bit	7	6	5	4	3	2	1	0
TWCR 0x36	**TWINT**	**TWEA**	**TWSTA**	**TWSTO**	**TWWC**	**TWEN**	**-**	**TWIE**
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W

TWINT ***TWI Interrupt Flag***

0 es wurde eine Operation gestartet, die noch nicht beendet ist.

1 wird von der Hardware auf Eins gesetzt, sobald eine TWI Operation beendet ist. Gleichzeitig wird ein TWI Interrupt ausgelöst, wenn Interrupts global erlaubt sind (**I** im **SREG**) und das **TWIE**-Bit gesetzt wurde. Durch das Schreiben einer Eins muss das Bit softwaremäßig gelöscht werden! Dies startet auch eine neue Operation. Vor dem Löschen müssen deshalb alle Register, die für die Operation benötigt werden schon beschrieben sein!
Auch beim Verwenden des Interrupts muss das Bit manuell gelöscht werden! Dies geschieht hier nicht automatisch durch die Hardware!

TWEA ***TWI Enable Acknowledge Bit***

0 **sendet ein NACK** nachdem Daten (Slave Adresse, globaler Anruf) angekommen sind.

1 **sendet ein ACK** nachdem Daten (Slave Adresse, globaler Anruf) angekommen sind.

TWSTA ***TWI START Condition Bit***

0 muss per Software wieder zurückgesetzt werden, wenn das Startbit erfolgreich versendet wurde!

1 **sendet eine Startbedingung** als Master falls der Bus frei ist (wartet sonst auf eine Stoppbedingung).

TWSTO ***TWI STOP Condition Bit***

0 wird nach dem Senden einer Stoppbedingung automatisch rückgesetzt.

1 **sendet eine Stoppbedingung** als Master. Kann im Slave-Modus benutzt werden um im Fehlerfall einen neuen definierten Zustand zu erreichen (SCL und SDA hochohmig).

TWCC ***TWI Write Collision Flag***

0 wird automatisch gelöscht wenn Daten in das Datenregister **TWDR** geschrieben werden und keine Operation anhängig ist (**TWINT** = 1).

1 **Kollision!** Zeigt an, dass versucht wurde in das Datenregister **TWDR** zu schreiben obschon die vorherige Operation noch nicht beendet wurde (**TWINT** = 0).

TWEN ***TWI Enable***

0 schaltet die TWI-**Schnittstelle aus** (Baugruppe verbraucht keinen Strom!).

1 schaltet die TWI-**Schnittstelle ein**.

ADIE ***TWI Interrupt Enable***

0 der TWI Interrupt ist gesperrt.

1 Das Setzen dieses Bit **ermöglicht das Auslösen eines Interrupts** in dem Moment, wo die TWI-Operation beendet ist (**TWINT**-Flag = 1 im **TWCR**).

TWSR = TWI Status Register

Bit	7	6	5	4	3	2	1	0
TWSR 0x01	**TWS7**	**TWS6**	**TWS5**	**TWS4**	**TWS3**	**-**	**TWPS1**	**TWPS0**
Startwert	1	1	1	1	1	0	0	0
Read/Write	R	R	R	R	R	R	R/W	R/W

TWS ***TWI Status TWS7-TWS3***

Die 5 Statusbits müssen aus dem Register ausmaskiert werden! Sie geben an ob eine Operation erfolgreich durchgeführt wurde und sollten kontrolliert werden.

Für den **Mastermode** gelten folgende Codes:

TWS	TWSR (maskiert)	Bedeutung
00001	**0x08**	Startbedingung wurde gesendet.
00010	**0x10**	Wiederholte Startbedingung gesendet (*repeated start*).
00011	**0x18**	Slave-Adresse fürs Schreiben gesendet. **ACK** empfangen.
00100	**0x20**	Slave-Adresse fürs Schreiben gesendet. **NACK** empfangen.
00101	**0x28**	Datenbyte gesendet. **ACK** empfangen.
00110	**0x30**	Datenbyte gesendet. **NACK** empfangen.
00111	**0x38**	Datenverlust!
01000	**0x40**	Slave-Adresse fürs Lesen gesendet. **ACK** empfangen.
01001	**0x48**	Slave-Adresse fürs Lesen gesendet. **NACK** empfangen.
01010	**0x50**	Datenbyte empfangen. **ACK** wurde gesendet.
01011	**0x58**	Datenbyte empfangen. **NACK** wurde gesendet.

TWPS ***TWI Prescaler Bits TWPS1,TWPS0***

Diese beiden Bits legen den Vorteiler für die Bitrate fest:

TWPS1	TWPS0	TWPS	Vorteiler (4^{TWPS})
0	0	**0**	**1**
0	1	**1**	**4**
1	0	**2**	**16**
1	1	**3**	**64**

TWBR = TWI Bit Rate Register

Bit	7	6	5	4	3	2	1	0
TWBR 0x00	**TWBR7**	**TWBR6**	**TWBR5**	**TWBR4**	**TWBR3**	**TWBR2**	**TWBR1**	**TWBR0**
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

TWBRn ***TWI Bit Rate Register Bit n***

Im Masterbetrieb errechnet sich die Bitrate mit:

$$f_{SCL} = \frac{\text{Systemtakt}}{16+2\cdot TWBR\cdot \text{Vorteiler}}$$

TWBR ist der Wert des Bitraten-Registers **TWBR**. Der Vorteiler wurde im Statusregister **TWSR** festgelegt. **TWBR** ermittelt man mit:

$$TWBR = \frac{\left(\frac{\text{Systemtakt}}{f_{SCL}} - 16\right)}{2\cdot \text{Vorteiler}}$$

TWDR = TWI Data Register

Bit	7	6	5	4	3	2	1	0
TWDR 0x03	**TWDR7**	**TWDR6**	**TWDR5**	**TWDR4**	**TWDR3**	**TWDR2**	**TWDR1**	**TWDR0**
Startwert	1	1	1	1	1	1	1	1
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

TWDRn ***TWI Data Register Bit n***

Das Datenbyte enthält das letzte zu sendende Byte oder das zuletzt empfangene Byte. Es kann nur beschrieben werden wenn **TWINT** = 1 (**TWCR**).

TWCR = TWI (Slave) Adress Register

Bit	7	6	5	4	3	2	1	0
TWAR 0x02	**TWA6**	**TWA5**	**TWA4**	**TWA3**	**TWA2**	**TWA1**	**TWA0**	**TWGCE**
Startwert	1	1	1	1	1	1	1	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

TWAn ***TWI (Slave) Adress Register Bit n***

Im Slavemodus wird in diesem Register die Slave-Adresse eingetragen

TWGCE ***TWI General Call Recognition Enable Bit***

Erlaubt das Erkennen eines globalen Rundrufs auf dem Bus.

SPI (3)

Für die SPI-Einheit werden nur drei SF-Register benötigt. Die gesamte Initialisierung kann im **SPI-Kontrollregister** **SPCR** vorgenommen werden. Das **SPI-Statusregister** **SPSR** dient hauptsächlich der Abfrage des SPI-Interrupt-Flags **SPIF** beim Polling. Daten werden über das **SPI-Datenregister** **SPDR** in den Sendepuffergeschrieben bzw. aus dem Empfangspuffer gelesen.

SPCR = SPI Control Register

Bit	7	6	5	4	3	2	1	0
SPCR 0x0D	**SPIE**	**SPE**	**DORD**	**MSTR**	**CPOL**	**CPHA**	**SPR1**	**SPR0**
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

SPIE ***SPI Interrupt Enable***

0 kein SPI-Interrupt erlaubt.

1 In dem Moment, wo die Übertragung aller 8 Bit beendet ist wird das SPI Interrupt-Flag **SPIF** im Statusregister **SPSR** gesetzt. Das Setzen des **SPIE**-Flag **ermöglicht das Auslösen eines Interrupts** sobald **SPIF** gesetzt wurde. Interrupts müssen dazu global frei gegeben sein (**I** = 1 im **SREG** mit "`sei`"). **SPIF** wird hardwaremäßig gelöscht wenn der Interrupt abgearbeitet wird.

SPE ***SPI Enable***

0 schaltet SPI aus.

1 schaltet SPI ein. Muss gesetzt werden, damit SPI-Operationen durchgeführt werden.

DORD ***Data ORDer***

0 das höchstwertigste Bit **MSB** wird als Erstes gesendet.

1 das niederwertigste Bit **LSB** wird als Erstes gesendet.

MSTR ***Master/Slave Select***

0 Controller ist Slave.

1 Controller ist Master. Das Bit muss vor oder gleichzeitig mit dem **SPE** Bit gesetzt werden!

CPOL ***Clock Polarity***

0 Ruhezustand der Taktleitung ist Low.

1 Ruhezustand der Taktleitung ist High.

CPHA ***Clock PHAse***

0 Daten werden sofort bei der ersten Flanke übernommen.

1 Daten werden bei der zweiten Flanke nach der halben Taktzeit (180°) übernommen.

Mit **CPOL** und **CPHA** wird der SPI-Modus festgelegt:

CPOL CPHA	SPI Modus:
00	0
01	1
10	2
11	3

SPRn ***SPI Clock Rate Select*** ***SPR2X, SPR1, SPR0***

Mit drei Bit wird die Taktgeschwindigkeit (Frequenz) des Busses **SCK** festgelegt. Dies betrifft nur den

Master. Das Bit **SPR2X** befindet sich im SPI Statusregister **SPSR** (Bit 0).

SPR2X SPR1 SPR0 2^2 2^1 2^0	SCK Frequenz:
000	***Systemtakt / 4***
001	***Systemtakt / 16***
010	***Systemtakt / 64***
011	***Systemtakt / 128***
100	**Systemtakt / 2**
101	**Systemtakt / 8**
110	**Systemtakt / 32**
111	**Systemtakt / 64**

SPSR = SPI Status Register

Bit	7	6	5	4	3	2	1	0
SPSR 0x0E	**SPIF**	**WCOL**	**-**	**-**	**-**	**-**	**-**	**SPI2X**
Startwert	0	0	0	0	0	0	0	0
Read/Write	R	R	R	R	R	R	R	R/W

SPIF ***SPI Interrupt Flag***

0 Übertragung noch nicht beendet.

1 In dem Moment, wo die Übertragung aller 8 Bit beendet ist wird das SPI Interrupt-Flag **SPIF** gesetzt. Das Setzen des **SPIE**-Flag in **SPCR** ermöglicht das Auslösen eines Interrupts. **SPIF** wird hardwaremäßig gelöscht wenn der Interrupt abgearbeitet wird. Wird **SPIF** im Polling-Betrieb abgefragt, so kann das Flag durch Lesen des Statusregisters **SPSR** bei gesetztem **SPIF** und anschließendem Lesen oder Schreiben des Datenregisters **SPDR** gelöscht werden.

WCOL ***Write COLlision Flag***

0 keine Schreibkollision aufgetreten.

1 Wird **SPDR** während der Datenübertragung beschrieben, so meldet das **WCOL**-Flag eine Schreibkollision. Das Flag kann (zusammen mit **SPIF**) durch Lesen des Statusregisters **SPSR** bei gesetztem **WCOL** und anschließendem Lesen oder Schreiben des Datenregisters **SPDR** gelöscht werden.

SPI2X ***Double SPI Speed Bit***

0 keine Verdopplung der Taktgeschwindigkeit des Busses.

1 Verdopplung der Taktgeschwindigkeit des Busses (siehe Tabelle **SPCR**).

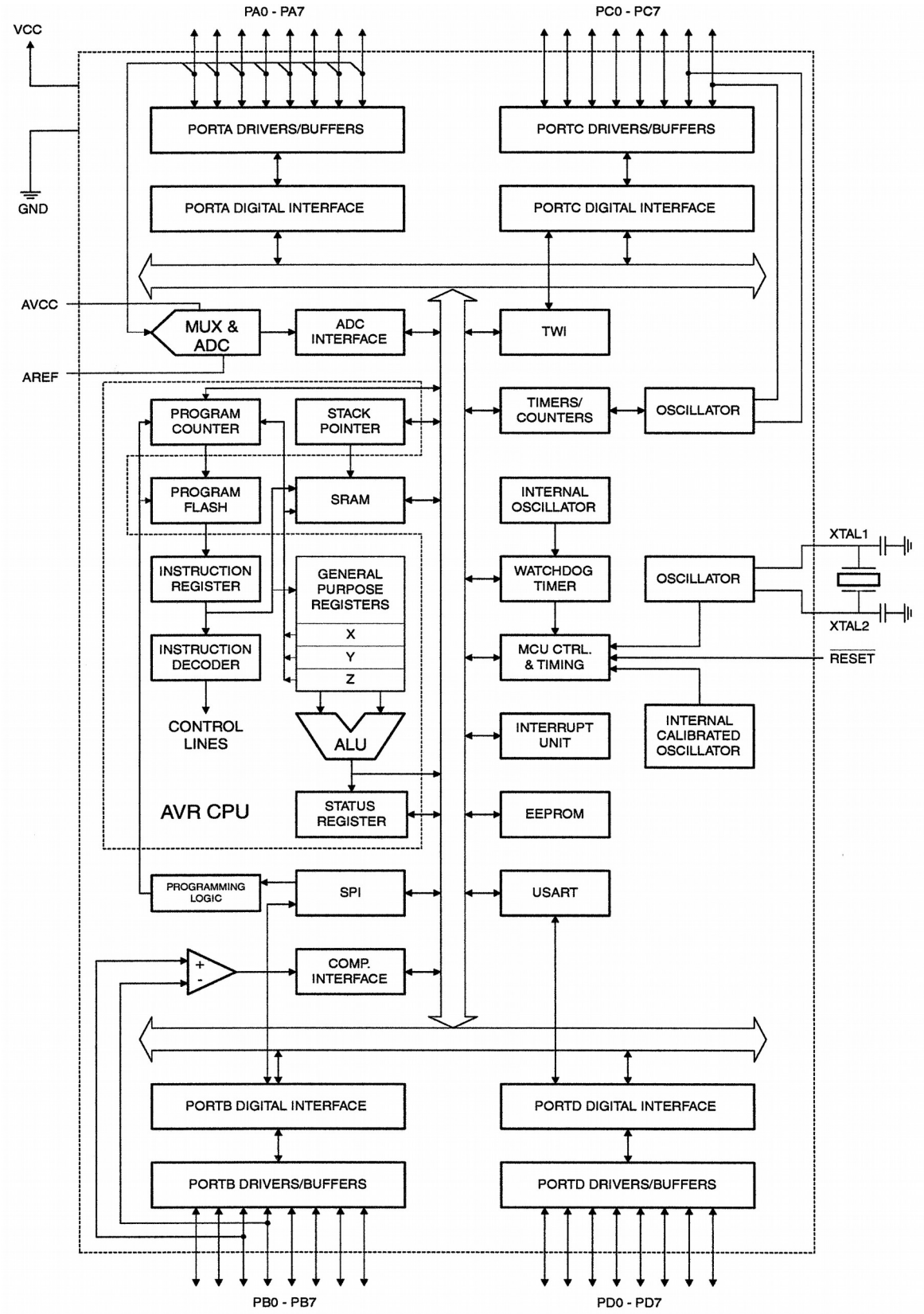
SPDR = SPI Data Register

Bit	7 (MSB)	6	5	4	3	2	1	0 (LSB)
SPDR 0x0F	**SPDR7**	**SPDR6**	**SPDR5**	**SPDR4**	**SPDR3**	**SPDR2**	**SPDR1**	**SPDR0**
Startwert	-	-	-	-	-	-	-	-
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Der Startwert ist undefiniert.

Blockschaltbild des ATmega32A

(Quelle: Atmel® Datenblatt ATmega32A)



F1 Erstes Programm

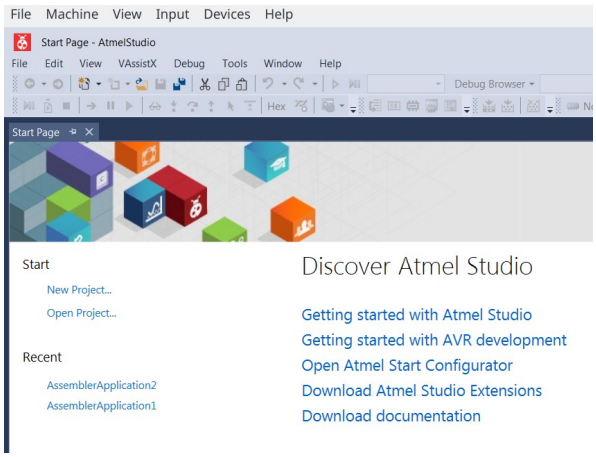
In diesem Kapitel wird erklärt wie man möglichst schnell mit dem Programm Studio 7 von Atmel® und einem ISP-Programmiergerät einen ATmega Controller programmiert. Es ist kein umfassendes Tutorial zum Programm Studio 7.

ATMEL® Studio 7

Studio 7 von ATMEL® ist eine umfassende Entwicklungsumgebung, um Programme für die AVR-Familie in Assembler und C zu erstellen und zu testen. Mit einem Suchprogramm und den Schlüsselwörtern "Atmel" und "Studio" findet man schnell die richtige Seite auf der ATMEL®-Homepage (atmel.com) im Internet.

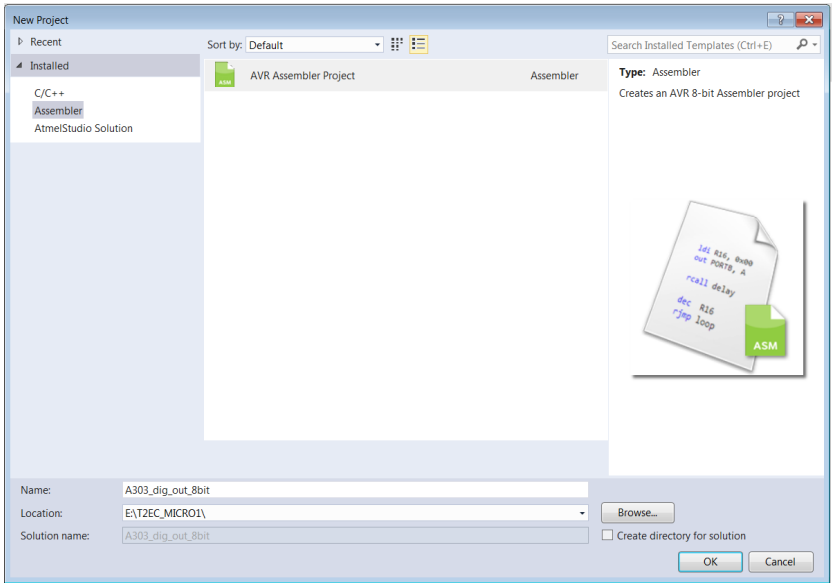
Nach der Registrierung kann das Programm auf den heimischen PC gesaugt und entpackt werden. Der USB-Treiber muss installiert werden, damit das Programmiergerät richtig funktioniert.

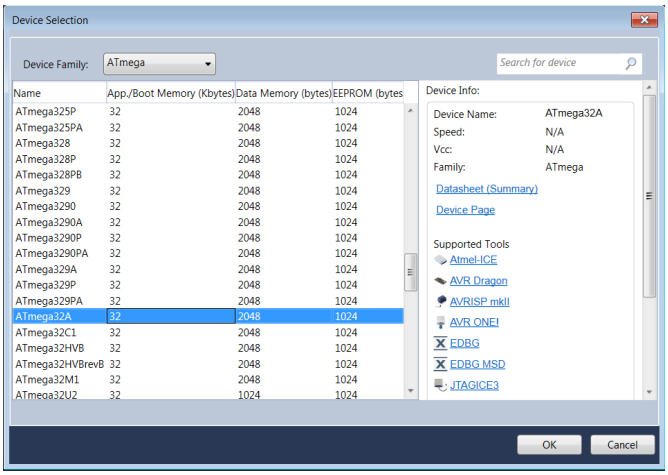
Nach einem Doppelklick auf ein angelegtes Desktop-Icon wählen wir auf der Startseite "Start New Project...".



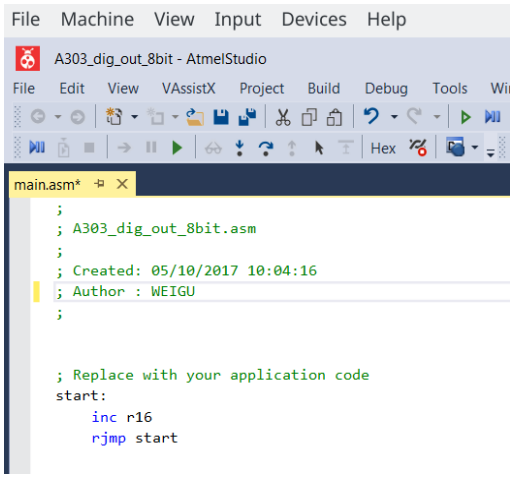
Im neuen Fenster (*New Project*) wählen wir Assembler aus, geben dem Projekt den einen Namen (hier **`A303_dig_out_8bit`**, da das Projekt unser erstes Assemblerprogramm enthalten soll). Um die Struktur 4einfach zu halten, soll kein "*solution"* Unterverzeichnis erstellt werden (ausklicken!). Unter "*Location*" kann das Verzeichnis wo das ganze abgespeichert werden soll ausgewählt werden (USB-Stick!).

Im nächsten Fenster (nach dem Anklicken von "*OK*") wählen wir den verwendeten Baustein (ATmega32 oder ATmega32A) aus.

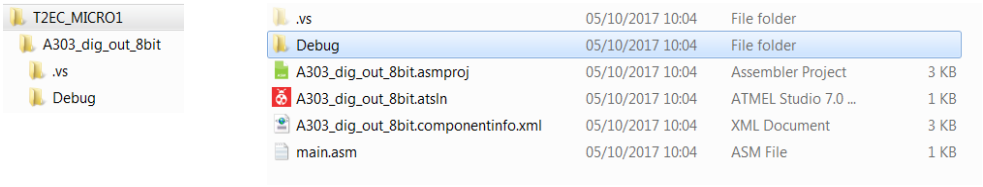




Nach dem nochmaligen Klicken von "*OK*" wird das Projekt erstellt, und wir befinden uns im Editorfenster des Assemblerprogramms. Das Hauptprogramm hat dabei den Titel **`main.asm`**.

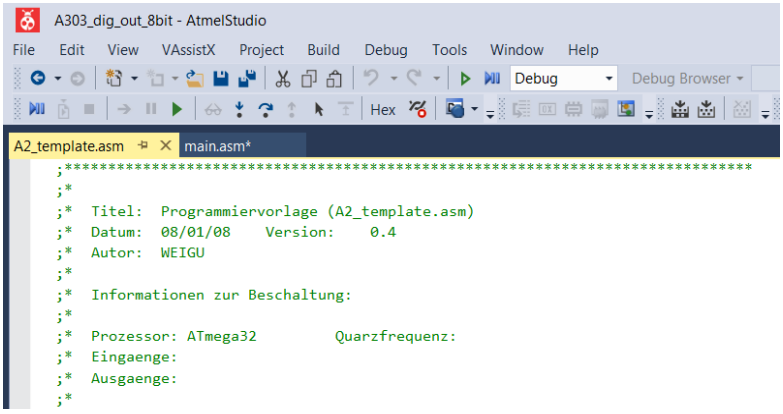


Da wir das Projekt im Verzeichnis "T2EC_MICRO1" abgespeichert haben sieht die Struktur des Projektes jetzt folgendermaßen aus:



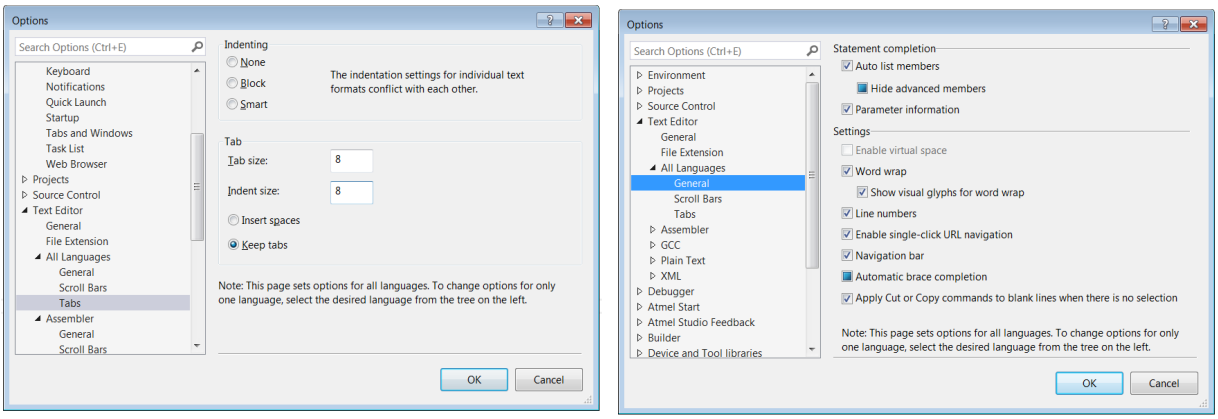
Das Assemblerprogramm befindet sich in der reinen Textdatei **`main.asm`**. Die 3 anderen Textdateien (xml) benötigt Studio7 zur Projektbeschreibung. Ein Klick auf die Datei mit der Endung ".asmproj" öffnet das Projekt (Alternativ kann man die Datei mit der Endung ".asmproj"(Atmel Studio 7 Solution File) auswählen, allerdings muss man dann die Assemblerdatei manuell auswählen). Das "*Debug*"-Verzeichnis ist momentan leer. Es wird nach dem Assemblieren die Ausgabedateien enthalten

Mit "*File/Open*" öffnen wir die Datei **`A2_template.asm`**[6]. Wir kopieren die Vorlage in unsere Datei **`main.asm`** und Schließen die Datei wieder.

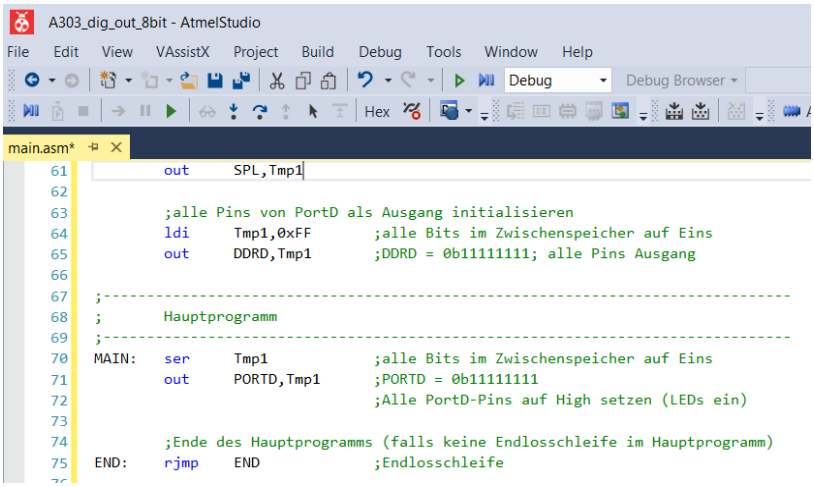


Um eine anständige Darstellung zu ermöglichen ist es jetzt an der Zeit im Menü "*Tools/Options*" den Tabulatorabstand auf 8 Zeichen zu setzen und die Zeilennummerierung einzuschalten. Nur so werden die Abstände aus der Vorlage richtig eingehalten.

6 Download von http://weigu.lu/tutorials/avr_assembler

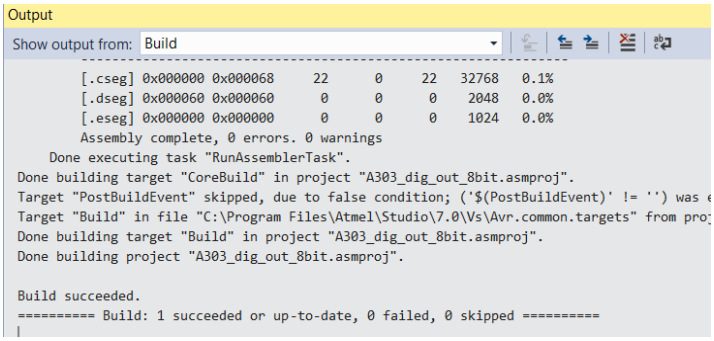


Im nächsten Schritt editieren wir die Vorlage und ergänzen den Initialisierungsteil und die Hauptschleife. Das Einbinden der AVR-Definitionsdatei ist in Studio 7 nicht mehr nötig, da dies automatisch geschieht (Die Zeile **`.INCLUDE "m32def.inc"`** muss auskommentiert bleiben).



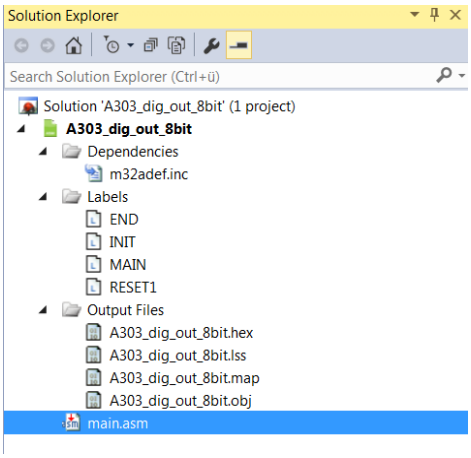
Ist das Programm fertiggestellt, so kann es assembliert werden. Dazu kann man eines der beiden "*Build*"-Icon nutzen  oder **F7** drücken (alternativ: Menupunkt "*Build* ")

War das Übersetzen erfolgreich, so ist dies im "*Output*"-Fenster sichtbar. Bei etwaigen Fehlern werden diese im Ausgabefenster angezeigt.



Im "*Solution Explorer*" (rechte Seite) sieht man, welche Dateien eingebunden wurden, und welche Ausgabe-Dateien erzeugt wurden.

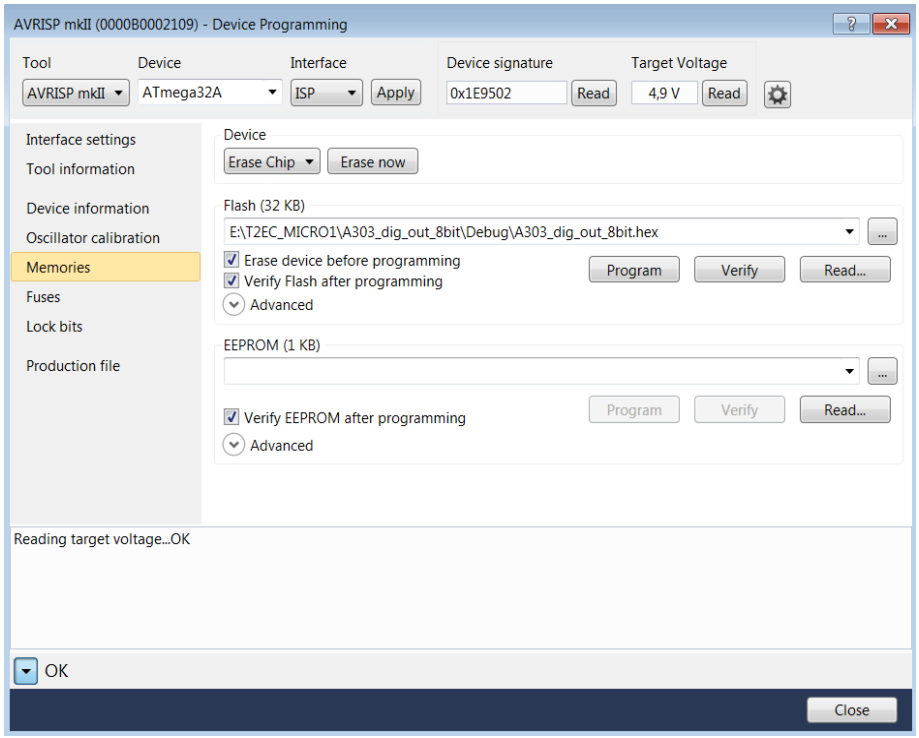
Die Ausgabe-Dateien befinden sich alle im "*Debug*"-Verzeichnis.



Mit dem "*Debug*"-Icon  oder **F5** (alternativ: Menupunkt "*Debug*") wird das Programm zum Mikrocontroller geschickt.

Benötigt man weitere Informationen, so kann man das "*Device Programming*"-Icon  (**CTRL+SHIFT+P**) nutzen (Menupunkt "*Tools*"). Das "*Device Programming*"-Fenster erlaubt es das Programmiergerät auszuwählen (*Apply*), die Verbindung zum Controller (*Device Signature*) zu testen und die Spannung zu messen.

Unter "*Memories*" kann die zu programmierende Hex-Datei ausgewählt werden. Auch können die Fuse und Lock-Bits gesetzt werden (siehe entsprechendes Kapitel).

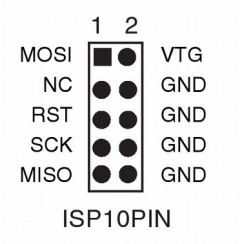


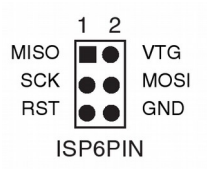
Hardware

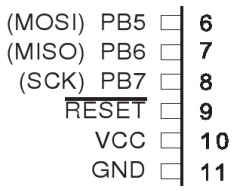
Zum Programmieren des ATmega-Controllers benötigen wir ein Programmiergerät. Dieses soll an eine serielle Schnittstelle oder noch besser an eine USB-Schnittstelle anschließbar sein, da diese Schnittstellen vom Studio-Programm unterstützt werden. Hierzu kann man zum Beispiel den originalen AVRISP mkII (Bild, Quelle ATMEL) oder das neue AVR Dragon-Board von ATMEL verwenden (beide USB). Aber auch das Entwicklungsboard STK500 oder der ältere AVRISP (beide seriell) eignen sich dazu.

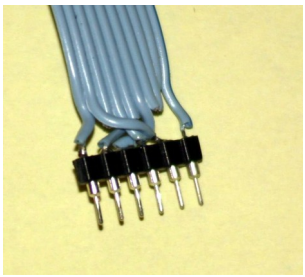
In einem nächsten Kapitel wird ein komplettes Entwicklungsboard mit integriertem USB Programmiergerät vorgestellt. Auf der Seite http://weigu.lu/microcontroller/avr_isp_mk2 ist zudem beschrieben, wie man leicht ein eigenes Programmiergerät bauen kann.

Besitzt man kein eigenes Board, um den Chip aufzunehmen, so kann man sich mit einem Sockel und einer Lochrasterplatine behelfen. Die einfachste Lösung ist jedoch die, eine DIL-Steckplatine zu verwenden. Da beim ATmega32A alle zum Programmieren und zur Stromversorgung benötigten Pins gleich nebeneinander liegen, benötigt man ein einfaches Adapterkabel, das die 6 benötigten Pins des Atmel-Programmiersteckers mit einer Stiftleiste (oder ein Stück IC-Sockel) verbindet.





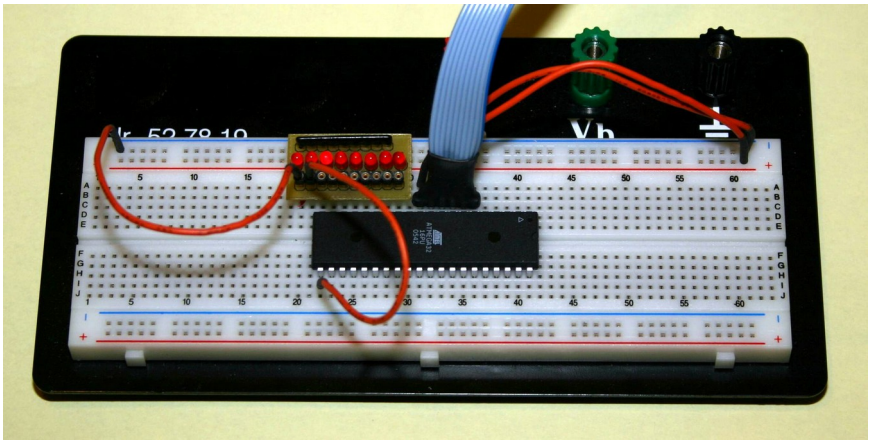




Stecker- und Pinbelegung Quelle: Atmel

Liefert das Programmiergerät nicht ausreichend Strom (AVRISP mkII, AVR Dragon-Board) verwenden, so muss eine zusätzliche Stromversorgung verwendet werden.

Die Schaltung für unser erstes bzw. zweites Programm kann dann folgendermaßen aussehen:



Bemerkungen: Es wurde eine hier eine kleine Platine für die 8 LEDs mit Vorwiderständen benutzt. Diese können natürlich auch ohne Platine verdrahtet werden. Einen Taster (für das dritte Programm) kann man einfach mit einer Steckverbindung gegen Masse simulieren. Es wird hier (noch) kein externer Quarz verwendet, sondern der interne Oszillator mit 1 MHz.

F2 Assembleranweisungen

Die Assembler-**Direktiven** sind Assembleranweisungen, die dem Assembler mitteilen was dieser beim Assemblieren tun soll. Sie sind nicht zu verwechseln mit den Befehlen.

Direktiven werden nicht in Maschinencode übersetzt (Pseudo-Befehle!), sondern dienen nur der Steuerung des Übersetzungsvorgangs. Sie beginnen mit einem Punkt und stehen meist gleich am Anfang der Zeile.

Folgende Direktiven werden verwendet:

Direktive	Operand	Beschreibung
.LIST		Listing-Ausgabe einschalten (default). Der produzierte Code wird von Menschen lesbar in einer ***.LST**-Datei ausgegeben.
.NOLIST		Listing-Ausgabe ausschalten.
.INCLUDE	"**Textdatei**" oder <**Textdatei**>	Fügt eine externe Textdatei ein (als ob deren Inhalt an dieser Stelle stünde). Es kann sich hierbei z.B. um einen Programmteil in Assembler, ein Unterprogramm oder eine Definitionsdatei (Header-Datei) mit zusätzliche Direktiven sein. Beispiel: **.INCLUDE <m32def.inc>**
.DEVICE	Bausteintyp	Definiert den Bausteintyp. Wird nicht benötigt, da schon in der Definitionsdatei enthalten.
.DEF	Name = Register	Definiere (engl.:“define“) einen Namen für ein Arbeitsregister (Variable!, r0 bis r31). Beispiel: **.DEF Tmp1 = r16**
.EQU	Name = Ausdruck	Definiert eine Konstante mit Namen. Dieser Name ist dann nicht mehr veränderbar. Beispiel: **.EQU Clock = 1000**
.SET	Name = Ausdruck	Definiert eine Konstante mit Namen. Dieser Name ist innerhalb des Programms (Übersetzungsvorgangs) veränderbar. Beispiel: **.SET Value = 500 ;alter Wert** **.SET Value = 200 ;neuer Wert**
.ORG	Adresse	Legt eine Anfangsadresse fest ab der der folgende Code abgespeichert wird. Hiermit kann der Speicher organisiert werden. Beispiel: **.ORG = 0xA00**
.EXIT		Ende des Quelltextes

Weiter Direktiven für die Organisation des Speicher-Bereichs:

Direktive	Operand	Beschreibung
.CSEG		Beginn des Codesegmentes. Alles Folgende wird als Code übersetzt und im Flash gespeichert
.DSEG		Beginn des Datensegmentes. Mit **.BYTE**-Direktiven und Labels wird hier der SRAM-Speicher organisiert.
.ESEG		Beginn des EEPROM-Segments. Mit **.DB** und **.DW**-Direktiven werden Variablen im EEPROM abgelegt.
.DB	`Liste mit Bytekonstanten`	(engl.: „define Byte“) Fügt konstante Bytes ein. Dabei ist es die Bedeutung der Bytes egal (Zahl von 0..255, ASCII-Zeichen 'b', eine Zeichenkette "Hallo"; alle Bytes werden durch Kommas getrennt). Im Flash muss eine gerade Zahl von Bytes eingefügt werden (16 Bit-Worte), sonst hängt der Assembler ein Nullbyte an.
.DW	`Liste mit Wortkonstanten`	(engl.: „define Word“) Fügt konstantes binäres Wort (16 Bit) ein. Im EEPROM zuerst das niederwertige Byte, dann das höherwertige Byte.
.BYTE	`Anzahl`	Reserviert Speicherplatz (Bytes) im SRAM (Datensegment).

F3 ASCII-Tabelle

Dec.	Oct.	Hex	Binary	Value		Dec.	Oct.	Hex	Binary	Value
000	000	000	00000000	NUL	(Null char.)	064	100	040	01000000	@
001	001	001	00000001	SOH	(Start of Header)	065	101	041	01000001	A
002	002	002	00000010	STX	(Start of Text)	066	102	042	01000010	B
003	003	003	00000011	ETX	(End of Text)	067	103	043	01000011	C
004	004	004	00000100	EOT	(End of Transmission)	068	104	044	01000100	D
005	005	005	00000101	ENQ	(Enquiry)	069	105	045	01000101	E
006	006	006	00000110	ACK	(Acknowledgment)	070	106	046	01000110	F
007	007	007	00000111	BEL	(Bell)	071	107	047	01000111	G
008	010	008	00001000	BS	(Backspace)	072	110	048	01001000	H
009	011	009	00001001	HT	(Horizontal Tab)	073	111	049	01001001	I
010	012	00A	00001010	LF	(Line Feed)	074	112	04A	01001010	J
011	013	00B	00001011	VT	(Vertical Tab)	075	113	04B	01001011	K
012	014	00C	00001100	FF	(Form Feed)	076	114	04C	01001100	L
013	015	00D	00001101	CR	(Carriage Return)	077	115	04D	01001101	M
014	016	00E	00001110	SO	(Shift Out)	078	116	04E	01001110	N
015	017	00F	00001111	SI	(Shift In)	079	117	04F	01001111	O
016	020	010	00010000	DLE	(Data Link Escape)	080	120	050	01010000	P
017	021	011	00010001	DC1	(XON) (Device Control 1)	081	121	051	01010001	Q
018	022	012	00010010	DC2	(Device Control 2)	082	122	052	01010010	R
019	023	013	00010011	DC3	(XOFF)(Device Control 3)	083	123	053	01010011	S
020	024	014	00010100	DC4	(Device Control 4)	084	124	054	01010100	T
021	025	015	00010101	NAK	(Negative Acknowledgement)	085	125	055	01010101	U
022	026	016	00010110	SYN	(Synchronous Idle)	086	126	056	01010110	V
023	027	017	00010111	ETB	(End of Trans. Block)	087	127	057	01010111	W
024	030	018	00011000	CAN	(Cancel)	088	130	058	01011000	X
025	031	019	00011001	EM	(End of Medium)	089	131	059	01011001	Y
026	032	01A	00011010	SUB	(Substitute)	090	132	05A	01011010	Z
027	033	01B	00011011	ESC	(Escape)	091	133	05B	01011011	[
028	034	01C	00011100	FS	(File Separator)	092	134	05C	01011100	\
029	035	01D	00011101	GS	(Group Separator)	093	135	05D	01011101	]
030	036	01E	00011110	RS	(Req to Send)(Rec Sep)	094	136	05E	01011110	^
031	037	01F	00011111	US	(Unit Separator)	095	137	05F	01011111	_
032	040	020	00100000	SP	(Space)	096	140	060	01100000	`
033	041	021	00100001	!		097	141	061	01100001	a
034	042	022	00100010	"		098	142	062	01100010	b
035	043	023	00100011	#		099	143	063	01100011	c
036	044	024	00100100	$		100	144	064	01100100	d
037	045	025	00100101	%		101	145	065	01100101	e
038	046	026	00100110	&		102	146	066	01100110	f
039	047	027	00100111	'		103	147	067	01100111	g
040	050	028	00101000	(		104	150	068	01101000	h
041	051	029	00101001	)		105	151	069	01101001	i
042	052	02A	00101010	*		106	152	06A	01101010	j
043	053	02B	00101011	+		107	153	06B	01101011	k
044	054	02C	00101100	,		108	154	06C	01101100	l
045	055	02D	00101101	-		109	155	06D	01101101	m
046	056	02E	00101110	.		110	156	06E	01101110	n
047	057	02F	00101111	/		111	157	06F	01101111	o
048	060	030	00110000	0		112	160	070	01110000	p
049	061	031	00110001	1		113	161	071	01110001	q
050	062	032	00110010	2		114	162	072	01110010	r
051	063	033	00110011	3		115	163	073	01110011	s
052	064	034	00110100	4		116	164	074	01110100	t
053	065	035	00110101	5		117	165	075	01110101	u
054	066	036	00110110	6		118	166	076	01110110	v
055	067	037	00110111	7		119	167	077	01110111	w
056	070	038	00111000	8		120	170	078	01111000	x
057	071	039	00111001	9		121	171	079	01111001	y
058	072	03A	00111010	:		122	172	07A	01111010	z
059	073	03B	00111011	;		123	173	07B	01111011	{
060	074	03C	00111100	<		124	174	07C	01111100	|
061	075	03D	00111101	=		125	175	07D	01111101	}
062	076	03E	00111110	>		126	176	07E	01111110	~
063	077	03F	00111111	?		127	177	07F	01111111	Nul

F4 Flussdiagramme

(Programmablaufplan PAP)

Die Software-Erstellung setzt sich aus folgenden Phasen zusammen:

1. Durchführung einer **Problemanalyse**
2. Anfertigen eines **Flussdiagramms (flow chart)**
3. **Kodieren** des Programms
4. Programm**dokumentation**
5. **Programmtest** mit Korrekturen

Das Flussdiagramm (flow Chart) erfasst alle zur Lösung einer Aufgabe nötigen Teiloperationen in übersichtlicher Darstellung und erforderlicher Reihenfolge.

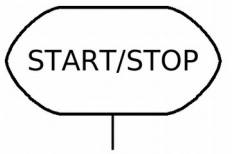
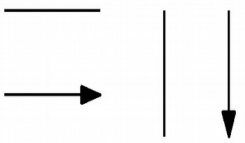
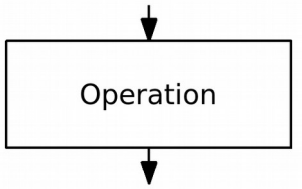
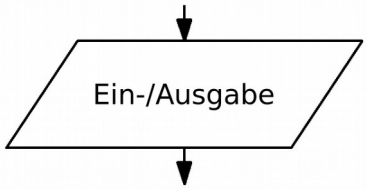
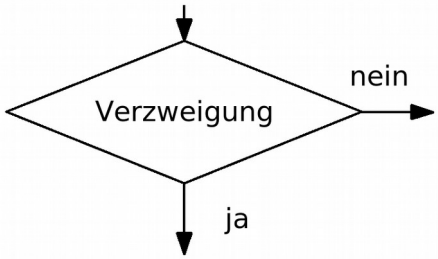
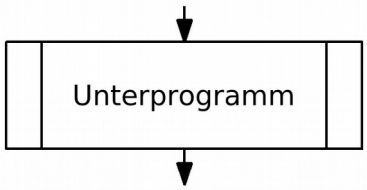
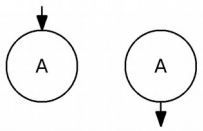
Bei großen Programmen arbeitet man mit groben Ablaufplänen, welche durch feinere Pläne für die einzelnen Programmschritte ergänzt werden können. In unseren kleinen Aufgaben arbeiten wir zum besseren Verständnis mit recht detaillierten Flussdiagrammen, wobei manchmal sogar ein Befehl als ein Symbol erscheint. Diese lassen sich auch einfacher ins endgültige Programm umsetzen.

Flussdiagramme sollen in Textform so allgemein wie nur möglich gehalten werden, damit die Programme mit (fast) jedem beliebigen Prozessor bzw. Controller umgesetzt werden können.

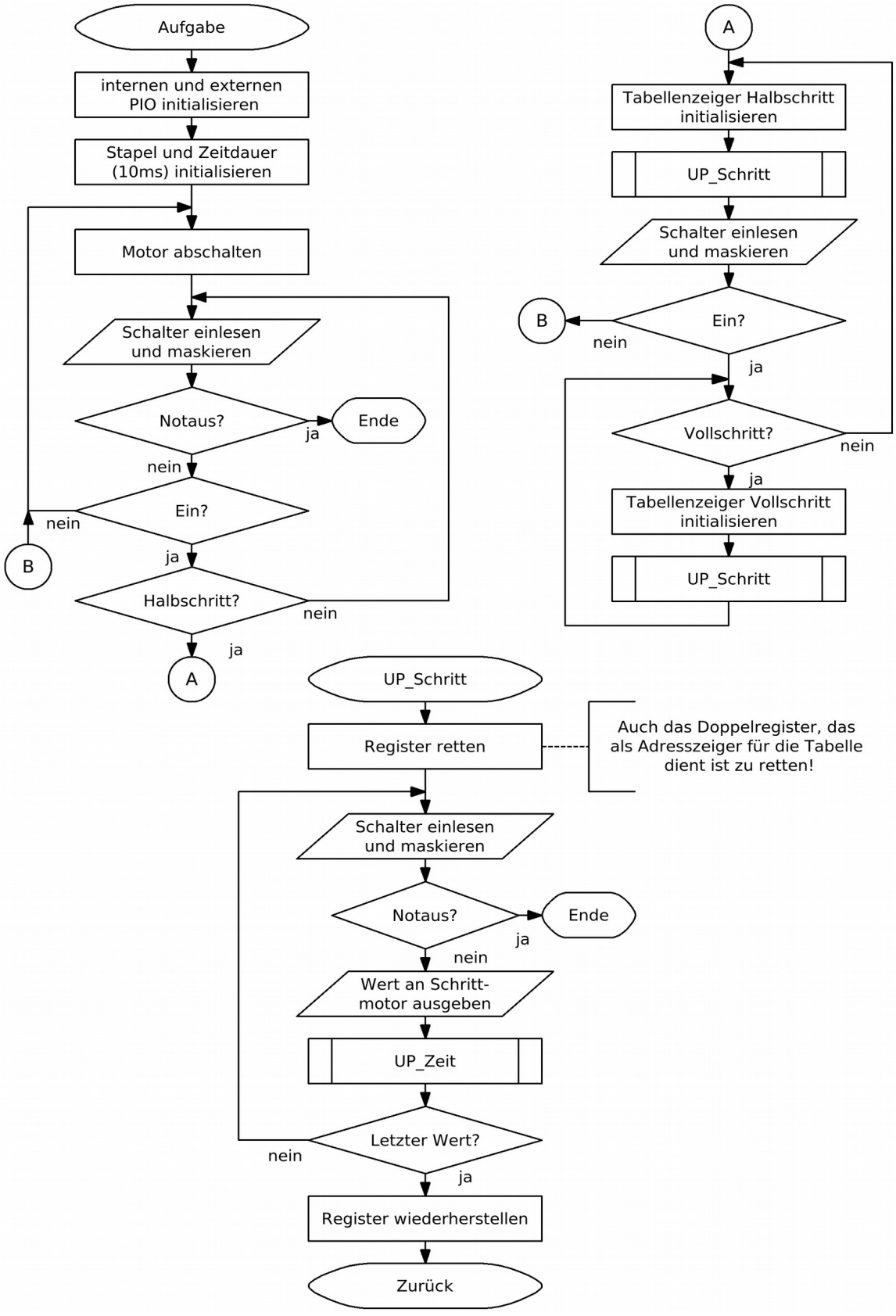
Zum besseren Verständnis werden allerdings manchmal in den Flussdiagrammen Funktionen der Befehle mit spezifischen Registern und Zuweisungen in Kommentarform zufügen.

Das Flussdiagramm besteht aus unterschiedlichen Sinnbildern (Blöcken), die durch Richtungspfeile untereinander verbunden werden.

Für Sinnbilder der Flussdiagramme sind genormt (DIN66001). Es werden in diesem Kurs folgende Sinnbilder verwendet:

	**Grenzstelle (Oval):** **Anfang- oder Ende des Programms bzw. andere Grenzpunkte**
	**Ablauflinie, Pfeillinie:** **Verbindung zum nächstfolgenden Element. Es ist besser immer Pfeile zur Richtungsangabe zu setzen.**
	**Operation allgemein (Rechteck):** **Beispiel: Rechenoperation, Datentransport...**
	**Ein-/Ausgabe-Operation (Parallelogramm):** **Ein-/Ausgabe über die Peripherie.**
	**Verzweigung (Raute):** **Nach einer Entscheidung bieten sich zwei Pfade an die mit "ja" (wahr, 1), und "nein" (unwahr, 0) bezeichnet werden.**
	**Unterprogramm (Rechteck doppelt vert. Linien):** **Aufruf eines Unterprogramms, der selbst wieder als Flussdiagramm zu dokumentieren ist.**
	**Übergangsstelle (Kreis):** **Verbindung von Ablauflinien (z.B. über mehrere Seiten)**
	**Bemerkung:** **Kann an alle Symbole angehängt werden um die Operation genauer zu dokumentieren.**

Beispiel für ein Flussdiagramm:



F5 Fuse- und Lock-Bits

Im AVR gibt es eine Reihe von programmierbaren Schaltern, die wichtige Eigenschaften des Controller festlegen. Diese Schalter können nur mit dem Programmiergerät verändert werden (Ausnahme: Bootloader-Programm)!

Beim Programmieren dieser Schalter ist äußerste Vorsicht geboten!!

Gesetzt (*programmed*) werden die Schalter mit Null, gelöscht (*unprogrammed*) mit einer Eins!

Werden diese Schalter falsch programmiert, so kann der Controller unter Umständen nicht mehr mit einem normalen Programmiergerät[7] angesprochen werden!

Fuse-Bits des ATMega32A

Beim ATmega32A, existieren 16 programmierbare Fuse-Bits (2 Byte). Wichtige Fuse-Bits beim höherwertigen Byte sind das **JTAGEN**- und das **SPIEN**-Bit.

Table 26-3. Fuse High Byte

Fuse High Byte	Bit No.	Description	Default Value
OCDEN(4)	7	Enable OCD	1 (unprogrammed, OCD disabled)
JTAGEN(5)	6	Enable JTAG	0 (programmed, JTAG enabled)
SPIEN(1)	5	Enable SPI Serial Program and Data Downloading	0 (programmed, SPI prog. enabled)
CKOPT(2)	4	Oscillator options	1 (unprogrammed)
EESAVE	3	EEPROM memory is preserved through the Chip Erase	1 (unprogrammed, EEPROM not preserved)
BOOTSZ1	2	Select Boot Size (see Table 25-6 for details)	0 (programmed)(3)
BOOTSZ0	1	Select Boot Size (see Table 25-6 for details)	0 (programmed)(3)
BOOTRST	0	Select reset vector	1 (unprogrammed)

Notes:
1. The SPIEN Fuse is not accessible in SPI Serial Programming mode.
2. The CKOPT Fuse functionality depends on the setting of the CKSEL bits. See See “Clock Sources” on page 26. for details.
3. The default value of BOOTSZ1:0 results in maximum Boot Size. See Table 25-6 on page 263.
4. Never ship a product with the OCDEN Fuse programmed regardless of the setting of Lock bits and the JTAGEN Fuse. A programmed OCDEN Fuse enables some parts of the clock system to be running in all sleep modes. This may increase the power consumption.
5. If the JTAG interface is left unconnected, the JTAGEN fuse should if possible be disabled. This to avoid static current at the TDO pin in the JTAG interface.

7 Mit Hilfe eines speziellen Programmiergerätes das den parallelen Brennmodus beherrscht (STK 500, AVR-Dragon) kann der Controller wieder zurückgesetzt werden.

JTAGEN sollte gelöscht werden, da sonst Probleme mit **PC2-PC5** (**TCK**, **TMS**, **TDO**, **TDI**) auftreten können. Ist **JTAGEN** gesetzt und das **JTD**-Bit im **MCUCSR**-Register gelöscht, so werden diese Pins mit einem internen Pull-Up-Widerstand auf High-Pegel gezogen. Werden sie dann gleichzeitig als Ausgangspin betrieben, so entsteht ein Spannungsteiler mit der Ausgangslast und die Ausgangsspannung hängt vom Widerstandsverhältnis ab.

SPIEN muss unbedingt gesetzt bleiben, da sonst das Programmieren über die SPI-Schnittstelle nicht mehr funktioniert.

CKOPT ist für die Oszillator-Verstärkung zuständig. Standardmäßig ist **CKOPT** nicht programmiert, also gelöscht (weniger Strom, geringere Störfestigkeit, geringere Bandbreite; siehe Datenblatt S 27). Diese Option ist allerdings nur bis 8 MHz zulässig! Wird ein Quarz mit höherer Frequenz eingesetzt, so muss **CKOPT** gesetzt werden!

Beim niederwertigen Byte sind **SUT0** und **SUT1** sowie **CKSEL0**-**CKSEL3** von Interesse. **SUT** steht für *Start-Up Time* und diese zwei Bit zusammen mit **CKSEL0** ermöglichen festzulegen, wie viele Taktzyklen der Controller nach dem Abschalten benötigt, um Einsatzbereit zu sein. Dies hängt natürlich vom verwendeten Quarz ab (Datenblatt S 28).

Table 26-4. Fuse Low Byte

Fuse Low Byte	Bit No.	Description	Default Value
BODLEVEL	7	Brown-out Detector trigger level	1 (unprogrammed)
BODEN	6	Brown-out Detector enable	1 (unprogrammed, BOD disabled)
SUT1	5	Select start-up time	1 (unprogrammed)(1)
SUT0	4	Select start-up time	0 (programmed)(1)
CKSEL3	3	Select Clock source	0 (programmed)(2)
CKSEL2	2	Select Clock source	0 (programmed)(2)
CKSEL1	1	Select Clock source	0 (programmed)(2)
CKSEL0	0	Select Clock source	1 (unprogrammed)(2)

Notes: 1. The default value of SUT1:0 results in maximum start-up time. SeeTable 8-9 on page 30 for details.
2. The default setting of CKSEL3:0 results in internal RC Oscillator @ 1MHz. See Table 8-1 on page 26 for details.

CKSEL0-**CKSEL3** ermöglicht es die Frequenz festzulegen. Bei der Auslieferung ist **SUT** = **0b10** (langsamste Aufwachzeit) und **CSEL** = **0b0001** (interner RC-Oszillator 1 MHz) eingestellt.

Table 8-1. Device Clocking Options Select(1)

Device Clocking Option	CKSEL3:0
External Crystal/Ceramic Resonator	1111 - 1010
External Low-frequency Crystal	1001
External RC Oscillator	1000 - 0101
Calibrated Internal RC Oscillator	0100 - 0001
External Clock	0000

Der Default Werte für die Fuse-Bytes bei der Auslieferung des Chips ist **0x99E1**.

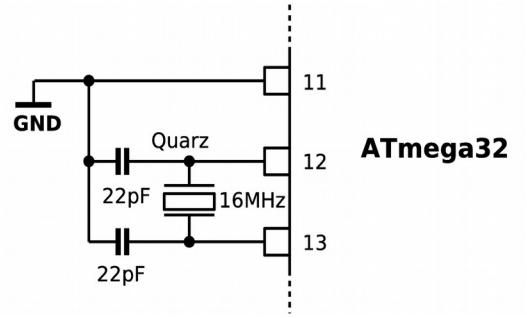
Mit Hilfe eines Online-Rechners unter **http://palmavr.sourceforge.net/cgi-bin/fc.cgi** kann man die Fuse-Bits einfach berechnen. Dies ist besonders hilfreich, wenn man ein anderes Programm als Studio 4 zur Programmierung der Fuse-Bits benutzt.

Nicht alle Fuse-Bits werden nach dem Brennen sofort wirksam. Nach dem Ändern der Fuse- oder Lock-Bits soll deshalb die Betriebsspannung aus- und wieder eingeschaltet werden.

Bemerkung: Ein Löschen des Chips beeinflusst die Fuse- und Lock-Bits nicht.

Externer Quarz mit 16 MHz

Aus dem Datenblatt lässt sich entnehmen, dass zwei Kondensatoren mit 12-22pF zu wählen sind. Beide sollen natürlich genau gleich sein. Anzuschließen ist der Quarz nach folgendem Schema:



Da eine langsame Aufwachzeit hier reicht, wird SUT = 0b11 ausgewählt. Für die hohe Frequenz wird CSEL = 0b1111 eingestellt. Dies lässt sich im Studio 4 im Programmier-Fenster (Menü "Tools/Programm AVR/Auto Connect..." oder Klick auf das entsprechende Icon (AVR)) einfach durch Auswählen von „*Ext. Crystal/Resonator High Freq.; Start-up time: 16K CK + 64 ms*“ unter SUT_CKSEL im "Fuses"-Reiter markieren (siehe Screenshot).

CKOPT muss für den 16 MHz-Quarz programmiert werden (Häkchen setzen)!

Table 8-3. Crystal Oscillator Operating Modes

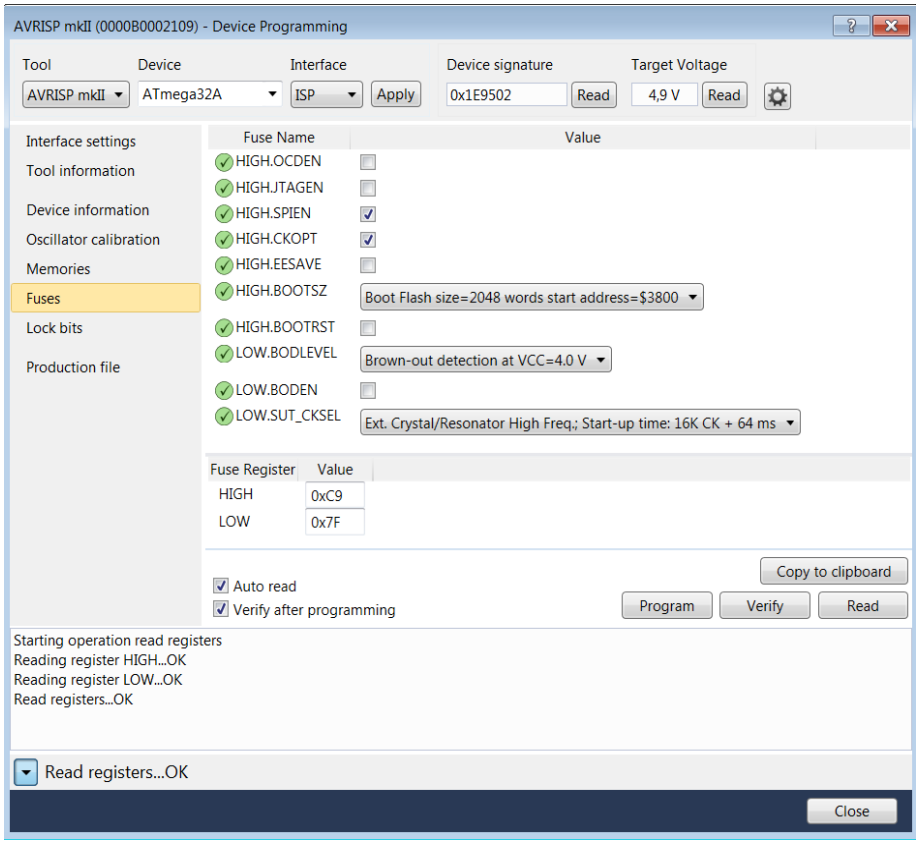
CKOPT	CKSEL3:1	Frequency Range (MHz)	Recommended Range for Capacitors C1 and C2 for Use with Crystals (pF)
1	101(1)	0.4 - 0.9	–
1	110	0.9 - 3.0	12 - 22
1	111	3.0 - 8.0	12 - 22
0	101, 110, 111	1.0 ≤	12 - 22

Note: 1. This option should not be used with crystals, only with ceramic resonators.

Zusätzlich soll das **JTAGEN**-Bit gelöscht werden, indem das entsprechende Häkchen entfernt wird.

Ein Klick auf die Schaltfläche programmiert den Controller.

Der neue Wert für die Fuse-Bytes nach dem Umprogrammieren ist **0xC9FF**.



Lock-Bits des ATMega32A

Die Lock-Bits dienen dazu den gesamten AVR gegen Auslesen oder Veränderung zu sperren. Dies tun sie zuverlässig, da noch kein Fall bekannt wurde, wo der Schutz geknackt wurde.

Das Löschen der Lock-Bits kann nur durch ein Löschen des gesamten Bausteins (*chip erease*) erfolgen !

Es existieren 6 Lock-Bits. Zwei davon (**LB1** und **LB2**) werden für normale externe Operationen benötigt. Die anderen vier Boot-Lock-Bit betreffen den Bootloader-Bereich.

Table 26-1. Lock Bit Byte[1]

Lock Bit Byte	Bit No.	Description	Default Value
	7	–	1 (unprogrammed)
	6	–	1 (unprogrammed)
BLB12	5	Boot Lock bit	1 (unprogrammed)
BLB11	4	Boot Lock bit	1 (unprogrammed)
BLB02	3	Boot Lock bit	1 (unprogrammed)
BLB01	2	Boot Lock bit	1 (unprogrammed)
LB2	1	Lock bit	1 (unprogrammed)
LB1	0	Lock bit	1 (unprogrammed)

Note: 1. "1" means unprogrammed, "0" means programmed

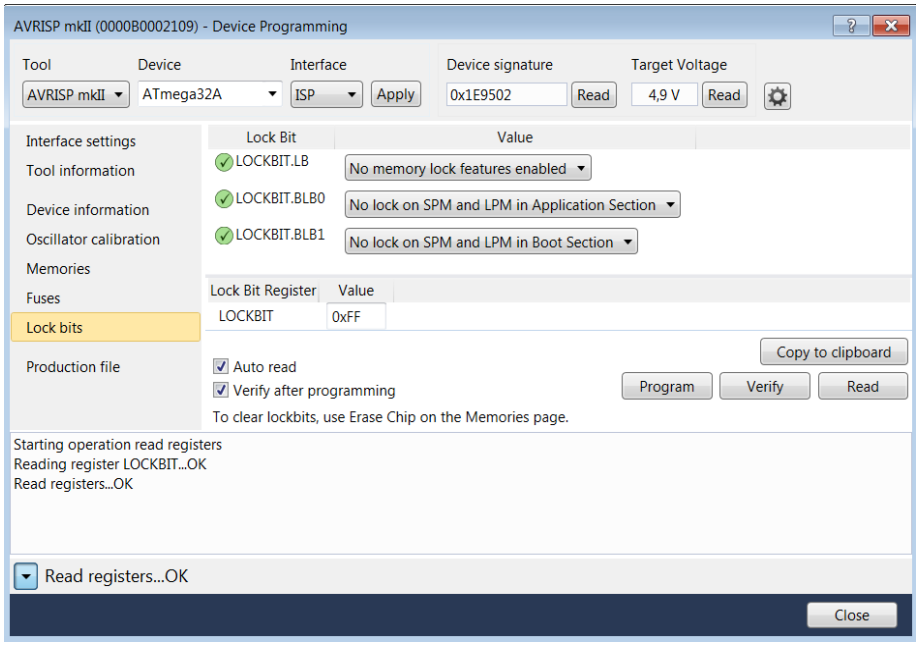
Mit den zwei LB-Bits lassen sich drei unterschiedliche Modi einstellen:

Modus 1 (LB = 0b11): Der AVR ist ungeschützt (Standard-Wert, *default*).

Modus 2 (LB = 0b10): Der Controller (Flash und EEPROM) kann nicht mehr programmiert werden (auch nicht parallel!). Dies gilt auch für die Fuse-Bit!

Modus 2 (LB = 0b00): Der Controller (Flash und EEPROM) kann nicht mehr programmiert werden (auch nicht parallel!) und nicht mehr ausgelesen (überprüft) werden. Dies gilt auch für die Fuse-Bit!

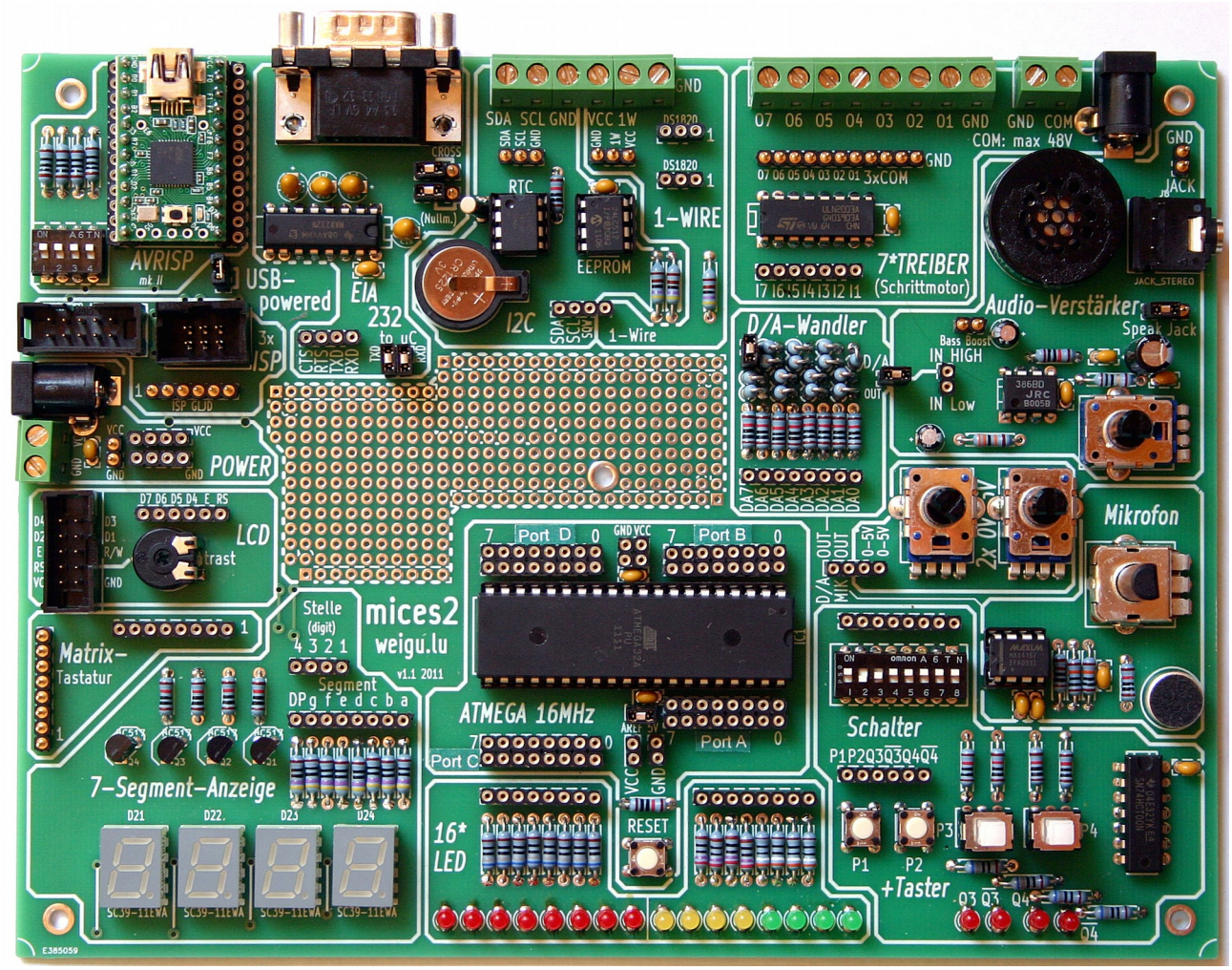
Genau wie die Fuse-Bits lassen sich die Lock-Bits im entsprechenden Reiter programmieren:

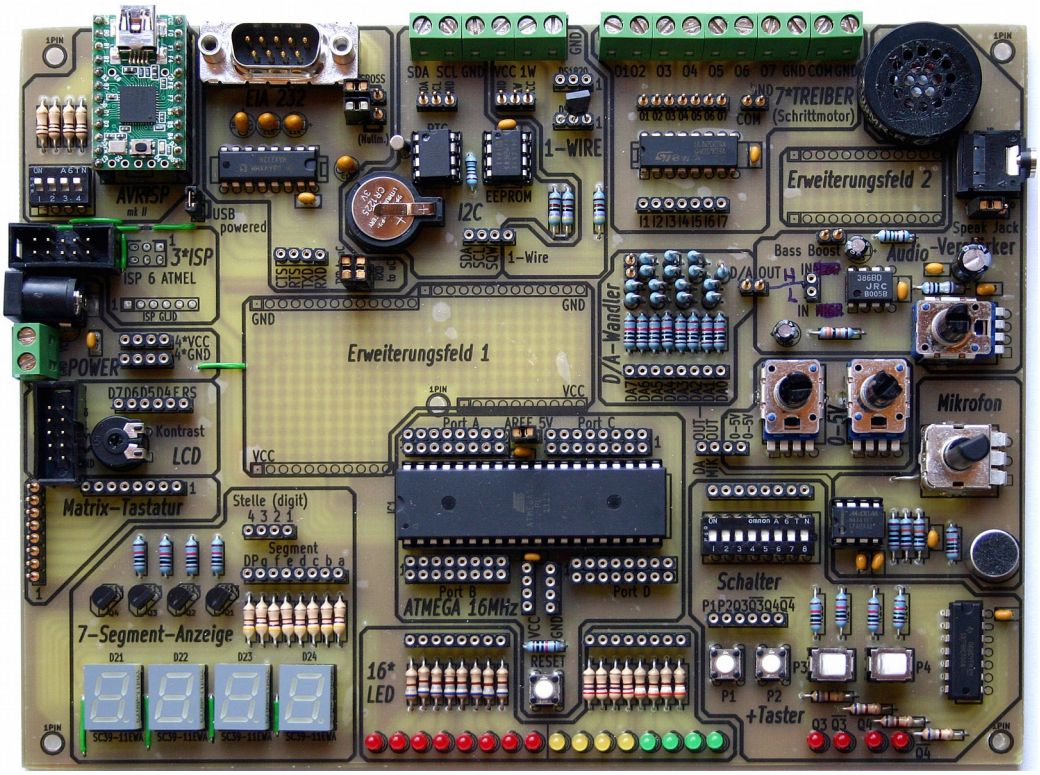


F6 MICES2-Board

Das die Herstellung des MICES-Board (siehe F7) teuer ist, wurde ein Low-Cost-Board entwickelt. Ziel war es eine einseitige leicht bestückbare Platine zu erhalten, mit der alle Aufgaben des Mikrocontroller-Kurses durchgeführt werden können. Die Platine wurde mit der Open-Source Software Kicad (kicad-pcb.org) entwickelt. Alle Dateien sind auf der Homepage (weigu.lu/microcontroller/avr_eval_board) verfügbar. Mit den Gerber-Dateien kann man die Platine professionell herstellen lassen.

In den beiden Bildern sieht man den voll bestückten Prototypen der professionell hergestellten Version und den ersten Prototypen der selbst geätzten Version.





Eigenschaften des MICES2-Board:

- Einseitige Platine (200 mm * 150 mm, 11 Drahtbrücken), gut lötbar, Dateien frei verfügbar
- Versorgung wahlweise über USB oder über externes Netzteil
- Verbindung der Baugruppen mit Drahtbrücken (oder flexiblen Steckbrücken)
- Integriertes Programmiergerät (USB, Kompatibel zum AVRISP mk2)
- 16 LEDs, 4 Taster (davon 2 entprellt), 8 Schalter, 4-stelliges Siebensegmentdisplay
- Serielle Schnittstelle, I2C-Schnittstelle, 1-Wire-Schnittstelle
- D/A-Wandler, Mikrofonschaltung, Audio-Verstärker, 2 Potentiometer 0 V - 5 V
- externes EEPROM, externe Echtzeituhr (RTC)
- Anschlüsse für Matrixtastatur, Schrittmotor (Treiber), LCD
- Feld mit Lötaugen um die Platine zu erweitern
- Schrittweiser Ausbau der Platine (beliebige Bestückung)
- Neben dem ATmega32A kann zusätzlich der für das Programmiergerät verwendete ATmega32U4 in der Schaltung verwendet werden (2 Controller!), zum Beispiel um Daten über USB an den PC zu senden (weigu.lu/b/usb/teensy2/vendor).

Herstellung der Platine

Am billigsten ist das Ätzen einer einseitig fotobeschichteten Platine. Auf die Bestückungsseite wird eine lasergedruckte Folie geklebt. Die gesamte Platine wird mit 0,8 mm gebohrt. Einige Löcher müssen dann noch mit 1 mm, 1,2 mm und 3 mm aufgebohrt werden.

Die Vorlagen zum Ätzen und für die Folie findet man auf weigu.lu/mices2.

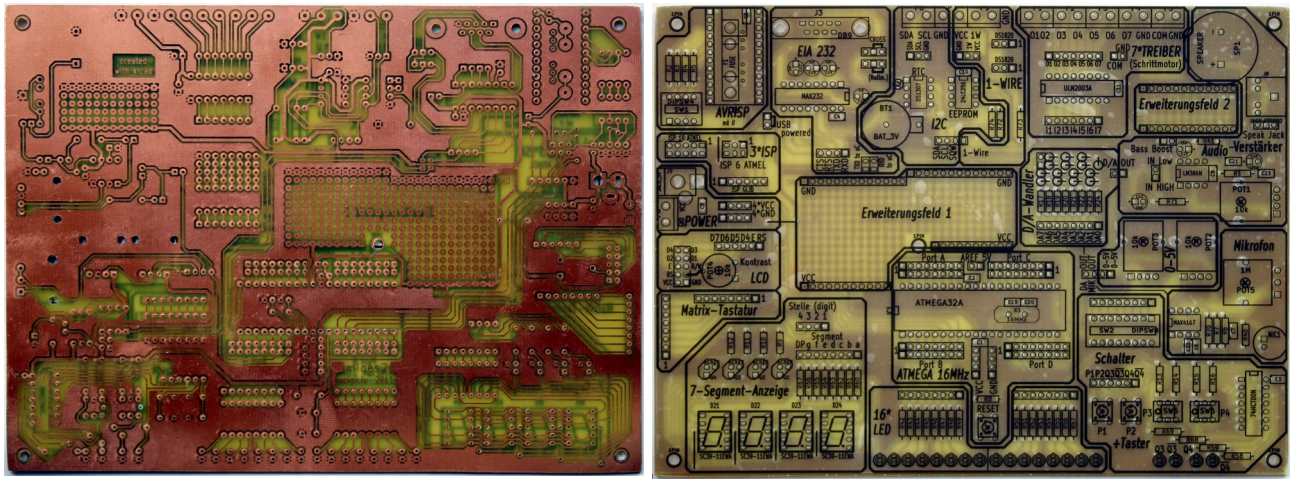
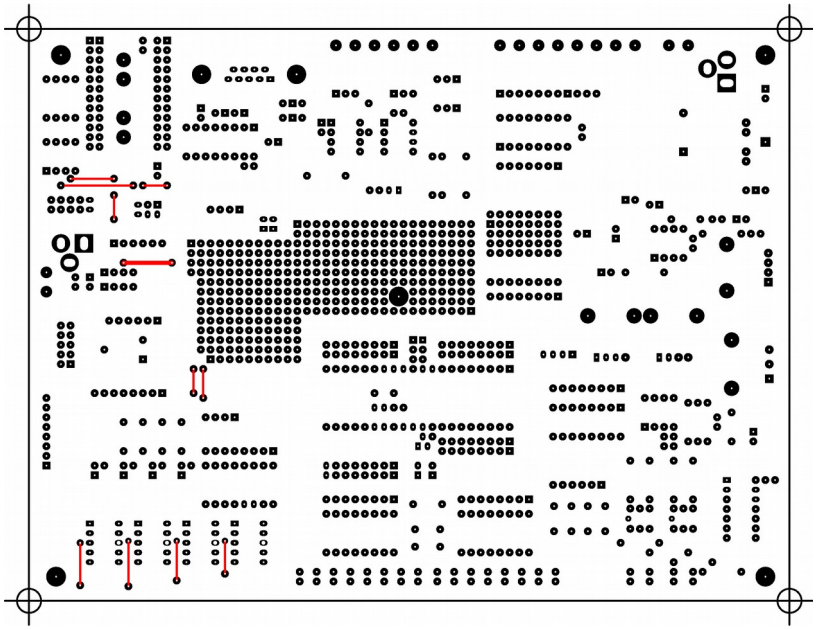


Bild: Platine des Prototypen

Bei der professionellen Version der Platine (beidseitig) müssen keine Drahtbrücken bestückt werden! Bei der einseitigen Platine werden die 11 Drahtbrücken (rot) nach der folgenden Graphik bestückt:



Danach werden zuerst die niedrigen Bauteile (Widerstände, Keramikkondensatoren, Sockel …), eingebaut, und dann erst die höheren Bauteile. Ein Platinenhalter ist sinnvoll und hilft beim Löten.

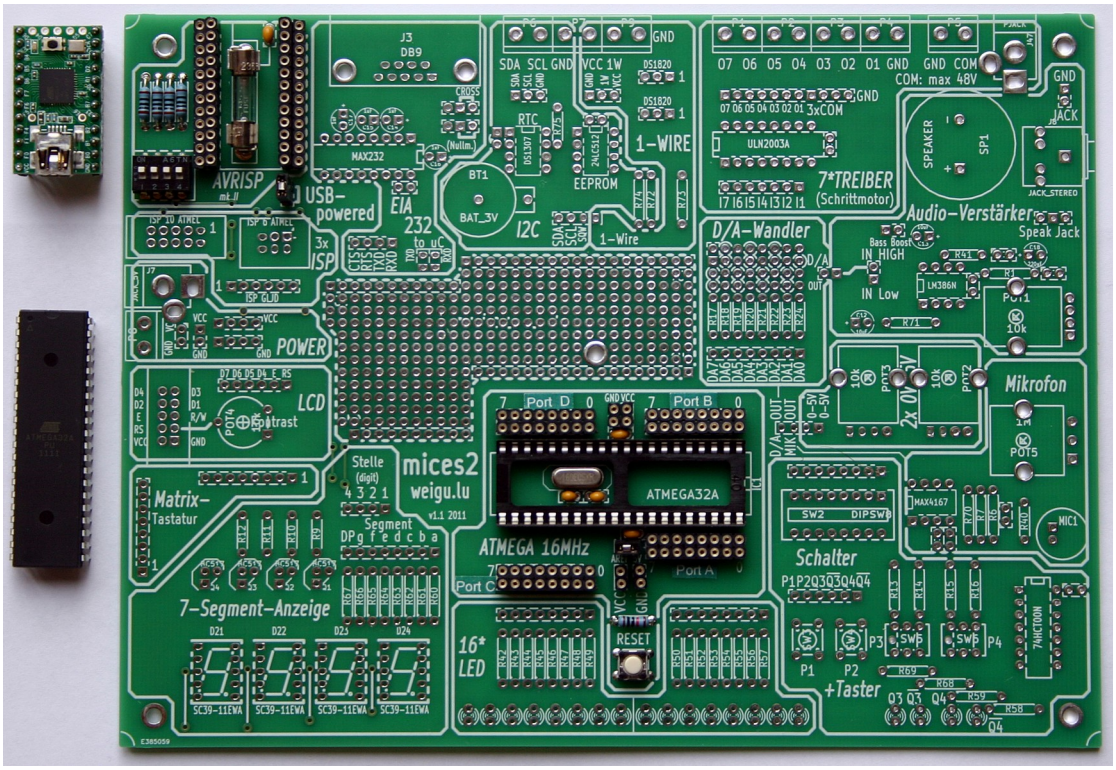
Bei den Tastern hilft es die Kontakte vor dem Einbau gerade zu biegen. Der Sicherungshalter muss mit einer Schlüsselfeile abgefeilt werden, damit er unter die Teensy-Platine passt. Die kleinen Plastikstifte bei der Stereo-Buchse eventuell abzuknipsen, damit die Buchse tiefer liegt.

Minimalbestückung

Eine Minimalbestückung des Boards kann folgendermaßen aussehen.

Es soll nur der Prozessor und das Programmiergerät bestückt werden. Die Spannungsversorgung erfolgt über USB (Jumper gesteckt). Die 4 Dip-Switches stehen auf „ON“.

Das Programmiergerät



Ein Teensy 2.0-Board von pjrc (pjrc.com/teensy) ist mit einem USB-fähigen ATmega32u4 bestückt und wird so programmiert, dass es das Programmiergerät AVRISP mk2 von ATMEL nachbildet[8] und so einwandfrei mit der Software "Studio 4" zusammenarbeitet..

Das Teensy-Board besitzt einen Bootloader und mit der Teensy-Loader Software (pjrc.com/teensy/loader.html) kann die Hex-Datei (mices2_teensy2_mk2.hex) einfach aufgespielt werden. Die Hex-Datei findet man auf „weigu.lu“. Die Software stammt von Dean Camera (fourwalledcubicle.com/LUFA.php).

Natürlich kann auch ein externes Programmiergerät verwendet werden. Dazu muss nur der entsprechende Programmierstecker aufgelötet werden (ATMEL 6-polig, ATMEL 10-polig oder proprietär (GLJD 6-polig)). Die 4 DIP-Switches stehen auf „OFF“.

8 Über ADC0 ermittelt das Teensy-Board den Wert der Systemspannung.

Spannungsversorgung

Je nach USB-Buchse am PC kann dieser 100 mA oder 500 mA liefern. Für die meisten Anwendungen reicht dieser Strom. Ist der Jumper „USB-powered“ gesteckt, so wird die gesamte Schaltung über die USB-Buchse mit Strom versorgt. Eine flinke Sicherung (500 mA) schützt den PC, auch wenn USB den Spezifikationen nach kurzschlussfest ist. Der Jumper ist außerdem sehr praktisch um bei USB-Versorgung mit einem Amperemeter den Stromverbrauch der Schaltung zu ermitteln.

Über eine Buchse, eine Schraubklemme oder eine Stiftleiste kann die Schaltung aber auch über ein externes Netzteil mit stabilisierten 5 V gefüttert werden. Der Jumper ist dann zu entfernen.

Der Mikrocontroller

Im Kurs wird ein ATmega32A verwendet. Natürlich können auch andere pinkompatible Controller (ATmega16, ATmega644p, …) eingesetzt werden. Es wird ein Quarz mit 16 MHz verwendet. JTAG soll ausgeschaltet sein, CKDIV muss beim ATmega32A aktiviert werden. Der neue Wert für die Fuse-Bytes nach dem Umprogrammieren ist dann **`0xC9FF`** (siehe Kapitel „F4 Fuse- und Lock-Bits“).

Über die vier 2x8 Sockelleisten sind die Pins des Mikrocontrollers erreichbar. Drahtbrücken werden einfach eingesteckt. Jedes Pin ist doppelt abgreifbar. In der Mitte steht noch 4 x VCC und 4 x GND zur Verfügung.

Mit dem oberen Jumper kann bei Bedarf der Eingang **AREF** des internen A/D-Wandlers mit 5 V verbunden werden. Es kann natürlich auch eine externe Referenz angelegt werden.

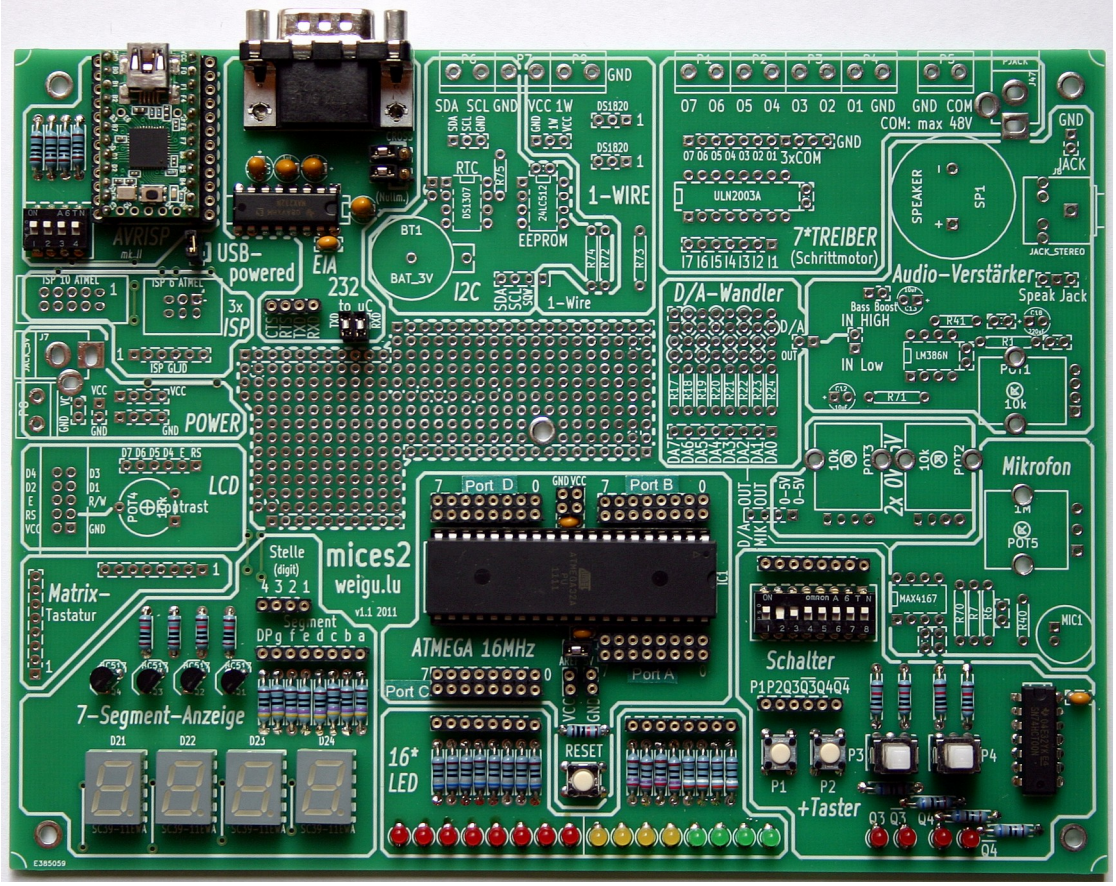
Bauteilliste:

Neben der Platine und den unten angeführten Bauteilen wird ein Teensy 2.0-Board benötigt. Dies kann bei PJRC (www.pjrc.com/teensy) bestellt werden. Möchte man die Bauteile bei der Firma „mouser.com“ bestellen, so kann dort die Bauteilliste leicht importiert werden (2 erste Kolonnen).

Bestellnr „mouser.com“	Anz.	Referenz	Wert	Beschreibung
80-C315C104Z5U5CA	3	C1-C3	100nF	Keramik-Kondensatoren (MLCC)
80-C315C220K2G	2	C19-C20	22pF	Keramik-Kondensatoren (MLCC)
576-0235.500HXP	1	F1	0,5A	Sicherung 5mmx20mm
556-ATMEGA32A-PU	1	IC1	ATMEGA32A	Mikrocontroller
575-31043164	2	J27-J30,...	64 pol.	SIP-Sockel 3A
575-010064	1	JP1,JP9	64 pol.	DIP-Headerleiste
660-MF1/4DC1500F	4	R2-R5	150	Metallschichtwiderstände 0,25W
660-MF1/4DC1002F	1	R8	10k	Metallschichtwiderstände 0,25W
653-A6TN-4104	1	SW1	4*1 pol.	4xDIP-Schalter
653-B3F-1000	1	SW7	1 pol.	Taster 1pol.
520-CSM1600-20-X	1	X1	16MHz	Quarz
576-04450001H	2	(F1)		Sicherungshalter
649-DILB40P223TLF	1	(IC1)	40 pol.	IC-Sockel
575-8014305010	1	Teensy	50 pol.	SIP-Sockel 4,5A
855-M7567-46	2	JP	2 pol.	Jumper

Standardbestückung

Um alle Aufgaben der ersten beiden Module lösen zu können sind neben Programmiergerät und Prozessor noch die LEDs, Taster und Schalter, die Sieben-Segmentanzeige und die serielle Schnittstelle zu bestücken.



Sieben-Segmentanzeige und LEDs

Bei der Sieben-Segmentanzeige werden die Stellen (digits) genau so wie die Segmente mit einer Eins (5 V) aktiviert. Die Leuchtstärke der Anzeige ist für Multiplexing ausgelegt. Die 16 LEDs stehen in drei Farben zur Verfügung.

Tasten und Schalter

Bei zwei der 4 Taster handelt es sich um entprellte Taster (P3 und P4). Sie wurden hardwaremäßig mittels Flip-Flops entprellt (siehe Schaltplan). Die Ausgänge stehen invertiert und nicht-invertiert zur Verfügung. 4 LEDs zeigen den jeweiligen Zustand der beiden Ausgänge. Diese LEDs dienen gleichzeitig als „Power“-LEDs da zwei davon bei Spannungsversorgung aufleuchten. Die beiden anderen Taster wie auch die Schalter sind nullaktiv (ziehen das Signal bei Betätigung gegen Masse).

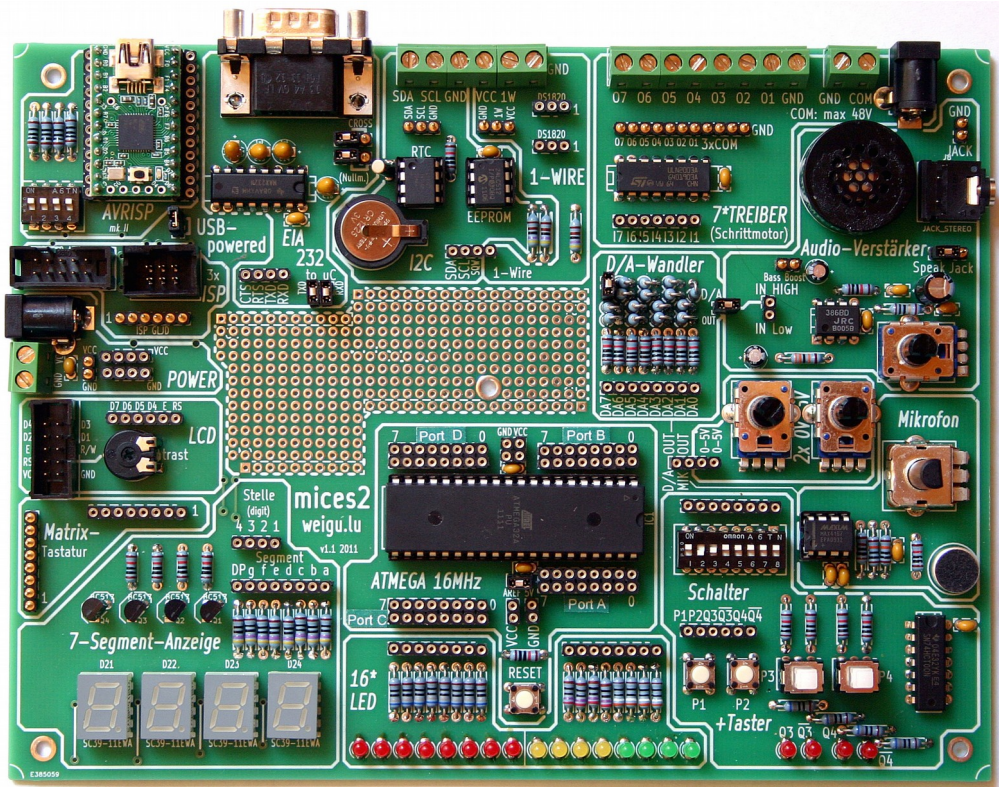
Serielle Schnittstelle

Bei der seriellen Schnittstelle können mit den beiden oberen Jumpern die Sendeleitungen **TxD** und die Empfangsleitung **RxD** gekreuzt werden (falls kein Nullmodemkabel verwendet wird!). Die beiden unteren Jumper verbinden **TxD** und **RxD** falls erwünscht mit den Pins **PD0** und **PD1** des Controllers. Sie müssen somit nicht über die Sockelleiste verbunden werden. Die beiden Pins des Controllers dürfen dann nicht mehr anderweitig angeschlossen werden. Die zwei Steuerleitungen **RTS** und **CTS** sind an der Sockelleiste abgreifbar.

Bauteilliste:

Bestellnr „mouser.com“	Anz.	Referenz	Wert	Beschreibung
80-C315C104Z5U5CA	2	C4-C5	100nF	Keramik-Kondensatoren (MLCC)
80-T350A105M035AT-TR	4	C14-C17	1uF	Tantalkondensatoren
78-TLHR4605	12	D1-D12	LED rot	LED rot 10mA 3mm 60deg
78-TLHG4605	4	D13-D16	LED grün	LED grün 10mA 3mm 60deg
78-TLHY4600	4	D17-D20	LED gelb	LED gelb 10mA 3mm 60deg
604-SC39-11EWA	4	D21-D24	SC39-11EWA	Siebensegmentanzeige CC
517-D2510-6002-AR	1	J2	2x5 pol.	Wannenstecker ISP 2x5
660-MF1/4DC1002F	8	R9-R16	10k	Metallschichtwiderstände 0,25W
660-MF1/4DC6800F	12	R42-R49,R58, ...	680 Ohm	Metallschichtwiderstände 0,25W
660-MF1/4DC2700F	4	R50-R53	270 Ohm	Metallschichtwiderstände 0,25W
660-MF1/4DC3900F	4	R54-R57	390 Ohm	Metallschichtwiderstände 0,25W
660-MF1/4DC4700F	8	R60-R67	470 Ohm	Metallschichtwiderstände 0,25W
512-BC517	4	Q1-Q4	BC517	Darlingtontransistor
653-A6TN-8104	1	SW2	8*1 pol.	8xDIP-Schalter
653-B3F-1000	2	SW3,SW4	1 pol.	Taster 1pol.
688-SPPH430200	2	SW5-SW6	2 pol.	Taster 2 pol.
595-SN74HCT00N	1	U2	74HCT00N	IC 4*NAND-Gatter
595-MAX232NE4	1	U5	MAX232	IC Pegelwandler (EIA232)
855-M7567-46	4	JP	2 pol.	Jumper
575-31043164	1	Jxx	64 pol.	SIP-Sockel 3A

Vollständige Bestückung



D/A-Wandler

Der 8-Bit D/A-Wandler ist mit einem R2R-Netzwerk realisiert. Am Jumper oben links kann das Signal hinter dem Netzwerk abgegriffen werden. Normalerweise ist dieser Jumper gesetzt und das Signal wird mit einem Operationsverstärker (½ MAX4167) verstärkt. Das verstärkte Signal kann an der unteren Sockelleiste abgegriffen werden (DA OUT) oder gleich mit dem Jumper oben rechts an den High-Eingang (0 V - 5 V) des Audio-Verstärkers weitergeleitet werden.

Mikrofon-Schaltung

Das Signal eines Electret-Mikrofons wird verstärkt (0 V - 5 V) und kann an der Sockelleiste abgegriffen werden (MIK OUT). Ein Potentiometer (unten rechts) ermöglicht die Veränderung des Aufnahmepegels. Zum Testen kann der Mikrofonausgang mit dem High-Eingang des Audio-Verstärkers verbunden werden (Achtung, ev. Rückkopplung).

2 Potentiometer 0 V - 5 V

Sie dienen zum Testen des internen A/D-Wandlers.

Audio-Verstärker

Der Audio-Verstärker besitzt 2 Eingänge. Für den D/A-Wandler und die Mikrofonschaltung ist der High-Eingang zu wählen (0 V - 5 V). Bei kleinen Spannungen der Low-Eingang. Das Ausgangssignal kann am Lautsprecher (8 Ω)oder an der 3,5 mm Stereo-Buchse (3,5 mm), bzw. der 2 poligen Stiftleiste, zum Anschluss eines externen Lautsprechers ausgegeben werden. Die Lautstärke lässt sich mit dem Potentiometer oben rechts einstellen. Mit dem Jumper „Bass Boost“ können tiefe Töne angehoben werden.

7xTreiber

Mit dem Treiberbaustein ULN2003 lässt sich sofort ein kleiner Schrittmotor ansteuern (max. 0,5 A, siehe Kapitel Schrittmotor). Natürlich können damit auch Relais oder andere Geräte die mehr Strom oder eine höhere Spannung benötigen ansteuern. Die externe Spannung für die Treiberstufen (max. 48 V) kann über Schraubklemmen, eine Stiftleiste oder eine DC-Spannungs-versorgungs-steckverbindung zugeführt werden. Die Geräte selbst können über Schraubklemmen oder die Stiftleiste angeschlossen werden.

1-Wire-Schnittstelle

Die 1- Wire-Schnittstelle besteht aus einem Pull-Up-Widerstand und zwei 3 polige SIP-Sockel für Bauteile. Zum Testen (siehe Testprogramme) wird ein Temperatursensor DS1820 (DS18S20) oder DS18B20 benötigt. Die Schnittstelle ist über Schraubklemmen oder über eine 3 polige Stiftleiste (VCC, Datenleitung und GND) erweiterbar.

I^2C-Schnittstelle

Eine batteriegepufferte Echtzeituhr und ein EEPROM mit 512 KiByte können mittels I^2C angesprochen werden. Die Echtzeituhr liefert auch bei Bedarf ein quarzgenaues Rechtecksignal von zum Beispiel 1 Hz am Ausgang SQW. Dies kann zum Beispiel genutzt werden, um Sekundengenaue Interrupts auszulösen. Die Echtzeituhr läuft nur, wenn eine Batterie (CR1225) vorhanden ist, oder der Pluspol der Batterie mit Masse verbunden ist (Stück Metall in der Batteriehalterung).

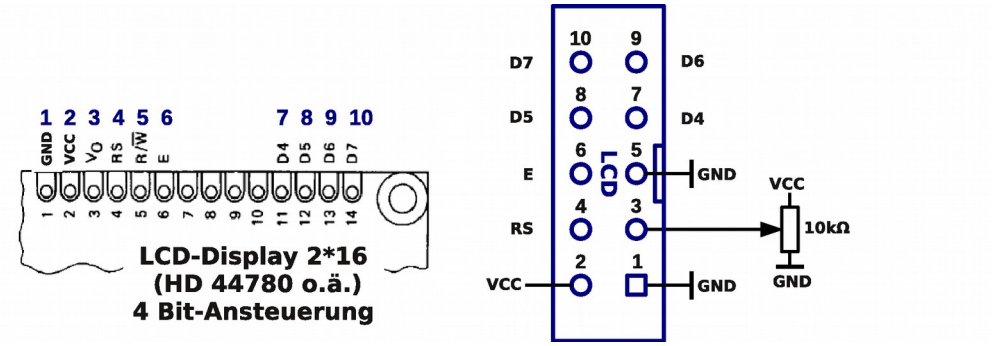
Die Schnittstelle ist über Schraubklemmen oder über eine 3 polige Stiftleiste (SDA, SCL und GND) erweiterbar.

Externe Spannungsversorgung

Soll das Board extern mit 5 V (VCC) versorgt werden, so kann dies über Schraubklemmen, eine Stiftleiste oder eine DC-Spannungsversorgungssteckverbindung erfolgen. Wird die Stiftleiste nicht verwendet, so ist sie nützlich um zum Beispiel die Masse eines Oszilloskops anzuschließen. Des weiteren stehen zwei 4 polige Sockelleisten mit 5 V und Masse zur Verfügung.

LCD-Display

Ein Standard-LCD-Diplay (z.B 2x16 Zeichen, HD44780 kompatibel) kann über den 10 poligen Wannenstecker angeschlossen werden. Es ist darauf zu achten , dass an Pin 1 Masse liegt![9] Mit dem Trimmer wird der Kontrast des Displays eingestellt.



Tastaturanschluss

Der 8 polige Anschluss kann zum Beispiel für eine 3x4 Matrixtastatur genutzt werden. Über die Stiftleiste lassen sich natürlich auch beliebige andere Geräte mit dem Controller verbinden.

Erweiterungsfeld

In der ersten Reihe des Feldes sind die Lötaugen mit VCC verbunden, in der untersten Reihe mit GND. Auf dem Erweiterungsfeld lassen sich so beliebige zusätzliche Schaltungen realisieren.

Bauteilliste:

Für die drei teuren ICs (RTC, EEPROM und MAX4167) werden Sockel verwendet.

9 Es existieren Displays bei denen Pin 1 (GND) und 2 (VCC) vertauscht sind! An Pin 3 liegt der Ausgang des Trimmers zum Anpassen des Kontrasts (VO). RS an Pin4, RW

Bestellnr „mouser.com“	Anz.	Referenz	Wert	Beschreibung
534-500	1	BT1	BAT	Lithium-Batteriehalter
80-C315C104Z5U5CA	6	C6-C11	100nF	Keramik-Kondensatoren (MLCC)
140-SS100M1A0405P	2	C12-C13	10uF	Elektrolytkondensatoren
667-EEU-FR1A221	1	C18	220uF	Elektrolytkondensator
80-C320C333K5R5CA	1	C21	33nF	Keramik-Kondensatoren (MLCC)
80-C315C390K2G	1	C22	39pF	Keramik-Kondensatoren (MLCC)
80-C320C473K5R5CA	1	C23	47nF	Keramik-Kondensatoren (MLCC)
571-5-102619-1	1	J1	AVRISP	Wannenstecker ISP 2x3
517-D2510-6002-AR	1	J9	2x5 pol.	Wannenstecker LCD
649-D09P13A4GV00LF	1	J3	9 pol.	DB9 Eia232 männl. 9 pol
575-31043164	1	Jxx	64 pol.	SIP-Sockel 3A
806-KLDHCX-0202-A	2	J7+J47	2 mm	DC-Spannungsversorgungssteckverb.
502-35RAPC4BHN2	1	J8	3,5 mm	Stereo-Steckverbinder Jacket
665AOM6545PR	1	MIC1	Electret	Elektret-Mikrofon
571-2828362	9	P1-P9	2 pol.	Schraubklemme
688-rk11k1130040	3	POT1-POT3	10k	Potentiometer
858-38PKABR10KLF30	1	POT4	10k	Trimmer
652-PDB12-M4251105BF	1	POT5	1M	Potentiometer
660-MF1/4DC10R0F	1	R1	10 Ohm	Metallschichtwiderstände 0,25W
660-MF1/4DC1003F	2	R6-R7	100k	Metallschichtwiderstände 0,25W
660-MF1/4DC1002F	26	R8-R41,R75	10k	Metallschichtwiderstände 0,25W
660-MF1/4DC1001F	1	R70	1k	Metallschichtwiderstände 0,25W
660-MF1/4DC3303F	1	R71	330k	Metallschichtwiderstände 0,25W
660-MF1/4DC4701F	3	R72-R74	4.7k	Metallschichtwiderstände 0,25W
665-AST-02308MR-R	1	SP1	8 Ohm	Laursprecher
579-24LC512-I/P	1	U1	24LC512	IC EEPROM
700-DS1307	1	U3	DS1307	IC Echtzeituhr RTC
513-NJM386BD	1	U4	LM386N	IC Verstärker
700-MAX4167EPA	1	U6	MAX4167	IC Operationsverstärker RTR
700-DS18B20+	1	IC2	DS18B20	Temperatursensor
511-ULN2003A	1	U7	ULN2003A	Treiber IC (0,5A)
520-TFC3X8-X	1	X2	32.768kHz	Uhrenquarz
649-DILB8P223TLF	3		8 pol.	IC-Sockel 8 pol.
855-M7567-46	4	JP	2 pol.	Jumper

Zusätzlich wird eine Batterie CR1225 benötigt. Flexible Steckbrücken (10 cm) können selbst hergestellt werden (starren Draht an flexiblen Draht löten und mit Schrumpfschlauch isolieren) oder zum Beispiel bei conrad.de bestellt werden.

Testen der Platine

Testen der Standardbestückung

Das Programm testet die Schalter, Taster, die serielle Schnittstelle und das Sieben-Segment-Display.

Mit Hilfe des Programmiergerätes ist das Programm „mices2_test1.hex“ (weigu.lu/mices2) in den Flash des ATmega32A zu brennen.

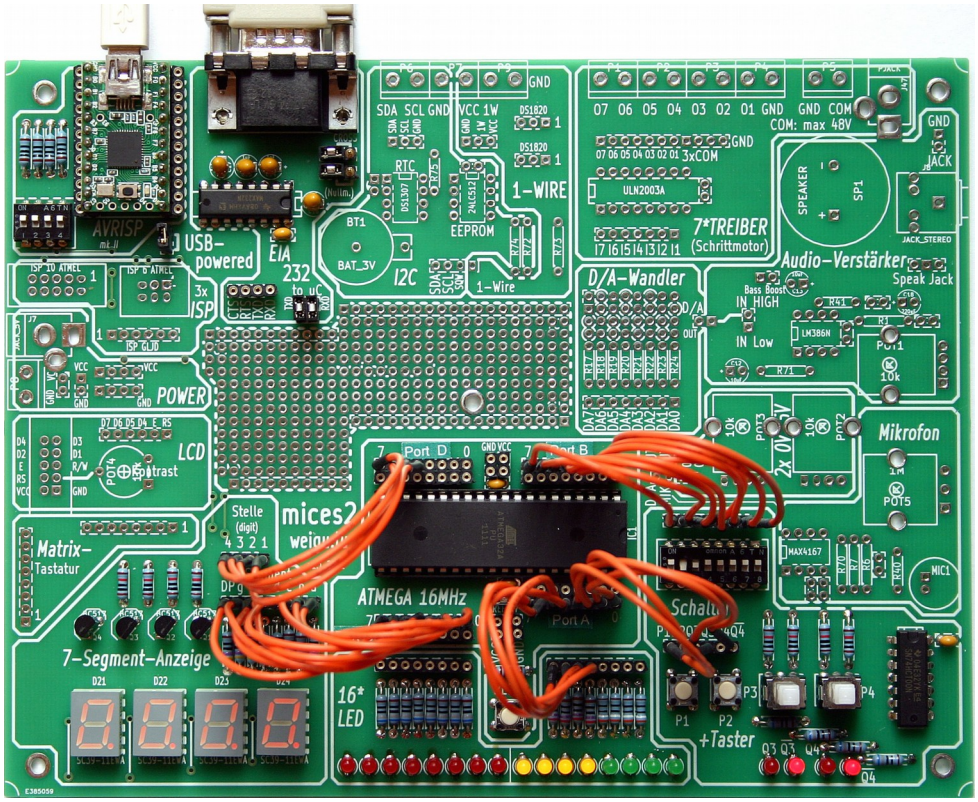
Erster Test

Die Software erlaubt es mit Hilfe der 8 Schalter die 7 Segmente des Displays sowie den Dezimalpunkt des Displays ein- bzw. auszuschalten. Die Zustände der 4 Taster (interne Pull-Ups sind eingeschaltet) können an den vier gelben LEDs abgelesen werden. Die restlichen LEDs werden zum Testen einfach nacheinander mit einer Drahtbrücke mit VCC verbunden. Die serielle Schnittstelle sendet alle empfangenen Zeichen wieder zurück (Echo-Mode). TxD und RxD werden über 2 Jumper mit PortD0 und PortD1 verbunden (to uC). Die serielle Schnittstelle wird mit Hilfe eines Nullmodemkabel mit einem PC verbunden auf dem eine Modemsoftware läuft. Das einzustellende Datenformat ist 8N1 mit 38400 Bit/s. Wird kein Nullmodemkabel, sondern eine direkte Verbindung genutzt, so können die Signale mit den oberen Wechseljumpern auf der Platine gekreuzt werden (Cross)[10].

Folgende Verbindungen müssen gesteckt werden:

- 8 Segmente (Display) ↔ Port C
- 4 Stellen (Display) ↔ Port D (4-7)
- 8 Schalter ↔ Port B
- 4 Taster (P1, P2, $\overline{Q3}$, $\overline{Q4}$) ↔ Port A (0-3)
- 4 gelbe LEDs ↔ Port A (4-7)
- 4 Jumper der seriellen Schnittstelle

10 Ein Fehler im Momentanen Layou verhindert dies. Die Steckbrücken müssen sich links befinden!!



Testen der vollständig bestückten Platine

Bei der vollständig bestückten Platine werden zwei Tests durchgeführt.

Das erste Programm (mices2_test2.hex) testet den D/A Wandler, die Mikrofonschaltung, den Audio-Verstärker, die variable Spannungsquelle und den I²C-Bus (EEPROM).

Das zweite Programm dient zum Testen der I²C-Schnittstelle (RTC), der 1-Wire Schnittstelle (DS1820), einer externen 3x4 Matrixtastatur, eines externen LCD-Displays und eines externen Schrittmotors (Treiberbaustein).

Zweiter Test

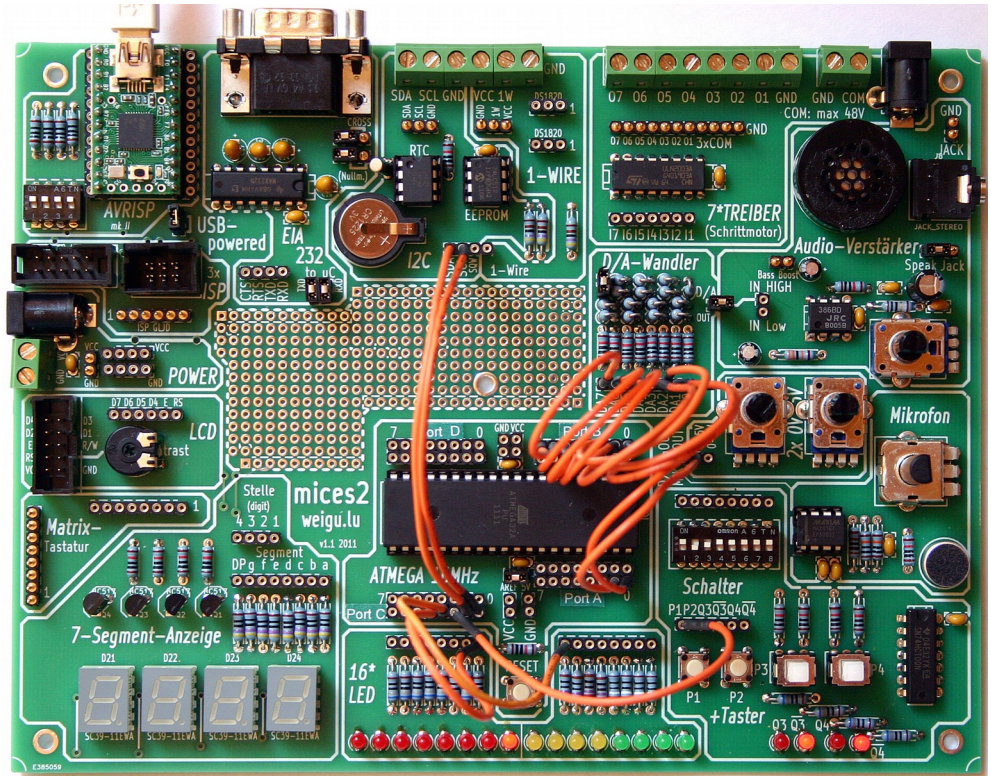
Das Programm nimmt Schallsignale (Sprache, Musik) auf und spielt sie mit variabler Geschwindigkeit wieder ab.

Wird der Taster P1 gedrückt, so leuchtet nach einer Zeit die Aufnahme-LED auf. Jetzt kann während einigen Sekunden mit dem Mikrofon aufgenommen werden. Gleich nach der Aufnahme wird das Signal über den D/A-Wandler abgespielt. Mit dem Potentiometer kann die Abspielgeschwindigkeit verstellt werden.

Das Signal der Mikrofonschaltung wird dabei mit dem internen A/D-Wandler digitalisiert und im externen EEPROM abgespeichert. Beim Abspielen übergibt der Controller das Signal an den D/A-Wandler.

Das Signal gelangt dann über einen Jumper zum High-Eingang des Audioverstärkers der das verstärkte Signal am Lautsprecher (Speaker[11]) ausgibt. Timer0 wird bei der Aufnahme verwendet, Timer2 beim Abspielen.

- D/A-Wandler (DA0-DA7) ↔ Port B
- Potentiometer (0-5V) ↔ PortA1
- Aufnahmetaster P1 ↔ PortC2
- Mikrofonschaltung Ausgang (Mic Out) ↔ PortA0
- EEPROM (I2C) SCL ↔ PortC0
- EEPROM (I2C) SDA ↔ PortC1
- LED Aufnahme (beliebige LED) an PortC3
- LED Abspielen (beliebige LED) an PortC4
- Jumper am D/A-Wandler, am Verstärker (IN High) und Auswahljumper Speaker



Dritter Test

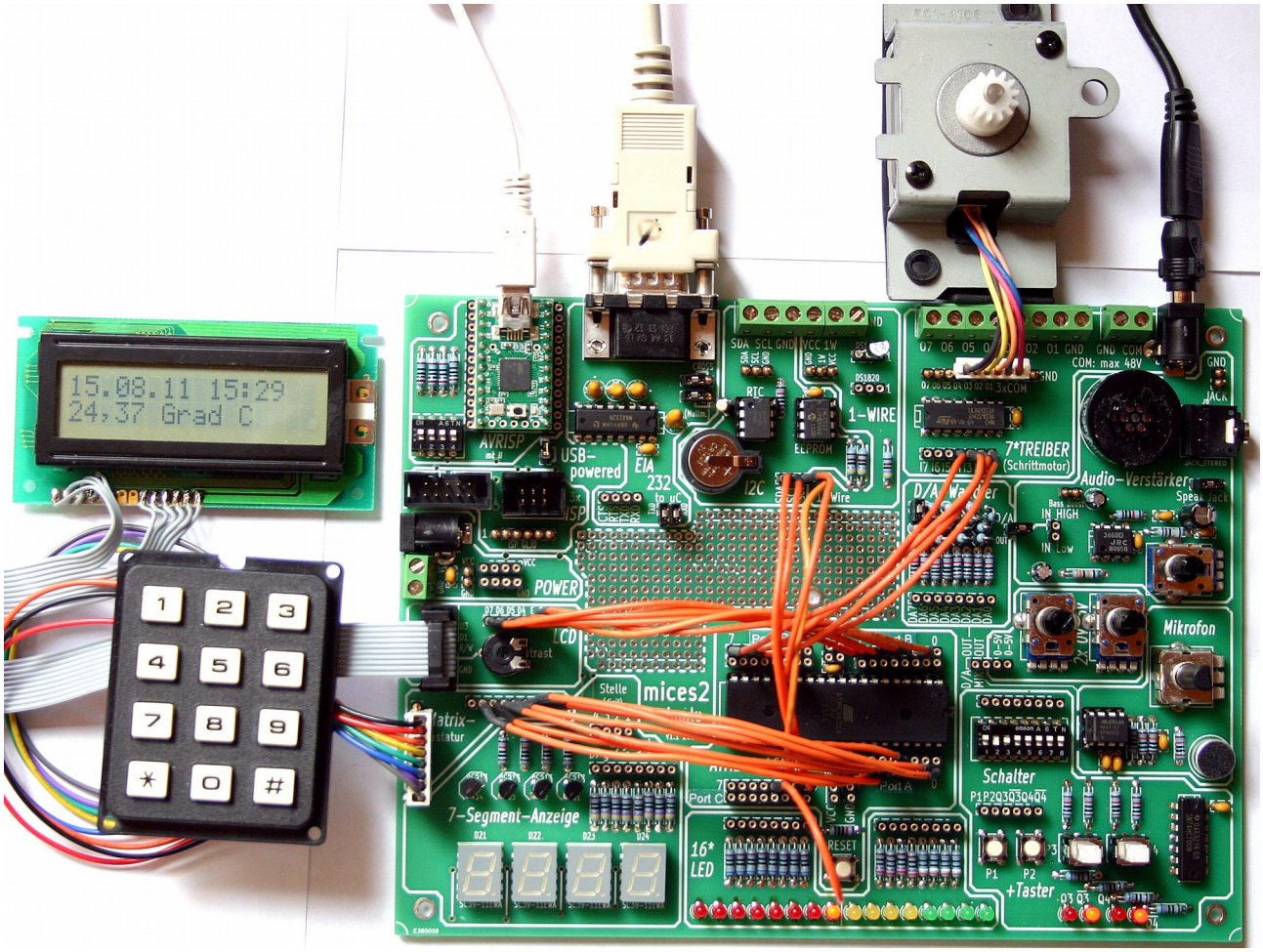
Das Programm benutzt die Echtzeituhr um eine LED im Sekundenschritt (quarzgenau) blinken zu lassen und den Schrittmotor einen Schritt pro Sekunde drehen zu lassen. Die Uhrzeit, das Datum und die Temperatur (DS18B20 über 1-Wire) werden auf dem LCD-Display und über die serielle Schnittstelle ausgegeben. Das Drücken einer Taste auf der Matrixtastatur bewirkt die Ausgabe des Tastenwertes auf dem LCD-Display und der Schnittstelle.

11 Oder auf Wunsch an einem externen Lautsprecher (Jumper umstecken)

TxD und RxD werden über 2 Jumper mit PortD0 und PortD1 verbunden (to uC). Die serielle Schnittstelle wird mit Hilfe eines Nullmodemkabel mit einem PC verbunden auf dem eine Modemsoftware läuft. Das einzustellende Datenformat ist 8N1 mit 38400 Bit/s. Wird kein Nullmodemkabel, sondern eine direkte Verbindung genutzt, so können die Signale mit den oberen Wechseljumpern auf der Platine gekreuzt werden (Cross).

Datum und Uhrzeit können über die serielle Schnittstelle gesetzt werden. Dazu muss der Großbuchstabe 'T'+Leerzeichen+Datum+Zeit (ohne Zwischenzeichen und ohne Sekunden) gesendet werden. Bsp: T 2207111012 (22 Juli 2011 10h12)

- Matrixtastatur ↔ Port A (Spalten 0-2, Zeilen 3-6)
- Schrittmotor (I1-I4) ↔ Port D (4-7)
- LCD-Display ↔ Port B (RS 2, EN 3, D4-D7 4-7)
- RTC (I2C) SCL ↔ PortC0
- RTC (I2C) SDA ↔ PortC1
- RTC (I2C) SQW ↔ PortD2 (INT0)
- Sekunden LED (beliebige LED) ↔ PortC3
- 1-Wire ↔ PortC2
- 4 Jumper der seriellen Schnittstelle
- DS18B20 muss gesteckt sein

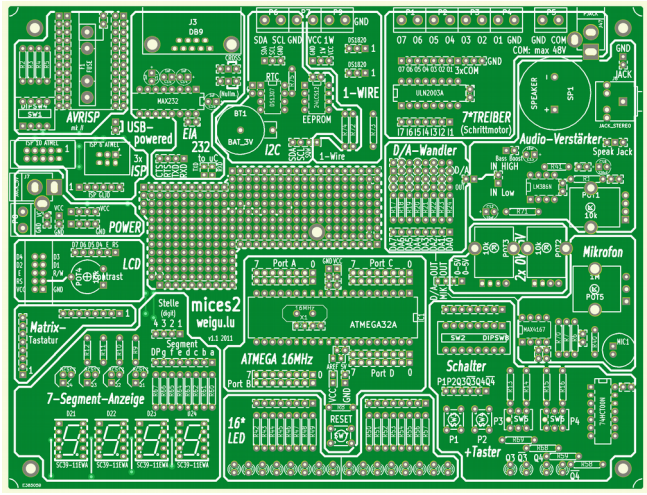


Kosten

Die Kosten für die Minimalversion betragen mindestens 55 € (Stand 2011). Sie setzen sich zusammen aus den Platinenkosten (ungeätzte Platine 7 €), den Kosten für das Teensy-Board (rund 16 € inkl. Versandkosten) und den Kosten für die Bauteile (rund 32 € ohne Versandkosten). Beträgt der Bestellwert bei „mouser.com“ über 75 € (ohne Mwst) so sind keine Versandkosten zu zahlen. Es ist also interessant Sammelbestellungen zu organisieren.

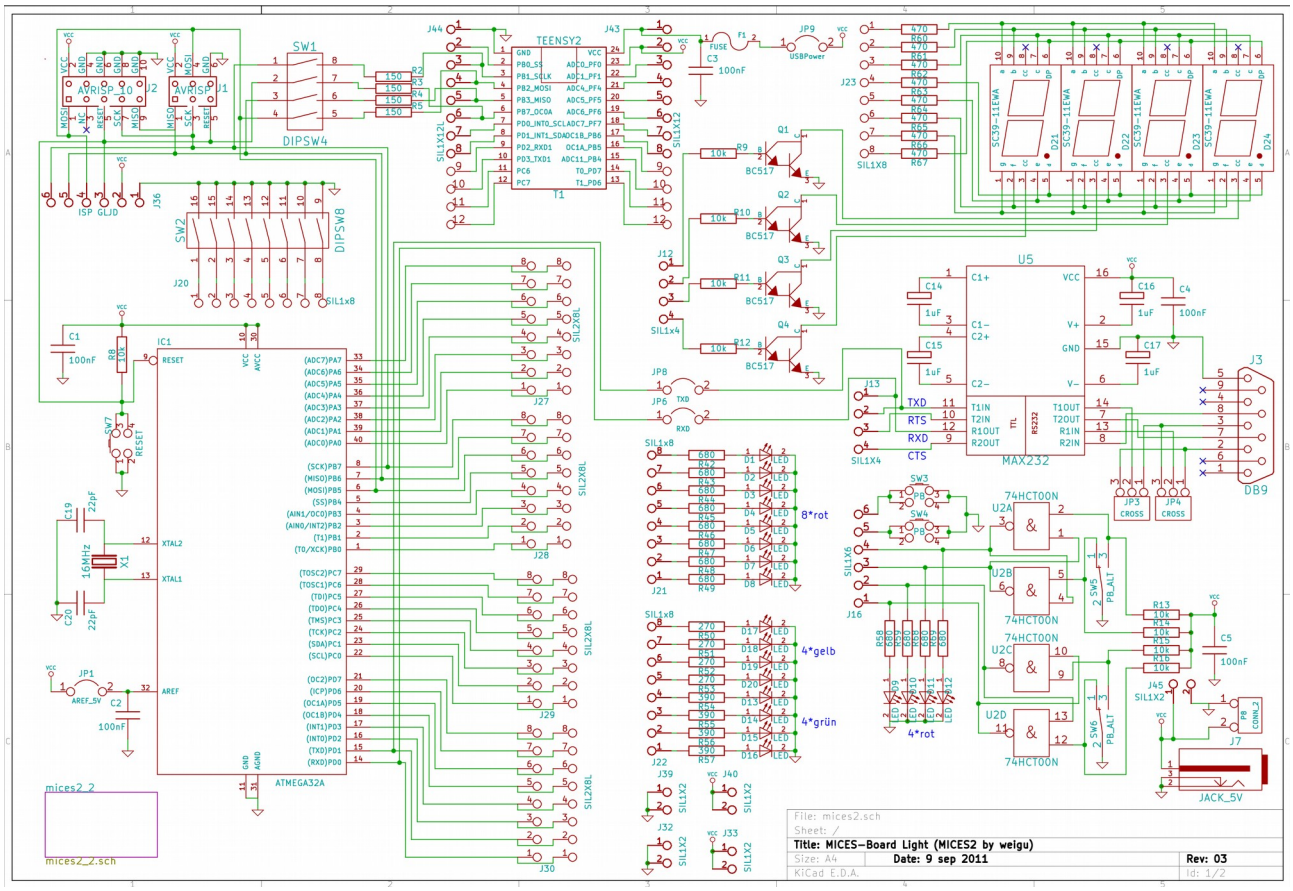
Die Kosten für die Standardversion betragen mindestens 82 €. Für die Vollversion sind mindestens 130 € zu zahlen.

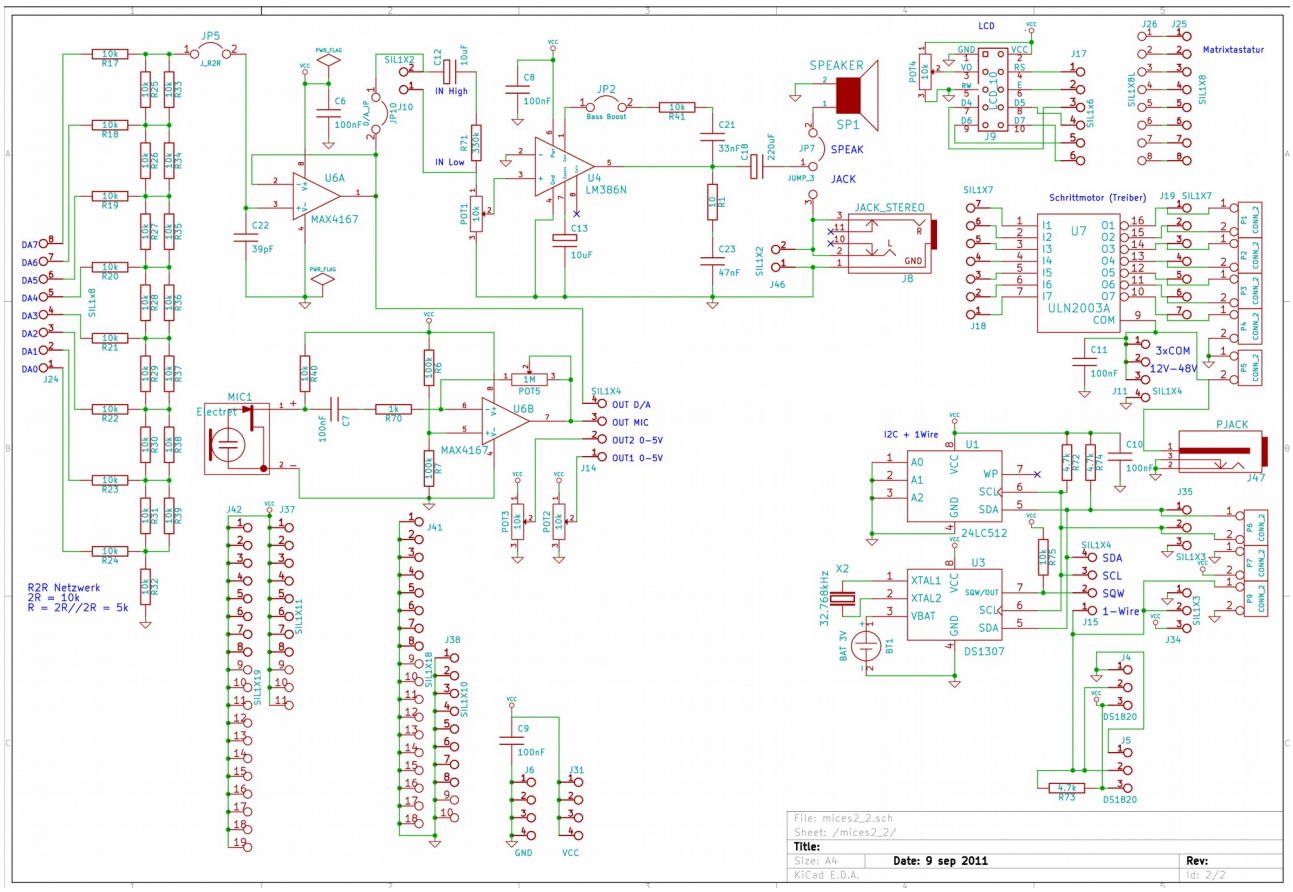
Da recht viele Löcher zu bohren sind, kann man die Platine auch professionell herstellen lassen. Bei einer Sammelbestellung kann man den Preis bei 10 Stück auf 23 € senken (z.B. eurocircuits.com, 25 Werktage). Bei höheren Stückzahlen wird die Platine noch billiger.



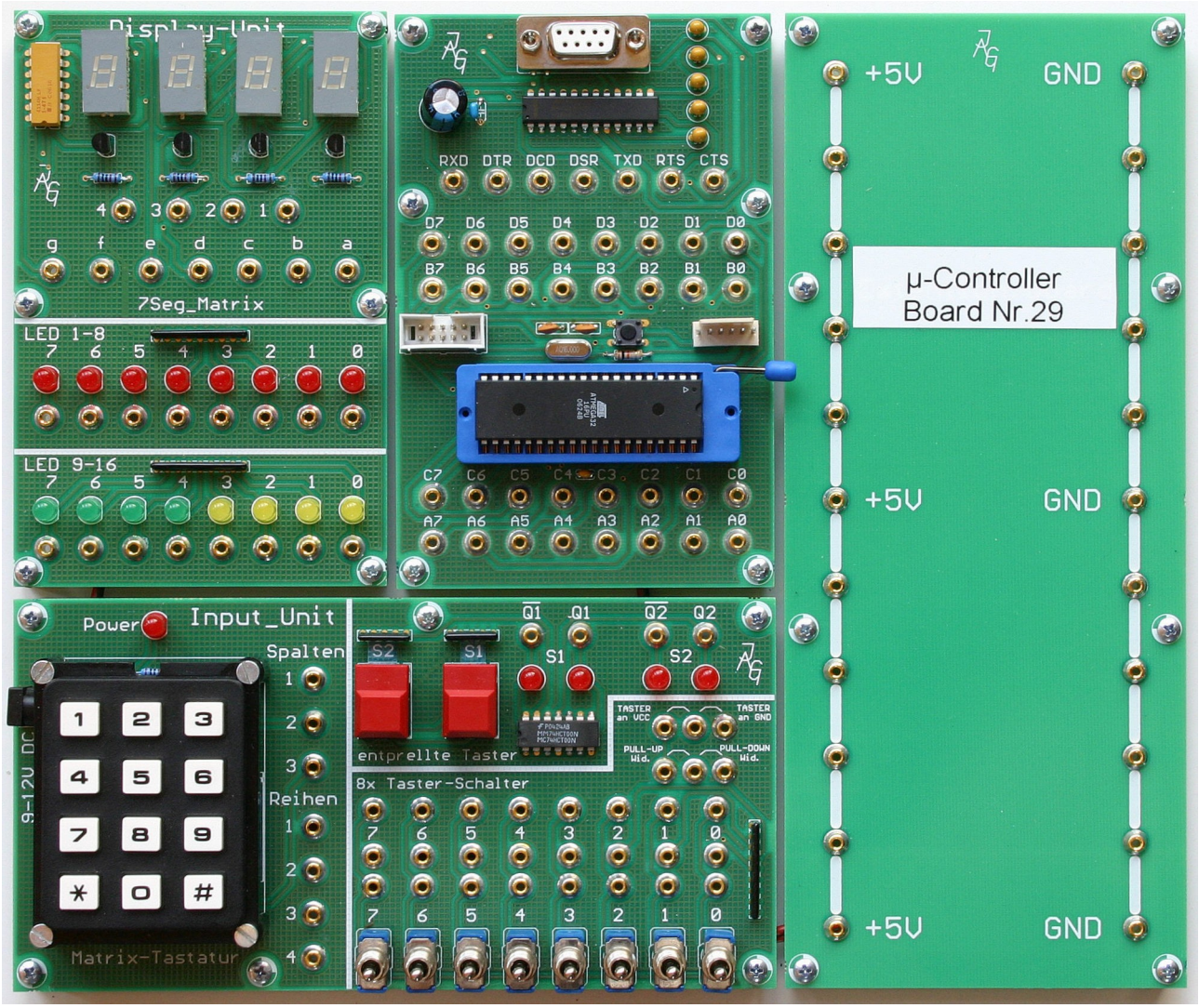
Schaltpläne

Siehe nächste Seite!


TEENSY2
ATMEGA32A
MAX232
74HCT00N
DIPSW4
DIPSW8
Title: MICES—Board Light (MICES2 by welgu)
Date: 9 sep 2011
Rev: 03


LM386N
MAX4167
ULN2003A
DS1307
24LC512
DS1820
Schrittmotor (Treiber)
Matrixtastatur
R2R Netzwerk
2R = 10k
R = 2R//2R = 5k
Title:
Date: 9 sep 2011
Rev:

F7 MICES Board



Das obige Board wurde von Georges Lauth und Jean Dauenfeld entwickelt und gebaut um den Schülern einen leichten Umgang mit dem Mikrocontroller zu ermöglichen. Es besteht aus vier Platinen. Alle relevanten Anschlüsse sind über 2 mm-Buchsen erreichbar.

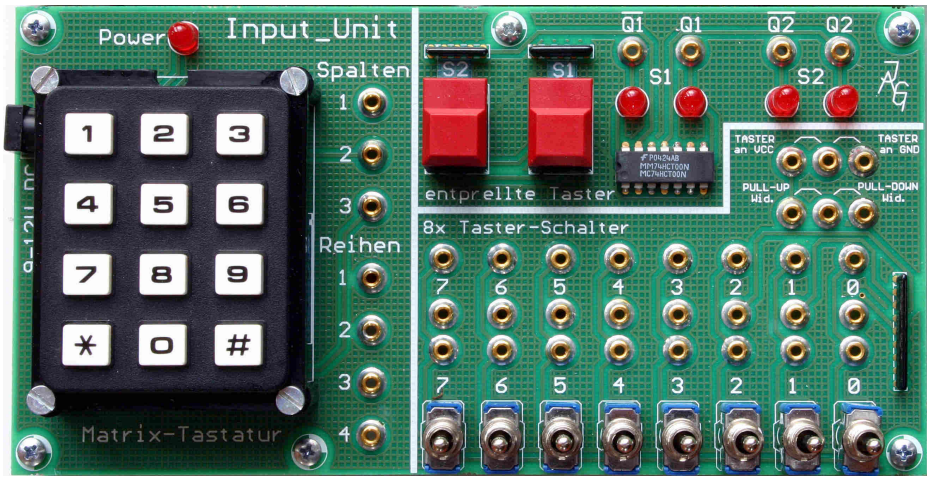
Die Eingabe-Einheit (Input_Unit) enthält eine Matrixtastatur mit 12 Tasten, zwei entprellte Schalter und acht Kombischalter welche als Schalter (nach oben) und Taster (nach unten) funktionieren. An diese Platine wird auch die Betriebsspannung von 9 V bis 12 V angeschlossen. Unter der Matrixtastatur befindet sich ein Spannungsregler (7805).

Auf der Anzeige-Einheit befinden sich 16 Low-Power-LEDs und 4 Sieben-Segment-Anzeigen.

Die Controllerplatine enthält neben dem Controller einen externen Quarz, zwei Buchsen für die ISP-Programmierung und eine serielle Schnittstelle zur Datenkommunikation.

Die vierte Platine ermöglicht es kleinere Platinen mit Zusatzfunktionen aufzustecken.

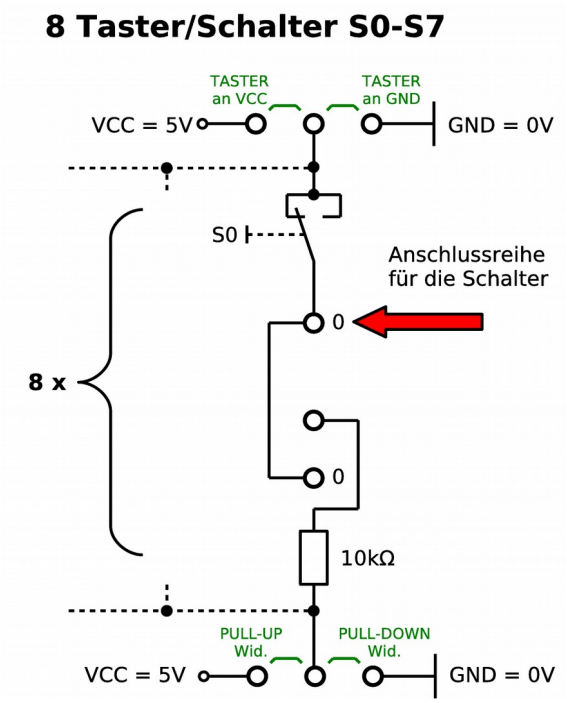
Eingabe-Einheit (Input-Unit)

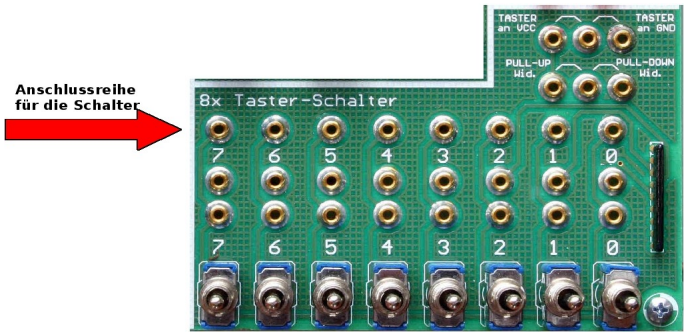


8 Taster bzw. Schalter

Um größtmögliche Flexibilität zu erreichen, kann man die Schalter/Taster beliebig mit Hilfe von 7,5 mm Steckbrücken verschalten. Leider leidet darunter die Übersichtlichkeit auf dem Board. Als Anschlussreihe für die Schalter/Taster dient die obere nummerierte Reihe auf dem Board. Die unteren beiden Reihen dienen dazu Steckbrücken zu setzen um gegeben, falls externe Pull-Up oder Pull-Down-Widerstände zuzuschalten. Die Schalter sind nicht entprellt.

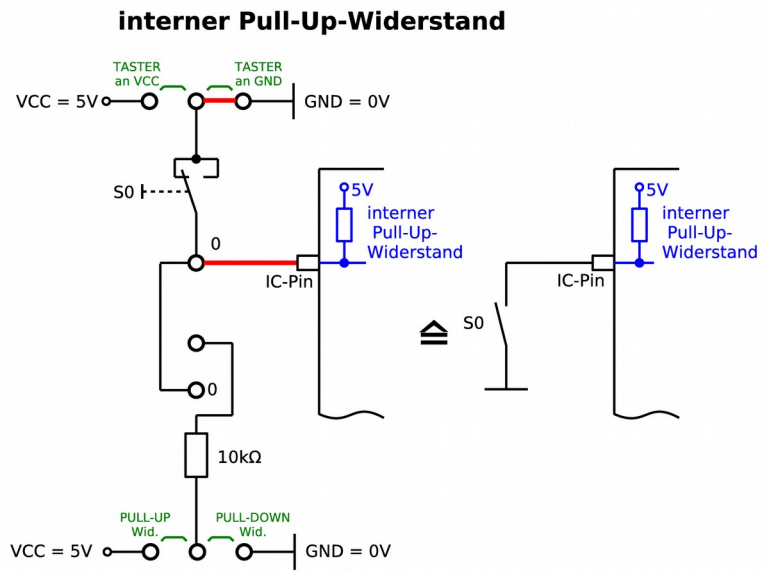
Schaltung:

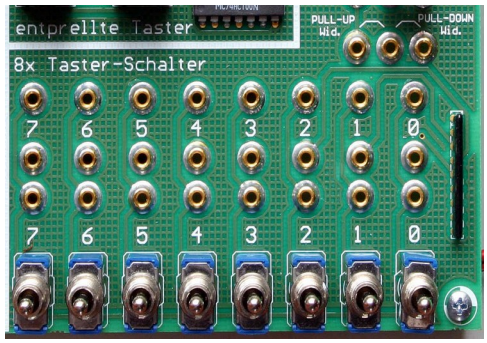




Schalter in Verbindung mit internen Pull-Up-Widerständen

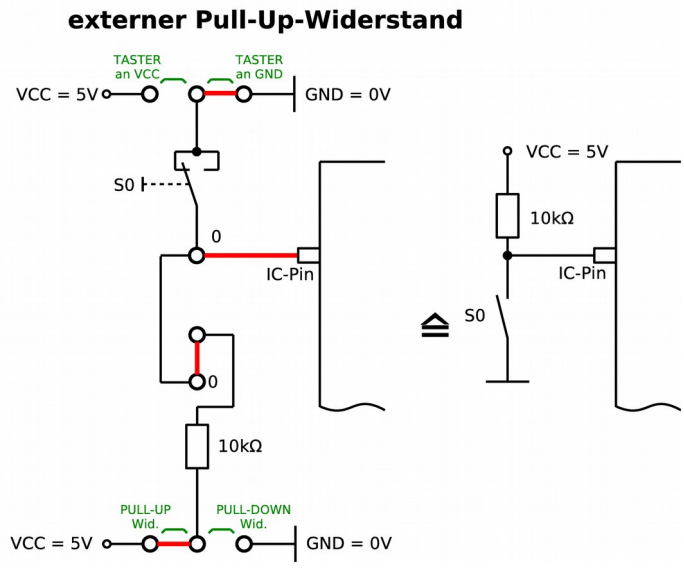
Da die Port-Ausgänge der ATmega-Controller die Möglichkeit bieten interne Pull-Up- Widerstände zuzuschalten kann man externe Widerstände sparen. Die externen Schalter müssen also einfach nach Masse verschaltet werden. Man arbeitet dann mit negativer Logik: Schalter betätigt entspricht 0 V; Schalter nicht betätigt entspricht 5 V.

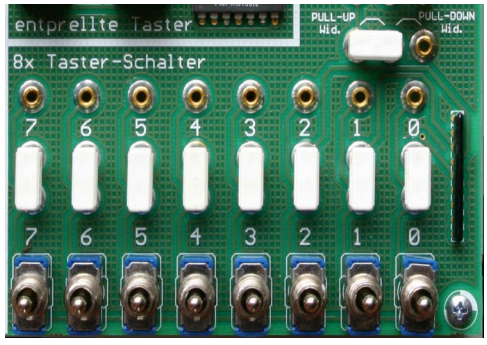




Schalter in Verbindung mit externen Pull-Ups

Ohne weiteres lassen sich natürlich auch externe Pull-Ups verwenden. Ist dabei der interne Pull-Up durch die Software eingeschaltet, so ergibt sich eine Parallelschaltung des externen und internen Widerstandes (40 kΩ || 10 kΩ). Dabei sinkt der Wert auf ungefähr 8 kΩ, was kein Problem darstellt.



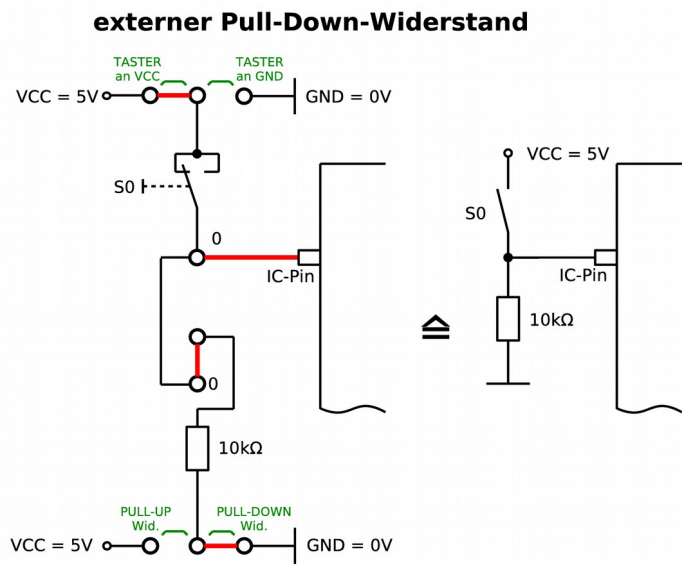


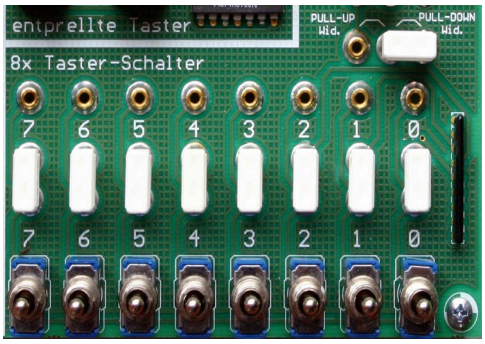
Schalter in Verbindung mit externen Pull-Downs

Möchte man mit positiver Logik (Schalter betätigt entspricht 5 V) arbeiten, so benötigt man einen externen Pull-Down-Widerstand.

Hier muss unbedingt darauf geachtet werden, dass der interne Pull-Up-Widerstand nicht aktiviert ist!!

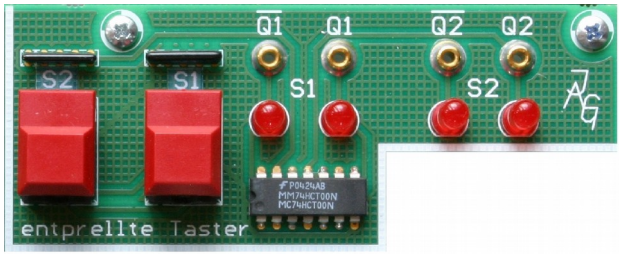
Sollte dies der Fall sein, so entsteht ein Spannungsteiler mit beiden Widerständen, der zu nicht definierten Pegeln führen könnte.



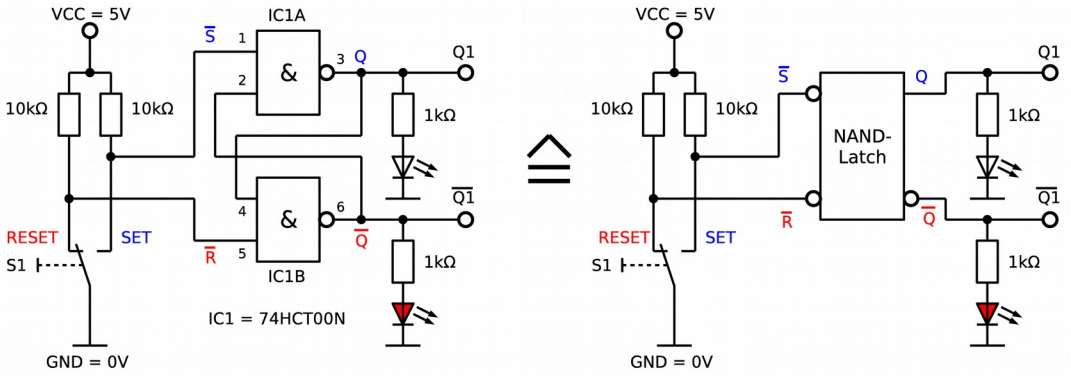


Zwei entprellte Taster

Am oberen rechten Rand des Feldes befinden sich zwei mit Hilfe von Flip-Flops entprellte Taster mit je zwei Ausgängen. Die Zustände der Ausgänge werden mit LEDs angezeigt. Der negierte Ausgang führt Null bei Betätigung der Taster (negative Logik).

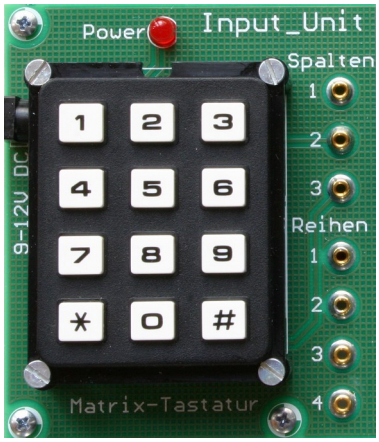


Schaltung:

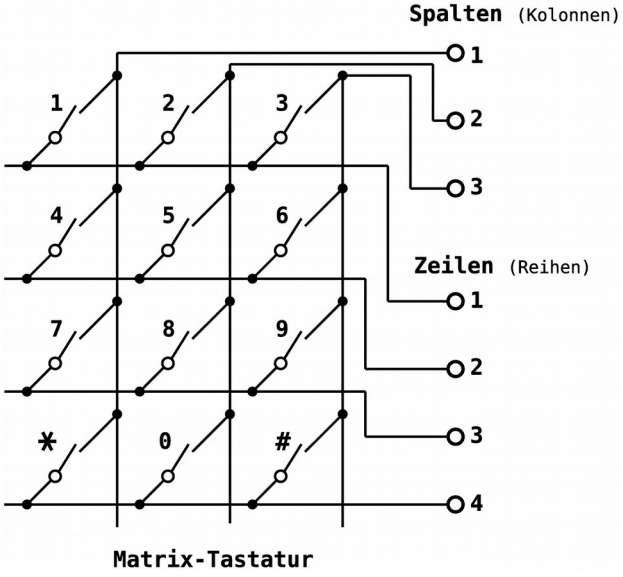


Matrix-Tastatur

Am linken Rand Feldes befindet sich die Matrix-Tastatur. Sie bietet 12 Tasten (10 Ziffern, Stern und Raute), die in vier Zeilen (Reihen) und drei Spalten (Kolonnen) angeordnet sind.



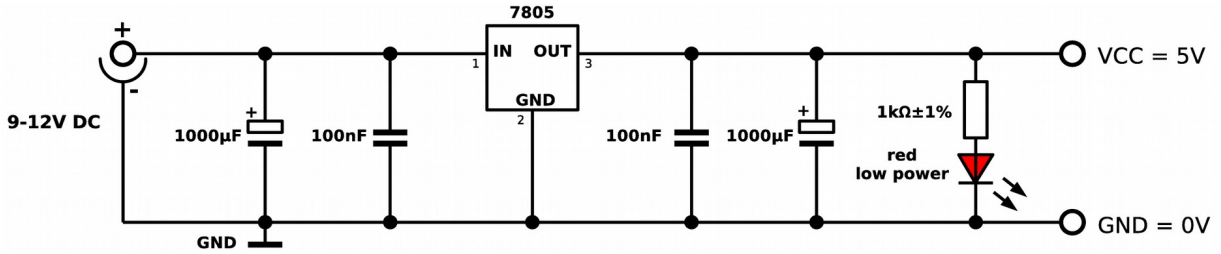
Schaltung:



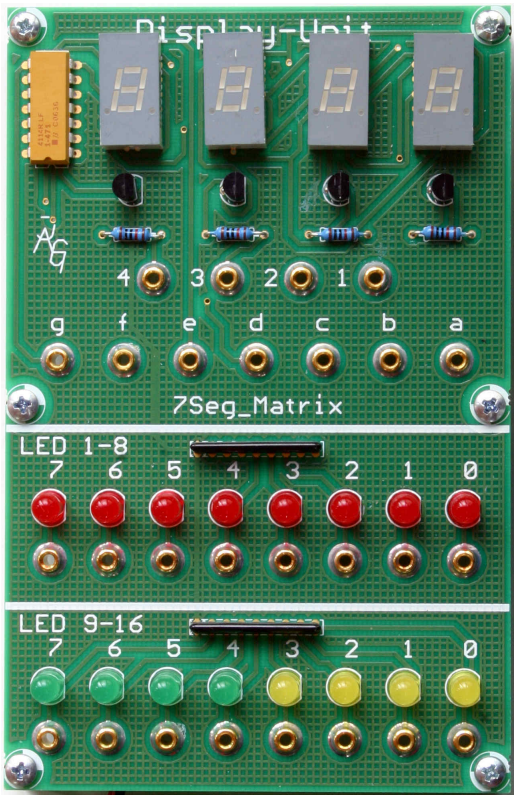
Spannungsstabilisierung

Unter der Matrix-Tastatur befindet sich eine Schaltung zur Stabilisierung der Eingangsgleichspannung auf 5V. Es handelt sich hier um eine klassische Spannungs-stabilisierungsschaltung mit einem 7805 IC. Die Eingangsgleichspannung soll zwischen 9 und 12V betragen.

Schaltung:

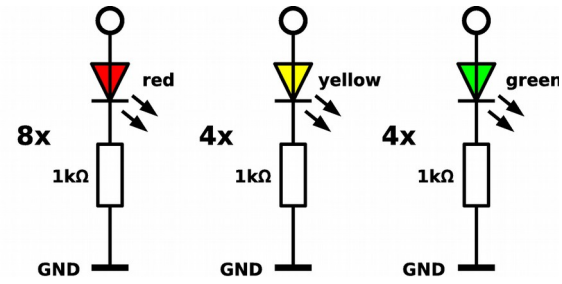


Anzeige-Einheit (Display-Unit)



LEDs

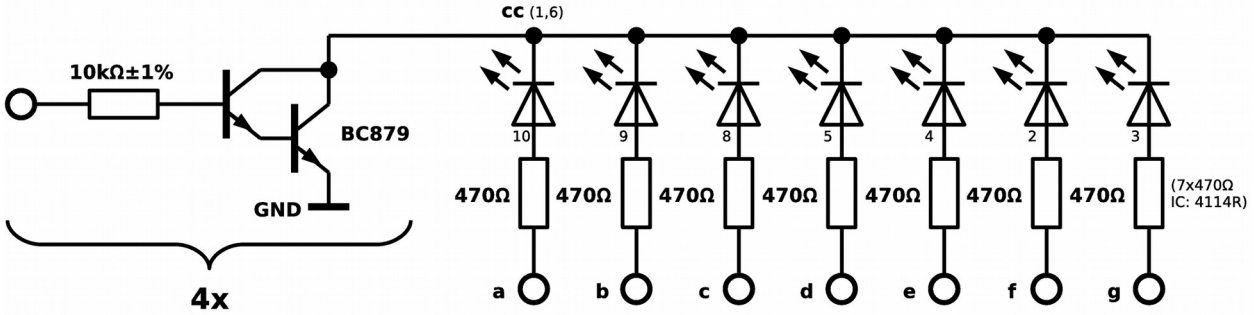
16 Low-Power-LEDs ermöglichen es 2 Byte darzustellen. Sie sind in drei Farben ausgeführt um unterschiedliche Signale besser voneinander unterscheiden zu können (Beispiel: Ampelsteuerung).

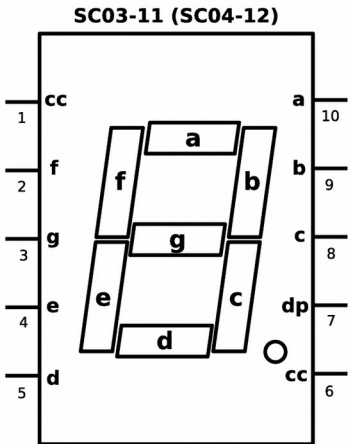


Sieben-Segment-Anzeige

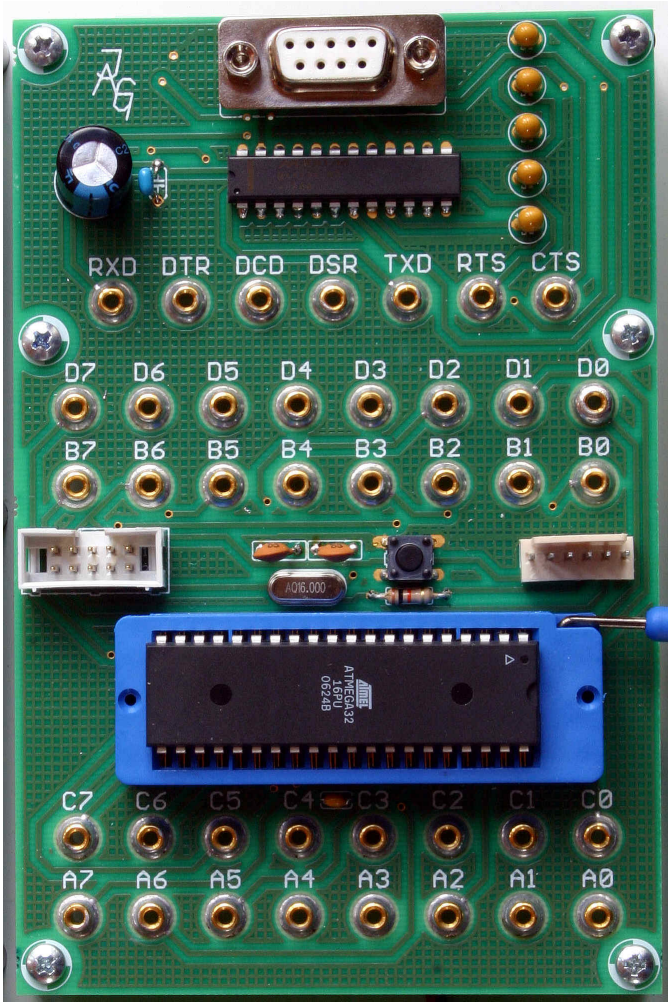
Die digitale Anzeige mit vier Stellen (digits) ermöglicht unter anderem die hexadezimale Darstellung eines Doppelregisters (z.B.: 16-Bit-Adresszeiger **Z**). Jede Stelle wird einzeln mit einer Eins (5 V) angesteuert, ebenso wie jedes einzelne Segment. Durch eine schnell aufeinander folgende Ausgabe auf alle vier Stellen kann dann eine ganze Zahl dargestellt werden.

Es werden Anzeigen mit hoher Leuchdichte bei geringem Strom und gemeinsamer Kathode verwendet. Der Darlingtontransistor invertiert das Signal und benötigt einen geringen Eingangsstrom (hohe Stromverstärkung).



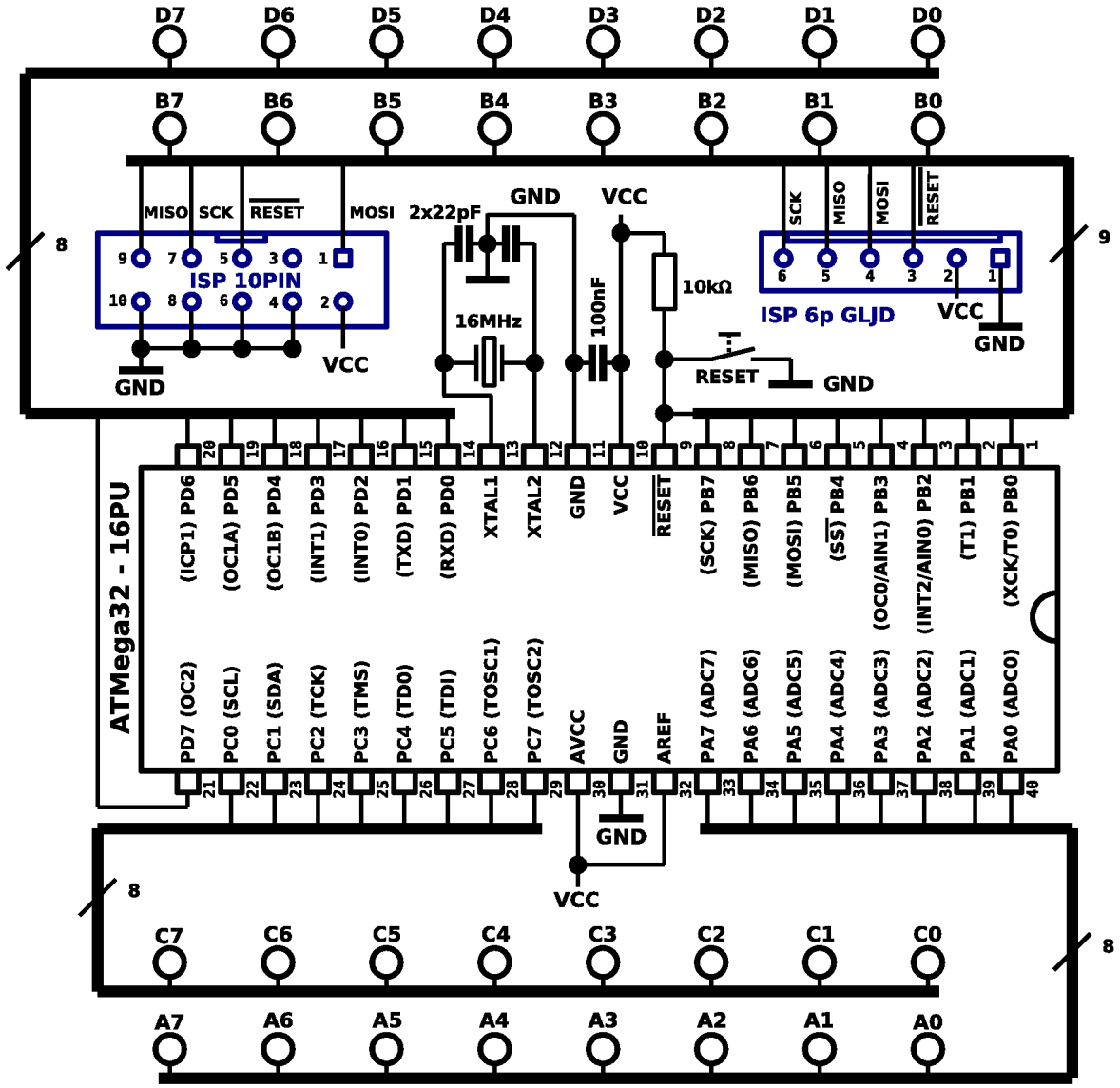


Controller-Einheit (Controller-Unit)



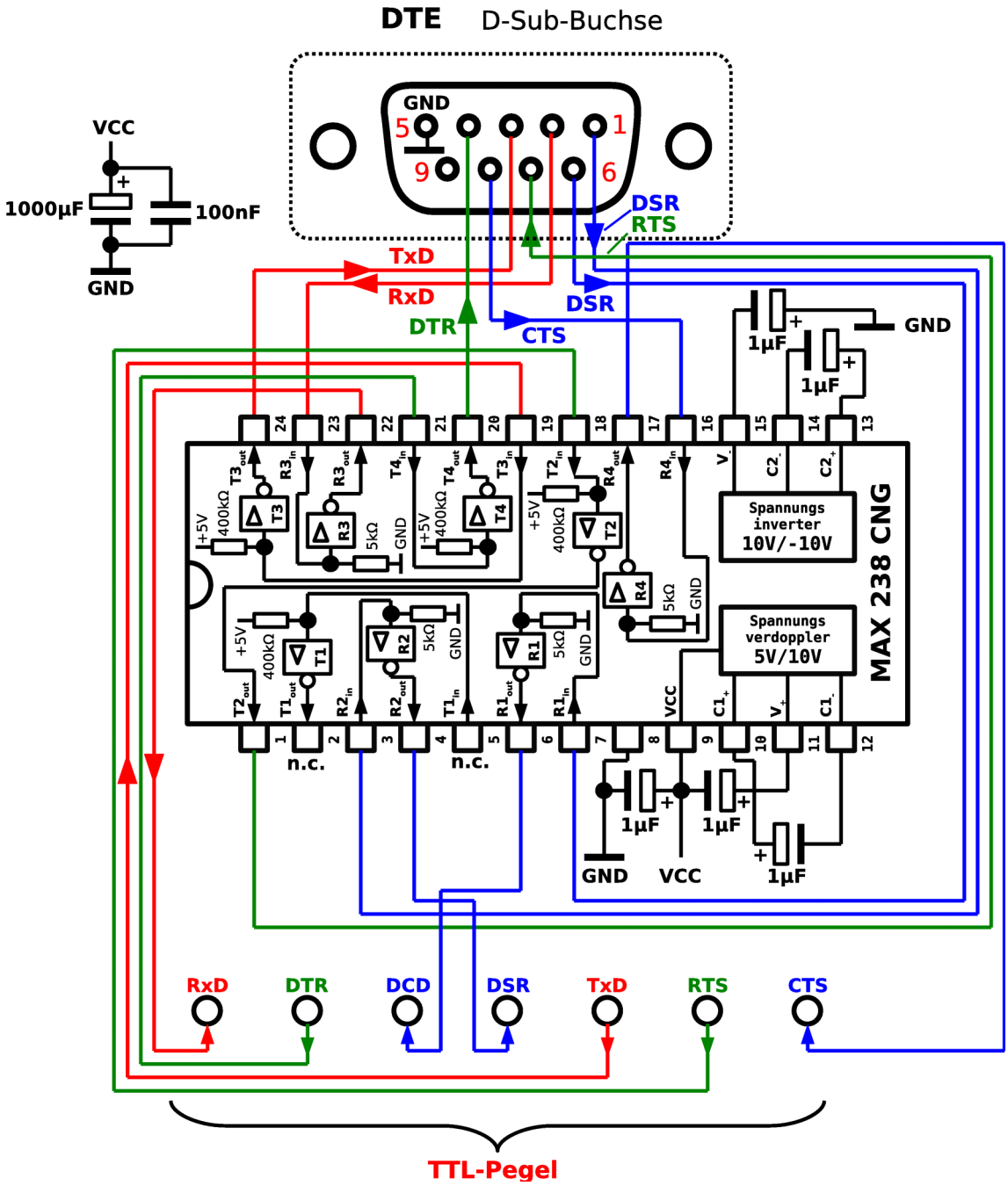
Mikrocontroller

Alle 32 Anschlüsse des Controllers sind einzeln mit 2 mm-Buchsen verbunden. Der Controller selbst sitzt in einer Nullkraft-Fassung und kann leicht gewechselt werden. Neben dem Controller sind ein 16 MHz Quarz, ein RESET-Taster sowie 2 ISP-Buchsen zum Anschließen eines Programmiergerätes vorhanden. Die 10-polige Buchse ist wie von ATMEL® vorgegeben belegt. Bei der 6-poligen Buchse handelt es sich um eine proprietäre Anschlussvariante.



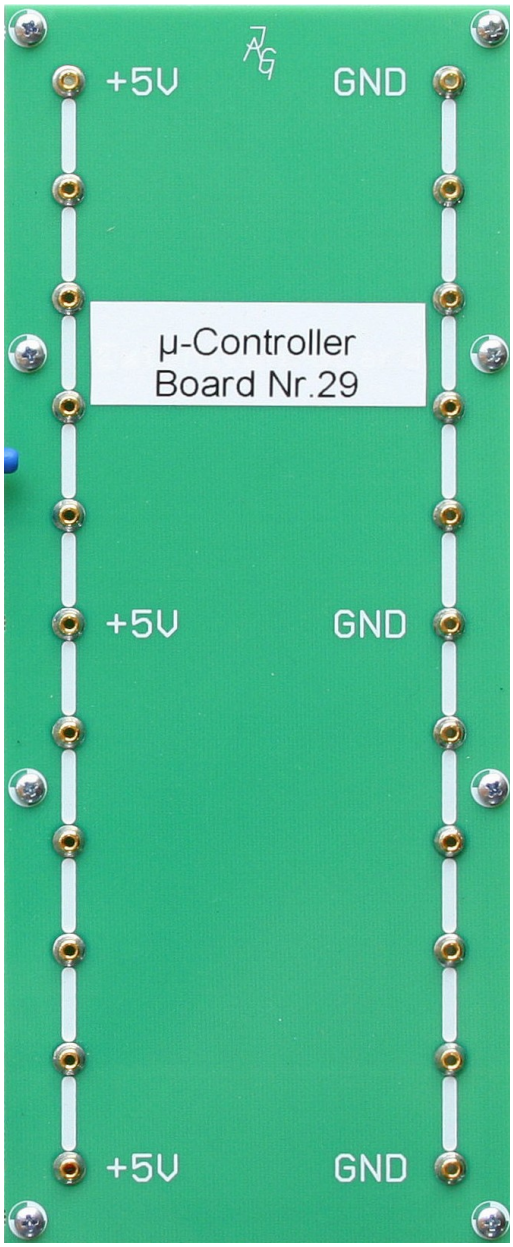
EIA232-Schnittstelle

Mit Hilfe des Bausteins MAX 238 CND werden die TTL-Schnittstellensignale in EIA232 Signale umgewandelt und umgekehrt. Außer RI sind alle Signale vorhanden. Das Mikrocontroller-Board fungiert als Daten-End-Einrichtung (**DEE**, engl.: **DTE**). Um ein klassisches Null-Modem-Kabel verwenden zu können ist ein "Gender Changer" nötig, da am Board eine Buchse zur Verfügung steht.


DTE D-Sub-Buchse
GND
5
1
9
6
DSR
RTS
TxD
RxD
DTR
DSR
CTS
VCC
1000µF
100nF
GND
1µF
1µF
GND
Spannungs inverter 10V/-10V
Spannungs verdoppler 5V/10V
MAX 238 CNG
n.c.
n.c.
1µF
1µF
1µF
GND
VCC
RxD
DTR
DCD
DSR
TxD
RTS
CTS
TTL-Pegel

Erweiterungs-Einheit (Expansion-Unit)

Die Erweiterungs-Einheit erlaubt es weitere kleine Schaltungen einfach einzubinden. Werden auf der Erweiterungsplatine 2mm-Stecker angebracht, so kann die Platine aufgesteckt werden. Über die Stecker wird die Platine mit Spannung versorgt.



F8 Anhang zum ATmega8A

Befehlssatz des ATmega8A

(Quelle: Atmel® Datenblatt ATmega8A)

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← 0xFF – Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← 0x00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • (0xFF - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd – 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← 0xFF	None	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr	Z,C	2
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC ← Z	None	3
RET		Subroutine Return	PC ← STACK	None	4
RETI		Interrupt Return	PC ← STACK	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← 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 ← PC + 2 or 3	None	1 / 2 / 3
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1 / 2 / 3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ← PC + 2 or 3	None	1 / 2 / 3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC ← PC + 2 or 3	None	1 / 2 / 3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC+k + 1	None	1 / 2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC+k + 1	None	1 / 2
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1 / 2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1 / 2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1 / 2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1 / 2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1 / 2
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1 / 2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1 / 2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1 / 2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1 / 2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1 / 2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1 / 2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1 / 2

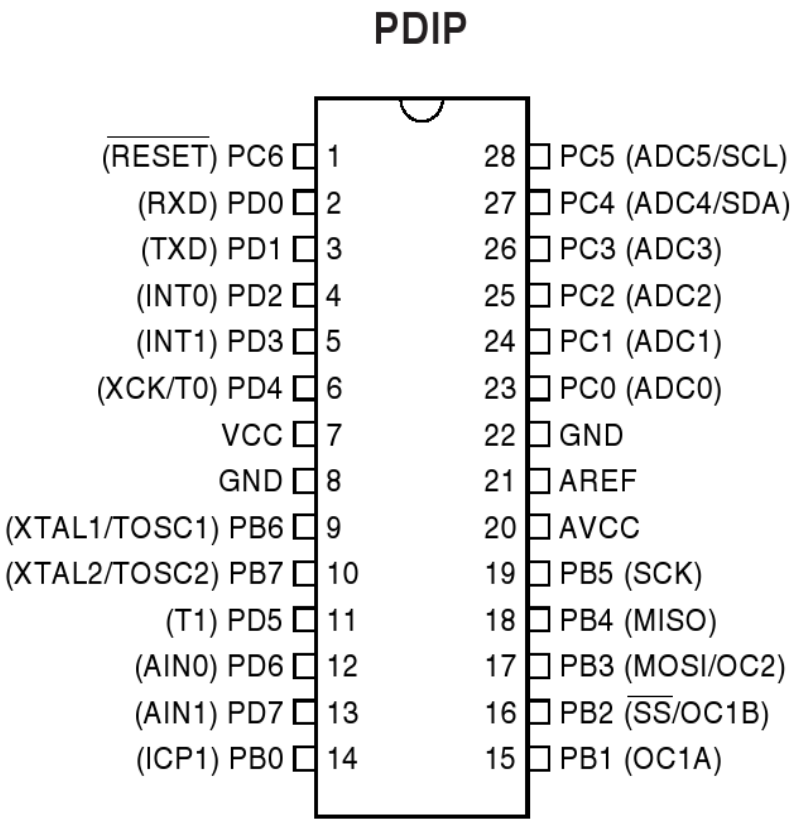
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1 / 2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1 / 2
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, 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
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

Im Befehlssatz werden für Operanden folgende Abkürzungen verwendet:

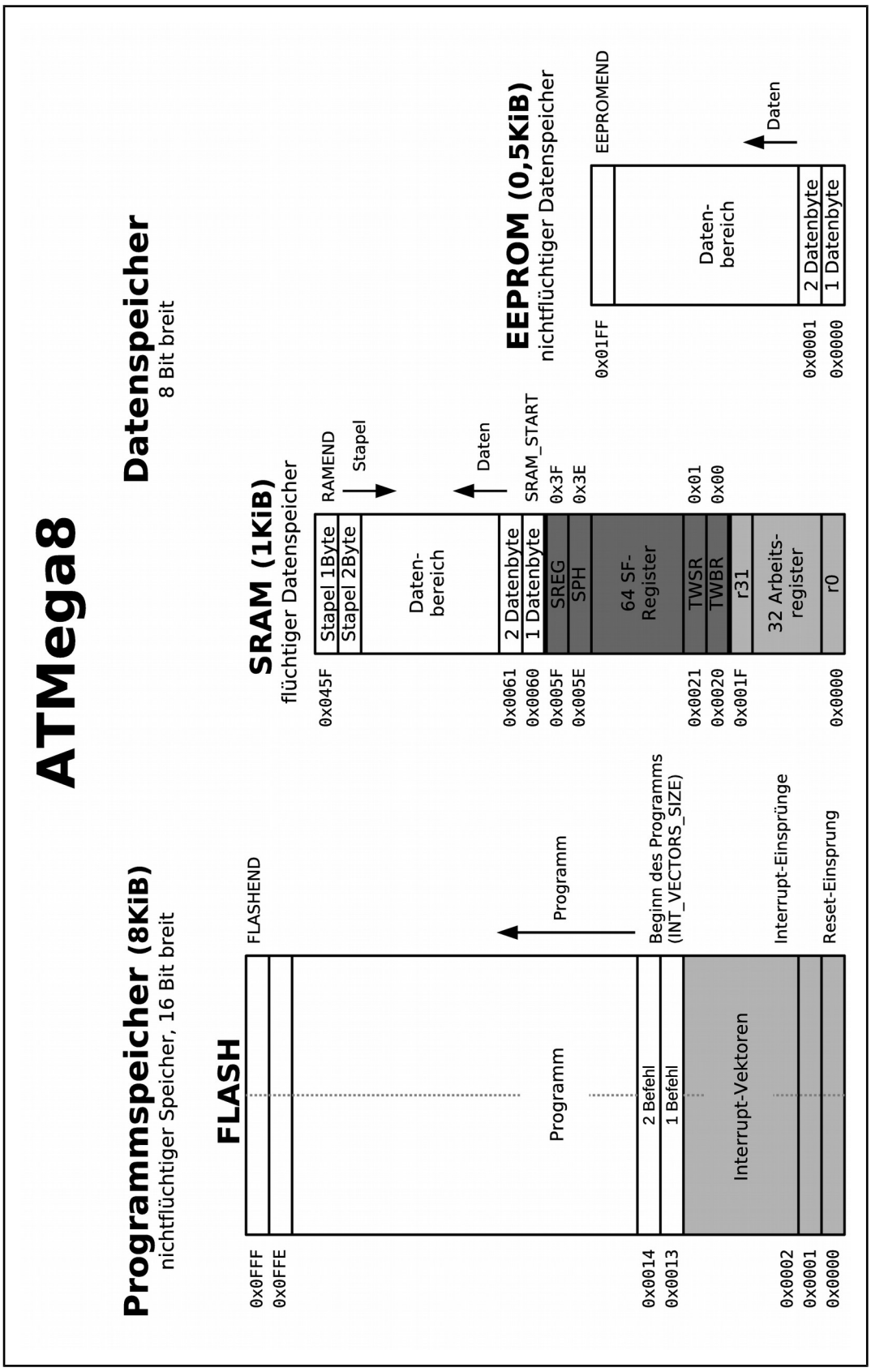
Rd	Meist Ziel-Arbeitsregister (destination, bei einigen Befehlen auch Quelle) **r0**-**r31** (bei immediate Befehlen nur **r16**-**r31**)
Rdl	Niederwertiges Byte (LByte) eines 16 Bit Ziel-Arbeitsregister
Rr	Quell- oder Sende-Arbeitsregister (**r0**-**r31**, source)
K	Daten-Konstante 8 Bit (0-255)
k	Adress-Konstante für Operationen mit dem "program counter PC". (Bsp.: Label für einen Sprung)
b	Bitkonstante 3 Bit (0-7), zum Auswählen eines Bits in einem Arbeits- oder SF-Registers
P	Adresse eines SF-Registers 6 Bit (0-63)
s	Bitkonstante 3 Bit (0-7), zum Auswählen eines Bits im Statusregister
X,Y,Z	Doppelregister (Pointer, Adresszeiger) zur direkten Adressierung **X** ⇒ **r27:r26**; **Y** ⇒ **r29:r28**, **Z** ⇒ **r31:r30**

Pinbelegung des ATmega8A

(Quelle: Atmel® Datenblatt ATmega8A)



Speicherorganisation des ATmega8A



Interrupt-Vektortabelle des ATmega8A

Vektor-nummer	Adresse im Flash	Name in "m32def.inc"	Quelle des Interrupts	Verantwortlich für den Interrupt
19	0x0024	SPMRaddr	SPM_RDY	Programmspeicher (Flash)-Programmierung fertig
18	0x0022	TWIaddr	TWI	I^2C-Interface (TWI-Interface)
17	0x0020	ACIaddr	ANA_COMP	Analog-Komparator
16	0x001E	ERDYaddr	EE_RDY	EEPROM-Programmierung (schreiben) fertig
15	0x001C	ADCCaddr	ADC	AD-Wandlung vollständig
14	0x001A	UTXCaddr	USART, TXC	USART, Zeichen gesendet (Senderegister leer)
13	0x0018	UDREaddr	USART, UDRE	USART, UDR-Register leer
12	0x0016	URXCaddr	USART, RXC	USART, Empfangsregister voll
11	0x0014	SPIaddr	SPI, STC	Serielle Übertragung beendet (SPI)
10	0x0012	OVF0addr	TIMER0 OVF	Overflow von Timer 0
9	0x0010	OVF1addr	TIMER1 OVF	Overflow von Timer 1
8	0x000E	OC1Baddr	TIMER1 COMPB	Compare Match B von Timer 1
7	0x000C	OC1Aaddr	TIMER1 COMPA	Compare Match A von Timer 1
6	0x000A	ICP1addr	TIMER1 CAPT	Capture Event von Timer 2
5	0x0008	OVF2addr	TIMER2 OVF	Overflow Match von Timer 2
4	0x0006	OC2addr	TIMER2 COMP	Compare Match von Timer 2
3	0x0004	INT1addr	INT1	Interrupt-Eingang 1 (externer PIN PD3)
2	0x0002	INT0addr	INT0	Interrupt-Eingang 0 (externer PIN PD2)
1	0x0000		RESET	RESET Pin (extern Pin 9), Power-On-Reset, Brown-Out-Reset, Watchdog Reset und JTAG AVR-Reset

Bemerkungen:

Das eigentliche Programm kann erst an der Adresse **0x0026** beginnen. Der in der Definitionsdatei vorgesehene Name für diese Adresse heißt: **INT_VECTORS_SIZE**.

Wenn das **BOOTRST**-Fuse-Bit programmiert wurde springt der Controller nach einem **RESET** automatisch in den Bootbereich (Bootloader-Programm) im oberen Adressbereich des Flash-Speichers. Mit Hilfe des **IVSEL**-Bit im SF-Register **GICR** kann die Interrupt-Vektortabelle in den Anfang des Bootbereichs verlegt werden.

SF-Registersatz des ATmega8A

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x3F (0x5F)	SREG	I	T	H	S	V	N	Z	C	8
0x3E (0x5E)	SPH	–	–	–	–	–	SP10	SP9	SP8	11
0x3D (0x5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	11
0x3C (0x5C)	Reserved									
0x3B (0x5B)	GICR	INT1	INT0	–	–	–	–	IVSEL	IVCE	48, 68
0x3A (0x5A)	GIFR	INTF1	INTF0	–	–	–	–	–	–	69
0x39 (0x59)	TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	–	TOIE0	73, 104, 124
0x38 (0x58)	TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	–	TOV0	74, 104, 104
0x37 (0x57)	SPMCR	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	224
0x36 (0x56)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	191
0x35 (0x55)	MCUCR	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	36, 67
0x34 (0x54)	MCUCSR	–	–	–	–	WDRF	BORF	EXTRF	PORF	43
0x33 (0x53)	TCCR0	–	–	–	–	–	CS02	CS01	CS00	73
0x32 (0x52)	TCNT0	Timer/Counter0 (8 Bits)								73
0x31 (0x51)	OSCCAL	Oscillator Calibration Register								31
0x30 (0x50)	SFIOR	–	–	–	–	ACME	PUD	PSR2	PSR10	57, 77, 125, 196
0x2F (0x4F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	99
0x2E (0x4E)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	101
0x2D (0x4D)	TCNT1H	Timer/Counter1 – Counter Register High byte								102
0x2C (0x4C)	TCNT1L	Timer/Counter1 – Counter Register Low byte								102
0x2B (0x4B)	OCR1AH	Timer/Counter1 – Output Compare Register A High byte								103
0x2A (0x4A)	OCR1AL	Timer/Counter1 – Output Compare Register A Low byte								103
0x29 (0x49)	OCR1BH	Timer/Counter1 – Output Compare Register B High byte								103
0x28 (0x48)	OCR1BL	Timer/Counter1 – Output Compare Register B Low byte								103
0x27 (0x47)	ICR1H	Timer/Counter1 – Input Capture Register High byte								103
0x26 (0x46)	ICR1L	Timer/Counter1 – Input Capture Register Low byte								103
0x25 (0x45)	TCCR2	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	121
0x24 (0x44)	TCNT2	Timer/Counter2 (8 Bits)								123
0x23 (0x43)	OCR2	Timer/Counter2 Output Compare Register								123
0x22 (0x42)	ASSR	–	–	–	–	AS2	TCN2UB	OCR2UB	TCR2UB	123
0x21 (0x41)	WDTCR	–	–	–	WDCE	WDE	WDP2	WDP1	WDP0	43
0x20[1] (0x40)[1]	UBRRH	URSEL	–	–	–	UBRR[11:8]				160
	UCSRC	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	159
0x1F (0x3F)	EEARH	–	–	–	–	–	–	–	EEAR8	19
0x1E (0x3E)	EEARL	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	19
0x1D (0x3D)	EEDR	EEPROM Data Register								19
0x1C (0x3C)	EECR	–	–	–	–	EERIE	EEMWE	EEWE	EERE	19
0x1B (0x3B)	Reserved									
0x1A (0x3A)	Reserved									
0x19 (0x39)	Reserved									
0x18 (0x38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	65
0x17 (0x37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	65
0x16 (0x36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	65
0x15 (0x35)	PORTC	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	65
0x14 (0x34)	DDRC	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	65
0x13 (0x33)	PINC	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	65
0x12 (0x32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	65
0x11 (0x31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	65
0x10 (0x30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	66
0x0F (0x2F)	SPDR	SPI Data Register								135
0x0E (0x2E)	SPSR	SPIF	WCOL	–	–	–	–	–	SPI2X	134
0x0D (0x2D)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	133
0x0C (0x2C)	UDR	USART I/O Data Register								156
0x0B (0x2B)	UCSRA	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	157
0x0A (0x2A)	UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	158
0x09 (0x29)	UBRRL	USART Baud Rate Register Low byte								160
0x08 (0x28)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	196
0x07 (0x27)	ADMUX	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	208
0x06 (0x26)	ADCSRA	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	209
0x05 (0x25)	ADCH	ADC Data Register High byte								210
0x04 (0x24)	ADCL	ADC Data Register Low byte								210
0x03 (0x23)	TWDR	Two-wire Serial Interface Data Register								193
0x02 (0x22)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	194
0x01 (0x21)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	193
0x00 (0x20)	TWBR	Two-wire Serial Interface Bit Rate Register								191

Note:
1. Refer to the USART description for details on how to access UBRRH and UCSRC.
2. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
3. Some of the Status Flags are cleared by writing a logical one to them. Note that the CBI and SBI instructions will operate on all bits in the I/O Register, writing a one back into any flag read as set, thus clearing the flag. The CBI and SBI instructions work with registers 0x00 to 0x1F only.

Blockschaltbild des ATmega8A

(Quelle: Atmel® Datenblatt ATmega8A)