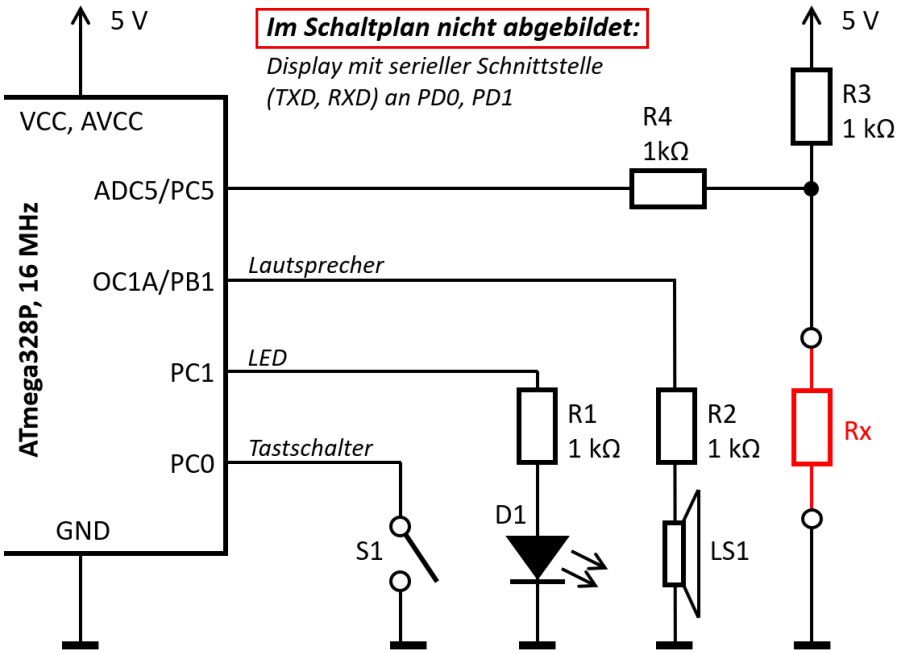
Hochschule München FK03	Intelligente Hardware und Eingebettete Systeme		Prof. Dr.-Ing. T. Küpper
Zugelassene Hilfsmittel: alle eigenen, Taschenrechner	**Matr.-Nr.:**	**Name, Vorname:**	
	Hörsaal:	**Unterschrift:**	

WiSe 2025/26
Viel Erfolg!!

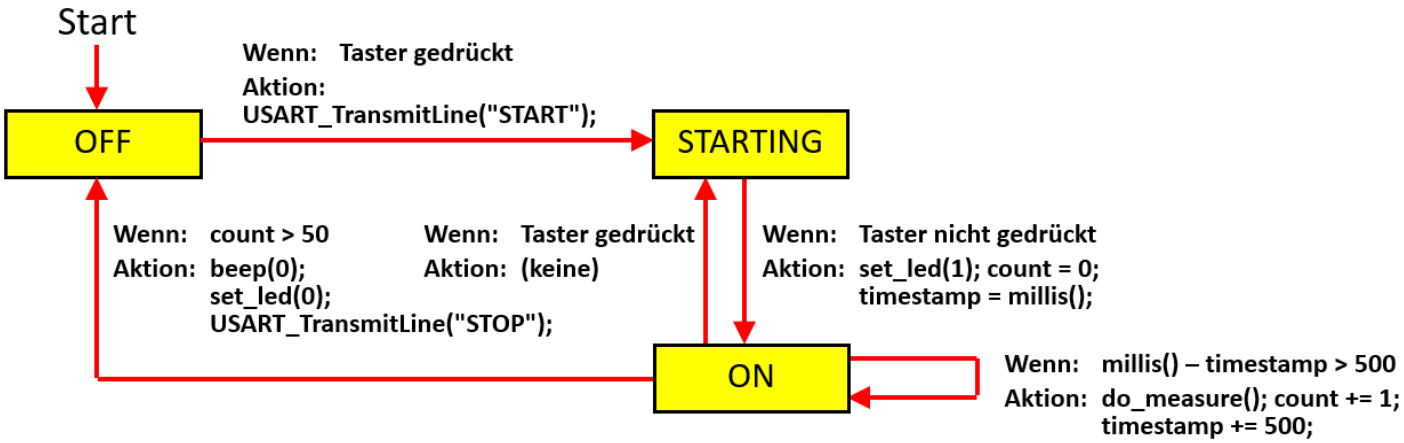
1	2	3	4	5	6		Σ	N

Aufgabe 1 von 5: Programmierung (ca. 40 Punkte ≙ 40 Minuten)

In dieser Aufgabe sollen Sie ein System entwickeln, das elektrische Widerstände misst und die Messergebnisse über ein Display ausgibt. Grundlage des Systems ist ein ATMega328P-Mikrocontroller, der mit einer Taktfrequenz von 16 MHz betrieben wird. Das System verfügt über mehrere Ein- und Ausgabeelemente: Neben dem zu messenden Widerstand Rx sind eine Leuchtdiode, ein Taster sowie ein Display angeschlossen. Bei Widerstandswerten von weniger als 50 Ω wird außerdem über einen kleinen Lautsprecher ein Piepton ausgegeben.



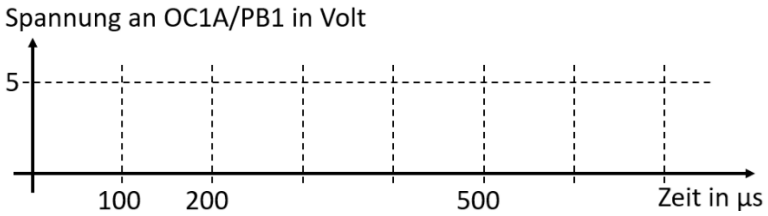
Durch Drücken des Tasters werden die Messungen gestartet. Das System besitzt eine automatische Abschaltfunktion, die nach einer gewissen Zeit den Betrieb wieder beendet. Die Messergebnisse werden auf einem seriellen Display angezeigt: Alle über die serielle Schnittstelle zum Display gesendeten Zeichen werden dort direkt angezeigt. Das Verhalten des Systems wird durch den folgenden endlichen Automaten beschrieben:



1.1 Wozu dient die Definition von F_CPU? Was passiert, wenn hier ein falscher Wert angegeben wird?

1.2 Der Timer/Counter1 ist ein 16-Bit-Zähler. In welchem Modus wird der Timer/Counter1 hier betrieben? Bis zu welchem Maximalwert zählt der Timer/Counter1 hier? Welche Frequenz wird an OC1A/PB1 ausgegeben?

1.3 Skizzieren Sie den Spannungsverlauf an OC1A/PB1, nachdem der Piepton mittels beep(1); eingeschaltet wurde.



1.4 Nehmen Sie an, dass in der Funktion init_timer1() der Wert OCR1A = 102; eingestellt wird. Skizzieren Sie den Spannungsverlauf an OC1A/PB1, der sich nun ergibt (im selben Diagramm).

```
// Widerstandsmessung mit seriellem Display und Lautsprecher
#define F_CPU 16000000UL  // *** Aufgabe 1.1 ***
#define BAUD 9600
#define MYUBRR F_CPU/16/BAUD-1
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <stdio.h>
#include <stdint.h>

// USART initialisieren, siehe Datenblatt, 24.6. USART Initialization
void USART_Init(uint16_t ubrr)
{
    UBRR0L = (uint8_t)(ubrr);
    UBRR0H = (uint8_t)(ubrr >> 8);          // Set baud rate
    UCSR0B = _BV( RXEN0) + _BV( TXEN0);     // Enable receiver & transmitter
    UCSR0C = _BV(UCSZ00) + _BV(UCSZ01);     // 8 data bits, 1 stop bit
}

// Ein Zeichen senden, siehe Datenblatt, 24.7. Data Transmission
void USART_Transmit(uint8_t data)
{
    while(!(UCSR0A & _BV(UDRE0))) { }       // Wait for empty transmit buffer
    UDR0 = data;                            // Put data into transmit buffer
}

// Ein Zeichen empfangen, siehe Datenblatt, 24.8. Data Reception
uint8_t USART_Receive(void)
{
    while (!(UCSR0A & _BV(RXC0))) { }       // Wait for data to be received
    return UDR0;                            // Get and return received data
}

// Komplette Textzeile inkl. Zeilenumbruch senden  *** Aufgabe 1.10 ***
void USART_TransmitLine(const char *text)
{
```

```
// Timer/Counter0 initialisieren, 1000 Interrupts pro Sekunde  *** Aufgabe 1.11 ***
void init_timer0(void)
{









}

// Timer-Interrupt, Millisekunden seit Programmstart zählen
volatile uint32_t __millis__internal__ = 0;  // *** Aufgabe 1.9 ***
ISR(TIMER0_COMPA_vect)
{
    __millis__internal__ += 1;
}

// Millisekunden seit Programmstart abfragen
uint32_t millis(void)
{
    uint32_t result = 0;
    cli(); result = __millis__internal__; sei();  // *** Aufgabe 1.5 ***
    return result;
}

// Timer/Counter1 initialisieren  *** Aufgabe 1.2 ***
void init_timer1(void)
{
    TCCR1A = _BV(WGM10) + _BV(WGM11);
    TCCR1B = _BV(WGM12) + _BV(CS11);
    OCR1A  = 511;                  *** Aufgabe 1.3 / 1.4 ***
}

// Piepton an OC1A/PB1 ein- (1) oder ausschalten (0)
void beep(int on)                  *** Aufgabe 1.3 / 1.4 ***
{
    if(on == 1) { TCCR1A |=  _BV(COM1A1); }
    if(on == 0) { TCCR1A &= ~_BV(COM1A1); }
}

// AD-Wandler initialisieren      *** Aufgabe 1.6 ***
void init_adc(void)
{
    ADMUX = _BV(REFS0) | _BV(MUX0) | _BV(MUX2);
    ADCSRA = _BV(ADEN) | _BV(ADPS2) | _BV(ADPS1) | _BV(ADPS0);
}

// Analogwert von ADC5 einlesen, Rückgabewert 0...1023
uint16_t read_adc5(void)
{
    ADCSRA |= _BV(ADSC);           // Wandlung starten
    while(ADCSRA & _BV(ADSC)) { }  // Auf das Ende der Wandlung warten
    return ADC;
}

// Eingang mit Pullup: Taster/PC0, Ausgänge: LED/PC1, Lautsprecher/PB1  *** Aufgabe 1.12 ***
void init_ports(void)
{









}

// Ist der Taster an PC0 aktuell ein- (1) oder ausgeschaltet (0)?  *** Aufgabe 1.13 ***
int switch_pressed(void)
{
```

```
// LED an PC1 ein- (1) oder ausschalten (0)  *** Aufgabe 1.14 ***
void set_led(int on)
{




}

// Unbekannten Widerstand Rx messen, Messung auf Display ausgeben
void do_measure(void)
{
    double supply_voltage = 4.85, low_resistance = 50.0;
    double vx = supply_voltage * read_adc5() / 1024;
    double ix = (supply_voltage - vx) / 1000;

    // Widerstand Rx berechnen, falls Strom ix wenigstens 10 µA beträgt
    double rx = -1;
    if(ix > 1e-5) { rx = vx / ix; }  // Widerstand angeschlossen, Stromfluss vorhanden?

    // Piepton bei kleinen Widerstandswerten einschalten
    int8_t beep_on_off = 0;
    if(0 <= rx && rx < low_resistance) { beep_on_off = 1; }
    beep(beep_on_off);

    // Ergebnis auf serieller Schnittstelle bzw. Display ausgeben
    char buf[32];
    snprintf(buf, sizeof buf, "R = %d Ohm", (int)rx);
    USART_TransmitLine(buf);  // *** Aufgabe 1.7 ***
}

// Zustände des endlichen Automaten
#define STATE_OFF      0
#define STATE_STARTING 1
#define STATE_ON       2

// Hauptprogramm
int main(void)
{
    USART_Init(MYUBRR);
    init_adc();
    init_timer0();
    init_timer1();
    init_ports();

    uint32_t timestamp = 0;
    uint8_t state = STATE_OFF, count = 0;

    while(1)
    {
        _delay_ms(10);  // quick & dirty debounce

        switch(state)   // *** Aufgabe 1.15 ***
        {
```

1.5. Warum sind die Aufrufe cli(); und sei(); in der Funktion uint32_t millis(void) erforderlich?

1.6. Bei der Initialisierung des Analog-Digital-Wandlers (ADC) werden die ADPS2-, ADPS1- und ADPS0-Bits gesetzt. Wozu dienen diese Bits? Warum ist es sinnvoll, sie genau auf diese Werte zu setzen?

1.7. Welche Ausgabe erfolgt auf dem Display, wenn gerade kein Widerstand Rx angeschlossen ist – wenn also die entsprechenden Anschlüsse offen bleiben?

1.8. Beschreiben Sie, nach welcher Zeit sich das System automatisch abschaltet? Wie kann das automatische Abschalten verhindert werden? Hinweis: Endlicher Automat …

1.9. Warum muss die Variable __millis__internal__ in diesem Programm als „volatile“ deklariert werden?

1.10. Schreiben Sie die Funktion USART_TransmitLine() zur Übertragung einer kompletten Textzeile über die serielle Schnittstelle. Nach dem Ende Textzeile soll immer auch noch ein Zeilenumbruch übertragen werden.

1.11. Schreiben Sie die Funktion init_timer0(). Mit dieser Funktion wird der Timer/Counter0 so initialisiert, dass die Interrupt-Funktion ISR(TIMER0_COMPA_vect) 1000 mal pro Sekunde aufgerufen wird.

1.12. Schreiben Sie die Funktion init_ports() zur Initialisierung der folgenden Ein-/Ausgänge des Mikrocontrollers: Taster/PC0 (Eingang mit Pullup), LED/PC1 und Lautsprecher/PB1 (Ausgänge).

1.13. Schreiben Sie die Funktion switch_pressed(). Diese Funktion liefert den Rückgabewert 1, falls in diesem Moment der Taster gerade gedrückt ist. Ansonsten beträgt der Rückgabewert 0.

1.14. Schreiben Sie die Funktion void set_led(int on); zum Ein-/Ausschalten der LED am Anschluss PC1.

1.15. Ergänzen Sie das Hauptprogramm int main(void) so, dass der Ablauf dem auf Seite 1 abgebildeten endlichen Automaten entspricht.

Aufgabe 2 von 5: C-Programmierung mit Mikrocontrollern (ca. 8 Punkte ≙ 8 Minuten)

2.1 Wie lauten die Ergebnisse der folgenden Berechnungen mit 8-Bit-Zahlen? Ergebnisse als Binärzahl schreiben!

a) _BV(5) | _BV(5) =

b) ~_BV(5) | _BV(5) =

c) 0b00011101 | 0b00001101 | 0b00001100 =

d) _BV(5) + _BV(5) =

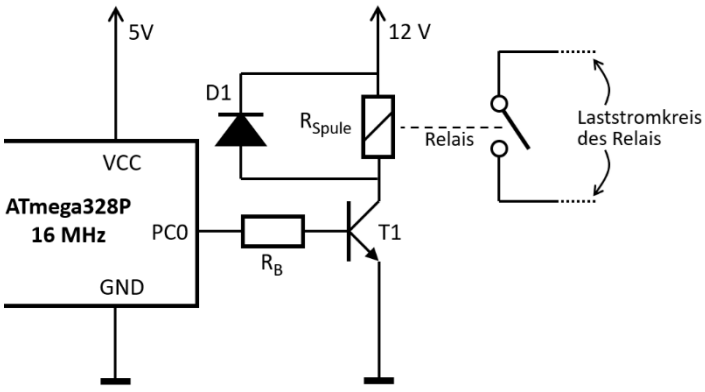
e) ~_BV(5) & _BV(5) =

f) 0b00011101 + 0b00001101 + 0b00001100 =

2.2 Ein 16-Bit-System addiert zwei Zahlen: 0xFFEF + 0x0013. Wie lautet das 16-Bit-Ergebnis als Hexadezimal-Zahl? Markieren Sie, ob ein Überlauf auftritt (Hinweis: Alle Register, inkl. Ergebnisregister, sind 16 Bit groß.)

Aufgabe 3 von 5: Ansteuerung von Transistoren und Relais (ca. 8 Punkte ≙ 8 Minuten)

Am Ausgang PC0 eines Mikrocontrollers ist ein Relais angeschlossen. Zur Ansteuerung der Relais-Spule wird ein NPN-Transistor und eine separate 12-Volt-Spannungsquelle verwendet. Die Spannung am Ausgang PC0 beträgt 4,5 V wenn dieser eingeschaltet ist. Für die Basis-Emitter-Spannung des Transistors kann in diesem Fall $U_{\mathrm{BE}} = 0{,}5\ \mathrm{V}$ angenommen werden.



3.1 Berechnen Sie einen geeigneten Vorwiderstand für die Basis des Transistors, wenn der Spulenstrom $I_{\mathrm{Spule}} = 50\ \mathrm{mA}$ beträgt und die Großsignalverstärkung des Transistors $B = 100$ ist. Wählen Sie dabei zum sicheren Einschalten des Transistors einen sinnvollen Übersteuerungsfaktor.

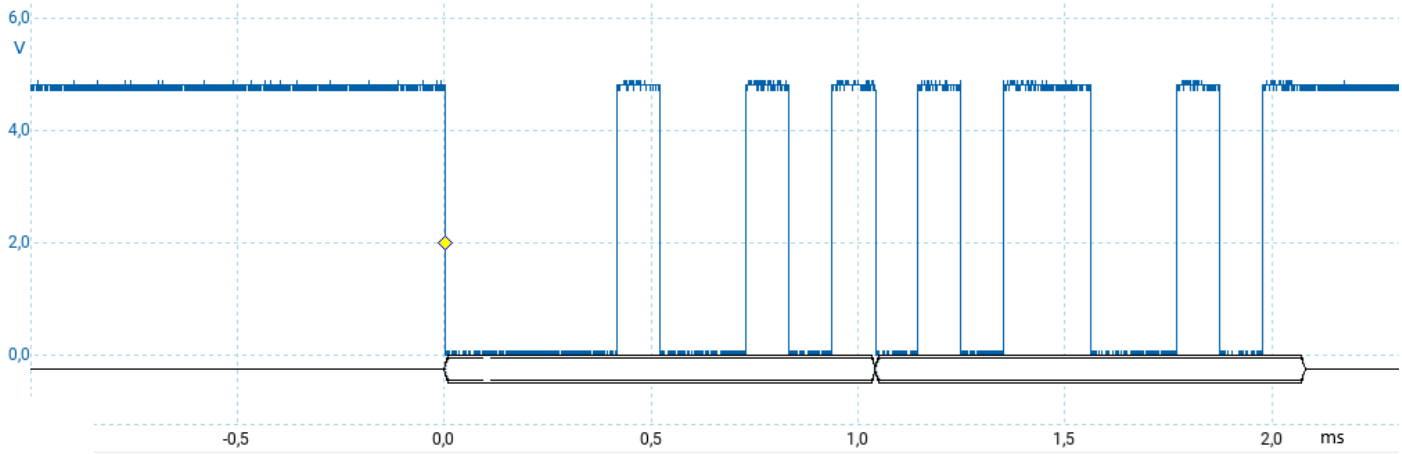
3.2 Im Bauteile-Lager ist der in 3.1 berechnete Widerstandswert R_{B} leider nicht vorrätig. Allerdings ist ein um 50% kleinerer Widerstand vorrätig. Begründen Sie, ob die Schaltung mit dem kleineren Widerstand funktioniert.

3.3 Im Laststromkreis des Relais wird nun eine LED angeschlossen. Diskutieren Sie, ob es mit dieser Schaltung möglich ist, die LED-Helligkeit per PWM einzustellen.

3.4 Wann bzw. wozu werden „Freilaufdioden“ verwendet?

Aufgabe 4 von 5: Serielle Schnittstelle (ca. 15 Punkte ≙ 15 Minuten)

Ein Mikrocontroller des Typs ATmega328P sendet zwei Zeichen über seine serielle Schnittstelle. Der dazugehörige Spannungsverlauf auf der TXD-Leitung ist in der folgenden Abbildung dargestellt:



4.1 Bestimmen Sie die beiden Zeichen (im ASCII-Format) die vom Mikrocontroller gesendet wurden. Geben Sie für beide Zeichen sowohl den Zeichen-Code als Hexadezimalzahl als auch das dazugehörige ASCII-Zeichen an. Die Parameter der seriellen Schnittstelle sind folgendermaßen gewählt:

- Baudrate: 9600 bps
- 1 Startbit, 8 Datenbits, 1 Stoppbit

4.2 Messdaten im Umfang von 5000 Bytes werden über die serielle Schnittstelle gesendet (Daten der Schnittstelle wie in 4.1). Wie lange dauert diese Übertragung mindestens? (Bitte kurze Berechnung durchführen!)

ASCII-Tabelle

DEZ	HEX	CHAR	DEZ	HEX	CHAR	DEZ	HEX	CHAR	DEZ	HEX	CHAR	DEZ	HEX	CHAR	DEZ	HEX	CHAR	DEZ	HEX	CHAR
32	20		33	21	!	34	22	"	35	23	#	36	24	$	37	25	%	38	26	&	39	27	'
40	28	(| 41 | 29 |)	42	2A	*	43	2B	+	44	2C	,	45	2D	-	46	2E	.	47	2F	/
48	30	0	49	31	1	50	32	2	51	33	3	52	34	4	53	35	5	54	36	6	55	37	7
56	38	8	57	39	9	58	3A	:	59	3B	;	60	3C	<	61	3D	=	62	3E	>	63	3F	?
64	40	@	65	41	A	66	42	B	67	43	C	68	44	D	69	45	E	70	46	F	71	47	G
72	48	H	73	49	I	74	4A	J	75	4B	K	76	4C	L	77	4D	M	78	4E	N	79	4F	O
80	50	P	81	51	Q	82	52	R	83	53	S	84	54	T	85	55	U	86	56	V	87	57	W
88	58	X	89	59	Y	90	5A	Z	91	5B	[	92	5C	\	93	5D	]	94	5E	^	95	5F	_
96	60	`	97	61	a	98	62	b	99	63	c	100	64	d	101	65	e	102	66	f	103	67	g
104	68	h	105	69	i	106	6A	j	107	6B	k	108	6C	l	109	6D	m	110	6E	n	111	6F	o
112	70	p	113	71	q	114	72	r	115	73	s	116	74	t	117	75	u	118	76	v	119	77	w
120	78	x	121	79	y	122	7A	z	123	7B	{	124	7C	|	125	7D	}	126	7E	~	127	7F	□

4.3. Wodurch kann es bei der seriellen Datenübertragung zu einem „Overrun-Error“ kommen? Wird ein solcher Fehler durch den Sender oder den Empfänger erkannt?

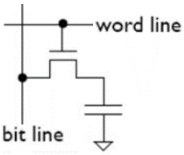
4.4. Wodurch kann es bei der seriellen Datenübertragung zu einem „Framing-Error“ kommen? Nennen Sie zwei Beispiele! Wird dieser Fehler durch den Sender oder den Empfänger erkannt?

4.5. Auf der Arduino-Platine aus dem Praktikum befindet sich ein ATmega328P-Mikrocontroller. Es ist problemlos möglich, Daten über die serielle Schnittstelle des Microcontrollers an einen PC zu senden. Dies funktioniert sogar, obwohl moderne PCs über gar keine „klassischen“ seriellen Schnittstellen mehr verfügen sondern nur über einige USB-Anschlüsse.

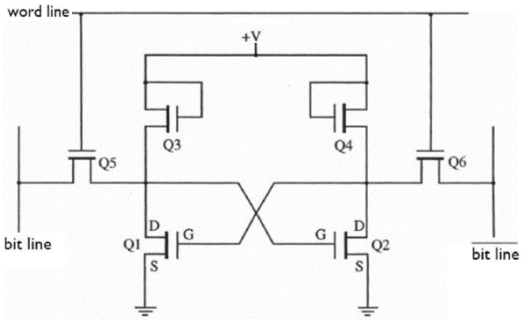
Beschreiben Sie in wenigen Stichworten (oder mit einer passenden Skizze), wie die serielle Datenübertragung vom Mikrocontroller bis zur PC-Applikation funktioniert.

Aufgabe 5 von 5: Speicher (ca. 9 Punkte ≙ 9 Minuten)

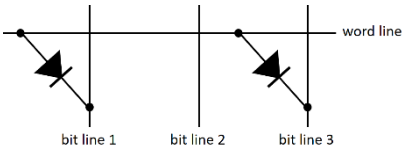
5.1. Betrachten Sie die folgenden Schaltungen von drei unterschiedlichen Speicherzellen. Kreuzen Sie jeweils an, um welchen Typ von Speicher es sich dabei handelt.



❒ EPROM
❒ EEPROM
❒ M-ROM
❒ dynamischer RAM-Speicher
❒ statischer RAM-Speicher



❒ EPROM
❒ EEPROM
❒ M-ROM
❒ dynamischer RAM-Speicher
❒ statischer RAM-Speicher



❒ EPROM
❒ EEPROM
❒ M-ROM
❒ dynamischer RAM-Speicher
❒ statischer RAM-Speicher

5.2. Welche Art von Speicherbausteinen besitzt ein Quarzfenster im Chip-Gehäuse? Wozu dient dieses Fenster?

5.3. Wozu benötigen dynamische RAM-Speicherzellen einen regelmäßigen „Refresh-Vorgang“?

5.4. Was versteht man unter der „modifizierten Harvard-Architektur“ bei einem Mikrocontroller des Typs ATmega328P, der im Praktikum verwendet wurde?

Auszüge aus dem ATmega328P-Datenblatt

Timer/Counter1

20.14.1. TC1 Control Register A (TCCR1A)

Bit	7	6	5	4	3	2	1	0
	COM1A[1:0]		COM1B[1:0]				WGM1[1:0]	
Access	R/W	R/W	R/W	R/W			R/W	R/W
Reset	0	0	0	0			0	0

20.14.2. TC1 Control Register B (TCCR1B)

Bit	7	6	5	4	3	2	1	0
	ICNC1	ICES1		WGM13	WGM12	CS12	CS11	CS10
Access	R/W	R/W		R/W	R/W	R/W	R/W	R/W
Reset	0	0		0	0	0	0	0

Table 20-6. Waveform Generation Mode Bit Description

Mode	WGM13	WGM12 (CTC1)(1)	WGM11 (PWM11)(1)	WGM10 (PWM10)(1)	Timer/ Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	-	-	-
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Table 20-4. Compare Output Mode, Fast PWM

COM1A1/ COM1B1	COM1A0/ COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	WGM1[3:0] = 14 or 15: Toggle OC1A on Compare Match, OC1B disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B disconnected.
1	0	Clear OC1A/OC1B on Compare Match, set OC1A/OC1B at BOTTOM (non-inverting mode)
1	1	Set OC1A/OC1B on Compare Match, clear OC1A/OC1B at BOTTOM (inverting mode)

Table 20-7. Clock Select Bit Description

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

ADC - Analog to Digital Converter

28.9.2. ADC Control and Status Register A (ADCSRA)

Bit	7	6	5	4	3	2	1	0
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit 7 – ADEN: ADC Enable

Writing this bit to one enables the ADC. By writing it to zero, the ADC is turned off. Turning the ADC off while a conversion is in progress, will terminate this conversion.

Bit 6 – ADSC: ADC Start Conversion

In Single Conversion mode, write this bit to one to start each conversion. In Free Running mode, write this bit to one to start the first conversion. The first conversion after ADSC has been written after the ADC has been enabled, or if ADSC is written at the same time as the ADC is enabled, will take 25 ADC clock cycles instead of the normal 13. This first conversion performs initialization of the ADC.

ADSC will read as one as long as a conversion is in progress. When the conversion is complete, it returns to zero. Writing zero to this bit has no effect.

Table 28-5. ADPSn: ADC Prescaler Select [n = 2:0]

ADPS[2:0]	Division Factor
000	2
001	2
010	4
011	8
100	16
101	32
110	64
111	128

By default, the successive approximation circuitry requires an input clock frequency between 50kHz and 200kHz to get maximum resolution. If a lower resolution than 10 bits is needed, the input clock frequency to the ADC can be higher than 200kHz to get a higher sample rate.

The ADC module contains a prescaler, which generates an acceptable ADC clock frequency from any CPU frequency above 100kHz. The prescaling is selected by the ADC Prescaler Select bits in the ADC Control and Status Register A (ADCSRA.ADPS).

28.9.1. ADC Multiplexer Selection Register (ADMUX)

Bit	7	6	5	4	3	2	1	0
	REFS1	REFS0	ADLAR		MUX3	MUX2	MUX1	MUX0
Access	R/W	R/W	R/W		R/W	R/W	R/W	R/W
Reset	0	0	0		0	0	0	0

Table 28-4. Input Channel Selection

MUX[3:0]	Single Ended Input
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	Temperature sensor
1001	Reserved
1010	Reserved
1011	Reserved
1100	Reserved
1101	Reserved
1110	1.1V (V_{BG})
1111	0V (GND)

TCCR0A **Timer/Counter0**

19.9.1. TC0 Control Register A

Bit	7	6	5	4	3	2	1	0
	COM0A1	COM0A0	COM0B1	COM0B0			WGM01	WGM00
Access	R/W	R/W	R/W	R/W			R/W	R/W
Reset	0	0	0	0			0	0

Table 19-9. Waveform Generation Mode Bit Description

Mode	WGM02	WGM01	WGM00	Timer/Counter Mode of Operation	TOP	Update of OCR0x at	TOV Flag Set on
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	-	-	-
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	-	-	-
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

19.9.2. TC0 Control Register B **TCCR0B**

Bit	7	6	5	4	3	2	1	0
	FOC0A	FOC0B			WGM02	CS0[2:0]		
Access	R/W	R/W			R/W	R/W	R/W	R/W
Reset	0	0			0	0	0	0

Table 19-10. Clock Select Bit Description

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

19.9.3. TC0 Interrupt Mask Register **TIMSK0**

Bit	7	6	5	4	3	2	1	0
						OCIEB	OCIEA	TOIE
Access						R/W	R/W	R/W
Reset						0	0	0

Bit 2 – OCIEB: Timer/Counter0, Output Compare B Match Interrupt Enable
When the OCIE0B bit is written to one, and the I-bit in the Status Register is set, the Timer/Counter Compare Match B interrupt is enabled. The corresponding interrupt is executed if a Compare Match in Timer/Counter occurs, i.e., when the OCF0B bit is set in TIFR0.

Bit 1 – OCIEA: Timer/Counter0, Output Compare A Match Interrupt Enable
When the OCIE0A bit is written to one, and the I-bit in the Status Register is set, the Timer/Counter0 Compare Match A interrupt is enabled. The corresponding interrupt is executed if a Compare Match in Timer/Counter0 occurs, i.e., when the OCF0A bit is set in TIFR0.

Bit 0 – TOIE: Timer/Counter0, Overflow Interrupt Enable
When the TOIE0 bit is written to one, and the I-bit in the Status Register is set, the Timer/Counter0 Overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter0 occurs, i.e., when the TOV0 bit is set in TIFR0.