Hochschule München FK03	Komponenten und Programmierung, 90 Minuten		Prof. Dr.-Ing. T. Küpper
Zugelassene Hilfsmittel: alle eigenen, Taschenrechner	**Matr.-Nr.:**	**Name, Vorname:**	
	Hörsaal:	**Unterschrift:**	

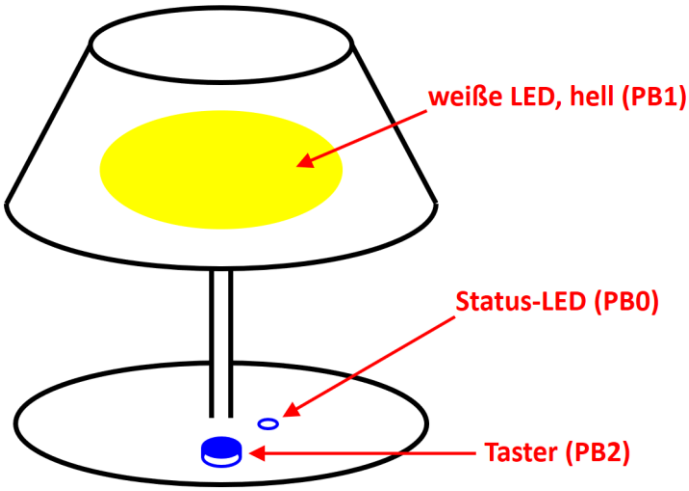
WS 2017/18

Viel Erfolg!!

1	2	3	4	5	6		Σ	N

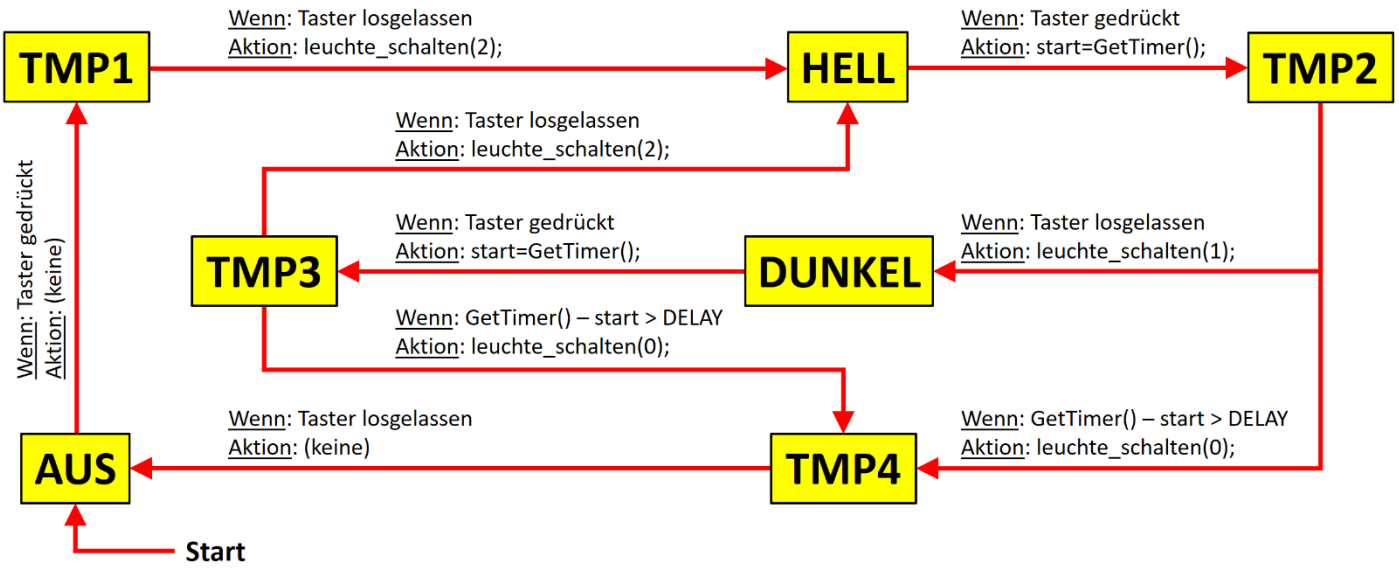
Aufgabe 1: Mikrocontroller-Programmierung (ca. 30 Punkte ≙ 30 Minuten)

Die Steuerung einer Tischlampe soll mit einem Mikrocontroller ATmega328P realisiert werden. An den Mikrocontroller sind die folgenden Komponenten angeschlossen:



- Anschluss PB1: Eine weiße, helle LED (Leuchtdiode). Diese LED dient als Leuchtmittel für die Tischlampe. Die Helligkeit der LED kann mittels PWM eingestellt werden.
- Anschluss PB2: Ein Taster zur Bedienung der Tischlampe. Durch Druck auf den Taster kann die Tischlampe eingeschaltet, durch nochmaligen Druck kann die Tischlampe zwischen „hell“ und „dunkel“ umgeschaltet werden. Zum Ausschalten der Tischlampe muss der Taster für eine längere Zeit gedrückt werden. (Hinweis: Der Taster schaltet wie im Praktikum gegen Masse – der interne Pullup-Widerstand ist aktiviert.)
- Anschluss PB0: An diesem Anschluss ist eine kleine Status-LED angeschlossen. Diese leutet immer dann, wenn die Tischlampe ausgeschaltet ist. (Diese Status-LED ist direkt neben dem Taster angebracht. Sie hilft dem Anwender, bei Dunkelheit den Taster zu finden.)

Das Verhalten der Tischlampe wird durch den folgenden endlichen Automaten beschrieben, der dazugehörige C-Quelltext ist ab Seite 3 abgedruckt:



1.1. Die Interrupt-Funktion ISR(TIMER0_COMPA_vect) wird vom Timer/Counter0 des Mikrocontrollers regelmäßig aufgerufen. Wie oft wird diese Interrupt-Funktion pro Sekunde aufgerufen, nachdem der Timer/Counter0 durch Aufruf der Funktion init_timer0() initialisiert wurde?

1.2. Wie viele Sekunden muss der Taster gedrückt werden, um die Tischleuchte auszuschalten?

1.3. Das PWM-Signal zur Ansteuerung der weißen, hellen LED am Anschluss PB1 wird mittels Timer/Counter1 generiert. Vervollständigen Sie die Funktion init_timer1() so, dass der „Fast PWM Mode“ (siehe Datenblatt, Tabelle 20-6: Mode = 5) mit einem Prescaler von 1:256 aktiviert wird.

1.4. Die Tischleuchte befinde sich im Zustand „DUNKEL“: Wie groß ist die Frequenz und wie groß ist der Tastgrad am Ausgang PB1?

1.5. Vervollständigen Sie die Funktion tastendruck() so, dass der Rückgabewert 1 (eins) beträgt, wenn der Taster gerade gedrückt ist. Der Rückgabewert soll 0 (null) betragen, wenn der Taster gerade nicht gedrückt ist.

1.6. Wozu dient die Verzögerung _delay_ms(100); zu Beginn der while-Schleife im Hauptprogramm (Funktion main)?

1.7. Programmieren Sie den fehlenden Rest des Hauptprogramms (Funktion main, Seiten 4 und 5).

```
// Steuerung für eine Tischleuchte (mit Taster und Status-LED)
#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

// Anschlüsse der Leuchtdioden und des Tasters
#define STATUS_LED PB0
#define LEUCHTE    PB1
#define TASTER     PB2

// Zeitschritte für "langen" Tastendruck
#define DELAY 10000

// Timer0-Zeitschritte seit Programmstart
volatile uint32_t timer0_intern__ = 0;

// Interrupt wird durch Timer0 ausgelöst
ISR(TIMER0_COMPA_vect)
{
    timer0_intern__++;
}

// Timer0-Zeitschritte seit Programmstart abfragen
uint32_t GetTimer(void)
{
    uint32_t result = 0;
    cli(); result = timer0_intern__; sei();
    return result;
}

// Timer0 initialisieren
void init_timer0(void)
{
    TCCR0A = _BV(WGM01);
    TCCR0B = _BV(CS01);
    OCR0A  = 199;

    TIMSK0 = _BV(OCIE0A);   // Compare Match Interrupt einschalten
    sei();                  // Interruptsystem einschalten
}

// Timer1 initialisieren: Fast-PWM am Anschluss OC1A/PB1
void init_timer1(void)
{
    TCCR1A = _BV(COM1A1);   // non-inverting mode, Tab. 20-4









}

// Rückgabe = 1, falls Taster gerade gedrückt ist;
// Rückgabe = 0, falls Taster gerade nicht gedrückt ist
int tastendruck(void)
{









}
```

```
// Leuchte hell (modus = 2), dunkel (modus = 1) oder ausschalten
// (modus = 0); bei ausgeschalteter Leuchte geht die Status-LED an
void leuchte_schalten(int modus)
{
    switch(modus)
    {
    case 2: OCR1A = 255; PORTB &= ~_BV(STATUS_LED); break;
    case 1: OCR1A =  25; PORTB &= ~_BV(STATUS_LED); break;
    case 0: OCR1A =   0; PORTB |=  _BV(STATUS_LED); break;
    }
}

// Ausgänge für Leuchte und Status-LED initialisieren;
// Pullup-Widerstand für Taster einschalten
void init_ports(void)
{
    DDRB = _BV(LEUCHTE) + _BV(STATUS_LED);
    PORTB = _BV(TASTER);
}

// Mögliche Zustände der Lampensteuerung
#define ST_AUS    0
#define ST_TMP1   1
#define ST_AN     2
#define ST_TMP2   3
#define ST_DUNKEL 4
#define ST_TMP3   5
#define ST_TMP4   6

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

    uint8_t state = ST_AUS;
    uint32_t start = 0;
    leuchte_schalten(0);

    while(1)
    {
        _delay_ms(100);  // *** Aufgabe 1.6 ***

        switch(state)
        {
```

Aufgabe 2: Mikrocontroller, Speicher (ca. 10 Punkte ≙ 10 Minuten)

2.1. Wodurch unterscheiden sich Mikroprozessoren und Mikrocontroller? Nennen Sie zwei Unterschiede!

2.2. Wozu dient bei der Arbeit mit Mikrocontrollern der sog. Bootloader?

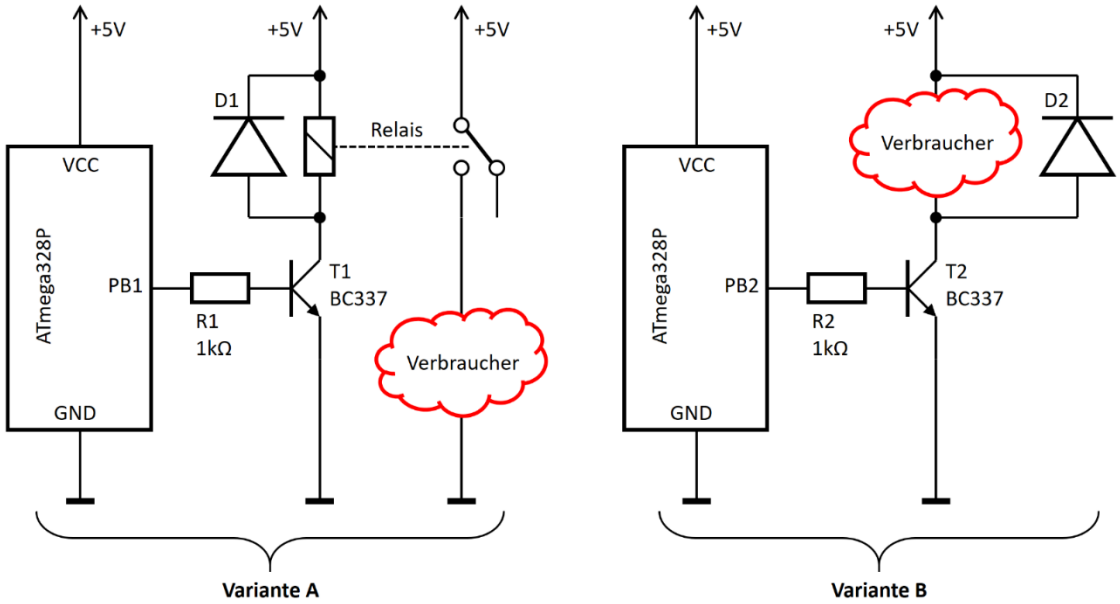
2.3. Beschreiben Sie in Stichworten die Schritte, die bei der Verwendung eines Bootloaders nacheinander ablaufen.

2.4. Zur Erweiterung des RAM-Speichers kann bei Bedarf ein externer SRAM-Baustein an den Mikrocontroller angeschlossen werden. Warum ist dies einfacher zu realisieren, als eine Speichererweiterung mittels DRAM?

2.5. Woher „weiß“ dieser externe SRAM-Baustein (aus Unterpunkt 2.4), ob an einer vom Mikrocontroller genannten Speicheradresse Daten geschrieben oder gelesen werden sollen?

Aufgabe 3: GPIO-Ports (ca. 10 Punkte ≙ 10 Minuten)

Die linke Abbildung zeigt die Ansteuerung eines Verbrauchers durch ein Relais (Variante A), die rechte Abbildung zeigt die Ansteuerung eines Verbrauchers durch einen einzelnen Transistor (Variante B). Für die Basis-Emitter-Spannung gilt bei beiden Transistoren: U_{BE} = 0,5 Volt. Die Spannung an den Anschlüssen PB1 und PB2 beträgt 4,5 Volt (falls diese eingeschaltet sind).



3.1. Welchen Strom I_{PB1} muss der Anschluss PB1 des Mikrocontrollers (Variante A) zum Einschalten des Verbrauchers liefern?

3.2. Welchen Strom I_{PB2} muss der Anschluss PB2 des Mikrocontrollers (Variante B) zum Einschalten des Verbrauchers liefern?

3.3. Nennen Sie jeweils einen Vorteil der Schaltungsvariante A und einen Vorteil der Schaltungsvariante B.

3.4. Unter welcher Bedingung ist es zulässig, bei Variante B auf die Freilaufdiode D2 zu verzichten?

3.5. Welche Schaltung (A oder B) ist nicht geeignet, einen Verbraucher mittels PWM anzusteuern? (Begründung!)

Aufgabe 4: Grundlagen (ca. 10 Punkte ≙ 10 Minuten)

4.1. Was ist der wesentliche Unterschied zwischen Von-Neumann- und Harvard-Rechnern?

4.2. Nennen Sie jeweils einen Vorteil, der Von-Neumann-Architektur bzw. der Harvard-Architektur.

4.3. In Digitalrechnern gibt es verschiedene Busse. Nennen Sie zwei Kriterien, wann ein Leitungsbündel in einem Digitalrechner als „Bus“ bezeichnet wird.

4.4. In einem 8-Bit-Prozessor haben viele Register – und insbesondere die Arbeitsregister – eine Größe von 8 Bit. Nur Befehlszähler und Adressregister sind oft deutlich größer (z. B. 16 oder 20 Bit). Warum ist dies so?

4.5. Wie viele Adressleitungen sind mindestens erforderlich, um 100.000 Speicherzellen eindeutig adressieren zu können?

Aufgabe 5: C-Programmierung für Mikrocontroller (ca. 10 Punkte ≙ 10 Minuten)

5.1. Betrachten Sie nochmals den C-Quelltext aus Aufgabe 1. Warum ist es erforderlich, in der Funktion GetTimer() die beiden Funktionen cli() und sei() aufzurufen?

5.2. Warum ist es in Aufgabe 1 erforderlich, die Variable timer0_intern__ als „volatile“ zu definieren?

5.3. Geben Sie die Ergebnisse der folgenden Operationen im Binärformat (!) an.

```
uint8_t var1 = _BV(3) | _BV(3);
```

var1 =

```
uint8_t var2 = _BV(3) + _BV(3);
```

var2 =

```
uint8_t var3 = 3 | 3;
```

var3 =

```
uint8_t var4 = 3 + 3;
```

var4 =

```
uint8_t var5 = 3 << 3;
```

var5 =

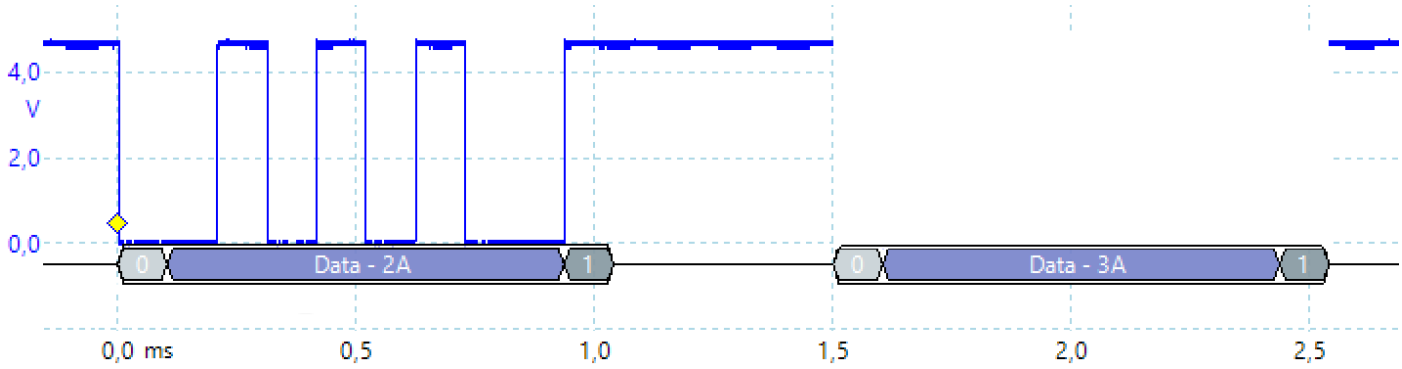
```
uint8_t var6 = _BV(3) << 3;
```

var6 =

Aufgabe 6: Serielle Schnittstelle (ca. 10 Punkte ≙ 10 Minuten)

Über die serielle Schnittstelle eines ATmega328P wird zunächst das Zeichen * (Stern, ASCII-Code = $2A_{HEX}$) übertragen. Anschließend soll das Zeichen : (Doppelpunkt, ASCII-Code = $3A_{HEX}$) gesendet werden. Die Übertragungsparameter sind: 9,6 kbit/s; 8 Datenbits; kein Paritätsbit; 1 Stoppbit.

6.1. Zeichnen Sie den Spannungsverlauf, der sich direkt am Anschluss TXD/PD1 (= Ausgang der seriellen Schnittstelle) beim Senden des zweiten Zeichens (Doppelpunkt, ASCII-Code = $3A_{HEX}$) ergibt in das vorbereitete Diagramm.



(Platz für Notizen..)

6.2. Wie unterscheidet sich der Spannungsverlauf im Vergleich zu Unterpunkt 6.1, wenn die Übertragung über eine externe serielle Schnittstelle nach RS232-Standard erfolgt. (Kurze Beschreibung oder geeignete Skizze angeben!)

Aus dem Datenblatt des Mikrocontrollers ATmega328P

Table 19-9. Waveform Generation Mode Bit Description **Register: TCCR0A**

Mode	WGM02	WGM01	WGM00	Timer/Counter Mode of Operation	TOP	Update of OCR0x at	TOV Flag Set on(1)(2)
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

Note: 1. MAX = 0xFF 2. BOTTOM = 0x00

Table 19-10. Clock Select Bit Description **Register: TCCR0B**

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.

Table 20-4. Compare Output Mode, Fast PWM **Register: TCCR1A**

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 20-6. Waveform Generation Mode Bit Description **Register: TCCR1A**

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 20-7. Clock Select Bit Description **Register: TCCR1B**

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
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.