Aufgabe 1: (14 Punkte)

Bei den Einfachauswahlfragen in dieser Aufgabe ist jeweils nur **eine** richtige Antwort eindeutig anzukreuzen. Auf die richtige Antwort gibt es die angegebene Punktzahl.

Wollen Sie eine Antwort korrigieren, streichen Sie bitte die falsche Antwort mit drei waagrechten Strichen durch (☒) und kreuzen die richtige an.

Lesen Sie die Frage genau, bevor Sie antworten.

a) Welche der folgenden Aussagen über den C-Präprozessor ist richtig? 2 Punkte

- ☐ Nach dem Übersetzen und dem Binden müssen C-Programme durch den Präprozessor nachbearbeitet werden, um Makros aufzulösen.
- ☒ Der Präprozessor ist eine Softwarekomponente, welche Texttransformationen auf dem Quellcode durchführt, die dann von einem C-Compiler übersetzt werden.
- ☐ Der Präprozessor optimiert Makros durch Zeigerarithmetik.
- ☐ Der Präprozessor ist eine Softwarekomponente, welche Java-Klassen durch C-Funktionen ersetzt, die dann von einem C-Compiler übersetzt werden.

b) Gegeben sei folgende Enumeration: 2 Punkte

```
enum LANGUAGE {Java, Python, C};
```

Welche Aussage ist richtig:

- ☐ Der Wert von `C` ist unbekannt; der Compiler weist zur Übersetzungszeit jedem `enum`-Element einen zufälligen, aber eindeutigen Wert zu.
- ☐ Der Wert von `C` ist `3`.
- ☐ Der Compiler meldet einen Fehler, weil den `enum`-Elementen kein Wert zugewiesen wurde.
- ☒ Der Wert von `C` ist `2`.

c) Folgende Makrodefinition findet sich in der AVR-Bibliothek: 2 Punkte

```
#define PINA (*(volatile uint8_t *)0x39)
```

Welche der folgenden Aussagen bezüglich der Verwendung des `volatile`-Schlüsselworts ist in diesem Fall richtig?

- ☐ Das Schlüsselwort `volatile` erlaubt dem Compiler bessere Optimierungen durchzuführen.
- ☒ Wird der Port A als Eingang konfiguriert, könnte sich der Wert von `PINA` jederzeit ändern. Durch volatile wird der Compiler angewiesen, stets den aktuellen Wert aus `PINA` zu lesen.
- ☐ Das `volatile`-Schlüsselwort ermöglicht den sicheren Zugriff auf einzelne Bits des Registers.
- ☐ Das `volatile`-Schlüsselwort stellt hier sicher, dass der Zugriff auf `PINA` mit Interrupts synchronisiert wird.

d) Was versteht man unter Polling? 2 Punkte

- ☐ Ein Konzept zur Abarbeitung von Interrupts.
- ☒ Wenn ein Programm regelmäßig eine Peripherie-Schnittstelle abfragt, ob Daten oder Zustandsänderungen vorliegen.
- ☐ Das regelmäßige Anheben eines Pegels, um einem Gerät einen bestimmten Zustand zu signalisieren.
- ☐ Wenn ein Programm zum Zugriff auf kritische Daten Interrupts sperrt.

e) Gegeben seien folgende zwei zu übersetzende C-Quelldateien: 2 Punkte

```
// foo.c:
uint8_t a = 42;
// main.c:
extern uint32_t a;
```

Welche Aussage zum Thema Übersetzer ist richtig:

- ☐ Beim Übersetzen werden insgesamt 5 Byte (1 Byte + 4 Byte) Speicherplatz für `a` reserviert.
- ☒ Beim Übersetzen werden insgesamt 1 Byte Speicherplatz für `a` reserviert.
- ☐ Beim Übersetzen werden insgesamt 4 Byte Speicherplatz für `a` reserviert.
- ☐ Der Übersetzer stellt in jedem Fall einen Typfehler fest und beendet sich mit einem Fehler.

f) Wie unterscheiden sich die wie folgt in einer C-Datei global definierten Variablen? 2 Punkte

```
int a;
static int b;
```

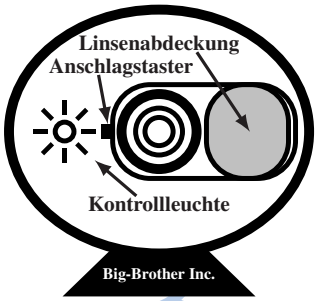
- ☐ Die Variable `b` ist nur zugreifbar aus Funktionen, die ebenfalls mit dem Schlüsselwort `static` deklariert wurden.
- ☐ Die Variable `a` ist nur für Funktionen in derselben Datei zugreifbar, während auf Variable `b` auch von Funktionen in anderen Modulen des Programms zugegriffen werden kann.
- ☐ Variable `b` kann nur einmalig zugewiesen werden und bleibt danach konstant.
- ☒ Die Variable `b` ist nur für Funktionen in derselben Datei zugreifbar, während auf Variable `a` auch von Funktionen in anderen Modulen des Programms zugegriffen werden kann.

g) Welche Aussage zu *Signalen* ist richtig? 2 Punkte

- ☐ Bei Signalen können keine Nebenläufigkeitsprobleme auftreten.
- ☐ Verschiedene Signale können sich nie gegenseitig unterbrechen.
- ☒ Signale können mittels `sigprocmask` während der Ausführung von kritischen Bereichen blockiert werden.
- ☐ Das Standardverhalten bei dem Empfang eines Signals ist der Abbruch des Prozesses.

Aufgabe 2: Webcam (0 Punkte)

Sie dürfen diese Seite zur besseren Übersicht bei der Programmierung heraustrennen!



Implementieren Sie die Steuerung einer Aktivitätsanzeige innerhalb einer Webcam, die die Privatsphäre der Nutzer schützen soll. Um gegen ungewollte Spionage vorzugehen, verfügt diese über eine Linsenabdeckung, welche bei Bedarf vor die Linse geschoben werden kann. Im Inneren der Webcam befindet sich ein Anschlagstaster, der jedes Öffnen/Schließen der Abdeckung registriert. Beim Öffnen der Abdeckung wird nun der Anschlagstaster losgelöst und die Webcam ist aktiv. Während der Aufzeichnung leuchtet nun auch eine rote Kontrollleuchte in Zeitintervallen von etwa einer Sekunde. Um die Bildqualität zu steigern, soll ebenfalls die Umgebungshelligkeit gemessen, verarbeitet und der eigentlichen Kamerasteuerung mitgeteilt werden. Wird die Abdeckung geschlossen und somit der Taster betätigt, soll die Kontrollleuchte ausgeschaltet werden und die Umgebungshelligkeit nicht mehr abgefragt werden.

Im Detail soll Ihr Programm wie folgt funktionieren:

- Initialisieren Sie die Hardware in der Funktion **`void`** `init(`**`void`**`)`. Treffen Sie hierbei keine Annahmen über den initialen Zustand der Hardware-Register.
- Der Eingang `PD2` (Interrupt 0) ist mit dem Anschlagstaster verbunden. Eine steigende Flanke tritt genau dann auf, wenn die Linsenabdeckung geöffnet wird und eine fallende dann, wenn diese geschlossen wird. Treffen Sie hier ebenfalls keine Annahmen über die initiale Position der Linsenabdeckung.
- Für die Zeittaktung soll ein 8-Bit Timer verwendet werden. Konfigurieren Sie diesen so, dass er alle Millisekunde einen Interrupt auslöst. Das Hauptprogramm soll dabei alle `500ms` (Siehe Makro `ACTIVATION_INTERVAL`) über das Verstreichen des geforderten Zeitintervalls informiert werden.
- Für die Zeittaktung soll ein 8-Bit Timer verwendet werden. Konfigurieren Sie diesen so, dass er alle Millisekunde einen Interrupt auslöst. Das Hauptprogramm soll dabei alle `500ms` (Siehe Makro `ACTIVATION_INTERVAL`) über das Verstreichen des geforderten Zeitintervalls informiert werden.
- Während der Aufzeichnung (= Linsenabdeckung wurde geöffnet), soll jede `500ms` der Zustand der Kontrollleuchte am Ausgang `PB0` getoggelt werden.
- In dem selben Zeitintervall soll ebenfalls die Umgebungshelligkeit ermittelt werden. Lesen Sie dafür zunächst durch Aufruf der Funktion **`int16_t`** `sb_adc_read(ADCDEV dev)` für das Geräte `PHOTO` den Wert h des Fototransistors als vorzeichenlose 10 Bit Ganzzahl. Während des Aufrufs müssen die Interrupts gesperrt sein.
- Bestimmen Sie den (approximierten) dekadischen Logarithmus des gemessenen Wertes h und übergeben Sie das Ergebnis an die externe Funktion **`void`** `set_iso(`**`uint16_t`** `value)`, welche die eigentliche Ansteuerung der Kamera kapselt.
- Implementieren Sie die Hilfsfunktion **`uint16_t`** `log10(`**`uint16_t`** `value)`, welche folgende ressourcenschonende Approximation umsetzt:

$$\log_{10} h = \frac{\log_2 h}{\log_2 10} \approx \left\lfloor \frac{\lfloor \log_2 h \rfloor}{3} \right\rfloor \tag{1}$$

  Hierbei entspricht der abgerundete binäre Logarithmus $\lfloor \log_2 h \rfloor$ der Position des höchstwertigen gesetzten Bits in h. Beispiel: $\lfloor \log_2 0b100 \rfloor = 2$. (Sonderfall: $\lfloor \log_2 0b0 \rfloor = 0$).

- Benutzen Sie **keine** Gleitkommazahlen oder sonstige Mathematikbibliotheksfunktionen.
- Stellen Sie sicher, dass sich der Mikrocontroller möglichst oft im Schlafmodus befindet.

Information über die Hardware

Sie dürfen diese Seite zur besseren Übersicht bei der Programmierung heraustrennen!

Anschlagstaster: Interruptleitung an **PORTD**, Pin 2

- Steigende Flanke: Linsenabdeckung wird geöffnet, Aufzeichnung beginnt
- Fallende Flanke: Linsenabdeckung wird geschlossen, Aufzeichnung endet
- Pin als Eingang konfigurieren: Entsprechendes Bit im **DDRD**-Register auf 0
- Internen Pull-Up-Widerstand aktivieren: Entsprechendes Bit im **PORTD**-Register auf 1
- Externe Interruptquelle **INT0**, ISR-Vektor-Makro: **INT0_vect**
- Aktivieren/Deaktivieren der Interruptquelle erfolgt durch Setzen/Löschen des **INT0**-Bits im Register **EIMSK**

Konfiguration der externen Interruptquelle **INT0** (Bits im Register **EICRA**)

Interrupt 0		Beschreibung
ISC01	ISC00	
0	0	Interrupt bei low Pegel
0	1	Interrupt bei beliebiger Flanke
1	0	Interrupt bei fallender Flanke
1	1	Interrupt bei steigender Flanke

Kontrollleuchte für Aufzeichnung: Ausgang an **PORTB**, Pin 0

- Signalisiert, ob gerade Aufzeichnung aktiv ist. Zum Leuchten der LED auf LOW setzen, sonst auf HIGH
- Pin als Ausgang konfigurieren: Entsprechendes Bit im **DDRB**-Register auf 1
- Kontrollleuchte initial ausschalten, entsprechendes Bit im **PORTB**-Register auf 1

Zeitgeber (8-bit): **TIMER0**

- Es soll die Überlaufunterbrechung verwendet werden (ISR-Vektor-Makro: **TIMER0_OVF_vect**)
- Der ressourcenschonendste Vorteiler (*prescaler*) ist 64, wodurch es bei dem 16 MHz CPU-Takt (hinreichend genau) alle $1ms$ zum Überlauf des 8-bit-Zählers **TCNT0** kommt.
- Aktivieren/Deaktivieren der Interruptquelle erfolgt durch Setzen/Löschen des **TOIE0**-Bits im Register **TIMSK0**

Konfiguration der Frequenz des Zeitgebers **TIMER0** (Bits im Register **TCCR0B**)

CS02	CS01	CS00	Beschreibung
0	0	0	Timer aus
0	0	1	CPU-Takt
0	1	0	CPU-Takt / 8
0	1	1	CPU-Takt / 64
1	0	0	CPU-Takt / 256
1	0	1	CPU-Takt / 1024
1	1	0	Ext. Takt (fallende Flanke)
1	1	1	Ext. Takt (steigende Flanke)

Ergänzen Sie das folgende Codegerüst so, dass ein vollständig übersetzbares Programm entsteht.

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <stdint.h>
#include <adc.h>

#define ACTIVATION_INTERVAL 500
#define LD_10 3

extern void set_iso(uint16_t value);

// Function Declarations, Global Variables, etc.
static void init(void);
static uint16_t log10(uint16_t value);

// Ignoriere ersten Durchlauf der Warteschleife
// Register in der main Funktion aus (siehe unten).
static volatile uint8_t event_shutter = 1;
static volatile uint8_t event_timer = 0;
// End Function Declarations, Global Variables, etc.

// Interrupt Service Routines
ISR(TIMER0_OVF_vect) {
  static uint16_t counter = 0;
  counter++;
  if (counter == ACTIVATION_INTERVAL) {
    counter = 0;
    event_timer = 1;
  }
}
ISR(INT0_vect) {
  event_shutter = 1;
}
// End Interrupt Service Routines
```

0

0

D: 0

```
// Function main
void main(void) {
// Initialization and Local Variables
  init();

  // Hier ist es ebenfalls ok, den aktuellen Wert des Pins
  // abzufragen und sich dann schlafen zu legen (siehe oben).
  uint8_t active = 0;



// Event Loop
  while (1) {
    cli();
    while (!event_timer && !event_shutter) {
      sleep_enable();
      sei(); // sleep_cpu() muss direkt nach sei() folgen
      sleep_cpu();
      sleep_disable(); // Optional
      cli();
    }
    sei();
```

0

0

```
// Process Events
    if (event_shutter) { // Shutter *vor* Timer
      event_shutter = 0;

      active = (PIND & (1 << PD2)) != 0;
      if (!active) {
        PORTB |= (1 << PB0);
      }
    }


    if (event_timer) {
      event_timer = 0;

      if (active) {
        PORTB ^= (1 << PB0);

        cli();
        int16_t sensor = sb_adc_read(PHOTO);
        sei();

        set_iso(log10(sensor));
      }
    }
  }
}

// End main
```

0

0

M: 0

```
// Logarithm to Base 10
static uint16_t log10(uint16_t value) {
  uint8_t log2 = 0;
  for (uint8_t i = 0; i < 16; i++) {
    if (value & (1 << i)) {
      log2 = i;
    }
  }

  return log2 / LD_10;
}

// End Logarithm to Base 10
```

0

L: 0

```
// Initialization Routine
static void init(void) {
  // Konfiguriere Prescaler (64)
  TCCR0B &= ~(1 << CS02);
  TCCR0B |= (1 << CS01) | (1 << CS00);

  // Aktiviere Timerinterrupts
  TIMSK0 |= (1 << TOIE0);

  // Konfiguriere PB0 (RED0) als Ausgang
  DDRB |= (1 << PB0);
  PORTB |= (1 << PB0);

  // Konfiguriere PD2 (Button0) als Eingang
  DDRD &= ~(1 << PD2);
  PORTD |= (1 << PD2);

  // Konfiguriere Interrupt für PD2 bei beliebiger Flanke
  EICRA |= (1 << ISC00);
  EICRA &= ~(1 << ISC01);

  // Aktive Button0 Interrupt
  EIMSK |= (1 << INT0);
}
// End Initialization Routine
```

0

I: 0

Aufgabe 3: alt (0 Punkte)

Personen, die an der FAU eine Lehrveranstaltung betreuen, erhalten üblicherweise viele hilfesuchende Emails von Studierenden. Da die Veranstaltungen in hohem Tempo voranschreiten, sind solche Mails häufig nach ein paar Tagen veraltet und eine Antwort ist nicht mehr nötig. Helfen Sie arbeitsscheuen Betreuenden, indem sie ein Programm `alt` schreiben, das in einem gegebenen Verzeichnis alles Dateien löscht, die älter als eine Woche sind.

```
$> ./alt /home/tutor/Maildir/spic
```

Das Programm soll im Detail wie folgt funktionieren:

- Das Programm prüft zu Beginn, ob genau ein Parameter übergeben wurden. Sollte dies nicht der Fall sein, gibt es eine entsprechende Fehlermeldung aus und beendet sich.
- Wurde das Programm korrekt aufgerufen, wird mittels `opendir()` und `readdir()` das übergebene Verzeichnis (nicht rekursiv) durchsucht.
- Mittels `stat()` soll für jede gefundene reguläre Datei die Modifikationszeit (`st_mtime`) abgefragt werden.
- Ist die Datei älter als 7 Tage, soll sie mittels `unlink()` entfernt werden.
- Nach erfolgreicher Ausführung endet das Programm mit dem Statuscode `EXIT_SUCCESS`.
- Bei Fehlern soll eine Fehlermeldung ausgegeben werden und das Programm mit einem Fehlerstatus (`EXIT_FAILURE`) beendet werden.
- Achten Sie darauf allokierte Ressourcen wieder freizugeben (`free()`, `closedir()`, ...).
- Implementieren Sie zudem die Hilfsfunktion `time_t calc_time_limit(void)`, welche den Zeitstempel für heute vor 7 Tagen ermittelt:
  - Zeit wird in UNIX-artigen Systemen in Sekunden seit dem 01.01.1970 gezählt.
  - Mittels `time(NULL)` können Sie sich den Zeitstempel für die aktuelle Sekunde zurückgeben lassen.
  - Gehen Sie von Standardtagen (24 Stunden pro Tag, 60 Minuten pro Stunde, 60 Sekunden pro Minute) aus.
  - Rufen Sie `calc_time_limit` nur einmal während eines Programmdurchlaufs auf, um eine konsistente Vergleichsbasis zu haben.

Achten Sie auf eine korrekte Fehlerbehandlung der verwendeten Funktionen. Fehlermeldungen sollen generell auf `stderr` erfolgen. Zur kompakten Fehlerbehandlung können die vorgegebenen Funktionen `die()` (`errno` gesetzt) und `err()` (`errno` nicht gesetzt) genutzt werden.

Ergänzen Sie das folgende Codegerüst so, dass ein vollständig übersetzbares Programm entsteht.

```
#include <sys/stat.h>
#include <sys/types.h>
#include <dirent.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>

static void die(const char *s) {
  perror(s);
  exit(EXIT_FAILURE);
}

static void err(const char *s) {
  fputs(s, stderr);
  exit(EXIT_FAILURE);
}

// Function calc_time_limit
static time_t calc_time_limit(void) {
  time_t now = time(NULL);
  time_t seven_days = 7*24*60*60;
  return now-seven_days;
}
```

C: 0

```c
// Function main
int main(int argc, char *argv[])
{
  // Überprüfe Argumente + Fehlerbehandlung
  if (argc != 2) {
    err("usage");
  }

  char *dirpath = argv[1];

  //Öffne Verzeichnisstrom
  DIR *dirp = opendir(dirpath);
  if (!dirp) {
    die("opendir");
  }

  time_t limit = calc_time_limit();

  errno = 0;
  struct dirent *de = readdir(dirp);
  while (de != NULL) {
    // Länge inklusive '/' und '\0'
    size_t len = strlen(dirpath) + strlen(de->d_name) + 2;
    char *filename = malloc(len);
    if (filename == NULL) {
      die("malloc");
    }

    // Setze Dateipfad zusammen
    strcpy(filename,dirpath);
    strcat(filename,"/");
    strcat(filename,de->d_name);
```

```
    // Check Timestamp
    struct stat sb;
    if (stat(filename, &sb) < 0) { // Aufruf + Bedingung
      die("stat");
    }


    if ((sb.st_mode & S_IFREG) && (sb.st_mtime < limit)) {
      if (unlink(filename) < 0) {
        die("unlink");
      }
    }

    free(filename);

    errno = 0;
    de = readdir(dirp);
  }

  if (errno != 0) {
    die("readdir");
  }

  closedir(dirp);

  return EXIT_SUCCESS;

}
// End main
```

L: 0

Aufgabe 4: Speicherorganisation (11 Punkte)

Hinweis: Lesen Sie zuerst die Aufgabenstellung – ein vollständiges Verständnis des Programms ist zur Bearbeitung der Aufgabe nicht notwendig.

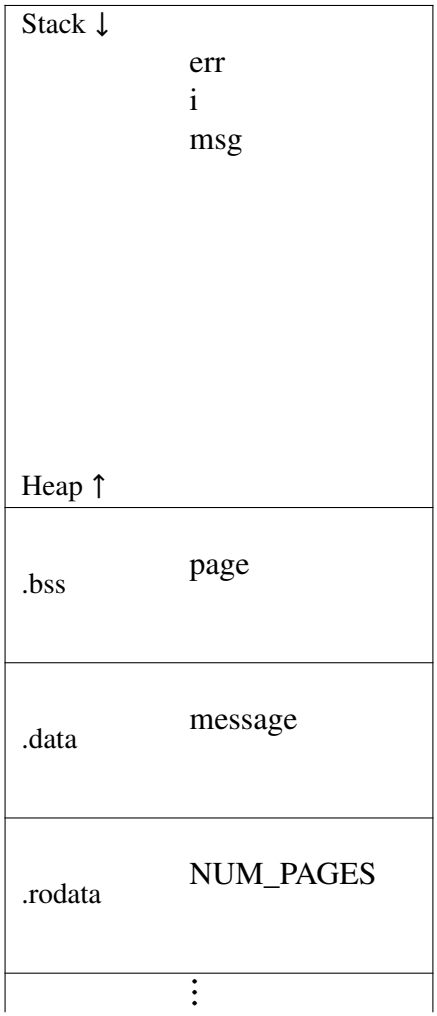
```
#include <stdint.h>
#include <display.h>
#include <timer.h>
#include <avr/interrupt.h>

static const uint8_t NUM_PAGES = 8;
static char message[] = "SPiC Exam!";

static void show_message(char msg[]) {
    static int page = 0;
    sb_display_showStringWide(page, 0, msg);
    page = (page + 1) % NUM_PAGES;
}

void main(void) {
    sei();
    int8_t err = sb_display_enable();
    if (err) {
        while(1) { /* ... */ }
    }
    for (uint8_t i = 0; i < NUM_PAGES; i++) {
        show_message(message);
        sb_timer_delay(5000);
    }
    while(1);
}
```

Speicheraufbau (vereinfacht):



a) Vervollständigen Sie den (vereinfachten) Speicheraufbau: 5 Punkte

1) Ergänzen Sie *Stack* und *Heap* sowie deren Wachstumsrichtung in der Abbildung. (2 Punkte)

2) Ordnen Sie alle im Quelltext vorkommenden Variablen den dargestellten Speichersektionen zu. (3 Punkte)

Korrekturhinweise: Reihenfolge der Variablen auf dem Stack wird nicht bewertet.

b) Wie muss der RAM-Speicher beim Start des Mikrocontrollers vorbereitet werden, ehe die eigentliche Programmausführung in `main()` beginnen kann? (2 Punkte)

- Kopieren von .data aus dem nichtflüchtigen Speicher (Flashspeicher/ROM) in den RAM

- Nullen von .bss

c) Die folgende Tabelle enthält vier Zeiger-Datentypen. Geben Sie für alle Datentypen an, ob der dereferenzierte Wert und der Zeiger veränderlich sind (jeweils **ja** oder **nein**). Sollte ein Datentyp so in C nicht möglich sein, kreuzen Sie bitte die Spalte **Syntaxfehler** an. (4 Punkte)

	Wert veränderlich	**Zeiger veränderlich**	**Syntaxfehler**
`uint8_t *`	ja	ja	
`const uint8_t *`	nein	ja	
`uint8_t * const`	ja	nein	
`const uint8_t * const`	nein	nein	

Korrekturhinweise: Keine Punkte wenn jemand Syntaxfehler ankreuzt und weitere Spalten ausfüllt

Aufgabe 5: Nebenläufigkeit (3 Punkte)

Die folgenden Beschreibungen sollen kurz und prägnant erfolgen (Stichworte, kurze Sätze).

a) Welche Schritte **in Hardware** umfasst typischerweise die Abarbeitung eines Interrupts auf einem Mikrokontroller? (3 Punkte)

1. Gerät signalisiert Interrupt,
   Programm wird unterbrochen
2. Weitere Interrupts werden gesperrt
3. Registerinhalte werden gesichert
4. Aufzurufende ISR wird ermittelt
5. ISR wird ausgeführt
6. ISR terminiert, Registerinhalte werden wiederhergestellt,
   Programm wird fortgesetzt

b) Die Verwendung von Interrupts kann zu verschiedenen Nebenläufigkeitsproblemen führen. Nennen und beschreiben Sie exemplarisch **zwei** solcher Probleme. Beschreiben Sie zusätzlich wie man mit ihnen umgehen kann, um die korrekte Funktionsweise von Programmen zu gewährleisten. (0 Punkte)

Lost-update:
Wert steht bereits in Register, ISR
veraendert Wert und restauriert bei return Register

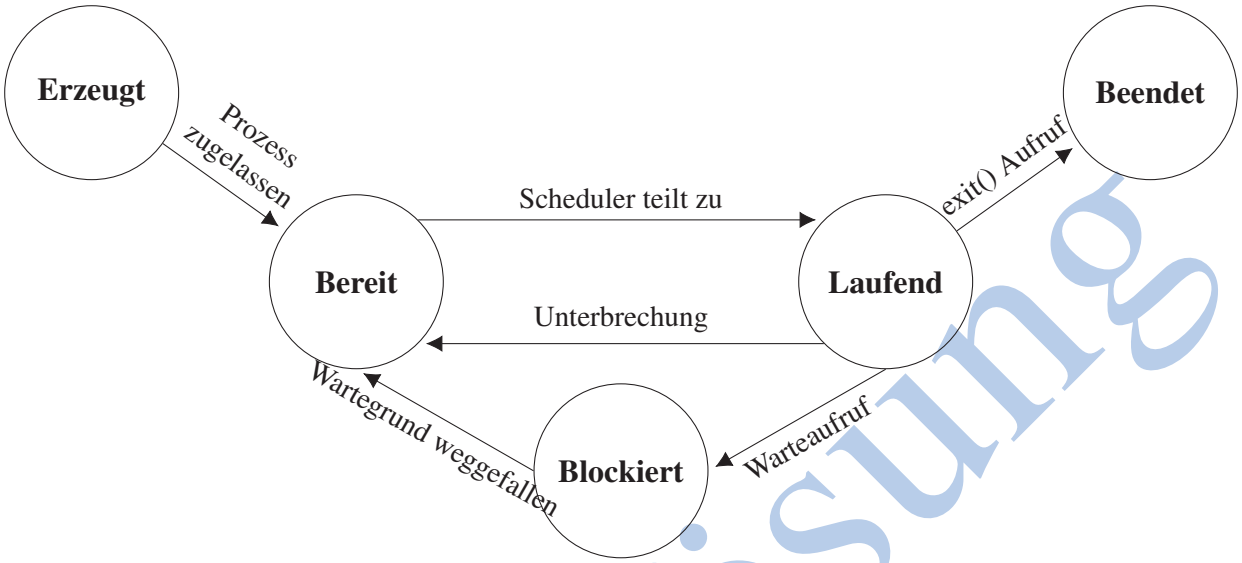
Read-Write:
Wert übersteigt Registerbreite und wird aktualisiert.
Unterbrechungen an dieser Stelle koennen zu Inkonsistenzen
führen (z.B. bei Operationen mit Überlauf)

Alternativ: Lost wakeup:
Interrupt zwischen Prüfen einer Bedingung und sleep

Lösung: Interrupts kurzzeitig blockieren

Aufgabe 6: Prozesse (0 Punkte)

a) Vervollständigen Sie den folgenden Graphen zu den Prozesszuständen unter Linux mit den entsprechenden Zustandsübergängen. Jeder Knoten entspricht einem Prozesszustand und jede Kante entspricht einem Zustandsübergang. Achten Sie darauf **alle** noch nicht beschrifteten Knoten und Kanten zu beschriften. (0 Punkte)



Korrekturhinweise:
Die Antworten für die Zustandsübergänge müssen nicht genau der Musterlösung entsprechen. Alle inhaltlich sinnvollen Antworten sind in Ordnung.

b) Nennen Sie jeweils ein charakteristisches Problem bei der Verwendung eines **Spin Locks** und eines **Sleeping Locks**.
(0 Punkte)

Spin Lock: Bei sehr langen kritischen Abschnitten wird übermäßig viel Rechenzeit vergeudet

Alternativ: In Uniprozessorsystemen wird Rechenzeit vergeudet, bis durch den Scheduler eine Umschaltung erfolgt

Sleeping Lock: Bei kurzen kritischen Abschnitten ist das Blockieren/Aufwegen und die Umschaltung unverhältnismäßig teuer