

Friedrich-Alexander-Universität Erlangen-Nürnberg

**Written exam for the module
„System-Level Programming (SLP)“
5 ECTS**

	achievable points	scored points					
Problem 1	14						
Problem 2	30	D	M	L	I		
		7	13	4	6		
Problem 3	20	C	L				
		3	17				
Problem 4	11	a	b	c			
		5	2	4			
Problem 5	7	a	b				
		3	4				
Problem 6	8	a	b				
		6	2				
Total	90						

(Last Name)

(First Name)

(Matriculation No.)

(Degree Program)

(Semester)

With my signature I confirm that

- I received the complete exam (19 pages including the cover page),
- I received the man pages (two sheets of paper with 5 man pages: opendir/readdir/closedir, stat, string, time und unlink),
- I have read the information on page 2.

Erlangen, July 28, 2023

(Signature)

Information

Please read the following information carefully and sign the declaration on the first page.

- You have 90 minutes to complete all problems.
- **No** auxiliary materials are permitted.
- The solution to a problem should be written on the exam sheet in the space provided. Please note that the space provided is generous and does not necessarily correspond to the expected answer length. If there is not enough space, you are allowed use the backside of the exam sheets. In this case, please mark the solution to a problem as such. If necessary, additional solution sheets (white) can be handed out. Before using them, be sure to write your name and your matriculation number on it!
- The solutions must be written in document-proof blue or black. Anything recognized as incorrect must be clearly crossed out. Ink eraser or Tippex must not be used. Do not use a pencil!
- Scratch paper (colored sheets) must **not** be submitted. If required, additional scratch paper (colored) is available from the supervisor.
- Content-related questions will not be answered.
- Write your first and second name, matriculation **number**, degree program, and number of semesters on the cover sheet of the exam.
- Please have your student ID and a photo ID ready for checking.
- You may not leave the room until your personal details have been checked and you have returned the exam papers to the exam supervisor.
- You cannot leave the room during the last 15 minutes of the exam. Stay seated, until all exam papers have been collected and the exam supervisor gives the signal to leave.

The **exam results** will be published in about a week on the course website under the item „Exam“. Alternatively, the results can be viewed using the WAFFEL-System (<https://waffel.informatik.uni-erlangen.de> - Access requires your login and matriculation number or password).

Problem 1: (14 Points)

For the single-choice questions in this problem, only one correct answer must be clearly marked with a cross. The correct answer is awarded the indicated number of points.

If you want to correct an answer, please cross out the incorrect answer with three horizontal lines (~~☒~~) and mark the correct answer with a cross.

Read the question carefully before you answer.

a) Which of the following statements about the C preprocessor is correct?

2 Points

- ☐ After compiling and binding, C programs must be post-processed by the pre-processor in order to resolve macros.
- ☒ The preprocessor is a software component that performs text transformations on the source code, which are then translated by a C compiler.
- ☐ The preprocessor optimizes macros using pointer arithmetic.
- ☐ The preprocessor is a software component that replaces Java classes with C functions, which are then translated by a C compiler.

b) The following enumeration is given:

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

2 Points

Which of the following statements is correct?

- ☐ The value of C is unknown; the compiler assigns a random but unique value to each enum element at compile time.
- ☐ The value of C is 3.
- ☐ The compiler reports an error because no value has been assigned to the enum elements.
- ☒ The value of C is 2.

c) The following macro definition can be found in the AVR library:

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

2 Points

Which of the following statements regarding the use of the `volatile` keyword is correct in this case?

- ☐ The keyword `volatile` allows the compiler to perform better optimizations.
- ☒ If port A is configured as an input, the value of `PINA` could change at any time. `volatile` instructs the compiler to always read the current value from `PINA`.
- ☐ The `volatile` keyword enables safe access to individual bits of the register.
- ☐ The `volatile` keyword ensures that access to `PINA` is synchronized with interrupts.

d) Which statement on the term „polling“ is correct?

2 Points

- ☐ A concept for processing interrupts.
- ☒ When a program periodically queries a peripheral device to determine whether data or status changes are pending.
- ☐ The periodic raising of a voltage level to signal a certain status to a device.
- ☐ When a program masks interrupts to access critical data.

e) Given are the following two C source files to be compiled:

2 Points

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

Which statement on the subject of compiling is correct?

- ☐ When compiling, a total of 5 bytes (1 byte + 4 bytes) of memory are reserved for a.
- ☒ When compiling, a single 1 byte of memory are reserved for a.
- ☐ When compiling, a total of 4 bytes of memory are reserved for a.
- ☐ In any case, the compiler detects a type error and exits with an error.

f) What is the difference between the following global variables defined in a C file?

2 Points

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

- ☐ The variable b can only be accessed from functions that have also been declared with the keyword `static`.
- ☐ The variable a can only be accessed by functions in the same file, while the variable b can also be accessed by functions in other modules of the program.
- ☐ Variable b can only be assigned once and then remains constant.
- ☒ The variable b can only be accessed by functions in the same file, while the variable a can also be accessed by functions in other modules of the program.

g) Which statement about *signals* is correct?

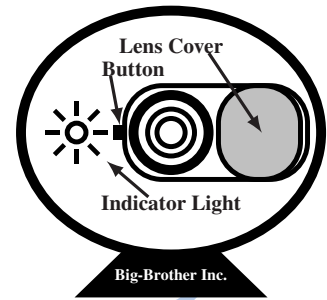
2 Points

- ☐ No concurrency problems can occur with signals.
- ☐ Different signals can never interrupt each other.
- ☒ Signals can be blocked using `sigprocmask` during the execution of critical areas.
- ☐ The default behavior when receiving a signal is to abort the process.

Problem 2: Webcam (30 Points)

You may detach this page for a better overview during programming!

Implement the control of an activity indicator within a webcam to protect the privacy of its users. To prevent unwanted spying, it has a lens cover that can be pushed in front of the lens if wanted. Inside the webcam is a stop button that registers every opening/closing of the cover. When the cover is opened, the button is released and the webcam is active. During recording, a red indicator light now also lights up at intervals of about one second. To increase the image quality, the ambient brightness should also be measured, processed, and communicated to the actual camera controller. When the cover is closed and, thus, the button is pressed, the indicator light should be switched off and the ambient brightness should no longer be queried.



In detail, your program should work as follows:

- Initialize the hardware in the **void** `init(void)` function. Do not make any assumptions about the initial state of the hardware registers.
- The input PD2 (interrupt 0) is connected to the button. A rising edge occurs when the lens cover is opened and a falling edge occurs when it is closed. Do not make any assumptions about the initial position of the lens cover.
- An 8-bit timer should be used for timing. Configure the timer so that it triggers an interrupt every millisecond. The main program should be informed every 500ms (see macro `ACTIVATION_INTERVAL`) when the required time interval has elapsed.
- An 8-bit timer should be used for the timing. Configure this in such a way so that it triggers an interrupt every millisecond. The main program should trigger an interrupt every 500ms (see macro `ACTIVATION_INTERVAL`) when the required time interval has elapsed.
- During recording (= lens cover opened), the state of the indicator light at output PB0 should be toggled every 500ms.
- The ambient brightness should also be determined at the same time interval. To do this, first call the function **int16_t** `sb_adc_read(ADCDEV dev)` for the PHOTO device to read the value h of the phototransistor as an unsigned 10-bit integer. Interrupts must be disabled during the function invocation.
- Determine the (approximated) decadic logarithm of the measured value h and pass the result to the external function **void** `set_iso(uint16_t value)`, which encapsulates the actual control of the camera.
- Implement the auxiliary function **uint16_t** `log10(uint16_t value)`, which implements the following resource-aware approximation:

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

The rounded binary logarithm $\lfloor \log_2 h \rfloor$ corresponds to the position of the most significant set bit in h . Example: $\lfloor \log_2 0b100 \rfloor = 2$. (Special case: $\lfloor \log_2 0b0 \rfloor = 0$).

- Do not use **any** floating point numbers or other math library functions.
- Make sure that the microcontroller enters a sleep mode as often as possible.

Information about the hardware

You may detach this page for a better overview during programming!

Button: interrupt line connected to **PORTD**, pin 2

- Rising edge: lens cover is opened, recording begins
- Falling edge: lens cover is closed, recording ends
- Configure pin as input: set the corresponding bit in the **DDRD** register to 0
- Activate internal pull-up resistor: set the corresponding bit in the **PORTD** register to 1
- External interrupt source **INT0**, ISR vector macro: **INT0_vect**
- Masking/Unmasking the interrupt is done by setting/clearing the **INT0** bit in the **EIMSK** register

Configuration of the external interrupt source **INT0** (bits in register **EICRA**)

interrupt 0		description
ISC01	ISC00	
0	0	interrupt at low level
0	1	interrupt at either edge
1	0	interrupt on falling edge
1	1	interrupt on rising edge

Indicator light for recording: output at **PORTB**, pin 0

- Signal active recording: set to LOW to switch on the LED, otherwise set to HIGH
- Configure pin as output: set the corresponding bit in the **DDRB** register to 1
- Initially switch off indicator light: set corresponding bit in **PORTB** register to 1

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

- The overflow interruption is to be used (ISR vector macro: **TIMER0_OVF_vect**).
- The most resource-efficient prescaler (*prescaler*) is 64, which causes the 8-bit counter **TCNT0** to overflow every *1ms* at the 16MHz CPU clock (sufficiently accurate).
- Activating/deactivating the interrupt source is done by setting/deleting the **TOIE0** bit in the register **TIMSK0**.

Configuration of the frequency of the timer **TIMER0** (bits in register **TCCR0B**)

CS02	CS01	CS00	description
0	0	0	timer off
0	0	1	CPU clock
0	1	0	CPU clock / 8
0	1	1	CPU clock / 64
1	0	0	CPU clock / 256
1	0	1	CPU clock / 1024
1	1	0	Ext. clock (falling edge)
1	1	1	Ext. clock (rising edge)

Complete the following code structure so that a fully compilable program is created.

```
#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); // Punkte dort
static uint16_t log10(uint16_t value); // Punkte dort
//
// Ignoriere ersten Durchlauf der Warteschleife
// (dafür extra halber Punkt). Alternativ: Lese Pin
// Register in der main Funktion aus (siehe unten).
static volatile uint8_t event_shutter = 1; // 1.5P
static volatile uint8_t event_timer = 0; // 1.0P
// End Function Declarations, Global Variables, etc.

// Interrupt Service Routines
ISR(TIMER0_OVF_vect) { // 0.5P
    static uint16_t counter = 0; // 1.0P
    counter++; // 0.5P
    if (counter == ACTIVATION_INTERVAL) { // 0.5P
        counter = 0; // 0.5P
        event_timer = 1; // 0.5P
    }
}

ISR(INT0_vect) { // 0.5P
    event_shutter = 1; // 0.5P
}

// End Interrupt Service Routines
```

1.5

1

4.5

D: 7

```
// Function main
void main(void) { // 0.5P
// Initialization and Local Variables
    init(); // 0.5P

    // Hier ist es ebenfalls ok, den aktuellen Wert des Pins
    // abzufragen und sich dann schlafen zu legen (siehe oben).
    uint8_t active = 0; // 0.5P

// Event Loop
while (1) { // 0.5P
    cli(); // Schütze Event-Abfrage // 0.5P
    while (!event_timer && !event_shutter) { // 1.0P
        sleep_enable(); // 0.5P
        sei(); // sleep_cpu() muss direkt nach sei() folgen
        sleep_cpu(); // 1.0P
        sleep_disable(); // Optional
        cli(); // Schütze Event-Abfrage // 0.5P
    }
    sei(); // 0.5P
```

1.5

4.5


```
// Process Events
if (event_shutter) { // Shutter *vor* Timer // 1.0P
    event_shutter = 0; // 0.5P

    active = (PIND & (1 << PD2)) != 0; // 0.5P
    if (!active) { // 0.5P
        PORTB |= (1 << PB0); // 0.5P
    }
}

if (event_timer) { // 0.5P
    event_timer = 0; // 0.5P

    if (active) { // 0.5P
        PORTB ^= (1 << PB0); // 0.5P

        cli(); // 0.5P
        int16_t sensor = sb_adc_read(PHOTO); // 0.5P
        sei(); // 0.5P

        set_iso(log10(sensor)); // 0.5P
    }
}
}

// End main
```

3

4

M: 13

```
// Logarithm to Base 10
static uint16_t log10(uint16_t value) { // 1.0P
    uint8_t log2 = 0; // 0.5P
    for (uint8_t i = 0; i < 16; i++) { // 1.0P
        if (value & (1 << i)) { // 0.5P
            log2 = i; // 0.5P
        }
    }

    return log2 / LD_10; // 0.5P
}

// End Logarithm to Base 10
```

4

L: 4

```
// Initialization Routine
static void init(void) { // 1.0P
    // Konfiguriere Prescaler (64)
    TCCR0B &= ~(1 << CS02); // 1.0P
    TCCR0B |= (1 << CS01) | (1 << CS00);

    // Aktiviere Timerinterrupts
    TIMSK0 |= (1 << TOIE0); // 0.5P

    // Konfiguriere PB0 (RED0) als Ausgang
    DDRB |= (1 << PB0); // 0.5P
    PORTB |= (1 << PB0); // 0.5P

    // Konfiguriere PD2 (Button0) als Eingang
    DDRD &= ~(1 << PD2); // 0.5P
    PORTD |= (1 << PD2); // 0.5P

    // Konfiguriere Interrupt für PD2 bei beliebiger Flanke
    EICRA |= (1 << ISC00); // 1.0P
    EICRA &= ~(1 << ISC01);

    // Aktive Button0 Interrupt
    EIMSK |= (1 << INT0); // 0.5P
}
// End Initialization Routine
```

Problem 3: alt (20 Points)

Course supervisors at FAU usually receive many emails from students asking for help. As courses progress at a fast pace, such emails are often outdated after a few days and a reply is no longer necessary. Help work-shy supervisors by writing a program `alt` that deletes all files in a given directory that are older than one week.

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

The program should work in detail as follows:

- The program checks at the beginning whether exactly one parameter has been passed. If this is not the case, it issues a corresponding error message and exits.
- If the program was called correctly, the passed directory is searched (not recursively) using `opendir()` and `readdir()`.
- The modification time (`st_mtime`) is queried for each regular file found using `stat()`.
- If the file is older than 7 days, it should be removed using `unlink()`.
- After successful execution, the program ends with the status code `EXIT_SUCCESS`.
- If errors occur, an error message should be displayed and the program should end with an error status (`EXIT_FAILURE`).
- Make sure to release allocated resources again (`free()`, `closedir()`, ...).
- Also implement the auxiliary function `time_t calc_time_limit(void)`, which determines the timestamp for 7 days ago from today:
 - Time is counted in UNIX-like systems in seconds since 01.01.1970.
 - You can use `time(NULL)` to return the timestamp for the current second.
 - Assume standard days (24 hours per day, 60 minutes per hour, 60 seconds per minute).
 - Call `calc_time_limit` only once during a program run in order to have a consistent basis for comparison.

Ensure correct error handling of the functions used. Error messages should generally be sent to `stderr`. The predefined functions `die()` (`errno` set) and `err()` (`errno` not set) can be used for compact error handling.

Complete the following code structure so that a fully compilable program is created.

```
#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) { // 1.0P
    time_t now = time(NULL); // 1.0P
    time_t seven_days = 7*24*60*60; // 0.5P
    return now-seven_days; // 0.5P
}
```

```
// Function main
int main(int argc, char *argv[]) // 0.5P
{
    // 1.0P: Überprüfe Argumente + Fehlerbehandlung
    if (argc != 2) {
        err("usage");
    }

    char *dirpath = argv[1];

    // 1.0P: Öffne Verzeichnisstrom
    DIR *dirp = opendir(dirpath);
    if (!dirp) { // 0.5P
        die("opendir"); // 0.5P
    }

    time_t limit = calc_time_limit(); // 0.5P

    errno = 0;
    struct dirent *de = readdir(dirp); // 1.0P
    while (de != NULL) { // 0.5P: Schleife
        // 1.0P: Länge inklusive '/' und '\0'
        size_t len = strlen(dirpath) + strlen(de->d_name) + 2;
        char *filename = malloc(len); // 0.5P
        if (filename == NULL) { // 0.5P
            die("malloc"); // 0.5P
        }

        // 1.0P: Setze Dateipfad zusammen
        strcpy(filename, dirpath);
        strcat(filename, "/");
        strcat(filename, de->d_name);
    }
}
```

```
// Check Timestamp
-----
struct stat sb;
-----
if (stat(filename, &sb) < 0) { // 1.5P: Aufruf + Bedingung
    die("stat"); // 0.5P
}
-----

// 1.0P check
-----
if ((sb.st_mode & S_IFREG) && (sb.st_mtime < limit)) {
    if (unlink(filename) < 0) { // 1.5P: Aufruf + Bedingung
        die("unlink"); // 0.5P
    }
}
-----

free(filename); // 0.5P
-----

errno = 0; // 0.5P
de = readdir(dirp); // Punkte oben
}
-----

if (errno != 0) { // 0.5P
    die("readdir"); // 0.5P
}
-----

// 0.5P: SchlieÙe Verzeichnisstrom
closedir(dirp);
-----

// 0.5P: Korrekter Rückgabewert
return EXIT_SUCCESS;
-----
}
-----
// End main
```

Problem 4: Memory Layout (11 Points)

Note: Read the task first - a complete understanding of the program is not necessary to complete the problem.

```
#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);
}
```

Memory Layout (simplified):

Stack ↓ ✓	
	err ✓ i ✓ msg ✓
Heap ↑ ✓	
.bss	page ✓
.data	message ✓
.rodata	NUM_PAGES ✓
⋮	

a) Complete the (simplified) memory layout:

- 1) Add the terms *stack* and *heap* and their direction of growth in the figure. (2 Points)
- 2) Assign all variables occurring in the source code to the corresponding memory segments. (3 Points)

5 Points

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

b) Upon starting the microcontroller, how must the RAM memory be prepared before the actual program execution in `main()` can begin? (2 Points)

- Copy .data from non-volatile memory
(flash/ROM) to RAM **1 Point**

- Zero .bss segment **1 Point**

c) The following table contains four pointer types. For all types, specify whether the dereferenced value and the pointer are mutable (i.e., not constant) (**yes** or **no** in each case). If a data type is not possible in C, please check the column **syntax error**. (4 Points)

	Value mutable	Pointer mutable	Syntax error
<code>uint8_t *</code>	yes✓	yes✓	
<code>const uint8_t *</code>	no✓	yes✓	
<code>uint8_t * const</code>	yes✓	no✓	
<code>const uint8_t * const</code>	no✓	no✓	

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

Problem 5: Concurrency (7 Points)

The following descriptions should be short and concise (keywords, short sentences).

a) Which steps **in hardware** typically comprise the processing of an interrupt on a microcontroller? (3 Points)

1. Device signals interrupt, Programm will be interrupted 0.5 Points
2. Further interrupts are blocked 0.5 Points
3. Register values will be saved 0.5 Points
4. ISR will be determined 0.5 Points
5. ISR will be executed 0.5 Points
6. ISR terminates, register values will be restored, program resumes 0.5 Points

b) The use of interrupts can lead to various concurrency problems. Name and describe **two** examples of such problems. In addition, describe how to deal with them to ensure the correct program behaviour. (4 Points)

Lost-update: 0.5 Points

Register contains value, ISR updates the value and restores the register upon returning **1 Point**

Read-Write: 0.5 Points

Value exceeds maximum representable value of a single register. Value will be updated. ISR interrupts update right in between and corrupts the result of the calculation (e.g. overflowing operation) **1 Point**

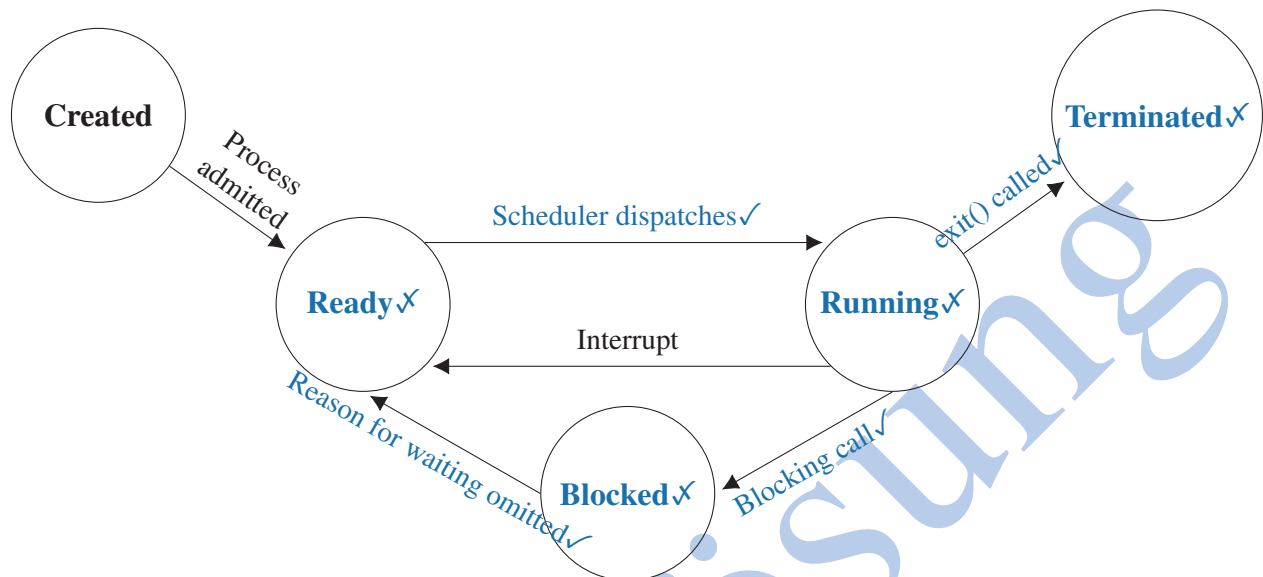
or alternatively: Lost wakeup:

Interrupt right between checking the sleep condition and going to sleep

Solution: Mask interrupts briefly 1 Point

Problem 6: Processes (8 Points)

a) Complete the following graph for the different Linux process states with the corresponding state transitions. Each node corresponds to a process state and each edge corresponds to a state transition. Make sure to label **all** nodes and edges that have not yet been labeled. (6 Points)

**Korrekturhinweise:**

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

b) Name one common problem each when using a **Spin Lock** and a **Sleeping Lock**. (2 Points)

Spin Lock: For long critical section,
CPU cycles are wasted **1 Point**

Sleeping Lock: For short critical section,
large overhead related to blocking/resuming **1 Point**