

This document includes solution hints, but **NOT** necessarily the full solution!

## Part I

# Evaluation Page Part B

Name: \_\_\_\_\_ Signature: \_\_\_\_\_

## 1 Evaluation

This part of the exam has 161 questions, with a total of 427 points and 6 bonus points.

Part	Max. Points	Scored Points
A	60	
B	180	
	Total	

## 2 Evaluation Part B

Page	Points	Bonus Points	Score
5	19	1	
6	1	0	
7	18	0	
8	11	0	
9	12	0	
10	14	0	
11	12	0	
12	14	0	
13	12	0	
14	12	0	
15	14	0	
16	2	1	
17	6	0	
18	6	0	
19	11	0	
20	9	0	
21	13	0	
22	2	0	
23	12	0	
24	7	0	
25	4	0	
Total:	211	2	

Page	Points	Bonus Points	Score
27	5	2	
28	10	0	
29	4	0	
30	5	0	
31	14	0	
32	16	0	
33	11	0	
34	7	0	
35	6	0	
36	10	0	
37	10	0	
38	6	0	
39	9	0	
40	12	0	
41	10	0	
42	7	0	
43	11	0	
44	6	0	
45	9	0	
46	6	0	
47	9	0	
49	9	0	
50	3	0	
51	18	0	
52	3	0	
53	0	2	
Total:	216	4	

## Part II

# Rules

Answer the questions within the space provided. If you do not have enough space, you can use the backside of the sheet. In that case clearly indicate that your answer continues on the backside.

### 3 Supporting Materials

This is an examination in writing, without the usage of any electronic devices, except a scientific pocket calculator. No restriction on the model of calculator that may be used, but no device with communication capability shall be accepted as a calculator. All other electronic devices are prohibited. Writing paper is available, writing instruments (pencil, pens, etc) have to be organized by the student.

- **Part A:** Without any supporting material, with calculator.
- **Part B:** With a self written summary (format A4, 8 sheets or up to 16 pages), with calculator

### 4 Procedure

1. Duration:  
**Part A:** 1 hour = 60 minutes = 60 points.  
(short break)  
**Part B:** 3 hours = 180 minutes = 180 points.
2. Sign the first page in the provided space. With this you certify that you are only using permitted support material and you are complying to the rules.
3. Write your name on any detached or additional paper sheets. Sheets without a name will not be evaluated.
4. Use the provided paper for your solutions. Use the provided space in the forms and tables. If needed use scratch paper. Document your way to your solution as appropriate.
5. Each question has a defined number of maximum points associated.
6. If a question is unclear, make reasonable assumptions. Document your assumptions and provide a rationale.
7. Write clearly and legibly. Unclear or multiple solutions will not be evaluated.
8. There is a short break between part A and B. You have to sign into a list for a needed break during the examination parts. Only one person can leave the room for a short time.
9. If something is unclear, ask your supervisor in the room.

## 5 Time Management

Read first all questions. Make sure you distribute your available time to all the questions. To reduce disturbance, ask questions in the first 15 minutes of the exam period.

## 6 Multiple-Choice Questions

1. Try to answer all questions if possible. If you are not sure, choose the answer which seems the best one.
2. For the questions of type ○: Choose **exactly one** option with ⊗ (or √), which you think is the best match. With a correct answer you get the given number of points for that question.
3. For the questions of type ±: After a question or possibly incomplete sentence there are four answers or extensions. Evaluate each of them if they are true or false and mark them accordingly with '+' (true) or '-' (false). Independent if the question is formulated grammatically in singular or plural, it is possible that **0, 1, 2, 3, 4** of the choices are true. For three correct answers out of four you receive half of the points.
4. Wrong answers will have no penalty. Each question which has no answer is treated like a wrong answers and will be evaluated with zero points.
5. If you are changing your mind: cross out your old answer and clearly mark which answer is the new one.

May Dilbert be with you! ☺

**Question 1 ..... 1 Point (Bonus)**

This could be your own bonus question ☺.

1. What would be the answer?

**Question 2 ..... Points: [2]**

In version control systems you are asked to provide a 'commit message': explain why you should provide such a message:

**Solution:** It allows to list them with the commits, so other users have a high level understanding of the commit.

**Question 3 ..... Points: [2]**

In our drivers we implemented `Deinit()` functions, for example for the shell:

```
void SHELL_Deinit(void);
```

Explain the purpose of such `Deinit()` functions:

**Solution:** Used to de-initialize the driver, free up memory or semaphores/etc. Get the driver back to the reset state.

**Question 4 ..... Points: [5]**

- (a) The boards we used (Robot and FRDM) have different number of LEDs available. Describe your strategy how you can write a common LED driver for all these boards (supporting multiple platforms with a single driver):

[3]

**Solution:** Using a platform define with macros to describe how many LEDs they have, and a macro to enable/disable it.

- (b) Describe with your approach how you can deal with a board which has no LED's at all:

[2]

**Solution:** Use a macro to disable it completely, or set number of LED to zero.

**Question 5 ..... Points: [10]**

Given the following interfaces:

```
bool KEY_Get(void); /* return FALSE if key is pressed, TRUE otherwise */
void WAIT_Waitms(uint16_t ms); /* realtime waiting for the given
    milliseconds */

typedef enum {
    NO_KEY,      /* no key pressed */
    SHORT_KEY,   /* short key press */
    LONG_KEY     /* long key press */
} KEY_State;

KEY_State GetKey(void);
```

Implement the function `GetKey()` without the usage of interrupts, with following requirements:

1. Return `NO_KEY` if the key is not pressed.
2. If the key is pressed, debounce it for 50 ms.
3. If the key is pressed for less or equal than 500 ms, return `SHORT_KEY`.
4. For a short key press or long key press detection, the function shall not block longer than needed.
5. If the key is pressed for more than 500 ms, return `LONG_KEY`.

#### Solution:

```
PTADD = 0x00; /* set port as input */
PTAPE = 0xFF; /* enable pull ups */
while (PTAD!=0xFF); /* wait until port pull ups are working */
EnableInterrupts;
```

#### Solution:

```
KEY_State GetKey(void) {
    if (!KEY_Get()) { /* press */
        WAIT1_Waitms(50); /* simple debounce */
        if (!KEY_Get()) { /* still pressed? */
            cnt = 0;
            while (!KEY_Get() && cnt<=500) {
                WAIT_Waitms(1);
                cnt++;
            } /* wait until released */
            if (cnt>=500) { /* long press */
                return LONG_KEY;
            } else {
                return SHORT_KEY;
            }
        }
    } else {
        return NO_KEY;
    }
}
```

#### Question 6.....Points: [7]

In INTRO we used a command line shell module and implementation.

- (a) What's the advantage of using a shell?

[1]

**Solution:** Universal and simple user interface to configure and control the application.

- (b) List three different USB *device* classes: [1]

**Solution:** Human interface device (HID), Communication device class (CDC), Mass storage class (MSC), Physical interface device (PID)

- (c) USB uses 'non-return-to-zero', so it inserts a zero marker after 6 consecutive one bits. Why? [2]

**Solution:** To allow the receiver to synchronize on the bits.

- (d) The USB bus protocol uses a frequency of 12 MHz for the bits. Why did we have to configure the microcontroller for 24 MHz operation? [2]

**Solution:** Nyquist-Shannon, need to sample it at least twice as fast.

- (e) What is special with the OTG USB class, compared to host or device classes? [1]

**Solution:** It allows two devices to connect, regardless of host or device (ad-hoc).

**Question 7.....Points: [4]**

Explain the difference between a windowed and a non-windowed Watch Dog Timer, and what kind of failures can be caught only with a windowed Watch Dog Timer.

**Solution:** The timer needs to be reset inside a defined window. Resetting it outside of the windows is considered as a failure and will reset the system. With this cases like code resetting the timer inside a fast loop will be prevented.

**Question 8.....Points: [8]**

You are using GIT as VCS. Explain what the following actions mean. Use '*remote repository*', '*local repository*', '*index*' and '*working directory*' in your answers.

- (a) clone [2]

**Solution:** Copy the remote repository as local repository.

- (b) add [2]

**Solution:** Add a new file to the index.

- (c) push/sync [2]

**Solution:** Add my changes from the index/local repository to the global repository.

- (d) commit [2]

**Solution:** Add my changes from the working directory/index to the local repository.

**Question 9.....Points: [4]**

Given the following Watchdog State Machine:

```
void wdt_a(void)
{
    if (state != 0x5555) {
        HALT;
    }
    state += 0x1111;
}

void wdt_b(void)
{
    if (state != 0x8888) {
        HALT;
    }
    KickDog();
    if (state != 0x8888) { /* Question: why? */
        HALT;
    }
    state = 0;
}

void main(void){
    for (;;) {
        state = 0x5555;
        wdt_a();
        ...
        state += 0x2222;
        wdt_b();
    }
}
```

Explain why that second check in `wdt_b()` on `(state != 0x8888)` is necessary:

**Solution:** Wrong code could vector onto the `KickDog()` line in `wdt_b()`.

**Question 10.....Points: [3]**

List three different clock sources which can be used for the microcontroller we used in INTRO:

**Solution:** External crystal, external oscillator, internal clock.

**Question 11.....Points: [4]**

You are using a MEMS accelerometer. This accelerometer should provide the value 0x7FFF (32767) for 0 g, and 0xCFFF (53247) for 1 g. But when you read from the sensor, it reports a value of 33000 for 0 g, and a value of 50000 for 1 g. Determine offset error and gain error factor at 1g:



**Solution:** Offset =  $32767 - 33000 = 233$ . Gain =  $(53247 - 32767) / (50000 - 33000) = 1.21$

**Question 12.....Points: [3]**

We used in our shell parser the following `xatoi()` interface:

```
byte UTIL1_xatoi(const unsigned char **str, long *res);
```

Explain why it is using `**str` and *not* `*str`:

**Solution:** It advances the parsing pointer, so `xatoi()` can be used sequentially on the input string.

**Question 13.....Points: [3]**

In our labs we used the RTOS with a tick timer period of 10 ms. List advantages and disadvantages for changing it to a period of 1 ms:

**Solution:** Pros: higher accuracy for wait/timing/scheduling (1). Cons: higher interrupt load (1). Performance tick counter needs to be changed too (1).

**Question 14.....Points: [3]**

Below is an extract of a header file:

```
/* drv.h */
#ifndef DRV_H
#define DRV_H
/* header file content follows here... */
#endif
```

Using the above header file, you find out that things are not working properly. Identify the problem.

**Solution:** Typo, should be `#ifndef`.

**Question 15.....Points: [3]**

Document for following code with doxygen (what the function does, parameter(s)):

```
void LED_On(LED_Set Leds);
```

**Solution:** `/*!`  
 \brief Switches on a set of LED.  
 \param[in] Leds The set of LED to be switched on.  
 `*/`

**Question 16.....Points: [3]**

FreeRTOS task definitions are using the macro `portTASK_FUNCTION`:

```
static portTASK_FUNCTION(RemoteTask, pvParameters) {
    ...
}
```

Explain the reason and advantage of using the macro `portTASK_FUNCTION`:

**Solution:** This is a macro to hide any compiler/architecture (1) specifics for function definitions (2).

**Question 17.....Points: [3]**

List advantages and disadvantages of using Processor Expert for a project we did in INTRO:

**Solution:** Pros: hardware abstraction, provided libraries. Cons: learning phase, code not that optimized.

**Question 18.....Points: [3]**

The microcontrollers we used in INTRO are using a table for the vectors: The vector table contains the address of the vector function. Explain what will happen during power-on reset if the whole vector table would be filled with zero (0x00) bytes:

**Solution:** Power-On reset will not work (no stack defined, no startup vector) (1.5). Code will divert to address 0x0, potentially crash. (1.5)

**Question 19.....Points: [3]**

Explain the fundamental differences between using `EnterCritical()` ... `ExitCritical()` and `xSemaphoreTake()` ... `xSemaphoreGive()` with respect to interrupts:

**Solution:** `EnterCritical` disables the interrupts (1), while the other method is using semaphores and itself does not disable interrupts (2).

**Question 20.....Points: [4]**

On the FRDM board we used an interrupt to detect button presses on the joystick shield.

(a) Why do we not know *which* button was pressed from the interrupt source? [1]

**Solution:** There is only one interrupt source for multiple interrupt pins (only one port interrupt, but multiple port pins).

(b) How can you know which button is pressed? [1]

**Solution:** Read the port pin value.

- (c) We get an interrupt for a button. But when we check the pin status in the debouncing state machine we see that this pin is not pressed. What could be the most likely problem? [2]

**Solution:** If at the interrupt execution time the switch state already has been closed (port does not reflect the state at the interrupt time).

**Question 21.....Points: [4]**

Figure 1 shows three different circuits how to connect a switch to a microcontroller. Rate them either 'good', 'fair' and 'poor' with a short explanation.

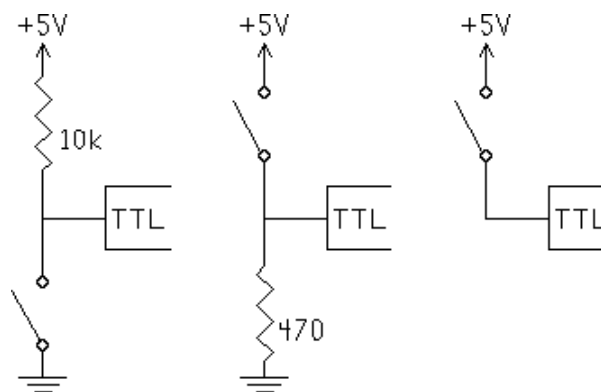


Figure 1: Three Switches

**Solution:** (good, fair, poor), Current, defined state, internal pullup.  
see <http://www.freewebs.com/maheshwankhede/ports.html>.

**Question 22.....Points: [3]**

The following `#define` might cause a problem using it:

```
#define INCREMENT(a)    a+1
```

Explain the problem and provide a solution how to fix it:

**Solution:** Textual replacement, order of operation might depend on usage. Better to use `((a)+1)`.

**Question 23.....Points: [3]**

Explain why FreeRTOS is using 'number of addressable units' as parameters to create a task, and not 'number of bytes':

**Solution:** Because stack needs to be aligned anyway on addressable units.

**Question 24** ..... **Points: [4]**

An RTOS is using following interface for a 'wait' API call:

```
void vTaskDelay(portTickType xTicksToDelay);
```

- (a) Explain why FreeRTOS is using 'number ticks' as parameters to the delay routines and not 'number of milliseconds': [2]

**Solution:** Because the tick timer can be in smaller units than milli-seconds.

- (b) You need to wait for 200 milliseconds with `vTaskDelay()`, but you do not want to depend on the frequency of the tick timer. How can you do this? [2]

**Solution:** Using something like `vTaskDelay(200/portTICK_RATE_MS);`

**Question 25** ..... **Points: [3]**

Evaluate the following statements about interrupt system for the FRDM (ARM Cortex M0+ core).

- ☐  $\pm$  I cannot change the interrupt priorities of the Cortex M0+ interrupts with positive interrupt vector numbers.
- ☐  $\pm$  Multiple interrupts can have the same priority on the Cortex-M0+.
- ☒  $\pm$  I cannot change the interrupt priority of the negative interrupt vector numbers.
- ☒  $\pm$  The ARM-Cortex M0+ core has only main priorities and no sub-priorities.

**Question 26** ..... **Points: [2]**

List two advantages of using an optical quadrature encoder compared to a mechanical one:

**Solution:** Optical encoders do not have the problem of bouncing, and they can operate at faster signal frequency as there are no mechanical parts involved. Additionally they are free of maintenance or wear-out.

**Question 27** ..... **Points: [2]**

List the 4 specific requirement for a realtime system:

**Solution:** The (a) right answer at the (b) right time, (c) independent of the system load, (c) in deterministic way.

**Question 28** ..... **Points: [3]**

Explain the fundamental difference between a hard and a soft realtime system:

**Solution:** A hard realtime system has to comply with a hard time boundary: missing that boundary means failure. A soft realtime system might miss that time boundary, and the system still would be considered working properly, but missing the time boundary would mean a system degradation (not a failure).

**Question 29**.....**Points: [2]**

Explain the advantages and disadvantages of using Processor Expert for a system we have developed in INTRO:

**Solution:** It allows to abstract from the hardware, and to focus on the higher level of the application. It generates the driver code and saves development time. On the other side it necessary to learn a tool, and to configure it properly.

**Question 30**.....**Points: [2]**

Explain reasons why you would use a VCS:

**Solution:** Backup, restore, synchronization, tracking, short/long term undo, ownership, sandbox, branching, merging.

**Question 31**.....**Points: [2]**

Explain reasons when you should use a VCS:

**Solution:** Every time you need control over your code and/or when you work together with other persons on the same code.

**Question 32**.....**Points: [2]**

Explain disadvantages and advantages using macros:

**Solution:** Cons: Code might be harder to read (additional textual replacement), there are traps and pitfalls using macros. Pros: Easy way to configure code at compilation time (versus run time), better/faster code, helping the compiler to optimize code generation.

**Question 33**.....**Points: [2]**

Explain why synchronization between systems is needed:

**Solution:** Systems need to get aligned and synchronized with the real world. The real world time is continuous, while the computer time is clock based. To have computers attached to the real world, they need to synchronize with the time of the real world.

**Question 34**.....**Points: [2]**

List three different synchronization methods:

**Solution:** Interrupt synchronization, realtime synchronization, gadfly/polling synchronization.

**Question 35.....Points: [1]**

List at least two different output formats produced by doxygen:

**Solution:** html, rtf, xml, L<sup>A</sup>T<sub>E</sub>X

**Question 36.....Points: [1]**

List a compelling reason why it makes sense to document an API in the sources with doxygen:

**Solution:** The documentation and the API are always in sync (if the documentation is properly updated), and everything is in one place.

**Question 37.....Points: [2]**

Explain a case where a synchronization between two systems does not make any sense:

**Solution:** If one system is so slow that it cannot react to the events from the other system. The system is overloaded and has no time to synchronize anyway.

**Question 38.....Points: [2]**

Explain reasons why we used a Shell in INTRO:

**Solution:** To communicate (i.e. inspect, configure, debug) our target.

**Question 39.....Points: [2]**

List reasons why you would use polling instead of interrupts for checking the state of a key:

**Solution:** Hardware does not support interrupts on that pin(s), no problem with missing a key press, plenty of computation time available.

**Question 40.....Points: [2]**

List key benefits of using open source operating system like FreeRTOS:

**Solution:** Sources available, free of charge, community working on development.

**Question 41.....Points: [2]**

Explain the difference between a pre-emptive and a cooperative multitasking:

**Solution:** Preemptive: scheduler is in charge who is executing, tasks get interrupted. In cooperative mode the task needs to return the control to kernel.

**Question 42.....Points: [2]**

What are stdin, stdout and stderr?

**Solution:** These are callbacks and standard handles (or pipes) used by the shell. Similar to what exists in Linux.

**Question 43.....Points: [3]**

Given in figure 2 a switch on a board: If the switch is **not** pressed, PTA5 will have

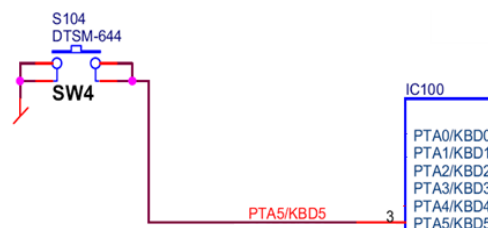


Figure 2: SRB Switch

a logical **HIGH** voltage level. How can this work?

**Solution:** The microcontroller port is configured to use internal pull-ups to Vdd.

**Question 44.....Points: [9]**

FreeRTOS is used in priority based preemptive mode. Tick timer is set up for 10 ms, and the processor is running at maximum speed. Task T3 has priority 3 (highest priority), task T2 has priority 2 and task T1 has priority 1. The tasks have been created with `xTaskCreate()` before time  $t=0$  ms, and the scheduler is started with `vTaskStartScheduler()` at the time  $t=0$  ms. Draw in Figure 3 a timing diagram for the execution of the two tasks for the first 100 ms (after  $t_0 = 0$  ms). Use a bar to indicate when a task is running. You can ignore the overhead in the task loop and the overhead in the RTOS/scheduler itself.

```
static portTASK_FUNCTION(T3, pvParameters) { /* priority 3 task */
    portTickType xLastTime = xTaskGetTickCount();
    for (;;) { /* task time is 5 ms including overhead */
        DoWorkFor5ms(); /* this needs 5 ms */
        vTaskDelayUntil(&xLastTime, 25/portTICK_RATE_MS);
    } /* loop forever */
}

static portTASK_FUNCTION(T2, pvParameters) { /* priority 2 task */
    for (;;) { /* task time is 10 ms including overhead */
        DoWorkFor10ms(); /* this takes 10 ms */
        vTaskDelay(30/portTICK_RATE_MS);
    } /* loop forever */
}
```

```
static portTASK_FUNCTION(T1, pvParameters) { /* priority 1 task */
    for (;;) { /* task time is 2 ms including overhead */
        DoWorkFor2ms(); /* this needs 2 ms */
        vTaskDelay(20/portTICK_RATE_MS);
    } /* loop forever */
}
```

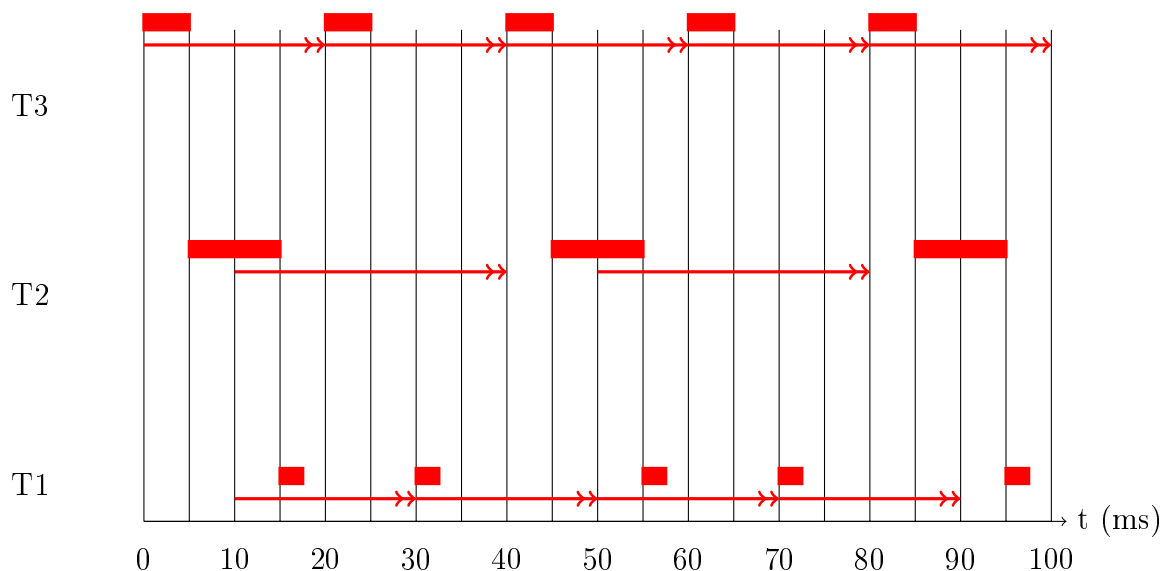


Figure 3: Task Timing

**Question 45.....Points: [2]**

Evaluate following statements for the ANSI `strcmp()` and `strncmp()` library functions as used in the INTRO program:

- $\ominus \pm$  `strcmp()` has one additional parameter compared to `strncmp()`.
- $\oplus \pm$  `strcmp()` and `strncmp()` return both zero, if the two argument strings are equal.
- $\oplus \pm$  Using `strncmp()` allow to compare two strings up to a given offset.
- $\ominus \pm$  The parameter passing for `strncmp()` is usually more efficient than the parameter passing for `strcmp()`.

**Question 46.....1 Point (Bonus)**

List a compelling reason why normally only debounced switches should be used at an interrupt generating input pin only:

**Solution:** As the switch is bouncing, it might create many (too many) interrupts on that input line. This will increase the system load and it might cause signaling multiple switch presses even if there was just one.



**Question 47.....Points: [2]**

Determine the value of variable j after calling the function foo():

```
#define SUM(a,b) (a)+b
#define MUL(a,b) a*b
#define CALL(a) bar(a)

int bar(int i) {
    return MUL(i,SUM(3,MUL(5,10)));
}
int j;

void foo(void) {
    j = CALL(SUM(10,2));
}
```

47. (12\*(3)+5\*10) = 86

**Question 48.....Points: [10]**

Consider following semaphore definition:

A semaphore S denotes an integer variable, which is accessed through the following two atomic operations:

```
#define DOWN(S)    while (S<=0); /* do nothing */ \
                  S = S-1;
#define UP(S)      S = S+1;
```

Using the two operations, it is possible for a program to implement a critical section or a mutex, as in the following example:

```
Semaphore S = 1; /* initialize semaphore to 1 */
```

Each process or task is able to protect its critical section:

```
while(1) {
    DOWN(S);
    /* in critical section */
    UP(S);
}
```

- (a) Is the implementation with UP() and DOWN() using a 'busy waiting'? Justify your answer. [2]

**Solution:** This semaphore definition involves the busy waiting because a waiting process is looping for the shared variable S.

- (b) The tasks are using a priority based scheduling. Is it possible that the above solution leads to a priority inversion problem? Provide an rationale for your answer and provide an example why. [2]

**Solution:** Yes, the above solution can lead to the priority inversion problem if a high-priority process is waiting for a low-priority process to leave the critical section and signal the shared variable S.

- (c) Implement in pseudo code the tasks A, B, C, D and E. Use UP() and DOWN() on the needed semaphores to implement following scenario: After both A and B have finished their work, task C can start. After task C has finish its work, both D and E can start. [6]

**Solution:** The relations among processes can be represented in the following diagrams.

```

A \    / D
S1 \  /
    C S3
S2 /  \
B /    \ E

```

The program can be as follows:

```
semaphore S1, S2, S3 = 0, 0, 0;
```

Process A: (2p)

-----

- do work of A

```
UP(S1); /* Let C start */
```

Process B: (2p)

-----

- do work of B

```
UP(S2); /* Let C start */
```

Process C: (4p)

-----

```
DOWN(S1); /* Block until A finished */
```

```
DOWN(S2); /* Block until B finished */
```

- do work of C

```
UP(S3); /* Let D or E start */
```

```
UP(S3); /* Let D or E start */
```

Process D: (2p)

-----

```
DOWN(S3); /* Block until C finished */
```

- do work of D

Process E: (2p)

-----

```
DOWN(S3); /* Block until C finished */
```

- do work of E

Total: 10

**Question 49.....Points: [4]**

Explain the *Priority Inversion* problem and illustrate it with an example:

**Solution:** A priority inversion problem is a situation in which a low-priority process is blocking a high priority process.

**Question 50.....Points: [2]**

We are using a tick timer with a period of 10ms a preemptive RTOS (FreeRTOS). This tick timer is as well used for the Trigger module as implemented in INTRO. Describe the consequences for the application and RTOS if this tick timer period gets changed from 10 ms to 200 ms, with the example of our INTRO application at the end of the semester.

**Solution:** A preemptive scheduling would happen only every 200 ms. Triggers would have the resolution of 200 ms, and this would impact our debouncing (it would take longer). Wait/Waituntil would have a resolution of 200 ms.

**Question 51.....Points: [10]**

A device is connected to a microcontroller as in Figure 4. It is using a data bus (PTB: Data) and two control signals (PTA0: status; PTA1: control). The protocol to send an 8bit data byte is illustrated in Figure 4. Implement the protocol using a

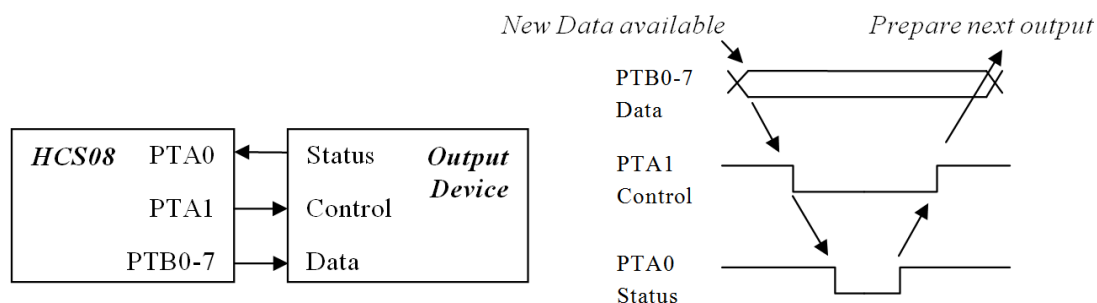


Figure 4: Output Device

*Gadfly* synchronization method.

*Note: use pseudo code as SetPTA0asInput(), but make sure with comments and naming to make clear what it does*

- (a) Implement the function `Init()` which initializes the ports and any other required variables. [5]

**Solution:**

```
void Init(void) { (3p)
    PTAD_A0 = 1; /* default pin value to 1 */
    SetPTA0asInput; /* pin as input */

    PTAD_A1 = 1; /* pin set to 1 */
```

```

SetPTA1asOutput ;

PTBD = 0; /* init port */
SetPTBasOutput; /* PTB as output */
}

```

- (b) Implement the function **Send(char)** which transmits a single byte.

[5]

**Solution:**

```

void Send(char ch) {
/* optional: wait until PTA0 is high */
PTBD = ch;
PTAD_A1 = 0; /* signal to device */
while (PTAD_A0==1); /* wait until signal goes down */
while (PTAD_A0==0); /* wait until signal goes up */
PTAD_A1 = 1; /* signal finish transfer */
}

```

Total: 10

**Question 52.....Points: [4]**

- (a) Determine for the 8bit (binary reflected) Gray code **0x47<sub>g</sub>** the corresponding 8bit Binary code<sub>b</sub>:

[2]

(a) 122/0x7A<sub>b</sub>**Solution:** Gray to Binary:0x47<sub>g</sub>, then use MSB, then add each next bit (ignore carry):0100'0111 → 0111'1010 → 0x7A<sub>b</sub>

- (b) Determine for the 8bit (binary reflected) Gray code **0x74<sub>g</sub>** Code the corresponding 8bit Binary code<sub>b</sub>:

[2]

(b) 88/0x58**Solution:** Gray to Binary:0x74<sub>g</sub>, then use MSB, then add each next bit (ignore carry):0111'0100 → 0101'1000 → 0x58<sub>b</sub>

Total: 4

**Question 53.....Points: [6]**

Consider following ANSI-C implementation:

```

typedef enum LED_Set {
    LED_0 = 1,
    LED_1 = 2,
    LED_2 = 4,
    LED_3 = 8
}

```

```
} LED_Set;
```

Alternatively it can be written as:

```
#define LED_0 1
#define LED_1 2
#define LED_2 4
#define LED_3 8
```

- (a) List pros and cons for using enumerations for above example:

[2]

**Solution:** Pros: symbolic debugging, using a type. Cons: it is using an int (which might be an overkill). Enums are int's in C.

- (b) List pros and cons for using #define's for above example:

[2]

**Solution:** Pros: normal int type, is portable, can calculate with int's (masking, binary operations). Con: no symbolic debugging, no type for memory allocation, no type

- (c) Explain the reason why the values 1, 2, 4, 8 are used (and not 0, 1, 2, 3):

[2]

**Solution:** Possible to use the LED's as bit set (efficient for passing multiple LED's in one argument).

**Question 54.....Points: [2]**

Which three critical hardware settings do you have to configure on the FRDM board to communicate using an RS-232-to-USB connection. Use the shell we used in INTRO as example.

**Solution:** Port/SCI to use, Baudrate, protocol (8N1).

**Question 55.....Points: [5]**

In this example FreeRTOS is used in priority based pre-emptive mode. Tick timer is set up to 10 ms, and the processor is running at maximum speed. The two LED's are switched off at the beginning. Task T1 has priority 3, and task T2 has priority 2. T1 has to work for 15 ms, and T2 for 2 ms (the overhead within the `for` loop can be ignored). Both tasks have been created with `xTaskCreate()` right after each other, and the scheduler has been started with `vTaskStartScheduler()` at the time `t=0` ms.

```
static portTASK_FUNCTION(T1, pvParameters) { /* priority 3 task */
    portTickType xLastTime = xTaskGetTickCount();
    LED1_Off();
    for (;;) { /* task time is 15 ms including overhead */
        DoWorkFor15ms();
        LED1_Neg();
        vTaskDelayUntil(&xLastTime, 25/portTICK_RATE_MS);
    } /* loop forever */
}
```

```

}

static portTASK_FUNCTION(T2, pvParameters) { /* priority 2 task */
    portTickType xLastTime = xTaskGetTickCount();
    LED2_Off();
    for (;;) { /* task time is 2 ms including overhead */
        LED2_Neg();
        DoWorkFor2ms();
        vTaskDelayUntil(&xLastTime, 30/portTICK_RATE_MS);
    } /* loop forever */
}

```

Draw in Figure 5 a timing diagram for the execution of the two tasks for the first 100 ms (after  $t_0=0\text{ms}$ ). Indicate the state of both LED's (LED1 and LED2). Use a bar to indicate when a task is running, and a bar to indicate the time when a LED is on. You can ignore the overhead in the task loop and the overhead in the RTOS/scheduler itself.

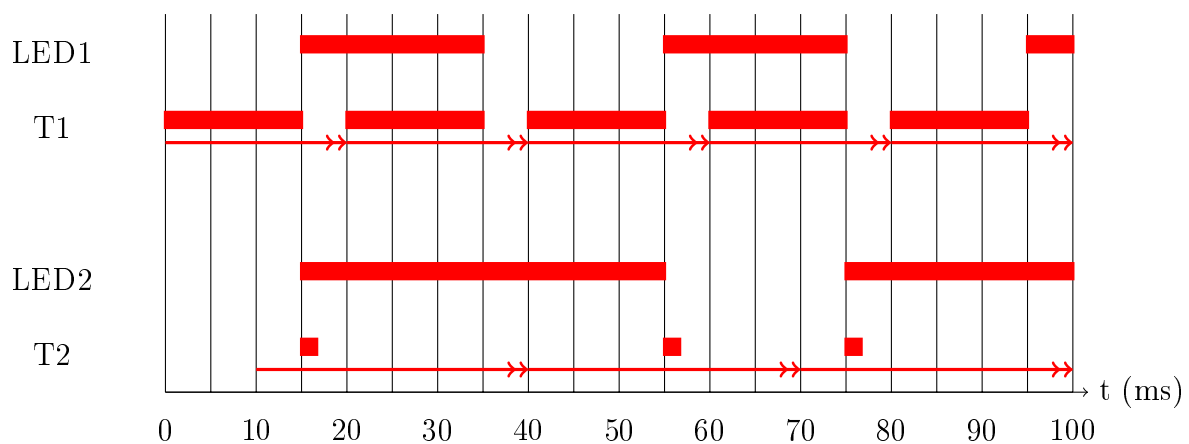


Figure 5: Task Timing

**Solution:** Keep in mind that the tick task counter resolution is 10ms: as such the `vTaskDelayUntil()` will take the current time at 10 ms boundary (e.g if the 'real' time is at 15ms, this corresponds to a tick counter of 1 (at 10 ms). If you make now a `vTaskDelayUntil()`, the time is taken from the tick counter and added. So if T2 calls `vTaskDelayUntil()` at the real time of 17 ms, this corresponds to a tick time of 10ms. Adding 30ms means that the task will be ready at 40 ms (however only, if not a higher priority task is running).

- correct LED toggling
- correct task length (15ms and 2ms)
- correct DelayUntil()

**Question 56.....Points: [2]**

Explain the difference between `xQueueReceive()` and `xQueuePeek()` in FreeRTOS:

**Solution:** `xQueueReceive()` removes the element from the queue. `xQueuePeek()` only checks if the queue contains an element, but does not remove it.

**Question 57.....Points: [12]**

Given a system using *Priority Ceiling* and three tasks ( $J_1$  (Priority 1, highest priority),  $J_2$  (priority 2) and  $J_3$  (priority 3)) and two semaphore  $S_1$  and  $S_2$ . The tasks are getting a lock of a semaphore with  $LS_x$ , and return the lock with  $US_x$ . The tasks are using the semaphores according to following timing and sequence:

- $J_1 = \{10\text{ms}, LS_1, 10\text{ms}, US_1, 10\text{ms}\}$ : total 30ms task time.
- $J_2 = \{10\text{ms}, LS_2, 10\text{ms}, LS_1, 10\text{ms}, US_1, 10\text{ms}, US_2, 10\text{ms}\}$ : total 50ms task time.
- $J_3 = \{10\text{ms}, LS_1, 30\text{ms}, US_1, 10\text{ms}\}$ : total 50ms task time.

(a) Calculate the *Priority Ceiling* for  $S_1$ : [2]

(a)  $J_1 \rightarrow 1$

(b) Calculate the *Priority Ceiling* for  $S_2$ : [2]

(b)  $J_2 \rightarrow 2$

(c) The delta between  $t_n$  and  $t_{n+1}$  in Figure 6 is 10 ms. Following times are defined: [8]

- $J_3$  starts at time  $t_0$ .
- $J_2$  starts at time  $t_2$ .
- $J_1$  starts at time  $t_4$ .

Complete and extend the timing diagram in Figure 6 indicating the task execution times. Mark with an  $LS_x$  the request for a semaphore  $x$ , and with  $US_x$  the release of a semaphore.

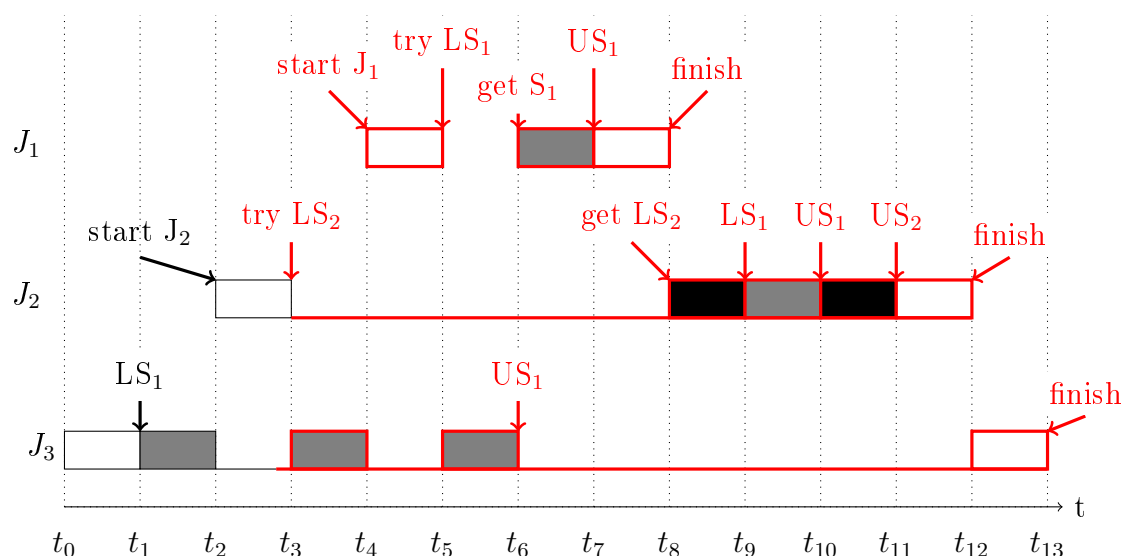


Figure 6: PCP

**Question 58.....Points: [2]**

The PWM signal used to drive the INTRO motor is 'low active'. Explain what this means.

**Solution:** The speed of the motor is proportional to the low level duty cycle of the signal: the more the signal is low, the higher the motor voltage will be.

**Question 59.....Points: [1]**

The following code

```
s = k0*e + k1*(e-eprev)/Ta;
eprev = e;
```

Implements in software a closed loop control using the following method:

- ☒ PD
- ☐ ID
- ☐ PI
- ☐ P
- ☐ PID

**Question 60.....Points: [2]**

Consider the H-Bridge we used for our INTRO lab. You have to implement a software which is able to drive the motor, from -100% (full speed backward) to +100% (full speed forward). Explain how you realize this with the signals you have available.

**Solution:** The range from 0 to 100% is realized with the PWM duty cycle (which is low active on our platform). 0 will be no speed, and 100% will be full speed. The direction signal (DIR) is used to drive the motor either forward (e.g. HIGH) or backward (e.g. LOW).

**Question 61.....Points: [2]**

- (a) Determine for the 8bit **Binary Code 0x31<sub>b</sub>** the corresponding 8bit (binary reflected) Gray code<sub>g</sub>:

[1]

(a) 0x29<sub>b</sub>

**Solution:** Binary to Gray:

0x31 EXOR (0x31»1)

= 0x31 EXOR 0x18 = **0x29<sub>b</sub>** (or **0b0010'1001** or **41<sub>d</sub>**)

- (b) Determine for the 8bit (Binary Reflected) **Gray Code 0x8F<sub>g</sub>** the corresponding 8bit Binary Code<sub>b</sub>:

[1]

(b) 0xF5<sub>b</sub>



**Solution:** Gray to Binary:

0x8F<sub>g</sub>, then use MSB, then add each next bit (ignore carry):

0xF5<sub>b</sub> (or 0b1111'0101 or 245<sub>d</sub>)

**Question 62.....Points: [1]**

Given the pseudo code for a closed loop control system:

```
esum += e;
s = a*e + b*Ta*esum;
```

This software controller implements a

- ☐ PD controller
- ☐ ID controller
- ☒ **PI controller**
- ☐ P Controller
- ☐ PID Controller

**Question 63.....Points: [1]**

For the digital encoder (using C1 and C2 signals) and the motor (as used in the INTRO lab) following applies:

- ☐  $\pm$  The quadrature encoder produces with C1 and C2 a 4 bit Gray encoded signal.
- ☐  $\pm$  In case C1 is not working or not available (only C2 is producing a signal), then we still can determine the motor direction.
- ☐  $\pm$  With minimal four consecutive codes it is possible to determine the motor direction.
- ☐  $\pm$  Using the two signals C1 and C2 it is possible to determine the absolute motor position.

**Question 64.....Points: [2]**

Given following program to debounce a switch on a board. RawKeyPressed() returns 0 for if the switch is pressed. The function DebounceSwitch() gets called periodically every 3 ms.

```
bool DebounceSwitch(void) {
    static word16 State = 0;
    State = (State<<1) | !RawKeyPressed() | 0xF800;
    if (State==0xFC00) return TRUE;
    return FALSE;
}
```

Determine the switch release debouncing time in milliseconds which is used to debounce the switch.

64. 30 ms, or number of zero bits (1111'1100'0

**Solution:**

```
1111'1000'0000'0000 FALSE (key not pressed)
1111'1000'0000'0001 FALSE (key pressed: !0 => 1 shifted in)
1111'1000'0000'0011 FALSE
1111'1000'0000'0111 FALSE
...
1111'1111'1111'1111 FALSE (key pressed stable for a long time)
Release (input 0)
1111'1101'1111'1110 FALSE
...
1111'1101'1010'0000 FALSE
...
1111'1010'0000'0000 FALSE
1111'1100'0000'0000 TRUE
```

**Question 65.....Points: [3]**

Processor Expert components are using the concept of *Method*, *Property* and *Event*. Give examples of this concept using the FreeRTOS component.

- (a) 3 FreeRTOS examples for *Methods*:

[1]

**Solution:** vTaskCreate(), vTaskStartScheduler(), vWaitUntil()

- (b) 3 FreeRTOS examples for *Properties*:

[1]

**Solution:** InitialStackSize, Taskname length, preemptive or non-preemptive

- (c) 3 FreeRTOS examples for *Events*:

[1]

**Solution:** onStackOverflow(), onTimerTick(), MemoryFailed()

**Question 66.....Points: [1]**

You connect a push button which is *not* debounced to interrupt input pin, using an external pull-up resistor. You configure the pin to generate an interrupt whenever there is a falling edge. What are the things you have to consider?

**Solution:** You have to disable immediately (as fast as possible) the interrupts for the input. Then you need to properly debounce the switch. Be prepared that you might have an interrupt flag pending, so you need to acknowledge the interrupt at the debouncing.

**Question 67.....Points: [1]**

Explain why there is the word 'quadrature' used to denote a quadrature encoder/decoder:

**Solution:** Because the signals consist of two sinus-like signals, which are 90 degrees out of phase (or a quadrant of a full circle).

**Question 68 ..... 2 Points (Bonus)**

You are implementing a state machine which performs a debouncing. You want to use the same state machine driver for different keys at the same, and depending on the application you want the state machine calling different things. What are you using to implement the *user/application defined* function calls which are evaluated at run time?

**Solution:** Function pointers.

Given the the FreeRTOS task scheduling pattern as shown in Figure 7 for the first 100 milliseconds. The RTOS is in preemptive scheduling mode. The RTOS IDLE task is running with priority `tskIDLE_PRIORITY`. The tasks T1 and T2 are implemented as following:

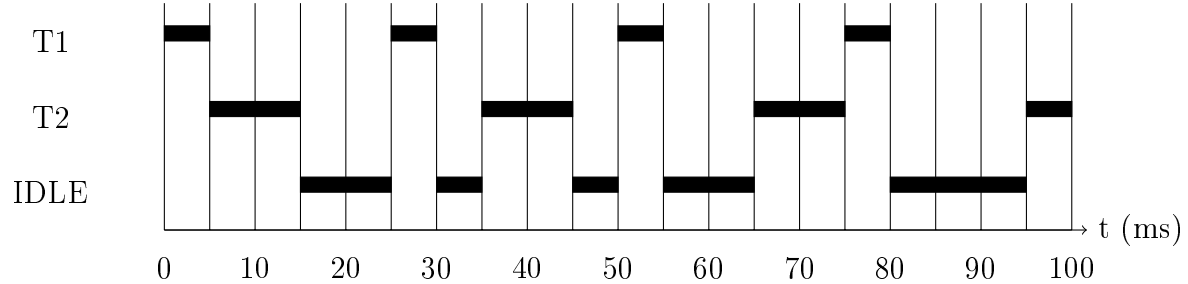
**Question 69.....Points: [8]**

Figure 7: FreeRTOS Task Schedule

```
static portTASK_FUNCTION(T1, pvParameters) {
    for (;;) {
        DoWorkT1();
        vTaskDelay(delayT1);
    } /* loop forever */
}

static portTASK_FUNCTION(T2, pvParameters) {
    for (;;) {
        DoWorkT2();
        vTaskDelay(delayT2);
    } /* loop forever */
}
```

(a) Determine the value of `delayT1` and `delayT2` in *milliseconds*: [2]

(a) delayT1: 20 ms, delayT2: 20 ms

(b) Each task does some work which is *not* using any blocking RTOS calls. Determine the time of work for `DoWorkT2()` and `DoWorkT1()` in *milliseconds*: [2]

(b) DoWorkT1: 5 ms, doWorkT2: 10 ms

(c) What is the tick timer period of the RTOS? Explain the reasoning behind your answer. [2]

**Solution:** Tick timer is 5 ms, as the delays are a multiple of it, and task switching happens from the IDLE task at 5 ms boundaries.

(d) Could it be possible that T1, T2 and IDLE share the same priority level? Justify your answer. [2]

**Solution:** Yes, this is possible, as within at least the first 100 ms there is no task running at the same time

**Question 70.....Points: [2]**

Given the following enumeration type:

```
typedef enum { RED, BLUE, GREEN, YELLOW, BLACK } ColorsEnum;
```

For the compiler we used in INTRO, the following applies for ANSI-C (with default compiler settings):

- ☐ sizeof(ColorsEnum)==1
- ☐ sizeof(ColorsEnum)==2
- ☒ sizeof(ColorsEnum)==4
- ☐ sizeof(ColorsEnum)==5
- ☒ sizeof(ColorsEnum)==sizeof(int)

**Question 71** ..... **Points: [4]**

Given following declaration of a trigger:

```
#define TRG_LAST      4
typedef void (*TRG_Callback)(void*);

typedef struct TriggerDesc {
    uint16_t triggerTicks;
    TRG_Callback callback;
} TriggerDesc;

static TriggerDesc TriggerList[TRG_LAST];
```

The triggers are initialized with following function:

```
void Init(void) {
    static volatile uint8_t i;

    for(i=sizeof(TriggerList)/sizeof(TriggerDesc); i>0; i--) {
        TriggerList[i].triggerTicks = 0;
        TriggerList[i].callback = NULL;
    }
}
```

(a) Evaluate the following statements:

[2]

- ☐  $\pm$  The function `Init()` will never finish.
- ☐  $\pm$  Every element of `Triggerlist` is initialized with with the same values.
- ☐  $\pm$  Using `static volatile` for the variable `i` ensures that the function `Init()` is reentrant.
- ☐  $\pm$  Using `volatile` for the variable `i` ensures that the function `Init()` is reentrant.

(b) Using above implementation, you notice that your application is working in a strange way, it even crashes after a while. Explain the reason:

[2]

**Solution:** The 'for' loop will iterate through index 4 to 1. But the array has the index 3 to 0. As such, the loop will write somewhere in memory, and part of the triggers are not initialized.

**Question 72.....Points: [5]**

Given following implementation of an event module:

```
#define GET_EVENT(event) \
    (bool)(EVNT_Events[(event)/8]&(0x80>>((uint8_t)((event)%8)))
#define CLR_EVENT(event) \
    EVNT_Events[(event)/8] &= ~(0x80>>((uint8_t)((event)%8)))

static uint8_t EVNT_Events[((EVNT_NOF_EVENTS-1)/8)+1];

bool EVNT_GetEvent(EVNT_Handle event) {
    bool isSet;
    EnterCritical();
    isSet = GET_EVENT(event);
    ExitCritical();
    return isSet;
}

void EVNT_ClearEvent(EVNT_Handle event) {
    EnterCritical();
    CLR_EVENT(event);
    ExitCritical();
}

uint8_t EVNT_CheckEvents(void) {
    uint8_t i;

    EnterCritical();
    for(i=0; i<sizeof(EVNT_Events)/sizeof(EVNT_Events[0]); i++) {
        if (EVNT_GetEvent(i)) {
            EVNT_ClearEvent(i);
            break;
        }
    }
    ExitCritical();
    return i; /* return the event which was set */
}
```

- (a) Using that implementation and calling `EVNT_CheckEvents()`, you find out that your periodic interrupt timer does not work any more. Explain the problem and how to fix it: [3]

**Solution:** The `Enter/ExitCritical()` are nested, thus it will disable the interrupts always. The solution is to use inside the `for()` loop the `GET_EVENT()` and `CLR_EVENT()` macros.

- (b) After you have fixed the previous problem, the application still does not work correctly. Somehow `EVNT_CheckEvents()` does not properly return events which have been set. Explain the problem and how to fix it: [2]

**Solution:** The `for()` loop is not properly iterating through the bit array. The loop should use a test on `i < EVNT_NOF_EVENTS` instead.

**Question 73.....Points: [2]**

Evaluate following statements in the context the RTOS startup, as used in the INTRO lab (FreeRTOS):

- ⊕ ± All interrupts are disabled during the (ANSI) startup code.
- ⊕ ± Global interrupts are enabled at the entry point of `main()`.
- ⊕ ± Interrupts for RTOS timers are disabled at the entry point of `main()`.
- ⊕ ± Interrupts for RTOS timers get enabled during startup of the scheduler.

**Question 74.....Points: [3]**

Explain why it is necessary to implement a timeout using the delta-time approach for measuring the speed with a quadrature encoder:

**Solution:** If there is no movement or the movement is very slow, then there would be no state change. As such, a timeout needs to be implemented in order to limit the delta time value.

**Question 75.....Points: [3]**

If you can choose between the delta-time and delta-pos method for quadrature decoding for fast (many steps) state changes, which one would you prefer and why?

**Solution:** The delta-pos approach, as there would be sufficient position changes in a small amount of time.

**Question 76.....Points: [3]**

If you can choose between the delta-time and delta-pos method for quadrature decoding for slow (few steps) state changes, which one would you prefer and why?

**Solution:** The delta-time approach, measuring the time for one or few steps. There would not be enough or now step for a given (short) time to count.

**Question 77.....Points: [3]**

Explain why a tacho implementation as we have implemented is not an exact measurement, but rather a speed estimation?

**Solution:** The sensor values are always lagging (behind in time, backward looking), the encoder itself has errors, there could be a drift between wheels and ground, and there are quantization errors.

**Question 78.....Points: [3]**

Name a data structure which can be used to efficiently implement a delta-pos algorithm for estimating a tachometer.

**Solution:** A ring buffer can be used for this: new values would overwrite the oldest entries.

**Question 79.....Points: [3]**

What is the difference between a Mutex and a normal semaphore in FreeRTOS?

**Solution:** The Mutex is a special case of a semaphore. A Mutex is used to grant exclusive access to a shared resource, while semaphores can be used for inter-process communication. The Mutex implements Priority Inheritance, which means that a process waiting on the Mutex inherits the priority of the process holding that Mutex. A Mutex has to be always returned, while for a semaphore this is not required.

**Question 80.....Points: [3]**

How many processes and priorities are needed to demonstrate a Priority Inversion?

**Solution:** At least three processes with different priorities are needed for a Priority Inversion. One process with low priority to hold the semaphore/resource, one process with medium priority which blocks the low priority process, and a process with high priority which waits for the semaphore/resource.

**Question 81.....Points: [3]**

Given the two Semaphores S1 and S2 and the three processes PL (high priority (priority 3)), PM (medium priority (priority 2)) and PL (low priority (priority 1)). Semaphore S1 is used by PL and PM, S2 is used by PL and PH. Determine the *Priority Ceiling* of S1 and S2.

**Solution:** S1 has a Priority Ceiling of 2, S2 has a Priority Ceiling of 3.

**Question 82.....Points: [3]**

If you use a Semaphore in FreeRTOS from an interrupt service routine (ISR), is there something special to consider?

**Solution:** To use RTOS API calls within an ISR, the API routines with a *fromISR* suffix has to be used, for example `xSemaphoreTakeFromISR()` and `xSemaphoreGiveFromISR()`.

**Question 83.....Points: [1]**

In which state is a process in FreeRTOS when it is waiting on a Semaphore?



**Solution:** It is in the *Blocked* state.

**Question 84.....Points: [3]**

Using a network stack like RNet, which things do you have to initialize?

**Solution:** Call the stack initialization routine (this initializes the low level driver), assign application message handler, assign own node address

**Question 85.....Points: [2]**

RNet uses four different layers in the stack. List the layers with their acronyms.

**Solution:** PHY (Physical Layer), MAC (Media Access Layer), NWK (Network Layer), APP (Application Layer)

**Question 86.....Points: [1]**

What is a CRC and for what it is used for?

**Solution:** CRC (Cyclic Redundancy Check) implements a check-sum to detect errors in data. In communication systems it is used to check the correctness of data packets transmitted.

**Question 87.....Points: [1]**

What is a so called *Ad-Hoc Network*?

**Solution:** A network with two or more nodes which does not need a centralized or fixed network service infrastructure. This differentiates it from other networks like a Wireless LAN with an access point.

**Question 88.....Points: [2]**

List pros and cons of an *Ad-Hoc Network*.

**Solution:** Pros: No central infrastructure required, network can be built 'on the fly' and on demand. No need for an access point or network coordinator which needs to run all the time. Cons: Resources and energy needed to run the access point or network coordinator. Network might not be scalable or suitable for large networks. No administration of the network.

**Question 89.....Points: [2]**

List the pins of the HC-SR04 Ultrasonic distance sensor and for what they are used.

**Solution:**

1. Vdd, 5V supply voltage
2. GND, Ground
3. Trig:  $10\mu$  input pulse signal to start the measurement
4. Echo: output signal where the pulse width is proportional to the distance measured (time of echo for reaching the object and back again).

**Question 90.....Points: [2]**

What happens if the HC-SR04 Ultrasonic distance sensor does not measure an echo (no object detected)?

**Solution:** The Echo signal will be high for a long time, causing a timer overflow. This can then be detected in the driver for 'no object in range'.

**Question 91.....Points: [2]**

Using the HC-SR04 Ultrasonic distance sensor, you are using for the speed of sound 58 cm/ $\mu$ . Why is the result not correct? Are there other factors influencing the result?

**Solution:** The time measured with the *Echo* signal is for the sound traveling to the object and back again. As such, the result needs to be divided by two. Other factors are air density, humidity and temperature.

**Question 92.....Points: [1]**

What is the purpose of a radio transceiver?

**Solution:** A radio transceiver is used for over-the-air communication. The transceiver transforms the data packets suitable for physical communication channel, encodes and decodes them.

**Question 93.....Points: [1]**

Name three software components/layers of the RNet communication stack we used.

**Solution:** PHY (physical), MAC (media access), NWK (network), APP (application).

**Question 94.....Points: [1]**

What does *Payload Packaging* mean?

**Solution:** With Payload Packaging each layer of a network stack can add/remove additional data at the beginning and/or end of the payload of the higher/lower network layers.

**Question 95.....Points: [1]**

You need to drive a H-Bridge IC with a PWM signal. What fundamental parameter you have to check in the data sheet of the IC?

**Solution:** The maximum/minimum PWM frequency you can use. Each IC has its own specific range.

**Question 96.....Points: [1]**

How can the power dissipation of the H-Bridge for of a PWM controlled device be reduced?

**Solution:** In general, the data sheet should provide information about the power dissipation at given frequencies. Typically higher PWM frequencies will cause higher dissipation due the switching activity, so a frequency in MHz range does not make any sense. Many motor controllers use a frequency above 20 kHz for quieter operation.

**Question 97.....Points: [1]**

If a closed control loop controller is using a PWM as output signal, what should be considered?

**Solution:** Depending on the PWM hardware, the PWM duty should be changed as such that a current running PWM duty cycle is not affected. Ideally the PWM duty cycle is changed at the end of a PWM period.

**Question 98.....Points: [1]**

What is the common purpose of using a Queue or a Semaphore/Mutex?

**Solution:** To ensure that access to data is done in a controlled way without race conditions or undesired side effects.

**Question 99.....Points: [1]**

Lock() and Unlock() are functions used for which programming primitive or design pattern?

**Solution:** For Semaphores or Mutex.

**Question 100.....Points: [1]**

The speed of a motor is proportional to the voltage applied to the motor. So why is it necessary to use a closed control loop?

**Solution:** To deal with different load, mechanical differences and external influences.

**Question 101.....Points: [2]**

You have to implement a position controller (closed loop)? Which type do you choose: PD or PI?

**Solution:** PD. The system already has an integral part in it and integrating the position error would either be a very large amount or would need to be scaled down so largely that it does not make much sense. As such, the PD would be sufficient.

**Question 102.....Points: [2]**

For our Sumo robot, identify the following parts of a closed loop control system: Controller, Actuator, Sensor, System.

**Solution:** Controller: Microcontroller with PID control loop implementation. Actuator: H-Bridge. Sensor: quadrature decoder (speed/position), System: Sumo Robot motors

**Question 103.....Points: [2]**

What is the purpose of the *Anti-Windup*?

**Solution:** To limit the total sum and impact of the integral value.

**Question 104.....Points: [2]**

Why is it better to use 'industrial qualified' SD cards instead of using 'consumer' ones?

**Solution:** The industrial ones usually are more robust and have well defined parameters or data sheet information. Consumer type cards might very differ in access time, wear out and programming time.

**Question 105.....Points: [1]**

Which fundamental values have to be calibrated for an accelerometer sensor as we have used it?

**Solution:** Offset and gain. Typically you need known values e.g. for 0g and 1g acceleration.

**Question 106.....Points: [1]**

Provide an implementation way where you can store calibration data?

**Solution:** Calibration data can be stored for example in the microcontroller internal flash, either at compilation time (constants in code) or at runtime with programming the FLASH memory.

**Question 107.....Points: [3]**

What do you have to consider if you are programming the internal FLASH memory at run time?

**Solution:** FLASH is typically organized in pages and blocks, and programming is done on page or block level. You need to make sure that you separate your program from the data. Interrupts typically need to be disabled during flash programming. Typically the program memory is not accessible during flash programming, so the flash programming 'applet' needs to run in RAM.

**Question 108.....Points: [1]**

Why did we use a digital interface to the reflectance sensor and not an analog one?

**Solution:** Digital pins were easily available, while analog pins are usually limited on a microcontroller.

**Question 109.....Points: [1]**

Even if all sensors of the reflectance sensor are exposed to the same conditions, the measurement will differ from sensor to sensor. Why?

**Solution:** The resistors and the capacitors used have their own tolerances, resulting in different measurements. This results in different discharging time.

**Question 110.....Points: [1]**

The reflectance sensor array has two red LED's. What's the purpose of it?

**Solution:** To visually indicate if the sensors are used.

**Question 111.....Points: [1]**

How could you modify the reflectance sensor array hardware to minimize the power consumption?

**Solution:** Removing the red LED's or the current limiting resistor for each red LED.

**Question 112.....Points: [3]**

List typical external errors for the reflectance sensor array we have used, and how the errors can be reduced or limited.

**Solution:** Sun light, room ambient light, light from other infrared LED's. Errors can be minimized with proper sensor placement and distance to the ground, using shielding, multiple reads of sensors (averaging, etc). Calibrate the sensors for the given environment.

**Question 113.....Points: [3]**

List several functions or macros which are used in FreeRTOS to control task execution.

**Solution:**

- `uxTaskPriorityGet/Set()`
- `vTaskSuspend/SuspendAll()`
- `vTaskResume()`, `vTaskResumeAll()`, `vTaskResumeFromISR()`,
- `taskYIELD()`
- `taskENABLE_INTERRUPTS()`, `taskDISABLE_INTERRUPTS()`
- `taskENTER_CRITICAL()`, `taskEXIT_CRITICAL()`
- `vTaskDelay()`, `vTaskDelayUntil()`

**Question 114.....Points: [1]**

If several tasks are pending, which task gets executed first by a priority-based preemptive scheduler?

**Solution:** The one with the highest priority.

**Question 115.....Points: [1]**

Why could be a call to `vTaskEndScheduler()` make sense?

**Solution:** If only for a certain time an RTOS is needed in an application. For example an application only needs to run multiple tasks and uses the RTOS services in a special mode, but otherwise the application does not want the runtime overhead (e.g. for going into a very deep low power mode). On the other hand, the application could suspend the scheduler too.

**Question 116.....Points: [1]**

List the four memory allocation schemes in FreeRTOS.

**Solution:**

1. Scheme 1: only allocation, no deallocation.

2. Scheme 2: blocks are deallocated, but not merged.
3. Scheme 3: wrapper to `malloc()` and `free()`.
4. Scheme 4: same as Scheme 3, but merges free blocks.

**Question 117.....Points: [1]**

What was the purpose of the Shell queue we have implemented in the INTRO application?

**Solution:** The queue is used to pass data from multiple tasks to the Shell task. That way the other tasks send the data through the queue, instead of directly accessing the UART resource. So the queue is used as synchronization design pattern.

**Question 118.....Points: [1]**

What are the two different categories of timeliness?

**Solution:** Absolute and relative.

**Question 119.....Points: [1]**

List three different clock sources for a microcontroller:

**Solution:** External crystal, external oscillator, internal reference clock.

**Question 120.....Points: [1]**

In a microcontroller you can have different kinds of clocks, list at least two:

**Solution:** Memory Bus clock, CPU clock, Peripheral clock (A/D, USB, ...).

**Question 121.....Points: [1]**

List a fundamental requirement for a realtime operating system:

**Solution:** Exact time base.

**Question 122.....Points: [1]**

List pros and cons of using an internal reference clock (compared to an external clock generator):

**Solution:** Pros: less parts or PCB components, simple, less expensive. Cons: less accurate, may need calibration, uses more power (not ideal for low power applications).

**Question 123.....Points: [3]**

List reasons why to use an RTOS:

**Solution:** Running multiple things easily in parallel, scalability of the application, using RTOS services (queues, semaphore, mutex, memory allocation, events, ...)

**Question 124.....Points: [2]**

What is the difference between preemptive and non-preemptive scheduling:

**Solution:** In a preemptive operating system the RTOS distributes the processing time, tasks get suspended by the RTOS. In a non-preemptive (cooperative) operating system the tasks are passing actively the control back to the operating system.

**Question 125.....Points: [2]**

List the three standard I/O channels used by the Shell:

**Solution:** stdin, stdout, stderr

**Question 126.....Points: [1]**

List the four fundamental states of a task in an operating system:

**Solution:** Run, Idle, Wait, Ready

**Question 127.....Points: [1]**

What does the abbreviation *HAL* mean?

**Solution:** Hardware Abstraction Layer: an interface or similar in software to abstract from the actual hardware used.

**Question 128.....Points: [2]**

What is the difference between two different vendors offering a microcontroller based on an ARM Cortex M0+?

**Solution:** The processor core is the same (instruction set, registers), although it can be configured by the vendor in some details (e.g. number of interrupt levels). But everything around the core (e.g peripherals like USB, CAN, UART, ...) are vendor specific.

**Question 129.....Points: [2]**

How many hardware stack pointers implements an ARM Cortex M0+ microcontroller?

**Solution:** Two (Main SP and Process SP).

**Question 130.....Points: [2]**

The ARM Cortex M0+ architecture has 16 32bit registers. Are they all freely usable by the application code?



**Solution:** No, only R0 - R12. Other registers are used by the hardware R13  $\Rightarrow$  SP, R14  $\Rightarrow$  LR, R15  $\Rightarrow$  PC R16  $\Rightarrow$  xPSR

**Question 131.....Points: [2]**  
Which Processor Expert generated macros can you use to protect a critical section?

**Solution:** EnterCritical() and ExitCritical()

**Question 132.....Points: [1]**  
What could a reason to use the `volatile` qualifier for a loop variable?

**Solution:** To prevent compiler optimizations.

**Question 133.....Points: [1]**  
What is the advantage of using *Gadfly Synchronization* compared to *Interrupt Synchronization*?

**Solution:** Less overhead with interrupt latency, simpler program flow, better if only to wait for a very short time.

**Question 134.....Points: [1]**  
What does this mean: *an instruction is atomic*?

**Solution:** It means that an instruction is always executed as a whole, that means it cannot be interrupted and in between something different can be executed.

**Question 135.....Points: [1]**  
What is the fundamental problem of 'shared data' in the context of interrupts?

**Solution:** An interrupt can happen any time. As such, if shared data is accessed from the main application and the interrupts, then access to the shared data could be interrupted too. As such, inconsistent data or race conditions could happen.

**Question 136.....Points: [2]**  
The implementation of EnterCritical() and ExitCritical() as we used it does not allow nesting. Why?

**Solution:** The macros are using a global variable to store the status of the interrupts (CCR register). If nested, then that variable gets overwritten, causing wrong behavior of the application.

**Question 137.....Points: [2]**  
Given the following source code:

```
#define CALC1 (2+5)
#define CALC2 (5*3)
```

Why is it important to use parenthesis?

**Solution:** The preprocessor does a textual replacement. Without the parenthesis it is possible that after the preprocessing the output is non-intentional. For example `10*CALC1` would result into `10*2+5`.

**Question 138**.....**Points: [2]**

List some advantages and disadvantages of using macros:

**Solution:** Pros: calculations at compilation time (faster, denser code), using it for configuration and compile time configuration. Cons: only textual replacement, overuse can make code harder to read, no type safety, debugging macros is a problem.

**Question 139**.....**Points: [2]**

You need to write a header file `LED.h`. Write it in such a way so *recursive* includes are not possible.

**Solution:**

```
#ifndef __LED_H_
#define __LED_H_
/* content of the header *.h file */
#endif
```

**Question 140**.....**Points: [2]**

List additional benefits of Doxygen beside of normal source documentation:

**Solution:** Dependency graphs (include and source file dependencies), message sequence diagrams, integration of external images.

**Question 141**.....**Points: [1]**

You need to use a variable in multiple modules (.c files). Where do you put the *declaration* of it? Make an example.

**Solution:** The declaration needs to be in an interface header (\*.h) file. E.g. `extern int MOT_currPWM;`. The name should be as such that there are no name collisions. Common practice is to use a prefix like `MOT_`.

**Question 142.....Points: [1]**

What is the fundamental difference between the two VCS: Git and SubVersion?

**Solution:** Git is a distributed VCS, while SubVersion is a centralized VCS.

**Question 143.....Points: [1]**

Two engineers are using Git as VCS. Now they change both the same line of code, and both perform a *commit*. What happens?

**Solution:** Nothing at the time of the commit. But at the time of *push*, the system will report a conflict which needs to be resolved manually.

**Question 144.....Points: [3]**

An engineer is generating the documentation with doxygen from the source files. He decides to put both the documentation and the source files into a VCS. Discuss this approach:

**Solution:** It is not recommended to put derived files (like generated files from source code) into a version control system, as the doxygen documentation can be re-generated from the source files. Putting derived files into a repository creates redundancies and increases the repository size, and increases the amount of communication with the repository.

**Question 145.....Points: [3]**

Below is the content of a .gitignore file. Explain what this file does with such a content:

```
*
!*.c
!*.h
!*.gitignore
!dev/
```

**Solution:** In the .gitignore the list of files and extensions tells Git which files to ignore. Git builds a list of files and goes from the beginning to the end of the file. The first line ignores all files. Then it removes (! operator) all \*.c and \*.h files, including the gitignore file. Then it removes all files in the dev folder. Using the ! operator gitignore explicitly specifies the files *not* to ignore.

**Question 146.....Points: [3]**

For multiple Eclipse projects, you want to use source files in a common folder (e.g. c:/mySources/common) which is *outside* of your Eclipse project. You consider to use 'Virtual Group', 'Linked Folder' or 'Linked File' methods to use the common file(s) in your project. Now you have added a new source file (e.g. accelerometer.c) to

that common folder. For each of the three above methods, explain what you have to do for *each of your eclipse projects* which want to use the new `accelerometer.c` file.

**Solution:**

- Virtual Group: need to add another link in the group
- Linked Folder: nothing, it will be automatically added
- Linked Files: need to add another link

**Question 147.....Points: [3]**

Given a Quadrature encoder as in Figure 8, generating two signals C0 and C1. Both signals get sampled with a periodic timer interrupt. The disk is turning with a maximum frequency of 100 Hz. Determine the minimal sampling frequency in Hz to guarantee an error free sampling of the two signals C0 and C1.

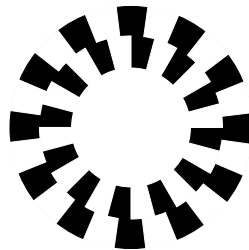


Figure 8: Quad Disk

**Solution:**  $(12 \cdot 4 \cdot 100 \text{ Hz} \cdot 2) = 9600 \text{ Hz}$ .

1p: 12 sectors

1p: 4 signals each sector

1p: Shanon/Nyquist

**Question 148.....Points: [3]**

Draw the resulting graph produced by doxygen for following DOT code:

```
\dot
digraph example_dot_graph {
    node [shape=triangle];
    rankdir=LR;
    A    [label="A"];
    B    [label="B"];
    C    [label="C"];
    A -> A [label="a"];
    A -> B -> C -> A -> C;
    B -> B [label="b"];
}
```

**Solution:**

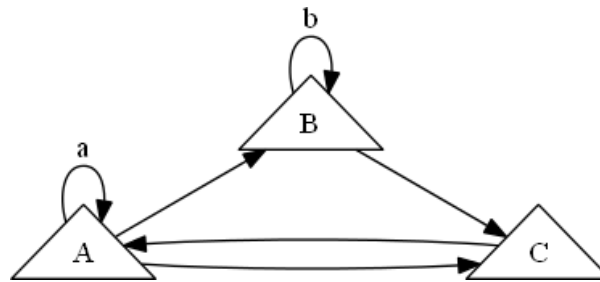


Figure 9: Doxygen Triangle

**Question 149.....Points: [3]**

You are using a wireless transceiver as used in the lab to transmit sensor data values to another system. Describe strategies to reduce the energy consumption for the communication. Limit your answer to the *communication* and *transceiver* only.

**Solution:** Data packets as small as possible, sending as few as possible. Using sleep mode of the transceiver.

**Question 150.....Points: [3]**

Calculate for the 16bit **Binary Code**  $0x1722_b$  the corresponding 16bit (binary reflected) Gray code<sub>g</sub>:

**Solution:** Binary to Gray:  
 $0x1722 \text{ EXOR } (0x1722 \gg 1)$   
 $= 0x1722 \text{ EXOR } 0x0B91 = \mathbf{0x1CB3_b}$  (or  $\mathbf{0b1'1100'1011'0011}$ )

**Question 151.....Points: [3]**

In Figure 10 are steps transforming a 6bit Gray Code into a 6bit Binary Code. Fill in the gaps:

**Question 152.....Points: [8]**

Below is the implementation of a task which reads and writes to an SCI (Serial Communication Interface). The SCI is using interrupts for RX and TX. As other tasks are using the `Read()` and `Write()` functions as well, it is necessary to use critical sections.

```
void Read(char *buf, size_t bufSize) {
    /* start critical section */
    SCI_Read(buf, bufSize);
    /* end critical section */
}

void Write(char *buf) {
    /* start critical section */
    SCI_Write(&buf[0]);
}
```

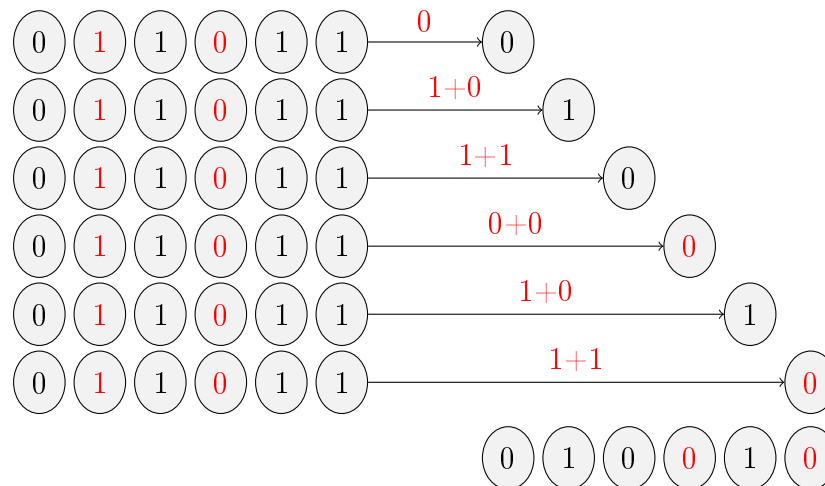


Figure 10: 6bit Gray Code

```

/* end critical section */
}

static portTASK_FUNCTION(Task1, pvParameters) {
    unsigned char buf[16];
    (void)pvParameters; /* parameter not used */
    for (;;) {
        /* start critical section */
        Read(buf, sizeof(buf));
        Write(buf, sizeof(buf));
        /* end critical section */
    }
}

```

You identified 3 different ways for implementing the `/* start critical section */` and `/* end critical section */` in above source. Evaluate following statements for the consequences of each implementation for the above source:

- (a) With using `DisableAllInterrupts()` and `EnableAllInterrupts()` the following applies: [2]
- ⊕ ± It increases the interrupt latency time.
  - ⊕ ± Is the best solution with respect to RAM/ROM footprint.
  - ⊕ ± Makes the usage of `SCI_Read()` and `SCI_Write()` reentrant.
  - ⊖ ± Allows the scheduler to run inside every critical section.
- (b) With using FreeRTOS *binary semaphore/mutex* the following applies: [2]
- ⊕ ± Calling the semaphore/mutex API might trigger a context switch.
  - ⊖ ± No context switch will happen within the critical section.
  - ⊕ ± Only one task at a time will be inside the critical section.
  - ⊕ ± Interrupts will occur inside the critical section.
- (c) With using FreeRTOS *recursive semaphore/mutex* the following applies: [2]

$\oplus \pm$  Calling the semaphore/mutex API might trigger a context switch.

$\ominus \pm$  Multiple tasks might be within the same critical section.

$\ominus \pm$  No context switch will happen within the critical section.

$\ominus \pm$  No interrupts will occur inside the critical section.

(d) From the 3 solutions a), b) and c), select the one you consider as the best one for above source, and explain briefly why: [2]

**Solution:** recursive semaphore/mutex.

**Question 153.....Points: [4]**

A system using *Priority Ceiling* has four tasks  $J_1, J_2, J_3$  and  $J_4$ . The task index denotes the task priority with 4 as the highest priority. Additionally four semaphores  $S_1, S_2, S_3$  and  $S_4$  are used. Task are getting a lock of a semaphore  $x$  using  $LS_x$ , the lock of a semaphore is released again with  $US_x$ . The task and semaphores are used according to following schedule:

- $J_1 = \{LS_4, US_4\}$
- $J_2 = \{LS_2, LS_1, US_1, US_2, LS_3, US_3\}$
- $J_3 = \{LS_1, US_1, LS_3, US_3\}$
- $J_4 = \{LS_1, US_1\}$

Determine the *Priority Ceiling* for each semaphore:

*Priority Ceiling* for  $S_1$ :

153. \_\_\_\_\_ 4

*Priority Ceiling* for  $S_2$ :

153. \_\_\_\_\_ 2

*Priority Ceiling* for  $S_3$ :

153. \_\_\_\_\_ 3

*Priority Ceiling* for  $S_4$ :

153. \_\_\_\_\_ 1

**Question 154.....Points: [3]**

Below is a way to prevent spurious keyboard interrupts on a microcontroller:

```
PTADD = 0x00; /* set port as input */
PTAD = 0xFF; /* write 1's to data port to have defined input values */
PTAPE = 0xFF; /* enable pull ups */
EnableInterrupts;
```

Transform the above code into an implementation using *Gadfly* synchronization:

**Solution:**

```
PTADD = 0x00; /* set port as input */  
PTAPE = 0xFF; /* enable pull ups */  
while (PTAD!=0xFF); /* wait until port pull ups are working */  
EnableInterrupts;
```



**Question 155.....Points: [6]**

A preemptive system has 3 tasks  $T_1$ ,  $T_2$  and  $T_3$  where the task index denotes the task priority, with 1 the lowest priority. The system is using *Priority Inheritance* for the semaphores. Create a timing diagram with the tasks in Figure 11 indicating the execution time for each task. The system has following timing:

- $t_0$ :  $T_1$  and  $T_2$  are ready,  $T_3$  is running,
- $t_1$ : The running task gets suspended
- $t_3$ : The running task gets suspended
- $t_4$ : The running task requests the semaphore
- $t_5$ :  $T_2$  and  $T_3$  get ready
- $t_7$ : The running task requests the semaphore
- $t_8$ : The task having the semaphore releases the semaphore
- $t_9$ : The task having the semaphore releases the semaphore
- $t_{10}$ : The running task terminates
- $t_{11}$ : The running task terminates
- $t_{12}$ : The running task terminates

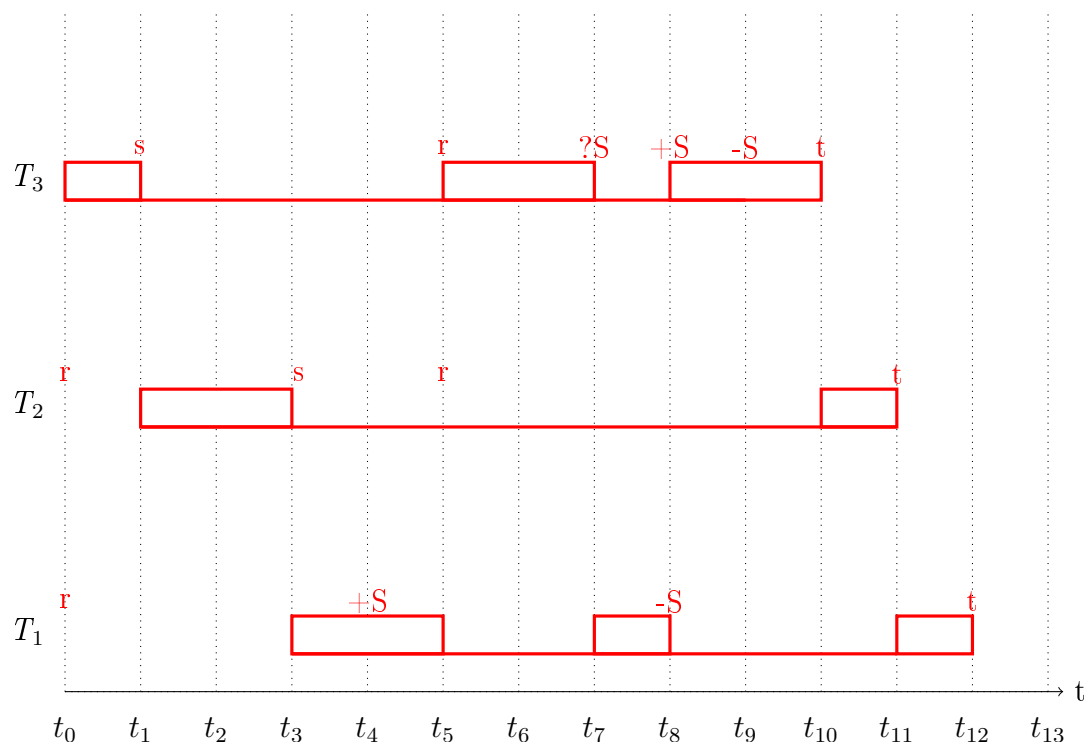


Figure 11: Tasks

**Question 156.....Points: [3]**

Explain for what the *Anti-Windup* is used in a PID control system:

**Solution:** To limit the impact of the integral term in the PID.

**Question 157.....Points: [3]**

Identify the three serious problems in the following source:

```
#define QUEUE_LENGTH 5
#define QUEUE_ITEM_SIZE sizeof(char_t*)

void QUEUE_SendMessage(const char *msg) {
    char *ptr = FRTOS1_pvPortMalloc(UTIL1_strlen(msg));
    UTIL1_strcpy(ptr, msg);
    if (FRTOS1_xQueueSendToBack(queueHandle, ptr, portMAX_DELAY)!=pdPASS)
    {
        for(;;) {} /* ups? */
    }
}
```

**Solution:** It should be `UTIL1_strlen(msg)+1`, it does not check for `malloc()` failure, it does not pass the address of the pointer

**Question 158.....Points: [8]**

Given a ultrasonic sensor. It has a TRIG input signal and an ECHO output signal. A microcontroller sets the TRIG signal HIGH for 1 ms. Then after some time, it can measure the duration of the ECHO pulse signal which is proportional to the distance measured (Figure 12).

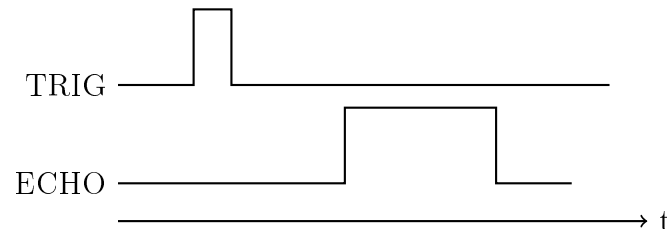


Figure 12: Ultrasonic Module Timing

- (a) Implement the generation of the TRIG signal using a *Realtime* synchronization method: [4]

**Solution:**

```
void Trig(void) {
    long i;
    SetTrigHigh();
    for (i=0;i<100000;i++) { asm("nop"); } /* wait for 1 ms */
    SetTrigLow();
}
```

- (b) After the TRIG signal generation, the microcontroller has to synchronize to the beginning of the ECHO signal. Implement this using a *Gadfly* synchronization: [4]

**Solution:**

```
void WaitEcho(void) {
    while (EchoIsLow()) {}
}
```

**Question 159 ..... Points: [10]**

FreeRTOS is used in priority based preemptive mode. Tick timer is set up for 5 ms, and the processor is running at maximum speed. Task T3 has priority 3 (highest priority), task T2 has priority 2 and task T1 has priority 1. The tasks have been created with `xTaskCreate()` before time  $t=0$  ms, and the scheduler is started with `vTaskStartScheduler()` at the time  $t=0$  ms. Draw in Figure 13 a timing diagram for the execution of the two tasks for the first 100 ms (after  $t_0 = 0$  ms). Use a bar to indicate when a task is running. You can ignore the overhead in the task loop and the overhead in the RTOS/scheduler itself.

```

static portTASK_FUNCTION(T3, pvParameters) { /* priority 3 task */
    portTickType xLastTime = xTaskGetTickCount();
    for (;;) { /* task time is 7 ms including overhead */
        DoWorkFor7ms(); /* this needs 7 ms */
        vTaskDelayUntil(&xLastTime, 30/portTICK_RATE_MS);
    } /* loop forever */
}

static portTASK_FUNCTION(T2, pvParameters) { /* priority 2 task */
    for (;;) { /* task time is 7 ms including overhead */
        DoWorkFor7ms(); /* this takes 7 ms */
        vTaskDelay(30/portTICK_RATE_MS);
    } /* loop forever */
}

static portTASK_FUNCTION(T1, pvParameters) { /* priority 1 task */
    for (;;) { /* task time is 4 ms including overhead */
        DoWorkFor4ms(); /* this needs 4 ms */
        vTaskDelay(28/portTICK_RATE_MS);
    } /* loop forever */
}

```

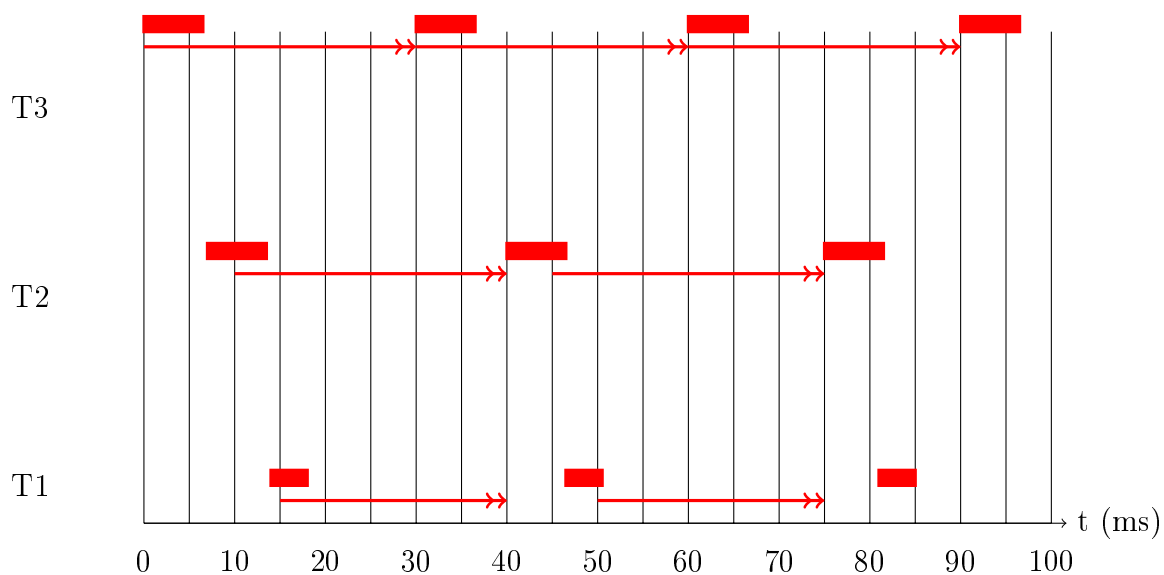


Figure 13: Task Timing

**Question 160.....Points: [3]**

You are working with an Eclipse based project (you can consider the one we used for INTRO). Which kind of files are you going to put into a VCS? Which ones not? Explain briefly why.

**Solution:** Put all files into the repository which are needed to build the product/project (source files, project settings, configuration files), but do not put any

derived or generated resources into it (generated documentation, object files, ELF file(s), generated make files, log files). The derived files can be re-generated, and putting them into VCS will just be duplication of content/consumes a lot of space.

**Question 161 ..... 2 Points (Bonus)**

Three men: a project manager, a software engineer, and a hardware engineer are helping out on a project. About midweek they decide to walk up and down the beach during their lunch hour. Halfway up the beach, they stumbled upon a lamp. As they rub the lamp a genie appears and says "Normally I would grant you three wishes, but since there are three of you, I will grant you each one wish."

The hardware engineer went first. "I would like to spend the rest of my life living in a huge house in Hawaii." The genie granted him his wish and sent him on off to Hawaii.

The software engineer went next. "I would like to spend the rest of my life living on a huge yacht cruising the Mediterranean." The genie granted him his wish and sent him off to the Mediterranean.

Last, but not least, it was the project manager's turn. "And what would your wish be?" asked the genie. The project manager replied:

- ☐ "I would like an expensive Ferrari car."
- ☐ "I would like to win the lottery so I have no money worries."
- ☐ "Send me to the hardware engineer to Hawaii"
- ☐ "Send me to the software engineer on the Mediterranean."
- ✓ "I want them both back after lunch."