1 Essä 2 poäng Embedded Systems

Give a short description of the term "embedded system" as used during the course. What makes an embedded system different from a general computing system such as a desktop computer or a server?

2 Essä 3 poäng Constraints

During the lecture, we have discussed multiple constraints when it comes to designing and developing embedded systems. Name 3 of them, and explain, how they affect each other.

3 Essä 3 poäng Button debouncing

Quite often when we work with embedded systems, we have to deal with signals transmitted through wires and physical connections. A common problem in those cases is, that whenever these signals change, they cannot do so instantaneously, which often leads to a short period of time, in which the signal is not stable. These problems appear especially when dealing with buttons and switches in the system. What do we mean by the term "debouncing"? Explain two methods of how we can achieve debouncing when dealing with buttons in our system.

4 Essä 4 poäng RTOS vs no RTOS

When developing embedded software, one often has to make the decision wether to use a realtime operating system (RTOS), or to just write the firmware on top of the vendor specific SDK for the hardware chip used. Compare the two different methods of embedded software development (i.e. "bare-metal" vs RTOS), what advantages and disadvantages do they have. Give two examples, one where using an RTOS is favourable, and one, where not using an RTOS might be advantageous. Outline shortly for each example, why you think that the chosen design is the better fit for the respective problem.

5 Essä 2 poäng Threads/Tasks

One concept, that is very important when using a realtime operating system (RTOS), are threads or tasks (as said during the lecture, for our purposes here these two terms can be seen as synonyms). In your own words, describe what a task/thread is in the context of an RTOS. Give an example of a system (you can be creative) where you would have two independent tasks running on the same microcontroller.

6 Essä 4 poäng Thread states

In the context of a realtime operating system (RTOS), threads/tasks can be in different states. Different RTOSs define different states, but most of them have at least the following:

- Ready
- Running
- Waiting/Blocked
- Terminated

Give a short description of each state, what it is used for, and how we go from and to this state.

7 Essä 2 poäng Thread priorities

Realtime operating systems often use the concept of thread/task priorities. Give a short description of what a priority level means when we talk about preemptive scheduling. What problems can we actually solve by using priorities?

8 Essä 2 poäng Preemptive and non-preemptive scheduler

Many realtime operating systems offer different scheduling algorithms to schedule the tasks/threads defined in your system. Two very common ones are preemptive and non-preemptive priority based scheduling. What is the difference between those two, and why would we sometimes want to favour one over the other and vice versa?

9 Essä 4 poäng Shared memory pitfall

When talking about inter-task communication in the context of a real-time operating system, which problem can arise by using shared memory (i.e. a global variable or data structure on the heap)? Outline a short example (either textually or with a short snippet of code), that showcases this problem.

Common strategies for solving the issue with shared memory are the following:

- Using cooperative (i.e. non-preemptive) tasks
- Locking interrupts before accessing shared memory

Give a short description of each method, and explain how they solve the aforementioned issue.
(You might be wondering why using a mutex is not mentioned as a method here. There is a separate question about mutexes in this exam, so you don't have to mention them here)

10 Essä 4 poäng Mutex

In the context of realtime operating systems, what do we mean by the term "mutex"? Which problems does a mutex solve?

When using a mutex, two common problems might occur:

- Priority inversion
- Deadly embrace

Give a short example (in C code or textually) for both of these problematic cases, and explain what the problem is and how we can solve it. I will not try to compile your C code, so it does not have to be syntactically fully correct. If you want, you can use the Zephyr API we used during the course, otherwise you can use generic API calls like mutex_lock( ) and mutex_unlock( ).

11 Essä 2 poäng Semaphore and Mutex

Give a short description of the term "semaphore" in the context of realtime operating systems. What are they used for? What is the difference between a mutex and a semaphore? In which use case scenario would you prefer to use a mutex, and in which would you use a semaphore?

12 Essä 4 poäng Reentrant function

In the context of realtime operating systems, (RTOS) what is a reentrant function? What criteria must a function fulfil to be reentrant?
Why are reentrant functions important when designing a system using an RTOS?

Give two examples (in C code) for a reentrant and a non-reentrant function. Your C code does not have to be fully syntactically correct, I will not try and compile your functions.

13 Essä 1 poäng Device tree

When using a realtime operating system such as Zephyr, one integral part is the device tree. What is the device tree used for, which problems does it solve?

14 Essä 2 poäng Deferred interrupt handling

In the context of realtime operating systems, what do we mean by the term "deferred interrupt handling"? Why is it a good design principle?
Give a short example of a system that might make use of deferred interrupt handling. You can do so textually or with a short C code snippet.

15 Essä 2 poäng Volatile

One C keyword, which is very common when programming embedded systems is "volatile". What does volatile mean, and why do we want to use it? Give a short C code example, that showcases the use of volatile, and where not using volatile would most likely result in a different functionality of the system.

16 Essä 2 poäng Memory

In a typical microcontroller, which two types of memory are present? What are these different types of memory used for?

17 Essä 1 poäng Memory management

Memory can be managed in different ways. Most commonly we distinguish between allocating memory at compile time, and allocating memory dynamically during runtime. Many coding standards prohibit (or at least strongly discourage) dynamic memory management for safety critical embedded systems. Why?

18 Essä 3 poäng Time space tradeoff

When talking about the efficiency of embedded software, very often we can either pick a solution that is very space efficient, but will increase the execution time, or a solution that favours fast code execution, but might require more memory. Give three examples of such cases, and quickly describe how/why they favour either space or time efficiency.

19 Essä 4 poäng Pre and post conditions

As you have seen during the first flipped classroom lecture, we can use pre and post conditions as a form of contract based programming, where we establish, that if a function fulfils certain pre conditions, we can logically conclude, that the post conditions will follow.

Have a look at this (very naive) implementation of a queue (i.e. ring buffer). You can also think about it as a First-In-First-Out buffer, where the ends are connected.

```c
#define QUEUE_SIZE 10

int queueData[QUEUE_SIZE];
int headPointer = 0;
int tailPointer = 0;

void queue_write(int data)
{
    queueData[headPointer] = data;
    headPointer = (headPointer + 1) % QUEUE_SIZE;
}

int queue_read()
{
    int data = queueData[tailPointer];
    tailPointer = (tailPointer + 1) % QUEUE_SIZE;

    return data;
}
```

As a first step, take either the queue_write or the queue_read function, and use ACSL (ANSI/ISO C Specification Language) to add one pre and one post condition. In a second step, rewrite the function you chose, and use the special assert( ) call to implement both the pre and post condition checking. You have to give the full function code, including the already existing function body.

Hint: The logic of a ring buffer actually requires, that multiple pre conditions are met in order to work properly. It is enough, if you just pick one here. Similarly, you can probably find multiple post conditions for each function. Again it is enough if you just pick one.

20 Essä 2 poäng Observer

In order to establish the correct functioning of an embedded system, a common method is to use an "observer". Give a short description of what we mean by that term, and explain with a short example (either textually or in C code, you **cannot** pick the one from the flipped classroom lecture or the one from the lab), where an observer can be used to establish the correct functioning of your overall system.

21 Essä 3 poäng Unit test with testing oracle

In the flipped classroom lecture about testing you saw, that in order to test a function, we usually have to put it into a test harness (a function that "wraps" around the function you want to test), which performs any necessary setups, then calls the function with given (probably random) inputs and then usually has an "oracle" to decide, if the function we are testing has returned the correct result. This is quite commonly called "unit testing".

Assume we want to test a function **int max(int * array, int size)**, which takes as inputs a pointer to an array of integers, and the size of the array, and returns the index of the maximum element in this array. You can assume, that each element in the array is unique (i.e. all elements in the array are different).

Write a test harness for this function, including proper setup (assume you have a function **int random(int minInt, int maxInt)**, which will return a random integer between and including the parameters minInt and maxInt), and then write a proper oracle to figure out, if **max** worked correctly. In case it did, the test harness should return 1, otherwise 0. I will not try and compile your C code, so it does not have to be fully syntactically correct. You do not have to strive for any coverage criteria here, so you can pick values for the random function that make sense for you, but that do not necessarily test out all corner cases.

22 Essä 4 poäng Statement coverage and branch coverage

In the context of testing software, we often talk about coverage criteria regarding our test cases. What do we mean by statement coverage, what do we mean by branch coverage? Write an example for a program in C code (you cannot use the one from the flipped classroom lecture), where statement coverage through test cases is impossible, and explain why.