(Till tentamensvakten: engelsk information behövs)

Exam

Embedded Systems I, DVA431
Västerås, 2018-08-15

Teacher:	Saad Mubeen, tel: 021-103191
Exam duration:	14:10 – 18:30
Help allowed:	calculator, language dictionary, ruler
Points:	90 p + extra lab points

Grading:

Swedish grades:		ECTS grades:	
0 – 54	→ failed	0 – 54	→ failed
55 – 76 p	→ 3	55 – 65	→ D
77 – 90 p	→ 4	66 – 79	→ C
91 – 100 p	→ 5	80 – 90	→ B
		91 – 100	→ A

Instructions:

- Answers should be written in English.
- Short and precise answers are preferred. Do not write more than necessary.
- If some assumptions are missing, or if you think the assumptions are unclear, write down what do you assume to solve the problem.
- Write clearly. If I cannot read it, it is wrong.

Good luck!!

Assignment 1: (16 points)

a) Designers of embedded systems have to develop products that are constrained in many different ways. Describe the constraints that are specific to embedded systems (i.e., the constraints that other type of computer systems do not have to consider). (6p)

b) Describe one application where you will prefer microcontroller over DSP. What could be the consequences of using DSP in this application? Motivate your answer. (4p)

c) Assume that your job is to develop software for resource-constrained embedded systems. Briefly explain two resource optimization techniques that you would use during the software development of these systems. (6p)

Assignment 2: (18 points)

a) Diodes and Zener diodes are used for protecting electronic boards and IC's. Describe how they are used and what type of protection they provide. Use circuit schematics to illustrate your answer. (8p)

b) Explain the concept of the "Open collector"/"Open Drain" mode to drive a digital output. Describe its role in the WIRED AND configuration and its working principle in the TWI communication bus. (10p)

Assignment 3: (12 points)

a) The build process of a microcontroller typically goes through three phases: compilation, linking and relocation, which are implemented respectively by the Compiler, the Linker and the Locator. Describe why the Linking phase is needed and how it is performed by the Linker. (8p)

b) Arrange in the correct order the following actions implemented by an Interrupt Service Routine (ISR). (4p)

i. Handle the interrupt

ii. Restore CPU context

iii. Acknowledge interrupt

iv. Save CPU context

Assignment 4: (16 points)

Assume three periodic tasks τ_1, τ_2 and τ_3 that communicate among each other by sending messages among their instances. The following is given:

Task τ_1:
- Has execution time 2 ms and period 12 ms.
- Sends 2 messages to a message queue MSGQ during each instance (job).
- All the messages are sent at the end of execution of each job.

Task τ_2:
- Has execution time 1 ms and a period 6 ms.
- Sends 2 messages to the message queue during each instance (job).
- All the messages are sent at the end of execution of each job.

Task τ_3:
- Has execution time 2 ms and a period 4 ms.
- Receives 2 messages from the message queue during each instance (job).
- All the messages are received at the end of execution of each job.

MSGQ:
- The queue contains the copy of the messages (not pointers).
- Has First In First Out (FIFO) order for inserting the messages.
- When a task reads a message from the message queue, the message is removed from the queue.

Questions:

a) Assume that the tasks are scheduled using the **Rate Monotonic** algorithm. What is the minimum possible size of the message queue (counted in number of messages) such that we are able to guarantee there will always be enough space in the queue for $\tau1$ and $\tau2$ to insert their messages? Motivate your answer by drawing an execution trace up to one hyper period and showing the number of messages in the queue after the execution of each task instance. (8p)

b) Assume that the tasks are scheduled using the **Shortest Job First** algorithm. What is the minimum possible size of the message queue (counted in number of messages) such that we are able to guarantee there will always be enough space in the queue for $\tau1$ and $\tau2$ to insert their messages? Motivate your answer by drawing an execution trace up to one hyper period and showing the number of messages in the queue after the execution of each task instance. (8p)

Assignment 5: (10 points)

a) What are the differences among executing the software, debugging the software and testing the software? (6p)

b) “Low coupling and high cohesion” is a good strategy for designing software modules in embedded systems. Do you agree or disagree with this statement? Justify your answer. (4p)

Assignment 6: (18 points)

Consider a real-time task set consisting of five tasks, A, B, C, D, E that share four resources protected by semaphores S1, S2, S3, S4. The tasks have different priorities and they are released for execution at different release times (see table below). All tasks use their semaphores as illustrated in the column "Execution sequence" below (clock ticks are counted relative to the start of the system). The execution times of tasks are as follows: A = 9 ticks, B = 4 ticks, C = 5 ticks, D = 5 ticks and E = 4 ticks as illustrated in the table below (in the "Execution sequence" column). The deadline of each task is relative to its release time. For example, the deadline of Task D is 25, which is relative to its release time 2 (see the table below). This means that relative to time 0, the deadline of Task D is 27.

Task	Priority	Release time	Deadline (relative to the release time)	Execution sequence								
A	5 (Highest)	10	11		S1		S2		S3		S4	
B	4	8	16		S4	S4						
C	3	5	20			S3	S3					
D	2	2	25		S2	S2	S2					
E	1 (Lowest)	0	27		S1	S1						

Clock Tick 0 1 2 3 4 5 6 7 8 9

For example, we can see in the table that task D has the fourth highest priority, Prio (D) = 2, it is released at time t = 2, and, once released, it will execute like described below:

- *tick 2+0*: tries to execute one clock tick without any semaphores.
- *tick 2+1*: tries to lock semaphore S2, and if ok, it enters its critical section with S2.
- *tick 2+2*: tries to continue its critical execution with S2.
- *tick 2+3*: tries to continue its critical execution with S2. It then releases S2 at the end of the tick.
- *tick 2+4*: tries to execute one clock tick without any semaphores.

The same reasoning applies to all other tasks.

Note that the execution scenarios for the tasks will be equal to the ones illustrated in the table above *only under the assumption* that the required semaphores are *free* when requested by a task, and the task is not pre-empted by a higher-priority task. However, from the release times above we can see that the tasks will interfere with each other. Besides, the semaphores will not be always available when requested by the tasks.

Assume the release times of the tasks, their priorities and the execution sequences from the table above:

a) Is the task set schedulable if the Immediate *Priority Ceiling Protocol (IPCP)* is used? If not, then why not? Draw the actual execution trace. You should run your trace from time t=0 until all of the tasks have completely executed once their execution sequence. (12p)

b) Does priority inversion occur in (a)? If yes, specify the times when the priority inversion occurs. If not, explain why not. (3p)

c) Does chained blocking occur in (a)? If yes, specify the times when the chained blocking occurs. If not, explain why not. (3p)

Assignment 7: (extra lab points)

You do not need to do anything here. This is for the extra points earned at the labs. Your extra lab points will be automatically added to your total exam score.
