Design and Integration of Embedded Systems exam SOLUTION **December 16, 2020.**

Available time: 60 minutes

Maximum points: 24. Minimum points needed: 12 points

Please start solving each exercise on a separate page, writing first your name and Neptun code.

Exam topics: System Design (12 points)

1. You are working for an embedded software company. You have to create a centralized version control system for your team's project. Your colleagues are beginners in version control and they would like you to help them:

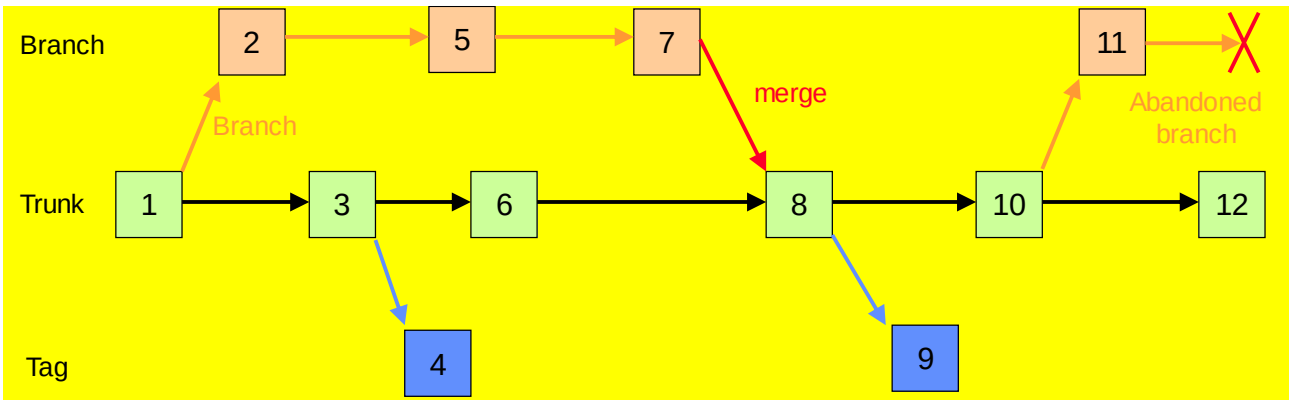
   a. Show them, what type of method you suggest during the parallel development of your team's project! Reason your decision! Draw them an example of two developers working in parallel on the same code part! **(2 points)**

   For a centralized version control system the copy-modify-merge method is the most efficient for parallel software development.

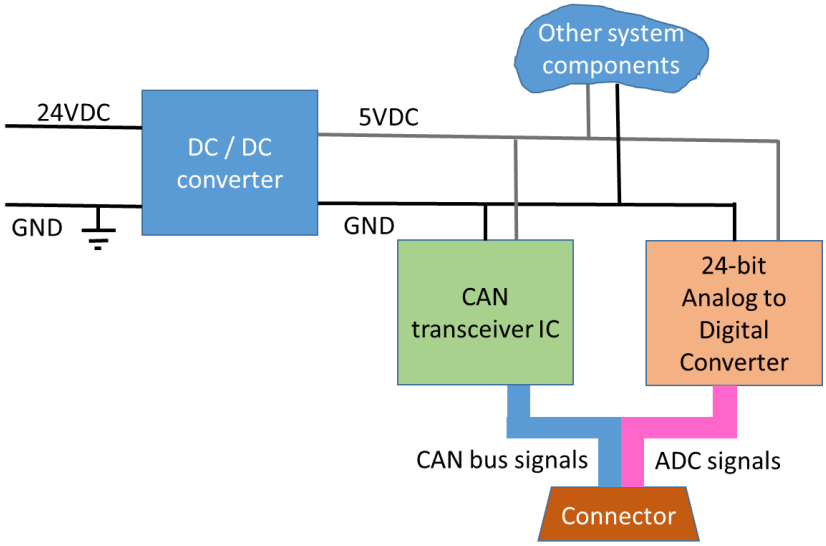
   Drawing example: VIMIMA11_DIES_02_ConfigurationManagement slides 26th to 33th.

   b. The colleagues are interested in creating releases time to time. What can you suggest to solve this problem? **(1 point)**

   In version control systems you can create branches. For example SVN suggests a special branch the TAG for creating releases.



2. You are working as a hardware system engineer at an embedded system company. The following block diagram is suggested by your colleagues. The block diagram does not contain minor components like filtering capacitors, diodes and inductors; therefore you do not need to deal with these. The colleagues said that the physical PCB layout of chips and wires is intended to follow the block diagram. (CAN is a communication bus with max. 1 Mbit/sec data rate.)



a. Can you identify possible problems on this block diagram? What are these problems, and why are they problematic? **(2 points)**

There are two possible main problem categories:

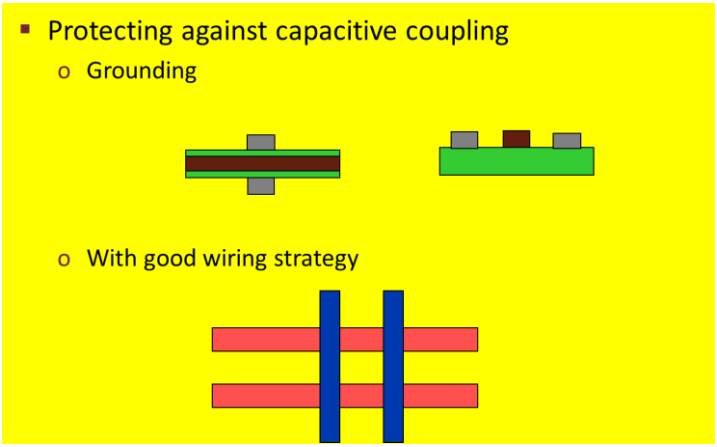
1. Power supply. The system uses the Power bus architecture which is a bad choice for a precision analog measurement (24-bit ADC). The power supply applies a DC/DC converter which is also too noisy for precision measurements.

2. The precision measurement signals and communication signals are wired close to each other in the same connector, which causes possible EMC problems.

b. What are your suggestions to improve the design? **(3 points)**

1. Power supply problems can be solved by using more sophisticated power architectures like the Power Star point, or the Star point with separate PSU. (Drawing is needed.)

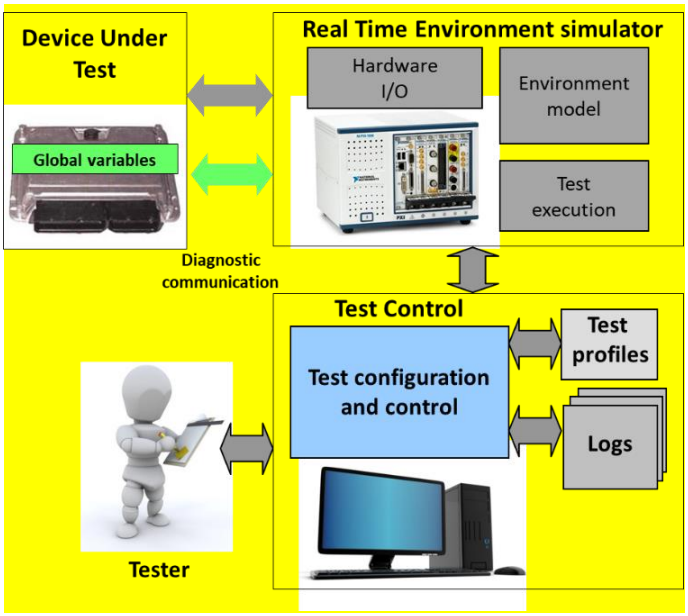
2. EMC problems can be solved by separating the connector by functions, or by using GND lines between the signals. Wiring strategies can also be used.



3. As a tester in an embedded system company, you have to test a complex automotive ECU (Electronic Control Unit, with integrated hardware and software). This ECU has interactions with other ECUs on the car's communication bus, it has many analog and digital signals to measure, and also has actuators to control parts of the car's mechanical system. The ECU is a safety critical unit: if it detects problems regarding its communication, measurements, or actuators, then it goes to a safe state.

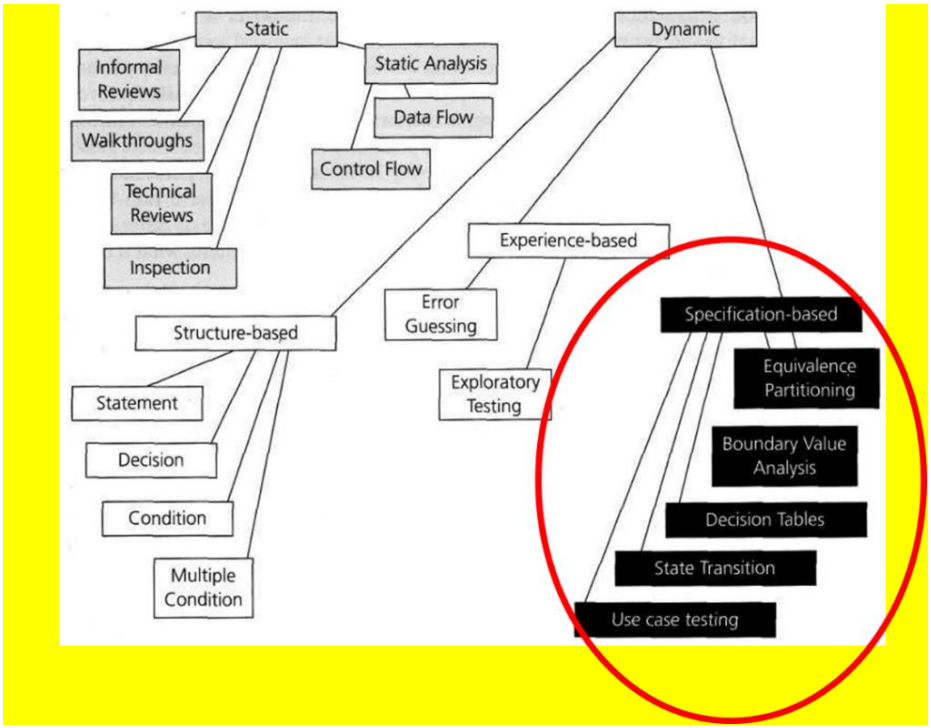
   a. How could you test such an ECU? Suggest and draw a test environment for the ECU! **(2 points)**

   This ECU requires a HIL environment, which simulates the analog and digital signals to measure, the communication, and the actuators.



   b. What test types suggested by the ISTQB can be used in this case? Give an example of using one of these techniques! **(2 points)**

   In this environment the black-box, specification based test types can be used: equivalence partitioning, boundary value analysis, decision tables, state transition testing.



   An example: sending communication messages containing a specified value of an equivalence class to the ECU and checking the ECU's response to this message (for example the ECU's actuator state changing).

Exam topics: Safety Critical Systems (12 points)

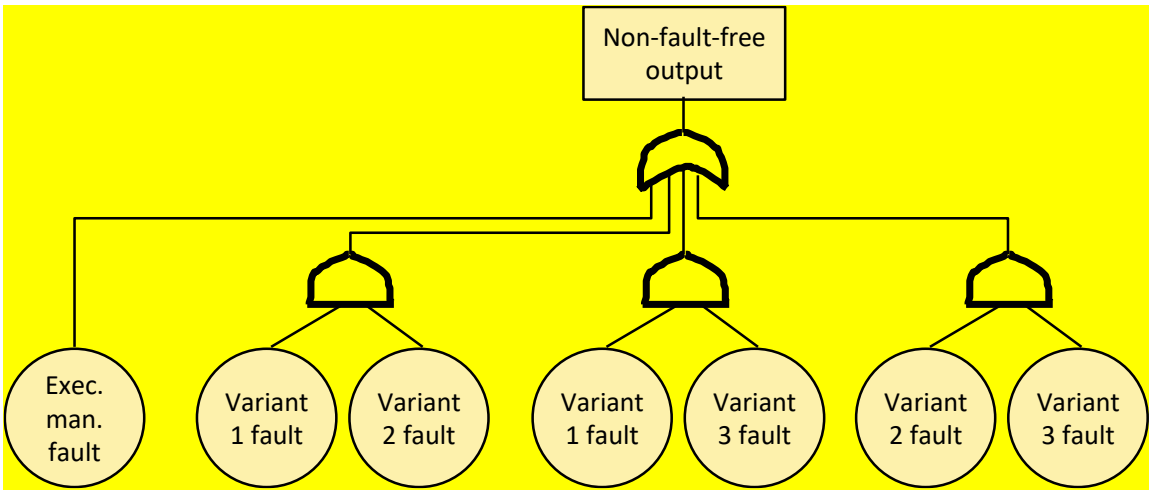
4. Consider a software system consisting of 3 software *variants* (diverse components implemented to solve the same problem) and an *execution manager* module, which executes the three variants according to the N-version programming (NVP) scheme.

   a. Describe the tasks that have to be implemented by the execution manager module according to the NVP scheme. **(1 point)**

   Execute the 3 software variants with the same input and collect their outputs. If there is a timeout during the execution of a variant then consider its output as faulty.
   Perform a majority voting on the outputs of the variants. If there is majority then provide it as output of the system, otherwise indicate that there is no fault-free output.

   b. Draw the *fault tree* belonging to the *non-fault-free output* of the system, considering the independent faults of the components. **(2 points)**

   The fault tree (the cut in which all variants are faulty can be reduced):



   c. List the *single points of failure* (or indicate, if there is no single point of failure). **(1 point)**

   Single point of failure: Fault of the execution manager (Exec. man. fault in the Figure).

5. Describe *shortly*, what determines the execution time of the following fault handling and fault tolerance techniques: forward recovery, backward recovery, N-version programming (NVP), recovery blocks (RB). **(2 points)**

   The execution time is determined as follows:

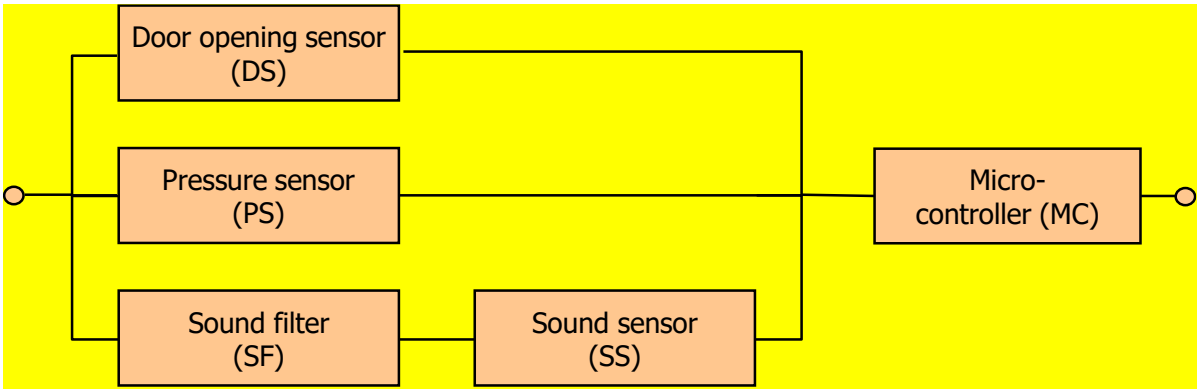
   - Forward recovery: The execution time of setting an error-free state by selective correction.
   - Backward recovery: The time needed to restore a prior error-free state (using a checkpoint that was saved earlier), and repeat the execution from that state.
   - N-version programming (NVP): The time needed to execute the variants (in parallel or serially each after the other; in the first case the slowest variant or timeout determines the execution time) and perform a majority voting on the outputs of the variants.
   - Recovery blocks (RB): The time needed to execute the primary variant and the acceptance check; if the output is not acceptable then execute additional variants until an acceptable output is provided by a variant (i.e., the execution time depends on the number of faulty variants). Optional: saving and then restoring the state before the execution of the next variant.

6. The intrusion detection system of a flat includes as detectors a *door opening sensor*, a *pressure sensor* on the floor, and a *sound sensor* with a *sound filter* on its input. The outputs of the sensors are processed by a *microcontroller* which provides an alarm when any of the sensors (i.e., at least one sensor) detects an intrusion. A faulty sensor or filter does not give output. The microcontroller does not provide alarm in case of its fault or missing input. The characteristics of these components are as follows:

	Door opening sensor	Pressure sensor	Sound filter	Sound sensor	Micro-controller
MTTF (hours)	25 000	25 000	30 000	30 000	40 000
MTTR (hours)	4	4	3	3	2

a. Construct the *reliability block diagram* of this system! **(3 points)**

The reliability block diagram:



b. Give the *formula* that can be used to compute the asymptotic *availability of a component* if its MTTF and MTTR characteristics are known. **(1 point)**

$A = MTTF / (MTTF+MTTR)$

c. Give the *formula* that can be used to compute the asymptotic *availability of this system* (in case of regular repairs of the failed components) using the asymptotic availabilities of the components! It is enough to give the formula, the result of computation shall not be provided. **(2 points)**

Let's denote the asymptotic availabilities of the components as A_{DS} (door opening sensor), A_{PS} (pressure sensor), A_{SF} (sound filter), A_{SS} (sound sensor), and A_{MC} (microcontroller).

The availability of the equipment:

$$A = \left(1 - (1-A_{DS})*(1-A_{PS})*(1-A_{SF}*A_{SS})\right) * A_{FM}$$