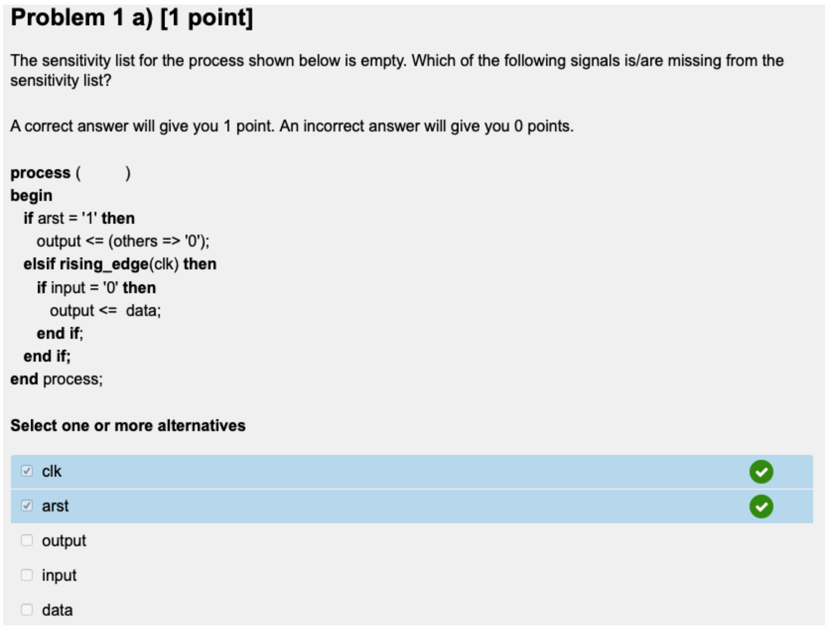
Version 0.1

FYS4220/9220 Real time and embedded data systems

- Date: Monday 2. December 2019
- Exam hours: 09:00 - 13:00 (4 hours)
- All exam questions must be answered. Each question may be weighted differently. For each question the maximum number of points is specified. The maximum number of points for the complete question set is 100 points
- Permitted material: None

For question where the answer include code, pseudo-code may be accepted if it can clearly demonstrate an understanding of the functionality.

Question set with suggested answers.



Problem 1 a) [1 point]

The sensitivity list for the process shown below is empty. Which of the following signals is/are missing from the sensitivity list?

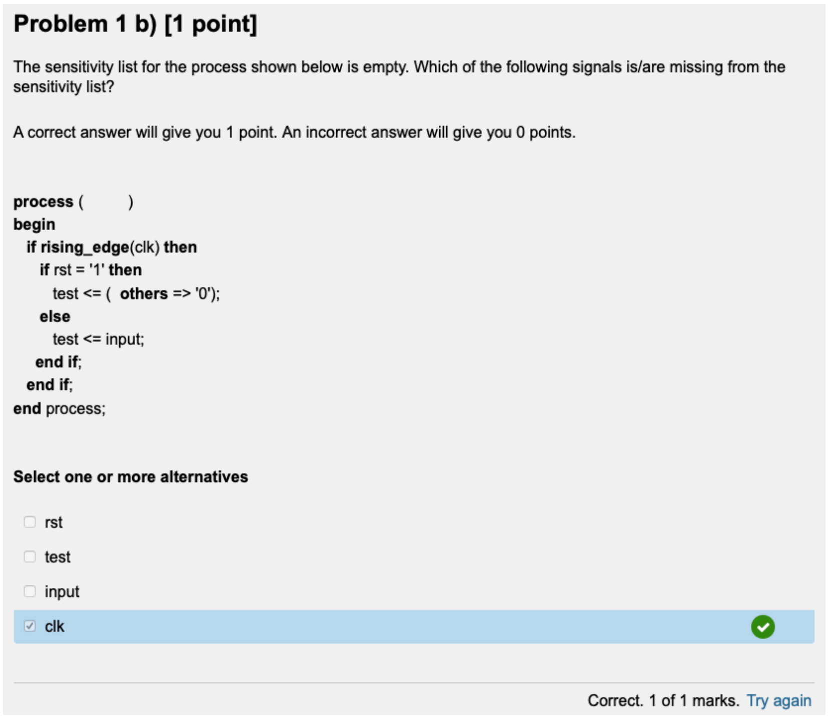
A correct answer will give you 1 point. An incorrect answer will give you 0 points.

```
process (        )
begin
  if arst = '1' then
    output <= (others => '0');
  elsif rising_edge(clk) then
    if input = '0' then
      output <=  data;
    end if;
  end if;
end process;
```

Select one or more alternatives

- [✓ clk] ✓
- [✓ arst] ✓
- output
- input
- data

This is a clocked process with asynchronous reset. Both the clock and the asynchronous reset shall trigger the process and thus be listed in the sensitivity list. Both signals are needed to obtain full score.



Problem 1 b) [1 point]

The sensitivity list for the process shown below is empty. Which of the following signals is/are missing from the sensitivity list?

A correct answer will give you 1 point. An incorrect answer will give you 0 points.

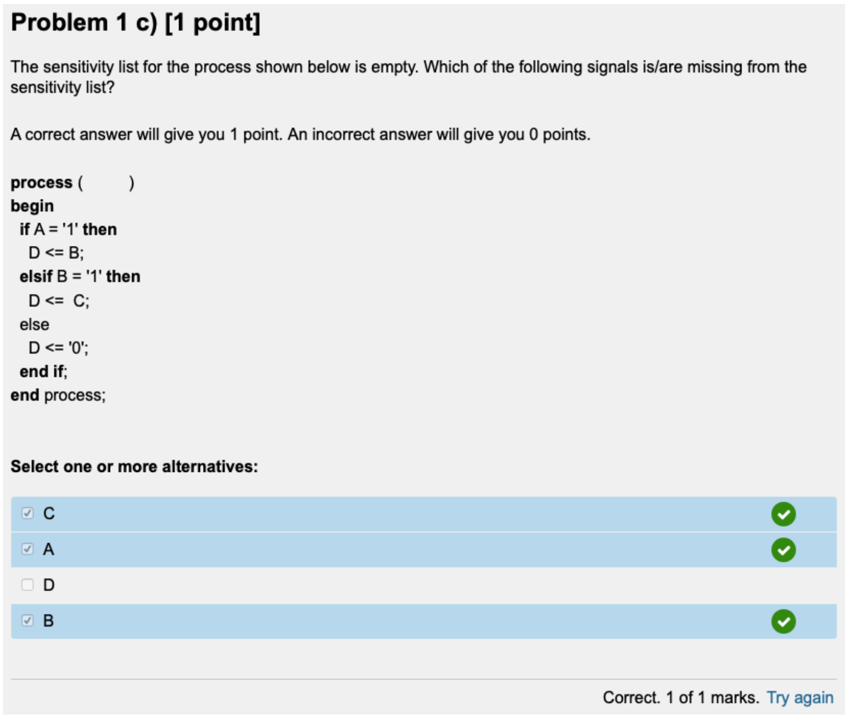
```
process (        )
begin
  if rising_edge(clk) then
    if rst = '1' then
      test <= ( others => '0');
    else
      test <= input;
    end if;
  end if;
end process;
```

Select one or more alternatives

- ☐ rst
- ☐ test
- ☐ input
- ☑ clk ✓

Correct. 1 of 1 marks. Try again

This is a synchronous process with synchronous reset. Only the clock signal shall trigger the process, thus only the clock signals shall be listed in the sensitivity list.



Problem 1 c) [1 point]

The sensitivity list for the process shown below is empty. Which of the following signals is/are missing from the sensitivity list?

A correct answer will give you 1 point. An incorrect answer will give you 0 points.

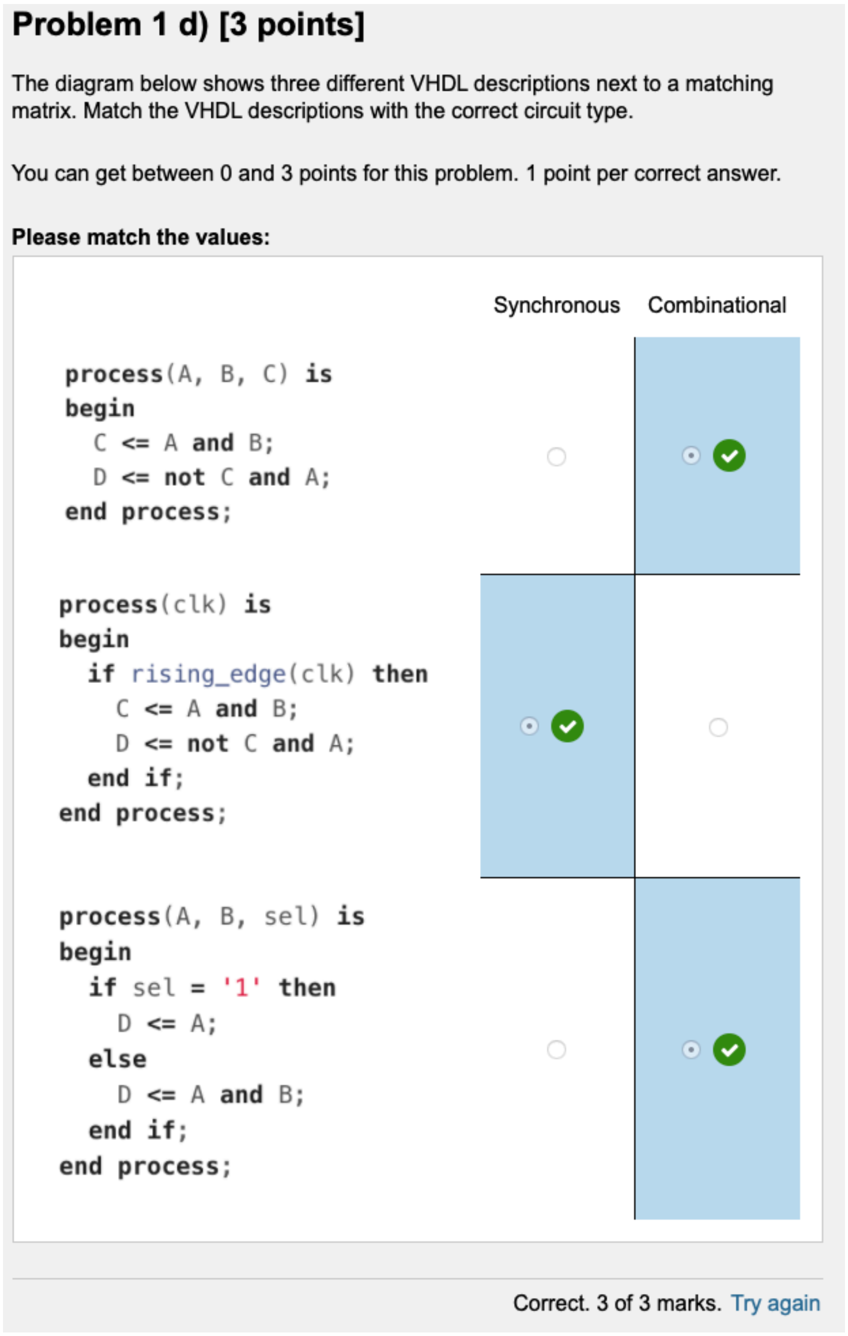
```
process (        )
begin
  if A = '1' then
    D <= B;
  elsif B = '1' then
    D <=  C;
  else
    D <= '0';
  end if;
end process;
```

Select one or more alternatives:

- ☑ C ✓
- ☑ A ✓
- ☐ D
- ☑ B ✓

Correct. 1 of 1 marks. Try again

This is a combinational process and thus all signals which are read by the process shall be listed in the sensitivity list. A, B, C are all inputs to the process (read by the process) and shall thus be listed in the sensitivity list. D is an output and shall not trigger the process and shall therefore not be listed in the sensitivity list.



Problem 1 d) [3 points]

The diagram below shows three different VHDL descriptions next to a matching matrix. Match the VHDL descriptions with the correct circuit type.

You can get between 0 and 3 points for this problem. 1 point per correct answer.

Please match the values:

```
process(A, B, C) is
begin
  C <= A and B;
  D <= not C and A;
end process;
```

Synchronous [✓ Combinational]

```
process(clk) is
begin
  if rising_edge(clk) then
    C <= A and B;
    D <= not C and A;
  end if;
end process;
```

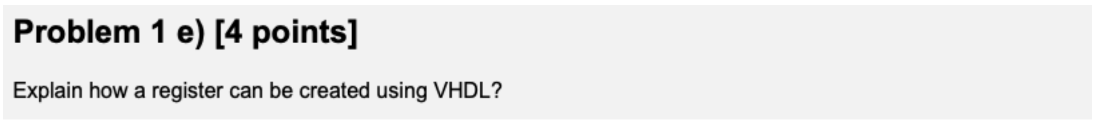
[✓ Synchronous] Combinational

```
process(A, B, sel) is
begin
  if sel = '1' then
    D <= A;
  else
    D <= A and B;
  end if;
end process;
```

Synchronous [✓ Combinational]

Correct. 3 of 3 marks. Try again

A process which contains conditional statement checking for a rising or falling edge on a clock signal is a clocked / synchronous process. This is the case for the process in the middle above. The two others are combinational processes as they are not dependent on a clock signal.

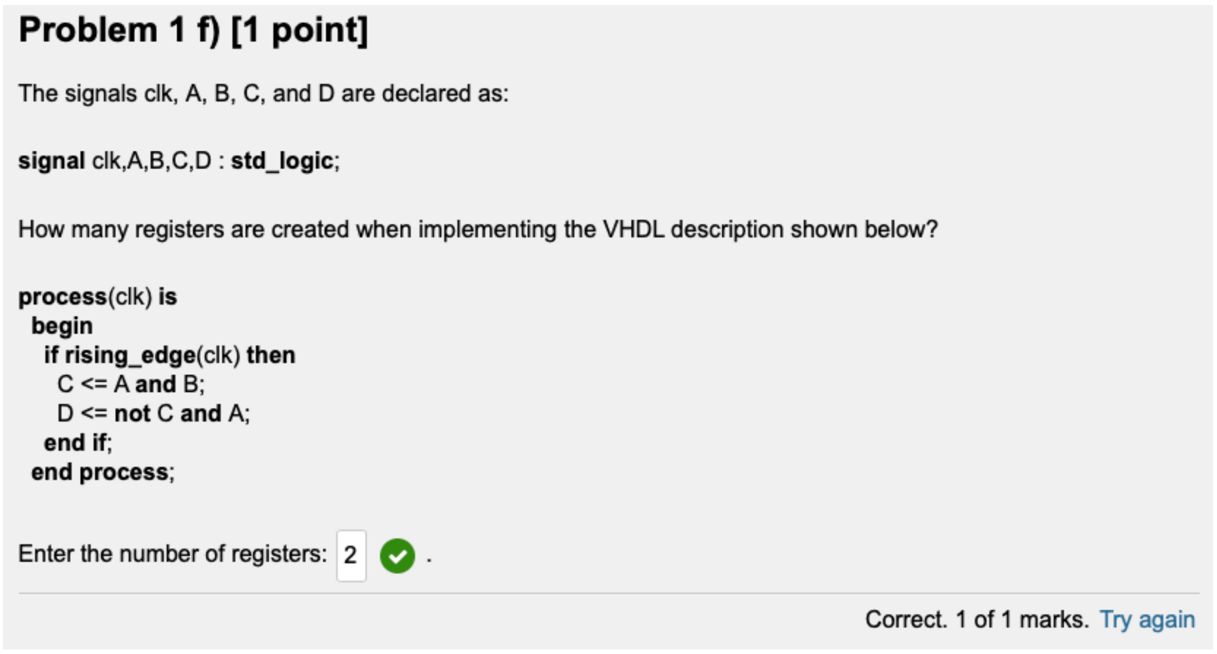


Problem 1 e) [4 points]

Explain how a register can be created using VHDL?

Answer:
A register can be created using a clocked process (synchronous logic) in VHDL. Every signal assignment within synchronous process, that is, within an *if rising_edge* or *if falling_edge* conditional statement, will create a register.

A partly correct answer demonstrating that the student has some understanding of this concept will give 1-3 points depending on the wording. E.g. writing that a clocked process will generate registers is correct, but a more precise answer is to add that it is the signal assignment within a clocked process, and within the rising or falling edge condition, that will result in a register being generated.



Problem 1 f) [1 point]

The signals clk, A, B, C, and D are declared as:

```
signal clk,A,B,C,D : std_logic;
```

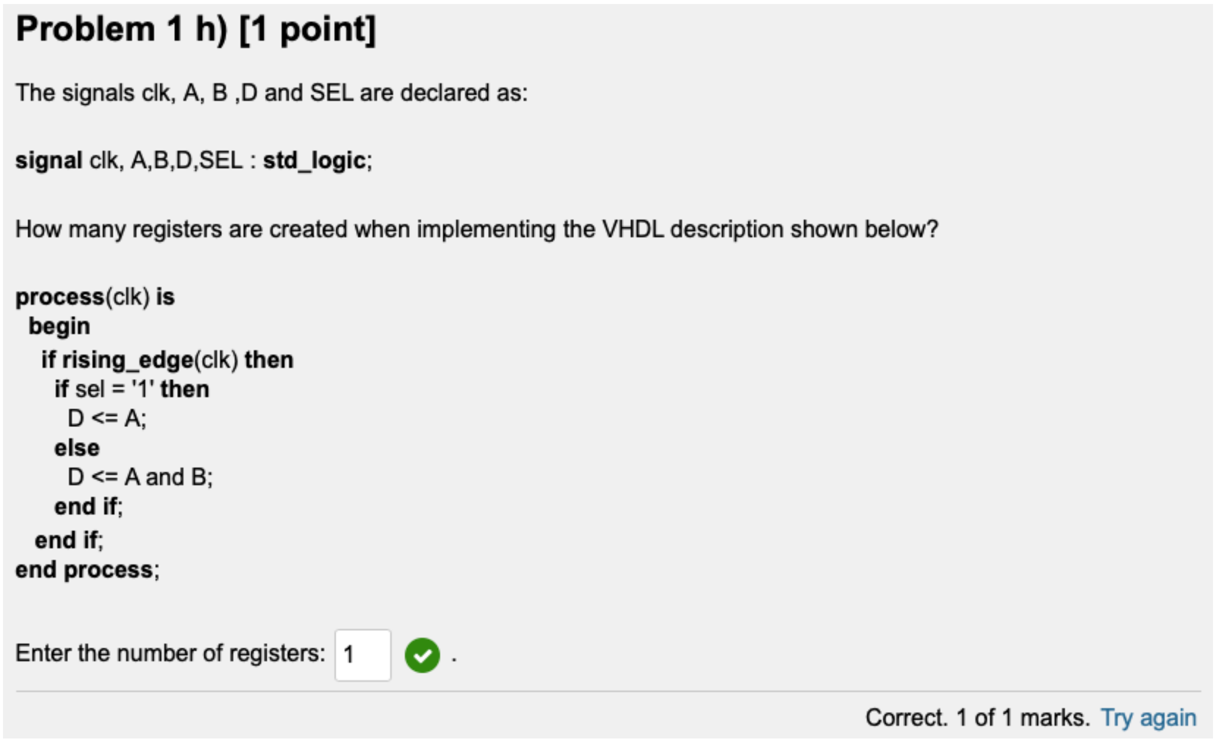
How many registers are created when implementing the VHDL description shown below?

```
process(clk) is
 begin
  if rising_edge(clk) then
   C <= A and B;
   D <= not C and A;
  end if;
 end process;
```

Enter the number of registers: 2 ✓ .

Correct. 1 of 1 marks. Try again

There are two assignment of std_logic signals within the clocked process and rising edge condition. Thus, two register will be generated.



Problem 1 h) [1 point]

The signals clk, A, B ,D and SEL are declared as:

```
signal clk, A,B,D,SEL : std_logic;
```

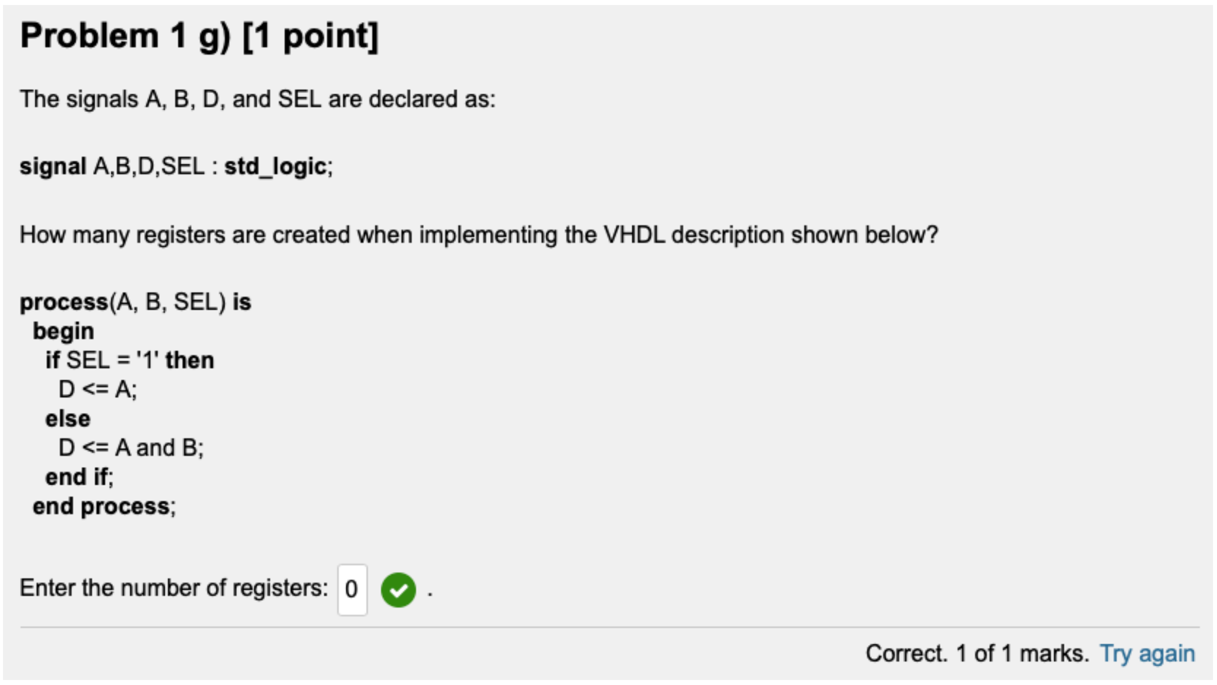
How many registers are created when implementing the VHDL description shown below?

```
process(clk) is
 begin
  if rising_edge(clk) then
   if sel = '1' then
    D <= A;
   else
    D <= A and B;
   end if;
  end if;
end process;
```

Enter the number of registers: 1 ✓ .

Correct. 1 of 1 marks. Try again

There is one assignment of a std_logic signal within the clocked process and rising edge condition. Thus, one register will be generated. While there are two lines of code assigning a value to the output D, these lines are part of one conditional statement and only one is true at any time.



Problem 1 g) [1 point]

The signals A, B, D, and SEL are declared as:

```
signal A,B,D,SEL : std_logic;
```

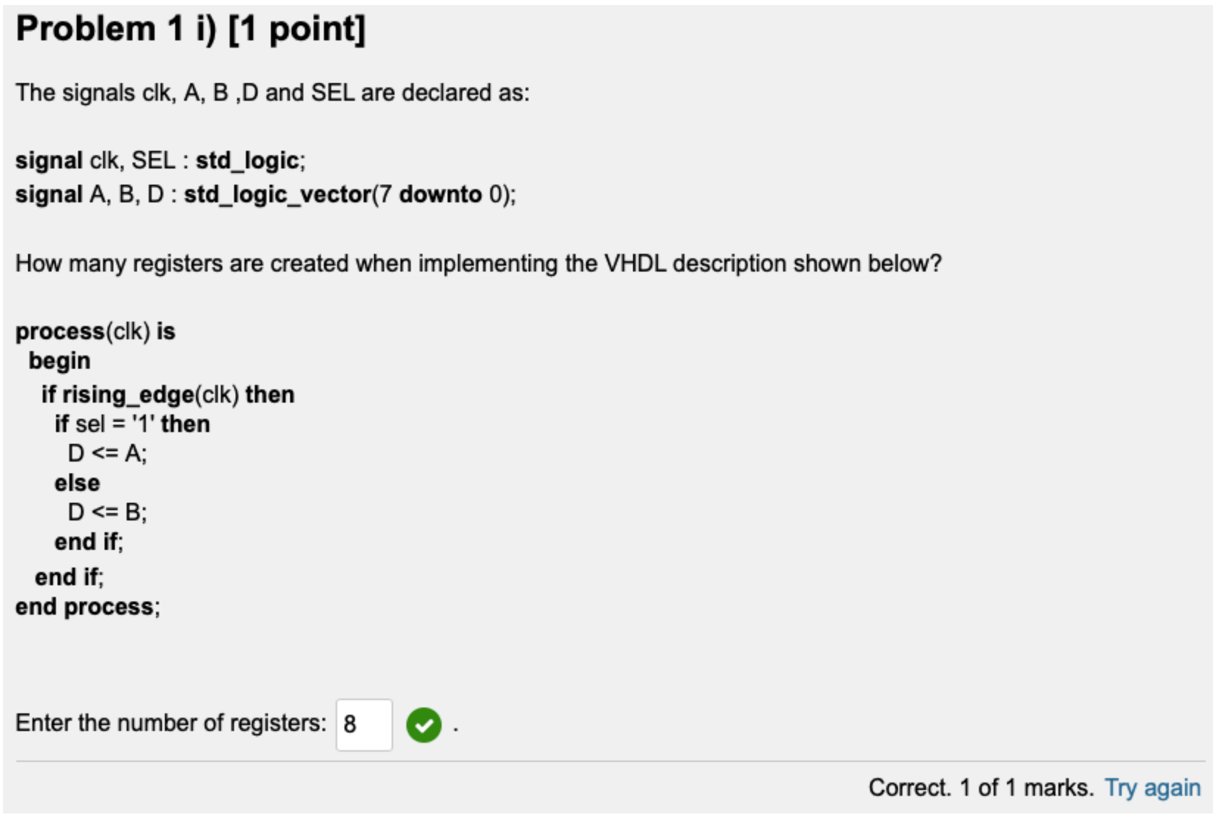
How many registers are created when implementing the VHDL description shown below?

```
process(A, B, SEL) is
 begin
  if SEL = '1' then
   D <= A;
  else
   D <= A and B;
  end if;
 end process;
```

Enter the number of registers: 0 ✓ .

Correct. 1 of 1 marks. Try again

This is a purely combinational process which is not dependent on a clock signal. Thus, no register will be generated.



Problem 1 i) [1 point]

The signals clk, A, B ,D and SEL are declared as:

```
signal clk, SEL : std_logic;
signal A, B, D : std_logic_vector(7 downto 0);
```

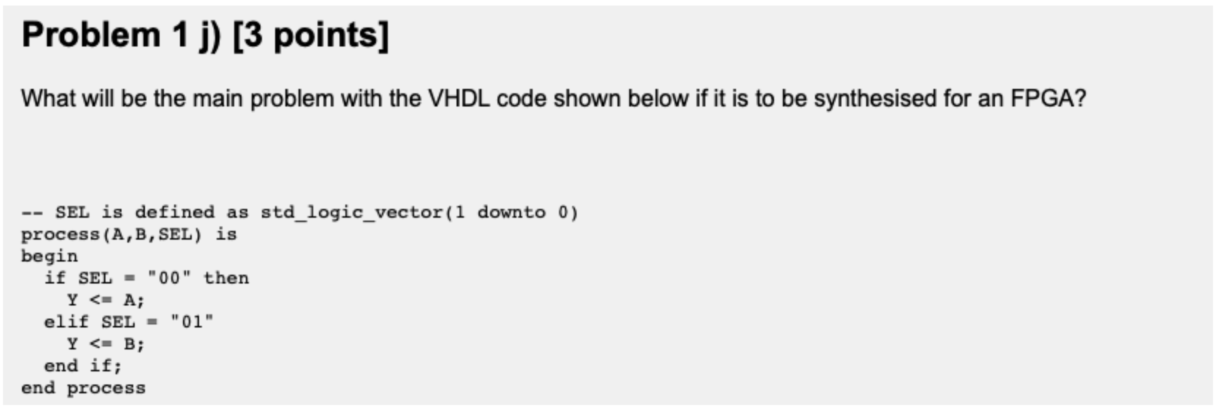
How many registers are created when implementing the VHDL description shown below?

```
process(clk) is
 begin
  if rising_edge(clk) then
   if sel = '1' then
    D <= A;
   else
    D <= B;
   end if;
  end if;
end process;
```

Enter the number of registers: 8 ✓ .

Correct. 1 of 1 marks. Try again

In this case there is an assignment of an 8-bit wide signal of the type std_logic_vector. This would be equivalent to assigning 8 individual std_logic signals. A total of 8 register, one for each bit in the vector, is generated.



Problem 1 j) [3 points]

What will be the main problem with the VHDL code shown below if it is to be synthesised for an FPGA?

```
-- SEL is defined as std_logic_vector(1 downto 0)
process(A,B,SEL) is
begin
  if SEL = "00" then
    Y <= A;
  elif SEL = "01"
    Y <= B;
  end if;
end process
```

Answer:
This is a combinational process where all outputs need to be assigned a value for all possible combinations / values of the signals that are read by the process. If the output is not specified for every possible set of input conditions, the option taken by the synthesis tool is to not change the current output. This is done by adding a latch, which adds an extra path for the signal to travel through and thus will impact the timing closure of the design. In the example code given, the output Y is not determined for all possible values of the input vector SEL.

A partly correct answer demonstrating that the student has some understanding of why an unintended latch is generated will give 1-2 point depending on the wording. To get full score the answer must demonstrate both an understanding of that the code has a problem and why this is a problem. E.g. writing only that the code has a problem because Y is not defined for all possible values of the inputs is only partly demonstrating and understanding. To get full score this must be combined with and understanding of that this code will generate an unintended latch and thus impact timing closure.

The VHDL description in this problem has 3 minor syntax errors.

- ***elif** should have been **elsif***
- ***then** is missing from the **elsif** condition*
- *a semicolon is missing after **end process***

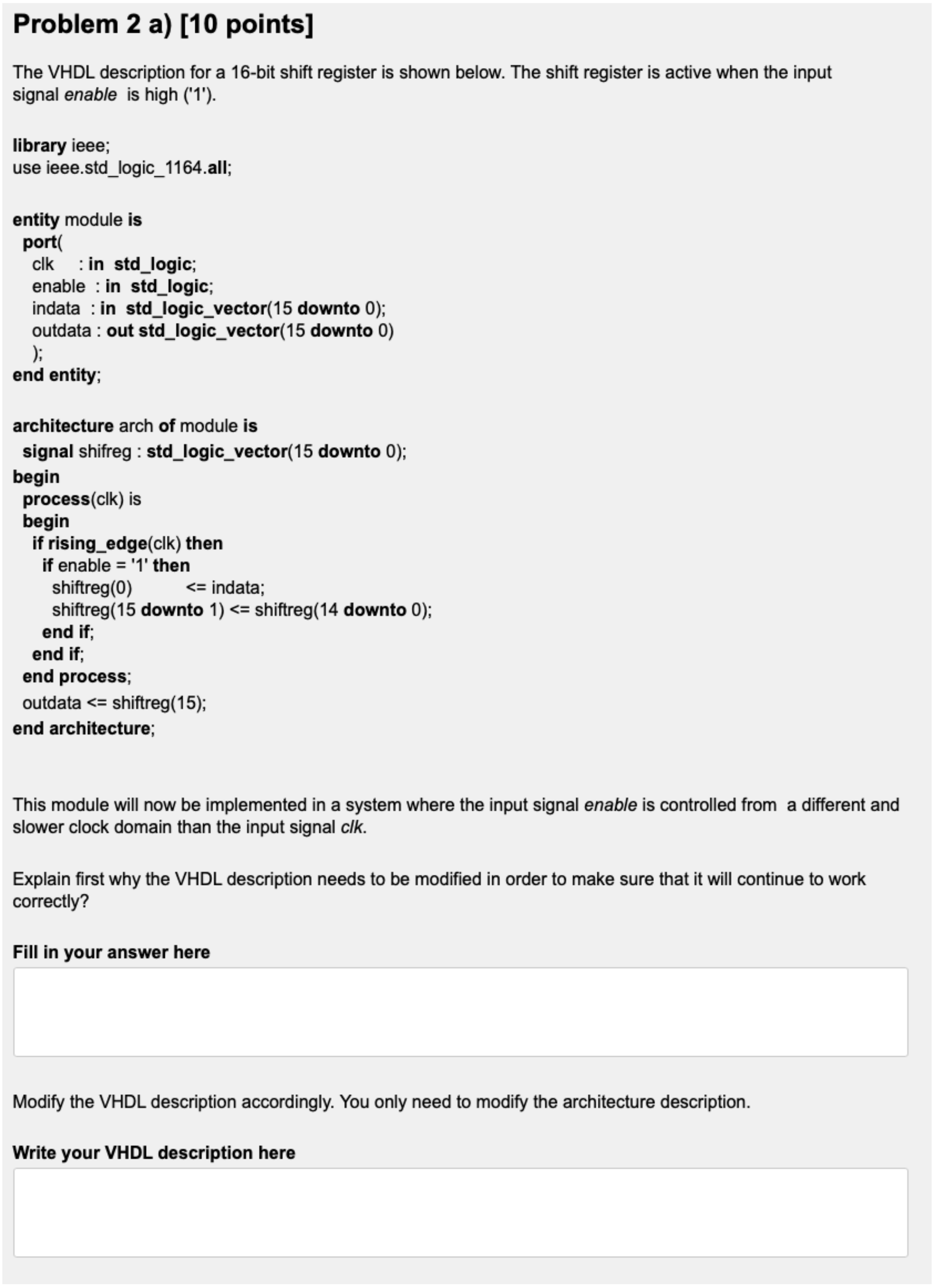
These syntax errors are however not considered to be significantly misleading from the real intended and main problem.

Problem 1 k) [3 points]

What is the purpose of an LUT in an FPGA?

Answer:
The Look-up Table (LUT) provides the capability of to implement combinational logic (Boolean functions) to the FPGA design. By storing the truth table of a Boolean function in the memory elements of a LUT, the inputs are used to address the various memory locations. The output will be the value stored at the memory location specified by the input values, and corresponding to the correct result of the given boolean operation.

A partly correct answer demonstrating that the student has some understanding of what an LUT is will give 1-3 points depending on the wording. To get full score the answer has to contain the point on “implementing combinational logic”.



Problem 2 a) [10 points]

The VHDL description for a 16-bit shift register is shown below. The shift register is active when the input signal *enable* is high ('1').

```
library ieee;
use ieee.std_logic_1164.all;

entity module is
 port(
  clk     : in  std_logic;
  enable  : in  std_logic;
  indata  : in  std_logic_vector(15 downto 0);
  outdata : out std_logic_vector(15 downto 0)
  );
end entity;

architecture arch of module is
 signal shifreg : std_logic_vector(15 downto 0);
begin
 process(clk) is
 begin
  if rising_edge(clk) then
   if enable = '1' then
    shiftreg(0)          <= indata;
    shiftreg(15 downto 1) <= shiftreg(14 downto 0);
   end if;
  end if;
 end process;
 outdata <= shiftreg(15);
end architecture;
```

This module will now be implemented in a system where the input signal *enable* is controlled from a different and slower clock domain than the input signal *clk*.

Explain first why the VHDL description needs to be modified in order to make sure that it will continue to work correctly?

Fill in your answer here

Modify the VHDL description accordingly. You only need to modify the architecture description.

Write your VHDL description here

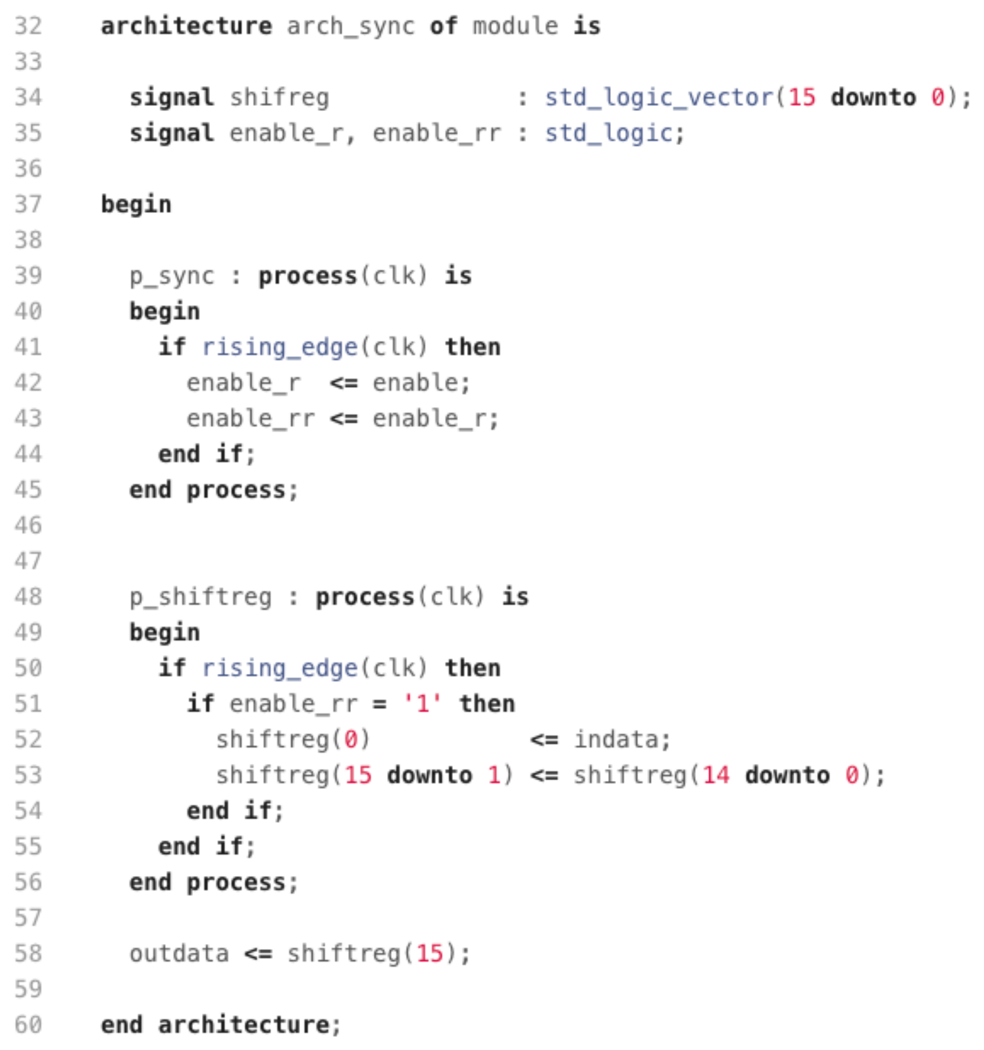
This question can score 10 points. Each sub-question can score 5 points each.

Answer part 1:

Since the input signal *enable* is controlled by a different and slower clock domain, it is necessary to synchronize the enable signal into the new clock domain. Without synchronization, any change in the *enable* signal will occur independent of the clock signal within the design it is used. If this change does not meet the setup and hold time requirements of the registers it is used to enable, the design may experience a metastable condition.

The important concept to demonstrate is that 1.) the enable signal can change independently of the clock signal, and 2.) therefore may not meet the setup and hold time requirements (e.g. arrive at the register input to close to the clock edge, and 3. therefore lead to a metastable state. Mentioning only one or two of these will give 1-3 points depending on the wording. Mentioning two will give 2-4 points depending on the wording, and mentioning all with clarity will give full score.

Answer part 2:



```vhdl
architecture arch_sync of module is

  signal shifreg               : std_logic_vector(15 downto 0);
  signal enable_r, enable_rr : std_logic;

begin

  p_sync : process(clk) is
  begin
    if rising_edge(clk) then
      enable_r  <= enable;
      enable_rr <= enable_r;
    end if;
  end process;


  p_shiftreg : process(clk) is
  begin
    if rising_edge(clk) then
      if enable_rr = '1' then
        shiftreg(0)          <= indata;
        shiftreg(15 downto 1) <= shiftreg(14 downto 0);
      end if;
    end if;
  end process;

  outdata <= shiftreg(15);

end architecture;
```

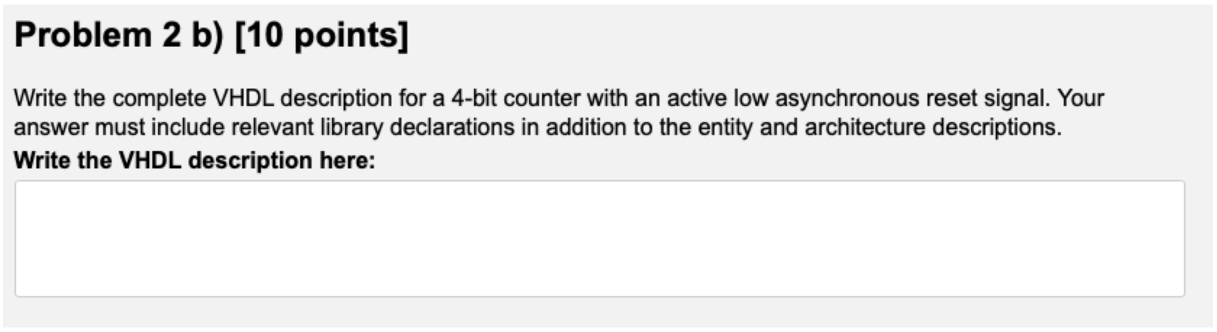
A correct answer is expected to demonstrate that the candidate knows how to implement synchronization registers (assignment inside process + declaration of signals), and that the check on the input signal *enable* also has to be changed to instead check on the synchronized signal.

The statements adding the two synchronization register can be placed either in a separate process or within the p_shiftreg process. Both are considered correct answers.

Pseudo-code may give points if it can demonstrate the understanding of that the signal needs to be synchronized using to registers.

There is no need to add edge-detection in this problem as the shift-register will be active as long as the enable signal is high/active. In-fact, adding edge detection will change the functionality of the shift-register and should therefore not give a full score.

The VHDL description presented in the problem has incorrect type declaration of the ports indata and outdata. In the example these are declared as std_logic_vector(15 downto 0), while the correct declaration should have been std_logic. However, this is not considered to be significantly misleading from the real intended and main problem.



Problem 2 b) [10 points]

Write the complete VHDL description for a 4-bit counter with an active low asynchronous reset signal. Your answer must include relevant library declarations in addition to the entity and architecture descriptions.

Write the VHDL description here:

Answer:

Entity description:

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity counter is
  port(
    clk     : in  std_logic;
    arst_n  : in  std_logic;
    counter : out std_logic_vector(3 downto 0)
    );
end entity;
```

Architecture alternative 1: Solved using counter as a signal vector.

```vhdl
architecture arch_signal of counter is

  signal cnt : unsigned(3 downto 0);
begin

  p_sig : process(arst_n, clk) is
  begin
    if arst_n = '0' then
      cnt <= (others => '0');
    elsif rising_edge(clk) then
      cnt <= cnt + 1;

    end if;
  end process;

  counter <= std_logic_vector(cnt);

end architecture;
```

Architecture alternative 2: Solved using counter as a variable vector.

```vhdl
architecture arch_variable of counter is
begin

  p_var : process(arst_n, clk) is
    variable cnt : unsigned(3 downto 0);
  begin
    if arst_n = '0' then
      cnt := (others => '0');
    elsif rising_edge(clk) then
      cnt := cnt + 1;
    end if;
    counter <= std_logic_vector(cnt);
  end process;

end architecture;
```

Her er vi på jakt etter forståelsen for hvordan skrive grunnleggende VHDL-kode med eksempelet for en teller, i tillegg ser vi etter dypere forståelse for at vi trenger unsigned ved implementasjon av telleren og at denne må konverters til std_logic_vector.

Det main purpose of this problem question is to check for a basic understanding of how to write a simple VHDL code from a specification. We are thus looking for structure and knowledge of how to write the entity, architecture and process VHDL description. For the process description we are looking for correct implementation of the sensitivity list and a clocked process with asynchronous reset. Furthermore, we are probing a deeper VHDL specific knowledge of that arithmetic operations should be carried out using the numeric_std library and that there is a need to convert from unsigned to the std_logic_vector output.

Problem 2 c) [10 points]

Write a the VHDL description for a basic test bench that can be used to simulate the VHDL description of the 4-bit counter from Problem 1 h).

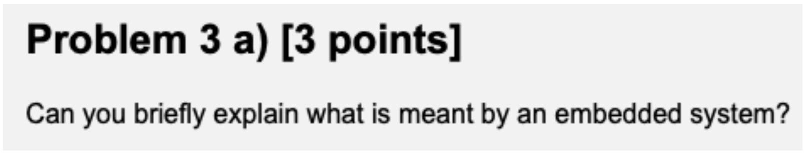
The test bench shall fulfil the following requirements:

- generate the input stimuli signals for the clock and reset input of the 4-bit counter.
- the clock period shall be 25 ns
- a reset pulse of 100 ns shall be used to reset the counter
- the counter shall run long enough for all possible values of the counter to be displayed on the output, including the roll over to zero.

```vhdl
library IEEE;
use ieee.std_logic_1164.all;


entity tb_counter is
end entity;

architecture tb of tb_counter is

  signal clk     : std_logic := '0';
  signal arst_n  : std_logic;
  signal counter : std_logic_vector(3 downto 0);

  signal clk_ena    : boolean := false;
  signal clk_period : time    := 25 ns;

begin

  counter_i : entity work.counter
    port map (
      clk     => clk,
      arst_n  => arst_n,
      counter => counter
      );

  clk <= not clk after clk_period/2 when clk_ena else '0';

  p_stimuli : process is
  begin
    --default values
    clk_ena <= true;
    arst_n  <= '1' after 0 ns, '0' after 10 ns, '1' after 110 ns;
    wait for 25*clk_period;
    clk_ena <= false;
    wait;
  end process;

end architecture;
```

The important part of this question is to demonstrate the key concepts/structure of how to write a test bench in VHDL. It is expected that the candidate shall demonstrate the knowledge about

- the need for an empty entity description
- component declaration or direct instantiation, and port map of the device under test
- how to generate a running clock of period 25 ns
- the need for a stimuli process without a sensitivity list, and correct ordering of statements to generate the reset signals and the number of clock cycles need for the counter to reach all possible values.
- The need to stop all signals transitions and end the process with a *wait* statement.



Problem 3 a) [3 points]

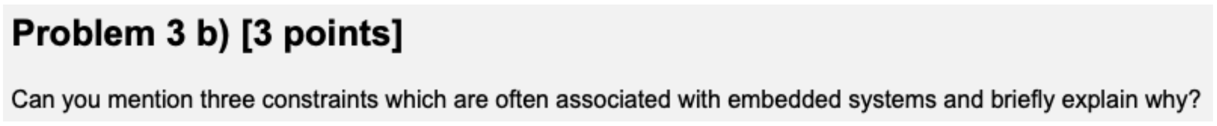
Can you briefly explain what is meant by an embedded system?

An embedded system is a computing system designed to perform one or more specific functions with specific system constraints.

Both the aspect of being a system with

- A specific task, and
- with specific system constraints

must be included/demonstrated in the answer. Only one of them will give 1-2 points depending on the wording.



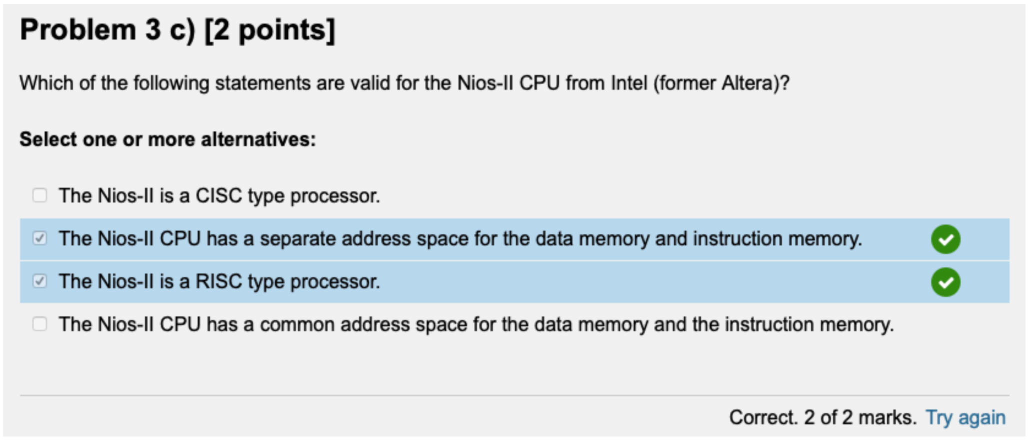
Problem 3 b) [3 points]

Can you mention three constraints which are often associated with embedded systems and briefly explain why?

Constraints that are often associated with embedded systems are:

- Physical (size, weight). As embedded systems tend to be small, there is often limited space for electronics or mechanical components. Likewise, there may be weight constraints for wearables and flight-based systems.
- Power. As embedded systems often run on batteries, there may be requirements to use as little power as possible to extend the life-time before a battery needs to be exchanged or charged.
- Cost. Embedded systems are commercial consumer products and to increase sales, there may be a requirement to keep the cost of the product low.
- Time. Some systems may control time critical functions and may have a requirement to be time deterministic. As such, they may be considered to the a Real-time embedded system.

Other constraints which are mentioned and explain in a convincing manner may also be accepted. 1 point will be given for each constraint. Only listing the constrain without an explanation will give 0.5 points per constraint.



Problem 3 c) [2 points]

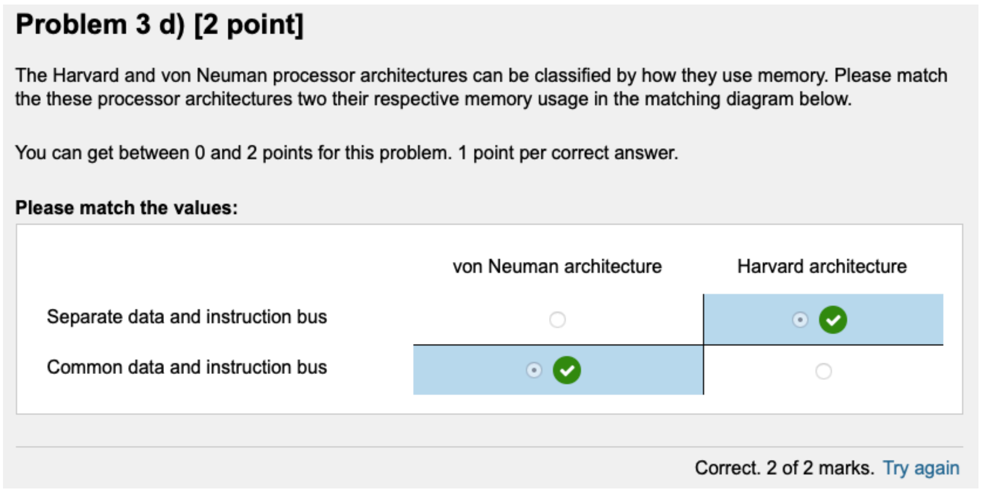
Which of the following statements are valid for the Nios-II CPU from Intel (former Altera)?

Select one or more alternatives:

- [ ] The Nios-II is a CISC type processor.
- [✓] The Nios-II CPU has a separate address space for the data memory and instruction memory. ✓
- [✓] The Nios-II is a RISC type processor. ✓
- [ ] The Nios-II CPU has a common address space for the data memory and the instruction memory.

Correct. 2 of 2 marks. Try again

There are two correct answers, each correct answer will give 1 point each.



Problem 3 d) [2 point]

The Harvard and von Neuman processor architectures can be classified by how they use memory. Please match the these processor architectures two their respective memory usage in the matching diagram below.

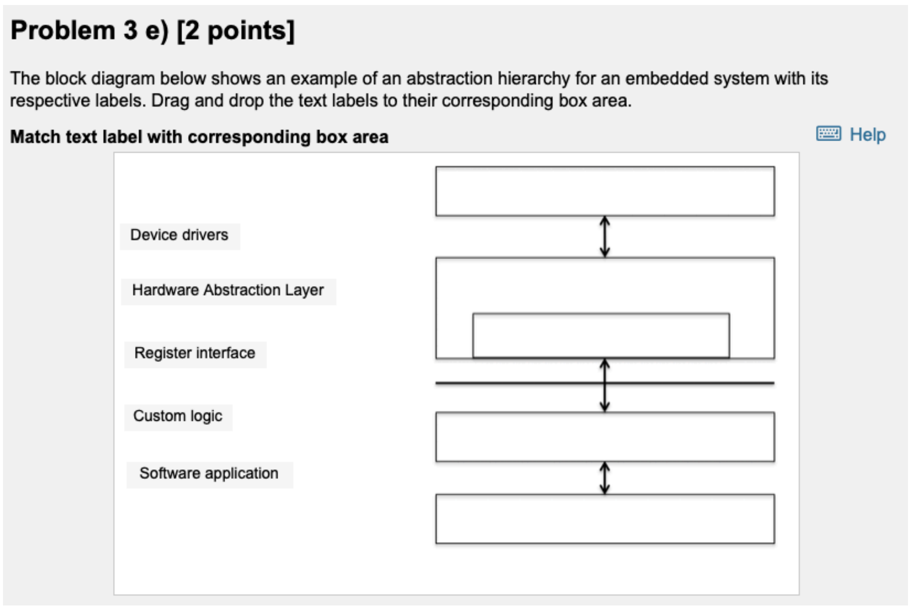
You can get between 0 and 2 points for this problem. 1 point per correct answer.

Please match the values:

	von Neuman architecture	Harvard architecture
Separate data and instruction bus	○	● ✓
Common data and instruction bus	● ✓	○

Correct. 2 of 2 marks. Try again

There are two correct answers, each correct answer will give 1 point each.

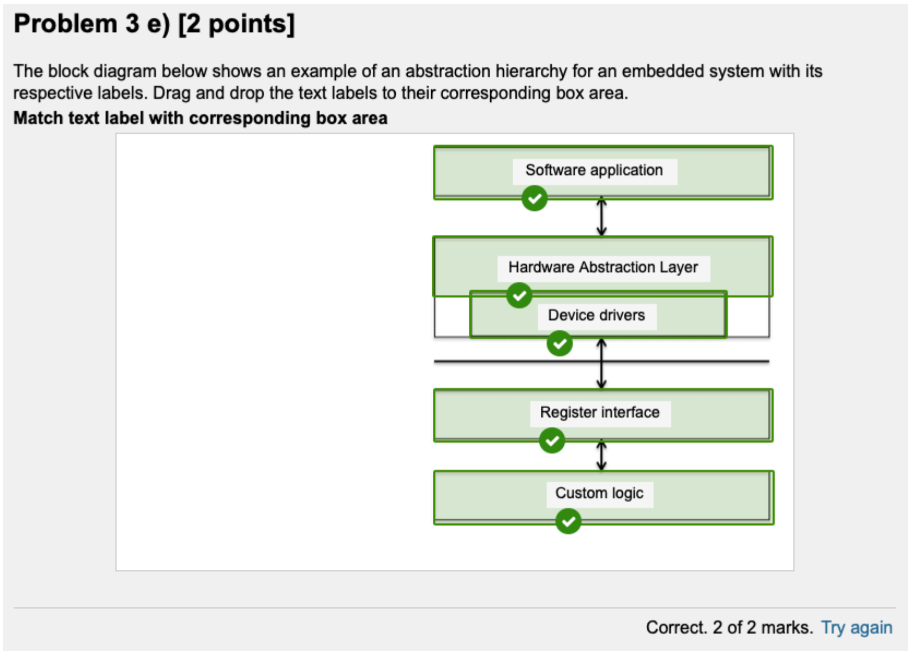


Problem 3 e) [2 points]

The block diagram below shows an example of an abstraction hierarchy for an embedded system with its respective labels. Drag and drop the text labels to their corresponding box area.

Match text label with corresponding box area

Each correct placing of the text labels will give 0.4 points to a total of 2 points. If the ordering is correct but slightly shifted, an overriding of the automatic score may be considered.

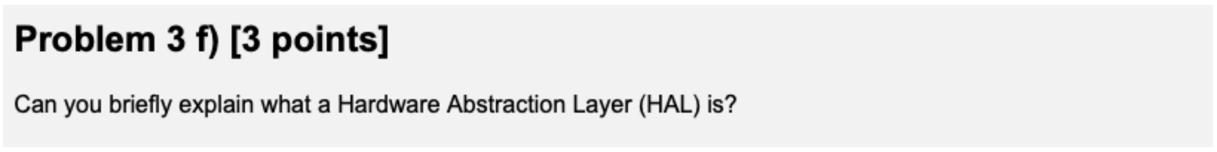


Problem 3 e) [2 points]

The block diagram below shows an example of an abstraction hierarchy for an embedded system with its respective labels. Drag and drop the text labels to their corresponding box area.

Match text label with corresponding box area

Correct. 2 of 2 marks. Try again



Problem 3 f) [3 points]

Can you briefly explain what a Hardware Abstraction Layer (HAL) is?

Answer:
The HAL is an abstraction layer implemented in software, between the physical hardware and the application software which runs on that computer. The HAL allows the computer operating system to interact with a hardware device at a general or abstract level rather than at a detailed level. It allows for device- independent programming by providing a simple device driver interface for programs to connect to the underlying hardware. It can be seen as the "glue" between the low-level devices and the standard libraries found on most system that provide c-compiles.

The answer shall demonstrate an understanding of that

1. the HAL is used as a middle/abstraction layer between the software and the hardware,
2. and that it provides a device-independent programming functionality. That is, the software designer does not need to have low-lever knowledge of the hardware.

Both of these aspects must be mentioned to get full score.

Problem 3 g) [10 points]

In this problem you will write the c-code to interface a module which is memory mapped to the Nios-II CPU. The module is used to read the data from a temperature sensor.

The module's base address in the CPU memory is:
#define MODULE_BASE_ADDR 0x00002000

Table 1 shows the register map description for the module's data and control register.

Table 1: Register map description

Register	Address/ Regnum	Type	Description
Data register [31:16]	0x0		Not in use
[15:0]		Read	**Sensor data**: Data from sensor. Stored as two's complement value
Control register [31:4]	0x1		Not in use
[2:3]		Write	**Resolution:** 2-bit resolution value 00: 0.5 degrees C / LSB 01: 0.25 degrees C / LSB 10: 0.125 degrees C / LSB 11: 0.0625 degrees C / LSB
[1]		Read	**Complete:** '1' – Transaction complete. This bit will be reset when read.
[0]		Write	**Run:** '1' – Start measurement. This bit will automatically be reset when the measurement has started.

The temperature is stored as a 16-bit two's complement value in the *Sensor data* register, and thus capable of representing both positive and negative values.

The IORD and IOWR macros in table 2, which are provided by the Nios-II Hardware Abstraction Layer (HAL), can be used to communicate with the memory mapped module.

Table 2: Definition of IORD and IOWR macro functions.

IOWR(BASE, REGNUM, DATA)	Writes the value DATA to the register at the offset REGNUM in a device with base address BASE. Registers are assumed to be offset by the address width of the bus.
IORD(BASE, REGNUM)	Reads the value of the register REGNUM in a device with base address BASE. Registers are assumed to be offset by the address width of the bus. IORD returns the 32-bit value of the register being read.

Your task is to write the c-code for the function *read_data()*. The function is defined below:

float read_data(alt_u8 resolution);

Function description:

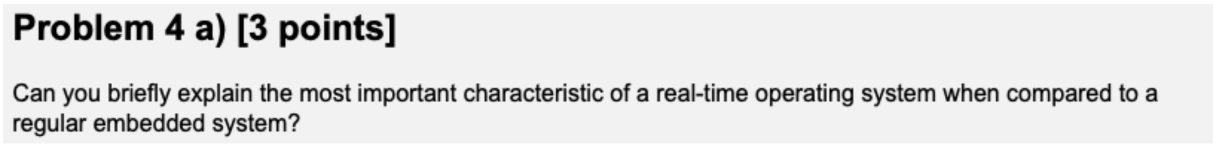
- The function shall configure the temperature sensor to use the value of the resolution variable which is provided as an argument to the function.
- The function shall then request a measurement to be performed by the sensor
- When the *Complete* bit is set, the function shall return a floating point value of the temperature.

Answer:

```c
float read_data(alt_u8 resolution){

    alt_u32 ctrl_reg = 0;
    alt_u32 data = 0;
    float value = 0;
    //Set resolution and request measurment
    //Must be performed in the same IOWR operation
    ctrl_reg = ((resolution & 0x3) << 2) | 0x1;
    IOWR(MODULE_BASE_ADDR,0x1,ctrl_reg)

    //Wait for complete bit to be set
    while (data != 1) {
        data = (IORD(MODULE_BASE_REG,0x1) >> 1) & 0x1;
    }

    // read temperature data (16-bit)
    data = IORD(MODULE_BASE_REG,0x0) & 0xffff;
    // cast to signed 16 bit in order to convert from two's complement
    value = (alt_16)data;
    // Set resolution

    // any other conditional sentence doing the same operation is also considered correct.
    // E.g. use of an if-sentence
    switch(resolution)
    {
    case 0:
        value = value * 0.5;
        break;
    case 1:
        value = value * 0.25;
        break;
    case 2:
        value = value * 0.125;
        break;
    case 3:
        value = value * 0.0625;
        break;
    }
    return value;
}
```

This question is meant to demonstrate a basic understanding of how to write low-level / hardware close c-code. That is, how to manipulate data on a register level. To get full score the required functionality needs to be implemented:

- write operation to control register (both resolution bits and run bit must be set in one operation)
- check and wait for complete bit to be set
- read data from data register and casting to 16-bit signed value
- apply resolution to read data to return float value

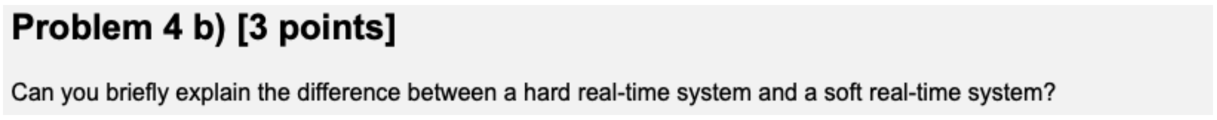


Problem 4 a) [3 points]

Can you briefly explain the most important characteristic of a real-time operating system when compared to a regular embedded system?

Compared to a regular embedded system, a real-time operating system also has time requirements. For a real-time system the correctness of the system does not only depend on the logical result of the computation, but also on the time when the result is generated.

The answer needs to demonstrate that the timing requirement is the essential characteristic that differentiates a real time system compared to a regular embedded system.



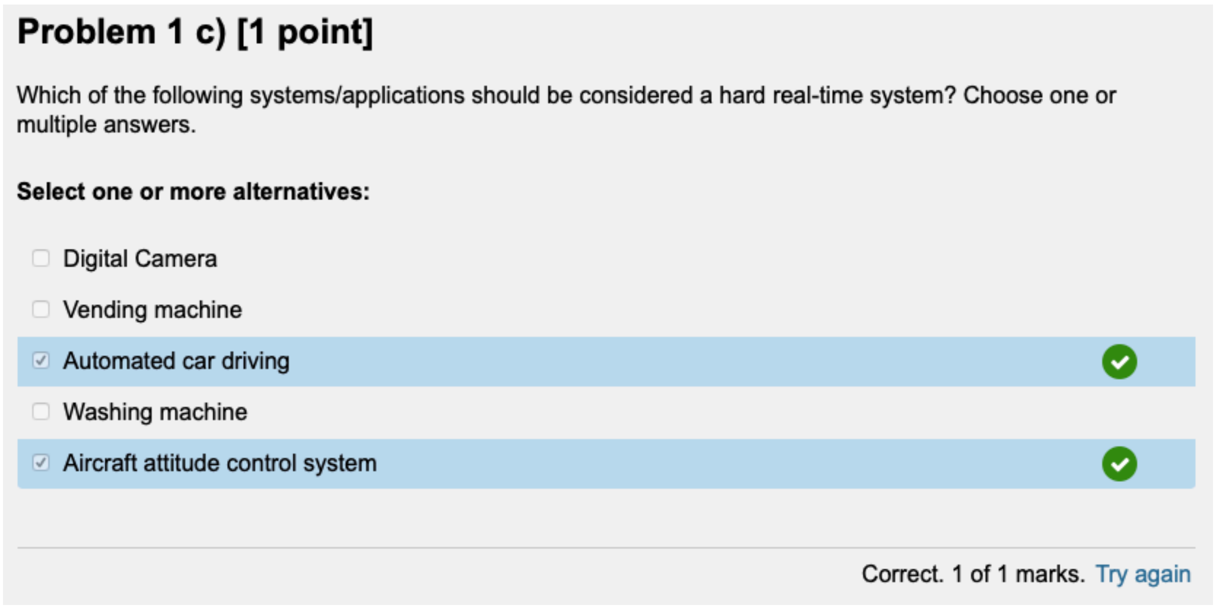
Problem 4 b) [3 points]

Can you briefly explain the difference between a hard real-time system and a soft real-time system?

Answer:

A hard real-time system is one in which failure to meet a single deadline or timing requirement may lead to a complete and catastrophic system failure. In contrast a soft real-time system is a system in which performance is degraded but not destroyed by failure to meet the deadline.

Describing only what a hard or soft real time system is not considered a correct answer but may give 1 point. It is essential to demonstrate the difference between the two types.



Problem 1 c) [1 point]

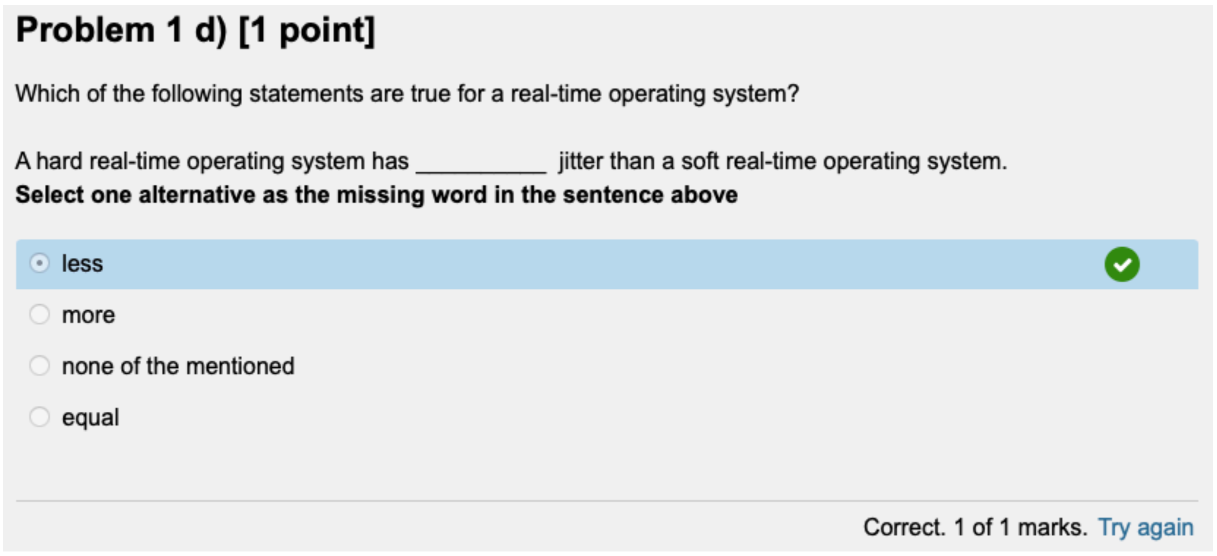
Which of the following systems/applications should be considered a hard real-time system? Choose one or multiple answers.

Select one or more alternatives:

- ☐ Digital Camera
- ☐ Vending machine
- ☑ Automated car driving ✓
- ☐ Washing machine
- ☑ Aircraft attitude control system ✓

Correct. 1 of 1 marks. Try again

Both an automated car driving system or an aircraft attitude control system may have serious or fatal consequences if failing. If failing, this is not acceptable. However, one can accept some degradation in the performance of digital camera, vending machine or washing machine for shorter periods.



Problem 1 d) [1 point]

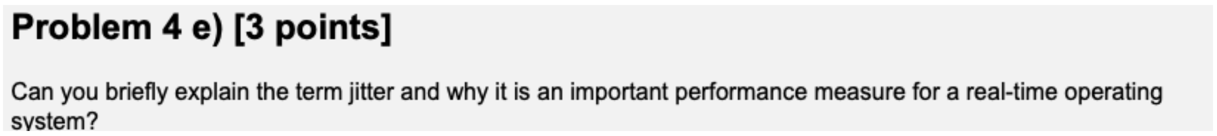
Which of the following statements are true for a real-time operating system?

A hard real-time operating system has ___________ jitter than a soft real-time operating system.
Select one alternative as the missing word in the sentence above

- [✓ less] ✓
- more
- none of the mentioned
- equal

Correct. 1 of 1 marks. Try again

Jitter is a measure of the error in the timing of subsequent iterations of a program loop. As a hard real-time system should be more time deterministic than a soft real-time system, a hard real-time system should have less error and therefore less jitter.

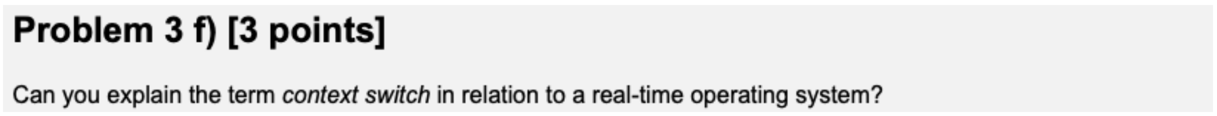


Problem 4 e) [3 points]

Can you briefly explain the term jitter and why it is an important performance measure for a real-time operating system?

Answer:
Jitter is the amount of error in the timing of a task over subsequent iterations of a program loop. Jitter is an important performance measure since it gives information about punctuality/accuracy of the system. Low jitter means that a task will take very close to the same amount of time to execute each time it runs.

Explaining only what jitter is, or only why it is important gives 1-2 points depending on the wording. Both needs to be included to get full score.

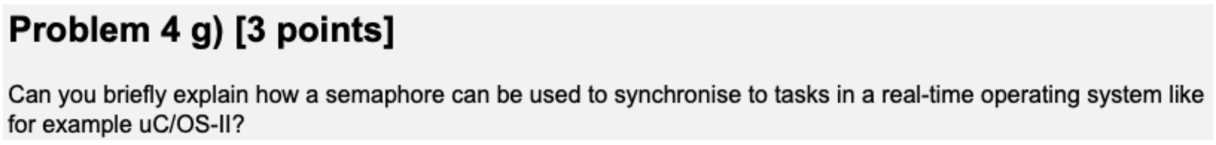


Problem 3 f) [3 points]

Can you explain the term *context switch* in relation to a real-time operating system?

Answer:
A context switch takes place when the system suspends the current running task in order to execute another higher priority task. In this case the RTOS must save all the information needed to eventually resume the suspended task. This is an essential part of an RTOS since it allows a system to multitask.

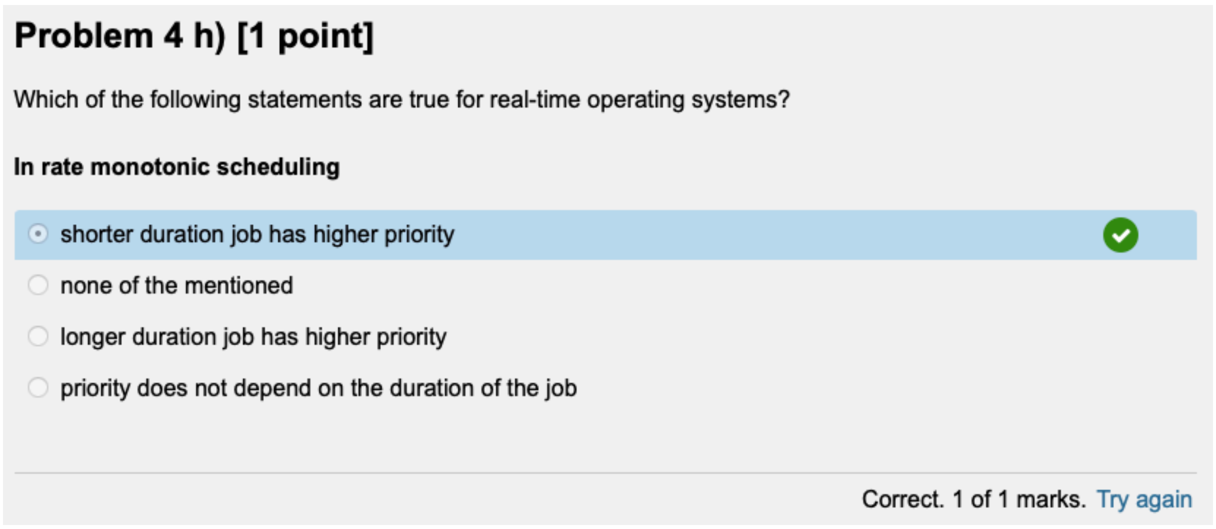
Explaining only that a context switch is suspending one task to run another task gives 1-2 points depending on the wording. To get full score the answer should also mention that a context switch also involves saving the state of the currently running task with the intention of eventually resuming this task.



Problem 4 g) [3 points]

Can you briefly explain how a semaphore can be used to synchronise to tasks in a real-time operating system like for example uC/OS-II?

Answer:
A semaphore can be used to synchronize two tasks if one task is pending the semaphore and another task is posting the semaphore. If the semaphore is initialized as 0, the task pending the semaphore will only be executed whenever the other task posts the same semaphore.



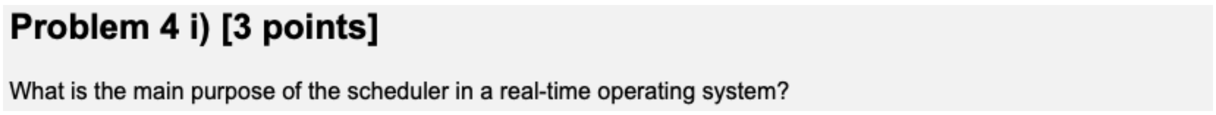
Problem 4 h) [1 point]

Which of the following statements are true for real-time operating systems?

In rate monotonic scheduling

- [✓ shorter duration job has higher priority] ✓
- none of the mentioned
- longer duration job has higher priority
- priority does not depend on the duration of the job

Correct. 1 of 1 marks. Try again

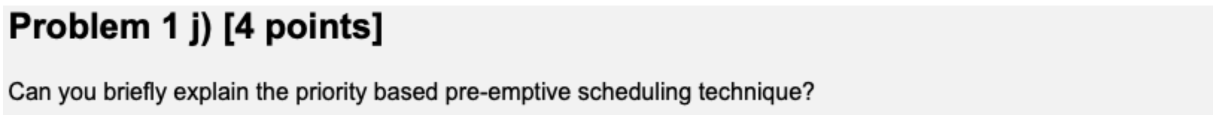


Problem 4 i) [3 points]

What is the main purpose of the scheduler in a real-time operating system?

The main purpose of the scheduler is to determine which task to run at any given moment.

A less precise answer may give 1-2 points depending on the wording. But the essential part is to demonstrate the understanding of that it is responsible to decided which tasks to run.



Problem 1 j) [4 points]

Can you briefly explain the priority based pre-emptive scheduling technique?

Using a priority based pre-emptive scheduling technique, ensures that at a given time the processor executes the highest priority task of all tasks which are currently ready to be executed. Here, pre-emption is the act of temporarily interrupting a task being carried out by the real-time kernel, in order to run a task with a higher priority.