Front page

1. Front page
2. FYS4220/9220 Real time and embedded data systems
3. Monday 3. December 2018
   Exam hours: 14:30 - 18:30 (4 hours)
4. All exam questions must be answered. Each question may be weighted differently. For each question the maximum number of points is specified.
   The maxium number of points for the complete question set is 100 points.
5. Permitted material: None

1 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 rising_edge(clk) then
    if enable = '1' then
      output <= data;
    end if;
  end if;
end process;
```

Select one or more alternatives:

- [ ] data
- [ ] output
- [ ] enable
- [ ] clk

Maximum marks: 1

2 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

Maximum marks: 1

3 Problem 1 c) [1point]

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 input = "00" then
    if test = '1' then
      data <= (others => '0');
    else
      data <= result_a;
    end if;
  elsif input = "01" then
    data <= result_b;
  elsif input = "10" then
    data <= result_c;
  else
    data <= (others => '0');
  end if;
end process;
```

Select one or more alternatives:

- [ ] input
- [ ] results_b
- [ ] test
- [ ] data
- [ ] result_a
- [ ] result_c

Maximum marks: 2

4 Problem 1 d) [8 points]

What is the purpose of the sensitivity when describing a VHDL process?

Write your answer here:

Explain why the sensitivity list should be used differently for a combinational process compared to a synchronous process.

Write your answer here:

Explain why the sensitivity list should be used differently for an asynchronous reset compared to a synchronous reset.

Write your answer here:

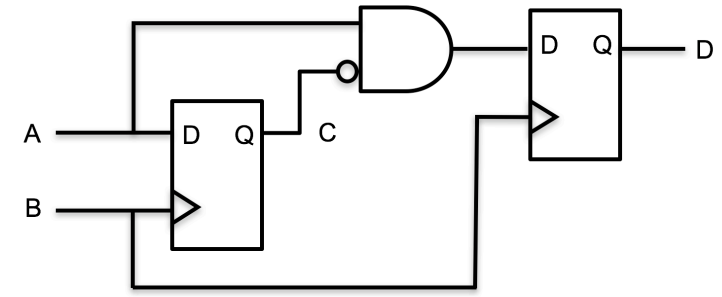
Maximum marks: 8

5 Problem 1 e) [3 points]

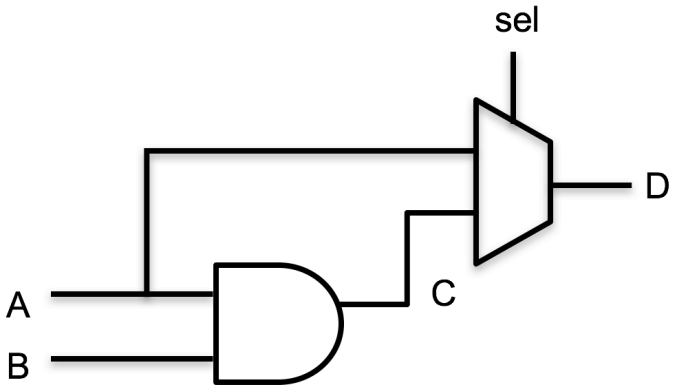
The diagram below shows different circuits next to a matching matrix. Match the circuit type with the correct circuit diagram.

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

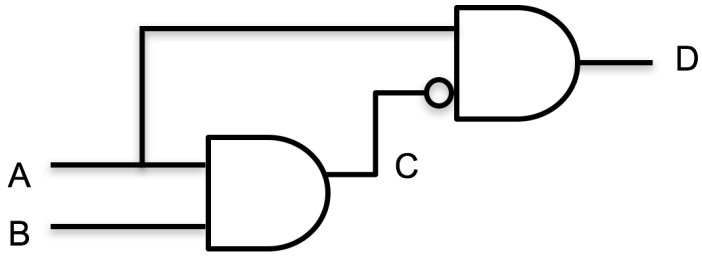
Please match the circuit diagram with the correct ciruit type:



Circuit B



Circuit C



Circuit A

	Synchronous	Combinational
Circuit B	○	○
Circuit C	○	○
Circuit A	○	○

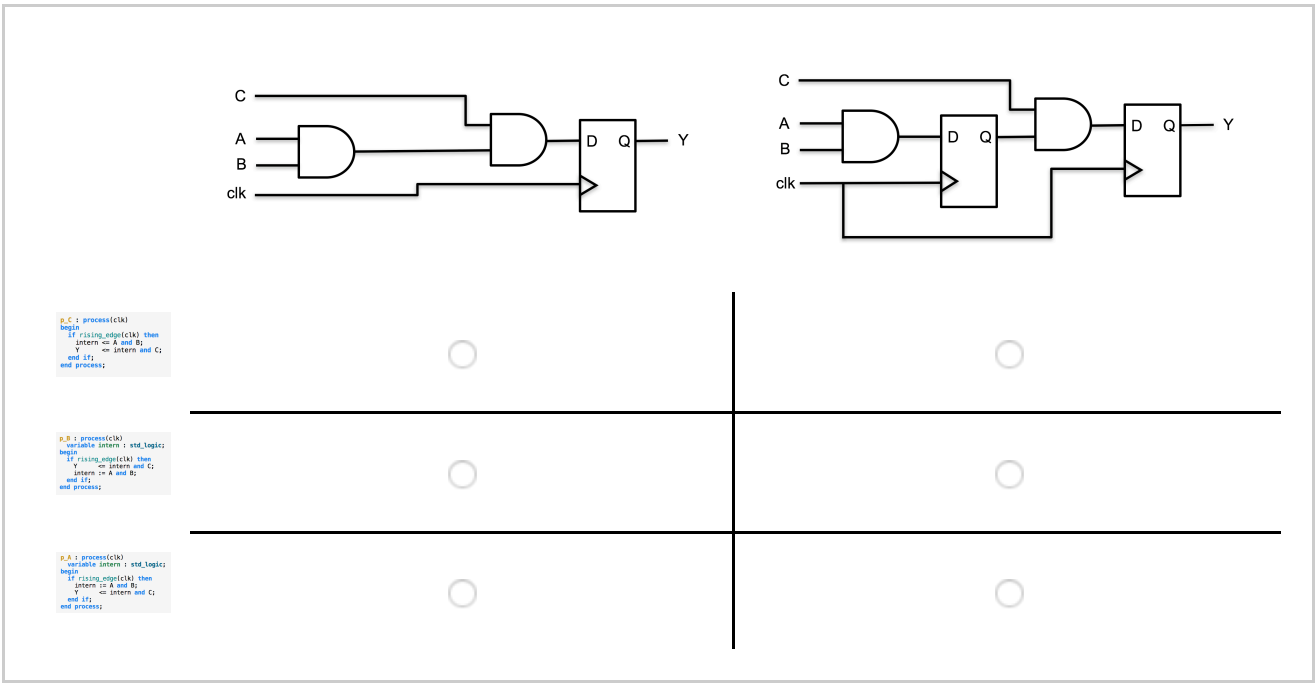
Maximum marks: 3

6 Problem 1 f) [3 points]

The diagram below shows two different circuits and three corresponding VHDL descriptions. Match each VHDL description with its correct circuit diagram.

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

Please match the values:

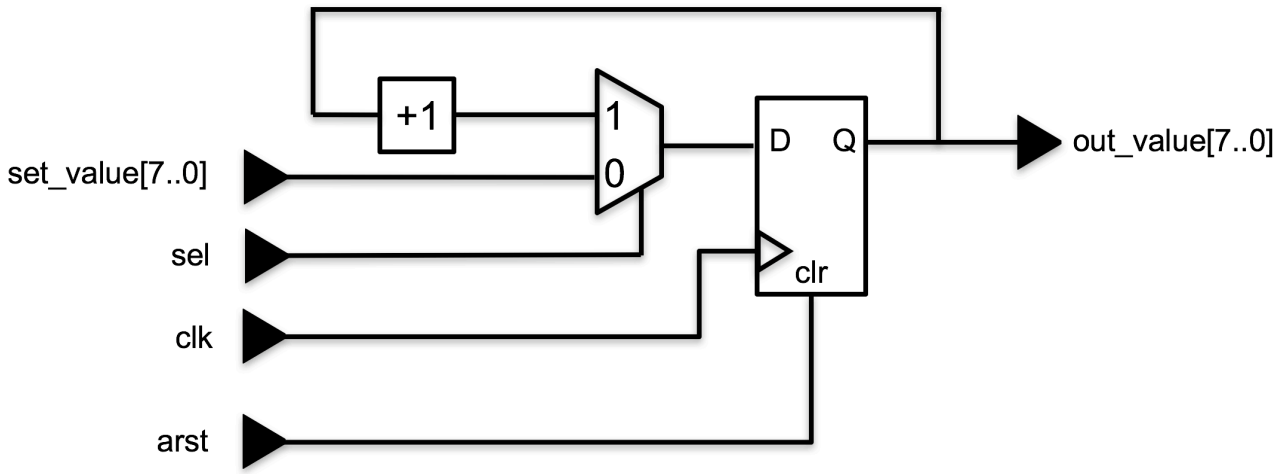


Maximum marks: 3

7 Problem 1 g) [3 points]

Write the VHDL description for the entity of the circuit shown below. Include the relevant library statements above the entity description. The name of the entity is *add1*.

Circuit description:
The circuit increments an internal 8-bits counter on every rising edge of the clock as long as the input *sel* = '1'. When *sel* = '0', the internal counter is set to the value given by the input *set_value*. The internal counter is connected to the output *out_value*. The counter is reset to zero using the input *arst*, which is an asynchronous active high reset signal.



Write your code here:

```
1
```

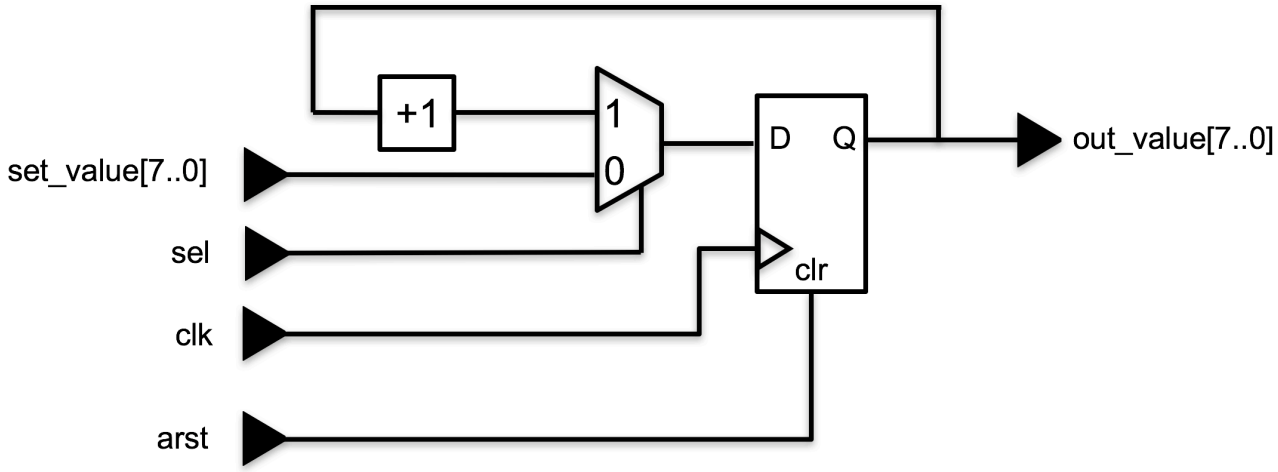
Maximum marks: 3

8 Problem 1 h) [10 points]

Write the VHDL description for the circuit shown below. Include also the architecture definition and any declarations of internal signals. The entity description has already been

written in the previous problem and should not be included here.

Circuit description:
The circuit increments an internal 8-bits counter on every rising edge of the clock as long as the input *sel* = '1'. When *sel* = '0', the internal counter is set to the value given by the input *set_value*. The internal counter is connected to the output *out_value*. The counter is reset to zero using the input *arst*, which is an asynchronous active high reset signal.



Write your code here:

```
1
```

Maximum marks: 10

Problem 2 - information

In this problem, you will write the VHDL description for a Finite State Machine which controls a basic SPI master module. The ASM diagram of the state machine is shown in the attached PDF file, while the entity and architecture description is shown below.

- The state machine has 3 states: spiStart, spiActive, spiEnd.
- The state machine has a synchronous active high reset. The state machine should be reset to the state spiStart.
- A control signal *tx_start* is used to enable the SPI transaction.
- A control signal *tx_done* is used to signal end of SPI transaction.

Your task is to:

Problem 2 a)

Write the necessary statements to declare the enumerated types and signals for the state machine.

Problem 2 b)

Write the VHDL description of the process proc_state according to the ASM diagram in the attached PDF file.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity spi_master is
6    generic(clk_period : integer := 50_000_000;
7            spi_period : integer := 1_000_000;
8            no_bits    : integer := 8
9            );
10   port(
11     clk      : in  std_logic;           -- system clk
12     reset    : in  std_logic;           -- synhronous active high
13     tx_data  : in  std_logic_vector(no_bits-1 downto 0);  -- data to be sent from master
14     rx_data  : out std_logic_vector(no_bits-1 downto 0);  -- data received from slave
15     mosi     : out std_logic;           -- serial line for master out slave in
16     miso     : in  std_logic;           -- serial line for master in slave out
17     sclk     : out std_logic;           -- spi clock
18     ss       : out std_logic;           -- slave select
19     tx_start : in  std_logic;           -- enable signal to start transaction
20     tx_done  : out std_logic   -- control signal indicating end of transaction
21     );
22 end entity;
23
24 architecture behave of spi_master is
25
26   -- declaration of spi relevant signals and constants
27   signal spi_clk     : std_logic;
28   signal spi_clk_r   : std_logic;
```

```
29    signal spi_counter : integer range 0 to (clk_period/(2*spi_period));
30    constant spi_delay : integer := (clk_period/(2*spi_period))-1;
31
32    signal bit_counter : integer range 0 to no_bits := spi_delay;
33    signal tx_reg      : std_logic_vector(no_bits downto 0);
34    signal rx_reg      : std_logic_vector(no_bits downto 0);
35
36    -- Problem 2 a): declaration of state machine
37
38
39    -- End Problem 2 a)
40
41  begin
42
43    sclk    <= spi_clk;
44    mosi    <= tx_reg(tx_reg'left);
45    rx_data <= rx_reg;
46
47    proc_spi_timing : process(clk)
48    begin
49      if rising_edge(clk) then
50        if spi_counter > 0 then
51          spi_counter <= spi_counter - 1;
52        else
53          spi_counter <= spi_delay;
54        end if;
55      end if;
56    end process;
57
58    proc_data_tx : process(clk)
59    begin
60      if rising_edge(clk) then
61        spi_clk_r <= spi_clk;
62        if state = spiStart then
63          tx_reg <= tx_data;
64        end if;
65        if spi_clk_r = '1' and spi_clk = '0' then
66          tx_reg <= tx_reg(tx_reg'left-1 downto 0) & tx_reg(0);
67        end if;
68      end if;
69    end process;
70
71    proc_data_rx : process(clk)
72    begin
73      if rising_edge(clk) then
74        if (spi_clk_r = '1' and spi_clk = '1') then
75          rx_reg <= rx_reg(rx_reg'left-1 downto 0) & miso;
76        end if;
77      end if;
78    end process;
79
80    -- Problem 2 b) : Write the VHDL description for the state machine process proc_state
81    proc_state :
82
83
84    -- End Problem 2 b)
85
86  end architecture;
87
88
```

9 Problem 2 a) [5 points]

Write the necessary statements to declare the enumerated types and signals for the state machine.

- The state machine has 3 states: *spiStart, spiActive, and spiEnd.*

Declare the signals and enumerated types for the state machine here:

```
1
```

Maximum marks: 5

10 Problem 2 b) [15 points]

Write the VHDL description for the process *proc_state* according to the ASM diagram in the attached PDF file.

- The state machine has 3 states: spiStart, spiActive, spiEnd.
- The state machine has a synchronous active high reset. The state machine should be reset to the state spiStart.
- A control signal *tx_start* is used to enable the SPI transaction.
- A control signal *tx_done* is used to signal end of SPI transaction.

Write your code here:

```
1
```

Maximum marks: 15

11 Problem 3 a) [4 points]

Can you explain what is meant by an embedded system?

Write your answer here:

Maximum marks: 4

12 Problem 3 b) [4 points]

Can you explain the difference between a softcore and a hardcore processor?

Write you answer here:

Maximum marks: 4

13 Problem 3 c) [2 points]

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

You can get 2 points for this problem. An incorrect answer gives 0 points.

Select one or more alternatives:

- [ ] The Nios-II is a hardcore processor.
- [ ] The Nios-II is a softcore processor.
- [ ] The Nios-II processor is a von Neuman type architecture
- [ ] The Nios-II processor is a Harvard type architecture

Maximum marks: 2

14 Problem 3 d) [4 points]

Harvard and *von Neuman* are two types of architectures for a CPU. Can you explain the main difference between these two types of architectures?

Write your answer here:

Maximum marks: 4

15 Problem 3 e) [4 points]

The Nios-II architecture provides memory mapped I/O access through the Avalon interface. Can you explain what memory mapped I/O access is?

Write your answer here:

Maximum marks: 4

16 Problem 3 f) [7 points]

In this problem you will write the c-code to interface an SPI master module which is memory mapped to the Nios-II CPU.

The module's base address in the CPU memory is:
#define SPI_BASE_ADDR 0x00013000

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
[15:8]		Read	**RX data:** Contains data received from SPI slave
[7:0]		Write	**TX data**: Data written to this register will be sent to SPI slave
Control register [31:3]	0x1		Not in use
[2]		Write	**Reset:** Will reset module when set to 1
[1]		Read	**TX/RX Done:** '1' – Transaction complete
[0]		Write	**TX/RX Start:** '1' – Start transactions

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:

alt_u32 read_data(alt_u8 *rx_data);

Function description:

- If the *TX/RX Done* bit is set, the function will read the data register and copy the 8-bit value of the *RX data* register into the pointer **rx_data* provided in the function's argument list.
- If the *RX/TX Done* bit is set to '1', the function returns the value 0.
- If the *RX/TX Done* bit is '0', the function returns the value 1.

Write the c-code for the function read_data() here:

```
1
```

Maximum marks: 7

17 Problem 4 a) [4 points]

What is the main difference between a real-time embedded system and a regular embedded system?

Write your answer here:

Maximum marks: 4

18 Problem 4 b) [1 point]

Which of the following scheduling types is used by the uC/OS-II Real Time Kernel?

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

Select one alternative:

- Priority-based non-preemptive
- Priority-based pre-emptive
- Round-robin
- Earliest deadline first
- Rate monotonic

Maximum marks: 1

19 Problem 4 c) [4 points]

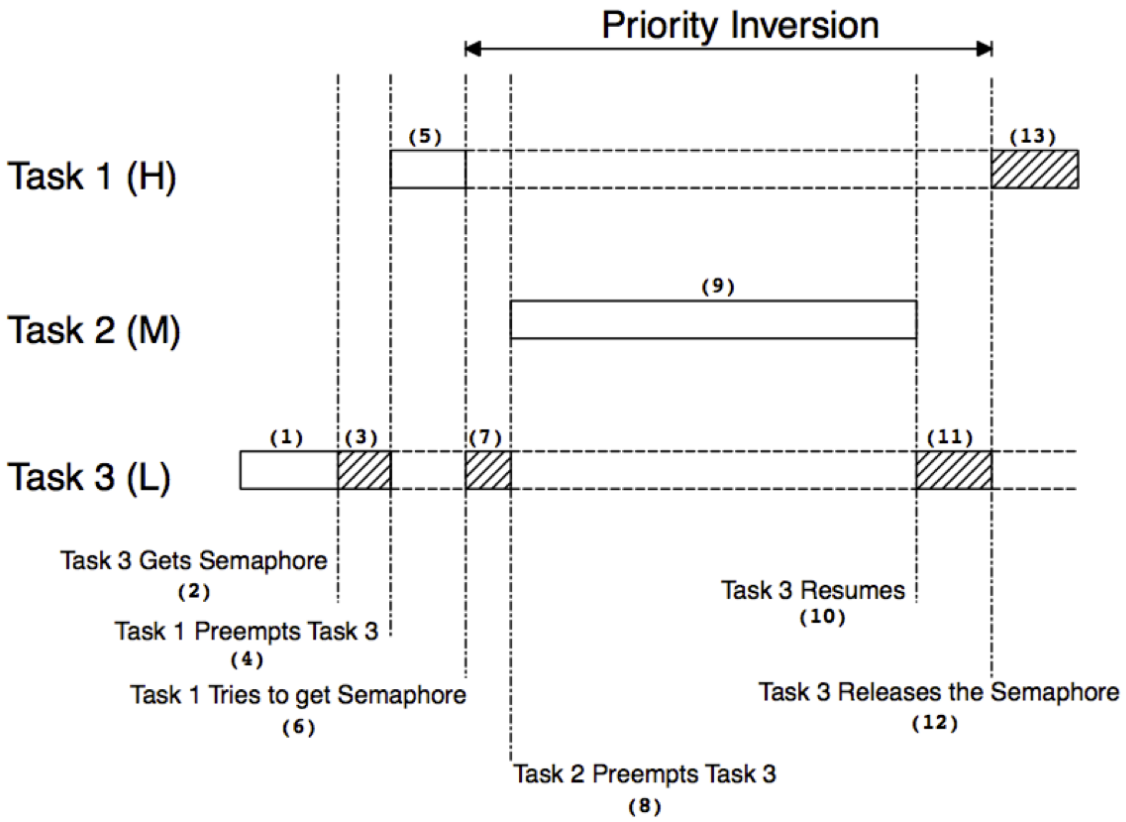
Can you explain the Round-robin scheduling technique?

Write your answer here:

Maximum marks: 4

20 Problem 4 d) [4 points]

Using the diagram below, can you explain the concept of priority inversion?



Write your answer here:

Maximum marks: 4

21 Problem 4 e) [4 points]

Can you explain how the problem of priority inversion be can be controlled / reduced?

Write your answer here:

Maximum marks: 4

22 Problem 4 f) [4 points]

Can you explain the two different ways a semaphore can be used in a real-time system?

Write your answer here:

Maximum marks: 4

23 Problem 4 g) [4 points]

The two tasks below use a mailbox service to send a message from *task1* to *task2.* The message is stored in the variable *t1*, and *task1* uses the OSMboxPost function to send a pointer to this location through the mailbox service.

Task2 makes uses of the message value to perform a calculation using the function *calc_message(),* which is defined in another file. Finally, *task2,* prints the message and calculation result to the terminal.

```
1
2  void task1 (void* pdata)
3  {
4    INT8U error_code = OS_NO_ERR;
5    int t1;
6
7    while(1)
8      {
9        t1 = OSTimeGet();
10       error_code = OSMboxPost(msg_box, (void*)&t1);
11       OSTimeDlyHMSM(0,0,1,0);
12     }
13 }
14
15
16 void task2 (void* pdata)
17 {
18   INT8U error_code = OS_NO_ERR;
19   int *msg_rx;
20   int result;
21
22   while(1)
23     {
24       //Pend message from task 1
25       msg_rx = (int*) OSMboxPend(msg_box,0,&error_code);
26       result = calc_message(*msg_rx);
27
28       OSSemPend(shared_terminal_sem,0,&error_code);
29       printf("Task2 received message: %d, result: \n", *msg_rx, result);
30       OSSemPost(shared_termial_sem);
31     }
32 }
33
```

What is the potential problem with this implementation, and can you briefly explain how this problem can be avoided?

Write your answer here:

Maximum marks: 4