UNIVERSITY OF OSLO

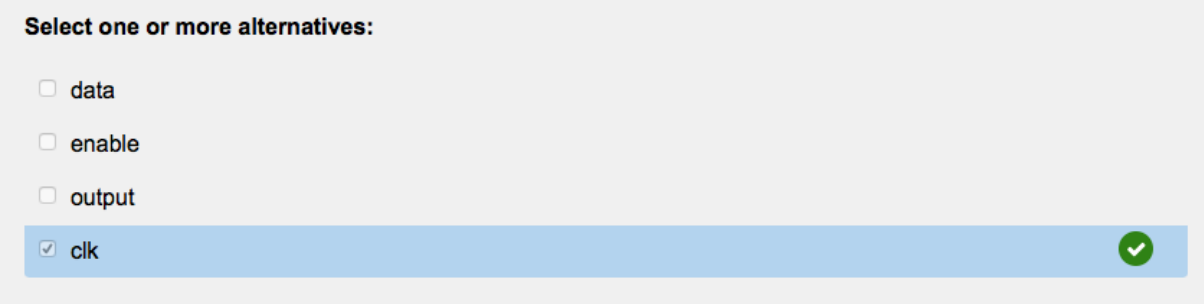
Faculty of Mathematics and Natural Sciences

Exam in:	**FYS4220**
Day of exam:	**3 December 2018**
Exam hours:	**14.30 – 18.30**
Appendices:	**none**
Permitted materials:	**none**

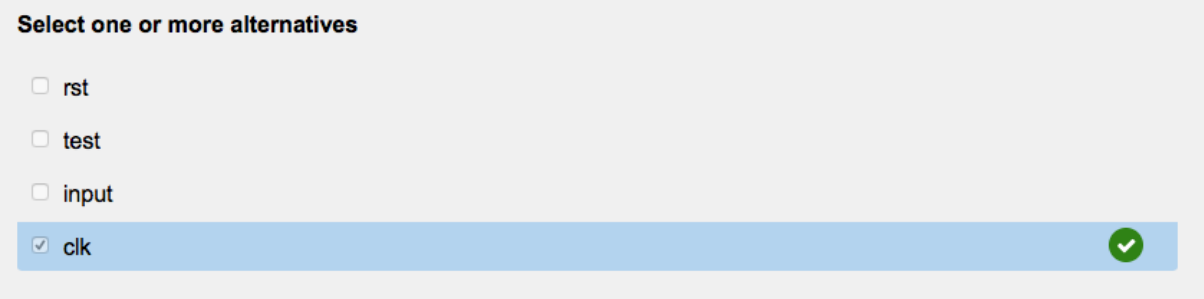
Answers

Problem 1 (30 points)

a) (1 point)



b) (1 point)



c) (1 point)



d) (8 points)

I (2 points)
The signal sensitivity list is used to specify which signal should cause the process to re-evaluate (activate). Whenever any event occurs on one of the signals in the sensitivity list, the process is re-evaluated.

II (3 points)
For a synchronous process the process should only be activated when there is an event on the clock signal. The clock signal is therefore the only signals that must be included in the sensitivity list. The only exception is when using an asynchronous reset, in this case, both the clock signal and the asynchronous reset signals will activate the process and must therefore be included in the sensitivity list.

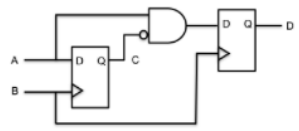
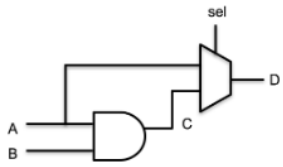
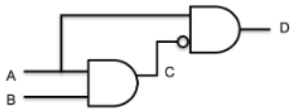
For a combinational process, an event on all the signals read by the process shall activate the process, and thus all signals read by the process must be included in the sensitivity list.

III (3 points)
For a process with a synchronous reset, the process will be reset synchronous to the clock. That is, the reset signal will be evaluated only when the condition rising_edge(clock) or falling_edge(clock) is true. The reset signal shall therefore not activate the process, and thus not be included in the sensitivity list.

For a process with an asynchronous reset, the process will be reset independent of the clock and must therefore also activate the process, and thus be included in the sensitivity list.

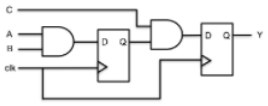
e) (3 points, 1 for each correct answer)

Please match the circuit diagram with the correct ciruit type:

	Synchronous	Combinational
 Circuit B	✓	
 Circuit C		✓
 Circuit A		✓

f) (3 points, 1 for each correct answer)

Please match the values:




```
p_C : process(clk)
begin
  if rising_edge(clk) then
    intern <= A and B;
    Y      <= intern and C;
  end if;
end process;
```

✓

```
p_B : process(clk)
  variable intern : std_logic;
begin
  if rising_edge(clk) then
    Y      <= intern and C;
    intern := A and B;
  end if;
end process;
```

✓

```
p_A : process(clk)
  variable intern : std_logic;
begin
  if rising_edge(clk) then
    intern := A and B;
    Y      <= intern and C;
  end if;
end process;
```

✓

g) (3 points)

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity add1 is
6    port(
7      clk       : in  std_logic;
8      sel       : in  std_logic;
9      arst     : in  std_logic;
10     set_value : in  std_logic_vector(7 downto 0);
11     out_value : out std_logic_vector(7 downto 0)
12     );
13 end entity;
14
```

h) (10 points)

```
15
16
17 architecture behav of add1 is
18
19
20 signal int_value : unsigned(7 downto 0);
21
22 begin
23   process(clk,arst) is
24   begin
25     if arst = '1' then
26       int_value <= (others => '0');
27     elsif rising_edge(clk) then
28       case sel is
29         when '0' =>
30           int_value <= unsigned(set_value);
31         when '1' =>
32           int_value <= int_value + 1;
33          when others =>
34           null;
35       end case;
36     end if;
37   end process;
38
39
40 out_value <= std_logic_vector(int_value);
41
42 end architecture;
```

Problem 2 (25%):

a) (5 points)

```
106   -- declaration of state machine
107   type state_type is (spiStart, spiActive, spiEnd);
108   signal state : state_type;
```

b) (15 points)

```
129   proc_state : process(clk)
130   begin
131     if rising_edge(clk) then
132       case state is
133         when spiStart =>
134           ss          <= '1';
135           spi_clk     <= '0';
136           bit_counter <= no_bits;
137           if tx_start = '1' then
138             ss      <= '0';
139             tx_done <= '0';
140             state   <= spiActive;
141           end if;
142
143         when spiActive =>
144           if spi_counter = 0 then
145             spi_clk <= not spi_clk;
146             if spi_clk = '1' then
147               bit_counter <= bit_counter - 1;
148             end if;
149             if bit_counter = 0 then
150               spi_clk <= '0';
151               state   <= spiEnd;
152             end if;
153           end if;
154         when spiEnd =>
155           ss      <= '1';
156           tx_done <= '1';
157           if tx_start = '0' then
158             state <= spiStart;
159           end if;
160
161         when others =>
162           state <= spiEnd;
163       end case;
164
165       if reset = '1' then
166         state <= spiStart;
167       end if;
168
169     end if;
170   end process;
```

Problem 3 (25%):

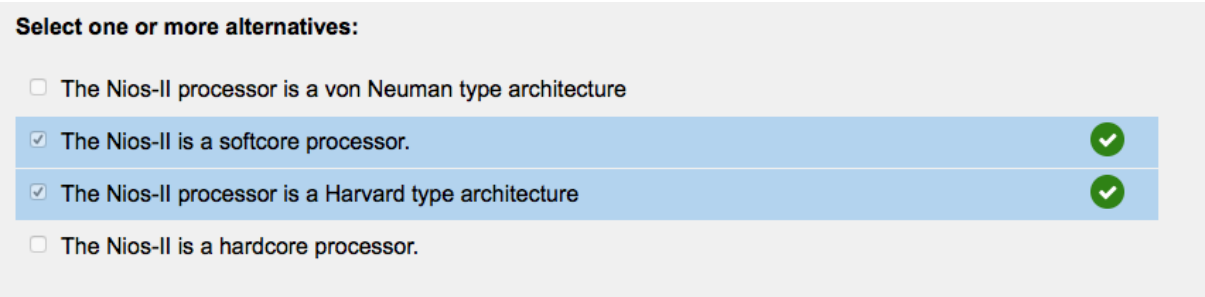
a) (4 points)

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

b) (4 points)

A softcore processor is configured from the logic resources inside the FPGA, and is as such implemented in the fabric of the FPGA. A hardcore processor is implemented as a dedicated predefined block in the actual silicon of the chip.

c) (2 points, 1 for each correct answer)



d) (4 points)

A Hardvard type architecture has a separate data and instruction bus, allowing transfers to be performed simultaneously on both busses. A von Neuman type architecture has only one bus which is used for both data and instructions.

e) (4 points)

Memory mapped I/O access means that all modules/peripherals connected to the Avalon bus are mapped into the address space of the master (cpu). These modules can thus be accessed by writing and reading data to the respective address location in the master address space.

f) (7 points)

```
8  alt_u32 read_data(alt_u8 *rx_data){
9    alt_u32 data = 0;
10   alt_u32 ctrl_reg = IORD(SPI_BASE_ADDR, 0x1);
11   if (ctrl_reg & (1 << 1))
12     {
13       data = IORD(SPI_BASE_ADDR, 0x0);
14       *rx_data = (alt_u8) ( (data >> 8) & 0xff );
15       return 0;
16     }
17   else
18     {
19       *rx_data = 0;
20       return 1;
21     }
22 }
```

Problem 4 (25%):

a) (4 points)

The main difference between a real-time system and a regular embedded system is that for the 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.

b)

Select one alternative:

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

c)

For an RTOS using the Round Robin scheduling technique, the tasks are executed in cyclic order. Time slices are assigned to each task in equal portions handling all tasks without priority. A task is interrupted if it is not completed by the given available time, and it is resumed next time a slot is assigned to the task.

d)

Priority inversion means that a high priority task (Task 1) has is pre-empted by a medium priority task (Task 2) effectively inverting the relative priorities of the two tasks. This is caused by a lower priority task (Task 3) which is blocking the usage of a shared resource (using a semaphore) also requested by the higher priority task. When a medium priority is

scheduled to run, and this task is not requesting the use of the shared resource, it pre-empts the low priority task and effectively behaves as it has a higher priority relative to the high priority task.

In the figure, Task 3 starts to run and gets the semaphore to use a shared resource. Task 1 then pre-empts Task 3. However, when task 1 wants to have access to the shared resource the semaphore is already taken, and Task 1 must wait for Task 3 to complete. Before Task 3 has complete the medium priority Task 2 pre-empts Task 3. Since Task 2 does is not using the shared resources, it is able to fully complete before Task 3. Task 3 then completes, releases the semaphore, and Task 1 can finally gain access to the semaphore.

e) (4 points)

A mutal exlusion semaphore (Mutex) can be used to reduce the priority inversion problem. Since a mutex knows its current owner, it is possible to promote the priority of the owner whenever a higher priority task starts waiting for the mutex. The lower priority task then effectively gets an equal or higher priority than the waiting task.

f) (4 points)

A semaphore can be used to synchronize the execution of tasks and to control the access to shared resources.

When used to control the access to a shared resource, the semaphore can be viewed as a key which is used to gain access to this resource. For a task to use a shared resource, it must first acquire the key. If the key is not available it has to wait until the key is available, that is, another task has completed to use the shared resource, before it can get access to it.
In this case, the semaphore must be initialized with the value 1.

When used to synchronize tasks, one task will post the semaphore and a second task will be waiting for the semaphore. In this way, the second task can be synchronized with the first task. In this case, the semaphore must be initialized with the value 0.

g) (4 points)

In this example, the mailbox is transferring a pointer to the actual messages stored in the variable t1 of task1. In task2 this pointer is used by the function calc_message(). The potential problem with this implementation is that the variable t1 may be updated with a new value before it is used by task2, or the function calc_message() has completed.
This can be solved by copying the messages to a local variable when received by task2.
An additional measure is for task1 to make sure not to update t1 if the mailbox is not empty.