UNIVERSITY OF OSLO

Faculty of Mathematics and Natural Sciences

Exam in: **FYS4220**
Day of exam: **17 December 2013**
Exam hours: **09.00 – 13.00**
This examination paper consists of 3 pages.
Appendices: **none**
Permitted materials: **none**

Make sure that your copy of this examination paper is complete before answering.

Problem 1 (25%):

a) What type of logic element (function) is implemented with the following conditional signal assignment?

Y <= A **when** C = '1' **else** B;

Answer:
This function implements a multiplexer where A and B is the input and C is the selecting signal.

b) Using a concurrent signal assignment (not process), write the VHDL description, (including **entity** and **architecture**) for the circuit diagram shown in Figure 1.

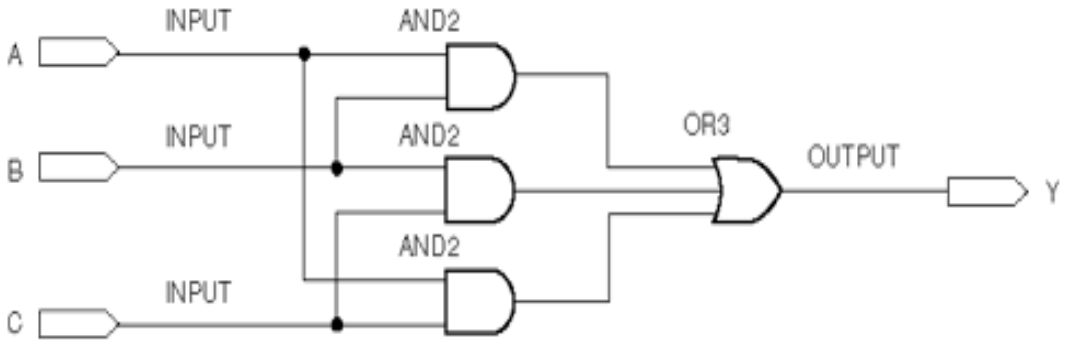


Figure 1

Answer:

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

entity voter is
  port(
    A : in  std_logic;
    B : in  std_logic;
    C : in  std_logic;
    Y : out std_logic);
end voter;

architecture behav of voter is
begin
  Y <= (A and B) or (B and C) or (A and C);
end behav;
```

c) Write the truth table for the digital circuit in Figure 1. Can you explain the functionality of this circuit and mention a possible application of it?

Answer:

A	B	C	Y
0	0	0	**0**
0	0	1	**0**
0	1	0	**0**
0	1	1	**1**
1	0	0	**0**
1	0	1	**1**
1	1	0	**1**
1	1	1	**1**
0	0	0	**0**
0	0	1	**0**
0	1	0	**0**
0	1	1	**1**
1	0	0	**0**
1	0	1	**1**
1	1	0	**1**
1	1	1	**1**

This circuit is a majority voter. The output Y of this circuit is equal to which ever value appears most frequently in at the inputs. This circuit can e.g. be used to implement fault tolerant designs. By making three identical instances of the same logic functionality, the output value of this logic functionality can be protected by adding a majority voter to the output of the three instances. This is also referred to as

Triple Modular Redundancy (TMR).

d) What is the purpose of the sensitivity list when describing a VHDL process? Explain why the sensitivity list should be used differently for a combinational process compared to a synchronous process.

Answer:
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.
In a combinational process the output should be directly dependent on the values on the input, and thus all signals that are read by the process (input signals of the combinational logic) must be included in the sensitivity list.
In a synchronous process, the inputs signals should only be evaluated when there is an event on the clock signal of that process. That is, the process is synchronous to the clock. Only the clock signal must be included in the sensitivity list of a synchronous process. The exception form this rule is if the synchronous process has an asynchronous reset. Then the sensitivity list must include both the clock signal and the asynchronous reset signal.

e) What is the difference between a variable and a signal in VHDL?

Answer:
When a signals or variable is assigned a value in VHDL, the value of the variable will be updated immediately while the value of the signal only will be updated at the end of the process when all statements have been evaluated.

f) Write a VHDL process statement that describes a 4-bit counter with a synchronous active high reset signal. Use a variable of type unsigned to implement the counter.

Answer:

```
process(clk)
  variable counter : unsigned(3 downto 0);
begin
  if rising_edge(clk) then
    if reset = '1' then
      counter := "0000";
    else
      counter := counter + 1;
    end if;
  end if;
end process;
```

Problem 2 (25%):

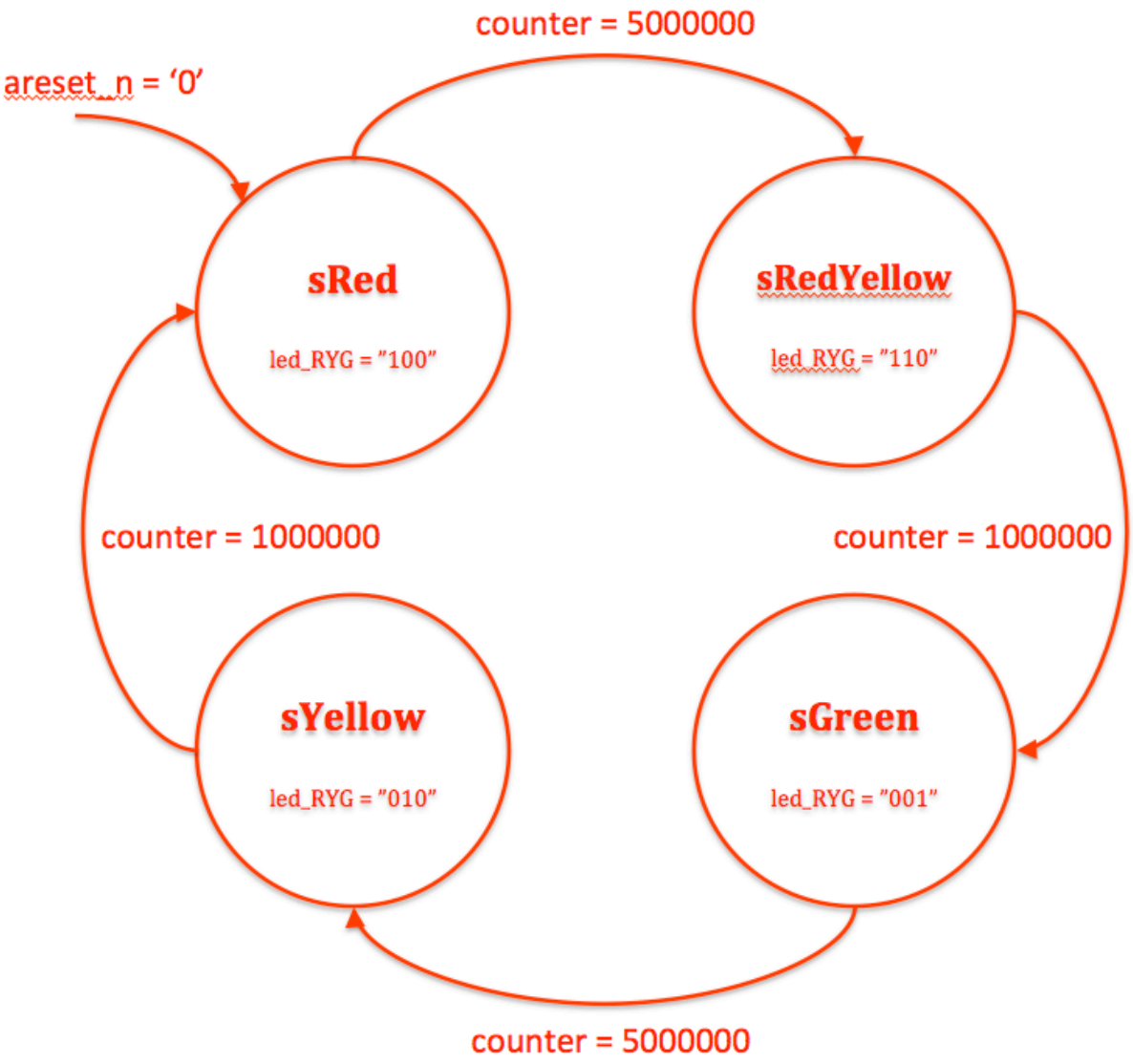
A basic traffic light controller is to be implemented in VHDL using a state machine architecture according to the following description:

- The state machine will have four states called: **sRed, sRedYellow, sGreen, sYellow**
- A 3-bit wide output port **led_RYG** will be used to switch ON ('1') or OFF ('0') a red, yellow and green light emitting diode (LED). The red LED will be connected to the most significant bit, the yellow LED to the central bit position, and the green LED to the least significant bit. The LEDs will be controlled according to the *current state* of the state machine:
  - **sRed**: the red LED is ON while the yellow and green LED is OFF
  - **sRedYellow**: the red and yellow LED is ON while the green LED is OFF
  - **sGreen**: the green LED is ON while the red and yellow LED is OFF
  - **sYellow**: the yellow LED is ON while the red and green LED is off
- The state transitions of the state machine are as follows:
  - sRed → sRedYellow
  - sRedYellow → sGreen
  - SGreen → sYellow
  - sYellow → sRed
- The state machine will be reset by an asynchronous active low reset signal. When the state machine is reset it will return to the **sRed** state.
- The state machine will be running on a system clock of 1 MHz. For practical reasons, the active period of each state must therefore be longer than the period of the system clock period. The state machine shall remain for 5 seconds in the **sRed** and **sGreen** states, and for 1 second in the **sRedYellow** and **sGreen** states. Use a counter to achieve correct timing of the state machine. The counter must be implemented as a *variable* of type integer within the state machine process.
- The entity of the VHDL code is shown in Figure 2.

```
entity traffic_light is
  port(
    clk      : in  std_logic;
    areset_n : in  std_logic;
    led_RYG  : out std_logic_vector(2 downto 0)
    );
end traffic_light;
```

Figure 2

a) **(10%)** Draw the state diagram for the traffic light controller described above.



b) **(10%)** Write the VHDL description of the traffic light controller.
Answer:

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

entity traffic_light is
  port(
    clk      : in  std_logic;
    areset_n : in  std_logic;
    led_RYG  : out std_logic_vector(2 downto 0)
    );
end traffic_light;

architecture behav of traffic_light is

  constant OneSec  : integer := 1000000;
  constant FiveSec : integer := 5000000;

  type statetype is (sGreen, sYellow, sRed, sRedYellow);
  signal state : statetype;

begin

  process(areset_n, clk)
    variable counter : integer range 0 to 5000000;
  begin
    if areset_n = '0' then
      state   <= sRed;
      led_RYG <= "100";
      counter := 0;
    elsif rising_edge(clk) then
      case state is

        when sRed =>
          led_RYG <= "100";
          counter := counter + 1;
          if counter = FiveSec then
            state   <= SRedYellow;
            counter := 0;
          end if;

        when sRedYellow =>
          led_RYG <= "110";
          counter := counter + 1;
          if counter = OneSec then
            state   <= SGreen;
            counter := 0;
          end if;

        when sYellow =>
          led_RYG <= "010";
          counter := counter + 1;
          if counter = OneSec then
            state   <= SRed;
            counter := 0;
          end if;

        when sGreen =>
          led_RYG <= "001";
          counter := counter + 1;
          if counter = FiveSec then
            state   <= SYellow;
            counter := 0;
          end if;

      end case;
    end if;
  end process;

end behav;
```

c) **(5%)** Is the traffic light controller a Mealy type of Moore type state machine? Explain your answer.

Answer:
The state machine of the traffic light controller is a Moore type state machine. This is because the value of the outputs are only dependent on the current state of the state machine.

Problem 3 (20%):

a) What is the difference between a softcore and a hardcore processor?

Answer:
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.

b) Is the NIOS II processor from Altera a softcore or a hardcore processor?
Answer:
It is a softcore processor.

c) What is meant by an embedded system? Name 3 examples of embedded systems.

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

Some examples:

- Digital watch
- Mobile phone
- Washing machine
- Mp3 player
- Climate control system in a care
- Anti-lock breaking system
- Motor control system

d) What is the purpose of the Hardware Abstraction Layer (HAL)?

Answer:
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. It provides a

e) The NIOS II processor is a general-purpose RISC processor core where the instructions can be pipelined. What does it mean that instructions are pipelined and why is this an advantage?

Answer:
A pipeline is a technique used to increase the instruction throughput (the number of instructions that can be executed in a unit of time). Pipelining does not reduce the time to complete an instruction, but increases instruction throughput by performing multiple operations in parallel.

f) Consider that you have a variable **int c=25**. If you want to write the value of this variable to the console (e.i. standard output) in hexadecimal format, how can you do this? What is the hexadecimal value of **c** in this case?

Answer:
printf("c= %x", c);
The hexadecimal value of c is 19.

Problem 4 (30%):

a) What makes a real-time embedded system different from a "normal" embedded system?

Answer:
For a real-time embedded system the correctness of the system depends not only on the logical result of a computation, but also on the time at which the results are generated.

b) What is 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 may lead to 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.

c) What is the main task of the scheduler in a real-time system?

Answer:
The main task of the scheduler is to determine which tasks run next.

d) Explain the term preemption in relation to a real-time kernel?

Answer:
Preemption is the act of temporarily interrupting a task being carried out by the real-time kernel. If a task of higher priority is made ready to run, the current task running will be suspended, and the higher priority task is immediately given control of the CPU. When the higher priority task is completed, the execution of the lower priority task is resumed. This is also known as a context switch.

e) Briefly explain the concepts of priority based scheduling and round robin scheduling.

Answer:
Priority based scheduling is when each task is assigned a priority based on its importance, and the control of the CPU is given to the highest priority task ready to run.

Round robin scheduling is when time slices are assigned to each task in equal portions and in a circular order, handling all tasks without priority. A task is interrupted if it is not completed by the given available time, and resumed next time a slot is assigned to that task.

f) What is a semaphore?

Answer:
A semaphore is a variable used for controlling access, by multiple processes, to a common shared resource.

g) Describe two possible applications of semaphores?

Answer:
A semaphore can be used to:

- Control the access to a common shared resource like e.g. a printer. In the case when two tasks/processes both need to access the printer, each task needs to acquired the semaphore before it can access the printer, and then release the semaphore when the access is completed. In this way, only one task can access the printer at the same time.
- Synchronize the execution of tasks, and make the task execute in a sequential fashion. Assuming you want to synchronize two tasks t1 and t2 such that t2 only will be executed after t1. A semaphore is then initially given the value 0 (which means that the semaphore is taken) and t2 is set to wait for this semaphore. When t1 releases the semaphore (sets the semaphore value to 1), the semaphore is available for t2 which then can be executed.

h) Figure 3 shows an example of two tasks that both print a message to the console (i.e. standard output). What is the purpose of the OSTimeDlyHMSM function?

Answer:
The OSTimeDlyHMSM function is used to suspend the execution of the task and thereby release the control of the CPU to another waiting task. This function can be

used to determine the execution frequency of the task by specifying how long the task should be suspended using the hour, minute, second, and millisecond argument of the function.

i) Assuming that task1 has a higher priority than task2, what would be the consequence if the OSTimeDlyHMSM function would be removed from task1?

Answer:
The consequence would be that task2 never will be executed because task1 will not be suspended, and since it has a higher priority than task two, task1 will always be in control of the CPU and therefore continuously running.

j) What is the potential problem with the code as it is implemented in Figure 3, and how can this problem be avoided? Add the appropriate modifications to the code.

```
void task1(void* pdata)
{
  while (1)
  {
    printf("Hello from task1\n");
    OSTimeDlyHMSM(0, 0, 0, 5);
  }
}

void task2(void* pdata)
{
  while (1)
  {
    printf("Hello from task2\n");
    OSTimeDlyHMSM(0, 0, 0, 5);
  }
}
```

Figure 3

Answer:
The problem with the codes as shown in Figure 3 is that two tasks are both accessing a commonly shared resource without this resource being protected by a mutual exclusion. In case the lower priority task is running and accessing the standard output at the same time as the higher priority task is made ready to run, the higher priority task will preempt the lower priority task in the middle of its access to the standard output. The result is that the text or value written to the standard output may be corrupted. This problem can be resolved by using a semaphore to protect the access to the standard output. See figure on next page.

```c
void task1(void* pdata)
{
  while (1)
  {
    OSSemPend(semaphore_name,0,&error_code);
    printf("Hello from task1\n");
    OSSemPost(semaphore_name);
    OSTimeDlyHMSM(0, 0, 0, 5);
  }
}

void task2(void* pdata)
{
  while (1)
  {
    OSSemPend(semaphore_name,0,&error_code);
    printf("Hello from task2\n");
    OSTimeDlyHMSM(0, 0, 0, 5);
    OSSemPost(semaphore_name);
  }
}
```