Exam - DDV3101 Comp.architecture and VHDL programming

Oppgave 1)
Dennard scaling states that as transistors are reduced in size, their power density stays constant. Meaning, power use stays in proportion with area, as both voltage and current scale (downward) with length. In layman's terms, Dennard scaling is what enabled chipmakers to increase the clock speed of their processors without increasing power draw as transistors got smaller.

With time however, and the breakdown of Dennard scaling and resulting inability to increase clock frequencies significantly has caused most CPU manufacturers to focus on multicore processors rather than a faster uniprocessor as an alternative way to improve performance. Before, the focus was on building a single core that was more complex and more powerful. Since then, the complexity of a processor is much more diverse and multi processors architectures and other forms of parallelism have become dominant.

Dennard scaling: as transistors are reduced in size, their power density stays constant. With the breakdown of this companies focus now on multicore processors for better performance, with the focus of parallelism

b)

With multiple processors per chip we get that they are more difficult to manage thermally than lower-density single-core designs. Furthermore, two processing cores sharing the same system bus and memory bandwidth limits the real-world performance advantage.

The number of transistors on a chip no longer doubles every few years, as Murphy predicted, neither the power density continues to be constant for a given area of silicon, independently of the number of transistors. Performance improved greatly over the years, but technology-driven improvements are now far more limited. In this way, we can realize that there is indeed a limit to performance improvements due to multiple processors per chip as the improvements are bounded by technology limitations.

Oppgave 2)
In today's world the term "architecture" refers to much more than instruction set design. Another important design factor that influences today's computer systems performance is for example, pipelining where the set of data processing elements are connected in series and the output of one element is the input of the next one.

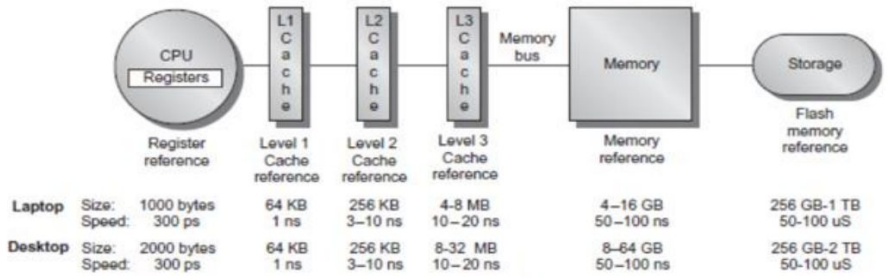
Oppgave 3)

a)

Memory hierarchy separates computer storage into a hierarchy based on response time.

b)

Below is an example of memory hierarchy for a laptop or a desktop computer where three cache levels are normally used. Some of the benefits we see is in terms of speed and size. As you go from registers inside the CPU down to the main storage medium we see the difference in speed and size. The speed increases the further we get from the cpu for example.



C) Why should multiple cache layers be used?

An issue is the fundamental tradeoff between cache latency and hit rate. Larger caches have better hit rates but longer latency. To address this tradeoff, many computers use multiple levels of cache, with small fast caches backed up by larger, slower caches.

D)

Cache lines = Memory size of cache / Size of 1 cache line
= 32KB / 4B
= 2^13 lines
b.p cache line = Log2(2^13)
= 13 bits
b.p word = Log2(4)
= 2 bits
b.p tag = 32 – 13 – 2
= 17 bits
tag memory = 17 * 2^13
= 139264 bits
32KB = 256K bits
Total memory = 256 000 + 139 264 **= 395264 bits**

Oppgave 4)

a)

Instruction pipelining is a technique for implementing instruction-level parallelism within a single processor. We do this in order to keep the processor busy at all times with some instruction by dividing incoming instructions into a series of sequential steps.

b) If the instruction pipeline has a depth of 4 stages, then it is correct that the execution time will be shortened four times.

c) There are cases when the next instruction cannot start in the following clock cycle, and in such cases we have a pipeline hazard. In this case during the execution of the two successive instructions we have a hazard because in $s1 the sub is performed and it must finish before it can be used again in add. This is why we get a hazard, because the add instruction must wait one clock cycle for the sub to finish.

d) In this case, we have one clock cycle, since s1 is used in both sub and add.

e) Not quite sure what is meant by this question

f) We prefer the fixed length because during dispatch of instructions it is more efficient

Oppgave 5)

a) A hazard i see from the figure is for example that t3 is used in add and then successively in sw. They both use t3, and so This will cause a hazard. This is the same for t5

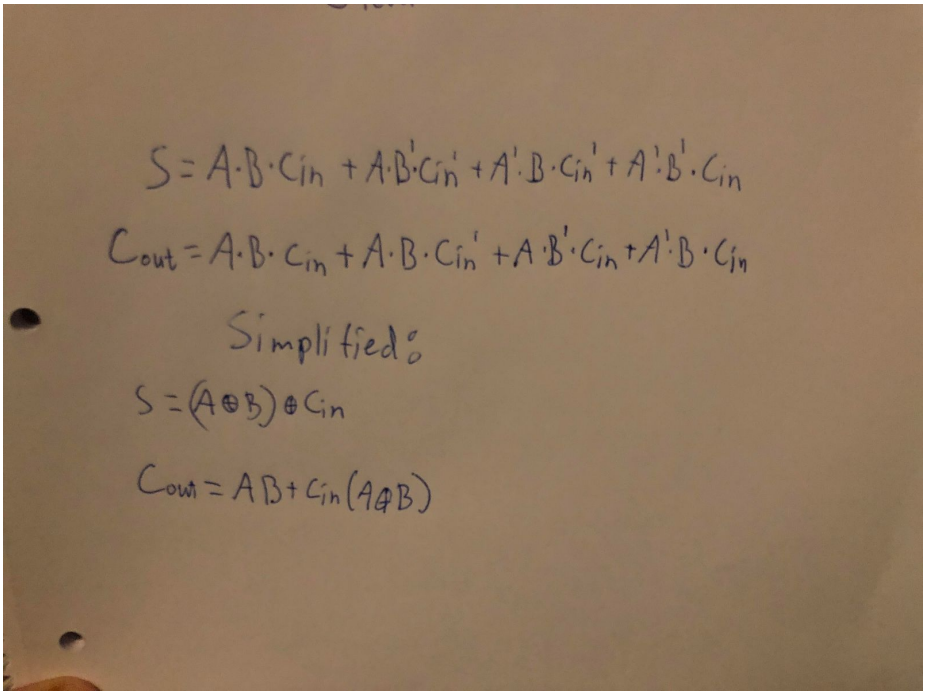
b) Since t3 and t5 cause hazards i separate them and rearrange in this order:

```
lw $t2, 4($t0)
add $t3, $t1, $t2
lw $t4, 8($t0)
sw $t3, 12($t0)
sub $t5, $t1, $t4
sw $t5, 16($t0)
```

Oppgave 6)
a)

Cin	A	B	S	Cout
0	0	0	0	0
0	1	0	1	0
0	0	1	1	0
0	1	1	0	1
1	0	0	1	0
1	1	0	0	1
1	0	1	0	1
1	1	1	1	1

b)



c)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity one_bit_full_adder is
 Port ( a : in STD_LOGIC;
 b : in STD_LOGIC;
 c_in : in STD_LOGIC;
 Sum_out : out STD_LOGIC;
 c_out : out STD_LOGIC);
end one_bit_full_adder;

architecture Behavioral of one_bit_full_adder is

begin

 Sum_out <= a XOR b XOR c_in ;
 c_out <= (a AND b) OR (c_in AND a) OR (c_in AND b) ;

end Behavioral;
```

d)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
ENTITY one_bit_full_adder_TB IS
END one_bit_full_adder_TB;

ARCHITECTURE behavior OF one_bit_full_adder_TB IS

-- Component Declaration for the Unit Under Test (UUT)

 COMPONENT one_bit_full_adder
 PORT(
 a : IN std_logic;
 b : IN std_logic;
 c_in : IN std_logic;
 Sum_out : OUT std_logic;
 c_out : OUT std_logic
 );
 END COMPONENT;

-- Inputs
 signal a : std_logic := '0';
 signal b : std_logic := '0';
 signal c_in : std_logic := '0';

-- Outputs
 signal Sum_out : std_logic;
 signal c_out : std_logic;

BEGIN

-- Instantiate the Unit Under Test (UUT)
 uut: one_bit_full_adder PORT MAP (
 a => a,
 b => b,
 c_in => c_in,
 Sum_out => Sum_out,
 c_out => c_out
 );

-- Stimulus process
 stim_proc: process
 begin
 -- hold reset state for 100 ns.
 wait for 100 ns;
-- insert stimulus here
 a <= '1';
 b <= '0';
 c_in <= '0';
```

```
 wait for 10 ns;

 a <= '0';
 b <= '1';
 c_in <= '0';
 wait for 10 ns;

 a <= '1';
 b <= '1';
 c_in <= '0';
 wait for 10 ns;

 a <= '0';
 b <= '0';
 c_in <= '1';
 wait for 10 ns;

 a <= '1';
 b <= '0';
 c_in <= '1';
 wait for 10 ns;

 a <= '0';
 b <= '1';
 c_in <= '1';
 wait for 10 ns;

 a <= '1';
 b <= '1';
 c_in <= '1';
 wait for 10 ns;

 end process;
END;
```

Oppgave 7)

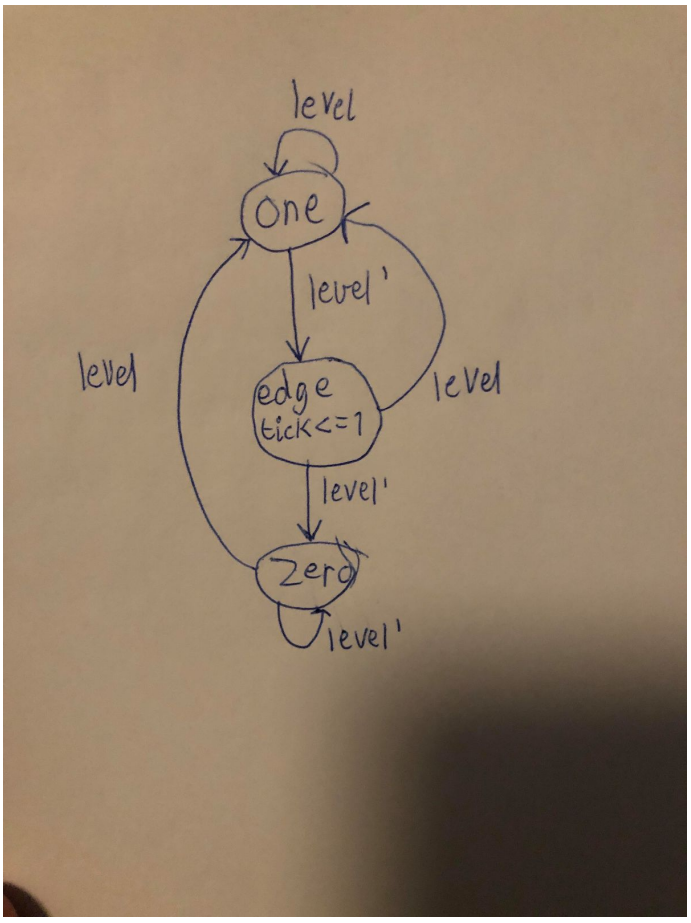
```
entity when else is
    Port ( x : in STD_LOGIC_VECTOR (2 downto 0);
           y : out STD_LOGIC_VECTOR (1 downto 0));
end when else;

architecture Behavioral of when_else is
begin y <= "11" when x(2) = "0" else
"01" when x(0) = "1" else
"00" when x(2) = "1" else
```

"10" when x(1) = "0" else
"10" when x = "110" else
"11";
end Behavioral;

Oppgave 8)

a) This is a Moore state machine because I see that the output values are determined solely by its current state.

b)



c) library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity fsm is
    Port (clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           level : in STD_LOGIC;
           tick : out STD_LOGIC);
end fsm;

```
architecture Behavioral of fsm is
type Moore_state is (zero, edge, one); -- 3 states are required for Moore
signal stateMoore_reg, stateMoore_next : Moore_state;

begin
   process(clk, reset)
   begin
      if (reset = '1') then -- go to state zero if reset
         stateMoore_reg <= zero;
      elsif (clk'event and clk = '1') then -- otherwise update the states
         stateMoore_reg <= stateMoore_next;
      end if;
   end process;

 -- Moore Design
   process(stateMoore_reg, level)
begin
      -- store current state as next
      stateMoore_next <= stateMoore_reg; -- required: when no case statement is
satisfied

      tick <= '0'; -- set tick to zero (so that 'tick = 1' is available for 1 cycle only)
      case stateMoore_reg is
         when zero => -- if state is zero,
            if level = '0' then  -- and level is 0
               stateMoore_next <= edge; -- then go to state edge.
            end if;
         when edge =>
            tick <= '1'; -- set the tick to 1.
            if level = '1' then -- if level is 1,
               stateMoore_next <= one; --go to state one,
            else
               stateMoore_next <= zero; -- else go to state zero.
            end if;
         when one =>
            if level = '0' then -- if level is 0,
               stateMoore_next <= zero; -- then go to state zero.
            end if;
      end case;
   end process;
end Behavioral
```

d)

mod m counter

```
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
```

```
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity mod_m_counter_en is
   generic(N : integer := 4;
           M : integer := 10);
   Port ( reset      : in STD_LOGIC;
          clk      : in STD_LOGIC;
          en       : in STD_LOGIC;
          q        : out STD_LOGIC_VECTOR (N-1 downto 0);
          max_tick : out STD_LOGIC);
end mod_m_counter_en;

architecture Behavioral of mod_m_counter_en is
------------internal memory--------
signal r_reg, r_next : unsigned(N-1 downto 0);    --unsgined not std_logic because
begin

   process(reset, clk)
      begin
----------------------sequential part----------
          if(reset = '1') then
             r_reg <= (others =>'0');
          elsif (rising_edge(clk)) then      --(clk'event and clk = '1'), event means if anything
changes in clock signal we set clk to 1, raising edge
              if(en = '1') then
             r_reg <= r_next;
            end if; --en
          end if; --rst
       end process;
--------------------next sate logic-----------
-------input: r_reg
-------output: r_next
     r_next <= (others => '0') when r_reg = M-1 else  --the counter value should be reset if it
breaks the maximum value m-1, else it goes up by one value
              r_reg +1;
--------------------output logic---------------
--------r_reg and external inout signal

   q <= std_logic_vector(r_reg);   --important here to have r_reg and not r_next in order to
have synchronous design
   max_tick <= '1' when r_reg = M-1 else
          '0';
end Behavioral;
```

hex2seg

```
entity hex_to_seg is
```

```
    Port ( hex : in STD_LOGIC_VECTOR (3 downto 0);
           seg : out STD_LOGIC_VECTOR (7 downto 0);
           an : out STD_LOGIC_VECTOR (3 downto 0));
end hex_to_seg;

architecture Behavioral of hex_to_seg is

begin
an <= "0111";   ---use all for seg, 0 means turn on for this circuit
seg(7) <= '1';  ---dp is turned off, because 1 means off for this circuit
    process(hex)
        begin
            case hex is
                when "0000" => --0
                seg(6 downto 0) <= "1000000";
                when "0001" =>   --1
                seg(6 downto 0) <= "1111001";
                when "0010" =>   --2
                seg(6 downto 0) <= "0100100";
                when "0011" =>   --3
                seg(6 downto 0) <= "0110000";
                when "0100" =>   --4
                seg(6 downto 0) <= "0011001";
                when "0101" =>   --5
                seg(6 downto 0) <= "0010010";
                when "0110" =>   --6
                seg(6 downto 0) <= "0000010";
                when "0111" =>   --7
                seg(6 downto 0) <= "1111000";
                when "1000" =>   --8
                seg(6 downto 0) <= "0000000";
                when "1001" =>   --9
                seg(6 downto 0) <= "0010000";
                when "1010" =>   --10 | A
                seg(6 downto 0) <= "0001000";
                when "1011" =>   --11 | B
                seg(6 downto 0) <= "0000011";
                when "1100" =>   --12 | C
                seg(6 downto 0) <= "1000110";
                when "1101" =>   --13 | D
                seg(6 downto 0) <= "0100001";
                when "1110" =>   --14 | E
                seg(6 downto 0) <= "0000110";
                when "1111" =>   --15 | F
                seg(6 downto 0) <= "0001110";
            end case;
        end process;
end Behavioral;
```

```
entity fsm is
    Port ( clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           level : in STD_LOGIC;
           tick : out STD_LOGIC);
end fsm;

architecture Behavioral of fsm is
type Moore_state is (zero, edge, one); -- 3 states are required for Moore
signal stateMoore_reg, stateMoore_next : Moore_state;

begin
    process(clk, reset)
    begin
        if (reset = '1') then -- go to state zero if reset
            stateMoore_reg <= zero;
        elsif (clk'event and clk = '1') then -- otherwise update the states
            stateMoore_reg <= stateMoore_next;
        end if;
    end process;

 -- Moore Design
    process(stateMoore_reg, level)
begin
        -- store current state as next
        stateMoore_next <= stateMoore_reg; -- required: when no case statement is
satisfied

        tick <= '0'; -- set tick to zero (so that 'tick = 1' is available for 1 cycle only)
        case stateMoore_reg is
            when zero => -- if state is zero,
                if level = '0' then  -- and level is 0
                    stateMoore_next <= edge; -- then go to state edge.
                end if;
            when edge =>
                tick <= '1'; -- set the tick to 1.
                if level = '1' then -- if level is 1,
                    stateMoore_next <= one; --go to state one,
                else
                    stateMoore_next <= zero; -- else go to state zero.
                end if;
            when one =>
                if level = '0' then -- if level is 0,
                    stateMoore_next <= zero; -- then go to state zero.
                end if;
        end case;
```

```
    end process;
end Behavioral
```

References:

https://en.wikipedia.org/wiki/Dennard_scaling
https://en.wikipedia.org/wiki/CPU_cache

https://gateoverflow.in/145078/how-many-total-bits-are-required-for-direct-mapped-cache-with
https://en.wikipedia.org/wiki/Mealy_machine