1. (10 points) Draw a transistor-level diagram (using n-FETs and p-FETs) of a CMOS NOR gate with 3 inputs, A, B and C.

   Suppose all three inputs are low initially and then one A goes high, causing the output of the NOR gate to go from high to low. Let $t_{HL}$ be the time for this high to low transition. Now, suppose that input A goes low again, causing the output to go from low to high. Let $t_{LH}$ be the time for this low to high transition. Which is larger, $t_{LH}$ or $t_{HL}$? Explain why.

How much larger? Assume that the on-resistance of an n-FET is the same as the on-resistance of a p-FET.
2. (10 points) Use Karnaugh maps to derive a minimal sum of products expression and a minimal product of sums expression for the function. \( F(A,B,C,D) = \Sigma m(0,4,5,12) \), \( d(A,B,C,D) = \Sigma m(3,7,8,11,13,14) \). Make full use of the don’t care conditions.

How many simple gates AND gates, OR gates and inverters are needed to implement circuits for the simpler of the two expressions you derived?

If CMOS gates are used in these circuits, how many transistors do they contain?
3. (15 points) Draw a logic diagram that directly corresponds to the VHDL module shown below. Every signal that appears in the VHDL should be labeled in your logic diagram. Your logic diagram may include gates, flip flops and multiplexors.

```vhdl
entity foo is port (  
    clk, enable: in std_logic;  
    A, B : in std_logic;  
    X: out std_logic);  
end foo;  
architecture a1 of foo is  
signal s0, s1: std_logic;  
begin  
    process(clk) begin  
        if rising_edge(clk) then  
            if enable = '0' then  
                s0 <= '0'; s1 <= '0';  
            else  
                if s0 = '1' and A = '1' then  
                    s1 <= s0;  
                elsif s0 > s1 then  
                    s0 <= A xor B;  
                end if;  
            end if;  
        end if;  
    end process;  
    X <= '1' when s1 > B else '0';  
end a1;
```
4. (20 points) The circuit shown below is a 3 digit BCD maximum circuit. The output \( \text{max}(2..0) \) is equal to the larger of \( A(2..0) \) and \( B(2..0) \).

Estimate the number of 4 input LUTs needed to implement one section of this circuit. Justify your estimate.

If the delay for a LUT is 1 ns, what is the worst-case delay for a 16 digit version of this circuit?

How would you change the circuit to work for excess-3 instead of standard BCD?

Complete the VHDL module on the following page to implement a 16 bit version of this circuit. Assume that the type \texttt{bcdWord} is declared as follows:

\begin{verbatim}
type bcdWord is array(15 downto 0) of std_logic_vector(3 downto 0);
\end{verbatim}
entity bcdMax is port (  
  A, B : in bcdWord;  
  max : out bcdWord;  
end bcdMax;

architecture a1 of bcdMax is

begin
  process (  
    eq(0), eq(1), eq(2), eq(3),  
    gt(0), gt(1), gt(2), gt(3)  
  ) begin

    end process;

end a1;
5. (10 points) For the circuit shown below, assume that the flip flop setup time is 2 ns, the hold time is 1 ns and the flip flop propagation delay is between 1 and 3 ns. Also, assume that the gate delay is between 0.3 and 1 ns (for all gates, including the mux) and that the clock skew is 1 ns.

Is this circuit subject to internal hold time violations? If so, show where you would add delay in order to eliminate them?

What is the smallest clock period for which the circuit is not subject to setup time violations? Be sure to take into account any modifications from the previous step.

During what period relative to the clock must input B be stable?
6. (14 points). Is the state table shown at right for a Mealy model sequential circuit, or a Moore model circuit?

<table>
<thead>
<tr>
<th>present state</th>
<th>input</th>
<th>output</th>
<th>next state</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_1S_0</td>
<td>AB</td>
<td>X</td>
<td>D_1D_0</td>
</tr>
<tr>
<td>00</td>
<td>0x</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>00</td>
<td>1x</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>1</td>
<td>01</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>1</td>
<td>01</td>
</tr>
<tr>
<td>01</td>
<td>1x</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>x0</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>10</td>
<td>x1</td>
<td>0</td>
<td>01</td>
</tr>
</tbody>
</table>

Write the next-state equations and the output equation for this sequential circuit, using the truth table. You may take advantage of the don’t care condition implied by the fact that there is no 11 state.

Draw a schematic of the circuit.
7. (20 points) The state diagram shown below is for a sequential circuit that counts the number periods where input $A$ is less than or equal to input $B$ for one or more time steps in a row ($leRun$), and the number of periods when $A > B$ for at least two time steps in a row ($gtRun$). $A$ and $B$ are four bits each and there is an eight bit output $X$ equal to $leRun - gtRun$. The circuit also has a reset signal which has been omitted from the diagram. When reset is high, the circuit should go to the le state and $leRun$ and $gtRun$ should both be set to 0.

Actually, the state diagram above is not quite correct. In particular, if $A \leq B$ right after reset, the value of $leRun$ ends up being one smaller than it should be. Show how to modify the state diagram above to correct this.

Complete the VHDL module outlined on the next page so that it implements the sequential circuit specified by the original version of the state diagram. Include code for the reset and output $X$. 
entity countRuns is Port (  
clk, reset: in std_logic;  
A, B : in std_logic_vector(3 downto 0);  
X : out std_logic_vector(7 downto 0));  
end countRuns;  

architecture al of ballgame is  

begin  
  process ( ) begin  

end process;  

end a1;
8. (12 points) The VHDL module defined below implements a circuit that counts the number of times that input A becomes larger than input B. So for example, with the sequence of input pairs \((A, B) = (7,5), (4,6), (8,6), (9,13), (15, 2)\) the final value of the output, \(nUpCrossings\) should be 2. Complete the schematic at the bottom of the page so it implements this circuit, using only simple gates.

```vhdl
entity upCross is Port (
    clk, reset: in std_logic;
    X, Y : in std_logic_vector(3 downto 0);
    nUpCrossings : out std_logic_vector(7 downto 0);
end upCross;
architecture a1 of upCross is
signal count: std_logic_vector(7 downto 0);
signal prevCompare: std_logic;
begin
    process (clk) begin
        if rising_edge(clk) then
            if reset = '1' then
                count <= (count'range =>'0');
                prevCompare <= '1';
            else
                if X > Y then
                    prevCompare <= '1';
                else
                    prevCompare <= '0';
                end if;
                if prevCompare = '0' and X > Y then
                    count <= count + '1';
                end if;
            end if;
        end if;
    end process;
    nUpCrossings <= count;
end a1;
```

---

**Schematic**

- \(X\)
- \(Y\)
- \(\text{compare}\)
- \(\text{prevCompare}\)
- \(\text{increment}\)
- \(\text{reset}\)
- \(\text{clk}\)
9. (12 points) The simulation output shows a program executing on the version of the processor from Design Problem 5. Recall that for this version of the processor, the inCtl register is accessed using address FFFC, the input register at FFFD, the LEDs at register FFFE and the output register at FFFF. Fill in the blanks below with the information that belongs in each of the labeled blank areas in the simulation output. Note that $32_{10}=20_{16}$.

A.  

B.  

C.  

D.  

E.  

F.  

\[
\begin{array}{c}
\text{/reset} & 0 \\
\text{/mem_en} & 0 \\
\text{/mem_rw} & 1 \\
\text{/abus} & \text{ZZZ} \\
\text{/dbus} & \text{ZZZ} \\
\text{/mclk} & 1 \\
\text{/pcu/state} & \text{brze} \\
\text{/pcu/tick} & 0 \\
\text{/pcu/pc} & 0010 \\
\text{/pcu/ireg} & 7016 \\
\text{/pcu/acc} & 0000 \\
\text{/pcu/alu} & 0000 \\
\text{/pcu/iar} & \text{FFFF} \\
\text{/pcu/preg} & 00 \\
\text{/swt} & FF \\
\text{/loadbtn} & 0 \\
\text{/loc/incfreg} & 0000 \\
\text{/loc/inreg} & \text{FFFF} \\
\text{/loc/outreg} & 0000 \\
\end{array}
\]

\[
\begin{array}{c}
\text{Now} \text{ps} & 51200 \text{ ns} & 51400 \text{ ns} & 51600 \text{ ns} \\
\end{array}
\]

0000 halt execution
01xx PREG := xx
0001 ACC := ~ACC
1xxx if sign bit of xxx is 0 then
\quad ACC := 0xxx else ACC := fxxx
2xxx ACC := M[Pxxx]
3xxx ACC := M[M[Pxxx]]
4xxx M[Pxxx] := ACC
5xxx M[M[Pxxx]] := ACC
6xxx PC := Pxxx
7xxx if ACC = 0 then PC := Pxxx
8xxx if ACC > 0 then PC := Pxxx
9xxx if ACC < 0 then PC := Pxxx
aXXX ACC := ACC + M[Pxxx]
dXXX ACC := ACC and M[Pxxx]
10. (12 points) The table shown below represents a direct-mapped instruction cache for our basic processor. Suppose the processor begins execution at location 0c15 and executes four instructions. Show the effect of this on the cache.

<table>
<thead>
<tr>
<th>tag</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0a4</td>
<td>401f</td>
</tr>
<tr>
<td>b23</td>
<td>1001</td>
</tr>
<tr>
<td>043</td>
<td>f312</td>
</tr>
<tr>
<td>0a4</td>
<td>c012</td>
</tr>
<tr>
<td>20c</td>
<td>1025</td>
</tr>
<tr>
<td>345</td>
<td>267b</td>
</tr>
<tr>
<td>0c1</td>
<td>9636</td>
</tr>
<tr>
<td>0c1</td>
<td>1004</td>
</tr>
<tr>
<td>0c1</td>
<td>0001</td>
</tr>
<tr>
<td>acb</td>
<td>c012</td>
</tr>
<tr>
<td>90c</td>
<td>23fb</td>
</tr>
<tr>
<td>03f</td>
<td>1003</td>
</tr>
<tr>
<td>0a3</td>
<td>2eec</td>
</tr>
<tr>
<td>083</td>
<td>dc16</td>
</tr>
<tr>
<td>abf</td>
<td>3201</td>
</tr>
<tr>
<td>0a4</td>
<td>a01d</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>address</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0636</td>
<td>683d</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>0c15</td>
<td>1fff</td>
</tr>
<tr>
<td>0c16</td>
<td>9636</td>
</tr>
</tbody>
</table>

How many values are retrieved from memory (not cache) during the execution of these four instructions? Account for both fetches and instruction execution.

What is the value in the ACC after these four instructions are executed?