1. (10 points) Draw a transistor-level diagram (using n-FETs and p-FETs) of a 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.

$t_{LH}$ is larger because for a low to high transition, charge must flow through three on pull-ups in series. For the high to low transition, charge must flow through just one pull-down. The lower resistance of the single pull-down allows for a faster transition.

How much larger? Assume that the on-resistance of an n-FET is the same as the on-resistance of a p-FET.

About three times larger.
2. (12 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.

\[
\begin{array}{c|cccc}
\text{CD} & 00 & 01 & 11 & 10 \\
\hline
00 & 1 & 0 & x & 0 \\
01 & 1 & 1 & x & 0 \\
11 & 1 & x & 0 & x \\
10 & x & 0 & x & 0 \\
\end{array}
\]

\[
\begin{array}{c|cccc}
\text{CD} & 00 & 01 & 11 & 10 \\
\hline
00 & 1 & 0 & x & 0 \\
01 & 1 & 1 & x & 0 \\
11 & 1 & x & 0 & x \\
10 & x & 0 & x & 0 \\
\end{array}
\]

The minimal sum of products expression is \( C'D' + BC' \).

The minimal product of sums expression is \( (C + B'D')' = C'(B + D') \).

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

One AND gate, one OR gate and two inverters.

If CMOS gates are used to implement these gates, how many transistors do they contain?

16
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.

```
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. (15 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 stage of this circuit.

*The compare block requires 3 LUTs for the equality comparison and 3 more for the greater-than comparison. The mux takes 4 LUTs (since there are 4 bits) and the outputs \( \text{eq}(1) \), \( \text{gt}(1) \) each require one LUT. So, the total is 12 LUTs per stage.

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

*The worst-case delay passes through the comparator in the stage for digit 15 (2 ns), then the ripple logic of all 16 stages (16 ns) and finally the output mux in the stage for digit 0 (1 ns). This gives a total of 19 ns.*

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

*No change is needed.*

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

```vhdl
type bcdWord is array(15 downto 0) of std_logic_vector(3 downto 0);
```
entity bcdMax is port (  
  A, B : in bcdWord;  
  max : out bcdWord;  
end bcdMax;
architecture a1 of bcdMax is  
signal eq, gt: std_logic_vector(16 downto 0);  
begin  
  process (A, B, eq, gt) begin  
  eq(16) <= '1';  
gt(16) <= '0';  
  for i in 15 downto 0 loop  
    if A(i) = B(i) and eq(i+1) = '1' then  
      eq(i) <= '1';  
    else  
      eq(i) <= '0';  
    end if;  
    if gt(i+1) = '1' or (A(i) > B(i) and eq(i+1) = '1') then  
      gt(i) <= '1';  
    else  
      gt(i) <= '0';  
    end if;  
    if gt(i) = '1' then  
      max(i) <= A(i);  
    else  
      max(i) <= B(i);  
    end if;  
  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?

There are no hold time violations. The s0-to-s0 path has a minimum combinational circuit delay is .6 ns and since 1+.6 > 1, there is no violation. The same holds for the s1-to-s1 path. For the two “cross paths”, the minimum combinational circuit delay is 1.2 ns and since 1+1.2>2 there is no violation.

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.

The maximum combinational circuit delays for the “cross paths” are 4 ns. This gives a minimum clock period of 3+4+2+1=10 ns.

During what period relative to the clock must input B be stable?

Input B has a short path to a flip flop containing three gates and a long path containing four gates. Consequently, the input must be stable starting at (clock edge)-2-4×1=-6 to (clock edge)+1-3×0.3=+0.1. The the stable period is [-6,+0.1].
6. (14 points). Is the state table shown at right for a Mealy model sequential circuit, or a Moore model circuit?

**Moore**

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.

\[ X = S_0 \]
\[ D_1 = S'_1S'_0 + S_0A \]
\[ D_0 = S_0A' + S_1B \]

Draw a schematic of the 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>
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 a1 of ballgame is  
  type state_type is (le, gt, gt2);  
  signal state: state_type;  
  signal leRun, gtRun: std_logic_vector(7 downto 0);  
begin  
  process (clk) begin  
    if rising_edge(clk) then  
      if reset = '1' then  
        state <= le; leRun <= x"00"; gtRun <= x"00";  
      else  
        case state is  
          when le =>  
            if A > B then  
              state <= gt;  
            endif;  
          when gt =>  
            if A > B then  
              state <= gt2; gtRun <= gtRun + 1;  
            else  
              state <= le; leRun <= leRun + 1;  
            endif;  
          when others => -- gt2 case  
            if A <= B then  
              state <= le; leRun <= leRun + 1;  
            endif;  
        end case;  
      end if;  
    end if;  
  end process;  
  X <= leRun - gtRun;  
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;
```
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. 701C     B. 101E     C. 1018
D. 701C     E. 5021     F. FFFF

```
0000  halt execution
01xx  PREG := xx
0001  ACC := -ACC
1xxx  if sign bit of xxx is 0 then
      ACC := 0xxx else ACC := fxxx
2xxx  ACC := M[Pxxx]
3xxx  ACC := M[M[Pxxx]]
4xxx  M[Pxxx] := ACC
```

```
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>a4</td>
<td>401f</td>
</tr>
<tr>
<td>b23</td>
<td>1001</td>
</tr>
<tr>
<td>043</td>
<td>f312</td>
</tr>
<tr>
<td>a4</td>
<td>c012</td>
</tr>
<tr>
<td>20c</td>
<td>1025</td>
</tr>
<tr>
<td>445 0c1</td>
<td>267b 1fff</td>
</tr>
<tr>
<td>0c1 063</td>
<td>6836 683d</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>a4</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.

3 in total. 2 instructions are fetched and one location is accessed during execution of the fourth instruction (an and instruction).

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

9636