1. (10 points) How many simple gates of each type (and, or, inverter) are needed to directly implement the expression $A(B + C'D) + (A + CD)'$.

3 ANDs, 3 ORs and 2 inverters

Find the simplest equivalent sum of products expression using Boolean algebra.

$$A(B + C'D) + (A + CD)' = AB + AC'D + A'C' + A'D'$$

$$= AB + AC'D + A'C'D + A'C'D' + A'D'$$

$$= AB + (A + A')C'D + (A'C'D' + A'D')$$

$$= AB + C'D + A'D'$$

How many simple gates of each type are needed to directly implement the simplified expression?

3 ANDs, 2 ORs and 3 inverters

How many transistors are needed to implement these gates, using CMOS?

$3 \times 6 + 2 \times 6 + 3 \times 2 = 36$

Show how to implement this expression using simple NANDs and NORs in addition to the other simple gate types in a way that requires no more than 30 transistors.
2. (10 points) Find a minimal sum of products expression for $A (B + C + D') + BC + A'C$ using a Karnaugh map.

What are the maxterms for the expression $A (B + C + D') + BC + A'C$? Write the maxterms out, don't just give their numerical designations.

$(A + B + C + D), (A + B + C + D'), (A' + B + C + D')$

Find a minimal product of sums expression for $A (B + C + D') + BC + A'C$ using a Karnaugh map.

From the Karnaugh map above, $F' = A'B'C' + B'C'D$, so $F = (A + B + C)(B + C + D')$.
3. (5 points) The circuit below is a four bit 2s-complement circuit. That is, the output \( Y=y_3y_2y_1y_0 \) is the 2s-complement of the input \( X=x_3x_2x_1x_0 \). What is the worst-case propagation delay for a 64 bit version of this circuit, if every simple gate has a delay of 1 ns?

The worst-case propagation delay for the 64 bit version is 64 ns.

The block diagram below is for a lookahead version of a 2s-complement circuit. Fill in the sub-circuit that goes in the leftmost block.

Consider a 64 bit version of this circuit. Assuming that every gate with more than two inputs is replaced with the fastest equivalent circuit using simple gates, what is the worst-case propagation delay for the 64 bit version?

7 ns.
4. (10 points) Draw a schematic diagram for each of the VHDL modules specified below, using simple gates.

entity foo is
  port (
    b, c, d: in std_logic;
    a: out std_logic
  );
end foo;

begin architecture bar of foo
  process(b,c,d) begin
    a <= b or c;
    if b /= d then
      a <= d;
    end if;
  end process;
end bar;

entity oof is
  port (
    b, c, d: in std_logic;
    a: out std_logic
  );
end oof;

begin architecture rab of oof
  process(b,c,d) begin
    if b /= d then
      a <= d;
    end if;
    a <= b or c;
  end process;
end rab;

Both of the circuits at left implement foo. The first is a direct implementation of the logic, the second is obtained by logic simplification. The circuit at right implements oof.
5. (10 points) Draw a schematic diagram for a positive edge-triggered D flip flop using nand gates and inverters. Label all inputs and outputs.

What is the definition of the setup time of a positive edge-triggered D flip flop? Be precise.

The setup time is the length of the time period preceding the rising clock edge during which the D input of the flip flop must be stable.

What is the definition of the hold time?

The hold time is the length of the time period following the rising clock edge during which the D input of the flip flop must be stable.
6. (15 points) Is the sequential circuit shown below, a Mealy model circuit or a Moore model circuit? Circle one.

Write down the next state and output equations

\[ D_0 = AS_0 + S'_1 \quad D_1 = (A + S'_0)S_1 \quad X = A' S_1 S_0' \]

Make a state transition table.

<table>
<thead>
<tr>
<th>(S_0S_1A)</th>
<th>(X)</th>
<th>(D_0D_1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0 10</td>
</tr>
<tr>
<td>00</td>
<td>1</td>
<td>0 10</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>0 01</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>0 01</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>1 10</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0 10</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0 00</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0 11</td>
</tr>
</tbody>
</table>
7. (15 points) Give a state transition diagram for the sequential circuit that is implemented by the VHDL module shown below. Be sure to use one of the two conventional formats for the state transition diagram. No hybrid formats, please. Include self-loops in your diagram and make it clear how the input and output signals correspond to the labels on the arrows.

```vhdl
text
```

```
entity class is
port (  
  clk: in STD_LOGIC;
  quiz: in STD_LOGIC;
  drone: in STD_LOGIC_VECTOR(1 downto 0);
  vim: out STD_LOGIC_VECTOR(1 downto 0)
);
end class;

architecture bored of class is
  type state_type is (snooze, yawn, panic);
  signal state: state_type;
begin
  process(clk) begin
    if clk'event and clk = '1' then
      if quiz = '1' then
        state <= panic;
      elsif state = yawn and drone = x"3" then
        state <= snooze;
      elsif state = snooze and drone <= x"1" then
        state <= yawn;
      elsif state = panic then
        state <= yawn;
      end if;
    end if;
  end process;
  vim <= x"0" when state = snooze else
       x"1" when state = yawn and drone > x"1" else
       x"2" when state = yawn and drone <= x"1" else
       x"3";
end class;
```
8. (15 points) The VHDL module shown below implements a scaled down version of the dual stack circuit from problem set 7. In the simulation output on the next page, fill in the missing waveforms. For the A and B outputs, write in the numerical values for each of the blanks.

```
entity twostax is
  Port (clk, reset : in std_logic;
        pushA, popA, pushB, popB : in std_logic;
        emptyA, fullA, emptyB, fullB : out std_logic;
        Din : in std_logic_vector(2 downto 0);
        A, B : out std_logic_vector(2 downto 0));
end twostax;

architecture arch1 of twostax is
  type mem4x4 is array(0 to 3) of std_logic_vector(2 downto 0);
  signal stackSpace: mem4x4;
  signal topA, topB: std_logic_vector(2 downto 0);
begin
  process(clk) begin
    if clk'event and clk = '1' then
      if reset = '1' then
        topA <= "111"; topB <= "100";
      elsif pushA = '1' then
        if topA /= "011" then
          stackSpace(conv_integer(unsigned(topA + "001"))) <= Din;
          if topB = topA + "001" then
            topB <= topB + "001";
          end if;
          topA <= topA + "001";
        end if;
      elsif popA = '1' then
        if topA /= "111" then topA <= topA - "001"; end if;
      elsif pushB = '1' then
        if topB /= "100" then
          stackSpace(conv_integer(unsigned(stackSpace(topB - "001")))) <= Din;
          topB <= topB - "001";
        end if;
      elsif popB = '1' then
        if topB /= "100" then topB <= topB + "001"; end if;
      end if;
      end if;
    end if;
  end process;
  emptyA <= '1' when topA = "111" else '0';
  fullA <= '1' when topA = "011" else '0';
  emptyB <= '1' when topB = "100" else '0';
  fullB <= '1' when topB = topA + "001" else '0';
  A <= "000" when topA = "111" else
       stackSpace(conv_integer(unsigned(topA)));
  B <= "000" when topB = "100" else
      stackSpace(conv_integer(unsigned(topB)));
end arch1;
```
The complete simulation output appears below.
9. (15 points) The simulation output is from the basic processor running a program. For each of the blanked out spots on the simulation, write the numerical value that should appear in that spot in the blanks given below.

- A. 1FF0
- B. 7002
- C. 1FF0
- D. 006E
- E. 5023
- F. 002F
- G. 002F

A. 1FF0
B. 7002
C. 1FF0
D. 006E
E. 5023
F. 002F
G. 002F

0000 halt execution
0001 negate the value in the ACC
1xxx change the value of the ACC to xxx
2xxx load the contents of memory location xxx into the ACC
3xxx load the ACC from the memory location whose address is stored in memory location xxx
4xxx store the value in the ACC in memory location xxx
5xxx store the value in the ACC in the memory location whose address is stored in memory location xxx
6xxx change the value of the PC to xxx
7xxx change the value of the PC to xxx if ACC = 0
8xxx change the value of the PC to xxx if ACC > 0
9xxx change the value of the PC to xxx if ACC < 0
axxx add the value in memory location xxx to the ACC
10. (10 points) The table shown below represents a 2-way set associative cache (combined instruction and data cache). The ‘<’ and ‘>’ symbols in the center point toward the least recently used of the two entries in a given row. So for example, the right entry in row 4 has been used more recently than the left entry.

<table>
<thead>
<tr>
<th>tag</th>
<th>instr.</th>
<th>&gt;</th>
<th>tag</th>
<th>instr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0a4</td>
<td>401f</td>
<td>0</td>
<td>0a5</td>
</tr>
<tr>
<td>1</td>
<td>20c</td>
<td>1001</td>
<td>1</td>
<td>0a5</td>
</tr>
<tr>
<td>2</td>
<td>0a4</td>
<td>a01d</td>
<td>2</td>
<td>0a5</td>
</tr>
<tr>
<td>3</td>
<td>0a4</td>
<td>c012</td>
<td>3</td>
<td>20b</td>
</tr>
<tr>
<td>4</td>
<td>20c</td>
<td>1025</td>
<td>4</td>
<td>20b</td>
</tr>
<tr>
<td>5</td>
<td>20c</td>
<td>0001</td>
<td>5</td>
<td>102</td>
</tr>
<tr>
<td>6</td>
<td>20c</td>
<td>a01d</td>
<td>6</td>
<td>20b</td>
</tr>
<tr>
<td>7</td>
<td>20c</td>
<td>701f</td>
<td>7</td>
<td>0a5</td>
</tr>
<tr>
<td>8</td>
<td>0a4</td>
<td>a01d</td>
<td>8</td>
<td>0a5</td>
</tr>
<tr>
<td>9</td>
<td>0a4</td>
<td>c012</td>
<td>9</td>
<td>20b</td>
</tr>
<tr>
<td>a</td>
<td>07b</td>
<td>1025</td>
<td>a</td>
<td>20b</td>
</tr>
<tr>
<td>b</td>
<td>20c</td>
<td>0001</td>
<td>b</td>
<td>20b</td>
</tr>
<tr>
<td>c</td>
<td>0a3</td>
<td>23ff</td>
<td>c</td>
<td>20b</td>
</tr>
<tr>
<td>d</td>
<td>0a4</td>
<td>401f</td>
<td>d</td>
<td>0a5</td>
</tr>
<tr>
<td>e</td>
<td>20c</td>
<td>1001</td>
<td>e</td>
<td>0a3</td>
</tr>
<tr>
<td>f</td>
<td>0a4</td>
<td>a01d</td>
<td>f</td>
<td>0a5</td>
</tr>
</tbody>
</table>

Show how the state of the cache changes, when the processor fetches and executes the instruction shown below, assuming that a new value replaces the least recently used entry. Also, assume that cache entries are selected using the low order 4 bits of the memory address and that the tag consists of the high order 12 bits. Be sure to update the ‘<’ and ‘>’ symbols where appropriate.

```
address   data
0a57      37ba
```

What is the value in the accumulator, after the instruction completes?

602c
11. (10 points) What expressions are implemented by the PLA shown below?

\[ x = b'c'd' + c'd + ab \]
\[ y = bc' + c'd \]
\[ z = b'c'd' + ad \]

If we were to implement these expressions in exactly the same way, using a PAL, what is the minimum number of product terms that the PAL must support? (Equivalently, how many AND gates must it have?)

Because \( x \) has three product terms, any PAL that implements these expressions must have nine product terms.

Could you “fit” these equations in a PAL with fewer product terms if you simplified the expressions? Explain why or why not.

The Karnaugh map for \( x \) below, makes it’s clear that there is no equivalent expression with fewer than three terms.