1. (10 points) Draw a state diagram for the sequential circuit shown below.

\[
\begin{array}{c|c|c|c|c|}
AB & EW & D_A & D_B & YZ \\
\hline
00 & 0x & 00 & 10 & \\
01 & 0x & 00 & 10 & \\
10 & 0x & 00 & 01 & \\
11 & 0x & 00 & 11 & \\
00 & 10 & 01 & 10 & \\
00 & 11 & 11 & 10 & \\
01 & 10 & 11 & 10 & \\
01 & 11 & 01 & 10 & \\
10 & 10 & 00 & 01 & \\
10 & 11 & 10 & 01 & \\
11 & 10 & 11 & 11 & \\
11 & 11 & 01 & 11 & \\
\end{array}
\]
2. (10 points) Answer the following questions, in connection with the circuit in problem 1. Assume that the flip flop setup time is 2 ns, hold time is 1 ns, maximum clock skew is 1 ns, flip flop propagation delay is between 1 and 3 ns and gate delay is between .4 and 1.5 ns.

(a) Is this circuit subject to hold time violations? If so, explain how to eliminate them.

Yes it is, since the flip flop propagation delay, plus the circuit delay from the output of flip flop B to the input of either flip flop can be just 1.8 ns, while the skew and hold time total 2 ns. The violation can be eliminated by adding a pair of inverters on the feedback path from B.

(b) What is the shortest clock period for which the circuit is not subject to setup time violations? Take into account the modifications made in part (a) to avoid hold time violations (if any).

3 + 4*1.5 + 2 + 1 = 12 ns.

(c) If the clock goes high at time 0, during what time period must input \( W \) be stable to guarantee that there are no setup or hold time violations? What about input \( E \)?

\( W \) must be stable from \(-5\) ns to \(+.2\) ns. \( E \) must be stable from \(-3.5\) ns to \(+.6\) ns.

(d) If the clock goes high at time 0, during what time period can output \( Y \) be changing? Output \( Z \)?

\( Y \) can be changing from \(+1.4\) ns to \(+6\) ns. \( Z \) can be changing from \(+1\) ns to \(+3\) ns.
3. (10 points) Draw a schematic for a sequential circuit corresponding to the state diagram shown below.

![State Diagram]

<table>
<thead>
<tr>
<th>$AB$</th>
<th>$X$</th>
<th>$D_A D_B$</th>
<th>$YZ$</th>
<th>$D_A = A' + X'$</th>
<th>$D_B = X$</th>
<th>$Y = A' + B'$</th>
<th>$Z = A'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>0</td>
<td>10</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>11</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>01</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>10</td>
<td>00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>01</td>
<td>00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Schematic Diagram]
4. (10 points) Give a state table that corresponds to the VHDL code shown below. For the state assignment, let add=00, deuce=01 and game=10.

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity tennis is
  port (
    serve, volley, clk: in STD_LOGIC;
    ace, fault: out STD_LOGIC
  );
end tennis;

architecture rally of tennis is
  type state_type is (add, deuce, game);
  signal state: state_type;
  begin
    process begin
      wait until clk = '1';
      if state = deuce and serve = '1' then
        state <= add;
      elsif state = deuce and serve /= volley then
        state <= game;
      elsif state = add and volley = '0' then
        state <= deuce;
      elsif state = add and volley = '1' and serve = '0' then
        state <= deuce;
      end if;
    end process;
    ace <= '1' when (state = add) and serve = '1' else '0';
    fault <= '1' when (state /= game) else '0';
  end rally;
```

<table>
<thead>
<tr>
<th>$s_0$</th>
<th>$s_1$</th>
<th>$S$</th>
<th>$V$</th>
<th>$A$</th>
<th>$F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>01</td>
<td>01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>01</td>
<td>01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>10</td>
<td>01</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>11</td>
<td>00</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>01</td>
<td>01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>10</td>
<td>01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>10</td>
<td>00</td>
<td>01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>11</td>
<td>00</td>
<td>01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>00</td>
<td>10</td>
<td>00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>01</td>
<td>10</td>
<td>00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>10</td>
<td>00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5. (10 points). The simulation output shown below came from the circuit with the partial VHDL specification shown below. Complete the specification in a way that is consistent with the simulation output shown.

library IEEE;
use IEEE.std_logic_1164.all;

entity whatsit is
  port (
    clk: in STD_LOGIC;
    reset: in STD_LOGIC;
    foo: in STD_LOGIC;
    bar: in STD_LOGIC;
    q: out STD_LOGIC_VECTOR(7 downto 0)
  );
end whatsit;

architecture whatsit_arch of whatsit is
signal reg: STD_LOGIC_VECTOR (7 downto 0);
begin
  process begin
    wait until clk = '1';
    if reset = '1' then
      reg <= x"00";
    elsif foo = '1' then
      reg <= not reg;
    elsif bar = '0' then
      reg <= reg(6 downto 0) & "0";
    end if;

    end process;
    q <= reg;
  end whatsit_arch;
6. (10 points) The simulation output below shows the control signals from the basic processor during the execution of five instructions. Each instruction is “boxed” for clarity. For each instruction, write the name of the instruction being executed. A list of the processor’s instructions appears below.

The first instruction loads the ACC from the IR, so it must be an immediate load, 8xxx.
The second instruction loads the ACC through the ALU. Since the memory is providing the operand, it must be a direct add, 2xxx.
The third instruction loads the PC from the IR. Since the ACC is not zero, it must be a direct branch, 4xxx.
The fourth instruction is an indirect operation, and since it is writing to the memory, it must be an indirect store, 7xxx.
The fifth instruction loads the ACC from the ALU and since neither the IR or the memory is supplying the operand, it must be a negate instruction, 3000.

0xxx load ACC with value stored in memory word xxx
1xxx store value in ACC into memory word xxx
2xxx add value in memory word xxx to value in ACC
3000 negate the value in ACC
3001 halt
4xxx change the value of PC to xxx
5xxx if the value of ACC is zero, change PC value to xxx
6xxx load ACC with value whose address is stored in word xxx
7xxx store ACC value into word whose address is in word xxx
8xxx change ACC value to xxx
9xxx add xxx to the value in ACC
7. (15 points) Consider the two sequential circuits $A$ and $B$ that are shown below. The timing diagrams show the input/output timing of the signals that go between the two blocks. Specifically, for the input signals, the shaded regions are the places where the signals may change without violating the setup and hold time requirements of the flip flops within the sequential circuits. For the output signals, the diagrams show when the outputs may be changing. Times shown are in ns.

What is the largest clock skew for which the combined circuit will work without risk of hold time violations?

If the clock rises more than 4 ns earlier at $A$ than at $B$, we can get a hold time violation at $B$. Similarly, if the clock rises more than 3 ns earlier at $B$ than at $A$, we can get a hold time violation at $A$. So, the maximum clock skew we can tolerate is 3 ns.

Suppose the maximum clock skew is actually .8 ns larger than this, how can you modify the circuit to eliminate the resulting hold time violations? Assume that an inverter has a delay of between .5 ns and 2 ns.

By adding a pair of inverters on the signal from $B$ to $A$, we can avoid the hold time violation.

With this modification, what is the smallest clock period for which the circuit can be expected to work without setup time violations? Assume the same clock skew as above.

From $A$ to $B$, we need $7 + 5 + 3.8 = 15.8$ ns to avoid setup time violations. From $B$ to $A$ we need $6 + 5 + 4 + 3.8 = 18.8$ ns. So, overall, we need 18.8 ns.
8. (6 points) Consider a 1 Mbit SRAM with 8 bit words. How many address bits are needed to address all the words? How many address bits are used for the row decoder, assuming the memory array has the same number of rows as it has columns?

1 Mbit is \(2^{20}\) bits or \(2^{17}\) bytes, so we need 17 address bits.

The memory array will have \(1024 = 2^{10}\) rows and columns, so 10 bits for the row decoder.
9. (15 points) Consider an extension to the basic processor in which there is an 8 entry 2-way set associative cache for both instructions and data. The cache entries are each one word long and are addressed using the low order three bits of the main memory address. The upper 13 bits of the main memory address are used as a tag. Assuming that the cache is empty initially, show the contents of the cache after the second pass through the loop. Assume that when the program starts, the values in words 0, 1 and 2 of the memory are 5, 6 and 7, respectively.

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0010</td>
<td>Store sum here</td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>Pointer to “next” value here</td>
<td></td>
</tr>
<tr>
<td>0012</td>
<td>start: 8000 (load “00”) initialize sum</td>
<td></td>
</tr>
<tr>
<td>0013</td>
<td>1010 (store 10)</td>
<td></td>
</tr>
<tr>
<td>0014</td>
<td>1011 (store 11) initialize pointer</td>
<td></td>
</tr>
<tr>
<td>0015</td>
<td>loop: 8010 (load “10”) if pointer=10, then quit</td>
<td></td>
</tr>
<tr>
<td>0016</td>
<td>3000 (negate)</td>
<td></td>
</tr>
<tr>
<td>0017</td>
<td>2011 (add 11)</td>
<td></td>
</tr>
<tr>
<td>0018</td>
<td>5026 (if 0 goto 26)</td>
<td></td>
</tr>
<tr>
<td>0019</td>
<td>6011 (load *11) sum = sum + *pointer</td>
<td></td>
</tr>
<tr>
<td>001a</td>
<td>2010 (add 10)</td>
<td></td>
</tr>
<tr>
<td>001b</td>
<td>1010 (store 10)</td>
<td></td>
</tr>
<tr>
<td>001c</td>
<td>8001 (load “1”) pointer = pointer + 1</td>
<td></td>
</tr>
<tr>
<td>001d</td>
<td>2011 (add 11)</td>
<td></td>
</tr>
<tr>
<td>001e</td>
<td>1011 (store 11)</td>
<td></td>
</tr>
<tr>
<td>001f</td>
<td>4015 (goto 15)</td>
<td></td>
</tr>
<tr>
<td>0020</td>
<td>end: 3001 (halt)</td>
<td></td>
</tr>
</tbody>
</table>