From the book,

10.1 (20 points) (a) & (b) for q s-a-0 and d s-a-1
(a) Determine the necessary inputs to the following network to test for q s-a-0 and d s-a-1.
(b) For this set of inputs, determine which other stuck-at faults can be tested.

<table>
<thead>
<tr>
<th>q s-a-0</th>
<th>ABCD = 1000 detects q s-a-0 and also detects c s-a-1, d s-a-1, r s-a-0, t s-a-1, v s-a-0, and F s-a-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>d s-a-1</td>
<td>ABCD = 1100 detects q s-a-0 and also detects a s-a-0, r s-a-0, t s-a-1, v s-a-0, and F s-a-0</td>
</tr>
<tr>
<td></td>
<td>ABCD = 1000 detects d s-a-1 and also detects q s-a-0, c s-a-1, r s-a-0, t s-a-1, v s-a-0, and F s-a-0</td>
</tr>
<tr>
<td></td>
<td>ABCD = 0000 detects d s-a-1 and also detects c s-a-1, p s-a-1, q s-a-0, r s-a-0, s s-a-0, and t s-a-1</td>
</tr>
<tr>
<td></td>
<td>ABCD = 0100 detects d s-a-1 and also detects a s-a-1, c s-a-1, p s-a-0, r s-a-1, t s-a-1, u s-a-1, v s-a-1, and F s-a-0</td>
</tr>
</tbody>
</table>

10.2 (20 points) Find a minimum set of tests that will test all single stuck-at-0 and stuck-at-1 faults in the following network. For each test, specify which faults are tested for s-a-1 and s-a-1.

<table>
<thead>
<tr>
<th>abcdefi</th>
<th>0001001 a s-a-1, b s-a-1, c s-a-1, g s-a-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>abcdefi</td>
<td>1000001 d s-a-1, e s-a-1, f s-a-1, h s-a-1</td>
</tr>
<tr>
<td>abcdefi</td>
<td>1001001 g s-a-0, h s-a-0, i s-a-0, Z s-a-0</td>
</tr>
<tr>
<td>abcdefi</td>
<td>0101001 b s-a-0</td>
</tr>
<tr>
<td>abcdefi</td>
<td>0011001 e s-a-0</td>
</tr>
<tr>
<td>abcdefi</td>
<td>1000101 e s-a-0</td>
</tr>
<tr>
<td>abcdefi</td>
<td>1000011 f s-a-0</td>
</tr>
<tr>
<td>abcdefi</td>
<td>1001000 i s-a-1, Z s-a-1</td>
</tr>
</tbody>
</table>

10.7 (10 points) State graphs for two sequential machines are given below. The first graph represents a correctly functioning machine, and the second represents the same machine with a malfunction. Assuming that the two machines can be reset to their starting states (S0 and T0), determine the shortest input sequence that will distinguish the two machines.

<table>
<thead>
<tr>
<th>Input:</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correct Output:</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Incorrect Output:</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
From other sources
(25 points) Design a hardware multiplier circuit (M) that computes the product of two, positive 2-bit binary numbers.

A1A0 represents one 2-bit number, B1B0 represents the second 2-bit number and M3M2M1M0 represents the 4-bit product. For example, if A1A0 = 10 and B1B0 = 11, then M3M2M1M0 = 0110. Model your circuit by performing the following steps.
   a. Develop a VHDL entity declaration for the multiplier.
   b. Develop an algorithmic behavioral architectural body for the multiplier.
   c. Simulate to verify the correctness of your model.

```vhdl
entity MULTIPLIER is
  port (A: in bit_vector (1 downto 0);
        B: in bit_vector (1 downto 0);
        M: out bit_vector (3 downto 0));
end MULTIPLIER;

architecture BEHAVE of MULTIPLIER is
begin
  variable temp : bit_vector (3 downto 0);
  begin
    temp := A & B;
    case TEMP is
    when "0000"|"0001"|"0010"|"0011"|"0100"|"1000"|"1100" => M <= "0000";
    when "0101" => M <= "0001";
    when "0110"|"1001" => M <= "0010";
    when "0111"|"1101" => M <= "0011";
    when "1010" => M <= "0100";
    when "1011"|"1110" => M <= "0110";
    when "1111" => M <= "1001";
    end case;
  end process;
end BEHAVE;
```
Design a sequential circuit that converts a 4-bit Gray code into a 4-bit BCD code. The inputs and outputs are timed by the same system clock. Assume that the device receives a START pulse coincident with the left most bit of the Gray code. The remaining bits of the Gray code are received one bit at a time (from left to right). At the end of each 4-bit Gray code, the device outputs the corresponding 4-bit BCD code in parallel along with a DAV signal. The next input could start during the clock period following the last input bit or at any time after that. The device must have a RESET input that initializes the device to the correct starting state. The figure given shows a block diagram and sample timing diagram. Use the table below for the Gray and BCD codes.

<table>
<thead>
<tr>
<th>Decimal Digit</th>
<th>Gray Code</th>
<th>BCD Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>0011</td>
<td>0010</td>
</tr>
<tr>
<td>3</td>
<td>0010</td>
<td>0011</td>
</tr>
<tr>
<td>4</td>
<td>0110</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>1110</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>1010</td>
<td>0110</td>
</tr>
<tr>
<td>7</td>
<td>1011</td>
<td>0111</td>
</tr>
<tr>
<td>8</td>
<td>1001</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>1000</td>
<td>1001</td>
</tr>
</tbody>
</table>

Model the device using VHDL and do a simulation to verify your model.

```vhdl
entity SGRAY2BCD is
  port (R, SI, START, CLK: in BIT;
        BCD: out BIT_VECTOR(3 downto 0);
        DAV: out BIT);
end SGRAY2BCD;

architecture FSM_RTL of SGRAY2BCD is
  type STATE_TYPE is (S0, S1, S2, S3, S4);
  signal STATE: STATE_TYPE;
  signal SHIFT_REG: BIT_VECTOR (3 downto 0);
begin
  -- Process to update state at end of each clock period.
```
NEXT_STATE: process (R, CLK)
begin
  if (R = '0') then
    STATE <= S0;
  elsif (CLK='1'and CLK'event) then
    case STATE is
      when S0 =>
        if (START = '1') then
          STATE <= S1;
          SHIFT_REG <= SHIFT_REG(2 downto 0) & SI;
        end if;
      when S1 =>
        SHIFT_REG <= SHIFT_REG(2 downto 0) & SI;
        STATE <= S2;
      when S2 =>
        SHIFT_REG <= SHIFT_REG(2 downto 0) & SI;
        STATE <= S3;
      when S3 =>
        SHIFT_REG <= SHIFT_REG(2 downto 0) & SI;
        STATE <= S4;
      when S4 =>
        if (START = '1') then
          STATE <= S1;
          SHIFT_REG <= SHIFT_REG(2 downto 0) & SI;
        else
          STATE <= S0;
        end if;
    end case;
  end if;
end process NEXT_STATE;
--
-- Output process
--
OUTPUT: process (STATE)
begin
  case STATE is
    when S0|S1|S2|S3 =>
      BCD <= "0000";
      DAV <= '0';
    when S4 =>
      DAV <= '1';
    case SHIFT_REG is
      when "0000" => BCD <= "0000";
      when "0001" => BCD <= "0001";
      when "0011" => BCD <= "0010";
      when "0010" => BCD <= "0011";
      when "0110" => BCD <= "0100";
      when "1110" => BCD <= "0101";
      when "1010" => BCD <= "0110";
      when "1011" => BCD <= "0111";
      when "1001" => BCD <= "1000";
      when "1000" => BCD <= "1001";
      when others => BCD <= "0000";
    end case;
  end case;
end process OUTPUT;
end FSM_RTL;