a) An ASM diagram has three types of box. A decision box must follow a state and is part of that state. A conditional output box must follow a decision box and is also part of the state. A state is executed in one clock cycle.

b)
c) module protocol (input logic n_reset, clk, valid, empty, done, output logic ready);

    enum (txwait, txload, txdata, txsend) state;

    always_comb
        if (state == txload && valid)
            ready = '1;
        else
            ready = '0;

    always_ff @(posedge clk, negedge n_reset)
        if (!n_reset) then
            state <= txwait;
        else
            case (state)
                txwait: if (valid)
                    state <= txload;
                txload: if (valid)
                    state <= txdata
                else
                    state <= txsend;
                txdata: if (empty)
                    state <= txload;
                txsend: if (done)
                    state <= txwait;
            endcase

endmodule
**a) SSFM assumes that a defect causes a node to be stuck at 1 or stuck at 0 and that only one node is affected (i.e. that faults don't mask).**

**b) Fault list:**
A/0, A/1, B/0, B/1, C/0, C/1, D/0, D/1, E/0, E/1, F/0, F/1, G/0, G/1, H/0, H/1, I/0, I/1, J/0, J/1, S/0, S/1

011 sets nodes as follows:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Detects S/1, J/1, I/1, H/1, G/1, F/1, E/1, C/0, B/0, A/1 (not D/0)

By analogy, D/1 would be found by 101 or 110. By inspection, we can see that G, H, I, J are 1 with 001, 010, 100, 111, respectively. Therefore need 6 patterns. Show working, as above, for each case.

(There are other ways to do this e.g. sensitive path algorithm.)

**c) Maybe. There is a difference between structure and function. The approach given attempts to sensitise nodes, based on the structure. A different implementation has the same function, but a different structure.**
a) The left device is a 4 bit shift register with parallel load.
   • SRG4 – 4 bit shift register
   • R active low asynchronous reset (no dependency)
   • M1/M2 mode control – mode 1 when input is 1, mode 2 when input is 0
   • C3 positive edge triggered clock.
   • 1→ shift right on clock edge.
   • 2,3D – D type load in mode 2 and clock

The right device is an exclusive NOR gate

b) After a reset, the sequence is:
   0000, 1000, 1100, 1110, 0111, 1011, 1101, 0110, 0011, 1001, 0100, 1010, 0101, 0010, 0001, 0000

The unused state is 1111. The circuit could enter this state following a power supply glitch.

c)

```verilog
module counter (input logic [3:0] a,
                output logic [3:0] q,
                input logic clk, n_reset, mode);

always_ff @(posedge clk, negedge n_reset)
  if (!n_reset)
    q <= '0;
  else
    if (mode)
      q <= ~((q[0] ^ q[1]), q[3:1]);
    else
      q <= a;
endmodule
```

| MARKS | 9 | 7 | 9 |
a) SISO allows registers of a sequential system to be reconfigured as a shift register. This allows the system to be put into any state and for the state of the circuit to be scanned out. The problem of testing the circuit is therefore reduced to that of testing the combinational logic. (A brief description of how SISO is used will gain full marks.)

\[ P^* = \overline{A} Q + P \overline{Q} \]
\[ Q^* = A \overline{Q} + A \overline{P} \]
\[ Z = P \]

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>A</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>10</td>
<td>1</td>
</tr>
</tbody>
</table>

Will remain stuck in this state if ever entered

Starting in state 00, sequence 101 will go to states 01, 11, 00, so Z will be 0, 0, 1, 0.

If X/0, state table becomes:

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>A</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>10</td>
<td>1</td>
</tr>
</tbody>
</table>

Circuit remains stuck in state 00. Z will stay at 0. NB this does not uniquely determine that X/0.

c) (One solution)
\[ P^* = \overline{A} \overline{B} Q + P \overline{B} \overline{Q} + B Q \]
\[ Q^* = \overline{A} \overline{B} Q + A \overline{B} \overline{P} + B A \]
\[ Z = P \]
Simplifies to:
\[ P^* = \overline{A} Q + P \overline{B} \overline{Q} + B Q \]
\[ Q^* = \overline{A} \overline{B} Q + A \overline{P} + B A \]
a) wire allows three-state logic to be modelled and to resolve contention. Logic allows 3 state logic, but does not resolve contention.

```markdown
assign y = enable ? a : 'z;
```

b) Circuit is a tristate bus register.

```markdown
module busreg (inout wire sysbus, 
             input logic clk, bus_ctrl, load, reset);

logic data;

assign sysbus = bus_ctrl ? data : 'z;

always_ff @(posedge clk, posedge reset)
  if (Reset)
    data <= '0;
  else
    if (load)
      data <= sysbus;

endmodule
```

c)

```markdown
module testbusreg;
  wire sysbus;
  logic clk, bus_ctrl, load, reset;

busreg b0 (.*);

initial
  begin
    reset = '0;
    #5ns reset = '1;
    #5ns reset = '0;
    end

initial
  begin
    clk = '0;
    forever #20ns clk = ~clk;
    end
```
Contention results in an X state. Therefore include a line like:

assert (sysbus != 'x);