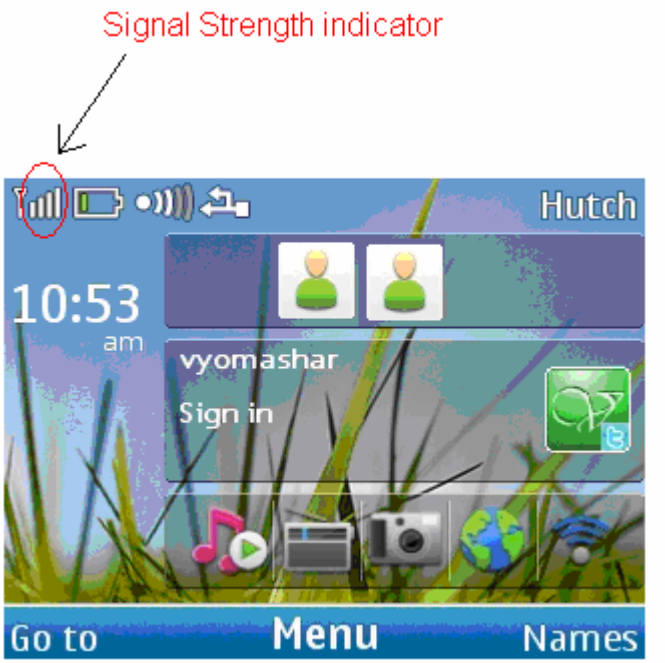
Project: Moore FSM for Signal Strength Indicator, Implemented in Logisim

---

Assume that you are an embedded systems designer at NOKIA.
Your job is to program the LCD display related to the signal-strength indicator on the cell-phone's screen.



You will need to create a Moore - FSM (Finite State Machine), which will control the signal strength indicator on the screen. Assume that the area on the LCD screen that displays the signal-strength bars, consists of a grid of 5x5 LCD pixels as shown below in Fig. 1(a). (Also, Fig. 1(a) illustrates the condition of having signal-strength of three bars). Fig.1(b) indicates the numbering of the pixels

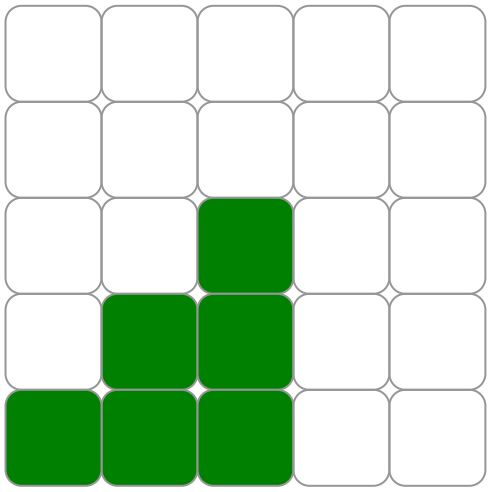


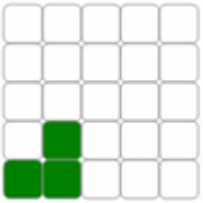
Fig. 1 (a)

Fig.1 (b)

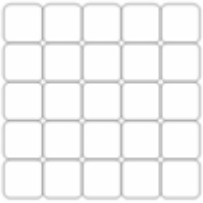
Specifications for the FSM:

- Assume that all multi-bit binary numbers/vectors are in little-endian format.
- The FSM will have a (multi-bit) input (*SS*), indicating Signal-Strength.
    - The range of valid values for *SS* is: $0 \leq SS \leq 100$
    - Also, assume that [6:0] *SS* is a binary vector of 7 bits
- The FSM will have a (multi-bit) output, [24:0] *pixels* (which is a binary vector of 25 bits), which will make the LCD pixel turn ON or OFF. (See Fig. 1(b) for pixel / vector numbering)
    - All pixels are green in color when ON (select green color in Logisim)

    - For example, if *pixel*s have value 0_0000_0000_0000_0000_110_0001 then it will display the following two signal-strength bars)

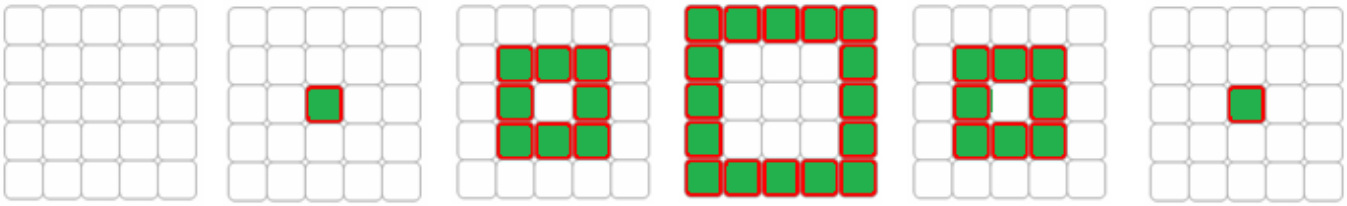


    - For example, if *pixel*s have value 0_0000_0000_0000_0000_0000_0000 then it will display the following: All LCD pixels are OFF in the image below – means a blank

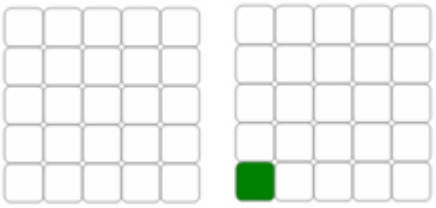


Functioning of the FSM:

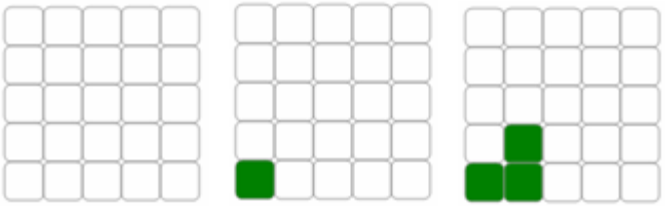
- If *SS* = 0, then the LCD-pixels will display the following images in a loop (the display jumps from one image to the other at every active clock edge). Obviously, each image can be generated by a particular value of the binary output vector *pixels*.



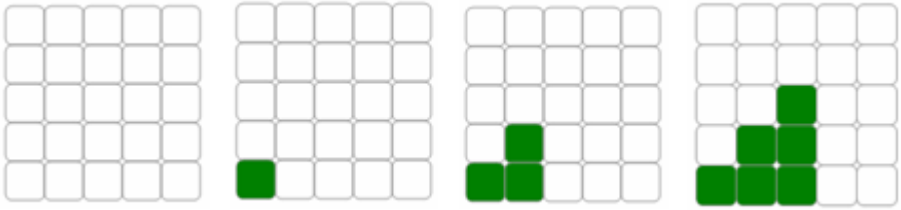
- If $1 \le SS \le 20$, the LCD-pixels will display the following images in a loop.



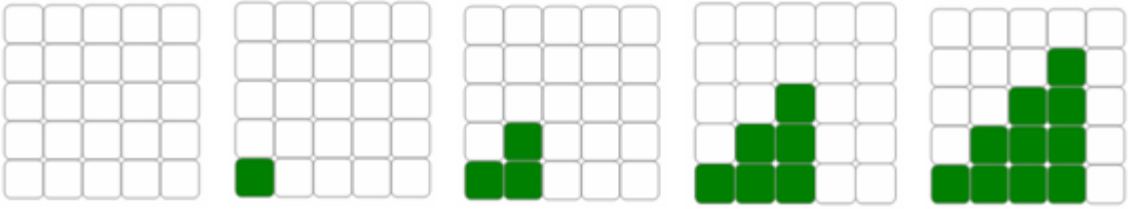
- If $21 \le SS \le 40$, the LCD-pixels will display the following images in a loop.



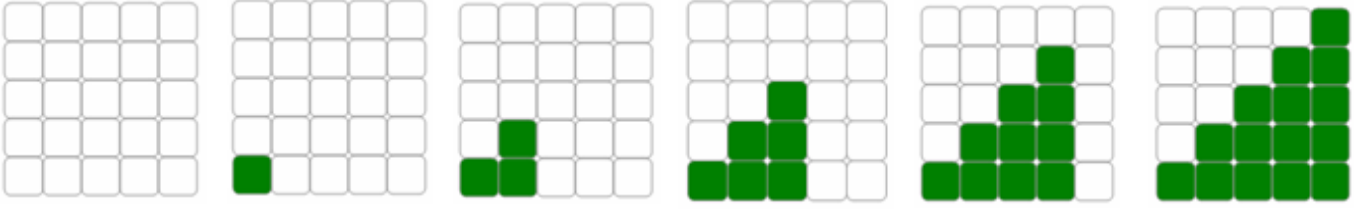
- If $41 \le SS \le 60$, the LCD-pixels will display the following images in a loop.



- If $61 \le SS \le 80$, the LCD-pixels will display the following images in a loop.

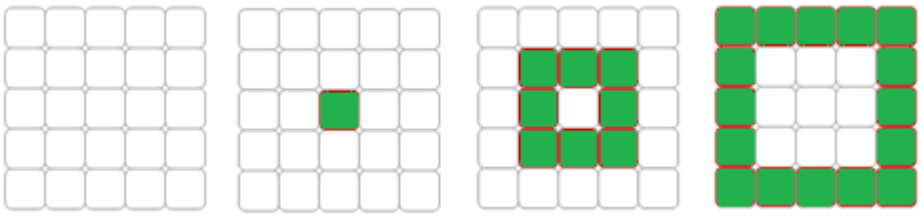


- If $81 \le SS \le 100$, the LCD-pixels will display the following images in a loop.



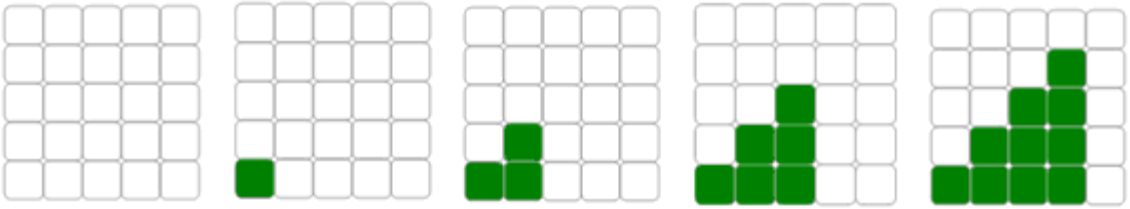
More specifications about the state diagram implementation:

- If you are in one of the states corresponding to the display of $1 \leq SS \leq 100$ (in your state diagram), and if the value of *SS* goes to 0, then at the next clock edge, you will jump to the first image corresponding to display of *SS* = 0 (shown on previous page)
- If you are in one of the states corresponding to the display of *SS* = 0, and if your *SS* changes to a value greater than 0, then you will jump to the last image corresponding to the display the of the particular SS range (in which your new SS value lies).
  - For example, when *SS* = 0, obviously you would be displaying the images for *SS* = 0 (shown on previous page). Now before you complete the loop of these images, …



  - Suddenly your *SS* = 78, then you should jump to the last image corresponding to the range $61 \leq SS \leq 80$, in which *SS* = 78 lies. Hence, you jump to the following image:

  - After this if *SS* remains constant (*SS* = 78), you would continue with your image loop, as shown below:

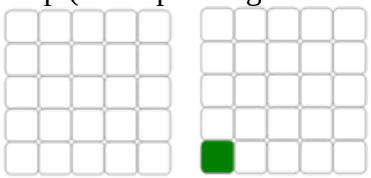


- If you are in one of the states corresponding to the display of $1 \leq SS \leq 100$ (in your state diagram), and if the value of *SS* changes:
  - If *SS* goes to a smaller value (such that the next state would become invalid in terms of the number of bars), then jump to first image of the valid range.
  - If *SS* goes to a larger value (such that the next state would still be valid in terms of the number of bars), then jump to the next state, and continue.
  - For example, if *SS* = 78 and you are in a state which displays the following image:

Now if SS goes to 95, then you should continue to the state which displays the following (and then continue with remaining images of the range $81 \leq SS \leq 100$):

- For example, if *SS* = 78 and you are in the state which displays the following image:

Now if *SS* goes to 12, then you should jump to the state which displays the first image of the following image loop (corresponding to the range $1 \leq SS \leq 20$):

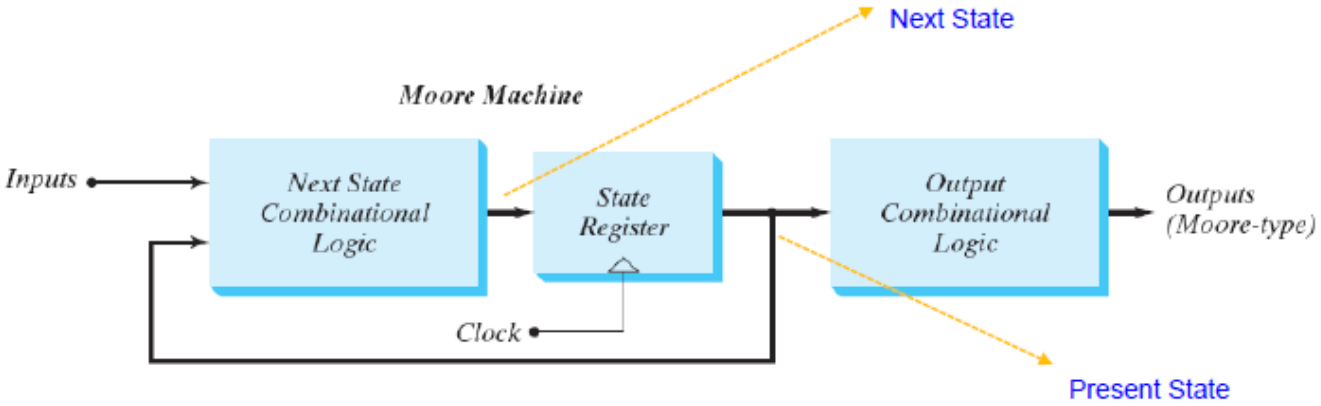


Question-1

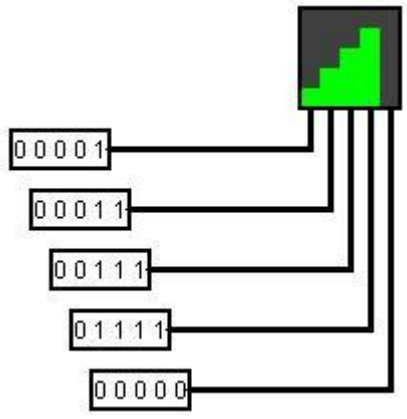
Draw the state diagram for the Moore FSM, which will operate according to the above specifications. Clearly label each state, the outputs at each state, and the inputs on the arcs.

Clearly mention the number of bits required to represent your present state (i.e. number of D-Flipflops required)

Question-2 (Logisim Implementation)



- Implement the complete Moore FSM in Logisim using three different methods:
  - The Next-state comb. logic is implemented using two levels of MUXs; And output comb. logic is implemented using MUX
  - The Next-state comb. logic is implemented using ROM; And output comb. logic is implemented using ROM (The ROM available in Logisim can be used)
  - The Next-state comb. logic is implemented using Decoders and OR gates; And output comb. logic is implemented using Decoder and OR gates
- The 25 pixels can be implemented in Logisim using LED matrix, and making its size 5x5. Shown below is an example of LED 5x5 matrix, which currently has *pixel*s value 0_0000_0111_1001_1100_0110_0001 (See Fig. 1(b) for pixel / vector numbering)



Notes/Hints:

1. Assume two separate states that display/output a blank (blank display is needed when SS = 0; and also needed when SS>0)
2. In the ROM based implementation of the next state comb. logic, the address for the ROM = {present state, FSM inputs}
3. The input to the Moore FSM is SS, which is 7 bits. If we assume that, we need 16 states for this FSM, then 4 bits would be needed for representing present state.
   Now, if we think in terms of the size of the truth table for next state:

Each next state bit would be a function of 7+4 bits = 11 bits. Such a truth table would have thousands of rows!!!

We need to solve this problem, by doing 1-hot coding, for each of the desired ranges for the SS. Then, we can further reduce this by providing the 1-hot coding (corresponding to the desired ranges for SS), to an encoder. The encoder's output can then be used as the actual input to the FSM, rather than directly using SS as the input to the FSM