Lab 3
The LightsOut Puzzle

With Basys 3 as your target platform, design, verify, implement, and test the LightsOut Puzzle, defined below using the following specification:

**Introduction**

The LightsOut Puzzle is a puzzle involving a rectangular (usually square) matrix of lights (squares) which can be turned on and off. A move consists of flipping a "switch" corresponding to a given square, thereby toggling the on/off state of this and all four vertically and horizontally adjacent squares.

An example of a move for a 3x3 matrix is shown below:

Starting from a randomly chosen light pattern, the aim is to turn all the lamps off.

The problem of determining if it is possible to start from the set of all lights being on to all lights being off is known as the "all-ones problem." As shown in 1989 by Klaus Sutner, this is always possible for a square lattice.

Below is an example of a sequence of steps leading to solving the puzzle, starting from a randomly selected initial state. The red dot denotes a square pointed by a player using the corresponding switch. After this square is selected, the corresponding light and the vertically and horizontally adjacent lights are toggled.

A nice demonstration of this puzzle, for the 11 x 11 matrix of lights, is available at [http://www.millstone.demon.co.uk/games/lightsout/lightsout.htm](http://www.millstone.demon.co.uk/games/lightsout/lightsout.htm)

However, please note that this demonstration has introduced one change compared to the rules of the puzzle listed above. The light corresponding to the selected square is not toggled. This is a valid variant of the same puzzle. Your task is to implement both variants, with the Basys 3 SW0 responsible for choosing between these two variants (SW0=0 : the light of the selected square unchanged, SW0=1 : the light of the selected square toggled).
Matching Resources of Basys 3

Lights should be numbered as shown in the diagram below:

```
 1 4 7
 2 5 8
 3 6 9
```

Lights should be represented using segments A, G, and D of the three leftmost seven-segment displays.

Switches SW1..SW9 correspond to lights 1..9. The game is played by turning a switch corresponding to the selected light from the position off to the position on and back.

Additionally, three buttons are used to control the flow of the puzzle.

BTNU is used to interrupt puzzle solving and return to the initial state, thus it is also referred to as a reset button.

BTNC is used to start the puzzle, and thus is also referred to as a start button.

BTND is used to force a win, i.e., move to the winning display pattern, involving all lights blinking with the period of 2 second, even if the win has not been accomplished. As a result, this button is also called the win button.

Four Circuit Parts

The entire circuit should be clearly divided into the following four parts: Input Interface, Output Interface, Datapath, and Controller. Signals SW1p..SW9p, RSTp, STARTp, and WINp are assumed to be in the form of pulses of the width of one clock period. This kind of signals are denoted in the following diagrams using the name ending with “p”.

Furthermore, it suffices to consider 0 and 1 as the only possible values for different types of matrices. Since matrix addition is commutative, it follows that the order in which the moves are performed is irrelevant.
**Input Interface: Button Debouncing and Rising Edge Detection**

Switches SW1-SW9 and all buttons used in the game must have an associated Debouncing circuit followed by the Rising Edge Detector (RED), to ensure accurate reading of the user input.

**Output Interface: SSD Driver**

Seven segment displays should be driven using a Seven Segment Display Driver (SSD_DRIVER), assuming that all segments of each display share the same anode. Please note that some small modifications are required compared to the traditional SSD_DRIVER used for displaying hexadecimal digits.

**Datapath: Random Pattern Generator**

The game should use a 9-bit Linear Feedback Shift Register (LFSR). This LFSR should be used in combination with the reset and start buttons to generate a truly random pattern of lights at the beginning of each game. The implemented LFSR should have the length L=9, the period $2^9-1=511$, and be set by the reset button to a non-zero state.

The LFSR shown in Fig. 1 fulfills all three mentioned above conditions.

![Block diagram of a 9-bit LFSR, with the period 511, initialized to the sequence “010101010”](image)

After the reset button is pressed, the value of LFSR should change every clock period of the 100 MHz clock, i.e., every 10 ns. When the start button (BTNC) is pressed, the current value of the LFSR output should be used to determine the initial pattern of lights.

*The sketch solution for the entire Datapath and the corresponding Controller is provided below. You are welcome to use this solution (with some possible changes and extensions) or develop an alternative solution. However, in both cases you are expected to use the same methodology based on:

1. Dividing the circuit into the Input Interface, Output Interface, Datapath, and Controller.
2. Describing the Datapath using a hierarchical block diagram
3. Describing the Controller using an ASM chart.
4. Converting the block diagram and ASM chart into synthesizable code following coding conventions discussed in this course or described in the textbook.
5. Verifying the code first using simulation, and then using Basys 3 board.*
Top-level Circuit:

A block diagram of the top-level circuit is shown in Fig. 2.

Datapath: Remaining Parts

The hierarchical block diagram of the LightsOut Datapath is shown in Fig. 3 (top-level), Fig. 4 (STORAGE), Fig. 5 (NEIGHBORS), and Fig. 1 (LFSR). The details of the circuit NEIGHBORS and the entire circuit SSD_CONVERTER are left up to you to design.
Fig. 3: Block diagram of the LightsOut_Datapath.

Fig. 4: Internal block diagram of the component STORAGE.
Fig. 5: Internal block diagram of the component NEIGHBORS.

Controller:

An ASM chart describing the operation of the LightsOut controller is shown in Fig. 6.
Fig. 6: An ASM chart describing the operation of LightsOut_Controller.
Required Tasks:

1. Revise the provided block diagram or draw a new block diagram describing the Datapath of the LightsOut Puzzle. Substantially different block diagrams, if correct and following good design practices will be rewarded with bonus points.
2. Revise the provided Algorithmic State Machine (ASM) chart or draw a new ASM chart describing the Controller of the LightsOut Puzzle. Please note that multiple state machines, working in parallel, can be used to achieve the required behavior.
3. Translate the block diagram and ASM charts to VHDL.
4. Develop a simple testbench with two versions of timing constants, one used for simulation, and the other used for the actual operation of the circuit on the board.
5. Perform functional simulation of your code
7. Prepare the correct XDC (Xilinx Design Constraint) file.
8. Implement your circuit using Xilinx Vivado.
9. Check thoroughly all implementation reports. Pay attention to timing, resource usage, and pin allocations.
10. Perform post-synthesis simulation of your circuit using ModelSim or Vivado Simulator.
11. Perform static timing analysis.
12. Check very carefully your pin allocations listed in the report files, and only if these pin allocations are correct, download your bitstream to the FPGA board.
13. Test the operation of your circuit experimentally using the Basys 3 FPGA Board.

Deliverables:

1. All block diagrams describing the Datapath of your circuit.
2. All ASM charts describing the Controller of your circuit.
3. All source files used for synthesis and implementation of your circuit.
4. A simple testbench.
5. User constraint files.
6. All synthesis and implementation report files.
7. RTL netlist.
8. Simulation waveforms from the functional and post-synthesis simulations, proving the correct operation of your circuit (in the PDF format).
10. Your own report containing at least the following additional information:
   - Resource utilization.
   - Minimum clock period and maximum clock frequency after synthesis and after implementation.
   - List of any deviations from the original specification.
   - Difficulties encountered and lessons learned.
## Important Dates

<table>
<thead>
<tr>
<th>Event</th>
<th>Tuesday Section</th>
<th>Wednesday Section</th>
<th>Friday Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hands-on Sessions and Introductions to the Experiment</td>
<td>02/11/2020</td>
<td>02/12/2020</td>
<td>02/14/2020</td>
</tr>
<tr>
<td></td>
<td>02/18/2020</td>
<td>02/19/2020</td>
<td>02/21/2020</td>
</tr>
<tr>
<td>Deliverables Due</td>
<td>02/25/2020</td>
<td>02/26/2020</td>
<td>02/28/2020</td>
</tr>
<tr>
<td></td>
<td>8:30am</td>
<td>6:50pm</td>
<td>8:10am</td>
</tr>
<tr>
<td>Demo and Q&amp;A</td>
<td>02/25/2015</td>
<td>02/26/2019</td>
<td>02/28/2019</td>
</tr>
<tr>
<td></td>
<td>9:00-11:40am</td>
<td>7:20-10:00pm</td>
<td>8:40-11:20am</td>
</tr>
</tbody>
</table>