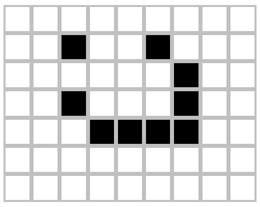
# Life on the far side of the Moon

## Theoretical background

The Game of Life is a cellular automaton devised by the John Horton Conway in 1970. This unique simulation is classified as a zero-player game, which means that once the initial setup is complete, the system evolves on its own without requiring any additional input. Users engage with the Game of Life by setting up an initial pattern and then watching how it develops over time. The system is notable for being Turing complete, meaning it has the capability to simulate a universal constructor or any other Turing machine, demonstrating its computational power despite its simple rules.

The universe is limited to [x,y]. Each cell can be in one of two states: alive or dead (alternatively described as populated or unpopulated) -> dead = 0, alive = 1. Every cell interacts with the eight cells surrounding it, including those touching its sides and corners. As time progresses, the following rules determine how the cells change:

1. A living cell with fewer than two living neighbors will die, simulating isolation or underpopulation.
2. A living cell with either two or three living neighbors will survive to the next round.
3. A living cell with more than three living neighbors will perish, mimicking overcrowding or overpopulation.
4. A dead cell that has exactly three living neighbors will come to life, representing reproduction.



## Goal

Simulate behaviour of life on the far side of the Moon. The time units value will appear on the first sample. Run the simulation for the specified duration and send results to an output port.

## Provided module

task_8.sv module is the top module of your design. You can modify it, add new modules and IPs.

You only get the outline of the life module. Inputs and outputs are declared as:

*Table 1 List of inputs and outputs of the* ***task_8*** *module.*

| Signal name | Signal type | Bit length |
|---|---|---|
| i_clk | Input | 1 bit |
| i_rst | Input | 1 bit |
| i_first | Input | 1 bit |
| I_last | Input | 1-bit |
| i_data | Input | 8-bit |
| i_valid | Input | 1 bit |
| o_data | Output | 8-bit |
| o_valid | Output | 1-bit |
| o_last | Output | 1-bit |

## Input and output interfaces

In one packet task receives 8-bit value. The matrix will be 64 by 64. When i_first and i_valid are high, value in i_data port is time units. After that in i_data port first population will be send.

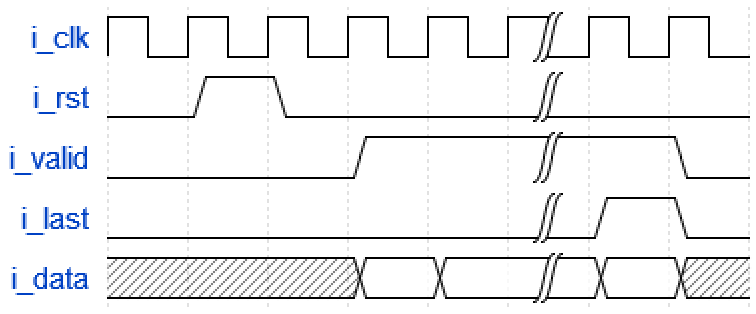
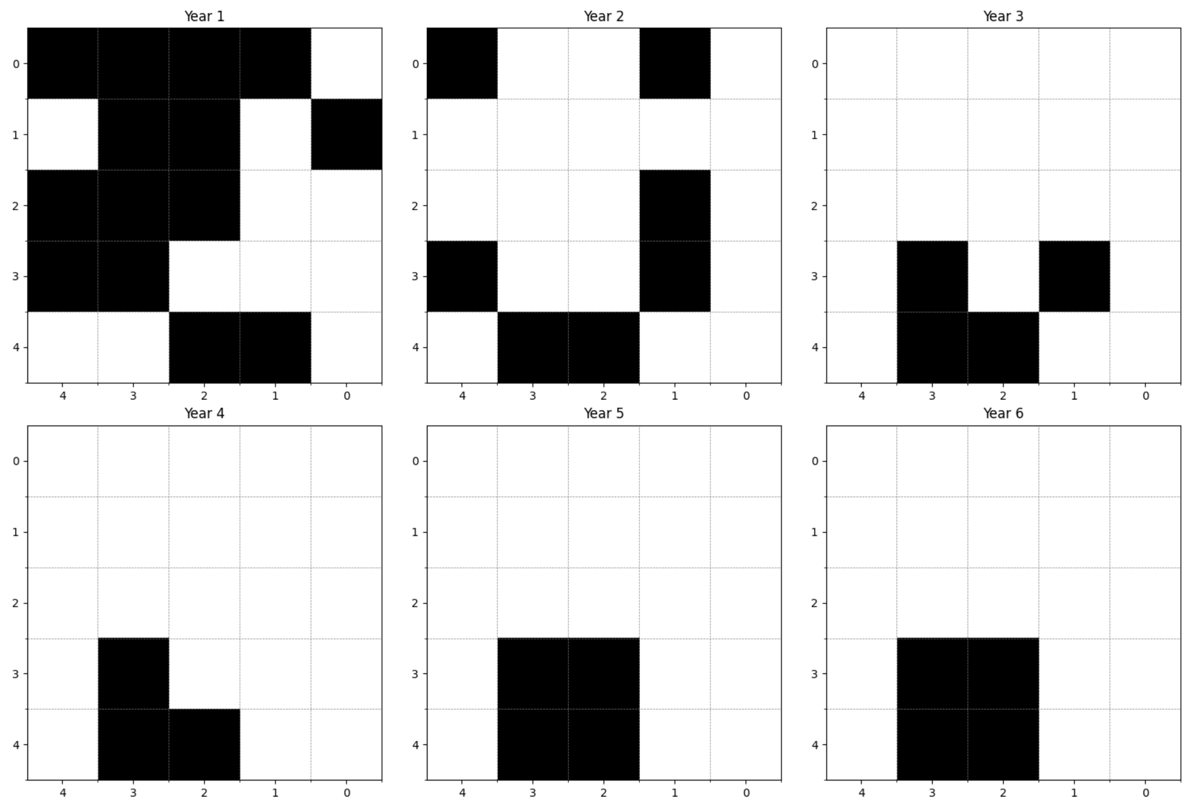


*Figure 1 Input waveforms.*

Data is sent in 8-bit chunks. First chunk is [7:0] of matrix, second is [15:8] of matrix and so on, row by row.

Example of game of life evolution:



## Evaluating the task

For the task to be considered correctly performed, the o_data value data packet must be exact with the reference output values.

In Development mode, the task will be tested using a single set of data that is randomized on each run. In Evaluation mode, the task will be tested using 3 fixed sets of data, which remain the same across all runs and are identical for every team (for more information, check chapter 2.3.1 **Testing modes** in the F**PGA_Hackathon_2025_Project_Guide** document).

You earn 5 points for correctly processing each test vector. In Development mode, this value simply indicates that the task passed the test. Evaluation mode, your total base score is calculated by multiplying 5 by the number of test vectors, so you can earn up to 15 points.

Additional points may be awarded for **resource utilization, timing performance** (for more information, check chapter 2.3.2 **Bonus Points** in the **FPGA_Hackathon_2025_Project_Guide** document).