Lunar Image Restorer

Theoretical background

On the other side of the Moon, there is a lot of cosmic radiation. Unfortunately, this has a negative impact on the operation of the camera on our rover. Especially on the Low-Resolution Camera (LowResCam). Some pixels in the image are damaged and very bright due to the high energy of particles hitting the matrix. Fortunately, we can easily fix this. To improve the image, we need to detect very bright pixels and replace them with the average value of the neighboring pixels.

Goal

Your task is to create a module that will receive a data packet, store it, restore broken pixels, and send it back to the Judge. You will get an image of 64x48 of 8-bit pixels. The image is sent to you **row by row** and should be sent back in the same way. If you find a dead pixel in the image, very bright with a value of 0xFF, you need to replace it with the average value of the neighboring pixels. If the broken pixel is somewhere in the center of the image, there are eight neighboring pixels, but be careful, if the pixel is somewhere on the edges or in the corner, there are fewer neighboring pixels. We can assume that the shielding of our camera is good enough that particle impacts are not too frequent and **do not occur close to each other**.

Example

	0	1	2	3	4	5	6		58	59	60	61	62	63
0														
1														
2														
3														
4														
44														
45														
46														
47														

Red – Bad pixel (0xff)

Green – Good pixel (not 0xff) used to calculate average and restore bad pixel

White – Good but not used in calculations

Provided module

You can edit only the *task_14.sv* file. If you wish, you may add other files for supportive submodules (like memories).

You only get the outline of the ***task_14*** module. Inputs and outputs are declared as shown below in Table 1.

Table 1 List of inputs and outputs of the task_14 module.

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

Input and output interfaces

In one packet the task receives a sequence of data on the *i_data* port. The maximal amount of data is 4KB. The beginning of the data block is marked by the *i_first* signal and the end by the *i_last* signal. Throughout the data transfer, the *i_valid* signal is high.

Task receives data in the way shown in the figure 1below:

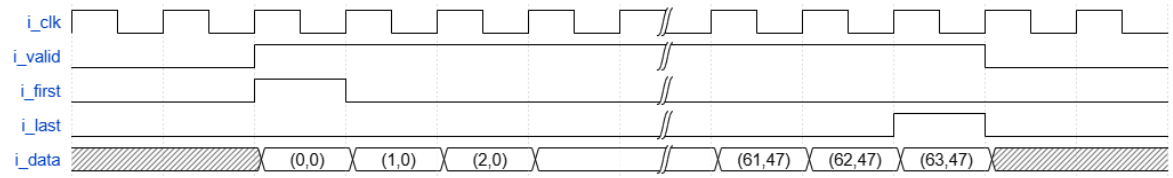

Figure 1 Input waveforms.

The task should return a sequence of data using the *o_data* port. Along with the data the *o_valid* signal should be set, and on the last cycle of data the *o_last* signal should go high.

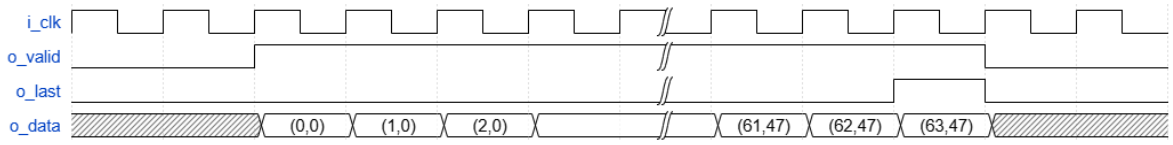

Figure 2 Output waveforms.

Evaluating the task

The evaluation is done by comparing the sequence returned by the task with the expected sequence calculated by the reference model in the Judge. If the sequences are the same, the task is marked as correctly solved.

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 **FPGA_Hackathon_2025_Project_Guide** document).

You earn 6 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 6 by the number of test vectors, so you can earn up to 18 points.

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