LED AND BUTTONS

To manipulate the process during the computation, we use 2 buttons to start and output results. By pressing the start button, XMOS should start to reading and processing an input image and evolve for numerous times. To help the user understand the processing cycle, we use a green LED to indicate reading and flashing for processing inputs, however to prevent a deadlock or infinite looping happening during runtime, we also make an alternative green LED that reversing its state for each evolution of the game, thus the progression of the game can be visualized. To stop at a certain stage of evolution and produce the current result, we use another button to end the game, for exporting the result, we use blue LED to present the progression. Nevertheless, the user need to be able to pause the game at any given time during processing, this has already been done by physically tilting the board, and resume if placed horizontal again. But we also present this state with a red LED, and also displaying information of the current game on the console so the user can check the status of the game by pausing the process.

When we approach the idea of displaying information, we would like to keep the process as fast as possible, which means more workers and less channel. However, since LED and Button functions require to be visited constantly so it can check the input and output constantly, if they are separated, then that indicates each workers will need 2 channels connect to each functions instead of 1. Thus we made a decision of merging them into 1 function with case() that decides which function the data should be operated on, which mean the channel will be reduced, and the amount of workers will be increased.

INTERFACES

The main aspect of the process is by converting the idea of parallel process into multiple independent workers by implementing interfaces. The main advantage of using interface is that we can use many functions like method in Java to communicate to other workers, these functions are already compact and provide concise and precise information.

We chose to divide the image into rows of information, so that we can minimize the amount of neighbours, not only it will reduce the bottleneck of communication, but also maximize the amount of workers available by minimizing channels requires. Since there are only 2 neighbour for each workers, then the data required from other workers are top and bottom rows for each corresponding neighbours. However in order to require data from other workers, workers itself need to be able to give out information, thus we created a server/client status for workers, when asked for data, the worker will act like a server which provide the data needed, and act as a client when ask for information from other workers.

However when we approach this idea of communication, we discover a deadlock where if all workers are asking for information from other workers, no workers will actually send information. So instead, we make all workers calculate the inner cells where cells require no information outside of the worker itself, then the system would create a queue where the top worker will be given permission to acquire data from other workers, this permission will be passed on to the next workers. Nevertheless, this is a bottleneck for the process, since the other workers which have already done the process of inner cells need to wait for the first worker. To avoid this issue, we can make sure that whenever a worker gather information from neighbours, its neighbour workers will not have permission to gather information but only given out. Thus we can shorten the process by a half at max, since the workers which have permission to ask for information are separated by the one who don’t.