Assignment 2 (XC-1A Game-of-Life Process Farm)

- This assignment is the 2nd assessed piece of unit coursework
- It is to be completed in pairs. (report any change to the team structure to the course director BEFORE starting your assignment)
- It is worth 15% of the unit mark
- Submission: Every student is required to upload their full piece of work (incl all XC source files) as a single ZIP file to SAFE before 23:59:59, Monday 8th December 2014. Make sure you submit it early enough (not last minute!) to avoid upload problems. Each member of a team has to upload an identical copy of the team's work.

- Assessment: You will present your submitted program running (using the XC-1A boards) in the labs on 12th and 19th December 2014. You will need to attend one of these lab sessions to get a mark. At these labs, we will ask you questions about your work and you will be able to showcase the merits of it.
- Do not attempt to plagiarise or copy code between teams etc. It is not worth it, be proud of your own work! We will ask you questions about your work in the labs - so you must understand the code your team developed in detail.

Your Task: XC-1A Game-of-Life Process Farm

Introduction: In 1970 the British mathematician John Horton Conway devised a cellular automaton named 'The Game of Life'. The "game" resides on a 2-valued 2D matrix, i.e. a binary image, where the matrix entries (call them cells, picture elements or pixels) can either be 'alive' (value white 255) or dead (value black 0). The game evolution is determined by its initial state and requires no further input. Every cell interacts with its eight neighbour pixels, that is cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions may occur:

- any live cell with fewer than two live neighbours dies
- any live cell with two or three live neighbours is unaffected
- any live cell with more than three live neighbours dies
- any dead cell with exactly three live neighbours becomes alive

A user can interact with the Game of Life by creating an initial configuration and observing how it evolves. The game has the power of a Universal Turing Machine: that is, anything that can be computed algorithmically can be calculated within it. Evolving such complex, deterministic systems is an important application of scientific computing, often making use of parallel architectures and concurrent programs running on large computing farms. Subjects as diverse as molecular biology, meteorology, biochemistry and aerospace engineering rely on simulations of this kind…

Task Overview: Your task is to design and implement a small process farm on the XC-1A board which simulates the 'Game of Life' on an image matrix. The board's buttons and LEDs should be used to control and visualize aspects of the game. The game matrix should be initialised from a PGM image file and the user should be able to export the game matrix as PGM image files. Your solution should make efficient and effective use of the available parallel hardware of the architecture by implementing farming and shared access to the game matrix from several cores based on message passing.

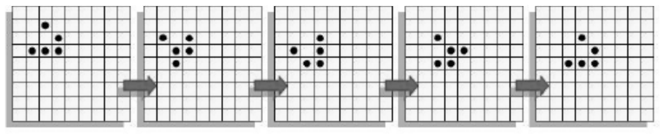

Figure 1) Example Evolution of the 'Game of Live'

Skeleton Code. To help you along, you are given a very simple code skeleton as a start that:

1) reads an image (from a PGM image file) using a thread DataInStream that produces a stream of pixel values (describing the image matrix left to right, line by line) sent on an XC channel;

2) writes a stream of pixel values from an XC channel to an image (PGM image file) using a function DataOutStream; The provided example code operates on example images of size 16x16 pixels only.

Feel free to change any of the skeleton code or add skeleton code from any previous assignment.

Process Control and User Feedback. In your application use buttons on the XC-1A board for control:

- Button A: starts reading and processing of a new image
- Button B: triggers processing to be paused and restarted by pressing Button B again, when paused use the LED clock on the board as visual feedback that indicates the progress of processing (number of rounds played)
- Button C: triggers the export of the current game state as a PGM image file
- Button D: gracefully terminates your program (all threads shut down)
- When not paused, use the LED clock on the board as visual feedback that indicates the current number of live cells in the game

Building the Process Farm. To perform the evolution of the game, your program should implement a 'distributor' or 'farmer thread' that tasks different 'worker threads' to operate on different parts of the image matrix which should be stored across the local memory of different cores to allow for images larger than permitted by 64k. The distributor is also responsible for reassembling the matrix for output.

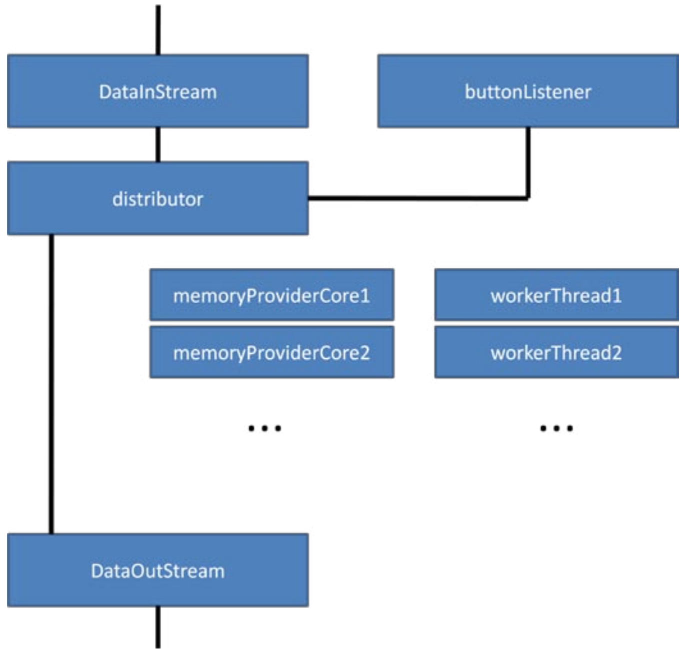

Figure 2) Some basic parts a process farm (to be extended)

The illustration above visualises some parts of the described design, feel free to add threads or channels to augment the layout.

For the subsystem distributor thread interacting with the buttonListener thread (hiding all other events and interactions) provide a basic transition diagram and a CSP specification.

Your Report. You need to submit a CONCISE (max 4 pages) report which should contain the following sections:

Functionality and Design: Outline what functionality you have implemented, which problems are solved by your implementation and how your program is designed to solve the problems. Give a basic CSP specification of the distributor thread interacting with the buttonListener thread here (hiding all other system events and interactions).

Tests and Experiments: Describe briefly the experiments you carried out, provide a selection of appropriate results and output images. This must be done at least for the example image provided and for two example images of your own choosing (showing the merit of your system). List the important factors for virtues and limitations of your system as indicated by the results.

Critical Analysis: Discuss the performance of your program with reference to the results obtained and indicate ways in which it might be improved. (Make sure your name, course, and email address appears on page 1 of the report.)

Further Task Details and Assessment Guideline.

The marker will take into account your lab presentation, your report and your source code. Find below a guideline for assessment. However, this is only a guideline and submissions will be marked on an individual basis.

To pass the assignment you need to implement a working, concurrent system that correctly evolves an image according to the rules of Game-of-Life and submit a report that discusses your implementation and results to a basic standard.

For a mark above 45 make sure you submit a clean, well documented piece of code that uses multiple worker threads.

For a mark above 50 also make sure your CSP specification is well formed and reflects your actual subsystem, your report is concise and discusses the key aspects of your system clearly.

For a mark above 55 also implement the correct button and LED behaviour as well as a graceful shutdown of all threads.

For a mark of merit above 60 also implement a system that can process images larger than 256x256 pixels and show a good understanding of the concurrency concepts relevant and/or implemented.

For a mark of upper merit above 65 use a timer to measure the throughput/processing speed of your system when evolving your game over 100 consecutive processing rounds (excluding I/O) and on different inputs (e.g. small vs. Large images, different initial conditions etc). Experiment with system parameters (e.g. synchronous vs. asynchronous channels etc) and draw conclusions about performance properties of your system in your report. You also should show a very good understanding of the concurrency concepts relevant and/or implemented.

A top first class mark (70+) is reserved for excellence. Here we would look for work that implements a more complex system allowing, for instance, dynamically adapting the strategy of storing image parts and distributing tasks to workers based on the game state or previous performance measurements, or your own ideas how this system can be turned into something more efficient, flexible and/or sophisticated … feel free to ask for the possibility of modifying the board etc to implement your own ideas. Note that for top marks you should also show an excellent understanding of the concurrency concepts relevant to the task and especially all those used in your implementation.

Keep in mind that together with the marking indications outlined above, your understanding of your own code during the lab presentation, as well as at least minimal code readability will be considered for assessment.