pde2hw user manual

Title	pde2hw (FPGA-based hardware prototypes in Processing)
Author	Nikolaos Kavvadias 2014
Contact	nikos@nkavvadias.com
Website	http://www.nkavvadias.com
Release Date	15 October 2014
Version	0.0.0
Rev. history	
v0.0.0	2014-10-15 First public release containing three hardware prototypes (gensca, imgsynth, rugca).

1. Introduction

`pde2hw` is a collection of standalone hardware prototypes that have been written in the Processing programming language (http://www.processing.org). The Processing codes that comprise this collection have been verified in FPGA-based hardware by using a high-level synthesis approach from ANSI/ISO C. The HLS tool that was used for the final implementation is HercuLeS: http://www.nkavvadias.com/hercules/

`pde2hw` comprises of the following applications:

- `gensca`: a 2D Generations cellular automaton (http://www.mirekw.com/ca/rullex_gene.html)
- `imgsynth`: an image synthesis application that does not use intermediate memory and is used for visualizing pixel generation functions of the form `z = f(x,y)`
- `rugca`: a 2D cellular automaton with 256 states using a rug-like rule (http://www.mirekw.com/ca/rullex_udll.html).

It should be mentioned that while the Processing prototypes are meant to serve as early-stage experiments prior dedicating to hardware, the first three cases (`gensca`, `imgsynth`, `rugca`) are post-mortem prototypes, i.e. the Processing codes had been developed after the FPGA implementations (which were synthesized to VHDL using HercuLeS).

2. File listing

The `pde2hw` distribution includes the following files:

/pde2hw	Top-level directory
AUTHORS	List of `pde2hw` authors.
LICENSE	The modified BSD license governs `pde2hw`.
README.html	HTML version of README.
README.pdf	PDF version of README.
README.rst	This file.
rst2docs.sh	Bash script for generating the HTML and PDF versions.
VERSION	Current version of the project sources.
/src/gensca	Processing code for `gensca`
gensca.pde	The `gensca` application.
/src/imgsynth	Processing code for `imgsynth`
imgsynth.pde	The `imgsynth` application.
/src/rugca	Processing code for `rugca`
rugca.pde	The `rugca` application.

3. Description

3.1 `gensca`

`gensca` is a generic implementation of the Generations automaton (2D, outer totalistic with decay).

Generations rules are defined in the "S/B/C" form, where:

- S - defines counts of alive neighbors necessary for a cell to survive,
- B - defines counts of alive neighbors necessary for a cell to be born.
- C - defines the count of states cells can have (including 0 state).

3.2 `imgsynth`

`imgsynth` is a static image synthesis engine for testing various algebraic effects.

The following triplets of two-variable functions which follow the form `R_f(x,y)`, `G_f(x,y)`, `B_f(x,y)` for each color component `R`, `G`, `B` have been tested:

GREY_XOR: R_f(x,y) = x ^ y, G_f(x,y) = x ^ y, B_f(x,y) = x ^ y

RGB_XOR: R_f(x,y) = x, G_f(x,y) = y, B_f(x,y) = x ^ y

RGB_ADD: R_f(x,y) = x, G_f(x,y) = x + y, B_f(x,y) = x + y

GREY_MUL: R_f(x,y) = x * y, G_f(x,y) = x * y, B_f(x,y) = x * y

RGB_MUL: R_f(x,y) = x, G_f(x,y) = x * y, B_f(x,y) = x * y

SEPIA: R_f(x,y) = 224, G_f(x,y) = 132, B_f(x,y) = 40

GRAD_RG: R_f(x,y) = x, G_f(x,y) = y, B_f(x,y) = 0

GRAD_RB: R_f(x,y) = x, G_f(x,y) = 0, B_f(x,y) = y

GREY_ADDSQ: R_f(x,y) = x*x+y*y, G_f(x,y) = x*x+y*y, B_f(x,y) = x*x+y*y

GRAD_GB: R_f(x,y) = 0, G_f(x,y) = x, B_f(x,y) = y

ADDSUBXOR: R_f(x,y) = x + y, G_f(x,y) = x - y, B_f(x,y) = x ^ y

GREY_AVG: R_f(x,y) = (x+y)»1, G_f(x,y) = (x+y)»1, B_f(x,y) = (x+y)»1

GREY_SUBSQ: R_f(x,y) = x*x-y*y, G_f(x,y) = x*x-y*y, B_f(x,y) = x*x-y*y

RGB_SUBSQ: R_f(x,y) = x, G_f(x,y) = x*x-y*y, B_f(x,y) = x*x-y*y

GREY_MAX: R_f(x,y) = MAX(x,y), G_f(x,y) = MAX(x,y), B_f(x,y) = MAX(x,y)

GREY_MIN: R_f(x,y) = MIN(x,y), G_f(x,y) = MIN(x,y), B_f(x,y) = MIN(x,y)

to which optionals masks can be applied.

3.3 `rugca`

`rugca` is the Generic implementation of a rug-like automaton (2D).

Rug rules are averaging rules using the full range of 256 possible states. To update itself in a Rug rule, every cell takes four steps.

1) Every cell calculates the sum of its 8-neighborhood states.
2) Every cell calculates the average neighbor state by dividing the sum by 8 and throwing out any remainder.
3) Every cell computes its new state by adding an increment (incr) to the average neighbour state.
4) As a final step, new state is taken modulo 256.

4. `pde2hw` usage

In order to execute the Processing applications, you have to invoke the Processing environment/IDE and then press the `Run` button with the application loaded (and visible in the editor).

Alternatively you can simply double-click on the `*.pde` file that contains the application (Windows).

5. Prerequisites

- Processing IDE (http://www.processing.org)

  The applications have been tested with version 2.2.1 of the Processing environment on Windows.