Linewars:

David Elliott and Evan Yeh

Linewars is a 2 player game where the players try to cut each other off by drawing lines. Each player starts as a single block with an initial direction. Once the game starts (with Player 1 pressing left to start the game), players start moving forward and leave behind a trailing line. Players may turn left or right by using pressing the respective buttons (Player 1: Left - Key0 Right - Key1, Player 2: Left - Key2 Right - Key3) Players lose when they collide with a drawn line (opponent’s or their own) or the edge of the game board. The color of the winning player is then shown on the screen until the reset button is pressed to reset the game. The game uses a VGA display of resolution 640 by 480 with tiles in the game being 32 by 32 pixels. There is one 20 by 15 2D array of tiles for each player indicating which tiles are occupied by that player. This game is indeed just like Tron, just renamed Linewars.

vga.vhd

**library** ieee**;**

**use** ieee**.**std\_logic\_1164**.all;**

**use** work**.**linewars\_package**.all;**

----------------------------------------------------------

**entity** vga **is**

**generic** **(**

Ha**:** integer **:=** 96**;** --Hpulse

Hb**:** integer **:=** 144**;** --Hpulse+HBP

Hc**:** integer **:=** 784**;** --Hpulse+HBP+Hactive

Hd**:** integer **:=** 800**;** --Hpulse+HBP+Hactive+HFP

Va**:** integer **:=** 2**;** --Vpulse

Vb**:** integer **:=** 35**;** --Vpulse+VBP

Vc**:** integer **:=** 515**;** --Vpulse+VBP+Vactive

Vd**:** integer **:=** 525**);** --Vpulse+VBP+Vactive+VFP

**port** **(**

clk**:** **in** std\_logic**;** --50MHz in our board

rst**:** **in** std\_logic**;**

Hsync**,** Vsync**:** **buffer** std\_logic**;**

R**,** G**,** B**:** **out** std\_logic\_vector**(**3 **downto** 0**);**

p1lswitch**,** p1rswitch**,** p2lswitch**,** p2rswitch**:** **in** std\_logic\_vector**);**

**end** vga**;**

----------------------------------------------------------

**architecture** vga **of** vga **is**

**signal** Hactive**,** Vactive**,** dena**:** std\_logic**;**

**signal** pixel\_clk**:** std\_logic**;**

**signal** game\_clk**:** std\_logic**;**

**shared** **variable** p1\_buffer**:** memory\_t **:=** **(others** **=>** **(others** **=>** '0'**));**

**shared** **variable** p2\_buffer**:** memory\_t **:=** **(others** **=>** **(others** **=>** '0'**));**

**shared** **variable** p1d**:** direction\_t **:=** 'd'**;**

**shared** **variable** p2d**:** direction\_t **:=** 'u'**;**

**shared** **variable** p1lost**:** std\_logic **:=** '0'**;**

**shared** **variable** p2lost**:** std\_logic **:=** '0'**;**

**signal** p1l\_down**:** std\_logic **:=** '0'**;**

**signal** p1r\_down**:** std\_logic **:=** '0'**;**

**signal** p2l\_down**:** std\_logic **:=** '0'**;**

**signal** p2r\_down**:** std\_logic **:=** '0'**;**

**signal** paused**:** std\_logic **:=** '1'**;**

**begin**

--player 1 button presses

**process(**clk**)**

**begin**

**if** rst **=** '0' **then**

p1d **:=** 'D'**;**

paused **<=** '1'**;**

**elsif** **rising\_edge(**clk**)** **then**

**if** p1lswitch **=** '0' **then**

**if** p1l\_down **=** '0' **then**

**if** paused **=** '1' **then**

paused **<=** '0'**;**

**elsif(**p1d **=** 'U'**)** **then**

p1d **:=** 'L'**;**

**elsif(**p1d **=** 'D'**)** **then**

p1d **:=** 'R'**;**

**elsif(**p1d **=** 'L'**)** **then**

p1d **:=** 'D'**;**

**elsif(**p1d **=** 'R'**)** **then**

p1d **:=** 'U'**;**

**end** **if;**

p1l\_down **<=** '1'**;**

**end** **if;**

**else**

p1l\_down **<=** '0'**;**

**end** **if;**

**if** p1rswitch **=** '0' **then**

**if** p1r\_down **=** '0' **then**

**if(**p1d **=** 'U'**)** **then**

p1d **:=** 'R'**;**

**elsif(**p1d **=** 'D'**)** **then**

p1d **:=** 'L'**;**

**elsif(**p1d **=** 'L'**)** **then**

p1d **:=** 'U'**;**

**elsif(**p1d **=** 'R'**)** **then**

p1d **:=** 'D'**;**

**end** **if;**

p1r\_down **<=** '1'**;**

**end** **if;**

**else**

p1r\_down **<=** '0'**;**

**end** **if;**

**end** **if;**

**end** **process;**

--player 2 button presses

**process(**clk**)**

**begin**

**if** rst **=** '0' **then**

p2d **:=** 'U'**;**

**elsif** **rising\_edge(**clk**)** **then**

**if** p2lswitch **=** '0' **then**

**if** p2l\_down **=** '0' **then**

**if(**p2d **=** 'U'**)** **then**

p2d **:=** 'L'**;**

**elsif(**p2d **=** 'D'**)** **then**

p2d **:=** 'R'**;**

**elsif(**p2d **=** 'L'**)** **then**

p2d **:=** 'D'**;**

**elsif(**p2d **=** 'R'**)** **then**

p2d **:=** 'U'**;**

**end** **if;**

p2l\_down **<=** '1'**;**

**end** **if;**

**else**

p2l\_down **<=** '0'**;**

**end** **if;**

**if** p2rswitch **=** '0' **then**

**if** p2r\_down **=** '0' **then**

**if(**p2d **=** 'U'**)** **then**

p2d **:=** 'R'**;**

**elsif(**p2d **=** 'D'**)** **then**

p2d **:=** 'L'**;**

**elsif(**p2d **=** 'L'**)** **then**

p2d **:=** 'U'**;**

**elsif(**p2d **=** 'R'**)** **then**

p2d **:=** 'D'**;**

**end** **if;**

p2r\_down **<=** '1'**;**

**end** **if;**

**else**

p2r\_down **<=** '0'**;**

**end** **if;**

**end** **if;**

**end** **process;**

--game clock generation

**process(**clk**)**

**constant** COUNTDOWN\_MAX**:** integer **:=** **(**CLOCK\_FREQ **/** GAME\_FREQ**)** **/** 2**;**

**variable** countdown**:** integer **range** 0 **to** COUNTDOWN\_MAX **:=** COUNTDOWN\_MAX**;**

**begin**

**if** **rising\_edge(**clk**)** **then**

countdown **:=** countdown **-** 1**;**

**if** countdown **=** 0 **then**

countdown **:=** COUNTDOWN\_MAX**;**

game\_clk **<=** not game\_clk**;**

**end** **if;**

**end** **if;**

**end** **process;**

--player movement and collision

**process(**game\_clk**)**

**variable** p1x**:** integer **range** 0 **to** BOARD\_W **-** 1 **:=** 3**;**

**variable** p1y**:** integer **range** 0 **to** BOARD\_H **-** 1 **:=** 3**;**

**variable** p2x**:** integer **range** 0 **to** BOARD\_W **-** 1 **:=** BOARD\_W **-** 1 **-** 3**;**

**variable** p2y**:** integer **range** 0 **to** BOARD\_H **-** 1 **:=** BOARD\_H **-** 1 **-** 3**;**

**variable** row**:** integer **range** 0 **to** BOARD\_H **-** 1**;**

**variable** col**:** integer **range** 0 **to** BOARD\_W **-** 1**;**

**begin**

**if** rst **=** '0' **then**

p1x **:=** 3**;**

p1y **:=** 3**;**

p2x **:=** BOARD\_W **-** 1 **-** 3**;**

p2y **:=** BOARD\_H **-** 1 **-** 3**;**

p1lost **:=** '0'**;**

p2lost **:=** '0'**;**

p1\_buffer **:=** **(others** **=>** **(others** **=>** '0'**));**

p2\_buffer **:=** **(others** **=>** **(others** **=>** '0'**));**

**elsif(rising\_edge(**game\_clk**))** **then**

**if** paused **=** '0' and p1lost **=** '0' and p2lost **=** '0' **then**

--updating player 1's position

**if(**p1d **=** 'U'**)** **then**

p1y **:=** p1y **-** 1**;**

**elsif(**p1d **=** 'D'**)** **then**

p1y **:=** p1y **+** 1**;**

**elsif(**p1d **=** 'L'**)** **then**

p1x **:=** p1x **-** 1**;**

**elsif(**p1d **=** 'R'**)** **then**

p1x **:=** p1x **+** 1**;**

**end** **if;**

--updating player 2's position

**if(**p2d **=** 'U'**)** **then**

p2y **:=** p2y **-** 1**;**

**elsif(**p2d **=** 'D'**)** **then**

p2y **:=** p2y **+** 1**;**

**elsif(**p2d **=** 'L'**)** **then**

p2x **:=** p2x **-** 1**;**

**elsif(**p2d **=** 'R'**)** **then**

p2x **:=** p2x **+** 1**;**

**end** **if;**

--collision detection for player 1

--tile is already occupied

**if(**p1\_buffer**(**p1y**)(**p1x**)** **=** '1' or p2\_buffer**(**p1y**)(**p1x**)** **=** '1'**)** or p1y **<** 0 or p1y **>** BOARD\_H **-** 1 or p1x **<** 0 or p1x **>** BOARD\_W **-** 1 **then**

p1lost **:=** '1'**;**

**end** **if;**

--collision detection for player 2

--tile is already occupied

**if(**p1\_buffer**(**p2y**)(**p2x**)** **=** '1' or p2\_buffer**(**p2y**)(**p2x**)** **=** '1'**)** or p2y **<** 0 or p2y **>** BOARD\_H **-** 1 or p2x **<** 0 or p2x **>** BOARD\_W **-** 1 **then**

p2lost **:=** '1'**;**

**end** **if;**

**end** **if;**

--turn on pixel

p1\_buffer**(**p1y**)(**p1x**)** **:=** '1'**;**

p2\_buffer**(**p2y**)(**p2x**)** **:=** '1'**;**

**end** **if;**

**end** **process;**

-------------------------------------------------------

--Part 1: CONTROL GENERATOR

-------------------------------------------------------

--Create pixel clock (50MHz->25MHz):

**process** **(**clk**)**

**begin**

**if** **(**clk'**event** and clk**=**'1'**)** **then**

pixel\_clk **<=** not pixel\_clk**;**

**end** **if;**

**end** **process;**

--Horizontal signals generation:

**process** **(**pixel\_clk**)**

**variable** hcount**:** integer **range** 0 **to** Hd**;**

**begin**

**if** **(**pixel\_clk'**event** and pixel\_clk**=**'1'**)** **then**

Hcount **:=** Hcount **+** 1**;**

**if** **(**Hcount**=**Ha**)** **then**

Hsync **<=** '1'**;**

**elsif** **(**Hcount**=**Hb**)** **then**

Hactive **<=** '1'**;**

**elsif** **(**Hcount**=**Hc**)** **then**

Hactive **<=** '0'**;**

**elsif** **(**Hcount**=**Hd**)** **then**

Hsync **<=** '0'**;**

Hcount **:=** 0**;**

**end** **if;**

**end** **if;**

**end** **process;**

--Vertical signals generation:

**process** **(**Hsync**)**

**variable** Vcount**:** integer **range** 0 **to** Vd**;**

**begin**

**if** **(**Hsync'**event** and Hsync**=**'0'**)** **then**

Vcount **:=** Vcount **+** 1**;**

**if** **(**Vcount**=**Va**)** **then**

Vsync **<=** '1'**;**

**elsif** **(**Vcount**=**Vb**)** **then**

Vactive **<=** '1'**;**

**elsif** **(**Vcount**=**Vc**)** **then**

Vactive **<=** '0'**;**

**elsif** **(**Vcount**=**Vd**)** **then**

Vsync **<=** '0'**;**

Vcount **:=** 0**;**

**end** **if;**

**end** **if;**

**end** **process;**

---Display enable generation:

dena **<=** Hactive and Vactive**;**

-------------------------------------------------------

--Part 2: IMAGE GENERATOR

-------------------------------------------------------

**process** **(**Hsync**,** Vsync**,** Vactive**,** dena**,** pixel\_clk**)**

**variable** row**:** integer **range** **-**1 **to** Vc **-** 1**;**

**variable** col**:** integer **range** 0 **to** Hc**;**

**begin**

--row updating

**if** **(**Vsync**=**'0'**)** **then**

row **:=** **-**1**;**

**elsif** **(**Hsync'**event** and Hsync**=**'1'**)** **then**

**if** **(**Vactive**=**'1'**)** **then**

row **:=** row **+** 1**;**

**end** **if;**

**end** **if;**

--col updating

**if** **(**Hsync**=**'0'**)** **then**

col **:=** 0**;**

**elsif** **(**pixel\_clk'**event** and pixel\_clk**=**'1'**)** **then**

**if** **(**Hactive**=**'1'**)** **then**

col **:=** col **+** 1**;**

**end** **if;**

**end** **if;**

**if** **(**dena**=**'1'**)** **then**

**if** **(**p1lost **=** '1'**)** **then**

--p2's color

r **<=** **(others** **=>** '0'**);**

g **<=** **(others** **=>** '1'**);**

b **<=** **(others** **=>** '1'**);**

**elsif** **(**p2lost **=** '1'**)** **then**

--p1's color

r **<=** **(others** **=>** '1'**);**

g **<=** **(others** **=>** '0'**);**

b **<=** **(others** **=>** '1'**);**

**else** --no one has won/lost, display game state

**if** **(**p1\_buffer**(**row **/** BLOCK\_H**)(**col **/** BLOCK\_W**)** **=** '1'**)** **then**

--p1's color

r **<=** **(others** **=>** '1'**);**

g **<=** **(others** **=>** '0'**);**

b **<=** **(others** **=>** '1'**);**

**elsif** **(**p2\_buffer**(**row **/** BLOCK\_H**)(**col **/** BLOCK\_W**)** **=** '1'**)** **then**

--p2's color

r **<=** **(others** **=>** '0'**);**

g **<=** **(others** **=>** '1'**);**

b **<=** **(others** **=>** '1'**);**

**else**

--color if game pixel is off

r **<=** **(others** **=>** '0'**);**

g **<=** **(others** **=>** '0'**);**

b **<=** **(others** **=>** '0'**);**

**end** **if;**

**end** **if;**

**else**

r **<=** **(others** **=>** '0'**);**

g **<=** **(others** **=>** '0'**);**

b **<=** **(others** **=>** '0'**);**

**end** **if;**

**end** **process;**

**end** vga**;**

----------------------------------------------------------

linewars\_package.vhd

**library** ieee**;**

**use** ieee**.**std\_logic\_1164**.all;**

----------------------------------------------------------

**package** linewars\_package **is**

**constant** CLOCK\_FREQ**:** integer **:=** 50\_000\_000**;**

**constant** GAME\_FREQ**:** integer **:=** 5**;**

**constant** SCREEN\_W**:** integer **:=** 640**;**

**constant** SCREEN\_H**:** integer **:=** 480**;**

**constant** BLOCK\_W**:** integer **:=** 32**;**

**constant** BLOCK\_H**:** integer **:=** BLOCK\_W**;**

**constant** BOARD\_W**:** integer **:=** SCREEN\_W **/** BLOCK\_W**;**

**constant** BOARD\_H**:** integer **:=** SCREEN\_H **/** BLOCK\_H**;**

**subtype** word\_t **is** std\_logic\_vector**(**0 **to** BOARD\_W**-**1**);**

**type** memory\_t **is** **array(**0 **to** BOARD\_H**-**1**)** **of** word\_t**;**

**type** direction\_t **is** **(**'U'**,** 'D'**,** 'L'**,** 'R'**);**

**end package;**