|  |
| --- |
| library ieee; |
|  | use ieee.numeric\_std.all; |
|  | use ieee.std\_logic\_1164.all; |
|  | use work.tetris\_package.all; |
|  |  |
|  | entity Tetris\_Controller is |
|  | port |
|  | ( |
|  | CLOCK : in std\_logic; |
|  | RESET\_N : in std\_logic; |
|  | TIME\_10MS : in std\_logic; |
|  |  |
|  | BUTTON\_LEFT : in std\_logic; |
|  | BUTTON\_RIGHT : in std\_logic; |
|  | BUTTON\_DOWN : in std\_logic; |
|  | BUTTON\_ROTATE : in std\_logic; |
|  | -- Connections with Tetris\_Datapath |
|  | CAN\_MOVE\_LEFT : in std\_logic; |
|  | CAN\_MOVE\_RIGHT : in std\_logic; |
|  | CAN\_MOVE\_DOWN : in std\_logic; |
|  | CAN\_ROTATE : in std\_logic; |
|  | ROW\_IS\_COMPLETE : in std\_logic; |
|  | CLEAR : out std\_logic; |
|  | MOVE\_DOWN : out std\_logic; |
|  | MOVE\_LEFT : out std\_logic; |
|  | MOVE\_RIGHT : out std\_logic; |
|  | ROTATE : out std\_logic; |
|  | MERGE : out std\_logic; |
|  | REMOVE\_ROW : out std\_logic; |
|  | ROW\_INDEX : out integer range 0 to (BOARD\_ROWS-1); |
|  | NEW\_PIECE : out std\_logic; |
|  | NEW\_PIECE\_TYPE : out piece\_type; |
|  | -- Connections with View |
|  | REDRAW : out std\_logic |
|  | ); |
|  |  |
|  | end entity; |
|  |  |
|  |  |
|  | architecture RTL of Tetris\_Controller is |
|  | constant STANDARD\_FALL\_SPEED : integer := 50; |
|  | constant FAST\_FALL\_SPEED : integer := 10; |
|  | constant MOVEMENT\_SPEED : integer := 20; |
|  |  |
|  | signal fall\_speed : integer range 1 to 100; |
|  | signal time\_to\_next\_fall : integer range 0 to (fall\_speed'high-1); |
|  | signal move\_piece\_down : std\_logic; |
|  |  |
|  | signal time\_to\_next\_move : integer range 0 to MOVEMENT\_SPEED-1; |
|  | signal move\_time : std\_logic; |
|  |  |
|  | type row\_check\_state\_type is (IDLE, CHECKING\_ROW, WAIT\_ROW\_REMOVAL); |
|  | signal row\_check\_req : std\_logic; |
|  | signal row\_check\_ack : std\_logic; |
|  | signal row\_check\_counter : integer range 0 to BOARD\_ROWS-1; |
|  | signal row\_check\_state : row\_check\_state\_type; |
|  |  |
|  | signal random\_piece : piece\_type; |
|  | signal rnd\_count\_r : integer range 0 to 6; |
|  | begin |
|  |  |
|  | CLEAR <= '0'; |
|  | ROW\_INDEX <= row\_check\_counter; |
|  |  |
|  | fall\_speed <= FAST\_FALL\_SPEED when (BUTTON\_DOWN = '1') |
|  | else STANDARD\_FALL\_SPEED; |
|  |  |
|  |  |
|  | TimedFall : process(CLOCK, RESET\_N) |
|  | begin |
|  | if (RESET\_N = '0') then |
|  | time\_to\_next\_fall <= 0; |
|  | move\_piece\_down <= '0'; |
|  | elsif rising\_edge(CLOCK) then |
|  | move\_piece\_down <= '0'; |
|  |  |
|  | if (TIME\_10MS = '1') then |
|  | if (time\_to\_next\_fall = 0) then |
|  | time\_to\_next\_fall <= fall\_speed - 1; |
|  | move\_piece\_down <= '1'; |
|  | else |
|  | time\_to\_next\_fall <= time\_to\_next\_fall - 1; |
|  | end if; |
|  | end if; |
|  | end if; |
|  | end process; |
|  |  |
|  |  |
|  | TimedMove : process(CLOCK, RESET\_N) |
|  | begin |
|  | if (RESET\_N = '0') then |
|  | time\_to\_next\_move <= 0; |
|  | move\_time <= '0'; |
|  | elsif rising\_edge(CLOCK) then |
|  | move\_time <= '0'; |
|  |  |
|  | if (TIME\_10MS = '1') then |
|  | if (time\_to\_next\_move = 0) then |
|  | time\_to\_next\_move <= MOVEMENT\_SPEED - 1; |
|  | move\_time <= '1'; |
|  | else |
|  | time\_to\_next\_move <= time\_to\_next\_move - 1; |
|  | end if; |
|  | end if; |
|  | end if; |
|  | end process; |
|  |  |
|  |  |
|  | Controller\_RTL : process (CLOCK, RESET\_N) |
|  | begin |
|  | if (RESET\_N = '0') then |
|  | MERGE <= '0'; |
|  | NEW\_PIECE <= '0'; |
|  | MOVE\_DOWN <= '0'; |
|  | MOVE\_LEFT <= '0'; |
|  | MOVE\_RIGHT <= '0'; |
|  | ROTATE <= '0'; |
|  | REDRAW <= '0'; |
|  | row\_check\_req <= '0'; |
|  | elsif rising\_edge(CLOCK) then |
|  | MERGE <= '0'; |
|  | NEW\_PIECE <= '0'; |
|  | MOVE\_DOWN <= '0'; |
|  | MOVE\_LEFT <= '0'; |
|  | MOVE\_RIGHT <= '0'; |
|  | ROTATE <= '0'; |
|  | REDRAW <= '0'; |
|  | row\_check\_req <= '0'; |
|  |  |
|  | if (move\_piece\_down = '1') then |
|  | if (CAN\_MOVE\_DOWN = '1') then |
|  | MOVE\_DOWN <= '1'; |
|  | REDRAW <= '1'; |
|  | else |
|  | MERGE <= '1'; |
|  | NEW\_PIECE <= '1'; |
|  | NEW\_PIECE\_TYPE <= random\_piece; |
|  | row\_check\_req <= '1'; |
|  | end if; |
|  | elsif (move\_time = '1') then |
|  | if (BUTTON\_ROTATE = '1' and CAN\_ROTATE = '1') then |
|  | ROTATE <= '1'; |
|  | REDRAW <= '1'; |
|  | elsif (BUTTON\_LEFT = '1' and CAN\_MOVE\_LEFT = '1') then |
|  | MOVE\_LEFT <= '1'; |
|  | REDRAW <= '1'; |
|  | elsif (BUTTON\_RIGHT = '1' and CAN\_MOVE\_RIGHT = '1') then |
|  | MOVE\_RIGHT <= '1'; |
|  | REDRAW <= '1'; |
|  | end if; |
|  | end if; |
|  |  |
|  | if (row\_check\_ack = '1') then |
|  | REDRAW <= '1'; |
|  | end if; |
|  | end if; |
|  | end process; |
|  |  |
|  |  |
|  | Row\_check : process(CLOCK, RESET\_N) |
|  | begin |
|  |  |
|  | if (RESET\_N = '0') then |
|  | REMOVE\_ROW <= '0'; |
|  | row\_check\_state <= IDLE; |
|  | row\_check\_ack <= '0'; |
|  |  |
|  | elsif rising\_edge(CLOCK) then |
|  | REMOVE\_ROW <= '0'; |
|  | row\_check\_ack <= '0'; |
|  |  |
|  | case (row\_check\_state) is |
|  |  |
|  | when IDLE => |
|  | if (row\_check\_req = '1') then |
|  | row\_check\_state <= CHECKING\_ROW; |
|  | row\_check\_counter <= BOARD\_ROWS - 1; |
|  | end if; |
|  |  |
|  | when CHECKING\_ROW => |
|  | if (ROW\_IS\_COMPLETE = '1') then |
|  | REMOVE\_ROW <= '1'; |
|  | row\_check\_state <= WAIT\_ROW\_REMOVAL; |
|  | elsif (row\_check\_counter /= 0) then |
|  | row\_check\_counter <= row\_check\_counter - 1; |
|  | else |
|  | row\_check\_state <= IDLE; |
|  | row\_check\_ack <= '1'; |
|  | end if; |
|  |  |
|  | when WAIT\_ROW\_REMOVAL => |
|  | row\_check\_state <= CHECKING\_ROW; |
|  |  |
|  | end case; |
|  | end if; |
|  | end process; |
|  |  |
|  | rand\_shape : process(RESET\_N, CLOCK) |
|  | begin |
|  | if (RESET\_N = '0') then |
|  | rnd\_count\_r <= 0; |
|  | elsif (rising\_edge(CLOCK)) then |
|  | if(TIME\_10MS = '1' or BUTTON\_RIGHT = '1' or BUTTON\_LEFT = '1' or BUTTON\_ROTATE = '1') then |
|  | if(rnd\_count\_r /= rnd\_count\_r'high-1) then |
|  | rnd\_count\_r <= rnd\_count\_r + 1; |
|  | else |
|  | rnd\_count\_r <= 0; |
|  | end if; |
|  | end if; |
|  | end if; |
|  | end process; |
|  |  |
|  |  |
|  | with rnd\_count\_r select random\_piece <= |
|  | PIECE\_T when 0, |
|  | PIECE\_SQUARE when 1, |
|  | PIECE\_STICK when 2, |
|  | PIECE\_L when 3, |
|  | PIECE\_LR when 4, |
|  | PIECE\_DOG\_L when 5, |
|  | PIECE\_DOG\_R when 6; |
|  |  |
|  | end architecture; |

|  |
| --- |
| library ieee; |
|  | use ieee.numeric\_std.all; |
|  | use ieee.std\_logic\_1164.all; |
|  | use work.tetris\_package.all; |
|  | use work.vga\_package.all; |
|  |  |
|  |  |
|  | entity Tetris\_DE1 is |
|  |  |
|  | port |
|  | ( |
|  | CLOCK\_50 : in std\_logic; |
|  | KEY : in std\_logic\_vector(3 downto 0); |
|  |  |
|  | SW : in std\_logic\_vector(9 downto 9); |
|  | VGA\_R : out std\_logic\_vector(3 downto 0); |
|  | VGA\_G : out std\_logic\_vector(3 downto 0); |
|  | VGA\_B : out std\_logic\_vector(3 downto 0); |
|  | VGA\_HS : out std\_logic; |
|  | VGA\_VS : out std\_logic; |
|  |  |
|  | SRAM\_ADDR : out std\_logic\_vector(17 downto 0); |
|  | SRAM\_DQ : inout std\_logic\_vector(15 downto 0); |
|  | SRAM\_CE\_N : out std\_logic; |
|  | SRAM\_OE\_N : out std\_logic; |
|  | SRAM\_WE\_N : out std\_logic; |
|  | SRAM\_UB\_N : out std\_logic; |
|  | SRAM\_LB\_N : out std\_logic |
|  | ); |
|  |  |
|  | end; |
|  |  |
|  |  |
|  | architecture RTL of Tetris\_DE1 is |
|  | signal clock : std\_logic; |
|  | signal clock\_vga : std\_logic; |
|  | signal RESET\_N : std\_logic; |
|  | signal fb\_ready : std\_logic; |
|  | signal fb\_clear : std\_logic; |
|  | signal fb\_flip : std\_logic; |
|  | signal fb\_draw\_rect : std\_logic; |
|  | signal fb\_fill\_rect : std\_logic; |
|  | signal fb\_draw\_line : std\_logic; |
|  | signal fb\_x0 : xy\_coord\_type; |
|  | signal fb\_y0 : xy\_coord\_type; |
|  | signal fb\_x1 : xy\_coord\_type; |
|  | signal fb\_y1 : xy\_coord\_type; |
|  | signal fb\_color : color\_type; |
|  | signal time\_10ms : std\_logic; |
|  | signal redraw : std\_logic; |
|  | signal rotate : std\_logic; |
|  | signal can\_move\_left : std\_logic; |
|  | signal can\_move\_right : std\_logic; |
|  | signal can\_move\_down : std\_logic; |
|  | signal can\_rotate : std\_logic; |
|  | signal row\_index : integer range 0 to (BOARD\_ROWS-1); |
|  | signal row\_is\_complete : std\_logic; |
|  | signal clear : std\_logic; |
|  | signal move\_left : std\_logic; |
|  | signal move\_right : std\_logic; |
|  | signal move\_down : std\_logic; |
|  | signal merge : std\_logic; |
|  | signal remove\_row : std\_logic; |
|  | signal new\_piece : std\_logic; |
|  | signal new\_piece\_type : piece\_type; |
|  | signal query\_cell : block\_pos\_type; |
|  | signal query\_cell\_content : board\_cell\_type; |
|  | signal reset\_sync\_reg : std\_logic; |
|  | begin |
|  |  |
|  | pll : entity work.PLL |
|  | port map ( |
|  | inclk0 => CLOCK\_50, |
|  | c0 => clock\_vga, |
|  | c1 => clock |
|  | ); |
|  |  |
|  |  |
|  | reset\_sync : process(CLOCK\_50) |
|  | begin |
|  | if (rising\_edge(CLOCK\_50)) then |
|  | reset\_sync\_reg <= SW(9); |
|  | RESET\_N <= reset\_sync\_reg; |
|  | end if; |
|  | end process; |
|  |  |
|  |  |
|  | vga : entity work.VGA\_Framebuffer |
|  | port map ( |
|  | CLOCK => clock\_vga, |
|  | RESET\_N => RESET\_N, |
|  | READY => fb\_ready, |
|  | COLOR => fb\_color, |
|  | CLEAR => fb\_clear, |
|  | DRAW\_RECT => fb\_draw\_rect, |
|  | FILL\_RECT => fb\_fill\_rect, |
|  | DRAW\_LINE => fb\_draw\_line, |
|  | FLIP => fb\_flip, |
|  | X0 => fb\_x0, |
|  | Y0 => fb\_y0, |
|  | X1 => fb\_x1, |
|  | Y1 => fb\_y1, |
|  |  |
|  | VGA\_R => VGA\_R, |
|  | VGA\_G => VGA\_G, |
|  | VGA\_B => VGA\_B, |
|  | VGA\_HS => VGA\_HS, |
|  | VGA\_VS => VGA\_VS, |
|  |  |
|  | SRAM\_ADDR => SRAM\_ADDR, |
|  | SRAM\_DQ => SRAM\_DQ, |
|  | SRAM\_CE\_N => SRAM\_CE\_N, |
|  | SRAM\_OE\_N => SRAM\_OE\_N, |
|  | SRAM\_WE\_N => SRAM\_WE\_N, |
|  | SRAM\_UB\_N => SRAM\_UB\_N, |
|  | SRAM\_LB\_N => SRAM\_LB\_N |
|  | ); |
|  |  |
|  |  |
|  | controller : entity work.Tetris\_Controller |
|  | port map ( |
|  | CLOCK => clock, |
|  | RESET\_N => RESET\_N, |
|  | TIME\_10MS => time\_10ms, |
|  | BUTTON\_LEFT => not(KEY(3)), |
|  | BUTTON\_RIGHT => not(KEY(2)), |
|  | BUTTON\_DOWN => not(KEY(1)), |
|  | BUTTON\_ROTATE => not(KEY(0)), |
|  | CAN\_MOVE\_LEFT => can\_move\_left, |
|  | CAN\_MOVE\_RIGHT => can\_move\_right, |
|  | CAN\_MOVE\_DOWN => can\_move\_down, |
|  | CAN\_ROTATE => can\_rotate, |
|  | ROW\_INDEX => row\_index, |
|  | ROW\_IS\_COMPLETE => row\_is\_complete, |
|  | CLEAR => clear, |
|  | ROTATE => rotate, |
|  | MOVE\_DOWN => move\_down, |
|  | MOVE\_LEFT => move\_left, |
|  | MOVE\_RIGHT => move\_right, |
|  | MERGE => merge, |
|  | NEW\_PIECE => new\_piece, |
|  | REMOVE\_ROW => remove\_row, |
|  | NEW\_PIECE\_TYPE => new\_piece\_type, |
|  | REDRAW => redraw |
|  | ); |
|  |  |
|  |  |
|  | datapath : entity work.Tetris\_Datapath |
|  | port map ( |
|  | CLOCK => clock, |
|  | RESET\_N => RESET\_N, |
|  | CLEAR => clear, |
|  | MOVE\_DOWN => move\_down, |
|  | MOVE\_LEFT => move\_left, |
|  | MOVE\_RIGHT => move\_right, |
|  | ROTATE => rotate, |
|  | MERGE => merge, |
|  | NEW\_PIECE => new\_piece, |
|  | REMOVE\_ROW => remove\_row, |
|  | NEW\_PIECE\_TYPE => new\_piece\_type, |
|  | CAN\_MOVE\_LEFT => can\_move\_left, |
|  | CAN\_MOVE\_RIGHT => can\_move\_right, |
|  | CAN\_MOVE\_DOWN => can\_move\_down, |
|  | CAN\_ROTATE => can\_rotate, |
|  | ROW\_INDEX => row\_index, |
|  | ROW\_IS\_COMPLETE => row\_is\_complete, |
|  | QUERY\_CELL => query\_cell, |
|  | CELL\_CONTENT => query\_cell\_content |
|  | ); |
|  |  |
|  | view : entity work.Tetris\_View |
|  | port map ( |
|  | CLOCK => clock, |
|  | RESET\_N => RESET\_N, |
|  | REDRAW => redraw, |
|  | FB\_READY => fb\_ready, |
|  | FB\_CLEAR => fb\_clear, |
|  | FB\_DRAW\_RECT => fb\_draw\_rect, |
|  | FB\_DRAW\_LINE => fb\_draw\_line, |
|  | FB\_FILL\_RECT => fb\_fill\_rect, |
|  | FB\_FLIP => fb\_flip, |
|  | FB\_COLOR => fb\_color, |
|  | FB\_X0 => fb\_x0, |
|  | FB\_Y0 => fb\_y0, |
|  | FB\_X1 => fb\_x1, |
|  | FB\_Y1 => fb\_y1, |
|  | QUERY\_CELL => query\_cell, |
|  | CELL\_CONTENT => query\_cell\_content |
|  | ); |
|  |  |
|  | timegen : process(CLOCK, RESET\_N) |
|  | variable counter : integer range 0 to (500000-1); |
|  | begin |
|  | if (RESET\_N = '0') then |
|  | counter := 0; |
|  | time\_10ms <= '0'; |
|  | elsif (rising\_edge(clock)) then |
|  | if(counter = counter'high) then |
|  | counter := 0; |
|  | time\_10ms <= '1'; |
|  | else |
|  | counter := counter+1; |
|  | time\_10ms <= '0'; |
|  | end if; |
|  | end if; |
|  | end process; |
|  |  |
|  | end architecture; |

|  |
| --- |
| library ieee; |
|  | use ieee.numeric\_std.all; |
|  | use ieee.std\_logic\_1164.all; |
|  | use work.tetris\_package.all; |
|  | --use work.vga\_package.all; |
|  |  |
|  |  |
|  | entity Tetris\_Datapath is |
|  | port |
|  | ( |
|  | CLOCK : in std\_logic; |
|  | RESET\_N : in std\_logic; |
|  |  |
|  | -- Connections for the Controller |
|  | CLEAR : in std\_logic; |
|  | MOVE\_DOWN : in std\_logic; |
|  | MOVE\_LEFT : in std\_logic; |
|  | MOVE\_RIGHT : in std\_logic; |
|  | ROTATE : in std\_logic; |
|  | MERGE : in std\_logic; |
|  | REMOVE\_ROW : in std\_logic; |
|  | NEW\_PIECE : in std\_logic; |
|  | NEW\_PIECE\_TYPE : in piece\_type; |
|  | ROW\_INDEX : in integer range 0 to (BOARD\_ROWS-1); |
|  | CAN\_MOVE\_LEFT : out std\_logic; |
|  | CAN\_MOVE\_RIGHT : out std\_logic; |
|  | CAN\_MOVE\_DOWN : out std\_logic; |
|  | CAN\_ROTATE : out std\_logic; |
|  | ROW\_IS\_COMPLETE : out std\_logic; |
|  | -- Connections for the View |
|  | QUERY\_CELL : in block\_pos\_type; |
|  | CELL\_CONTENT : out board\_cell\_type |
|  | ); |
|  |  |
|  | end entity; |
|  |  |
|  |  |
|  | architecture RTL of Tetris\_Datapath is |
|  | signal board : board\_type; |
|  | signal falling\_piece : piece\_type; |
|  | signal next\_falling\_piece : piece\_type; |
|  | type affected\_by\_merge\_type is array(natural range <>, natural range <>) of std\_logic; |
|  | signal cell\_affected\_by\_merge : affected\_by\_merge\_type(0 to BOARD\_COLUMNS-1, 0 to BOARD\_ROWS-1); |
|  |  |
|  | begin |
|  |  |
|  | FallingPiece\_RTL : process(CLOCK, RESET\_N) |
|  | constant PIECE\_AT\_RESET : piece\_type := PIECE\_SQUARE; |
|  | constant NEW\_PIECE\_OFFSET : integer := BOARD\_COLUMNS/2 - 2; |
|  | begin |
|  | if (RESET\_N = '0') then |
|  |  |
|  | falling\_piece <= PIECE\_AT\_RESET; |
|  |  |
|  | elsif (rising\_edge(CLOCK)) then |
|  |  |
|  | if (NEW\_PIECE = '1') then |
|  |  |
|  | falling\_piece.shape <= NEW\_PIECE\_TYPE.shape; |
|  | for i in 0 to BLOCKS\_PER\_PIECE-1 loop |
|  | falling\_piece.blocks(i).row <= NEW\_PIECE\_TYPE.blocks(i).row; |
|  | falling\_piece.blocks(i).col <= NEW\_PIECE\_TYPE.blocks(i).col + NEW\_PIECE\_OFFSET; |
|  | end loop; |
|  |  |
|  | else |
|  |  |
|  | falling\_piece <= next\_falling\_piece; |
|  |  |
|  | end if; |
|  | end if; |
|  | end process; |
|  |  |
|  |  |
|  | NextFallingPiece : process(falling\_piece, MOVE\_DOWN, MOVE\_LEFT, MOVE\_RIGHT, ROTATE) |
|  | variable pivot : block\_pos\_type; |
|  | begin |
|  | next\_falling\_piece <= falling\_piece; |
|  | pivot := falling\_piece.blocks(0); |
|  |  |
|  | for i in 0 to BLOCKS\_PER\_PIECE-1 loop |
|  | if (MOVE\_DOWN = '1') then |
|  | next\_falling\_piece.blocks(i).row <= falling\_piece.blocks(i).row + 1; |
|  |  |
|  | elsif (ROTATE = '1') then |
|  | if(i /= 0) then -- the pivot does not require any transformation |
|  | next\_falling\_piece.blocks(i).col <= |
|  | pivot.col - (falling\_piece.blocks(i).row - pivot.row); |
|  |  |
|  | next\_falling\_piece.blocks(i).row <= |
|  | pivot.row + (falling\_piece.blocks(i).col - pivot.col); |
|  | end if; |
|  | elsif (MOVE\_LEFT = '1') then |
|  | next\_falling\_piece.blocks(i).col <= falling\_piece.blocks(i).col - 1; |
|  |  |
|  | elsif (MOVE\_RIGHT = '1') then |
|  | next\_falling\_piece.blocks(i).col <= falling\_piece.blocks(i).col + 1; |
|  | end if; |
|  | end loop; |
|  | end process; |
|  |  |
|  |  |
|  | CanMove\_Signals : process(falling\_piece, board) |
|  | variable cur\_block : block\_pos\_type; |
|  | variable left\_cell\_filled : std\_logic; |
|  | variable right\_cell\_filled : std\_logic; |
|  | variable bottom\_cell\_filled : std\_logic; |
|  | begin |
|  | CAN\_MOVE\_LEFT <= '1'; |
|  | CAN\_MOVE\_RIGHT <= '1'; |
|  | CAN\_MOVE\_DOWN <= '1'; |
|  | CAN\_ROTATE <= '1'; --TODO rotation detection |
|  |  |
|  | for i in 0 to BLOCKS\_PER\_PIECE-1 loop |
|  | cur\_block := falling\_piece.blocks(i); |
|  |  |
|  | if (cur\_block.col = 0) then |
|  | CAN\_MOVE\_LEFT <= '0'; |
|  | else |
|  | left\_cell\_filled := board.cells((cur\_block.col-1),cur\_block.row).filled; |
|  | if (left\_cell\_filled = '1') then |
|  | CAN\_MOVE\_LEFT <= '0'; |
|  | end if; |
|  | end if; |
|  |  |
|  | if (cur\_block.col = (BOARD\_COLUMNS-1)) then |
|  | CAN\_MOVE\_RIGHT <= '0'; |
|  | else |
|  | right\_cell\_filled := board.cells((cur\_block.col+1),cur\_block.row).filled; |
|  | if (right\_cell\_filled = '1') then |
|  | CAN\_MOVE\_RIGHT <= '0'; |
|  | end if; |
|  | end if; |
|  |  |
|  | if (cur\_block.row = (BOARD\_ROWS-1)) then |
|  | CAN\_MOVE\_DOWN <= '0'; |
|  | else |
|  | bottom\_cell\_filled := board.cells(cur\_block.col,(cur\_block.row+1)).filled; |
|  | if (bottom\_cell\_filled = '1') then |
|  | CAN\_MOVE\_DOWN <= '0'; |
|  | end if; |
|  | end if; |
|  | end loop; |
|  | end process; |
|  |  |
|  |  |
|  | Board\_rtl : process(CLOCK, RESET\_N) |
|  | begin |
|  | if (RESET\_N = '0') then |
|  |  |
|  | for col in 0 to BOARD\_COLUMNS-1 loop |
|  | for row in 0 to BOARD\_ROWS-1 loop |
|  | board.cells(col,row).filled <= '0'; |
|  | end loop; |
|  | end loop; |
|  |  |
|  | elsif (rising\_edge(CLOCK)) then |
|  |  |
|  | for col in 0 to BOARD\_COLUMNS-1 loop |
|  | for row in 0 to BOARD\_ROWS-1 loop |
|  |  |
|  | if (CLEAR = '1') then |
|  | board.cells(col,row).filled <= '0'; |
|  |  |
|  | elsif (REMOVE\_ROW = '1') then |
|  | if (row = 0) then |
|  | board.cells(col, row).filled <= '0'; |
|  | elsif (row <= ROW\_INDEX) then |
|  | board.cells(col, row) <= board.cells(col, row-1); |
|  | end if; |
|  |  |
|  | elsif (MERGE = '1') then |
|  | if(cell\_affected\_by\_merge(col, row) = '1') then |
|  | board.cells(col, row).filled <= '1'; |
|  | board.cells(col, row).shape <= falling\_piece.shape; |
|  | end if; |
|  |  |
|  | end if; |
|  |  |
|  | end loop; |
|  | end loop; |
|  |  |
|  | end if; |
|  | end process; |
|  |  |
|  |  |
|  | AffectedByMerge : process(board, falling\_piece) |
|  | begin |
|  | cell\_affected\_by\_merge <= ((others=> (others=>'0'))); |
|  |  |
|  | for i in 0 to BLOCKS\_PER\_PIECE-1 loop |
|  | cell\_affected\_by\_merge( |
|  | falling\_piece.blocks(i).col, |
|  | falling\_piece.blocks(i).row |
|  | ) <= '1'; |
|  | end loop; |
|  |  |
|  | end process; |
|  |  |
|  |  |
|  | RowCheck : process(board, ROW\_INDEX) |
|  | begin |
|  | ROW\_IS\_COMPLETE <= '1'; |
|  | for i in 0 to (BOARD\_COLUMNS-1) loop |
|  | if(board.cells(i,ROW\_INDEX).filled = '0') then |
|  | ROW\_IS\_COMPLETE <= '0'; |
|  | end if; |
|  | end loop; |
|  | end process; |
|  |  |
|  |  |
|  | CellQuery : process(QUERY\_CELL, board, falling\_piece) |
|  | variable selected\_cell : board\_cell\_type; |
|  | begin |
|  | CELL\_CONTENT.filled <= '0'; |
|  | CELL\_CONTENT.shape <= SHAPE\_T; --indifferent |
|  |  |
|  | selected\_cell := board.cells(QUERY\_CELL.col, QUERY\_CELL.row); |
|  | -- At first attempt output the selected board cell |
|  | CELL\_CONTENT <= selected\_cell; |
|  |  |
|  | -- Override the output if one of the blocks of |
|  | -- the falling\_piece occupy the selected cell |
|  | for i in 0 to BLOCKS\_PER\_PIECE-1 loop |
|  | if(falling\_piece.blocks(i) = QUERY\_CELL) then |
|  | CELL\_CONTENT.filled <= '1'; |
|  | CELL\_CONTENT.shape <= falling\_piece.shape; |
|  | end if; |
|  | end loop; |
|  | end process; |
|  |  |
|  | end architecture; |

|  |
| --- |
| library ieee; |
|  | use ieee.numeric\_std.all; |
|  | use ieee.std\_logic\_1164.all; |
|  | use work.tetris\_package.all; |
|  | use work.vga\_package.all; |
|  |  |
|  |  |
|  | entity Tetris\_View is |
|  | port |
|  | ( |
|  | CLOCK : in std\_logic; |
|  | RESET\_N : in std\_logic; |
|  |  |
|  | REDRAW : in std\_logic; |
|  |  |
|  | FB\_READY : in std\_logic; |
|  | FB\_CLEAR : out std\_logic; |
|  | FB\_DRAW\_RECT : out std\_logic; |
|  | FB\_DRAW\_LINE : out std\_logic; |
|  | FB\_FILL\_RECT : out std\_logic; |
|  | FB\_FLIP : out std\_logic; |
|  | FB\_COLOR : out color\_type; |
|  | FB\_X0 : out xy\_coord\_type; |
|  | FB\_Y0 : out xy\_coord\_type; |
|  | FB\_X1 : out xy\_coord\_type; |
|  | FB\_Y1 : out xy\_coord\_type; |
|  |  |
|  | QUERY\_CELL : out block\_pos\_type; |
|  | CELL\_CONTENT : in board\_cell\_type |
|  |  |
|  | ); |
|  | end entity; |
|  |  |
|  |  |
|  | architecture RTL of Tetris\_View is |
|  | constant LEFT\_MARGIN : integer := 8; |
|  | constant TOP\_MARGIN : integer := 8; |
|  | constant BLOCK\_SIZE : integer := 20; |
|  | constant BLOCK\_SPACING : integer := 1; |
|  |  |
|  | type state\_type is (IDLE, WAIT\_FOR\_READY, DRAWING); |
|  | type substate\_type is (CLEAR\_SCENE, DRAW\_BOARD\_OUTLINE, DRAW\_BOARD\_BLOCKS, FLIP\_FRAMEBUFFER); |
|  | signal state : state\_type; |
|  | signal substate : substate\_type; |
|  | signal query\_cell\_r : block\_pos\_type; |
|  |  |
|  | begin |
|  |  |
|  | QUERY\_CELL <= query\_cell\_r; |
|  |  |
|  | process(CLOCK, RESET\_N) |
|  | begin |
|  |  |
|  | if (RESET\_N = '0') then |
|  | state <= IDLE; |
|  | substate <= CLEAR\_SCENE; |
|  | FB\_CLEAR <= '0'; |
|  | FB\_DRAW\_RECT <= '0'; |
|  | FB\_DRAW\_LINE <= '0'; |
|  | FB\_FILL\_RECT <= '0'; |
|  | FB\_FLIP <= '0'; |
|  | query\_cell\_r.col <= 0; |
|  | query\_cell\_r.row <= 0; |
|  |  |
|  | elsif (rising\_edge(CLOCK)) then |
|  |  |
|  | FB\_CLEAR <= '0'; |
|  | FB\_DRAW\_RECT <= '0'; |
|  | FB\_DRAW\_LINE <= '0'; |
|  | FB\_FILL\_RECT <= '0'; |
|  | FB\_FLIP <= '0'; |
|  |  |
|  | case (state) is |
|  | when IDLE => |
|  | if (REDRAW = '1') then |
|  | state <= WAIT\_FOR\_READY; |
|  | substate <= CLEAR\_SCENE; |
|  | end if; |
|  |  |
|  | when WAIT\_FOR\_READY => |
|  | if (FB\_READY = '1') then |
|  | state <= DRAWING; |
|  | end if; |
|  |  |
|  | when DRAWING => |
|  | state <= WAIT\_FOR\_READY; |
|  |  |
|  | case (substate) is |
|  | when CLEAR\_SCENE => |
|  | FB\_COLOR <= COLOR\_BLACK; |
|  | FB\_CLEAR <= '1'; |
|  | substate <= DRAW\_BOARD\_OUTLINE; |
|  |  |
|  | when DRAW\_BOARD\_OUTLINE => |
|  | FB\_COLOR <= COLOR\_RED; |
|  | FB\_X0 <= LEFT\_MARGIN; |
|  | FB\_Y0 <= TOP\_MARGIN; |
|  | FB\_X1 <= LEFT\_MARGIN + (BOARD\_COLUMNS \* BLOCK\_SIZE); |
|  | FB\_Y1 <= TOP\_MARGIN + (BOARD\_ROWS \* BLOCK\_SIZE); |
|  | FB\_DRAW\_RECT <= '1'; |
|  | substate <= DRAW\_BOARD\_BLOCKS; |
|  |  |
|  | when DRAW\_BOARD\_BLOCKS => |
|  | if(CELL\_CONTENT.filled = '1') then |
|  | FB\_COLOR <= Lookup\_color(CELL\_CONTENT.shape); |
|  | FB\_X0 <= LEFT\_MARGIN + (query\_cell\_r.col \* BLOCK\_SIZE) + BLOCK\_SPACING; |
|  | FB\_Y0 <= TOP\_MARGIN + (query\_cell\_r.row \* BLOCK\_SIZE) + BLOCK\_SPACING; |
|  | FB\_X1 <= LEFT\_MARGIN + (query\_cell\_r.col \* BLOCK\_SIZE) + BLOCK\_SIZE - BLOCK\_SPACING; |
|  | FB\_Y1 <= TOP\_MARGIN + (query\_cell\_r.row \* BLOCK\_SIZE) + BLOCK\_SIZE - BLOCK\_SPACING; |
|  | FB\_FILL\_RECT <= '1'; |
|  | end if; |
|  |  |
|  | if (query\_cell\_r.col /= BOARD\_COLUMNS-1) then |
|  | query\_cell\_r.col <= query\_cell\_r.col + 1; |
|  | else |
|  | query\_cell\_r .col <= 0; |
|  | if (query\_cell\_r.row /= BOARD\_ROWS-1) then |
|  | query\_cell\_r.row <= query\_cell\_r.row + 1; |
|  | else |
|  | query\_cell\_r.row <= 0; |
|  | substate <= FLIP\_FRAMEBUFFER; |
|  | end if; |
|  | end if; |
|  |  |
|  | when FLIP\_FRAMEBUFFER => |
|  | FB\_FLIP <= '1'; |
|  | state <= IDLE; |
|  |  |
|  | end case; |
|  | end case; |
|  |  |
|  | end if; |
|  | end process; |
|  |  |
|  | end architecture; |

|  |
| --- |
| library ieee; |
|  | use ieee.std\_logic\_1164.all; |
|  | use ieee.numeric\_std.all; |
|  | use work.vga\_package.all; |
|  |  |
|  | -- TODO aggiungere bound sulle coordinate |
|  |  |
|  | entity VGA\_Framebuffer is |
|  | generic |
|  | ( |
|  | SCREEN\_WIDTH : positive := 512; |
|  | SCREEN\_HEIGHT : positive := 480; |
|  | COLOR\_DEPTH : positive := 12 |
|  | ); |
|  | port |
|  | ( |
|  | CLOCK : in std\_logic; |
|  | RESET\_N : in std\_logic; |
|  |  |
|  | -- Commands |
|  | CLEAR : in std\_logic; |
|  | DRAW\_RECT : in std\_logic; |
|  | DRAW\_LINE : in std\_logic; |
|  | FILL\_RECT : in std\_logic; |
|  | FLIP : in std\_logic; |
|  |  |
|  |  |
|  | -- Commands parameters |
|  | COLOR : in color\_type; |
|  | X0 : in xy\_coord\_type; |
|  | Y0 : in xy\_coord\_type; |
|  | X1 : in xy\_coord\_type; |
|  | Y1 : in xy\_coord\_type; |
|  |  |
|  | -- Outputs |
|  | READY : out std\_logic; |
|  | VGA\_R : out std\_logic\_vector(3 downto 0); |
|  | VGA\_G : out std\_logic\_vector(3 downto 0); |
|  | VGA\_B : out std\_logic\_vector(3 downto 0); |
|  | VGA\_HS : out std\_logic; |
|  | VGA\_VS : out std\_logic; |
|  |  |
|  | SRAM\_ADDR : out std\_logic\_vector(17 downto 0); |
|  | SRAM\_DQ : inout std\_logic\_vector(15 downto 0); |
|  | SRAM\_CE\_N : out std\_logic; |
|  | SRAM\_OE\_N : out std\_logic; |
|  | SRAM\_WE\_N : out std\_logic; |
|  | SRAM\_UB\_N : out std\_logic; |
|  | SRAM\_LB\_N : out std\_logic |
|  | ); |
|  |  |
|  | end; |
|  |  |
|  |  |
|  | architecture RTL of VGA\_Framebuffer is |
|  | type state\_type is (IDLE, DRAWING\_RECT, FILLING\_RECT, DRAWING\_LINE); |
|  | type substate\_type is (INIT, DRAWING, DRAWING\_R1, DRAWING\_R2); |
|  | signal state : state\_type; |
|  | signal substate : substate\_type; |
|  |  |
|  | signal vga\_blank : std\_logic; |
|  | signal vga\_strobe : std\_logic; |
|  | signal vga\_x : std\_logic\_vector(10 downto 0); |
|  | signal vga\_y : std\_logic\_vector(10 downto 0); |
|  | signal vga\_vsync : std\_logic; |
|  |  |
|  | signal fb\_buffer\_idx : std\_logic; |
|  | signal fb\_wr\_req : std\_logic; |
|  | signal fb\_wr\_ack : std\_logic; |
|  | signal fb\_wr\_x : std\_logic\_vector(10 downto 0); |
|  | signal fb\_wr\_y : std\_logic\_vector(10 downto 0); |
|  | signal fb\_wr\_color : std\_logic\_vector(COLOR\_DEPTH-1 downto 0); |
|  |  |
|  | signal fb\_rd\_req : std\_logic; |
|  | signal fb\_rd\_ack : std\_logic; |
|  | signal fb\_rd\_x : std\_logic\_vector(10 downto 0); |
|  | signal fb\_rd\_y : std\_logic\_vector(10 downto 0); |
|  | signal fb\_rd\_color : std\_logic\_vector(COLOR\_DEPTH-1 downto 0); |
|  |  |
|  | signal flip\_on\_next\_vs : std\_logic; |
|  | signal latched\_color : std\_logic\_vector(COLOR\_DEPTH-1 downto 0); |
|  | signal x\_cursor : integer range 0 to SCREEN\_WIDTH; |
|  | signal y\_cursor : integer range 0 to SCREEN\_HEIGHT; |
|  | signal x\_start : integer range 0 to SCREEN\_WIDTH; |
|  | signal y\_start : integer range 0 to SCREEN\_HEIGHT; |
|  | signal x\_end : integer range 0 to SCREEN\_WIDTH; |
|  | signal y\_end : integer range 0 to SCREEN\_HEIGHT; |
|  |  |
|  | begin |
|  |  |
|  | vga\_timing : entity work.VGA\_Timing |
|  | port map( |
|  | CLOCK => CLOCK, |
|  | RESET\_N => RESET\_N, |
|  | H\_SYNC => VGA\_HS, |
|  | V\_SYNC => vga\_vsync, |
|  | BLANK => vga\_blank, |
|  | PIXEL\_STROBE => vga\_strobe, |
|  | PIXEL\_X => vga\_x, |
|  | PIXEL\_Y => vga\_y |
|  | ); |
|  |  |
|  | vga\_fb : entity work.VGA\_RAMDAC |
|  | port map( |
|  | CLOCK => CLOCK, |
|  | RESET\_N => RESET\_N, |
|  | BUFFER\_INDEX => fb\_buffer\_idx, |
|  | WR\_REQ => fb\_wr\_req, |
|  | WR\_ACK => fb\_wr\_ack, |
|  | WR\_X => fb\_wr\_x, |
|  | WR\_Y => fb\_wr\_y, |
|  | WR\_COLOR => fb\_wr\_color, |
|  |  |
|  | RD\_REQ => fb\_rd\_req, |
|  | RD\_ACK => fb\_rd\_ack, |
|  | RD\_X => fb\_rd\_x, |
|  | RD\_Y => fb\_rd\_y, |
|  | RD\_COLOR => fb\_rd\_color, |
|  |  |
|  | SRAM\_ADDR => SRAM\_ADDR, |
|  | SRAM\_DQ => SRAM\_DQ, |
|  | SRAM\_CE\_N => SRAM\_CE\_N, |
|  | SRAM\_OE\_N => SRAM\_OE\_N, |
|  | SRAM\_WE\_N => SRAM\_WE\_N, |
|  | SRAM\_UB\_N => SRAM\_UB\_N, |
|  | SRAM\_LB\_N => SRAM\_LB\_N |
|  | ); |
|  |  |
|  |  |
|  |  |
|  | fb\_rd\_x <= vga\_x; |
|  | fb\_rd\_y <= vga\_y; |
|  | --fb\_rd\_req <= vga\_strobe; --TODO here |
|  |  |
|  | -- only to avoid synthesis warnings; |
|  | fb\_rd\_req <= not(vga\_blank) and (vga\_strobe or fb\_rd\_ack or '1'); |
|  |  |
|  |  |
|  | fb\_wr\_color <= latched\_color; |
|  | fb\_wr\_x <= std\_logic\_vector(to\_unsigned(x\_cursor, fb\_wr\_x'length)); |
|  | fb\_wr\_y <= std\_logic\_vector(to\_unsigned(y\_cursor, fb\_wr\_y'length)); |
|  |  |
|  | VGA\_VS <= vga\_vsync; |
|  | VGA\_R <= fb\_rd\_color(11 downto 8) when (vga\_blank = '0') else (others => '0'); |
|  | VGA\_G <= fb\_rd\_color(7 downto 4) when (vga\_blank = '0') else (others => '0'); |
|  | VGA\_B <= fb\_rd\_color(3 downto 0) when (vga\_blank = '0') else (others => '0'); |
|  | READY <= '1' when (state = IDLE and (CLEAR or DRAW\_LINE or DRAW\_RECT or FILL\_RECT or FLIP) = '0') else '0'; |
|  |  |
|  | draw\_logic : process(CLOCK, RESET\_N) |
|  | begin |
|  |  |
|  | if (RESET\_N = '0') then |
|  |  |
|  | state <= IDLE; |
|  | fb\_wr\_req <= '0'; |
|  | fb\_buffer\_idx <= '0'; |
|  | flip\_on\_next\_vs <= '0'; |
|  |  |
|  | elsif (rising\_edge(CLOCK)) then |
|  |  |
|  | fb\_wr\_req <= '0'; |
|  |  |
|  | case (state) is |
|  | when IDLE => |
|  |  |
|  | latched\_color <= COLOR; |
|  | if (CLEAR = '1') then |
|  | x\_start <= 0; |
|  | y\_start <= 0; |
|  | x\_end <= SCREEN\_WIDTH-1; |
|  | y\_end <= SCREEN\_HEIGHT-1; |
|  | state <= FILLING\_RECT; |
|  | substate <= INIT; |
|  |  |
|  | elsif (DRAW\_LINE = '1') then |
|  | x\_start <= X0; |
|  | y\_start <= Y0; |
|  | x\_end <= X1; |
|  | y\_end <= Y1; |
|  | state <= DRAWING\_LINE; |
|  | substate <= INIT; |
|  |  |
|  | elsif (DRAW\_RECT = '1') then |
|  | x\_start <= X0; |
|  | y\_start <= Y0; |
|  | x\_end <= X1; |
|  | y\_end <= Y1; |
|  | state <= DRAWING\_RECT; |
|  | substate <= INIT; |
|  |  |
|  | elsif (FILL\_RECT = '1') then |
|  | x\_start <= X0; |
|  | y\_start <= Y0; |
|  | x\_end <= X1; |
|  | y\_end <= Y1; |
|  | state <= FILLING\_RECT; |
|  | substate <= INIT; |
|  |  |
|  | elsif (FLIP = '1') then |
|  | flip\_on\_next\_vs <= '1'; |
|  |  |
|  | end if; |
|  |  |
|  | if (flip\_on\_next\_vs = '1' and vga\_vsync = '0') then |
|  | fb\_buffer\_idx <= not(fb\_buffer\_idx); |
|  | flip\_on\_next\_vs <= '0'; |
|  | end if; |
|  |  |
|  |  |
|  | when DRAWING\_RECT => |
|  | fb\_wr\_req <= '1'; |
|  | if (substate = INIT) then |
|  | x\_cursor <= x\_start; |
|  | y\_cursor <= y\_start; |
|  | substate <= DRAWING\_R1; |
|  | elsif (substate = DRAWING\_R1) then |
|  | if (fb\_wr\_ack = '1') then |
|  | if (x\_cursor = x\_end) then |
|  | if (y\_cursor = y\_end) then |
|  | fb\_wr\_req <= '0'; |
|  | substate <= DRAWING\_R2; |
|  | else |
|  | y\_cursor <= y\_cursor + 1; |
|  | end if; |
|  | else |
|  | x\_cursor <= x\_cursor + 1; |
|  | end if; |
|  | end if; |
|  | elsif (substate = DRAWING\_R2) then |
|  | if (fb\_wr\_ack = '1') then |
|  | if (x\_cursor = x\_start) then |
|  | if (y\_cursor = y\_start) then |
|  | fb\_wr\_req <= '0'; |
|  | state <= IDLE; |
|  | else |
|  | y\_cursor <= y\_cursor - 1; |
|  | end if; |
|  | else |
|  | x\_cursor <= x\_cursor - 1; |
|  | end if; |
|  | end if; |
|  | end if; |
|  |  |
|  |  |
|  | when DRAWING\_LINE => |
|  | fb\_wr\_req <= '1'; |
|  | if (substate = INIT) then |
|  | x\_cursor <= x\_start; |
|  | y\_cursor <= y\_start; |
|  | substate <= DRAWING; |
|  | else |
|  | if (fb\_wr\_ack = '1') then |
|  | if (x\_cursor = x\_end) then |
|  | if (y\_cursor = y\_end) then |
|  | fb\_wr\_req <= '0'; |
|  | state <= IDLE; |
|  | else |
|  | y\_cursor <= y\_cursor + 1; |
|  | end if; |
|  | else |
|  | x\_cursor <= x\_cursor + 1; |
|  | end if; |
|  | end if; |
|  | end if; |
|  |  |
|  |  |
|  | when FILLING\_RECT => |
|  | fb\_wr\_req <= '1'; |
|  |  |
|  | if (substate = INIT) then |
|  | x\_cursor <= x\_start; |
|  | y\_cursor <= y\_start; |
|  | substate <= DRAWING; |
|  | else |
|  | if (fb\_wr\_ack = '1') then |
|  | if (x\_cursor = x\_end) then |
|  | x\_cursor <= x\_start; |
|  | if (y\_cursor = y\_end) then |
|  | fb\_wr\_req <= '0'; |
|  | state <= IDLE; |
|  | else |
|  | y\_cursor <= y\_cursor + 1; |
|  | end if; |
|  | else |
|  | x\_cursor <= x\_cursor + 1; |
|  | end if; |
|  | end if; |
|  | end if; |
|  |  |
|  |  |
|  | when others => |
|  | assert false severity failure; |
|  |  |
|  | end case; |
|  |  |
|  |  |
|  | end if; |
|  |  |
|  | end process; |
|  |  |
|  |  |
|  | end architecture; |

|  |
| --- |
| library ieee; |
|  | use ieee.std\_logic\_1164.all; |
|  | use ieee.numeric\_std.all; |
|  |  |
|  | entity VGA\_RAMDAC is |
|  | generic |
|  | ( |
|  | FB\_WIDTH : natural := 512; |
|  | FB\_HEIGHT : natural := 480; |
|  | FB\_DEPTH : natural := 12 |
|  | ); |
|  |  |
|  | port |
|  | ( |
|  | CLOCK : in std\_logic; |
|  | RESET\_N : in std\_logic; |
|  |  |
|  | BUFFER\_INDEX : in std\_logic; |
|  | WR\_X : in std\_logic\_vector(10 downto 0); |
|  | WR\_Y : in std\_logic\_vector(10 downto 0); |
|  | WR\_COLOR : in std\_logic\_vector(FB\_DEPTH-1 downto 0); |
|  | WR\_REQ : in std\_logic; |
|  | WR\_ACK : out std\_logic; |
|  |  |
|  | RD\_X : in std\_logic\_vector(10 downto 0); |
|  | RD\_Y : in std\_logic\_vector(10 downto 0); |
|  | RD\_COLOR : out std\_logic\_vector(FB\_DEPTH-1 downto 0); |
|  | RD\_REQ : in std\_logic; |
|  | RD\_ACK : out std\_logic; |
|  |  |
|  | SRAM\_ADDR : out std\_logic\_vector(17 downto 0); |
|  | SRAM\_DQ : inout std\_logic\_vector(15 downto 0); |
|  | SRAM\_CE\_N : out std\_logic; |
|  | SRAM\_OE\_N : out std\_logic; |
|  | SRAM\_WE\_N : out std\_logic; |
|  | SRAM\_UB\_N : out std\_logic; |
|  | SRAM\_LB\_N : out std\_logic |
|  | ); |
|  |  |
|  | end; |
|  |  |
|  |  |
|  | architecture RTL of VGA\_RAMDAC is |
|  |  |
|  | signal wr\_addr : std\_logic\_vector(SRAM\_ADDR'range); |
|  | signal rd\_addr : std\_logic\_vector(SRAM\_ADDR'range); |
|  | signal rd\_buf\_idx : std\_logic; |
|  | signal wr\_buf\_idx : std\_logic; |
|  | signal encoded\_pixel : std\_logic\_vector(7 downto 0); |
|  | signal mem\_dir\_wr : std\_logic; |
|  | signal mem\_dir\_rd : std\_logic; |
|  | signal latch\_ram\_rd : std\_logic; |
|  | signal ram\_rd\_word : std\_logic\_vector(7 downto 0); |
|  | signal latched\_ram : std\_logic\_vector(7 downto 0); |
|  |  |
|  | type ram\_state\_type is (IDLE, READING, WRITING, RW\_COMPLETED); |
|  | signal ram\_state : ram\_state\_type; |
|  | signal next\_ram\_state : ram\_state\_type; |
|  |  |
|  | function coords\_to\_addr |
|  | ( |
|  | x : std\_logic\_vector; |
|  | y : std\_logic\_vector |
|  | ) |
|  | return std\_logic\_vector is |
|  | begin |
|  | return y(8 downto 0) & x(8 downto 0); |
|  | end function; |
|  |  |
|  |  |
|  | function encode\_pixel(rgb : std\_logic\_vector(FB\_DEPTH-1 downto 0)) |
|  | return std\_logic\_vector |
|  | is |
|  | constant BPC : natural := FB\_DEPTH/3; |
|  | variable red : std\_logic\_vector(BPC-1 downto 0); |
|  | variable green : std\_logic\_vector(BPC-1 downto 0); |
|  | variable blue : std\_logic\_vector(BPC-1 downto 0); |
|  | begin |
|  | blue := rgb(BPC-1 downto 0); |
|  | green := rgb(BPC\*2-1 downto BPC); |
|  | red := rgb(BPC\*3-1 downto BPC\*2); |
|  |  |
|  | return red(red'high downto red'high-2) |
|  | & green(green'high downto green'high-1) |
|  | & blue(blue'high downto blue'high-2); |
|  | end function; |
|  |  |
|  |  |
|  | function decode\_pixel(pixel : std\_logic\_vector(7 downto 0)) |
|  | return std\_logic\_vector |
|  | is |
|  | constant BPC : natural := FB\_DEPTH/3; |
|  | variable red : std\_logic\_vector(BPC-1 downto 0); |
|  | variable green : std\_logic\_vector(BPC-1 downto 0); |
|  | variable blue : std\_logic\_vector(BPC-1 downto 0); |
|  | begin |
|  |  |
|  | red := (others => pixel(5)); |
|  | red(red'high downto red'high-2) := pixel(7 downto 5); |
|  |  |
|  | green := (others => pixel(3)); |
|  | green(green'high downto green'high-1) := pixel(4 downto 3); |
|  |  |
|  | blue := (others => pixel(0)); |
|  | blue(blue'high downto blue'high-2) := pixel(2 downto 0); |
|  |  |
|  | return red & green & blue; |
|  | end function; |
|  |  |
|  |  |
|  | begin |
|  |  |
|  | rd\_buf\_idx <= BUFFER\_INDEX; |
|  | wr\_buf\_idx <= not(BUFFER\_INDEX); |
|  | wr\_addr <= coords\_to\_addr(WR\_X,WR\_Y); |
|  | rd\_addr <= coords\_to\_addr(RD\_X,RD\_Y); |
|  | encoded\_pixel <= encode\_pixel(WR\_COLOR); |
|  | --RD\_COLOR <= decode\_pixel(latched\_ram) when latch\_ram\_rd = '0' else decode\_pixel(ram\_rd\_word); |
|  | RD\_COLOR <= decode\_pixel(latched\_ram); --TODO here |
|  |  |
|  |  |
|  |  |
|  |  |
|  | ram\_fsm : process(RD\_REQ, WR\_REQ, rd\_addr, wr\_addr, encoded\_pixel, ram\_state, CLOCK) |
|  | begin |
|  | mem\_dir\_wr <= '0'; |
|  | mem\_dir\_rd <= '0'; |
|  | RD\_ACK <= '0'; |
|  | WR\_ACK <= '0'; |
|  | SRAM\_OE\_N <= '1'; |
|  | SRAM\_WE\_N <= '1'; |
|  | latch\_ram\_rd <= '0'; |
|  | next\_ram\_state <= ram\_state; |
|  |  |
|  | case (ram\_state) is |
|  |  |
|  | when IDLE => --TODO remove state |
|  |  |
|  | if (RD\_REQ = '1') then |
|  | mem\_dir\_rd <= '1'; |
|  | SRAM\_OE\_N <= '0'; |
|  | latch\_ram\_rd <= '1'; |
|  | RD\_ACK <= '1'; |
|  |  |
|  | elsif (WR\_REQ = '1') then |
|  | mem\_dir\_wr <= '1'; |
|  | SRAM\_WE\_N <= '0'; |
|  | WR\_ACK <= '1'; |
|  |  |
|  | end if; |
|  |  |
|  | when others => |
|  | assert false severity failure; |
|  |  |
|  | end case; |
|  |  |
|  | end process; |
|  |  |
|  |  |
|  | mem\_dir\_ctrl : process(mem\_dir\_rd, mem\_dir\_wr, rd\_addr, SRAM\_DQ, wr\_addr, rd\_buf\_idx, wr\_buf\_idx, encoded\_pixel) |
|  | begin |
|  | SRAM\_CE\_N <= '0'; -- We don't care of power saving, RAM is always enabled! |
|  | SRAM\_ADDR <= (others => '-'); |
|  | ram\_rd\_word <= (others => '-'); |
|  | SRAM\_DQ <= (others => 'Z'); |
|  | SRAM\_LB\_N <= '1'; |
|  | SRAM\_UB\_N <= '1'; |
|  |  |
|  | if (mem\_dir\_rd = '1') then |
|  | SRAM\_ADDR <= rd\_addr(SRAM\_ADDR'range); |
|  | if (rd\_buf\_idx = '0') then |
|  | ram\_rd\_word <= SRAM\_DQ(7 downto 0); |
|  | SRAM\_LB\_N <= '0'; |
|  | else |
|  | ram\_rd\_word <= SRAM\_DQ(15 downto 8); |
|  | SRAM\_UB\_N <= '0'; |
|  | end if; |
|  |  |
|  | elsif (mem\_dir\_wr = '1') then |
|  | SRAM\_ADDR <= wr\_addr(SRAM\_ADDR'range); |
|  | SRAM\_LB\_N <= wr\_buf\_idx; |
|  | SRAM\_UB\_N <= not(wr\_buf\_idx); |
|  | SRAM\_DQ <= encoded\_pixel & encoded\_pixel; |
|  | end if; |
|  |  |
|  | end process; |
|  |  |
|  |  |
|  | ram\_regs : process(CLOCK, RESET\_N) |
|  | begin |
|  | if (RESET\_N = '0') then |
|  |  |
|  | latched\_ram <= (others => '0'); |
|  | ram\_state <= IDLE; |
|  |  |
|  | elsif (rising\_edge(CLOCK)) then |
|  |  |
|  | if (latch\_ram\_rd = '1') then |
|  | latched\_ram <= ram\_rd\_word; |
|  | end if; |
|  |  |
|  | ram\_state <= next\_ram\_state; |
|  |  |
|  | end if; |
|  | end process; |
|  |  |
|  |  |
|  | end architecture; |

|  |
| --- |
| library ieee; |
|  | use ieee.std\_logic\_1164.all; |
|  | use ieee.numeric\_std.all; |
|  |  |
|  | entity VGA\_Timing is |
|  |  |
|  | generic |
|  | ( |
|  | CLOCK\_DIV : natural := 4; |
|  | H\_DISP : natural := 640-128; |
|  | H\_FRONT\_PORCH : natural := 16+64; |
|  | H\_SYNC\_LEN : natural := 96; |
|  | H\_BACK\_PORCH : natural := 48+64; |
|  | V\_DISP : natural := 480; |
|  | V\_FRONT\_PORCH : natural := 11; |
|  | V\_SYNC\_LEN : natural := 2; |
|  | V\_BACK\_PORCH : natural := 34 |
|  | ); |
|  |  |
|  | port |
|  | ( |
|  | CLOCK : in std\_logic; |
|  | RESET\_N : in std\_logic; |
|  | H\_SYNC : out std\_logic; |
|  | V\_SYNC : out std\_logic; |
|  | BLANK : out std\_logic; |
|  | PIXEL\_STROBE : out std\_logic; |
|  | PIXEL\_X : out std\_logic\_vector(10 downto 0); |
|  | PIXEL\_Y : out std\_logic\_vector(10 downto 0) |
|  | ); |
|  | end; |
|  |  |
|  | architecture RTL of VGA\_Timing is |
|  | constant H\_LENGTH : natural := H\_DISP + H\_FRONT\_PORCH + H\_SYNC\_LEN + H\_BACK\_PORCH; |
|  | constant V\_LENGTH : natural := V\_DISP + V\_FRONT\_PORCH + V\_SYNC\_LEN + V\_BACK\_PORCH; |
|  |  |
|  | signal clock\_count : integer range 0 to CLOCK\_DIV; |
|  |  |
|  | type state\_type is (FRONT\_PORCH, SYNC, BACK\_PORCH, DATA); |
|  | signal h\_state : state\_type; |
|  | signal h\_counter : integer range 0 to H\_LENGTH; |
|  | signal h\_pixel : integer range 0 to H\_DISP; |
|  |  |
|  | signal v\_state : state\_type; |
|  | signal v\_counter : integer range 0 to H\_LENGTH; |
|  | signal v\_pixel : integer range 0 to V\_DISP; |
|  |  |
|  | signal new\_line : std\_logic; |
|  |  |
|  | begin |
|  |  |
|  | h\_timing : process(CLOCK, RESET\_N) |
|  | begin |
|  |  |
|  | if (RESET\_N = '0') then |
|  | h\_counter <= 0; |
|  | h\_pixel <= 0; |
|  | h\_state <= FRONT\_PORCH; |
|  | new\_line <= '0'; |
|  | clock\_count <= 0; |
|  | elsif (rising\_edge(CLOCK)) then |
|  | new\_line <= '0'; |
|  |  |
|  | if (clock\_count /= CLOCK\_DIV-1) then |
|  | clock\_count <= clock\_count + 1; |
|  | else |
|  | clock\_count <= 0; |
|  |  |
|  | if (h\_counter = 0) then |
|  | h\_state <= FRONT\_PORCH; |
|  | new\_line <= '1'; --TODO really correct here?-- |
|  | elsif (h\_counter = (H\_FRONT\_PORCH - 1)) then |
|  | h\_state <= SYNC; |
|  | elsif (h\_counter = (H\_FRONT\_PORCH + H\_SYNC\_LEN - 1)) then |
|  | h\_state <= BACK\_PORCH; |
|  | elsif (h\_counter = (H\_FRONT\_PORCH + H\_SYNC\_LEN + H\_BACK\_PORCH - 1)) then |
|  | h\_state <= DATA; |
|  | end if; |
|  |  |
|  | if(h\_state = DATA and h\_counter /= 0) then |
|  | h\_pixel <= h\_pixel + 1; |
|  | else |
|  | h\_pixel <= 0; |
|  | end if; |
|  |  |
|  | if (h\_counter = H\_LENGTH-1) then |
|  | h\_counter <= 0; |
|  | else |
|  | h\_counter <= h\_counter + 1; |
|  | end if; |
|  | end if; |
|  | end if; |
|  |  |
|  | end process; |
|  |  |
|  |  |
|  | v\_timing : process(CLOCK, RESET\_N) |
|  | begin |
|  |  |
|  | if (RESET\_N = '0') then |
|  | v\_counter <= 0; |
|  | v\_pixel <= 0; |
|  | v\_state <= FRONT\_PORCH; |
|  | elsif (rising\_edge(CLOCK)) then |
|  |  |
|  | if(new\_line = '1') then |
|  | if (v\_counter = 0) then |
|  | v\_state <= FRONT\_PORCH; |
|  | elsif (v\_counter = (V\_FRONT\_PORCH - 1)) then |
|  | v\_state <= SYNC; |
|  | elsif (v\_counter = (V\_FRONT\_PORCH + V\_SYNC\_LEN - 1)) then |
|  | v\_state <= BACK\_PORCH; |
|  | elsif (v\_counter = (V\_FRONT\_PORCH + V\_SYNC\_LEN + V\_BACK\_PORCH - 1)) then |
|  | v\_state <= DATA; |
|  | end if; |
|  |  |
|  | if(v\_state = DATA and v\_counter /= 0) then |
|  | v\_pixel <= v\_pixel + 1; |
|  | else |
|  | v\_pixel <= 0; |
|  | end if; |
|  |  |
|  | if (v\_counter = V\_LENGTH-1) then |
|  | v\_counter <= 0; |
|  | else |
|  | v\_counter <= v\_counter + 1; |
|  | end if; |
|  | end if; |
|  | end if; |
|  |  |
|  | end process; |
|  |  |
|  | BLANK <= '0' when (h\_state = DATA and v\_state = DATA) else '1'; |
|  | H\_SYNC <= '0' when (h\_state = SYNC) else '1'; |
|  | V\_SYNC <= '0' when (v\_state = SYNC) else '1'; |
|  | PIXEL\_X <= std\_logic\_vector(to\_unsigned(h\_pixel, PIXEL\_X'LENGTH)); |
|  | PIXEL\_Y <= std\_logic\_vector(to\_unsigned(v\_pixel, PIXEL\_Y'LENGTH)); |
|  | PIXEL\_STROBE <= '1' when (h\_state = DATA and v\_state = DATA and clock\_count = 0) else '0'; |
|  | --<= '1' when ((h\_state = BACK\_PORCH or h\_state = DATA) and v\_state = DATA) else '0'; |
|  |  |
|  |  |
|  | end architecture; |

|  |
| --- |
|  |
| library ieee; |
|  | use ieee.numeric\_std.all; |
|  | use ieee.std\_logic\_1164.all; |
|  | use work.vga\_package.all; |
|  |  |
|  | package tetris\_package is |
|  | constant BOARD\_COLUMNS : positive := 10; |
|  | constant BOARD\_ROWS : positive := 20; |
|  | constant BLOCKS\_PER\_PIECE : positive := 4; |
|  |  |
|  | type shape\_type is (SHAPE\_T, SHAPE\_SQUARE, SHAPE\_STICK, SHAPE\_L\_L, SHAPE\_L\_R, SHAPE\_DOG\_L, SHAPE\_DOG\_R); |
|  | attribute enum\_encoding : string; |
|  | attribute enum\_encoding of shape\_type : type is "one-hot"; |
|  |  |
|  | -- Board declarations |
|  | type board\_cell\_type is record |
|  | filled : std\_logic; |
|  | shape : shape\_type; |
|  | end record; |
|  |  |
|  | type board\_cell\_array is array(natural range <>, natural range <>) of board\_cell\_type; |
|  |  |
|  | type board\_type is record |
|  | cells : board\_cell\_array(0 to (BOARD\_COLUMNS-1), 0 to (BOARD\_ROWS-1)); |
|  | end record; |
|  |  |
|  | -- Piece declarations |
|  | type block\_pos\_type is record |
|  | col : integer range 0 to (BOARD\_COLUMNS-1); |
|  | row : integer range 0 to (BOARD\_ROWS-1); |
|  | end record; |
|  |  |
|  | type block\_pos\_array is array(natural range <>) of block\_pos\_type; |
|  |  |
|  | type piece\_type is record |
|  | shape : shape\_type; |
|  | blocks : block\_pos\_array(0 to (BLOCKS\_PER\_PIECE-1)); |
|  | end record; |
|  |  |
|  | -- Piece definitions |
|  | constant PIECE\_T : piece\_type := |
|  | ( |
|  | shape => SHAPE\_T, |
|  | blocks => |
|  | ( |
|  | (col => 1, row => 0), |
|  | (col => 0, row => 0), |
|  | (col => 2, row => 0), |
|  | (col => 1, row => 1) |
|  | ) |
|  | ); |
|  |  |
|  | constant PIECE\_SQUARE : piece\_type := |
|  | ( |
|  | shape => SHAPE\_SQUARE, |
|  | blocks => |
|  | ( |
|  | (col => 0, row => 0), |
|  | (col => 1, row => 0), |
|  | (col => 0, row => 1), |
|  | (col => 1, row => 1) |
|  | ) |
|  | ); |
|  |  |
|  | constant PIECE\_STICK : piece\_type := |
|  | ( |
|  | shape => SHAPE\_STICK, |
|  | blocks => |
|  | ( |
|  | (col => 0, row => 1), |
|  | (col => 0, row => 0), |
|  | (col => 0, row => 2), |
|  | (col => 0, row => 3) |
|  | ) |
|  | ); |
|  |  |
|  | constant PIECE\_L : piece\_type := |
|  | ( |
|  | shape => SHAPE\_L\_L, |
|  | blocks => |
|  | ( |
|  | (col => 1, row => 0), |
|  | (col => 0, row => 0), |
|  | (col => 2, row => 0), |
|  | (col => 0, row => 1) |
|  | ) |
|  | ); |
|  |  |
|  | constant PIECE\_LR : piece\_type := |
|  | ( |
|  | shape => SHAPE\_L\_R, |
|  | blocks => |
|  | ( |
|  | (col => 1, row => 0), |
|  | (col => 0, row => 0), |
|  | (col => 2, row => 0), |
|  | (col => 2, row => 1) |
|  | ) |
|  | ); |
|  |  |
|  | constant PIECE\_DOG\_L : piece\_type := |
|  | ( |
|  | shape => SHAPE\_DOG\_L, |
|  | blocks => |
|  | ( |
|  | (col => 1, row => 0), |
|  | (col => 1, row => 1), |
|  | (col => 2, row => 0), |
|  | (col => 0, row => 1) |
|  | ) |
|  | ); |
|  |  |
|  | constant PIECE\_DOG\_R : piece\_type := |
|  | ( |
|  | shape => SHAPE\_DOG\_R, |
|  | blocks => |
|  | ( |
|  | (col => 1, row => 0), |
|  | (col => 1, row => 1), |
|  | (col => 0, row => 0), |
|  | (col => 2, row => 1) |
|  | ) |
|  | ); |
|  |  |
|  | function Lookup\_color(shape : shape\_type) return color\_type; |
|  | end package; |
|  |  |
|  |  |
|  | package body tetris\_package is |
|  |  |
|  | function Lookup\_color(shape : shape\_type) |
|  | return color\_type is |
|  | variable color : color\_type; |
|  | begin |
|  | case (shape) is |
|  | when SHAPE\_T => |
|  | color := COLOR\_YELLOW; |
|  | when SHAPE\_SQUARE => |
|  | color := COLOR\_MAGENTA; |
|  | when SHAPE\_STICK => |
|  | color := COLOR\_ORANGE; |
|  | when SHAPE\_L\_L | SHAPE\_L\_R => |
|  | color := COLOR\_CYAN; |
|  | when SHAPE\_DOG\_L | SHAPE\_DOG\_R => |
|  | color := COLOR\_GREEN; |
|  | end case; |
|  | return color; |
|  | end function; |
|  |  |
|  | end package body; |
|  |  |

|  |
| --- |
| library ieee; |
|  | use ieee.numeric\_std.all; |
|  | use ieee.std\_logic\_1164.all; |
|  |  |
|  | package vga\_package is |
|  | subtype color\_type is std\_logic\_vector(11 downto 0); |
|  | subtype xy\_coord\_type is integer range 0 to 512; |
|  | constant COLOR\_BLACK : color\_type := X"000"; |
|  | constant COLOR\_WHITE : color\_type := X"FFF"; |
|  | constant COLOR\_RED : color\_type := X"F00"; |
|  | constant COLOR\_ORANGE : color\_type := X"F80"; |
|  | constant COLOR\_GREEN : color\_type := X"0F0"; |
|  | constant COLOR\_BLUE : color\_type := X"00F"; |
|  | constant COLOR\_YELLOW : color\_type := X"FF0"; |
|  | constant COLOR\_CYAN : color\_type := X"0FF"; |
|  | constant COLOR\_MAGENTA : color\_type := X"F0F"; |
|  |  |
|  |  |
|  |  |
|  | end package; |