`timescale 1ns / 1ns // `timescale time\_unit/time\_precision  
  
////////////////  
// TOP MODULE //  
////////////////  
  
module project  
 (  
 SW,  
 KEY,  
 LEDR, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5,  
 CLOCK\_50, // On Board 50 MHz  
 // Your inputs and outputs here   
 // The ports below are for the VGA output. Do not change.  
 VGA\_CLK, // VGA Clock  
 VGA\_HS, // VGA H\_SYNC  
 VGA\_VS, // VGA V\_SYNC  
 VGA\_BLANK\_N, // VGA BLANK  
 VGA\_SYNC\_N, // VGA SYNC  
 VGA\_R, // VGA Red[9:0]  
 VGA\_G, // VGA Green[9:0]  
 VGA\_B // VGA Blue[9:0]  
 );  
   
 input [9:0] SW;  
 input [3:0] KEY;  
   
 output [9:0] LEDR;  
 output [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;  
  
 input CLOCK\_50; // 50 MHz  
 // Declare your inputs and outputs here  
 // Do not change the following outputs  
 output VGA\_CLK; // VGA Clock  
 output VGA\_HS; // VGA H\_SYNC  
 output VGA\_VS; // VGA V\_SYNC  
 output VGA\_BLANK\_N; // VGA BLANK  
 output VGA\_SYNC\_N; // VGA SYNC  
 output [9:0] VGA\_R; // VGA Red[9:0]  
 output [9:0] VGA\_G; // VGA Green[9:0]  
 output [9:0] VGA\_B; // VGA Blue[9:0]  
   
 wire resetn;  
 assign resetn = ~SW[9];  
   
 // Create the colour, x, y and writeEn wires that are inputs to the controller.  
  
 wire [2:0] colour;  
 wire [7:0] x;  
 wire [6:0] y;  
 wire writeEn;  
  
 // Create an Instance of a VGA controller - there can be only one!  
 // Define the number of colours as well as the initial background  
 // image file (.MIF) for the controller.  
 vga\_adapter VGA(  
 .resetn(resetn),  
 .clock(CLOCK\_50),  
 .colour(colour),  
 .x(x),  
 .y(y),  
 .plot(writeEn),  
 /\* Signals for the DAC to drive the monitor. \*/  
 .VGA\_R(VGA\_R),  
 .VGA\_G(VGA\_G),  
 .VGA\_B(VGA\_B),  
 .VGA\_HS(VGA\_HS),  
 .VGA\_VS(VGA\_VS),  
 .VGA\_BLANK(VGA\_BLANK\_N),  
 .VGA\_SYNC(VGA\_SYNC\_N),  
 .VGA\_CLK(VGA\_CLK));  
 defparam VGA.RESOLUTION = "160x120";  
 defparam VGA.MONOCHROME = "FALSE";  
 defparam VGA.BITS\_PER\_COLOUR\_CHANNEL = 1;  
 defparam VGA.BACKGROUND\_IMAGE = "black.mif";  
   
 // Put your code here. Your code should produce signals x,y,colour and writeEn  
 // for the VGA controller, in addition to any other functionality your design may require.  
  
 /\*  
 posX, posY : coords  
 col : colour  
 opX, opY : selection alu operations  
 selX, selY, selCol : select what input connects to data path's X reg, Y reg, colour reg  
 ldX, ldY, ldCol : enable X reg, Y reg, colour reg to load input  
 cx, cy : cursor position  
 \*/  
   
 //Into data path  
 reg [7:0] posX;  
 reg [6:0] posY;  
 reg [2:0] col, opX, opY;  
 reg [1:0] selX, selY, plot;  
 reg selCol, ldX, ldY, ldCol;  
   
 //Out of main control  
 wire [7:0] posX1;  
 wire [6:0] posY1;  
 wire [2:0] col1, opX1, opY1;  
 wire [1:0] selX1, selY1, plot1;  
 wire selCol1, ldX1, ldY1, ldCol1;  
   
 //Out of letter decoder control  
 wire [7:0] posX2;  
 wire [6:0] posY2;  
 wire [2:0] col2, opX2, opY2;  
 wire [1:0] selX2, selY2, plot2;  
 wire selCol2, ldX2, ldY2, ldCol2;  
   
 wire [2:0] cx, cy;  
 wire selCtrl;  
   
 wire [5:0] cur;  
 hex\_decoder h0(x[3:0], HEX0);  
 hex\_decoder h1(x[7:4], HEX1);  
 hex\_decoder h2(y[3:0], HEX2);  
 hex\_decoder h3({1'b0, y[6:4]}, HEX3);  
 hex\_decoder h4(cur[3:0], HEX4);  
 hex\_decoder h5({2'b00,cur[5:4]}, HEX5);  
 assign LEDR[6:4] = cy;  
 assign LEDR[3:1] = cx;  
   
 always @ (\*) begin  
 if (selCtrl == 1) begin  
 posX <= posX2;  
 posY <= posY2;  
 col <= col2;  
 opX <= opX2;  
 opY <= opY2;  
 selX <= selX2;  
 selY <= selY2;  
 selCol <= selCol2;  
 ldX <= ldX2;  
 ldY <= ldY2;  
 ldCol <= ldCol2;  
 plot <= plot2;  
 end  
 else begin  
 posX <= posX1;  
 posY <= posY1;  
 col <= col1;  
 opX <= opX1;  
 opY <= opY1;  
 selX <= selX1;  
 selY <= selY1;  
 selCol <= selCol1;  
 ldX <= ldX1;  
 ldY <= ldY1;  
 ldCol <= ldCol1;  
 plot <= plot1;  
 end  
 end  
   
 control c0(  
 SW[0], ~KEY[0], ~KEY[1], ~KEY[2], ~KEY[3], resetn, CLOCK\_50, posX1, posY1,  
 col1, opX1, opY1, selX1, selY1, selCol1,  
 ldX1, ldY1, ldCol1, plot1, cx, cy, selCtrl, cur, SW[6:1], LEDR[9:8], LEDR[7]  
 );  
  
 data d0(  
 posX, posY, col, opX, opY, selX, selY, selCol,  
 ldX, ldY, ldCol, plot, cx, cy, resetn, CLOCK\_50,  
 x, y, colour, writeEn  
 );  
   
 letter\_decoder l0(  
 resetn, CLOCK\_50, posX2, posY2,  
 col2, opX2, opY2, selX2, selY2, selCol2,  
 ldX2, ldY2, ldCol2, plot2  
 );  
  
endmodule  
  
  
  
//////////////////  
// CONTROL PATH //  
//////////////////  
  
module control(  
 toggle\_keys, key0, key1, key2, key3, resetn, clock, posX, posY,  
 col, opX, opY, selX, selY, selCol,  
 ldX, ldY, ldCol, plot, cxOut, cyOut, selCtrl, outState, switches, outBoard, outTurn  
 );  
  
 input toggle\_keys, key0, key1, key2, key3, resetn, clock;  
  
 output reg [7:0] posX;  
 output reg [6:0] posY;  
 output reg [2:0] col, opX, opY;  
 output reg [1:0] selX, selY, plot;  
 output reg selCol, ldX, ldY, ldCol, selCtrl;  
   
 output reg [2:0] cxOut, cyOut;  
   
 reg [5:0] current, next;  
   
 reg [7:0] countX;  
 reg [6:0] countY;  
 reg resX, resY, enX, enY;  
  
 reg [2:0] cursorX, cursorY, oldCX, oldCY;  
 reg curRes, curCentre, cxAdd, cxSub, cyAdd, cySub;  
   
 reg [1:0] borderStep;  
 reg resBStep, bStepChange;  
   
 //Board data - access with board[y][x] to get the state  
 reg [1:0] board [7:0][7:0];  
 reg resBoard, ldBoard;  
   
 reg turn, resTurn, changeTurn;  
   
 reg [5:0] tileReturn;  
 reg selTileRet, ldTileRet;  
   
 reg [2:0] count;  
 reg resCount, enCount;  
   
 //assign cxOut = cursorX;  
 //assign cyOut = cursorY;  
   
 output [5:0] outState;  
 assign outState = current;  
   
 input [5:0] switches;  
 output [1:0] outBoard;  
 assign outBoard = board[switches[5:3]][switches[2:0]];  
   
 output outTurn;  
 assign outTurn = turn;  
   
 reg enXBox, resetXBox, enYBox, resetYBox;  
 reg [6:0] xBox, yBox;

reg [3:0] ckFromX, ckFromY, ckX, ckY;

reg [1:0] ckDirX, ckDirY;

reg[5:0] ck8d\_doneReturn, ck8d\_valReturn;

localparam  
 IDLE = 6'b000000,  
   
 PRE\_BLACK = 6'b000001,  
   
 FILL\_BLACK = 6'b000010,  
   
 PRE\_GAME = 6'b000011,  
   
 DRAW\_BB = 6'b000100,  
   
 WAIT\_INPUT = 6'b000101,  
   
 UP\_W = 6'b000110,  
 DOWN\_W = 6'b000111,  
 LEFT\_W = 6'b001000,  
 RIGHT\_W = 6'b001001,  
 PLACE\_W = 6'b001010,  
   
 UP = 6'b001011,  
 DOWN = 6'b001100,  
 LEFT = 6'b001101,  
 RIGHT = 6'b001110,  
 PLACE = 6'b001111,  
   
 DRAW\_CURSOR = 6'b010000,  
 CURSOR\_T = 6'b010001,  
 CURSOR\_R = 6'b010010,  
 CURSOR\_B = 6'b010011,  
 CURSOR\_L = 6'b010100,  
   
 PLOT\_TILE = 6'b010101,  
   
 INVAL\_MOVE = 6'b010110,  
 VAL\_MOVE = 6'b010111,  
   
 INIT\_TILES = 6'b011000,  
 TILE\_LOOP = 6'b011001;

PRE\_CK8D = 6'b011010,

CK8D\_LOOP = 6'b011011,

CK8D\_NEXT = 6'b011100;

PRE\_FLIP = ;

FLIP = ;

//Circuit A - determine next state  
 always @(\*)  
 begin  
 case (current)  
 IDLE: begin  
 next = PRE\_BLACK;  
 end

PRE\_BLACK: next = FILL\_BLACK;  
   
 FILL\_BLACK: begin  
 if (countX >= 159 & countY >= 119) next = PRE\_GAME;  
 else next = FILL\_BLACK;  
 end  
   
 PRE\_GAME: next = DRAW\_BB;  
   
 DRAW\_BB: begin  
 next = (xBox > 7 & yBox > 6) ? INIT\_TILES : DRAW\_BB;  
 end  
  
 WAIT\_INPUT: begin  
 if (toggle\_keys == 1) begin  
 if (key3 == 1) next = PLACE\_W;  
 else next = WAIT\_INPUT;  
 end  
 else if (key3 == 1) next = LEFT\_W;  
 else if (key2 == 1) next = UP\_W;  
 else if (key1 == 1) next = DOWN\_W;  
 else if (key0 == 1) next = RIGHT\_W;  
 else next = WAIT\_INPUT;  
 end  
  
 UP\_W: next = key2 ? UP\_W : UP;  
   
 DOWN\_W: next = key1 ? DOWN\_W : DOWN;  
   
 LEFT\_W: next = key3 ? LEFT\_W : LEFT;  
  
 RIGHT\_W: next = key0 ? RIGHT\_W : RIGHT;  
   
 PLACE\_W: next = key3 ? PLACE\_W : PLACE;  
   
 UP: next = DRAW\_CURSOR;  
   
 DOWN: next = DRAW\_CURSOR;  
   
 LEFT: next = DRAW\_CURSOR;  
   
 RIGHT: next = DRAW\_CURSOR;  
   
 PLACE: begin  
 if (board[cursorY][cursorX] != 2'b10) next = INVAL\_MOVE;  
 else next = VAL\_MOVE;  
 end  
  
 DRAW\_CURSOR: next = CURSOR\_T;  
   
 CURSOR\_T: begin  
 if (countX < 8) next = CURSOR\_T;  
 else next = CURSOR\_R;  
 end  
   
 CURSOR\_R: begin  
 if (countY < 8) next = CURSOR\_R;  
 else next = CURSOR\_B;  
 end  
   
 CURSOR\_B: begin  
 if (countX < 8) next = CURSOR\_B;  
 else next = CURSOR\_L;  
 end  
   
 CURSOR\_L: begin  
 if (countY < 8) next = CURSOR\_L;  
 else begin  
 if (borderStep < 2) next = DRAW\_CURSOR;  
 else next = WAIT\_INPUT;  
 end  
 end  
   
 PLOT\_TILE: begin  
 if (countX >= 7 & countY >= 7) next = tileReturn;  
 else next = PLOT\_TILE;  
 end

INVAL\_MOVE: next = WAIT\_INPUT;  
   
 VAL\_MOVE: next = PLOT\_TILE;  
   
 INIT\_TILES: next = TILE\_LOOP;  
   
 TILE\_LOOP: begin  
 if (count < 4) next = PLOT\_TILE;  
 else next = DRAW\_CURSOR;  
 end  
   
 default: next = IDLE;  
 endcase  
 end  
  
 //Circuit B - determine outputs  
 always @ (\*)  
 begin  
 posX = 8'b00000000;  
 posY = 7'b0000000;  
 col = 3'b000;  
 opX = 3'b000;  
 opY = 3'b000;  
 selX = 2'b00;  
 selY = 2'b00;  
 selCol = 0;  
 ldX = 0;  
 ldY = 0;  
 ldCol = 0;  
 plot = 2'b00;  
   
 resX = 0;  
 resY = 0;  
 enX = 0;  
 enY = 0;  
   
 curRes = 0;  
 curCentre = 0;  
 cxAdd = 0;  
 cxSub = 0;  
 cyAdd=0;  
 cySub=0;  
   
 cxOut = 0;  
 cyOut = 0;  
   
 resBStep = 0;  
 bStepChange = 0;  
   
 resBoard = 0;  
 ldBoard = 0;  
   
 resTurn = 0;  
 changeTurn = 0;  
   
 selTileRet = 0;  
 ldTileRet = 0;  
   
 resCount = 0;  
 enCount = 0;  
   
 selCtrl = 0;  
   
 enXBox = 0;  
 resetXBox = 0;  
 enYBox = 0;  
 resetYBox = 0;  
   
 case (current)  
   
 PRE\_BLACK: begin  
 col = 3'b000;  
 selX = 2'b10;  
 selY = 2'b10;  
 ldX = 1;  
 ldY = 1;  
 ldCol = 1;  
   
 resX = 1;  
 resY = 1;  
 end  
   
 FILL\_BLACK: begin  
 if (countX < 159 & countY <= 119) begin  
 selX = 2'b01;  
 ldX = 1;  
 opX = 3'b000;  
   
 enX = 1;  
 end  
 else if (countY < 119) begin  
 selX = 2'b10;  
 ldX = 1;  
   
 selY = 2'b01;  
 ldY = 1;  
 opY = 3'b000;  
   
 resX = 1;  
 enY = 1;  
 end  
  
 plot = 1;  
 end  
   
 PRE\_GAME: begin  
 curRes = 1;  
   
 resX = 1;  
 resY = 1;  
  
 resBoard = 1;  
 resTurn = 1;  
   
 posX = 40;  
 posY = 35;  
 col = 3'b001;  
 ldX = 1;  
 ldY = 1;  
 ldCol = 1;  
 selX = 2'b00; // load starting board coord  
 selY = 2'b00;  
   
 resetXBox = 1;  
 resetYBox = 1;  
 end  
   
 DRAW\_BB: begin  
 plot = 1;  
   
 ldCol = 1;  
 selCol = 0;  
 if (countX < 0 | countX > 7 | countY < 1 | countY > 8) // if within range, color  
 col = 3'b001;  
 else   
 col = 3'b011;  
   
 if (countX < 9 & countY <= 9) // same line keep going   
 begin  
 opX = 3'b000;  
 ldX = 1;  
 selX = 2'b01;  
 enX = 1;  
 if (countX == 8 & countY == 9)  
 enXBox = 1;  
 end  
 else if (countY < 9) // finishes one line   
 begin  
 opX = 3'b011;  
 opY = 3'b000;  
 ldX = 1;  
 ldY = 1;  
 selX = 2'b01;  
 selY = 2'b01;  
 resX = 1;  
 enY = 1;  
 end  
 else // end of one box  
 begin  
 opX = 3'b000;  
 opY = 3'b011;  
 ldX = 1;  
 ldY = 1;  
 selX = 2'b01;  
 selY = 2'b01;  
 resX = 1;  
 resY = 1;  
   
 if (xBox > 7) // if one row is completed  
 begin  
 opY = 3'b000;  
 posX = 40;  
 ldX = 1;  
 selX = 2'b00;  
 resetXBox = 1;  
 enYBox = 1;  
 end  
   
 end  
   
 end  
  
 WAIT\_INPUT: begin  
 resBStep = 1;  
 end  
   
 UP: begin  
 if (cursorY > 0) cySub=1;  
 end  
  
 DOWN: begin  
 if (cursorY < 7) cyAdd=1;  
 end  
  
 LEFT: begin  
 if (cursorX > 0) cxSub=1;  
 end  
  
 RIGHT: begin  
 if (cursorX < 7) cxAdd=1;  
 end  
  
 DRAW\_CURSOR: begin  
 if (borderStep == 0) begin  
 posX = 40 + 10 \* oldCX;  
 posY = 35 + 10 \* oldCY;  
 col = 3'b001;  
 end  
 else begin  
 posX = 40 + 10 \* cursorX;  
 posY = 35 + 10 \* cursorY;  
 col = 3'b100;  
 end  
 selX = 2'b00;  
 selY = 2'b00;  
 ldX = 1;  
 ldY = 1;  
 ldCol = 1;  
  
 resX = 1;  
 resY = 1;  
   
 bStepChange = 1;  
 end  
   
 CURSOR\_T: begin  
 selX = 2'b01;  
 ldX = 1;  
 opX = 3'b000;  
 plot = 2'b01;  
  
 enX = 1;  
 end  
  
 CURSOR\_R: begin  
 selY = 2'b01;  
 ldY = 1;  
 opY = 3'b000;  
 plot = 2'b01;  
  
 enY = 1;  
 resX = 1;  
 end  
  
 CURSOR\_B: begin  
 selX = 2'b01;  
 ldX = 1;  
 opX = 3'b001;  
 plot = 2'b01;  
  
 enX = 1;  
 resY = 1;  
 end  
  
 CURSOR\_L: begin  
 selY = 2'b01;  
 ldY = 1;  
 opY = 3'b001;  
 plot = 2'b01;  
  
 enY = 1;  
 end  
   
 PLOT\_TILE: begin  
 if (countX < 7 & countY <= 7) begin  
 selX = 2'b01;  
 ldX = 1;  
 opX = 3'b000;  
   
 enX = 1;  
 end  
 else if (countY < 7) begin  
 selX = 2'b01;  
 ldX = 1;  
 opX = 3'b010;  
   
 selY = 2'b01;  
 ldY = 1;  
 opY = 3'b000;  
   
 resX = 1;  
 enY = 1;  
 end  
   
 cxOut = cursorX;  
 cyOut = cursorY;  
   
 plot = 2'b10;  
 end  
   
 VAL\_MOVE: begin  
 posX = 41 + 10 \* cursorX;  
 posY = 36 + 10 \* cursorY;  
   
 if(turn == 0) col = 3'b000;  
 else col = 3'b111;  
   
 selX = 2'b00;  
 selY = 2'b00;  
 ldX = 1;  
 ldY = 1;  
 ldCol = 1;  
  
 resX = 1;  
 resY = 1;  
   
 ldBoard = 1;  
 changeTurn = 1;  
   
 selTileRet = 0;  
 ldTileRet = 1;  
 end  
   
 INIT\_TILES: begin  
   
 curCentre = 1;  
   
 resCount = 1;  
 end  
   
 TILE\_LOOP: begin  
 case (count)  
 0: begin  
 posX = 41 + 10 \* 3;  
 posY = 36 + 10 \* 3;  
 col = 3'b111;  
 end  
 1: begin  
 posX = 41 + 10 \* 4;  
 posY = 36 + 10 \* 3;  
 col = 3'b000;  
   
 cxAdd = 1;  
 end  
 2: begin  
 posX = 41 + 10 \* 4;  
 posY = 36 + 10 \* 4;  
 col = 3'b111;  
   
 cyAdd = 1;  
 end  
 3: begin  
 posX = 41 + 10 \* 3;  
 posY = 36 + 10 \* 4;  
 col = 3'b000;  
   
 cxSub = 1;  
 end  
 default: begin  
 col = 3'b000;  
 end  
 endcase  
   
 selX = 2'b00;  
 selY = 2'b00;  
 ldX = 1;  
 ldY = 1;  
 ldCol = 1;  
   
 resX = 1;  
 resY = 1;  
   
 selTileRet = 1;  
 ldTileRet = 1;  
   
 enCount = 1;  
 end  
   
 endcase  
 end  
  
 //State FFs  
 always @ (posedge clock)  
 begin  
 if (resetn == 0)  
 current <= IDLE;  
 else  
 current <= next;  
 end  
  
 // x box counter  
 always @(posedge clock)  
 begin  
 if (resetn == 0 | resetXBox == 1)  
 xBox <= 0;  
 else if (enXBox)  
 xBox <= xBox + 1;  
 end  
   
 // y box counter  
 always @(posedge clock)  
 begin  
 if (resetn == 0 | resetYBox == 1)  
 yBox <= 0;  
 else if (enYBox)  
 yBox <= yBox + 1;  
 end  
   
 //PLOT\_TILE return state  
 always @ (posedge clock) begin  
 if (resetn == 0)  
 tileReturn <= IDLE;  
 else if (ldTileRet == 1) begin  
 case (selTileRet)  
 0: tileReturn <= WAIT\_INPUT;  
 1: tileReturn <= TILE\_LOOP;  
 default: tileReturn <= IDLE;  
 endcase  
 end  
 end  
   
 //General counters  
 always @ (posedge clock)  
 begin  
 if (resetn == 0) begin  
 countX <= 0;  
 countY <= 0;  
 count <= 0;  
 end  
 else begin  
 if (resX == 1)  
 countX <= 0;  
 else if (enX == 1)  
 countX <= countX + 1;  
   
 if (resY == 1)  
 countY <= 0;  
 else if (enY == 1)  
 countY <= countY + 1;  
   
 if (resCount == 1)  
 count <= 0;  
 else if (enCount == 1)  
 count <= count + 1;  
 end  
 end  
   
 //Cursor position  
 always @ (posedge clock) begin  
 if(resetn == 0 | curRes == 1) begin  
 cursorX <= 0;  
 cursorY <= 0;  
   
 oldCX <= 0;  
 oldCY <= 0;  
 end  
 else if (curCentre == 1) begin  
 cursorX <= 3;  
 cursorY <= 3;  
   
 oldCX <= 3;  
 oldCY <= 3;  
 end  
 else if (cxAdd == 1 | cxSub == 1 | cyAdd == 1 | cySub == 1) begin  
   
 oldCX <= cursorX;  
 oldCY <= cursorY;  
   
 if(cxAdd == 1) begin  
 cursorX <= cursorX + 1;  
 end  
 else if(cxSub == 1) begin  
 cursorX <= cursorX - 1;  
 end  
   
 if(cyAdd == 1) begin  
 cursorY <= cursorY + 1;  
 end  
 else if(cySub == 1) begin  
 cursorY <= cursorY - 1;  
 end  
 end  
 end  
   
 //Border step  
 always @ (posedge clock) begin  
 if (resetn == 0 | resBStep == 1)  
 borderStep <= 0;  
 else if (bStepChange == 1)  
 borderStep <= borderStep + 1;  
 end  
   
 //Game board data  
 reg [3:0] i, j;  
 always @ (posedge clock) begin  
 if (resetn == 0 | resBoard == 1) begin  
 for (i = 0; i < 8; i = i + 1)  
 for (j = 0; j < 8; j = j + 1) begin  
 if (i == 3 & j == 3 | i == 4 & j == 4)  
 board[i][j] <= 2'b01;  
 else if (i == 3 & j == 4 | i == 4 & j == 3)  
 board[i][j] <= 2'b00;  
 else  
 board[i][j] <= 2'b10;  
 end  
 end  
 else if (ldBoard == 1)  
 board[cursorY][cursorX] <= turn;  
 end  
   
 //Turn  
 always @ (posedge clock) begin  
 if (resetn == 0 | resTurn == 1)  
 turn <= 0;  
 else if (changeTurn == 1)  
 turn <= ~turn;  
 end  
   
endmodule  
  
  
  
///////////////  
// DATA PATH //  
///////////////  
  
module data(  
 inX, inY, inCol, opX, opY, selX, selY, selCol,  
 ldX, ldY, ldCol, plot, cx, cy, resetn, clock,  
 outX, outY, outCol, writeEn  
 );  
  
 input [7:0] inX;  
 input [6:0] inY;  
 input [2:0] inCol, opX, opY, cx, cy;  
 input [1:0] selX, selY, plot;  
 input selCol, ldX, ldY, ldCol, resetn, clock;  
  
 output [7:0] outX;  
 output [6:0] outY;  
 output [2:0] outCol;  
 output reg writeEn;  
  
 reg [7:0] regX;  
 reg [6:0] regY;  
 reg [2:0] regCol;  
  
 reg [7:0] aluX;  
 reg [6:0] aluY;  
  
 wire inCircle;  
 assign inCircle = ((10\*regX - 100\*cx - 445)\*\*2 + (10\*regY - 100\*cy - 395)\*\*2 <= 1600) ? 1 : 0;  
   
 assign outX = regX;  
 assign outY = regY;  
 assign outCol = regCol;  
  
 //regX  
 always @ (posedge clock) begin  
 if (resetn == 0)  
 regX <= 0;  
 else if (ldX == 1) begin  
 case (selX)  
 2'b00: regX <= inX;  
 2'b01: regX <= aluX;  
 2'b10: regX <= 0;  
 default: regX <= 0;  
 endcase  
 end  
 end  
   
 //regY  
 always @ (posedge clock) begin  
 if (resetn == 0)  
 regY <= 0;  
 else if (ldY == 1) begin  
 case (selY)  
 2'b00: regY <= inY;  
 2'b01: regY <= aluY;  
 2'b10: regY <= 0;  
 default: regY <= 0;  
 endcase  
 end  
 end  
   
 //regCol  
 always @ (posedge clock) begin  
 if (resetn == 0)  
 regCol <= 3'b000;  
 else if (ldCol == 1) begin  
 case (selCol)  
 1'b0: regCol <= inCol;  
 1'b1: regCol <= 3'b000;  
 default: regCol <= 3'b000;  
 endcase  
 end  
 end  
   
  
 //aluX  
 always @ (\*) begin  
 case (opX)  
 3'b000: aluX <= regX + 1;  
 3'b001: aluX <= regX - 1;  
 3'b010: aluX <= regX - 7;  
 3'b011: aluX <= regX - 9;  
 default: aluX <= regX;  
 endcase  
 end  
  
 //aluY  
 always @ (\*) begin  
 case (opY)  
 3'b000: aluY <= regY + 1;  
 3'b001: aluY <= regY - 1;  
 3'b010: aluY <= regY - 7;  
 3'b011: aluY <= regY - 9;  
 default: aluY <= regY;  
 endcase  
 end  
   
 //plot  
 always @ (\*) begin  
 case (plot)  
 2'b00: writeEn <= 0;  
 2'b01: writeEn <= 1;  
 2'b10: writeEn <= inCircle;  
 default: writeEn <= 0;  
 endcase  
 end  
   
endmodule  
  
module letter\_decoder(  
 resetn, clock, posX, posY,  
 col, opX, opY, selX, selY, selCol,  
 ldX, ldY, ldCol, plot  
 );  
  
 input resetn, clock;  
  
 output reg [7:0] posX;  
 output reg [6:0] posY;  
 output reg [2:0] col, opX, opY;  
 output reg [1:0] selX, selY, plot;  
 output reg selCol, ldX, ldY, ldCol;  
   
endmodule  
  
//HEX display  
module hex\_decoder(hex\_digit, segments);  
 input [3:0] hex\_digit;  
 output reg [6:0] segments;  
  
 always @(\*)  
 case (hex\_digit)  
 4'h0: segments = 7'b100\_0000;  
 4'h1: segments = 7'b111\_1001;  
 4'h2: segments = 7'b010\_0100;  
 4'h3: segments = 7'b011\_0000;  
 4'h4: segments = 7'b001\_1001;  
 4'h5: segments = 7'b001\_0010;  
 4'h6: segments = 7'b000\_0010;  
 4'h7: segments = 7'b111\_1000;  
 4'h8: segments = 7'b000\_0000;  
 4'h9: segments = 7'b001\_1000;  
 4'hA: segments = 7'b000\_1000;  
 4'hB: segments = 7'b000\_0011;  
 4'hC: segments = 7'b100\_0110;  
 4'hD: segments = 7'b010\_0001;  
 4'hE: segments = 7'b000\_0110;  
 4'hF: segments = 7'b000\_1110;   
 default: segments = 7'h7f;  
 endcase  
endmodule