`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

cx, cy : cursor position

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

\*/

wire [7:0] posX;

wire [6:0] posY;

wire [2:0] col, opX, opY, cx, cy;

wire [1:0] selX, selY, plot;

wire selCol, ldX, ldY, ldCol;

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(0, HEX4);

hex\_decoder h5(0, HEX5);

control c0(

SW[0], ~KEY[0], ~KEY[1], ~KEY[2], ~KEY[3], resetn, CLOCK\_50, posX, posY,

col, opX, opY, selX, selY, selCol,

ldX, ldY, ldCol, plot, cx, cy, LEDR[4:0]

);

data d0(

posX, posY, col, opX, opY, selX, selY, selCol,

ldX, ldY, ldCol, plot, cx, cy, resetn, CLOCK\_50,

x, y, colour, writeEn

);

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, outState

);

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;

output [2:0] cxOut, cyOut;

reg [4:0] current, next;

reg [7:0] countX;

reg [6:0] countY;

reg resX, resY, enX, enY;

reg [2:0] cursorX, cursorY, oldCX, oldCY;

reg cxRes, cxAdd, cxSub, cyRes, cyAdd, cySub, oldCXld, oldCYld;

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 [2:0] boardX, boardY;

assign cxOut = cursorX;

assign cyOut = cursorY;

output [4:0] outState;

assign outState = current;

localparam

IDLE = 5'b0000,

PRE\_BLACK = 5'b00001,

FILL\_BLACK = 5'b00010,

PRE\_GAME = 5'b00011,

DRAW\_BB = 5'b00100,

WAIT\_INPUT = 5'b00101,

UP\_W = 5'b00110,

DOWN\_W = 5'b00111,

LEFT\_W = 5'b01000,

RIGHT\_W = 5'b01001,

PLACE\_W = 5'b01010,

UP = 5'b01011,

DOWN = 5'b01100,

LEFT = 5'b01101,

RIGHT = 5'b01110,

PLACE = 5'b01111,

DRAW\_CURSOR = 5'b10000,

CURSOR\_T = 5'b10001,

CURSOR\_R = 5'b10010,

CURSOR\_B = 5'b10011,

CURSOR\_L = 5'b10100,

PLOT\_CIRCLE = 5'b10101;

//Circuit A - determine next state

always @(\*)

begin

case (current)

IDLE: begin

next = PRE\_GAME;

end

PRE\_BLACK: next = FILL\_BLACK;

FILL\_BLACK: begin

if (countX >= 159 & countY >= 119) next = DRAW\_BB;

else next = FILL\_BLACK;

end

PRE\_GAME: next = PRE\_BLACK;

DRAW\_BB: begin

next = DRAW\_CURSOR;

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: next = PLOT\_CIRCLE;

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\_CIRCLE: begin

if (countX >= 7 & countY >= 7) next = WAIT\_INPUT;

else next = PLOT\_CIRCLE;

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;

cxRes = 0;

cxAdd = 0;

cxSub = 0;

cyRes=0;

cyAdd=0;

cySub=0;

oldCXld = 0;

oldCYld = 0;

resBStep = 0;

bStepChange = 0;

case (current)

IDLE: begin

end

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

cxRes = 1;

cyRes = 1;

end

DRAW\_BB: begin

end

WAIT\_INPUT: begin

oldCXld = 1;

oldCYld = 1;

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

PLACE: begin

posX = 41 + 10 \* cursorX;

posY = 36 + 10 \* cursorY;

col = 3'b111;

selX = 2'b00;

selY = 2'b00;

ldX = 1;

ldY = 1;

ldCol = 1;

resX = 1;

resY = 1;

end

DRAW\_CURSOR: begin

if (borderStep == 0) begin

posX = 40 + 10 \* oldCX;

posY = 35 + 10 \* oldCY;

col = 3'b000;

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\_CIRCLE: 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

plot = 2'b10;

end

endcase

end

//State FFs

always @ (posedge clock)

begin

if (resetn == 0)

current <= IDLE;

else

current <= next;

end

//General counters

always @ (posedge clock)

begin

if (resetn == 0) begin

countX <= 0;

countY <= 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;

end

end

//Cursor position

always @ (posedge clock) begin

if(resetn == 0) begin

cursorX <= 0;

cursorY <= 0;

end

else begin

if(cxRes == 1)

cursorX <= 0;

else if(cxAdd == 1)

cursorX <= cursorX+1;

else if(cxSub == 1)

cursorX <= cursorX-1;

if(cyRes==1)

cursorY <= 0;

else if(cyAdd == 1)

cursorY <= cursorY+1;

else if(cySub == 1)

cursorY <= cursorY-1;

end

end

//Previous cursor position

always @ (posedge clock) begin

if (resetn == 0) begin

oldCX <= 0;

oldCY <= 0;

end

else begin

if (oldCXld == 1)

oldCX <= cursorX;

if (oldCYld == 1)

oldCY <= cursorY;

end

end

//Border step

always @ (posedge clock) begin

if (resetn == 0 | resBStep == 1) begin

borderStep <= 0;

end

else if (bStepChange == 1)

borderStep <= borderStep + 1;

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;

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;

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

//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