`timescale 1ns / 1ns // `timescale time\_unit/time\_precision

//////////////////////////////

// TOP MODULE

//////////////////////////////

module project

(

SW,

KEY,

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;

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 = KEY[0];

// 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

\*/

wire [7:0] posX;

wire [6:0] posY;

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

wire [1:0] selX, selY;

wire selCol, ldX, ldY, ldCol;

control c0(

SW[0], SW[1], SW[2], SW[3], SW[4], KEY[0], CLOCK\_50, posX, posY,

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

ldX, ldY, ldCol, writeEn

);

data d0(

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

ldX, ldY, ldCol, KEY[0], CLOCK\_50,

x, y, colour

);

endmodule

/////////////////////////////////

// CONTROL PATH

//////////////////////////////////

module control(

enter, key\_right, key\_down, key\_up, key\_left, resetn, clock, posX, posY,

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

ldX, ldY, ldCol, plot

);

input enter, key\_up, key\_left, key\_right, key\_down, 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;

output reg selCol, ldX, ldY, ldCol, plot;

reg [4:0] current, next;

reg [7:0] countX;

reg [6:0] countY;

reg resX, resY, enX, enY;

reg playerX, playerY;

localparam IDLE = 4’b0000;

DRAW\_BB = 4’b0001,

WAIT\_INPUT = 4’b0010,

UP = 4’b0011,

DOWN = 4’b0100,

LEFT = 4’b0101,

RIGHT = 4’b0110,

PLACE = 4’b0111,

DRAW\_BORDER = 4'b1000,

BORDER\_T = 4'b1000,

BORDER\_R = 4'b1000,

BORDER\_B = 4'b1000,

BORDER\_L = 4'b1000;

//Circuit A - determine next state

always @(\*)

begin

case (current)

IDLE: begin

if (enter == 1) next = DRAW\_BB;

else next = IDLE;

end

DRAW\_BB: begin

if (xCoord < 7 & yCoord < 7) next = DRAW\_BB; else next = WAIT\_INPUT;

end

WAIT\_INPUT: begin

if (key\_up == 1) next = UP;

else if (key\_down == 1) next = DOWN;

else if (key\_left == 1) next = LEFT;

else if (key\_right == 1) next = RIGHT;

else if (key\_enter == 1) next = PLACE;

else next = WAIT\_INPUT;

end

UP: begin

if (key\_up == 1) next = UP;

else next = DRAW\_BORDER;

end

DOWN: begin

if (key\_down == 1) next = DOWN;

else next = DRAW\_BORDER;

end

LEFT: begin

if (key\_left == 1) next = LEFT;

else next = DRAW\_BORDER;

end

RIGHT: begin

if (key\_right == 1) next = RIGHT;

else next = DRAW\_BORDER;

end

PLACE: begin

next = WAIT\_INPUT;

end

DRAW\_BORDER: begin

next = BORDER\_T;

end

BORDER\_T: begin

if (countX < 7) next = BORDER\_T;

else next = BORDER\_R;

end

BORDER\_R: begin

if (countY < 7) next = BORDER\_R;

else next = BORDER\_B;

end

BORDER\_B: begin

if (countX < 7) next = BORDER\_B;

else next = BORDER\_L;

end

BORDER\_L: begin

if (countY < 7) next = BORDER\_L;

else next = WAIT\_INPUT;

end

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 = 0;

resX = 0;

resY = 0;

enX = 0;

enY = 0;

case (current)

IDLE: begin

playerX = 0;

playerY = 0;

end

DRAW\_BB: begin

drawX = 1;

drawY = 1;

plot = 1;

posX = 41 + 10\*xCoord + bxX;

posY = 36 + 10\*yCoord + bxY;

col = 3’b010;

ldX = 1;

ldY = 1;

ldCol = 1;

end

UP: begin

if (playerY > 0) playerY = playerY - 1;

end

DOWN: begin

if (playerY < 7) playerY = playerY + 1;

end

LEFT: begin

if (playerX > 0) playerX = playerX - 1;

end

RIGHT: begin

if (playerX < 7) playerX = playerX + 1;

end

PLACE: begin

end

DRAW\_BORDER: begin

posX = 40 + 10 \* playerX;

posY = 35 + 10 \* playerY;

col = 3'b100;

ldX = 1;

ldY = 1;

ldCol = 1;

resX = 1;

resY = 1;

end

BORDER\_T: begin

selX = 2'b01;

ldX = 1;

plot = 1;

enX = 1;

end

BORDER\_R: begin

selY = 2'b01;

ldY = 1;

plot = 1;

enY = 1;

resX = 1;

end

BORDER\_B: begin

selX = 2'b01;

ldX = 1;

opX = 3'b001;

plot = 1;

enX = 1;

resY = 1;

end

BORDER\_L: begin

selY = 2'b01;

ldY = 1;

opY =3'b001;

plot = 1;

enY = 1;

end

endcase

end

//State FFs

always @ (posedge clock)

begin

if (resetn == 0)

current <= IDLE;

else

current <= next;

end

//x pixel board counter

always @(posedge)

begin

if (reset == 0 | bxX == 7)

bxX <= 0;

else if (drawX == 1 & bxX < 7)

bxX <= bxX + 1;

end

// x coord counter

always @(posedge)

begin

if (reset == 0 | xCoord == 7)

xCoord <= 0;

else if (xCoord < 7 & bxX == 7 & bxY == 7)

xCoord <= xCoord + 1;

end

//y pixel board counter

always @(posedge)

begin

if (reset == 0 | bxY == 7)

bxY <= 0;

else if (drawY == 1 & bxY < 7 & bxX == 7)

bxY <= bxY + 1;

end

// y coord counter

always @(posedge)

begin

if (reset == 0 | yCoord == 7)

yCoord <= 0;

else if (yCoord < 7 & xCoord == 7 & bxY == 7 & bxX == 7)

yCoord <= yCoord + 1;

end

//Pixel 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

endmodule

/////////////////////////

// DATA PATH

/////////////////////////

module data(

inX, inY, inCol, opX, opY, selX, selY, selCol,

ldX, ldY, ldCol, resetn, clock,

outX, outY, outCol

);

input [7:0] inX;

input [6:0] inY;

input [2:0] inCol, opX, opY;

input [1:0] selX, selY;

input selCol, ldX, ldY, ldCol, resetn, clock;

output [7:0] outX;

output [6:0] outY;

output [2:0] outCol;

reg [7:0] regX;

reg [6:0] regY;

reg [2:0] regCol;

reg [7:0] aluX;

reg [6:0] aluY;

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;

endcase

end

//aluY

always @ (\*) begin

case (opY)

3'b000: aluY = regY + 1;

3'b001: aluY = regY - 1;

endcase

end

endmodule