Members:

Patrick Tayag

Brandon Hoang

Mohammad Nasiri

Diego Buenrostro

California State University, Fullerton

Computer Engineering

**EGCP 446 – Advanced Digital Design using Verilog HDL**

**(Fall 2019)**

**Project C: State Machine for Score**

1. **Lab No 8 – Task**

* Create a group of four students. Please note this group can’t be change until the end of the semester.
* Create a LabProjC project and import all the provided Verilog files as well as the provided constraints file.
* Modify the pong\_graph\_animate.v to display zero, one, two, three, four, five, six, seven, eight and nine in a sequential order on the VGA screen every time the ball hits the paddle.
* Generate the bit stream and program the FPGA with your design.
* Verify that the hardware’s behavior works as expected.
* Submit all your Verilog code and constrain files. Also, include the picture of your VGA screen for all the ten numbers.
* Please include the names of each group members involved.

Submit your Verilog code here

Zero.v  
module Zero\_0(clk, reset,video\_on,refr\_tick, pix\_x, pix\_y,zero\_on,zero\_rgb);

input clk, reset;

input video\_on, refr\_tick;

input [9:0] pix\_x, pix\_y;

output zero\_on;

output [2:0] zero\_rgb;

//--------------------------------------------

// Zero Logo

//--------------------------------------------

wire [3:0] Zero\_rom\_addr;

wire [4:0] Zero\_rom\_col;

reg [31:0] Zero\_rom\_data;

wire Zero\_rom\_bit,Zero\_Logo\_on, Logo\_on ;

//--------------------------------------------

// Zero Logo Positioning

//--------------------------------------------

// Logo Position left, Top boundary

localparam Zero\_X\_L = 300;

localparam Zero\_X\_R = 331;

localparam Zero\_X\_T = 10;

localparam Zero\_X\_B = 25;

localparam Zero\_Logo\_x = 32;

localparam Zero\_Logo\_y = 16;

// Zero left, right boundary

wire [9:0] Zero\_x\_l, Zero\_x\_r;

// Zero top, bottom boundary

wire [9:0] Zero\_y\_t, Zero\_y\_b;

// reg to track left, top position

reg [9:0] Zero\_x\_reg, Zero\_y\_reg;

wire [9:0] Zero\_x\_next, Zero\_y\_next;

// body

//--------------------------------------------

// Zero Logo

//--------------------------------------------

always @\*

case (Zero\_rom\_addr)

6'h0: Zero\_rom\_data = 32'b00000000000011111111000000000000; // \*\*\*\*

6'h1: Zero\_rom\_data = 33'b00000000000011111111000000000000; // \*\*\*\*\*\*

6'h2: Zero\_rom\_data = 32'b00000000001100000000110000000000; // \*\*\*\*\*\*\*\*

6'h3: Zero\_rom\_data = 32'b00000000001100000000110000000000; // \*\*\*\*\*\*\*\*

6'h4: Zero\_rom\_data = 32'b00000000001100000000110000000000; // \*\*\*\*\*\*\*\*

6'h5: Zero\_rom\_data = 32'b00000000001100000000110000000000; // \*\*\*\*\*\*\*\*

6'h6: Zero\_rom\_data = 32'b00000000001100000000110000000000; // \*\*\*\*\*\*

6'h7: Zero\_rom\_data = 32'b00000000001100000000110000000000; // \*\*\*\*

6'h8: Zero\_rom\_data = 32'b00000000000000000000000000000000; // \*\*\*\*

6'h9: Zero\_rom\_data = 32'b00000000001100000000110000000000; // \*\*\*\*\*\*

6'hA: Zero\_rom\_data = 32'b00000000001100000000110000000000; // \*\*\*\*\*\*\*\*

6'hB: Zero\_rom\_data = 32'b00000000001100000000110000000000; // \*\*\*\*\*\*\*\*

6'hC: Zero\_rom\_data = 32'b00000000001100000000110000000000; // \*\*\*\*\*\*\*\*

6'hD: Zero\_rom\_data = 32'b00000000001100000000110000000000; // \*\*\*\*\*\*\*\*

6'hE: Zero\_rom\_data = 32'b00000000000011111111000000000000; // \*\*\*\*\*\*

6'hF: Zero\_rom\_data = 32'b00000000000011111111000000000000; // \*\*\*\*

endcase

//------------------------------------------

// Register for Zero logo

//------------------------------------------

// registers

always @(posedge clk, posedge reset)

if (reset)

begin

Zero\_y\_reg <= 0;

end

else

begin

Zero\_x\_reg <= Zero\_x\_next;

Zero\_y\_reg <= Zero\_y\_next;

end

//--------------------------------------------

// Zero Logo

//--------------------------------------------

// boundary

assign Zero\_x\_l = Zero\_x\_reg;

assign Zero\_y\_t = Zero\_y\_reg;

assign Zero\_x\_r = Zero\_x\_l + Zero\_Logo\_x - 1;

assign Zero\_y\_b = Zero\_y\_t + Zero\_Logo\_y - 1;

// pixel within logo

assign Logo\_on =

(Zero\_X\_L<=pix\_x) && (pix\_x<=Zero\_x\_r) &&

(Zero\_X\_T<=pix\_y) && (pix\_y<=Zero\_y\_b);

//assign Logo\_on =

// (Zero\_x\_l<=Zero\_X\_L) && (pix\_x<=Zero\_x\_r) &&

// (Zero\_y\_t<=Zero\_X\_T) && (pix\_y<=Zero\_y\_b);

// map current pixel location to ROM addr/col

assign Zero\_rom\_addr = pix\_y[3:0] - Zero\_y\_t[3:0];

assign Zero\_rom\_col = pix\_x[4:0] - Zero\_x\_l[4:0];

assign Zero\_rom\_bit = Zero\_rom\_data[Zero\_rom\_col];

// pixel within logo

assign zero\_on = (Logo\_on & Zero\_rom\_bit);

assign zero\_rgb = 3'b101;

// assign Zero\_x\_next = (refr\_tick)? Zero\_x\_reg + 1'b1:Zero\_x\_reg;

// assign Zero\_y\_next = (refr\_tick) ? Zero\_y\_reg + 1'b1:Zero\_y\_reg;

assign Zero\_x\_next = (refr\_tick) ? Zero\_x\_reg: Zero\_X\_L;

assign Zero\_y\_next = (refr\_tick) ? Zero\_y\_reg : Zero\_X\_T ;

endmodule

One.v

module One\_1(clk, reset,video\_on, refr\_tick,pix\_x,pix\_y,one\_on,one\_rgb);

input clk, reset;

input video\_on, refr\_tick;

input [9:0] pix\_x;

input [9:0] pix\_y;

output one\_on;

output [2:0] one\_rgb;

//--------------------------------------------

// One Logo

//--------------------------------------------

wire [3:0] One\_rom\_addr;

wire [4:0] One\_rom\_col;

reg [31:0] One\_rom\_data;

wire One\_rom\_bit,One\_Logo\_on, Logo\_on ;

//--------------------------------------------

// One Logo Positioning

//--------------------------------------------

// Logo Position left, Top boundary

localparam One\_X\_L = 300;

localparam One\_X\_R = 331;

localparam One\_X\_T = 10;

localparam One\_X\_B = 25;

localparam One\_Logo\_x = 32;

localparam One\_Logo\_y = 16;

// One left, right boundary

wire [9:0] One\_x\_l, One\_x\_r;

// One top, bottom boundary

wire [9:0] One\_y\_t, One\_y\_b;

// reg to track left, top position

reg [9:0] One\_x\_reg, One\_y\_reg;

wire [9:0] One\_x\_next, One\_y\_next;

// body

//--------------------------------------------

// One Logo

//--------------------------------------------

always @\*

case (One\_rom\_addr)

6'h0: One\_rom\_data = 32'b00000000000000000000000000000110; // \*\*\*\*

6'h1: One\_rom\_data = 33'b00000000000000000000000000000110; // \*\*\*\*\*\*

6'h2: One\_rom\_data = 32'b00000000000000000000000000000110; // \*\*\*\*\*\*\*\*

6'h3: One\_rom\_data = 32'b00000000000000000000000000000110; // \*\*\*\*\*\*\*\*

6'h4: One\_rom\_data = 32'b00000000000000000000000000000110; // \*\*\*\*\*\*\*\*

6'h5: One\_rom\_data = 32'b00000000000000000000000000000110; // \*\*\*\*\*\*\*\*

6'h6: One\_rom\_data = 32'b00000000000000000000000000000110; // \*\*\*\*\*\*

6'h7: One\_rom\_data = 32'b00000000000000000000000000000000; // \*\*\*\*

6'h8: One\_rom\_data = 32'b00000000000000000000000000000000; // \*\*\*\*

6'h9: One\_rom\_data = 32'b00000000000000000000000000000110; // \*\*\*\*\*\*

6'hA: One\_rom\_data = 32'b00000000000000000000000000000110; // \*\*\*\*\*\*\*\*

6'hB: One\_rom\_data = 32'b00000000000000000000000000000110; // \*\*\*\*\*\*\*\*

6'hC: One\_rom\_data = 32'b00000000000000000000000000000110; // \*\*\*\*\*\*\*\*

6'hD: One\_rom\_data = 32'b00000000000000000000000000000110; // \*\*\*\*\*\*\*\*

6'hE: One\_rom\_data = 32'b00000000000000000000000000000110; // \*\*\*\*\*\*

6'hF: One\_rom\_data = 32'b00000000000000000000000000000110; // \*\*\*\*

endcase

//------------------------------------------

// Register for One logo

//------------------------------------------

// registers

always @(posedge clk, posedge reset)

if (reset)

begin

One\_y\_reg <= 0;

end

else

begin

One\_x\_reg <= One\_x\_next;

One\_y\_reg <= One\_y\_next;

end

//--------------------------------------------

// One Logo

//--------------------------------------------

// boundary

assign One\_x\_l = One\_x\_reg;

assign One\_y\_t = One\_y\_reg;

assign One\_x\_r = One\_x\_l + One\_Logo\_x - 1;

assign One\_y\_b = One\_y\_t + One\_Logo\_y - 1;

// pixel within logo

assign Logo\_on =

(One\_X\_L<=pix\_x) && (pix\_x<=One\_x\_r) &&

(One\_X\_T<=pix\_y) && (pix\_y<=One\_y\_b);

//assign Logo\_on =

// (One\_x\_l<=One\_X\_L) && (pix\_x<=One\_x\_r) &&

// (One\_y\_t<=One\_X\_T) && (pix\_y<=One\_y\_b);

// map current pixel location to ROM addr/col

assign One\_rom\_addr = pix\_y[3:0] - One\_y\_t[3:0];

assign One\_rom\_col = pix\_x[4:0] - One\_x\_l[4:0];

assign One\_rom\_bit = One\_rom\_data[One\_rom\_col];

// pixel within logo

assign one\_on = (Logo\_on & One\_rom\_bit);

assign one\_rgb = 3'b101;

// assign One\_x\_next = (refr\_tick)? One\_x\_reg + 1'b1:One\_x\_reg;

// assign One\_y\_next = (refr\_tick) ? One\_y\_reg + 1'b1:One\_y\_reg;

assign One\_x\_next = (refr\_tick) ? One\_x\_reg: One\_X\_L;

assign One\_y\_next = (refr\_tick) ? One\_y\_reg : One\_X\_T ;

endmodule

Two.v

module Two\_2(clk, reset,video\_on, refr\_tick,pix\_x,pix\_y,two\_on,two\_rgb);

input clk, reset;

input video\_on, refr\_tick;

input [9:0] pix\_x;

input [9:0] pix\_y;

output two\_on;

output [2:0] two\_rgb;

//--------------------------------------------

// Two Logo

//--------------------------------------------

wire [3:0] Two\_rom\_addr;

wire [4:0] Two\_rom\_col;

reg [31:0] Two\_rom\_data;

wire Two\_rom\_bit,Two\_Logo\_on, Logo\_on ;

//--------------------------------------------

// Two Logo Positioning

//--------------------------------------------

// Logo Position left, Top boundary

localparam Two\_X\_L = 300;

localparam Two\_X\_R = 331;

localparam Two\_X\_T = 10;

localparam Two\_X\_B = 25;

localparam Two\_Logo\_x = 32;

localparam Two\_Logo\_y = 16;

// Two left, right boundary

wire [9:0] Two\_x\_l, Two\_x\_r;

// Two top, bottom boundary

wire [9:0] Two\_y\_t, Two\_y\_b;

// reg to track left, top position

reg [9:0] Two\_x\_reg, Two\_y\_reg;

wire [9:0] Two\_x\_next, Two\_y\_next;

// body

//--------------------------------------------

// Two Logo

//--------------------------------------------

always @\*

case (Two\_rom\_addr)

6'h0: Two\_rom\_data = 32'b00000000000011111111100000000000; // \*\*\*\*

6'h1: Two\_rom\_data = 32'b00000000001111111111110000000000; // \*\*\*\*\*\*

6'h2: Two\_rom\_data = 32'b00000000011100000000111000000000; // \*\*\*\*\*\*\*\*

6'h3: Two\_rom\_data = 32'b00000000011100000000011100000000; // \*\*\*\*\*\*\*\*

6'h4: Two\_rom\_data = 32'b00000000001110000000001100000000; // \*\*\*\*\*\*\*\*

6'h5: Two\_rom\_data = 32'b00000000000111000000000000000000; // \*\*\*\*\*\*\*\*

6'h6: Two\_rom\_data = 32'b00000000000011100000000000000000; // \*\*\*\*\*\*

6'h7: Two\_rom\_data = 32'b00000000000001110000000000000000; // \*\*\*\*

6'h8: Two\_rom\_data = 32'b00000000000001110000000000000000; // \*\*\*\*

6'h9: Two\_rom\_data = 32'b00000000000000011100000000000000; // \*\*\*\*\*\*

6'hA: Two\_rom\_data = 32'b00000000000000001110000000000000; // \*\*\*\*\*\*\*\*

6'hB: Two\_rom\_data = 32'b00000000000000000111000000000000; // \*\*\*\*\*\*\*\*

6'hC: Two\_rom\_data = 32'b00000000000000000011100000000000; // \*\*\*\*\*\*\*\*

6'hD: Two\_rom\_data = 32'b00000000000000000001110000000000; // \*\*\*\*\*\*\*\*

6'hE: Two\_rom\_data = 32'b00000000011111111111111000000000; // \*\*\*\*\*\*

6'hF: Two\_rom\_data = 32'b00000000011111111111111000000000; // \*\*\*\*

endcase

//------------------------------------------

// Register for Two logo

//------------------------------------------

// registers

always @(posedge clk, posedge reset)

if (reset)

begin

Two\_y\_reg <= 0;

end

else

begin

Two\_x\_reg <= Two\_x\_next;

Two\_y\_reg <= Two\_y\_next;

end

//--------------------------------------------

// Two Logo

//--------------------------------------------

// boundary

assign Two\_x\_l = Two\_x\_reg;

assign Two\_y\_t = Two\_y\_reg;

assign Two\_x\_r = Two\_x\_l + Two\_Logo\_x - 1;

assign Two\_y\_b = Two\_y\_t + Two\_Logo\_y - 1;

// pixel within logo

assign Logo\_on =

(Two\_X\_L<=pix\_x) && (pix\_x<=Two\_x\_r) &&

(Two\_X\_T<=pix\_y) && (pix\_y<=Two\_y\_b);

//assign Logo\_on =

// (Two\_x\_l<=Two\_X\_L) && (pix\_x<=Two\_x\_r) &&

// (Two\_y\_t<=Two\_X\_T) && (pix\_y<=Two\_y\_b);

// map current pixel location to ROM addr/col

assign Two\_rom\_addr = pix\_y[3:0] - Two\_y\_t[3:0];

assign Two\_rom\_col = pix\_x[4:0] - Two\_x\_l[4:0];

assign Two\_rom\_bit = Two\_rom\_data[Two\_rom\_col];

// pixel within logo

assign two\_on = (Logo\_on & Two\_rom\_bit);

assign two\_rgb = 3'b101;

// assign Two\_x\_next = (refr\_tick)? Two\_x\_reg + 1'b1:Two\_x\_reg;

// assign Two\_y\_next = (refr\_tick) ? Two\_y\_reg + 1'b1:Two\_y\_reg;

assign Two\_x\_next = (refr\_tick) ? Two\_x\_reg: Two\_X\_L;

assign Two\_y\_next = (refr\_tick) ? Two\_y\_reg : Two\_X\_T ;

endmodule

Three.v

module Three\_3(clk, reset,video\_on, refr\_tick,pix\_x,pix\_y,three\_on,three\_rgb);

input clk, reset;

input video\_on, refr\_tick;

input [9:0] pix\_x;

input [9:0] pix\_y;

output three\_on;

output [2:0] three\_rgb;

//--------------------------------------------

// Three Logo

//--------------------------------------------

wire [3:0] Three\_rom\_addr;

wire [4:0] Three\_rom\_col;

reg [31:0] Three\_rom\_data;

wire Three\_rom\_bit,Three\_Logo\_on, Logo\_on ;

//--------------------------------------------

// Three Logo Positioning

//--------------------------------------------

// Logo Position left, Top boundary

localparam Three\_X\_L = 300;

localparam Three\_X\_R = 331;

localparam Three\_X\_T = 10;

localparam Three\_X\_B = 25;

localparam Three\_Logo\_x = 32;

localparam Three\_Logo\_y = 16;

// Three left, right boundary

wire [9:0] Three\_x\_l, Three\_x\_r;

// Three top, bottom boundary

wire [9:0] Three\_y\_t, Three\_y\_b;

// reg to track left, top position

reg [9:0] Three\_x\_reg, Three\_y\_reg;

wire [9:0] Three\_x\_next, Three\_y\_next;

// body

//--------------------------------------------

// Three Logo

//--------------------------------------------

always @\*

case (Three\_rom\_addr)

6'h0: Three\_rom\_data = 32'b00000000000111111111100000000000; // \*\*\*\*

6'h1: Three\_rom\_data = 33'b00000000001111111111110000000000; // \*\*\*\*\*\*

6'h2: Three\_rom\_data = 32'b00000000001110000001110000000000; // \*\*\*\*\*\*\*\*

6'h3: Three\_rom\_data = 32'b00000000001110000000000000000000; // \*\*\*\*\*\*\*\*

6'h4: Three\_rom\_data = 32'b00000000001110000000000000000000; // \*\*\*\*\*\*\*\*

6'h5: Three\_rom\_data = 32'b00000000001110000000000000000000; // \*\*\*\*\*\*\*\*

6'h6: Three\_rom\_data = 32'b00000000001110000000000000000000; // \*\*\*\*\*\*

6'h7: Three\_rom\_data = 32'b00000000000111111111100000000000; // \*\*\*\*

6'h8: Three\_rom\_data = 32'b00000000000111111111100000000000; // \*\*\*\*

6'h9: Three\_rom\_data = 32'b00000000001110000000000000000000; // \*\*\*\*\*\*

6'hA: Three\_rom\_data = 32'b00000000001110000000000000000000; // \*\*\*\*\*\*\*\*

6'hB: Three\_rom\_data = 32'b00000000001110000000000000000000; // \*\*\*\*\*\*\*\*

6'hC: Three\_rom\_data = 32'b00000000001110000000000000000000; // \*\*\*\*\*\*\*\*

6'hD: Three\_rom\_data = 32'b00000000001110000001110000000000; // \*\*\*\*\*\*\*\*

6'hE: Three\_rom\_data = 32'b00000000001111111111110000000000; // \*\*\*\*\*\*

6'hF: Three\_rom\_data = 32'b00000000000111111111100000000000; // \*\*\*\*

endcase

//------------------------------------------

// Register for Three logo

//------------------------------------------

// registers

always @(posedge clk, posedge reset)

if (reset)

begin

Three\_y\_reg <= 0;

end

else

begin

Three\_x\_reg <= Three\_x\_next;

Three\_y\_reg <= Three\_y\_next;

end

//--------------------------------------------

// Three Logo

//--------------------------------------------

// boundary

assign Three\_x\_l = Three\_x\_reg;

assign Three\_y\_t = Three\_y\_reg;

assign Three\_x\_r = Three\_x\_l + Three\_Logo\_x - 1;

assign Three\_y\_b = Three\_y\_t + Three\_Logo\_y - 1;

// pixel within logo

assign Logo\_on =

(Three\_X\_L<=pix\_x) && (pix\_x<=Three\_x\_r) &&

(Three\_X\_T<=pix\_y) && (pix\_y<=Three\_y\_b);

//assign Logo\_on =

// (Three\_x\_l<=Three\_X\_L) && (pix\_x<=Three\_x\_r) &&

// (Three\_y\_t<=Three\_X\_T) && (pix\_y<=Three\_y\_b);

// map current pixel location to ROM addr/col

assign Three\_rom\_addr = pix\_y[3:0] - Three\_y\_t[3:0];

assign Three\_rom\_col = pix\_x[4:0] - Three\_x\_l[4:0];

assign Three\_rom\_bit = Three\_rom\_data[Three\_rom\_col];

// pixel within logo

assign three\_on = (Logo\_on & Three\_rom\_bit);

assign three\_rgb = 3'b101;

// assign Three\_x\_next = (refr\_tick)? Three\_x\_reg + 1'b1:Three\_x\_reg;

// assign Three\_y\_next = (refr\_tick) ? Three\_y\_reg + 1'b1:Three\_y\_reg;

assign Three\_x\_next = (refr\_tick) ? Three\_x\_reg: Three\_X\_L;

assign Three\_y\_next = (refr\_tick) ? Three\_y\_reg : Three\_X\_T ;

endmodule

Four.v

module Four\_4(clk, reset,video\_on, refr\_tick,pix\_x,pix\_y,Four\_on,Four\_rgb);

input clk, reset;

input video\_on, refr\_tick;

input [9:0] pix\_x;

input [9:0] pix\_y;

output Four\_on;

output [2:0] Four\_rgb;

//--------------------------------------------

// Four Logo

//--------------------------------------------

wire [3:0] Four\_rom\_addr;

wire [4:0] Four\_rom\_col;

reg [31:0] Four\_rom\_data;

wire Four\_rom\_bit,Four\_Logo\_on, Logo\_on ;

//--------------------------------------------

// Four Logo Positioning

//--------------------------------------------

// Logo Position left, Top boundary

localparam Four\_X\_L = 300;

localparam Four\_X\_R = 331;

localparam Four\_X\_T = 10;

localparam Four\_X\_B = 25;

localparam Four\_Logo\_x = 32;

localparam Four\_Logo\_y = 16;

// Four left, right boundary

wire [9:0] Four\_x\_l, Four\_x\_r;

// Four top, bottom boundary

wire [9:0] Four\_y\_t, Four\_y\_b;

// reg to track left, top position

reg [9:0] Four\_x\_reg, Four\_y\_reg;

wire [9:0] Four\_x\_next, Four\_y\_next;

// body

//--------------------------------------------

// Four Logo

//--------------------------------------------

always @\*

case (Four\_rom\_addr)

6'h0: Four\_rom\_data = 32'b00000000000111111111000000000000; // \*\*\*\*

6'h1: Four\_rom\_data = 33'b00000000000111111111100000000000; // \*\*\*\*\*\*

6'h2: Four\_rom\_data = 32'b0000000000011100001111000000000; // \*\*\*\*\*\*\*\*

6'h3: Four\_rom\_data = 32'b0000000000011100000111100000000; // \*\*\*\*\*\*\*\*

6'h4: Four\_rom\_data = 32'b00000000000111000000111100000000; // \*\*\*\*\*\*\*\*

6'h5: Four\_rom\_data = 32'b00000000000111000000011110000000; // \*\*\*\*\*\*\*\*

6'h6: Four\_rom\_data = 32'b00000000000111000000001111000000; // \*\*\*\*\*\*

6'h7: Four\_rom\_data = 32'b00000111111111111111111111000000; // \*\*\*\*

6'h8: Four\_rom\_data = 32'b00000111111111111111111111000000; // \*\*\*\*

6'h9: Four\_rom\_data = 32'b00000000000111000000000000000000; // \*\*\*\*\*\*

6'hA: Four\_rom\_data = 32'b00000000000111000000000000000000; // \*\*\*\*\*\*\*\*

6'hB: Four\_rom\_data = 32'b00000000000111000000000000000000; // \*\*\*\*\*\*\*\*

6'hC: Four\_rom\_data = 32'b00000000000111000000000000000000; // \*\*\*\*\*\*\*\*

6'hD: Four\_rom\_data = 32'b00000000000111000000000000000000; // \*\*\*\*\*\*\*\*

6'hE: Four\_rom\_data = 32'b00000000000111000000000000000000; // \*\*\*\*\*\*

6'hF: Four\_rom\_data = 32'b00000000000111000000000000000000; // \*\*\*\*

endcase

//------------------------------------------

// Register for Four logo

//------------------------------------------

// registers

always @(posedge clk, posedge reset)

if (reset)

begin

Four\_y\_reg <= 0;

end

else

begin

Four\_x\_reg <= Four\_x\_next;

Four\_y\_reg <= Four\_y\_next;

end

//--------------------------------------------

// Four Logo

//--------------------------------------------

// boundary

assign Four\_x\_l = Four\_x\_reg;

assign Four\_y\_t = Four\_y\_reg;

assign Four\_x\_r = Four\_x\_l + Four\_Logo\_x - 1;

assign Four\_y\_b = Four\_y\_t + Four\_Logo\_y - 1;

// pixel within logo

assign Logo\_on =

(Four\_X\_L<=pix\_x) && (pix\_x<=Four\_x\_r) &&

(Four\_X\_T<=pix\_y) && (pix\_y<=Four\_y\_b);

//assign Logo\_on =

// (Four\_x\_l<=Four\_X\_L) && (pix\_x<=Four\_x\_r) &&

// (Four\_y\_t<=Four\_X\_T) && (pix\_y<=Four\_y\_b);

// map current pixel location to ROM addr/col

assign Four\_rom\_addr = pix\_y[3:0] - Four\_y\_t[3:0];

assign Four\_rom\_col = pix\_x[4:0] - Four\_x\_l[4:0];

assign Four\_rom\_bit = Four\_rom\_data[Four\_rom\_col];

// pixel within logo

assign Four\_on = (Logo\_on & Four\_rom\_bit);

assign Four\_rgb = 3'b101;

// assign Four\_x\_next = (refr\_tick)? Four\_x\_reg + 1'b1:Four\_x\_reg;

// assign Four\_y\_next = (refr\_tick) ? Four\_y\_reg + 1'b1:Four\_y\_reg;

assign Four\_x\_next = (refr\_tick) ? Four\_x\_reg: Four\_X\_L;

assign Four\_y\_next = (refr\_tick) ? Four\_y\_reg : Four\_X\_T ;

endmodule

Five.v

module Five\_5(clk, reset,video\_on,refr\_tick, pix\_x, pix\_y,five\_on,five\_rgb);

input clk, reset;

input video\_on, refr\_tick;

input [9:0] pix\_x, pix\_y;

output five\_on;

output [2:0] five\_rgb;

//--------------------------------------------

// Five Logo

//--------------------------------------------

wire [3:0] Five\_rom\_addr;

wire [4:0] Five\_rom\_col;

reg [31:0] Five\_rom\_data;

wire Five\_rom\_bit,Five\_Logo\_on, Logo\_on ;

//--------------------------------------------

// Five Logo Positioning

//--------------------------------------------

// Logo Position left, Top boundary

localparam Five\_X\_L = 300;

localparam Five\_X\_R = 331;

localparam Five\_X\_T = 10;

localparam Five\_X\_B = 25;

localparam Five\_Logo\_x = 32;

localparam Five\_Logo\_y = 16;

// Five left, right boundary

wire [9:0] Five\_x\_l, Five\_x\_r;

// Five top, bottom boundary

wire [9:0] Five\_y\_t, Five\_y\_b;

// reg to track left, top position

reg [9:0] Five\_x\_reg, Five\_y\_reg;

wire [9:0] Five\_x\_next, Five\_y\_next;

// body

//--------------------------------------------

// Five Logo

//--------------------------------------------

always @\*

case (Five\_rom\_addr)

6'h0: Five\_rom\_data = 32'b00000000111111111111111000000000; // \*\*\*\*

6'h1: Five\_rom\_data = 33'b00000000111111111111111000000000; // \*\*\*\*\*\*

6'h2: Five\_rom\_data = 32'b00000000000000000000111000000000; // \*\*\*\*\*\*\*\*

6'h3: Five\_rom\_data = 32'b00000000000000000000111000000000; // \*\*\*\*\*\*\*\*

6'h4: Five\_rom\_data = 32'b00000000000000000000111000000000; // \*\*\*\*\*\*\*\*

6'h5: Five\_rom\_data = 32'b00000000000000000000111000000000; // \*\*\*\*\*\*\*\*

6'h6: Five\_rom\_data = 32'b00000000000000000000111000000000; // \*\*\*\*\*\*

6'h7: Five\_rom\_data = 32'b00000000111111111111111000000000; // \*\*\*\*

6'h8: Five\_rom\_data = 32'b00000000111111111111111000000000; // \*\*\*\*

6'h9: Five\_rom\_data = 32'b00000000111000000000000000000000; // \*\*\*\*\*\*

6'hA: Five\_rom\_data = 32'b00000000111000000000000000000000; // \*\*\*\*\*\*\*\*

6'hB: Five\_rom\_data = 32'b00000000111000000000000000000000; // \*\*\*\*\*\*\*\*

6'hC: Five\_rom\_data = 32'b00000000111000000000000000000000; // \*\*\*\*\*\*\*\*

6'hD: Five\_rom\_data = 32'b00000000111000000000000000000000; // \*\*\*\*\*\*\*\*

6'hE: Five\_rom\_data = 32'b00000000111111111111111000000000; // \*\*\*\*\*\*

6'hF: Five\_rom\_data = 32'b00000000111111111111111000000000; // \*\*\*\*

endcase

//------------------------------------------

// Register for Five logo

//------------------------------------------

// registers

always @(posedge clk, posedge reset)

if (reset)

begin

Five\_y\_reg <= 0;

end

else

begin

Five\_x\_reg <= Five\_x\_next;

Five\_y\_reg <= Five\_y\_next;

end

//--------------------------------------------

// Five Logo

//--------------------------------------------

// boundary

assign Five\_x\_l = Five\_x\_reg;

assign Five\_y\_t = Five\_y\_reg;

assign Five\_x\_r = Five\_x\_l + Five\_Logo\_x - 1;

assign Five\_y\_b = Five\_y\_t + Five\_Logo\_y - 1;

// pixel within logo

assign Logo\_on =

(Five\_X\_L<=pix\_x) && (pix\_x<=Five\_x\_r) &&

(Five\_X\_T<=pix\_y) && (pix\_y<=Five\_y\_b);

//assign Logo\_on =

// (Five\_x\_l<=Five\_X\_L) && (pix\_x<=Five\_x\_r) &&

// (Five\_y\_t<=Five\_X\_T) && (pix\_y<=Five\_y\_b);

// map current pixel location to ROM addr/col

assign Five\_rom\_addr = pix\_y[3:0] - Five\_y\_t[3:0];

assign Five\_rom\_col = pix\_x[4:0] - Five\_x\_l[4:0];

assign Five\_rom\_bit = Five\_rom\_data[Five\_rom\_col];

// pixel within logo

assign five\_on = (Logo\_on & Five\_rom\_bit);

assign five\_rgb = 3'b101;

// assign Five\_x\_next = (refr\_tick)? Five\_x\_reg + 1'b1:Five\_x\_reg;

// assign Five\_y\_next = (refr\_tick) ? Five\_y\_reg + 1'b1:Five\_y\_reg;

assign Five\_x\_next = (refr\_tick) ? Five\_x\_reg: Five\_X\_L;

assign Five\_y\_next = (refr\_tick) ? Five\_y\_reg : Five\_X\_T ;

endmodule

Six.v

module Six\_6(clk, reset,video\_on, refr\_tick,pix\_x, pix\_y,six\_on,six\_rgb);

input clk, reset;

input video\_on, refr\_tick;

input [9:0] pix\_x, pix\_y;

output six\_on;

output [2:0] six\_rgb;

//--------------------------------------------

// Six Logo

//--------------------------------------------

wire [3:0] Six\_rom\_addr;

wire [4:0] Six\_rom\_col;

reg [31:0] Six\_rom\_data;

wire Six\_rom\_bit,Six\_Logo\_on, Logo\_on ;

//--------------------------------------------

// Six Logo Positioning

//--------------------------------------------

// Logo Position left, Top boundary

localparam Six\_X\_L = 300;

localparam Six\_X\_R = 331;

localparam Six\_X\_T = 10;

localparam Six\_X\_B = 25;

localparam Six\_Logo\_x = 32;

localparam Six\_Logo\_y = 16;

// Six left, right boundary

wire [9:0] Six\_x\_l, Six\_x\_r;

// Six top, bottom boundary

wire [9:0] Six\_y\_t, Six\_y\_b;

// reg to track left, top position

reg [9:0] Six\_x\_reg, Six\_y\_reg;

wire [9:0] Six\_x\_next, Six\_y\_next;

// body

//--------------------------------------------

// Six Logo

//--------------------------------------------

always @\*

case (Six\_rom\_addr)

6'h0: Six\_rom\_data = 32'b00000000000111111111100000000000; // \*\*\*\*

6'h1: Six\_rom\_data = 33'b00000000001111111111110000000000; // \*\*\*\*\*\*

6'h2: Six\_rom\_data = 32'b00000000001110000001110000000000; // \*\*\*\*\*\*\*\*

6'h3: Six\_rom\_data = 32'b00000000001110000001110000000000; // \*\*\*\*\*\*\*\*

6'h4: Six\_rom\_data = 32'b00000000000000000001110000000000; // \*\*\*\*\*\*\*\*

6'h5: Six\_rom\_data = 32'b00000000000000000001110000000000; // \*\*\*\*\*\*\*\*

6'h6: Six\_rom\_data = 32'b00000000000000000001110000000000; // \*\*\*\*\*\*

6'h7: Six\_rom\_data = 32'b00000000000111111111110000000000; // \*\*\*\*

6'h8: Six\_rom\_data = 32'b00000000000111111111110000000000; // \*\*\*\*

6'h9: Six\_rom\_data = 32'b00000000001110000001110000000000; // \*\*\*\*\*\*

6'hA: Six\_rom\_data = 32'b00000000001110000001110000000000; // \*\*\*\*\*\*\*\*

6'hB: Six\_rom\_data = 32'b00000000001110000001110000000000; // \*\*\*\*\*\*\*\*

6'hC: Six\_rom\_data = 32'b00000000001110000001110000000000; // \*\*\*\*\*\*\*\*

6'hD: Six\_rom\_data = 32'b00000000001110000001110000000000; // \*\*\*\*\*\*\*\*

6'hE: Six\_rom\_data = 32'b00000000001111111111110000000000; // \*\*\*\*\*\*

6'hF: Six\_rom\_data = 32'b00000000000111111111100000000000; // \*\*\*\*

endcase

//------------------------------------------

// Register for Six logo

//------------------------------------------

// registers

always @(posedge clk, posedge reset)

if (reset)

begin

Six\_y\_reg <= 0;

end

else

begin

Six\_x\_reg <= Six\_x\_next;

Six\_y\_reg <= Six\_y\_next;

end

//--------------------------------------------

// Six Logo

//--------------------------------------------

// boundary

assign Six\_x\_l = Six\_x\_reg;

assign Six\_y\_t = Six\_y\_reg;

assign Six\_x\_r = Six\_x\_l + Six\_Logo\_x - 1;

assign Six\_y\_b = Six\_y\_t + Six\_Logo\_y - 1;

// pixel within logo

assign Logo\_on =

(Six\_X\_L<=pix\_x) && (pix\_x<=Six\_x\_r) &&

(Six\_X\_T<=pix\_y) && (pix\_y<=Six\_y\_b);

//assign Logo\_on =

// (Six\_x\_l<=Six\_X\_L) && (pix\_x<=Six\_x\_r) &&

// (Six\_y\_t<=Six\_X\_T) && (pix\_y<=Six\_y\_b);

// map current pixel location to ROM addr/col

assign Six\_rom\_addr = pix\_y[3:0] - Six\_y\_t[3:0];

assign Six\_rom\_col = pix\_x[4:0] - Six\_x\_l[4:0];

assign Six\_rom\_bit = Six\_rom\_data[Six\_rom\_col];

// pixel within logo

assign six\_on = (Logo\_on & Six\_rom\_bit);

assign six\_rgb = 3'b101;

// assign Six\_x\_next = (refr\_tick)? Six\_x\_reg + 1'b1:Six\_x\_reg;

// assign Six\_y\_next = (refr\_tick) ? Six\_y\_reg + 1'b1:Six\_y\_reg;

assign Six\_x\_next = (refr\_tick) ? Six\_x\_reg: Six\_X\_L;

assign Six\_y\_next = (refr\_tick) ? Six\_y\_reg : Six\_X\_T ;

endmodule

Seven.v

module Seven\_7(clk, reset,video\_on, refr\_tick,pix\_x,pix\_y,Seven\_on,Seven\_rgb);

input clk, reset;

input video\_on, refr\_tick;

input [9:0] pix\_x;

input [9:0] pix\_y;

output Seven\_on;

output [2:0] Seven\_rgb;

//--------------------------------------------

// Seven Logo

//--------------------------------------------

wire [3:0] Seven\_rom\_addr;

wire [4:0] Seven\_rom\_col;

reg [31:0] Seven\_rom\_data;

wire Seven\_rom\_bit,Seven\_Logo\_on, Logo\_on ;

//--------------------------------------------

// Seven Logo Positioning

//--------------------------------------------

// Logo Position left, Top boundary

localparam Seven\_X\_L = 300;

localparam Seven\_X\_R = 331;

localparam Seven\_X\_T = 10;

localparam Seven\_X\_B = 25;

localparam Seven\_Logo\_x = 32;

localparam Seven\_Logo\_y = 16;

// Seven left, right boundary

wire [9:0] Seven\_x\_l, Seven\_x\_r;

// Seven top, bottom boundary

wire [9:0] Seven\_y\_t, Seven\_y\_b;

// reg to track left, top position

reg [9:0] Seven\_x\_reg, Seven\_y\_reg;

wire [9:0] Seven\_x\_next, Seven\_y\_next;

// body

//--------------------------------------------

// Seven Logo

//--------------------------------------------

always @\*

case (Seven\_rom\_addr)

6'h0: Seven\_rom\_data = 32'b00000011111111111111111000000000; // \*\*\*\*

6'h1: Seven\_rom\_data = 33'b00000011111111111111111000000000; // \*\*\*\*\*\*

6'h2: Seven\_rom\_data = 32'b00000001110000000000000000000000; // \*\*\*\*\*\*\*\*

6'h3: Seven\_rom\_data = 32'b00000000111000000000000000000000; // \*\*\*\*\*\*\*\*

6'h4: Seven\_rom\_data = 32'b00000000011100000000000000000000; // \*\*\*\*\*\*\*\*

6'h5: Seven\_rom\_data = 32'b00000000001110000000000000000000; // \*\*\*\*\*\*\*\*

6'h6: Seven\_rom\_data = 32'b00000000000111000000000000000000; // \*\*\*\*\*\*

6'h7: Seven\_rom\_data = 32'b00000000000011100000000000000000; // \*\*\*\*

6'h8: Seven\_rom\_data = 32'b00000000000001110000000000000000; // \*\*\*\*

6'h9: Seven\_rom\_data = 32'b00000000000000111000000000000000; // \*\*\*\*\*\*

6'hA: Seven\_rom\_data = 32'b00000000000000011100000000000000; // \*\*\*\*\*\*\*\*

6'hB: Seven\_rom\_data = 32'b00000000000000001110000000000000; // \*\*\*\*\*\*\*\*

6'hC: Seven\_rom\_data = 32'b00000000000000000111000000000000; // \*\*\*\*\*\*\*\*

6'hD: Seven\_rom\_data = 32'b00000000000000000011100000000000; // \*\*\*\*\*\*\*\*

6'hE: Seven\_rom\_data = 32'b00000000000000000001110000000000; // \*\*\*\*\*\*

6'hF: Seven\_rom\_data = 32'b00000000000000000000111000000000; // \*\*\*\*

endcase

//------------------------------------------

// Register for Seven logo

//------------------------------------------

// registers

always @(posedge clk, posedge reset)

if (reset)

begin

Seven\_y\_reg <= 0;

end

else

begin

Seven\_x\_reg <= Seven\_x\_next;

Seven\_y\_reg <= Seven\_y\_next;

end

//--------------------------------------------

// Seven Logo

//--------------------------------------------

// boundary

assign Seven\_x\_l = Seven\_x\_reg;

assign Seven\_y\_t = Seven\_y\_reg;

assign Seven\_x\_r = Seven\_x\_l + Seven\_Logo\_x - 1;

assign Seven\_y\_b = Seven\_y\_t + Seven\_Logo\_y - 1;

// pixel within logo

assign Logo\_on =

(Seven\_X\_L<=pix\_x) && (pix\_x<=Seven\_x\_r) &&

(Seven\_X\_T<=pix\_y) && (pix\_y<=Seven\_y\_b);

//assign Logo\_on =

// (Seven\_x\_l<=Seven\_X\_L) && (pix\_x<=Seven\_x\_r) &&

// (Seven\_y\_t<=Seven\_X\_T) && (pix\_y<=Seven\_y\_b);

// map current pixel location to ROM addr/col

assign Seven\_rom\_addr = pix\_y[3:0] - Seven\_y\_t[3:0];

assign Seven\_rom\_col = pix\_x[4:0] - Seven\_x\_l[4:0];

assign Seven\_rom\_bit = Seven\_rom\_data[Seven\_rom\_col];

// pixel within logo

assign Seven\_on = (Logo\_on & Seven\_rom\_bit);

assign Seven\_rgb = 3'b101;

// assign Seven\_x\_next = (refr\_tick)? Seven\_x\_reg + 1'b1:Seven\_x\_reg;

// assign Seven\_y\_next = (refr\_tick) ? Seven\_y\_reg + 1'b1:Seven\_y\_reg;

assign Seven\_x\_next = (refr\_tick) ? Seven\_x\_reg: Seven\_X\_L;

assign Seven\_y\_next = (refr\_tick) ? Seven\_y\_reg : Seven\_X\_T ;

endmodule

Eight.v

module Eight\_8(clk, reset,video\_on,refr\_tick, pix\_x, pix\_y,eight\_on,eight\_rgb);

input clk, reset;

input video\_on, refr\_tick;

input [9:0] pix\_x, pix\_y;

output eight\_on;

output [2:0] eight\_rgb;

//--------------------------------------------

// Eight Logo

//--------------------------------------------

wire [3:0] Eight\_rom\_addr;

wire [4:0] Eight\_rom\_col;

reg [31:0] Eight\_rom\_data;

wire Eight\_rom\_bit,Eight\_Logo\_on, Logo\_on ;

//--------------------------------------------

// Eight Logo Positioning

//--------------------------------------------

// Logo Position left, Top boundary

localparam Eight\_X\_L = 300;

localparam Eight\_X\_R = 331;

localparam Eight\_X\_T = 10;

localparam Eight\_X\_B = 25;

localparam Eight\_Logo\_x = 32;

localparam Eight\_Logo\_y = 16;

// Eight left, right boundary

wire [9:0] Eight\_x\_l, Eight\_x\_r;

// Eight top, bottom boundary

wire [9:0] Eight\_y\_t, Eight\_y\_b;

// reg to track left, top position

reg [9:0] Eight\_x\_reg, Eight\_y\_reg;

wire [9:0] Eight\_x\_next, Eight\_y\_next;

// body

//--------------------------------------------

// Eight Logo

//--------------------------------------------

always @\*

case (Eight\_rom\_addr)

6'h0: Eight\_rom\_data = 32'b00000000000111111111100000000000; // \*\*\*\*

6'h1: Eight\_rom\_data = 33'b00000000001111111111110000000000; // \*\*\*\*\*\*

6'h2: Eight\_rom\_data = 32'b00000000001110000001110000000000; // \*\*\*\*\*\*\*\*

6'h3: Eight\_rom\_data = 32'b00000000001110000001110000000000; // \*\*\*\*\*\*\*\*

6'h4: Eight\_rom\_data = 32'b00000000001110000001110000000000; // \*\*\*\*\*\*\*\*

6'h5: Eight\_rom\_data = 32'b00000000001110000001110000000000; // \*\*\*\*\*\*\*\*

6'h6: Eight\_rom\_data = 32'b00000000001110000001110000000000; // \*\*\*\*\*\*

6'h7: Eight\_rom\_data = 32'b00000000000111111111100000000000; // \*\*\*\*

6'h8: Eight\_rom\_data = 32'b00000000000111111111100000000000; // \*\*\*\*

6'h9: Eight\_rom\_data = 32'b00000000001110000001110000000000; // \*\*\*\*\*\*

6'hA: Eight\_rom\_data = 32'b00000000001110000001110000000000; // \*\*\*\*\*\*\*\*

6'hB: Eight\_rom\_data = 32'b00000000001110000001110000000000; // \*\*\*\*\*\*\*\*

6'hC: Eight\_rom\_data = 32'b00000000001110000001110000000000; // \*\*\*\*\*\*\*\*

6'hD: Eight\_rom\_data = 32'b00000000001110000001110000000000; // \*\*\*\*\*\*\*\*

6'hE: Eight\_rom\_data = 32'b00000000001111111111110000000000; // \*\*\*\*\*\*

6'hF: Eight\_rom\_data = 32'b00000000000111111111100000000000; // \*\*\*\*

endcase

//------------------------------------------

// Register for Eight logo

//------------------------------------------

// registers

always @(posedge clk, posedge reset)

if (reset)

begin

Eight\_y\_reg <= 0;

end

else

begin

Eight\_x\_reg <= Eight\_x\_next;

Eight\_y\_reg <= Eight\_y\_next;

end

//--------------------------------------------

// Eight Logo

//--------------------------------------------

// boundary

assign Eight\_x\_l = Eight\_x\_reg;

assign Eight\_y\_t = Eight\_y\_reg;

assign Eight\_x\_r = Eight\_x\_l + Eight\_Logo\_x - 1;

assign Eight\_y\_b = Eight\_y\_t + Eight\_Logo\_y - 1;

// pixel within logo

assign Logo\_on =

(Eight\_X\_L<=pix\_x) && (pix\_x<=Eight\_x\_r) &&

(Eight\_X\_T<=pix\_y) && (pix\_y<=Eight\_y\_b);

//assign Logo\_on =

// (Eight\_x\_l<=Eight\_X\_L) && (pix\_x<=Eight\_x\_r) &&

// (Eight\_y\_t<=Eight\_X\_T) && (pix\_y<=Eight\_y\_b);

// map current pixel location to ROM addr/col

assign Eight\_rom\_addr = pix\_y[3:0] - Eight\_y\_t[3:0];

assign Eight\_rom\_col = pix\_x[4:0] - Eight\_x\_l[4:0];

assign Eight\_rom\_bit = Eight\_rom\_data[Eight\_rom\_col];

// pixel within logo

assign eight\_on = (Logo\_on & Eight\_rom\_bit);

assign eight\_rgb = 3'b101;

// assign Eight\_x\_next = (refr\_tick)? Eight\_x\_reg + 1'b1:Eight\_x\_reg;

// assign Eight\_y\_next = (refr\_tick) ? Eight\_y\_reg + 1'b1:Eight\_y\_reg;

assign Eight\_x\_next = (refr\_tick) ? Eight\_x\_reg: Eight\_X\_L;

assign Eight\_y\_next = (refr\_tick) ? Eight\_y\_reg : Eight\_X\_T ;

endmodule

Nine.v

module Nine\_9(clk, reset,video\_on, refr\_tick,pix\_x,pix\_y,Nine\_on,Nine\_rgb);

input clk, reset;

input video\_on, refr\_tick;

input [9:0] pix\_x;

input [9:0] pix\_y;

output Nine\_on;

output [2:0] Nine\_rgb;

//--------------------------------------------

// Nine Logo

//--------------------------------------------

wire [3:0] Nine\_rom\_addr;

wire [4:0] Nine\_rom\_col;

reg [31:0] Nine\_rom\_data;

wire Nine\_rom\_bit,Nine\_Logo\_on, Logo\_on ;

//--------------------------------------------

// Nine Logo Positioning

//--------------------------------------------

// Logo Position left, Top boundary

localparam Nine\_X\_L = 300;

localparam Nine\_X\_R = 331;

localparam Nine\_X\_T = 10;

localparam Nine\_X\_B = 25;

localparam Nine\_Logo\_x = 32;

localparam Nine\_Logo\_y = 16;

// Nine left, right boundary

wire [9:0] Nine\_x\_l, Nine\_x\_r;

// Nine top, bottom boundary

wire [9:0] Nine\_y\_t, Nine\_y\_b;

// reg to track left, top position

reg [9:0] Nine\_x\_reg, Nine\_y\_reg;

wire [9:0] Nine\_x\_next, Nine\_y\_next;

// body

//--------------------------------------------

// Nine Logo

//--------------------------------------------

always @\*

case (Nine\_rom\_addr)

6'h0: Nine\_rom\_data = 32'b00000000000011111110000000000000; // \*\*\*\*

6'h1: Nine\_rom\_data = 33'b00000000000111100111110000000000; // \*\*\*\*\*\*

6'h2: Nine\_rom\_data = 32'b00000000001110000001110000000000; // \*\*\*\*\*\*\*\*

6'h3: Nine\_rom\_data = 32'b00000000011100000000110000000000; // \*\*\*\*\*\*\*\*

6'h4: Nine\_rom\_data = 32'b00000000011100000000110000000000; // \*\*\*\*\*\*\*\*

6'h5: Nine\_rom\_data = 32'b00000000011110000011110000000000; // \*\*\*\*\*\*\*\*

6'h6: Nine\_rom\_data = 32'b00000000011111100111110000000000; // \*\*\*\*\*\*

6'h7: Nine\_rom\_data = 32'b00000000011111111110000000000000; // \*\*\*\*

6'h8: Nine\_rom\_data = 32'b00000000011100000000000000000000; // \*\*\*\*

6'h9: Nine\_rom\_data = 32'b00000000011100000000000000000000; // \*\*\*\*\*\*

6'hA: Nine\_rom\_data = 32'b00000000011100000000000000000000; // \*\*\*\*\*\*\*\*

6'hB: Nine\_rom\_data = 32'b00000000011100000000000000000000; // \*\*\*\*\*\*\*\*

6'hC: Nine\_rom\_data = 32'b00000000011100000000000000000000; // \*\*\*\*\*\*\*\*

6'hD: Nine\_rom\_data = 32'b00000000001110000000110000000000; // \*\*\*\*\*\*\*\*

6'hE: Nine\_rom\_data = 32'b00000000000111111111110000000000; // \*\*\*\*\*\*

6'hF: Nine\_rom\_data = 32'b00000000000011111111000000000000; // \*\*\*\*

endcase

//------------------------------------------

// Register for Nine logo

//------------------------------------------

// registers

always @(posedge clk, posedge reset)

if (reset)

begin

Nine\_y\_reg <= 0;

end

else

begin

Nine\_x\_reg <= Nine\_x\_next;

Nine\_y\_reg <= Nine\_y\_next;

end

//--------------------------------------------

// Nine Logo

//--------------------------------------------

// boundary

assign Nine\_x\_l = Nine\_x\_reg;

assign Nine\_y\_t = Nine\_y\_reg;

assign Nine\_x\_r = Nine\_x\_l + Nine\_Logo\_x - 1;

assign Nine\_y\_b = Nine\_y\_t + Nine\_Logo\_y - 1;

// pixel within logo

assign Logo\_on =

(Nine\_X\_L<=pix\_x) && (pix\_x<=Nine\_x\_r) &&

(Nine\_X\_T<=pix\_y) && (pix\_y<=Nine\_y\_b);

//assign Logo\_on =

// (Nine\_x\_l<=Nine\_X\_L) && (pix\_x<=Nine\_x\_r) &&

// (Nine\_y\_t<=Nine\_X\_T) && (pix\_y<=Nine\_y\_b);

// map current pixel location to ROM addr/col

assign Nine\_rom\_addr = pix\_y[3:0] - Nine\_y\_t[3:0];

assign Nine\_rom\_col = pix\_x[4:0] - Nine\_x\_l[4:0];

assign Nine\_rom\_bit = Nine\_rom\_data[Nine\_rom\_col];

// pixel within logo

assign Nine\_on = (Logo\_on & Nine\_rom\_bit);

assign Nine\_rgb = 3'b101;

// assign Nine\_x\_next = (refr\_tick)? Nine\_x\_reg + 1'b1:Nine\_x\_reg;

// assign Nine\_y\_next = (refr\_tick) ? Nine\_y\_reg + 1'b1:Nine\_y\_reg;

assign Nine\_x\_next = (refr\_tick) ? Nine\_x\_reg: Nine\_X\_L;

assign Nine\_y\_next = (refr\_tick) ? Nine\_y\_reg : Nine\_X\_T ;

endmodule

Seven\_Segment.v

module Seven\_Segment(clk, reset, An, Disp);

input clk, reset;

output [3:0] An;

//output S\_reset;

output [7:0] Disp;

reg [3:0] state\_reg, state\_next;

parameter S0 = 4'b0000;

parameter S1 = 4'b0001;

parameter S2 = 4'b0010;

parameter S3 = 4'b0011;

parameter S4 = 4'b0100;

parameter S5 = 4'b0101;

parameter S6 = 4'b0110;

parameter S7 = 4'b0111;

parameter S8 = 4'b1000;

parameter S9 = 4'b1001;

//Clk\_divider c0 (.Clk(clk), .Reset(reset), .Slow\_clk(s\_clk));

always @(posedge clk, posedge reset)

if (reset)

state\_reg <= S0;

else

state\_reg <= state\_next;

always@(\*)

case (state\_reg)

S0: state\_next = S1;

S1: state\_next = S2;

S2: state\_next = S3;

S3: state\_next = S4;

S4: state\_next = S5;

S5: state\_next = S6;

S6: state\_next = S7;

S7: state\_next = S8;

S8: state\_next = S9;

S9: state\_next = S0;

default: state\_next= S0;

endcase

assign Disp = (state\_reg==S0) ? 8'b11000000:

(state\_reg==S1) ? 8'b11111001:

(state\_reg==S2) ? 8'b10100100:

(state\_reg==S3) ? 8'b10110000:

(state\_reg==S4) ? 8'b10011001:

(state\_reg==S5) ? 8'b10010010:

(state\_reg==S6) ? 8'b10000010:

(state\_reg==S7) ? 8'b11111000:

(state\_reg==S8) ? 8'b10000000:

(state\_reg==S9) ? 8'b10011000:

8'b11000000;

//assign S\_reset = (state\_reg==S9 && state\_next==S0) ? 1'b1: 1'b0;

assign An = 4'b0111;

endmodule

vga\_sync.v

// Listing 13.1

module vga\_sync

(

input wire clk, reset,

output wire hsync, vsync, video\_on, p\_tick,

output wire [9:0] pixel\_x, pixel\_y

);

// constant declaration

// VGA 640-by-480 sync parameters

localparam HD = 640; // horizontal display area

localparam HF = 48 ; // h. front (left) border

localparam HB = 16 ; // h. back (right) border

localparam HR = 96 ; // h. retrace

localparam VD = 480; // vertical display area

localparam VF = 10; // v. front (top) border

localparam VB = 33; // v. back (bottom) border

localparam VR = 2; // v. retrace

// mod-2 counter

reg mod2\_reg;

wire mod2\_next;

// sync counters

reg [9:0] h\_count\_reg, h\_count\_next;

reg [9:0] v\_count\_reg, v\_count\_next;

// output buffer

reg v\_sync\_reg, h\_sync\_reg;

wire v\_sync\_next, h\_sync\_next;

// status signal

wire h\_end, v\_end, pixel\_tick;

// body

// registers

always @(posedge clk, posedge reset)

if (reset)

begin

mod2\_reg <= 2'b00;

v\_count\_reg <= 0;

h\_count\_reg <= 0;

v\_sync\_reg <= 1'b0;

h\_sync\_reg <= 1'b0;

end

else

begin

mod2\_reg <= mod2\_next;

v\_count\_reg <= v\_count\_next;

h\_count\_reg <= h\_count\_next;

v\_sync\_reg <= v\_sync\_next;

h\_sync\_reg <= h\_sync\_next;

end

// mod-2 circuit to generate 25 MHz enable tick

assign mod2\_next = ~mod2\_reg;

// assign mod2\_next = mod2\_reg + 1'b1;

assign pixel\_tick = mod2\_reg;

// status signals

// end of horizontal counter (799)

assign h\_end = (h\_count\_reg==(HD+HF+HB+HR-1));

// end of vertical counter (524)

assign v\_end = (v\_count\_reg==(VD+VF+VB+VR-1));

// next-state logic of mod-800 horizontal sync counter

always @\*

if (pixel\_tick) // 25 MHz pulse

if (h\_end)

h\_count\_next = 0;

else

h\_count\_next = h\_count\_reg + 1;

else

h\_count\_next = h\_count\_reg;

// next-state logic of mod-525 vertical sync counter

always @\*

if (pixel\_tick & h\_end)

if (v\_end)

v\_count\_next = 0;

else

v\_count\_next = v\_count\_reg + 1;

else

v\_count\_next = v\_count\_reg;

// horizontal and vertical sync, buffered to avoid glitch

// h\_sync\_next asserted between 656 and 751

assign h\_sync\_next = (h\_count\_reg>=(HD+HB) &&

h\_count\_reg<=(HD+HB+HR-1));

// vh\_sync\_next asserted between 490 and 491

assign v\_sync\_next = (v\_count\_reg>=(VD+VB) &&

v\_count\_reg<=(VD+VB+VR-1));

// video on/off

assign video\_on = (h\_count\_reg<HD) && (v\_count\_reg<VD);

// output

assign hsync = h\_sync\_reg;

assign vsync = v\_sync\_reg;

assign pixel\_x = h\_count\_reg;

assign pixel\_y = v\_count\_reg;

assign p\_tick = pixel\_tick;

endmodule

ucf file

# Xilinx Spartan3 Starter Board (Rev E) UCF file

# by Brian Szmyd

# Updated by Wimbo

# Clock

Net "clk" LOC = "T9";

NET "clk" PERIOD = 20ns; # 20ns = 50Mhz

#Net "Socket" LOC = "D9";

#NET "Socket" PERIOD = ??????? ns;

# User Switches

NET "reset" LOC = "F12";

#NET "sel<0>" LOC = "G12";

#NET "sel<1>" LOC = "H14";

#NET "SW3" LOC = "H13";

#NET "SW4" LOC = "J14";

#NET "SW5" LOC = "J13";

#NET "SW6" LOC = "K14";

#NET "SW7" LOC = "K13";

# User Buttons

NET "btn<0>" LOC = "M13";

#NET "BTN1" LOC = "M14";

#NET "BTN2" LOC = "L13";

NET "btn<1>" LOC = "L14";

# LEDs

#NET "LD0" LOC = "K12";

#NET "LD1" LOC = "P14";

#NET "LD2" LOC = "L12";

#NET "LD3" LOC = "N14";

#NET "LD4" LOC = "P13";

#NET "LD5" LOC = "N12";

#NET "LD6" LOC = "P12";

#NET "LD7" LOC = "P11";

# 7 Segment

NET "An<0>" LOC = "D14";

NET "An<1>" LOC = "G14";

NET "An<2>" LOC = "F14";

NET "An<3>" LOC = "E13";

NET "Disp<0>" LOC = "E14";

NET "Disp<1>" LOC = "G13";

NET "Disp<2>" LOC = "N15";

NET "Disp<3>" LOC = "P15";

NET "Disp<4>" LOC = "R16";

NET "Disp<5>" LOC = "F13";

NET "Disp<6>" LOC = "N16";

NET "Disp<7>" LOC = "P16";

# VGA Signals

NET "rgb<2>" LOC = "R12";

NET "rgb<1>" LOC = "T12";

NET "rgb<0>" LOC = "R11";

NET "hsync" LOC = "R9";

NET "vsync" LOC = "T10";

pong\_top.v

// Listing 13.6

module pong\_top\_an

(

input wire clk, reset,

input wire [1:0] btn,

output wire hsync, vsync,

output wire [3:0] An,

output wire [7:0] Disp,

output wire [2:0] rgb

);

// signal declaration

wire [9:0] pixel\_x, pixel\_y;

wire video\_on, pixel\_tick;

reg [2:0] rgb\_reg;

wire [2:0] rgb\_next;

// body

// instantiate vga sync circuit

vga\_sync vsync\_unit

(.clk(clk), .reset(reset), .hsync(hsync), .vsync(vsync),

.video\_on(video\_on), .p\_tick(pixel\_tick),

.pixel\_x(pixel\_x), .pixel\_y(pixel\_y));

// instantiate graphic generator

pong\_graph\_animate pong\_graph\_an\_unit

(.clk(clk), .reset(reset), .btn(btn),

.video\_on(video\_on), .pix\_x(pixel\_x),

.pix\_y(pixel\_y), .An(An), .Disp(Disp), .graph\_rgb(rgb\_next));

// rgb buffer

always @(posedge clk)

if (pixel\_tick)

rgb\_reg <= rgb\_next;

// output

assign rgb = rgb\_reg;

endmodule

debounce.v

module debounce(clk,btn,btn\_clr);

input clk;

input btn;

output reg btn\_clr;

parameter delay = 6500; //6.5ms delay

integer count=0;

reg xnew=0;

always @(posedge clk)

if (btn != xnew)

begin

xnew <= btn;

count <= 0;

end

else if (count == delay) btn\_clr <= xnew;

else count <= count + 1;

endmodule

pong\_graph\_animate.v

/ Listing 13.5

module pong\_graph\_animate

(

input wire clk, reset,

input wire video\_on,

input wire [1:0] btn,

input wire [9:0] pix\_x, pix\_y,

output wire [3:0] An,

output wire [7:0] Disp,

output reg [2:0] graph\_rgb

);

reg [3:0] counter = 4'b0000;

// constant and signal declaration

// x, y coordinates (0,0) to (639,479)

localparam MAX\_X = 640;

localparam MAX\_Y = 480;

wire refr\_tick;

//--------------------------------------------

// vertical stripe as a wall

//--------------------------------------------

// wall left, right boundary

localparam WALL\_X\_L = 32;

localparam WALL\_X\_R = 35;

//--------------------------------------------

// right vertical bar

//--------------------------------------------

// bar left, right boundary

localparam BAR\_X\_L = 600;

localparam BAR\_X\_R = 603;

// bar top, bottom boundary

wire [9:0] bar\_y\_t, bar\_y\_b;

localparam BAR\_Y\_SIZE = 72;

// register to track top boundary (x position is fixed)

reg [9:0] bar\_y\_reg, bar\_y\_next;

// bar moving velocity when a button is pressed

localparam BAR\_V = 4;

//--------------------------------------------

// square ball

//--------------------------------------------

localparam BALL\_SIZE = 8;

// ball left, right boundary

wire [9:0] ball\_x\_l, ball\_x\_r;

// ball top, bottom boundary

wire [9:0] ball\_y\_t, ball\_y\_b;

// reg to track left, top position

reg [9:0] ball\_x\_reg, ball\_y\_reg;

wire [9:0] ball\_x\_next, ball\_y\_next;

// reg to track ball speed

reg [9:0] x\_delta\_reg, x\_delta\_next;

reg [9:0] y\_delta\_reg, y\_delta\_next;

// ball velocity can be pos or neg)

localparam BALL\_V\_P = 2;

localparam BALL\_V\_N = -2;

//--------------------------------------------

// round ball

//--------------------------------------------

wire [2:0] rom\_addr, rom\_col;

reg [7:0] rom\_data;

wire rom\_bit;

//--------------------------------------------

// Score Count Clock

//--------------------------------------------

wire score\_clk;

wire d\_clk;

//--------------------------------------------

// Score Count Reset

//--------------------------------------------

wire score\_reset;

//--------------------------------------------

// object output signals

//--------------------------------------------

wire wall\_on, bar\_on, sq\_ball\_on, Score\_on,rd\_ball\_on, Zero\_Logo\_on,One\_Logo\_on, Two\_Logo\_on, Three\_Logo\_on;

wire [2:0] wall\_rgb, bar\_rgb, ball\_rgb, Zero\_rgb, One\_rgb, Two\_rgb,Three\_rgb, Score\_rgb;

// body

//--------------------------------------------

// round ball image ROM

//--------------------------------------------

always @\*

case (rom\_addr)

3'h0: rom\_data = 8'b00111100; // \*\*\*\*

3'h1: rom\_data = 8'b01111110; // \*\*\*\*\*\*

3'h2: rom\_data = 8'b11111111; // \*\*\*\*\*\*\*\*

3'h3: rom\_data = 8'b11111111; // \*\*\*\*\*\*\*\*

3'h4: rom\_data = 8'b11111111; // \*\*\*\*\*\*\*\*

3'h5: rom\_data = 8'b11111111; // \*\*\*\*\*\*\*\*

3'h6: rom\_data = 8'b01111110; // \*\*\*\*\*\*

3'h7: rom\_data = 8'b00111100; // \*\*\*\*

endcase

Zero\_0 Z0 (.clk(clk), .reset(reset),.video\_on(video\_on), .refr\_tick(refr\_tick), .pix\_x(pix\_x),.pix\_y(pix\_y), .zero\_on(Zero\_Logo\_on),.zero\_rgb(Zero\_rgb));

One\_1 Z1 (.clk(clk), .reset(reset),.video\_on(video\_on), .refr\_tick(refr\_tick), .pix\_x(pix\_x),.pix\_y(pix\_y), .one\_on(One\_Logo\_on),.one\_rgb(One\_rgb));

Two\_2 Z2 (.clk(clk), .reset(reset),.video\_on(video\_on), .refr\_tick(refr\_tick), .pix\_x(pix\_x),.pix\_y(pix\_y), .two\_on(Two\_Logo\_on),.two\_rgb(Two\_rgb));

Three\_3 Z3 (.clk(clk), .reset(reset),.video\_on(video\_on), .refr\_tick(refr\_tick), .pix\_x(pix\_x),.pix\_y(pix\_y), .three\_on(Three\_Logo\_on),.three\_rgb(Three\_rgb));

Four\_4 Z4 (.clk(clk), .reset(reset),.video\_on(video\_on), .refr\_tick(refr\_tick), .pix\_x(pix\_x),.pix\_y(pix\_y), .Four\_on(Four\_Logo\_on),.Four\_rgb(Four\_rgb));

Five\_5 Z5 (.clk(clk), .reset(reset),.video\_on(video\_on), .refr\_tick(refr\_tick), .pix\_x(pix\_x),.pix\_y(pix\_y), .five\_on(Five\_Logo\_on),.five\_rgb(Five\_rgb));

Six\_6 Z6 (.clk(clk), .reset(reset),.video\_on(video\_on), .refr\_tick(refr\_tick), .pix\_x(pix\_x),.pix\_y(pix\_y), .six\_on(Six\_Logo\_on),.six\_rgb(Six\_rgb));

Seven\_7 Z7 (.clk(clk), .reset(reset),.video\_on(video\_on), .refr\_tick(refr\_tick), .pix\_x(pix\_x),.pix\_y(pix\_y), .Seven\_on(Seven\_Logo\_on),.Seven\_rgb(Seven\_rgb));

Eight\_8 Z8 (.clk(clk), .reset(reset),.video\_on(video\_on), .refr\_tick(refr\_tick), .pix\_x(pix\_x),.pix\_y(pix\_y), .eight\_on(Eight\_Logo\_on),.eight\_rgb(Eight\_rgb));

Nine\_9 Z9 (.clk(clk), .reset(reset),.video\_on(video\_on), .refr\_tick(refr\_tick), .pix\_x(pix\_x),.pix\_y(pix\_y), .Nine\_on(Nine\_Logo\_on),.Nine\_rgb(Nine\_rgb));

assign Score\_on = (counter==4'b0000) ? Zero\_Logo\_on:

(counter==4'b0001) ? One\_Logo\_on:

(counter==4'b0010) ? Two\_Logo\_on:

(counter==4'b0011) ? Three\_Logo\_on:

(counter==4'b0100) ? Four\_Logo\_on:

(counter==4'b0101) ? Five\_Logo\_on:

(counter==4'b0110) ? Six\_Logo\_on:

(counter==4'b0111) ? Seven\_Logo\_on:

(counter==4'b1000) ? Eight\_Logo\_on:

(counter==4'b1001) ? Nine\_Logo\_on :

Zero\_Logo\_on;

assign Score\_rgb = (counter==4'b0000) ? Zero\_rgb:

(counter==4'b0001) ? One\_rgb:

(counter==4'b0010) ? Two\_rgb:

(counter==4'b0011) ? Three\_rgb:

(counter==4'b0100) ? Four\_rgb:

(counter==4'b0101) ? Five\_rgb:

(counter==4'b0110) ? Six\_rgb:

(counter==4'b0111) ? Seven\_rgb:

(counter==4'b1000) ? Eight\_rgb:

(counter==4'b1001) ? Nine\_rgb :

Zero\_rgb;

// registers

always @(posedge clk, posedge reset)

if (reset)

begin

bar\_y\_reg <= 0;

ball\_x\_reg <= 0;

ball\_y\_reg <= 0;

x\_delta\_reg <= 10'h004;

y\_delta\_reg <= 10'h004;

end

else

begin

bar\_y\_reg <= bar\_y\_next;

ball\_x\_reg <= ball\_x\_next;

ball\_y\_reg <= ball\_y\_next;

x\_delta\_reg <= x\_delta\_next;

y\_delta\_reg <= y\_delta\_next;

end

// refr\_tick: 1-clock tick asserted at start of v-sync

// i.e., when the screen is refreshed (60 Hz)

assign refr\_tick = (pix\_y==481) && (pix\_x==0);

//--------------------------------------------

// (wall) left vertical strip

//--------------------------------------------

// pixel within wall

assign wall\_on = (WALL\_X\_L<=pix\_x) && (pix\_x<=WALL\_X\_R);

// wall rgb output

assign wall\_rgb = 3'b001; // blue

//--------------------------------------------

// right vertical bar

//--------------------------------------------

// boundary

assign bar\_y\_t = bar\_y\_reg;

assign bar\_y\_b = bar\_y\_t + BAR\_Y\_SIZE - 1;

// pixel within bar

assign bar\_on = (BAR\_X\_L<=pix\_x) && (pix\_x<=BAR\_X\_R) &&

(bar\_y\_t<=pix\_y) && (pix\_y<=bar\_y\_b);

// bar rgb output

assign bar\_rgb = 3'b010; // green

// new bar y-position

always @\*

begin

bar\_y\_next = bar\_y\_reg; // no move

if (refr\_tick)

if (btn[1] & (bar\_y\_b < (MAX\_Y-1-BAR\_V)))

bar\_y\_next = bar\_y\_reg + BAR\_V; // move down

else if (btn[0] & (bar\_y\_t > BAR\_V))

bar\_y\_next = bar\_y\_reg - BAR\_V; // move up

end

//--------------------------------------------

// square ball

//--------------------------------------------

// boundary

assign ball\_x\_l = ball\_x\_reg;

assign ball\_y\_t = ball\_y\_reg;

assign ball\_x\_r = ball\_x\_l + BALL\_SIZE - 1;

assign ball\_y\_b = ball\_y\_t + BALL\_SIZE - 1;

// pixel within ball

assign sq\_ball\_on =

(ball\_x\_l<=pix\_x) && (pix\_x<=ball\_x\_r) &&

(ball\_y\_t<=pix\_y) && (pix\_y<=ball\_y\_b);

// map current pixel location to ROM addr/col

assign rom\_addr = pix\_y[2:0] - ball\_y\_t[2:0];

assign rom\_col = pix\_x[2:0] - ball\_x\_l[2:0];

assign rom\_bit = rom\_data[rom\_col];

// pixel within ball

assign rd\_ball\_on = sq\_ball\_on & rom\_bit;

// ball rgb output

assign ball\_rgb = 3'b100; // red

// new ball position

assign ball\_x\_next = (refr\_tick) ? ball\_x\_reg+x\_delta\_reg :

ball\_x\_reg ;

assign ball\_y\_next = (refr\_tick) ? ball\_y\_reg+y\_delta\_reg :

ball\_y\_reg ;

// new ball velocity

always @\*

begin

x\_delta\_next = x\_delta\_reg;

y\_delta\_next = y\_delta\_reg;

if (ball\_y\_t < 1) // reach top

y\_delta\_next = BALL\_V\_P;

else if (ball\_y\_b > (MAX\_Y-1)) // reach bottom

y\_delta\_next = BALL\_V\_N;

else if (ball\_x\_l <= WALL\_X\_R) // reach wall

x\_delta\_next = BALL\_V\_P; // bounce back

else if ((BAR\_X\_L<=ball\_x\_r) && (ball\_x\_r<=BAR\_X\_R) &&

(bar\_y\_t<=ball\_y\_b) && (ball\_y\_t<=bar\_y\_b))

begin

//counter = counter+1; //asdasdsadsad

// reach x of right bar and hit, ball bounce back

x\_delta\_next = BALL\_V\_N;

end

end

//--------------------------------------------

// Score clock gemerated here

//--------------------------------------------

assign score\_clk = ((BAR\_X\_L<=ball\_x\_r) && (ball\_x\_r<=BAR\_X\_R) &&

(bar\_y\_t<=ball\_y\_b) && (ball\_y\_t<=bar\_y\_b)) ? 1'b1: 1'b0;

//--------------------------------------------

// Debounce for ball hitting the paddle

//--------------------------------------------

debounce db (.clk(clk),.btn(score\_clk),.btn\_clr(d\_clk));

Seven\_Segment ss (.clk(d\_clk), .reset(reset), .An(An), .Disp(Disp));

always @(posedge d\_clk, posedge reset)

begin

if (reset)

counter = 4'b0000;

else

if (counter < 9)

counter = counter + 1;

else

counter = 4'b0000;

end

//--------------------------------------------

// rgb multiplexing circuit

//--------------------------------------------

always @\*

if (~video\_on)

graph\_rgb = 3'b000; // blank

else

if (wall\_on)

graph\_rgb = wall\_rgb;

else if (bar\_on)

graph\_rgb = bar\_rgb;

else if (rd\_ball\_on)

graph\_rgb = ball\_rgb;

else if (Score\_on)

graph\_rgb = Score\_rgb;

else

graph\_rgb = 3'b110; // yellow background

endmodule

















