```
`timescale 1ns / 1ps

// VGA Pong v1.0     Chris Fallin <cfallin@nd.edu>

module box_detect(vpos, hpos, v, vsize, h, hsize, q);
   input [9:0] vpos;
   input [9:0] hpos;
   input [9:0] v;
   input [9:0] vsize;
   input [9:0] h;
   input [9:0] hsize;
   output q;

  wire h_ok;
  wire v_ok;

  assign q = h_ok & v_ok;

  assign v_ok =
    ( vpos >= v ) &
    ( vpos < v + vsize);

  assign h_ok =
    ( hpos >= h ) &
    ( hpos < h + hsize);

endmodule
```

```
`timescale 1ns / 1ps

// VGA Pong v1.0     Chris Fallin <cfallin@nd.edu>

module kbd(clk, reset,
        ps2_clk, ps2_data,
        paddle1_down, paddle1_up, paddle2_down, paddle2_up);
  input clk;
  input reset;
  input ps2_clk;
  input ps2_data;
  output paddle1_down;
  output paddle1_up;
  output paddle2_down;
  output paddle2_up;

  wire ps2_clk;
  wire ps2_data;

  reg paddle1_down;
  reg paddle1_up;
  reg paddle2_down;
  reg paddle2_up;
  initial paddle1_down = 0;
  initial paddle1_up = 0;
  initial paddle2_down = 0;
  initial paddle2_up = 0;

  reg [10:0] shiftreg;
  reg [3:0] bitcount;
  initial bitcount = 0;
  initial shiftreg = 0;

  wire [7:0] kbd_data;
  assign kbd_data = shiftreg[8:1];

  reg keyup_prefix;
  initial keyup_prefix = 0;

  reg byte_complete;
  initial byte_complete = 0;

  reg last_ps2_clk;
  initial last_ps2_clk = 1;

  always @(posedge clk or posedge reset) begin
    if(reset) begin
      shiftreg <= 0;
      bitcount <= 0;
      keyup_prefix <= 0;
      byte_complete <= 0;
      paddle1_up <= 0;
      paddle1_down <= 0;
      paddle2_up <= 0;
      paddle2_down <= 0;
      last_ps2_clk <= 1;
    end
    else begin

      last_ps2_clk <= ps2_clk;
      if(last_ps2_clk == 1 && ps2_clk == 0) begin
        shiftreg <= (shiftreg >> 1) | (ps2_data << 10);
        if(bitcount == 10) byte_complete <= 1;
        if(bitcount == 11)
          bitcount <= 1;
```

```
        else
          bitcount <= bitcount + 1;
      end


      if(byte_complete) begin
        byte_complete <= 0;

        // process keycode
        if(kbd_data == 8'hF0)
          keyup_prefix <= 1;
        else begin
          keyup_prefix <= 0;
          if(kbd_data == 8'h1D) // W: paddle 1 up
            paddle1_up <= ~keyup_prefix;
          if(kbd_data == 8'h1B) // S: paddle 1 down
            paddle1_down <= ~keyup_prefix;
          if(kbd_data == 8'h44) // O: paddle 2 up
            paddle2_up <= ~keyup_prefix;
          if(kbd_data == 8'h4B) // L: paddle 2 down
            paddle2_down <= ~keyup_prefix;
        end

      end // byte_complete

    end // posedge clk

  end

endmodule
```

```
‘timescale 1ns / 1ps

// VGA Pong v1.0     Chris Fallin <cfallin@nd.edu>

module motion(
  clk,
  paddle1_up,
  paddle1_down,
  paddle2_up,
  paddle2_down,
  paddle1_v,
  paddle2_v,
  ball_v_dir,
  ball_h_dir,
  ball_v,
  ball_h,
  game, reset);

   input clk;
  input paddle1_up;
   input paddle1_down;
   input paddle2_up;
   input paddle2_down;
  output ball_v_dir;
  output ball_h_dir;
   output [9:0] paddle1_v;
   output [9:0] paddle2_v;
   output [9:0] ball_v;
   output [9:0] ball_h;
   output game;
  input reset;

  reg [9:0] paddle1_v;
  reg [9:0] paddle2_v;
  reg [9:0] ball_v;
  reg [9:0] ball_h;
  reg ball_v_dir;
  reg ball_h_dir;
  reg game; // game over

  wire paddle1_inrange;
  assign paddle1_inrange = (ball_v + 16) >= paddle1_v && ball_v < (paddle1_v + 48);

  wire paddle2_inrange;
  assign paddle2_inrange = (ball_v + 16) >= paddle2_v && ball_v < (paddle2_v + 48);

  // these collision-detections leave 1-pixel buffer because of 1-cycle delay on bounces

  wire collide_top;
  assign collide_top = (ball_v == 1) && (ball_v_dir == 0);

  wire collide_bottom;
  assign collide_bottom = ((ball_v + 16) == 479) && (ball_v_dir == 1);

  wire collide_side1;
  assign collide_side1 = ball_h == (16 + 1) && (ball_h_dir == 0);

  wire collide_side2;
  assign collide_side2 = (ball_h + 16) == (640 - 16 - 1) && (ball_h_dir == 1);

  initial begin
    // keep in sync with reset block below
    paddle1_v <= 480 / 2;
    paddle2_v <= 280;
    ball_v <= 480 / 2;
```

```
    ball_h <= 480 / 2;
    ball_v_dir <= 1;
    ball_h_dir <= 1;
    game <= 0;
  end

  always @(posedge reset or posedge clk) begin

    if(reset) begin
      paddle1_v <= 480 / 2;
      paddle2_v <= 280; // so the ball gets the first bounce
      ball_v <= 480 / 2;
      ball_h <= 480 / 2;
      ball_v_dir <= 1;
      ball_h_dir <= 1;
      game <= 0;
    end

    else if(~game) begin

      if(collide_top | collide_bottom)
        ball_v_dir <= ~ball_v_dir;

      if(collide_side1) begin
        if(paddle1_inrange)
          ball_h_dir <= ~ball_h_dir;
        else
          game <= 1;
      end
      else if(collide_side2) begin
        if(paddle2_inrange)
          ball_h_dir <= ~ball_h_dir;
        else
          game <= 1;
      end

      if(ball_v_dir)
        ball_v <= ball_v + 1;
      else
        ball_v <= ball_v - 1;

      if(ball_h_dir)
        ball_h <= ball_h + 1;
      else
        ball_h <= ball_h - 1;

      if(paddle1_up)
        paddle1_v <= paddle1_v - 1;
      else if(paddle1_down)
        paddle1_v <= paddle1_v + 1;

      if(paddle2_up)
        paddle2_v <= paddle2_v - 1;
      else if(paddle2_down)
        paddle2_v <= paddle2_v + 1;

    end

  end

endmodule
```

```
`timescale 1ns / 1ps

// VGA Pong v1.0     Chris Fallin <cfallin@nd.edu>

module motion_clock(clk_50MHz, clk_motion);
   input clk_50MHz;
   output clk_motion;

  reg [17:0] cnt;

  initial cnt = 0;

  always @(posedge clk_50MHz) cnt <= cnt + 1;

  assign clk_motion = cnt[17];
endmodule
```

```
NET "reset" LOC = "p69" ;
NET "game" LOC = "p15" ;

NET "clk_50MHz" LOC = "p36" ;

NET "ps2_clk" LOC = "p85" ;
NET "ps2_data" LOC = "p86" ;

NET "vga_hsync" LOC = "p41" ;
NET "vga_vsync" LOC = "p40" ;
NET "vga_r" LOC = "p49" ;
NET "vga_g" LOC = "p47" ;
NET "vga_b" LOC = "p48" ;

NET "paddle1_up" LOC = "p5" ;
NET "paddle1_down" LOC = "p4" ;
NET "paddle2_up" LOC = "p3" ;
NET "paddle2_down" LOC = "p2" ;
```

```
`timescale 1ns / 1ps

// VGA Pong v1.0     Chris Fallin <cfallin@nd.edu>

module top(reset, clk_50MHz,
        ps2_clk, ps2_data,
        vga_hsync, vga_vsync, vga_r, vga_g, vga_b,
        game,
        paddle1_up, paddle1_down, paddle2_up, paddle2_down);
  input reset;
  input clk_50MHz;
  input ps2_clk;
  input ps2_data;
   output vga_hsync;
   output vga_vsync;
   output vga_r;
   output vga_g;
   output vga_b;
  output game;
  output paddle1_up;
  output paddle1_down;
  output paddle2_up;
  output paddle2_down;

  wire vga_hsync;
  wire vga_vsync;
  wire vga_r;
  wire vga_g;
  wire vga_b;
  wire game;

  wire motion_clk;

  // -----
  // registers

  // VGA timing stuff
  wire active;
  wire [9:0] hpos;
  wire [9:0] vpos;

  // item positions
  wire [9:0] paddle1_v;
  wire [9:0] paddle2_v;

  wire [9:0] ball_v;
  wire [9:0] ball_h;
  wire ball_v_dir;
  wire ball_h_dir;

  wire paddle1_active;
  wire paddle2_active;
  wire ball_active;

  // -----
  // module instantiations

  motion_clock mtn_clk(
    .clk_50MHz(clk_50MHz),
    .clk_motion(motion_clk)
    );

  vga_sync sync(
    .vsync(vga_vsync),
    .hsync(vga_hsync),
```

```
    .vpos(vpos),
    .hpos(hpos),
    .active(active),
    .clk_50MHz(clk_50MHz),
    //.reset(reset)  --- don't reset VGA with the rest; monitor resync takes time
    .reset(0)
    );

  box_detect paddle1_box(
    .v(paddle1_v),
    .vsize(48),
    .h(0),
    .hsize(16),
    .vpos(vpos),
    .hpos(hpos),
    .q(paddle1_active)
    );

  assign vga_r = paddle1_active & active;

  box_detect paddle2_box(
    .v(paddle2_v),
    .vsize(48),
    .h(624),
    .hsize(16),
    .vpos(vpos),
    .hpos(hpos),
    .q(paddle2_active)
    );

  assign vga_b = paddle2_active & active;

  box_detect ball_box(
    .v(ball_v),
    .vsize(16),
    .h(ball_h),
    .hsize(16),
    .vpos(vpos),
    .hpos(hpos),
    .q(ball_active)
    );

  assign vga_g = ball_active & active;

  wire paddle1_up;
  wire paddle1_down;
  wire paddle2_up;
  wire paddle2_down;

  motion mtn(
    .reset(reset),
    .clk(motion_clk),
    .game(game),
    .ball_v(ball_v),
    .ball_h(ball_h),
    .ball_v_dir(ball_v_dir),
    .ball_h_dir(ball_h_dir),
    .paddle1_v(paddle1_v),
    .paddle2_v(paddle2_v),
    .paddle1_up(paddle1_up),
    .paddle1_down(paddle1_down),
    .paddle2_up(paddle2_up),
    .paddle2_down(paddle2_down)
    );
```

```
  kbd k(
    .clk(clk_50MHz),
    .reset(reset),
    .ps2_clk(ps2_clk),
    .ps2_data(ps2_data),
    .paddle1_up(paddle1_up),
    .paddle1_down(paddle1_down),
    .paddle2_up(paddle2_up),
    .paddle2_down(paddle2_down)
    );

endmodule
```

```
`timescale 1ns / 1ps

// VGA Pong v1.0     Chris Fallin <cfallin@nd.edu>

module vga_sync(clk_50MHz, reset, hsync, vsync, hpos, vpos, active);
  input clk_50MHz;
  input reset;
  output hsync;
  output vsync;
  output [9:0] hpos;
  output [9:0] vpos;
  output active;

  wire hsync;
  wire vsync;
  reg [9:0] hpos;
  reg [9:0] vpos;
  wire active;

  wire active_h, active_v;

  /*
    The sync patterns for 640x480 @ 60Hz (from the Basys board manual) are:

    H: (50Mhz pixel clock)
    0 - 639: active
    640 - 687: front porch
    688 - 783: H sync pulse (low)
    784 - 799: back porch

    V: (clocked from H sync pulse)
     0 - 479: active
    480 - 489: front porch
    490 - 491: V sync pulse (low)
    492 - 520: back porch
  */

  reg clk; // 25 MHz pixel clock

  always @(posedge clk_50MHz)
    clk <= ~clk;

  // H counter
  always @(posedge clk or posedge reset) begin

    if(reset == 1) begin
      hpos <= 0;
      vpos <= 0;
    end
    else
    // counter modulo 800
    if(hpos == 799) begin
      hpos <= 0;

      // V counter, driven by H counter rollover
      if(vpos == 520)
        vpos <= 0;
      else
        vpos <= vpos + 1;
    end
    else
      hpos <= hpos + 1;
  end

  // H sync
```

```
  assign hsync = ~ (hpos >= 656 && hpos < 752);
  //assign hsync = ~ (hpos >= 688 && hpos <= 783);

  // V sync
  assign vsync = ~ (vpos == 490 || vpos == 491);

  // H blank
  assign active_h = (hpos < 640);

  // V blank
  assign active_v = (vpos < 480);

  // blanking
  assign active = active_h & active_v;

  // setup
  initial begin
    vpos <= 0;
    hpos <= 0;
  end

endmodule
```