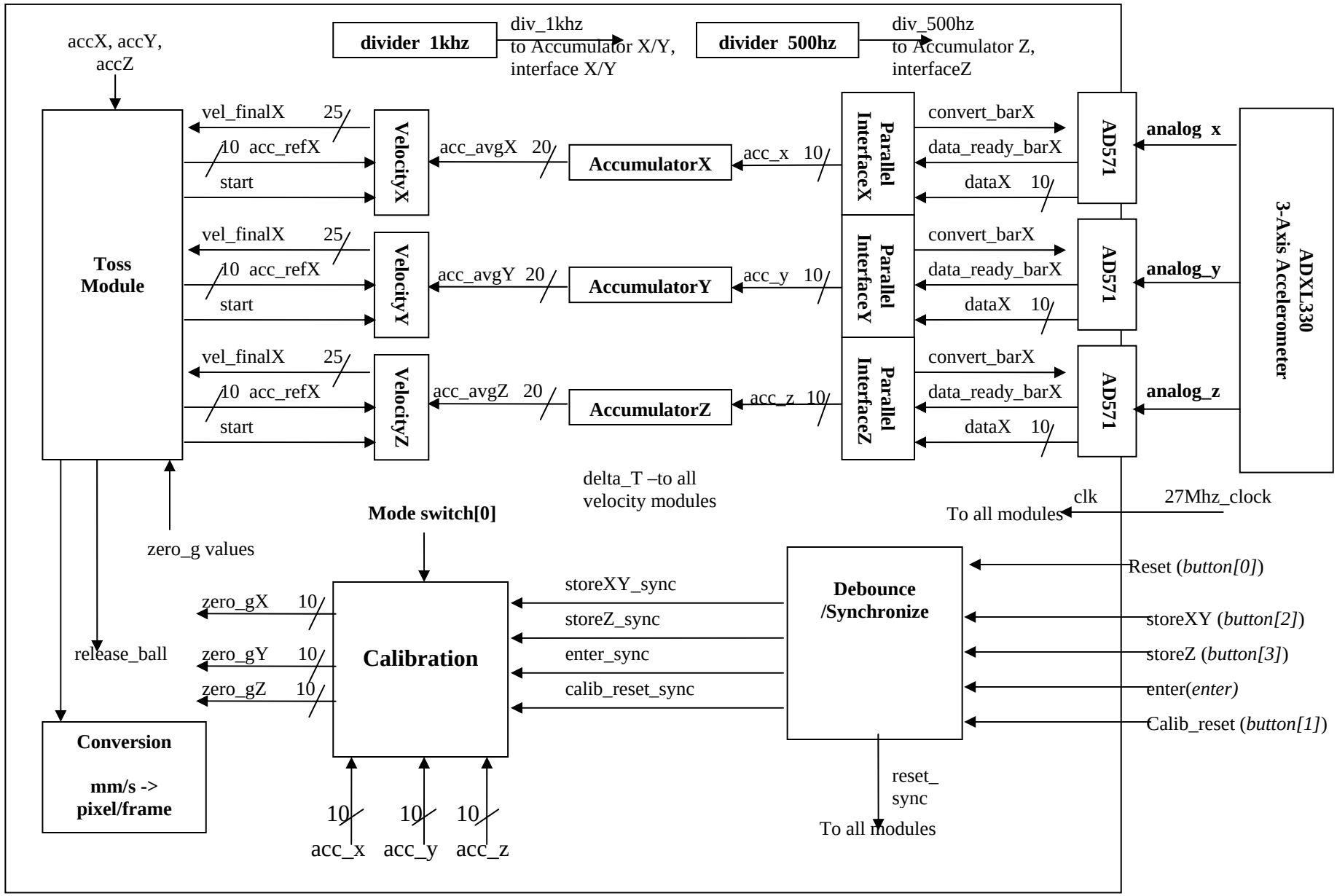
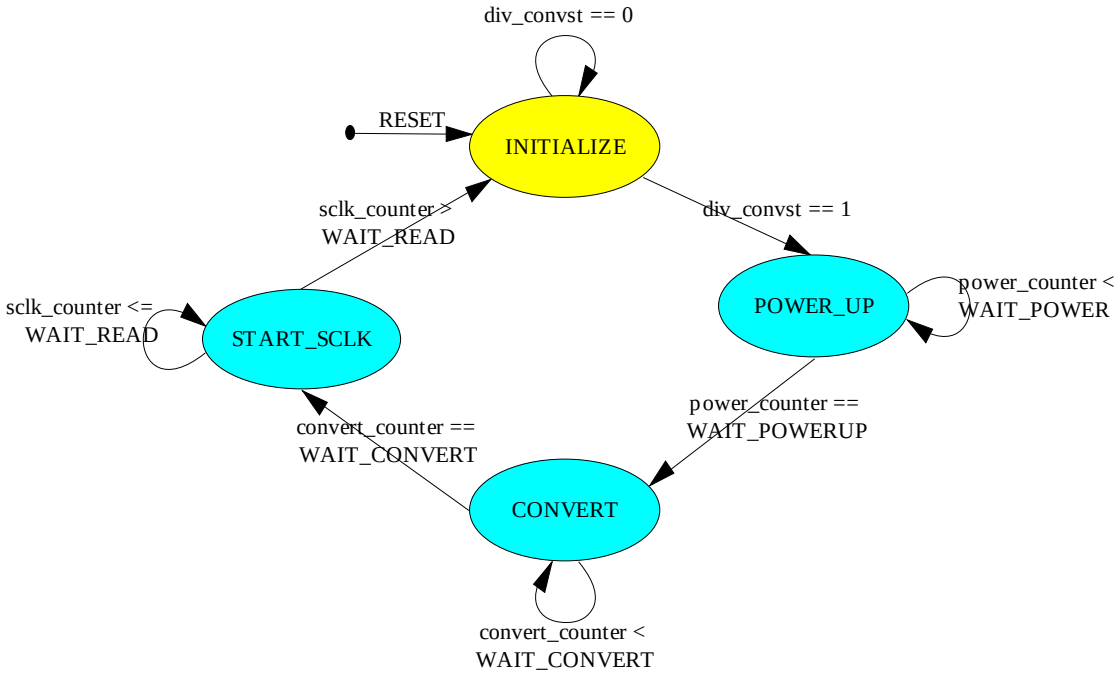
APPENDIX

APPENDIX A


accX, accY, accZ
divider 1khz
div_1khz to Accumulator X/Y, interface X/Y
divider 500hz
div_500hz to Accumulator Z, interfaceZ
Toss Module
vel_finalX 25
10 acc_refX
start
VelocityX
VelocityY
VelocityZ
acc_avgX 20
acc_avgY 20
acc_avgZ 20
AccumulatorX
AccumulatorY
AccumulatorZ
acc_x 10
acc_y 10
acc_z 10
Parallel InterfaceX
Parallel InterfaceY
Parallel InterfaceZ
convert_barX
data_ready_barX
dataX 10
AD571
analog x
analog_y
analog_z
ADXL330 3-Axis Accelerometer
delta_T –to all velocity modules
clk
27Mhz_clock
To all modules
Mode switch[0]
zero_g values
Calibration
zero_gX 10
zero_gY 10
zero_gZ 10
storeXY_sync
storeZ_sync
enter_sync
calib_reset_sync
Debounce /Synchronize
Reset (button[0])
storeXY (button[2])
storeZ (button[3])
enter(enter)
Calib_reset (button[1])
release_ball
Conversion
mm/s -> pixel/frame
10 acc_x
10 acc_y
10 acc_z
reset_ sync
To all modules

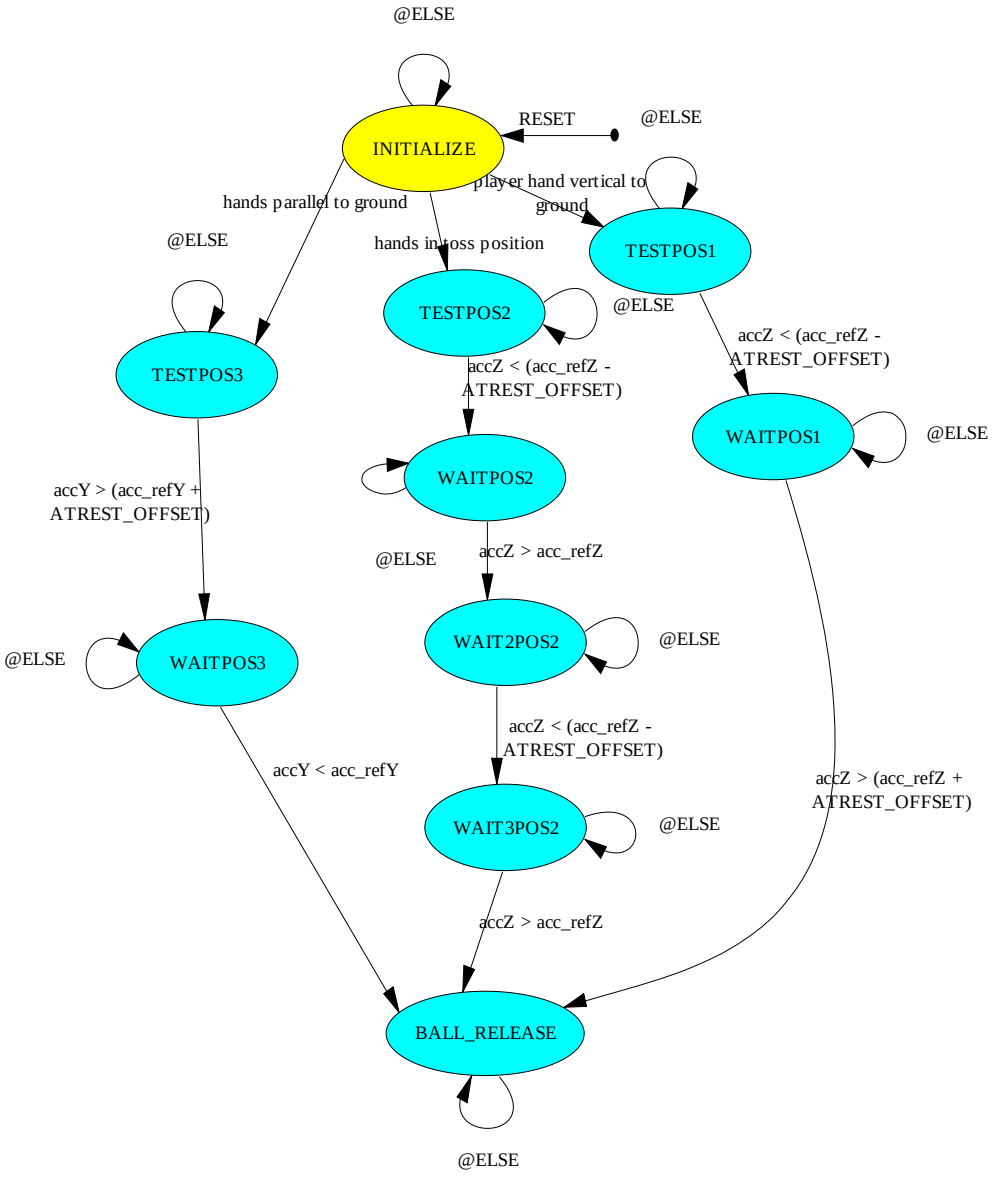
APPENDIX B

Serial Interface State Transition Diagram



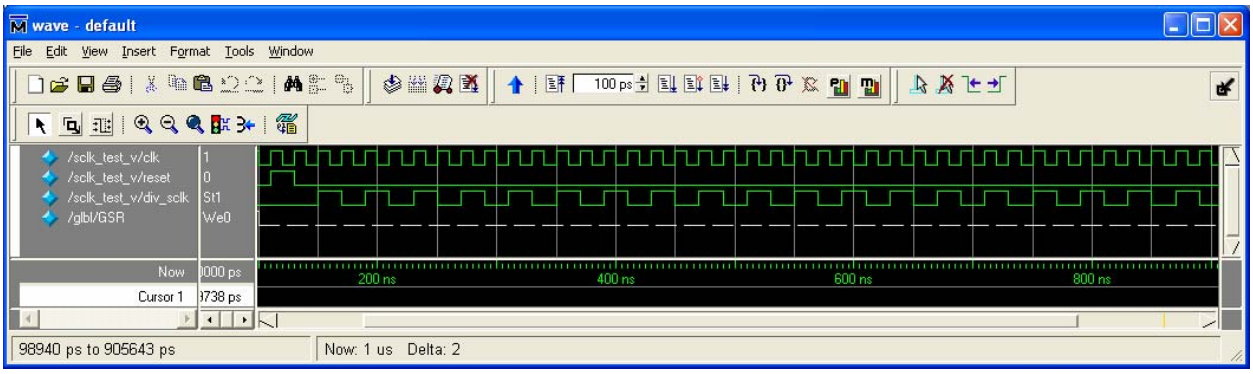
APPENDIX C

Toss Module State Transition Diagrams



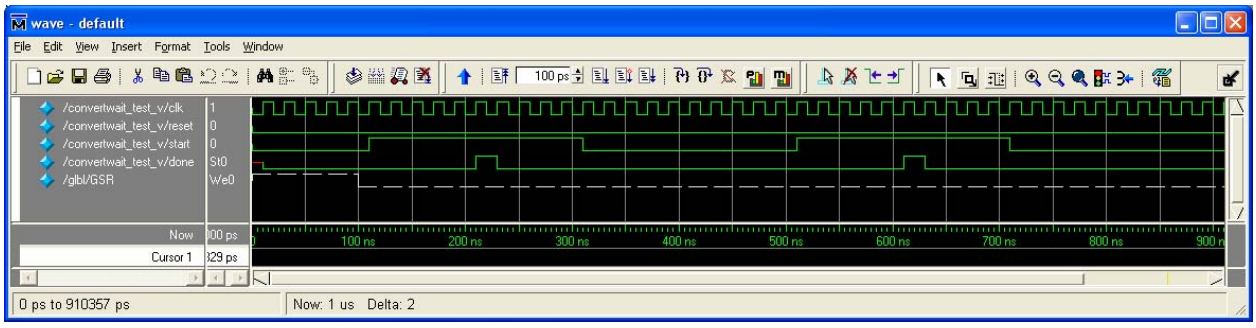
APPENDIX D

D.1 ModelSim waveform of *sclk*—sclk requires a minimum high and low pulse width of 25 ns each, which means sclk must have at least a 50 ns period. The 27MHz clock gives only 37 ns per period. Thus by dividing the 27MHz clock in half (74 ns period), there should be sufficient time to meet the specifications of sclk.

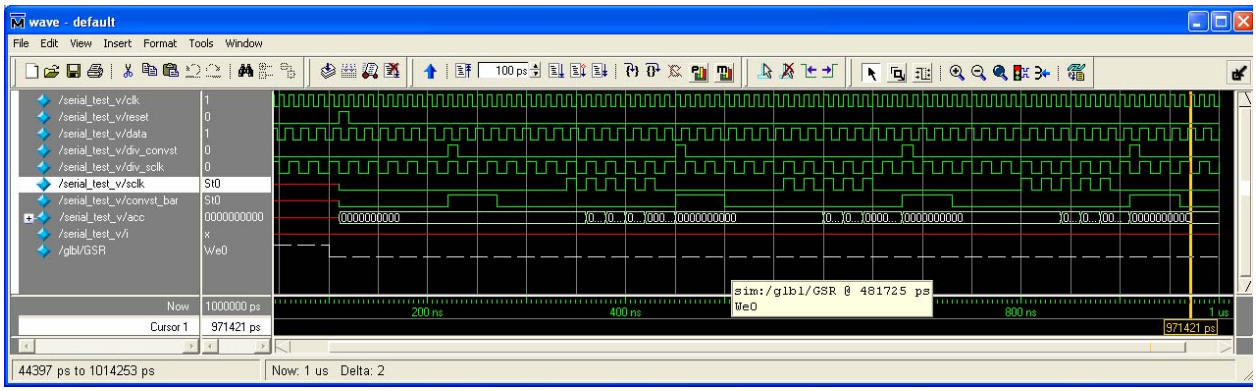


D.2 ModelSim waveform of convst_bar—Because we obtain data at such low bandwidths, using mode 2 of the AD7810 operation is sufficient and will also save power. However, Mode 2 requires a power-up time of 1.5us in addition to the conversion time of 2.3us. The total time required from conversion start to read finish requires approximately 5us which makes a 2 kHz *convst* enough to allow complete conversion and reading. The waveform shows that clock cycle has been slowed by half.

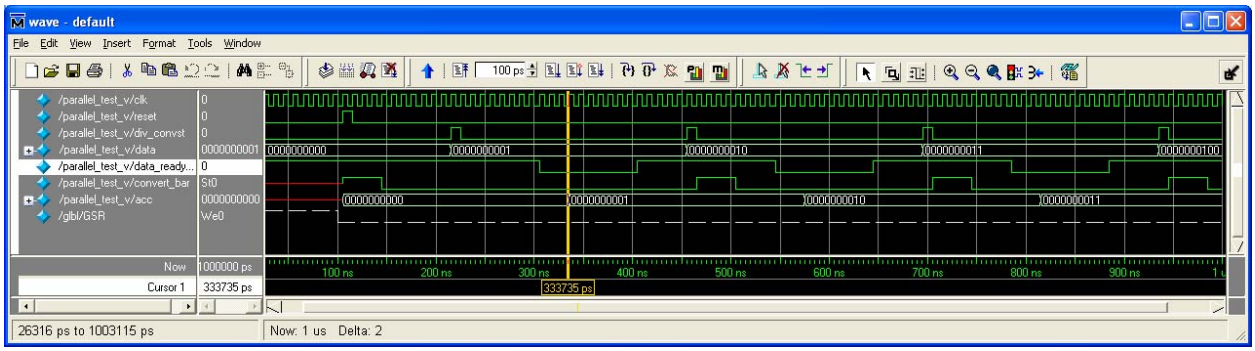
Once conversion has been started by convst, then it needs to wait until the device has powered up and finished the conversion before sclk can go high and clock the output. Convert_wait takes in a start signal that indicates when the conversion has begun and outputs a done signal when the waiting time is done. This ModelSim simulation uses a wait time of 5 clock cycles before done is outputted high.



D.3 ModelSim waveform of serial_interface--Generates the expected waveform for the serial interface, the power-up, conversion, and reading times were all changed to 5 clock cycles for testing

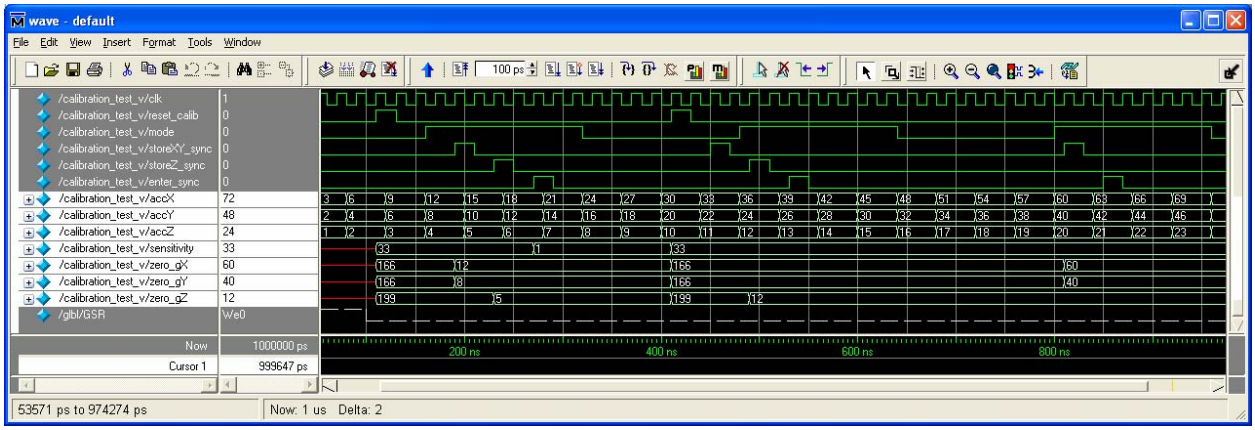


D.4 Parallel interface



When div_convst goes high, a conversion is started on the next clock cycle indicated by convert_bar going high. It is then followed by a blanking period. For testing purposes, the blanking time (time required to clear data and get ready to convert) was set to 4 such that 4 clock cycles later, convert_bar goes low and initiates the actual conversion. Data_ready is an input signal from the ADC that indicates the device has finished converting on a high to low transition. However, there is a 500 ns delay before the data becomes active. For simulation, I've set that delay to 2 clock cycles such that from the time *data_ready* goes low, the data is ready during the active window. In the waveform above, an extra clock cycle is introduced because the code tests when data_ready is low, not the transition edge. For unipolar operations, the bipolar control signal on each AD571 was connected to ground.

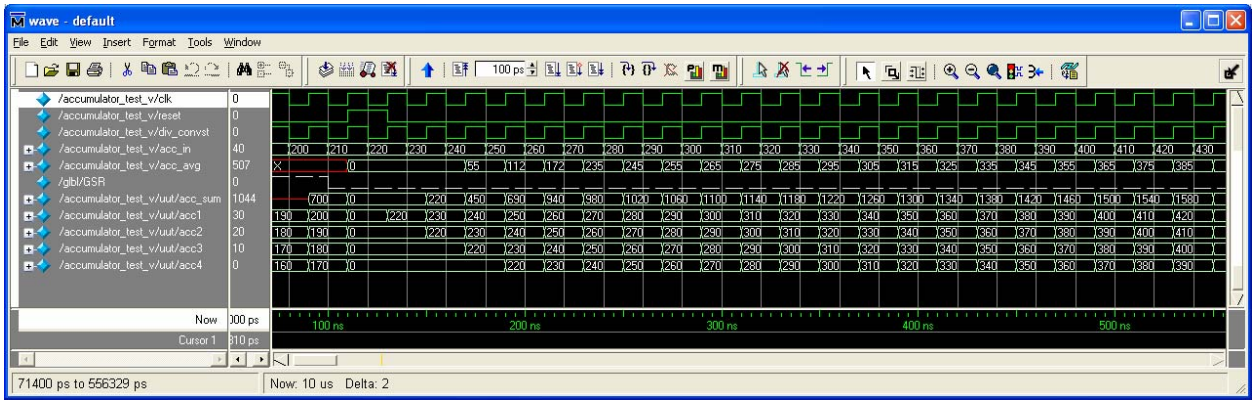
D.5 Calibration



Controlled by 3 buttons. Button1 resets only the calibration mode. The user can calibrate the zero g bias in the X, Y, Z, directions as follows. First, the user enters the calibration mode by setting mode (switch[0]) high. The user can only change zero-g bias values when under this mode. To calibrate the X/Y axis, the user must lay the chip flat facing up and then press button2. To calibrate the Z axis, the user needs to stand the chip on its top side (see picture). Finally, to calculate the sensitivity, the user should press button_enter.

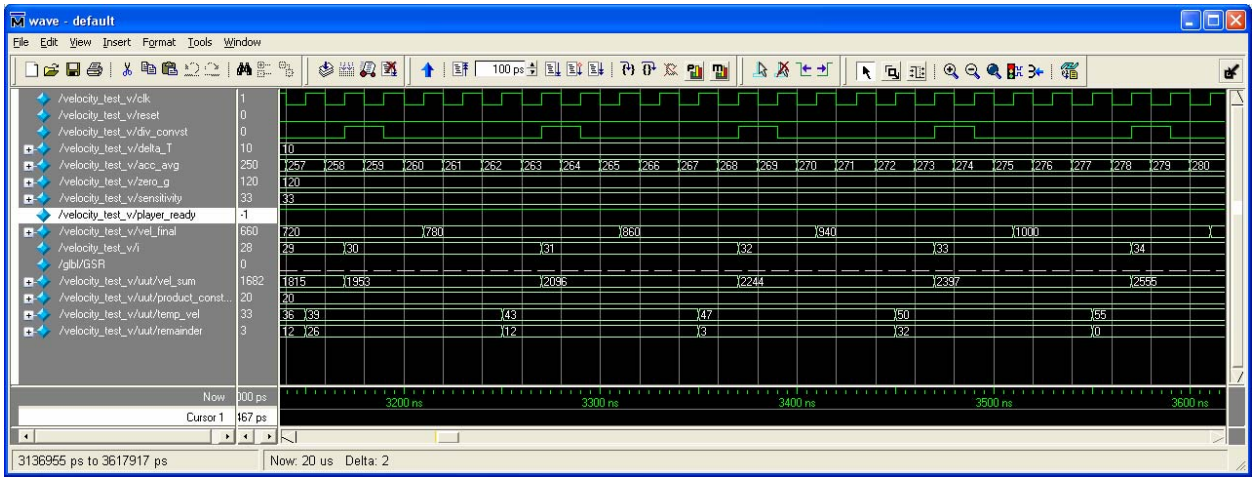
When mode goes high the first time, the store signals operate normally. With the rising edge of each signal, a new value for zero-g bias is stored into the corresponding axis. When enter goes high, the sensitivity is changed. The second time mode goes high, storeXY_sync goes high before mode goes high. As a result, XY axis zero-g bias does not change, but Z-axis does change. Notice that sensitivity remains unchanged because there were no storeXY values from which to calculate the sensitivity. Similarly for the third mode high signal, there is no storeZ_sync high, therefore no sensitivity gets calculated.

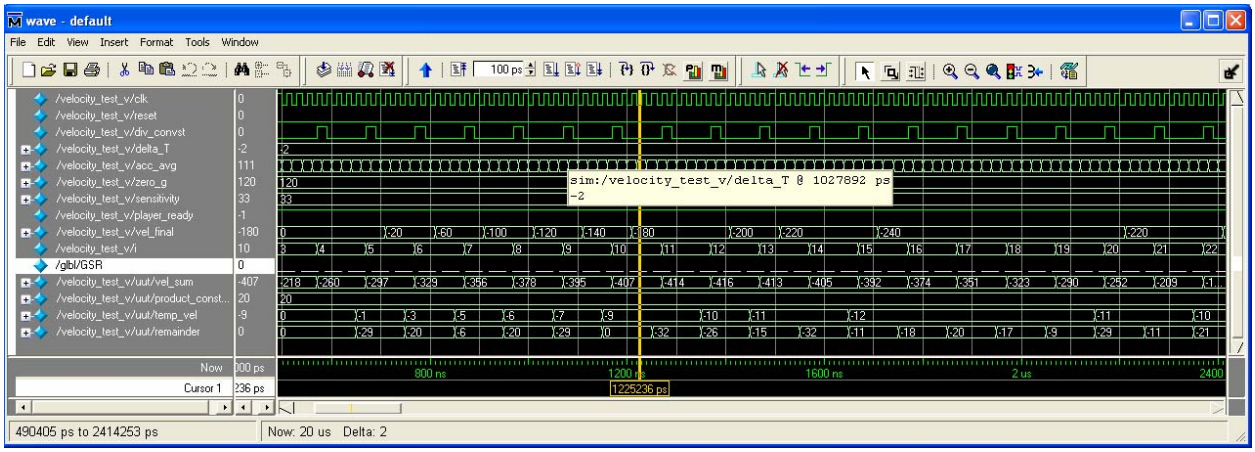
D.6 Accumulator



Accumulator shift registers—the accumulator's values are a little bit delayed, but it functions properly as a time averager over 4 clock cycles.

D.7 Velocity

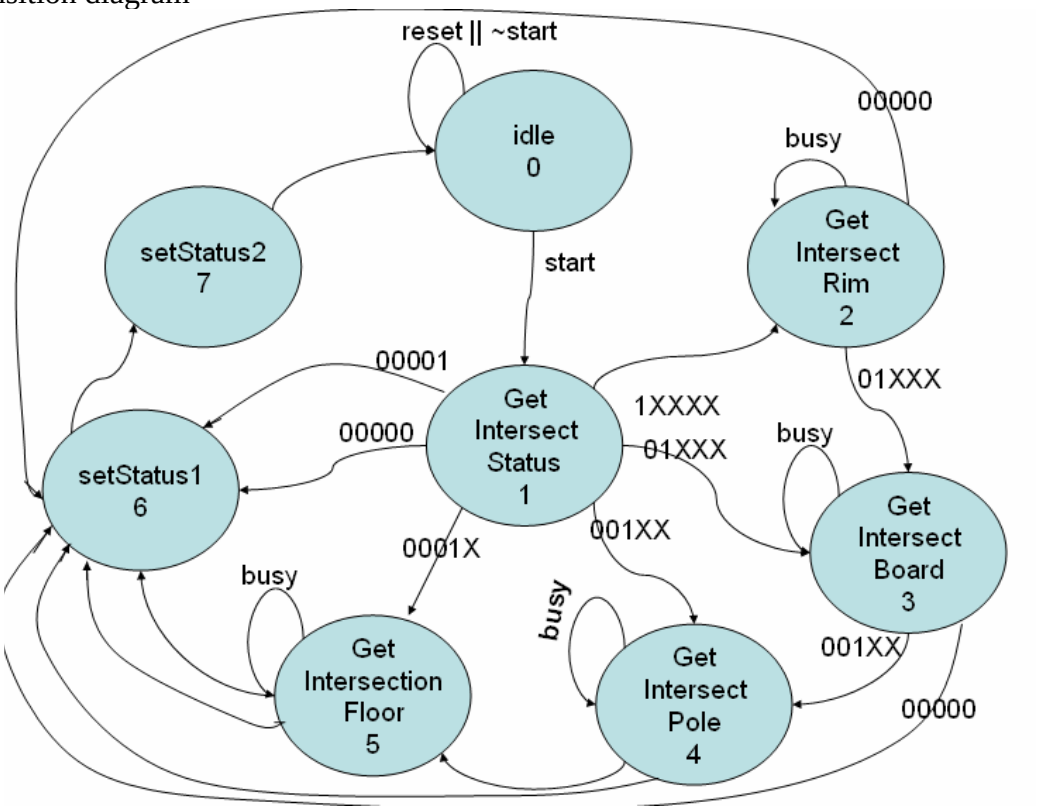

wave - default
File Edit View Insert Format Tools Window
/velocity_test_v/clk
/velocity_test_v/reset
/velocity_test_v/div_convst
/velocity_test_v/delta_T
/velocity_test_v/acc_avg
/velocity_test_v/zero_g
/velocity_test_v/sensitivity
/velocity_test_v/player_ready
/velocity_test_v/vel_final
/velocity_test_v/i
/glbl/GSR
/velocity_test_v/uut/vel_sum
/velocity_test_v/uut/product_const...
/velocity_test_v/uut/temp_vel
/velocity_test_v/uut/remainder
Now
Cursor 1
3200 ns
3300 ns
3400 ns
3500 ns
3600 ns
3136955 ps to 3617917 ps
Now: 20 us Delta: 2


wave - default
File Edit View Insert Format Tools Window
/velocity_test_v/clk
/velocity_test_v/reset
/velocity_test_v/div_convst
/velocity_test_v/delta_T
/velocity_test_v/acc_avg
/velocity_test_v/zero_g
/velocity_test_v/sensitivity
/velocity_test_v/player_ready
/velocity_test_v/vel_final
/velocity_test_v/i
/glbl/GSR
/velocity_test_v/uut/vel_sum
/velocity_test_v/uut/product_const...
/velocity_test_v/uut/temp_vel
/velocity_test_v/uut/remainder
sim:/velocity_test_v/delta_T @ 1027892 ps
-2
Now
Cursor 1
800 ns
1200 ns
1600 ns
2 us
2400
1225236 ps
490405 ps to 2414253 ps
Now: 20 us Delta: 2

APPENDIX E

1 *getStatus*

a) State transition diagram



b) Notes:

i. For simplification of the state diagram, I only showed a combination of a set of signals, they are set as {intersectWithRimPlane, intersectWithBoard, intersectWIthPole,intersectWithFloor, intersectBounary}.

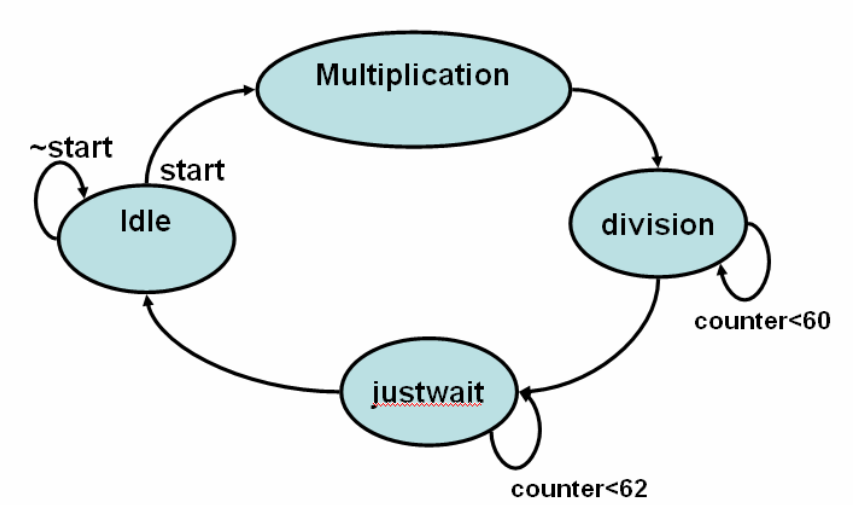
ii. The states 2, can also go to state 4, 5, depend on the set of signals. Because that makes the transition diagram really complicated and each state actually act in a similar manner, I neglected some of them in the representative transition diagram.

c) Outputs:

i. The output *status* is set to proper value depending on where the ball intersects, when a state goes to state 6, *setStatus*.

2 *getIntersection*

a) State transition diagram



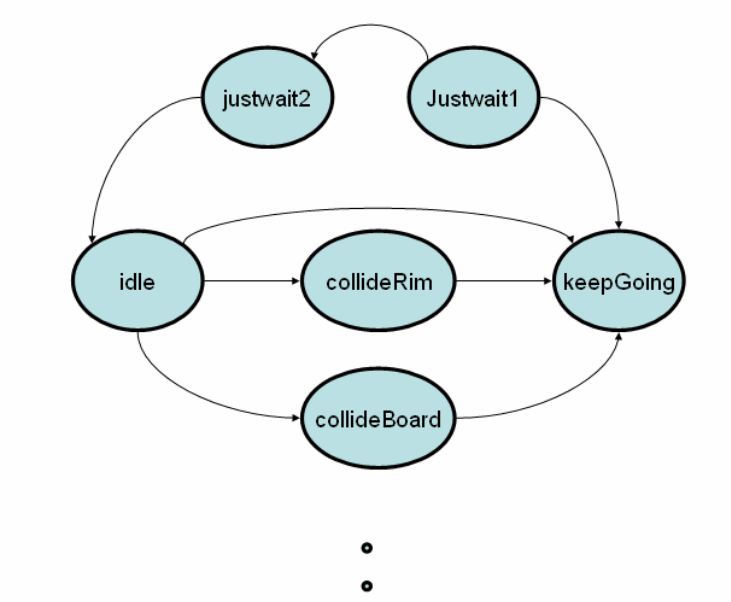
b) Outputs

i. The states *multiplication* and *division* outputs the proper signals to *multiplier* and *divide.*

ii. Counter is updated in *division* and *justwait.*

3 *getBallPosition*

a) transition diagram



b) control signals and outputs

i. There is no input dependence other than the *idle* state. In *idle* state, depending on what is the *status* for the ball, it either goes to a collidXXXX state to update the velocity due to collision, or goes to *keepGoing* state to update the position, and the velocity due to gravity.

APPENDIX F

parallel_interface.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Chun Li
// 6.111 Final Project: Virtual Basketball
//
// parallel_interface.v
//
// Used to control the timing and conversion of the AD571 ADC.
// The AD571 ADC has parallel 10-bit outputs. It takes in a convert start signal
// and after a 2us blanking period, starts the conversion. The conversion takes
// a maximum of 40us, after which the data ready signal goes low. After data ready
// goes low, there is an additional 500ns before data is active. This module
// controls all the timing required to sample and read from the ADC.
//
//////////////////////////////////////////////////////////////////////////////////

module parallel_interface(clk, reset, div_convst, data, data_ready_bar,
                      convert_bar, acc);

  input clk, reset;
  input div_convst;
  input [9:0] data;
  input data_ready_bar;

  output convert_bar;   // active low
  output [9:0] acc;

  reg convert_bar;
  reg [9:0] acc;
  reg [2:0] state;
  reg [5:0] counter; // same counter can be used multiple times; no functions overlap.

  parameter BLANK = 0;
  parameter CONVERTING = 1;
  parameter WAIT_ACTIVE = 2;
  parameter DONE = 3;

  parameter DATAREADY_TIME = 14; // clk cycles, 500 max delay after data ready
               // before it can actually be read

  parameter BLANK_TIME = 54;   // 54 clk cycles (2us) blanking period after
               // the rising edge of convert_bar before a conversion can start

  always @ (posedge clk) begin

    if (reset)
      begin
        convert_bar <= 1;
        acc <= 10'd0;
        counter <= 11'd1;  // counting starts at 1
        state <= BLANK;
```

```
    end
  else
    case(state)

    BLANK:
      begin

        if(counter == BLANK_TIME) // 2us pulse blank
          begin
            convert_bar <= 0;  // conversion starts on falling edge
            counter <= 0;
            state <= CONVERTING;
          end

        else
          begin
            convert_bar <= 1;
            counter <= counter + 1;
            state <= BLANK;
          end
      end

    CONVERTING:
      begin

        if(!data_ready_bar) state <= WAIT_ACTIVE; // active low

        else state <= CONVERTING;

      end

    WAIT_ACTIVE:
      begin

        // 500ns before data active from the time data_ready_bar goes low

        if(counter == DATAREADY_TIME)
          begin
            acc <= data;
            counter <= 0;
            state <= DONE;
          end

        else
          begin
            acc <= acc;
            counter <= counter + 1;
            state <= WAIT_ACTIVE;
          end
      end

    DONE:
      begin

        // conversion is done, so now wait for the *div_convst* signal
        // to begin the blanking period of another conversion

        if(div_convst == 1) state <= BLANK;    // to start the blanking period
        else state <= DONE;   // wait until next conversion requested

      end
```

```
       default: state <= DONE;

     endcase
   end
endmodule
```

parallel_test.v

```
module parallel_test_v;

   // Inputs
   reg clk;
   reg reset;
   reg div_convst;
   reg [9:0] data;
   reg data_ready_bar;

   // Outputs
   wire convert_bar;
   wire [9:0] acc;

   // Instantiate the Unit Under Test (UUT)
   parallel_interface uut (
      .clk(clk),
      .reset(reset),
      .div_convst(div_convst),
      .data(data),
      .data_ready_bar(data_ready_bar),
      .convert_bar(convert_bar),
      .acc(acc)
   );

   always #5 clk = ~clk;

   initial begin
      // Initialize Inputs
      clk = 0;
      reset = 0;
      div_convst = 0;
      data = 0;
      data_ready_bar = 1;

      // Wait 100 ns for global reset to finish
      #105;

      reset = 1;
      #10;
      reset = 0;
      #100;

      data = 1;
      div_convst = 1;
      #10;
      div_convst = 0;
      #80
      data_ready_bar = 0;
      #100;
      data_ready_bar = 1;
      #50;

      data = 2;
      div_convst = 1;
      #10;
      div_convst = 0;
      #80
      data_ready_bar = 0;
      #100;
      data_ready_bar = 1;
      #50;

      data = 3;
      div_convst = 1;
      #10;
      div_convst = 0;
```

```
      #80
      data_ready_bar = 0;
      #100;
      data_ready_bar = 1;
      #50;

      data = 4;
      div_convst = 1;
      #10;
      div_convst = 0;
      #80
      data_ready_bar = 0;
      #100;
      data_ready_bar = 1;
      #50;

   end

endmodule
```

serial_interface.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Chun Li
// 6.111 Final Project: Virtual Basketball
//
// serial_interface.v
//
// This module is used to initiate conversion on the ADC and to read data from it
// *convst* tells the ADC to start converting whatever analog data it was reading
// at the time. Using Mode 2 of the AD7810, it requires a power-up time of 1.5us
// and a guaranteed conversion time of 2.3us. After the conversion, *sclk* is
// assigned to half the speed of the 27Mhz clock (37 ns period)in order to satisfy
// the restraints of the chip (at least 50ns clock period). *sclk* is used by the
// chip to read out the bits of the converted results starting from the MSB.
// Thus, using shift registers, the data is shifted into a 10bit result *acc* to
// be outputted.
//
//////////////////////////////////////////////////////////////////////////////////

module serial_interface(clk, reset, data, div_convst, div_sclk, convst_bar, sclk, acc);

   input clk, reset;
   input div_convst, div_sclk;
   input data;

   output convst_bar, sclk;
    output [9:0] acc;    //10 bit final output

   reg sclk;
   reg convst_bar;
   reg [9:0] acc;
   reg [9:0] acc_temp;

   reg [2:0] state;
   reg [5:0] power_counter;
   reg [5:0] convert_counter;
   reg [3:0] sclk_counter;

   parameter INITIALIZE = 0;
   parameter POWER_UP = 1;
   parameter CONVERT = 2;
   parameter START_SCLK = 3;

   parameter WAIT_POWERUP = 41; //clock cycles: 1.5us to powerup
   parameter WAIT_CONVERT = 63; //clock cylces: 2.3us to convert
   parameter WAIT_READ = 11;    //read data for 11 sclk cycles get all 10 bits

   always @ (posedge clk) begin

      if (reset)
        begin
          acc <= 10'd0;
          acc_temp <= 10'd0;
          convst_bar <= 0;
          sclk <= 0;
          power_counter <= 1;
          convert_counter <= 1;
          sclk_counter <= 1;
          state <= INITIALIZE;
        end

      else
        begin
          case (state)
```

```
INITIALIZE:
  begin
    if (div_convst == 1)    // indicates start of a conversion cycle
      begin
        convst_bar <= 1;
        sclk <= 0;
        state <= POWER_UP;      // starts ADC powerup
      end

    else state <= INITIALIZE;
  end

POWER_UP:   // requires 1.5us to powerup
  begin
    if(power_counter == WAIT_POWERUP)
      begin
        power_counter <= 1;
        convst_bar <= 0;
        state <= CONVERT;
      end

    else
      begin
        power_counter <= power_counter + 1;
        state <= POWER_UP;
      end

  end

CONVERT:  // ADC needs 2.3us to convert analog to digital
  begin
    if(convert_counter == WAIT_CONVERT)
      begin
        convert_counter <= 1;
        state <= START_SCLK;
      end

    else
      begin
        convert_counter <= convert_counter + 1;
        state <= CONVERT;
      end

  end

START_SCLK:   //
  begin

    sclk <= div_sclk;

    // sclk high to read in a bit
    // the first clock doesn't read any data
    // if sclk counter = 11, then all 10 bits should be read it
    // and the first garbage bit will be eliminated

    if (sclk_counter <= WAIT_READ)
      begin

        if(div_sclk == 1)  // get next bit
          begin
            acc_temp <= {acc_temp[8:0],data}; // shift register
            sclk_counter <= sclk_counter + 1;
            state <= START_SCLK;
          end

        else
          begin
            acc_temp <= acc_temp;
            sclk_counter <= sclk_counter;
```

```
                        state <= START_SCLK;
                      end
                  end

               else   // sclk_counter > WAIT_READ : done reading
                  begin
                     acc <= acc_temp;
                     sclk_counter <= 1; // reset the counter
                     state <= INITIALIZE;  // go back to wait for conversion start
                  end
            end

         default: state <= INITIALIZE;

         endcase
      end
   end

endmodule
```

serial_ test.v

```
module serial_test_v;

   // Inputs
   reg clk;
   reg reset;
   reg data;
   reg div_convst;
   reg div_sclk;

   // Outputs
   wire sclk, convst_bar;
   wire [9:0] acc;

   // Instantiate the Unit Under Test (UUT)
   serial_interface uut (
      .clk(clk),
      .reset(reset),
      .data(data),
      .div_convst(div_convst),
      .div_sclk(div_sclk),
      .convst_bar(convst_bar),
      .sclk(sclk),
      .acc(acc)
   );
   always #5 clk = ~clk;
   always #10 div_sclk = ~div_sclk;
   always #8 data = ~data;

   integer i;
   initial begin
      // Initialize Inputs
      clk = 1;
      reset = 0;
      data = 0;
      div_convst = 0;
      div_sclk = 1;

      // Wait 100 ns for global reset to finish
      #110;
      reset = 1;
      #10;
      reset = 0;
      #100;

      div_convst = 1;
      #10;
      div_convst = 0;
      #230;

      div_convst = 1;
      #10;
      div_convst = 0;
      #230;

      div_convst = 1;
      #10;
      div_convst = 0;
      #230;

      div_convst = 1;
      #10;
      div_convst = 0;
      #230;

   end
endmodule
```

divider_1khz.v

```
//////////////////////////////////////////////////////////////////////////////
//
// Chun Li
// 6.111 Lab 2
// divider_1khz.v
//
// Design a module, counter, that takes two inputs, a 27 MHz *clock*
// and *reset*, and outputs a 1 kHz *div_1khz* signal that will tell
// the AD571 ADC when to start conversions. This signal is used for the
// X and Y axis (each with a max bandwidth of 1.6kHz).
//
//////////////////////////////////////////////////////////////////////////////

`timescale 1ns/10ps

module divider_1khz(clk, reset, div_1khz);

   input clk, reset;
   output div_1khz;

   reg div_1khz;
   reg [14:0] count;            // 27 MHz clock between 24 and 25 bits

   always @ (posedge clk) begin

     if (reset)              // if reset high
     begin
       count <= 14'd0;       // set count back to 0
       div_1khz <= 1'd0;       // set enable to 0
     end                   // if

     else if (count == 26999)  // use (count == 17900) for 1.5 kHz enable rate
     begin
       div_1khz <= 1'd1;       // enable high only for one period of 27MHz
       count <= 14'b0;       // reset count to 0
     end                   // end if

     else                  // if nothing
     begin
       count <= count + 1;   // increment count
       div_1khz <= 1'b0;       // enable low for all other cases

     end                   // else

   end                     // always

endmodule
```

divider_500hz.v

```
//////////////////////////////////////////////////////////////////////////////
//
// Chun Li
// 6.111 Lab 2
// divider_500hz.v
//
// Design a module, counter, that takes two inputs, a 27 MHz *clock*
// and *reset*, and outputs a 500 hz *div_500hz* signal that will tell
// the AD571 ADC to start a conversion. The 500hz signal is used to
// sample the Z coordinate because it has a max bandwidth of 550hz.
//
//////////////////////////////////////////////////////////////////////////////

`timescale 1ns/10ps

module divider_500hz(clk, reset, div_500hz);

   input clk, reset;
   output div_500hz;

   reg div_500hz;
   reg [15:0] count;            // 27 MHz clock between 24 and 25 bits

   always @ (posedge clk) begin

     if (reset)              // if reset high
     begin
       count <= 14'd0;       // set count back to 0
       div_500hz <= 1'd0;      // set enable to 0
     end                   // if

     else if (count == 53999)   // 500hz Z direction bandwidth
     begin
       div_500hz <= 1'd1;      // enable high only for one period of 27MHz
       count <= 14'b0;       // reset count to 0
     end                   // end if

     else                  // if nothing
     begin
       count <= count + 1;   // increment count
       div_500hz <= 1'b0;      // enable low for all other cases

     end                   // else

   end                     // always

endmodule
```

calibration.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////
//
// Chun Li
// 6.111 Final Project: Virtual Basketball
//
// calibration.v
//
// Each accelerometer has different sensitivities and 0-g bias points. This module
// will allow the user to use the calibration mode to store and later use the value
// for calculations. The user can change 0-g bais and sensitivity only when the
// user enters this mode.
//
// There is the option of using the BRAM on the labkit which is fast and stores
// value until the user changes it or the labkit powers off. However, since
// calibration only stores a few 10-bit values, I chose to use registers. It saves
// interfacing time and is easier to manipulate and change.
//
// Calibration will be taken using the following protocol. Button1 is a calibration
// reset signal. Only when it is pressed will the stored calibrations revert to
// default. Pressing button0 (reset_sync) will not affect the value, thus the device
// has only to be calibrated once as long as the labkit is on.
//
// Once in calibration mode, there are two layouts used. First lay the device flat
// on a table top, press *store* (button2) to store the first value. Then lay it
// vertically on the top edge, press *store* again to remember the second value.
// The program below will output the final zero-g bias in each direction and
// a calculated sensitivity based on the measurements. The theory behind it is to
// use gravity. when the chip is flat, gravity acts on the Z axis, but all the other
// axis will be the zero-g value. When the chip is vertical,
//
//////////////////////////////////////////////////////////////////////////////

module calibration(clk, reset_calib, mode, storeXY_sync, storeZ_sync, enter_sync,
               accX, accY, accZ, sensitivity, zero_gX, zero_gY, zero_gZ);

    input clk;
    input reset_calib;  // button1;
   input mode;     // enters calibration mode when switch0 high
   input storeXY_sync, storeZ_sync, enter_sync;
   input [9:0] accX, accY, accZ; // values from the parallel interface
                            // to get the actual voltage in mV
                            // multiply numbers by 10.
   output [9:0] sensitivity;
   output [9:0] zero_gX;
   output [9:0] zero_gY;
   output [9:0] zero_gZ;

   reg [9:0] sensitivity;
   reg [9:0] zero_gX;
   reg [9:0] zero_gY;
   reg [9:0] zero_gZ;

   reg [9:0] zero_gZ0;
   reg [9:0] zero_gZ1;

   // using 3.3V to power the accelerometer, the ADXL330 device should have a
   // a sensitivity of 330 mV/g and laying the chip flat and facing up,
   // it has a zero-g bias of X = 1.66V, Y = 1.66V, Z = 1.66V.

   parameter DEFAULT_SENS = 33;  // mV/10
   parameter DEFAULT_ZEROX = 166;// mV/10
   parameter DEFAULT_ZEROY = 166;// mV/10
   parameter DEFAULT_ZEROZ = 166;// mV/10

   always @ (posedge clk) begin
```

```
    if(reset_calib)
      begin
        sensitivity <= DEFAULT_SENS;
        zero_gX <= DEFAULT_ZEROX;
        zero_gY <= DEFAULT_ZEROY;
        zero_gZ <= DEFAULT_ZEROZ;
        zero_gZ0 <= 0;
        zero_gZ1 <= 0;
      end

    else if(mode)  // mode high = in calibration mode
      begin
        if(storeXY_sync) // lay device facing up and flat on the table
          begin
            zero_gX <= accX;    // should be appx. 1.66V
            zero_gY <= accY;    // should be appx. 1.66V
            zero_gZ0 <= accZ;     // should be appx. 1.66V
          end

        // set device vertical
        else if(storeZ_sync)
          begin
            zero_gZ <= accZ;    // should be appx. 1.66V
            zero_gZ1 <= accZ;     // temp variable used to calculate sensit.
          end


        else if(enter_sync)
          begin
            sensitivity <= (((zero_gZ0 == 0) || (zero_gZ1 == 0))?
                          sensitivity:
                          ((zero_gZ0 > zero_gZ1)? (zero_gZ0 - zero_gZ1):
                                                  (zero_gZ1 - zero_gZ0)));

            zero_gZ0 <= 0; // reset temporary variables
            zero_gZ1 <= 0;
          end
      end

    else // if not in calibration mode (switch0 low), then keep same values
      begin
        zero_gX <= zero_gX;
        zero_gY <= zero_gY;
        zero_gZ <= zero_gZ;
        sensitivity <= sensitivity;
      end

  end
endmodule
```

calibration_test.v

```
module calibration_test_v;

   // Inputs
   reg clk;
   reg reset_calib;
   reg mode;
   reg storeXY_sync;
   reg storeZ_sync;
   reg enter_sync;
   reg [9:0] accX;
   reg [9:0] accY;
   reg [9:0] accZ;

   // Outputs
   wire [9:0] sensitivity;
   wire [9:0] zero_gX;
   wire [9:0] zero_gY;
   wire [9:0] zero_gZ;

   // Instantiate the Unit Under Test (UUT)
   calibration uut (
      .clk(clk),
      .reset_calib(reset_calib),
      .mode(mode),
      .storeXY_sync(storeXY_sync),
      .storeZ_sync(storeZ_sync),
      .enter_sync(enter_sync),
      .accX(accX),
      .accY(accY),
      .accZ(accZ),
      .sensitivity(sensitivity),
      .zero_gX(zero_gX),
      .zero_gY(zero_gY),
      .zero_gZ(zero_gZ)
   );
   always #10 clk = ~clk;
   always #160 mode = ~mode;
   always #40 accX = accX + 3;
   always #40 accY = accY + 2;
   always #40 accZ = accZ + 1;
   initial begin
      // Initialize Inputs
      clk = 0;
      reset_calib = 0;
      mode = 0;
      storeXY_sync = 0;
      storeZ_sync = 0;
      enter_sync = 0;
      accX = 0;
      accY = 0;
      accZ = 0;

      // Wait 100 ns for global reset to finish
           #110;
       reset_calib = 1;
      #20;
      reset_calib = 0;
      #60;

      storeXY_sync = 1;
      #20;
      storeXY_sync = 0;
      #20;

      storeZ_sync = 1;
      #20;
```

```
      storeZ_sync = 0;
      #20;
      enter_sync = 1;
      #20;
      enter_sync = 0;
      #120;


      reset_calib = 1;
      #20;
      reset_calib = 0;
      #20;

      storeXY_sync = 1;
      #20;
      storeXY_sync = 0;
      #20;

      storeZ_sync = 1;
      #20;
      storeZ_sync = 0;
      #20;
      enter_sync = 1;
      #20;
      enter_sync = 0;
      #260;

      storeXY_sync = 1;
      #20;
      storeXY_sync = 0;
      #20;

      enter_sync = 1;
      #20;
      enter_sync = 0;

   end

endmodule
```

accumulator.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Chun Li
// 6.111 Final Project: Virtual Basketball
//
// accumulator.v
//
// a simple file that registers 4 acceleration inputs and averages them
//
//////////////////////////////////////////////////////////////////////////////////
module accumulator(clk, reset, div_convst, acc_in, acc_avg);

   input clk;
   input reset;
   input div_convst;    // can be div_1khz for X/Y or div_500hz for Z
   input [9:0] acc_in;

   output [9:0] acc_avg;

  reg [9:0] acc_avg;
  reg [12:0] acc_sum;   // sum of 4 10-bit numbers is a 12 bit number

  reg [9:0] acc1, acc2, acc3, acc4;   // 4 shift registers to do an average acc

  always @ (posedge clk) begin

    if (reset)
      begin
        acc_avg <= 0;    // set all values to zero-g bias acceleration
        acc_sum <= 0;
        acc1 <= 0;
        acc2 <= 0;
        acc3 <= 0;
        acc4 <= 0;
      end

    else if (div_convst)    // tell registers when to shift data to the next register
      begin
        acc1 <= acc_in;
        acc2 <= acc1;
        acc3 <= acc2;
        acc4 <= acc3;

        // calculate sum and average using data from previous clock cycle

        acc_sum <= acc1 + acc2 + acc3 + acc4;
        acc_avg <= {2'b00, acc_sum[10:2]};  // remove last 2 bits to divide by 4

        // determine which direction the object is traveling in
      end

    else acc_avg <= acc_avg;

  end

endmodule
```

accumulator_test.v

```
module accumulator_test_v;

   // Inputs
   reg clk;
   reg reset;
   reg div_convst;
   reg [9:0] acc_in;

   // Outputs
   wire [9:0] acc_avg;

   // Instantiate the Unit Under Test (UUT)
   accumulator uut (
      .clk(clk),
      .reset(reset),
      .div_convst(div_convst),
      .acc_in(acc_in),
      .acc_avg(acc_avg)
   );

   always #10 clk = ~clk;
   always #20 acc_in = acc_in + 10;
   always #10 div_convst = ~div_convst;

   initial begin
      // Initialize Inputs
      clk = 0;
      reset = 0;
      div_convst = 0;
      acc_in = 160;

      // Wait 100 ns for global reset to finish
      #110;
      reset = 1;
      #20;
      reset = 0;
      #20;

      // Add stimulus here

   end

endmodule
```

velocity.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Chun Li
// 6.111 Final Project
// Virtual Basketball
//
// velocity.v
//
// This module does the calculations from acceleration in millivolts to an actual
// velocity measured in mm/s. The math goes as follows.
// acc_avg (mV/10)
// sensitivity (mV/10)
// delta_T (ms)
// gravity (m/s^2)
// SUM(change in acc_avg) / sensitivity gives me acceleration in units of g
// multiply the above results by ~10m/s^2 (10*10^3 mm/s^2)
// and delta_T (units converted to seconds)
// simplifying the units gives me units of mm/s such that the delta_T input
// can be given in units of (ms); this greatly simplifies the math.
//
//////////////////////////////////////////////////////////////////////////////////
module velocity(clk, reset, div_convst, delta_T, acc_avg, acc_ref, sensitivity,
                start, vel_final);

   input clk, reset;
   input div_convst;
   input [1:0] delta_T;    // in milliseconds (ms) 1khz => 1ms; 500hz => 2ms
   input [9:0] acc_avg;
   input [9:0] acc_ref;
   input [9:0] sensitivity;
   input start;

   output signed [25:0] vel_final;   // final output
   //output signed [19:0] vel_sum;
   //output signed [5:0] product_constants;

   reg signed [19:0] vel_sum;     // allow plenty of bits (20)for the sum

   parameter GRAVITY = 4'd10;   // gravity 9.8 m/s^2 approximated to 10 m/s^2.

   always @ (posedge clk) begin

      if (reset)
        begin
          vel_sum <= 0;
        end

      else if (start)
        begin
          if (div_convst)
            begin

              if(acc_avg < acc_ref) vel_sum <= vel_sum - (acc_ref - acc_avg);

              else if (acc_avg >= acc_ref) vel_sum <= vel_sum + (acc_avg - acc_ref);
              else vel_sum <= vel_sum;

            end

          else vel_sum <= vel_sum;
        end

      else vel_sum <= 0;

   end
```

```
   wire [5:0] product_constants;

   multiplier M1(
      .sclr(reset),
      .clk(clk),
      .a(GRAVITY),
      .b(delta_T),
      .o(product_constants)
   );

   wire [19:0] temp_vel;
   wire [10:0] remainder;

   divide_acc D1(
      .clk(clk),
      .sclr(reset),
      .dividend(vel_sum),
      .divisor({1'b0,sensitivity}),
      .quotient(temp_vel),
      .remainder(remainder),
      .rfd(rfd)
   );

   multiplier2 M2(
      .sclr(reset),
      .clk(clk),
      .a(temp_vel),
      .b(product_constants),
      .o(vel_final)
   );

Endmodule
```

velocity_test.v

```
module velocity_test_v;

   // Inputs
   reg clk;
   reg reset;
   reg div_convst;
   reg [1:0] delta_T;
   reg [9:0] acc_avg;
   reg [9:0] acc_ref;
   reg [9:0] sensitivity;
   reg player_ready;

   // Outputs
   wire [25:0] vel_final;

   // Instantiate the Unit Under Test (UUT)
   velocity uut (
      .clk(clk),
      .reset(reset),
      .div_convst(div_convst),
      .delta_T(delta_T),
      .acc_avg(acc_avg),
      .acc_ref(acc_ref),
      .sensitivity(sensitivity),
      .player_ready(player_ready),
      .vel_final(vel_final)
   );

   always #10 clk = ~clk;
   always #20 acc_avg = acc_avg + 1;
   integer i;
   initial begin
      // Initialize Inputs
      clk = 0;
      reset = 0;
      div_convst = 0;
      delta_T = 2'd2;
      acc_avg = 10'd50;
      acc_ref = 10'd120;
      sensitivity = 10'd33;
      player_ready = 0;

      // Wait 100 ns for global reset to finish
      #110;
       reset = 1;
      #20;
      reset = 0;
      #20;
      player_ready = 1;
      #20;

      for(i=0; i< 50; i=i+1)
      begin
         div_convst = 1;
         #20;
         div_convst = 0;
         #80;
      end
      #1000;
      player_ready = 0;
      // Add stimulus here


   end

endmodule
```

toss_module.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Chun Li
// 6.111 Final Project: Virtual Basketball
//
// toss_module.v
//
// Take input from the calibration module and the acceleration from the serial
// interface and outputs signals that indicate to the display module the different
// positions of the player's throw. At each position, an enable signal tells the
// display what position to move to next. The last position is essentially the
// release of the ball, at which point I will output the velocity.
//
// The acceleration data coming in should have a value between 0 and 330. Multiply
// these values by 10 to get 0 - 3300 mV range of the accelerometer. Because the
// device is pretty sensitive,
//
// The release velocities X, Y, Z directions are relative to real axis
// the accelerometer X, Y, Z directions may change relative to the real axis
// confusing...i know.
//
//////////////////////////////////////////////////////////////////////////////////
module toss_module(clk, reset, zero_gX, zero_gY, zero_gZ, sensitivity,
                   accX, accY, accZ, velX_in, velY_in, velZ_in,
                   player_ready, acc_refX, acc_refY, acc_refZ, release_ball,
                   release_velX, release_velY, release_velZ, start);

    input clk, reset;
   input [9:0] accX, accY, accZ;
   input [9:0] zero_gX, zero_gY, zero_gZ, sensitivity;
   input signed [25:0] velX_in, velY_in, velZ_in;

   output player_ready; // should turn one led light on
   output release_ball;
   output [9:0] acc_refX, acc_refY, acc_refZ;
   output [25:0] release_velX, release_velY, release_velZ;
   output start;

   reg player_ready, release_ball;
   reg [25:0] release_velX, release_velY, release_velZ;
   reg [9:0] acc_refX, acc_refY, acc_refZ;
   reg start;

   reg [4:0] state;
   reg [24:0] counter;

   parameter INITIALIZE = 0;
   parameter TESTPOS1 = 1;
   parameter WAITPOS1 = 2;
   parameter TESTPOS2 = 3;
   parameter WAITPOS2 = 4;
   parameter WAIT2POS2 = 5;
   parameter WAIT3POS2 = 6;
   parameter TESTPOS3 = 7;
   parameter WAITPOS3 = 8;
   parameter BALL_RELEASE = 9;

   parameter INIT_ONESEC = 26999999;
   parameter ATREST_OFFSET = 8;  // = to a 160mV fluctuation range at rest

   always @ (posedge clk) begin

     if(reset)
       begin
         player_ready <= 0;
```

```
      release_ball <= 0;
      release_velX <= 25'd0;
      release_velY <= 25'd0;
      release_velZ <= 25'd0;
      acc_refX <= zero_gX;
      acc_refY <= zero_gY;
      acc_refZ <= zero_gZ;
      counter <= 25'd0;
      state <= INITIALIZE;
    end

  else
    case(state)

      INITIALIZE:
        begin

          release_velY <= 25'd0; // assign output Y velocity to display
          release_velZ <= 25'd0;
          release_velX <= 25'd0;
          release_ball <= 0;
          player_ready <= 0;

          // player needs to stay in one of the starting positions
          // for at least one second before the system signals that
          // the player can begin shooting

          // detect first mode: test (player has hands vertical to
          // ground and then push forward. look at accY)
          // if (~199-offset) =< accY <= (~199+offset)

          if((accY <= (zero_gY + sensitivity + ATREST_OFFSET))
          && (accY >= (zero_gY + sensitivity - ATREST_OFFSET)))
            begin
              if(counter == INIT_ONESEC)
                begin
                  acc_refY <= zero_gY + sensitivity;
                  acc_refX <= zero_gX;
                  acc_refZ <= zero_gZ;
                  counter <= 0;
                  player_ready <= 1;
                  state <= TESTPOS1;
                end
              else
                begin
                  counter <= counter + 1;
                  state <= INITIALIZE;
                end
            end

          // game mode: player must have hands poised to throw
          // back of the hand pointing down
          // z-axis must meet the criteria of ~ 1.33V and
          // y-axis must meet the criteria of ~1.66V for one second
          // before the go ahead LED gets turned

          else if((accZ <= (zero_gZ - sensitivity + ATREST_OFFSET))
          && (accZ >= (zero_gZ - sensitivity - ATREST_OFFSET)))
            begin
              if(counter == INIT_ONESEC)
                begin
                  acc_refY <= zero_gY;
                  acc_refX <= zero_gX;
                  acc_refZ <= zero_gZ - sensitivity;
                  counter <= 0;
                  player_ready <= 1;
                  state <= TESTPOS2;
                end
              else
```

```
                begin
                  counter <= counter + 1;
                  state <= INITIALIZE;
                end
            end

          // Used to test with the broken accelerometer such that
          // when the hands are parallel to the ground, and player moves forward
          // the ball will move forward (similar to position 1)

          else if((accZ <= (zero_gZ + sensitivity + ATREST_OFFSET))
          && (accZ >= (zero_gZ + sensitivity - ATREST_OFFSET)))
            begin
              if(counter == INIT_ONESEC)
                begin
                  acc_refY <= zero_gY;
                  acc_refX <= zero_gX;
                  acc_refZ <= zero_gZ + sensitivity;
                  counter <= 0;
                  player_ready <= 1;
                  state <= TESTPOS3;
                end
              else
                begin
                  counter <= counter + 1;
                  state <= INITIALIZE;
                end
            end
          else
            begin
              counter <= 0;
              state <= INITIALIZE;
              player_ready <= 0;
            end

        end

      // FOR POSITION ONE, WHERE THE PLAYER PUSHES THE BALL FORWARD
      TESTPOS1:
        begin
          if(accZ < (acc_refZ - ATREST_OFFSET)) // allow enough offset in -Z acc
            begin
              start <= 1; // assume the acceleration is decreasing
              state <= WAITPOS1; // wait until it comes back up to zerog
            end
          else
            begin
              start <= 0;
              state <= TESTPOS1;
            end
        end

      WAITPOS1:
        begin
          if(accZ > acc_refZ)
            begin
              release_velY <= ({25{velZ_in[25]}} ^ velZ_in) + velZ_in[25];
              release_velZ <= 25'd0;
              release_velX <= 25'd0;
              state <= BALL_RELEASE;
            end
          else state <= WAITPOS1;

        end

      // FOR POSITION TWO, PUSH UP AND FORWARD
      TESTPOS2:
        begin
          if(accZ < (acc_refZ - ATREST_OFFSET))
```

```
        begin
          start <= 1;
          state <= WAITPOS2;
        end
      else
        begin
          start <= 0;
          state <= TESTPOS2;
        end
    end

  WAITPOS2: // end with hands upright
    begin
      if(accZ > acc_refZ)
        begin
          release_velZ <= ({25{velZ_in[25]}} ^ velZ_in) + velZ_in[25];
          acc_refZ <= zero_gZ;
          acc_refY <= zero_gY + sensitivity;
          start <= 0; // zero the velocity
          state <= WAIT2POS2;
        end
      else state <= WAITPOS2;
    end

  WAIT2POS2:
    begin
      if (accZ < (acc_refZ - ATREST_OFFSET))
        begin
          start <= 1;
          state <= WAIT3POS2;
        end
      else state <= WAIT2POS2;
    end

  WAIT3POS2:
    begin
      if(accZ > acc_refZ)
        begin
          release_velY <= ({25{velZ_in[25]}} ^ velZ_in) + velZ_in[25];
          release_velX <= 0;
          state <= BALL_RELEASE;
        end
      else state <= WAIT3POS2;
    end

  // POSITION 3: PLAYER HANDS FLAT AND PUSH FORWARD
  // USED TO TEST THE SYSTEM WITH THE BROKEN Z AXIS ACCELEROMETER

  TESTPOS3:
    begin
      if (accY > (acc_refY + ATREST_OFFSET)) // y position positive
        begin
          start <= 1;
          state <= WAITPOS3;
        end
      else state <= TESTPOS3;
    end

  WAITPOS3:
    begin
      if (accY < acc_refY)  // comes back down past the refline
        begin
          release_velY <= velY_in;
          release_velZ <= 25'd0;
          release_velX <= 25'd0;
          state <= BALL_RELEASE;
        end
      else state <= WAITPOS3;
    end
```

```
          BALL_RELEASE:
            begin
               release_ball <= 1;
               state <= BALL_RELEASE;  // can only be reset by reset button
            end

          default: state <= INITIALIZE;

        endcase
    end

endmodule
```

conversion.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Chun Li
// 6.111 Final Project: Virtual Basketball
//
// Used to convert the output signals (release_velX, Y, Z) (mm/s) from the toss
// module into pixels per frame clock, so the display can use this velocity to
// calculate the trajectory of the ball.
// -1 inch = 25.4 mm => 25 + 4/10
// -80 pixels = 42 inches
// -1 second = 60 frame clocks
// -want output in pixels per frame clock
//
// -simplifies to 1/800 (to be as precise as we can b/c we already lose
// information during the integration.
//////////////////////////////////////////////////////////////////////////////////

module conversions(clk, reset, input_vel, output_vel);

    input clk;
    input reset;
    input [15:0] input_vel;   // 25 bit velocity from the toss module (mm/s)

    output [9:0] output_vel;  // pixels/frame clock

    wire [15:0] quotient;
    wire [9:0] remainder;

    division_conv Dconv1(
      .clk(clk),
      .sclr(reset),
      .dividend(input_vel),
      .divisor(10'd800),
      .quotient(quotient),
      .remainder(remainder),
      .rfd(rfd)
    );

    assign output_vel = quotient[9:0];

endmodule
```

basicLogic.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Jingwen Ouyang
// Module Name:    basicLogic
// Additional Comments: This is a basic control logic that calculates the ball's
//          the ball bounces off the 4 edges
//
//////////////////////////////////////////////////////////////////////////////////
module basicLogicBall(clock, frameClock, reset, vY, vZ, pY, pZ, ballY, ballZ);
    input clock;
    input frameClock;
    input reset;
    input [9:0] vY;
    input [9:0] vZ;
    input [9:0] pY;
    input [9:0] pZ;
    output reg [9:0] ballY;
    output reg [9:0] ballZ;
    reg [9:0] tempVY;
    reg [9:0] tempVZ;
    reg movingDown;
    reg movingRight;
    reg [1:0] state;

    parameter idle = 2'd0;
    parameter updateVelocity = 2'd1;
    parameter updatePosition = 2'd2;

    parameter deltaVZ = 1'd1;  //change of velocity each frame clock
    parameter floorPosZ = 10'd360;
    parameter rightLimit = 10'd639;

    reg [29:0] counter;

    always @ (posedge clock)

      begin

         if (counter < 10)    counter <= counter + 1;
         else counter <=0;

         if (reset)

            begin
               //default to idle state, and reset starting position and velocity for ball
               state <= idle;
               ballY <= pY;
               ballZ <= pZ;
               // for 3D i need to indicate 3 axis, so this need to be changed.
               // but for this basic 2D versoin, down means positive Z direction, Right means positive Y
direction
               // and there is no use of X direction which is into the screen
               movingDown <= 0;
               movingRight <= 1;
               tempVY <= vY;
               tempVZ <= vZ;
               counter <= 0; // delay velocity change
            end

         else

            begin

               case (state)

                  idle:
                     begin
                        if (~frameClock)  state <= idle;
                        else state<= updateVelocity;
                     end

                  //only z direction has acceleration so only Z direction has speed change
                  updateVelocity:
```

```
                  begin
                     // go to the next state without condition
                     state <= updatePosition;
                     // change velocity only when counter = 0;
                     // because the frameClock is too fast.. and i want to keep the deltaV
                     // as a whole number, the counter is used to make a slower clock
                     if (counter==0)
                        begin
                           if (~movingDown)
                              begin
                                 if (tempVZ < deltaVZ)
                                    begin
                                       movingDown <= 1;
                                       tempVZ <= deltaVZ - tempVZ;
                                    end
                                 else tempVZ <= tempVZ - deltaVZ;
                              end
                           else tempVZ <= tempVZ + deltaVZ;
                        end
                  end

               updatePosition:
                  begin
                     state <= idle;

                     //checking if it will bounce(this will be done in status module in final version

                     //for the Z direction, check upper and lower boundery
                     //in case of bouncing off the ceiling
                     if ((movingDown==0) & (ballZ < tempVZ))
                        begin
                           movingDown <= 1; // tell it to change direction, no change in speed
                           ballZ <= tempVZ - ballZ; // update the position
                        end
                     //in case of bouncing off the ground
                     else if ((movingDown==1)& ((ballZ+tempVZ)> floorPosZ))
                        begin
                           movingDown <=0; // tell it to change directoin
                           tempVZ <= {1'b0, tempVZ[9:1]}; //the bounce absorbs energy, devide V by half (need
to change)
                           ballZ <= floorPosZ-(tempVZ-(floorPosZ-ballZ)); //update the position
                        end
                     //normal movement
                     else if (movingDown) ballZ <= ballZ + tempVZ;
                     else ballZ <= ballZ - tempVZ;

                     //for the Y direction, check left and right boundery
                     //in case of bouncing off the left side
                     if ((movingRight==0) & (ballY < tempVY))
                        begin
                           movingRight <= 1; // tell it to change direction, no change in speed
                           ballY <= tempVY - ballY; // update the position
                        end
                     //in case of bouncing off the right
                     else if ((movingRight==1)& ((ballY+tempVY)> rightLimit))
                        begin
                           movingRight <=0; // tell it to change directoin, no change in speed
                           ballY <= rightLimit-(tempVY-(rightLimit-ballY)); //update the position
                        end
                     //normal movement
                     else if (movingRight) ballY <= ballY + tempVY;
                     else ballY <= ballY - tempVY;
                  end

               default:  state <= idle;

            endcase

         end

      end

endmodule
```

calculateForTwoD.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Engineer: Jingwen Ouyang
// Module Name:    calculateForTwoD
// Additional Comments: converts a 3D cordinate into 2D
//////////////////////////////////////////////////////////////////////////////////
module calculateForTwoD(clock, reset, start, frameClock, inX, inY, inZ,outX, outY,busy);
    input clock;
    input start;
    input reset;
    input frameClock;
    input [9:0] inX;
    input [9:0] inY;
    input [9:0] inZ;
    output reg busy;
    output reg [9:0] outX;
    output reg [9:0] outY;

    wire busyMul;
    wire busyDiv;
    wire [16:0] tempXsinA;
    wire [16:0] tempYsinB;
    wire [16:0] tempXcosA;
    wire [16:0] tempYcosB;
    wire [16:0] XsinA;
    wire [16:0] YsinB;
    wire [16:0] XcosA;
    wire [16:0] YcosB;

    reg enableMul;
    reg enableDiv;
    reg [3:0] state;
    reg [9:0] tempX;
    reg [9:0] tempY;

    //parameters for state
    parameter idle = 4'd0;
    parameter doMultiplication = 4'd1;
    parameter startMultiplication = 4'd2;
    parameter doDivision = 4'd3;
    parameter startDivision = 4'd4;
    parameter doSummation = 4'd5;
    parameter doOutput = 4'd6;
    parameter wait1 = 4'd7;
    parameter wait2 = 4'd8;

    //TODO: initial positions, subject to change
    //the two things need to go through 3Dto2D is ball and shaddow
    //i set it to the initial position of the ball because at that time, shadow won't be drawn
    parameter initialX = 10'd371;
    parameter initialY = 10'd359;


    //instantiates a divider and a multiplier
    twoDDiv myDivider(clock, reset, enableDiv, tempXsinA, tempYsinB, tempXcosA, tempYcosB, busyDiv, XsinA,
YsinB, XcosA, YcosB);
    twoDMul myMultiplier(clock, reset, enableMul, inX, inY, busyMul, tempXsinA, tempYsinB, tempXcosA,
tempYcosB);



    always @ (posedge clock)

      begin

         if (reset)
            begin
               state <= 3'd0;
               enableMul <= 1'd0;
               enableDiv <= 1'd0;
               outX <= initialX;
               outY <= initialY;
               tempX <= initialX;
               tempY <= initialY;
```

```
   end

else

   begin

      case (state)

         idle:

            begin
               if (~start)  state <= idle;
               else
                  begin
                     busy <= 1;
                     state <= startMultiplication;
                     enableMul <= 1;
                     enableDiv <= 0;
                  end
            end

         startMultiplication:

            begin
               state <= doMultiplication;
            end

         doMultiplication:

            begin
               if (busyMul) state <= doMultiplication;
               else
                  begin
                     state <= startDivision;
                     enableMul <= 0;
                     enableDiv <= 1;
                  end
            end

         startDivision:

            begin
               state <= doDivision;
            end

         doDivision:

            begin
               if (busyDiv)   state <= doDivision;
               else
                  begin
                     state <= doSummation;
                     enableDiv <= 0;
                     enableMul <= 0;
                  end
            end

         doSummation:

            begin
               state <= doOutput;
               //TODO: need to varify. this is depend on how the court is displayed, its view angle
               //(217,479) is where the origin of the 3D coordinate is in the 2D
               tempY<= 10'd479 - inZ - XsinA[9:0] - YsinB[9:0];
               tempX<= 10'd217 + XcosA[9:0] - YcosB[9:0];
            end

         doOutput:

            begin
               busy <= 0;
               state <= wait1;
               outX <= tempX;
               outY <= tempY;
            end
```

```
                //add delay so that the start signal is turned off before returning to the idle state
                wait1: state<=wait2;
                wait2: state<=idle;
                default: state <= idle;

            endcase

        end

    end


endmodule
```

colorMapping.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Engineer: Jingwen Ouyang
// Module Name:    colorMapping
// Additional Comments: maps a 4bit color index with a 24bit real color
//////////////////////////////////////////////////////////////////////////////////
module colorMapping(clock, colorIndex, color);
    input clock;
    input [3:0] colorIndex;
    output reg [23:0] color;

    //specify colors
    //these are the original mappng created by Paint when i creat this BMP file
    //just for testing
    parameter black = 24'h000000;               //0
    parameter darkerRed = 24'h800000;           //1
    parameter darkerGreen = 24'h008000;         //2
    parameter darkerYellow = 24'h808000;        //3
    parameter darkerNavy = 24'h000080;          //4
    parameter darkerPink = 24'h800080;          //5 reserve as transparent
    parameter darkerBlue = 24'h008080;          //6
    parameter darkerGray = 24'h808080;          //7
    parameter gray = 24'hC0C0C0;                //8
    parameter red = 24'hFF0000;                 //9
    parameter green = 24'h00FF00;               //10
    parameter yellow = 24'hFFFF00;              //11
    parameter navy = 24'h0000FF;                //12
    parameter pink = 24'hFF00FF;                //13
    parameter blue = 24'h00FFFF;                //14
    parameter white = 24'hFFFFFF;               //15

    //the colors I want
    parameter ball = 24'hFF3300; //reg replace green
    parameter background = 24'hFFFF99; // light yellow replace yellow
    parameter floor = 24'hFF9933; //orange replace darker yellow
    parameter rim = 24'hCC0000; //maroon replace red



    always @ (posedge clock)
      begin
        if (colorIndex == 4'b0000) color<=black;                //0
        else if (colorIndex == 4'b0001)  color <= darkerRed;    //1
        else if (colorIndex == 4'b0010)  color <= darkerGreen;  //2
        else if (colorIndex == 4'b0011)  color <= floor;  //3
        else if (colorIndex == 4'b0100)  color <= darkerNavy;   //4
        else if (colorIndex == 4'b0101)  color <= darkerPink;   //5 - transparent
        else if (colorIndex == 4'b0110)  color <= darkerBlue;   //6
        else if (colorIndex == 4'b0111)  color <= darkerGray;   //7
        else if (colorIndex == 4'b1000)  color <= gray;         //8
        else if (colorIndex == 4'b1001)  color <= rim;          //9
        else if (colorIndex == 4'b1010)  color <= ball;        //10
        else if (colorIndex == 4'b1011)  color <= background;       //11
        else if (colorIndex == 4'b1100)  color <= navy;         //12
        else if (colorIndex == 4'b1101)  color <= pink;         //13
        else if (colorIndex == 4'b1110)  color <= blue;         //14
        else if (colorIndex == 4'b1111)  color <= white;        //15
        else color <= darkerPink;
      end




Endmodule
```

colorMappingShadow.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Engineer: Jingwen Ouyang
// Module Name:    colorMappingShadow
// Additional Comments: maps a 4bit color index with a 24bit real color for the shadow
//
//////////////////////////////////////////////////////////////////////////////////
module colorMappingShadow(clock, colorIndex, color);
    input clock;
    input [3:0] colorIndex;
    output reg [23:0] color;

    //specify colors
    parameter darkerPink = 24'h800080;          //5 reserve as transparent
    parameter gray = 24'hC0C0C0;                //8


    //the shadow only has one color, if the color is transparent for the ball
    // it is transparent for the shadow; if the ball color is drawn, then
    // set the shadow to gray.
    // the shape for the shadow really should be a ellipse...
    always @ (posedge clock)
      begin
        if (colorIndex == 4'b0101)  color <= darkerPink;    //5 - transparent
        else color <= gray;
      end

endmodule
```

controllerVGA.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Engineer:   Jingwen Ouyang
// Create Date:    20:45:55 03/11/2007
// Module Name:    controllerVGA
// Additional Comments: created in lab4, guaranteed to work
//////////////////////////////////////////////////////////////////////////////////
module controllerVGA(reset,dcm, hsync, vsync, pixel_count, line_count, hblank, vblank);
    input reset;
    input dcm;
    output hblank;
    output vblank;
    output hsync;
    output vsync;
    output [9:0] pixel_count;
    output [9:0] line_count;
    wire hsync;
    wire vsync;
    wire hblank;
    wire vblank;
    reg [9:0] pixel_count;
    reg [9:0] line_count;

    //for real
    parameter activeVideoH = 640;
    parameter frontPorchH = 16;
    parameter syncPulseH = 96;
    parameter backPorchH = 48;
    parameter activeVideoV = 480;
    parameter frontPorchV = 11;
    parameter syncPulseV = 2;
    parameter backPorchV = 32;

//  //for testing
//  parameter activeVideoH = 4;
//  parameter frontPorchH = 1;
//  parameter syncPulseH = 2;
//  parameter backPorchH = 1;
//  parameter activeVideoV = 2;
//  parameter frontPorchV = 1;
//  parameter syncPulseV = 1;
//  parameter backPorchV = 1;

    // this is for future use in case the parameter change...
    parameter downH=activeVideoH+frontPorchH;
    parameter upH=downH+syncPulseH;
    parameter downV=activeVideoV+frontPorchV;
    parameter upV=downV+syncPulseV;
    parameter endPixel=upH+backPorchH-1;
    parameter endLine=upV+backPorchV;

    //just deal with the count
    always @ (posedge dcm)
      begin
         if (reset)
            begin
               pixel_count <= 10'd0;
               line_count <= 10'd0;
            end
         else
            begin
               pixel_count <= pixel_count +1;
               if (pixel_count==endPixel)
                  begin
                     pixel_count <= 0;
                     line_count<= (line_count==endLine)? 0 : line_count+1;
                  end
            end
      end


      //always block has delayas, assign avoids it
      //set them high (or low) base on the counts
      assign hblank = (pixel_count<activeVideoH || reset) ? 1 : 0;
```

```
      assign vblank = (line_count<activeVideoV || reset) ? 1 : 0;
      assign hsync = (pixel_count<downH || pixel_count>=upH || reset) ? 1 : 0;
      assign vsync = (line_count<downV || line_count>=upV || reset) ? 1 : 0;

endmodule
```

delay.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Engineer:   Jingwen Ouyang
// Create Date:    20:45:55 03/11/2007
// Module Name:    delay
// Additional Comments: created in lab 4, this is the delay for lab 4; I found that for this project
//       the delay should be fixed. that is the reason why there was a strap of misalignment
//       in the VGA display. However, i did not get time to fix it...
//////////////////////////////////////////////////////////////////////////////////
module delay(reset,clk, hsync_b, vsync_b, vga_out_hsync, vga_out_vsync);
    input clk;
    input reset;
    input hsync_b;
    input vsync_b;
    output vga_out_hsync;
    output vga_out_vsync;

    wire vga_out_hsync;
    wire vga_out_vsync;
    reg hsync1, hsync2, vsync1,vsync2;

    //delay
    always @ (posedge clk)
      begin
         if (reset)
            begin
               vsync2<=0;
               hsync2<=0;
            end
         else
            begin
               vsync1<=vsync_b;
               hsync1<=hsync_b;
               vsync2<=vsync1;
               hsync2<=hsync1;
            end
      end

    //assignment
    assign vga_out_hsync = hsync2;
    assign vga_out_vsync = vsync2;

endmodule
```

drawBeaver_backUp.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Engineer: Jingwen Ouyang
// Module Name:    drawbeaver
// Additional Comments: A base version that displays the beaver that shoots the ball with its tail
//
//////////////////////////////////////////////////////////////////////////////////
module drawbeaver(reset, clock, start, next, pixel_count, line_count, rgbBeaver);
    input clock;
    input reset;
    input start;
    input next;
    input [9:0] pixel_count;
    input [9:0] line_count;
    output [23:0] rgbBeaver;

    reg [3:0] colorIndex;
    reg [6:0] oneHzCount;
    reg [12:0] romAddress;
    reg [1:0] state;
    wire [3:0] romColorIndex;

    parameter startingX = 10'd346;
    parameter startingY = 10'd361;

    parameter idleBeaver = 2'd0;
    parameter startShooting = 2'd1;
    parameter movingBeaver = 2'd2;
    parameter finalBeaver = 2'd3;

    parameter idleRomAddress = 13'd0;
    parameter movingRomAddress = 13'd1650;
    parameter finalRomAddress = 13'd3300;

    parameter transparentIndex = 4'b0101;

    //start from Chun
    always @ (posedge clock)
      begin
         if (reset)
            begin
               state <= idleBeaver;
               romAddress <= idleRomAddress;
               colorIndex <= transparentIndex;
               oneHzCount <= 0;
            end
         else
            begin

               case (state)

                  idleBeaver:

                        if (~start)
                           begin
                              state<= idleBeaver;
                              //draw the beaver at the right place
                              if ((line_count>=startingY)&&(line_count<startingY+33))
                                 begin
                                    if ((pixel_count>=startingX)&&(pixel_count<startingX+50))
                                       begin
                                          romAddress <= romAddress + 1;
                                          colorIndex <= romColorIndex; //TODO: the timing here may not be right,
one clock cycle delay
                                       end
                                    else
                                       begin
                                          romAddress <= romAddress;
                                          colorIndex <= transparentIndex;
                                       end
                                 end
                              else
                                 begin
```

```
                              //reset address
                              romAddress <= idleRomAddress;
                              colorIndex <= transparentIndex;
                           end
                     end
                  else
                     begin
                        //go to a intermediate state
                        state <= startShooting;
                        romAddress <= movingRomAddress;
                        colorIndex <= transparentIndex;
                        //point the image to the right address
                     end

            startShooting:
                  //go to the next state unconditionally, it is needed, because start signal is short
                  state <= movingBeaver;

            movingBeaver:
                  //display it for a second
                  if (oneHzCount < 120)
                     begin
                        state <= movingBeaver;
                        //increment the couter
                        if ((line_count==479)&&(pixel_count==639)) oneHzCount <= oneHzCount + 1;
                        if ((line_count>=startingY)&&(line_count<startingY+33))
                           begin
                              if ((pixel_count>=startingX)&&(pixel_count<startingX+50))
                                    begin
                                       romAddress <= romAddress + 1;
                                       colorIndex <= romColorIndex; //TODO: the timing here may not be
right, one clock cycle delay
                                    end
                                 else
                                    begin
                                       romAddress <= romAddress;
                                       colorIndex <= transparentIndex;
                                    end
                              end
                           else
                              begin
                                 //reset address
                                 romAddress <= movingRomAddress;
                                 colorIndex <= transparentIndex;
                              end
                     end
                  else
                     begin
                        //stop the counter so it won't go too large
                        oneHzCount <= 6'd60;
                        state <= finalBeaver;
                        romAddress <= finalRomAddress;
                        colorIndex <= transparentIndex;
                     end

            finalBeaver:
                  //display it for a second
                  if (~next)
                     begin
                        state <= finalBeaver;
                        if ((line_count>=startingY)&&(line_count<startingY+33))
                           begin
                              if ((pixel_count>=startingX)&&(pixel_count<startingX+50))
                                 begin
                                    romAddress <= romAddress + 1;
                                    colorIndex <= romColorIndex; //TODO: the timing here may not be right,
one clock cycle delay
                                 end
                              else
                                 begin
                                    romAddress <= romAddress;
                                    colorIndex <= transparentIndex;
                                 end
                           end
                        else
                           begin
```

```
                                    //reset address
                                    romAddress <= finalRomAddress;
                                    colorIndex <= transparentIndex;
                                 end
                           end
                        else
                           begin
                              state <= idleBeaver;
                              romAddress <= idleRomAddress;
                              colorIndex <= transparentIndex;
                           end

                  default: state<= idleBeaver;

               endcase



            end
         end



    beaverrom myBeaverRom(
      .addr(romAddress),
      .clk(clock),
      .dout(romColorIndex)
    );

    colorMapping mybeaverColorMapping(clock, colorIndex, rgbBeaver); //can use a different mapping

endmodule
```

drawRectangle.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Engineer: Jingwen Ouyang
// Create Date:    22:28:55 03/22/2007
// Module Name:    drawRectangle
// Additional Comments:     Created in lab4
//////////////////////////////////////////////////////////////////////////////////
module drawRectangle(clk, reset, posX, posY, pixel_count, line_count, rgbSignal);
    input clk;
    input reset;
    input [9:0] posX;
    input [8:0] posY;
    input [9:0] pixel_count;
    input [9:0] line_count;
    output [23:0] rgbSignal;

    // default to ball color and width/length
    parameter width = 8;
    parameter length = 8;
    parameter color = 24'hFFFFFF;

    reg draw;

    always @ (posedge clk)
         if (reset)
            //globle reset
            draw <= 0;
         else
            //check if it is within drawing area, if so, signal to draw
            draw <= ((pixel_count >= posX)               //left edge
                     && (pixel_count < (posX+width))     //right edge
                     && (line_count >= posY)             //top edge
                     && (line_count < (posY+length)));   //button edge

    //if signal for draw is high, paint it to desired color
    assign rgbSignal= (draw)? color : 24'd0;

endmodule
```

lab4_labkit.v

```
//////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- the changes that made comparing to pong
// this is a basic version
//
//////////////////////////////////////////////////////////////////////////////


   //////////////////////////////////////////////////////////////////////////
   //
   // Lab 4 Components within I/O module
   // Jingwen Ouyang
   //
   //////////////////////////////////////////////////////////////////////////

   //
   // Generate a 31.5MHz pixel clock from clock_27mhz
   //

   wire pclk, pixel_clock;
   DCM pixel_clock_dcm (.CLKIN(clock_27mhz), .CLKFX(pclk));
   // synthesis attribute CLKFX_DIVIDE of pixel_clock_dcm is 6
   // synthesis attribute CLKFX_MULTIPLY of pixel_clock_dcm is 7
   // synthesis attribute CLK_FEEDBACK of pixel_clock_dcm  is "NONE"
   BUFG pixel_clock_buf (.I(pclk), .O(pixel_clock));

   //
   // VGA output signals
   //

   // Inverting the clock to the DAC provides half a clock period for signals
   // to propagate from the FPGA to the DAC.
   assign vga_out_pixel_clock = ~pixel_clock;

      assign vga_out_sync_b = 1'b1;


   wire [9:0] pixel_count;
   wire [9:0] line_count;
   wire theFrameClock;
   wire [9:0] ballPosY;
   wire [9:0] ballPosX;


   // power-on reset generation
   wire power_on_reset; // remain high for first 16 clocks
   SRL16 reset_sr (.D(1'b0), .CLK(clock_27mhz), .Q(power_on_reset),
   .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
   defparam reset_sr.INIT = 16'hFFFF;
   wire reset = power_on_reset | ~button_enter;

   //make sure to creat all the connection for the blocks need to color
   // wire [23:0] rgbSignal, rgbWall01, rgbBall, rgbWall02,rgbWall03;
   //wire [23:0] rgbSignal, rgbPaddle, rgbBall, rgbWall01, rgbWall02,rgbWall03;
   //   wire [23:0] rgbMIT01, rgbMIT02, rgbMIT03, rgbMIT04, rgbMIT05, rgbMIT06, rgbMIT07;
   wire [23:0] rgbSignal, rgbBall, rgbCourt, rgbBall,rgbBeaver;

   controllerVGA checkerBoardVGA (reset,pixel_clock, hsync, vsync, pixel_count,
      line_count, hblank, vblank);

   delay delayVGA (reset,pixel_clock, ~hsync, ~vsync, vga_out_hsync, vga_out_vsync);

   assign theFrameClock = (pixel_count == 639) && (line_count == 479);

   //get the position of the ball
   basicLogicBall myBallPositions(pixel_clock, theFrameClock, reset, 10'd2, 10'd10, 10'd50, 10'd360, ballPosX,
ballPosY);

   //draw the ball
   tempDrawBall myTestDrawBall(pixel_clock, reset, ballPosX, ballPosY, pixel_count, line_count, rgbBall);

   tempDrawBall myTempDrawBall(pixel_clock, reset, 10'd28, tempBallPosY, pixel_count, line_count, rgbBall);
   testBackground myTestBackground(pixel_clock, reset, pixel_count, line_count, rgbCourt);
```

```
   //TODO: get the start and next signal for this, next is bottom, start from Chun
   //drawbeaver(reset, clock, start, next, pixel_count, line_count, rgbBeaver)
   drawbeaver myDrawBeaver(reset, pixel_clock, 1'b1, 1'b0, pixel_count, line_count, rgbBeaver);

   rgbController myRGbController(pixel_clock, rgbBall,rgbCourt,rgbBeaver,rgbSignal);




   // VGA Output
   assign vga_out_red = rgbSignal[23:16];
   assign vga_out_green = rgbSignal[15:8];
   assign vga_out_blue = rgbSignal[7:0];
   assign vga_out_blank_b = hblank && vblank;

endmodule
```

tempDrawBall.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Engineer: Jingwen Ouyang
// Module Name:    tempDrawBall
// Additional Comments: this is a basic version
//
//////////////////////////////////////////////////////////////////////////////////
module tempDrawBall(clock, reset, posX, posY, pixel_count, line_count, rgbOut);
    input clock;
    input reset;
    input [9:0] posX;
    input [8:0] posY;
    input [9:0] pixel_count;
    input [9:0] line_count;
    output [23:0] rgbOut;

    reg [8:0] romAddress;
    wire [3:0] romColorIndex;
    reg [3:0] colorIndex;
    //ball is 18*18 pixel


    always @ (posedge clock)
      begin
         if (reset)
            romAddress <= 9'd0;
         //should draw ball
         else if ((line_count>=posY)&&(line_count<posY+18))
            begin
               if ((pixel_count>=posX)&&(pixel_count<posX+18))
                  begin
                     romAddress <= romAddress + 1;
                     colorIndex <= romColorIndex; //TODO: the timing here may not be right, one clock cycle
delay
                  end
               else
                  begin
                     romAddress <= romAddress;
                     colorIndex <= 4'b0101; //will be treat as transparent
                  end
            end
         else
            begin
               romAddress <= 9'd0;
               colorIndex <= 4'b0101; //will be treat as transparent
            end
      end

    ballrom myBallRom(
      .addr(romAddress),
      .clk(clock),
      .dout(romColorIndex)
    );

    colorMapping myTestBallColorMapping(clock, colorIndex, rgbOut);
endmodule
```

twoDDiv.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Engineer: Jingwen Ouyang
// Module Name:    twoDDiv
// Additional Comments: divides for the 3D to 2D
//
//////////////////////////////////////////////////////////////////////////////////
module twoDDiv(clock, reset, enableDivider, tempXsinA, tempYsinB, tempXcosA, tempYcosB, busyDiv, XsinA, YsinB,
XcosA, YcosB);
    input clock;
    input reset;
    input enableDivider;
    input [16:0] tempXsinA;
    input [16:0] tempYsinB;
    input [16:0] tempXcosA;
    input [16:0] tempYcosB;
    output reg busyDiv;
    output [16:0] XsinA;
    output [16:0] YsinB;
    output [16:0] XcosA;
    output [16:0] YcosB;

    reg [4:0] count;
    reg state;

    parameter divisor = 7'd100;

    //parameters for states
    parameter idle = 1'b0;
    parameter working = 1'b1;

    //instantiate dividors
    divider divXsinA (
      .clk(clock),
      .dividend(tempXsinA),
      .divisor(divisor),
      .quotient(XsinA),
      .ce(enableDivider)
    );

    divider divYsinB (
      .clk(clock),
      .dividend(tempYsinB),
      .divisor(divisor),
      .quotient(YsinB),
      .ce(enableDivider)
    );

    divider divXcosA (
      .clk(clock),
      .dividend(tempXcosA),
      .divisor(divisor),
      .quotient(XcosA),
      .ce(enableDivider)
    );

    divider divYcosB (
      .clk(clock),
      .dividend(tempYcosB),
      .divisor(divisor),
      .quotient(YcosB),
      .ce(enableDivider)
    );

    always @ (posedge clock)

      begin

         if (reset)
            begin
               state <= idle;
               busyDiv <= 0;
               count<= 0;
```

```
            end

         else
            begin

               case (state)

                  idle:
                     begin
                        if (~enableDivider) state <= idle;
                        else
                           begin
                              state <= working;
                              busyDiv <= 1;
                           end
                     end

                  working:
                     begin
                        if (count<25)
                           begin
                              busyDiv <=1;
                              state <= working;
                              count <=count+1;
                           end
                        else
                           begin
                              state <= idle;
                              count <= 0;
                              busyDiv <= 0;
                           end
                     end

                  default: state <= idle;
               endcase
            end
      end



endmodule
```

twoDMul.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Engineer: Jingwen Ouyang
// Module Name:    twoDMul
// Additional Comments: divides for the 3D to 2D
//
//////////////////////////////////////////////////////////////////////////////////
module twoDMul(clock, reset, enableMultiplier, inX, inY, busyMul, tempXsinA, tempYsinB, tempXcosA, tempYcosB);
    input clock;
    input reset;
    input enableMultiplier;
    input [9:0] inX;
    input [9:0] inY;
    output reg busyMul;
    output [16:0] tempXsinA;
    output [16:0] tempYsinB;
    output [16:0] tempXcosA;
    output [16:0] tempYcosB;
    reg state;
    reg [3:0] count;

    // a fixed frame projection
    // depend on how the court is displayed, the view angle
    parameter sina = 7'd49;
    parameter sinb = 7'd87;
    parameter cosa = 7'd24;
    parameter cosb = 7'd97;


    //parameters for states
    parameter idle = 1'b0;
    parameter working = 1'b1;

    //instantiating the multipliers
    // no enable, no clock
    multiplier mulXsinA (
      .a(inX),
      .b(sina),
      .o(tempXsinA)
    );

    multiplier mulYsinB (
      .a(inY),
      .b(sinb),
      .o(tempYsinB)
    );

    multiplier mulXcosA (
      .a(inX),
      .b(cosa),
      .o(tempXcosA)
    );

    multiplier mulYcosB (
      .a(inY),
      .b(cosb),
      .o(tempYcosB)
    );

    always @ (posedge clock)

      begin

         if (reset)
            begin
               state <= idle;
               busyMul <= 0;
               count<= 0;
            end

         else
            begin
```

```
            case (state)

               idle:
                  begin
                     if (~enableMultiplier) state <= idle;
                     else
                        begin
                           state <= working;
                           busyMul <= 1;
                        end
                  end

               working:
                  begin
                     if (count<10)
                        begin
                           state <= working;
                           count <= count+1;
                           busyMul <=1;
                        end
                     else
                        begin
                           state <= idle;
                           count <= 0;
                           busyMul <= 0;
                        end
                  end
               default: state <= idle;
            endcase
         end
      end

endmodule
```

UPGRADE

drawBall.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Engineer: Jingwen Ouyang
// Module Name:    drawBall
// Additional Comments: This is the version that did work properly yet...
//
//////////////////////////////////////////////////////////////////////////////////
module drawBall(clock, reset, frameClock, start, ballX, ballY, ballZ, line_count, pixel_count,
                  ballVX, ballVY, ballVZ, repeat1, repeat2, repeat3, busy, rgbBall);
    input clock;
    input reset;
    input frameClock;
    input start;
    input repeat1;
    input repeat2;
    input repeat3;
    input [9:0] pixel_count;
    input [9:0] line_count;
    input [9:0] ballX;
    input [9:0] ballY;
    input [9:0] ballZ;
    input [0:0] ballVX;
    input [0:0] ballVY;
    input [0:0] ballVZ;
    output reg busy;
    output [23:0] rgbBall;

    wire [9:0] twoDX;
    wire [9:0] twoDY;
    wire [9:0] myBallX;
    wire [9:0] myBallY;
    wire [9:0] myBallZ;
    wire [4:0] status;
    wire positiveX;
    wire positiveY;
    wire positiveZ;
    wire [3:0] scoreRomColorIndex;
    wire [3:0] sadFaceRomColorIndex;
```

```
wire [3:0] ballRomColorIndex;
reg [3:0] colorIndex;
reg [12:0] scoreRomAddress;
reg [13:0] sadFaceRomAddress;
reg [8:0] ballRomAddress;
//start signals
reg startGetBallPos;
reg startGetStatus;
reg start2D;
wire busy2D;
wire busyGetBallPos;
wire busyGetStatus;
//states
reg [4:0] state;
parameter idle = 5'd0;
parameter getStatus = 5'd1;
parameter getBallPos = 5'd2;
parameter w21 = 5'd3;
parameter w22 = 5'd4;
parameter threeD2twoD = 5'd5;
parameter w31 = 5'd6;
parameter w32 = 5'd7;
parameter getRGB = 5'd8;

parameter transparent = 24'h800080;

//status parameters
parameter keepGoing = 5'd1;
parameter score = 5'd2;
parameter miss = 5'd3;
parameter collideFloor = 5'd4;
parameter collideRim = 5'd5;
parameter collidePole = 5'd6;
parameter collideBoard = 5'd7;

//start from draw beaver
always @ (posedge clock)
  begin
     if (reset)
        begin
           state <= 0;
           busy <= 0;
           colorIndex <= 4'b0101; //will be treat as transparent
        end
     else
        begin
           case (state)

              idle:

                 if (~start)
                    begin
                       state <= idle;
                       busy <= 0;
                       colorIndex <= 4'b0101; //will be treat as transparent;
                    end
                 else
                    begin
                       state <= getStatus;
                       startGetStatus <= 1;
                       busy <= 1;
                       //TODO: set the inputs to getStatus
                    end

              getStatus:

                 begin
                    startGetStatus<=0;
                    if (busyGetStatus) state<=getStatus;
                    else if (status == miss)
                       begin
                          state<=w21;
                          busy<=0;
                          if ((line_count>=200)&&(line_count<300))
                             begin
                                if ((pixel_count>=200)&&(pixel_count<300))
```

```
                                    begin
                                       sadFaceRomAddress <= sadFaceRomAddress + 1;
                                       colorIndex <= sadFaceRomColorIndex; //TODO: the timing here may not be
right, one clock cycle delay
                                    end
                                 else
                                    begin
                                       sadFaceRomAddress <= sadFaceRomAddress;
                                       colorIndex <= 4'b0101; //will be treat as transparent
                                    end
                              end
                        end
                     else if (status == score)
                        begin
                           state<=w21;
                           busy<=0;
                           if ((line_count>=200)&&(line_count<300))
                              begin
                                 if ((pixel_count>=200)&&(pixel_count<280))
                                    begin
                                       scoreRomAddress <= scoreRomAddress + 1;
                                       colorIndex <= scoreRomColorIndex; //TODO: the timing here may not be
right, one clock cycle delay
                                    end
                                 else
                                    begin
                                       sadFaceRomAddress <= sadFaceRomAddress;
                                       colorIndex <= 4'b0101; //will be treat as transparent
                                    end
                              end
                        end
                     else
                        begin
                           startGetBallPos <= 1;
                           state<=getBallPos;
                        end
                  end



               getBallPos:

                  begin
                     startGetBallPos <= 0;
                     if (busyGetBallPos) state <= getBallPos;
                     //if it is done go the the wait states
                     else
                        begin
                           state <= threeD2twoD;
                           start2D <=1;
                        end
                  end

               threeD2twoD:
                  begin
                     start2D<=0;
                     if (busy2D) state<= threeD2twoD;
                     else
                        busy <= 0;
                        state<=w31;
                        begin
                           busy <= 0;
                           state <= threeD2twoD;
                           if ((line_count>=twoDY)&&(line_count<twoDY+18))
                              begin
                                 if ((pixel_count>=twoDX)&&(pixel_count<twoDX+18))
                                    begin
                                       ballRomAddress <= ballRomAddress + 1;
                                       colorIndex <= ballRomColorIndex; //TODO: the timing here may not be
right, one clock cycle delay
                                    end
                                 else
                                    begin
                                       ballRomAddress <= ballRomAddress;
                                       colorIndex <= 4'b0101; //will be treat as transparent
                                    end
```

```
                              end
                           else
                              begin
                                 ballRomAddress <= 9'd0;
                                 colorIndex <= 4'b0101; //will be treat as transparent
                              end
                        end
                  end

                //wait so that the start signal is turned off outside
                w21:    state <= w22;
                w22:    state <= idle;
                w31:    state <= w32;
                w32:    state <= idle;
                default : state<=idle;

             endcase

          end
       end

    calculateForTwoD myCalculateForTwoD(clock, reset, start2D,frameClock, myBallX, myBallY, myBallZ,twoDX,
twoDY, busy2D);

    getBallPosition myGetBallPos(clock, frameClock, reset, startGetBallPos, status, repeat1, repeat2, repeat3,
         ballX, ballY, ballZ, ballVX, ballVY, ballVZ, myBallX, myBallY, myBallZ, myVX, myVY, myVZ,
busyGetBallPos);

    getStatusLogic myGetStatusLogic(clock, reset, startGetStatus, myBallX, myBallY, myBallZ, positiveX,
positiveY, positiveZ,
                  myVX, myVY, myVZ, status, busyGetStatus);

    ballrom myFinalBallRom(
      .addr(ballRomAddress),
      .clk(clock),
      .dout(ballRomColorIndex)
    );

    sadfacerom mySadFaceRom(
    .addr(sadFaceRomAddress),
      .clk(clock),
      .dout(sadFaceRomColorIndex)
    );

    scoredrom myScoreRom(
    .addr(scoreRomAddress),
      .clk(clock),
      .dout(scoreRomColorIndex)
    );

   colorMapping myBallColorMapping(clock, colorIndex, rgbBall);

endmodule
```

drawBeaver.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Engineer: Jingwen Ouyang
// Module Name:    drawbeaver
// Additional Comments: A updated version that displays the beaver that shoots the ball with its tail
//          then it will stay there, but the ball supposed to travel along the calculated trajectory
//
//////////////////////////////////////////////////////////////////////////////////
module drawbeaver(reset, clock, frameClock, start, next,
                  pixel_count, line_count, ballX, ballY, ballZ,
                  ballVX, ballVY, ballVZ, repeat1, repeat2, repeat3,
                  rgbBall, rgbBeaver);
    input clock;
    input reset;
    input start;
    input next;
    input frameClock;
    input [9:0] pixel_count;
    input [9:0] line_count;
    input repeat1;
    input repeat2;
    input repeat3;
    input [9:0] ballX;
    input [9:0] ballY;
    input [9:0] ballZ;
    input [0:0] ballVX;
    input [0:0] ballVY;
    input [0:0] ballVZ;
    output [23:0] rgbBall;
    output [23:0] rgbBeaver;

    reg [3:0] colorIndex;
    reg [6:0] oneHzCount;
    reg [12:0] romAddress;
    reg [1:0] state;
    wire [3:0] romColorIndex;

    parameter startingX = 10'd346;
    parameter startingY = 10'd361;

    parameter idleBeaver = 2'd0;
    parameter startShooting = 2'd1;
    parameter movingBeaver = 2'd2;
    parameter finalBeaver = 2'd3;

    parameter idleRomAddress = 13'd0;
    parameter movingRomAddress = 13'd1650;
    parameter finalRomAddress = 13'd3300;

    reg startDrawBall;

    parameter transparentIndex = 4'b0101;

    //start from Chun
    always @ (posedge clock)
      begin
          if (reset)
             begin
                state <= idleBeaver;
                romAddress <= idleRomAddress;
                colorIndex <= transparentIndex;
                oneHzCount <= 0;
             end
          else
             begin

                case (state)

                   idleBeaver:

                         if (~start)
                            begin
                               state<= idleBeaver;
```

```
                        //draw the beaver at the right place
                        if ((line_count>=startingY)&&(line_count<startingY+33))
                           begin
                              if ((pixel_count>=startingX)&&(pixel_count<startingX+50))
                                 begin
                                    romAddress <= romAddress + 1;
                                    colorIndex <= romColorIndex; //TODO: the timing here may not be right,
one clock cycle delay
                                 end
                              else
                                 begin
                                    romAddress <= romAddress;
                                    colorIndex <= transparentIndex;
                                 end
                           end
                        else
                           begin
                              //reset address
                              romAddress <= idleRomAddress;
                              colorIndex <= transparentIndex;
                           end
                     end
                  else
                     begin
                        //go to a intermediate state
                        state <= startShooting;
                        romAddress <= movingRomAddress;
                        colorIndex <= transparentIndex;
                        //point the image to the right address
                     end

            startShooting:
                  //go to the next state unconditionally, it is needed, because start signal is short
                  state <= movingBeaver;

            movingBeaver:
                  //display it for a second
                  if (oneHzCount < 120)
                     begin
                        state <= movingBeaver;
                        //increment the couter
                        if ((line_count==479)&&(pixel_count==639)) oneHzCount <= oneHzCount + 1;
                        if ((line_count>=startingY)&&(line_count<startingY+33))
                           begin
                              if ((pixel_count>=startingX)&&(pixel_count<startingX+50))
                                    begin
                                       romAddress <= romAddress + 1;
                                       colorIndex <= romColorIndex; //TODO: the timing here may not be
right, one clock cycle delay
                                    end
                                 else
                                    begin
                                       romAddress <= romAddress;
                                       colorIndex <= transparentIndex;
                                    end
                              end
                           else
                              begin
                                 //reset address
                                 romAddress <= movingRomAddress;
                                 colorIndex <= transparentIndex;
                              end
                     end
                  else
                     begin
                        //stop the counter so it won't go too large
                        oneHzCount <= 6'd120;
                        state <= finalBeaver;
                        romAddress <= finalRomAddress;
                        colorIndex <= transparentIndex;
                        startDrawBall<=1;
                     end

            finalBeaver:
                  //display it for a second
                  begin
```

```
                        if (~next)
                           begin
                              state <= finalBeaver;

                              if ((line_count>=startingY)&&(line_count<startingY+33))
                                 begin
                                    if ((pixel_count>=startingX)&&(pixel_count<startingX+50))
                                       begin
                                          romAddress <= romAddress + 1;
                                          colorIndex <= romColorIndex; //TODO: the timing here may not be
right, one clock cycle delay
                                       end
                                    else
                                       begin
                                          romAddress <= romAddress;
                                          colorIndex <= transparentIndex;
                                       end
                                 end
                              else
                                 begin
                                    //reset address
                                    romAddress <= finalRomAddress;
                                    colorIndex <= transparentIndex;
                                 end
                           end
                        else
                           begin
                              state <= idleBeaver;
                              romAddress <= idleRomAddress;
                              colorIndex <= transparentIndex;
                              startDrawBall<=0;
                           end
                     end

                  default: state<= idleBeaver;

               endcase



            end
         end

    //start, ballX, ballY, ballZ, ballVX, ballVY, ballVZ, come from Chun
    //repeat1, repeat2, repeat3, come from labkid
    drawBall myDrawBall(clock, ballReset, frameClock, start, ballX, ballY, ballZ, line_count, pixel_count,
                  ballVX, ballVY, ballVZ, repeat1, repeat2, repeat3, rgbBall);

    beaverrom myBeaverRom(
      .addr(romAddress),
      .clk(clock),
      .dout(romColorIndex)
    );

    assign ballReset = (reset || next);

    colorMapping mybeaverColorMapping(clock, colorIndex, rgbBeaver); //can use a different mapping

endmodule
```

drawNumbers.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Engineer: Jingwen Ouyang
// Module Name:    drawNumbers
// Additional Comments: this is in the upgrated version
//
//////////////////////////////////////////////////////////////////////////////////
module drawNumbers(clock, reset, start, next, pixel_count, line_count, status, rgbNumber);
    input clock;
    input reset;
    input start;
    input next;
    input [9:0] pixel_count;
    input [9:0] line_count;
    input [4:0] status;
    output [23:0] rgbNumber;

//  reg [4:0] count1;
//  reg [4:0] count2;
//  reg [4:0] count3;
//  reg [4:0] count4;
//  reg [4:0] count5;
//  reg [4:0] count6;

    reg [10:0] romAddress;
    reg [10:0] romAddress1;
    reg [10:0] romAddress2;
    reg [10:0] romAddress3;
    reg [10:0] romAddress4;
    reg [10:0] romAddress5;
    reg [10:0] romAddress6;
    reg [10:0] tempRomAddress1;
    reg [10:0] tempRomAddress2;
    reg [10:0] tempRomAddress3;
    reg [10:0] tempRomAddress4;
    reg [10:0] tempRomAddress5;
    reg [10:0] tempRomAddress6;

    reg [1:0] state;
    parameter idle = 2'd0;
    parameter setInitialRomAddress = 2'd1;
    parameter updateTempAddresses = 2'd2;
    parameter getRGB =2'd3;

    //status checklist parameters
    parameter score = 5'd2;
    parameter miss = 5'd3;

    //parameters for positions
    parameter line1 = 10'd70;
    parameter line2 = 10'd82;
    parameter pixel1 = 10'd458;
    parameter pixel2 = 10'd469;
    parameter pixel3 = 10'd480;
    parameter pixel4 = 10'd541;
    parameter pixel5 = 10'd552;
    parameter pixel6 = 10'd563;

    parameter number9location = 11'd864;
    parameter transparentIndex = 4'b0101;
    reg [3:0] colorIndex;
    wire [3:0] romColorIndex;

    always @ (posedge clock)
      begin
         if (reset)
            begin
               romAddress <=0;
               romAddress1 <=0;
               romAddress2 <=0;
               romAddress3 <=0;
               romAddress4 <=0;
               romAddress5 <=0;
```

```
      romAddress6 <=0;
   end
else
   begin
      case (state)
         idle:
            if (status == miss)
               begin
                  //go to the next state
                  state <= setInitialRomAddress;
                  // display format: 12 font, 8*12 size, and says: "score" out of "total"
                  //only the total move to the next score
                  //if the 3rd digit is less than number9location,
                  //it goes to the next digit, otherwise it becomes 0
                  //and the 2nd digit increments, so on so forth
                  //but there is no 4th digit so 3rd just go from 9-0 meaning 999->0
                  if (romAddress6<number9location)   romAddress6<= romAddress6+96;
                  else
                     begin
                        romAddress6<= 0;
                        if (romAddress5<number9location)   romAddress5<= romAddress5+96;
                        else
                           begin
                              romAddress5<=0;
                              if (romAddress5<number9location)   romAddress4<= romAddress4+96;
                              else romAddress4<=0;
                           end
                     end
               end
            else if (status == score)
               begin
                  //go to the next state
                  state <= setInitialRomAddress;
                  //both the total and score move up
                  //if the 3rd digit is less than number9location,
                  //it goes to the next digit, otherwise it becomes 0
                  //and the 2nd digit increments, so on so forth
                  //but there is no 4th digit so 3rd just go from 9-0 meaning 999->0
                  //for total
                  if (romAddress6<number9location)   romAddress6<= romAddress6+96;
                  else
                     begin
                        romAddress6<= 0;
                        if (romAddress5<number9location)   romAddress5<= romAddress5+96;
                        else
                           begin
                              romAddress5<=0;
                              if (romAddress5<number9location)   romAddress4<= romAddress4+96;
                              else romAddress4<=0;
                           end
                     end
                  //for score
                  if (romAddress3<number9location)   romAddress3<= romAddress3+96;
                  else
                     begin
                        romAddress3<= 0;
                        if (romAddress2<number9location)   romAddress2<= romAddress5+96;
                        else
                           begin
                              romAddress2<=0;
                              if (romAddress1<number9location)   romAddress1<= romAddress4+96;
                              else romAddress1<=0;
                           end
                     end
               end
            else
               // no change in numbers
               state <= idle;

         setInitialRomAddress:

            begin

               state <= updateTempAddresses;
               //romAddress1 -> romaddress6 stores the initial address for each digit
               //they shouldn't be changes when drawing
```

```
      //so i need another temperary variable
      tempRomAddress1 <= romAddress1;
      tempRomAddress2 <= romAddress2;
      tempRomAddress3 <= romAddress3;
      tempRomAddress4 <= romAddress4;
      tempRomAddress5 <= romAddress5;
      tempRomAddress6 <= romAddress6;

   end


updateTempAddresses:
   //this one gets the address value for the rom
   begin

      state<= getRGB;

      if ((line_count>=line1) && (line_count<line2))
         begin
            //update them if they are in range
            if ((pixel_count>=pixel1)&&(pixel_count<pixel1+8))
                  tempRomAddress1 <= tempRomAddress1 +1;
            else tempRomAddress1 <= tempRomAddress1;
            if ((pixel_count>=pixel2)&&(pixel_count<pixel2+8))
                  tempRomAddress2 <= tempRomAddress2 +1;
            else tempRomAddress2 <= tempRomAddress2;
            if ((pixel_count>=pixel3)&&(pixel_count<pixel3+8))
                  tempRomAddress3 <= tempRomAddress3 +1;
            else tempRomAddress3 <= tempRomAddress3;
            if ((pixel_count>=pixel4)&&(pixel_count<pixel4+8))
                  tempRomAddress4 <= tempRomAddress4 +1;
            else tempRomAddress4 <= tempRomAddress4;
            if ((pixel_count>=pixel5)&&(pixel_count<pixel5+8))
                  tempRomAddress5 <= tempRomAddress5 +1;
            else tempRomAddress5 <= tempRomAddress5;
            if ((pixel_count>=pixel6)&&(pixel_count<pixel6+8))
                  tempRomAddress6 <= tempRomAddress6 +1;
            else tempRomAddress6 <= tempRomAddress6;
         end
      else
         begin
            tempRomAddress1 <= tempRomAddress1;
            tempRomAddress2 <= tempRomAddress2;
            tempRomAddress3 <= tempRomAddress3;
            tempRomAddress4 <= tempRomAddress4;
            tempRomAddress5 <= tempRomAddress5;
            tempRomAddress6 <= tempRomAddress6;
         end
   end


 getRGB:

   begin
      //TODO
      state <= idle;

      if ((line_count>=line1) && (line_count<line2))
         begin
            if ((pixel_count>=pixel1)&&(pixel_count<pixel1+8))
               begin
                  romAddress <= tempRomAddress1;
                  colorIndex <= romColorIndex;
               end
            else if ((pixel_count>=pixel2)&&(pixel_count<pixel2+8))
               begin
                  romAddress <= tempRomAddress2;
                  colorIndex <= romColorIndex;
               end
            else if ((pixel_count>=pixel3)&&(pixel_count<pixel3+8))
               begin
                  romAddress <= tempRomAddress3;
                  colorIndex <= romColorIndex;
               end
            else if ((pixel_count>=pixel4)&&(pixel_count<pixel4+8))
               begin
```

```
                                romAddress <= tempRomAddress4;
                                colorIndex <= romColorIndex;
                             end
                          else if ((pixel_count>=pixel5)&&(pixel_count<pixel5+8))
                             begin
                                romAddress <= tempRomAddress5;
                                colorIndex <= romColorIndex;
                             end
                          else if ((pixel_count>=pixel6)&&(pixel_count<pixel6+8))
                             begin
                                romAddress <= tempRomAddress6;
                                colorIndex <= romColorIndex;
                             end
                          else
                             begin
                                romAddress <= romAddress;
                                colorIndex <= transparentIndex;
                             end
                       end
                    else
                       begin
                          romAddress <= romAddress;
                          colorIndex <= transparentIndex;
                       end
                 end

              default: state<= idle;
           endcase
        end
     end


    numberrom myNumberRom(
      .addr(romAddress),
      .clk(clock),
      .dout(romColorIndex)
    );

    colorMapping myNumberColorMapping(clock, colorIndex, rgbNumber); //can use a different mapping


endmodule
```

drawShadow.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Engineer: Jingwen Ouyang
// Module Name:    drawShadow
// Additional Comments: Similar as the tempDrawBall£¬ also in the updated version
//
//////////////////////////////////////////////////////////////////////////////////
module drawShadow(reset, clock, shadowX, shadowY, shadowZ, pixel_count, line_count, rgbShadow);
    input reset;
    input clock;
    input [9:0] shadowX;
    input [9:0] shadowY;
    input [9:0] shadowZ;
    input [9:0] pixel_count;
    input [9:0] line_count;
    output reg [23:0] rgbShadow;

    parameter transparentColor = 24'h800080;
    parameter gray = 24'hC0C0C0;


    //TODO: this need to be updated for 3D
    always @ (posedge clock)
      begin
         if (pixel_count > (shadowX+9))
            begin
               if (line_count > (shadowY+9))
                  begin
                     if (((pixel_count-shadowX-9)*(pixel_count-shadowX-9)
                           +(line_count-shadowY-9)*(line_count-shadowY-9))< 81)
                           rgbShadow <= gray;
                     else rgbShadow <= transparentColor;
                  end
               else
                  begin
                     if (((pixel_count-shadowX-9)*(pixel_count-shadowX-9)
                           +(shadowY-line_count+9)*(shadowY-line_count+9))< 81)
                           rgbShadow <= gray;
                     else rgbShadow <= transparentColor;
                  end
            end
         else
            begin
               if (line_count > (shadowY+9))
                  begin
                     if (((shadowX-pixel_count+9)*(shadowX-pixel_count+9)
                           +(line_count-shadowY-9)*(line_count-shadowY-9))< 81)
                           rgbShadow <= gray;
                     else rgbShadow <= transparentColor;
                  end
               else
                  begin
                     if (((shadowX-pixel_count+9)*(shadowX-pixel_count+9)
                           +(shadowY-line_count+9)*(shadowY-line_count+9))< 81)
                           rgbShadow <= gray;
                     else rgbShadow <= transparentColor;
                  end
            end
      end


endmodule
```

getBallPosition.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    04:09:36 05/15/2007
// Design Name:
// Module Name:    getBallPosition
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module getBallPosition(clock, frameClock, reset, start, status, repeat1, repeat2, repeat3,
          initialBallX, initialBallY, initialBallZ, initialBallVX, initialBallVY, initialBallVZ,
          ballX, ballY, ballZ, ballVX, ballVY, ballVZ, busy);
    input clock;
    input frameClock;
    input reset;
    input start;
    input [4:0] status;
    input repeat1;
    input repeat2;
    input repeat3;
    input [9:0] initialBallX;
    input [9:0] initialBallY;
    input [9:0] initialBallZ;
    input [9:0] initialBallVX;
    input [9:0] initialBallVY;
    input [9:0] initialBallVZ;
    //TODO: may need a temperary variable, if need delay
    output reg [9:0] ballX;
    output reg [9:0] ballY;
    output reg [9:0] ballZ;
    output reg busy;
    output reg [9:0] ballVX;
    output reg [9:0] ballVY;
    output reg [9:0] ballVZ;
    //indicates the direction of speed
    reg positiveX;
    reg positiveY;
    reg positiveZ;
    //for replay
    reg [1:0] count;
    reg [1:0] countLimit;

    //state assignment
    //TODO: state/status parameter assignment, should be the same as the ones in getStatus
    reg [4:0] state;
    parameter idle = 5'd0;
    parameter keepGoing = 5'd1;
    parameter score = 5'd2;
    parameter miss = 5'd3;
    parameter collideFloor = 5'd4;
    parameter collideRim = 5'd5;
    parameter collidePole = 5'd6;
    parameter collideBoard = 5'd7;
    parameter startGame = 5'd8;
    parameter justWait1 = 5'd9;
    parameter justWait2 = 5'd10;

    //TODO: get the right gravity
    //from calculation 10m/s^2 = 1pixel/frameClock^2 =)
    parameter gravity = 1;
```

```
always @ (posedge clock)

  begin

     if (reset)

        begin
           //default to idle state, and reset starting position and velocity for ball
           state <= idle;
           //get the initial values
           ballX <= initialBallX;
           ballY <= initialBallY;
           ballZ <= initialBallZ;
           ballVX <= initialBallVX;
           ballVY <= initialBallVY;
           ballVZ <= initialBallVZ;
           positiveX <= 1;
           positiveY <= 1;
           positiveZ <= 1;
        end

     else

        begin
              case (state)

                 idle:
                    begin

                       count<= 2'd0;
                       busy<=0;
                       if (start)
                          //should only be called if it is not miss/score
                          begin
                             state<= startGame;
                             //set the count parameter;
                             //to help slow down the clock for replay
                             //remember to update count each frameClock
                             countLimit <= 2'd0;
                             busy<=1;
                          end
                       else if ((~start) && (repeat1))
                          begin
                             state<= startGame;
                             countLimit <= 2'd0;
                             busy<=1;
                          end
                       else if ((~start)&&(~repeat1) && (repeat2))
                          begin
                             state<= startGame;
                             countLimit <= 2'd1;
                             busy<=1;
                          end
                       else if ((~start)&&(~repeat1) && (~repeat2) && (repeat3))
                          begin
                             state<= startGame;
                             countLimit <= 2'd2;
                             busy<=1;
                          end
                       else
                          begin
                             state<=idle;
                             countLimit <= 2'd0;
                          end
                    end


                 //this add a delay
                 startGame:

                    begin

                       if (frameClock)
                          begin
                             //update count each frameClock
                             //frameClock should only be high for one clock cycle
```

```
                        //TODO: check the case when countLimit = 0;
                        if (count < countLimit) count <= count + 1 ;
                        else
                           begin
                              count <= 0;
                              //the priority takes into account of the possible movement of the ball

                              //this can be expanded to different parts for the Board/pole,
                              if (status == collideRim) state <= collideRim;
                              else if (status == collideBoard) state <= collideBoard;
                              else if (status == collidePole)  state <= collidePole;
                              else if (status == collideFloor) state <= collideFloor;
                              //any other situation, let the ball keep going
                              //however, since it is only called if it doesn't not collide anywhere and
not game over
                              //which actually only leaves the situation of keepGoing
                              else state <= keepGoing;
                           end

                     end
               end

            collideRim:

               begin

                  //set next state
                  state <= keepGoing;

                  //change directions for all axies
                  //no change for speed, assume energy is conserved here
                  if (positiveX) positiveX <= 0;
                  else positiveX <= 1;
                  if (positiveY) positiveY <= 0;
                  else positiveY <= 1;
                  if (positiveZ) positiveZ <= 0;
                  else positiveZ <= 1;

               end


            collideBoard:

               begin

                  //set next state
                  state <= keepGoing;

                  //change directions for y axies only
                  //no change for speed, assume energy is conserved here
                  if (positiveY) positiveY <= 0;
                  else positiveY <= 1;

               end

            collidePole:

               begin

                  //set next state
                  state <= keepGoing;

                  //change directions for y axies only
                  //no change for speed, assume energy is conserved here
                  if (positiveY) positiveY <= 0;
                  else positiveY <= 1;

               end

            collideFloor:

               begin

                  //set next state
                  state <= keepGoing;
```

```
                        //change direction for z axis
                        if (positiveZ) positiveZ <= 0;
                        else positiveZ <= 1;
                        //change the speeds also, because energy loss here is quite obvious
                        //i want to divide by 2, here i just removed the last bit,
                        //however, this gives me possible osillation
                        //TODO: see if there is a better way to change this
//                      ballVX <= {0, ballVX[9:1]};
//                      ballVY <= {0, ballVY[9:1]};
//                      ballVY <= {0, ballVY[9:1]};
                        ballVX <= ballVX[9:1];
                        ballVY <= ballVY[9:1];
                        ballVY <= ballVY[9:1];
                     end

                  keepGoing:

                     begin

                        // calculate the position to output
                        if (positiveX) ballX <= ballX + ballVX;
                        else ballX <= ballX - ballVX;

                        if (positiveY) ballY <= ballY + ballVY;
                        else ballY <= ballY - ballVY;

                        if (positiveZ) ballZ <= ballZ + ballVZ;
                        else ballZ <= ballZ - ballVZ;

                        //update ballVZ due to gravity
                        if (positiveZ)
                           begin
                              if (gravity > ballVZ)
                                 begin
                                    //reverse direction
                                    positiveZ <= 0;
                                    ballVZ <= gravity - ballVZ;
                                 end
                              else ballVZ <= ballVZ - gravity;
                           end
                        else  ballVZ <= ballVZ + gravity;

                        state <= justWait1;
                        busy <= 0;

                     end

                  //adds delay to make sure the start signal is turned off before return to idle
                  //other wise it will start again, which means it will never stop
                  justWait1:     state <= justWait2;

                  justWait2:     state <= idle;

                  default:       state <= idle;

               endcase

            end
         end


endmodule
```

getIntersectionInd.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Engineer: Jingwen Ouyang
// Module Name:    getIntersectionInd
// Additional Comments: gets the intersection of the ball with the possible plane in one axis
//
//////////////////////////////////////////////////////////////////////////////////

module getIntersectionInd(clock, reset, start, ballX1, ballX2, ball1, ballV0, ballV1, int1, busy);
    input clock;
    input reset;
    input start;
    //ball0, ballV0, int0 are the reference values
    //the others are the ones need to be calculated
    input [9:0] ballX1; //ballX1 > ballX2;
    input [9:0] ballX2;
    input [9:0] ball1;
    input [9:0] ballV0;
    input [9:0] ballV1;
    output reg [9:0] int1;
    output reg busy;


    //ball0, ballV0 are the reference values
    //the others are the ones need to be calculated

    //internal variables
    reg [9:0] difference;
    reg [9:0] counter;
    reg [19:0] dividend1;
    wire [19:0] tempDividend1;
    wire [19:0] quotient1;

    //states are for timing, to add delay
    reg [1:0] state;
    parameter idle = 2'd0;
    parameter multiplication = 2'd1;
    parameter division = 2'd2;
    parameter justwait = 2'd3;

    //the equation needs to be done is int1 = ball1 + ballV1 * (int0 - ball0) / ballV0

    always @ (posedge clock)
      begin
         if (reset)
            begin
               state <= idle;
               int1 <= 10'd0;
               counter <= 5'd0;
               busy <= 0;
            end
         else
            begin

               case (state)

                  idle:

                     if (~start)
                        begin
                           state<= idle;
                           counter <= 5'd0;
                           busy <= 0;
                        end
                     else
                        begin
                           state<= multiplication;
                           busy <= 1'b1;
                           difference <= ballX1 - ballX2;
                        end


                  multiplication:
```

```
                  begin
                     state <= division;
                     dividend1 <= tempDividend1;
                  end


               division:

                  //division has latency, needs to wait
                  if (counter<60)
                     begin
                        state <= division;
                        counter<= counter+1;
                     end
                  else
                     begin
                        state <= justwait;
                        busy <= 0;
                        int1 <= ball1+quotient1[9:0];
                        counter<= counter+1;
                     end


               justwait:

                  // make sure the major machine has turned off the start signal
                  if (counter< 32)
                     begin
                        counter <= counter + 1;
                        state <=justwait;
                     end
                  else
                     begin
                        state <= idle;
                        counter <= 10'd0;
                     end


               default: state<=idle;

            endcase
         end
      end

      intersectionmul mul1(
         .a(difference),
         .b(ballV1),
         .o(tempDividend1)
      );

      intersectiondiv div1(
         .clk(clock),
         .dividend(dividend1),
         .divisor(ballV0),
         .quotient(quotient1)
      );


endmodule
```

getShadowPos.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Engineer£ºJingwen Ouyang
// Module Name:    getShadowPos
//
//////////////////////////////////////////////////////////////////////////////////
module getShadowPos(clock, ballX, ballY, shadowX, shadowY, shadowZ);
    input clock;
    input [9:0] ballX;
    input [9:0] ballY;
    output reg [9:0] shadowX = 10'd0;
    output reg [9:0] shadowY = 10'd0;
    output wire [9:0] shadowZ;

    //no matter what vertical height (Z) the ball is at, shadow is always on the ground
    always @ (posedge clock)
      begin
         shadowX<= ballX;
         shadowY<= ballY;
      end

    assign shadowZ = 10'd0;

endmodule
```

getStatus.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Engineer: Jingwen Ouyang
// Module Name:    getStatus
// Additional Comments: I think this piece of code can be shortened if i creat some smaller modules
//          for each situation... but did not have time to clean this
//
//////////////////////////////////////////////////////////////////////////////////
module getStatus(clock, reset, start,ballX, ballY, ballZ, positiveX, positiveY, positiveZ,
                 ballVX, ballVY, ballVZ, status, busy);
    //inputs
    input clock;
    input reset;
    input start;
    input [9:0] ballX;
    input [9:0] ballY;
    input [9:0] ballZ;
    input positiveX;
    input positiveY;
    input positiveZ;
    input [9:0] ballVX;
    input [9:0] ballVY;
    input [9:0] ballVZ;

    //outputs
    output reg [4:0] status;
    output reg busy;

    //talk to getIntersection
    reg startGetIntersection;
    wire busyGetIntersection1;
    wire busyGetIntersection2;
    reg [9:0] ball11; //ballX1 > ballX2;
    reg [9:0] ball12;
    reg [9:0] ball21
    reg [9:0] ball22;
    reg [9:0] ball1;
    reg [9:0] ball2;
    reg [9:0] ballV0;
    reg [9:0] ballV1;
    reg [9:0] ballV2;
    wire [9:0] int1;
    wire [9:0] int2;



    //bits to indicate which plane it might intersect
    //set them to right one, not just flip it
    reg intersectWithRimPlane;
    reg intersectWithBoardPlane;
    reg intersectWithPolePlane;
    reg intersectWithFloorPlane;
    reg intersectWithBoundary;

    //the important x points
    parameter XStart =10'd0;
    parameter XBoard1 =10'd108;
    parameter XPole1 =10'd165;
    parameter XPole2 =10'd189;
    parameter XBoard2 =10'd246;
    parameter XEnd =10'd400;

    //the important x points
    parameter YStart =10'd0;
    parameter YBoard =10'd217;
    parameter YPole =10'd225;
    parameter YEnd =10'd400;

    //the important z points
    parameter ZFloor =10'd0;
    parameter ZBoard1 = 10'd206;
    parameter ZBoard2 = 10'd286;
    parameter ZEnd =10'd400;
```

```
//TODO: for rim
parameter rimX = 10'd177;
parameter rimY = 10'd188;
parameter rimZ = 10'd230;

parameter ballRadiusSquared = 10'd324;

//state assignemnt
reg [2:0] state;
parameter idle = 3'd0;
parameter getIntersectStatus = 3'd1;
parameter getIntersectRim = 3'd2;
parameter getIntersectBoard = 3'd3;
parameter getIntersectPole = 3'd4;
parameter getIntersectFloor = 3'd5;
parameter setStatus1 = 3'd6;
parameter setStatus2 = 3'd7;



//TODO: output status parameter assignment
parameter keepGoing = 4'd0;
parameter score = 4'd2;
parameter miss = 4'd3;
parameter collideBoard1 = 4'd4;
parameter collideBoard2 = 4'd5;
parameter collideBoard3 = 4'd6;
parameter collideBoard4 = 4'd7;
parameter collideFloor = 4'd8;
parameter collideRim = 4'd9;
parameter collidePole = 4'd10;


always @ (posedge clock)

  begin

      //based on how the ball travels, i simplified it to a few cases,
      //ignored some irregular travel path
      //I check to see if it scores first, then see if it will run into the board or the pole
      //then it could be either bounce off floor or out of bounds

      //the ball might collide with the board/pole, rim, or the floor
      //1. colliding with the floor the speed change is one case
      //2. to make it real colliding with the board/pole will have some variaties.
      //   I divided the ball into 9 boxes (3 x 3) so that the velocity will change depend on
      //   which part the ball is colliding, the change of velocity will be different
      //3. the ball should stop moving if it roll out of bounds (out of 0-400 in any axis)
      //   or bouce on the floor twice (which means there will have a counter)

      //need state machine because some calculation takes time, especially when
      //it computes the possible intersection points

      //TODO: in this basic version, there is actually no change of velocity in Z...
      //because video lack that axie, so i will not check the case where it bounce off the floor
      //at negative axis...

      if (reset)
        begin
           state <= idle;
           startGetIntersection <= 0;
           busy <= 0;
           intersectWithRimPlane<= 0;
           intersectWithBoardPlane<= 0;
           intersectWithPolePlane<= 0;
           intersectWithFloorPlane<= 0;
           intersectWithBoundary<= 0;
        end
      else

        begin

           case (state)

              idle:
```

```
if (~start)
   begin
      state<= idle;
      busy<= 0;
   end

else

   begin

      state<= getIntersectStatus;
      busy <= 1;

      //check where it might intersect

      //check with Rim
      if (positiveZ)
         begin
            if ((ballZ <= rimZ) && ((ballZ+ballVZ) >= rimZ))
               intersectWithRimPlane <= 1;
            else intersectWithRimPlane <= 0;
         end
      else
         begin
            if ((ballZ >= rimZ) && ((ballZ+ballVZ) <= rimZ))
               intersectWithRimPlane <= 1;
            else intersectWithRimPlane <= 0;
         end

      //check with board
      if (positiveY)
         begin
            if ((ballY <= YBoard) && ((ballY+ballVY) >= YBoard))
               intersectWithBoardPlane <= 1;
            else intersectWithBoardPlane <= 0;
         end
      else
         begin
            if ((ballY >= YBoard) && ((ballY+ballVY) <= YBoard))
               intersectWithBoardPlane <= 1;
            else intersectWithBoardPlane <= 0;
         end

      //check with pole
      if (positiveY)
         begin
            if ((ballY <= YPole) && ((ballY+ballVY) >= YPole))
               intersectWithPolePlane <= 1;
            else intersectWithBoardPlane <= 0;
         end
      else
         begin
            if ((ballY >= YPole) && ((ballY+ballVY) <= YPole))
               intersectWithPolePlane <= 1;
            else intersectWithBoardPlane <= 0;
         end


      //check with floor
      if ((~positiveZ) && (ballZ >= ZFloor) && (ballVZ >= ballZ))
               intersectWithFloorPlane <= 1;
      else intersectWithFloorPlane <= 0;

      //check with x,y,z boundarys except floor; 0-400
      //Z
      if (positiveZ && (ballZ <= ZEnd) && ((ballZ+ballVZ) >= ZEnd))
         intersectWithBoundary <= 1;
      //Y
      else if (positiveY && (ballY <= YEnd) && ((ballY+ballVY) >= YEnd))
         intersectWithBoundary <= 1;
      else if ((~positiveY) && (ballY >= YStart) && (ballY<=ballVY))
         intersectWithBoundary <= 1;
      //X
      else if (positiveX && (ballX <= XEnd) && ((ballX+ballVX) >= XEnd))
         intersectWithBoundary <= 1;
```

```
            else if ((~positiveX) && (ballX >= XStart) && (ballX<=ballVX))
               intersectWithBoundary <= 1;
            else intersectWithBoundary <= 0;

         end

      getIntersectStatus:

         if ((~intersectWithRimPlane) && (~intersectWithBoardPlane) &&
            (~intersectWithPolePlane) && (~intersectWithFloorPlane) &&
            (intersectWithBoundary))
            begin
               //always set the status and set busy to 0 when goes to state "setStatus1"
               state <= setStatus1;
               status <= miss;
               busy1<=0;
               busy2<=0;
            end
         else if ((~intersectWithRimPlane) && (~intersectWithBoardPlane) &&
            (~intersectWithPolePlane) && (intersectWithFloorPlane))
            begin
               // if going to getIntersectXXX then set the right signal
               state <= getIntersectFloor;
               startGetIntersection<=1;
               //ballX1 > ballX2 , and ballV0 in Z axies,
               //set ball1 in X, ball2 in Y
               ballV0 <= ballVZ;
               ballV1 <= ballVX;
               ballV2 <= ballVY;
               //i know it has to be (~positiveZ) and ballVZ > ballZ
               if (positiveX)
                  begin
                     ball1 <= ballX;
                     ball11<= ballZ;
                     ball12<= ZFloor;
                  end
               else
                  begin
                     ball1 <= ballX - ballVX;
                     ball11<= ballVZ
                     ball12<= ballZ;
                  end
               if (positiveY)
                  begin
                     ball2 <= ballY;
                     ball21<=ballZ;
                     ball12<=ZFloor;
                  end
               else
                  begin
                     ball2 <= ballY - ballVY;
                     ball21<=ballVZ;
                     ball22<=ballZ;
                  end
            end
         else if ((~intersectWithRimPlane) && (~intersectWithBoardPlane) &&
            (intersectWithPolePlane))
            begin
               state <= getIntersectPole;
               startGetIntersection<=1;
               //ballX1 > ballX2 , and ballV0 in Y axies,
               //set ball1 in X, ball2 in Z
               ballV0 <= ballVY;
               ballV1 <= ballVX;
               ballV2 <= ballVZ;
               if (positiveX)
                  begin
                     ball1 <= ballX;
                     if (positiveY)
                        //same direction
                        begin
                           ball11<= YPole;
                           ball12<= ballY;
                        end
                     else
                        //opposite direction
```

```
                begin
                   ball11<=ballY;
                   ball12<=Ypole;
                end
          end
       else
          begin
             ball1 <= ballX - ballVX;
             if (positiveY)
                begin
                   ball11<=ballVY+ballY;
                   ball12<=Ypole;
                end
             else
                begin
                   ball11<=YPole;
                   ball12<=ballY-ballVY;
                end
          end
       if (positiveZ)
          begin
             ball2 <= ballZ;
             if (positiveY)
                //same direction
                begin
                   ball11<= YPole;
                   ball12<= ballY;
                end
             else
                //opposite direction
                begin
                   ball11<=ballY;
                   ball12<=Ypole;
                end
          end
       else
          begin
             ball2 <= ballZ - ballVZ;
             if (positiveY)
                begin
                   ball11<=ballVY+ballY;
                   ball12<=Ypole;
                end
             else
                begin
                   ball11<=YPole;
                   ball12<=ballY-ballVY;
                end
          end
   end
else if ((~intersectWithRimPlane) && (intersectWithBoardPlane))
   begin
      state <= getIntersectBoard;
      startGetIntersection<=1;
      //ballX1 > ballX2 , and ballV0 in Y axies,
      //set ball1 in X, ball2 in Z
      ballV0 <= ballVY;
      ballV1 <= ballVX;
      ballV2 <= ballVZ;
      if (positiveX)
         begin
            ball1 <= ballX;
            if (positiveY)
               //same direction
               begin
                  ball11<= YBoard;
                  ball12<= ballY;
               end
            else
               //opposite direction
               begin
                  ball11<=ballY;
                  ball12<=YBoard;
               end
         end
       else
```

```
            begin
               ball1 <= ballX - ballVX;
               if (positiveY)
                  begin
                     ball11<=ballVY+ballY;
                     ball12<=Ypole;
                  end
               else
                  begin
                     ball11<=YPole;
                     ball12<=ballY-ballYZ;
                  end
            end
         if (positiveZ)
            begin
               ball2 <= ballZ;
               if (positiveY)
                  //same direction
                  begin
                     ball11<= YBoard;
                     ball12<= ballY;
                  end
               else
                  //opposite direction
                  begin
                     ball11<=ballY;
                     ball12<=YBoard;
                  end
            end
         else
            begin
               ball2 <= ballZ - ballVZ;
               if (positiveY)
                  begin
                     ball11<=ballVY+ballY;
                     ball12<=YBoard;
                  end
               else
                  begin
                     ball11<=YBoard;
                     ball12<=ballY-ballVY;
                  end
            end
      end
   else if (intersectWithRimPlane)
      begin
         state <= getIntersectRim;
         startGetIntersection<=1;
         //ballX1 > ballX2 , and ballV0 in Z axies,
         //set ball1 in X, ball2 in Y
         ballV0 <= ballVZ;
         ballV1 <= ballVX;
         ballV2 <= ballVY;
         if (positiveX)
            begin
               ball2 <= ballX;
               if (positiveZ)
                  //same direction
                  begin
                     ball11<= rimZ;
                     ball12<= ballZ;
                  end
               else
                  //opposite direction
                  begin
                     ball11<=ballZ;
                     ball12<=rimZ;
                  end
            end
         else
            begin
               ball2 <= ballX - ballVX;
               if (positiveZ)
                  begin
                     ball11<=ballVZ+ballZ;
                     ball12<=rimZ;
```

```
                        end
                     else
                        begin
                           ball11<=rimZ;
                           ball12<=ballZ-ballVZ;
                        end
                  end
               if (positiveY)
                  begin
                     ball2 <= ballY;
                     if (positiveZ)
                        //same direction
                        begin
                           ball11<= rimZ;
                           ball12<= ballZ;
                        end
                     else
                        //opposite direction
                        begin
                           ball11<=ballZ;
                           ball12<=rimZ;
                        end
                  end
               else
                  begin
                     ball2 <= ballY - ballVY;
                     if (positiveZ)
                        begin
                           ball11<=ballVZ+ballZ;
                           ball12<=rimZ;
                        end
                     else
                        begin
                           ball11<=rimZ;
                           ball12<=ballZ-ballVZ;
                        end
                  end
            end
         else
            // in case of ((~intersectWithRimPlane) &&
            //(~intersectWithBoardPlane) && (~intersectWithPolePlane) &&
            //(~intersectWithFloorPlane) && (~intersectWithBoundary))
            begin
               //always set the status and set busy to 0 when goes to state "setStatus1"
               state <= setStatus1;
               status <= keepGoing;
               busy <= 0;
            end

      getIntersectRim:

         begin
            //it is ok if it is just high for one clock cycle
            //see simulation wave
            startGetIntersection<=0;
            //busy won't be high right away, it need one clock cycle
            //if getIntersection is working, stay in this one
            if (startGetIntersection || busyGetIntersection1 || busyGetIntersection2) state <=
getIntersectRim;
            //if it is done, check if it does intersect with the rim
            //check with the radius
            //int0 is in Z, int1 is in X, int2 is in Y
            //TODO: check this
            //there is built in multiplier, but need positive inputs
            else if ((~positiveZ) &&
                     ((((int1 > rimX) && (int2 > rimY) &&
                      (((int1-rimX)*(int1-rimX)+(int2-rimY)*(int2-rimY))< ballRadiusSquared))
                      |||((int1 > rimX) && (int2 < rimY) &&
                         (((int1-rimX)*(int1-rimX)+(rimY-int2)*(rimY-int2))< ballRadiusSquared))
                      |||((int1 < rimX) && (int2 < rimY) &&
                         (((rimX-int1)*(rimX-int1)+(rimY-int2)*(rimY-int2))< ballRadiusSquared))
                      |||((int1 < rimX) && (int2 > rimY) &&
                         (((rimX-int1)*(rimX-int1)+(int2-rimY)*(int2-rimY))<ballRadiusSquared))))
               //if it is in range then we scores
               begin
                  state <= setStatus1;
```

```
                        status <= score;
                        busy <= 0;
                     end
                  // other wise go to the others
                  else if ((~intersectWithBoardPlane) && (~intersectWithPolePlane) &&
(~intersectWithFloorPlane) &&
                        (intersectWithBoundary))
                     begin
                        //always set the status and set busy to 0 when goes to state "setStatus1"
                        state <= setStatus1;
                        status <= miss;
                        busy1<=0;
                        busy2<=0;
                     end
                  else if ((~intersectWithBoardPlane) && (~intersectWithPolePlane) &&
(intersectWithFloorPlane))
                     begin
                        // if going to getIntersectXXX then set the right signal
                        state <= getIntersectFloor;
                        startGetIntersection<=1;
                        //ballX1 > ballX2 , and ballV0 in Z axies,
                        //set ball1 in X, ball2 in Y
                        ballV0 <= ballVZ;
                        ballV1 <= ballVX;
                        ballV2 <= ballVY;
                        //i know it has to be (~positiveZ) and ballVZ > ballZ
                        if (positiveX)
                           begin
                              ball1 <= ballX;
                              ball11<= ballZ;
                              ball12<= ZFloor;
                           end
                        else
                           begin
                              ball1 <= ballX - ballVX;
                              ball11<= ballVZ
                              ball12<= ballZ;
                           end
                        if (positiveY)
                           begin
                              ball2 <= ballY;
                              ball21<=ballZ;
                              ball12<=ZFloor;
                           end
                        else
                           begin
                              ball2 <= ballY - ballVY;
                              ball21<=ballVZ;
                              ball22<=ballZ;
                           end
                     end
                  else if ((~intersectWithBoardPlane) && (intersectWithPolePlane))
                     begin
                        state <= getIntersectPole;
                        startGetIntersection<=1;
                        //ballX1 > ballX2 , and ballV0 in Y axies,
                        //set ball1 in X, ball2 in Z
                        ballV0 <= ballVY;
                        ballV1 <= ballVX;
                        ballV2 <= ballVZ;
                        if (positiveX)
                           begin
                              ball1 <= ballX;
                              if (positiveY)
                                 //same direction
                                 begin
                                    ball11<= YPole;
                                    ball12<= ballY;
                                 end
                              else
                                 //opposite direction
                                 begin
                                    ball11<=ballY;
                                    ball12<=Ypole;
                                 end
                           end
```

```
            else
               begin
                  ball1 <= ballX - ballVX;
                  if (positiveY)
                     begin
                        ball11<=ballVY+ballY;
                        ball12<=Ypole;
                     end
                  else
                     begin
                        ball11<=YPole;
                        ball12<=ballY-ballVY;
                     end
               end
            if (positiveZ)
               begin
                  ball2 <= ballZ;
                  if (positiveY)
                     //same direction
                     begin
                        ball11<= YPole;
                        ball12<= ballY;
                     end
                  else
                     //opposite direction
                     begin
                        ball11<=ballY;
                        ball12<=Ypole;
                     end
               end
            else
               begin
                  ball2 <= ballZ - ballVZ;
                  if (positiveY)
                     begin
                        ball11<=ballVY+ballY;
                        ball12<=Ypole;
                     end
                  else
                     begin
                        ball11<=YPole;
                        ball12<=ballY-ballVY;
                     end
               end
         end
      else if (intersectWithBoardPlane)
         begin
            state <= getIntersectBoard;
            startGetIntersection<=1;
            //ballX1 > ballX2 , and ballV0 in Y axies,
            //set ball1 in X, ball2 in Z
            ballV0 <= ballVY;
            ballV1 <= ballVX;
            ballV2 <= ballVZ;
            if (positiveX)
               begin
                  ball1 <= ballX;
                  if (positiveY)
                     //same direction
                     begin
                        ball11<= YBoard;
                        ball12<= ballY;
                     end
                  else
                     //opposite direction
                     begin
                        ball11<=ballY;
                        ball12<=YBoard;
                     end
               end
            else
               begin
                  ball1 <= ballX - ballVX;
                  if (positiveY)
                     begin
                        ball11<=ballVY+ballY;
```

```
                                ball12<=Ypole;
                             end
                          else
                             begin
                                ball11<=YPole;
                                ball12<=ballY-ballYZ;
                             end
                       end
                    if (positiveZ)
                       begin
                          ball2 <= ballZ;
                          if (positiveY)
                             //same direction
                             begin
                                ball11<= YBoard;
                                ball12<= ballY;
                             end
                          else
                             //opposite direction
                             begin
                                ball11<=ballY;
                                ball12<=YBoard;
                             end
                       end
                    else
                       begin
                          ball2 <= ballZ - ballVZ;
                          if (positiveY)
                             begin
                                ball11<=ballVY+ballY;
                                ball12<=YBoard;
                             end
                          else
                             begin
                                ball11<=YBoard;
                                ball12<=ballY-ballVY;
                             end
                       end
                 end
              else
                 // in case of ((~intersectWithRimPlane) &&
                 //(~intersectWithBoardPlane) && (~intersectWithPolePlane) &&
                 //(~intersectWithFloorPlane) && (~intersectWithBoundary))
                 begin
                    //always set the status and set busy to 0 when goes to state "setStatus1"
                    state <= setStatus1;
                    status <= keepGoing;
                    busy <= 0;
                 end



           getIntersectBoard:

              begin
                 //it is ok if it is just high for one clock cycle
                 //see simulation wave
                 startGetIntersection<=0;
                 //busy won't be high right away, it need one clock cycle
                 //if getIntersection is working, stay in this one
                 if (startGetIntersection || busyGetIntersection1 || busyGetIntersection2) state <=
getIntersectBoard;
                 //if it is done, check if it does intersect with the board
                 //check with the
                 //int1 is in X, int2 is in Z
                 //TODO: check this
                 else if ((int1<XBoard2)&&(int1>XBoard1)&&(int2<ZBoard2)&&(int2>ZBoard1))
                    //if it is in range then it bounce off the board
                    //TODO: depend on where it ball hit, it should give different possibilities for
acceleration
                    begin
                       state <= setStatus1;
                       status <= collideBoard1;
                       busy <= 0;
                    end
```

```
                        // other wise go to the others
                        else if ((~intersectWithPolePlane) && (~intersectWithFloorPlane) &&
(intersectWithBoundary))
                           begin
                              //always set the status and set busy to 0 when goes to state "setStatus1"
                              state <= setStatus1;
                              status <= miss;
                              busy1<=0;
                              busy2<=0;
                           end
                        else if ((~intersectWithPolePlane) && (intersectWithFloorPlane))
                           begin
                              // if going to getIntersectXXX then set the right signal
                              state <= getIntersectFloor;
                              startGetIntersection<=1;
                              //ballX1 > ballX2 , and ballV0 in Z axies,
                              //set ball1 in X, ball2 in Y
                              ballV0 <= ballVZ;
                              ballV1 <= ballVX;
                              ballV2 <= ballVY;
                              //i know it has to be (~positiveZ) and ballVZ > ballZ
                              if (positiveX)
                                 begin
                                    ball1 <= ballX;
                                    ball11<= ballZ;
                                    ball12<= ZFloor;
                                 end
                              else
                                 begin
                                    ball1 <= ballX - ballVX;
                                    ball11<= ballVZ
                                    ball12<= ballZ;
                                 end
                              if (positiveY)
                                 begin
                                    ball2 <= ballY;
                                    ball21<=ballZ;
                                    ball12<=ZFloor;
                                 end
                              else
                                 begin
                                    ball2 <= ballY - ballVY;
                                    ball21<=ballVZ;
                                    ball22<=ballZ;
                                 end
                           end
                        else if (intersectWithPolePlane)
                           begin
                              state <= getIntersectPole;
                              startGetIntersection<=1;
                              //ballX1 > ballX2 , and ballV0 in Y axies,
                              //set ball1 in X, ball2 in Z
                              ballV0 <= ballVY;
                              ballV1 <= ballVX;
                              ballV2 <= ballVZ;
                              if (positiveX)
                                 begin
                                    ball1 <= ballX;
                                    if (positiveY)
                                       //same direction
                                       begin
                                          ball11<= YPole;
                                          ball12<= ballY;
                                       end
                                    else
                                       //opposite direction
                                       begin
                                          ball11<=ballY;
                                          ball12<=Ypole;
                                       end
                                 end
                              else
                                 begin
                                    ball1 <= ballX - ballVX;
                                    if (positiveY)
                                       begin
```

```
                              ball11<=ballVY+ballY;
                              ball12<=Ypole;
                           end
                        else
                           begin
                              ball11<=YPole;
                              ball12<=ballY-ballVY;
                           end
                     end
                  if (positiveZ)
                     begin
                        ball2 <= ballZ;
                        if (positiveY)
                           //same direction
                           begin
                              ball11<= YPole;
                              ball12<= ballY;
                           end
                        else
                           //opposite direction
                           begin
                              ball11<=ballY;
                              ball12<=Ypole;
                           end
                     end
                  else
                     begin
                        ball2 <= ballZ - ballVZ;
                        if (positiveY)
                           begin
                              ball11<=ballVY+ballY;
                              ball12<=Ypole;
                           end
                        else
                           begin
                              ball11<=YPole;
                              ball12<=ballY-ballVY;
                           end
                     end
               end
            else
               // in case of ((~intersectWithRimPlane) &&
               //(~intersectWithBoardPlane) && (~intersectWithPolePlane) &&
               //(~intersectWithFloorPlane) && (~intersectWithBoundary))
               begin
                  //always set the status and set busy to 0 when goes to state "setStatus1"
                  state <= setStatus1;
                  status <= keepGoing;
                  busy <= 0;
               end



         getIntersectPole:

            begin
               //it is ok if it is just high for one clock cycle
               //see simulation wave
               startGetIntersection<=0;
               //busy won't be high right away, it need one clock cycle
               //if getIntersection is working, stay in this one
               if (startGetIntersection || busyGetIntersection1 || busyGetIntersection2) state <=
getIntersectPole;
               //if it is done, check if it does intersect with the board
               //check with the
               //int1 is in X, int2 is in Z
               //TODO: check this
               else if ((int1<XPole2)&&(int1>XPole1)&&(int2<ZBoard1)&&(int2>ZFloor))
                  //if it is in range then it bounce off the pole
                  //TODO: depend on where it ball hit, it should give different possibilities for
acceleration
                  begin
                     state <= setStatus1;
                     status <= collidePole;
                     busy <= 0;
```

```
                        end
                     // other wise go to the others
                     else if ((~intersectWithFloorPlane) && (intersectWithBoundary))
                        begin
                           //always set the status and set busy to 0 when goes to state "setStatus1"
                           state <= setStatus1;
                           status <= miss;
                           busy1<=0;
                           busy2<=0;
                        end
                     else if (intersectWithFloorPlane)
                        begin
                           // if going to getIntersectXXX then set the right signal
                           state <= getIntersectFloor;
                           startGetIntersection<=1;
                           //ballX1 > ballX2 , and ballV0 in Z axies,
                           //set ball1 in X, ball2 in Y
                           ballV0 <= ballVZ;
                           ballV1 <= ballVX;
                           ballV2 <= ballVY;
                           //i know it has to be (~positiveZ) and ballVZ > ballZ
                           if (positiveX)
                              begin
                                 ball1 <= ballX;
                                 ball11<= ballZ;
                                 ball12<= ZFloor;
                              end
                           else
                              begin
                                 ball1 <= ballX - ballVX;
                                 ball11<= ballVZ
                                 ball12<= ballZ;
                              end
                           if (positiveY)
                              begin
                                 ball2 <= ballY;
                                 ball21<=ballZ;
                                 ball12<=ZFloor;
                              end
                           else
                              begin
                                 ball2 <= ballY - ballVY;
                                 ball21<=ballVZ;
                                 ball22<=ballZ;
                              end
                        end
                     else
                        // in case of ((intersectWithRimPlane) &&
                        //(intersectWithBoardPlane) && (~intersectWithPolePlane) &&
                        //(~intersectWithFloorPlane) && (~intersectWithBoundary))
                        begin
                           //always set the status and set busy to 0 when goes to state "setStatus1"
                           state <= setStatus1;
                           status <= keepGoing;
                           busy <= 0;
                        end
                     end

               getIntersectFloor:

                     begin
                        //it is ok if it is just high for one clock cycle
                        //see simulation wave
                        startGetIntersection <=0;
                        //busy won't be high right away, it need one clock cycle
                        //if getIntersection is working, stay in this one
                        if (startGetIntersection || busyGetIntersection1 || busyGetIntersection2) state <=
getIntersectFloor;
                        //if it is done, check if it does intersect with the board
                        //check with the
                        //int1 is in X, int2 is in Y
                        //TODO: check this what does getIntersection give me if it is negative?
                        else if ((int1>=XStart)&&(int1<=XEnd)&&(int2>=ZFloor)&&(int2<=ZEnd))
                           //if it is in range then it bounce off the floor
                           //TODO: depend on where it ball hit, it should give different possibilities for
acceleration
```

```
                        begin
                           state <= setStatus1;
                           status <= collideFloor;
                           busy <= 0;
                        end
                     // other wise go to the others
                     else if (intersectWithBoundary)
                        begin
                           //always set the status and set busy to 0 when goes to state "setStatus1"
                           state <= setStatus1;
                           status <= miss;
                           busy<=0;
                        end
                     else
                        // in case of ((intersectWithRimPlane) &&
                        //(~intersectWithBoardPlane) && (~intersectWithPolePlane) &&
                        //(~intersectWithFloorPlane) && (~intersectWithBoundary))
                        begin
                           //always set the status and set busy to 0 when goes to state "setStatus1"
                           state <= setStatus1;
                           status <= keepGoing;
                           busy <= 0;
                        end
                  end

               //adds delay, so that before it return to idle, the start signal is turned off
               //outside, other wise it will start again unexpectedly
               //TODO: check if the second one is needed
               setStatus1:   state <= setStatus2;

               setStatus2:   state <= idle;

               default: state<= idle;

            endcase
         end
      end


   //getIntersectionInd(clock, reset, start, ballX1, ballX2, ball1, ballV0, ballV1, int1, busy);
   getIntersectionInd myIntInd1(clock, reset, startGetIntersection, ball11, ball12,
               ball1, ballV0, ballV1, int1, busyGetIntersection1);

   getIntersectionInd myIntInd2(clock, reset, startGetIntersection, ball21, ball22,
               ball2, ballV0, ballV2, int2, busyGetIntersection2);

endmodule
```

getStatusLogic.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Engineer: Jingwen Ouyang
// Module Name:    getStatusLogic
// Additional Comments: I think this piece of code can be shortened if i creat some smaller modules
//          for each situation... but did not have time to clean this
//
//////////////////////////////////////////////////////////////////////////////////
module getStatusLogic(clock, reset, start,ballX, ballY, ballZ, positiveX, positiveY, positiveZ,
                  ballVX, ballVY, ballVZ, status, busy);
    //inputs
    input clock;
    input reset;
    input start;
    input [9:0] ballX;
    input [9:0] ballY;
    input [9:0] ballZ;
    input positiveX;
    input positiveY;
    input positiveZ;
    input [9:0] ballVX;
    input [9:0] ballVY;
    input [9:0] ballVZ;

    //outputs
    output reg [4:0] status;
    output reg busy;

    //talk to getIntersection
    reg startGetIntersection;
    wire busyGetIntersection1;
    wire busyGetIntersection2;
    reg [9:0] ball11; //ballX1 > ballX2;
    reg [9:0] ball12;
    reg [9:0] ball21;
    reg [9:0] ball22;
    reg [9:0] ball1;
    reg [9:0] ball2;
    reg [9:0] ballV0;
    reg [9:0] ballV1;
    reg [9:0] ballV2;
    wire [9:0] int1;
    wire [9:0] int2;

//  //internal variables, they are reused through out of the calculation
//  reg [9:0] tempPosX;
//  reg [9:0] tempPosY;
//  reg [9:0] tempPosZ;

    //bits to indicate which plane it might intersect
    //set them to right one, not just flip it
    reg intersectWithRimPlane;
    reg intersectWithBoardPlane;
    reg intersectWithPolePlane;
    reg intersectWithFloorPlane;
    reg intersectWithBoundary;

    //the important x points
    parameter XStart =10'd0;
    parameter XBoard1 =10'd108;
    parameter XPole1 =10'd165;
    parameter XPole2 =10'd189;
    parameter XBoard2 =10'd246;
    parameter XEnd =10'd400;

    //the important x points
    parameter YStart =10'd0;
    parameter YBoard =10'd217;
    parameter YPole =10'd225;
    parameter YEnd =10'd400;

    //the important z points
    parameter ZFloor =10'd0;
```

```
    parameter ZBoard1 = 10'd206;
    parameter ZBoard2 = 10'd286;
    parameter ZEnd =10'd400;

    //TODO: for rim
    parameter rimX = 10'd177;
    parameter rimY = 10'd188;
    parameter rimZ = 10'd230;

    parameter ballRadiusSquaredSmall = 10'd196;
    parameter ballRadiusSquaredBig = 10'd720;

    //state assignemnt
    reg [2:0] state;
    parameter idle = 3'd0;
    parameter getIntersectStatus = 3'd1;
    parameter getIntersectRim = 3'd2;
    parameter getIntersectBoard = 3'd3;
    parameter getIntersectPole = 3'd4;
    parameter getIntersectFloor = 3'd5;
    parameter setStatus1 = 3'd6;
    parameter setStatus2 = 3'd7;



    //TODO: output status parameter assignment
//  parameter keepGoing = 5'd0;
//  parameter score = 5'd2;
//  parameter miss = 5'd3;
//  parameter collideBoard1 = 4'd4;
//  parameter collideBoard2 = 4'd5;
//  parameter collideBoard3 = 4'd6;
//  parameter collideBoard4 = 4'd7;
//  parameter collideFloor = 4'd8;
//  parameter collideRim = 4'd9;
//  parameter collidePole = 4'd10;
    parameter keepGoing = 5'd1;
    parameter score = 5'd2;
    parameter miss = 5'd3;
    parameter collideFloor = 5'd4;
    parameter collideRim = 5'd5;
    parameter collidePole = 5'd6;
    parameter collideBoard = 5'd7;


    always @ (posedge clock)

      begin

          //based on how the ball travels, i simplified it to a few cases,
          //ignored some irregular travel path
          //I check to see if it scores first, then see if it will run into the board or the pole
          //then it could be either bounce off floor or out of bounds

          //the ball might collide with the board/pole, rim, or the floor
          //1. colliding with the floor the speed change is one case
          //2. to make it real colliding with the board/pole will have some variaties.
          //   I divided the ball into 9 boxes (3 x 3) so that the velocity will change depend on
          //   which part the ball is colliding, the change of velocity will be different
          //3. the ball should stop moving if it roll out of bounds (out of 0-400 in any axis)
          //   or bouce on the floor twice (which means there will have a counter)

          //need state machine because some calculation takes time, especially when
          //it computes the possible intersection points

          //TODO: in this basic version, there is actually no change of velocity in Z...
          //because video lack that axie, so i will not check the case where it bounce off the floor
          //at negative axis...

          if (reset)
            begin
               state <= idle;
               startGetIntersection <= 0;
               busy <= 0;
               intersectWithRimPlane<= 0;
               intersectWithBoardPlane<= 0;
```

```
      intersectWithPolePlane<= 0;
      intersectWithFloorPlane<= 0;
      intersectWithBoundary<= 0;
      ball11<=10'd0;
      ball12<=0;
      ball21<=0;
      ball22<=0;
      ball1 <=0;
      ball2 <=0;
      ballV0<=0;
      ballV1<=0;
      ballV2<=0;
   end
else

   begin

      case (state)

         idle:

            if (~start)
               begin
                  state<= idle;
                  busy<= 0;
               end

            else

               begin

                  state<= getIntersectStatus;
                  busy <= 1;

                  //check where it might intersect

                  //check with Rim
                  if (positiveZ)
                     begin
                        if ((ballZ <= rimZ) && ((ballZ+ballVZ) >= rimZ))
                           intersectWithRimPlane <= 1;
                        else intersectWithRimPlane <= 0;
                     end
                  else
                     begin
                        if ((ballZ >= rimZ) && ((ballZ-ballVZ) <= rimZ))
                           intersectWithRimPlane <= 1;
                        else intersectWithRimPlane <= 0;
                     end

                  //check with board
                  if (positiveY)
                     begin
                        if ((ballY <= YBoard) && ((ballY+ballVY) >= YBoard))
                           intersectWithBoardPlane <= 1;
                        else intersectWithBoardPlane <= 0;
                     end
                  else
                     begin
                        if ((ballY >= YBoard) && ((ballY-ballVY) <= YBoard))
                           intersectWithBoardPlane <= 1;
                        else intersectWithBoardPlane <= 0;
                     end

                  //check with pole
                  if (positiveY)
                     begin
                        if ((ballY <= YPole) && ((ballY+ballVY) >= YPole))
                           intersectWithPolePlane <= 1;
                        else intersectWithBoardPlane <= 0;
                     end
                  else
                     begin
                        if ((ballY >= YPole) && ((ballY-ballVY) <= YPole))
                           intersectWithPolePlane <= 1;
                        else intersectWithBoardPlane <= 0;
```

```
            end


            //check with floor
            if ((~positiveZ) && (ballZ >= ZFloor) && (ballVZ >= ballZ))
                    intersectWithFloorPlane <= 1;
            else intersectWithFloorPlane <= 0;

            //check with x,y,z boundarys except floor; 0-400
            //Z
            if (positiveZ && (ballZ <= ZEnd) && ((ballZ+ballVZ) >= ZEnd))
               intersectWithBoundary <= 1;
            //Y
            else if (positiveY && (ballY <= YEnd) && ((ballY+ballVY) >= YEnd))
               intersectWithBoundary <= 1;
            else if ((~positiveY) && (ballY >= YStart) && (ballY<=ballVY))
               intersectWithBoundary <= 1;
            //X
            else if (positiveX && (ballX <= XEnd) && ((ballX+ballVX) >= XEnd))
               intersectWithBoundary <= 1;
            else if ((~positiveX) && (ballX >= XStart) && (ballX<=ballVX))
               intersectWithBoundary <= 1;
            else intersectWithBoundary <= 0;

         end

      getIntersectStatus:

         if ((~intersectWithRimPlane) && (~intersectWithBoardPlane) &&
            (~intersectWithPolePlane) && (~intersectWithFloorPlane) &&
            (intersectWithBoundary))
            begin
               //always set the status and set busy to 0 when goes to state "setStatus1"
               state <= setStatus1;
               status <= miss;
               busy<=0;
            end
         else if ((~intersectWithRimPlane) && (~intersectWithBoardPlane) &&
            (~intersectWithPolePlane) && (intersectWithFloorPlane))
            begin
               // if going to getIntersectXXX then set the right signal
               state <= getIntersectFloor;
               startGetIntersection<=1;
               //ballX1 > ballX2 , and ballV0 in Z axies,
               //set ball1 in X, ball2 in Y
               ballV0 <= ballVZ;
               ballV1 <= ballVX;
               ballV2 <= ballVY;
               //i know it has to be (~positiveZ) and ballVZ > ballZ
               if (positiveX)
                  begin
                     ball1 <= ballX;
                     ball11<= ballZ;
                     ball12<= ZFloor;
                  end
               else
                  begin
                     ball1 <= ballX - ballVX;
                     ball11<= ballVZ;
                     ball12<= ballZ;
                  end
               if (positiveY)
                  begin
                     ball2 <= ballY;
                     ball21<=ballZ;
                     ball22<=ZFloor;
                  end
               else
                  begin
                     ball2 <= ballY - ballVY;
                     ball21<=ballVZ;
                     ball22<=ballZ;
                  end
            end
         else if ((~intersectWithRimPlane) && (~intersectWithBoardPlane) &&
            (intersectWithPolePlane))
```

```
   begin
      state <= getIntersectPole;
      startGetIntersection<=1;
      //ballX1 > ballX2 , and ballV0 in Y axies,
      //set ball1 in X, ball2 in Z
      ballV0 <= ballVY;
      ballV1 <= ballVX;
      ballV2 <= ballVZ;
      if (positiveX)
         begin
            ball1 <= ballX;
            if (positiveY)
               //same direction
               begin
                  ball11<= YPole;
                  ball12<= ballY;
               end
            else
               //opposite direction
               begin
                  ball11<=ballY;
                  ball12<=YPole;
               end
         end
      else
         begin
            ball1 <= ballX - ballVX;
            if (positiveY)
               begin
                  ball11<=ballVY+ballY;
                  ball12<=YPole;
               end
            else
               begin
                  ball11<=YPole;
                  ball12<=ballY-ballVY;
               end
         end
      if (positiveZ)
         begin
            ball2 <= ballZ;
            if (positiveY)
               //same direction
               begin
                  ball21<= YPole;
                  ball22<= ballY;
               end
            else
               //opposite direction
               begin
                  ball21<=ballY;
                  ball22<=YPole;
               end
         end
      else
         begin
            ball2 <= ballZ - ballVZ;
            if (positiveY)
               begin
                  ball21<=ballVY+ballY;
                  ball22<=YPole;
               end
            else
               begin
                  ball21<=YPole;
                  ball22<=ballY-ballVY;
               end
         end
   end
else if ((~intersectWithRimPlane) && (intersectWithBoardPlane))
   begin
      state <= getIntersectBoard;
      startGetIntersection<=1;
      //ballX1 > ballX2 , and ballV0 in Y axies,
      //set ball1 in X, ball2 in Z
      ballV0 <= ballVY;
```

```
            ballV1 <= ballVX;
            ballV2 <= ballVZ;
            if (positiveX)
               begin
                  ball1 <= ballX;
                  if (positiveY)
                     //same direction
                     begin
                        ball11<= YBoard;
                        ball12<= ballY;
                     end
                  else
                     //opposite direction
                     begin
                        ball11<=ballY;
                        ball12<=YBoard;
                     end
               end
            else
               begin
                  ball1 <= ballX - ballVX;
                  if (positiveY)
                     begin
                        ball11<=ballVY+ballY;
                        ball12<=YPole;
                     end
                  else
                     begin
                        ball11<=YPole;
                        ball12<=ballY-ballVY;
                     end
               end
            if (positiveZ)
               begin
                  ball2 <= ballZ;
                  if (positiveY)
                     //same direction
                     begin
                        ball21<= YBoard;
                        ball22<= ballY;
                     end
                  else
                     //opposite direction
                     begin
                        ball21<=ballY;
                        ball22<=YBoard;
                     end
               end
            else
               begin
                  ball2 <= ballZ - ballVZ;
                  if (positiveY)
                     begin
                        ball21<=ballVY+ballY;
                        ball22<=YBoard;
                     end
                  else
                     begin
                        ball21<=YBoard;
                        ball22<=ballY-ballVY;
                     end
               end
         end
      else if (intersectWithRimPlane)
         begin
            state <= getIntersectRim;
            startGetIntersection<=1;
            //ballX1 > ballX2 , and ballV0 in Z axies,
            //set ball1 in X, ball2 in Y
            ballV0 <= ballVZ;
            ballV1 <= ballVX;
            ballV2 <= ballVY;
            if (positiveX)
               begin
                  ball1 <= ballX;
                  if (positiveZ)
```

```
                        //same direction
                        begin
                           ball11<= rimZ;
                           ball12<= ballZ;
                        end
                     else
                        //opposite direction
                        begin
                           ball11<=ballZ;
                           ball12<=rimZ;
                        end
                  end
               else
                  begin
                     ball1 <= ballX - ballVX;
                     if (positiveZ)
                        begin
                           ball11<=ballVZ+ballZ;
                           ball12<=rimZ;
                        end
                     else
                        begin
                           ball21<=rimZ;
                           ball22<=ballZ-ballVZ;
                        end
                  end
               if (positiveY)
                  begin
                     ball2 <= ballY;
                     if (positiveZ)
                        //same direction
                        begin
                           ball21<= rimZ;
                           ball22<= ballZ;
                        end
                     else
                        //opposite direction
                        begin
                           ball21<=ballZ;
                           ball22<=rimZ;
                        end
                  end
               else
                  begin
                     ball2 <= ballY - ballVY;
                     if (positiveZ)
                        begin
                           ball21<=ballVZ+ballZ;
                           ball22<=rimZ;
                        end
                     else
                        begin
                           ball21<=rimZ;
                           ball22<=ballZ-ballVZ;
                        end
                  end
            end
         else
            // in case of ((~intersectWithRimPlane) &&
            //(~intersectWithBoardPlane) && (~intersectWithPolePlane) &&
            //(~intersectWithFloorPlane) && (~intersectWithBoundary))
            begin
               //always set the status and set busy to 0 when goes to state "setStatus1"
               state <= setStatus1;
               status <= keepGoing;
               busy <= 0;
            end

   getIntersectRim:

         begin
            //it is ok if it is just high for one clock cycle
            //see simulation wave
            startGetIntersection<=0;
            //busy won't be high right away, it need one clock cycle
            //if getIntersection is working, stay in this one
```

```
                        if (startGetIntersection || busyGetIntersection1 || busyGetIntersection2) state <=
getIntersectRim;
                        //if it is done, check if it does intersect with the rim
                        //check with the radius
                        //int0 is in Z, int1 is in X, int2 is in Y
                        //TODO: check this
                        //there is built in multiplier, but need positive inputs
                        else if ((~positiveZ) &&
                                 (((int1 > (rimX+9)) && (int2 > (rimY+9)) &&
                                  (((int1-(rimX+9))*(int1-(rimX+9))+(int2-(rimY+9))*(int2-(rimY+9)))<
ballRadiusSquaredSmall))
                                  ||((int1 > (rimX+9)) && (int2 < (rimY+9)) &&
                                     (((int1-(rimX+9))*(int1-(rimX+9))+((rimY+9)-int2)*((rimY+9)-int2))<
ballRadiusSquaredSmall))
                                  ||((int1 < (rimX+9)) && (int2 < (rimY+9)) &&
                                     ((((rimX+9)-int1)*((rimX+9)-int1)+((rimY+9)-int2)*((rimY+9)-int2))<
ballRadiusSquaredSmall))
                                  ||((int1 < (rimX+9)) && (int2 > (rimY+9)) &&
                                     ((((rimX+9)-int1)*((rimX+9)-int1)+(int2-(rimY+9))*(int2-
(rimY+9)))<ballRadiusSquaredSmall))))
                           //if it is in range then we scores
                           begin
                              state <= setStatus1;
                              status <= score;
                              busy <= 0;
                           end
                        // other wise go to the others
                        else if ((~intersectWithBoardPlane) && (~intersectWithPolePlane) &&
(~intersectWithFloorPlane) &&
                                 (intersectWithBoundary))
                           begin
                              //always set the status and set busy to 0 when goes to state "setStatus1"
                              state <= setStatus1;
                              status <= miss;
                              busy<=0;
                           end
                        else if ((~intersectWithBoardPlane) && (~intersectWithPolePlane) &&
(intersectWithFloorPlane))
                           begin
                              // if going to getIntersectXXX then set the right signal
                              state <= getIntersectFloor;
                              startGetIntersection<=1;
                              //ballX1 > ballX2 , and ballV0 in Z axies,
                              //set ball1 in X, ball2 in Y
                              ballV0 <= ballVZ;
                              ballV1 <= ballVX;
                              ballV2 <= ballVY;
                              //i know it has to be (~positiveZ) and ballVZ > ballZ
                              if (positiveX)
                                 begin
                                    ball1 <= ballX;
                                    ball11<= ballZ;
                                    ball12<= ZFloor;
                                 end
                              else
                                 begin
                                    ball1 <= ballX - ballVX;
                                    ball11<= ballVZ;
                                    ball12<= ballZ;
                                 end
                              if (positiveY)
                                 begin
                                    ball2 <= ballY;
                                    ball21<=ballZ;
                                    ball12<=ZFloor;
                                 end
                              else
                                 begin
                                    ball2 <= ballY - ballVY;
                                    ball21<=ballVZ;
                                    ball22<=ballZ;
                                 end
                           end
                        else if ((~intersectWithBoardPlane) && (intersectWithPolePlane))
                           begin
                              state <= getIntersectPole;
```

```
            startGetIntersection<=1;
            //ballX1 > ballX2 , and ballV0 in Y axies,
            //set ball1 in X, ball2 in Z
            ballV0 <= ballVY;
            ballV1 <= ballVX;
            ballV2 <= ballVZ;
            if (positiveX)
               begin
                  ball1 <= ballX;
                  if (positiveY)
                     //same direction
                     begin
                        ball11<= YPole;
                        ball12<= ballY;
                     end
                  else
                     //opposite direction
                     begin
                        ball11<=ballY;
                        ball12<=YPole;
                     end
               end
            else
               begin
                  ball1 <= ballX - ballVX;
                  if (positiveY)
                     begin
                        ball11<=ballVY+ballY;
                        ball12<=YPole;
                     end
                  else
                     begin
                        ball11<=YPole;
                        ball12<=ballY-ballVY;
                     end
               end
            if (positiveZ)
               begin
                  ball2 <= ballZ;
                  if (positiveY)
                     //same direction
                     begin
                        ball21<= YPole;
                        ball22<= ballY;
                     end
                  else
                     //opposite direction
                     begin
                        ball21<=ballY;
                        ball22<=YPole;
                     end
               end
            else
               begin
                  ball2 <= ballZ - ballVZ;
                  if (positiveY)
                     begin
                        ball21<=ballVY+ballY;
                        ball22<=YPole;
                     end
                  else
                     begin
                        ball21<=YPole;
                        ball22<=ballY-ballVY;
                     end
               end
         end
      else if (intersectWithBoardPlane)
         begin
            state <= getIntersectBoard;
            startGetIntersection<=1;
            //ballX1 > ballX2 , and ballV0 in Y axies,
            //set ball1 in X, ball2 in Z
            ballV0 <= ballVY;
            ballV1 <= ballVX;
            ballV2 <= ballVZ;
```

```
            if (positiveX)
               begin
                  ball1 <= ballX;
                  if (positiveY)
                     //same direction
                     begin
                        ball11<= YBoard;
                        ball12<= ballY;
                     end
                  else
                     //opposite direction
                     begin
                        ball11<=ballY;
                        ball12<=YBoard;
                     end
               end
            else
               begin
                  ball1 <= ballX - ballVX;
                  if (positiveY)
                     begin
                        ball11<=ballVY+ballY;
                        ball12<=YPole;
                     end
                  else
                     begin
                        ball11<=YPole;
                        ball12<=ballY-ballVY;
                     end
               end
            if (positiveZ)
               begin
                  ball2 <= ballZ;
                  if (positiveY)
                     //same direction
                     begin
                        ball21<= YBoard;
                        ball22<= ballY;
                     end
                  else
                     //opposite direction
                     begin
                        ball21<=ballY;
                        ball22<=YBoard;
                     end
               end
            else
               begin
                  ball2 <= ballZ - ballVZ;
                  if (positiveY)
                     begin
                        ball21<=ballVY+ballY;
                        ball22<=YBoard;
                     end
                  else
                     begin
                        ball21<=YBoard;
                        ball22<=ballY-ballVY;
                     end
               end
         end
      else
         // in case of ((~intersectWithRimPlane) &&
         //(~intersectWithBoardPlane) && (~intersectWithPolePlane) &&
         //(~intersectWithFloorPlane) && (~intersectWithBoundary))
         begin
            //always set the status and set busy to 0 when goes to state "setStatus1"
            state <= setStatus1;
            status <= keepGoing;
            busy <= 0;
         end
   end
```

```
                getIntersectBoard:

                   begin
                      //it is ok if it is just high for one clock cycle
                      //see simulation wave
                      startGetIntersection<=0;
                      //busy won't be high right away, it need one clock cycle
                      //if getIntersection is working, stay in this one
                      if (startGetIntersection || busyGetIntersection1 || busyGetIntersection2) state <=
getIntersectBoard;
                      //if it is done, check if it does intersect with the board
                      //check with the
                      //int1 is in X, int2 is in Z
                      //TODO: check this
                      else if ((int1<XBoard2)&&(int1>XBoard1)&&(int2<ZBoard2)&&(int2>ZBoard1))
                         //if it is in range then it bounce off the board
                         //TODO: depend on where it ball hit, it should give different possibilities for
acceleration
                         begin
                            state <= setStatus1;
                            status <= collideBoard;
                            busy <= 0;
                         end
                      // other wise go to the others
                      else if ((~intersectWithPolePlane) && (~intersectWithFloorPlane) &&
(intersectWithBoundary))
                         begin
                            //always set the status and set busy to 0 when goes to state "setStatus1"
                            state <= setStatus1;
                            status <= miss;
                            busy<=0;
                         end
                      else if ((~intersectWithPolePlane) && (intersectWithFloorPlane))
                         begin
                            // if going to getIntersectXXX then set the right signal
                            state <= getIntersectFloor;
                            startGetIntersection<=1;
                            //ballX1 > ballX2 , and ballV0 in Z axies,
                            //set ball1 in X, ball2 in Y
                            ballV0 <= ballVZ;
                            ballV1 <= ballVX;
                            ballV2 <= ballVY;
                            //i know it has to be (~positiveZ) and ballVZ > ballZ
                            if (positiveX)
                               begin
                                  ball1 <= ballX;
                                  ball11<= ballZ;
                                  ball12<= ZFloor;
                               end
                            else
                               begin
                                  ball1 <= ballX - ballVX;
                                  ball11<= ballVZ;
                                  ball12<= ballZ;
                               end
                            if (positiveY)
                               begin
                                  ball2 <= ballY;
                                  ball21<=ballZ;
                                  ball12<=ZFloor;
                               end
                            else
                               begin
                                  ball2 <= ballY - ballVY;
                                  ball21<=ballVZ;
                                  ball22<=ballZ;
                               end
                         end
                      else if (intersectWithPolePlane)
                         begin
                            state <= getIntersectPole;
                            startGetIntersection<=1;
                            //ballX1 > ballX2 , and ballV0 in Y axies,
                            //set ball1 in X, ball2 in Z
                            ballV0 <= ballVY;
                            ballV1 <= ballVX;
```

```
                ballV2 <= ballVZ;
                if (positiveX)
                   begin
                      ball1 <= ballX;
                      if (positiveY)
                         //same direction
                         begin
                            ball11<= YPole;
                            ball12<= ballY;
                         end
                      else
                         //opposite direction
                         begin
                            ball11<=ballY;
                            ball12<=YPole;
                         end
                   end
                else
                   begin
                      ball1 <= ballX - ballVX;
                      if (positiveY)
                         begin
                            ball11<=ballVY+ballY;
                            ball12<=YPole;
                         end
                      else
                         begin
                            ball11<=YPole;
                            ball12<=ballY-ballVY;
                         end
                   end
                if (positiveZ)
                   begin
                      ball2 <= ballZ;
                      if (positiveY)
                         //same direction
                         begin
                            ball21<= YPole;
                            ball22<= ballY;
                         end
                      else
                         //opposite direction
                         begin
                            ball21<=ballY;
                            ball22<=YPole;
                         end
                   end
                else
                   begin
                      ball2 <= ballZ - ballVZ;
                      if (positiveY)
                         begin
                            ball21<=ballVY+ballY;
                            ball22<=YPole;
                         end
                      else
                         begin
                            ball21<=YPole;
                            ball22<=ballY-ballVY;
                         end
                   end
             end
          else
             // in case of ((~intersectWithRimPlane) &&
             //(~intersectWithBoardPlane) && (~intersectWithPolePlane) &&
             //(~intersectWithFloorPlane) && (~intersectWithBoundary))
             begin
                //always set the status and set busy to 0 when goes to state "setStatus1"
                state <= setStatus1;
                status <= keepGoing;
                busy <= 0;
             end
       end


    getIntersectPole:
```

```
                    begin
                       //it is ok if it is just high for one clock cycle
                       //see simulation wave
                       startGetIntersection<=0;
                       //busy won't be high right away, it need one clock cycle
                       //if getIntersection is working, stay in this one
                       if (startGetIntersection || busyGetIntersection1 || busyGetIntersection2) state <=
getIntersectPole;
                       //if it is done, check if it does intersect with the board
                       //check with the
                       //int1 is in X, int2 is in Z
                       //TODO: check this
                       else if ((int1<XPole2)&&(int1>XPole1)&&(int2<ZBoard1)&&(int2>ZFloor))
                          //if it is in range then it bounce off the pole
                          //TODO: depend on where it ball hit, it should give different possibilities for
acceleration
                          begin
                             state <= setStatus1;
                             status <= collidePole;
                             busy <= 0;
                          end
                       // other wise go to the others
                       else if ((~intersectWithFloorPlane) && (intersectWithBoundary))
                          begin
                             //always set the status and set busy to 0 when goes to state "setStatus1"
                             state <= setStatus1;
                             status <= miss;
                             busy<=0;
                          end
                       else if (intersectWithFloorPlane)
                          begin
                             // if going to getIntersectXXX then set the right signal
                             state <= getIntersectFloor;
                             startGetIntersection<=1;
                             //ballX1 > ballX2 , and ballV0 in Z axies,
                             //set ball1 in X, ball2 in Y
                             ballV0 <= ballVZ;
                             ballV1 <= ballVX;
                             ballV2 <= ballVY;
                             //i know it has to be (~positiveZ) and ballVZ > ballZ
                             if (positiveX)
                                begin
                                   ball1 <= ballX;
                                   ball11<= ballZ;
                                   ball12<= ZFloor;
                                end
                             else
                                begin
                                   ball1 <= ballX - ballVX;
                                   ball11<= ballVZ;
                                   ball12<= ballZ;
                                end
                             if (positiveY)
                                begin
                                   ball2 <= ballY;
                                   ball21<=ballZ;
                                   ball12<=ZFloor;
                                end
                             else
                                begin
                                   ball2 <= ballY - ballVY;
                                   ball21<=ballVZ;
                                   ball22<=ballZ;
                                end
                          end
                       else
                          // in case of ((intersectWithRimPlane) &&
                          //(intersectWithBoardPlane) && (~intersectWithPolePlane) &&
                          //(~intersectWithFloorPlane) && (~intersectWithBoundary))
                          begin
                             //always set the status and set busy to 0 when goes to state "setStatus1"
                             state <= setStatus1;
                             status <= keepGoing;
                             busy <= 0;
                          end
```

```
                        end

                getIntersectFloor:

                      begin
                         //it is ok if it is just high for one clock cycle
                         //see simulation wave
                         startGetIntersection <=0;
                         //busy won't be high right away, it need one clock cycle
                         //if getIntersection is working, stay in this one
                         if (startGetIntersection || busyGetIntersection1 || busyGetIntersection2) state <=
getIntersectFloor;
                         //if it is done, check if it does intersect with the board
                         //check with the
                         //int1 is in X, int2 is in Y
                         //TODO: check this what does getIntersection give me if it is negative?
                         else if ((int1>=XStart)&&(int1<=XEnd)&&(int2>=ZFloor)&&(int2<=ZEnd))
                            //if it is in range then it bounce off the floor
                            //TODO: depend on where it ball hit, it should give different possibilities for
acceleration
                            begin
                               state <= setStatus1;
                               status <= collideFloor;
                               busy <= 0;
                            end
                         // other wise go to the others
                         else if (intersectWithBoundary)
                            begin
                               //always set the status and set busy to 0 when goes to state "setStatus1"
                               state <= setStatus1;
                               status <= miss;
                               busy<=0;
                            end
                         else
                            // in case of ((intersectWithRimPlane) &&
                            //(~intersectWithBoardPlane) && (~intersectWithPolePlane) &&
                            //(~intersectWithFloorPlane) && (~intersectWithBoundary))
                            begin
                               //always set the status and set busy to 0 when goes to state "setStatus1"
                               state <= setStatus1;
                               status <= keepGoing;
                               busy <= 0;
                            end
                      end

                   //adds delay, so that before it return to idle, the start signal is turned off
                   //outside, other wise it will start again unexpectedly
                   //TODO: check if the second one is needed
                   setStatus1:   state <= setStatus2;

                   setStatus2:   state <= idle;

                   default: state<= idle;

                endcase
             end
          end


      //getIntersectionInd(clock, reset, start, ballX1, ballX2, ball1, ballV0, ballV1, int1, busy);
      getIntersectionInd myIntInd1(clock, reset, startGetIntersection, ball11, ball12,
                      ball1, ballV0, ballV1, int1, busyGetIntersection1);

      getIntersectionInd myIntInd2(clock, reset, startGetIntersection, ball21, ball22,
                      ball2, ballV0, ballV2, int2, busyGetIntersection2);

endmodule
```

lab4_labkit.v

```
LabKit

   //////////////////////////////////////////////////////////////////////////
   //
   // Lab 4 Components within I/O module
   // Jingwen Ouyang
   //
```

```
   //////////////////////////////////////////////////////////////////////////

   //
   // Generate a 31.5MHz pixel clock from clock_27mhz
   //



   wire pclk, pixel_clock;
   DCM pixel_clock_dcm (.CLKIN(clock_27mhz), .CLKFX(pclk));
   // synthesis attribute CLKFX_DIVIDE of pixel_clock_dcm is 6
   // synthesis attribute CLKFX_MULTIPLY of pixel_clock_dcm is 7
   // synthesis attribute CLK_FEEDBACK of pixel_clock_dcm  is "NONE"
   BUFG pixel_clock_buf (.I(pclk), .O(pixel_clock));

   //
   // VGA output signals
   //

   // Inverting the clock to the DAC provides half a clock period for signals
   // to propagate from the FPGA to the DAC.
   assign vga_out_pixel_clock = ~pixel_clock;

   // The composite sync signal is used to encode sync data in the green
   // channel analog voltage for older monitors.  It does not need to be
   // implemented for the monitors in the 6.111 lab, and can be left at 1'b1.
   assign vga_out_sync_b = 1'b1;



   wire [9:0] pixel_count;
   wire [9:0] line_count;
   wire theFrameClock;
   //wire [8:0] paddlePosY;
   wire [9:0] ballPosY;
   wire [9:0] ballPosX;


   // power-on reset generation
   wire power_on_reset; // remain high for first 16 clocks
   SRL16 reset_sr (.D(1'b0), .CLK(clock_27mhz), .Q(power_on_reset),
   .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
   defparam reset_sr.INIT = 16'hFFFF;
   wire reset = power_on_reset | ~button_enter;

   //make sure to creat all the connection for the blocks need to color
   // wire [23:0] rgbSignal, rgbWall01, rgbBall, rgbWall02,rgbWall03;
   //wire [23:0] rgbSignal, rgbPaddle, rgbBall, rgbWall01, rgbWall02,rgbWall03;
   //   wire [23:0] rgbMIT01, rgbMIT02, rgbMIT03, rgbMIT04, rgbMIT05, rgbMIT06, rgbMIT07;
   wire [23:0] rgbSignal, rgbCourt;
   wire [23:0] rgbBall, rgbSignal;
   //rgbBall,rgbBeaver;
   wire [23:0] rgbNumber;
   wire [23:0] rgbShadow;


   //this one is the thing made the ball to go up and down
   //first get the Y position for paddle getPaddlePosY(frameClock, reset,up,down,poxY)
//   getPaddlePosY myBallPosY(pixel_clock, theFrameClock, reset, ~button_up, ~button_down, tempBallPosY);
// tempDrawBall(clock, reset, posX, posY, pixel_count, line_count, rgbOut);
//tempDrawBall myTempDrawBall(pixel_clock, reset, outX, outY, pixel_count, line_count, rgbBall);

   //wire [23:0] rgbPaddle;

   controllerVGA checkerBoardVGA (reset,pixel_clock, hsync, vsync, pixel_count,
      line_count, hblank, vblank);

   delay delayVGA (reset,pixel_clock, ~hsync, ~vsync, vga_out_hsync, vga_out_vsync);

   assign theFrameClock = (pixel_count == 639) && (line_count == 479);



   calculateForTwoD myCalculateBallTwoD(pixel_clock, reset, 1'd1, frameClock, 10'd177, ballPosX, ballPosY,outX,
outY,busy);
   //module basicLogicBall(clock, frameClock, reset, vY, vZ, pY, pZ, ballY, ballZ);
```

```
   //basicLogicBall myBallPositions(pixel_clock, theFrameClock, reset, 10'd2, 10'd10, 10'd50, 10'd360, ballPosX,
ballPosY);
   //draw the ball
   //drawRectangle drawBall(pixel_clock, reset, ballPosX, ballPosY, pixel_count, line_count, rgbBall);
   //drawRectangle drawBall(pixel_clock, reset, outX, outY, pixel_count, line_count, rgbBall);
   assign rgbSignal = rgbBall;

   wire [9:0] outX, outY;
   wire busy;



   //getShadowPos(clock, ballX, ballY, shadowX, shadowY, shadowZ);
    getShadowPos myGetShadowPos(pixel_clock, 10'd28, tempBallPosY, shadowX,shadowY,shadowZ);
    //drawShadow(reset, clock, shadowX, shadowY, shadowZ, pixel_count, line_count, rgbShadow);
    calculateForTwoD myCalculateShadowTwoD(pixel_clock, reset, 1'd1, frameClock, shadowX,shadowY,shadowZ,outSX,
outSY,busy)
   drawShadow myDrawShadow(reset, pixel_clock, 10'd28, 10'd400, 10'd0, pixel_count, line_count, rgbShadow);
      tempDrawShadow myTempDrawShadow(pixel_clock, reset, outSX, outSY, pixel_count, line_count, rgbShadow);
   testBackground(clock,  reset, pixel_count, line_count, hblank, vblank, rgbOut)



   //TODO: get the start and next signal for this, next is bottom, start from Chun
   drawbeaver myDrawBeaver(reset, pixel_clock, frameClock, ~button0, ~button0,
                  pixel_count, line_count, 10'd177, 10'd0,10'd33,
                  10'd0, 10'd4, 10'd3, switch[1],switch[2],switch[3],
                  rgbBall, rgbBeaver);

// tempDrawSad mytempDrawSad(pixel_clock, reset, posX, posY, pixel_count, line_count, rgbBall1);
// tempDrawScore mytempDrawScore(pixel_clock, reset, posX, posY, pixel_count, line_count, rgbBall2);
// assign rgbNumber = 24'h800080;
// assign rgbShadow = 24'h800080;


   rgbController myRGbController(pixel_clock, rgbBall,rgbCourt,rgbBeaver,rgbShadow, rgbNumber,rgbSignal);


   testBackground myTestBackground(pixel_clock, reset, pixel_count, line_count, rgbCourt);
   // assign rgbSignal = rgbCourt;

   // VGA Output
   assign vga_out_red = rgbSignal[23:16];
   assign vga_out_green = rgbSignal[15:8];
   assign vga_out_blue = rgbSignal[7:0];


     assign vga_out_blank_b = hblank && vblank;



endmodule
```

tempDrawSad.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Engineer: Jingwen Ouyang
// Create Date:    20:19:07 05/15/2007
// Module Name:    tempDrawSad
// Additional Comments: Similar to the temp draw ball
//
//////////////////////////////////////////////////////////////////////////////////
module tempDrawSad(pixel_clock, reset, posX, posY, pixel_count, line_count, rgbOut);
    input pixel_clock;
    input reset;
    input [9:0] posX;
    input [8:0] posY;
    input [9:0] pixel_count;
    input [9:0] line_count;
    output [23:0] rgbOut;

    reg [8:0] romAddress;
    wire [3:0] romColorIndex;
    reg [3:0] colorIndex;
    //ball is 18*18 pixel


    always @ (posedge pixel_clock)
      begin
         if (reset)
            romAddress <= 9'd0;
         //should draw ball
         else if ((line_count>=posY)&&(line_count<posY+100))
            begin
               if ((pixel_count>=posX)&&(pixel_count<posX+100))
                  begin
                     romAddress <= romAddress + 1;
                     colorIndex <= romColorIndex; //TODO: the timing here may not be right, one clock cycle
delay
                  end
               else
                  begin
                     romAddress <= romAddress;
                     colorIndex <= 4'b0101; //will be treat as transparent
                  end
            end
         else
            begin
               romAddress <= 9'd0;
               colorIndex <= 4'b0101; //will be treat as transparent
            end
      end

    sadfacerom myfaceRom(
      .addr(romAddress),
      .clk(clock),
      .dout(romColorIndex)
    );

    colorMapping myTestFaceColorMapping(clock, colorIndex, rgbOut);
endmodule
```

tempDrawScore.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Engineer: Jingwen Ouyang
// Module Name:    tempDrawScore
// Additional Comments: Similar to the temp draw ball
//
//////////////////////////////////////////////////////////////////////////////////
module tempDrawScore(pixel_clock, reset, posX, posY, pixel_count, line_count, rgbOut);

    input pixel_clock;
    input reset;
    input [9:0] posX;
    input [8:0] posY;
    input [9:0] pixel_count;
    input [9:0] line_count;
    output [23:0] rgbOut;

    reg [8:0] romAddress;
    wire [3:0] romColorIndex;
    reg [3:0] colorIndex;
    //ball is 18*18 pixel


    always @ (posedge pixel_clock)
      begin
         if (reset)
            romAddress <= 9'd0;
         //should draw ball
         else if ((line_count>=posY)&&(line_count<posY+100))
            begin
               if ((pixel_count>=posX)&&(pixel_count<posX+80))
                  begin
                     romAddress <= romAddress + 1;
                     colorIndex <= romColorIndex; //TODO: the timing here may not be right, one clock cycle
delay
                  end
               else
                  begin
                     romAddress <= romAddress;
                     colorIndex <= 4'b0101; //will be treat as transparent
                  end
            end
         else
            begin
               romAddress <= 9'd0;
               colorIndex <= 4'b0101; //will be treat as transparent
            end
      end

    sadfacerom myScoreRom(
      .addr(romAddress),
      .clk(clock),
      .dout(romColorIndex)
    );

    colorMapping myTestScoreColorMapping(clock, colorIndex, rgbOut);
endmodule
```